Red Hat Enterprise Linux OpenStack Platform 6
Deploying OpenStack: Learning Environments (Manual Setup)

Manually installing and configuring Red Hat Enterprise Linux OpenStack Platform

OpenStack Documentation Team

Red Hat Enterprise Linux OpenStack Platform 6 Deploying OpenStack: Learning Environments (Manual Setup)

Manually installing and configuring Red Hat Enterprise Linux OpenStack Platform

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 on Red Hat Enterprise Linux. In particular, this guide focuses on providing an instructional walkthrough of the OpenStack deployment process. As such, this guide provides instructions on how to install each OpenStack component manually. Manually installing OpenStack components will result in a fully functional OpenStack deployment, but may not be suitable for most production environments.

Table of Contents

Part I. Introduction

Chapter 1. Introduction

Red Hat Enterprise Linux OpenStack Platform provides the foundation to build a private or public Infrastructure-as-a-Service (IaaS) cloud on top of Red Hat Enterprise Linux. It offers a massively scalable, fault-tolerant platform for the development of cloud-enabled workloads.

The current Red Hat system is based on OpenStack Juno, and packaged so that available physical hardware can be turned into a private, public, or hybrid cloud platform including:

- Fully distributed object storage
- Persistent block-level storage
- Virtual-machine provisioning engine and image storage
- Authentication and authorization mechanism
- Integrated networking
- Web browser-based GUI for both users and administration.

The Red Hat Enterprise Linux OpenStack Platform IaaS cloud is implemented by a collection of interacting services that control its computing, storage, and networking resources. The cloud is managed using a web-based interface which allows administrators to control, provision, and automate OpenStack resources. Additionally, the OpenStack infrastructure is facilitated through an extensive API, which is also available to end users of the cloud.

Note

For an overview of the OpenStack components and their interfaces, see the "Component Overview"(https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux_OpenStack_Platform/).

1.1. Supported Virtual Machine Operating Systems

All guest operating systems that are certified with KVM in Red Hat Enterprise Linux 6 and Red Hat Enterprise Linux 7.1 are supported by RHEL OpenStack Platform 6. A detailed list of the supported guest operating systems can be found here: Certified Guest Operating Systems in Red Hat Enterprise Linux OpenStack Platform and Red Hat Enterprise Virtualization.

Chapter 2. Prerequisites

2.1. Software Requirements

2.1.1. Operating System Requirements

This version of Red Hat Enterprise Linux OpenStack Platform is supported on Red Hat Enterprise Linux 7.1.

For detailed information on installing Red Hat Enterprise Linux, see the corresponding installation guide at:

https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/

2.1.2. Software Repository Configuration

2.1.2.1. Customer Portal Subscription Management

Red Hat Enterprise Linux OpenStack Platform requires each system in the OpenStack environment be running Red Hat Enterprise Linux Server and that all systems be signed up to receive updates from the Customer Portal Subscription Management using Subscription Manager. For further information on managing Red Hat subscriptions, see the Red Hat Subscription Management documentation at the following link:

https://access.redhat.com/site/documentation/en-US/Red_Hat_Subscription_Management/

All steps in this procedure must be executed while logged in to the account of the **root** user on the system being registered.

Procedure 2.1. Registering a Red Hat Enterprise Linux system using Subscription Management

1. Run the **subscription-manager register** command to register the system with Customer Portal Subscription Management.

```
# subscription-manager register
```

2. Enter your Red Hat Customer Portal user name when prompted.

```
Username: admin@example.com
```

Important

Your Red Hat Subscription must have Red Hat Enterprise Linux OpenStack Platform entitlements. If your subscription does not have Red Hat Enterprise Linux OpenStack entitlements, you can register for access to the evaluation program at http://www.redhat.com/products/enterprise-linux/openstack-platform/.

3. Enter your Red Hat Customer Portal password when prompted.

```
Password:
```

4. When registration completes successfully, the system is assigned a unique identifier.

```
The system has been registered with id: IDENTIFIER
```

The system has been registered with Customer Portal Subscription Management, and you can now attach software subscriptions.

2.1.2.2. Configure Content Delivery Network (CDN) Channels

This section discusses channel and repository settings required for deploying Red Hat Enterprise Linux OpenStack Platform 6.

Warning

Although older Red Hat OpenStack repositories are available, you must ensure that your system can no longer access them before installing Red Hat Enterprise Linux OpenStack Platform 6. For example, unsubscribe from or disable the following:

- Red Hat Enterprise Linux OpenStack Platform 4 (Havana) -- rhel-6-server-openstack-4.0-rpms
- Red Hat Enterprise Linux OpenStack Platform 5 (Icehouse) -- rhel-7-server-openstack-5.0-rpms

Packstack registers systems to Red Hat Network using Subscription Manager. You might encounter problems if your systems have already been registered and subscribed to the Red Hat OpenStack channels using RHN Classic.

2.1.2.2.1. Content Delivery Network (CDN) Channels

To install Red Hat Enterprise Linux OpenStack Platform 6 through the Content Delivery Network (CDN), configure the correct channels (repositories):

Run the following command to enable a CDN channel:

```
# subscription-manager repos --enable=[reponame]
```

Run the following command to disable a CDN channel:

```
# subscription-manager repos --disable=[reponame]
```

Red Hat Enterprise Linux 7.1

The following tables outline the channels for Red Hat Enterprise Linux 7.1.

Table 2.1. Required Channels

Channel	Repository Name
Red Hat Enterprise Linux 7 Server (RPMS)	`rhel-7-server-rpms`
Red Hat OpenStack 6.0 for Server 7 (RPMS)	`rhel-7-server-openstack-6.0-rpms`
Red Hat Enterprise Linux 7 Server - RH Common (RPMs)	`rhel-7-server-rh-common-rpms`

Table 2.2. Optional Channels

Channel	Repository Name
Red Hat Enterprise Linux 7 Server - Optional	**rhel-7-server-optional-rpms**

Red Hat Enterprise Linux OpenStack Platform Installer

The following tables outline the channels for the Red Hat Enterprise Linux OpenStack Platform installer.

Table 2.3. Required Channels

Channel	Repository Name
Red Hat OpenStack 6.0 for RHEL 7 Platform Installer (RPMs)	**rhel-7-server-openstack-6.0-installer-rpms**
Red Hat Enterprise Linux 7 Server (RPMS)	**rhel-7-server-rpms**
Red Hat Software Collections RPMs for Red Hat Enterprise Linux 7 Server	**rhel-server-rhscl-7-rpms**
Images on CDN (Optional)	**rhel-7-server-openstack-6.0-files**

Disable Channels

The following table outlines the channels you must disable to ensure Red Hat Enterprise Linux OpenStack Platform 6 functions correctly.

Table 2.4. Disable Channels

Channel	Repository Name
Red Hat CloudForms Management Engine	**"cf-me-*"**
Red Hat CloudForms Tools for RHEL 6	**"rhel-6-server-cf-*"**
Red Hat Enterprise Virtualization	**"rhel-6-server-rhev*"**
Red Hat Enterprise Linux 6 Server - Extended Update Support	**"*-eus-rpms"**

2.1.2.3. Red Hat Enterprise Linux Repository Configuration

Log in as the **root** user and follow the steps in this procedure to configure a Red Hat Enterprise Linux system to receive updates from Red Hat Network. Repeat these steps on each system in the OpenStack environment.

Procedure 2.2. Attaching Pool IDs to your subscription

1. Use the **subscription-manager list** command to locate the pool identifier of the Red Hat Enterprise Linux subscription.

```
# subscription-manager list --available
+-------------------------------------------+
    Available Subscriptions
+-------------------------------------------+

Product Name:          Red Hat Enterprise Linux Server
Product Id:            69
Pool Id:               POOLID
```

```
Quantity:               1
Service Level:          None
Service Type:           None
Multi-Entitlement:      No
Expires:                01/01/2022
Machine Type:           physical
...
```

The pool identifier is indicated in the **Pool Id** field associated with the **Red Hat Enterprise Linux Server** product. The identifier will be unique to your subscription. Take note of this identifier as it will be required to perform the next step.

Note

The output displayed in this step has been truncated to conserve space. All other available subscriptions will also be listed in the output of the command.

2. Use the **subscription-manager attach** command to attach the subscription identified in the previous step.

```
# subscription-manager attach --pool=POOLID
Successfully attached a subscription for Red Hat Enterprise Linux
Server.
```

Replace *POOLID* with the unique identifier associated with your Red Hat Enterprise Linux Server subscription. This is the identifier that was located in the previous step.

3. Run the **yum repolist** command. This command ensures that the repository configuration file **/etc/yum.repos.d/redhat.repo** exists and is up to date.

```
# yum repolist
```

Once repository metadata has been downloaded and examined, the list of repositories enabled will be displayed, along with the number of available packages.

```
repo id                  repo name
status
rhel-7-server-rpms       Red Hat Enterprise Linux 7 Server (RPMs)
8,816
repolist: 8,816
```

Note

The output displayed in this step may differ from that which appears when you run the **yum repolist** command on your system. In particular the number of packages listed will vary if or when additional packages are added to the **rhel-7-server-rpms** repository.

You have successfully configured your system to receive Red Hat Enterprise Linux updates from Red Hat Network.

2.1.2.4. Red Hat Enterprise Linux OpenStack Platform Repository Configuration

Follow the steps in this procedure to configure a Red Hat Enterprise Linux system to receive OpenStack packages and updates from Content Delivery Network or Red Hat Network. Access to a Red Hat software entitlement that includes Red Hat Enterprise Linux OpenStack Platform is required, such entitlements include:

- Red Hat Cloud Infrastructure
- Red Hat Cloud Infrastructure (without Guest OS)
- Red Hat Enterprise Linux OpenStack Platform
- Red Hat Enterprise Linux OpenStack Platform Preview
- Red Hat Enterprise Linux OpenStack Platform (without Guest OS)

Important

Required and optional repository names for each version are listed in Section 2.1.2.2, "Configure Content Delivery Network (CDN) Channels".

These steps must be run while logged in as the **root** user. Repeat these steps on each system in the environment.

Procedure 2.3. Attaching Pool IDs to your subscription

1. Use the **subscription-manager list** command to locate the pool identifier of the relevant Red Hat Cloud Infrastructure or Red Hat Enterprise Linux OpenStack Platform entitlement.

```
# subscription-manager list --available
+-------------------------------------------+
    Available Subscriptions
+-------------------------------------------+
...
Product Name:          ENTITLEMENT
Product Id:            ID_1
Pool Id:               POOLID_1
Quantity:              3
Service Level:         None
Service Type:          None
Multi-Entitlement:     No
Expires:               DATE
Machine Type:          physical

Product Name:          ENTITLEMENT
Product Id:            ID_2
Pool Id:               POOLID_2
Quantity:              unlimited
Service Level:         None
Service Type:          None
```

```
Multi-Entitlement:     No
Expires:               DATE
Machine Type:          virtual
...
```

Locate the entry in the list where the **Product Name** matches the name of the entitlement that will be used to access Red Hat Enterprise Linux OpenStack Platform packages. Take note of the pool identifier associated with the entitlement, this value is indicated in the **Pool Id** field. The pool identifier is unique to your subscription and will be required to complete the next step.

Note

The output displayed in this step has been truncated to conserve space. All other available subscriptions will also be listed in the output of the command.

2. Use the **subscription-manager attach** command to attach the subscription identified in the previous step.

```
# subscription-manager attach --pool=POOLID
Successfully attached a subscription for ENTITLEMENT.
```

Replace *POOLID* with the unique identifier associated with your Red Hat Cloud Infrastructure or Red Hat Enterprise Linux OpenStack Platform entitlement. This is the identifier that was located in the previous step.

3. Use either the **subscription-manager** or **yum-config-manager** commands to enable or disable the appropriate software repositories (channels).

For example, to ensure that the repository for Red Hat Enterprise Linux OpenStack Platform 3 (Grizzly) has been disabled, run:

```
# subscription-manager repos --disable rhel-server-ost-6-3-rpms
Loaded plugins: product-id
==== repo: rhel-server-ost-6-3-rpms ====
[rhel-server-ost-6-3-rpms]
bandwidth = 0
base_persistdir = /var/lib/yum/repos/x86_64/6Server
baseurl =
https://cdn.redhat.com/content/dist/rhel/server/6/6Server/x86_64/op
enstack/3/os
cache = 0
cachedir = /var/cache/yum/x86_64/6Server/rhel-server-ost-6-3-rpms
cost = 1000
enabled = False
...
```

Note

The values **True** and **1** are equivalent. As a result the output on your system may instead contain this string:

```
enabled = 1
```

4. Run the **yum repolist** command. This command ensures that the repository configuration file **/etc/yum.repos.d/redhat.repo** exists and is up to date.

```
# yum repolist
```

Once repository metadata has been downloaded and examined, the current list of enabled repositories will be displayed, along with the number of available packages. For example:

```
repo id                repo name
status
rhel-7-server-rpms     Red Hat Enterprise Linux 7 Server (RPMs)
11,610+460
rhel-7-server-openstack-6.0-rpms  \
Red Hat OpenStack 6.0 for Server 7 (RPMS)   487+143
```

5. Use the **subscription-manager** command to enable the Red Hat Enterprise Linux OpenStack Platform repository.

```
# subscription-manager repos --enable=[reponame]
```

Replace *[reponame]* with the appropriate repository name. For details, see Section 2.1.2.2, “Configure Content Delivery Network (CDN) Channels”.

6. Install the *yum-plugin-priorities* package provided by the **rhel-6-server-optional-rpms** channel for Red Hat Enterprise Linux 6.6 or the **rhel-7-server-optional-rpms** channel for Red Hat Enterprise Linux 7.1:

```
# yum install yum-plugin-priorities
```

7. If not yet installed, use the following to install yum-config-manager:

```
# yum install yum-utils
```

8. Use the **yum-config-manager** command to set the priority of the Red Hat Enterprise Linux OpenStack Platform software repository to **1**. This is the highest priority value supported by the *yum-plugin-priorities* plug-in.

```
# yum-config-manager --enable [reponame] --
setopt="[reponame].priority=1"
```

For example:

```
# yum-config-manager --enable rhel-7-server-openstack-6.0-rpms \
```

```
        --setopt="rhel-7-server-openstack-6.0-rpms.priority=1"
Loaded plugins: product-id
==== repo: rhel-7-server-openstack-6.0-rpms ====
[rhel-7-server-openstack-6.0-rpms]
bandwidth = 0
base_persistdir = /var/lib/yum/repos/x86_64/6Server
...
cost = 1000
enabled = True
...
priority = 1
...
```

9. Run the **yum *update*** command and reboot to ensure that the most up-to-date packages, including the kernel, are installed and running.

```
# yum update
# reboot
```

You have successfully configured your system to receive Red Hat Enterprise Linux OpenStack Platform packages. You may use the **yum repolist** command to confirm the repository configuration again at any time.

2.1.3. Reserved UIDs and GIDs

The reserved range for UIDs and GIDs in Red Hat Enterprise Linux is currently 0-500. Because your organization may need to change one or more of these, the following tables are provided for OpenStack and third-party components that it uses. If you are assigning UIDs and GIDS, it is best to start at a number higher than 1000 (higher than 5000 might be a good strategy).

Table 2.5. OpenStack Daemons

Component	Code	Reserved UID	Reserved GID
Identity	keystone	163	163
Block Storage	cinder	165	165
Compute	nova	162	162
Image	glance	161	161
Object Storage	swift	160	160
Telemetry	ceilometer	166	166
Orchestration	heat	187	187

Table 2.6. Third-party Components

Component	Reserved UID	Reserved GID
MongoDB	184	184
Memcached	497	496
MariaDB	27	27
Nagios	496	495
RabbitMQ	103	106

2.2. Hardware Requirements

2.2.1. 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.

Memory

A minimum of 2 GB of RAM is recommended.

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 100 GB of available disk space is recommended.

Add additional disk space to this requirement based on the amount of space that you intend to make available to virtual machine instances. This figure varies based on both the size of each disk image you intend to create and whether you intend to share one or more disk images between multiple instances.

1 TB of disk space is recommended for a realistic environment capable of hosting multiple instances of varying sizes.

Network Interface Cards

A minimum of 2 x 1 Gbps Network Interface Cards, although it is ideal to have 3 x 1 Gbps Network Interface Cards.

2.2.2. Network Node Requirements

Network nodes are responsible for hosting the services that provide networking functionality to compute instances. In particular, they host the DHCP agent, layer 3 agent, and metadata proxy services. Like all systems that handle networking in an OpenStack environment, they also host an instance of the layer 2 agent.

The hardware requirements of network nodes vary widely depending on the networking workload of the environment. The requirements listed here are intended as a guide to the minimum requirements of a network node.

Processor

No specific CPU requirements are imposed by the networking services.

Memory

A minimum of 2 GB of RAM is recommended.

Disk Space

A minimum of 10 GB of available disk space is recommended.

No additional disk space is required by the networking services other than that required to install the packages themselves. However, some disk space must be available to store log files and temporary files.

Network Interface Cards

2 x 1 Gbps Network Interface Cards.

2.2.3. Block Storage Node Requirements

Block storage nodes are nodes that host the volume service (**`openstack-cinder-volume`**) and provide volumes for use by virtual machine instances or other cloud users. The block storage API (**`openstack-cinder-api`**) and scheduling services (**`openstack-cinder-scheduler`**) can run on the same nodes as the volume service, or on separate nodes. In either case, the primary hardware requirement of the block storage nodes is that there is enough block storage available to serve the needs of the environment.

The amount of block storage required in an environment varies in accordance with the following factors:

- The number of volumes that will be created in the environment.
- The average size of the volumes that will be created in the environment.
- Whether or not the storage back end will be configured to support redundancy.
- Whether or not the storage back end will be configured to create sparse volumes by default.

Use the following formula to assist with estimating the initial block storage needs of your environment:

```
VOLUMES * SIZE * REDUNDANCY * UTILIZATION = TOTAL
```

- Replace *VOLUMES* with the number of volumes that are expected to exist in the environment at any one time.
- Replace *SIZE* with the expected average size of the volumes (in gigabytes) that will exist in the environment at any one time.
- Replace *REDUNDANCY* with the expected number of redundant copies of each volume that the back end storage will be configured to store. Use **1**, or skip this multiplication operation if no redundancy is required.
- Replace *UTILIZATION* with the expected percentage of each volume that will actually be used. Use **1**, indicating 100%, if the use of sparse volumes will not be enabled.

The resultant figure represents an **estimate** of the block storage needs of your environment in gigabytes. It is recommended that some additional space is allowed for future growth. Addition of further block storage after the environment has been deployed is facilitated by adding more block storage providers and, if necessary, additional instances of the volume service.

2.3. Prerequisite Database Server

Each OpenStack component requires a running MariaDB service. As such, you will need to deploy one before deploying a full OpenStack cloud service or installing any single OpenStack component.

2.3.1. Install the MariaDB Database Packages

The following packages are required by the MariaDB database server:

mariadb-galera-server

Provides the MariaDB database server.

mariadb-galera-common

Provides the MariaDB server shared files. Installed as a dependency of the *mariadb-galera-server* package.

galera

Installs the Galera wsrep provider. Installed as a dependency of the *mariadb-galera-server* package.

To install the required packages, log in as the **root** user and run:

```
# yum install mariadb-galera-server
```

The database server is installed and ready to be configured.

2.3.2. Configure the Firewall to Allow Database Traffic

As the database service is used by all of the components in the OpenStack environment it must be accessible by them.

To allow this the firewall on the system hosting the database service must be altered to allow network traffic on the required port. All steps in this procedure must be run while logged in to the server hosting the database service as the **root** user.

Procedure 2.4. Configuring the firewall to allow database traffic

1. Open the **/etc/sysconfig/iptables** file in a text editor.
2. Add an INPUT rule allowing TCP traffic on port **3306** to the file. The new rule must appear before any INPUT rules that REJECT traffic.

   ```
   -A INPUT -p tcp -m multiport --dports 3306 -j ACCEPT
   ```

3. Save the changes to the **/etc/sysconfig/iptables** file.
4. Restart the **iptables** service to ensure that the change takes effect.

   ```
   # service iptables restart
   ```

The firewall is now configured to allow incoming connections to the MariaDB database service on port **3306**.

2.3.3. Start the Database Service

All steps in this procedure must be performed while logged in to the server hosting the database service as the **root** user.

Procedure 2.5. Start the database service (for Red Hat Enterprise Linux 7-based systems)

1. Use the **systemctl** command to start the **mariadb.service**:

```
# systemctl start mariadb.service
```

2. Use the **systemctl** command to ensure that **mariadb.service** will be started automatically in the future:

```
# systemctl enable mariadb.service
```

The database service has been started, and is configured to start automatically on boot.

2.3.4. Configuring the Database Administrator Account

Summary

By default, MariaDB creates a database user account called **root** that provides access to the MariaDB server from the machine on which the MariaDB server was installed. You must manually set a password for this account to secure access to the MariaDB server. Moreover, you must create a user account that provides access to the MariaDB server from machines other than the machine on which the MariaDB server is installed if required.

Procedure 2.6. Configuring the Database Administrator Account

1. Log in to the machine on which the MariaDB server is installed.
2. Set a password for the local **root** user:

```
# mysqladmin -u root password "PASSWORD"
```

3. Optionally, configure a user account for remote access:
   a. Use the **mysql** client to connect to the MariaDB server:

```
# mysql -p
```

   b. Enter the newly configured password for the **root** database user account:

```
Enter password:
```

   c. Create a user called **root** that can access the MariaDB server from remote machines:

```
MariaDB [(none)]> CREATE USER 'root'@'%' IDENTIFIED BY
'PASSWORD';
```

   d. Grant the newly created user access to resources in the database:

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON *.* TO
'root'@'%';
```

   e. Exit the **mysql** client:

```
MariaDB [(none)]> \q
```

Summary

You have set a password for the **root** database user with permission to access the MariaDB server from the machine on which the MariaDB server is installed, and optionally created a new database user account with permission to access the MariaDB server from a machine other than the machine on which the MariaDB server is installed.

Note

You can also use the **mysqladmin** command to change the password of a database user if required. In the following example, replace *OLDPASS* with the existing password of the user and *NEWPASS* with a new password, leaving no space between **-p** and the old password:

```
# mysqladmin -u root -pOLDPASS password NEWPASS
```

2.3.5. Testing Connectivity

To ensure a database user account has been correctly configured, you can test the connectivity of that user account with the MariaDB server from the machine on which the MariaDB server is installed (local connectivity), and from a machine other than the machine on which the MariaDB server is installed (remote connectivity).

2.3.5.1. Testing Local Connectivity

Summary

Test whether you can connect to the MariaDB server from the machine on which the MariaDB server is installed.

Procedure 2.7. Testing Local Connectivity

1. Log in to the machine on which the MariaDB server is installed.
2. Use the **mysql** client tool to connect to the MariaDB server, replacing **USER** with the user name by which to connect:

   ```
   $ mysql -u USER -p
   ```

3. Enter the password of the database user when prompted.

   ```
   Enter password:
   ```

Result

If the permissions for the database user are correctly configured, the connection succeeds and the MariaDB welcome screen and prompt are displayed. If the permissions for the database user are not correctly configured, an error message is displayed that explains that the database user is not allowed to connect to the MariaDB server.

2.3.5.2. Testing Remote Connectivity

Summary

Test whether you can connect to the MariaDB server from a machine other than the machine on which the MariaDB server is installed.

Procedure 2.8. Testing Remote Connectivity

1. Log in to a machine other than the machine on which the MariaDB server is installed.
2. Install the MySQL client tools provided by the *mysql* package:

```
# yum install mysql
```

3. Use the **mysql** client tool to connect to the MariaDB server, replacing *USER* with the user name and *HOST* with the IP address or fully qualified domain name of the MariaDB server:

```
# mysql -u USER -h HOST -p
```

4. Enter the password of the database user when prompted:

```
Enter password:
```

Result

If the permissions for the database user are correctly configured, the connection succeeds and the MariaDB welcome screen and prompt are displayed. If the permissions for the database user are not correctly configured, an error message is displayed that explains that the database user is not allowed to connect to the MariaDB server.

2.4. Prerequisite Message Broker

If you are deploying a full OpenStack cloud service, you will need to set up a working message broker for the following OpenStack components:

- Block Storage
- Compute
- OpenStack Networking
- Orchestration
- Image Service
- Telemetry

For Red Hat Enterprise Linux OpenStack Platform 5, the default message broker is RabbitMQ.

2.4.1. Configure the Firewall for Message Broker Traffic

Before installing and configuring the message broker, you must allow incoming connections on the port it will use. The default port for message broker (AMQP) traffic is **5672**.

To allow this the firewall must be altered to allow network traffic on the required port. All steps must be run while logged in to the server as the **root** user.

Procedure 2.9. Configuring the firewall for message broker traffic

1. Open the **/etc/sysconfig/iptables** file in a text editor.
2. Add an **INPUT** rule allowing incoming connections on port **5672** to the file. The new rule must appear before any **INPUT** rules that **REJECT** traffic.

```
-A INPUT -p tcp -m tcp --dport 5672  -j ACCEPT
```

3. Save the changes to the **/etc/sysconfig/iptables** file.
4. Restart the **iptables** service for the firewall changes to take effect.

```
# service iptables restart
```

The firewall is now configured to allow incoming connections to the MariaDB database service on port **5672**.

2.4.2. Install and Configure the RabbitMQ Message Broker

As of Red Hat Enterprise Linux OpenStack Platform 5, RabbitMQ replaces QPid as the default (and recommended) message broker. The RabbitMQ messaging service is provided by the *rabbitmq-server* package.

To install RabbitMQ, run:

```
# yum install rabbitmq-server
```

Important

When installing the *rabbitmq-server* package, a **guest** user with a default **guest** password will automatically be created for the RabbitMQ service. Red Hat strongly advises that you change this default password, especially if you have IPv6 available. With IPv6, RabbitMQ may be accessible from outside the network.

You should be able to change the **guest** password after launching the **rabbitmq-server** server. See Section 2.4.2.1, “Launch the RabbitMQ Message Broker” for details.

2.4.2.1. Launch the RabbitMQ Message Broker

After installing the RabbitMQ message broker and configuring the firewall to accept message broker traffic, launch the **rabbitmq-server** service and configure it to launch on boot:

For Red Hat Enterprise Linux 7 systems, use the **systemctl** command to start and enable the **rabbitmq-server** service

```
# systemctl start rabbitmq-server.service
# systemctl enable rabbitmq-server
```

Important

When installing the *rabbitmq-server* package, a **guest** user with a default **guest** password will automatically be created for the RabbitMQ service. Red Hat strongly advises that you change this default password, especially if you have IPv6 available. With IPv6, RabbitMQ may be accessible from outside the network.

To change the default guest password of RabbitMQ:

```
# rabbitmqctl change_password guest NEW_RABBITMQ_PASS
```

Replace *NEW_RABBITMQ_PASS* with a more secure password.

After launching RabbitMQ and changing the default password of the **guest** user, perform the following procedure:

Procedure 2.10. Configuring the RabbitMQ message broker for OpenStack use

1. Create a RabbitMQ user account for the Block Storage, Compute, OpenStack Networking, Orchestration, Image, and Telemetry services:

   ```
   # rabbitmqctl add_user cinder CINDER_PASS
   # rabbitmqctl add_user nova NOVA_PASS
   # rabbitmqctl add_user neutron NEUTRON_PASS
   # rabbitmqctl add_user heat HEAT_PASS
   # rabbitmqctl add_user glance GLANCE_PASS
   # rabbitmqctl add_user ceilometer CEILOMETER_PASS
   ```

   Replace *CINDER_PASS*, *NOVA_PASS*, *NEUTRON_PASS*, *HEAT_PASS*, *GLANCE_PASS*, and *CEILOMETER_PASS* with secure passwords for each service.

2. Next, grant each of these RabbitMQ users read/write permissions to all resources:

   ```
   # rabbitmqctl set_permissions cinder ".*" ".*" ".*"
   # rabbitmqctl set_permissions nova ".*" ".*" ".*"
   # rabbitmqctl set_permissions neutron ".*" ".*" ".*"
   # rabbitmqctl set_permissions heat ".*" ".*" ".*"
   # rabbitmqctl set_permissions glance ".*" ".*" ".*"
   # rabbitmqctl set_permissions ceilometer ".*" ".*" ".*"
   ```

2.4.2.2. Enable SSL on the RabbitMQ Message Broker

The RabbitMQ message broker features built-in support for SSL, which you can use to secure traffic. You can configure SSL on RabbitMQ through the **/etc/rabbitmq/rabbitmq.config** configuration file. Before doing so, you will first need to create the certificates required for SSL communication.

The following procedure demonstrates how to create the necessary certificates using **certutil**.

Procedure 2.11. Creating and adding the certificates required for SSL communication

1. Create a directory in which to store the required certificates:

```
# mkdir /etc/pki/rabbitmq
```

2. Choose a secure certificate password and store it in a file within **/etc/pki/rabbitmq**:

```
# echo SSL_RABBITMQ_PW > /etc/pki/rabbitmq/certpw
```

Replace *SSL_RABBITMQ_PW* with the certificate password you wish to use. This password will be used later for further securing the necessary certificates..

3. Set the necessary file mode bits of the certificate directory and password file:

```
# chmod 700 /etc/pki/rabbitmq
# chmod 600 /etc/pki/rabbitmq/certpw
```

4. Create the certificate database files (***.db**) in **/etc/pki/rabbitmq**, using the password in **/etc/pki/rabbitmq/certpw**:

```
# certutil -N -d /etc/pki/rabbitmq -f /etc/pki/rabbitmq/certpw
```

5. For a production environment, it is recommended that you use a reputable third-party Certificate Authority (CA) to sign your certificates. A third-party CA will require a Certificate Signing Request (CSR), which you can create by running:

```
# certutil -R -d /etc/pki/rabbitmq -s "CN=RABBITMQ_HOST" \
 -a -f /etc/pki/rabbitmq/certpw > RABBITMQ_HOST.csr
```

Replace *RABBITMQ_HOST* with the IP or hostname of the system hosting the RabbitMQ message broker. This command will produce a CSR named ***RABBITMQ_HOST*.csr** and a key file (*keyfile.key*). The key file will be used later when configuring the RabbitMQ message broker to use SSL.

Note

Some CAs may require additional values other than **"CN=*RABBITMQ_HOST*"**.

6. Provide ***RABBITMQ_HOST*.csr** to your third-party CA for signing. Your CA should provide you with a signed certificate (*server.crt*) and a CA file (*ca.crt*). Add these files to your certificate database:

```
# certutil -A -d /etc/pki/rabbitmq -n RABBITMQ_HOST -f
/etc/pki/rabbitmq/certpw \
 -t u,u,u -a -i /path/to/server.crt
# certutil -A -d /etc/pki/rabbitmq -n "Your CA certificate" \
 -f /etc/pki/rabbitmq/certpw -t CT,C,C -a -i /path/to/ca.crt
```

Alternatively, you can also create a self-signed certificate and key file if you do not have a Certificate Authority. To do so, run:

```
# certutil -S -d /etc/pki/rabbitmq -n RABBITMQ_HOST -s
"CN=RABBITMQ_HOST" \
 -t "CT,," -x -f /etc/pki/rabbitmq/certpw -z /usr/bin/certutil
```

After creating the required certificates and key file, you can now configure the RabbitMQ message broker to use those certificates for secure communications. To do so, add the following section to the **/etc/rabbitmq/rabbitmq.config** configuration file:

```
[
  {rabbit, [
     {ssl_listeners, [5671]},
     {ssl_options, [{cacertfile,"/path/to/ca.crt"},
                    {certfile,"/path/to/server.crt"},
                    {keyfile,"/path/to/keyfile.key"},
                    {verify,verify_peer},
                    {fail_if_no_peer_cert,false}]}
   ]}
].
```

Note

If you are using a self-signed certificate, omit the following string:

```
{cacertfile,"/path/to/ca.crt"},
```

Procedure 2.12. Disable SSLv3

The following procedure disables SSLv3 by explicitly listing supported versions in the **rabbitmq.config** file:

1. Edit **rabbitmq.config** to only include support for specific TLS encryption versions:

```
{rabbit, [
{ssl_options, [{versions, ['tlsv1.2','tlsv1.1',tlsv1]}]},
]}
```

2. Restart the RabbitMQ service for the change to take effect:

```
# systemctl restart rabbitmq-server.service
```

2.4.2.3. Export an SSL Certificate for Clients

When SSL is enabled on a server, the clients require a copy of the SSL certificate to establish a secure connection.

The following example commands can be used to export a client certificate and the private key from the message broker's certificate database:

```
# pk12util -o <p12exportfile> -n <certname> -d <certdir> -w
<p12filepwfile>
# openssl pkcs12 -in <p12exportfile> -out <clcertname> -nodes -clcerts
-passin pass:<p12pw>
```

For more information on SSL commands and options, refer to the OpenSSL Documentation. On Red Hat Enterprise Linux type: **man openssl**.

2.5. Installation Prerequisites Checklists

The following tables describe prerequisites for successfully installing a Red Hat Enterprise Linux OpenStack Platform cloud. Checklist items are the minimum that should be known or verified before the installation is started.

The *Value/Verified* column can be used to provide the appropriate value or a 'check' that the item has been verified.

Note

If installing single components after the initial Red Hat Enterprise Linux OpenStack Platform installation, ensure that you have:

- **root** access to the host machine (to install components, as well other administrative tasks such as updating the firewall).
- Administrative access to the Identity service.
- Administrative access to the database (ability to add both databases and users).

Table 2.7. OpenStack Installation-General

Item	Description	Value/Verified
Hardware Requirements	Requirements in Section 2.2.1, “Compute Node Requirements”, Section 2.2.2, “Network Node Requirements”, and Section 2.2.3, “Block Storage Node Requirements” must be verified.	Yes | No
Operating System	Red Hat Enterprise Linux 7.1 Server	Yes | No
Red Hat Subscription	You must have a subscription to: - Receive package updates from Red Hat Network or an equivalent source such as a Red Hat Network Satellite server. - Receive software updates for both Red Hat Enterprise Linux 7.1 Server and Red Hat Enterprise Linux OpenStack Platform	Yes | No
Administrative access on all installation machines	Almost all procedures in this guide must be performed as the root user, so the installer must have root access.	Yes | No
Red Hat Subscriber Name/Password	You must know the Red Hat subscriber name and password.	- Name: - Password:

Item	Description	Value/Verified
Machine addresses	You must know the host IP address of the machine or machines on which any OpenStack components and supporting software will be installed.	Provide host addresses for the following: - Identity service - OpenStack Networking service - Block Storage service - Compute service - Image service - Object Storage service - Dashboard service - MariaDB server (default database for this guide)

Table 2.8. OpenStack Identity service

Item	Description	Value
Host Access	The system hosting the Identity service must have: - Access to Red Hat Network or equivalent service. - Network interface addressable by all OpenStack hosts. - Network access to the database server. - If using LDAP, network access to the directory server .	Verify whether the system has: - Yes | No - Yes | No - Yes | No - Yes | No
SSL Certificates	If using external SSL certificates, you must know where the database and certificates are located, and have access to them.	Yes | No
LDAP Information	If using LDAP, you must have administrative access to configure a new directory server schema.	Yes | No
Connections	The system hosting the Identity service must have a connection to all other OpenStack services.	Yes | No

Table 2.9. OpenStack Object Storage service

Item	Description	Value
File System	Red Hat currently supports the **XFS** and **ext4** file systems for object storage; one of these must be available.	- XFS - ext4
Mount Point	The /srv/node mount point must be available.	Yes | No
Connections	For the cloud installed in this guide, the system hosting the Object Storage service will need a connection to the Identity service.	Yes | No

Table 2.10. OpenStack Image Service

Item	Description	Value
Backend Storage	The Image service supports a number of storage backends. You must decide on one of the following: - file (local directory) - OpenStack Object Storage service	Storage:
Connections	The system hosting the Image service must have a connection to the Identity, Dashboard , and Compute services, as well as to the Object Storage service if using OpenStack Object Storage as its backend.	Yes | No

Table 2.11. OpenStack Block Storage service

Item	Description	Value
Backend Storage	The Block Storage service supports a number of storage backends. You must decide on one of the following: - LVM - NFS - Red Hat Storage	Storage:
Connections	The system hosting the Block Storage service must have a connection to the Identity, Dashboard, and Compute services.	Yes | No

Table 2.12. OpenStack Networking service

Item	Description	Value
Plugin agents	In addition to the standard OpenStack Networking components, a wide choice of plugin agents are also available that implement various networking mechanisms. You'll need to decide which of these apply to your network and install them.	Circle appropriate: - Open vSwitch - Cisco UCS/Nexus - Linux Bridge - VMware NSX virtualized network platform - Ryu OpenFlow Controller - NEC OpenFlow - Big Switch Controller Plugin - Cloudbase Hyper-V - MidoNet - Brocade Neutron Plugin - PLUMgrid
Connections	The system hosting the OpenStack Networking service must have a connection to the Identity, Dashboard, and Compute services.	Yes | No

Table 2.13. OpenStack Compute service

Item	Description	Value
Hardware virtualization support	The OpenStack Compute service requires hardware virtualization support. Note: a procedure is included in this Guide to verify this (refer to Section 8.1.1, “Check for Hardware Virtualization Support”).	Yes | No
VNC client	The Compute service supports the Virtual Network Computing (VNC) console access to instances through a web browser. You must decide whether this will be provided to your users.	Yes | No
Resources: CPU and Memory	OpenStack supports overcommitting of CPU and memory resources on Compute nodes (refer to Section 8.3.5.4, “Configure Resource Overcommitment”). • The default CPU overcommit ratio of 16 means that up to 16 virtual cores can be assigned to a node for each physical core. • The default memory overcommit ratio of 1.5 means that instances can be assigned to a physical node if the total instance memory usage is less than 1.5 times the amount of physical memory available.	Decide: • CPU setting: • Memory setting:
Resources: Host	You can reserve resources for the host, to prevent a given amount of memory and disk resources from being automatically assigned to other resources on the host (refer to Section 8.3.5.5, “Reserve Host Resources”).	Decide: • Host Disk (default 0MB): • Host Memory (default 512MB):
libvirt Version	You will need to know the version of your libvirt for the configuration of Virtual Interface Plugging (refer to Section 8.3.5.6.4, “Configure Virtual Interface Plugging”).	Version:
Connections	The system hosting the Compute service must have a connection to all other OpenStack services.	Yes | No

Table 2.14. OpenStack dashboard service

Item	Description	Value
Host software	The system hosting the dashboard service must have the following already installed: • httpd • mod_wsgi • mod_ssl	Yes | No
Connections	The system hosting the dashboard service must have a connection to all other OpenStack services.	Yes | No

Note

To install **mod_wsgi**, **httpd**, and **mod_ssl**, execute as root:

```
# yum install -y mod_wsgi httpd mod_ssl
```

Part II. Install OpenStack Services Manually

Chapter 3. OpenStack Identity Service Installation

3.1. Identity Service Requirements

The system hosting the Identity service must have:

- Access to Red Hat Network or equivalent service provided by a tool such as Satellite.
- A network interface that is addressable by all other systems that will host OpenStack services.
- Network access to the database server.
- Network access to the directory server if using an LDAP backend.

Ensure that these requirements are met before proceeding with installation and configuration of the Identity service.

3.2. Install the Identity Packages

The packages that provide the components of the Identity service are:

openstack-keystone

Provides the OpenStack Identity service.

openstack-utils

Provides supporting utilities to assist with a number of tasks including the editing of configuration files.

openstack-selinux

Provides OpenStack specific SELinux policy modules.

To install these packages, log in as the **root** user and run:

```
# yum install -y openstack-keystone \
    openstack-utils \
    openstack-selinux
```

The OpenStack Identity service is installed and ready to be configured.

3.3. Create the Identity Database

In this procedure the database and database user that will be used by the Identity service will be created. These steps must be performed while logged in to the database server as the **root** user (or at least as a user with the correct permissions: **create db**, **create user**, **grant permissions**).

Procedure 3.1. Creating the Identity Service database

1. Connect to the database service using the **mysql** command.

```
# mysql -u root -p
```

2. Create the **keystone** database.

```
mysql> CREATE DATABASE keystone;
```

3. Create a **keystone** database user and grant it access to the **keystone** database.

```
mysql> GRANT ALL ON keystone.* TO 'keystone'@'%' IDENTIFIED BY
'PASSWORD';
```

```
mysql> GRANT ALL ON keystone.* TO 'keystone'@'localhost'
IDENTIFIED BY 'PASSWORD';
```

   Replace *PASSWORD* with a secure password that will be used to authenticate with the database server as this user.

4. Flush the database privileges to ensure that they take effect immediately.

```
mysql> FLUSH PRIVILEGES;
```

5. Exit the **mysql** client.

```
mysql> quit
```

The database has been created. The database will be populated during service configuration.

3.4. Configure the Identity Service

3.4.1. Set the Identity Service Administration Token

Before the Identity service is started for the first time you must define an administrative token in an environment variable. This value will be used to authenticate before user and service accounts have been defined using the Identity service.

All steps listed in this procedure must be performed while logged in as the **root** user into the server that will host the Identity service.

Procedure 3.2. Setting the Identity Service administrative token

1. Use OpenSSL to generate an initial service token and save it in the **OS_SERVICE_TOKEN** environment variable.

```
# export OS_SERVICE_TOKEN=$(openssl rand -hex 10)
```

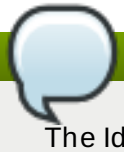
2. Store the value of the administration token in a file for future use.

```
# echo $OS_SERVICE_TOKEN > ~/ks_admin_token
```

3. Use the **openstack-config** tool to set the value of the **admin_token** configuration key to that of the newly created token.

```
# openstack-config --set /etc/keystone/keystone.conf \
DEFAULT admin_token $OS_SERVICE_TOKEN
```

The administration token for the Identity service has been created. This value will be used in subsequent Identity configuration procedures.



Note

The Identity server's token database table grows unconditionally over time as new tokens are generated. To clear the token table, the administrator must run the **keystone-manage token_flush** command to flush the tokens. Flushing tokens simply deletes expired tokens, eliminating any means of traceability. It is recommended that this command be run approximately once per minute.

```
# keystone-manage token_flush
```

3.4.2. Configure the Identity Service Database Connection

The database connection string used by the Identity service is defined in the **/etc/keystone/keystone.conf** file. It must be updated to point to a valid database server before starting the service.

All commands in this procedure must be run while logged in as the **root** user on the server hosting the Identity service.

Procedure 3.3. Configuring the Identity Service SQL database connection

- Use the **openstack-config** command to set the value of the **connection** configuration key.

```
# openstack-config --set /etc/keystone/keystone.conf \
  sql connection mysql://USER:PASS@IP/DB
```

Replace:

- *USER* with the database user name the Identity service is to use, usually **keystone**.
- *PASS* with the password of the chosen database user.
- *IP* with the IP address or host name of the database server.
- *DB* with the name of the database that has been created for use by the Identity service, usually **keystone**.

Important

The IP address or host name specified in the connection configuration key must match the IP address or host name to which the keystone database user was granted access when creating the keystone database. Moreover, if the database is hosted locally and you granted permissions to 'localhost' when creating the keystone database, you must enter 'localhost'.

The database connection string has been set and will be used by the Identity service.

3.4.3. Configure the Public Key Infrastructure

3.4.3.1. Public Key Infrastructure Overview

The Identity service generates tokens which are cryptographically signed documents users and other services use for authentication. The tokens are signed using a private key while the public key is made available in an X509 certificate.

The certificates and relevant configuration keys are automatically generated by the **keystone-manage pki_setup** command. It is however possible to manually create and sign the required certificates using a third party certificate authority. If using third party certificates the Identity service configuration must be manually updated to point to the certificates and supporting files.

The configuration keys relevant to PKI setup appear in the **[signing]** section of the **/etc/keystone/keystone.conf** configuration file. These keys are:

ca_certs

Specifies the location of the certificate for the authority that issued the certificate denoted by the **certfile** configuration key. The default value is **/etc/keystone/ssl/certs/ca.pem**.

ca_key

Specifies the key of the certificate authority that issued the certificate denoted by the **certfile** configuration key. The default value is **/etc/keystone/ssl/certs/cakey.pem**.

ca_password

Specifies the password, if applicable, required to open the certificate authority file. The default action if no value is specified is not to use a password.

certfile

Specifies the location of the certificate that must be used to verify tokens. The default value of **/etc/keystone/ssl/certs/signing_cert.pem** is used if no value is specified.

keyfile

Specifies the location of the private key that must be used when signing tokens. The default value of **/etc/keystone/ssl/private/signing_key.pem** is used if no value is specified.

token_format

Specifies the algorithm to use when generating tokens. Possible values are **UUID** and **PKI**. The default value is **PKI**.

3.4.3.2. Create the Public Key Infrastructure Files

This section explains how to create and configure the PKI files to be used by the Identity service. All steps listed in this procedure must be performed while logged into the system hosting the Identity service as the **root** user.

Procedure 3.4. Creating the PKI files to be used by the Identity service

1. Run the **keystone-manage pki_setup** command.

```
# keystone-manage pki_setup \
  --keystone-user keystone \
  --keystone-group keystone
```

2. Ensure that the **keystone** user owns the **/var/log/keystone/** and **/etc/keystone/ssl/** directories.

```
# chown -R keystone:keystone /var/log/keystone \
  /etc/keystone/ssl/
```

The Identity service PKI files have been created and will be used when generating and signing tokens.

3.4.3.3. Configure the Identity Service to Use Public Key Infrastructure Files

After generating the PKI files for use by the Identity service, you will need to enable the Identity service to use them.

Set the values of the attributes in the **/etc/keystone/keystone.conf** file by using the following commands:

```
# openstack-config --set /etc/keystone/keystone.conf \
 signing token_format PKI
# openstack-config --set /etc/keystone/keystone.conf \
 signing certfile /etc/keystone/ssl/certs/signing_cert.pem
# openstack-config --set /etc/keystone/keystone.conf \
 signing keyfile /etc/keystone/ssl/private/signing_key.pem
# openstack-config --set /etc/keystone/keystone.conf \
 signing ca_certs /etc/keystone/ssl/certs/ca.pem
# openstack-config --set /etc/keystone/keystone.conf \
 signing key_size 1024
# openstack-config --set /etc/keystone/keystone.conf \
 signing valid_days 3650
# openstack-config --set /etc/keystone/keystone.conf \
 signing ca_password None
```

You can also update these values directly by editing the **/usr/share/keystone/keystone-dist.conf** file.

3.4.4. Configure the Firewall to Allow Identity Service Traffic

As the Identity service is used by all of the components in the OpenStack environment for authentication it must be accessible by them.

To allow this the firewall on the system hosting the Identity service must be altered to allow network traffic on the required ports. All steps in this procedure must be run while logged in to the server hosting the Identity service as the **root** user.

Procedure 3.5. Configuring the firewall to allow Identity Service traffic

1. Open the **/etc/sysconfig/iptables** file in a text editor.
2. Add an INPUT rule allowing TCP traffic on ports **5000** and **35357** to the file. The new rule must appear before any INPUT rules that REJECT traffic.

```
-A INPUT -p tcp -m multiport --dports 5000,35357 -j ACCEPT
```

3. Save the changes to the **/etc/sysconfig/iptables** file.
4. Restart the **iptables** service to ensure that the change takes effect.

```
# service iptables restart
```

The firewall is now configured to allow incoming connections to the Identity service on ports **5000** and **35357**.

3.4.5. Populate the Identity Service Database

You can populate the Identity service database after you have successfully configured the Identity service database connection string (refer to Section 3.4.2, “Configure the Identity Service Database Connection”).

Procedure 3.6. Populating the Identity Service database

1. Log in to the system hosting the Identity service.
2. Use the **su** command to switch to the **keystone** user and run the **keystone-manage db_sync** command to initialize and populate the database identified in **/etc/keystone/keystone.conf**.

```
# su keystone -s /bin/sh -c "keystone-manage db_sync"
```

The Identity service database has been initialized and populated.

3.5. Start the Identity Service

All steps in this procedure must be performed while logged in to the server hosting the Identity service as the **root** user.

Procedure 3.7. Launching the Identity Service (for Red Hat Enterprise Linux 6-based systems)

1. Use the **service** command to start the **openstack-keystone** service.

```
# service openstack-keystone start
```

2. Use the **chkconfig** command to ensure that the **openstack-keystone** service will be started automatically in the future.

```
# chkconfig openstack-keystone on
```

Procedure 3.8. Launching the Identity Service (for Red Hat Enterprise Linux 7-based systems)

1. Use the **systemctl** command to start the **openstack-keystone** service.

```
# systemctl start openstack-keystone.service
```

2. Use the **systemctl** command to ensure that the **openstack-keystone** service will be started automatically in the future.

```
# systemctl enable openstack-keystone
```

The **openstack-keystone** service has been started.

3.6. Create the Identity Service Endpoint

Once the Identity service has been started its API endpoint must be defined. Some OpenStack services including the dashboard will not work unless this record is present.

All steps listed in this procedure must be performed while logged in to the Identity server as the **root** user.

Procedure 3.9. Creating the Identity Service Endpoint

1. **Set the OS_SERVICE_TOKEN Environment Variable**

   Set the **OS_SERVICE_TOKEN** environment variable to the administration token. This is done by reading the token file created when setting the administration token.

```
# export OS_SERVICE_TOKEN=`cat ~/ks_admin_token`
```

2. **Set the OS_SERVICE_ENDPOINT Environment Variable**

   Set the **OS_SERVICE_ENDPOINT** environment variable to point to the server hosting the Identity service.

```
# export OS_SERVICE_ENDPOINT="http://IP:35357/v2.0"
```

   Replace *IP* with the IP address or host name of your Identity server.

3. **Create a Service Entry**

   Create a service entry for the Identity service using the **keystone service-create** command.

```
# keystone service-create --name=keystone --type=identity \
       --description="Keystone Identity service"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |     Keystone Identity service    |
| id          | a8bff1db381f4751bd8ac126464511ae |
| name        |             keystone             |
| type        |             identity             |
+-------------+----------------------------------+
```

4. **Create an Endpoint for the API**

   Create an endpoint entry for the v2.0 API Identity service using the **keystone endpoint-create** command.

```
# keystone endpoint-create \
        --service keystone \
        --publicurl 'https://IP:443/keystone/main' \
        --adminurl 'https://IP:443/keystone/admin' \
        --internalurl 'http://IP:5000/v2.0'
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| adminurl    |       https://IP:443/keystone/admin       |
| id          | 1295011fdc874a838f702518e95a0e13 |
| internalurl |       http://IP:5000/v2.0        |
| publicurl   |       https://IP:443/keystone/main        |
| region      |             regionOne            |
| service_id  |                ID                |
+-------------+----------------------------------+
```

Replace *IP* with the IP address or host name of the Identity server.

Note

By default, the endpoint is created in the default region, **regionOne**. If you need to specify a different region when creating an endpoint use the ***--region*** argument to provide it.

The Identity service and endpoint entry has been created. The final step in Identity service configuration is the creation of the default user accounts, roles, and tenants.

3.6.1. Service Regions

Each service cataloged in the Identity service is identified by its region, which typically represents a geographical location, and its endpoint. In a cloud with multiple Compute deployments, regions allow for the discrete separation of services, and are a robust way to share some infrastructure between Compute installations, while allowing for a high degree of failure tolerance.

Administrators determine which services are shared between regions and which services are used only with a specific region. By default when an endpoint is defined and no region is specified it is created in the region named **regionOne**.

To begin using separate regions specify the ***--region*** argument when adding service endpoints.

```
$ keystone endpoint-create --region REGION \
  --service SERVICENAME\
  --publicurl PUBLICURL
  --adminurl ADMINURL
  --internalurl INTERNALURL
```

Replace *REGION* with the name of the region that the endpoint belongs to. When sharing an endpoint between regions create an endpoint entry containing the same URLs for each applicable region. For information on setting the URLs for each service refer to the Identity service configuration information of the service in question.

Example 3.1. Endpoints within Discrete Regions

In this example the **APAC** and **EMEA** regions share an Identity server (**identity.example.com**) endpoint while providing region specific compute API endpoints.

```
$ keystone endpoint-list
+---------+--------+-------------------------------------------------
----+
|   id    | region |                      publicurl
|
+---------+--------+-------------------------------------------------
----+
| 0d8b... |  APAC  |         http://identity.example.com:5000/v3
|
| 769f... |  EMEA  |         http://identity.example.com:5000/v3
|
| 516c... |  APAC  |  http://nova-
apac.example.com:8774/v2/$(tenant_id)s  |
| cf7e... |  EMEA  |  http://nova-
emea.example.com:8774/v2/$(tenant_id)s  |
+---------+--------+-------------------------------------------------
----+
```

3.7. Create an Administrator Account

Executing the following procedure will result in the creation of an administrative user as well as an associated tenant and role.

The steps listed in this procedure must be performed while logged in to the system hosting the Identity service as a user who has access to a file containing the administration token.

Procedure 3.10. Creating an Administrator account

1. Set the **OS_SERVICE_TOKEN** environment variable to the value of the administration token. This is done by reading the token file created when setting the administration token:

   ```
   # export OS_SERVICE_TOKEN=`cat ~/ks_admin_token`
   ```

2. Set the **OS_SERVICE_ENDPOINT** environment variable to point to the server hosting the Identity service:

   ```
   # export OS_SERVICE_ENDPOINT="http://IP:35357/v2.0"
   ```

   Replace *IP* with the IP address or host name of your Identity server.

3. Use the **keystone user-create** command to create an **admin** user:

   ```
   # keystone user-create --name admin --pass PASSWORD
   +----------+----------------------------------+
   | Property |              Value               |
   +----------+----------------------------------+
   | email    |                                  |
   | enabled  |              True                |
   ```

```
| id       | 94d659c3c9534095aba5f8475c87091a  |
| name     |              admin                |
| tenantId |                                   |
+----------+-----------------------------------+
```

Replace *PASSWORD* with a secure password for the account.

4. Use the **keystone role-create** command to create an **admin** role:

```
# keystone role-create --name admin
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
| id       | 78035c5d3cd94e62812d6d37551ecd6a |
| name     |              admin               |
+----------+----------------------------------+
```

5. Use the **keystone tenant-create** command to create an **admin** tenant:

```
# keystone tenant-create --name admin
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |                                  |
| enabled     |              True                |
| id          | 6f8e3e36c4194b86b9a9b55d4b722af3 |
| name        |              admin               |
+-------------+----------------------------------+
```

6. Now that the user account, role, and tenant have been created, the relationship between them must be explicitly defined using the **keystone user-role-add** command:

```
# keystone user-role-add --user admin --role admin --tenant admin
```

7. The newly created **admin** account will be used for future management of the Identity service. To facilitate authentication, create a **keystonerc_admin** file in a secure location such as the home directory of the **root** user.

   Add these lines to the file to set the environment variables that will be used for authentication:

```
unset OS_SERVICE_TOKEN
unset OS_SERVICE_ENDPOINT
export OS_USERNAME=admin
export OS_TENANT_NAME=admin
export OS_PASSWORD=PASSWORD
export OS_AUTH_URL=http://IP:35357/v2.0/
export PS1='[\u@\h \W(keystone_admin)]\$ '
```

   Replace *PASSWORD* with the password of the **admin** user and replace *IP* with the IP address or host name of the Identity server.

8. Run the **source** command on the file to load the environment variables used for authentication:

```
# source ~/keystonerc_admin
```

An administration user account, role, and tenant have been defined in the Identity server. The **keystonerc_admin** file has also been created for authenticating as the **admin** user.

3.8. Create a Regular User Account

Executing the following procedure will result in the creation of a regular user and tenant, and associating that user with Identity's default *_member_* role.

The steps listed in this procedure must be performed while logged in to the system hosting the Identity service as a user that has access to a file containing the administration token.

Procedure 3.11. Creating a regular user account

1. Load identity credentials from the **~/keystonerc_admin** file that was generated when the administrative user was created:

```
# source ~/keystonerc_admin
```

2. Use the **keystone user-create** to create a regular user:

```
# keystone user-create --name USER --pass PASSWORD
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
| email    |                                  |
| enabled  |              True                |
| id       | b8275d7494dd4c9cb3f69967a11f9765 |
| name     |              USER                |
| tenantId |                                  |
+----------+----------------------------------+
```

Replace *USER* with the user name that you would like to use for the account. Replace *PASSWORD* with a secure password for the account.

3. Use the **keystone tenant-create** command to create a tenant:

```
# keystone tenant-create --name TENANT
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |                                  |
| enabled     |              True                |
| id          | 6f8e3e36c4194b86b9a9b55d4b722af3 |
| name        |              TENANT              |
+-------------+----------------------------------+
```

Replace *TENANT* with the name that you wish to give to the tenant.

4. Now that the user account and tenant have been created, the relationship between them and the default *_member_* role must be explicitly defined using the **keystone user-role-add** command:

```
# keystone user-role-add --user USER --role _member_ --tenant
TENANT
```

where:

- *USER* is the same user name specified earlier during user creation.
- *TENANT* is the same tenant name specified earlier during tenant creation.

5. To facilitate authentication, create a **keystonerc_user** file in a secure location (for example, the home directory of the **root** user).

   Set these environment variables that will be used for authentication:

```
export OS_USERNAME=USER
export OS_TENANT_NAME=TENANT
export OS_PASSWORD=PASSWORD
export OS_AUTH_URL=http://IP:5000/v2.0/
export PS1='[\u@\h \W(keystone_user)]\$ '
```

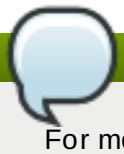
   where:

   - *PASSWORD* is the same password specified earlier during user creation.
   - *IP* is the IP address or host name of the Identity server.

A regular user account and tenant have been defined in the Identity server, and associated with the default **_member_** role. A **keystonerc_user** file has also been created for authenticating as the created user.

3.9. Create the Services Tenant

Tenants are used to aggregate service resources (tenants are also known as projects). Per tenant, quota controls can be used to limit the numbers of resources.



Note

For more information about quotas, see the *View and manage quotas* section in the *Red Hat Enterprise Linux OpenStack Platform Administration User Guide*. This document is available from the following page:

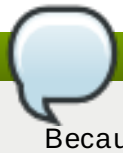
https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux_OpenStack_Platform

Each user is assigned to a tenant. For regular users, their tenant typically represents their group, project, or organisation. For service users (the entity accessing the Identity service on behalf of the service), the tenant represents a service's geographical region. This means that if your cloud's services are:

- Distributed, then typically one service tenant is created for each endpoint on which services are running (excepting the Identity and dashboard services).

- Deployed on a single node, then only one service tenant is required (but of course this is just one option; more can be created for administrative purposes).

The service setup examples in the *Deploying OpenStack: Learning Environments (Manual Set Up)* guide assume that all services are deployed on one node, therefore only one service tenant is required. All such examples use the **services** tenant.



Note

Because administrators, regular users, and service users all need a tenant, at least three tenants are typically created, one for each group. To create administrative and regular users and tenants, see Section 3.7, “Create an Administrator Account” and Section 3.8, “Create a Regular User Account”.

To create the **services** tenant:

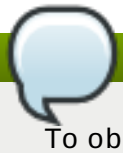
Procedure 3.12. Creating the services tenant

1. Run the **source** command on the file containing the environment variables used to identify the Identity service administrator.

```
# source ~/keystonerc_admin
```

2. Create the **services** tenant in the Identity service:

```
# keystone tenant-create --name services --description "Services
Tenant"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |         Services Tenant          |
| enabled     |              True                |
| id          | 7e193e36c4194b86b9a9b55d4b722af3 |
| name        |             services             |
+-------------+----------------------------------+
```



Note

To obtain a list of all Identity service tenants and their IDs, execute:

```
# keystone tenant-list
```

3.10. Validate the Identity Service Installation

Follow the steps outlined in this procedure to verify that an Identity service installation is functioning correctly. These steps must be performed while logged in to either the Identity server or another system.

The logged in user must have access to **keystonerc_admin** and **keystonerc_user** files containing the environment variables required to authenticate as the administrator user and a regular user respectively. Also, the system must have the following already installed: httpd, mod_wsgi, and mod_ssl (for security purposes).

Procedure 3.13. Validating the Identity Service installation

1. Run the **source** command on the file containing the environment variables used to identify the Identity service administrator.

```
# source ~/keystonerc_admin
```

2. Run the **keystone user-list** command to authenticate with the Identity service and list the users defined in the system.

```
# keystone user-list
+----------------------------------+--------+---------+-----------
-------+
|                id                |  name  | enabled |      email
|
+----------------------------------+--------+---------+-----------
-------+
| 94d659c3c9534095aba5f8475c87091a | admin  |   True  |
|
| b8275d7494dd4c9cb3f69967a11f9765 |  USER  |   True  |
|
+----------------------------------+--------+---------+-----------
-------+
```

The list of users defined in the system is displayed. If the list is not displayed then there is an issue with the installation.

a. If the message returned indicates a permissions or authorization issue then check that the administrator user account, tenant, and role were created properly. Also ensure that the three objects are linked correctly.

```
Unable to communicate with identity service: {"error":
{"message": "You are not authorized to perform the requested
action: admin_required", "code": 403, "title": "Not
Authorized"}}. (HTTP 403)
```

b. If the message returned indicates a connectivity issue then verify that the **openstack-keystone** service is running and that the firewall service is configured to allow connections on ports **5000** and **35357**.

```
Authorization Failed: [Errno 111] Connection refused
```

3. Run the **source** command on the file containing the environment variables used to identify the regular Identity service user.

```
# source ~/keystonerc_user
```

4. Run the **keystone user-list** command to authenticate with the Identity service and list the users defined in the system.

```
# keystone user-list
Unable to communicate with identity service: {"error": {"message":
"You are not authorized to perform the requested action:
admin_required", "code": 403, "title": "Not Authorized"}}. (HTTP
403)
```

An error message is displayed indicating that the user is **Not Authorized** to run the command. If the error message is not displayed but instead the user list appears then the regular user account was incorrectly attached to the **admin** role.

5. Run the **keystone token-get** command to verify that the regular user account is able to run commands that it is authorized to access.

```
# keystone token-get
+-----------+----------------------------------+
|  Property |              Value               |
+-----------+----------------------------------+
|  expires  |       2013-05-07T13:00:24Z       |
|     id    | 5f6e089b24d94b198c877c58229f2067 |
| tenant_id | f7e8628768f2437587651ab959fbe239 |
|  user_id  | 8109f0e3deaf46d5990674443dcf7db7 |
+-----------+----------------------------------+
```

The Identity service is installed and functioning correctly.

3.10.1. Troubleshoot Identity Client (keystone) Connectivity Problems

When the Identity client (**keystone**) is unable to contact the Identity service it returns an error:

```
Unable to communicate with identity service: [Errno 113] No route to
host. (HTTP 400)
```

To debug the issue check for these common causes:

Identity service is down

On the system hosting the Identity service check the service status:

```
# openstack-status | grep keystone
openstack-keystone:                     active
```

If the service is not running then log in as the **root** user and start it.

```
# service openstack-keystone start
```

Firewall is not configured properly

The firewall might not be configured to allow TCP traffic on ports **5000** and **35357**. If so, refer to Section 3.4.4, “Configure the Firewall to Allow Identity Service Traffic” for instructions on how to correct this.

Service Endpoints not defined correctly

On the system hosting the Identity service check that the endpoints are defined correctly.

Procedure 3.14. Verifying Identity Service endpoints

1. Obtain the administration token:

```
$ grep admin_token /etc/keystone/keystone.conf
admin_token = 0292d404a88c4f269383ff28a3839ab4
```

2. Determine the correct administration endpoint for the Identity service:

```
http://IP:35357/VERSION
```

Replace *IP* with the IP address or host name of the system hosting the Identity service. Replace *VERSION* with the API version (**v2.0**, or **v3**) that is in use.

3. Unset any pre-defined Identity service related environment variables:

```
$ unset OS_USERNAME OS_TENANT_NAME OS_PASSWORD
OS_AUTH_URL
```

4. Use the administration token and endpoint to authenticate with the Identity service. Confirm that the Identity service endpoint is correct:

```
$ keystone --os-token=TOKEN \
           --os-endpoint=ENDPOINT \
           endpoint-list
```

Verify that the listed **publicurl**, **internalurl**, and **adminurl** for the Identity service are correct. In particular ensure that the IP addresses and port numbers listed within each endpoint are correct and reachable over the network.

If these values are incorrect then refer to Section 3.6, “Create the Identity Service Endpoint” for information on adding the correct endpoint. Once the correct endpoints have been added, remove any incorrect endpoints using the **endpoint-delete** action of the **keystone** command.

```
$ keystone --os-token=TOKEN \
           --os-endpoint=ENDPOINT \
           endpoint-delete ID
```

Replace *TOKEN* and *ENDPOINT* with the values identified previously. Replace *ID* with the identity of the endpoint to remove as listed by the **endpoint-list** action.

Chapter 4. OpenStack Object Storage Service Installation

4.1. Services that Make Up the Object Storage Service

The Object Storage Service is made up of 4 services that work together to manage the storage of data objects.

Proxy Service

The proxy service uses the object ring to decide where to direct newly uploaded objects. It updates the relevant container database to reflect the presence of a new object. If a newly uploaded object goes to a new container, the proxy service updates the relevant account database to reflect the new container.

The proxy service also directs get requests to one of the nodes where a replica of the requested object is stored, either randomly, or based on response time from the node.

Object Service

The object service is responsible for storing data objects in partitions on disk devices. Each partition is a directory. Each object is held in a subdirectory of its partition directory. A MD5 hash of the path to the object is used to identify the object itself.

Container Service

The container service maintains databases of objects in containers. There is one database file for each container, and the database files are replicated across the cluster. Containers are defined when objects are put in them. Containers make finding objects faster by limiting object listings to specific container namespaces.

Account Service

The account service maintains databases of all of the containers accessible by any given account. There is one database file for each account, and the database files are replicated across the cluster. Any account has access to a particular group of containers. An account maps to a tenant in the Identity service.

Load Balancer

The load balancer is used primarily by the Object Storage service to terminate SSL connections. Almost any load balancing service for HTTP can be used; it only needs to allow additional HTTP requests. However, using **`nginx`** is not recommended because it stores entire bodies of transferred objects, resulting in poor latency.

4.2. Architecture of the Object Storage Service

The OpenStack Object Storage service is a modular group of services, including **openstack-swift-proxy**, **openstack-swift-object**, **openstack-swift-container**, and **openstack-swift-account**.

All of the services can be installed on each node. Alternatively, services can be run on dedicated nodes.

Common Object Storage Service Deployment Configurations

All services on all nodes.

Simplest to set up.

Dedicated proxy nodes, all other services combined on other nodes.

The proxy service is CPU and I/O intensive. The other services are disk and I/O intensive. This configuration allows you to optimize your hardware usage.

Dedicated proxy nodes, dedicated object service nodes, container and account services combined on other nodes.

The proxy service is CPU and I/O intensive. The container and account services are more disk and I/O intensive than the object service. This configuration allows you to optimize your hardware usage even more.

The following diagram provides an overview of the third option, where the proxy and object nodes are split out from those containing the container and account services:

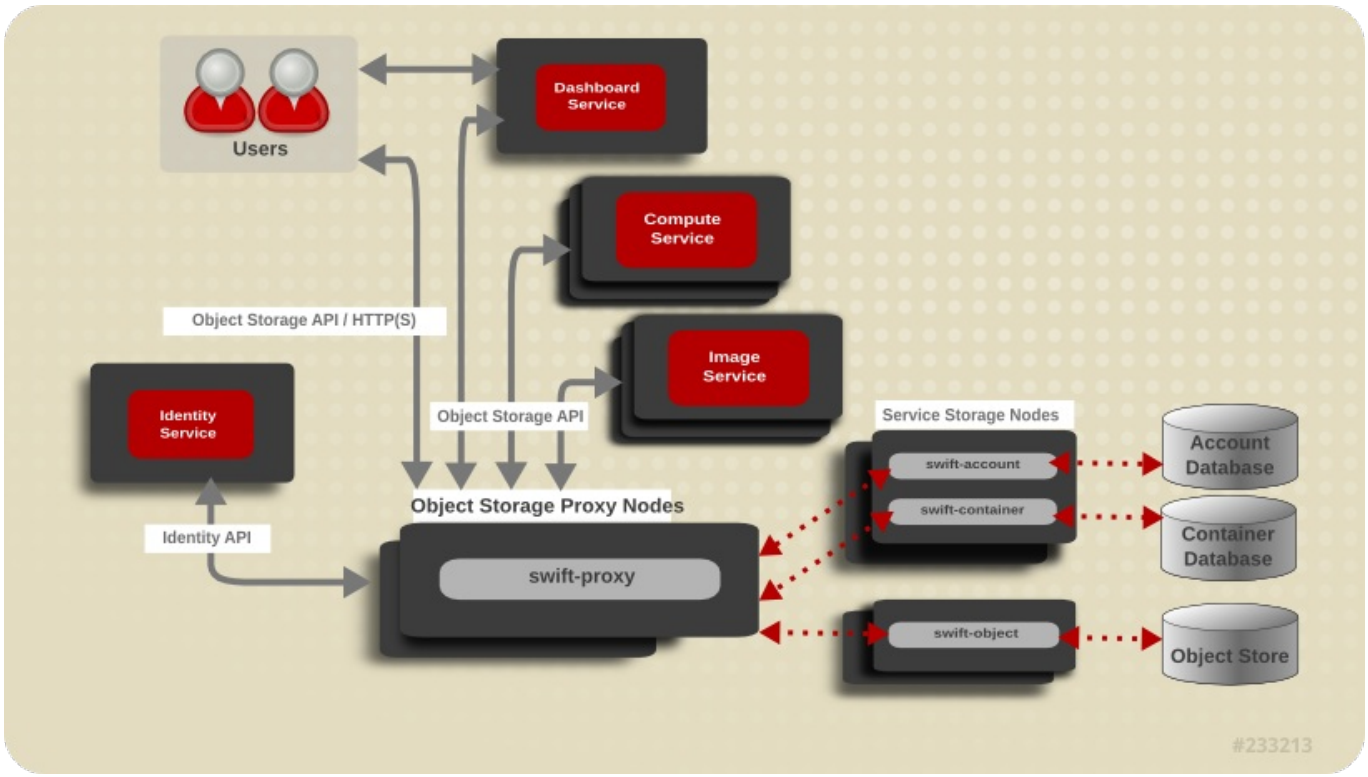


Figure 4.1. Service Architecture

4.3. Object Storage Service Requirements

Supported Filesystems

The Object Storage Service stores objects in filesystems. Currently, **XFS** and **ext4** are supported. Your filesystem must be mounted with *Extended Attributes* (**xattr**) enabled.

We recommend **XFS**. You can configure this in **/etc/fstab**:

Example 4.1. Sample /etc/fstab Entry for One XFS Storage Disk

```
/dev/sdb1 /srv/node/d1 xfs inode64,noatime,nodiratime 0 0
```

Note

Extended Attributes are already enabled on **XFS** by default. As such, you no longer need to specify **user_xattr** in your **/etc/fstab** entry.

Acceptable Mountpoints

The Object Storage service expects devices to be mounted at **/srv/node/**.

4.4. Configure Rsyncd

To ensure replication, you must set up **rsyncd** for your filesystems before you install and configure Object Storage. The following procedure assumes that at least two XFS storage disks have been mounted on each storage node. Example **/etc/fstab**:

Example 4.2. Sample /etc/fstab Entry for Two XFS Storage Disks

```
/dev/sdb1 /srv/node/d1 xfs inode64,noatime,nodiratime 0 0
/dev/sdb2 /srv/node/d2 xfs inode64,noatime,nodiratime 0 0
```

On each storage node:

1. Copy addresses from the controller's **/etc/hosts** file, and add storage node IP addresses (ensure all nodes have all addresses in their **/etc/hosts** file as well).
2. Install the rsync and xinetd packages:

```
# yum install rsync xineted
```

3. Put the following in **/etc/rsync.conf**:

```
##assumes 'swift' has been used as the Object Storage user/group
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
##address on which the rsync daemon listens
address = LOCAL_MGT_NETWORK_IP

[account]
max connections = 2
path = /srv/node/
read only = false
write only      = no
list            = yes
incoming chmod  = 0644
outgoing chmod  = 0644
lock file = /var/lock/account.lock

[container]
max connections = 2
```

```
path = /srv/node/
read only = false
write only      = no
list            = yes
incoming chmod  = 0644
outgoing chmod  = 0644
lock file = /var/lock/container.lock

[object]
max connections = 2
path = /srv/node/
read only = false
write only      = no
list            = yes
incoming chmod  = 0644
outgoing chmod  = 0644
lock file = /var/lock/object.lock
```

Note

Multiple account, container and object sections can be used.

4. Add the following to **/etc/xinetd.d/rsync**:

```
service rsync
 {
     port            = 873
     disable         = no
     socket_type     = stream
     protocol        = tcp
     wait            = no
     user            = root
     group           = root
     groups          = yes
     server          = /usr/bin/rsync
     bind            = LOCAL_MGT_NETWORK_IP
     server_args = --daemon --config /etc/rsync.conf
 }
```

5. Enable xinetd, and start the service:

```
# systemctl enable xinetd.service
# systemctl start xinetd.service
```

4.5. Install the Object Storage Service Packages

The OpenStack Object Storage service is packaged in the following packages.

Primary OpenStack Object Storage packages

openstack-swift-proxy

Proxies requests for objects.

openstack-swift-object

Stores data objects of up to 5GB.

openstack-swift-container

Maintains a database that tracks all of the objects in each container.

openstack-swift-account

Maintains a database that tracks all of the containers in each account.

OpenStack Object Storage dependencies

openstack-swift

Contains code common to the specific services.

openstack-swift-plugin-swift3

The swift3 plugin for OpenStack Object Storage.

memcached

Soft dependency of the proxy server, caches authenticated clients rather than making them reauthorize at every interaction.

openstack-utils

Provides utilities for configuring OpenStack.

Procedure 4.1. Installing the Object Storage Service packages

- Install the required packages using the **yum** command as the root user:

```
# yum install -y openstack-swift-proxy \
  openstack-swift-object \
  openstack-swift-container \
  openstack-swift-account \
  openstack-utils \
  memcached
```

The services that make up the OpenStack Object Storage service are installed and ready to be configured.

4.6. Configure the Object Storage Service

4.6.1. Create the Object Storage Service Identity Records

This section assumes that you have already created an administrator account and **services** tenant. For more information, refer to:

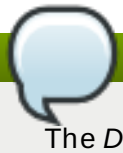
- Section 3.7, “Create an Administrator Account”
- Section 3.9, “Create the Services Tenant”

In this procedure, you will:

1. Create the **swift** user, who has the **admin** role in the **services** tenant.
2. Create the **swift** service entry and assign it an endpoint.

In order to proceed, you should have already performed the following (using the Identity service):

1. Created an Administrator role named **admin** (refer to Section 3.7, "Create an Administrator Account" for instructions)
2. Created the **services** tenant (refer to Section 3.9, "Create the Services Tenant" for instructions)



Note

The *Deploying OpenStack: Learning Environments (Manual Set Up)* guide uses one tenant for all service users. For more information, refer to Section 3.9, "Create the Services Tenant".

You can perform this procedure from your Identity service server or on any machine where you've copied the **keystonerc_admin** file (which contains administrator credentials) and the **keystone** command-line utility is installed.

Procedure 4.2. Configuring the Object Storage Service to authenticate through the Identity Service

1. Set up the shell to access Keystone as the admin user:

```
# source ~/keystonerc_admin
```

2. Create the **swift** user and set its password by replacing *PASSWORD* with your chosen password:

```
# keystone user-create --name swift --pass PASSWORD
```

3. Add the **swift** user to the **services** tenant with the **admin** role:

```
# keystone user-role-add --user swift --role admin --tenant
services
```

4. Create the **swift** Object Storage service entry:

```
# keystone service-create --name swift --type object-store \
   --description "Swift Storage Service"
```

5. Create the **swift** endpoint entry:

```
# keystone endpoint-create \
   --service swift \
   --publicurl "http://IP:8080/v1/AUTH_\$(tenant_id)s" \
   --adminurl "http://IP:8080/v1" \
   --internalurl "http://IP:8080/v1/AUTH_\$(tenant_id)s"
```

Replace *IP* with the IP address or fully qualified domain name of the system hosting the Object Storage Proxy service.

You have configured the Identity service to work with the Object Storage service.

4.6.2. Configure the Object Storage Service Storage Nodes

The Object Storage Service stores objects on the filesystem, usually on a number of connected physical storage devices. All of the devices which will be used for object storage must be formatted **ext4** or **XFS**, and mounted under the **/srv/node/** directory. All of the services that will run on a given node must be enabled, and their ports opened.

While you can run the proxy service alongside the other services, the proxy service is not covered in this procedure.

Procedure 4.3. Configuring the Object Storage Service storage nodes

1. Format your devices using the **ext4** or **XFS** filesystem. Make sure that **xattr**s are enabled.
2. Add your devices to the **/etc/fstab** file to ensure that they are mounted under **/srv/node/** at boot time.

   Use the **blkid** command to find your device's unique ID, and mount the device using its unique ID.

   Note

   If using **ext4**, ensure that extended attributes are enabled by mounting the filesystem with the **user_xattr** option. (In **XFS**, extended attributes are enabled by default.)

3. Configure the firewall to open the TCP ports used by each service running on each node.

   By default, the account service uses port 6202, the container service uses port 6201, and the object service uses port 6200.

   a. Open the **/etc/sysconfig/iptables** file in a text editor.

   b. Add an **INPUT** rule allowing TCP traffic on the ports used by the account, container, and object service. The new rule must appear before any **reject-with icmp-host-prohibited** rule.

   ```
   -A INPUT -p tcp -m multiport --dports 6200,6201,6202,873 -j
   ACCEPT
   ```

   c. Save the changes to the **/etc/sysconfig/iptables** file.

   d. Restart the **iptables** service for the firewall changes to take effect.

   ```
   # service iptables restart
   ```

4. Change the owner of the contents of **/srv/node/** to **swift:swift** with the **chown** command.

   ```
   # chown -R swift:swift /srv/node/
   ```

5. Set the **SELinux** context correctly for all directories under **/srv/node/** with the **restorcon** command.

```
# restorecon -R /srv
```

6. Use the **openstack-config** command to add a hash prefix (swift_hash_path_prefix) to your **/etc/swift/swift.conf**:

```
# openstack-config --set /etc/swift/swift.conf swift-hash
swift_hash_path_prefix \
             $(openssl rand -hex 10)
```

7. Use the **openstack-config** command to add a hash suffix (swift_hash_path_suffix) to your **/etc/swift/swift.conf**:

```
# openstack-config --set /etc/swift/swift.conf swift-hash
swift_hash_path_suffix \
             $(openssl rand -hex 10)
```

These details are required for finding and placing data on all of your nodes. Back **/etc/swift/swift.conf** up.

8. Use the **openstack-config** command to set the IP address your storage services will listen on. Run these commands for every service on every node in your Object Storage cluster.

```
# openstack-config --set /etc/swift/object-server.conf \
  DEFAULT bind_ip node_ip_address
# openstack-config --set /etc/swift/account-server.conf \
  DEFAULT bind_ip node_ip_address
# openstack-config --set /etc/swift/container-server.conf \
  DEFAULT bind_ip node_ip_address
```

The **DEFAULT** argument specifies the **DEFAULT** section of the service configuration file. Replace *node_ip_address* with the IP address of the node you are configuring.

9. Copy **/etc/swift/swift.conf** from the node you are currently configuring, to all of your Object Storage Service nodes.

Important

The **/etc/swift/swift.conf** file must be identical on all of your Object Storage Service nodes.

10. Start the services which will run on your node.

```
# service openstack-swift-account start
# service openstack-swift-container start
# service openstack-swift-object start
```

11. Use the **chkconfig** command to make sure the services automatically start at boot time.

```
# chkconfig openstack-swift-account on
# chkconfig openstack-swift-container on
# chkconfig openstack-swift-object on
```

All of the devices that your node will provide as object storage are formatted and mounted under **/srv/node/**. Any service running on the node has been enabled, and any ports used by services on the node have been opened.

4.6.3. Configure the Object Storage Service Proxy Service

The Object Storage proxy service determines to which node **gets** and **puts** are directed.

Although you can run the account, container, and object services alongside the proxy service, only the proxy service is covered in the following procedure.

Note

Because the SSL capability built into the Object Storage service is intended primarily for testing, it is not recommended for use in production. In a production cluster, Red Hat recommends that you use the load balancer to terminate SSL connections.

Procedure 4.4. Configuring the Object Storage Service proxy service

1. Update the configuration file for the proxy server with the correct authentication details for the appropriate service user:

```
# openstack-config --set /etc/swift/proxy-server.conf \
    filter:authtoken auth_host IP
# openstack-config --set /etc/swift/proxy-server.conf \
    filter:authtoken admin_tenant_name services
# openstack-config --set /etc/swift/proxy-server.conf \
    filter:authtoken admin_user swift
# openstack-config --set /etc/swift/proxy-server.conf \
    filter:authtoken admin_password PASSWORD
```

   Where:

   - *IP* - The IP address or host name of the Identity server.
   - *services* - The name of the tenant that was created for the use of the Object Storage service (previous examples set this to **services**).
   - *swift* - The name of the service user that was created for the Object Storage service (previous examples set this to **swift**).
   - *PASSWORD* - The password associated with the service user.

2. Start the **memcached** and **openstack-swift-proxy** services using the **service** command:

```
# service memcached start
# service openstack-swift-proxy start
```

3. Enable the **memcached** and **openstack-swift-proxy** services permanently using the **chkconfig** command:

```
# chkconfig memcached on
# chkconfig openstack-swift-proxy on
```

4. Allow incoming connections to the Swift proxy server by adding this firewall rule to the **/etc/sysconfig/iptables** configuration file:

```
-A INPUT -p tcp -m multiport --dports 8080 -j ACCEPT
```

Important

This rule allows communication from all remote hosts to the system hosting the Swift proxy on port **8080**. For information regarding the creation of more restrictive firewall rules refer to the *Red Hat Enterprise Linux Security Guide* from the following link:

https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/

5. Use the **service** command to restart the **iptables** service for the new rule to take effect:

```
# service iptables save
# service iptables restart
```

The Object Storage Service proxy service is now listening for HTTP put and get requests on port 8080, and directing them to the appropriate nodes.

4.6.4. Object Storage Service Rings

Rings determine where data is stored in a cluster of storage nodes. Ring files are generated using the **swift-ring-builder** tool. Three ring files are required, one each for the **object**, **container**, and **account** services.

Each storage device in a cluster is divided into partitions, with a recommended minimum of 100 partitions per device. Each partition is physically a directory on disk. A configurable number of bits from the MD5 hash of the filesystem path to the partition directory, known as the *partition power*, is used as a partition index for the device. The *partition count* of a cluster that has 1000 devices, where each device has 100 partitions on it, is 100 000.

The partition count is used to calculate the partition power, where 2 to the partition power is the partition count. If the partition power is a fraction, it is rounded up. If the partition count is 100 000, the part power is 17 (16.610 rounded up). This can be expressed mathematically as: 2 ^ partition power = partition count.

4.6.5. Build Object Storage Service Ring Files

Three ring files need to be created: one to track the objects stored by the Object Storage Service, one to track the containers in which objects are placed, and one to track which accounts can access which containers. The ring files are used to deduce where a particular piece of data is stored.

Ring files are generated using four possible parameters: partition power, replica count, zone, and the amount of time that must pass between partition reassignments.

Table 4.1. Parameters used when building ring files

Ring File Parameter	Description
part_power	2 ^ partition power = partition count. The partition is rounded up after calculation.
replica_count	The number of times that your data will be replicated in the cluster.
min_part_hours	Minimum number of hours before a partition can be moved. This parameter increases availability of data by not moving more than one copy of a given data item within that min_part_hours amount of time.
zone	Used when adding devices to rings (optional). Zones are a flexible abstraction, where each zone should be separated from other zones as possible in your deployment. You can use a zone to represent sites, cabinet, nodes, or even devices.

Procedure 4.5. Building Object Storage service ring files

1. Use the **swift-ring-builder** command to build one ring for each service. Provide a builder file, a *partition power*, a *replica count*, and the *minimum hours between partition re-assignment*:

```
# swift-ring-builder /etc/swift/object.builder create part_power
replica_count min_part_hours
# swift-ring-builder /etc/swift/container.builder create
part_power replica_count min_part_hours
# swift-ring-builder /etc/swift/account.builder create part_power
replica_count min_part_hours
```

2. When the rings are created, add devices to the account ring.

```
# swift-ring-builder /etc/swift/account.builder add
zX-SERVICE_IP:6202/dev_mountpt part_count
```

   Where:

   - *X* is the corresponding integer of a specified zone (for example, **z1** would correspond to Zone One).
   - *SERVICE_IP* is the IP on which the account, container, and object services should listen. This IP should match the **bind_ip** value set during the configuration of the Object Storage service Storage Nodes.
   - *dev_mountpt* is the **/srv/node** subdirectory under which your device is mounted.
   - *part_count* is the partition count you used to calculate your partition power.

   For example, if:

   - The account, container, and object services are configured to listen on **10.64.115.44** on Zone One,

- Your device is mounted on **/srv/node/accounts**, and
- You wish to set a partition count of 100.

Then run:

```
# swift-ring-builder /etc/swift/account.builder add z1-
10.64.115.44:6202/accounts 100
```

Note

Repeat this step for each device (on each node in the cluster) you want added to the ring.

3. In a similar fashion, add devices to both containers and objects rings:

```
# swift-ring-builder /etc/swift/container.builder add
zX-SERVICE_IP:6201/dev_mountpt part_count
```

```
# swift-ring-builder /etc/swift/object.builder add
zX-SERVICE_IP:6200/dev_mountpt part_count
```

Replace the variables herein with the same ones used in the previous step.

Note

Repeat these commands for each device (on each node in the cluster) you want added to the ring.

4. Distribute the partitions across the devices in the ring using the **swift-ring-builder** command's **rebalance** argument.

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

5. Check to see that you now have 3 ring files in the directory **/etc/swift**. The command:

```
# ls /etc/swift/*gz
```

should reveal:

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

6. Start the **openstack-swift-proxy** service:

```
# service openstack-swift-proxy start
```

7. Ensure that all files in the **/etc/swift/** directory including those that you have just created are owned by the **root** user and **swift** group.

Important

All mount points must be owned by **root**; all roots of mounted file systems must be owned by **swift**. Before running the following command, ensure that all devices are already mounted and owned by **root**.

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

8. Copy each ring builder file to each node in the cluster, storing them under **/etc/swift/**.

```
# scp /etc/swift/*.gz node_ip_address:/etc/swift
```

You have created rings for each of the services that require them. You have used the builder files to distribute partitions across the nodes in your cluster, and have copied the ring builder files themselves to each node in the cluster.

4.7. Validate the Object Storage Service Installation

After installing and configuring the Block Storage service, perform the following steps to validate it:

Procedure 4.6. Validating the Object Storage Service installation

1. On your proxy server node, use the **openstack-config** command to turn on debug level logging:

```
# openstack-config --set /etc/swift/proxy-server.conf DEFAULT
log_level debug
```

2. Set up the shell to access Keystone as the admin user:

```
$ source ~/keystonerc_admin
```

3. Use the **swift** list to make sure you can connect to your proxy server:

```
$ swift list
      Message from syslogd@example-swift-01 at Jun 14 02:46:00 ...
 135 proxy-server Server reports support for api versions: v3.0,
v2.0
```

4. Use the **swift** command to upload some files to your Object Storage Service nodes

```
$ head -c 1024 /dev/urandom > data1.file ; swift upload c1
data1.file
$ head -c 1024 /dev/urandom > data2.file ; swift upload c1
data2.file
$ head -c 1024 /dev/urandom > data3.file ; swift upload c1
data3.file
```

5. Use the **swift** command to take a listing of the objects held in your Object Storage Service cluster.

```
$ swift list
$ swift list c1
data1.file
data2.file
data3.file
```

You have now uploaded 3 files into 1 container. If you check the other storage devices, you will find more **.data** files, depending on your replica count.

```
$ find /srv/node/ -type f -name "*data"
```

Chapter 5. OpenStack Image Service Installation

5.1. Image Service Requirements

To install the Image service, you must have access to:

- MariaDB database server root credentials and IP address
- Identity service administrator credentials and endpoint URL

If using the OpenStack Object Storage service as the storage backend, you will also need to know the service's endpoint public URL. This endpoint is configured as part of Section 4.6.1, “Create the Object Storage Service Identity Records”.

5.2. Install the Image Service Packages

The OpenStack Image service requires the following packages:

openstack-glance

Provides the OpenStack Image service.

openstack-utils

Provides supporting utilities to assist with a number of tasks including the editing of configuration files.

openstack-selinux

Provides OpenStack specific SELinux policy modules.

To install all of the above packages, execute the following command while logged in as the **root** user.

```
# yum install -y openstack-glance openstack-utils openstack-selinux
```

The OpenStack Image service is now installed and ready to be configured.

5.3. Create the Image Service Database

In this procedure, the database and database user that will be used by the Image service will be created. These steps must be performed while logged in to the database server as the **root** user (or as a user with suitable access: **create db**, **create user**, **grant permissions**).

Procedure 5.1. Creating the Image Service database

1. Connect to the database service using the **mysql** command.

```
# mysql -u root -p
```

2. Create the **glance** database.

```
mysql> CREATE DATABASE glance;
```

3. Create a **glance** database user and grant it access to the **glance** database.

```
mysql> GRANT ALL ON glance.* TO 'glance'@'%' IDENTIFIED BY
'PASSWORD';
mysql> GRANT ALL ON glance.* TO 'glance'@'localhost' IDENTIFIED
BY 'PASSWORD';
```

   Replace *PASSWORD* with a secure password that will be used to authenticate with the database server as this user.

4. Flush the database privileges to ensure that they take effect immediately.

```
mysql> FLUSH PRIVILEGES;
```

5. Exit the **mysql** client.

```
mysql> quit
```

The Image Service database has been created. The database will be populated during service configuration.

5.4. Configuring the Image Service

5.4.1. Configuration Overview

To configure the Image service, the following must be completed:

- Configure TLS/SSL.
- Configure the Identity service for Image service authentication (create database entries, set connection strings, and update configuration files).
- Configure the disk-image storage backend (this guide uses the Object Storage service).
- Configure the firewall for Image service access.
- Populate the Image service database.

5.4.2. Create the Image Identity Records

This section outlines the steps for creating and configuring Identity service records required by the Image service.

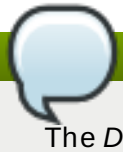
1. Create the **glance** user, who has the **admin** role in the **services** tenant.
2. Create the **glance** service entry and assign it an endpoint.

These entries assist other OpenStack services attempting to locate and access the volume functionality provided by the Image service.

In order to proceed, you should have already performed the following (through the Identity service):

1. Created an Administrator role named **admin** (refer to Section 3.7, “Create an Administrator Account” for instructions)

2. Created the **services** tenant (refer to Section 3.9, "Create the Services Tenant" for instructions)



Note

The *Deploying OpenStack: Learning Environments (Manual Set Up)* guide uses one tenant for all service users. For more information, refer to Section 3.9, "Create the Services Tenant".

You can perform this procedure from your Identity service host or on any machine where you've copied the **keystonerc_admin** file (which contains administrator credentials) and the **keystone** command-line utility is installed.

Procedure 5.2. Configuring the Image Service to authenticate through the Identity Service

1. Authenticate as the administrator of the Identity service by running the **source** command on the **keystonerc_admin** file:

```
# source ~/keystonerc_admin
```

2. Create a user named **glance** for the Image Service to use:

```
# keystone user-create --name glance --pass PASSWORD
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|  email   |                                  |
| enabled  |               True               |
|    id    | 8091eaf121b641bf84ce73c49269d2d1 |
|   name   |              glance              |
| tenantId |                                  |
+----------+----------------------------------+
```

Replace *PASSWORD* with a secure password that will be used by the Image Service when authenticating with the Identity service.

3. Use the **keystone user-role-add** command to link the **glance** user and the **admin** role together within the context of the **services** tenant:

```
# keystone user-role-add --user glance --role admin --tenant
services
```

4. Create the **glance** service entry:

```
# keystone service-create --name glance \
       --type image \
       --description "Glance Image Service"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |       Glance Image Service       |
```

```
|      id     | 7461b83f96bd497d852fb1b85d7037be |
|     name    |              glance              |
|     type    |               image              |
+-------------+----------------------------------+
```

5. Create the **glance** endpoint entry:

```
# keystone endpoint-create \
        --service glance \
        --publicurl "http://IP:9292" \
        --adminurl "http://IP:9292" \
        --internalurl "http://IP:9292"
```

Replace *IP* with the IP address or host name of the system hosting the Image Service.

All supporting Identity service entries required by the Image Service have been created.

5.4.3. Configure the Image Service Database Connection

The database connection string used by the Image service is defined in the **/etc/glance/glance-api.conf** and **/etc/glance/glance-registry.conf** files. It must be updated to point to a valid database server before starting the service.

All commands in this procedure must be run while logged in as the **root** user on the server hosting the Image service.

Procedure 5.3. Configuring the Image Service SQL database connection

1. Use the **openstack-config** command to set the value of the **sql_connection** configuration key in the **/etc/glance/glance-api.conf** file.

```
# openstack-config --set /etc/glance/glance-api.conf \
  DEFAULT sql_connection mysql://USER:PASS@IP/DB
```

Replace:

- *USER* with the database user name the Image service is to use, usually **glance**.
- *PASS* with the password of the chosen database user.
- *IP* with the IP address or host name of the database server.
- *DB* with the name of the database that has been created for use by the Image service, usually **glance**.

2. Use the **openstack-config** command to set the value of the **sql_connection** configuration key in the **/etc/glance/glance-registry.conf** file.

```
# openstack-config --set /etc/glance/glance-registry.conf \
  DEFAULT sql_connection mysql://USER:PASS@IP/DB
```

Replace the placeholder values *USER*, *PASS*, *IP*, and *DB* with the same values used in the previous step.

Important

The IP address or host name specified in the connection configuration key must match the IP address or host name to which the glance database user was granted access when creating the glance database. Moreover, if the database is hosted locally and you granted permissions to 'localhost' when creating the glance database, you must enter 'localhost'.

The database connection string has been set and will be used by the Image service.

5.4.4. Configure Image Service Authentication

To update the Image configuration files for Identity usage, execute the following commands as the **root** user on each node hosting the Image service:

Procedure 5.4. Configuring the Image Service to authenticate through the Identity Service

1. Configure the **glance-api** service:

```
# openstack-config --set /etc/glance/glance-api.conf \
  paste_deploy flavor keystone
# openstack-config --set /etc/glance/glance-api.conf \
  keystone_authtoken auth_host IP
# openstack-config --set /etc/glance/glance-api.conf \
  keystone_authtoken auth_port 35357
# openstack-config --set /etc/glance/glance-api.conf \
  keystone_authtoken auth_protocol http
# openstack-config --set /etc/glance/glance-api.conf \
  keystone_authtoken admin_tenant_name services
# openstack-config --set /etc/glance/glance-api.conf \
  keystone_authtoken admin_user glance
# openstack-config --set /etc/glance/glance-api.conf \
  keystone_authtoken admin_password PASSWORD
```

2. Configure the **glance-registry** service:

```
  # openstack-config --set /etc/glance/glance-registry.conf \
  paste_deploy flavor keystone
# openstack-config --set /etc/glance/glance-registry.conf \
  keystone_authtoken auth_host IP
# openstack-config --set /etc/glance/glance-registry.conf \
  keystone_authtoken auth_port 35357
# openstack-config --set /etc/glance/glance-registry.conf \
  keystone_authtoken auth_protocol http
# openstack-config --set /etc/glance/glance-registry.conf \
  keystone_authtoken admin_tenant_name services
# openstack-config --set /etc/glance/glance-registry.conf \
  keystone_authtoken admin_user glance
# openstack-config --set /etc/glance/glance-registry.conf \
  keystone_authtoken admin_password PASSWORD
```

where:

- *IP* - The IP address or host name of the Identity server.
- *services* - The name of the tenant that was created for the use of the Image service (previous examples set this to **services**).
- *glance* - The name of the service user that was created for the Image service (previous examples set this to **glance**).
- *PASSWORD* - The password associated with the service user.

5.4.5. Using the Object Storage Service for Image Storage

By default, the Image service uses the local file system (**file**) for its storage backend. However, either of the following storage backends can be used to store uploaded disk images:

- **file** - Local file system of the Image server (**/var/lib/glance/images/** directory)
- **swift** - OpenStack Object Storage service

Note

The configuration procedure below uses the **openstack-config** command. However, the **/etc/glance/glance-api.conf** file can also be manually updated. If manually updating the file:

1. Ensure that the **default_store** parameter is set to the correct backend (for example, '**default_store=rbd**').
2. Update the parameters in that backend's section (for example, under '**RBD Store Options**').

To change the configuration to use the Object Storage service, execute the following steps as the **root** user:

1. Set the **default_store** configuration key to **swift**:

```
# openstack-config --set /etc/glance/glance-api.conf \
   DEFAULT default_store swift
```

2. Set the **swift_store_auth_address** configuration key to the public endpoint for the Identity service:

```
# openstack-config --set /etc/glance/glance-api.conf \
   DEFAULT swift_store_auth_address http://IP:5000/v2.0/
```

3. Add the container for storing images in the Object Storage Service:

```
# openstack-config --set /etc/glance/glance-api.conf \
 DEFAULT swift_store_create_container_on_put True
```

4. Set the **swift_store_user** configuration key to contain the tenant and user to use for authentication in the format *TENANT*:*USER*:

- If you followed the instructions in this guide to deploy Object Storage, these values must be replaced with the **services** tenant and the **swift** user respectively.
- If you did not follow the instructions in this guide to deploy Object Storage, these values must be replaced with the appropriate Object Storage tenant and user for your environment.

5. Set the **swift_store_key** configuration key to the password of the user to be used for authentication (that is, the password that was set for the **swift** user when deploying the Object Storage service.

```
# openstack-config --set /etc/glance/glance-api.conf \
  DEFAULT swift_store_key PASSWORD
```

5.4.6. Configure the Firewall to Allow Image Service Traffic

The Image Service should be accessible over the network through port **9292**.

To allow this, the Image service should be configured to recognize the 9292 port, and the firewall on the system hosting the image storage service should also allow network traffic on the port. All steps in this procedure must be run while logged in to the server hosting the image storage service as the **root** user.

Procedure 5.5. Configuring the firewall to allow Image Service traffic

1. Open the **/etc/glance/glance-api.conf** file in a text editor, and remove any comment characters preceding the following parameters:

```
bind_host = 0.0.0.0
bind_port = 9292
```

2. Open the **/etc/sysconfig/iptables** file in a text editor.
3. Add an INPUT rule allowing TCP traffic on port **9292** to the file. The new rule must appear before any INPUT rules that REJECT traffic.

```
-A INPUT -p tcp -m multiport --dports 9292 -j ACCEPT
```

4. Save the changes to the **/etc/sysconfig/iptables** file.
5. Restart the **iptables** service to ensure that the change takes effect.

```
# service iptables restart
```

The firewall is now configured to allow incoming connections to the Image Service on port **9292**.

5.4.7. Configure RabbitMQ Message Broker Settings for the Image Service

As of Red Hat Enterprise Linux OpenStack Platform 5, RabbitMQ replaces QPid as the default (and recommended) message broker. The RabbitMQ messaging service is provided by the *rabbitmq-server* package.

Procedure 5.6. Configuring the Image Service (glance) to use the RabbitMQ message broker

1. Log in as **root** to the Image Service node.
2. In **glance-api.conf** of that system, set RabbitMQ as the notifier:

```
# openstack-config --set /etc/glance/glance-api.conf \
  DEFAULT notification_driver messaging
```

3. Set the name of the RabbitMQ host. Replace **rabbitmq-hostname** with the name of the RabbitMQ host:

```
# openstack-config --set /etc/glance/glance-api.conf \
  DEFAULT rabbit_host RABBITMQ_HOST
```

   Replace *RABBITMQ_HOST* with the IP address or host name of the message broker.

4. Set the message broker port to **5672**:

```
# openstack-config --set /etc/glance/glance-api.conf \
  DEFAULT rabbit_port 5672
```

5. Set the RabbitMQ username and password created for the Image Service:

```
# openstack-config --set /etc/glance/glance-api.conf \
  DEFAULT rabbit_userid glance
```

```
# openstack-config --set /etc/glance/glance-api.conf \
  DEFAULT rabbit_password GLANCE_PASS
```

   Where **glance** and *GLANCE_PASS* are the RabbitMQ username and password created for the Image Service (in Section 2.4.2, “Install and Configure the RabbitMQ Message Broker”).

6. In Section 2.4.2.1, “Launch the RabbitMQ Message Broker”, we gave the **glance** user read/write permissions to all resources -- specifically, through the virtual host **/**. Configure the Image Service to connect to this virtual host:

```
# openstack-config --set /etc/glance/glance-api.conf \
  DEFAULT rabbit_virtual_host /
```

5.4.8. Configure the Image Service to Use SSL

Use the following options in the **glance-api.conf** file to configure SSL.

Table 5.1. SSL options for Image Service

Configuration Option	Description
cert_file	Path to certificate file to use when starting API server securely.
key_file	Path to private key file to use when starting API server securely.
ca_file	Path to CA certificate file to use to verify connecting clients.

5.4.9. Populate the Image Service Database

You can populate the Image Service database after you have successfully configured the Image Service database connection string (refer to Section 5.4.3, "Configure the Image Service Database Connection").

Procedure 5.7. Populating the Image Service database

1. Log in to the system hosting the Image service.
2. Use the **su** command to switch to the **glance** user.

```
# su glance -s /bin/sh
```

3. Run the **glance-manage db_sync** command to initialize and populate the database identified in **/etc/glance/glance-api.conf** and **/etc/glance/glance-registry.conf**.

```
# glance-manage db_sync
```

The Image service database has been initialized and populated.

5.4.10. Enable Image Loading Through the Local File System

By default, the Image service provides images to instances using the HTTP protocol. That is, image data is transmitted from the image store to the local disk of the Compute node using HTTP.

This process is typical for most deployments where the Image and Compute services are installed on different hosts.

Note

You can use direct image access even if Image and Compute services are not installed on the same hosts, but are sharing a shared file system. The only requirement in this case would be to have the file system mounted in the same location.

In deployments where both services are installed on the same host (and, consequently, share the same file system), it is more efficient to skip the HTTP steps altogether. Instead, you will need to configure both Image and Compute services to send and receive images using the local file system.

To do this:

Procedure 5.8. Configuring Image and Compute services to send/receive images through the local file system

Important

The Image file system metadata to be generated for this procedure will only apply to new images. Any existing images will not use this metadata.

1. Create a JSON document that exposes the Image file system metadata required by **openstack-nova-compute**.
2. Configure the Image service to use the JSON document.

3. Configure **openstack-nova-compute** to use the file system metadata provided by the Image service.

Note

If both Image and Compute services are hosted on different nodes, you can emulate local file system sharing through Gluster.

The following sections describe this in more detail.

5.4.10.1. Configure File System Sharing Across Different Image and Compute Nodes

If both the Image and Compute services are hosted on different nodes, you can still enable them to share images locally. To do so, you will have to use Gluster (Red Hat Storage shares).

With this configuration, both Image and Compute services will have to share the same Gluster volume. For this, the same volume must be mounted on their respective nodes. Doing so will allow both services to access the same volume locally, thereby allowing image loading through local file system.

This configuration requires that you:

1. Install and configure the packages required for Gluster on the nodes hosting Image and Compute services.
2. Create the GlusterFS volume to be shared by both Image and Compute services.
3. Mount the GlusterFS volume on the Image and Compute service nodes.

Note

For instructions on this procedure, refer to the latest version of the *Configuring Red Hat OpenStack with Red Hat Storage* guide available from:

https://access.redhat.com/site/documentation/en-US/Red_Hat_Storage/

Once you have configured the GlusterFS volume and mounted it on the Image service node, you will also have to configure the Compute service node to use the mounted Gluster volume. To do so:

Procedure 5.9. Configuring the Compute service node to use a mounted Gluster volume

1. Log in to the Compute service node.
2. From there, install the packages required for Gluster:

```
# yum install -y glusterfs glusterfs-fuse
```

3. Ensure that the drivers required to load the Gluster volume are enabled. To do so:
    a. Open the **/etc/nova/nova.conf** configuration file.

b. Search for the **Libvirt** handlers for remote volumes (specifically, **volume_drivers**). The value for this parameter should be a comma-delimited list of drivers for different types of volumes.

c. Depending on your Compute service deployment, the **volume_drivers** may already be enabled (as in, uncommented). If so, ensure that the Gluster volume driver (namely **glusterfs=nova.virt.libvirt.volume.LibvirtGlusterfsVolumeDriver** is also listed.

   If the **volume_drivers** parameter is disabled or is not listed, edit the file accordingly.

4. Configure the Compute service to use the mounted Gluster volume:

```
# openstack-config --set /etc/nova/nova-conf \
 DEFAULT glusterfs_mount_point_base GLUSTER_MOUNT
```

   Replace *GLUSTER_MOUNT* with the directory where the Gluster volume is mounted.

5. Restart the Compute service.

```
# service openstack-nova-compute restart
```

After completing these procedures, both Image and Compute services can now emulate accessing the same file system as if it were a local file system. You can then enable image loading through the local file system as normal.

5.4.10.2. Configure the Image Service to Provide Images Through the Local File System

In order to enable image loading through the local file system (as opposed to HTTP), the Image service needs to first expose its local file-system metadata to the **openstack-nova-compute** service. To do so:

Procedure 5.10. Configuring the Image service to expose local file system metadata to the Compute service

1. Determine the mount point of the file system used by the Image service:

```
# df
Filesystem     1K-blocks     Used Available Use% Mounted on
/dev/sda3       51475068 10905752  37947876  23% /
devtmpfs         2005504        0   2005504   0% /dev
tmpfs            2013248      668   2012580   1% /dev/shm
```

   For example, if the Image service uses the **/dev/sda3** file system, then its corresponding mount point is **/**.

2. Create a unique ID for the mount point using:

```
# uuidgen
ad5517ae-533b-409f-b472-d82f91f41773
```

   Note the output of the **uuidgen**, as this will be used in the next step.

3. Create a file with the **.json** extension.

4. Open the file and add the following information:

```
{
"id": "UID",
"mountpoint": "MOUNTPT"
}
```

Where:

- *UID* is the unique ID created in the previous step.
- *MOUNTPT* is the mount point of the Image service's file system (as determined in the first step).

5. Configure the Image service to use this JSON file:

```
# openstack-config --set /etc/glance/glance-api.conf \
 DEFAULT show_multiple_locations True
```

```
# openstack-config --set /etc/glance/glance-api.conf \
 DEFAULT filesystem_store_metadata_file JSON_PATH
```

Replace *JSON_PATH* with the full path to the JSON file.

6. Restart the Image service (if already running).

```
# service openstack-glance-registry restart
# service openstack-glance-api restart
```

Important

The Image file-system metadata generated for this procedure only applies to new images. Any image that exists (that is, prior to this procedure) will not use this metadata.

5.4.10.3. Configure the Compute Service to Use Local File System Metadata

After configuring the Image Service to expose local file system metadata (as part of Section 5.4.10.2, "Configure the Image Service to Provide Images Through the Local File System"), you can then configure the Compute service to use this metadata. Doing so allows **openstack-nova-compute** to load images from the local file system.

To do so:

Procedure 5.11. Configuring the Compute service to use file system metadata provided by the Image Service

1. Configure **openstack-nova-compute** to enable the use of direct URLs that have the **file://** scheme:

```
# openstack-config --set /etc/nova/nova.conf \
 DEFAULT allowed_direct_url_schemes file
```

2. Create an entry for the Image service's file system:

```
# openstack-config --set /etc/nova/nova.conf \
image_file_url filesystems FSENTRY
```

Replace *FSENTRY* with the name you wish to assign to the Image service's file system.

3. Open the **.json** file used by the Image service to expose its local file system metadata. The information in this file will be used in the next step.

4. Associate the entry created in the previous step to the file system metadata exposed by the Image service:

```
# openstack-config --set /etc/nova/nova.conf \
image_file_url:FSENTRY id UID
```

```
# openstack-config --set /etc/nova/nova.conf \
image_file_url:FSENTRY mountpoint MOUNTPT
```

Where:

- *UID* is the unique ID used by the Image service. In the **.json** file used by the Image service, the *UID* is the **"id":** value.
- *MOUNTPT* is the mount point used by the Image service's file system. In the **.json** file used by the Image service, the *MOUNTPT* is the **"mountpoint":** value.

5.5. Launch the Image API and Registry Services

Now that Glance has been configured, start the **glance-api** and **glance-registry** services as the **root** user:

```
# service openstack-glance-registry start
# service openstack-glance-api start
# chkconfig openstack-glance-registry on
# chkconfig openstack-glance-api on
```

5.6. Validate the Image Service Installation

5.6.1. Obtain a Test Disk Image

A disk image can be downloaded from Red Hat, which can be used as a test in the import of images into the Image service (see Section 5.6.3, “Upload a Disk Image”).

A new image is provided with each minor Red Hat Enterprise Linux 6 release, and is available in the download section of the Red Hat Enterprise Linux 6 Server channel:

https://rhn.redhat.com/rhn/software/channel/downloads/Download.do?cid=16952

The **wget** command below uses an example URL.

```
# mkdir /tmp/images
# cd /tmp/images
```

```
# wget -c -O rhel-6-server-x86_64-disc1.iso "https://content-
web.rhn.redhat.com/rhn/isos/xxxx/rhel-6-server-x86_64-
disc1.isoxxxxxxxx"
```

5.6.2. Build a Custom Virtual Machine Image

Red Hat Enterprise Linux OpenStack Platform includes Oz, a set of libraries and utilities for performing automated operating system installations with limited input from the user. Oz is also useful for building virtual machine images that can be uploaded to the Image service and used to launch virtual machine instances.

- Template Description Language (TDL) Files - Oz accepts input in the form of XML-based TDL files, which describe the operating system being installed, the installation media's source, and any additional packages or customization changes that must be applied to the image.
- **virt-sysprep** - It is also recommended that the **virt-sysprep** command is run on Linux-based virtual machine images prior to uploading them to the Image service. The **virt-sysprep** command re-initializes a disk image in preparation for use in a virtual environment. Default operations include the removal of SSH keys, removal of persistent MAC addresses, and removal of user accounts.

  The **virt-sysprep** command is provided by the *libguestfs-tools* package.

Important

Oz makes use of the **default** Libvirt network. It is recommended that you do not build images using Oz on a system that is running either the **nova-network** service or any of the OpenStack Networking components.

Procedure 5.12. Building Images using Oz

1. Use the **yum** command to install the *oz* and *libguestfs-tools* packages.

   ```
   # yum install -y oz libguestfs-tools
   ```

2. Download the Red Hat Enterprise Linux 6 Server installation DVD ISO file.

   Although Oz supports the use of network-based installation media, in this procedure a Red Hat Enterprise Linux 6 DVD ISO will be used.

3. Use a text editor to create a TDL file for use with Oz. The following example displays the syntax for a basic TDL file.

   Example 5.1. TDL File

   The template below can be used to create a Red Hat Enterprise Linux 6 disk image. In particular, note the use of the **rootpw** element to set the password for the **root** user and the **iso** element to set the path to the DVD ISO.

   ```
   <template>
    <name>rhel66_x86_64</name>
    <description>Red Hat 6.6 x86_64 template</description>
    <os>
   ```

```
  <name>RHEL-6</name>
  <version>6</version>
  <arch>x86_64</arch>
  <rootpw>PASSWORD</rootpw>
  <install type='iso'>
    <iso>file:///home/user/rhel-server-6.6-x86_64-dvd.iso</iso>
  </install>
 </os>
 <commands>
   <command name='console'>
sed -i 's/ rhgb//g' /boot/grub/grub.conf
sed -i 's/ quiet//g' /boot/grub/grub.conf
sed -i 's/ console=tty0 / console=ttyS0,115200n8 console=tty0 /g'
/boot/grub/grub.conf
   </command>
 </commands>
</template>
```

4. Run the **oz-install** command to build an image:

```
# oz-install -u -d3 TDL_FILE
```

Syntax:

- **-u** ensures any required customization changes to the image are applied after guest operating installation.
- **-d3** enables the display of errors, warnings, and informational messages.
- *TDL_FILE* provides the path to your TDL file.

By default, Oz stores the resultant image in the **/var/lib/libvirt/images/** directory. This location is configurable by editing the **/etc/oz/oz.cfg** configuration file.

5. Run the **virt-sysprep** command on the image to re-initialize it in preparation for upload to the Image service. Replace *FILE* with the path to the disk image.

```
# virt-sysprep --add FILE
```

Refer to the **virt-sysprep** manual page by running the **man virt-sysprep** command for information on enabling and disabling specific operations.

You have successfully created a Red Hat Enterprise Linux based image that is ready to be added to the Image service.

Note

The Red Hat Enterprise Linux OpenStack Platform 5 release included the **Disk Image Builder** as a Technology Preview. This tool provides automation to build RAM disks and disk images for deploying instances through OpenStack. While users and operators can manually script or put together RAM disks and disk images, automation makes customization and testing easier.

For more information on the support scope for features marked as technology previews, refer to https://access.redhat.com/support/offerings/techpreview/.

5.6.3. Upload a Disk Image

To launch instances based on images stored in the Image service, you must first upload one or more images into the Image service.

To carry out this procedure, you must already have created or downloaded images suitable for use in the OpenStack environment. For more information, refer to:

- Section 5.6.1, “Obtain a Test Disk Image”
- Section 5.6.2, “Build a Custom Virtual Machine Image”

Important

It is recommended that the **virt-sysprep** command be run on all Linux-based virtual machine images prior to uploading them to the Image service. The **virt-sysprep** command re-initializes a disk image in preparation for use in a virtual environment. Default operations include the removal of SSH keys, removal of persistent MAC addresses, and removal of user accounts.

The **virt-sysprep** command is provided by the Red Hat Enterprise Linux *libguestfs-tools* package. As the **root** user, execute:

```
# yum install -y libguestfs-tools
# virt-sysprep --add FILE
```

For information on enabling and disabling specific operations, refer to the command's manual page by executing:

```
# man virt-sysprep
```

To upload an image to the Image service:

1. Set the environment variables used for authenticating with the Identity service by loading them from the **keystonerc** file associated with your user (an administrative account is not required):

```
# source ~/keystonerc_userName
```

2. Use the **glance image-create** command to import your disk image:

```
# glance image-create --name "NAME" \
      --is-public IS_PUBLIC \
      --disk-format DISK_FORMAT \
      --container-format CONTAINER_FORMAT \
      --file IMAGE
```

Where:

- *NAME* = The name by which users will refer to the disk image.
- *IS_PUBLIC* = Either **true** or **false**:
  - **true** - All users will be able to view and use the image.
  - **false** - Only administrators will be able to view and use the image.
- *DISK_FORMAT* = The disk image's format. Valid values include: **aki**, **ami**, **ari**, **iso**, **qcow2**, **raw**, **vdi**, **vhd**, and **vmdk**.

  If the format of the virtual machine disk image is unknown, use the **qemu-img info** command to try and identify it.

  Example 5.2. Using qemu-img info

  In the following example, the **qemu-img info** is used to determine the format of a disk image stored in the file **./RHEL66.img**.

```
# qemu-img info ./RHEL66.img
image: ./RHEL66.img
file format: qcow2
virtual size: 5.0G (5368709120 bytes)
disk size: 136K
cluster_size: 65536
```

- *CONTAINER_FORMAT* = The container format of the image. The container format is **bare** unless the image is packaged in a file format such as **ovf** or **ami** that includes additional metadata related to the image.
- *IMAGE* = The local path to the image file (for uploading).

For more information about the **glance image-create** syntax, execute:

```
# glance help image-create
```

Note

If the image being uploaded is not locally accessible but is available using a remote URL, provide the URL using the **--location** parameter instead of using the **--file** parameter.

However, unless you also specify the **--copy-from** argument, the Image service will not copy the image into the object store. Instead, the image will be accessed remotely each time it is required.

Example 5.3. Uploading an Image to the Image service

In this example the **qcow2** format image in the file named **RHEL66.img** is uploaded to the Image service. It is created in the service as a publicly accessible image named **RHEL 6.6**.

```
# glance image-create --name "RHEL 6.6" --is-public true --
disk-format qcow2 \
          --container-format bare \
          --file RHEL66.img
+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
| checksum         | 2f81976cae15c16ef0010c51e3a6c163     |
| container_format | bare                                 |
| created_at       | 2013-01-25T14:45:48                  |
| deleted          | False                                |
| deleted_at       | None                                 |
| disk_format      | qcow2                                |
| id               | 0ce782c6-0d3e-41df-8fd5-39cd80b31cd9 |
| is_public        | True                                 |
| min_disk         | 0                                    |
| min_ram          | 0                                    |
| name             | RHEL 6.6                             |
| owner            | b1414433c021436f97e9e1e4c214a710     |
| protected        | False                                |
| size             | 25165824                             |
| status           | active                               |
| updated_at       | 2013-01-25T14:45:50                  |
+------------------+--------------------------------------+
```

3. To verify that your image was successfully uploaded, use the **glance image-list** command:

```
 # glance image-list
+--------------+----------+-------------+------------------+------
----+--------+
| ID           | Name     | Disk Format | Container Format |Size
| Status |
+--------------+----------+-------------+------------------+------
----+--------+
```

```
| 0ce782c6-... | RHEL 6.6 | qcow2       | bare
|213581824 | active |
+--------------+----------+-------------+------------------+------
----+--------+
```

To view detailed information about an uploaded image, execute the **glance image-show** command using the image's identifier:

```
# glance image-show 0ce782c6-0d3e-41df-8fd5-39cd80b31cd9
+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
| checksum         | 2f81976cae15c16ef0010c51e3a6c163     |
| container_format | bare                                 |
| created_at       | 2013-01-25T14:45:48                  |
| deleted          | False                                |
| disk_format      | qcow2                                |
| id               | 0ce782c6-0d3e-41df-8fd5-39cd80b31cd9 |
| is_public        | True                                 |
| min_disk         | 0                                    |
| min_ram          | 0                                    |
| name             | RHEL 6.6                             |
| owner            | b1414433c021436f97e9e1e4c214a710     |
| protected        | False                                |
| size             | 25165824                             |
| status           | active                               |
| updated_at       | 2013-01-25T14:45:50                  |
+------------------+--------------------------------------+
```

You have successfully uploaded a disk image to the Image service. This disk image can now be used as the basis for launching virtual machine instances in your OpenStack environment.

Chapter 6. OpenStack Block Storage Installation

6.1. Block Storage Installation Overview

Block storage functionality is provided in OpenStack by three separate services collectively referred to as the Block Storage service or **`cinder`**. The three services are:

The API service (`openstack-cinder-api`)

The API service provides a HTTP endpoint for block storage requests. When an incoming request is received the API verifies identity requirements are met and translates the request into a message denoting the required block storage actions. The message is then sent to the message broker for processing by the other Block Storage services.

The scheduler service (`openstack-cinder-scheduler`)

The scheduler service reads requests from the message queue and determines on which block storage host the request must be actioned. The scheduler then communicates with the volume service on the selected host to process the request.

The volume service (`openstack-cinder-volume`)

The volume service manages the interaction with the block storage devices. As requests come in from the scheduler, the volume service creates, modifies, and removes volumes as required.

Although the three services can be co-located in a production environment, it is more common to deploy many instances of the volume service with one or more instances of the API and scheduler services managing them.

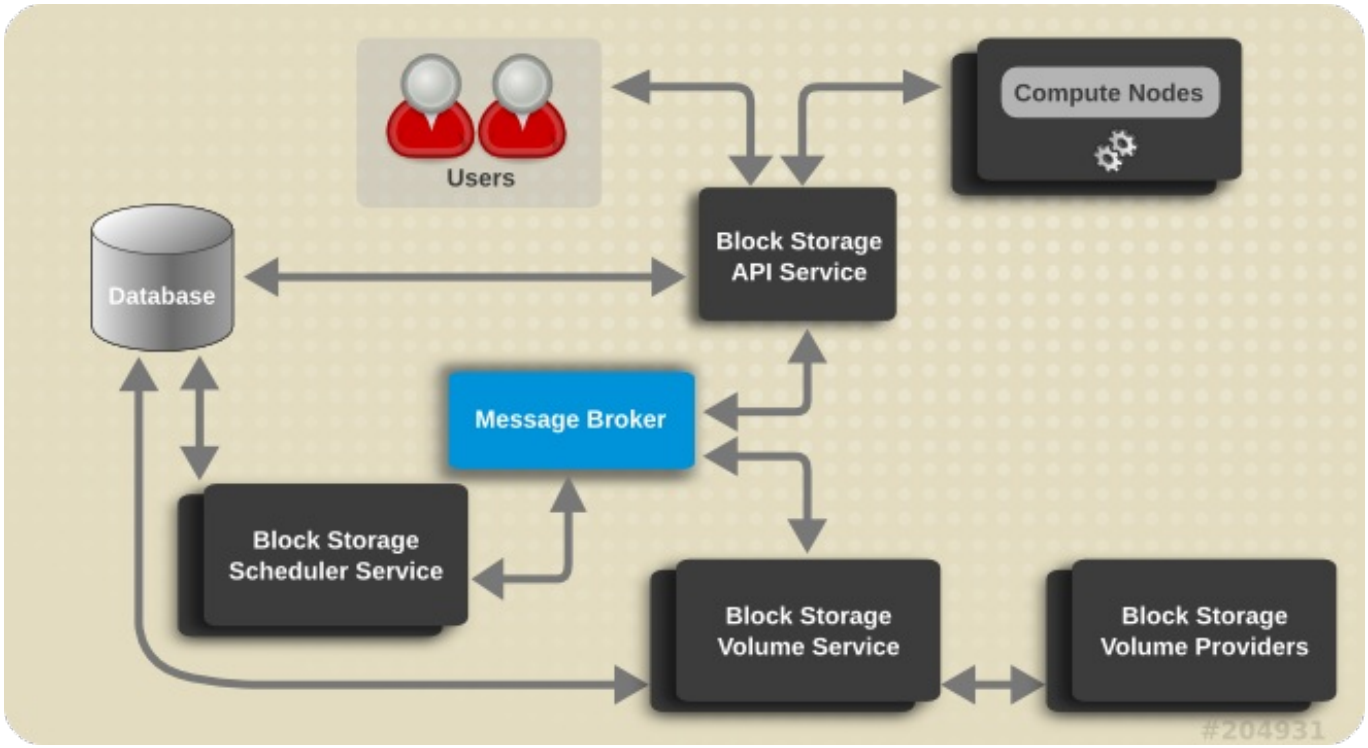


Figure 6.1. Block Storage Architecture

Deploying the Block Storage service is made up of three major phases:

Preparing for Block Storage Installation

Steps that must be performed before installing any of the Block Storage services. These procedures include the creation of identity records, the database, and a database user.

Common Block Storage Configuration

Steps that are common to all of the Block Storage services and as such must be performed on all block storage nodes in the environment. These procedures include configuring the services to refer to the correct database and message broker. Additionally, they include the initialization and population of the database which must only be performed once but can be performed from any of the block storage systems.

Volume Service Specific Configuration

Steps that are specific to systems that will be hosting the volume service and as such require direct access to block storage devices.

6.2. Block Storage Prerequisite Configuration

6.2.1. Create the Block Storage Database

In this procedure the database and database user that will be used by the Block Storage services will be created. These steps must be performed while logged in to the database server as the **root** user.

Procedure 6.1. Creating the database to be used by Block Storage Services

1. Connect to the database service using the **mysql** command.

```
# mysql -u root -p
```

2. Create the **cinder** database.

```
mysql> CREATE DATABASE cinder;
```

3. Create a **cinder** database user and grant it access to the **cinder** database.

```
mysql> GRANT ALL ON cinder.* TO 'cinder'@'%' IDENTIFIED BY
'PASSWORD';
```

```
mysql> GRANT ALL ON cinder.* TO 'cinder'@'localhost' IDENTIFIED
BY 'PASSWORD';
```

Replace *PASSWORD* with a secure password that will be used to authenticate with the database server as this user.

4. Flush the database privileges to ensure that they take effect immediately.

```
mysql> FLUSH PRIVILEGES;
```

5. Exit the **mysql** client.

```
mysql> quit
```

The block storage database has been created. The database will be populated during service configuration.

6.2.2. Create the Block Storage Service Identity Records

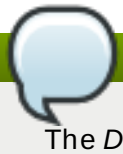
The steps outlined in this procedure cover the creation of Identity records to support the Block Storage service:

1. Create the **cinder** user, who has the **admin** role in the **services** tenant.
2. Create the **cinder** service entry and assign it an endpoint.

These entries provide authentication for the Block Storage services, and guide other OpenStack services attempting to locate and access the volume functionality provided by Block Storage.

In order to proceed, you should have already performed the following (through the Identity service):

1. Created an Administrator role named **admin** (refer to Section 3.7, "Create an Administrator Account" for instructions)
2. Created the **services** tenant (refer to Section 3.9, "Create the Services Tenant" for instructions)



Note

The *Deploying OpenStack: Learning Environments (Manual Set Up)* guide uses one tenant for all service users. For more information, refer to Section 3.9, "Create the Services Tenant".

You can perform the following procedure from your Identity service host or on any machine where you've copied the **keystonerc_admin** file (which contains administrator credentials) and the **keystone** command-line utility is installed.

Procedure 6.2. Creating Identity records for the Block Storage service

1. Authenticate as the administrator of the Identity service by running the **source** command on the **keystonerc_admin** file containing the required credentials.

```
# source ~/keystonerc_admin
```

2. Create a user named **cinder** for the Block Storage service to use.

```
# keystone user-create --name cinder --pass PASSWORD
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|  email   |                                  |
| enabled  |               True               |
|    id    | e1765f70da1b4432b54ced060139b46a |
|   name   |              cinder              |
| tenantId |                                  |
+----------+----------------------------------+
```

Replace *PASSWORD* with a secure password that will be used by the Block Storage service when authenticating with the Identity service.

3. Use the **keystone user-role-add** command to link the **cinder** user, **admin** role, and **services** tenant together:

```
# keystone user-role-add --user cinder --role admin --tenant
services
```

4. Create the **cinder** service entry:

```
# keystone service-create --name cinder \
       --type volume \
       --description "Cinder Volume Service"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |      Cinder Volume Service       |
|      id     | dfde7878671e484c9e581a3eb9b63e66 |
|     name    |              cinder              |
|     type    |              volume              |
+-------------+----------------------------------+
```

5. Create the **cinder** endpoint entry.

```
# keystone endpoint-create \
       --service cinder \
       --publicurl "http://IP:8776/v1/\$(tenant_id)s" \
       --adminurl "http://IP:8776/v1/\$(tenant_id)s" \
       --internalurl "http://IP:8776/v1/\$(tenant_id)s"
```

Replace *IP* with the IP address or host name of the system that will be hosting the Block Storage service API (**openstack-cinder-api**).

Important

If you intend to install and run multiple instances of the API service then you must repeat this step for the IP address or host name of each instance.

All supporting Identity service entries required by the Block Storage services have been created.

6.3. Common Block Storage Configuration

6.3.1. Install the Block Storage Service Packages

The OpenStack Block Storage service requires the following packages:

openstack-cinder

Provides the Block Storage services and associated configuration files.

openstack-utils

Provides supporting utilities to assist with a number of tasks including the editing of configuration files.

openstack-selinux

Provides OpenStack specific SELinux policy modules.

To install all of the above packages, execute the following command while logged in as the **root** user:

```
# yum install -y openstack-cinder openstack-utils openstack-selinux
```

The Block Storage services are now installed and ready to be configured.

6.3.2. Configure Block Storage Service Authentication

The Block Storage service must be explicitly configured to use the Identity service for authentication. Follow the steps listed in this procedure to configure this.

All steps listed in this procedure must be performed on each system hosting Block Storage services while logged in as the **root** user.

Procedure 6.3. Configuring the Block Storage Service to authenticate through the Identity Service

1. Set the authentication strategy (**auth_strategy**) configuration key to **keystone** using the **openstack-config** command.

   ```
   # openstack-config --set /etc/cinder/cinder.conf \
     DEFAULT auth_strategy keystone
   ```

2. Set the authentication host (**auth_host**) configuration key to the IP address or host name of the Identity server.

   ```
   # openstack-config --set /etc/cinder/cinder.conf \
     keystone_authtoken auth_host IP
   ```

   Replace *IP* with the IP address or host name of the Identity server.

3. Set the administration tenant name (**admin_tenant_name**) configuration key to the name of the tenant that was created for the use of the Block Storage service. In this guide, examples use *services*.

   ```
   # openstack-config --set /etc/cinder/cinder.conf \
     keystone_authtoken admin_tenant_name services
   ```

4. Set the administration user name (**admin_user**) configuration key to the name of the user that was created for the use of the Block Storage service. In this guide, examples use *cinder*.

   ```
   # openstack-config --set /etc/cinder/cinder.conf \
     keystone_authtoken admin_user cinder
   ```

5. Set the administration password (**admin_password**) configuration key to the password that is associated with the user specified in the previous step.

   ```
   # openstack-config --set /etc/cinder/cinder.conf \
     keystone_authtoken admin_password PASSWORD
   ```

The authentication keys used by the Block Storage services have been set and will be used when the services are started.

6.3.3. Configure the Block Storage Service to Use SSL

Use the following options in the **cinder.conf** file to configure SSL.

Table 6.1. SSL options for Block Storage

Configuration Option	Description
backlog	Number of backlog requests to configure the socket with.
tcp_keepidle	Sets the value of TCP_KEEPIDLE in seconds for each server socket.
ssl_ca_file	CA certificate file to use to verify connecting clients.
ssl_cert_file	Certificate file to use when starting the server securely.
ssl_key_file	Private key file to use when starting the server securely.

6.3.4. Configure RabbitMQ Message Broker Settings for the Block Storage Service

As of Red Hat Enterprise Linux OpenStack Platform 5, RabbitMQ replaces QPid as the default (and recommended) message broker. The RabbitMQ messaging service is provided by the *rabbitmq-server* package.

This section assumes that you have already configured a RabbitMQ message broker. For more information, refer to:

- Section 2.4, “Prerequisite Message Broker”
- Section 2.4.2, “Install and Configure the RabbitMQ Message Broker”
- Section 2.4.1, “Configure the Firewall for Message Broker Traffic”
- Section 2.4.2.1, “Launch the RabbitMQ Message Broker”

Procedure 6.4. Configuring the Block Storage service to use the RabbitMQ message broker

1. Log in as **root** to the system hosting the Block Storage services.
2. In **/etc/cinder/cinder.conf**, set RabbitMQ as the RPC back end.

```
# openstack-config --set /etc/cinder/cinder.conf \
  DEFAULT rpc_backend cinder.openstack.common.rpc.impl_kombu
```

3. Set the Block Storage service to connect to the RabbitMQ host:

```
# openstack-config --set /etc/cinder/cinder.conf \
  DEFAULT rabbit_host RABBITMQ_HOST
```

   Replace *RABBITMQ_HOST* with the IP address or host name of the message broker.

4. Set the message broker port to **5672**:

```
# openstack-config --set /etc/cinder/cinder.conf \
```

```
   DEFAULT rabbit_port 5672
```

5. Set the RabbitMQ username and password created for the Block Storage service:

```
# openstack-config --set /etc/cinder/cinder.conf \
   DEFAULT rabbit_userid cinder
```

```
# openstack-config --set /etc/cinder/cinder.conf \
   DEFAULT rabbit_password CINDER_PASS
```

   Where **cinder** and *CINDER_PASS* are the RabbitMQ username and password created for Block Storage (in Section 2.4.2, “Install and Configure the RabbitMQ Message Broker”).

6. In Section 2.4.2.1, “Launch the RabbitMQ Message Broker”, we gave the **cinder** user read/write permissions to all resources -- specifically, through the virtual host **/**. Configure the Block Storage service to connect to this virtual host:

```
# openstack-config --set /etc/cinder/cinder.conf \
   DEFAULT rabbit_virtual_host /
```

6.3.4.1. Enable SSL Communication Between Block Storage Service and Message Broker

If you enabled SSL on the message broker, you will also have to configure the Block Storage service accordingly. For this, you will need the exported client certificates and key file. See Section 2.4.2.3, “Export an SSL Certificate for Clients” for instructions on how to export these files.

Once you have the necessary certificates and key, run the following commands to enable SSL communication with the message broker:

```
# openstack-config --set /etc/cinder/cinder.conf \
 DEFAULT rabbit_use_ssl True
# openstack-config --set /etc/cinder/cinder.conf \
 DEFAULT kombu_ssl_certfile /path/to/client.crt
# openstack-config --set /etc/cinder/cinder.conf \
 DEFAULT kombu_ssl_keyfile /path/to/clientkeyfile.key
```

Where:

- */path/to/client.crt* is the absolute path to the exported client certificate.
- */path/to/clientkeyfile.key* is the absolute path to the exported client key file.

If your certificates were signed by a third-party Certificate Authority (CA), then you will also need to run the following command:

```
# openstack-config --set /etc/cinder/cinder.conf \
 DEFAULT kombu_ssl_ca_certs /path/to/ca.crt
```

Replace */path/to/ca.crt* with the absolute path to the CA file provided by the third-party CA (see Section 2.4.2.2, “Enable SSL on the RabbitMQ Message Broker” for more information).

6.3.5. Configure the Block Storage Service Database Connection

The database connection string used by the Block Storage services (the value of the **sql_connection** configuration key) is defined in the **/etc/cinder/cinder.conf** file. The string must be updated to point to a valid database server before starting the service.

The following command must be executed as the **root** user on each system hosting Block Storage services:

```
# openstack-config --set /etc/cinder/cinder.conf \
  DEFAULT sql_connection mysql://USER:PASS@IP/DB
```

Replace:

- *USER* with the database user name the Block Storage services are to use, usually **cinder**.
- *PASS* with the password of the chosen database user.
- *IP* with the IP address or host name of the database server.
- *DB* with the name of the database that has been created for use by the Block Storage services, usually **cinder**.

Important

The IP address or host name specified in the connection configuration key must match the IP address or host name to which the cinder database user was granted access when creating the cinder database. Moreover, if the database is hosted locally and you granted permissions to 'localhost' when creating the cinder database, you must enter 'localhost'.

The database connection string has been set and will be used by the Block Storage services.

6.3.6. Configure the Firewall to Allow Block Storage Service Traffic

Systems attempting to use the functionality provided by the Block Storage services access it over the network using ports **3260** and **8776**.

To allow this the firewall on the system hosting the Block Storage service must be altered to allow network traffic on these ports. All steps in this procedure must be run on each system hosting Block Storage services while logged in as the **root** user.

Procedure 6.5. Configuring the firewall to allow Block Storage Service traffic

1. Open the **/etc/sysconfig/iptables** file in a text editor.
2. Add an INPUT rule allowing TCP traffic on ports **3260** and **8776** to the file. The new rule must appear before any INPUT rules that REJECT traffic.

   ```
   -A INPUT -p tcp -m multiport --dports 3260,8776 -j ACCEPT
   ```

3. Save the changes to the **/etc/sysconfig/iptables** file.
4. Restart the **iptables** service to ensure that the change takes effect.

   ```
   # service iptables restart
   ```

The firewall is now configured to allow incoming connections to the Block Storage service on ports **3260** and **8776**.

6.3.7. Populate the Block Storage Database

You can populate the Block Storage database after you have successfully configured the Block Storage service database connection string (refer to Section 6.3.5, "Configure the Block Storage Service Database Connection").

Important

This procedure only needs to be followed once to initialize and populate the database. You do not need to perform these steps again when adding additional systems hosting Block Storage services.

Procedure 6.6. Populating the Block Storage Service database

1. Log in to the system hosting one of the Block Storage services.
2. Use the **su** command to switch to the **cinder** user.

```
# su cinder -s /bin/sh
```

3. Run the **cinder-manage db sync** command to initialize and populate the database identified in **/etc/cinder/cinder.conf**.

```
$ cinder-manage db sync
```

The Block Storage service database has been initialized and populated.

6.3.8. Increase the Throughput of the Block Storage API Service

By default, the Block Storage API service (**openstack-cinder-api**) runs in one process. This limits the number of API requests that the Block Storage service can process at any given time. In a production environment, you should increase the Block Storage API throughput by allowing **openstack-cinder-api** to run in as many processes as the machine capacity allows.

Red Hat Enterprise Linux OpenStack Platform 5 adds a new Block Storage API service option to address this, namely ***osapi_volume_workers***. This option allows you to specify the number of API service workers (or OS processes) to launch for **openstack-cinder-api**.

To set this option, run the following command on the **openstack-cinder-api** host:

```
# openstack-config --set /etc/cinder/cinder.conf \
 DEFAULT osapi_volume_workers CORES
```

Replace *CORES* with the number of CPU cores/threads on a machine.

6.4. Volume Service Configuration

6.4.1. Block Storage Driver Support

The volume service (**openstack-cinder-volume**) requires access to suitable block storage. As such, Red Hat Enterprise Linux OpenStack Platform provides volume drivers for several supported block storage types. These types include:

- LVM/iSCSI
- ThinLVM
- NFS
- NetAPP NFS
- Red Hat Storage (Gluster)
- Dell EqualLogic

For more detailed information on configuring volume drivers for the Block Storage service, refer to the *Volume Drivers* section of the *Red Hat Enterprise Linux OpenStack Platform Configuration Reference Guide*. For instructions on how to set up an NFS or GlusterFS back end, refer to the *Manage volumes* section of the *Red Hat Enterprise Linux OpenStack Platform Cloud Administration Guide*.

Both documents are available from the following link:

https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux_OpenStack_Platform

For instructions on how to set up an LVM back end, refer to Section 6.4.2, “Configure OpenStack Block Storage to use an LVM Storage Back End”.

6.4.2. Configure OpenStack Block Storage to use an LVM Storage Back End

The **openstack-cinder-volume** service is able to make use of a volume group attached directly to the server on which the service runs. This volume group must be created exclusively for use by the Block Storage service and the configuration updated to point to the name of the volume group.

The following steps must be performed while logged into the system hosting the **openstack-cinder-volume** service as the **root** user:

Procedure 6.7. Configuring openstack-cinder-volume to use LVM storage as a back end

1. Use the **pvcreate** command to create a physical volume.

```
# pvcreate DEVICE
  Physical volume "DEVICE" successfully created
```

   Replace *DEVICE* with the path to a valid, unused, device. For example:

```
# pvcreate /dev/sdX
```

2. Use the **vgcreate** command to create a volume group.

```
# vgcreate cinder-volumes DEVICE
  Volume group "cinder-volumes" successfully created
```

   Replace *DEVICE* with the path to the device used when creating the physical volume. Optionally replace *cinder-volumes* with an alternative name for the new volume group.

3. Set the **volume_group** configuration key to the name of the newly created volume group.

```
# openstack-config --set /etc/cinder/cinder.conf \
DEFAULT volume_group cinder-volumes
```

The name provided must match the name of the volume group created in the previous step.

4. Ensure that the correct volume driver for accessing LVM storage is in use by setting the **volume_driver** configuration key to **cinder.volume.drivers.lvm.LVMISCSIDriver**.

```
# openstack-config --set /etc/cinder/cinder.conf \
DEFAULT volume_driver cinder.volume.drivers.lvm.LVMISCSIDriver
```

The **openstack-cinder-volume** service has been configured to use LVM storage.

6.4.3. Configure SCSI Target Daemon

The volume storage service uses a SCSI target daemon for mounting storage. A SCSI target daemon must be installed on each system hosting an instance of the volume service. Red Hat Enterprise Linux 6.6 and 7.1 use different SCSI target daemons.

Procedure 6.8. Configure a SCSI target daemon for Red Hat Enterprise Linux 7.1

This procedure must be performed on each system hosting an instance of the volume service.

1. Log in as the **root** user.
2. Install the *targetcli* package.

```
# yum install targetcli
```

3. Launch the **target** daemon and configure it to start at boot time.

```
# systemctl start target
# systemctl enable target
```

4. Configure the volume service to use the **lioadm** iSCSI target user-land tool.

```
# openstack-config --set /etc/cinder/cinder.conf \
  DEFAULT iscsi_helper lioadm
```

5. Set the correct IP address on which the iSCSI daemon should listen (*ISCSIIP*).

```
# openstack-config --set /etc/cinder/cinder.conf \
  DEFAULT iscsi_ip_address ISCSIIP
```

6.5. Launch the Block Storage Services

To bring up the Block Storage functionality at least one instance of each of the three services must be started:

- The API service (**openstack-cinder-api**).
- The scheduler service (**openstack-cinder-scheduler**).

- The volume service (**openstack-cinder-volume**).

The services do not need to be located on the same system, but must be configured to communicate using the same message broker and database instance. Once the services are running, the API will accept incoming volume requests, the scheduler will assign them as appropriate, and the volume service will action them.

Procedure 6.9. Launching Block Storage Services

1. **Starting the API Service**

   Log in to each server that you intend to run the API on as the **root** user and start the API service.

   a. Use the **service** command to start the API service (**openstack-cinder-api**).

   ```
   # service openstack-cinder-api start
   ```

   b. Use the **chkconfig** command to enable the API service permanently (**openstack-cinder-api**).

   ```
   # chkconfig openstack-cinder-api on
   ```

2. **Starting the Scheduler Service**

   Log in to each server that you intend to run the scheduler on as the **root** user and start the scheduler service.

   a. Use the **service** command to start the scheduler (**openstack-cinder-scheduler**).

   ```
   # service openstack-cinder-scheduler start
   ```

   b. Use the **chkconfig** command to enable the scheduler permanently (**openstack-cinder-scheduler**).

   ```
   # chkconfig openstack-cinder-scheduler on
   ```

3. **Starting the Volume Service**

   Log in to each server that Block Storage has been attached to as the **root** user and start the volume service.

   a. Use the **service** command to start the volume service (**openstack-cinder-volume**).

   ```
   # service openstack-cinder-volume start
   ```

   b. Use the **service** command to start the The SCSI target daemon (**tgtd**).

   ```
   # service tgtd start
   ```

   c. Use the **chkconfig** command to enable the volume service permanently (**openstack-cinder-volume**).

```
# chkconfig openstack-cinder-volume on
```

d. Use the **chkconfig** command to enable the SCSI target daemon permanently (**tgtd**).

```
# chkconfig tgtd on
```

The volume service is running and ready to begin allocating volumes as they are requested.

6.6. Validate the Block Storage Service Installation

This procedure lists steps for validating that the block storage installation is complete and ready for use.

Procedure 6.10. Validating the Block Storage Service installation

1. **Testing Locally**

   The steps outlined in this section of the procedure must be performed while logged in to the server hosting the block storage API service as the **root** user or a user with access to a **keystonerc_admin** file containing the credentials of the OpenStack administrator. Transfer the **keystonerc_admin** file to the system before proceeding.

   a. Run the **source** command on the **keystonerc_admin** file to populate the environment variables used for identifying and authenticating the user.

   ```
   # source ~/keystonerc_admin
   ```

   b. Run the **cinder list** command and verify that no errors are returned.

   ```
   # cinder list
   ```

   c. Run the **cinder create** command to create a volume.

   ```
   # cinder create SIZE
   ```

   Replace *SIZE* with the size of the volume to create in Gigabytes (GB).

   d. Run the **cinder delete** command to remove the volume.

   ```
   # cinder delete ID
   ```

   Replace *ID* with the identifier returned when the volume was created.

2. **Testing Remotely**

   The steps outlined in this section of the procedure must be performed while logged in to a system other than the server hosting the block storage API service. Transfer the **keystonerc_admin** file to the system before proceeding.

   a. Install the *python-cinderclient* package using the **yum** command. You will need to authenticate as the **root** user for this step.

```
# yum install -y python-cinderclient
```

b. Run the **source** command on the **keystonerc_admin** file to populate the environment variables used for identifying and authenticating the user.

```
$ source ~/keystonerc_admin
```

c. Run the **cinder list** command and verify that no errors are returned.

```
$ cinder list
```

d. Run the **cinder create** command to create a volume.

```
$ cinder create SIZE
```

Replace *SIZE* with the size of the volume to create in Gigabytes (GB).

e. Run the **cinder delete** command to remove the volume.

```
$ cinder delete ID
```

Replace *ID* with the identifier returned when the volume was created.

Chapter 7. OpenStack Networking Service Installation

7.1. OpenStack Networking Installation Overview

7.1.1. OpenStack Networking Architecture

OpenStack Networking provides cloud administrators with flexibility in deciding which individual services should run on which physical systems. All service daemons can be run on a single physical host for evaluation purposes. Alternatively each service can have its own physical host or even be replicated across multiple hosts for redundancy.

This chapter focuses on an architecture that combines the role of cloud controller with that of the network host while allowing for multiple compute nodes on which virtual machine instances run. The OpenStack Networking services deployed on the cloud controller in this chapter may also be deployed on a separate network host entirely. This is recommended for environments where it is expected that significant amounts of network traffic will need to be routed from virtual machine instances to external networks.

Example network diagram

The placement of OpenStack Networking services and agents can vary depending on requirements. This diagram is an example of a common deployment model, utilizing a dedicated OpenStack Networking node and tenant networks:

Two Compute nodes run the Open vSwitch (ovs-agent), and one OpenStack Networking node performs the network functions: L3 routing, DHCP, NAT, including services such as FWaaS and LBaaS. The Compute nodes have two physical network cards each, one for tenant traffic, and another for management connectivity. The OpenStack Networking node has a third network card specifically for provider traffic:

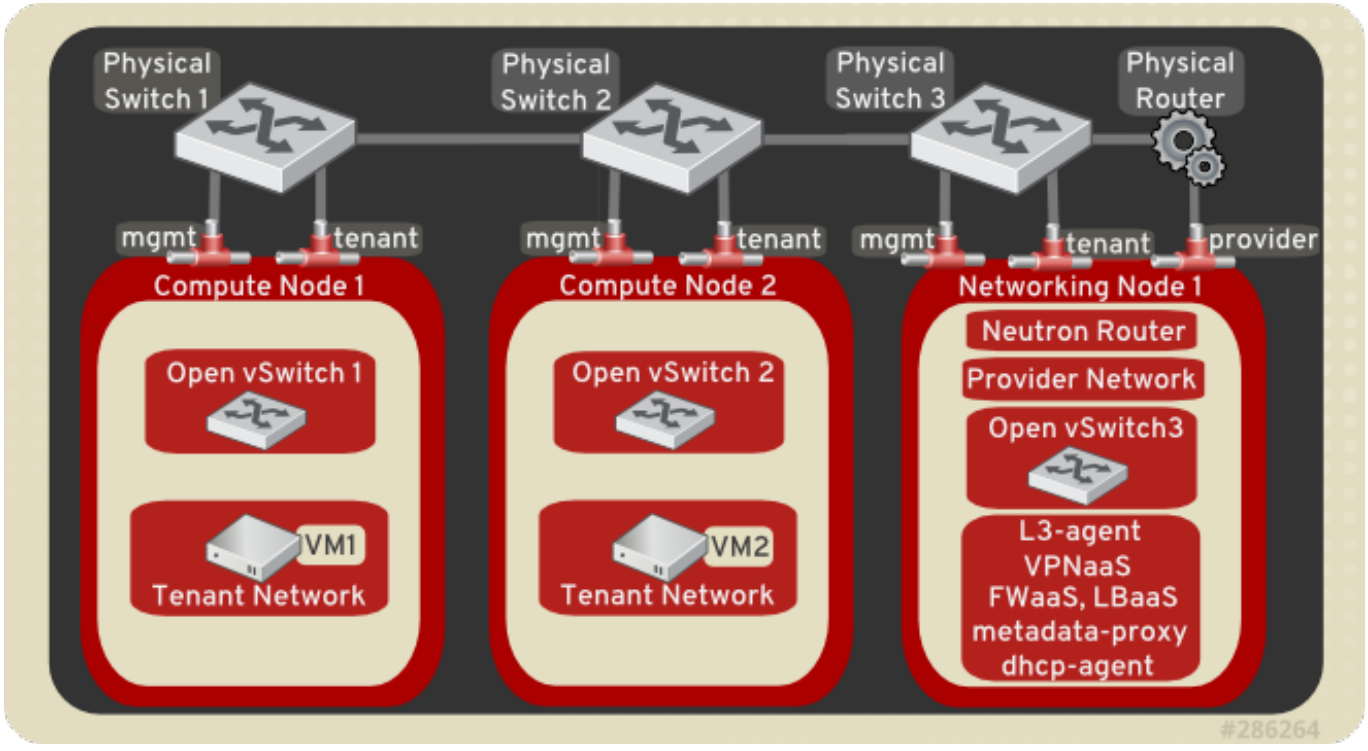


Figure 7.1. Example network diagram

7.1.2. OpenStack Networking API

OpenStack Networking provides a powerful API to define the network connectivity and addressing used by devices from other services, such as OpenStack Compute.

The OpenStack Compute API has a virtual server abstraction to describe compute resources. Similarly, the OpenStack Networking API has virtual network, subnet, and port abstraction layers to describe network resources. In more detail:

Network

An isolated L2 segment, analogous to VLAN in the physical networking world.

Subnet

A block of v4 or v6 IP addresses and associated configuration state.

Port

A connection point for attaching a single device, such as the NIC of a virtual server, to a virtual network. Also describes the associated network configuration, such as the MAC and IP addresses to be used on that port.

You can configure rich network topologies by creating and configuring networks and subnets, and then instructing other OpenStack services like OpenStack Compute to attach virtual devices to ports on these networks. In particular, OpenStack Networking supports each tenant having multiple private networks, and allows tenants to choose their own IP addressing scheme, even if those IP addresses overlap with those used by other tenants. This enables very advanced cloud networking use cases, such as building multi-tiered web applications and allowing applications to be migrated to the cloud without changing IP addresses.

Even if a cloud administrator does not intend to expose the above capabilities to tenants directly, the OpenStack Networking API can be very useful for administrative purposes. The API provides significantly more flexibility for the cloud administrator when customizing network offerings.

7.1.3. OpenStack Networking API Extensions

The OpenStack Networking API allows plug-ins to provide extensions to enable additional networking functional not available in the core API itself.

Provider Networks

Provider networks allow the creation of virtual networks that map directly to networks in the physical data center. This allows the administrator to give tenants direct access to a public network such as the Internet or to integrate with existing VLANs in the physical networking environment that have a defined meaning or purpose.

When the provider extension is enabled OpenStack Networking users with administrative privileges are able to see additional provider attributes on all virtual networks. In addition such users have the ability to specify provider attributes when creating new provider networks.

Both the Open vSwitch and Linux Bridge plug-ins support the provider networks extension.

Layer 3 (L3) Routing and Network Address Translation (NAT)

The L3 routing API extensions provides abstract L3 routers that API users are able to dynamically provision and configure. These routers are able to connect to one or more Layer 2 (L2) OpenStack Networking controlled networks. Additionally the routers are able to

provide a gateway that connects one or more private L2 networks to an common public or external network such as the Internet.

The L3 router provides basic NAT capabilities on gateway ports that connect the router to external networks. The router supports floating IP addresses which give a static mapping between a public IP address on the external network and the private IP address on one of the L2 networks attached to the router.

This allows the selective exposure of compute instances to systems on an external public network. Floating IP addresses are also able to be reallocated to different OpenStack Networking ports as necessary.

Security Groups

Security groups and rules filter the type and direction of network traffic sent to a given network port. This provides an additional layer of security to complement any firewall rules present on the Compute instance. The security group is a container object with one or more security rules. A single security group can manage traffic to multiple compute instances.

Ports created for floating IP addresses, OpenStack Networking LBaaS VIPs, router interfaces, and instances are associated with a security group. If none is specified, then the port is associated with the **`default`** security group. By default, this group will drop all inbound traffic and allow all outbound traffic. Additional security rules can be added to the **`default`** security group to modify its behavior or new security groups can be created as necessary.

The Open vSwitch, Linux Bridge, VMware NSX, NEC, and Ryu networking plug-ins currently support security groups.

Note

Unlike Compute security groups, OpenStack Networking security groups are applied on a per port basis rather than on a per instance basis.

7.1.4. OpenStack Networking Plug-ins

The original OpenStack Compute network implementation assumed a basic networking model where all network isolation was performed through the use of Linux VLANs and firewalls. OpenStack Networking introduces the concept of a *plug-in*, which is a pluggable back-end implementation of the OpenStack Networking API. A plug-in can use a variety of technologies to implement the logical API requests. Some OpenStack Networking plug-ins might use basic Linux VLANs and firewalls, while others might use more advanced technologies, such as L2-in-L3 tunneling or OpenFlow, to provide similar benefits.

Plug-ins for these network technologies are currently tested and supported for use with Red Hat Enterprise Linux OpenStack Platform:

- Open vSwitch (*openstack-neutron-openvswitch*)
- Linux Bridge (*openstack-neutron-linuxbridge*)

Other plug-ins that are also packaged and available include:

- Cisco (*openstack-neutron-cisco*)
- NEC OpenFlow (*openstack-neutron-nec*)

- VMware (*openstack-neutron-vmware*)
- Ryu (*openstack-neutron-ryu*)

Plug-ins enable the cloud administrator to weigh different options and decide which networking technology is right for the deployment.

7.1.5. VMware NSX Integration

OpenStack Networking uses the NSX plugin for Networking to integrate with an existing VMware vCenter deployment. When installed on the network nodes, the NSX plugin enables a NSX controller to centrally manage configuration settings and push them to managed network nodes. Network nodes are considered managed when they're added as hypervisors to the NSX controller.

The diagram below depicts an example NSX deployment and illustrates the route inter-VM traffic takes between separate Compute nodes. Note the placement of the VMware NSX plugin and the **`neutron-server`** service on the network node. The NSX controller features centrally with a green line to the network node to indicate the management relationship:

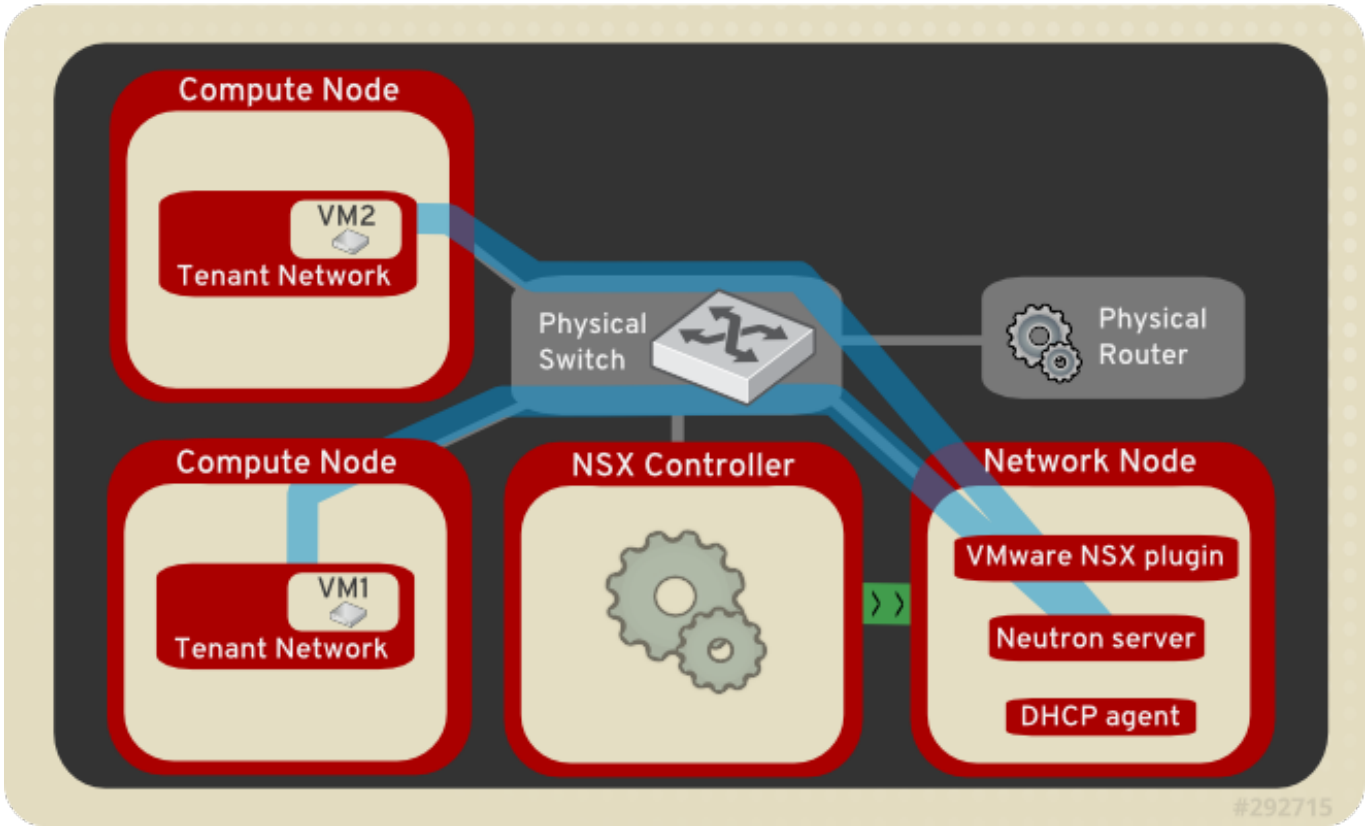


Figure 7.2. VMware NSX overview

7.1.6. Open vSwitch Overview

Open vSwitch is a *software-defined networking* (SDN) virtual switch designed to supersede the heritage Linux software bridge. It provides switching services to virtualized networks with support for industry standard NetFlow, OpenFlow, and sFlow. Open vSwitch is also able to integrate with physical switches due to its support for the layer 2 features STP, LACP, and 802.1Q VLAN tagging.

Open vSwitch tunneling is supported with Open vSwitch version 1.11.0-1.el6 or later. Refer to the table below for specific kernel requirements:

Feature	Kernel Requirement
GRE Tunneling	2.6.32-358.118.1.openstack.el6 or later
VXLAN Tunneling	2.6.32-358.123.4.openstack.el6 or later

7.1.7. Modular Layer 2 (ML2) Overview

ML2 is the new OpenStack Networking core plug-in introduced in OpenStack's Havana release. Superseding the previous model of singular plug-ins, ML2's modular design enables the concurrent operation of mixed network technologies. The monolithic Open vSwitch and linuxbridge plug-ins have been deprecated and will be removed in a future release; their functionality has instead been re-implemented as ML2 mechanisms.

Note

ML2 is the default OpenStack Networking plug-in, with Open vSwitch configured as the default mechanism driver.

The requirement behind ML2

Previously, OpenStack Networking deployments were only able to use the plug-in that had been selected at implementation time. For example, a deployment running the Open vSwitch plug-in was only able to use Open vSwitch exclusively; it wasn't possible to simultaneously run another plug-in such as linuxbridge. This was found to be a limitation in environments with heterogeneous requirements.

ML2 network types

Multiple network segment types can be operated concurrently. In addition, these network segments can interconnect using ML2's support for multi-segmented networks. Ports are automatically bound to the segment with connectivity; it is not necessary to bind them to a specific segment. Depending on the mechanism driver, ML2 supports the following network segment types:

- flat
- GRE
- local
- VLAN
- VXLAN

The various Type drivers are enabled in the ML2 section of the **`ml2_conf.ini`** file:

```
[ml2]
type_drivers = local,flat,vlan,gre,vxlan
```

ML2 Mechanisms

Plug-ins have been reimplemented as mechanisms with a common code base. This approach enables code reuse and eliminates much of the complexity around code maintenance and testing. Supported mechanism drivers currently include:

- Arista

- Cisco
- Nexus
- Hyper-V Agent
- L2 Population
- Linuxbridge Agent
- Open vSwitch Agent
- Tail-f NCS

The various mechanism drivers are enabled in the ML2 section of the **`ml2_conf.ini`** file:

```
[ml2]
mechanism_drivers = openvswitch,linuxbridge,l2population
```

7.1.8. Choose a Network Back-end

The Red Hat Enterprise Linux OpenStack Platform offers two distinctly different networking back-ends: Nova networking and OpenStack Networking (neutron). Nova networking has been deprecated in the OpenStack technology roadmap, but still remains currently available. OpenStack Networking is considered the core software-defined networking (SDN) component of OpenStack's forward-looking roadmap and is under active development.

It is important to consider that there is currently no migration path between Nova networking and OpenStack Networking. This would impact an operator's plan to deploy Nova networking with the intention of upgrading to OpenStack Networking at a later date. At present, any attempt to switch between these technologies would need to be performed manually, and would likely require planned outages.

Choose OpenStack Networking (neutron)

- **If you require an overlay network solution:** OpenStack Networking supports GRE or VXLAN tunneling for virtual machine traffic isolation. With GRE or VXLAN, no VLAN configuration is required on the network fabric and the only requirement from the physical network is to provide IP connectivity between the nodes. Furthermore, VXLAN or GRE allows a theoretical scale limit of 16 million unique IDs which is far beyond the 4094 limitation of 802.1q VLAN ID. Nova networking bases the network segregation on 802.1q VLANs and does not support tunneling with GRE or VXLAN.
- **If you require overlapping IP addresses between tenants:** OpenStack Networking uses the network namespace capabilities in the Linux kernel, which allows different tenants to use the same subnet range (e.g. 192.168.1/24) on the same Compute node without any risk of overlap or interference. This is suited for large multi-tenancy deployments. By comparison, Nova networking offers flat topologies that must remain mindful of subnets used by all tenants.
- **If you require a Red Hat-certified third-party OpenStack Networking plug-in:** By default, Red Hat Enterprise OpenStack Platform 5 uses the open source ML2 core plug-in with the Open vSwitch (OVS) mechanism driver. Based on the physical network fabric and other network requirements, third-party OpenStack Networking plug-ins can be deployed instead of the default ML2/Open vSwitch driver due to the pluggable architecture of OpenStack Networking. Red Hat is

constantly working to enhance our Partner Certification Program to certify more OpenStack Networking plugins against Red Hat Enterprise OpenStack Platform. You can learn more about our Certification Program and the certified OpenStack Networking plug-ins at http://marketplace.redhat.com.

- **If you require VPN-as-a-service (VPNaaS), Firewall-as-a-service (FWaaS), or Load-Balancing-as-a-service (LBaaS):** These network services are only available in OpenStack Networking and are not available for Nova networking. The dashboard allows tenants to manage these services with no need for administrator intervention.

Choose Nova networking

- **If your deployment requires flat (untagged) or VLAN (802.1q tagged) networking:**This implies scalabilty requirements (theoretical scale limit of 4094 VLAN IDs, where in practice physical switches tend to support a much lower number) as well as management and provisioning requirements. Specific configuration is necessary on the physical network to trunk the required set of VLANs between the nodes.
- **If your deployment does not require overlapping IP addresses between tenants:** This is usually suitable only for small, private deployments.
- **If you do not need a software-defined networking (SDN) solution, or the ability to interact with the physical network fabric.**
- **If you do not need self-service VPN, Firewall, or Load-Balancing services.**

7.1.9. Configure the L2 Population mechanism driver

The L2 Population driver enables broadcast, multicast, and unicast traffic to scale out on large overlay networks. By default, Open vSwitch GRE and VXLAN replicate broadcasts to every agent, including those that do not host the destination network. This design requires the acceptance of significant network and processing overhead. The alternative design introduced by the L2 Population driver implements a partial mesh for ARP resolution and MAC learning traffic. This traffic is sent only to the necessary agent by encapsulating it as a targeted unicast.

Enable the L2 population driver by adding it to the list of mechanism drivers. You also need to have at least one tunneling driver enabled; either GRE, VXLAN, or both. Add the appropriate configuration options to the **`ml2_conf.ini`** file:

```
[ml2]
type_drivers = local,flat,vlan,gre,vxlan
mechanism_drivers = openvswitch,linuxbridge,l2population
```

Enable L2 population in the **`ovs_neutron_plugin.ini`** file. This must be enabled on each node running the L2 agent:

```
[agent]
l2_population = True
```

7.1.10. OpenStack Networking Agents

As well as the OpenStack Networking service and the installed plug-in a number of components combine to provide networking functionality in the OpenStack environment.

L3 Agent

The L3 agent is part of the *openstack-neutron* package. It acts as an abstract L3 router that can connect to and provide gateway services for multiple L2 networks.

The nodes on which the L3 agent is to be hosted must not have a manually configured IP address on a network interface that is connected to an external network. Instead there must be a range of IP addresses from the external network that are available for use by OpenStack Networking. These IP addresses will be assigned to the routers that provide the link between the internal and external networks.

The range selected must be large enough to provide a unique IP address for each router in the deployment as well as each desired floating IP.

DHCP Agent

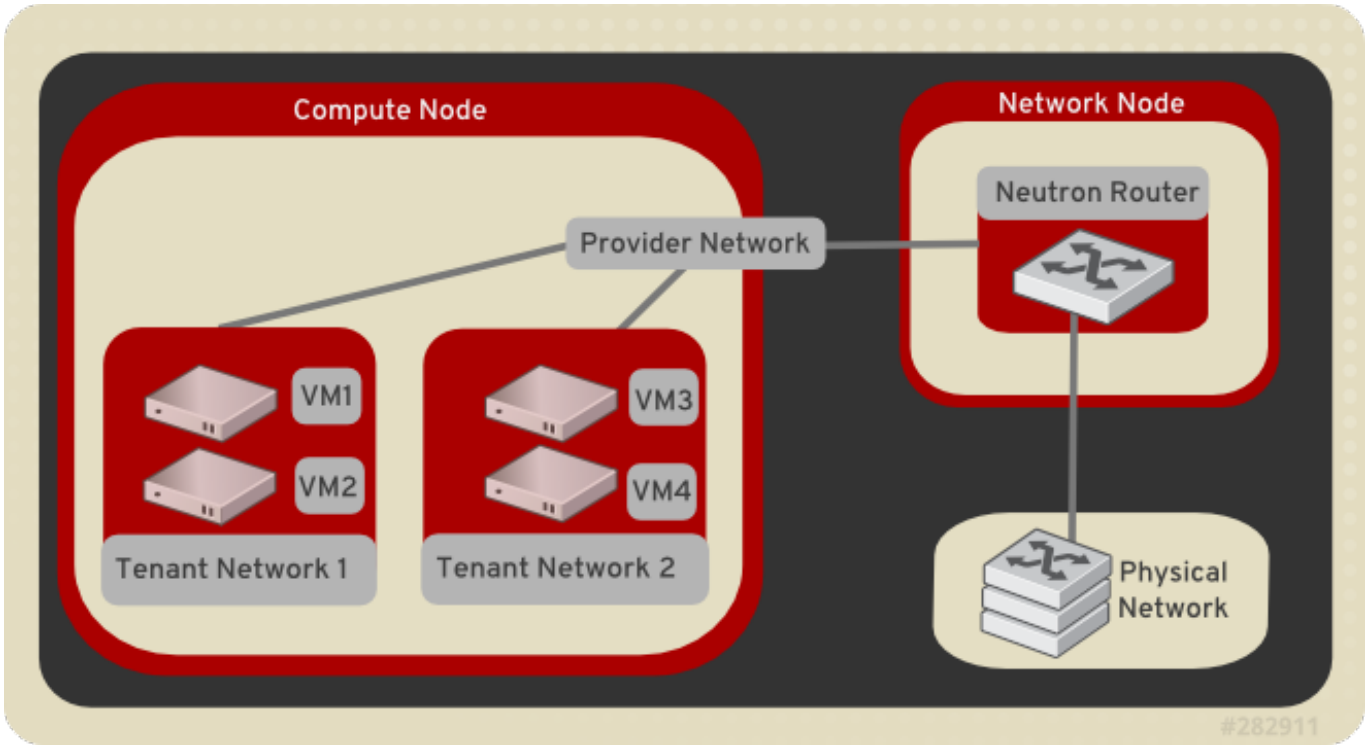
The OpenStack Networking DHCP agent is capable of allocating IP addresses to virtual machines running on the network. If the agent is enabled and running when a subnet is created then by default that subnet has DHCP enabled.

Plug-in Agent

Many of the OpenStack Networking plug-ins, including Open vSwitch and Linux Bridge, utilize their own agent. The plug-in specific agent runs on each node that manages data packets. This includes all compute nodes as well as nodes running the dedicated agents **`neutron-dhcp-agent`** and **`neutron-l3-agent`**.

7.1.11. Tenant and Provider networks

The following diagram presents an overview of the tenant and provider network types, and illustrates how they interact within the overall OpenStack Networking topology:



Tenant networks

Tenant networks are created by users for connectivity within projects; they are fully isolated by default and are not shared with other projects. OpenStack Networking supports a range of tenant network types:

Flat

All instances reside on the same network, which can also be shared with the hosts. No VLAN tagging or other network segregation takes place.

Local

Instances reside on the local Compute host and are effectively isolated from any external networks.

VLAN

OpenStack Networking allows users to create multiple provider or tenant networks using VLAN IDs (802.1Q tagged) that correspond to VLANs present in the physical network. This allows instances to communicate with each other across the environment. They can also communicate with dedicated servers, firewalls, load balancers and other network infrastructure on the same layer 2 VLAN.

VXLAN/GRE

VXLAN and GRE use network overlays to support private communication between instances. An OpenStack Networking router is required to enable traffic to traverse outside of the GRE or VXLAN tenant network. A router is also required to connect directly-connected tenant networks with external networks, including the Internet; the router provides the ability to connect to instances directly from an external network using floating IP addresses.

Provider networks

Provider networks are created by the OpenStack administrator and map directly to an existing physical network in the data center. Useful network types in this category are flat (untagged) and VLAN (802.1Q tagged). It is possible to allow provider networks to be shared among tenants as part of the network creation process.

7.1.12. Multiple Networks on a Single Node

Multiple external networks can now be operated on a single OpenStack Networking node running either the ML2 Open vSwitch mechanism driver, or the deprecated Open vSwitch plug-in.

Procedure 7.1. Create multiple Provider networks

1. Create an Open vSwitch bridge, which will provide connectivity to the new external network using **eth1**:

```
# ovs-vsctl add-br br-eth1
```

```
# ovs-vsctl add-port br-eth1 eth1
```

```
# ip link set eth1 up
```

2. The bridge and router interfaces are automatically mapped by the L3 agent, by relaying any physnet-to-bridge mappings created in **/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini**. For example, in tunneling mode, the L3 agent will patch **br-int** to the external bridges, and set the external **q** router interfaces on **br-int**. Enable this behavior in **/etc/neutron/l3_agent.ini**, by setting the options below as empty values:

```
gateway_external_network_id =
external_network_bridge =
```

On the OpenStack Networking nodes, edit **ovs_neutron_plugin.ini** to map the physical networks to the corresponding external bridges.

```
bridge_mappings = physnet1:br-ex,physnet2:br-eth1
```

Restart the L3 agent and Open vSwitch agent services for the changes to take effect:

```
# service neutron-l3-agent restart
```

```
# service neutron-openvswitch-agent restart
```

The following steps demonstrate the creation of two external networks on the same OpenStack Networking node. Ensure the **l3_agent.ini** configuration is present before proceeding.

3. Create the new Provider network **physnet1**, and the related topology:

```
# neutron net-create ext_net --provider:network_type flat --
provider:physical_network physnet1 --router:external=True
```

```
# neutron subnet-create ext_net --gateway 172.16.0.1
172.16.0.0/24 -- --enable_dhcp=False
```

```
# neutron router-create router1
```

```
# neutron router-gateway-set router1 ext_net
```

```
# neutron net-create privnet
```

```
# neutron subnet-create privnet --gateway 192.168.123.1
192.168.123.0/24 --name privnet_subnet
```

```
# neutron router-interface-add router1 privnet_subnet
```

4. Create the new Provider network **physnet2**, and the related topology:

```
# neutron net-create ext_net2 --provider:network_type flat --
provider:physical_network physnet2 --router:external=True
```

```
# neutron subnet-create ext_net2 --allocation-pool
start=192.168.1.200,end=192.168.1.222 --gateway=192.168.1.1 --
enable_dhcp=False 192.168.1.0/24
```

```
# neutron router-create router2
```

```
# neutron router-gateway-set router2 ext_net2
```

```
# neutron net-create privnet2
```

```
# neutron subnet-create privnet2 --gateway 192.168.125.1
192.168.125.0/24 --name privnet2_subnet
```

```
# neutron router-interface-add router2 privnet2_subnet
```

7.1.13. Recommended OpenStack Networking Deployment

OpenStack Networking provides an extreme amount of flexibility when deploying networking in support of a compute environment. As a result, the exact layout of a deployment will depend on a combination of expected workloads, expected scale, and available hardware.

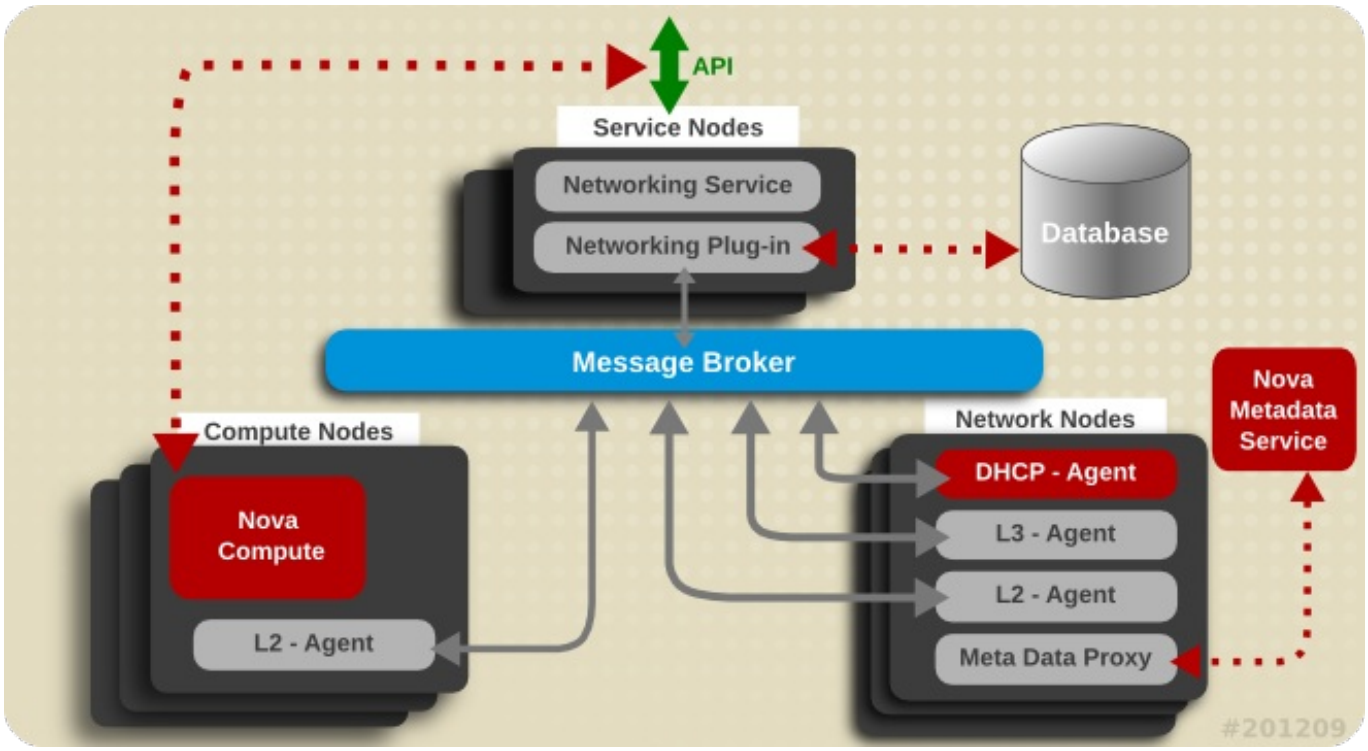


Figure 7.3. Deployment Architecture

For demonstration purposes, this chapter concentrates on a OpenStack Networking deployment that consists of these types of nodes:

Service Node

The service node exposes the OpenStack Networking API to clients and handles incoming requests before forwarding them to a message queue to be actioned by the other nodes. The service node hosts both the OpenStack Networking service itself and the active OpenStack Networking plug-in.

In environments that use controller nodes to host the client-facing APIs and schedulers for all services, the controller node would also fulfil the role of service node as it is applied in this chapter.

Network Node

The network node handles the majority of the OpenStack Networking workload. It hosts the DHCP agent, the Layer 3 (L3) agent, the Layer 2 (L2) Agent, and the metadata proxy. In addition to plug-ins that require an agent, it runs an instance of the plug-in agent (as do all other systems that handle data packets in an environment where such plug-ins are in use). Both the Open vSwitch and Linux Bridge plug-ins include an agent.

Compute Node

The Compute hosts run the Compute instances themselves. To connect Compute instances to the OpenStack Networking services, Compute nodes must also run the L2 agent. Like all other systems that handle data packets it must also run an instance of the plug-in agent.

This deployment type and division of responsibilities is made only as a suggestion. Other divisions are equally valid, in particular in some environments the network and service nodes may be combined.

Warning

Environments that have been configured to use nova-network, either using the **packstack** utility or manually, can be reconfigured to use OpenStack Networking. This is however currently **not** recommended for environments where Compute instances have already been created and configured to use nova-network. If you wish to proceed with such a conversion, stop the **openstack-nova-network** service on each Compute node first; to do so, run:

```
# service openstack-nova-network stop
```

You must also disable the **openstack-nova-network** service permanently on each node using the **chkconfig** command:

```
# chkconfig openstack-nova-network off
```

Important

When running 2 or more active controller nodes, do not run **nova-consoleauth** on more than one node. Running more than one instance of **nova-consoleauth** causes a conflict between nodes with regard to token requests which may cause errors.

7.1.14. Kernel Requirements

Nodes that handle OpenStack Networking traffic require a Red Hat Enterprise Linux kernel that supports the use of network namespaces. In addition, the Open vSwitch plug-in requires kernel version **2.6.32-431.el6.x86_64** or later.

The default kernel included in supported Red Hat Enterprise Linux versions fulfills both requirements.

Previous Red Hat Enterprise Linux OpenStack Platform releases required manually installing and booting to a custom Red Hat Enterprise Linux kernel that supported network namespaces and other OpenStack features. As of this release, doing so is no longer required. All OpenStack Networking requirements are now built into the default kernel of supported Red Hat Enterprise Linux versions.

7.2. Networking Prerequisite Configuration

7.2.1. Configure OpenStack Networking Authentication

This section outlines the steps for creating and configuring Identity service records required by the OpenStack Networking service.

1. Create the **neutron** user, who has the **admin** role in the **services** tenant.
2. Create the **neutron** service entry and assign it an endpoint.

These entries will assist other OpenStack services attempting to locate and access the network functionality provided by the OpenStack Networking service. In order to proceed, you should have already performed the following (through the Identity service):

1. Created an Administrator role named **admin** (refer to Section 3.7, "Create an Administrator Account" for instructions)
2. Created the **services** tenant (refer to Section 3.9, "Create the Services Tenant" for instructions)

Note

The *Deploying OpenStack: Learning Environments (Manual Set Up)* guide uses one tenant for all service users. For more information, refer to Section 3.9, "Create the Services Tenant".

You can perform the following procedure from your Identity service host or on any machine where you've copied the **keystonerc_admin** file (which contains administrator credentials) and the **keystone** command-line utility is installed.

Procedure 7.2. Configuring OpenStack Networking to authenticate through the Identity Service

1. Authenticate as the administrator of the Identity service by running the **source** command on the **keystonerc_admin** file containing the required credentials:

```
# source ~/keystonerc_admin
```

2. Create a user named **neutron** for the OpenStack Networking service to use:

```
# keystone user-create --name neutron --pass PASSWORD
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|  email   |                                  |
| enabled  |               True               |
|    id    | 1df18bcd14404fa9ad954f9d5eb163bc |
|   name   |              neutron             |
| tenantId |                                  |
+----------+----------------------------------+
```

Replace *PASSWORD* with a secure password that will be used by the OpenStack Networking service when authenticating with the Identity service.

3. Use the **keystone user-role-add** command to link the **neutron** user, **admin** role, and **services** tenant together:

```
# keystone user-role-add --user neutron --role admin --tenant
services
```

4. Create the **neutron** service entry:

```
# keystone service-create --name neutron \
       --type network \
       --description "OpenStack Networking Service"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |   OpenStack Networking Service   |
|      id     | 134e815915f442f89c39d2769e278f9b |
|     name    |              neutron             |
|     type    |              network             |
+-------------+----------------------------------+
```

5. Create the **network** endpoint entry:

```
# keystone endpoint-create --service-id SERVICE_ID \
       --service neutron \
       --publicurl "http://IP:9696" \
       --adminurl "http://IP:9696" \
       --internalurl "http://IP:9696"
```

   Replace SERVICE_ID with the service identifier from the previous step, and replace *IP* with the IP address or host name of the system that will be acting as the network node.

All supporting Identity service entries required by the OpenStack Networking service have been created.

7.3. Common Networking Configuration

7.3.1. Disable Network Manager

OpenStack Networking currently does not work on systems that have the Network Manager service enabled.

Follow the steps listed in this procedure while logged in as the **root** user on each system in the environment that will handle network traffic. This includes the system that will host the OpenStack Networking service, all network nodes, and all compute nodes.

These steps determine the state of the **NetworkManager** service, disable it from running, and replace it with the standard network service:

Procedure 7.3. Disabling the Network Manager service

1. Verify Network Manager is currently enabled:

```
# systemctl status NetworkManager.service | grep Active:
```

A. The system displays an error if the Network Manager service is not currently installed:

```
error reading information on service NetworkManager: No such file
or directory
```

If this error is displayed then no further action is required to disable the Network Manager service.

B. The system displays **`Active: active (running)`** if Network Manager is running, or **`Active: inactive (dead)`** if it is not.

For example, if Network Manager is active:

```
Active: active (running) since Thu 2014-06-26 19:34:00 EDT; 2s
ago
```

If Network Manager is inactive, then no further action is required.

2. If Network Manager is running, then you must first stop it and then disable it:

```
# systemctl stop NetworkManager.service
# systemctl disable NetworkManager.service
```

3. Open each interface configuration file on the system in a text editor. Interface configuration files are found in the **`/etc/sysconfig/network-scripts/`** directory and have names of the form **`ifcfg-X`** where *X* is replaced by the name of the interface. Valid interface names include **`eth0`**, **`p1p5`**, and **`em1`**.

In each file ensure that the **`NM_CONTROLLED`** configuration key is set to **`no`** and the **`ONBOOT`** configuration key is set to **`yes`**. Add these keys manually if they do not already exist in each file.

```
NM_CONTROLLED=no
ONBOOT=yes
```

This action ensures that the standard network service will take control of the interfaces and automatically activate them on boot.

4. Ensure that the standard network service is started using the **`systemctl`** command:

```
# systemctl start network.service
```

5. Ensure the network service is enabled:

```
# systemctl enable network.service
```

The Network Manager service is disabled. The standard network service is enabled and configured to control the required network interfaces.

7.3.2. Disable firewalld

Disable the firewalld service for Compute and OpenStack Networking nodes running Red Hat Enterprise Linux 7.1, and replace with iptables.

Procedure 7.4. Disable the firewalld service

1. Install the iptables service:

```
# yum install iptables-services
```

2. Review the iptables rules defined in **/etc/sysconfig/iptables**

Note

You can review your current firewalld configuration by running the **firewall-cmd** command.

```
# firewall-cmd --list-all
```

3. Once satisfied with the iptables rules, disable firewalld:

```
# systemctl disable firewalld.service
```

4. Stop the firewalld service and start iptables:

```
# systemctl stop firewalld.service; systemctl start iptables;
systemctl start ip6tables
```

5. Permanently enable the iptables services:

```
# systemctl enable iptables
```

```
# systemctl enable ip6tables
```

The firewalld service has been disabled, and the iptables service has been enabled and configured to start on boot.

7.3.3. Install the OpenStack Networking Service Packages

The OpenStack Networking service requires the following packages:

openstack-neutron

Provides the OpenStack Networking service and associated configuration files.

openstack-neutron-PLUGIN

Provides an OpenStack Networking plug-in. Replace *PLUGIN* with one of the recommended plug-ins (**openvswitch** and **linuxbridge**).

openstack-utils

Provides supporting utilities to assist with a number of tasks including the editing of configuration files.

openstack-selinux

Provides OpenStack specific SELinux policy modules.

The packages must be installed on all systems that will handle network traffic. This includes the OpenStack Networking service node, all network nodes, and all Compute nodes.

To install all of the above packages, execute the following command while logged in as the **root** user:

```
# yum install -y openstack-neutron \
   openstack-neutron-PLUGIN \
   openstack-utils \
   openstack-selinux
```

Replace *PLUGIN* with **openvswitch** or **linuxbridge** (determines which plug-in is installed).

The OpenStack Networking services are installed and ready to be configured.

7.3.4. Configure the Firewall to Allow OpenStack Networking Traffic

Remote systems requiring integration with the OpenStack Networking service will need to be granted access to **TCP** port **9696**.

All steps in this procedure must be run while logged in to the server hosting the OpenStack Networking service as the **root** user.

Procedure 7.5. Configuring the firewall to allow OpenStack Networking traffic

1. Open the **/etc/sysconfig/iptables** file in a text editor.
2. Add an INPUT rule allowing TCP traffic on port **9696** to the file. The new rule must appear before any INPUT rules that REJECT traffic.

   ```
   -A INPUT -p tcp -m multiport --dports 9696 -j ACCEPT
   ```

3. Save the changes to the **/etc/sysconfig/iptables** file.
4. Restart the **iptables** service to ensure that the change takes effect.

   ```
   # service iptables restart
   ```

The firewall is now configured to allow incoming connections to the OpenStack Networking service on port **9696**.

7.4. Configure the Networking Service

7.4.1. Configure OpenStack Networking Service Authentication

The OpenStack Networking service must be explicitly configured to use the Identity service for authentication. To accomplish this, perform the following procedure on the network node while logged in as **root** on the DHCP agent's host:

Procedure 7.6. Configuring the OpenStack Networking Service to authenticate through the Identity Service

1. Set the authentication strategy (**auth_strategy**) configuration key to **keystone** using the **openstack-config** command.

```
# openstack-config --set /etc/neutron/neutron.conf \
  DEFAULT auth_strategy keystone
```

2. Set the authentication host (**auth_host** configuration key) to the IP address or host name of the Identity server.

```
# openstack-config --set /etc/neutron/neutron.conf \
  keystone_authtoken auth_host IP
```

   Replace *IP* with the IP address or host name of the Identity server.

3. Set the administration tenant name (**admin_tenant_name**) configuration key to the name of the tenant that was created for the use of the OpenStack Networking service. Examples in this guide use *services*.

```
# openstack-config --set /etc/neutron/neutron.conf \
  keystone_authtoken admin_tenant_name services
```

4. Set the administration user name (**admin_user**) configuration key to the name of the user that was created for the use of the OpenStack Networking services. Examples in this guide use *neutron*.

```
# openstack-config --set /etc/neutron/neutron.conf \
  keystone_authtoken admin_user neutron
```

5. Set the administration password (**admin_password**) configuration key to the password that is associated with the user specified in the previous step.

```
# openstack-config --set /etc/neutron/neutron.conf \
  keystone_authtoken admin_password PASSWORD
```

The authentication keys used by the OpenStack Networking service have been set and will be used when the services are started.

7.4.2. Configure RabbitMQ Message Broker Settings for the Networking Service

As of Red Hat Enterprise Linux OpenStack Platform 5, RabbitMQ replaces Qpid as the default (and recommended) message broker. The RabbitMQ messaging service is provided by the *rabbitmq-server* package.

This section assumes that you have already configured a RabbitMQ message broker. For more information, refer to:

- Section 2.4, “Prerequisite Message Broker”
- Section 2.4.2, “Install and Configure the RabbitMQ Message Broker”
- Section 2.4.1, “Configure the Firewall for Message Broker Traffic”
- Section 2.4.2.1, “Launch the RabbitMQ Message Broker”

Procedure 7.7. Configuring the OpenStack Networking service to use the RabbitMQ message broker

Perform these steps on all Red Hat Enterprise Linux OpenStack Platform nodes.

1. In **/etc/neutron/neutron.conf**, set RabbitMQ as the RPC back end.

```
# openstack-config --set /etc/neutron/neutron.conf \
  DEFAULT rpc_backend neutron.openstack.common.rpc.impl_kombu
```

2. Set the **neutron-server** service to connect to the RabbitMQ host:

```
# openstack-config --set /etc/neutron/neutron.conf \
  DEFAULT rabbit_host RABBITMQ_HOST
```

   Replace *RABBITMQ_HOST* with the IP address or host name of the message broker.

3. Set the message broker port to **5672**:

```
# openstack-config --set /etc/neutron/neutron.conf \
  DEFAULT rabbit_port 5672
```

4. Set the RabbitMQ username and password created for the OpenStack Networking service:

```
# openstack-config --set /etc/neutron/neutron.conf \
  DEFAULT rabbit_userid neutron
```

```
# openstack-config --set /etc/neutron/neutron.conf \
  DEFAULT rabbit_password NEUTRON_PASS
```

   Where **neutron** and *NEUTRON_PASS* are the RabbitMQ username and password created for OpenStack Networking (in Section 2.4.2, “Install and Configure the RabbitMQ Message Broker”).

5. In Section 2.4.2.1, “Launch the RabbitMQ Message Broker”, we gave the **neutron** user read/write permissions to all resources -- specifically, through the virtual host **/**. Configure the Networking service to connect to this virtual host:

```
# openstack-config --set /etc/neutron/neutron.conf \
  DEFAULT rabbit_virtual_host /
```

7.4.2.1. Enable SSL Communication Between the OpenStack Networking Service and Message Broker

If you enabled SSL on the message broker, you will also have to configure the OpenStack Networking service accordingly. For this, you will need the exported client certificates and key file. See Section 2.4.2.3, “Export an SSL Certificate for Clients” for instructions on how to export these files.

Once you have the necessary certificates and key, run the following commands to enable SSL communication with the message broker:

```
# openstack-config --set /etc/neutron/neutron.conf \
 DEFAULT rabbit_use_ssl True
```

```
# openstack-config --set /etc/neutron/neutron.conf \
 DEFAULT kombu_ssl_certfile /path/to/client.crt
# openstack-config --set /etc/neutron/neutron.conf \
 DEFAULT kombu_ssl_keyfile /path/to/clientkeyfile.key
```

Where:

- */path/to/client.crt* is the absolute path to the exported client certificate.
- */path/to/clientkeyfile.key* is the absolute path to the exported client key file.

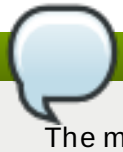
If your certificates were signed by a third-party Certificate Authority (CA), then you will also need to run the following command:

```
# openstack-config --set /etc/neutron/neutron.conf \
 DEFAULT kombu_ssl_ca_certs /path/to/ca.crt
```

Replace */path/to/ca.crt* with the absolute path to the CA file provided by the third-party CA (see Section 2.4.2.2, “Enable SSL on the RabbitMQ Message Broker” for more information).

7.4.3. Set the OpenStack Networking Service Plug-in

Additional configuration settings must be applied to enable the desired plug-in. Below are the procedures for enabling the ML2, Open vSwitch (OVS), and Linux Bridge plug-ins.



Note

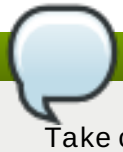
The monolithic Open vSwitch and linuxbridge plug-ins have been deprecated and will be removed in a future release; their functionality has instead been re-implemented as ML2 mechanisms.

OpenStack Networking plug-ins can be referenced in **neutron.conf** by their nominated short names, instead of their lengthy class names. For example:

```
core_plugin = neutron.plugins.ml2.plugin:Ml2Plugin
```

will become:

```
core_plugin = ml2
```



Note

Take care not to introduce errant whitespace characters, as these could result in parse errors.

Table 7.1. core_plugin

Short name	Class name
bigswitch	neutron.plugins.bigswitch.plugin:NeutronRestProxyV2
brocade	neutron.plugins.brocade.NeutronPlugin:BrocadePluginV2

Short name	Class name
cisco	neutron.plugins.cisco.network_plugin:PluginV2
embrane	neutron.plugins.embrane.plugins.embrane_ovs_plugin:EmbraneOvsPlugin
hyperv	neutron.plugins.hyperv.hyperv_neutron_plugin:HyperVNeutronPlugin
linuxbridge	neutron.plugins.linuxbridge.lb_neutron_plugin:LinuxBridgePluginV2
midonet	neutron.plugins.midonet.plugin:MidonetPluginV2
ml2	neutron.plugins.ml2.plugin:Ml2Plugin
mlnx	neutron.plugins.mlnx.mlnx_plugin:MellanoxEswitchPlugin
nec	neutron.plugins.nec.nec_plugin:NECPluginV2
openvswitch	neutron.plugins.openvswitch.ovs_neutron_plugin:OVSNeutronPluginV2
plumgrid	neutron.plugins.plumgrid.plumgrid_plugin.plumgrid_plugin:NeutronPluginPLUMgridV2
ryu	neutron.plugins.ryu.ryu_neutron_plugin:RyuNeutronPluginV2
vmware	neutron.plugins.vmware.plugin:NsxPlugin

The **service_plugins** option accepts a comma-delimited list of multiple service plugins.

Table 7.2. service_plugins

Short name	Class name
dummy	neutron.tests.unit.dummy_plugin:DummyServicePlugin
router	neutron.services.l3_router.l3_router_plugin:L3RouterPlugin
firewall	neutron.services.firewall.fwaas_plugin:FirewallPlugin
lbaas	neutron.services.loadbalancer.plugin:LoadBalancerPlugin
metering	neutron.services.metering.metering_plugin:MeteringPlugin

Procedure 7.8. Enabling the ML2 plug-in

Follow these steps on the node running the **neutron-server** service.

1. Install the openstack-neutron-ml2 package:

```
# yum install openstack-neutron-ml2
```

2. Create a symbolic link to direct OpenStack Networking to the ML2 config file **ml2_conf.ini**:

```
# ln -s /etc/neutron/plugins/ml2/ml2_conf.ini
/etc/neutron/plugin.ini
```

3. Add the appropriate configuration options to the **ml2_conf.ini** file. Available options are listed below. Refer to Section 7.1.7, “Modular Layer 2 (ML2) Overview” for further information on these settings.

```
[ml2]
type_drivers = local,flat,vlan,gre,vxlan
mechanism_drivers = openvswitch,linuxbridge,l2population
[agent]
l2_population = True
```

4. Enable the ML2 plug-in and L3 router in the **neutron.conf** file:

```
core_plugin = neutron.plugins.ml2.plugin.Ml2Plugin
service_plugins =
neutron.services.l3_router.l3_router_plugin.L3RouterPlugin
```

5. Refer to Section 7.4.7, "Create the OpenStack Networking Database" to configure the ML2 database.

6. Restart the OpenStack Networking service:

```
# service neutron-server restart
```

Procedure 7.9. Enabling the Open vSwitch plug-in

Follow these steps on the node running the **neutron-server** service.

Note

The monolithic Open vSwitch plug-in has been deprecated and will be removed in a future release; its functionality has instead been re-implemented as a ML2 mechanism.

1. Create a symbolic link between the **/etc/neutron/plugin.ini** path referred to by the OpenStack Networking service and the plug-in specific configuration file.

```
# ln -s /etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini
\
  /etc/neutron/plugin.ini
```

2. Update the value of the **tenant_network_type** configuration key in the **/etc/neutron/plugin.ini** file to refer to the type of network that must be used for tenant networks. Supported values are **gre**, **local**, **vlan**, and **vxlan**.

   The default is **local** but this is not recommended for real deployments.

   Open vSwitch Tunneling allows virtual machines across multiple hosts to share a single layer 2 network. GRE and VXLAN tunnels are supported for encapsulating traffic between Open vSwitch endpoints on each host. Ensure that MTUs are an appropriate size from end-to-end, including those on the virtual machines.

```
# openstack-config --set /etc/neutron/plugin.ini \
  OVS tenant_network_type TYPE
```

   Replace *TYPE* with the type chosen tenant network type.

3. If **flat** or **vlan** networking was chosen, the value of the **network_vlan_ranges** configuration key must also be set. This configuration key maps physical networks to VLAN ranges.

   Mappings are of the form *NAME*:*START*:*END* where *NAME* is replaced by the name of the physical network, *START* is replaced by the VLAN identifier that starts the range, and *END* is replaced by the replaced by the VLAN identifier that ends the range.

```
# openstack-config --set /etc/neutron/plugin.ini \
  OVS network_vlan_ranges NAME:START:END
```

Multiple ranges can be specified using a comma separated list, for example:

```
physnet1:1000:2999,physnet2:3000:3999
```

4. Update the value of the **core_plugin** configuration key in the **/etc/neutron/neutron.conf** file to refer to the Open vSwitch plug-in.

```
# openstack-config --set /etc/neutron/neutron.conf \
  DEFAULT core_plugin \

neutron.plugins.openvswitch.ovs_neutron_plugin.OVSNeutronPluginV
2
```

If you are using a Linux Bridge plug-in, perform the following procedure instead:

Procedure 7.10. Enabling the Linux Bridge plug-in

Follow these steps on the node running the **neutron-server** service.

Note

The monolithic linuxbridge plug-in has been deprecated and will be removed in a future release; its functionality has instead been re-implemented as a ML2 mechanism.

1. Create a symbolic link between the **/etc/neutron/plugin.ini** path referred to by the OpenStack Networking service and the plug-in specific configuration file.

```
# ln -s /etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini \
       /etc/neutron/plugin.ini
```

2. Update the value of the **tenant_network_type** configuration key in the **/etc/neutron/plugin.ini** file to refer to the type of network that must be used for tenant networks. Supported values are **flat**, **vlan**, and **local**.

   The default is **local** but this is not recommended for real deployments.

```
# openstack-config --set /etc/neutron/plugin.ini \
  VLAN tenant_network_type TYPE
```

   Replace *TYPE* with the type chosen tenant network type.

3. If **flat** or **vlan** networking was chosen, the value of the **network_vlan_ranges** configuration key must also be set. This configuration key maps physical networks to VLAN ranges.

   Mappings are of the form *NAME*:*START*:*END* where *NAME* is replaced by the name of the physical network, *START* is replaced by the VLAN identifier that starts the range, and *END* is replaced by the replaced by the VLAN identifier that ends the range.

```
# openstack-config --set /etc/neutron/plugin.ini \
  LINUX_BRIDGE network_vlan_ranges NAME:START:END
```

Multiple ranges can be specified using a comma separated list, for example:

```
physnet1:1000:2999,physnet2:3000:3999
```

4. Update the value of the **core_plugin** configuration key in the **/etc/neutron/neutron.conf** file to refer to the Linux Bridge plug-in.

```
# openstack-config --set /etc/neutron/neutron.conf \
  DEFAULT core_plugin \

neutron.plugins.linuxbridge.lb_neutron_plugin.LinuxBridgePlugin
V2
```

7.4.4. VXLAN and GRE tunnels

Overlay links enable instances in different networks to communicate with each other by tunneling traffic through the underlying network. **GRE** and **VXLAN** are two supported tunnel encapsulation technologies:

GRE is an established encapsulation technology, with general acceptance in the industry. The GRE headers (RFC 2784 and 2890) perform encapsulation of Layer 2 frames.

VXLAN was submitted to the IETF in 2011 and is expected to become the default standard for multitenant cloud-scale networks. VXLAN uses UDP encapsulation and random values in the UDP source port, resulting in automatic equal-cost load balancing in every switch device that uses a 5-tuple hash calculation to load balance. This allows VXLAN-encapsulated traffic to be load balanced by compatible physical network hardware.

Table 7.3. VXLAN and GRE comparison

Feature	VXLAN	GRE
Segmentation	24-bit VNI (VXLAN Network Identifier)	Uses different Tunnel IDs
Theoretical scale limit	16 million unique IDs	16 million unique IDs
Transport	UDP (default port 4789)	IP Protocol 47
Filtering	VXLAN uses UDP with a well-known destination port; firewalls and switch/router ACLs can be tailored to block only VXLAN traffic.	Firewalls and layer 3 switches and routers with ACLs will typically not parse deeply enough into the GRE header to distinguish tunnel traffic types; all GRE would need to be blocked indiscriminately.
Protocol overhead	50 bytes over IPv4 (8 bytes VXLAN header, 20 bytes IPv4 header, 8 bytes UDP header, 14 bytes Ethernet).	42 bytes over IPv4 (8 bytes GRE header, 20 bytes IPv4 header, 14 bytes Ethernet).

Feature	VXLAN	GRE
Handling unknown destination packets, broadcasts, and multicast	VXLAN uses IP multicast to manage flooding from these traffic types. Ideally, one logical Layer 2 network (VNI) is associated with one multicast group address on the physical network. This requires end-to-end IP multicast support on the physical data center network.	GRE has no built-in protocol mechanism to address these. This type of traffic is simply replicated to all nodes.
IETF specification	http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-01	http://tools.ietf.org/html/rfc2784.html

Note

To avoid packet fragmentation, it is recommended to increase the MTU at the vSwitch and on all physical network devices that the VXLAN or GRE traffic will traverse.

7.4.5. Configure Open vSwitch tunneling

Tunneling encapsulates network traffic between physical OpenStack Networking hosts and allows VLANs to span multiple physical hosts. Instances communicate as if they share the same layer 2 network. Open vSwitch supports tunneling with the VXLAN and GRE encapsulation protocols.

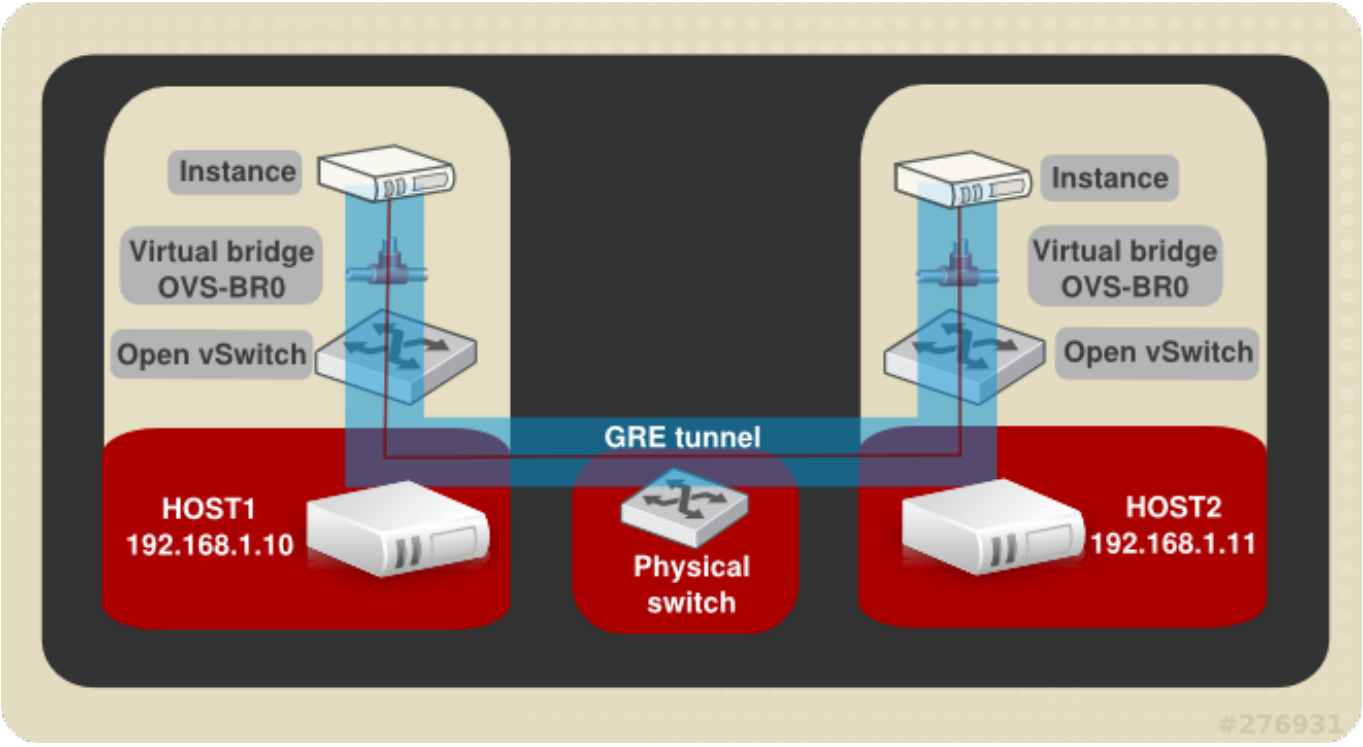


Figure 7.4. Example VXLAN tunnel

This diagram shows two instances running on separate hosts connected by a VXLAN tunnel. The required physical and virtual components are also illustrated. The following procedure creates a VXLAN or GRE tunnel between two Open vSwitches running on separate OpenStack Networking

hosts:

Procedure 7.11. Example tunnel configuration

1. Create a virtual bridge named OVS-BR0 on each participating host:

```
ovs-vsctl add-br OVS-BR0
```

2. Create a tunnel to link the OVS-BR0 virtual bridges. Run the ovs-vsctl command on HOST1 to create the tunnel and link it to the bridge on HOST2:

   GRE tunnel command:

```
ovs-vsctl add-port OVS-BR0 gre1 -- set Interface gre1 type=gre
options:remote_ip=192.168.1.11
```

   VXLAN tunnel command:

```
ovs-vsctl add-port OVS-BR0 vxlan1 -- set Interface vxlan1
type=vxlan options:remote_ip=192.168.1.11
```

3. Run the ovs-vsctl command on HOST2 to create the tunnel and link it to the bridge on HOST1.

   GRE tunnel command:

```
ovs-vsctl add-port OVS-BR0 gre1 -- set Interface gre1 type=gre
options:remote_ip=192.168.1.10
```

   VXLAN tunnel command:

```
ovs-vsctl add-port OVS-BR0 vxlan1 -- set Interface vxlan1
type=vxlan options:remote_ip=192.168.1.10
```

Successful completion of these steps results in the two instances sharing a layer 2 network.

7.4.6. Configure the OpenStack Networking Service Database Connection

The database connection string used by the OpenStack Networking service is defined in the **`/etc/neutron/plugin.ini`** file. It must be updated to point to a valid database server before starting the service.

Procedure 7.12. Configuring the OpenStack Networking SQL database connection

- Use the **`openstack-config`** command to set the value of the **`connection`** configuration key.

```
# openstack-config --set /etc/neutron/plugin.ini \
  DATABASE sql_connection mysql://USER:PASS@IP/DB
```

  Replace:

  - *USER* with the database user name the OpenStack Networking service is to use, See Section 7.4.7, “Create the OpenStack Networking Database” for recommended database naming conventions.

- *PASS* with the password of the chosen database user.
- *IP* with the IP address or host name of the database server.
- *DB* with the name of the database that will be created for use by the OpenStack Networking service (**ovs_neutron** is used as the example in the upcoming *Creating the OpenStack Networking Database* section).

Important

The IP address or host name specified in the connection configuration key must match the IP address or host name to which the neutron database user was granted access when creating the neutron database. Moreover, if the database is hosted locally and you granted permissions to 'localhost' when creating the neutron database, you must enter 'localhost'.

7.4.7. Create the OpenStack Networking Database

In this procedure the database and database user that will be used by the OpenStack Networking service will be created. These steps must be performed while logged in to the database server as the **root** user, and prior to starting the **neutron-server** service.

Procedure 7.13. Creating the OpenStack Networking database

1. Connect to the database service using the **mysql** command.

```
# mysql -u root -p
```

2. Create the database. If you intend to use the:
   - ML2 plug-in, the recommended database name is **neutron_ml2**
   - Open vSwitch plug-in, the recommended database name is **ovs_neutron**.
   - Linux Bridge plug-in, the recommended database name is **neutron_linux_bridge**.

   This example creates the ML2 **neutron_ml2** database.

```
mysql> CREATE DATABASE neutron_ml2 character set utf8;
```

3. Create a **neutron** database user and grant it access to the **neutron_ml2** database.

```
mysql> "GRANT ALL ON neutron_ml2.* TO 'neutron'@'%';"
```

4. Flush the database privileges to ensure that they take effect immediately.

```
mysql> FLUSH PRIVILEGES;
```

5. Exit the **mysql** client.

```
mysql> quit
```

6. Run the **neutron-db-manage** command:

```
# neutron-db-manage --config-file /usr/share/neutron/neutron-
dist.conf \
  --config-file /etc/neutron/neutron.conf --config-file
/etc/neutron/plugin.ini upgrade head
```

The OpenStack Networking database **neutron_ml2** has been created. The database will be populated during service configuration.

See Section 7.4.6, "Configure the OpenStack Networking Service Database Connection" to configure OpenStack Networking to use the newly created database.

See Section 7.4.3, "Set the OpenStack Networking Service Plug-in" for plug-in selection and configuration.

7.4.8. Launch the OpenStack Networking Service

Once the required settings are configured, you can now launch the OpenStack Networking service (**neutron**) using the **service** command:

```
# service neutron-server start
```

Enable the OpenStack Networking service permanently using the **chkconfig** command.

```
# chkconfig neutron-server on
```

The OpenStack Networking service is configured and running. Further action is however required to configure and run the various networking agents that are also fundamental to providing networking functionality.

Important

By default, OpenStack Networking does not enforce Classless Inter-Domain Routing (CIDR) checking of IP addresses. This is to maintain backwards compatibility with previous releases. If you require such checks set the value of the **force_gateway_on_subnet** configuration key to **True** in the **/etc/neutron/neutron.conf** file.

7.5. Configure the DHCP Agent

Follow the steps listed in this procedure to configure the DHCP Agent. All steps listed in this procedure must be performed on the network node while logged in as the **root** user on the system hosting the DHCP agent.

Procedure 7.14. Configuring the DHCP Agent

1. **Configuring Authentication**

   The DHCP agent must be explicitly configured to use the Identity service for authentication.

   a. Set the authentication strategy (**auth_strategy**) configuration key to **keystone** using the **openstack-config** command.

```
# openstack-config --set /etc/neutron/dhcp_agent.ini \
   DEFAULT auth_strategy keystone
```

b. Set the authentication host (**auth_host** configuration key) to the IP address or host name of the Identity server.

```
# openstack-config --set /etc/neutron/dhcp_agent.ini \
   keystone_authtoken auth_host IP
```

Replace *IP* with the IP address or host name of the Identity server.

c. Set the administration tenant name (**admin_tenant_name**) configuration key to the name of the tenant that was created for the use of the OpenStack Networking services. Examples in this guide use *services*.

```
# openstack-config --set /etc/neutron/dhcp_agent.ini \
   keystone_authtoken admin_tenant_name services
```

d. Set the administration user name (**admin_user**) configuration key to the name of the user that was created for the use of the OpenStack Networking services. Examples in this guide use *neutron*.

```
# openstack-config --set /etc/neutron/dhcp_agent.ini \
   keystone_authtoken admin_user neutron
```

e. Set the administration password (**admin_password**) configuration key to the password that is associated with the user specified in the previous step.

```
# openstack-config --set /etc/neutron/dhcp_agent.ini \
   keystone_authtoken admin_password PASSWORD
```

2. **Configuring the Interface Driver**

Set the value of the **interface_driver** configuration key in the **/etc/neutron/dhcp_agent.ini** file based on the OpenStack Networking plug-in being used. Execute only the configuration step that applies to the plug-in used in your environment.

A. **Open vSwitch Interface Driver**

```
# openstack-config --set /etc/neutron/dhcp_agent.ini \
   DEFAULT interface_driver
neutron.agent.linux.interface.OVSInterfaceDriver
```

B. **Linux Bridge Interface Driver**

```
# openstack-config --set /etc/neutron/dhcp_agent.ini \
   DEFAULT interface_driver
neutron.agent.linux.interface.BridgeInterfaceDriver
```

3. **Starting the DHCP Agent**

a. Use the **service** command to start the **neutron-dhcp-agent** service.

```
# service neutron-dhcp-agent start
```

b. Use the **chkconfig** command to ensure that the **neutron-dhcp-agent** service will be started automatically in the future.

```
# chkconfig neutron-dhcp-agent on
```

The DHCP agent has been configured and started.

7.6. Create an External Network

OpenStack networking provides two mechanisms for connecting the Layer 3 (L3) agent to an external network. The first, attaching it to an external bridge (**br-ex**) directly, is only supported when the Open vSwitch plug-in is in use. The second method, which is supported by both the Open vSwitch plug-in and the Linux Bridge plug-in, is to use an external provider network.

To use an external provider network it is first necessary to create one. Follow the steps outlined in this procedure while logged in to a system with the OpenStack networking client - provided by the *python-neutronclient* package installed. You must also have access to a **keystonerc_admin** file containing the authentication details of the OpenStack administrative user.

Take note of the unique identifiers generated by the steps listed in this procedure. These identifiers will be required when configuring the L3 agent.

Procedure 7.15. Creating and configuring an external network

1. Use the **source** command to load the credentials of the administrative user.

```
$ source ~/keystonerc_admin
```

2. Use the **net-create** action of the **neutron** command line client to create a new provider network.

```
$ neutron net-create EXTERNAL_NAME \
  --router:external True \
  --provider:network_type TYPE \
  --provider:physical_network PHYSNET \
  --provider:segmentation_id VLAN_TAG
```

Replace these strings with the appropriate values for your environment:

- Replace *EXTERNAL_NAME* with a name for the new external network provider.
- Replace *PHYSNET* with a name for the physical network. This is not applicable if you intend to use a local network type. *PHYSNET* must match one of the values defined under **bridge_mappings** in the **/etc/neutron/plugin.ini** file.
- Replace *TYPE* with the type of provider network you wish to use. Supported values are **flat** (for flat networks), **vlan** (for VLAN networks), and **local** (for local networks).
- Replace *VLAN_TAG* with the VLAN tag that will be used to identify network traffic. The VLAN tag specified must have been defined by the network administrator.

If the **network_type** was set to a value other than **vlan** then this parameter is not required.

Take note of the unique external network identifier returned, this will be required in subsequent steps.

3. Use the **subnet-create** action of the command line client to create a new subnet for the new external provider network.

```
$ neutron subnet-create --gateway GATEWAY \
   --allocation-pool start=IP_RANGE_START,end=IP_RANGE_END \
   --disable-dhcp EXTERNAL_NAME EXTERNAL_CIDR
```

Replace these strings with the appropriate values for your environment:

- Replace *GATEWAY* with the IP address or hostname of the system that is to act as the gateway for the new subnet.
- Replace *IP_RANGE_START* with the IP address that denotes the start of the range of IP addresses within the new subnet that floating IP addresses will be allocated from.
- Replace *IP_RANGE_END* with the IP address that denotes the end of the range of IP addresses within the new subnet that floating IP addresses will be allocated from.
- Replace *EXTERNAL_NAME* with the name of the external network the subnet is to be associated with. This must match the name that was provided to the **net-create** action in the previous step.
- Replace *EXTERNAL_CIDR* with the Classless Inter-Domain Routing (CIDR) representation of the block of IP addresses the subnet represents. An example would be **192.168.100.0/24**.

Take note of the unique subnet identifier returned, this will be required in subsequent steps.

Important

The IP address used to replace the string *GATEWAY* **must** be within the block of IP addresses specified in place of the *EXTERNAL_CIDR* string but outside of the block of IP addresses specified by the range started by *IP_RANGE_START* and ended by *IP_RANGE_END*.

The block of IP addresses specified by the range started by *IP_RANGE_START* and ended by *IP_RANGE_END* **must** also fall within the block of IP addresses specified by *EXTERNAL_CIDR*.

4. Use the **router-create** action of the **neutron** command line client to create a new router.

```
$ neutron router-create NAME
```

Replace *NAME* with the name to give the new router. Take note of the unique router identifier returned, this will be required in subsequent steps.

5. Use the **router-gateway-set** action of the **neutron** command line client to link the newly created router to the external provider network.

```
$ neutron router-gateway-set ROUTER NETWORK
```

Replace *ROUTER* with the unique identifier of the router, replace *NETWORK* with the unique identifier of the external provider network.

6. Use the **router-interface-add** action of the **neutron** command line client to link the newly created router to each private network subnet.

```
$ neutron router-interface-add ROUTER SUBNET
```

Replace *ROUTER* with the unique identifier of the router, replace *SUBNET* with the unique identifier of a private network subnet. Perform this step for each existing private network subnet to which you wish to link the router.

An external provider network has been created. Use the unique identifier of the router when configuring the L3 agent.

7.7. Configuring the Plug-in Agent

7.7.1. Configure the Open vSwitch Plug-in Agent

If you have not yet enabled the Open vSwitch plug-in, see Section 7.4.3, “Set the OpenStack Networking Service Plug-in”.

The Open vSwitch plug-in has a corresponding agent. When the Open vSwitch plug-in is in use all nodes in the environment that handle data packets must have the agent installed and configured.

This includes all compute nodes and systems hosting the dedicated DHCP and L3 agents.

Note

Open vSwitch support for TCP segmentation offload (TSO) and Generic Segmentation Offload (GSO) to VXLAN and GRE is enabled by default.

Procedure 7.16. Configuring the Open vSwitch Plug-in Agent

1. Confirm that the *openvswitch* package is installed. This is normally installed as a dependency of the *neutron-plugin-openvswitch* package.

```
# rpm -qa | grep openvswitch
openvswitch-1.10.0-1.el6.x86_64
openstack-neutron-openvswitch-2013.1-3.el6.noarc
```

2. Start the **openvswitch** service.

```
# service openvswitch start
```

3. Enable the **openvswitch** service permanently.

```
# chkconfig openvswitch on
```

4. Each host running the Open vSwitch agent also requires an Open vSwitch bridge named **br-int**. This bridge is used for private network traffic. Use the **ovs-vsctl** command to create this bridge before starting the agent.

```
# ovs-vsctl add-br br-int
```

Warning

The **br-int** bridge is required for the agent to function correctly. Once created do not remove or otherwise modify the **br-int** bridge.

5. Ensure that the **br-int** device persists on reboot by creating a **/etc/sysconfig/network-scripts/ifcfg-br-int** file with these contents:

```
DEVICE=br-int
DEVICETYPE=ovs
TYPE=OVSBridge
ONBOOT=yes
BOOTPROTO=none
```

6. Set the value of the **bridge_mappings** configuration key. This configuration key must contain a list of physical networks and the network bridges associated with them.

   The format for each entry in the comma separated list is:

```
PHYSNET:BRIDGE
```

   Where *PHYSNET* is replaced with the name of a physical network, and *BRIDGE* is replaced by the name of the network bridge.

   The physical network must have been defined in the **network_vlan_ranges** configuration variable on the OpenStack Networking server.

```
# openstack-config --set /etc/neutron/plugin.ini \
  OVS bridge_mappings MAPPINGS
```

   Replace *MAPPINGS* with the physical network to bridge mappings.

7. Use the **service** command to start the **neutron-openvswitch-agent** service.

```
# service neutron-openvswitch-agent start
```

8. Use the **chkconfig** command to ensure that the **neutron-openvswitch-agent** service is started automatically in the future.

```
# chkconfig neutron-openvswitch-agent on
```

9. Use the **chkconfig** command to ensure that the **neutron-ovs-cleanup** service is started automatically on boot. When started at boot time this service ensures that the OpenStack Networking agents maintain full control over the creation and management of tap devices.

```
# chkconfig neutron-ovs-cleanup on
```

The networking configuration has been updated to use the Open vSwitch plug-in.

7.7.2. Configure the Linux Bridge Plug-in Agent

The Linux Bridge plug-in has a corresponding agent. When the Linux Bridge plug-in is in use all nodes in the environment that handle data packets must have the agent installed and configured.

This includes all compute nodes and systems hosting the dedicated DHCP and L3 agents.

Procedure 7.17. Configuring the Linux Bridge plug-in agent

1. Set the value of the **physical_interface_mappings** configuration key. This configuration key must contain a list of physical networks and the VLAN ranges associated with them that are available for allocation to tenant networks.

   The format for each entry in the comma separated list is:

   ```
   PHYSNET:VLAN_START:VLAN_END
   ```

   Where *PHYSNET* is replaced with the name of a physical network, *VLAN_START* is replaced by an identifier indicating the start of the VLAN range, and *VLAN_END* is replaced by an identifier indicating the end of the VLAN range.

   The physical networks must have been defined in the **network_vlan_ranges** configuration variable on the OpenStack Networking server.

   ```
   # openstack-config --set /etc/neutron/plugin.ini \
     LINUX_BRIDGE physical_interface_mappings MAPPINGS
   ```

   Replace *MAPPINGS* with the physical network to VLAN range mappings.

2. Use the **service** command to start the **neutron-linuxbridge-agent** service.

   ```
   # service neutron-linuxbridge-agent start
   ```

3. Use the **chkconfig** command to ensure that the **neutron-linuxbridge-agent** service is started automatically in the future.

   ```
   # chkconfig neutron-linuxbridge-agent on
   ```

The networking configuration has been updated to use the Linux Bridge plug-in.

7.8. Configure the L3 Agent

Follow the steps listed in this procedure to configure the L3 agent. All steps listed in this procedure must be performed on the network node while logged in as the **root** user.

Procedure 7.18. Configuring the L3 Agent

1. **Configuring Authentication**

   a. Set the authentication strategy (**auth_strategy**) configuration key to **keystone** using the **openstack-config** command.

```
# openstack-config --set /etc/neutron/metadata_agent.ini \
  DEFAULT auth_strategy keystone
```

b. Set the authentication host (**auth_host**) configuration key to the IP address or host name of the Identity server.

```
# openstack-config --set /etc/neutron/metadata_agent.ini \
  keystone_authtoken auth_host IP
```

Replace *IP* with the IP address or host name of the Identity server.

c. Set the administration tenant name (**admin_tenant_name**) configuration key to the name of the tenant that was created for the use of the OpenStack Networking services. Examples in this guide use *services*.

```
# openstack-config --set /etc/neutron/metadata_agent.ini \
  keystone_authtoken admin_tenant_name services
```

d. Set the administration user name (**admin_user**) configuration key to the name of the user that was created for the use of the OpenStack Networking services. Examples in this guide use *neutron*.

```
# openstack-config --set /etc/neutron/metadata_agent.ini \
  keystone_authtoken admin_user neutron
```

e. Set the administration password (**admin_password**) configuration key to the password that is associated with the user specified in the previous step.

```
# openstack-config --set /etc/neutron/metadata_agent.ini \
  keystone_authtoken admin_password PASSWORD
```

Note

When used in **admin_password**, the $ and " " (space-bar) special characters must be properly escaped. Escape the $ in **complex$password** with a second $ to become **complex$$password**. Escape the " " (space-bar) char with quotation marks: **complex password** becomes **"complex password"**.

2. **Configuring the Interface Driver**

Set the value of the **interface_driver** configuration key in the **/etc/neutron/l3_agent.ini** file based on the OpenStack Networking plug-in being used. Execute only the configuration step that applies to the plug-in used in your environment.

A. **Open vSwitch Interface Driver**

```
# openstack-config --set /etc/neutron/l3_agent.ini \
  DEFAULT interface_driver
neutron.agent.linux.interface.OVSInterfaceDriver
```

B. **Linux Bridge Interface Driver**

```
# openstack-config --set /etc/neutron/l3_agent.ini \
  DEFAULT interface_driver
neutron.agent.linux.interface.BridgeInterfaceDriver
```

3. **Configuring External Network Access**

The L3 agent connects to external networks using either an external bridge or an external provider network. When using the Open vSwitch plug-in either approach is supported. When using the Linux Bridge plug-in only the use of an external provider network is supported. Choose the approach that is most appropriate for the environment.

A. **Using an External Bridge**

To use an external bridge you must create and configure it. Finally the OpenStack Networking configuration must be updated to use it. This must be done on each system hosting an instance of the L3 agent.

a. Use the **ovs-vsctl** command to create the external bridge named **br-ex**.

```
# ovs-vsctl add-br br-ex
```

b. Ensure that the **br-ex** device persists on reboot by creating a **/etc/sysconfig/network-scripts/ifcfg-br-ex** file with these contents:

```
DEVICE=br-ex
DEVICETYPE=ovs
TYPE=OVSBridge
ONBOOT=yes
BOOTPROTO=none
```

c. Ensure that the value of the **external_network_bridge** configuration key in the **/etc/neutron/l3_agent.ini** file is **br-ex**. This ensures that the L3 agent will use the external bridge.

```
# openstack-config --set /etc/neutron/l3_agent.ini \
  DEFAULT external_network_bridge br-ex
```

B. **Using a Provider Network**

To connect the L3 agent to external networks using a provider network you must first have created the provider network. You must also have created a subnet and router to associate with it. The unique identifier of the router will be required to complete these steps.

The value of the **external_network_bridge** configuration key in the **/etc/neutron/l3_agent.ini** file must be blank. This ensures that the L3 agent does not attempt to use an external bridge.

```
# openstack-config --set /etc/neutron/l3_agent.ini \
  DEFAULT external_network_bridge ""
```

4. **Starting the L3 Agent**

a. Use the **service** command to start the **neutron-l3-agent** service.

```
# service neutron-l3-agent start
```

b. Use the **chkconfig** command to ensure that the **neutron-l3-agent** service will be started automatically in the future.

```
# chkconfig neutron-l3-agent on
```

5. **Starting the Metadata Agent**

The OpenStack Networking metadata agent allows virtual machine instances to communicate with the compute metadata service. It runs on the same hosts as the Layer 3 (L3) agent.

a. Use the **service** command to start the **neutron-metadata-agent** service.

```
# service neutron-metadata-agent start
```

b. Use the **chkconfig** command to ensure that the **neutron-metadata-agent** service will be started automatically in the future.

```
# chkconfig neutron-metadata-agent on
```

The L3 agent has been configured and started.

6. **Enable leastrouter scheduling**

The leastrouter scheduler enumerates L3 Agent router assignment, and consequently schedules the router to the L3 Agent with the fewest routers. This differs from the ChanceScheduler behavior, which randomly selects from the candidate pool of L3 Agents. Enable the leastrouter scheduler by changing the **router_scheduler_driver** option in the **neutron.conf** file:

```
router_scheduler_driver=neutron.scheduler.l3_agent_scheduler.LeastR
outersScheduler
```

The router is scheduled once connected to a network. Unschedule the router using the **neutron** command:

```
# neutron l3-agent-router-remove [l3 node] [router]
```

Assign the router with using the **neutron** command:

```
# neutron l3-agent-router-add [l3 node] [router]
```

7.9. Validate the OpenStack Networking Installation

To begin using OpenStack Networking it is necessary to deploy networking components to compute nodes. Initial networks and routers must also be defined. It is however possible to perform basic sanity checking of the OpenStack Networking deployment by following the steps outline in this procedure.

Procedure 7.19. Validating the OpenStack Networking installation

1. **All Nodes**

   a. Verify that the customized Red Hat Enterprise Linux kernel intended for use with Red Hat Enterprise Linux OpenStack Platform is running:

   ```
   $ uname --kernel-release
   2.6.32-358.6.2.openstack.el6.x86_64
   ```

   If the kernel release value returned does not contain the string **openstack** then update the kernel and reboot the system.

   b. Ensure that the installed IP utilities support network namespaces:

   ```
   $ ip netns
   ```

   If an error indicating that the argument is not recognised or supported is returned then update the system using **yum**.

2. **Service Nodes**

   a. Ensure that the **neutron-server** service is running:

   ```
   $ openstack-status | grep neutron-server
   neutron-server:                         active
   ```

3. **Network Nodes**

   Ensure that the following services are running:

   - DHCP agent (**neutron-dhcp-agent**)
   - L3 agent (**neutron-l3-agent**)
   - Plug-in agent, if applicable (**neutron-openvswitch-agent** or **neutron-linuxbridge-agent**)
   - Metadata agent (**neutron-metadata-agent**)

   To do so, run:

   ```
   # openstack-status | grep SERVICENAME
   ```

   Replace *SERVICENAME* with the appropriate service name. For example, to check whether the L3 agent is running:

   ```
   # openstack-status | grep neutron-l3-agent
   ```

All required services on the service and network nodes are operational. Proceed to deploy some compute nodes, define networks, and define routers to begin using OpenStack Networking.

7.9.1. Troubleshoot OpenStack Networking Issues

This section discusses the different commands you can use and procedures you can follow to troubleshoot the OpenStack Networking service issues.

Debugging Networking Device

- Use the **ip a** command to display all the physical and virtual devices.
- Use the **ovs-vsctl show** command to display the interfaces and bridges in a virtual switch.
- Use the **ovs-dpctl show** command to show datapaths on the switch.

Tracking Networking Packets

- Use the **tcpdump** command to see where packets are not getting through.

```
# tcpdump -n -i INTERFACE -e -w FILENAME
```

Replace *INTERFACE* with the name of the network interface to see where the packets are not getting through. The interface name can be the name of the bridge or host Ethernet device.

The **-e** flag ensures that the link-level header is dumped (in which the vlan tag will appear).

The **-w** flag is optional. You can use it only if you want to write the output to a file. If not, the output is written to the standard output (**stdout**).

For more information about **tcpdump**, refer to its manual page by running **man tcpdump**.

Debugging Network Namespaces

- Use the **ip netns list** command to list all known network namespaces.
- Use the **ip netns exec** command to show routing tables inside specific namespaces.

```
# ip netns exec NAMESPACE_ID bash
# route -n
```

Start the **ip netns exec** command in a bash shell so that subsequent commands can be invoked without the **ip netns exec** command.

Chapter 8. OpenStack Compute Service Installation

8.1. Compute Service Requirements

8.1.1. Check for Hardware Virtualization Support

The OpenStack Compute Service requires hardware virtualization support and certain kernel modules. Follow this procedure to determine whether your system has hardware virtualization support and the correct kernel modules available.

All steps listed must be performed while logged into the system as the **root** user.

Procedure 8.1. Verifying hardware virtualization support

1. Use the **grep** command to check for the presence of the **svm** or **vmx** CPU extensions by inspecting the **/proc/cpuinfo** file generated by the kernel:

   ```
   # grep -E 'svm|vmx' /proc/cpuinfo
   ```

   If any output is shown after running this command then the CPU is hardware virtualization capable and the functionality is enabled in the system BIOS.

2. Use the **lsmod** command to list the loaded kernel modules and verify that the **kvm** modules are loaded:

   ```
   # lsmod | grep kvm
   ```

   If the output includes **kvm_intel** or **kvm_amd** then the **kvm** hardware virtualization modules are loaded and your kernel meets the module requirements for the OpenStack Compute Service.

This completes the required checks to ensure hardware virtualization support is available and enabled, and that you have the correct kernel modules loaded.

If the checks indicated that either hardware virtualization support or the required kernel modules are not available or not enabled then you must take action to either find a system that does have the required hardware virtualization support and modules, or enable it on the system that failed the checks.

8.2. Install a Compute VNC Proxy

8.2.1. Install the Compute VNC Proxy Packages

The VNC proxy is available to users of the Compute service. Two types of VNC proxy server packages are available. The *openstack-nova-novncproxy* package provides VNC support to instances through a web browser (using Websockets), while the *openstack-nova-console* package provides access to instances through a traditional VNC client (through the **openstack-nova-xvpvncproxy** service).

The console authentication service, also provided by the *openstack-nova-console* package, is used to authenticate the VNC connections. Typically the console authentication service and the proxy utilities are installed on the same host as the Compute API service.

The following steps must be performed while logged in as the **root** user.

Procedure 8.2. Installing the Compute VNC proxy packages

- Install the VNC proxy utilities and the console authentication service:

  A. Install the *openstack-nova-novncproxy* package using the **yum** command:

  ```
  # yum install -y openstack-nova-novncproxy
  ```

  B. Install the *openstack-nova-console* package using the **yum** command:

  ```
  # yum install -y openstack-nova-console
  ```

The VNC proxy packages and the console authentication service are now installed and ready for configuration.

8.2.2. Configure the Firewall to Allow Compute VNC Proxy Traffic

The node that hosts VNC access to instances must be configured to allow VNC traffic through its firewall. By default, the **openstack-nova-novncproxy** service listens on TCP port 6080 and the **openstack-nova-xvpvncproxy** service listens on TCP port 6081.

The following procedure allows traffic on TCP port 6080 to traverse through the firewall for use by the *openstack-nova-novncproxy* package:

The following steps must be performed while logged in as the **root** user.

Procedure 8.3. Configuring the firewall to allow Compute VNC proxy traffic

1. Edit the **/etc/sysconfig/iptables** file and add the following on a new line underneath the *-A INPUT -i lo -j ACCEPT* line and before any *-A INPUT -j REJECT* rules:

   ```
   -A INPUT -m state --state NEW -m tcp -p tcp --dport 6080 -j ACCEPT
   ```

2. Save the file and exit the editor.

- Similarly, when using the **openstack-nova-xvpvncproxy** service, enable traffic on TCP port 6081 with the following on a new line in the same location:

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 6081 -j ACCEPT
```

Once the file has been edited with the new firewall rule or rules, run the following commands as the **root** user to apply the changes:

```
# service iptables restart
```

```
# iptables-save
```

The firewall is now configured to allow VNC proxy traffic.

8.2.3. Configure the VNC Proxy Service

VNC access to instances is available over a web browser or with a traditional VNC client. The **/etc/nova/nova.conf** file holds the following VNC options:

- *vnc_enabled* - Default is true.
- *vncserver_listen* - The IP address to which VNC services will bind.
- *vncserver_proxyclient_address* - The IP address of the compute host used by proxies to connect to instances.
- *novncproxy_base_url* - The browser address where clients connect to instance.
- *novncproxy_port* - The port listening for browser VNC connections. Default is 6080.
- *xvpvncproxy_port* - The port to bind for traditional VNC clients. Default is 6081.

As the **root** user, use the **service** command to start the console authentication service:

```
# service openstack-nova-consoleauth start
```

Use the **chkconfig** command to permanently enable the service:

```
# chkconfig openstack-nova-consoleauth on
```

As the **root** user, use the **service** command on the nova node to start the browser-based service:

```
# service openstack-nova-novncproxy start
```

Use the **chkconfig** command to permanently enable the service:

```
# chkconfig openstack-nova-novncproxy on
```

To control access to the VNC service that uses a traditional client (non browser-based), substitute *openstack-nova-xvpvncproxy* into the previous commands.

8.2.4. Access Instances with the Compute VNC Proxy

Browse to the *novncproxy_base_url* URL provided in the **/etc/nova/nova.conf** file to access instance consoles.

The following image shows VNC access to a Fedora Linux instance with a web browser. It is provided only as an example, and settings such as IP addresses will be different in your environment.

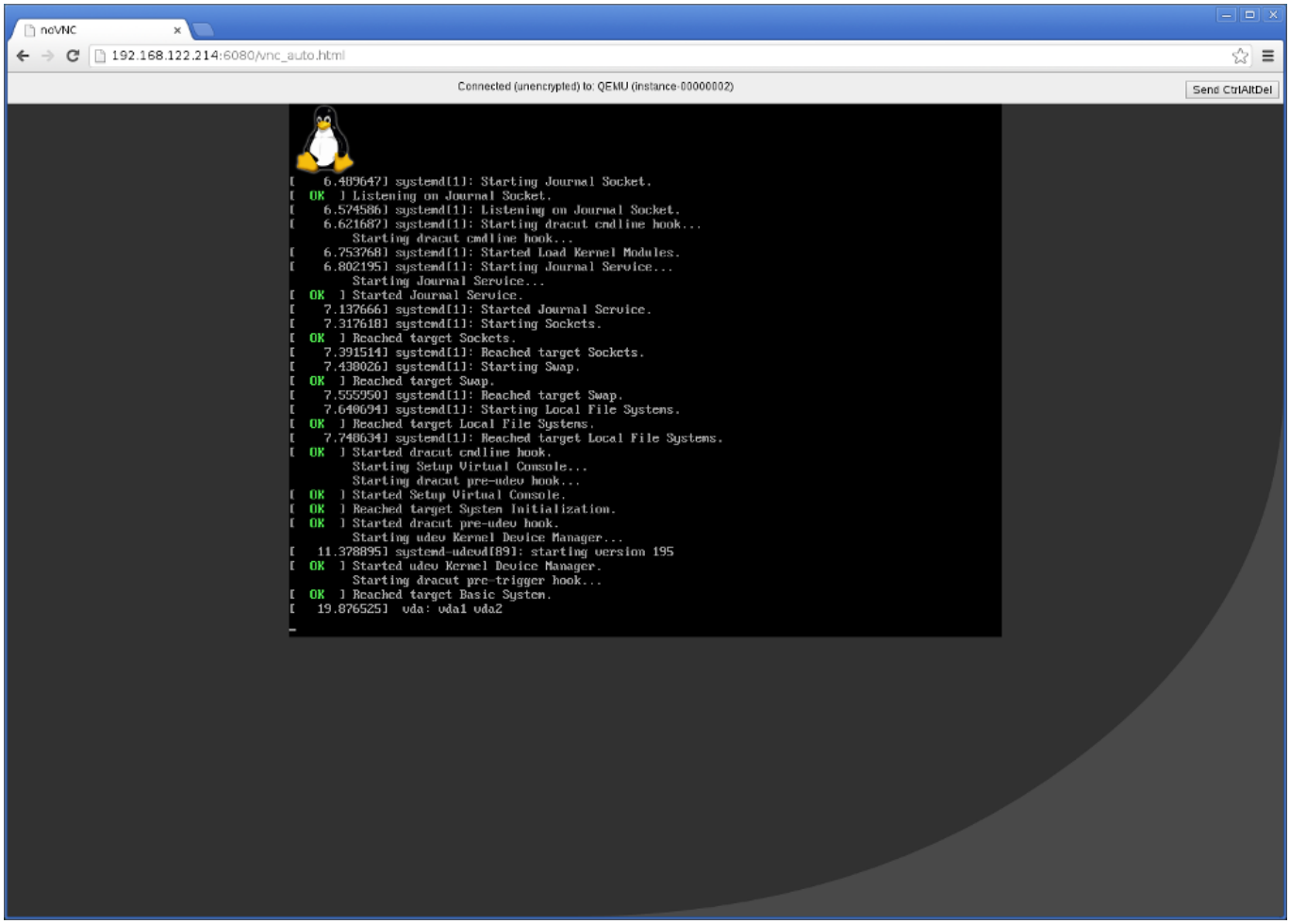


Figure 8.1. VNC instance access

8.3. Install a Compute Node

8.3.1. Create the Compute Service Database

The following procedure creates the database and database user used by the Compute service. These steps must be performed while logged in to the database server as the **root** user.

Procedure 8.4. Creating the Compute Service database

1. Connect to the database service using the **mysql** command.

```
# mysql -u root -p
```

2. Create the **nova** database.

```
mysql> CREATE DATABASE nova;
```

3. Create a **nova** database user and grant it access to the **nova** database.

```
mysql> GRANT ALL ON nova.* TO 'nova'@'%' IDENTIFIED BY
'PASSWORD';
```

```
mysql> GRANT ALL ON nova.* TO 'nova'@'localhost' IDENTIFIED BY
'PASSWORD';
```

Replace *PASSWORD* with a secure password that will be used to authenticate with the database server as this user.

4. Flush the database privileges to ensure that they take effect immediately.

```
mysql> FLUSH PRIVILEGES;
```

5. Exit the **mysql** client.

```
mysql> quit
```

The Compute database has been created. The database will be populated during service configuration.

8.3.2. Configure Compute Service Authentication

This section outlines the steps for creating and configuring Identity service records required by the Compute service.

1. Create the **compute** user, who has the **admin** role in the **services** tenant.
2. Create the **compute** service entry and assign it an endpoint.

These entries will assist other OpenStack services attempting to locate and access the functionality provided by the Compute service. In order to proceed, you should have already performed the following (through the Identity service):

1. Created an Administrator role named **admin** (refer to Section 3.7, “Create an Administrator Account” for instructions)
2. Created the **services** tenant (refer to Section 3.9, “Create the Services Tenant” for instructions)

Note

The *Deploying OpenStack: Learning Environments (Manual Set Up)* guide uses one tenant for all service users. For more information, refer to Section 3.9, “Create the Services Tenant”.

You can perform the following procedure from your Identity service host or on any machine where you've copied the **keystonerc_admin** file (which contains administrator credentials) and the **keystone** command-line utility is installed.

Procedure 8.5. Configuring the Compute Service to authenticate through the Identity Service

1. Authenticate as the administrator of the Identity service by running the **source** command on the **keystonerc_admin** file containing the required credentials:

```
# source ~/keystonerc_admin
```

2. Create a user named **compute** for the OpenStack Compute service to use:

```
# keystone user-create --name compute --pass PASSWORD
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|  email   |                                  |
| enabled  |               True               |
|    id    | 96cd855e5bfe471ce4066794bbafb615 |
|   name   |              compute             |
| tenantId |                                  |
+----------+----------------------------------+
```

   Replace *PASSWORD* with a secure password that will be used by the Compute service when authenticating against the Identity service.

3. Use the **keystone user-role-add** command to link the **compute** user, **admin** role, and **services** tenant together:

```
# keystone user-role-add --user compute --role admin --tenant
services
```

4. Create the **compute** service entry:

```
# keystone service-create --name compute \
       --type compute \
       --description "OpenStack Compute Service"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |     OpenStack Compute Service    |
|      id     | 8dea97f5ee254b309c1792d2bd821e59 |
|     name    |              compute             |
|     type    |              compute             |
+-------------+----------------------------------+
```

5. Create the **compute** endpoint entry:

```
# keystone endpoint-create \
         --service compute
         --publicurl "http://IP:8774/v2/\$(tenant_id)s" \
         --adminurl "http://IP:8774/v2/\$(tenant_id)s" \
         --internalurl "http://IP:8774/v2/\$(tenant_id)s"
```

   Replace *IP* with the IP address or host name of the system that will be acting as the compute node.

All supporting Identity service entries required by the OpenStack Compute service have been created.

8.3.3. Install the Compute Service Packages

The OpenStack Compute services are provided the following packages:

openstack-nova-api

Provides the OpenStack Compute API service. At least one node in the environment must host an instance of the API service. This must be the node pointed to by the Identity service endpoint definition for the Compute service.

openstack-nova-compute

Provides the OpenStack Compute service.

openstack-nova-conductor

Provides the Compute conductor service. The conductor handles database requests made by Compute nodes, ensuring that individual Compute nodes do not require direct database access. At least one node in each environment must act as a Compute conductor.

openstack-nova-scheduler

Provides the Compute scheduler service. The scheduler handles scheduling of requests made to the API across the available Compute resources. At least one node in each environment must act as a Compute scheduler.

python-cinderclient

Provides client utilities for accessing storage managed by the OpenStack Block Storage service. This package is not required if you do not intend to attach block storage volumes to your instances or you intend to manage such volumes using a service other than the OpenStack Block Storage service.

To install the above packages, execute the following command while logged in as the **root** user:

```
# yum install -y openstack-nova-api openstack-nova-compute \
  openstack-nova-conductor openstack-nova-scheduler \
  python-cinderclient
```

Note

In the command presented here, all Compute service packages are installed on a single node. In a production deployment, Red Hat recommends that the API, conductor, and scheduler services be installed on a separate controller node or on separate nodes entirely. The Compute service itself must be installed on each node that is expected to host virtual machine instances.

The Compute service package is now installed.

8.3.4. Configure the Compute Service to Use SSL

Use the following options in the **nova.conf** file to configure SSL.

Table 8.1. SSL options for Compute

Configuration Option	Description
enabled_ssl_apis	A list of APIs with enabled SSL.
ssl_ca_file	CA certificate file to use to verify connecting clients.

Configuration Option	Description
ssl_cert_file	SSL certificate of API server.
ssl_key_file	SSL private key of API server.
tcp_keepidle	Sets the value of TCP_KEEPIDLE in seconds for each server socket. Defaults to 600.

8.3.5. Configure the Compute Service

8.3.5.1. Configure Compute Service Authentication

The Compute service must be explicitly configured to use the Identity service for authentication. Follow the steps listed in this procedure to configure this.

All steps listed in this procedure must be performed on each system hosting Compute services while logged in as the **root** user.

Procedure 8.6. Configuring the Compute Service to authenticate through the Identity Service

1. Set the authentication strategy (**auth_strategy**) configuration key to **keystone** using the **openstack-config** command.

```
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT auth_strategy keystone
```

2. Set the authentication host (**auth_host**) configuration key to the IP address or host name of the Identity server.

```
# openstack-config --set /etc/nova/api-paste.ini \
  filter:authtoken auth_host IP
```

   Replace *IP* with the IP address or host name of the Identity server.

3. Set the administration tenant name (**admin_tenant_name**) configuration key to the name of the tenant that was created for the use of the Compute service. In this guide, examples use *services*.

```
# openstack-config --set /etc/nova/api-paste.ini \
  filter:authtoken admin_tenant_name services
```

4. Set the administration user name (**admin_user**) configuration key to the name of the user that was created for the use of the Compute service. In this guide, examples use *compute*.

```
# openstack-config --set /etc/nova/api-paste.ini \
  filter:authtoken admin_user compute
```

5. Set the administration password (**admin_password**) configuration key to the password that is associated with the user specified in the previous step.

```
# openstack-config --set /etc/nova/api-paste.ini \
  filter:authtoken admin_password PASSWORD
```

The authentication keys used by the Compute services have been set and will be used when the services are started.

8.3.5.2. Configure the Compute Service Database Connection

The database connection string used by the Compute service is defined in the **/etc/nova/nova.conf** file. It must be updated to point to a valid database server before starting the service.

The database connection string only needs to be set on nodes that will be hosting the conductor service (**openstack-nova-conductor**). Compute nodes communicate with the conductor using the messaging infrastructure, the conductor in turn orchestrates communication with the database. As a result individual compute nodes do not require direct access to the database. This procedure only needs to be followed on nodes that will host the conductor service. There must be at least one instance of the conductor service in any compute environment.

All commands in this procedure must be run while logged in as the **root** user on the server hosting the Compute service.

Procedure 8.7. Configuring the Compute Service SQL database connection

- Use the **openstack-config** command to set the value of the **sql_connection** configuration key.

  ```
  # openstack-config --set /etc/nova/nova.conf \
    DEFAULT sql_connection mysql://USER:PASS@IP/DB
  ```

  Replace:

  - *USER* with the database user name the Compute service is to use, usually **nova**.
  - *PASS* with the password of the chosen database user.
  - *IP* with the IP address or host name of the database server.
  - *DB* with the name of the database that has been created for use by the compute, usually **nova**.

Important

The IP address or host name specified in the connection configuration key must match the IP address or host name to which the nova database user was granted access when creating the nova database. Moreover, if the database is hosted locally and you granted permissions to 'localhost' when creating the nova database, you must enter 'localhost'.

The database connection string has been set and will be used by the Compute service.

8.3.5.3. Configure RabbitMQ Message Broker Settings for the Compute Service

As of Red Hat Enterprise Linux OpenStack Platform 5, RabbitMQ replaces QPid as the default (and recommended) message broker. The RabbitMQ messaging service is provided by the *rabbitmq-server* package.

This section assumes that you have already configured a RabbitMQ message broker. For more information, refer to:

- Section 2.4, “Prerequisite Message Broker”
- Section 2.4.2, “Install and Configure the RabbitMQ Message Broker”
- Section 2.4.1, “Configure the Firewall for Message Broker Traffic”
- Section 2.4.2.1, “Launch the RabbitMQ Message Broker”

Procedure 8.8. Configuring the Compute service to use the RabbitMQ message broker

Perform the following steps as **root** on the Compute controller, and all Compute nodes.

1. In **/etc/nova/nova.conf** of that system, set RabbitMQ as the RPC back end.

```
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT rpc_backend rabbit
```

2. Set the Compute service to connect to the RabbitMQ host:

```
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT rabbit_host RABBITMQ_HOST
```

   Replace *RABBITMQ_HOST* with the IP address or host name of the message broker.

3. Set the message broker port to **5672**:

```
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT rabbit_port 5672
```

4. Set the RabbitMQ username and password created for the Compute service:

```
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT rabbit_userid nova
```

```
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT rabbit_password NOVA_PASS
```

   Where **nova** and *NOVA_PASS* are the RabbitMQ username and password created for Compute (in Section 2.4.2, “Install and Configure the RabbitMQ Message Broker”).

5. In Section 2.4.2.1, “Launch the RabbitMQ Message Broker”, we gave the **nova** user read/write permissions to all resources -- specifically, through the virtual host **/**. Configure the Compute service to connect to this virtual host:

```
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT rabbit_virtual_host /
```

8.3.5.3.1. Enable SSL Communication Between Compute Service and Message Broker

If you enabled SSL on the message broker, you will also have to configure the Compute service accordingly. For this, you will need the exported client certificates and key file. See Section 2.4.2.3, “Export an SSL Certificate for Clients” for instructions on how to export these files.

Once you have the necessary certificates and key, run the following commands to enable SSL communication with the message broker:

```
# openstack-config --set /etc/nova/nova.conf \
 DEFAULT rabbit_use_ssl True
# openstack-config --set /etc/nova/nova.conf \
 DEFAULT kombu_ssl_certfile /path/to/client.crt
# openstack-config --set /etc/nova/nova.conf \
 DEFAULT kombu_ssl_keyfile /path/to/clientkeyfile.key
```

Where:

- */path/to/client.crt* is the absolute path to the exported client certificate.
- */path/to/clientkeyfile.key* is the absolute path to the exported client key file.

If your certificates were signed by a third-party Certificate Authority (CA), then you will also need to run the following command:

```
# openstack-config --set /etc/nova/nova.conf \
 DEFAULT kombu_ssl_ca_certs /path/to/ca.crt
```

Replace */path/to/ca.crt* with the absolute path to the CA file provided by the third-party CA (see Section 2.4.2.2, “Enable SSL on the RabbitMQ Message Broker” for more information).

8.3.5.4. Configure Resource Overcommitment

OpenStack supports overcommitting of CPU and memory resources on compute nodes. Overcommitting is a technique of allocating more virtualized CPUs and/or memory than there are physical resources.

Important

Overcommitting increases the amount of instances you are able to run, but reduces instance performance.

CPU and memory overcommit settings are represented as a ratio. OpenStack uses the following ratios by default:

- Default CPU overcommit ratio - 16
- Default memory overcommit ratio - 1.5

These default settings have the following implications:

- The default CPU overcommit ratio of 16 means that up to 16 virtual cores can be assigned to a node for each physical core.
- The default memory overcommit ratio of 1.5 means that instances can be assigned to a physical node if the total instance memory usage is less than 1.5 times the amount of physical memory available.

Use the **cpu_allocation_ratio** and **ram_allocation_ratio** directives in **/etc/nova/nova.conf** to change these default settings.

8.3.5.5. Reserve Host Resources

You can reserve host memory and disk resources as always available to OpenStack. To prevent a

given amount of memory and disk resources from being considered as available to be allocated for usage by virtual machines, edit the following directives in **/etc/nova/nova.conf**:

- **reserved_host_memory_mb** - Defaults to 512MB.
- **reserved_host_disk_mb** - Defaults to 0MB.

8.3.5.6. Configure Compute Networking

8.3.5.6.1. Compute Networking Overview

Unlike Nova-only deployments, when OpenStack Networking is in use, the **nova-network** service **must** not run. Instead all network related decisions are delegated to the OpenStack Networking Service.

Therefore, it is very important that you refer to this guide when configuring networking, rather than relying on Nova networking documentation or past experience with Nova networking. In particular, using CLI tools like **nova-manage** and **nova** to manage networks or IP addressing, including both fixed and floating IPs, is not supported with OpenStack Networking.

Important

It is strongly recommended that you uninstall **nova-network** and reboot any physical nodes that were running **nova-network** before using these nodes to run OpenStack Network. Problems can arise from inadvertently running the **nova-network** process while using OpenStack Networking service; for example, a previously running **nova-network** could push down stale firewall rules.

8.3.5.6.2. Update the Compute Configuration

Each time a Compute instance is provisioned or deprovisioned, the service communicates with OpenStack Networking through its API. To facilitate this connection, it is necessary to configure each Compute node with the connection and authentication details outlined in this procedure.

These steps must be performed on each Compute node while logged in as the **root** user.

Procedure 8.9. Updating the connection and authentication settings of Compute nodes

1. Modify the **network_api_class** configuration key to indicate that the OpenStack Networking service is in use.

```
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT network_api_class nova.network.neutronv2.api.API
```

2. Set the value of the **neutron_url** configuration key to point to the endpoint of the OpenStack Networking API.

```
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT neutron_url http://IP:9696/
```

   Replace *IP* with the IP address or host name of the server hosting the API of the OpenStack Networking service.

3. Set the value of the **neutron_admin_tenant_name** configuration key to the name of the tenant used by the OpenStack Networking service. Examples in this guide use *services*.

```
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT neutron_admin_tenant_name services
```

4. Set the value of the **neutron_admin_username** configuration key to the name of the administrative user for the OpenStack Networking service. Examples in this guide use *neutron*.

```
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT neutron_admin_username neutron
```

5. Set the value of the **neutron_admin_password** configuration key to the password associated with the administrative user for the OpenStack Networking service.

```
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT neutron_admin_password PASSWORD
```

6. Set the value of the **neutron_admin_auth_url** configuration key to the URL associated with the Identity service endpoint.

```
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT neutron_admin_auth_url http://IP:35357/v2.0
```

   Replace *IP* with the IP address or host name of the Identity service endpoint.

7. Set the value of the **security_group_api** configuration key to **neutron**.

```
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT security_group_api neutron
```

   This enables the use of OpenStack Networking security groups.

8. Set the value of the **firewall_driver** configuration key to **nova.virt.firewall.NoopFirewallDriver**.

```
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT firewall_driver nova.virt.firewall.NoopFirewallDriver
```

   This **must** be done when OpenStack Networking security groups are in use.

The configuration has been updated and the Compute service will use OpenStack Networking when it is next started.

8.3.5.6.3. Configure the L2 Agent

Each compute node must run an instance of the Layer 2 (L2) agent appropriate to the networking plug-in that is in use.

- Section 7.7.1, “Configure the Open vSwitch Plug-in Agent”
- Section 7.7.2, “Configure the Linux Bridge Plug-in Agent”

8.3.5.6.4. Configure Virtual Interface Plugging

When **nova-compute** creates an instance, it must 'plug' each of the vNIC associated with the instance into a OpenStack Networking controlled virtual switch. Compute must also inform the virtual switch of the OpenStack Networking port identifier associated with each vNIC.

A generic virtual interface driver, **nova.virt.libvirt.vif.LibvirtGenericVIFDriver**, is provided in Red Hat Enterprise Linux OpenStack Platform. This driver relies on OpenStack Networking being able to return the type of virtual interface binding required. The following plug-ins support this operation:

- Linux Bridge
- Open vSwitch
- NEC
- BigSwitch
- CloudBase Hyper-V
- Brocade

To use the generic driver, execute the **openstack-config** command to set the value of the **vif_driver** configuration key appropriately:

```
# openstack-config --set /etc/nova/nova.conf \
  libvirt vif_driver \
  nova.virt.libvirt.vif.LibvirtGenericVIFDriver
```

Important

Considerations for Open vSwitch and Linux Bridge deployments:

- If running Open vSwitch with security groups enabled, use the Open vSwitch specific driver, **nova.virt.libvirt.vif.LibvirtHybridOVSBridgeDriver**, instead of the generic driver.
- For Linux Bridge environments, you must add the following to the **/etc/libvirt/qemu.conf** file to ensure that the virtual machine launches properly:

```
user = "root"
group = "root"
cgroup_device_acl = [
   "/dev/null", "/dev/full", "/dev/zero",
   "/dev/random", "/dev/urandom",
   "/dev/ptmx", "/dev/kvm", "/dev/kqemu",
   "/dev/rtc", "/dev/hpet", "/dev/net/tun",
]
```

8.3.5.7. Configure the Firewall to Allow Compute Service Traffic

Connections to virtual machine consoles, whether direct or through the proxy, are received on ports **5900** to **5999**.

To allow this, the firewall on the service node must be configured to allow network traffic on these ports. Log in as the **root** user to the server hosting the Compute service and perform the following procedure:

Procedure 8.10. Configuring the firewall to allow Compute Service traffic

1. Open the **/etc/sysconfig/iptables** file in a text editor.
2. Add an INPUT rule allowing TCP traffic on ports in the ranges **5900** to **5999** by adding the following line to the file.

   ```
   -A INPUT -p tcp -m multiport --dports 5900:5999 -j ACCEPT
   ```

   The new rule must appear before any INPUT rules that REJECT traffic.
3. Save the changes to the **/etc/sysconfig/iptables** file.
4. Restart the **iptables** service to ensure that the change takes effect.

   ```
   # service iptables restart
   ```

The firewall is now configured to allow incoming connections to the Compute services. Repeat this process for each compute node.

8.3.6. Populate the Compute Service Database

You can populate the Compute Service database after you have successfully configured the Compute Service database connection string (refer to Section 8.3.5.2, “Configure the Compute Service Database Connection”).

Important

This procedure only needs to be followed once to initialize and populate the database. You do not need to perform these steps again when adding additional systems hosting Compute services.

Procedure 8.11. Populating the Compute Service database

1. Log in to a system hosting an instance of the **openstack-nova-conductor** service.
2. Use the **su** command to switch to the **nova** user.

   ```
   # su nova -s /bin/sh
   ```
3. Run the **nova-manage db sync** command to initialize and populate the database identified in **/etc/nova/nova.conf**.

   ```
   $ nova-manage db sync
   ```

The Compute service database has been initialized and populated.

8.3.7. Launch the Compute Services

Procedure 8.12. Launching Compute services

1. **Starting the Message Bus Service**

   Libvirt requires that the **messagebus** service be enabled and running.

   a. Use the **service** command to start the **messagebus** service.

   ```
   # service messagebus start
   ```

   b. Use the **chkconfig** command to enable the **messagebus** service permanently.

   ```
   # chkconfig messagebus on
   ```

2. **Starting the Libvirtd Service**

   The Compute service requires that the **libvirtd** service be enabled and running.

   a. Use the **service** command to start the **libvirtd** service.

   ```
   # service libvirtd start
   ```

   b. Use the **chkconfig** command to enable the **libvirtd** service permanently.

   ```
   # chkconfig libvirtd on
   ```

3. **Starting the API Service**

   Start the API service on each system that will be hosting an instance of it. Note that each API instance should either have its own endpoint defined in the Identity service database or be pointed to by a load balancer that is acting as the endpoint.

   a. Use the **service** command to start the **openstack-nova-api** service.

   ```
   # service openstack-nova-api start
   ```

   b. Use the **chkconfig** command to enable the **openstack-nova-api** service permanently.

   ```
   # chkconfig openstack-nova-api on
   ```

4. **Starting the Scheduler**

   Start the scheduler on each system that will be hosting an instance of it.

   a. Use the **service** command to start the **openstack-nova-scheduler** service.

   ```
   # service openstack-nova-scheduler start
   ```

   b. Use the **chkconfig** command to enable the **openstack-nova-scheduler** service permanently.

   ```
   # chkconfig openstack-nova-scheduler on
   ```

5. **Starting the Conductor**

   The conductor is intended to minimize or eliminate the need for Compute nodes to access the database directly. Compute nodes instead communicate with the conductor through a message broker and the conductor handles database access.

   Start the conductor on each system that is intended to host an instance of it. Note that it is recommended that this service is not run on each and every Compute node as this eliminates the security benefits of restricting direct database access from the Compute nodes.

   a. Use the **service** command to start the **openstack-nova-conductor** service.

   ```
   # service openstack-nova-conductor start
   ```

   b. Use the **chkconfig** command to enable the **openstack-nova-conductor** service permanently.

   ```
   # chkconfig openstack-nova-conductor on
   ```

6. **Starting the Compute Service**

   Start the Compute service on every system that is intended to host virtual machine instances.

   a. Use the **service** command to start the **openstack-nova-compute** service.

   ```
   # service openstack-nova-compute start
   ```

   b. Use the **chkconfig** command to enable the **openstack-nova-compute** service permanently.

   ```
   # chkconfig openstack-nova-compute on
   ```

7. **Starting Optional Services**

   Depending on environment configuration you may also need to start these services:

   openstack-nova-cert

   The X509 certificate service, required if you intend to use the EC2 API to the Compute service.

   Note

   If you intend to use the EC2 API to the Compute service, you need to set the options in the **nova.conf** configuration file. For more information, see *Configuring the EC2 API* section in the *Red Hat Enterprise Linux OpenStack Platform Configuration Reference Guide*. This document is available from the following link:

   https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux_OpenStack_Platform

   openstack-nova-network

The Nova networking service. Note that you **must** not start this service if you have installed and configured, or intend to install and configure, OpenStack Networking.

`openstack-nova-objectstore`

The Nova object storage service. It is recommended that the OpenStack Object Storage service (Swift) is used for new deployments.

The Compute services have been started and are ready to accept virtual machine instance requests.

Chapter 9. OpenStack Orchestration Installation

9.1. Install the Orchestration Service Packages

The OpenStack Orchestration service requires the following packages:

openstack-heat-api

Provides the OpenStack-native REST API to the **heat** Engine.

openstack-heat-api-cfn

Provides the AWS CloudFormation-compatible API to the **heat** Engine.

openstack-heat-common

Provides components common to all **heat** services.

openstack-heat-engine

Provides the OpenStack API for launching templates and submitting events back to the API.

openstack-heat-api-cloudwatch

Provides the AWS CloudWatch-compatible API to the **heat** Engine.

heat-cfntools

Provides the tools required on **heat**-provisioned cloud instances.

python-heatclient

Provides a Python API and command-line script, both of which make up a client for the OpenStack **heat** API.

openstack-utils

Provides supporting utilities to assist with a number of tasks including the editing of configuration files.

To install these required packages, run the following command as the **root** user:

```
# yum install -y openstack-heat-* python-heatclient openstack-utils
```

9.2. Configure the Orchestration Service

To configure the Orchestration service, you will need to:

- Configure a database for the Orchestration service.
- Bind each Orchestration API service to a corresponding IP address.
- Create and configure the Orchestration service Identity records.
- Configure how Orchestration services authenticate with the Identity service.

The following sections describe each procedure in detail.

9.2.1. Configure the Orchestration Service Database Connection

The OpenStack Orchestration service requires a database in order to work. The database connection settings used by this service are configured in **/etc/heat/heat.conf**.

After using the MariaDB **root** account to automatically create the required databases, the Orchestration service will use its own MariaDB account (specifically, **heat**) to use those databases.

Note

In order to connect to the MariaDB database, create a database, and configure those database, you will need the password of the MariaDB database **root** account.

The following procedure illustrates this in more detail:

Procedure 9.1. Configuring the Orchestration service SQL database connection

1. Connect to the database service as **root** using the **mysql** utility.

```
# mysql -u root -p
```

2. After logging in, create the **heat** database:

```
mysql> CREATE DATABASE heat;
```

3. Create a database user named **heat** and grant that user access to the newly-created **heat** database:

```
mysql> GRANT ALL ON heat.* TO 'heat'@'%' IDENTIFIED BY
'HEATPW';
mysql> GRANT ALL ON heat.* TO 'heat'@'localhost' IDENTIFIED BY
'HEATPW';
```

   Replace *HEATPW* with a secure password that the **heat** user should use to authenticate to the database server.

4. Flush the database privileges to ensure that they take effect immediately:

```
mysql> FLUSH PRIVILEGES;
```

5. Exit the **mysql** utility:

```
mysql> quit
```

6. Update the database connection string used by **heat** with the *HEATPW* used earlier:

```
# openstack-config --set /etc/heat/heat.conf \
 DEFAULT sql_connection mysql://heat:HEATPW@MARIADBHOST/heat
```

   Where:

   - *HEATPW* is the password set earlier for the **heat** account.

- *MARIADBHOST* is the IP address or hostname of the MariaDB database server (**localhost** if it is on the same host).

7. As the **heat** user, sync the database using the **heat-manage** utility:

```
# runuser -s /bin/sh heat -c "heat-manage db_sync"
```

9.2.2. Restrict the Bind Addresses of each Orchestration API Service

After configuring the database, set the **bind_host** setting of each Orchestration API service. This setting controls which IP address a service should use for incoming connections.

The **/etc/heat/heat.conf** configuration file contains a section for each Orchestration API service. the **bind_host** should be set in each one, as in:

```
# openstack-config --set /etc/heat/heat.conf
heat_api bind_host IP
```

```
# openstack-config --set /etc/heat/heat.conf
heat_api_cfn bind_host IP
```

```
# openstack-config --set /etc/heat/heat.conf
heat_api_cloudwatch bind_host IP
```

Replace each *IP* with the address that the corresponding API should use.

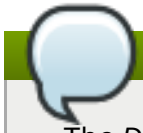
9.2.3. Create the Orchestration Service Identity Records

In this procedure, you will:

1. Create the **heat** user, who has the **admin** role in the **services** tenant.
2. Create the **heat** service entry and assign it an endpoint.

In order to proceed, you should have already performed the following (through the Identity service):

1. Created an Administrator role named **admin** (refer to Section 3.7, “Create an Administrator Account” for instructions)
2. Created the **services** tenant (refer to Section 3.9, “Create the Services Tenant” for instructions)



Note

The *Deploying OpenStack: Learning Environments (Manual Set Up)* guide uses one tenant for all service users. For more information, refer to Section 3.9, “Create the Services Tenant”.

You can perform this procedure from your Identity service server or on any machine where you've copied the **keystonerc_admin** file (which contains administrator credentials) and the **keystone** command-line utility is installed.

Procedure 9.2. Creating Identity records for the Orchestration service

1. Set up the shell to access Keystone as the **admin** user:

```
# source ~/keystonerc_admin
```

2. Create a **heat** user in **keystone**:

```
# keystone user-create \
  --name=heat \
  --pass=SERVICE_PASSWORD \
```

Replace *SERVICE_PASSWORD* with the password the Orchestration service should use when authenticating with the Identity service.

3. Establish the relationship between the Orchestration service, the **services** tenant, and **admin** role:

```
# keystone user-role-add --user heat \
  --role admin \
  --tenant services \
```

4. Create Identity service entries for Orchestration and Cloud Formation:

```
# keystone service-create --name heat \
  --type orchestration
```

```
# keystone service-create --name heat-cfn \
  --type cloudformation
```

Once the **heat** and **heat-cfn** services are created, note their respective IDs. These will be used later.

5. Create service endpoint entries for both **heat** and **heat-cfn** services:

```
# keystone endpoint-create \
  --service heat-cfn \
  --publicurl "HEAT_CFN_IP:8000/v1" \
  --adminurl "HEAT_CFN_IP:8000/v1" \
  --internalurl "HEAT_CFN_IP:8000/v1"
```

```
# keystone endpoint-create \
  --service heat \
  --publicurl "HEAT_IP:8004/v1/%(tenant_id)s" \
  --adminurl "HEAT_IP:8004/v1/%(tenant_id)s" \
  --internalurl "HEAT_IP:8004/v1/%(tenant_id)s"
```

Where:

- *HEAT_CFN_IP* is the IP or host name of the system hosting the **heat-cfn** service.
- *HEAT_IP* is the IP or host name of the system hosting the **heat** service.

Important

Include the **http://** prefix for *HEAT_CFN_IP* and *HEAT_IP* values.

9.2.3.1. Create the Required Identity Domain for the Orchestration Service

As of Red Hat Enterprise Linux OpenStack Platform 5, the Orchestration service now requires its own Identity domain. The use of Identity domains provides a framework through which users can be created, associated with credentials deployed inside instances owned by **heat** stacks.

The use of a separate domain allows for separation between the instances and the user deploying the stack. This, in turn, allows regular users without administrative rights to deploy **heat** stacks that require such credentials.

Note

At present, the creation of Identity service domains is only supported on Red Hat Enterprise Linux 7.1. As such, if you wish to set up the **heat** domain for an OpenStack service installed on Red Hat Enterprise Linux 6.6, you will have to do so from a Red Hat Enterprise Linux 7.1 host. From there, you can specify the Red Hat Enterprise Linux 6.6 host on which to set up the domain.

Procedure 9.3. Creating an Identity service domain for the Orchestration service

1. Obtain the administrative token used by the Identity service. This token is the value of the **admin_token** configuration key in the **/etc/keystone/keystone.conf** file of the Identity server:

   ```
   # cat /etc/keystone/keystone.conf | grep admin_token
   admin_token = 0292d404a88c4f269383ff28a3839ab4
   ```

   The administrative token (in this example, *0292d404a88c4f269383ff28a3839ab4*) will be used later to perform all actions requiring administrative credentials.

2. Install the *python-openstackclient* package on the Red Hat Enterprise Linux 7.1 host you will use to create and configure the domain.

   ```
   # yum install python-openstackclient
   ```

   Run the rest of the steps in this procedure from that Red Hat Enterprise Linux 7.1 host.

3. Create the **heat** domain.

   ```
   # openstack --os-token ADMIN_TOKEN --os-url=IDENTITY_IP:5000/v3 \
     --os-identity-api-version=3 domain create heat \
     --description "Owns users and projects created by heat"
   ```

   Where:

   - *ADMIN_TOKEN* is the administrative token obtained earlier (*0292d404a88c4f269383ff28a3839ab4* from the first step's example).

- *IDENTITY_IP* is the IP or host name of the Identity server. The *IDENTITY_IP* can be a Red Hat Enterprise Linux 6.6 host, even if you are running the command from a Red Hat Enterprise Linux 7.1 host.

This command should return the domain ID of the newly created **heat** domain. This ID (*HEAT_DOMAIN_ID*) will be used in the next step.

4. Create a user named **heat_domain_admin** that can have administrative rights within the **heat** domain:

```
# openstack --os-token ADMIN_TOKEN --os-url=IDENTITY_IP:5000/v3 \
  --os-identity-api-version=3 user create heat_domain_admin \
  --password DOMAIN_PASSWORD \
  --domain HEAT_DOMAIN_ID
  --description "Manages users and projects created by heat"
```

Replace *DOMAIN_PASSWORD* with a secure password for this user. This command should return a user ID (*DOMAIN_ADMIN_ID*), which will be used in the next step.

5. Grant the user created in the previous step administrative rights within the **heat** domain:

```
# openstack --os-token ADMIN_TOKEN --os-url=IDENTITY_IP:5000/v3 \
  --os-identity-api-version=3 role add --user DOMAIN_ADMIN_ID \
  --domain HEAT_DOMAIN_ID admin
```

After setting up the **heat** domain and user, you can now configure the Orchestration service to use them. To do so, run the following commands on the server hosting the **heat** services:

```
# openstack-config --set /etc/heat/heat.conf \
  DEFAULT stack_domain_admin_password DOMAIN_PASSWORD
```

```
# openstack-config --set /etc/heat/heat.conf \
  DEFAULT stack_domain_admin heat_domain_admin
```

```
# openstack-config --set /etc/heat/heat.conf \
  DEFAULT stack_user_domain HEAT_DOMAIN_ID
```

9.2.4. Configure Orchestration Service Authentication

After creating and configuring the required Orchestration service users and roles (namely, Identity records), configure the Orchestration service to authenticate with the Identity service. Doing so involves setting the required Identity tokens for the **heat-api**, **heat-api-cfn**, and **heat-api-cloudwatch** services. These services use the token settings configured in **/etc/heat/heat.conf**.

Procedure 9.4. Configuring the Orchestration service to authenticate through the Identity service

1. Set the Orchestration services to authenticate as the correct tenant:

```
# openstack-config --set /etc/heat/heat.conf \
 keystone_authtoken admin_tenant_name services
```

Where *services* is the name of the tenant created for the use of the Orchestration service. Examples in this guide use *services*.

2. Set the Orchestration services to authenticate using the **heat** administration user account:

```
# openstack-config --set /etc/heat/heat.conf \
keystone_authtoken admin_user heat
```

3. Set the Orchestration services to use the correct **heat** administration user account password:

```
# openstack-config --set /etc/heat/heat.conf \
keystone_authtoken admin_password SERVICE_PASSWORD
```

Where *SERVICE_PASSWORD* is the password set during the creation of the **heat** user.

4. Set the correct Identity service host that the Orchestration services should use:

```
# openstack-config --set /etc/heat/heat.conf \
keystone_authtoken service_host KEYSTONE_HOST
```

```
# openstack-config --set /etc/heat/heat.conf \
keystone_authtoken auth_host KEYSTONE_HOST
```

```
# openstack-config --set /etc/heat/heat.conf \
keystone_authtoken auth_uri http://KEYSTONE_HOST:35357/v2.0 \
```

```
# openstack-config --set /etc/heat/heat.conf \
keystone_authtoken keystone_ec2_uri
http://KEYSTONE_HOST:35357/v2.0
```

Where *KEYSTONE_HOST* is the hostname of the Identity service. If the Identity service is hosted on the same system, use **127.0.0.1**.

5. Configure the **heat-api-cfn** and **heat-api-cloudwatch** service hostnames to which VMs should connect.

```
# openstack-config --set /etc/heat/heat.conf \
DEFAULT heat_metadata_server_url HEAT_CFN_HOST:8000
```

```
# openstack-config --set /etc/heat/heat.conf \
DEFAULT heat_waitcondition_server_url
HEAT_CFN_HOST:8000/v1/waitcondition
```

```
# openstack-config --set /etc/heat/heat.conf \
DEFAULT heat_watch_server_url HEAT_CLOUDWATCH_HOST:8003
```

Where:

- *HEAT_CFN_HOST* is the IP or hostname of the system hosting the **heat-api-cfn** service.
- *HEAT_CLOUDWATCH_HOST* is the IP or hostname of the system hosting the **heat-api-cloudwatch** service.

Important

Even if all services are hosted on the same system, *do not* use **127.0.0.1** for either service hostname. This IP address would refer to the local host of each VM, and would therefore prevent the VM from reaching the actual service.

6. Application templates use wait conditions and signaling for orchestration. You will need to define the Identity role for users that will receive progress data. By default, this role is **heat_stack_user**.

```
# openstack-config --set /etc/heat/heat.conf \
 DEFAULT heat_stack_user_role heat_stack_user
```

9.2.5. Configure RabbitMQ Message Broker Settings for the Orchestration Service

As of Red Hat Enterprise Linux OpenStack Platform 5, RabbitMQ replaces QPid as the default (and recommended) message broker. The RabbitMQ messaging service is provided by the *rabbitmq-server* package.

This section assumes that you have already configured a RabbitMQ message broker. For more information, refer to:

- Section 2.4, “Prerequisite Message Broker”
- Section 2.4.2, “Install and Configure the RabbitMQ Message Broker”
- Section 2.4.1, “Configure the Firewall for Message Broker Traffic”
- Section 2.4.2.1, “Launch the RabbitMQ Message Broker”

Procedure 9.5. Configuring the Orchestration service to use the RabbitMQ message broker

1. Log in as **root** to the Orchestration controller node.
2. In **/etc/heat/heat.conf** of that system, set RabbitMQ as the RPC back end.

```
# openstack-config --set /etc/heat/heat.conf \
   DEFAULT rpc_backend heat.openstack.common.rpc.impl_kombu
```

3. Set the Orchestration service to connect to the RabbitMQ host:

```
# openstack-config --set /etc/heat/heat.conf \
   DEFAULT rabbit_host RABBITMQ_HOST
```

   Replace *RABBITMQ_HOST* with the IP address or host name of the message broker.

4. Set the message broker port to **5672**:

```
# openstack-config --set /etc/heat/heat.conf \
   DEFAULT rabbit_port 5672
```

5. Set the RabbitMQ username and password created for the Orchestration service:

```
# openstack-config --set /etc/heat/heat.conf \
  DEFAULT rabbit_userid heat
```

```
# openstack-config --set /etc/heat/heat.conf \
  DEFAULT rabbit_password HEAT_PASS
```

   Where **heat** and *HEAT_PASS* are the RabbitMQ username and password created for Orchestration (in Section 2.4.2, “Install and Configure the RabbitMQ Message Broker”).

6. In Section 2.4.2.1, “Launch the RabbitMQ Message Broker”, we gave the **heat** user read/write permissions to all resources -- specifically, through the virtual host **/**. Configure the Orchestration service to connect to this virtual host:

```
# openstack-config --set /etc/heat/heat.conf \
  DEFAULT rabbit_virtual_host /
```

9.2.5.1. Enable SSL Communication Between Orchestration Service and Message Broker

If you enabled SSL on the message broker, you will also have to configure the Orchestration service accordingly. For this, you will need the exported client certificates and key file. See Section 2.4.2.3, “Export an SSL Certificate for Clients” for instructions on how to export these files.

Once you have the necessary certificates and key, run the following commands to enable SSL communication with the message broker:

```
# openstack-config --set /etc/heat/heat.conf \
 DEFAULT rabbit_use_ssl True
# openstack-config --set /etc/heat/heat.conf \
 DEFAULT kombu_ssl_certfile /path/to/client.crt
# openstack-config --set /etc/heat/heat.conf \
 DEFAULT kombu_ssl_keyfile /path/to/clientkeyfile.key
```

Where:

- */path/to/client.crt* is the absolute path to the exported client certificate.
- */path/to/clientkeyfile.key* is the absolute path to the exported client key file.

If your certificates were signed by a third-party Certificate Authority (CA), then you will also need to run the following command:

```
# openstack-config --set /etc/heat/heat.conf \
 DEFAULT kombu_ssl_ca_certs /path/to/ca.crt
```

Replace */path/to/ca.crt* with the absolute path to the CA file provided by the third-party CA (see Section 2.4.2.2, “Enable SSL on the RabbitMQ Message Broker” for more information).

9.3. Launch the Orchestration Service

Once the required settings are configured, you can now launch the services that consist of the Orchestration service:

- **openstack-heat-api**
- **openstack-heat-api-cfn**
- **openstack-heat-api-cloudwatch**
- **openstack-heat-engine**

To do so for each service:

```
# service SERVICENAME start
```

Then, configure each service to launch automatically at boot:

```
# chkconfig SERVICENAME on
```

Replace *SERVICENAME* with the corresponding name of each Orchestration service.

9.4. Deploy a Stack Using Orchestration Templates

The Orchestration engine uses templates (defined as **.template** files) to launch instances, IPs, volumes, or other types of stacks. The **heat** utility is a command-line interface that allows you to create, configure, and launch stacks.

Note

The *openstack-heat-templates* package provides sample templates that you can use to test core Orchestration features. It also contains template-related scripts and conversion tools. To install this package, run the following command as **root**:

```
# yum install -y openstack-heat-templates
```

Some Orchestration templates launch instances that require access to **openstack-heat-api-cfn**. Such instances will need to be able to communicate with the **openstack-heat-api-cloudwatch** and **openstack-heat-api-cfn** services. The IPs and ports used by these services are the values set in the **/etc/heat/heat.conf** as **heat_metadata_server_url** and **heat_watch_server_url** (refer to Section 9.2.4, “Configure Orchestration Service Authentication” for details).

To allow access to these services, you may need to configure your firewall accordingly. Specifically, you will need to open the ports used by the **openstack-heat-api-cloudwatch** (8003) and **openstack-heat-api-cfn** (8000). To do so, perform the following tasks as **root**:

Procedure 9.6. Configuring the firewall for Orchestration services traffic

1. Open the **/etc/sysconfig/iptables** file in a text editor.
2. Configure the firewall to allow TCP traffic on ports **8003** and **8000**. To do so, add the following **INPUT** rules to **/etc/sysconfig/iptables**:

```
-A INPUT -i BR -p tcp --dport 8003 -j ACCEPT
```

```
-A INPUT -i BR -p tcp --dport 8000 -j ACCEPT
```

Replace *BR* with the interface of the bridge used by your instances (as in, the instances launched from Orchestration templates).

Note

Do not include the **-i *BR*** parameter in the **INPUT** rules if:

- you are not using **nova-network**, or
- the Orchestration service and **nova-compute** are not hosted on the same server.

3. Save the changes to the **/etc/sysconfig/iptables** file.
4. Restart the **iptables** service for the firewall changes to take effect.

```
# service iptables restart
```

To use **heat** to launch an application:

```
# heat stack-create STACKNAME \
--template-file=PATH_TEMPLATE \
--parameters="PARAMETERS"
```

Where:

- *STACKNAME* is the name you wish to assign to the stack. This name will appear when you run the **heat stack-list** command.
- *PATH_TEMPLATE* is the path to your **.template** file.
- *PARAMETERS* is a semicolon-delimeted list of stack creation parameters you wish to use. Supported parameters are defined in the template file itself.

For example, to launch a stack named **myapplication** using the template file **/home/user/myapplication.template** with specific database login parameters:

```
# heat stack-create myapplication \
--template-file=/home/user/myapplication.template \
--parameters="DBUsername=root;DBPassword=PASSWORD"
```

9.5. Integrate Telemetry and Orchestration Services

The Orchestration service can also use the Telemetry service (and its alarms) in monitoring the resource usage of stacks created using the **heat stack-create** command. To enable this, the Orchestration service must be installed and configured accordingly (see Section 12.2, “Overview of Telemetry Service Deployment” for more information) .

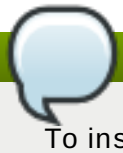
Telemetry service alarms are used by the *autoscaling* feature. This feature allows the Orchestration service to automatically create stacks when the usage of a specific resource reaches a certain level. To allow Orchestration to use Telemetry alarms, the following line must be uncommented in/added to the **resource_registry** section of **/etc/heat/environment.d/default.yaml**:

```
"AWS::CloudWatch::Alarm":
"file:///etc/heat/templates/AWS_CloudWatch_Alarm.yaml"
```

Chapter 10. Dashboard Installation

10.1. Dashboard Service Requirements

- The system hosting the dashboard service must have:
  - The following already installed: **httpd**, **mod_wsgi**, and **mod_ssl** (for security purposes).
  - A connection to the Identity service, as well as to the other OpenStack API services (OpenStack Compute, Block Storage, Object Storage, Image, and Networking services).
- The installer must know the URL of the Identity service endpoint.



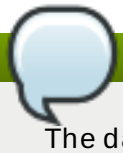
Note

To install **mod_wsgi**, **httpd**, and **mod_ssl**, execute as root:

```
# yum install -y mod_wsgi httpd mod_ssl
```

10.2. Install the Dashboard Packages

The steps in this procedure install the packages required by the OpenStack dashboard service.



Note

The dashboard service uses a configurable backend session store. This installation uses memcached as the session store. However, other options do exist. For more details, refer to *Session Storage Options*.

The only required package is:

openstack-dashboard

Provides the OpenStack dashboard service.

If using memcached, the following must also be installed:

memcached

Memory-object caching system, which speeds up dynamic web applications by alleviating database load.

python-memcached

Python interface to the memcached daemon.

This installation must be performed while logged in as the **root** user.

1. Install the memcached object caching system:

```
# yum install -y memcached python-memcached
```

2. Install the dashboard package:

```
# yum install -y openstack-dashboard
```

The OpenStack dashboard service is installed and ready to be configured.

10.3. Launch the Apache Web Service

Because the dashboard is a Django (Python) web application, it is hosted by the **httpd** service. To start the service, execute the following commands as the **root** user:

1. To start the **service**, execute on the command line:

```
# service httpd start
```

2. To ensure that the httpd service starts automatically in the future, execute:

```
# chkconfig httpd on
```

3. You can confirm that **httpd** is running by executing:

```
# service --status-all | grep httpd
```

10.4. Configure the Dashboard

10.4.1. Configure Connections and Logging

Before users connect to the dashboard for the first time, the following must be configured in the **/etc/openstack-dashboard/local_settings** file (sample files are available in the *Configuration Reference Guide* at https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux_OpenStack_Platform):

Note

Update the following changes as a root user.

1. Allowed hosts - Set the **ALLOWED_HOSTS** parameter with a comma-seperated list of host/domain names that the application can serve. For example:

```
ALLOWED_HOSTS = ['horizon.example.com', 'localhost',
'192.168.20.254', ]
```

2. Cache Backend - As the **root** user, update the **CACHES** settings with the memcached values:

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
CACHES = {
 'default': {
  'BACKEND' :
```

```
'django.core.cache.backends.memcached.MemcachedCache',
  'LOCATION' : 'memcacheURL:port',
 }
}
```

Where:

- **memcacheURL** is the host on which memcache was installed
- **port** is the value from the **PORT** parameter in the **/etc/sysconfig/memcached** file.

3. Dashboard Host - Specify the host URL for your OpenStack Identity service endpoint. For example:

```
OPENSTACK_KEYSTONE_URL="127.0.0.1"
```

4. Time Zone - To change the dashboard's timezone, update the following (the time zone can also be changed using the dashboard GUI):

```
TIME_ZONE="UTC"
```

5. To ensure the configuration changes take effect, restart the Apache web server.

Note

The **HORIZON_CONFIG** dictionary contains all the settings for the Dashboard. Whether or not a service is in the dashboard depends on the Service Catalog configuration in the Identity service. For a full listing, refer to http://docs.openstack.org/developer/horizon/topics/settings.html (*Horizon Settings and Configuration*).

Note

It is recommended that you use **django-secure** module to ensure that most of the best practices and modern browser protection mechanisms are enabled. For more infomation http://django-secure.readthedocs.org/en/latest/ (*django-secure*).

10.4.2. Configure the Dashboard to Use HTTPS

Although the default installation uses a non-encrypted channel (HTTP), it is possible to enable SSL support for the OpenStack dashboard. Use the following steps for enable HTTPS (switch out the example domain 'openstack.example.com' for that of your current setup):

1. Edit the **/etc/openstack-dashboard/local_settings** file, and uncomment the following parameters:

```
SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTOCOL', 'https')
CSRF_COOKIE_SECURE = True
SESSION_COOKIE_SECURE = True
```

The latter two settings instruct the browser to only send dashboard cookies over HTTPS

connections, ensuring that sessions will not work over HTTP.

2. Edit the **/etc/httpd/conf/httpd.conf** file, and add the following line:

```
NameVirtualHost *:443
```

3. Edit the **/etc/httpd/conf.d/openstack-dashboard.conf** file, and replace the 'Before' section with 'After':

   Before:

```
WSGIScriptAlias /dashboard /usr/share/openstack-
dashboard/openstack_dashboard/wsgi/django.wsgi
Alias /static /usr/share/openstack-dashboard/static/

<Directory /usr/share/openstack-dashboard/openstack_dashboard/wsgi>
    <IfModule mod_deflate.c>
      SetOutputFilter DEFLATE
      <IfModule mod_headers.c>
        # Make sure proxies donât deliver the wrong content
        Header append Vary User-Agent env=!dont-vary
      </IfModule>
    </IfModule>

  Order allow,deny
  Allow from all
</Directory>
```

   After:

```
<VirtualHost *:80>
  ServerName openstack.example.com
  RedirectPermanent / https://openstack.example.com/
</VirtualHost>

<VirtualHost *:443>
    ServerName openstack.example.com
    SSLEngine On
    SSLCertificateFile /etc/httpd/SSL/openstack.example.com.crt
    SSLCACertificateFile /etc/httpd/SSL/openstack.example.com.crt
    SSLCertificateKeyFile /etc/httpd/SSL/openstack.example.com.key
    SetEnvIf User-Agent ".*MSIE.*" nokeepalive ssl-unclean-shutdown
    WSGIScriptAlias / /usr/share/openstack-
dashboard/openstack_dashboard/wsgi/django.wsgi
    WSGIDaemonProcess horizon user=apache group=apache processes=3
threads=10
    RedirectPermanent /dashboard https://openstack.example.com
    Alias /static /usr/share/openstack-dashboard/static/
    <Directory /usr/share/openstack-
dashboard/openstack_dashboard/wsgi>
      Order allow,deny
      Allow from all
    </Directory>
</VirtualHost>
```

In the 'After' configuration, Apache listens on port 443 and redirects all non-secured requests to the HTTPs protocol. The **<VirtualHost *:443>** section defines the required options for this protocol, including private key, public key, and certificates.

4. As the **root** user, restart Apache and memcached:

```
# service httpd restart
# service memcached restart
```

When using the HTTP version of the dashboard (through the browser), the user should be redirected to the HTTPs version of the page.

10.4.3. Configure the Dashboard to Use Keystone v3 Authentication

In rare cases, when having many or all services enabled all at once, it may be required to include the following configuration settings for logging in to the dashboard, especially when using keystone v3 authentication.

1. In /usr/share/openstack-dashboard/openstack_dashboard/settings.py add the following configuration:

```
DATABASES =
{
  'default':
  {
   'ENGINE': 'django.db.backends.mysql',
    'NAME': 'horizondb',
    'USER': 'User Name',
    'PASSWORD': 'Password',
    'HOST': 'localhost',
   }
}
```

2. In the same file, change SESSION_ENGINE to:

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cached_db'
```

3. Connect to the database service using the mysql command, replacing USER with the user name by which to connect. The *USER* must be a **root** user (or at least as a user with the correct permission: **create db**).

```
# mysql -u USER -p
```

4. Create the Horizon database.

```
mysql > create database horizondb;
```

5. Exit the mysql client.

```
mysql > exit
```

6. Change to the openstack_dashboard directory and sync the database using:

```
# cd /usr/share/openstack-dashboard/openstack_dashboard
$ ./manage.py syncdb
```

You do not need to create a superuser, so answer 'n' to the question.

7. Restart Apache http server. For Red Hat Enterprise Linux:

```
#service httpd restart
```

10.4.4. Change dashboard's default role

By default, the dashboard service uses the Identity role named **_member_** which is created automatically by Identity. This is adequate for regular users.

If you choose to create a different role and wish to set dashboard to use this role, you must create this role in the Identity service prior to using the dashboard, then configure dashboard to use it.

1. Log in to the system on which your **keystonerc_admin** file resides and authenticate as the Identity administrator:

```
# source ~/keystonerc_admin
```

2. Use the **keystone role-create** command to create the new role:

```
# keystone role-create --name NEW_ROLE
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
| id       | 8261ac4eabcc4da4b01610dbad6c038a |
| name     |              NEW_ROLE            |
+----------+----------------------------------+
```

Replace *NEW_ROLE* with the name that you wish to give to the role.

3. To configure the dashboard service to use a role other than the default **_member_** role, change the value of the **OPENSTACK_KEYSTONE_DEFAULT_ROLE** configuration key, which is stored in:

/etc/openstack-dashboard/local_settings

4. Restart the **httpd** service for the change to take effect.

10.4.5. Configure SELinux

SELinux is a security feature of Red Hat Enterprise Linux, which provides access control. Possible status values are Enforcing, Permissive, and Disabled. If SELinux is configured in 'Enforcing' mode, you must modify the SELinux policy to allow connections from the httpd service to the Identity server. This is also recommended if SELinux is configured in 'Permissive' mode.

1. Use the **getenforce** command to check the status of SELinux on the system:

```
# getenforce
```

2. If the resulting value is 'Enforcing' or 'Permissive', use the **setsebool** command as the **root** user to allow **httpd**-Identity service connections:

```
# setsebool -P httpd_can_network_connect on
```

Note

You can also view the status of SELinux using:

```
# sestatus
SELinux status:                 enabled
SELinuxfs mount:                /selinux
Current mode:                   permissive
Mode from config file:          enforcing
Policy version:                 24
Policy from config file:        targeted
```

For more information, refer to the *Red Hat Enterprise Linux SELinux Users and Administrators Guide* at the following link:

https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/

10.4.6. Configure the dashboard firewall

To allow users to connect to the dashboard, you must configure the system firewall to allow connections. The httpd service, and the dashboard, support both HTTP and HTTPS connections.

Note

To protect authentication credentials and other data, it is highly recommended that you only enable HTTPS connections.

Execute the following as the **root** user:

Procedure 10.1. Configuring the firewall to allow dashboard traffic

1. Edit the **/etc/sysconfig/iptables** configuration file:
   - Allow incoming connections using just HTTPS by adding this firewall rule to the file:

     ```
     -A INPUT -p tcp --dport 443 -j ACCEPT
     ```

   - Allow incoming connections using both HTTP and HTTPS by adding this firewall rule to the file:

     ```
     -A INPUT -p tcp -m multiport --dports 80,443 -j ACCEPT
     ```

2. Restart the iptables service for the changes to take effect.

```
# service iptables restart
```

Important

These rules allow communication from all remote hosts to the system running the dashboard service on ports 80 or 443. For information regarding the creation of more restrictive firewall rules, refer to the *Red Hat Enterprise Linux Security Guide* at the following link:

https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/

10.4.7. Session Storage Options

10.4.7.1. Configure Local Memory Cache Session Storage

Local memory storage is the quickest and easiest session backend to set up, because it has no external dependencies. However, it does have two significant drawbacks:

- No shared storage across processes or workers.
- No persistence after a process terminates.

The local memory backend is enabled as the default for the dashboard service solely because it has no dependencies. However, it is not recommended for production use, or even for serious development work.

To use local memory for storage, include the following in the **`/etc/openstack-dashboard/local_settings`** file:

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
CACHES = {
     'default': {
          'BACKEND': 'django.core.cache.backends.locmem.LocMemCache'
     }
}
```

10.4.7.2. Configure Memcached Session Storage

External caching using an application such as memcached offers persistence and shared storage, and can be very useful for small-scale deployment and/or development. The Dashboard installation process in this guide recommends the use of memcached for external caching (for configuration details, refer to *Configuring Connections and Logging*).

However, for distributed and high-availability scenarios, memcached has inherent problems which are beyond the scope of this documentation. Memcached is an extremely fast and efficient cache backend for cases where it fits the deployment need, but it's not appropriate for all scenarios.

10.4.7.3. Configure Database Session Storage

Database-backed sessions are scalable (using an appropriate database strategy), persistent, and can be made high-concurrency and highly-available. The downside to this approach is that database-backed sessions are one of the slower session storages, and incur a high overhead under heavy usage. Proper configuration of your database deployment can also be a substantial

undertaking and is far beyond the scope of this documentation.

To enable database session storage, follow the below steps as the **root** user to initialize the database and configure it for use:

1. Start the MariaDB command-line client, by executing:

```
# mysql -u root -p
```

2. Specify the MariaDB root user's password when prompted.

3. Create the **dash** database:

```
mysql> CREATE DATABASE dash;
```

4. Create a MariaDB user for the newly-created dash database who has full control of the database.

```
mysql> GRANT ALL ON dash.* TO 'dash'@'%' IDENTIFIED BY
'PASSWORD';
```

```
mysql> GRANT ALL ON dash.* TO 'dash'@'localhost' IDENTIFIED BY
'PASSWORD';
```

   Replace *PASSWORD* with a secure password for the new database user to authenticate with.

5. Enter quit at the **mysql>** prompt to exit the MariaDB client.

6. In the **/etc/openstack-dashboard/local_settings** file, change the following options to refer to the new MariaDB database:

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cached_db'
DATABASES = {
 'default': {
  # Database configuration here
  'ENGINE': 'django.db.backends.mysql',
  'NAME': 'dash',
  'USER': 'dash',
  'PASSWORD': 'PASSWORD',
  'HOST': 'HOST',
  'default-character-set': 'utf8'
 }
}
```

   Replace *PASSWORD* with the password of the **dash** database user and replace *HOST* with the IP address or fully qualified domain name of the database server.

7. Populate the new database by executing:

```
# cd /usr/share/openstack-dashboard
# python manage.py syncdb
```

   Note: You will be asked to create an admin account; this is not required.

   As a result, the following should be displayed:

```
Installing custom SQL ...
Installing indexes ...
DEBUG:django.db.backends:(0.008) CREATE INDEX
`django_session_c25c2c28` ON
`django_session` (`expire_date`);; args=()
No fixtures found.
```

8. Restart Apache to pick up the default site and symbolic link settings:

```
# service httpd restart
```

9. Restart the **openstack-nova-api** service to ensure the API server can connect to the dashboard and to avoid an error displayed in the dashboard.

```
# service openstack-nova-api restart
```

10.4.7.4. Configure Cached Database Session Storage

To mitigate the performance issues of database queries, Django's **cached_db session** backend can be used, which utilizes both the database and caching infrastructure to perform write-through caching and efficient retrieval.

Enable this hybrid setting by configuring both your database and cache as discussed above and then using:

```
SESSION_ENGINE = "django.contrib.sessions.backends.cached_db"
```

10.4.7.5. Configure Cookies Session Storage

The cookies-session backend avoids server load and scaling problems because it stores session data in a cookie, which is stored by the user's browser. The backend uses a cryptographic signing technique, together with the SECRET_KEY, to ensure session data is not tampered with during transport (this is not the same as encryption, session data is still readable by an attacker).

- Advantages:
  - Does not require additional dependencies or infrastructure overhead.
  - Scales indefinitely as long as the quantity of session data being stored fits into a normal cookie.
- Disadvantages:
  - Places session data into storage on the user's machine and transports it over the wire.
  - Limits the quantity of session data which can be stored.

Note

For a thorough discussion of the security implications of this session backend, please read the Django documentation on cookie-based sessions:

https://docs.djangoproject.com/en/dev/topics/http/sessions/#s-using-cookie-based-sessions

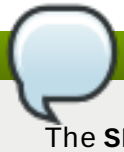
To enable cookie-session storage:

1. In the **/etc/openstack-dashboard/local_settings** file, set:

```
SESSION_ENGINE =
"django.contrib.sessions.backends.signed_cookies"
```

2. Add a randomly-generated **SECRET_KEY** to the project by executing on the command line:

```
$ django-admin.py startproject
```



Note

The **SECRET_KEY** is a text string, which can be specified manually or automatically generated (as in this procedure). You will just need to ensure that the key is unique (that is, does not match any other password on the machine).

10.5. Validate Dashboard Installation

After the dashboard has been successfully installed and configured, you can access the user interface with your web browser. Replace *HOSTNAME* with the host name or IP address of the server on which you installed the Dashboard service:

- HTTPS

```
https://HOSTNAME/dashboard/
```

- HTTP

```
http://HOSTNAME/dashboard/
```

When prompted, log in using the credentials of your OpenStack user. For information on creating an OpenStack user, refer to Section 3.8, “Create a Regular User Account”.

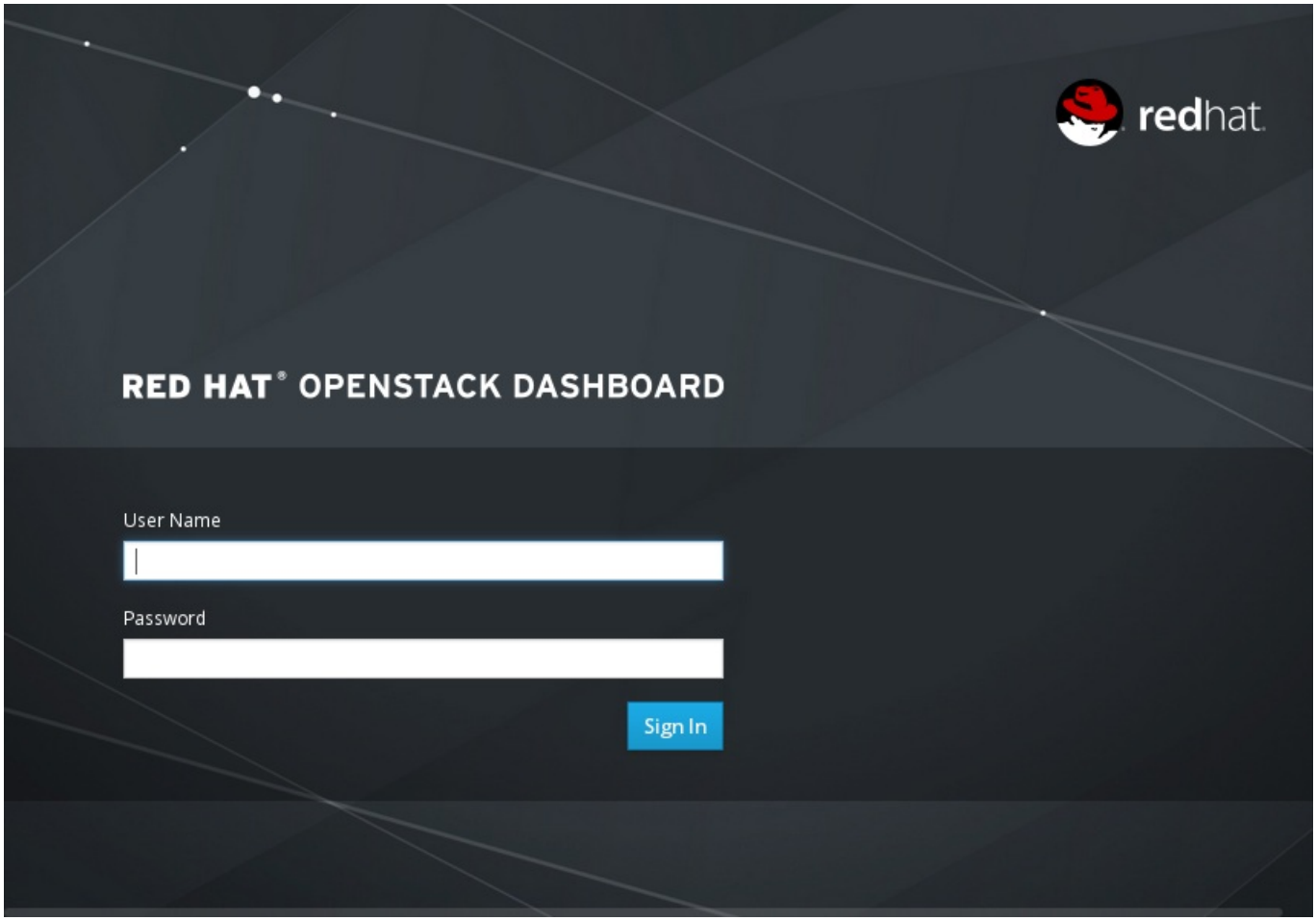


Figure 10.1. Dashboard Login Screen

10.5.1. OpenStack Dashboard - Admin Tab

The **Admin** tab provides an interface where the administrators can view usage and manage instances, volumes, flavors, images, projects, users, services, and quotas.

Note

The **Admin** tab is only displayed in the main window if you have logged in as an administrator.

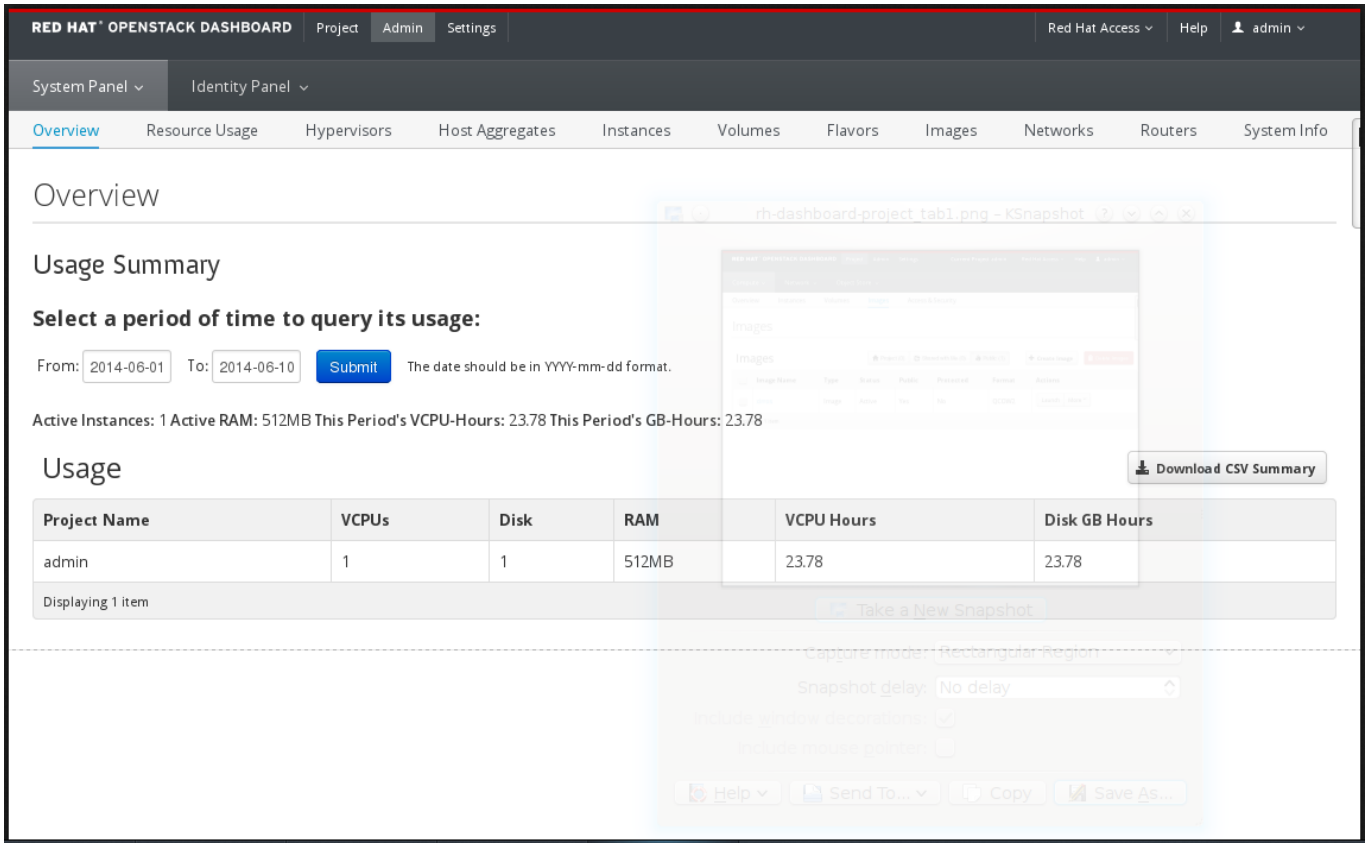


Figure 10.2. Dashboard - Admin Tab

You can access the following options available in the **Admin** tab.

Table 10.1. System Panel

Parameter Name	Description
Overview	View basic reports.
Resource Usage	Use the following tabs to view the following usages: - **Daily Report** - View the daily report. - **Stats** - View the statistics of all resources.
Hypervisors	View the hypervisor summary.
Host Aggregates	View, create, and edit host aggregates. View the list of availability zones.
Instances	View, pause, resume, suspend, migrate, soft or hard reboot, and delete running instances that belong to users of some, but not all, projects. Also, view the log for an instance or access an instance using the console.
Volumes	View, create, edit, and delete volumes, and volume types.
Flavors	View, create, edit, view extra specs for, and delete flavors. Flavors are the virtual hardware templates in OpenStack.
Images	View, create, edit properties for, and delete custom images.
Networks	View, create, edit properties for, and delete networks.
Routers	View, create, edit properties for, and delete routers.

Parameter Name	Description
System Info	Contains the following tabs: - **Services** - View a list of the services. - **Compute Services** - View a list of all Compute services. - **Network Agents** - View the network agents. - **Default Quotas** - View default quota values. Quotas are hard-coded in OpenStack Compute and define the maximum allowable size and number of resources.

Table 10.2. Identity Panel

Parameter Name	Description
Projects	View, create, assign users to, remove users from, and delete projects.
Users	View, create, enable, disable, and delete users.

10.5.2. OpenStack Dashboard - Project Tab

The Project tab provides an interface for viewing and managing the resources of a project. Select a project from the **CURRENT PROJECT** drop-down list on the top right side to view and manage resources in that project.

The **Project** tab displays the details of the selected project.

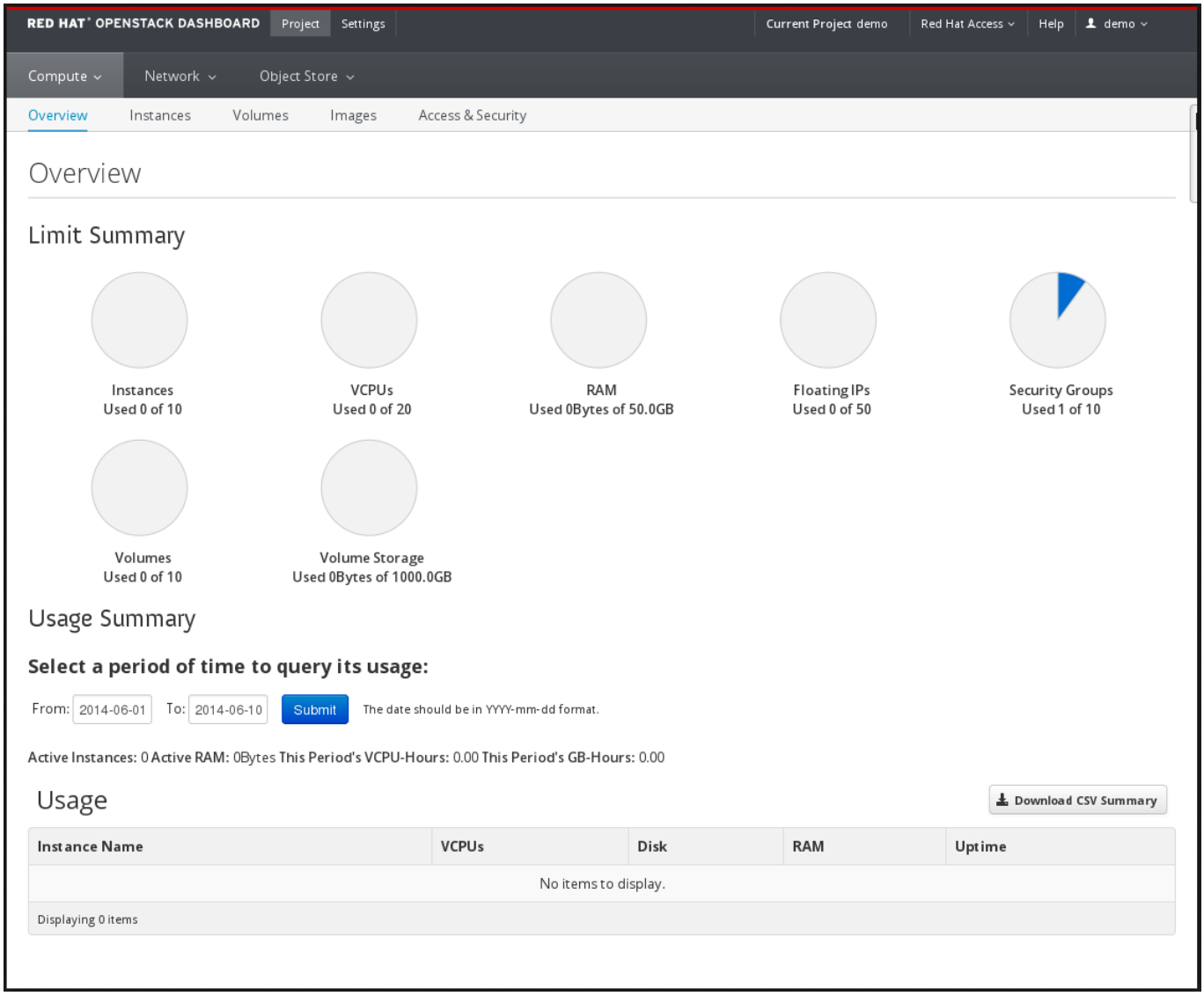


Figure 10.3. Dashboard - Project Tab

Access the following tabs:

Table 10.3. Compute

Parameter Name	Description
Overview	View reports for the project.
Instances	View, launch, create a snapshot from, stop, pause, or reboot instances, or connect to them through the console.
Volumes	Use the following tabs to complete these tasks: • **Volumes** - View, create, edit, and delete volumes. • **Volume Snapshots** - View, create, edit, and delete volume snapshots.
Images	View images, instance snapshots, and volume snapshots created project users, and any images that are publicly available. Create, edit, and delete images, and launch instances from images and snapshots.

Parameter Name	Description
Access & Security	Use these tabs to complete these tasks: - **Security Groups** tab- View, create, edit, and delete security groups, and security group rules. - **Keypairs** tab- View, create, edit, and import keypairs, and delete keypairs. - **Floating IPs** tab- Allocate an IP address to or release it from a project. - **API Access** tab- View API endpoints.

Table 10.4. Network

Parameter Name	Description
Network Topology	View the interactive topology of the network.
Networks	Create and manage public and private networks.
Routers	Create and manage subnets.

Table 10.5. Object Store

Parameter Name	Description
Containers	Create and manage storage containers. A container is a storage compartment for your data and provides a way for you to organize your data. It is similar to the concept as a Linux file directory but cannot be nested.

Table 10.6. Orchestration

Parameter Name	Description
Stacks	Orchestrate multiple composite cloud applications using templates, through both an OpenStack-native REST API and a CloudFormation-compatible Query API.

10.5.3. OpenStack Dashboard - Red Hat Access Tab

Red Hat Access tab gives you the option to sign in to the Red Hat Customer Portal and search for articles and solutions.

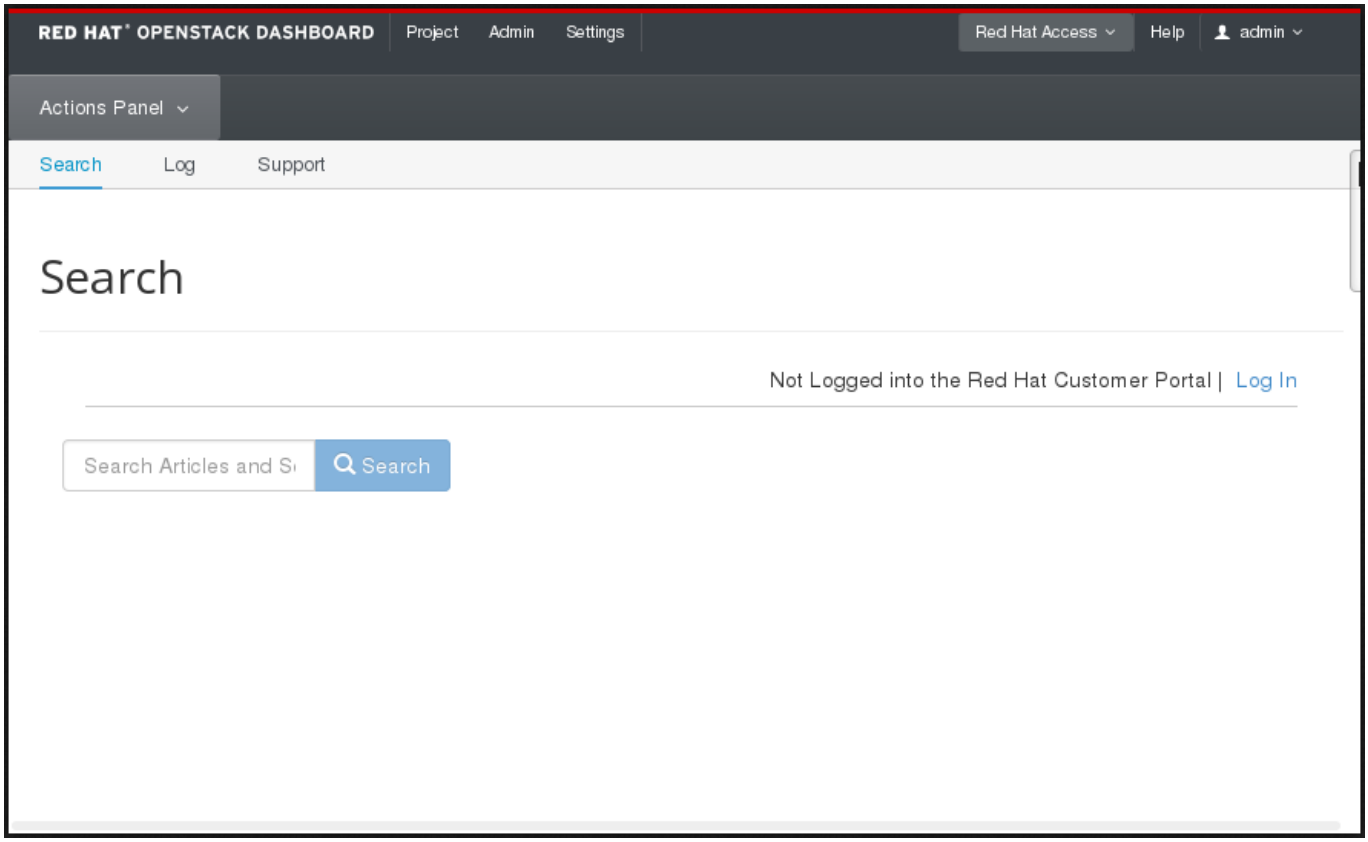


Figure 10.4. Red Hat Access Tab

This tab contains a **Log** option that you can use to check the console log file for your instances and the **Support** option to check for support cases for issues. You can also use this to open new support cases using the **Open a New Support Case** button.

10.6. Troubleshoot Dashboard Installation Issues

10.6.1. No Networks or Routers Tab Appears in the Dashboard

The **Networks** and **Routers** tabs only appear in the dashboard when the environment is configured to use OpenStack Networking. In particular note that by default the PackStack utility currently deploys Nova Networking and as such in environments deployed in this manner the tab will not be visible.

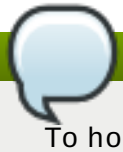
If OpenStack Networking is deployed in the environment but the tabs still do not appear ensure that the service endpoints are defined correctly in the Identity service, that the firewall is allowing access to the endpoints, and that the services are running.

10.6.2. Dashboard Reports ERROR When Launching Instances

When using the dashboard to launch instances if the operation fails, a generic **ERROR** message is displayed. Determining the actual cause of the failure requires the use of the command line tools.

Use the **nova list** to locate the unique identifier of the instance. Then use this identifier as an argument to the **nova show** command. One of the items returned will be the error condition. The most common value is **NoValidHost**.

This error indicates that no valid host was found with enough available resources to host the instance. To work around this issue, consider choosing a smaller instance size or increasing the overcommit allowances for your environment.



Note

To host a given instance, the compute node must have not only available CPU and RAM resources but also enough disk space for the ephemeral storage associated with the instance.

Chapter 11. OpenStack Data Processing Installation

11.1. Install the OpenStack Data Processing Service Packages

On the server hosting the OpenStack Data Processing service, install the *openstack-sahara* package:

```
# yum install openstack-sahara
```

This package provides the OpenStack Data Processing CLI clients (**sahara** and **sahara-db-manage**) and the **openstack-sahara-api** service.

11.2. Configure the Data Processing Service

To configure the Data Processing service (Sahara), you will need to:

- Configure the OpenStack Data Processing database connection.
- Configure the OpenStack Data Processing API server to authenticate with the Identity service.
- Configure the firewall to allow service traffic for OpenStack Data Processing (through port **8386**).

The following sections describe each step in greater detail.

11.2.1. Configure the OpenStack Data Processing Database Connection

The database connection URL used by the OpenStack Data Processing service is defined in the **/etc/sahara/sahara.conf** file. This URL must be set to point to a valid database server before launching the OpenStack Data Processing API server (**openstack-sahara-api**). Typically, if a database already exists for OpenStack Data Processing you would set the database connection URL using the following command:

```
# openstack-config --set /etc/sahara/sahara.conf \
   database connection DB_TYPE://USER:PASS@IP/sahara
```

Where:

- *DB_TYPE* is the type of the database server (for example, **mysql**).
- *USER* and *PASS* are the username and password required by the OpenStack Data Processing service to log on to the database server. Supply these credentials only when required by the database server (for example, when the database server is hosted on another system or node).
- *IP* is the IP address/hostname of the system hosting the database server.

If you have not created a database yet for OpenStack Data Processing, perform the following steps instead:

Procedure 11.1. Creating and configuring a database for OpenStack Data Processing

If you are installing the OpenStack Data Processing service for testing purposes, create the database on the same server hosting the **openstack-sahara-api** service. The following instructions show how to do this on MariaDB, which is the default database used by most OpenStack components.

1. From the command line, log on to the database as an administrator:

```
# mysql -u root -p
```

2. Create the **sahara** database.

```
mysql> CREATE DATABASE sahara;
```

3. Create a **sahara** database user and grant it access to the **sahara** database.

```
MariaDB > GRANT ALL ON sahara.* TO 'sahara'@'%' IDENTIFIED BY
'PASSWORD';
```

```
MariaDB > GRANT ALL ON sahara.* TO 'sahara'@'localhost' IDENTIFIED
BY 'PASSWORD';
```

Replace *PASSWORD* with a secure password that will be used to authenticate with the database server as this user.

4. Exit the **mysql** client.

```
mysql> quit
```

5. On the system hosting the OpenStack Data Processing service, set the database connection URL to the new database:

```
# openstack-config --set /etc/sahara/sahara.conf \
    database connection mysql://sahara:PASSWORD@IP/sahara
```

6. Use the **sahara-db-manage** client to configure the schema of the **sahara** database:

```
# sahara-db-manage --config-file /etc/sahara/sahara.conf upgrade
head
INFO  [alembic.migration] Context impl MySQLImpl.
INFO  [alembic.migration] Will assume non-transactional DDL.
INFO  [alembic.migration] Running upgrade None -> 001, Icehouse
release
INFO  [alembic.migration] Running upgrade 001 -> 002, placeholder
INFO  [alembic.migration] Running upgrade 002 -> 003, placeholder
INFO  [alembic.migration] Running upgrade 003 -> 004, placeholder
INFO  [alembic.migration] Running upgrade 004 -> 005, placeholder
INFO  [alembic.migration] Running upgrade 005 -> 006, placeholder
INFO  [alembic.migration] Running upgrade 006 -> 007, convert
clusters.status_description to LongText
INFO  [alembic.migration] Running upgrade 007 -> 008, add
security_groups field to node groups
INFO  [alembic.migration] Running upgrade 008 -> 009, add rollback
info to cluster
INFO  [alembic.migration] Running upgrade 009 -> 010, add
auto_security_groups flag to node group
INFO  [alembic.migration] Running upgrade 010 -> 011, add Sahara
settings info to cluster
```

11.2.2. Create the OpenStack Data Processing Service Identity Records

This section assumes that you have already created an administrator account and **services** tenant. For more information, refer to:

- Section 3.7, “Create an Administrator Account”
- Section 3.9, “Create the Services Tenant”

In this procedure, you will:

1. Create the **sahara** user, who has the **admin** role in the **services** tenant.
2. Create the **sahara** service entry and assign it an endpoint.

In order to proceed, you should have already performed the following (using the Identity service):

1. Created an Administrator role named **admin** (refer to Section 3.7, “Create an Administrator Account” for instructions)
2. Created the **services** tenant (refer to Section 3.9, “Create the Services Tenant” for instructions)

Note

The *Deploying OpenStack: Learning Environments (Manual Set Up)* guide uses one tenant for all service users. For more information, refer to Section 3.9, “Create the Services Tenant”.

You can perform this procedure from your Identity service server or on any machine where you've copied the **keystonerc_admin** file (which contains administrator credentials) and the **keystone** command-line utility is installed.

Procedure 11.2. Configuring the OpenStack Data Processing service to authenticate through the Identity Service

1. Set up the shell to access Keystone as the admin user:

```
# source ~/keystonerc_admin
```

2. Create the **sahara** user and set its password by replacing *PASSWORD* with your chosen password:

```
# keystone user-create --name sahara --pass PASSWORD
```

3. Add the **sahara** user to the **services** tenant with the **admin** role:

```
# keystone user-role-add --user sahara --role admin --tenant
services
```

4. Create the **sahara** system item:

```
# keystone service-create --name=sahara --type=data_processing --
description="OpenStack Data Processing"
```

5. Create the **sahara** endpoint entry:

```
# keystone endpoint-create \
    --service sahara \
    --publicurl "http://SAHARA_HOST:8386/v1.1/%(tenant_id)s" \
    --adminurl "http://SAHARA_HOST:8386/v1.1/%(tenant_id)s" \
    --internalurl "http://SAHARA_HOST:8386/v1.1/%(tenant_id)s"
```

Replace *SAHARA_HOST* with the IP address or fully qualified domain name of the system hosting the OpenStack Data Processing service.

Note

By default, the endpoint is created in the default region, **regionOne**. This is a case-sensitive value.

If you need to specify a different region when creating an endpoint, use the ***--region*** argument to provide it. In particular, if you are manually installing OpenStack Data Processing in an OpenStack environment deployed through Packstack, specify the region as **RegionOne** (as in, ***--region RegionOne***).

See Section 3.6.1, “Service Regions” for more information.

You have now configured the Identity service to work with the OpenStack Data Processing service.

11.2.3. Configure OpenStack Data Processing Authentication

After creating and configuring the required OpenStack Data Processing service users and roles (namely, Identity records), configure the OpenStack Data Processing API server (**openstack-sahara-api**) to authenticate with the Identity service. Doing so involves setting the required Identity credentials in the **/etc/sahara/sahara.conf** configuration file.

To configure the required Identity credentials for the OpenStack Data Processing API server, perform the following procedure.

Procedure 11.3. Configuring the OpenStack Data Processing API server to authenticate through the Identity service

1. First, set the OpenStack Data Processing API to connect to the Identity API service:

```
# openstack-config --set /etc/sahara/sahara.conf \
   keystone_authtoken auth_uri
http://KEYSTONE_HOST:PUBLICPORT/v2.0/
# openstack-config --set /etc/sahara/sahara.conf \
   keystone_authtoken identity_uri http://KEYSTONE_HOST:ADMINPORT
```

Where:

- *KEYSTONE_HOST* is the IP address of the Identity API host.
- *PUBLICPORT* is the port used for the Identity API service public endpoint.
- *ADMINPORT* is the port used for the Identity API service admin endpoint (typically **35357**).

See Section 3.6, "Create the Identity Service Endpoint" for more details on setting up the Identity API service endpoints.

2. Set the OpenStack Data Processing service to authenticate as the correct tenant:

```
# openstack-config --set /etc/sahara/sahara.conf \
  keystone_authtoken admin_tenant_name services
```

Where *services* is the name of the tenant created for the use of the OpenStack Data Processing service. Examples in this guide use *services*.

3. Set the OpenStack Data Processing API server to authenticate using the **sahara** administration user account:

```
# openstack-config --set /etc/sahara/sahara.conf \
  keystone_authtoken admin_user sahara
```

4. Set the OpenStack Data Processing API server to use the correct **sahara** administration user account password:

```
# openstack-config --set /etc/sahara/sahara.conf \
  keystone_authtoken admin_password SERVICE_PASSWORD
```

Where *SERVICE_PASSWORD* is the password set during the creation of the **sahara** user.

11.2.4. Configure the Firewall to Allow OpenStack Data Processing Service Traffic

OpenStack Data Processing uses port **8386**. As such, Red Hat Enterprise Linux should be configured to allow traffic on this port.

To do so, perform the following steps as the **root** user.

Procedure 11.4. Configuring the firewall to allow OpenStack Data Processing traffic

1. Open the **/etc/sysconfig/iptables** file in a text editor.
2. Add an INPUT rule allowing TCP traffic on port **8386** to the file. The new rule must appear before any INPUT rules that REJECT traffic.

```
-A INPUT -p tcp -m multiport --dports 8386 -j ACCEPT
```

3. Save the changes to the **/etc/sysconfig/iptables** file.
4. Restart the **iptables** service to ensure that the change takes effect.

```
# service iptables restart
```

The firewall is now configured to allow incoming connections to the OpenStack Data Processing service on port **8386**.

11.3. Configure and Launch the OpenStack Data Processing Service

If your OpenStack deployment uses OpenStack Networking (**neutron**), then you must configure the OpenStack Data Processing service accordingly. To do so, run:

```
# openstack-config --set /etc/sahara/sahara.conf \
   DEFAULT use_neutron true
```

To launch OpenStack Data Processing, run the following command on the server hosting it:

```
# systemctl start openstack-sahara-all
```

Then, configure the service to start at boot:

```
# systemctl enable openstack-sahara-all
```

Part III. Set Up the OpenStack Monitoring

Chapter 12. OpenStack Telemetry Installation

12.1. Telemetry Service Overview

The Telemetry service provides user-level usage data for OpenStack-based clouds, which can be used for customer billing, system monitoring, or alerts. Data can be collected by notifications sent by existing OpenStack components (for example, usage events emitted from Compute) or by polling the infrastructure (for example, libvirt).

Telemetry includes a storage daemon that communicates with authenticated agents through a trusted messaging system, to collect and aggregate data. Additionally, the service uses a plug-in system, which makes it easy to add new monitors.

Table 12.1. Telemetry service components

Component	Description
ceilometer-agent-compute	An agent that runs on each Compute node to poll for resource utilization statistics.
ceilometer-agent-central	An agent that runs on a central management server to poll for utilization statistics about resources not tied to instances or Compute nodes.
ceilometer-collector	An agent that runs on one or more central management servers to monitor the message queues. Notification messages are processed and turned into Telemetry messages, and sent back out on to the message bus using the appropriate topic. Telemetry messages are written to the data store without modification.
ceilometer-notification	An agent that pushes metrics to the collector service from various OpenStack services.
MongoDB database	For collected usage data from collector agents. Only the collector agents and the API server have access to the database.
API Server	Runs on one or more central management servers to provide access to data in the database.
RabbitMQ server (`rabbitmq-server`)	Provides the AMQP message queue. This server (also used by Block Storage) handles the OpenStack transaction management, including queuing, distribution, security, management, clustering, and federation. Messaging becomes especially important when an OpenStack deployment is scaled and its services are running on multiple machines.

Telemetry service dependencies

- Each **`nova-compute`** node must have a **`ceilometer-compute`** agent deployed and running.
- All nodes running the **`ceilometer-api`** service must have firewall rules granting appropriate access.
- The **`ceilometer-central-agent`** cannot currently be horizontally scaled, so only a single instance of this service should be running at any given moment.
- You can choose where to locate the additional Telemetry agents, as all intra-agent communication is either based on AMQP or REST calls to the **`ceilometer-api`** service; as is the case for the **`ceilometer-alarm-evaluator`** service.

12.2. Overview of Telemetry Service Deployment

The OpenStack Telemetry service is composed of an API server, three *openstack-ceilometer* agents, and two alarm services. The API Server (provided by the *openstack-ceilometer-api*) runs on one or more central management servers to provide access to the back-end repository.

Note

At present, **`mongod`** is the only back-end repository supported by the Telemetry service.

The three Telemetry agents (and their respective packages) are:

- Central agent (provided by *openstack-ceilometer-central*): runs on a central management server to poll public REST APIs for utilization statistics about resources that are not visible (either through notifications or from the hypervisor layer)
- Collector (provided by *openstack-ceilometer-collector*): runs on one or more central management servers to receive notifications on resource usage. The Collector also parses resource usage statistics and saves them as datapoints in the Telemetry store.
- Compute agent (provided by *openstack-ceilometer-compute*): runs on each Compute service node to poll for instance utilization statistics. You must install and configure the Compute service first before installing the *openstack-ceilometer-compute* on any node.

The two alarm services (and their respective packages) that comprise the rest of the Telemetry service are:

- Evaluator (provided by *ceilometer-alarm-evaluator*): triggers state transitions on alarms.
- Notifier (provided by *ceilometer-alarm-notifier*): executes required actions when alarms are triggered.

You can deploy the API Server, Central agent, data store service, and Collector on different hosts. In addition, each Compute node must also have a Compute agent installed; this agent gathers detailed usage metrics on instances running on the Compute node.

Each of these components must have the following settings configured:

- Authentication, as in Identity service tokens and Telemetry secret
- Database connection URL, for connecting to the Telemetry store

The authentication settings (for example, Identity credentials) and database connection string for these components are all configured in **`/etc/ceilometer/ceilometer.conf`**. As such, components deployed on the same host will share the same settings.

Conversely, if Telemetry components are deployed on multiple hosts then you will have to replicate any authentication changes to these hosts. To do so, you can copy the **`/etc/ceilometer/ceilometer.conf`** file across hosts after applying the new settings.

Once the Telemetry service (as in, all of its components, wherever each is hosted) is deployed and configured, you will need to configure each monitored service (Image, Networking, Object Storage, Block Storage, and each Compute node) to submit data to the Telemetry service. The related settings are configured in each service's configuration file.

12.3. Install the Telemetry Service Packages

The OpenStack Telemetry service requires the following packages:

mongodb-server

Provides the MongoDB database server. The Telemetry service uses **mongodb** as its back-end data repository.

openstack-ceilometer-api

Provides the **ceilometer** API Server.

openstack-ceilometer-central

Provides the Central **ceilometer** agent.

openstack-ceilometer-collector

Provides the **ceilometer** Collector agent.

openstack-ceilometer-common

Provides components common to all **ceilometer** services.

openstack-ceilometer-compute

Provides the **ceilometer** agent that should run on each Compute node.

openstack-ceilometer-alarm

Provides the **ceilometer** alarm notification and evaluation services.

openstack-ceilometer-notification

Provides the Notification agent. This agent provides metrics to the Collector agent from different OpenStack services.

python-ceilometer

Provides the **ceilometer** python library.

python-ceilometerclient

Provides the **ceilometer** command-line tool and a Python API (specifically, the **ceilometerclient** module).

To install these required packages on the same host, run the following command as the **root** user:

```
# yum install -y mongodb-server openstack-ceilometer-* python-
ceilometer python-ceilometerclient
```

You can deploy the API Server, Central agent, data store service, and Collector on different hosts. In this case, log in to a host and install the corresponding package of the component you wish to deploy there:

```
# yum install -y PACKAGE
```

Replace *PACKAGE* with the corresponding package of the component you wish to install on the host. All nodes hosting services that you wish to monitor with the Telemetry services must also have the *python-ceilometer* and *python-ceilometerclient* packages installed (see Section 12.9, “Configure Monitored Services” for more information).

12.4. Create the Telemetry Identity Records

12.4. Create the Telemetry Identity Records

In this section, you will:

1. Create the **ceilometer** user, who has the **ResellerAdmin** role in the **services** tenant.
2. Create the **ceilometer** service entry and assign it an endpoint.

In order to proceed, you need to have already completed the following steps:

1. Created an Administrator role named **admin** (refer to Section 3.7, "Create an Administrator Account" for instructions)
2. Created the **services** tenant (refer to Section 3.9, "Create the Services Tenant" for instructions)

Note

The *Deploying OpenStack: Learning Environments (Manual Set Up)* guide uses one tenant for all service users. For more information, refer to Section 3.9, "Create the Services Tenant".

You can perform this procedure from your Identity service host or on any machine where you've copied the **keystonerc_admin** file (which contains administrator credentials) and the **keystone** command-line utility is installed. For more information about the **keystonerc_admin** file, refer to Section 3.7, "Create an Administrator Account".

Procedure 12.1. Creating Identity records for the Telemetry service

1. Set up the shell to access Keystone as the admin user:

```
# source ~/keystonerc_admin
```

2. Create a **ceilometer** user using the following command:

```
# keystone user-create --name=ceilometer \
   --pass=SERVICE_PASSWORD \
   --email=CEILOMETER_EMAIL
```

   Where:

   - *SERVICE_PASSWORD* is the password the Telemetry service should use when authenticating with the Identity service.
   - *CEILOMETER_EMAIL* is the email address used by the Telemetry service.

3. Create a **ResellerAdmin** role:

```
# keystone role-create --name=ResellerAdmin
```

4. Establish the relationship between the Telemetry service, the **services** tenant, and **ResellerAdmin** role:

```
# keystone user-role-add --user ceilometer \
    --role ResellerAdmin \
    --tenant services
```

5. Establish the relationship between the Telemetry service, the **services** tenant, and **admin** role:

```
# keystone user-role-add --user ceilometer \
   --role admin \
   --tenant services
```

6. Create the **ceilometer** service entry:

```
# keystone service-create --name=ceilometer \
  --type=metering \
  --description="OpenStack Telemetry Service"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |     OpenStack Telemetry Service  |
| id          | a511aea8bc1264641f4dff1db38751br |
| name        |             ceilometer           |
| type        |             metering             |
+-------------+----------------------------------+
```

7. Create the **ceilometer** endpoint entry:

```
# keystone endpoint-create \
 --service ceilometer \
 --publicurl "IP:8777" \
 --adminurl "IP:8777" \
 --internalurl "IP:8777"
```

Replace *IP* with the IP address or host name of the system hosting the Telemetry service.

Note

By default, the endpoint is created in the default region, **regionOne**. This is a case-sensitive value.

If you need to specify a different region when creating an endpoint, use the ***--region*** argument to provide it. In particular, if you are manually installing the Telemetry service in an OpenStack environment deployed through Packstack, specify the region as **RegionOne** (as in, ***--region RegionOne***).

See Section 3.6.1, “Service Regions” for more information.

12.5. Configure Telemetry Service Authentication

After creating and configuring the required Telemetry service users and roles (namely, Identity records), configure the Telemetry API service (**openstack-ceilometer-api**) to authenticate with the Identity service. Doing so involves setting the required Identity credentials in the **/etc/ceilometer/ceilometer.conf** configuration file.

To configure the required Identity credentials for the Telemetry API service, perform the following procedure.

Procedure 12.2. Configuring the Telemetry service to authenticate through the Identity service

1. Set the Telemetry service's authentication host (**auth_host**) configuration key:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  keystone_authtoken auth_host KEYSTONE_HOST
```

   Replace *KEYSTONE_HOST* with the IP address or host name of the Identity server.

2. Set the Telemetry service's authentication port (**auth_port**) configuration key:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  keystone_authtoken auth_port PORT
```

   Replace *PORT* with the authentication port used by the Identity server.

3. Set the Telemetry service to use the **http** protocol for authenticating:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  keystone_authtoken auth_protocol PORT
```

4. Set the Telemetry service to authenticate as the correct tenant:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  keystone_authtoken admin_tenant_name services
```

   Where *services* is the name of the tenant created for the use of the Telemetry service. Examples in this guide use *services*.

5. Set the Telemetry service to authenticate using the **ceilometer** administration user account:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  keystone_authtoken admin_user ceilometer
```

6. Set the Telemetry service to use the correct **ceilometer** administration user account password:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  keystone_authtoken admin_password SERVICE_PASSWORD
```

   Where *SERVICE_PASSWORD* is the password set during the creation of the **ceilometer** user.

7. Set the Telemetry secret. This is a string used to help secure communication between all components of the Telemetry service across multiple hosts (for example, between the Collector agent and a Compute node agent). To set the Telemetry secret:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
 publisher_rpc metering_secret SECRET
```

Replace *SECRET* with the string that all Telemetry service components should use to sign and verify messages that are sent or received over AMQP.

After configuring the credentials of the Telemetry API service, configure the service endpoints to be used by the Central agent, Compute agents, and Alarm Evaluator. To do so, run the following commands on each host where these components are deployed:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  DEFAULT os_auth_url http://IP:35357/v2.0
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  DEFAULT os_username ceilometer
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  DEFAULT os_tenant_name services
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  DEFAULT os_password SERVICE_PASSWORD
```

Where:

- *IP* is the IP address or host name of the Identity server.
- *SERVICE_PASSWORD* is the password set during the creation of the **ceilometer** user.

12.6. Configure the MongoDB Back-End and Create the Telemetry Database

The Telemetry service uses MongoDB as its back-end data repository. As such, you will need to start the **mongod** service.

Before doing so, you may want to configure **mongod** to run with the **--smallfiles** parameter. This parameter configures MongoDB to use a smaller default data file and journal size. With this, MongoDB will limit the size of each data file, creating and writing to a new one when it reaches 512MB.

You can configure **mongod** to run with this parameter using **/etc/sysconfig/mongod**. To do so, specify **--smallfiles** in the **OPTIONS** section of **/etc/sysconfig/mongod**, as in:

```
OPTIONS="--smallfiles /etc/mongodb.conf"
```

MongoDB will use all parameters specified in the **OPTIONS** section when **mongod** launches. To start the MongoDB service:

```
# service mongod start
```

After configuring and launching the MongoDB back-end, create a suitable MongoDB database for the Telemetry service:

```
# mongo --host MONGOHOST --eval'
db = db.getSiblingDB("ceilometer");
db.addUser({user: "ceilometer",
 pwd: "MONGOPASS",
 roles: [ "readWrite", "dbAdmin" ]})'
```

This command will create the MongoDB database for the Telemetry service, along with a database user named **ceilometer** (whose corresponding password is *MONGOPASS*). Replace *MONGOHOST* with the IP address/hostname of the system hosting the MongoDB database.

12.6.1. Configure the Telemetry Database Connection

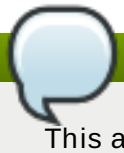
The database connection URL used by the Telemetry service is defined in the **/etc/ceilometer/ceilometer.conf** file. This URL must be set to point to a valid database server before launching the Telemetry API service (**openstack-ceilometer-api**), Notification agent (**openstack-ceilometer-notification**) and Collector agent (**openstack-ceilometer-collector**). The Telemetry service uses this database server as its back-end data repository; by default, the Telemetry service uses MongoDB.

To set the database connection string, run the following command as the **root** user on the system hosting the **openstack-ceilometer-api** and **openstack-ceilometer-collector** services:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  database connection
mongodb://ceilometer:MONGOPASS@MONGOHOST/ceilometer
```

Where:

- *MONGOPASS* is the password of the **ceilometer** user, required by the Telemetry service to log on to the database server. Supply these credentials only when required by the database server (for example, when the database server is hosted on another system or node).
- *MONGOHOST* is the IP address/hostname and port of the system hosting the database server.



Note

This assumes that you created a MongoDB database user named **ceilometer** with the password *MONGOPASS*, as instructed in Section 12.6, “Configure the MongoDB Back-End and Create the Telemetry Database”.

If MongoDB is hosted locally on the same host, the required database connection string would be:

```
mongodb://localhost:27017/ceilometer
```

As in:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  database connection mongodb://localhost:27017/ceilometer
```

12.7. Configure RabbitMQ Message Broker Settings for the Telemetry Service

As of Red Hat Enterprise Linux OpenStack Platform 5, RabbitMQ replaces QPid as the default (and recommended) message broker. The RabbitMQ messaging service is provided by the *rabbitmq-server* package.

This section assumes that you have already configured a RabbitMQ message broker. For more information, refer to:

- Section 2.4, “Prerequisite Message Broker”
- Section 2.4.2, “Install and Configure the RabbitMQ Message Broker”
- Section 2.4.1, “Configure the Firewall for Message Broker Traffic”
- Section 2.4.2.1, “Launch the RabbitMQ Message Broker”

Procedure 12.3. Configuring the Telemetry service to use the RabbitMQ message broker

1. Log in as **root** to the Telemetry service node.
2. In **/etc/ceilometer/ceilometer.conf** of that system, set RabbitMQ as the RPC back end.

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  DEFAULT rpc_backend ceilometer.openstack.common.rpc.impl_kombu
```

3. Set the Telemetry service to connect to the RabbitMQ host:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  DEFAULT rabbit_host RABBITMQ_HOST
```

   Replace *RABBITMQ_HOST* with the IP address or host name of the message broker.

4. Set the message broker port to **5672**:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  DEFAULT rabbit_port 5672
```

5. Set the RabbitMQ username and password created for the Telemetry service:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  DEFAULT rabbit_userid ceilometer
```

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  DEFAULT rabbit_password CEILOMETER_PASS
```

   Where **ceilometer** and *CEILOMETER_PASS* are the RabbitMQ username and password created for Telemetry (in Section 2.4.2, “Install and Configure the RabbitMQ Message Broker”).

6. In Section 2.4.2.1, “Launch the RabbitMQ Message Broker”, we gave the **ceilometer** user read/write permissions to all resources -- specifically, through the virtual host **/**. Configure the Telemetry service to connect to this virtual host:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  DEFAULT rabbit_virtual_host /
```

12.8. Configure the Compute Node

The Telemetry service monitors each node by collecting usage data from the Compute agent (**openstack-ceilometer-compute**) installed on that node. You can configure a node's Compute agent by replicating the **/etc/ceilometer/ceilometer.conf** from another host (as in, a host whose Telemetry components have already been configured).

You will also have to configure the Compute node itself to enable notifications. This is set in the Compute agent's configuration file, namely **/etc/nova/nova.conf**:

Procedure 12.4. Enabling notifications on a Compute node

1. Install *python-ceilometer* and *python-ceilometerclient* on the node:

```
# yum install python-ceilometer python-ceilometerclient
```

2. Enable auditing on the node:

```
# openstack-config --set /etc/nova/nova.conf \
DEFAULT instance_usage_audit True
```

3. Configure the audit frequency:

```
# openstack-config --set /etc/nova/nova.conf \
DEFAULT instance_usage_audit_period hour
```

4. Configure what type of state changes should trigger a notification:

```
# openstack-config --set /etc/nova/nova.conf \
DEFAULT notify_on_state_change vm_and_task_state
```

5. Set the node to use the correct notification drivers:

```
# openstack-config --set /etc/nova/nova.conf \
DEFAULT notification_driver
nova.openstack.common.notifier.rpc_notifier
```

```
# openstack-config --set /etc/nova/nova.conf \
DEFAULT notification_driver ceilometer.compute.nova_notifier
```

Once the Compute node is configured for Telemetry, start/restart the Compute agent:

```
# service openstack-ceilometer-compute restart
```

Configure the agent to launch automatically at boot:

```
# chkconfig openstack-ceilometer-compute on
```

Finally, restart the **openstack-nova-compute** service to apply all changes to **/etc/nova/nova.conf**:

```
# service openstack-nova-compute restart
```

12.9. Configure Monitored Services

The Telemetry service can also monitor the Image, Networking, Object Storage, and Block Storage service. To enable this, you will need to configure each service to submit samples to the Collector services. The following commands demonstrate how to do this for each service. Before configuring any of these services, install the *python-ceilometer* and *python-ceilometerclient* packages on the node hosting the service first:

```
# yum install python-ceilometer python-ceilometerclient
```

Note

Restart a service after configuring it to be monitored by the Telemetry service. Doing so will allow the configuration to take effect.

Image service (`glance`)

```
# openstack-config --set /etc/glance/glance-api.conf \
 DEFAULT notifier_strategy NOTIFYMETHOD
```

Replace *NOTIFYMETHOD* with a notification queue: **`rabbit`** (to use a **`rabbitmq`** queue) or **`qpid`** (to use a **`qpid`** message queue).

Block Storage service (`cinder`)

```
# openstack-config --set /etc/cinder/cinder.conf \
 DEFAULT notification_driver
cinder.openstack.common.notifier.rpc_notifier
# openstack-config --set /etc/cinder/cinder.conf \
 DEFAULT rpc_backend cinder.openstack.common.rpc.impl_qpid
# openstack-config --set /etc/cinder/cinder.conf \
 DEFAULT control_exchange cinder
```

Object Storage service (`swift`)

The Telemetry service collects samples from the Object Storage service (**`swift`**) through the **`ResellerAdmin`** role. The steps to configure this should already have been performed when configuring the required Identity records for Telemetry.

In addition, you will also need to configure the Object Storage service to process traffic from **`ceilometer`**. To do so:

Procedure 12.5. Configuring the Object Storage service to process traffic from the Telemetry service

1. Add the following line to **`/etc/swift/proxy-server.conf`**:

```
[filter:ceilometer]
use = egg:ceilometer#swift
```

2. Add **`ceilometer`** to the **`pipeline`** directive of the same file:

```
[pipeline:main]
pipeline = healthcheck cache authtoken keystoneauth proxy-
server ceilometer
```

Networking service (neutron)

Telemetry supports the use of labels for distinguishing IP ranges. Use **openstack-config** to enable Networking integration with Telemetry:

```
# openstack-config --set /etc/neutron/neutron.conf \
 DEFAULT notification_driver
neutron.openstack.common.notifier.rpc_notifier
```

12.10. Launch the Telemetry API and Agents

Once the authentication settings and database connection strings have been configured for each component on each host, you can now launch the Telemetry service. To do so, launch the corresponding service of each component:

```
# service SERVICENAME start
```

Then, configure each service to launch automatically at boot:

```
# chkconfig SERVICENAME on
```

Replace *SERVICENAME* with the corresponding name of each Telemetry component service:

Table 12.2. Telemetry Component Service Names

Component	Service Name
openstack-ceilometer-compute	Compute agent (runs on each Compute node)
openstack-ceilometer-central	Central **ceilometer** agent
openstack-ceilometer-collector	Collector agent
openstack-ceilometer-api	API Server
openstack-ceilometer-alarm-evaluator	Evaluator (triggers alarm state transitions)
openstack-ceilometer-alarm-notifier	Notifier (executes alarm actions)
openstack-ceilometer-notification	Notification agent

Chapter 13. Nagios Installation

13.1. Install the Nagios Service

The Nagios monitoring system can be used to provide monitoring and alerts for the OpenStack network and infrastructure. The following installation procedure installs:

nagios

Nagios program that monitors hosts and services on the network, and which can send email or page alerts when a problem arises and when a problem is resolved.

nagios-devel

Includes files which can be used by Nagios-related applications.

nagios-plugins*

Nagios plugins for Nagios-related applications (including ping and nrpe).

gd

Graphics Library, for dynamically creating images

gd-devel

Development libraries for Graphics Library (*gd*)

php

HTML-embedded scripting language, used by Nagios for the web interface.

gcc*, *glibc* and *glibc-common

GNU compiler collection, together with standard programming libraries and binaries (including locale support).

openssl

OpenSSL toolkit, which provides support for secure communication between machines.

Install the required packages as the **root** user, using the **yum** command:

```
# yum install nagios nagios-devel nagios-plugins* gd gd-devel php
gcc glibc glibc-common openssl
```

Note

If any of the packages are not immediately available (for example, *gd-devel* or *gcc*), you might have to enable the optional Red Hat channel using **subscription-manager**:

```
# subscription-manager repos --enable rhel-6-server-optional-rpms
```

Nagios service placement

Consider deploying Nagios to a server that is external to the OpenStack environment, allowing it to receive diagnostic information in the event of system issues. In addition, there are a number of points to review for optimal Nagios placement:

1. Nagios services can have high CPU overhead if SSH is used.
2. Nagios should be hosted on a securely locked down server, especially if security events are being monitored. The Nagios server will receive traffic from a broad scope of systems. If security segmentation is a requirement, then this would be considered a privileged system, subject to additional firewall rules than what would apply to an OpenStack node.
3. Nagios servers may receive a considerable amount of network traffic, resulting in resource contention.

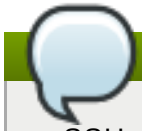
13.1.1. Install the NRPE Addon

NRPE (Nagios Remote Plugin Executor) plugins are compiled executables or scripts that are used to check the status of a host's service, and report back to the Nagios service. If the OpenStack cloud is distributed across machines, the NPRE addon can be used to run access plugin information on those remote machines.

NRPE and the Nagios plugins must be installed on each remote machine to be monitored. On the remote machine, and as the **root** user, execute the following:

```
# yum install -y nrpe nagios-plugins* openssl
```

After the installation, you can view all available plugins in the **/usr/lib64/nagios/plugins** directory (depending on the machine, they may be in **/usr/lib/nagios/plugins**).



Note

SSH can also be used to access remote Nagios plugins. However, this can result in too high a CPU load on both the Nagios host and remote machine, and is not recommended.

13.2. Configure Nagios

Nagios is composed of a server, plugins that report object/host information from both local and remote machines back to the server, a web interface, and configuration that ties all of it together.

At a minimum, the following must be done:

1. Check web-interface user name and password, and check basic configuration.
2. Add OpenStack monitoring to the local server.
3. If the OpenStack cloud includes distributed hosts:
    a. Install and configure NRPE on each remote machine (that has services to be monitored).
    b. Tell Nagios which hosts are being monitored.
    c. Tell Nagios which services are being monitored for each host.

Table 13.1. Nagios Configuration Files

File Name	Description
/etc/nagios/nagios.cfg	Main Nagios configuration file.
/etc/nagios/cgi.cfg	CGI configuration file.
/etc/httpd/conf.d/nagios.conf	Nagios configuration for httpd.
/etc/nagios/passwd	Password file for Nagios users.
/usr/local/nagios/etc/ResourceName.cfg	Contains user-specific settings.
/etc/nagios/objects/ObjectsDir/ObjectsFile.cfg	Object definition files that are used to store information about items such as services or contact groups.
/etc/nagios/nrpe.cfg	NRPE configuration file.

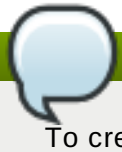
13.2.1. Configure HTTPD for Nagios

By default, when Nagios is installed, the default httpd user and password is: **nagiosadmin / nagiosadmin**. This value can be viewed in the **/etc/nagios/cgi.cfg** file.

Procedure 13.1. Configuring HTTPD for Nagios

1. Log in as the **root** user.
2. To change the default password for the user nagiosadmin, execute:

```
# htpasswd -c /etc/nagios/passwd nagiosadmin
```



Note

To create a new user, use the following command with the new user's name:

```
# htpasswd /etc/nagios/passwd newUserName
```

3. Update the **nagiosadmin** email address in **/etc/nagios/objects/contacts.cfg**

```
define contact{
    contact_name    nagiosadmin             ; Short name of user
    [...snip...]
    email           yourName@example.com    ; <<*****CHANGE
THIS******
    }
```

4. Verify that the basic configuration is working:

```
# nagios -v /etc/nagios/nagios.cfg
```

If errors occur, check the parameters set in **/etc/nagios/nagios.cfg**

5. Ensure that Nagios is started automatically when the system boots.

```
# chkconfig --add nagios
# chkconfig nagios on
```

6. Start up Nagios and restart httpd:

```
# service httpd restart
# service nagios start
```

7. Check your Nagios access by using the following URL in your browser, and using the nagiosadmin user and the password that was set in Step 1:

```
http://nagiosHostURL/nagios
```

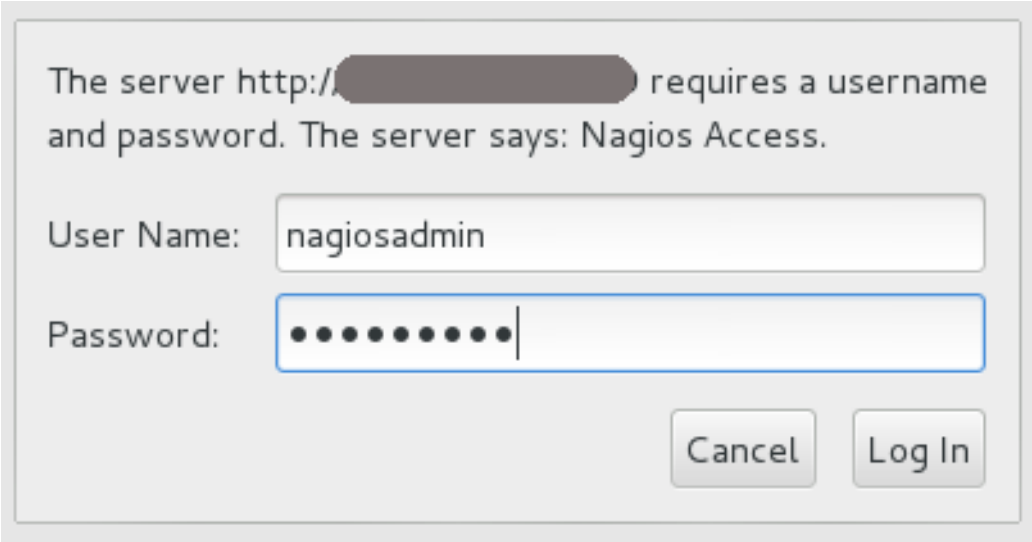


Figure 13.1. Nagios Login

Note

If the Nagios URL cannot be accessed, ensure your firewall rules has been set up correctly (refer to *Configuring the Dashboard Firewall*).

13.2.2. Configure Nagios to Monitor OpenStack Services

By default, on the Nagios server, the **/etc/nagios/objects/localhost.cfg** file is used to define services for basic local statistics (for example, swap usage or the number of current users). You can always comment these services out if they are no longer needed by prefacing each line with a '#' character. This same file can be used to add new OpenStack monitoring services.

Note

Additional service files can be used, but they must be specified as a **cfg_file** parameter in the **/etc/nagios/nagios.cfg** file.

Procedure 13.2. Configuring Nagios to monitor an OpenStack service

1. Log in as the **root** user.
2. Write a short script for the item to be monitored (for example, whether a service is running), and place it in the **/usr/lib64/nagios/plugins** directory.

   For example, the following script checks the number of Compute instances, and is stored in a file named **nova-list**:

```
#!/bin/env bash
export OS_USERNAME=userName
export OS_TENANT_NAME=tenantName
export OS_PASSWORD=password
export OS_AUTH_URL=http://identityURL:35357/v2.0/

data=$(nova list  2>&1)
rv=$?

if [ "$rv" != "0" ] ; then
    echo $data
    exit $rv
fi

echo "$data" | grep -v -e '--------' -e '| Status |' -e '^$' | wc
-l
```

3. Ensure the script is executable:

```
# chmod u+x nova-list
```

4. In the **/etc/nagios/objects/commands.cfg** file, specify a command section for each new script:

```
define command {
        command_line
/usr/lib64/nagios/plugins/nova-list
        command_name                    nova-list
}
```

5. In the **/etc/nagios/objects/localhost.cfg** file, define a service for each new item, using the defined command. For example:

```
define service {
        check_command   nova-list
        host_name       localURL
        name            nova-list
```

```
        normal_check_interval   5
        service_description     Number of nova vm instances
        use             generic-service
        }
```

6. Restart nagios using:

```
# service nagios restart
```

13.2.3. Configure NRPE

To set up monitoring on each remote machine, execute the following as the **root** user:

Procedure 13.3. Configuring monitoring on a remote machine

1. In the **/etc/nagios/nrpe.cfg** file, add the central Nagios server IP address in the **allowed_hosts** line:

```
allowed_hosts=127.0.0.1, NagiosServerIP
```

2. In the **/etc/nagios/nrpe.cfg** file, add any commands to be used to monitor the OpenStack services. For example:

```
command[keystone]=/usr/lib64/nagios/plugins/check_procs -c 1: -w 3:
-C keystone-all
```

Each defined command can then be specified in the **services.cfg** file on the Nagios monitoring server (refer to Section 13.2.5, “Create Service Definitions for Remote Services”).

Note

Any complicated monitoring can be placed into a script, and then referred to in the command definition. For an OpenStack script example, refer to Section 13.2.2, “Configure Nagios to Monitor OpenStack Services”.

3. Next, configure the firewall to allow **nrpe** traffic.

4. Start the NRPE service:

```
# service nrpe start
```

13.2.4. Create Host Definitions

If additional machines are being used in the cloud, in addition to the host on which Nagios is installed, they must be made known to Nagios by configuring them in an objects file.

Procedure 13.4. Creating host definitions through an objects file

1. Log in as the **root** user.

2. In the **/etc/nagios/objects** directory, create a **hosts.cfg** file.

3. In the file, specify a **host** section for each machine on which an OpenStack service is running and should be monitored:

```
define host{
    use linux-server
    host_name remoteHostName
    alias remoteHostAlias
    address remoteAddress
}
```

where:

- **host_name** = Name of the remote machine to be monitored (typically listed in the local **/etc/hosts** file). This name is used to reference the host in service and host group definitions.
- **alias** = Name used to easily identify the host (typically the same as the **host_name**).
- **address** = Host address (typically its IP address, although a FQDN can be used instead, just make sure that DNS services are available).

For example:

```
define host{
  host_name     Server-ABC
  alias  OS-ImageServices
  address 192.168.1.254
}
```

4. In the **/etc/nagios/nagios.cfg** file, under the **OBJECT CONFIGURATION FILES** section, specify the following line:

```
cfg_file=/etc/nagios/objects/hosts.cfg
```

13.2.5. Create Service Definitions for Remote Services

To monitor remote services, you must define those services in a new file (in this procedure, **/etc/nagios/objects/services.cfg**)

Procedure 13.5. Creating service definitions

1. Log in as the **root** user.
2. In the **/etc/nagios/objects/commands.cfg** file, specify the following to handle the use of the **check_nrpe** plugin with remote scripts or plugins:

```
define command{
        command_name    check_nrpe
        command_line    $USER1$/check_nrpe -H $HOSTADDRESS$ -c
$ARG1$
 }
```

3. In the **/etc/nagios/objects** directory, create the **services.cfg** file.

4. In the file, specify the following **service** sections for each remote OpenStack host to be monitored:

```
##Basic remote checks#############
##Remember that remoteHostName is defined in the hosts.cfg file.

define service{
 use generic-service
 host_name remoteHostName
 service_description PING
 check_command check_ping!100.0,20%!500.0,60%
}

define service{
 use generic-service
 host_name remoteHostName
 service_description Load Average
 check_command check_nrpe!check_load
}

##OpenStack Service Checks#######
define service{
 use generic-service
 host_name remoteHostName
 service_description Identity Service
 check_command check_nrpe!keystone
}
```

   The above sections ensure that a server heartbeat, load check, and the OpenStack Identity service status are reported back to the Nagios server. All OpenStack services can be reported, just ensure that a matching command is specified in the remote server's **nrpe.cfg** file.

5. In the **/etc/nagios/nagios.cfg** file, under the **OBJECT CONFIGURATION FILES** section, specify the following line:

```
cfg_file=/etc/nagios/objects/services.cfg
```

13.2.6. Verify the Nagios Configuration

Procedure 13.6. Verifying the Nagios configuration

1. Log in as the **root** user.
2. Verify that the updated configuration is working:

```
# nagios -v /etc/nagios/nagios.cfg
```

   If errors occur, check the parameters set in **/etc/nagios/nagios.cfg**, **/etc/nagios/services.cfg**, and **/etc/nagios/hosts.cfg**.

3. Restart Nagios:

```
# service nagios restart
```

4. Log into the Nagios dashboard again by using the following URL in your browser, and using the **`nagiosadmin`** user and the password that was set in Step 1:

```
http://nagiosHostURL/nagios
```

Chapter 14. Remote Logging Installation and Configuration

14.1. Introduction to Remote Logging

All systems generate and update log files recording their actions and any problems they encounter. In a distributed or cloud computing environment that contains many systems collecting these log files in a central location simplifies debugging.

The **rsyslog** service provides facilities both for running a centralized logging server and for configuring individual systems to send their log files to the centralized logging server. This is referred to as configuring the systems for "remote logging".

14.2. Install rsyslog Server

The *rsyslog* package must be installed on the system that you intend to use as a centralized logging server and all systems that will be configured to send logs to it. To do so, log in as the **root** user and install the *rsyslog* package:

```
# yum install rsyslog
```

The *rsyslog* package is installed and ready to be configured.

14.3. Configure rsyslog on the Centralized Logging Server

The steps in this procedure must be followed on the system that you intend to use as your centralized logging sever. All steps in this procedure must be run while logged in as the **root** user.

Procedure 14.1. Configuring rsyslog on the centralized logging server

1. Configure SELinux to allow **rsyslog** traffic.

   ```
   # semanage -a -t syslogd_port_t -p udp 514
   ```

2. Open the **/etc/rsyslog.conf** file in a text editor.

   a. Add this line to the file, defining the location logs will be saved to:

   ```
   $template TmplMsg, "/var/log/%HOSTNAME%/%PROGRAMNAME%.log"
   $template TmplAuth, "/var/log/%HOSTNAME%/%PROGRAMNAME%.log"

   authpriv.*   ?TmplAuth
   *.info,mail.none,authpriv.none,cron.none   ?TmplMsg
   ```

   b. Remove the comment character (#) from the beginning of these lines in the file:

   ```
   #$ModLoad imudp
   #$UDPServerRun 514
   ```

   Save the changes to the **/etc/rsyslog.conf** file.

Your centralized log server is now configured to receive and store log files from the other systems in your environment.

14.4. Configure rsyslog on Individual Nodes

Apply the steps listed in this procedure to each of your systems to configure them to send logs to a centralized log server. All steps listed in this procedure must be performed while logged in as the **root** user.

Procedure 14.2. Configuring a node to send logs to a centralized log server

- Edit the **/etc/rsyslog.conf**, and specify the address of your centralized log server by adding the following:

```
*.*    @YOURSERVERADDRESS:YOURSERVERPORT
```

Replace *YOURSERVERADDRESS* with the address of the centralized logging server. Replace *YOURSERVERPORT* with the port on which the **rsyslog** service is listening. For example:

```
*.*    @192.168.20.254:514
```

Or:

```
*.*    @log-server.company.com:514
```

The single @ specifies the UDP protocol for transmission. Use a double @@ to specify the TCP protocol for transmission.

Important

The use of the wildcard * character in these example configurations indicates to **rsyslog** that log entries from all log facilities and of all log priorities must be sent to the remote **rsyslog** server.

For information on applying more precise filtering of log files refer to the manual page for the **rsyslog** configuration file, **rsyslog.conf**. Access the manual page by running the command **man rsyslog.conf**.

Once the **rsyslog** service is started or restarted the system will send all log messages to the centralized logging server.

14.5. Start the rsyslog Server

The **rsyslog** service must be running on both the centralized logging server and the systems attempting to log to it.

The steps in this procedure must be performed while logged in as the **root** user.

Procedure 14.3. Launching the rsyslog service

1. Use the **service** command to start the **rsyslog** service.

```
# service rsyslog start
```

2. Use the **chkconfig** command to ensure the **rsyslog** service starts automatically in future.

```
# chkconfig rsyslog on
```

The **rsyslog** service has been started. The service will start sending or receiving log messages based on its local configuration.

Appendix

The following sections provide supplementary information for this guide.

A.1. Backup and Restore

Both a backup and a recovery policy should be formulated to minimize data loss and system downtime.

When determining your backup strategy, you will need to answer the following questions:

- How quickly will you need to recover from data loss? If you cannot have data loss at all, you should focus on High Availability as a deployment strategy, in addition to using backups.
- How many backups should you keep?
- Should your backups be kept off-site?
- How often should backups be tested?
- What will be backed up? Critical data in OpenStack is backed up per component (or service).

The following sections describe database and file-system backups for components, as well as information on recovering backups.

A.1.1. Database Backup

The installation process in this guide uses the MariaDB server, which hosts the databases for the separate OpenStack components. With all these databases in one place, it is easy to create a database backup.

Note

If you are using a different database server, refer to your server's notes for more information. For example, if you are using MariaDB and MyISAM tables, consider using **`mysqlhotcopy`** to improve backup and recovery times.

Procedure A.1. Backing up all databases

1. Log in as the database root user.
2. Export database information into a backup file:

```
# mysqldump --opt --all-databases > fileName
```

For example:

```
# mysqldump --opt --all-databases > openstack.sql
```

Procedure A.2. Backing up a single database

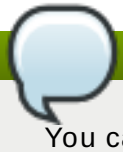
1. Log in as the database root user.

2. Export a single database into a backup file:

```
# mysqldump --opt databaseName > fileName
```

For example:

```
# mysqldump --opt nova > nova.sql
```



Note

You can automate the backup process by creating a cron job that runs the following script once per day:

```
#!/bin/bash
backup_dir="/var/lib/backups/mysql"
filename="${backup_dir}/mysql-`hostname`-`eval date +%Y%m%d`.sql.gz"

# Dump the entire MariaDB database
/usr/bin/mysqldump --opt --all-databases | gzip > $filename

# Delete backups older than 7 days
find $backup_dir -ctime +7 -type f -name mysql-\*sql.gz -delete
```

A.1.2. File System Backup

Each OpenStack component has its own directory and files under **/etc**, **/var/lib**, and **/var/log**. Files in these directories should be regularly backed up:

/etc/component

Files in this directory (for example, **/etc/glance**) should be regularly backed up. This will be the case for all nodes on which the component is housed. In particular:

- **/etc/nova** will need to be backed up on both the cloud controller and all Compute nodes.
- **/etc/swift** is very important, because it contains Object Storage configuration files, ring files, and ring builder files. Losing these files will render the data on your cluster inaccessible. A best practice is to copy the builder files to all storage nodes along with the ring files. Try to spread multiple backup copies throughout your storage cluster.

/var/lib/component

It is good practice to back up this directory (for example, **/var/lib/keystone**) for all components. In particular:

- **/var/lib/glance/images** - If you are using a file-based backend for the Image service, this directory stores its images and should be backed up regularly. Using both of the following should ensure image availability:
  - Run the directory on a RAID array; if a disk fails, the directory is available.
  - Use a tool such as **rsync** to replicate the images to another server:

```
# rsync -az --progress /var/lib/glance/images backup-
server: \
/var/lib/glance/images/
```

- **/var/lib/nova** - All files in this directory should be backed up. The only exception is the **/var/lib/nova/instances** subdirectory on Compute nodes. This subdirectory contains the KVM images of running instances. You would only want to back up this directory if you need to maintain backup copies of all instances. Under most circumstances, you do not need to do this, but this can vary from cloud to cloud and your service levels. Also, be aware that making a backup of a live KVM instance can cause that instance to not boot properly if it is ever restored from a backup.

/var/log/component

Log files should be regularly backed up, unless you have all logs going to a central server. It is highly recommended to use a central logging server.

Note

Various tools can be used to backup files. Red Hat Enterprise Linux supports the following:

- **tar**

  For example, to create an archive file called **var-lib-cinder-backup.tar** in **/mnt/backup/**, run:

  ```
  # tar cf /mnt/backup/var-lib-cinder-backup.tar /var/lib/cinder
  ```

  The resulting archive file contains the contents of the **/var/lib/cinder** directory, and will be nearly as large as the data being backed up. Depending on the type of data being backed up, compressing the archive file can result in significant size reductions. The archive file can be compressed by adding a single option to the previous command:

  ```
  # tar czf /mnt/backup/var-lib-cinder-backup.tar.gz
  /var/lib/cinder
  ```

  The resulting **var-lib-cinder-backup.tar.gz** archive file is now **gzip** compressed.

- **cpio**

  Unlike **tar**, **cpio** reads the names of the files it processes through standard input. For example:

  ```
  # find /var/lib/cinder/ | cpio -o > /mnt/backup/var-lib-
  cinder-backup.cpio
  ```

  This command creates a **cpio** archive file (containing everything in **/var/lib/cinder/**) called **var-lib-cinder-backup.cpio** and places it in the **/mnt/backup/** directory.

A.1.3. Recovery

Recovery involves the simple procedure of stopping services, restoring files and databases for the component, then restarting the services. All procedures must be carried out as the root user.

Procedure A.3. Recover database backups and files

1. Ensure all services are stopped for the component. For example, for the Image service:

```
# service openstack-glance-registry stop
# service openstack-glance-api stop
```

2. Stop MariaDB:

```
# service mariadb stop
```

3. Import the previously backed-up database. For example, to import an Image service back up:

```
# mysql glance < glance.sql
```

4. Copy backup files over to the necessary directories. For example, for Image files:

```
# mv /etc/glance{,.orig}
# cp -a /path/to/backup/glance /etc/
```

5. Restart the component's services. For example, to restart the Image service you will have to restart the SQL database, Image registry, and Image API.

```
# service mariadb start
# service openstack-glance-registry start
# service openstack-glance-api start
```

A.2. Miscellaneous Service Log Files

The following table enumerates the different log files used by the dashboard, Identity, Image, and Object Storage services. These files are useful for troubleshooting startup/installation issues.

Each file is located on the host of its corresponding service.

Table A.1. Log Files

File name	Description
/var/log/glance/api.log	The log of the Image API service (**openstack-glance-api**).
/var/log/glance/registry.log	The log of the Image registry service (**openstack-glance-registry**).
/var/log/keystone/keystone.log	The log of the Identity service (**openstack-keystone**).
/var/log/swift/swift-startup.log	The log for the object storage proxy service (**openstack-swift-proxy**).

For a list of the different log files used by all other services, refer to the *Red Hat Enterprise Linux OpenStack Platform 5 Configuration Reference Guide*. This document is available from the following link:

https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux_OpenStack_Platform

Revision History

Revision 6.0.0-8 **Tue Apr 21 2015** **Don Domingo**

BZ#1205021 - Corrected service name for OpenStack Data Processing
BZ#1208622 - Updated commands used for OpenStack Data Processing authentication setup

Revision 6.0.0-7 **Fri Apr 17 2015** **Don Domingo**

BZ#1210544 - Corrected default ports used by Object Storage service

Revision 6.0.0-6 **Wed Apr 15 2015** **Don Domingo**

BZ#1210544 - Corrected recommended filesystem format for Swift storage nodes (XFS)

Revision 6.0.0-5 **Wed Mar 11 2015** **Martin Lopes**

BZ#1191817 - Added step for 'iptables-services' installation

Revision 6.0.0-4 **Mon Feb 23 2015** **Summer Long**

BZ#1175503 - Added new section 'Configure Rsyncd' for Object Storage.

Revision 6.0.0-3 **Mon Jan 19 2015** **Summer Long**

BZ#1181347 - Removed architecture information, added reference to new "Component Overview".

Revision 6.0.0-2 **Thu Dec 18 2014** **Martin Lopes**

BZ#1175569 - Updated neutron service descriptions.

Revision 6.0.0-1 **Tue Dec 9 2014** **Don Domingo**

Red Hat Enterprise Linux OpenStack 6.0 initial build.