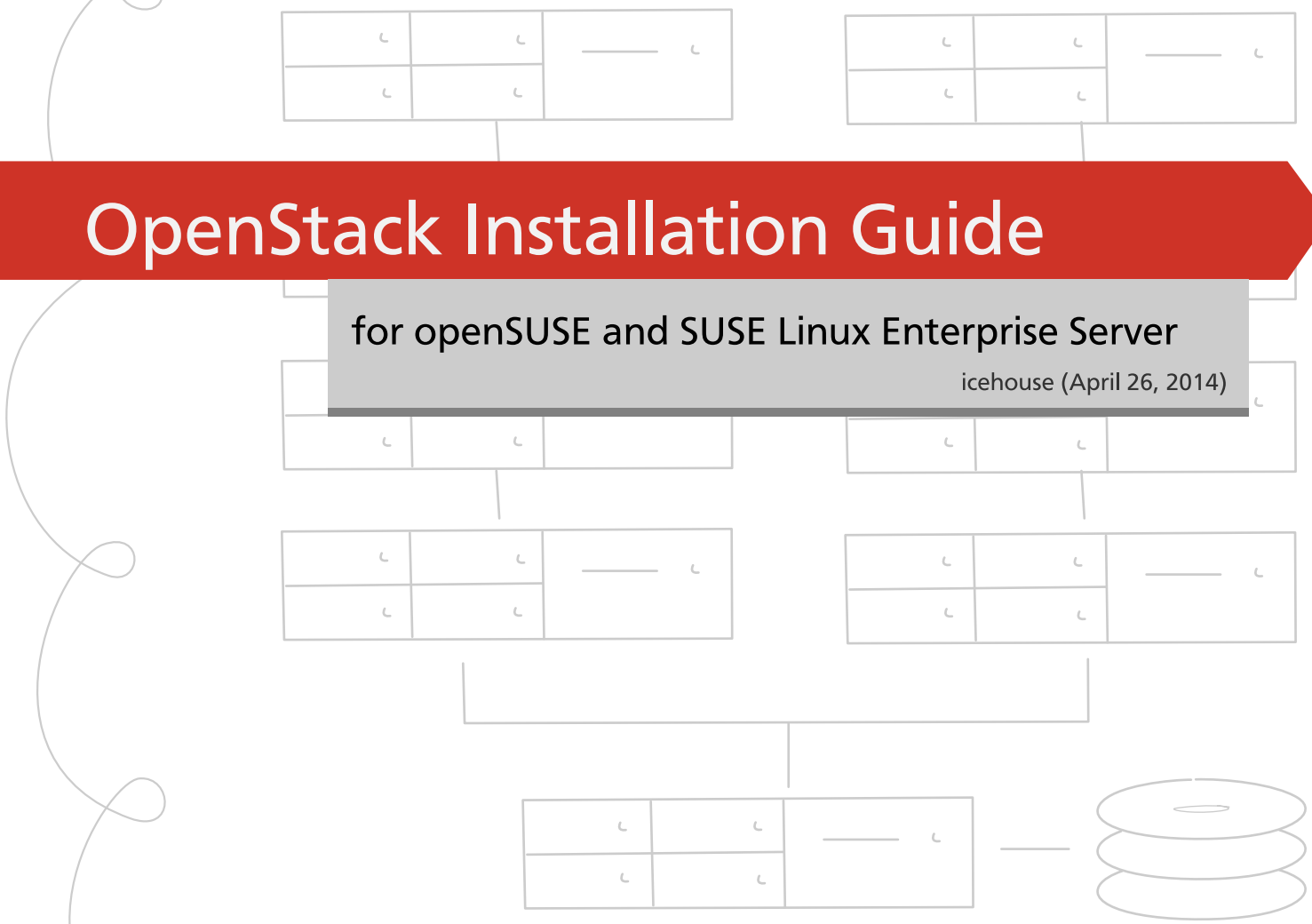
OpenStack Installation Guide

for openSUSE and SUSE Linux Enterprise Server

icehouse (April 26, 2014)



BUILT FOR

openstack™

CLOUD SOFTWARE

openstack™

docs.openstack.org

OpenStack Installation Guide for openSUSE and SUSE Linux Enterprise Server

icehouse (2014-04-26)

The OpenStack® system consists of several key projects that you install separately but that work together depending on your cloud needs. These projects include Compute, Identity Service, Networking, Image Service, Block Storage, Object Storage, Telemetry, Orchestration, and Database. You can install any of these projects separately and configure them stand-alone or as connected entities. This guide shows you how to install OpenStack by using packages on openSUSE through the Open Build Service Cloud repository. Explanations of configuration options and sample configuration files are included.

Table of Contents

List of Figures

List of Tables

Preface

Conventions

The OpenStack documentation uses several typesetting conventions.

Notices

Notices take three forms:

Note

The information in a note is usually in the form of a handy tip or reminder.

Important

The information in an important notice is something you must be aware of before proceeding.

Warning

The information in warnings is critical. Warnings provide additional information about risk of data loss or security issues.

Command prompts

Commands prefixed with the # prompt are to be executed by the `root` user. These examples can also be executed by using the **sudo** command, if available.

Commands prefixed with the $ prompt can be executed by any user, including `root`.

Document change history

This version of the guide replaces and obsoletes all previous versions. The following table describes the most recent changes:

Revision Date	Summary of Changes
April 16, 2014	• Update for Icehouse, rework Networking setup to use ML2 as plugin, add new chapter for Database Service setup, improved basic configuration.
October 25, 2013	• Added initial Debian support.
October 17, 2013	• Havana release.
October 16, 2013	• Add support for SUSE Linux Enterprise.
October 8, 2013	• Complete reorganization for Havana.
September 9, 2013	• Build also for openSUSE.
August 1, 2013	• Fixes to Object Storage verification steps. Fix bug 1207347.
July 25, 2013	• Adds creation of cinder user and addition to the service tenant. Fix bug 1205057.
May 8, 2013	• Updated the book title for consistency.

Revision Date	Summary of Changes
May 2, 2013	• Updated cover and fixed small errors in appendix.

1. Architecture

Table of Contents

Overview

The *OpenStack* project is an open source cloud computing platform that supports all types of cloud environments. The project aims for simple implementation, massive scalability, and a rich set of features. Cloud computing experts from around the world contribute to the project.

OpenStack provides an Infrastructure-as-a-Service (*IaaS*) solution through a variety of complemental services. Each service offers an application programming interface (*API*) that facilitates this integration. The following table provides a list of OpenStack services:

Table 1.1. OpenStack services

Service	Project name	Description
Dashboard	*Horizon*	Provides a web-based self-service portal to interact with underlying OpenStack services, such as launching an instance, assigning IP addresses and configuring access controls.
Compute	*Nova*	Manages the lifecycle of compute instances in an OpenStack environment. Responsibilities include spawning, scheduling and decomissioning of virtual machines on demand.
Networking	*Neutron*	Enables network connectivity as a service for other OpenStack services, such as OpenStack Compute. Provides an API for users to define networks and the attachments into them. Has a pluggable architecture that supports many popular networking vendors and technologies.
Storage		
Object Storage	*Swift*	Stores and retrieves arbitrary unstructured data objects via a *RESTful*, HTTP based API. It is highly fault tolerant with its data replication and scale out architecture. Its implementation is not like a file server with mountable directories.
Block Storage	*Cinder*	Provides persistent block storage to running instances. Its pluggable driver architecture facilitates the creation and management of block storage devices.
Shared services		
Identity service	*Keystone*	Provides an authentication and authorization service for other OpenStack services. Provides a catalog of endpoints for all OpenStack services.
Image Service	*Glance*	Stores and retrieves virtual machine disk images. OpenStack Compute makes use of this during instance provisioning.
Telemetry	*Ceilometer*	Monitors and meters the OpenStack cloud for billing, benchmarking, scalability, and statistical purposes.

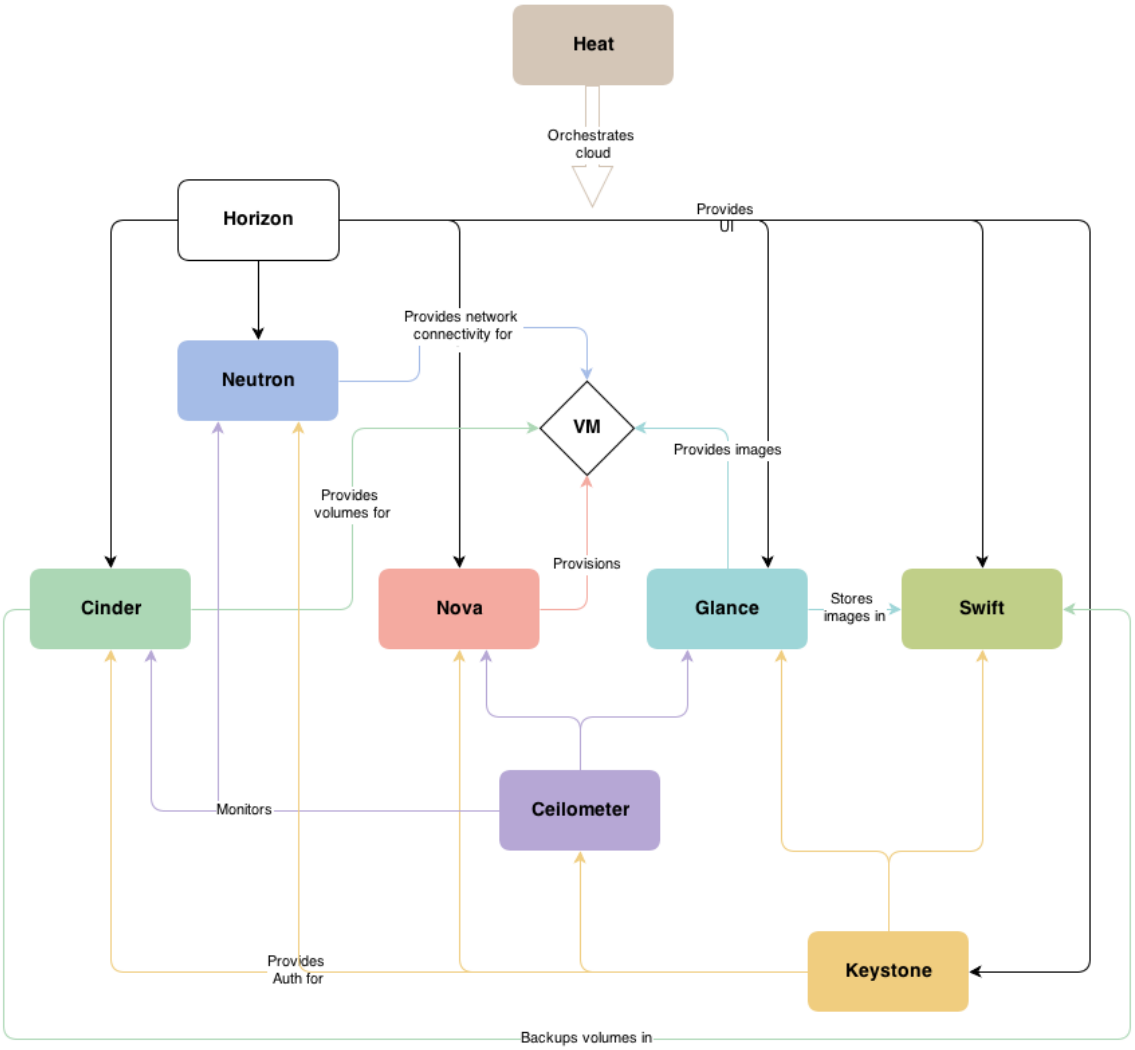
Service	Project name	Description
Higher-level services		
Orchestration	*Heat*	Orchestrates multiple composite cloud applications by using either the native *HOT* template format or the AWS CloudFormation template format, through both an OpenStack-native REST API and a CloudFormation-compatible Query API.
Database Service	*Trove*	Provides scalable and reliable Cloud Database-as-a-Service functionality for both relational and non-relational database engines.

This guide describes how to deploy these services in a functional test environment and, by example, teaches you how to build a production environment.

Conceptual architecture

Launching a virtual machine or instance involves many interactions among several services. The following diagram provides the conceptual architecture of a typical OpenStack environment.

Figure 1.1. Conceptual architecture



Example architectures

OpenStack is highly configurable to meet different needs with various compute, networking, and storage options. This guide enables you to choose your own OpenStack adventure using a combination of basic and optional services. This guide uses the following example architectures:

- Three-node architecture with OpenStack Networking (neutron). See Figure 1.2, "Three-node architecture with OpenStack Networking (neutron)" [4].

  - The basic controller node runs the Identity service, Image Service, management portions of Compute and Networking, Networking plug-in, and the dashboard. It also includes supporting services such as a database, *message broker*, and *Network Time Protocol (NTP)*.

    Optionally, the controller node also runs portions of Block Storage, Object Storage, Database Service, Orchestration, and Telemetry. These components provide additional features for your environment.

  - The network node runs the Networking plug-in, layer 2 agent, and several layer 3 agents that provision and operate tenant networks. Layer 2 services include provisioning of virtual networks and tunnels. Layer 3 services include routing, *NAT* , and *DHCP*. This node also handles external (internet) connectivity for tenant virtual machines or instances.

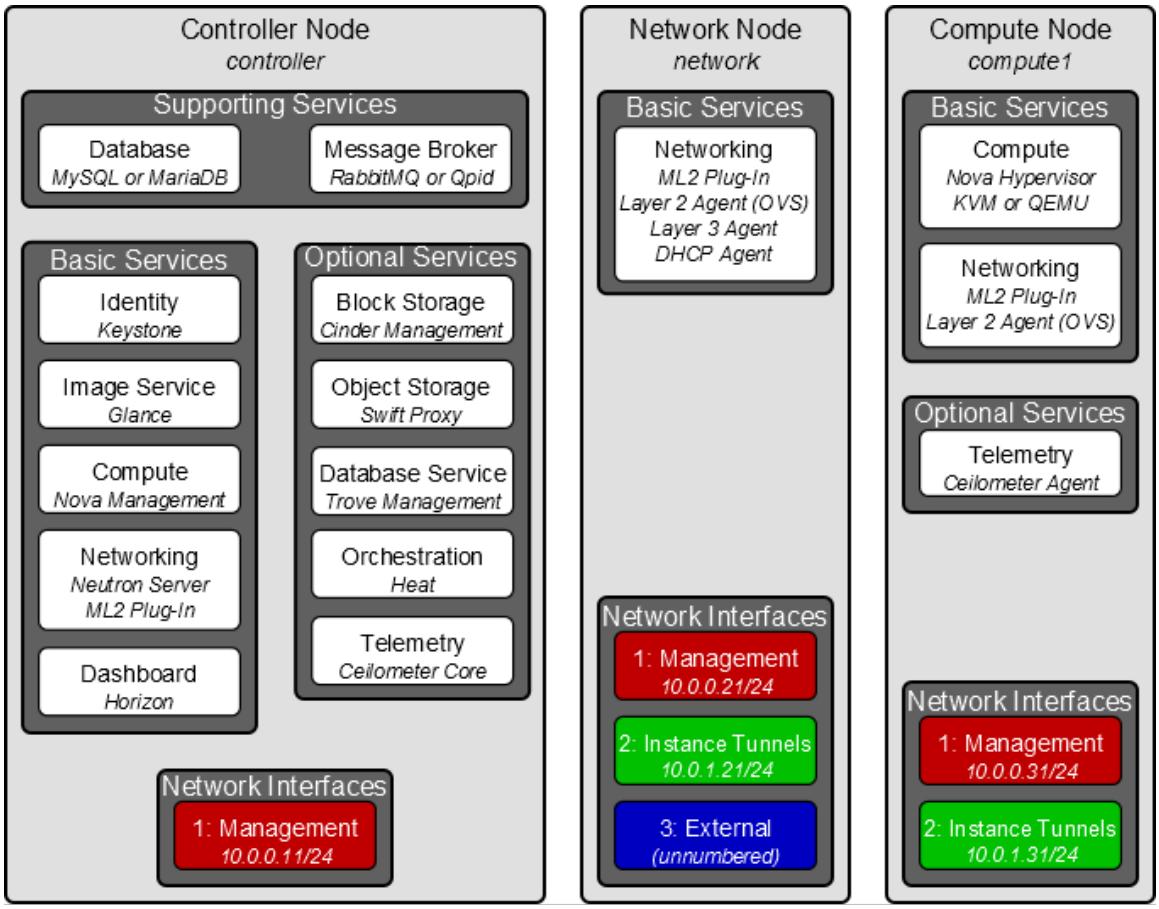
  - The compute node runs the hypervisor portion of Compute, which operates tenant virtual machines or instances. By default Compute uses KVM as the hypervisor. The compute node also runs the Networking plug-in and layer 2 agent which operate tenant networks and implement security groups. You can run more than one compute node.

    Optionally, the compute node also runs the Telemetry agent. This component provides additional features for your environment.

Note

When you implement this architecture, skip the section called "Legacy networking (nova-network)" [67] in Chapter 7, "Add a networking service" [48]. To use optional services, you might need to install additional nodes, as described in subsequent chapters.

Figure 1.2. Three-node architecture with OpenStack Networking (neutron)



- Two-node architecture with legacy networking (nova-network). See Figure 1.3, "Two-node architecture with legacy networking (nova-network)" [5].

  - The basic *controller node* runs the Identity service, Image Service, management portion of Compute, and the dashboard necessary to launch a simple instance. It also includes supporting services such as a database, message broker, and NTP.

    Optionally, the controller node also runs portions of Block Storage, Object Storage, Database Service, Orchestration, and Telemetry. These components provide additional features for your environment.

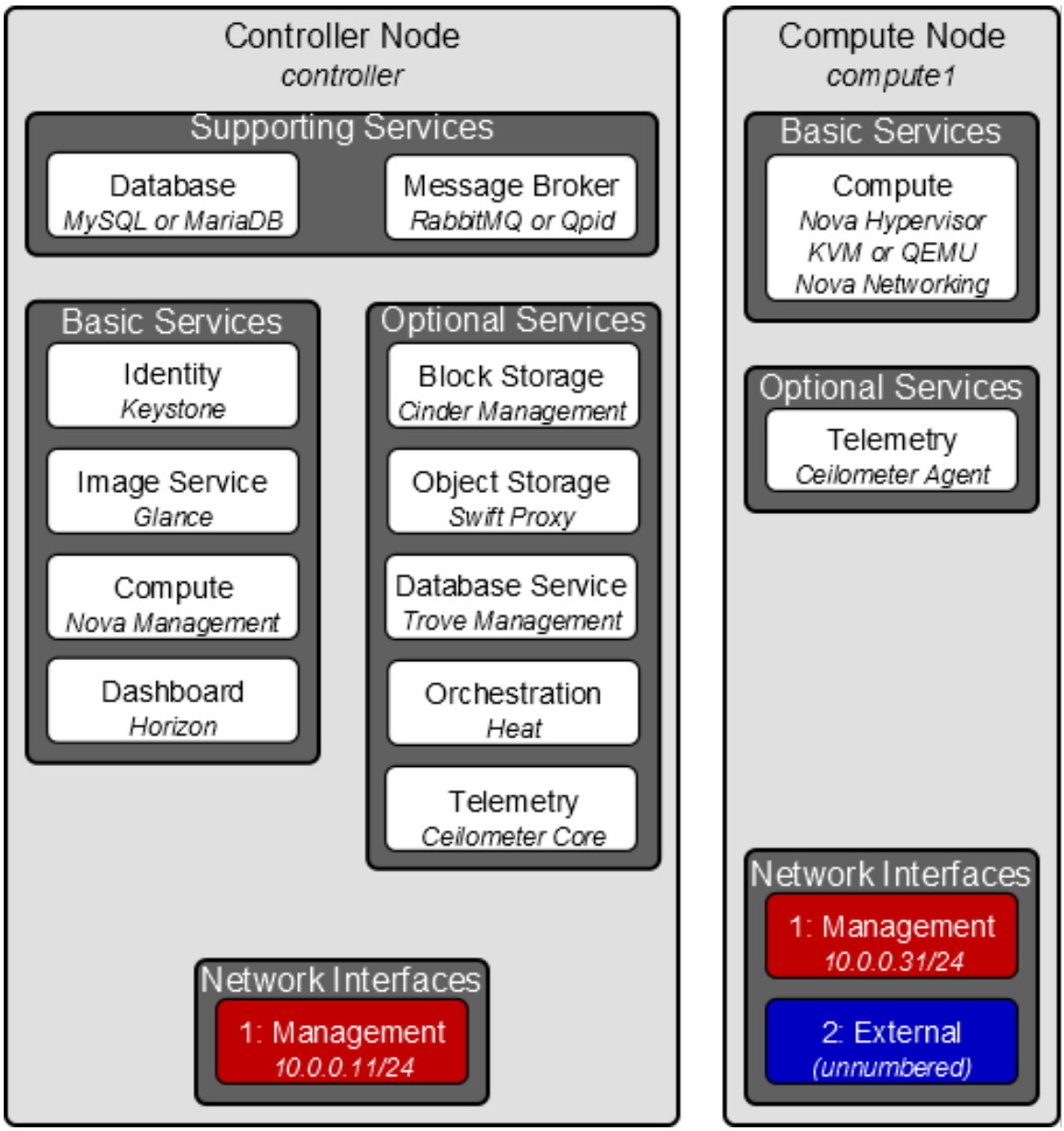
  - The basic *compute node* runs the *hypervisor* portion of Compute, which operates *tenant virtual machines* or instances. By default, Compute uses *KVM* as the *hypervisor*. Compute also provisions and operates tenant networks and implements *security groups*. You can run more than one compute node.

    Optionally, the compute node also runs the Telemetry agent. This component provides additional features for your environment.

Note

When you implement this architecture, skip the section called "OpenStack Networking (neutron)" [48] in Chapter 7, "Add a networking service" [48]. To use optional services, you might need to install additional nodes, as described in subsequent chapters.

Figure 1.3. Two-node architecture with legacy networking (nova-network)



2. Basic environment configuration

Table of Contents

This chapter explains how to configure each node in the example architectures including the two-node architecture with legacy networking and three-node architecture with OpenStack Networking (neutron).

Note

Although most environments include OpenStack Identity, Image Service, Compute, at least one networking service, and the dashboard, OpenStack Object Storage can operate independently of most other services. If your use case only involves Object Storage, you can skip to the section called "System requirements for Object Storage" [84]. However, the dashboard will not work without at least OpenStack Image Service and Compute.

Note

You must use an account with administrative privileges to configure each node. Either run the commands as the `root` user or configure the `sudo` utility.

Before you begin

For a functional environment, OpenStack doesn't require a significant amount of resources. We recommend that your environment meets or exceeds the following minimum requirements which can support several minimal *CirrOS* instances:

- Controller Node: 1 processor, 2 GB memory, and 5 GB storage
- Network Node: 1 processor, 512 MB memory, and 5 GB storage
- Compute Node: 1 processor, 2 GB memory, and 10 GB storage

To minimize clutter and provide more resources for OpenStack, we recommend a minimal installation of your Linux distribution. Also, we strongly recommend that you install a 64-bit version of your distribution on at least the compute node. If you install a 32-bit version of your distribution on the compute node, attempting to start an instance using a 64-bit image will fail.

Note

A single disk partition on each node works for most basic installations. However, you should consider *Logical Volume Manager (LVM)* for installations with optional services such as Block Storage.

Many users build their test environments on *virtual machines (VMs)*. The primary benefits of VMs include the following:

- One physical server can support multiple nodes, each with almost any number of network interfaces.
- Ability to take periodic "snap shots" throughout the installation process and "roll back" to a working configuration in the event of a problem.

However, VMs will reduce performance of your instances, particularly if your hypervisor and/or processor lacks support for hardware acceleration of nested VMs.

Note

If you choose to install on VMs, make sure your hypervisor permits *promiscuous mode* on the *external network*.

For more information about system requirements, see the OpenStack Operations Guide.

Networking

After installing the operating system on each node for the architecture that you choose to deploy, you must configure the network interfaces. We recommend that you disable any automated network management tools and manually edit the appropriate configuration files for your distribution. For more information on how to configure networking on your distribution, see the SLES 11 or openSUSE documentation.

To disable `NetworkManager`:

- Use the YaST network module:

```
# yast2 network
```

  For more information, see the SLES or the openSUSE documentation.

Proceed to network configuration for the example OpenStack Networking (neutron) or legacy networking (nova-network) architecture.

OpenStack Networking (neutron)

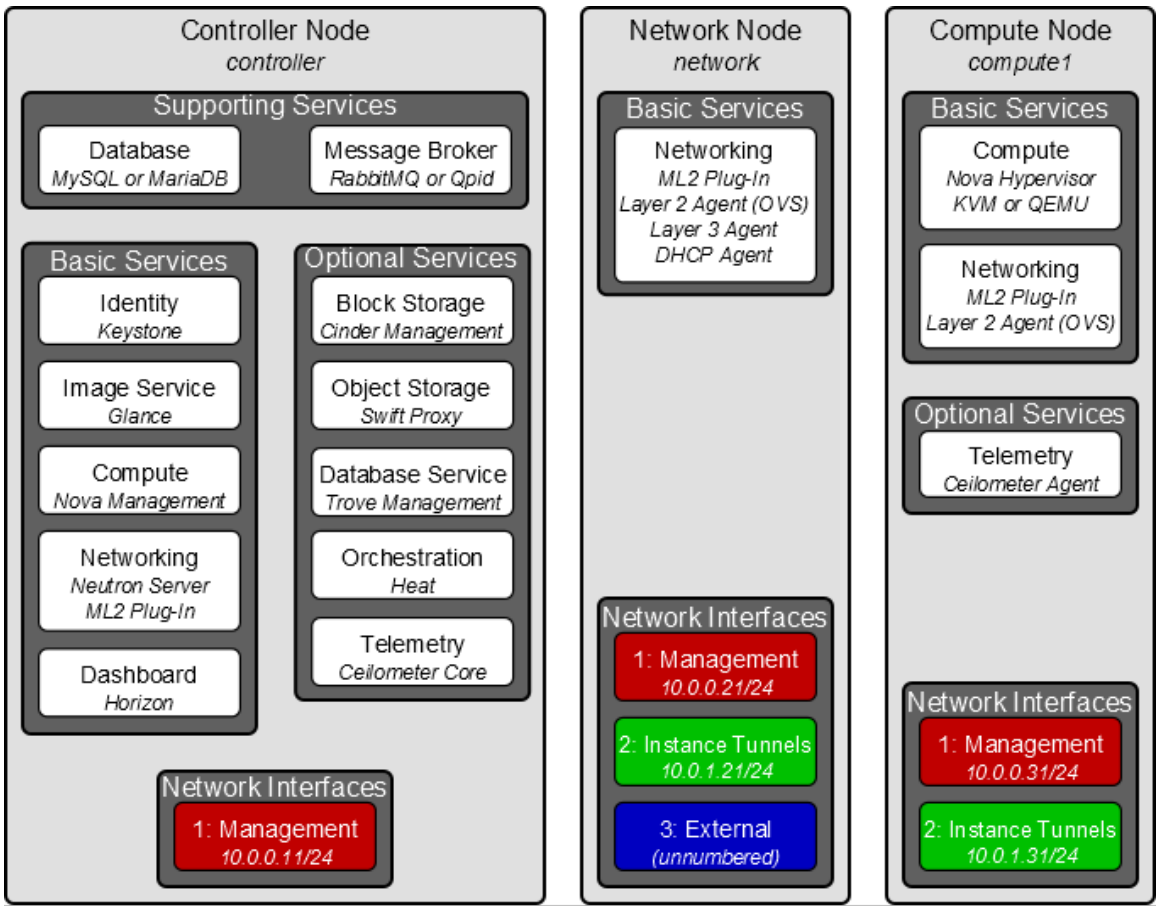
The example architecture with OpenStack Networking (neutron) requires one controller node, one network node, and at least one compute node. The controller node contains one network interface on the *management network*. The network node contains one network interface on the management network, one on the *instance tunnels network*, and

one on the *external network*. The compute node contains one network interface on the management network and one on the instance tunnels network.

Note

Network interface names vary by distribution. Traditionally, interfaces use "eth" followed by a sequential number. To cover all variations, this guide simply refers to the first interface as the interface with the lowest number, the second interface as the interface with the middle number, and the third interface as the interface with the highest number.

Figure 2.1. Three-node architecture with OpenStack Networking (neutron)



Unless you intend to use the exact configuration provided in this example architecture, you must modify the networks in this procedure to match your environment. Also, each node must resolve the other nodes by name in addition to IP address. For example, the `controller` name must resolve to `10.0.0.11`, the IP address of the management interface on the controller node.

Warning

Reconfiguring network interfaces will interrupt network connectivity. We recommend using a local terminal session for these procedures.

Controller node

To configure networking:

- Configure the first interface as the management interface:

  IP address: 10.0.0.11

  Network mask: 255.255.255.0 (or /24)

  Default gateway: 10.0.0.1

To configure name resolution:

- Edit the `/etc/hosts` file to contain the following:

```
# controller
10.0.0.11       controller

# network
10.0.0.21       network

# compute1
10.0.0.31       compute1
```

Network node

To configure networking:

1. Configure the first interface as the management interface:

   IP address: 10.0.0.21

   Network mask: 255.255.255.0 (or /24)

   Default gateway: 10.0.0.1

2. Configure the second interface as the instance tunnels interface:

   IP address: 10.0.1.21

   Network mask: 255.255.255.0 (or /24)

3. The external interface uses a special configuration without an IP address assigned to it. Configure the third interface as the external interface:

   Replace *`INTERFACE_NAME`* with the actual interface name. For example, *eth2* or *ens256*.

   - Edit the `/etc/sysconfig/network/ifcfg-`*`INTERFACE_NAME`* file to contain the following:

```
STARTMODE='auto'
BOOTPROTO='static'
```

4. Restart networking:

```
# service network restart
```

To configure name resolution:

- Edit the `/etc/hosts` file to contain the following:

```
# network
10.0.0.21       network

# controller
10.0.0.11       controller

# compute1
10.0.0.31       compute1
```

Compute node

To configure networking:

1. Configure the first interface as the management interface:

   IP address: 10.0.0.31

   Network mask: 255.255.255.0 (or /24)

   Default gateway: 10.0.0.1

   Note

   Additional compute nodes should use 10.0.0.32, 10.0.0.33, and so on.

2. Configure the second interface as the instance tunnels interface:

   IP address: 10.0.1.31

   Network mask: 255.255.255.0 (or /24)

   Note

   Additional compute nodes should use 10.0.1.32, 10.0.1.33, and so on.

To configure name resolution:

- Edit the `/etc/hosts` file to contain the following:

```
# compute1
10.0.0.31       compute1

# controller
10.0.0.11       controller

# network
10.0.0.21       network
```

Verify connectivity

We recommend that you verify network connectivity to the internet and among the nodes before proceeding further.

1. From the *controller* node, **ping** a site on the internet:

```
# ping -c 4 openstack.org
PING openstack.org (174.143.194.225) 56(84) bytes of data.
64 bytes from 174.143.194.225: icmp_seq=1 ttl=54 time=18.3 ms
64 bytes from 174.143.194.225: icmp_seq=2 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=3 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=4 ttl=54 time=17.4 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3022ms
rtt min/avg/max/mdev = 17.489/17.715/18.346/0.364 ms
```

2. From the *controller* node, **ping** the management interface on the *network* node:

```
# ping -c 4 network
PING network (10.0.0.21) 56(84) bytes of data.
64 bytes from network (10.0.0.21): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from network (10.0.0.21): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from network (10.0.0.21): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from network (10.0.0.21): icmp_seq=4 ttl=64 time=0.202 ms

--- network ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

3. From the *controller* node, **ping** the management interface on the *compute* node:

```
# ping -c 4 compute1
PING compute1 (10.0.0.31) 56(84) bytes of data.
64 bytes from compute1 (10.0.0.31): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from compute1 (10.0.0.31): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from compute1 (10.0.0.31): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from compute1 (10.0.0.31): icmp_seq=4 ttl=64 time=0.202 ms

--- network ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

4. From the *network* node, **ping** a site on the internet:

```
# ping -c 4 openstack.org
PING openstack.org (174.143.194.225) 56(84) bytes of data.
64 bytes from 174.143.194.225: icmp_seq=1 ttl=54 time=18.3 ms
64 bytes from 174.143.194.225: icmp_seq=2 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=3 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=4 ttl=54 time=17.4 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3022ms
rtt min/avg/max/mdev = 17.489/17.715/18.346/0.364 ms
```

5. From the *network* node, **ping** the management interface on the *controller* node:

```
# ping -c 4 controller
```

```
PING controller (10.0.0.11) 56(84) bytes of data.
64 bytes from controller (10.0.0.11): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from controller (10.0.0.11): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from controller (10.0.0.11): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from controller (10.0.0.11): icmp_seq=4 ttl=64 time=0.202 ms

--- controller ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

6. From the *network* node, **ping** the instance tunnels interface on the *compute* node:

```
# ping -c 4 10.0.1.31
PING 10.0.1.31 (10.0.1.31) 56(84) bytes of data.
64 bytes from 10.0.1.31 (10.0.1.31): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from 10.0.1.31 (10.0.1.31): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from 10.0.1.31 (10.0.1.31): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from 10.0.1.31 (10.0.1.31): icmp_seq=4 ttl=64 time=0.202 ms

--- 10.0.1.31 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

7. From the *compute* node, **ping** a site on the internet:

```
# ping -c 4 openstack.org
PING openstack.org (174.143.194.225) 56(84) bytes of data.
64 bytes from 174.143.194.225: icmp_seq=1 ttl=54 time=18.3 ms
64 bytes from 174.143.194.225: icmp_seq=2 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=3 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=4 ttl=54 time=17.4 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3022ms
rtt min/avg/max/mdev = 17.489/17.715/18.346/0.364 ms
```

8. From the *compute* node, **ping** the management interface on the *controller* node:

```
# ping -c 4 controller
PING controller (10.0.0.11) 56(84) bytes of data.
64 bytes from controller (10.0.0.11): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from controller (10.0.0.11): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from controller (10.0.0.11): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from controller (10.0.0.11): icmp_seq=4 ttl=64 time=0.202 ms

--- controller ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

9. From the *compute* node, **ping** the instance tunnels interface on the *network* node:

```
# ping -c 4 10.0.1.21
PING 10.0.1.21 (10.0.1.21) 56(84) bytes of data.
64 bytes from 10.0.1.21 (10.0.1.21): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from 10.0.1.21 (10.0.1.21): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from 10.0.1.21 (10.0.1.21): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from 10.0.1.21 (10.0.1.21): icmp_seq=4 ttl=64 time=0.202 ms

--- 10.0.1.21 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
```

```
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

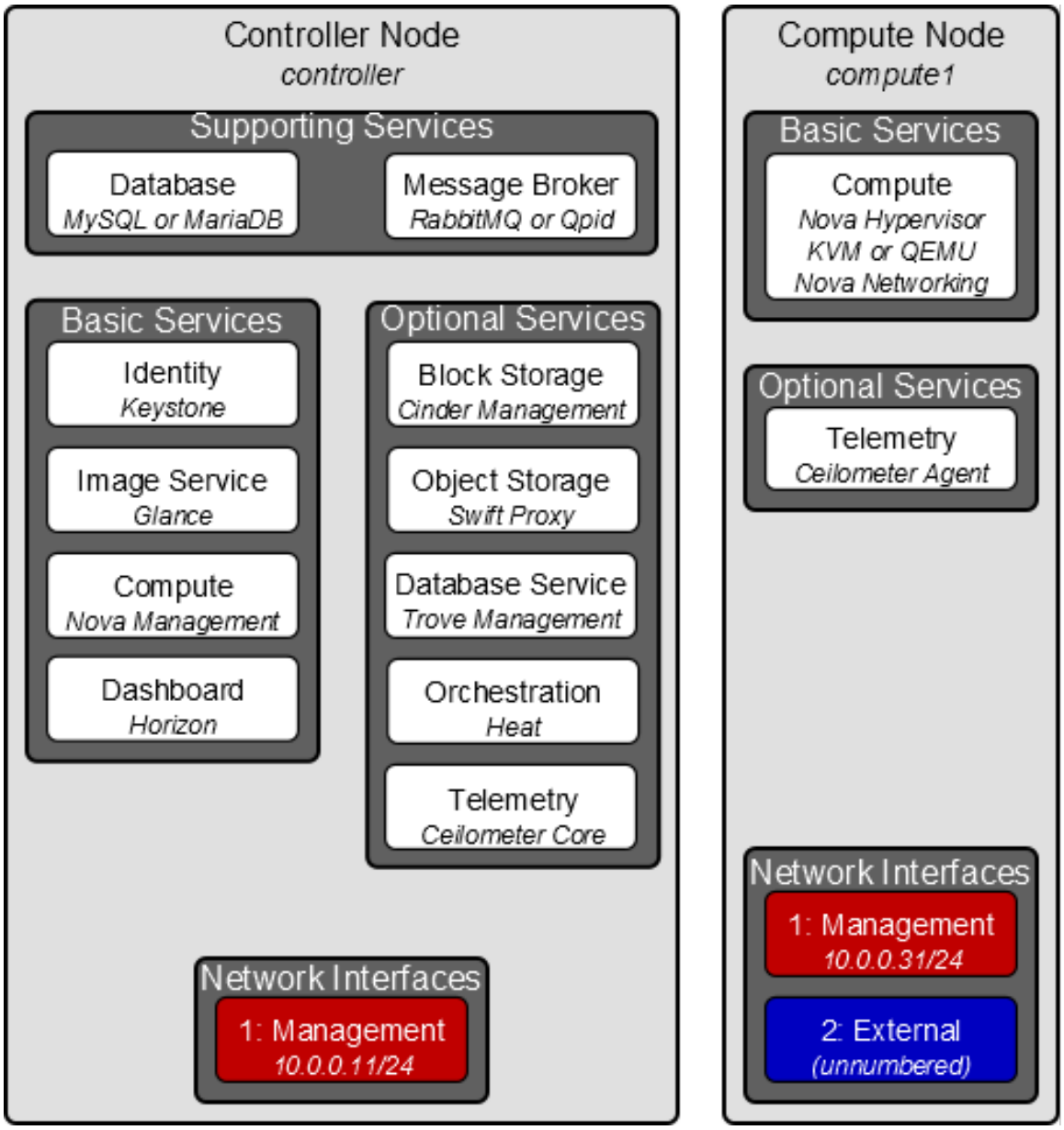
Legacy networking (nova-network)

The example architecture with legacy networking (nova-network) requires a controller node and at least one compute node. The controller node contains one network interface on the *management network*. The compute node contains one network interface on the management network and one on the *external network*.

Note

Network interface names vary by distribution. Traditionally, interfaces use "eth" followed by a sequential number. To cover all variations, this guide simply refers to the first interface as the interface with the lowest number and the second interface as the interface with the highest number.

Figure 2.2. Two-node architecture with legacy networking (nova-network)



Unless you intend to use the exact configuration provided in this example architecture, you must modify the networks in this procedure to match your environment. Also, each node must resolve the other nodes by name in addition to IP address. For example, the *`controller`* name must resolve to `10.0.0.11`, the IP address of the management interface on the controller node.

Warning

Reconfiguring network interfaces will interrupt network connectivity. We recommend using a local terminal session for these procedures.

Controller node

To configure networking:

- Configure the first interface as the management interface:

  IP address: 10.0.0.11

  Network mask: 255.255.255.0 (or /24)

  Default gateway: 10.0.0.1

To configure name resolution:

- Edit the `/etc/hosts` file to contain the following:

```
# controller
10.0.0.11       controller

# compute1
10.0.0.31       compute1
```

Compute node

To configure networking:

1. Configure the first interface as the management interface:

   IP address: 10.0.0.31

   Network mask: 255.255.255.0 (or /24)

   Default gateway: 10.0.0.1

   ### Note

   Additional compute nodes should use 10.0.0.32, 10.0.0.33, and so on.

2. The external interface uses a special configuration without an IP address assigned to it. Configure the second interface as the external interface:

   Replace *`INTERFACE_NAME`* with the actual interface name. For example, *eth1* or *ens224*.

   - Edit the `/etc/sysconfig/network/ifcfg-`*`INTERFACE_NAME`* file to contain the following:

     ```
     STARTMODE='auto'
     BOOTPROTO='static'
     ```

3. Restart networking:

   ```
   # service network restart
   ```

To configure name resolution:

- Edit the `/etc/hosts` file to contain the following:

```
# compute1
10.0.0.31       compute1

# controller
10.0.0.11       controller
```

Verify connectivity

We recommend that you verify network connectivity to the internet and among the nodes before proceeding further.

1. From the *controller* node, **ping** a site on the internet:

```
# ping -c 4 openstack.org
PING openstack.org (174.143.194.225) 56(84) bytes of data.
64 bytes from 174.143.194.225: icmp_seq=1 ttl=54 time=18.3 ms
64 bytes from 174.143.194.225: icmp_seq=2 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=3 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=4 ttl=54 time=17.4 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3022ms
rtt min/avg/max/mdev = 17.489/17.715/18.346/0.364 ms
```

2. From the *controller* node, **ping** the management interface on the *compute* node:

```
# ping -c 4 compute1
PING compute1 (10.0.0.31) 56(84) bytes of data.
64 bytes from compute1 (10.0.0.31): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from compute1 (10.0.0.31): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from compute1 (10.0.0.31): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from compute1 (10.0.0.31): icmp_seq=4 ttl=64 time=0.202 ms

--- compute1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

3. From the *compute* node, **ping** a site on the internet:

```
# ping -c 4 openstack.org
PING openstack.org (174.143.194.225) 56(84) bytes of data.
64 bytes from 174.143.194.225: icmp_seq=1 ttl=54 time=18.3 ms
64 bytes from 174.143.194.225: icmp_seq=2 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=3 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=4 ttl=54 time=17.4 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3022ms
rtt min/avg/max/mdev = 17.489/17.715/18.346/0.364 ms
```

4. From the *compute* node, **ping** the management interface on the *controller* node:

```
# ping -c 4 controller
PING controller (10.0.0.11) 56(84) bytes of data.
64 bytes from controller (10.0.0.11): icmp_seq=1 ttl=64 time=0.263 ms
```

```
64 bytes from controller (10.0.0.11): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from controller (10.0.0.11): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from controller (10.0.0.11): icmp_seq=4 ttl=64 time=0.202 ms

--- controller ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

Network Time Protocol (NTP)

To synchronize services across multiple machines, you must install *NTP*. The examples in this guide configure the controller node as the reference server and any additional nodes to set their time from the controller node.

Install the `ntp` package on each system running OpenStack services:

```
# zypper install ntp
```

Set up the NTP server on your controller node so that it receives data by modifying the `ntp.conf` file and restarting the service:

```
# service ntp start
# chkconfig ntp on
```

It is advised that you configure additional nodes to synchronize their time from the controller node rather than from outside of your LAN. To do so, install the ntp daemon as above, then edit `/etc/ntp.conf` and change the `server` directive to use the controller node as internet time source.

Passwords

The various OpenStack services and the required software like the database and the messaging server have to be password protected. You use these passwords when configuring a service and then again to access the service. You have to choose a password while configuring the service and later remember to use the same password when accessing it. Optionally, you can generate random passwords with the pwgen program. Or, to create passwords one at a time, use the output of this command repeatedly:

```
$ openssl rand -hex 10
```

This guide uses the convention that *SERVICE_PASS* is the password to access the service SERVICE and *SERVICE_DBPASS* is the database password used by the service SERVICE to access the database.

The complete list of passwords you need to define in this guide are:

Table 2.1. Passwords

Password name	Description
Database password (no variable used)	Root password for the database
RABBIT_PASS	Password of user guest of RabbitMQ
KEYSTONE_DBPASS	Database password of Identity service
DEMO_PASS	Password of user `demo`
ADMIN_PASS	Password of user `admin`

Password name	Description
GLANCE_DBPASS	Database password for Image Service
GLANCE_PASS	Password of Image Service user `glance`
NOVA_DBPASS	Database password for Compute service
NOVA_PASS	Password of Compute service user `nova`
DASH_DBPASS	Database password for the dashboard
CINDER_DBPASS	Database password for the Block Storage service
CINDER_PASS	Password of Block Storage service user `cinder`
NEUTRON_DBPASS	Database password for the Networking service
NEUTRON_PASS	Password of Networking service user `neutron`
HEAT_DBPASS	Database password for the Orchestration service
HEAT_PASS	Password of Orchestration service user `heat`
CEILOMETER_DBPASS	Database password for the Telemetry service
CEILOMETER_PASS	Password of Telemetry service user `ceilometer`
TROVE_DBPASS	Database password of Database service
TROVE_PASS	Password of Database Service user `trove`

Database

Most OpenStack services require a database to store information. This guide uses a MySQL database on SUSE Linux Enterprise Server and a compatible database on openSUSE running on the controller node. This compatible database for openSUSE is MariaDB. You must install the MariaDB database on the controller node. You must install the MySQL Python library on any additional nodes that access MySQL or MariaDB.

Controller setup

For SUSE Linux Enterprise Server: On the controller node, install the MySQL client and server packages, and the Python library.

```
# zypper install mysql-client mysql python-mysql
```

For openSUSE: On the controller node, install the MariaDB client and database server packages, and the MySQL Python library.

```
# zypper install mariadb-client mariadb python-mysql
```

The MySQL configuration requires some changes to work with OpenStack.

- Edit the `/etc/my.cnf` file:

  a. Under the `[mysqld]` section, set the `bind-address` key to the management IP address of the controller node to enable access by other nodes via the management network:

  ```
  [mysqld]
  ...
  bind-address = 10.0.0.11
  ```

  b. Under the `[mysqld]` section, set the following keys to enable InnoDB, UTF-8 character set, and UTF-8 collation by default:

```
[mysqld]
...
default-storage-engine = innodb
collation-server = utf8_general_ci
init-connect = 'SET NAMES utf8'
character-set-server = utf8
```

Start the MariaDB or MySQL database server and set it to start automatically when the system boots:

```
# service mysql start
# chkconfig mysql on
```

Finally, you should set a root password for your MariaDB or MySQL database. The OpenStack programs that set up databases and tables prompt you for this password if it is set.

You must delete the anonymous users that are created when the database is first started. Otherwise, database connection problems occur when you follow the instructions in this guide. To do this, use the **mysql_secure_installation** command. Note that if **mysql_secure_installation** fails you might need to use **mysql_install_db** first:

```
# mysql_install_db
# mysql_secure_installation
```

If you have not already set a root database password, press **ENTER** when you are prompted for the password. This command presents a number of options for you to secure your database installation. Respond **yes** to all prompts unless you have a good reason to do otherwise.

Node setup

On all nodes other than the controller node, install the MySQL Python library:

```
# zypper install python-mysql
```

OpenStack packages

Distributions might release OpenStack packages as part of their distribution or through other methods because the OpenStack and distribution release times are independent of each other.

This section describes the configuration you must complete after you configure machines to install the latest OpenStack packages.

Use the Open Build Service repositories for *Icehouse* based on your openSUSE or SUSE Linux Enterprise Server version.

For openSUSE 13.1 use:

```
# zypper addrepo -f obs://Cloud:OpenStack:Icehouse/openSUSE_13.1 Icehouse
```

If you use SUSE Linux Enterprise Server 11 SP3, use:

```
# zypper addrepo -f obs://Cloud:OpenStack:Icehouse/SLE_11_SP3 Icehouse
```

The openstack-utils package contains utility programs that make installation and configuration easier. These programs are used throughout this guide. Install openstack-utils. This verifies that you can access the Open Build Service repository:

```
# zypper install openstack-utils
```

Warning

The openstack-config program in the openstack-utils package uses crudini to manipulate configuration files. However, crudini version 0.3 does not support multi valued options. See https://bugs.launchpad.net/openstack-manuals/+bug/1269271. As a work around, you must manually set any multi valued options or the new value overwrites the previous value instead of creating a new option.

Upgrade your system packages:

```
# zypper refresh
# zypper update
```

If the upgrade included a new kernel package, reboot the system to ensure the new kernel is running:

```
# reboot
```

Messaging server

OpenStack uses a *message broker* to coordinate operations and status information among services. The message broker service typically runs on the controller node. OpenStack supports several message brokers including RabbitMQ, Qpid, and ZeroMQ. However, most distributions that package OpenStack support a particular message broker. This guide covers the message broker supported by each distribution. If you prefer to implement a different message broker, consult the documentation associated with it.

- RabbitMQ
- Qpid
- ZeroMQ

To install the message broker service

- SUSE Linux Enterprise Server (SLES) and openSUSE use RabbitMQ.

  ```
  # zypper install rabbitmq-server
  ```

To configure the message broker service

1. Start the message broker service:

   ```
   # service rabbitmq-server start
   ```

2. The message broker creates a default account that uses `guest` for the username and password. To simplify installation of your test environment, we recommend that you use this account, but change the password for it.

Run the following command:

Replace *RABBIT_PASS* with a suitable password.

```
# rabbitmqctl change_password guest RABBIT_PASS
```

You must configure the `rabbit_password` key in the configuration file for each OpenStack service that uses the message broker.

Note

For production environments, you should create a unique account with suitable password. For more information on securing the message broker, see the documentation.

If you decide to create a unique account with suitable password for your test environment, you must configure the `rabbit_userid` and `rabbit_password` keys in the configuration file of each OpenStack service that uses the message broker.

To finalize installation

- Configure the message broker service to start when the system boots:

```
# chkconfig rabbitmq-server on
```

Congratulations, now you are ready to install OpenStack services!

3. Configure the Identity Service

Table of Contents

Identity Service concepts

The *Identity Service* performs the following functions:

- User management. Tracks users and their permissions.
- *Service catalog*. Provides a catalog of available services with their API endpoints.

To understand the Identity Service, you must understand the following concepts:

User	Digital representation of a person, system, or service who uses OpenStack cloud services. The Identity Service validates that incoming requests are made by the user who claims to be making the call. Users have a login and may be assigned tokens to access resources. Users can be directly assigned to a particular tenant and behave as if they are contained in that tenant.
Credentials	Data that is known only by a user that proves who they are. In the Identity Service, examples are: User name and password, user name and API key, or an authentication token provided by the Identity Service.
Authentication	The act of confirming the identity of a user. The Identity Service confirms an incoming request by validating a set of credentials supplied by the user. These credentials are initially a user name and password or a user name and API key. In response to these credentials, the Identity Service issues an authentication token to the user, which the user provides in subsequent requests.
Token	An arbitrary bit of text that is used to access resources. Each token has a scope which describes which resources are accessible with it. A token may be revoked at any time and is valid for a finite duration. While the Identity Service supports token-based authentication in this release, the intention is for it to support additional protocols in the future. The

intent is for it to be an integration service foremost, and not aspire to be a full-fledged identity store and management solution.

Tenant — A container used to group or isolate resources and/or identity objects. Depending on the service operator, a tenant may map to a customer, account, organization, or project.

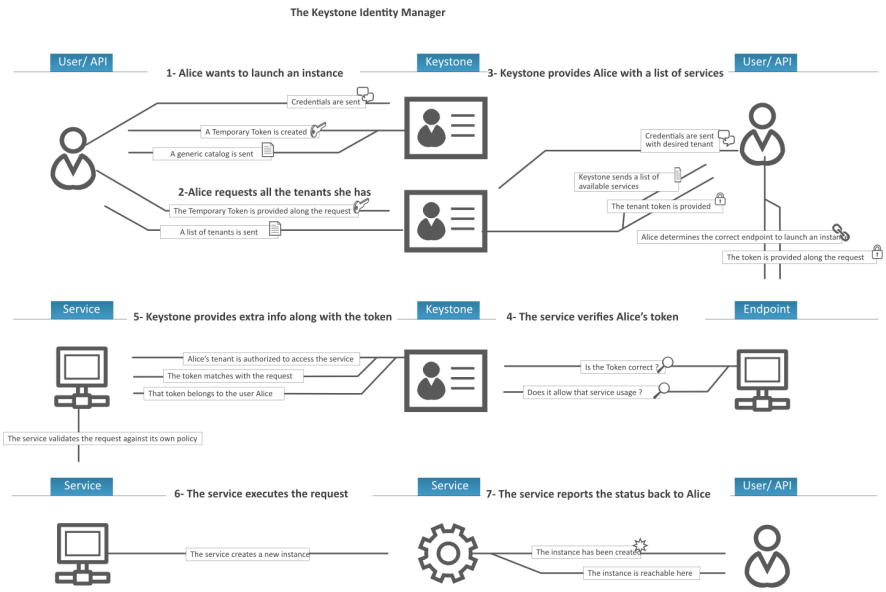
Service — An OpenStack service, such as Compute (Nova), Object Storage (Swift), or Image Service (Glance). Provides one or more endpoints through which users can access resources and perform operations.

Endpoint — A network-accessible address, usually described by a URL, from where you access a service. If using an extension for templates, you can create an endpoint template, which represents the templates of all the consumable services that are available across the regions.

Role — A personality that a user assumes that enables them to perform a specific set of operations. A role includes a set of rights and privileges. A user assuming that role inherits those rights and privileges.

In the Identity Service, a token that is issued to a user includes the list of roles that user has. Services that are being called by that user determine how they interpret the set of roles a user has and to which operations or resources each role grants access.

The following diagram shows the Identity Service process flow:



Install the Identity Service

1. Install the OpenStack Identity Service on the controller node, together with python-keystoneclient (which is a dependency):

```
# zypper install openstack-keystone python-keystoneclient openstack-utils
```

2. The Identity Service uses a database to store information. Specify the location of the database in the configuration file. In this guide, we use a MySQL database on the controller node with the username `keystone`. Replace *KEYSTONE_DBPASS* with a suitable password for the database user.

```
# openstack-config --set /etc/keystone/keystone.conf \
   database connection mysql://keystone:KEYSTONE_DBPASS@controller/
keystone
```

3. Use the password that you set previously to log in as root. Create a `keystone` database user:

```
$ mysql -u root -p
mysql> CREATE DATABASE keystone;
mysql> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost' \
  IDENTIFIED BY 'KEYSTONE_DBPASS';
mysql> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' \
  IDENTIFIED BY 'KEYSTONE_DBPASS';
mysql> exit
```

4. Define an authorization token to use as a shared secret between the Identity Service and other OpenStack services. Use **openssl** to generate a random token and store it in the configuration file:

```
# ADMIN_TOKEN=$(openssl rand -hex 10)
# echo $ADMIN_TOKEN
# openstack-config --set /etc/keystone/keystone.conf DEFAULT \
   admin_token $ADMIN_TOKEN
```

For SUSE Linux Enterprise use instead as first command:

```
# ADMIN_TOKEN=$(openssl rand 10|hexdump  -e '1/1 "%.2x"')
```

5. By default, Keystone uses PKI tokens. Create the signing keys and certificates and restrict access to the generated data:

```
# keystone-manage pki_setup --keystone-user keystone --keystone-group
 keystone
# chown -R keystone:keystone /etc/keystone/ssl
# chmod -R o-rwx /etc/keystone/ssl
```

6. Start the Identity Service and enable it to start when the system boots:

```
# service openstack-keystone start
# chkconfig openstack-keystone on
```

7. By default, the Identity Service stores expired tokens in the database indefinitely. While potentially useful for auditing in production environments, the accumulation of expired tokens will considerably increase database size and may decrease service performance, particularly in test environments with limited resources. We recommend configuring a periodic task using `cron` to purge expired tokens hourly.

- Run the following command to purge expired tokens every hour and log the output to `/var/log/keystone/keystone-tokenflush.log`:

```
# (crontab -l 2>&1 | grep -q token_flush) || \
echo '@hourly /usr/bin/keystone-manage token_flush >/var/log/keystone/
keystone-tokenflush.log 2>&1' >> /var/spool/cron/tabs/root
```

Define users, tenants, and roles

After you install the Identity Service, set up *users*, *tenants*, and *roles* to authenticate against. These are used to allow access to services and *endpoints*, described in the next section.

Typically, you would indicate a user and password to authenticate with the Identity Service. At this point, however, you have not created any users, so you have to use the authorization token created in an earlier step, see the section called "Install the Identity Service" [24] for further details. You can pass this with the `--os-token` option to the **keystone** command or set the `OS_SERVICE_TOKEN` environment variable. Set `OS_SERVICE_TOKEN`, as well as `OS_SERVICE_ENDPOINT` to specify where the Identity Service is running. Replace ***ADMIN_TOKEN*** with your authorization token.

```
$ export OS_SERVICE_TOKEN=ADMIN_TOKEN
$ export OS_SERVICE_ENDPOINT=http://controller:35357/v2.0
```

Create an administrative user

Follow these steps to create an administrative user, role, and tenant. You will use this account for administrative interaction with the OpenStack cloud.

By default, the Identity Service creates a special `_member_` role. The OpenStack dashboard automatically grants access to users with this role. You will give the `admin` user access to this role in addition to the `admin` role.

Note

Any role that you create must map to roles specified in the `policy.json` file included with each OpenStack service. The default policy file for most services grants administrative access to the `admin` role.

1. Create the `admin` user:

```
$ keystone user-create --name=admin --pass=ADMIN_PASS --email=ADMIN_EMAIL
```

   Replace *ADMIN_PASS* with a secure password and replace *ADMIN_EMAIL* with an email address to associate with the account.

2. Create the `admin` role:

```
$ keystone role-create --name=admin
```

3. Create the `admin` tenant:

```
$ keystone tenant-create --name=admin --description="Admin Tenant"
```

4. You must now link the `admin` user, `admin` role, and `admin` tenant together using the `user-role-add` option:

```
$ keystone user-role-add --user=admin --tenant=admin --role=admin
```

5. Link the `admin` user, `_member_` role, and `admin` tenant:

```
$ keystone user-role-add --user=admin --role=_member_ --tenant=admin
```

Create a normal user

Follow these steps to create a normal user and tenant, and link them to the special `_member_` role. You will use this account for daily non-administrative interaction with the OpenStack cloud. You can also repeat this procedure to create additional cloud users with different usernames and passwords. Skip the tenant creation step when creating these users.

1. Create the `demo` user:

```
$ keystone user-create --name=demo --pass=DEMO_PASS --email=DEMO_EMAIL
```

Replace *DEMO_PASS* with a secure password and replace *DEMO_EMAIL* with an email address to associate with the account.

2. Create the `demo` tenant:

```
$ keystone tenant-create --name=demo --description="Demo Tenant"
```

Note

Do not repeat this step when adding additional users.

3. Link the `demo` user, `_member_` role, and `demo` tenant:

```
$ keystone user-role-add --user=demo --role=_member_ --tenant=demo
```

Create a `service` tenant

OpenStack services also require a username, tenant, and role to access other OpenStack services. In a basic installation, OpenStack services typically share a single tenant named `service`.

You will create additional usernames and roles under this tenant as you install and configure each service.

- Create the `service` tenant:

```
$ keystone tenant-create --name=service --description="Service Tenant"
```

Define services and API endpoints

So that the Identity Service can track which OpenStack services are installed and where they are located on the network, you must register each service in your OpenStack installation. To register a service, run these commands:

- **keystone service-create**. Describes the service.
- **keystone endpoint-create**. Associates *API endpoints* with the service.

You must also register the Identity Service itself. Use the `OS_SERVICE_TOKEN` environment variable, as set previously, for authentication.

1. Create a service entry for the Identity Service:

```
$ keystone service-create --name=keystone --type=identity \
  --description="OpenStack Identity"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description | OpenStack Identity               |
| id          | 15c11a23667e427e91bc31335b45f4bd |
| name        | keystone                         |
| type        | identity                         |
+-------------+----------------------------------+
```

   The service ID is randomly generated and is different from the one shown here.

2. Specify an API endpoint for the Identity Service by using the returned service ID. When you specify an endpoint, you provide URLs for the public API, internal API, and admin API. In this guide, the `controller` host name is used. Note that the Identity Service uses a different port for the admin API.

```
$ keystone endpoint-create \
  --service-id=$(keystone service-list | awk '/ identity / {print $2}') \
  --publicurl=http://controller:5000/v2.0 \
  --internalurl=http://controller:5000/v2.0 \
  --adminurl=http://controller:35357/v2.0
+-------------+----------------------------------+
|   Property  |             Value                |
+-------------+----------------------------------+
| adminurl    | http://controller:35357/v2.0     |
| id          | 11f9c625a3b94a3f8e66bf4e5de2679f |
| internalurl | http://controller:5000/v2.0      |
| publicurl   | http://controller:5000/v2.0      |
| region      | regionOne                        |
| service_id  | 15c11a23667e427e91bc31335b45f4bd |
+-------------+----------------------------------+
```

Note

You will need to create an additional endpoint for each service added to your OpenStack environment. The sections of this guide associated with the installation of each service include the endpoint creation step specific to the service.

Verify the Identity Service installation

1. To verify that the Identity Service is installed and configured correctly, clear the values in the `OS_SERVICE_TOKEN` and `OS_SERVICE_ENDPOINT` environment variables:

```
$ unset OS_SERVICE_TOKEN OS_SERVICE_ENDPOINT
```

These variables, which were used to bootstrap the administrative user and register the Identity Service, are no longer needed.

2. You can now use regular user name-based authentication.

   Request a authentication token by using the `admin` user and the password you chose for that user:

   ```
   $ keystone --os-username=admin --os-password=ADMIN_PASS \
     --os-auth-url=http://controller:35357/v2.0 token-get
   ```

   In response, you receive a token paired with your user ID. This verifies that the Identity Service is running on the expected endpoint and that your user account is established with the expected credentials.

3. Verify that authorization behaves as expected. To do so, request authorization on a tenant:

   ```
   $ keystone --os-username=admin --os-password=ADMIN_PASS \
     --os-tenant-name=admin --os-auth-url=http://controller:35357/v2.0 \
     token-get
   ```

   In response, you receive a token that includes the ID of the tenant that you specified. This verifies that your user account has an explicitly defined role on the specified tenant and the tenant exists as expected.

4. You can also set your `--os-*` variables in your environment to simplify command-line usage. Set up a `admin-openrc.sh` file with the admin credentials and admin endpoint:

   ```
   export OS_USERNAME=admin
   export OS_PASSWORD=ADMIN_PASS
   export OS_TENANT_NAME=admin
   export OS_AUTH_URL=http://controller:35357/v2.0
   ```

5. Source this file to read in the environment variables:

   ```
   $ source admin-openrc.sh
   ```

6. Verify that your `admin-openrc.sh` file is configured correctly. Run the same command without the `--os-*` arguments:

   ```
   $ keystone token-get
   ```

   The command returns a token and the ID of the specified tenant. This verifies that you have configured your environment variables correctly.

7. Verify that your admin account has authorization to perform administrative commands:

   ```
   $ keystone user-list
   +----------------------------------+-------+---------+-------------------+
   |                id                |  name | enabled |    email          |
   +----------------------------------+-------+---------+-------------------+
   | afea5bde3be9413dbd60e479fddf9228 | admin |   True  | admin@example.com |
   | 32aca1f9a47540c29d6988091f76c934 |  demo |   True  | demo@example.com  |
   +----------------------------------+-------+---------+-------------------+
   ```

```
$ keystone user-role-list --user admin --tenant admin
+----------------------------------+----------
+----------------------------------+----------------------------------+
|                id                |   name   |             user_id
       |            tenant_id             |
+----------------------------------+----------
+----------------------------------+----------------------------------+
| 9fe2ff9ee4384b1894a90878d3e92bab | _member_ |
 afea5bde3be9413dbd60e479fddf9228 | e519b772cb43474582fa303da62559e5 |
| 5d3b60b66f1f438b80eaae41a77b5951 |  admin   |
 afea5bde3be9413dbd60e479fddf9228 | e519b772cb43474582fa303da62559e5 |
+----------------------------------+----------
+----------------------------------+----------------------------------+
```

Seeing that the `id` in the output from the **keystone user-list** command matches the `user_id` in the **keystone user-role-list** command, and that the admin role is listed for that user, for the related tenant, this verifies that your user account has the `admin` role, which matches the role used in the Identity Service `policy.json` file.

Note

As long as you define your credentials and the Identity Service endpoint through the command line or environment variables, you can run all OpenStack client commands from any machine. For details, see Chapter 4, “Install and configure the OpenStack clients” [30].

4. Install and configure the OpenStack clients

Table of Contents

The following sections contain information about working with the OpenStack clients. Recall: in the previous section, you used the **keystone** client.

You must install the client tools to complete the rest of the installation.

Configure the clients on your desktop rather than on the server so that you have a similar experience to your users.

Overview

You can use the OpenStack command-line clients to run simple commands that make API calls. You can run these commands from the command line or in scripts to automate tasks. If you provide OpenStack credentials, you can run these commands on any computer.

Internally, each client command runs cURL commands that embed API requests. The OpenStack APIs are RESTful APIs that use the HTTP protocol, including methods, URIs, media types, and response codes.

These open-source Python clients run on Linux or Mac OS X systems and are easy to learn and use. Each OpenStack service has its own command-line client. On some client commands, you can specify a **debug** parameter to show the underlying API request for the command. This is a good way to become familiar with the OpenStack API calls.

The following table lists the command-line client for each OpenStack service with its package name and description.

Table 4.1. OpenStack services and clients

Service	Client	Package	Description
Block Storage	**cinder**	python-cinderclient	Create and manage volumes.
Compute	**nova**	python-novaclient	Create and manage images, instances, and flavors.
Database Service	**trove**	python-troveclient	Create and manage databases.
Identity	**keystone**	python-keystoneclient	Create and manage users, tenants, roles, endpoints, and credentials.
Image Service	**glance**	python-glanceclient	Create and manage images.

Service	Client	Package	Description
Networking	**neutron**	python-neutronclient	Configure networks for guest servers. This client was previously called **quantum**.
Object Storage	**swift**	python-swiftclient	Gather statistics, list items, update metadata, and upload, download, and delete files stored by the Object Storage service. Gain access to an Object Storage installation for ad hoc processing.
Orchestration	**heat**	python-heatclient	Launch stacks from templates, view details of running stacks including events and resources, and update and delete stacks.
Telemetry	**ceilometer**	python-ceilometerclient	Create and collect measurements across OpenStack.

An OpenStack **common** client is in development.

Install the OpenStack command-line clients

Install the prerequisite software and the Python package for each OpenStack client.

Install the prerequisite software

The following table lists the software that you need to have to run the command-line clients, and provides installation instructions as needed.

Table 4.2. Prerequisite software

Prerequisite	Description
Python 2.6 or later	Currently, the clients do not support Python 3.
setuptools package	Installed by default on Mac OS X. Many Linux distributions provide packages to make setuptools easy to install. Search your package manager for setuptools to find an installation package. If you cannot find one, download the setuptools package directly from http://pypi.python.org/pypi/setuptools. The recommended way to install setuptools on Microsoft Windows is to follow the documentation provided on the setuptools website. Another option is to use the unofficial binary installer maintained by Christoph Gohlke (http://www.lfd.uci.edu/~gohlke/pythonlibs/#setuptools).

pip package	To install the clients on a Linux, Mac OS X, or Microsoft Windows system, use pip. It is easy to use, ensures that you get the latest version of the clients from the Python Package Index, and lets you update or remove the packages later on.
	
	Install pip through the package manager for your system:
	
	MacOS.
	
	`# easy_install pip`
	
	Microsoft Windows. Ensure that the `C:\Python27\Scripts` directory is defined in the PATH environment variable, and use the **easy_install** command from the setuptools package:
	
	`C:\>easy_install pip`
	
	Another option is to use the unofficial binary installer provided by Christoph Gohlke (http://www.lfd.uci.edu/~gohlke/pythonlibs/#pip).
	
	Ubuntu 12.04/14.04. A packaged version enables you to use dpkg or aptitude to install the python-novaclient:

Prerequisite	Description

	`# aptitude install python-novaclient`
	
	Ubuntu and Debian.
	
	`# aptitude install python-pip`
	
	Red Hat Enterprise Linux, CentOS, or Fedora. A packaged version available in RDO enables you to use yum to install the clients, or you can install pip and use it to manage client installation:
	
	`# yum install python-pip`
	
	openSUSE 12.2 and earlier. A packaged version available in the Open Build Service enables you to use rpm or zypper to install the clients, or you can install pip and use it to manage client installation:
	
	`# zypper install python-pip`
	
	openSUSE 12.3 and later. A packaged version enables you to use rpm or zypper to install the clients. See the section called "Install the clients" [32]

Install the clients

When following the instructions in this section, replace *PROJECT* with the lowercase name of the client to install, such as **nova**. Repeat for each client. The following values are valid:

- `ceilometer` - Telemetry API
- `cinder` - Block Storage API and extensions
- `glance` - Image Service API
- `heat` - Orchestration API
- `keystone` - Identity service API and extensions
- `neutron` - Networking API
- `nova` - Compute API and extensions
- `swift` - Object Storage API
- `trove` - Database Service API

The following example shows the command for installing the nova client with `pip`.

```
# pip install python-novaclient
```

Installing with pip

Use pip to install the OpenStack clients on a Linux, Mac OS X, or Microsoft Windows system. It is easy to use and ensures that you get the latest version of the client from the Python Package Index. Also, pip enables you to update or remove a package.

Install each client separately by using the following command:

- For Mac OS X or Linux:

```
# pip install python-PROJECTclient
```

- For Microsoft Windows:

```
C:\>pip install python-PROJECTclient
```

Installing from packages

RDO and openSUSE have client packages that can be installed without `pip`.

On Red Hat Enterprise Linux, CentOS, or Fedora, use **yum** to install the clients from the packaged versions available in RDO:

```
# yum install python-PROJECTclient
```

For openSUSE, use rpm or zypper to install the clients from the packaged versions available in the Open Build Service:

```
# zypper install python-PROJECT
```

Upgrade or remove clients

To upgrade a client, add the `--upgrade` option to the **pip install** command:

```
# pip install --upgrade python-PROJECTclient
```

To remove the a client, run the **pip uninstall** command:

```
# pip uninstall python-PROJECTclient
```

Set environment variables using the OpenStack RC file

To set the required environment variables for the OpenStack command-line clients, you must create an environment file called an OpenStack rc file, or `openrc.sh` file. This project-specific environment file contains the credentials that all OpenStack services use.

When you source the file, environment variables are set for your current shell. The variables enable the OpenStack client commands to communicate with the OpenStack services that run in the cloud.

Note

Defining environment variables using an environment file is not a common practice on Microsoft Windows. Environment variables are usually defined in the **Advanced** tab of the System Properties dialog box.

Create and source the OpenStack RC file

1. In a text editor, create a file named *PROJECT*`-openrc.sh` file and add the following authentication information:

   The following example shows the information for a project called `admin`, where the OS username is also `admin`, and the identity host is located at `controller`.

```
export OS_USERNAME=admin
export OS_PASSWORD=ADMIN_PASS
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://controller:35357/v2.0
```

2. On any shell from which you want to run OpenStack commands, source the *PROJECT*-openrc.sh file for the respective project. In this example, you source the admin-openrc.sh file for the *admin* project:

```
$ source admin-openrc.sh
```

Override environment variable values

When you run OpenStack client commands, you can override some environment variable settings by using the options that are listed at the end of the **help** output of the various client commands. For example, you can override the OS_PASSWORD setting in the *PROJECT*-openrc.sh file by specifying a password on a **keystone** command, as follows:

```
$ keystone --os-password PASSWORD service-list
```

Where *PASSWORD* is your password.

Create openrc.sh files

As explained in the section called "Create and source the OpenStack RC file" [33], use the credentials from the section called "Define users, tenants, and roles" [25] and create the following *PROJECT*-openrc.sh files:

- admin-openrc.sh for the administrative user
- demo-openrc.sh for the normal user:

```
export OS_USERNAME=demo
export OS_PASSWORD=DEMO_PASS
export OS_TENANT_NAME=demo
export OS_AUTH_URL=http://controller:35357/v2.0
```

5. Configure the Image Service

Table of Contents

The OpenStack Image Service enables users to discover, register, and retrieve virtual machine images. Also known as the glance project, the Image Service offers a *REST* API that enables you to query virtual machine image metadata and retrieve an actual image. You can store virtual machine images made available through the Image Service in a variety of locations from simple file systems to object-storage systems like OpenStack Object Storage.

Important

For simplicity, this guide configures the Image Service to use the `file` back end. This means that images uploaded to the Image Service are stored in a directory on the same system that hosts the service. By default, this directory is `/var/lib/glance/images/`.

Before you proceed, ensure that the system has sufficient space available in this directory to store virtual machine images and snapshots. At an absolute minimum, several gigabytes of space should be available for use by the Image Service in a proof of concept deployment. To see requirements for other back ends, see *Configuration Reference*.

Image Service overview

The Image Service includes the following components:

- `glance-api`. Accepts Image API calls for image discovery, retrieval, and storage.
- `glance-registry`. Stores, processes, and retrieves metadata about images. Metadata includes items such as size and type.

Security note

The registry is a private internal service meant only for use by the Image Service itself. Do not expose it to users.

- Database. Stores image metadata. You can choose your database depending on your preference. Most deployments use MySQL or SQlite.
- Storage repository for image files. The Image Service supports a variety of repositories including normal file systems, Object Storage, RADOS block devices, HTTP, and Amazon S3. Some types of repositories support only read-only usage.

A number of periodic processes run on the Image Service to support caching. Replication services ensures consistency and availability through the cluster. Other periodic processes include auditors, updaters, and reapers.

As shown in Figure 1.1, "Conceptual architecture" [2], the Image Service is central to the overall IaaS picture. It accepts API requests for images or image metadata from end users or Compute components and can store its disk files in the Object Storage Service.

Install the Image Service

The OpenStack Image Service acts as a registry for virtual disk images. Users can add new images or take a snapshot of an image from an existing server for immediate storage. Use snapshots for back up and as templates to launch new servers. You can store registered images in Object Storage or in other locations. For example, you can store images in simple file systems or external web servers.

Note

This procedure assumes you set the appropriate environment variables to your credentials as described in the section called "Verify the Identity Service installation" [27].

1. Install the Image Service on the controller node:

```
# zypper install openstack-glance python-glanceclient
```

2. The Image Service stores information about images in a database. The examples in this guide use the MySQL database that is used by other OpenStack services.

   Configure the location of the database. The Image Service provides the `glance-api` and `glance-registry` services, each with its own configuration file. You must update both configuration files throughout this section. Replace *GLANCE_DBPASS* with your Image Service database password.

```
# openstack-config --set /etc/glance/glance-api.conf database \
  connection mysql://glance:GLANCE_DBPASS@controller/glance
# openstack-config --set /etc/glance/glance-registry.conf database \
  connection mysql://glance:GLANCE_DBPASS@controller/glance
```

3. Configure the Image Service to use the message broker:

   Replace *RABBIT_PASS* with the password you chose for the `guest` account in RabbitMQ.

```
# openstack-config --set /etc/glance/glance-api.conf DEFAULT \
  rpc_backend rabbit
# openstack-config --set /etc/glance/glance-api.conf DEFAULT \
  rabbit_host controller
# openstack-config --set /etc/glance/glance-api.conf DEFAULT \
  rabbit_password RABBIT_PASS
```

4. Use the password you created to log in as root and create a `glance` database user:

```
$ mysql -u root -p
mysql> CREATE DATABASE glance;
```

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

5. Create a `glance` user that the Image Service can use to authenticate with the Identity service. Choose a password and specify an email address for the `glance` user. Use the `service` tenant and give the user the `admin` role:

```
$ keystone user-create --name=glance --pass=GLANCE_PASS \
   --email=glance@example.com
$ keystone user-role-add --user=glance --tenant=service --role=admin
```

6. Configure the Image Service to use the Identity Service for authentication.

   Run the following commands and replace *GLANCE_PASS* with the password you chose for the `glance` user in the Identity Service:

```
# openstack-config --set /etc/glance/glance-api.conf keystone_authtoken \
  auth_uri http://controller:5000
# openstack-config --set /etc/glance/glance-api.conf keystone_authtoken \
  auth_host controller
# 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 service
# 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 GLANCE_PASS
# openstack-config --set /etc/glance/glance-api.conf paste_deploy \
  flavor keystone
# openstack-config --set /etc/glance/glance-registry.conf
 keystone_authtoken \
  auth_uri http://controller:5000
# openstack-config --set /etc/glance/glance-registry.conf
 keystone_authtoken \
  auth_host controller
# 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 service
# 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 GLANCE_PASS
# openstack-config --set /etc/glance/glance-registry.conf paste_deploy \
  flavor keystone
```

7. Register the Image Service with the Identity service so that other OpenStack services can locate it. Register the service and create the endpoint:

```
$ keystone service-create --name=glance --type=image \
  --description="OpenStack Image Service"
$ keystone endpoint-create \
  --service-id=$(keystone service-list | awk '/ image / {print $2}') \
  --publicurl=http://controller:9292 \
  --internalurl=http://controller:9292 \
  --adminurl=http://controller:9292
```

8. Start the `glance-api` and `glance-registry` services and configure them to start when the system boots:

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

Verify the Image Service installation

To test the Image Service installation, download at least one virtual machine image that is known to work with OpenStack. For example, CirrOS is a small test image that is often used for testing OpenStack deployments (CirrOS downloads). This walk through uses the 64-bit CirrOS QCOW2 image.

For more information about how to download and build images, see *OpenStack Virtual Machine Image Guide*. For information about how to manage images, see the *OpenStack User Guide*.

1. Download the image into a dedicated directory using **wget** or **curl**:

```
$ mkdir images
$ cd images/
$ wget http://cdn.download.cirros-cloud.net/0.3.2/cirros-0.3.2-x86_64-
disk.img
```

2. Upload the image to the Image Service:

```
$ glance image-create --name=IMAGELABEL --disk-format=FILEFORMAT \
  --container-format=CONTAINERFORMAT --is-public=ACCESSVALUE < IMAGEFILE
```

Where:

IMAGELABEL Arbitrary label. The name by which users refer to the image.

FILEFORMAT Specifies the format of the image file. Valid formats include `qcow2`, `raw`, `vhd`, `vmdk`, `vdi`, `iso`, `aki`, `ari`, and `ami`.

You can verify the format using the **file** command:

```
$ file cirros-0.3.2-x86_64-disk.img
cirros-0.3.2-x86_64-disk.img: QEMU QCOW Image (v2),
 41126400 bytes
```

CONTAINERFORMAT Specifies the container format. Valid formats include: `bare`, `ovf`, `aki`, `ari` and `ami`.

Specify `bare` to indicate that the image file is not in a file format that contains metadata about the virtual machine. Although this field is currently required, it is not actually used by any of the OpenStack services and has no effect on system behavior. Because the value is not used anywhere, it is safe to always specify `bare` as the container format.

ACCESSVALUE Specifies image access:

- true - All users can view and use the image.
- false - Only administrators can view and use the image.

IMAGEFILE Specifies the name of your downloaded image file.

For example:

```
$ source admin-openrc.sh
$ glance image-create --name "cirros-0.3.2-x86_64" --disk-format qcow2 \
  --container-format bare --is-public True --progress < cirros-0.3.2-
x86_64-disk.img
+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
| checksum         | 64d7c1cd2b6f60c92c14662941cb7913     |
| container_format | bare                                 |
| created_at       | 2014-04-08T18:59:18                  |
| deleted          | False                                |
| deleted_at       | None                                 |
| disk_format      | qcow2                                |
| id               | acafc7c0-40aa-4026-9673-b879898e1fc2 |
| is_public        | True                                 |
| min_disk         | 0                                    |
| min_ram          | 0                                    |
| name             | cirros-0.3.2-x86_64                  |
| owner            | efa984b0a914450e9a47788ad330699d     |
| protected        | False                                |
| size             | 13167616                             |
| status           | active                               |
| updated_at       | 2014-01-08T18:59:18                  |
+------------------+--------------------------------------+
```

Note

Because the returned image ID is generated dynamically, your deployment generates a different ID than the one shown in this example.

3. Confirm that the image was uploaded and display its attributes:

```
$ glance image-list
```

```
+--------------------------------------+---------------------
+-------------+-----------------+----------+--------+
| ID                                   | Name                | Disk Format
 | Container Format | Size     | Status |
+--------------------------------------+---------------------
+-------------+-----------------+----------+--------+
```

```
| acafc7c0-40aa-4026-9673-b879898e1fc2 | cirros-0.3.2-x86_64 | qcow2
 | bare             | 13167616 | active |
+--------------------------------------+--------------------
+-------------+-----------------+----------+--------+
```

Alternatively, the upload to the Image Service can be done without having to use local disk space to store the file, by use of the *--copy-from* parameter.

For example:

```
$ glance image-create --name="cirros-0.3.2-x86_64" --disk-format=qcow2 \
  --container-format=bare --is-public=true \
  --copy-from http://cdn.download.cirros-cloud.net/0.3.2/cirros-0.3.2-x86_64-
disk.img
+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
| checksum         | 64d7c1cd2b6f60c92c14662941cb7913     |
| container_format | bare                                 |
| created_at       | 2014-04-08T06:13:18                  |
| deleted          | False                                |
| disk_format      | qcow2                                |
| id               | 3cce1e32-0971-4958-9719-1f92064d4f54 |
| is_public        | True                                 |
| min_disk         | 0                                    |
| min_ram          | 0                                    |
| name             | cirros-0.3.2-x86_64                  |
| owner            | efa984b0a914450e9a47788ad330699d     |
| protected        | False                                |
| size             | 13167616                             |
| status           | active                               |
| updated_at       | 2014-04-08T06:13:20                  |
+------------------+--------------------------------------+
```

6. Configure Compute services

Table of Contents

Compute service

The Compute service is a cloud computing fabric controller, which is the main part of an IaaS system. Use it to host and manage cloud computing systems. The main modules are implemented in Python.

Compute interacts with the Identity Service for authentication, Image Service for images, and the Dashboard for the user and administrative interface. Access to images is limited by project and by user; quotas are limited per project (for example, the number of instances). The Compute service scales horizontally on standard hardware, and downloads images to launch instances as required.

The Compute service is made up of the following functional areas and their underlying components:

API

- `nova-api` service. Accepts and responds to end user compute API calls. Supports the OpenStack Compute API, the Amazon EC2 API, and a special Admin API for privileged users to perform administrative actions. Also, initiates most orchestration activities, such as running an instance, and enforces some policies.
- `nova-api-metadata` service. Accepts metadata requests from instances. The `nova-api-metadata` service is generally only used when you run in multi-host mode with `nova-network` installations. For details, see Metadata service in the *Cloud Administrator Guide*.

  On Debian systems, it is included in the `nova-api` package, and can be selected through debconf.

Compute core

- `nova-compute` process. A worker daemon that creates and terminates virtual machine instances through hypervisor APIs. For example, XenAPI for XenServer/XCP, libvirt for KVM or QEMU, VMwareAPI for VMware, and so on. The process by which it does so is fairly complex but the basics are simple: Accept actions from the queue and perform a series of system commands, like launching a KVM instance, to carry them out while updating state in the database.
- `nova-scheduler` process. Conceptually the simplest piece of code in Compute. Takes a virtual machine instance request from the queue and determines on which compute server host it should run.

- `nova-conductor` module. Mediates interactions between `nova-compute` and the database. Aims to eliminate direct accesses to the cloud database made by `nova-compute`. The `nova-conductor` module scales horizontally. However, do not deploy it on any nodes where `nova-compute` runs. For more information, see A new Nova service: nova-conductor.

Networking for VMs

- `nova-network` worker daemon. Similar to `nova-compute`, it accepts networking tasks from the queue and performs tasks to manipulate the network, such as setting up bridging interfaces or changing iptables rules. This functionality is being migrated to OpenStack Networking, which is a separate OpenStack service.

- `nova-dhcpbridge` script. Tracks IP address leases and records them in the database by using the dnsmasq `dhcp-script` facility. This functionality is being migrated to OpenStack Networking. OpenStack Networking provides a different script.

Console interface

- `nova-consoleauth` daemon. Authorizes tokens for users that console proxies provide. See `nova-novncproxy` and `nova-xvpnvcproxy`. This service must be running for console proxies to work. Many proxies of either type can be run against a single `nova-consoleauth` service in a cluster configuration. For information, see About nova-consoleauth.
- `nova-novncproxy` daemon. Provides a proxy for accessing running instances through a VNC connection. Supports browser-based novnc clients.
- `nova-xvpnvncproxy` daemon. A proxy for accessing running instances through a VNC connection. Supports a Java client specifically designed for OpenStack.
- `nova-cert` daemon. Manages x509 certificates.

Image management (EC2 scenario)

- `nova-objectstore` daemon. Provides an S3 interface for registering images with the Image Service. Mainly used for installations that must support euca2ools. The euca2ools tools talk to `nova-objectstore` in *S3 language*, and `nova-objectstore` translates S3 requests into Image Service requests.
- euca2ools client. A set of command-line interpreter commands for managing cloud resources. Though not an OpenStack module, you can configure `nova-api` to support this EC2 interface. For more information, see the Eucalyptus 3.4 Documentation.

Command-line clients and other interfaces

- nova client. Enables users to submit commands as a tenant administrator or end user.
- nova-manage client. Enables cloud administrators to submit commands.

Other components

- The queue. A central hub for passing messages between daemons. Usually implemented with RabbitMQ, but could be any AMQP message queue, such as Apache Qpid or Zero MQ.
- SQL database. Stores most build-time and runtime states for a cloud infrastructure. Includes instance types that are available for use, instances in use, available networks, and projects. Theoretically, OpenStack Compute can support any database that SQL-Alchemy supports, but the only databases widely used are SQLite3 databases (only appropriate for test and development work), MySQL, and PostgreSQL.

The Compute service interacts with other OpenStack services: Identity Service for authentication, Image Service for images, and the OpenStack dashboard for a web interface.

Install Compute controller services

Compute is a collection of services that enable you to launch virtual machine instances. You can configure these services to run on separate nodes or the same node. In this guide, most services run on the controller node and the service that launches virtual machines runs on a

dedicated compute node. This section shows you how to install and configure these services on the controller node.

1. Install the Compute packages necessary for the controller node.

```
# zypper install openstack-nova-api openstack-nova-scheduler \
  openstack-nova-cert openstack-nova-conductor openstack-nova-console \
  openstack-nova-consoleauth openstack-nova-novncproxy python-novaclient
```

2. Compute stores information in a database. In this guide, we use a MySQL database on the controller node. Configure Compute with the database location and credentials. Replace *NOVA_DBPASS* with the password for the database that you will create in a later step.

```
# openstack-config --set /etc/nova/nova.conf \
  database connection mysql://nova:NOVA_DBPASS@controller/nova
```

3. Set these configuration keys to configure Compute to use the RabbitMQ message broker:

```
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT rpc_backend nova.rpc.impl_kombu
# openstack-config --set /etc/nova/nova.conf DEFAULT rabbit_host
 controller
# openstack-config --set /etc/nova/nova.conf DEFAULT
 rabbit_password RABBIT_PASS
```

4. Set the `my_ip`, `vncserver_listen`, and `vncserver_proxyclient_address` configuration options to the management interface IP address of the controller node:

```
# openstack-config --set /etc/nova/nova.conf DEFAULT my_ip 10.0.0.11
# openstack-config --set /etc/nova/nova.conf DEFAULT vncserver_listen 10.
0.0.11
# openstack-config --set /etc/nova/nova.conf DEFAULT
 vncserver_proxyclient_address 10.0.0.11
```

5. Use the password you created previously to log in as root. Create a `nova` database user:

```
$ mysql -u root -p
mysql> CREATE DATABASE nova;
mysql> GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'localhost' \
IDENTIFIED BY 'NOVA_DBPASS';
mysql> GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%' \
IDENTIFIED BY 'NOVA_DBPASS';
```

6. Create a `nova` user that Compute uses to authenticate with the Identity Service. Use the `service` tenant and give the user the `admin` role:

```
$ keystone user-create --name=nova --pass=NOVA_PASS --email=nova@example.
com
$ keystone user-role-add --user=nova --tenant=service --role=admin
```

7. Configure Compute to use these credentials with the Identity Service running on the controller. Replace *NOVA_PASS* with your Compute password.

```
# openstack-config --set /etc/nova/nova.conf DEFAULT auth_strategy
 keystone
# openstack-config --set /etc/nova/nova.conf keystone_authtoken auth_uri
 http://controller:5000
```

```
# openstack-config --set /etc/nova/nova.conf keystone_authtoken
 auth_host controller
# openstack-config --set /etc/nova/nova.conf keystone_authtoken
 auth_protocol http
# openstack-config --set /etc/nova/nova.conf keystone_authtoken auth_port
 35357
# openstack-config --set /etc/nova/nova.conf keystone_authtoken admin_user
 nova
# openstack-config --set /etc/nova/nova.conf keystone_authtoken
 admin_tenant_name service
# openstack-config --set /etc/nova/nova.conf keystone_authtoken
 admin_password NOVA_PASS
```

8. You must register Compute with the Identity Service so that other OpenStack services can locate it. Register the service and specify the endpoint:

```
$ keystone service-create --name=nova --type=compute \
  --description="OpenStack Compute"
$ keystone endpoint-create \
  --service-id=$(keystone service-list | awk '/ compute / {print $2}') \
  --publicurl=http://controller:8774/v2/%\(tenant_id\)s \
  --internalurl=http://controller:8774/v2/%\(tenant_id\)s \
  --adminurl=http://controller:8774/v2/%\(tenant_id\)s
```

9. Start Compute services and configure them to start when the system boots:

```
# service openstack-nova-api start
# service openstack-nova-cert start
# service openstack-nova-consoleauth start
# service openstack-nova-scheduler start
# service openstack-nova-conductor start
# service openstack-nova-novncproxy start
# chkconfig openstack-nova-api on
# chkconfig openstack-nova-cert on
# chkconfig openstack-nova-consoleauth on
# chkconfig openstack-nova-scheduler on
# chkconfig openstack-nova-conductor on
# chkconfig openstack-nova-novncproxy on
```

10. To verify your configuration, list available images:

```
$ nova image-list
+--------------------------------------+---------------------+--------
+--------+
| ID                                   | Name                | Status |
 Server |
+--------------------------------------+---------------------+--------
+--------+
| acafc7c0-40aa-4026-9673-b879898e1fc2 | cirros-0.3.2-x86_64 | ACTIVE |
     |
+--------------------------------------+---------------------+--------
+--------+
```

Configure a compute node

After you configure the Compute service on the controller node, you must configure another system as a compute node. The compute node receives requests from the controller node and hosts virtual machine instances. You can run all services on a single

node, but the examples in this guide use separate systems. This makes it easy to scale horizontally by adding additional Compute nodes following the instructions in this section.

The Compute service relies on a hypervisor to run virtual machine instances. OpenStack can use various hypervisors, but this guide uses KVM.

1. Install the Compute packages:

```
# zypper install openstack-nova-compute kvm openstack-utils
```

2. Edit the `/etc/nova/nova.conf` configuration file:

```
# openstack-config --set /etc/nova/nova.conf database connection mysql://
nova:NOVA_DBPASS@controller/nova
# openstack-config --set /etc/nova/nova.conf DEFAULT auth_strategy
 keystone
# openstack-config --set /etc/nova/nova.conf keystone_authtoken auth_uri
 http://controller:5000
# openstack-config --set /etc/nova/nova.conf keystone_authtoken
 auth_host controller
# openstack-config --set /etc/nova/nova.conf keystone_authtoken
 auth_protocol http
# openstack-config --set /etc/nova/nova.conf keystone_authtoken auth_port
 35357
# openstack-config --set /etc/nova/nova.conf keystone_authtoken admin_user
 nova
# openstack-config --set /etc/nova/nova.conf keystone_authtoken
 admin_tenant_name service
# openstack-config --set /etc/nova/nova.conf keystone_authtoken
 admin_password NOVA_PASS
```

3. Configure the Compute service to use the RabbitMQ message broker by setting these configuration keys:

```
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT rpc_backend nova.rpc.impl_kombu
# openstack-config --set /etc/nova/nova.conf DEFAULT rabbit_host
 controller
# openstack-config --set /etc/nova/nova.conf DEFAULT
 rabbit_password RABBIT_PASS
```

4. Configure Compute to provide remote console access to instances.

```
# openstack-config --set /etc/nova/nova.conf DEFAULT my_ip 10.0.0.31
# openstack-config --set /etc/nova/nova.conf DEFAULT vnc_enabled True
# openstack-config --set /etc/nova/nova.conf DEFAULT vncserver_listen 0.0.
0.0
# openstack-config --set /etc/nova/nova.conf DEFAULT
 vncserver_proxyclient_address 10.0.0.31
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT novncproxy_base_url http://controller:6080/vnc_auto.html
```

5. Specify the host that runs the Image Service.

```
# openstack-config --set /etc/nova/nova.conf DEFAULT
 glance_host controller
```

6. If you install Compute on a virtual machine for testing purposes, you must determine whether your hypervisor and/or CPU support nested hardware acceleration using the following command:

```
$ egrep -c '(vmx|svm)' /proc/cpuinfo
```

If this command returns a value of *one or greater*, your hypervisor and/or CPU support nested hardware acceleration which requires no additional configuration.

If this command returns a value of *zero*, your hypervisor and/or CPU do not support nested hardware acceleration and `libvirt` must use QEMU instead of KVM. Configure `libvirt` to use QEMU:

```
# openstack-config --set /etc/nova/nova.conf libvirt virt_type qemu
```

7. Start the Compute service and configure it to start when the system boots:

```
# service libvirtd start
# chkconfig libvirtd on
# service openstack-nova-compute start
# chkconfig openstack-nova-compute on
```

7. Add a networking service

Table of Contents

Configuring networking in OpenStack can be a bewildering experience. This guide provides step-by-step instructions for both OpenStack Networking (neutron) and the legacy networking (nova-network) service. If you are unsure which to use, we recommend trying OpenStack Networking because it offers a considerable number of features and flexibility including *plug-ins* for a variety of emerging products supporting *virtual networking*. See the Networking chapter of the *OpenStack Cloud Administrator Guide* for more information.

OpenStack Networking (neutron)

Networking concepts

OpenStack Networking (neutron) manages all of the networking facets for the Virtual Networking Infrastructure (VNI) and the access layer aspects of the Physical Networking Infrastructure (PNI) in your OpenStack environment. OpenStack Networking allows tenants to create advanced virtual network topologies including services such as *firewalls*, *load balancers*, and *virtual private networks (VPNs)*.

Networking provides the following object abstractions: networks, subnets, and routers. Each has functionality that mimics its physical counterpart: networks contain subnets, and routers route traffic between different subnet and networks.

Any given Networking set up has at least one external network. This network, unlike the other networks, is not merely a virtually defined network. Instead, it represents the view into a slice of the external network that is accessible outside the OpenStack installation. IP addresses on the Networking external network are accessible by anybody physically on the outside network. Because this network merely represents a slice of the outside network, DHCP is disabled on this network.

In addition to external networks, any Networking set up has one or more internal networks. These software-defined networks connect directly to the VMs. Only the VMs on any given internal network, or those on subnets connected through interfaces to a similar router, can access VMs connected to that network directly.

For the outside network to access VMs, and vice versa, routers between the networks are needed. Each router has one gateway that is connected to a network and many interfaces that are connected to subnets. Like a physical router, subnets can access machines on other subnets that are connected to the same router, and machines can access the outside network through the gateway for the router.

Additionally, you can allocate IP addresses on external networks to ports on the internal network. Whenever something is connected to a subnet, that connection is called a port. You can associate external network IP addresses with ports to VMs. This way, entities on the outside network can access VMs.

Networking also supports *security groups*. Security groups enable administrators to define firewall rules in groups. A VM can belong to one or more security groups, and Networking applies the rules in those security groups to block or unblock ports, port ranges, or traffic types for that VM.

Each plug-in that Networking uses has its own concepts. While not vital to operating Networking, understanding these concepts can help you set up Networking. All Networking installations use a core plug-in and a security group plug-in (or just the No-Op security group plug-in). Additionally, Firewall-as-a-service (FWaaS) and Load-balancing-as-a-service (LBaaS) plug-ins are available.

Modular Layer 2 (ML2) plug-in

Configure controller node

Prerequisites

Before you configure OpenStack Networking (neutron), you must create a database and Identity service credentials including a user and service.

1. Connect to the database as the root user, create the `neutron` database, and grant the proper access to it:

   Replace *NEUTRON_DBPASS* with a suitable password.

   ```
   $ mysql -u root -p
   mysql> CREATE DATABASE neutron;
   mysql> GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'localhost' \
   IDENTIFIED BY 'NEUTRON_DBPASS';
   mysql> GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'%' \
   IDENTIFIED BY 'NEUTRON_DBPASS';
   ```

2. Create Identity service credentials for Networking:

   a. Create the `neutron` user:

      Replace *NEUTRON_PASS* with a suitable password and *`neutron@example.com`* with a suitable e-mail address.

      ```
      $ keystone user-create --name neutron --pass NEUTRON_PASS --
      email neutron@example.com
      ```

   b. Link the `neutron` user to the `service` tenant and `admin` role:

      ```
      $ keystone user-role-add --user neutron --tenant service --role admin
      ```

   c. Create the `neutron` service:

      ```
      $ keystone service-create --name neutron --type network --description
       "OpenStack Networking"
      ```

d. Create the service endpoint:

```
$ keystone endpoint-create \
  --service-id $(keystone service-list | awk '/ network / {print $2}')
 \
  --publicurl http://controller:9696 \
  --adminurl http://controller:9696 \
  --internalurl http://controller:9696
```

To install the Networking components

- `# zypper install openstack-neutron openstack-neutron-server`

Note

SUSE does not use a separate ML2 plug-in package.

To configure the Networking server component

The Networking server component configuration includes the database, authentication mechanism, message broker, topology change notifier, and plug-in.

1. Configure Networking to use the database:

   Replace *NEUTRON_DBPASS* with a suitable password.

   ```
   # openstack-config --set /etc/neutron/neutron.conf database connection \
     mysql://neutron:NEUTRON_DBPASS@controller/neutron
   ```

2. Configure Networking to use the Identity service for authentication:

   Replace *NEUTRON_PASS* with the password you chose for the `neutron` user in the Identity service.

   ```
   # openstack-config --set /etc/neutron/neutron.conf DEFAULT \
     auth_strategy keystone
   # openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
     auth_uri http://controller:5000
   # openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
     auth_host controller
   # openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
     auth_protocol http
   # openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
     auth_port 35357
   # openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
     admin_tenant_name service
   # openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
     admin_user neutron
   # openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
     admin_password NEUTRON_PASS
   ```

3. Configure Networking to use the message broker:

   Replace *RABBIT_PASS* with the password you chose for the `guest` account in RabbitMQ.

   ```
   # openstack-config --set /etc/neutron/neutron.conf DEFAULT \
   ```

```
  rpc_backend neutron.openstack.common.rpc.impl_kombu
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  rabbit_host controller
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  rabbit_userid guest
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  rabbit_password RABBIT_PASS
```

4. Configure Networking to notify Compute about network topology changes:

```
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  notify_nova_on_port_status_changes True
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  notify_nova_on_port_data_changes True
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  nova_url http://controller:8774/v2
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  nova_admin_username nova
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  nova_admin_tenant_id $(keystone tenant-list | awk '/ service / { print
 $2 }')
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  nova_admin_password NOVA_PASS
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  nova_admin_auth_url http://controller:35357/v2.0
```

5. Configure Networking to use the Modular Layer 2 (ML2) plug-in and associated services:

```
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  core_plugin ml2
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  service_plugins router
```

Note

We recommend adding `verbose = True` to the `[DEFAULT]` section in `/etc/neutron/neutron.conf` to assist with troubleshooting.

6. Comment out any lines in the `[service_providers]` section.

To configure the Modular Layer 2 (ML2) plug-in

The ML2 plug-in uses the Open vSwitch (OVS) mechanism (agent) to build the virtual networking framework for instances. However, the controller node does not need the OVS agent or service because it does not handle instance network traffic.

- Run the following commands:

```
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 \
  type_drivers gre
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 \
  tenant_network_types gre
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 \
  mechanism_drivers openvswitch
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini
 ml2_type_gre \
  tunnel_id_ranges 1:1000
```

```
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini
 securitygroup \
  firewall_driver neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini
 securitygroup \
  enable_security_group True
```

To configure Compute to use Networking

By default, most distributions configure Compute to use legacy networking. You must reconfigure Compute to manage networks through Networking.

- Run the following commands:

  Replace *NEUTRON_PASS* with the password you chose for the `neutron` user in the Identity service.

```
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  network_api_class nova.network.neutronv2.api.API
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  neutron_url http://controller:9696
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  neutron_auth_strategy keystone
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  neutron_admin_tenant_name service
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  neutron_admin_username neutron
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  neutron_admin_password NEUTRON_PASS
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  neutron_admin_auth_url http://controller:35357/v2.0
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  linuxnet_interface_driver nova.network.linux_net.LinuxOVSInterfaceDriver
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  firewall_driver nova.virt.firewall.NoopFirewallDriver
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  security_group_api neutron
```

Note

By default, Compute uses an internal firewall service. Since Networking includes a firewall service, you must disable the Compute firewall service by using the `nova.virt.firewall.NoopFirewallDriver` firewall driver.

To finalize installation

1. The Networking service initialization scripts expect the variable `NEUTRON_PLUGIN_CONF` in file `/etc/sysconfig/neutron` to reference the configuration file associated with your chosen plug-in. Using ML2, for example, edit the `/etc/sysconfig/neutron` file and add the following:

```
NEUTRON_PLUGIN_CONF="/etc/neutron/plugins/ml2/ml2_conf.ini"
```

2. Restart the Compute services:

```
# service openstack-nova-api restart
# service openstack-nova-scheduler restart
# service openstack-nova-conductor restart
```

3. Start the Networking service and configure it to start when the system boots:

```
# service openstack-neutron start
# chkconfig openstack-neutron on
```

Configure network node

Prerequisites

Before you configure OpenStack Networking, you must enable certain kernel networking functions.

1. Edit `/etc/sysctl.conf` to contain the following:

```
net.ipv4.ip_forward=1
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0
```

2. Implement the changes:

```
# sysctl -p
```

To install the Networking components

- ```
  # zypper install openstack-neutron-openvswitch-agent openstack-neutron-l3-agent \
    openstack-neutron-dhcp-agent openstack-neutron-metadata-agent
  ```

Note

SUSE does not use a separate ML2 plug-in package.

To configure the Networking common components

The Networking common component configuration includes the authentication mechanism, message broker, and plug-in.

1. Configure Networking to use the Identity service for authentication:

   Replace *NEUTRON_PASS* with the password you chose for the `neutron` user in the Identity service.

```
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  auth_strategy keystone
# openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
  auth_uri http://controller:5000
# openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
  auth_host controller
# openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
  auth_protocol http
# openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
```
```

```
  auth_port 35357
# openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
  admin_tenant_name service
# openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
  admin_user neutron
# openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
  admin_password NEUTRON_PASS
```

2. Configure Networking to use the message broker:

   Replace *RABBIT_PASS* with the password you chose for the `guest` account in RabbitMQ.

```
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  rpc_backend neutron.openstack.common.rpc.impl_kombu
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  rabbit_host controller
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  rabbit_userid guest
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  rabbit_password RABBIT_PASS
```

3. Configure Networking to use the Modular Layer 2 (ML2) plug-in and associated services:

```
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  core_plugin ml2
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  service_plugins router
```

Note

We recommend adding `verbose = True` to the `[DEFAULT]` section in `/etc/neutron/neutron.conf` to assist with troubleshooting.

4. Comment out any lines in the `[service_providers]` section.

To configure the Layer-3 (L3) agent

The *Layer-3 (L3) agent* provides routing services for instance virtual networks.

- Run the following commands:

```
# openstack-config --set /etc/neutron/l3_agent.ini DEFAULT \
  interface_driver neutron.agent.linux.interface.OVSInterfaceDriver
# openstack-config --set /etc/neutron/l3_agent.ini DEFAULT \
  use_namespaces True
```

Note

We recommend adding `verbose = True` to the `[DEFAULT]` section in `/etc/neutron/l3_agent.ini` to assist with troubleshooting.

To configure the DHCP agent

The *DHCP agent* provides *DHCP* services for instance virtual networks.

- Run the following commands:

```
# openstack-config --set /etc/neutron/dhcp_agent.ini DEFAULT \
  interface_driver neutron.agent.linux.interface.OVSInterfaceDriver
# openstack-config --set /etc/neutron/dhcp_agent.ini DEFAULT \
  dhcp_driver neutron.agent.linux.dhcp.Dnsmasq
# openstack-config --set /etc/neutron/dhcp_agent.ini DEFAULT \
  use_namespaces True
```

Note

We recommend adding `verbose = True` to the `[DEFAULT]` section in `/etc/neutron/dhcp_agent.ini` to assist with troubleshooting.

To configure the metadata agent

The *metadata agent* provides configuration information such as credentials for remote access to instances.

1. Run the following commands:

   Replace *NEUTRON_PASS* with the password you chose for the `neutron` user in the Identity service. Replace *METADATA_SECRET* with a suitable secret for the metadata proxy.

```
# openstack-config --set /etc/neutron/metadata_agent.ini DEFAULT \
  auth_url http://controller:5000/v2.0
# openstack-config --set /etc/neutron/metadata_agent.ini DEFAULT \
  auth_region regionOne
# openstack-config --set /etc/neutron/metadata_agent.ini DEFAULT \
  admin_tenant_name service
# openstack-config --set /etc/neutron/metadata_agent.ini DEFAULT \
  admin_user neutron
# openstack-config --set /etc/neutron/metadata_agent.ini DEFAULT \
  admin_password NEUTRON_PASS
# openstack-config --set /etc/neutron/metadata_agent.ini DEFAULT \
  nova_metadata_ip controller
# openstack-config --set /etc/neutron/metadata_agent.ini DEFAULT \
  metadata_proxy_shared_secret METADATA_SECRET
```

Note

We recommend adding `verbose = True` to the `[DEFAULT]` section in `/etc/neutron/metadata_agent.ini` to assist with troubleshooting.

2. 

   Note

   Perform the next two steps on the *controller* node.

3. On the *controller* node, configure Compute to use the metadata service:

   Replace *METADATA_SECRET* with the secret you chose for the metadata proxy.

```
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  service_neutron_metadata_proxy true
# openstack-config --set /etc/nova/nova.conf DEFAULT \
```

```
  neutron_metadata_proxy_shared_secret METADATA_SECRET
```

4. On the *controller* node, restart the Compute *API* service:

```
# service openstack-nova-api restart
```

To configure the Modular Layer 2 (ML2) plug-in

The ML2 plug-in uses the Open vSwitch (OVS) mechanism (agent) to build virtual networking framework for instances.

- Run the following commands:

  Replace *INSTANCE_TUNNELS_INTERFACE_IP_ADDRESS* with the IP address of the instance tunnels network interface on your network node. This guide uses `10.0.1.21` for the IP address of the instance tunnels network interface on the network node.

```
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 \
  type_drivers gre
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 \
  tenant_network_types gre
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 \
  mechanism_drivers openvswitch
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini
 ml2_type_gre \
  tunnel_id_ranges 1:1000
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ovs \
  local_ip INSTANCE_TUNNELS_INTERFACE_IP_ADDRESS
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ovs \
  tunnel_type gre
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ovs \
  enable_tunneling True
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini
 securitygroup \
  firewall_driver neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini
 securitygroup \
  enable_security_group True
```

To configure the Open vSwitch (OVS) service

The OVS service provides the underlying virtual networking framework for instances. The integration bridge `br-int` handles internal instance network traffic within OVS. The external bridge `br-ext` handles external instance network traffic within OVS. The external bridge requires a port on the physical external network interface to provide instances with external network access. In essence, this port bridges the virtual and physical external networks in your environment.

1. Start the OVS service and configure it to start when the system boots:

```
# service openvswitch-switch start
# chkconfig openvswitch-switch on
```

2. Add the integration bridge:

```
# ovs-vsctl add-br br-int
```

3. Add the external bridge:

```
# ovs-vsctl add-br br-ex
```

4. Add a port to the external bridge that connects to the physical external network interface:

   Replace *INTERFACE_NAME* with the actual interface name. For example, *eth2* or *ens256*.

```
# ovs-vsctl add-port br-ex INTERFACE_NAME
```

Note

Depending on your network interface driver, you may need to disable *Generic Receive Offload (GRO)* to achieve suitable throughput between your instances and the external network.

To temporarily disable GRO on the external network interface while testing your environment:

```
# ethtool -K INTERFACE_NAME gro off
```

To finalize the installation

1. The Networking service initialization scripts expect the variable `NEUTRON_PLUGIN_CONF` in the `/etc/sysconfig/neutron` file to reference the configuration file associated with your chosen plug-in. Using ML2, for example, edit the `/etc/sysconfig/neutron` file and add the following:

```
NEUTRON_PLUGIN_CONF="/etc/neutron/plugins/ml2/ml2_conf.ini"
```

2. Start the Networking services and configure them to start when the system boots:

```
# service openstack-neutron-openvswitch-agent start
# service openstack-neutron-l3-agent start
# service openstack-neutron-dhcp-agent start
# service openstack-neutron-metadata-agent start
# chkconfig openstack-neutron-openvswitch-agent on
# chkconfig openstack-neutron-l3-agent on
# chkconfig openstack-neutron-dhcp-agent on
# chkconfig openstack-neutron-metadata-agent on
```

Configure compute node

Prerequisites

Before you configure OpenStack Networking, you must enable certain kernel networking functions.

1. Edit `/etc/sysctl.conf` to contain the following:

```
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0
```

2. Implement the changes:

```
# sysctl -p
```

To install the Networking components

- ```
  # zypper install openstack-neutron-openvswitch-agent
  ```

Note

SUSE does not use a separate ML2 plug-in package.

To configure the Networking common components

The Networking common component configuration includes the authentication mechanism, message broker, and plug-in.

1. Configure Networking to use the Identity service for authentication:

   Replace *NEUTRON_PASS* with the password you chose for the `neutron` user in the Identity service.

   ```
   # openstack-config --set /etc/neutron/neutron.conf DEFAULT \
     auth_strategy keystone
   # openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
     auth_uri http://controller:5000
   # openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
     auth_host controller
   # openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
     auth_protocol http
   # openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
     auth_port 35357
   # openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
     admin_tenant_name service
   # openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
     admin_user neutron
   # openstack-config --set /etc/neutron/neutron.conf keystone_authtoken \
     admin_password NEUTRON_PASS
   ```

2. Configure Networking to use the message broker:

   Replace *RABBIT_PASS* with the password you chose for the `guest` account in RabbitMQ.

   ```
   # openstack-config --set /etc/neutron/neutron.conf DEFAULT \
     rpc_backend neutron.openstack.common.rpc.impl_kombu
   # openstack-config --set /etc/neutron/neutron.conf DEFAULT \
     rabbit_host controller
   # openstack-config --set /etc/neutron/neutron.conf DEFAULT \
     rabbit_userid guest
   # openstack-config --set /etc/neutron/neutron.conf DEFAULT \
     rabbit_password RABBIT_PASS
   ```

3. Configure Networking to use the Modular Layer 2 (ML2) plug-in and associated services:

   ```
   # openstack-config --set /etc/neutron/neutron.conf DEFAULT \
     core_plugin ml2
   ```
   ```

```
# openstack-config --set /etc/neutron/neutron.conf DEFAULT \
  service_plugins router
```

Note

We recommend adding `verbose = True` to the `[DEFAULT]` section in `/etc/neutron/neutron.conf` to assist with troubleshooting.

4. Comment out any lines in the `[service_providers]` section.

To configure the Modular Layer 2 (ML2) plug-in

The ML2 plug-in uses the Open vSwitch (OVS) mechanism (agent) to build the virtual networking framework for instances.

- Run the following commands:

  Replace *INSTANCE_TUNNELS_INTERFACE_IP_ADDRESS* with the IP address of the instance tunnels network interface on your compute node. This guide uses `10.0.1.31` for the IP address of the instance tunnels network interface on the first compute node.

```
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 \
  type_drivers gre
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 \
  tenant_network_types gre
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 \
  mechanism_drivers openvswitch
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini
 ml2_type_gre \
  tunnel_id_ranges 1:1000
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ovs \
  local_ip INSTANCE_TUNNELS_INTERFACE_IP_ADDRESS
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ovs \
  tunnel_type gre
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini ovs \
  enable_tunneling True
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini
 securitygroup \
  firewall_driver neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver
# openstack-config --set /etc/neutron/plugins/ml2/ml2_conf.ini
 securitygroup \
  enable_security_group True
```

To configure the Open vSwitch (OVS) service

The OVS service provides the underlying virtual networking framework for instances. The integration bridge `br-int` handles internal instance network traffic within OVS.

1. Start the OVS service and configure it to start when the system boots:

```
# service openvswitch-switch start
# chkconfig openvswitch-switch on
```

2. Add the integration bridge:

```
# ovs-vsctl add-br br-int
```

To configure Compute to use Networking

By default, most distributions configure Compute to use legacy networking. You must reconfigure Compute to manage networks through Networking.

- Run the following commands:

  Replace *NEUTRON_PASS* with the password you chose for the `neutron` user in the Identity service.

```
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  network_api_class nova.network.neutronv2.api.API
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  neutron_url http://controller:9696
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  neutron_auth_strategy keystone
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  neutron_admin_tenant_name service
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  neutron_admin_username neutron
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  neutron_admin_password NEUTRON_PASS
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  neutron_admin_auth_url http://controller:35357/v2.0
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  linuxnet_interface_driver nova.network.linux_net.LinuxOVSInterfaceDriver
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  firewall_driver nova.virt.firewall.NoopFirewallDriver
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  security_group_api neutron
```

Note

By default, Compute uses an internal firewall service. Since Networking includes a firewall service, you must disable the Compute firewall service by using the `nova.virt.firewall.NoopFirewallDriver` firewall driver.

To finalize the installation

1. The Networking service initialization scripts expect the variable `NEUTRON_PLUGIN_CONF` in the `/etc/sysconfig/neutron` file to reference the configuration file associated with your chosen plug-in. Using ML2, for example, edit the `/etc/sysconfig/neutron` file and add the following:

```
NEUTRON_PLUGIN_CONF="/etc/neutron/plugins/ml2/ml2_conf.ini"
```

2. Restart the Compute service:

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

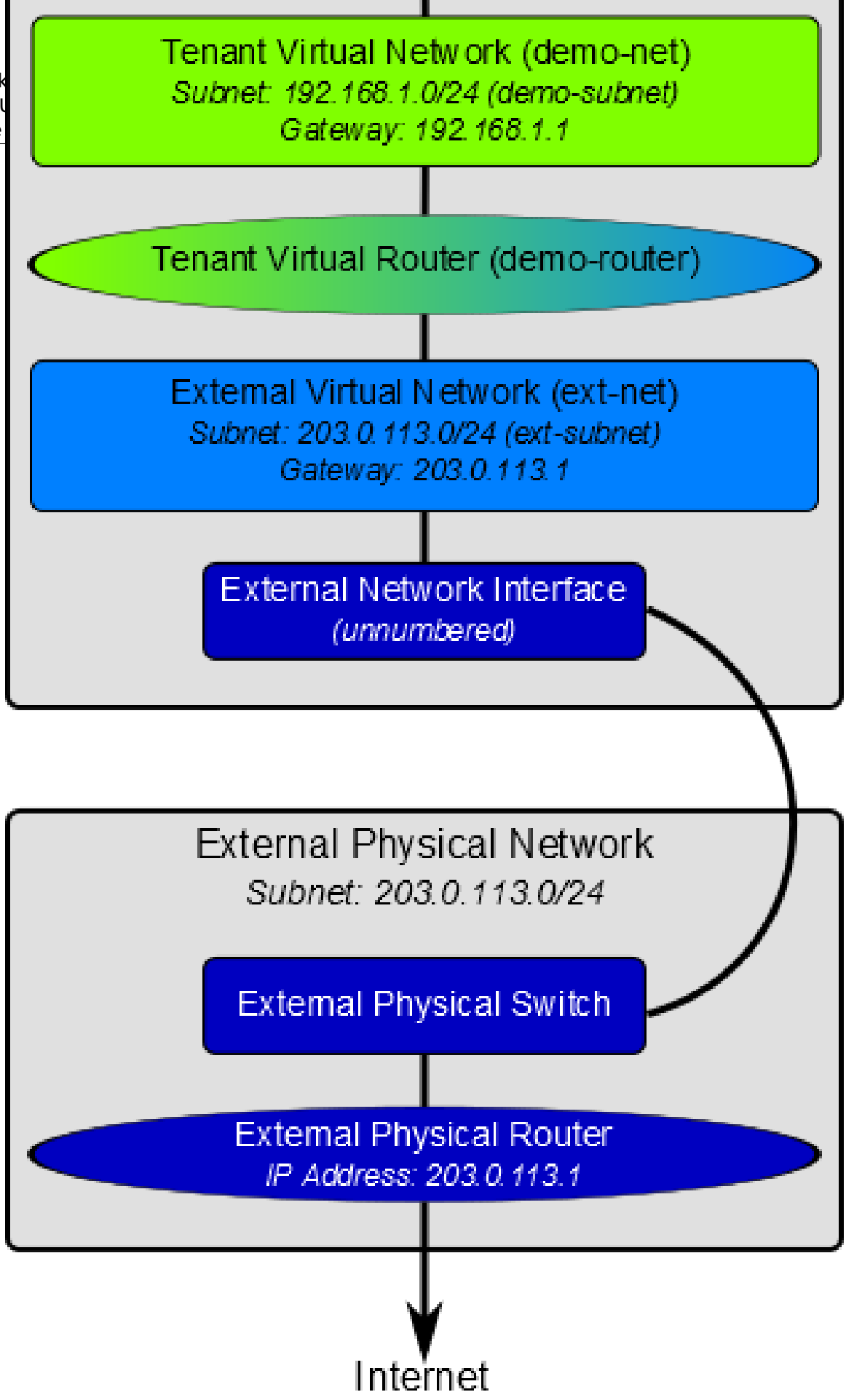
3. Start the Open vSwitch (OVS) agent and configure it to start when the system boots:

```
# service openstack-neutron-openvswitch-agent start
```

```
# chkconfig openstack-neutron-openvswitch-agent on
```

Create initial networks

Before launching your first instance, you must create the necessary virtual network infrastructure to which the instance will connect, including the external network and tenant network. See Figure 7.1, “Initial networks” [62]. After creating this infrastructure, we recommend that you verify connectivity and resolve any issues before proceeding further.


Tenant Virtual Network (demo-net)
Subnet: 192.168.1.0/24 (demo-subnet)
Gateway: 192.168.1.1
Tenant Virtual Router (demo-router)
External Virtual Network (ext-net)
Subnet: 203.0.113.0/24 (ext-subnet)
Gateway: 203.0.113.1
External Network Interface
(unnumbered)
External Physical Network
Subnet: 203.0.113.0/24
External Physical Switch
External Physical Router
IP Address: 203.0.113.1
Internet

External network

The external network typically provides internet access for your instances. By default, this network only allows internet access *from* instances using *Network Address Translation (NAT)*. You can enable internet access *to* individual instances using a *floating IP address* and suitable *security group* rules. The `admin` tenant owns this network because it provides external network access for multiple tenants. You must also enable sharing to allow access by those tenants.

Note

Perform these commands on the controller node.

To create the external network

1. Source the `admin` tenant credentials:

```
$ source admin-openrc.sh
```

2. Create the network:

```
$ neutron net-create ext-net --shared --router:external=True
Created a new network:
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | True                                 |
| id                        | 893aebb9-1c1e-48be-8908-6b947f3237b3 |
| name                      | ext-net                              |
| provider:network_type     | gre                                  |
| provider:physical_network |                                      |
| provider:segmentation_id  | 1                                    |
| router:external           | True                                 |
| shared                    | True                                 |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tenant_id                 | 54cd044c64d5408b83f843d63624e0d8     |
+---------------------------+--------------------------------------+
```

Like a physical network, a virtual network requires a *subnet* assigned to it. The external network shares the same subnet and *gateway* associated with the physical network connected to the external interface on the network node. You should specify an exclusive slice of this subnet for *router* and floating IP addresses to prevent interference with other devices on the external network.

Replace *`FLOATING_IP_START`* and *`FLOATING_IP_END`* with the first and last IP addresses of the range that you want to allocate for floating IP addresses. Replace *`EXTERNAL_NETWORK_CIDR`* with the subnet associated with the physical network. Replace *`EXTERNAL_NETWORK_GATEWAY`* with the gateway associated with the physical network, typically the ".1" IP address. You should disable *DHCP* on this subnet because instances do not connect directly to the external network and floating IP addresses require manual assignment.

To create a subnet on the external network

- Create the subnet:

```
$ neutron subnet-create ext-net --name ext-subnet \
  --allocation-pool start=FLOATING_IP_START,end=FLOATING_IP_END \
  --disable-dhcp --gateway EXTERNAL_NETWORK_GATEWAY EXTERNAL_NETWORK_CIDR
```

For example, using `203.0.113.0/24` with floating IP address range `203.0.113.101` to `203.0.113.200`:

```
$ neutron subnet-create ext-net --name ext-subnet \
  --allocation-pool start=203.0.113.101,end=203.0.113.200 \
  --disable-dhcp --gateway 203.0.113.1 203.0.113.0/24
Created a new subnet:
+-------------------
+-----------------------------------------------------+
| Field             | Value
 |
+-------------------
+-----------------------------------------------------+
| allocation_pools  | {"start": "203.0.113.101", "end": "203.0.113.200"}
 |
| cidr              | 203.0.113.0/24
 |
| dns_nameservers   |
 |
| enable_dhcp       | False
 |
| gateway_ip        | 203.0.113.1
 |
| host_routes       |
 |
| id                | 9159f0dc-2b63-41cf-bd7a-289309da1391
 |
| ip_version        | 4
 |
| ipv6_address_mode |
 |
| ipv6_ra_mode      |
 |
| name              | ext-subnet
 |
| network_id        | 893aebb9-1c1e-48be-8908-6b947f3237b3
 |
| tenant_id         | 54cd044c64d5408b83f843d63624e0d8
 |
+-------------------
+-----------------------------------------------------+
```

Tenant network

The tenant network provides internal network access for instances. The architecture isolates this type of network from other tenants. The `demo` tenant owns this network because it only provides network access for instances within it.

Note

Perform these commands on the controller node.

To create the tenant network

1. Source the demo tenant credentials:

```
$ source demo-openrc.sh
```

2. Create the network:

```
$ neutron net-create demo-net
Created a new network:
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| admin_state_up | True                                 |
| id             | ac108952-6096-4243-adf4-bb6615b3de28 |
| name           | demo-net                             |
| shared         | False                                |
| status         | ACTIVE                               |
| subnets        |                                      |
| tenant_id      | cdef0071a0194d19ac6bb63802dc9bae     |
+----------------+--------------------------------------+
```

Like the external network, your tenant network also requires a subnet attached to it. You can specify any valid subnet because the architecture isolates tenant networks. Replace *TENANT_NETWORK_CIDR* with the subnet you want to associate with the tenant network. Replace *TENANT_NETWORK_GATEWAY* with the gateway you want to associate with this network, typically the ".1" IP address. By default, this subnet will use DHCP so your instances can obtain IP addresses.

To create a subnet on the tenant network

- Create the subnet:

```
$ neutron subnet-create demo-net --name demo-subnet \
  --gateway TENANT_NETWORK_GATEWAY TENANT_NETWORK_CIDR
```

Example using 192.168.1.0/24:

```
$ neutron subnet-create demo-net --name demo-subnet \
  --gateway 192.168.1.1 192.168.1.0/24
Created a new subnet:
+-------------------
+-----------------------------------------------------+
| Field             | Value
 |
+-------------------
+-----------------------------------------------------+
| allocation_pools  | {"start": "192.168.1.2", "end": "192.168.1.254"}
 |
| cidr              | 192.168.1.0/24
 |
| dns_nameservers   |
 |
| enable_dhcp       | True
 |
| gateway_ip        | 192.168.1.1
 |
| host_routes       |
 |
```

```
| id                | 69d38773-794a-4e49-b887-6de6734e792d
 |
| ip_version        | 4
 |
| ipv6_address_mode |
 |
| ipv6_ra_mode      |
 |
| name              | demo-subnet
 |
| network_id        | ac108952-6096-4243-adf4-bb6615b3de28
 |
| tenant_id         | cdef0071a0194d19ac6bb63802dc9bae
 |
+-------------------
+-----------------------------------------------------+
```

A virtual router passes network traffic between two or more virtual networks. Each router requires one or more *interfaces* and/or gateways that provide access to specific networks. In this case, you will create a router and attach your tenant and external networks to it.

To create a router on the tenant network and attach the external and tenant networks to it

1. Create the router:

```
$ neutron router-create demo-router
Created a new router:
+-----------------------+--------------------------------------+
| Field                 | Value                                |
+-----------------------+--------------------------------------+
| admin_state_up        | True                                 |
| external_gateway_info |                                      |
| id                    | 635660ae-a254-4feb-8993-295aa9ec6418 |
| name                  | demo-router                          |
| status                | ACTIVE                               |
| tenant_id             | cdef0071a0194d19ac6bb63802dc9bae     |
+-----------------------+--------------------------------------+
```

2. Attach the router to the `demo` tenant subnet:

```
$ neutron router-interface-add demo-router demo-subnet
Added interface b1a894fd-aee8-475c-9262-4342afdc1b58 to router demo-
router.
```

3. Attach the router to the external network by setting it as the gateway:

```
$ neutron router-gateway-set demo-router ext-net
Set gateway for router demo-router
```

Verify connectivity

We recommend that you verify network connectivity and resolve any issues before proceeding further. Following the external network subnet example using `203.0.113.0/24`, the tenant router gateway should occupy the lowest IP address in the floating IP address range, `203.0.113.101`. If you configured your external physical network and virtual networks correctly, you you should be able to **ping** this IP address from any host on your external physical network.

Note

If you are building your OpenStack nodes as virtual machines, you must configure the hypervisor to permit promiscuous mode on the external network.

To verify network connectivity

- Ping the tenant router gateway:

```
$ ping -c 4 203.0.113.101
PING 203.0.113.101 (203.0.113.101) 56(84) bytes of data.
64 bytes from 203.0.113.101: icmp_req=1 ttl=64 time=0.619 ms
64 bytes from 203.0.113.101: icmp_req=2 ttl=64 time=0.189 ms
64 bytes from 203.0.113.101: icmp_req=3 ttl=64 time=0.165 ms
64 bytes from 203.0.113.101: icmp_req=4 ttl=64 time=0.216 ms

--- 203.0.113.101 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.165/0.297/0.619/0.187 ms
```

Legacy networking (nova-network)

Configure controller node

Legacy networking primarily involves compute nodes. However, you must configure the controller node to use it.

To configure legacy networking

1. Run the following commands:

```
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  network_api_class nova.network.api.API
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  security_group_api nova
```

2. Restart the Compute services:

```
# service openstack-nova-api restart
# service openstack-nova-scheduler restart
# service openstack-nova-conductor restart
```

Configure compute node

This section covers deployment of a simple *flat network* that provides IP addresses to your instances via *DHCP*. If your environment includes multiple compute nodes, the *multi-host* feature provides redundancy by spreading network functions across compute nodes.

To install legacy networking components

- `# zypper install openstack-nova-network openstack-nova-api`

To configure legacy networking

1. Run the following commands:

Replace *INTERFACE_NAME* with the actual interface name for the external network. For example, *eth1* or *ens224*.

```
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  network_api_class nova.network.api.API
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  security_group_api nova
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  network_manager nova.network.manager.FlatDHCPManager
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  firewall_driver nova.virt.libvirt.firewall.IptablesFirewallDriver
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  network_size 254
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  allow_same_net_traffic False
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  multi_host True
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  send_arp_for_ha True
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  share_dhcp_address True
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  force_dhcp_release True
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  flat_network_bridge br100
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  flat_interface INTERFACE_NAME
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  public_interface INTERFACE_NAME
```

2. Start the services and configure them to start when the system boots:

```
# service openstack-nova-network start
# service openstack-nova-api-metadata start
# chkconfig openstack-nova-network on
# chkconfig openstack-nova-api-metadata on
```

Create initial network

Before launching your first instance, you must create the necessary virtual network infrastructure to which the instance will connect. This network typically provides internet access *from* instances. You can enable internet access *to* individual instances using a *floating IP address* and suitable *security group* rules. The `admin` tenant owns this network because it provides external network access for multiple tenants.

This network shares the same *subnet* associated with the physical network connected to the external *interface* on the compute node. You should specify an exclusive slice of this subnet to prevent interference with other devices on the external network.

Note

Perform these commands on the controller node.

To create the network

1. Source the `admin` tenant credentials:

```
$ source admin-openrc.sh
```

2. Create the network:

   Replace *NETWORK_CIDR* with the subnet associated with the physical network.

```
$ nova network-create demo-net --bridge br100 --multi-host T \
  --fixed-range-v4 NETWORK_CIDR
```

   For example, using an exclusive slice of `203.0.113.0/24` with IP address range `203.0.113.24` to `203.0.113.32`:

```
$ nova network-create demo-net --bridge br100 --multi-host T \
  --fixed-range-v4 203.0.113.24/29
```

Note

This command provides no output.

3. Verify creation of the network:

```
$ nova net-list
+--------------------------------------+----------+------------------+
| ID                                   | Label    | CIDR             |
+--------------------------------------+----------+------------------+
| 84b34a65-a762-44d6-8b5e-3b461a53f513 | demo-net | 203.0.113.24/29  |
+--------------------------------------+----------+------------------+
```

Next steps

Your OpenStack environment now includes the core components necessary to launch a basic instance. You can launch an instance or add more services to your environment in the following chapters.

8. Add the dashboard

Table of Contents

The OpenStack dashboard, also known as Horizon, is a Web interface that enables cloud administrators and users to manage various OpenStack resources and services.

The dashboard enables web-based interactions with the OpenStack Compute cloud controller through the OpenStack APIs.

These instructions show an example deployment configured with an Apache web server.

After you install and configure the dashboard, you can complete the following tasks:

- Customize your dashboard. See section Customize the dashboard in the *OpenStack Cloud Administrator Guide*.
- Set up session storage for the dashboard. See the section called "Set up session storage for the dashboard" [72].

System requirements

Before you install the OpenStack dashboard, you must meet the following system requirements:

- OpenStack Compute installation. Enable the Identity Service for user and project management.

  Note the URLs of the Identity Service and Compute endpoints.
- Identity Service user with sudo privileges. Because Apache does not serve content from a root user, users must run the dashboard as an Identity Service user with sudo privileges.
- Python 2.6 or 2.7. The Python version must support Django. The Python version should run on any system, including Mac OS X. Installation prerequisites might differ by platform.

Then, install and configure the dashboard on a node that can contact the Identity Service.

Provide users with the following information so that they can access the dashboard through a web browser on their local machine:

- The public IP address from which they can access the dashboard
- The user name and password with which they can access the dashboard

Your web browser, and that of your users, must support HTML5 and have cookies and JavaScript enabled.

Note

To use the VNC client with the dashboard, the browser must support HTML5 Canvas and HTML5 WebSockets.

For details about browsers that support noVNC, see https://github.com/kanaka/noVNC/blob/master/README.md, and https://github.com/kanaka/noVNC/wiki/Browser-support, respectively.

Install the dashboard

Before you can install and configure the dashboard, meet the requirements in the section called "System requirements" [70].

Note

When you install only Object Storage and the Identity Service, even if you install the dashboard, it does not pull up projects and is unusable.

For more information about how to deploy the dashboard, see deployment topics in the developer documentation.

1. Install the dashboard on the node that can contact the Identity Service as root:

```
# zypper install memcached python-python-memcached apache2-mod_wsgi
 openstack-dashboard
```

2. Modify the value of `CACHES['default']['LOCATION']` in `/srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py` to match the ones set in `/etc/sysconfig/memcached`.

   Open and look for this line:

```
CACHES = {
'default': {
'BACKEND' : 'django.core.cache.backends.memcached.MemcachedCache',
'LOCATION' : '127.0.0.1:11211'
}
}
```

Notes

- The address and port must match the ones set in `/etc/sysconfig/memcached`.

  If you change the memcached settings, you must restart the Apache web server for the changes to take effect.

- You can use options other than memcached option for session storage. Set the session back-end through the *SESSION_ENGINE* option.

- To change the timezone, use the dashboard or edit the `/srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py` file.

Change the following parameter: `TIME_ZONE = "UTC"`

3. Update the `ALLOWED_HOSTS` in `local_settings.py` to include the addresses you wish to access the dashboard from.

   Edit `/srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py`:

   ```
   ALLOWED_HOSTS = ['localhost', 'my-desktop']
   ```

4. This guide assumes that you are running the Dashboard on the controller node. You can easily run the dashboard on a separate server, by changing the appropriate settings in `local_settings.py`.

   Edit `/srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py` and change `OPENSTACK_HOST` to the hostname of your Identity Service:

   ```
   OPENSTACK_HOST = "controller"
   ```

5. Setup Apache configuration:

   ```
   # cp /etc/apache2/conf.d/openstack-dashboard.conf.sample \
     /etc/apache2/conf.d/openstack-dashboard.conf
   # a2enmod rewrite;a2enmod ssl;a2enmod wsgi
   ```

6. By default, the `openstack-dashboard` package enables a database as session store. Before you continue, either change the session store set up as described in the section called “Set up session storage for the dashboard” [72] or finish the setup of the database session store as explained in the section called “Initialize and configure the database” [74].

7. Start the Apache web server and memcached:

   ```
   # service apache2 start
   # service memcached start
   # chkconfig apache2 on
   # chkconfig memcached on
   ```

8. You can now access the dashboard at `http://controller`.

   Login with credentials for any user that you created with the OpenStack Identity Service.

Set up session storage for the dashboard

The dashboard uses Django sessions framework to handle user session data. However, you can use any available session back end. You customize the session back end through the `SESSION_ENGINE` setting in your `local_settings` file (on Fedora/RHEL/CentOS: `/etc/openstack-dashboard/local_settings`, on Ubuntu and Debian: `/etc/openstack-dashboard/local_settings.py` and on openSUSE: `/srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py`).

The following sections describe the pros and cons of each option as it pertains to deploying the dashboard.

Local memory cache

Local memory storage is the quickest and easiest session back end to set up, as it has no external dependencies whatsoever. It has the following significant drawbacks:

- No shared storage across processes or workers.
- No persistence after a process terminates.

The local memory back end is enabled as the default for Horizon solely because it has no dependencies. It is not recommended for production use, or even for serious development work. Enabled by:

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
CACHES = {
    'BACKEND': 'django.core.cache.backends.locmem.LocMemCache'
}
```

Key-value stores

You can use applications such as Memcached or Redis for external caching. These applications offer persistence and shared storage and are useful for small-scale deployments and/or development.

Memcached

Memcached is a high-performance and distributed memory object caching system providing in-memory key-value store for small chunks of arbitrary data.

Requirements:

- Memcached service running and accessible.
- Python module `python-memcached` installed.

Enabled by:

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
CACHES = {
    'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache'
    'LOCATION': 'my_memcached_host:11211',
}
```

Redis

Redis is an open source, BSD licensed, advanced key-value store. It is often referred to as a data structure server.

Requirements:

- Redis service running and accessible.
- Python modules `redis` and `django-redis` installed.

Enabled by:

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
CACHES = {
    "default": {
        "BACKEND": "redis_cache.cache.RedisCache",
        "LOCATION": "127.0.0.1:6379:1",
        "OPTIONS": {
            "CLIENT_CLASS": "redis_cache.client.DefaultClient",
        }
    }
}
```

Initialize and configure the database

Database-backed sessions are scalable, persistent, and can be made high-concurrency and highly-available.

However, 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.

1. Start the mysql command-line client:

   ```
   $ mysql -u root -p
   ```

2. Enter the MySQL root user's password when prompted.

3. To configure the MySQL database, create the dash database:

   ```
   mysql> CREATE DATABASE dash;
   ```

4. Create a MySQL user for the newly-created dash database that has full control of the database. Replace *DASH_DBPASS* with a password for the new user:

   ```
   mysql> GRANT ALL PRIVILEGES ON dash.* TO 'dash'@'%' IDENTIFIED BY
    'DASH_DBPASS';
   mysql> GRANT ALL PRIVILEGES ON dash.* TO 'dash'@'localhost' IDENTIFIED BY
    'DASH_DBPASS';
   ```

5. Enter quit at the `mysql>` prompt to exit MySQL.

6. In the `local_settings` file (on Fedora/RHEL/CentOS: `/etc/openstack-dashboard/local_settings`, on Ubuntu/Debian: `/etc/openstack-dashboard/local_settings.py` and on openSUSE: `/srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py`), change these options:

   ```
   SESSION_ENGINE = 'django.core.cache.backends.db.DatabaseCache'
   DATABASES = {
       'default': {
           # Database configuration here
           'ENGINE': 'django.db.backends.mysql',
           'NAME': 'dash',
           'USER': 'dash',
           'PASSWORD': 'DASH_DBPASS',
           'HOST': 'localhost',
           'default-character-set': 'utf8'
       }
   }
   ```

7. After configuring the `local_settings` as shown, you can run the **manage.py syncdb** command to populate this newly-created database.

```
$ /usr/share/openstack-dashboard/manage.py syncdb
```

Note on openSUSE the path is `/srv/www/openstack-dashboard/manage.py`.

As a result, the following output is returned:

```
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. On Ubuntu: If you want to avoid a warning when you restart apache2, create a blackhole directory in the dashboard directory, as follows:

```
# mkdir -p /var/lib/dash/.blackhole
```

9. Restart Apache to pick up the default site and symbolic link settings:

On Ubuntu:

```
# /etc/init.d/apache2 restart
```

On Fedora/RHEL/CentOS:

```
# service httpd restart
```

```
# service apache2 restart
```

On openSUSE:

```
# systemctl restart apache2.service
```

10. On Ubuntu, restart the `nova-api` service to ensure that the API server can connect to the dashboard without error:

```
# service nova-api restart
```

Cached database

To mitigate the performance issues of database queries, you can use the Django **cached_db** session back end, which utilizes both your 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 previously. Then, set the following value:

```
SESSION_ENGINE = "django.contrib.sessions.backends.cached_db"
```

Cookies

If you use Django 1.4 or later, the **signed_cookies** back end avoids server load and scaling problems.

This back end stores session data in a cookie, which is stored by the user's browser. The back end uses a cryptographic signing technique 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.

The pros of this engine are that it requires no additional dependencies or infrastructure overhead, and it scales indefinitely as long as the quantity of session data being stored fits into a normal cookie.

The biggest downside is that it places session data into storage on the user's machine and transports it over the wire. It also limits the quantity of session data that can be stored.

See the Django cookie-based sessions documentation.

Next steps

Your OpenStack environment now includes the dashboard. You can launch an instance or add more services to your environment in the following chapters.

9. Add the Block Storage service

Table of Contents

The OpenStack Block Storage service works through the interaction of a series of daemon processes named `cinder-*` that reside persistently on the host machine or machines. You can run the binaries from a single node or across multiple nodes. You can also run them on the same node as other OpenStack services. The following sections introduce Block Storage service components and concepts and show you how to configure and install the Block Storage service.

Block Storage

The Block Storage service enables management of volumes, volume snapshots, and volume types. It includes the following components:

- `cinder-api`: Accepts API requests and routes them to `cinder-volume` for action.
- `cinder-volume`: Responds to requests to read from and write to the Block Storage database to maintain state, interacting with other processes (like `cinder-scheduler`) through a message queue and directly upon block storage providing hardware or software. It can interact with a variety of storage providers through a driver architecture.
- `cinder-scheduler` daemon: Like the `nova-scheduler`, picks the optimal block storage provider node on which to create the volume.
- Messaging queue: Routes information between the Block Storage service processes.

The Block Storage service interacts with Compute to provide volumes for instances.

Configure a Block Storage service controller

Note

This scenario configures OpenStack Block Storage services on the *Controller node* and assumes that a second node provides storage through the `cinder-volume` service.

For instructions on how to configure the second node, see the section called "Configure a Block Storage service node" [79].

You can configure OpenStack to use various storage systems. This example uses LVM.

1. Install the appropriate packages for the Block Storage service:

```
# zypper install  openstack-cinder-api openstack-cinder-scheduler
```

2. Configure Block Storage to use your database.

   Run the following command to set `connection` option in the `[database]` section, which is in the `/etc/cinder/cinder.conf` file, replace *`CINDER_DBPASS`* with the password for the Block Storage database that you will create in a later step:

```
# openstack-config --set /etc/cinder/cinder.conf \
  database connection mysql://cinder:CINDER_DBPASS@controller/cinder
```

3. Use the password that you set to log in as root to create a `cinder` database:

```
# mysql -u root -p
mysql> CREATE DATABASE cinder;
mysql> GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'localhost' \
  IDENTIFIED BY 'CINDER_DBPASS';
mysql> GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'%' \
  IDENTIFIED BY 'CINDER_DBPASS';
```

4. Create a `cinder` user.

   The Block Storage service uses this user to authenticate with the Identity service.

   Use the `service` tenant and give the user the `admin` role:

```
$ keystone user-create --name=cinder --pass=CINDER_PASS --
email=cinder@example.com
$ keystone user-role-add --user=cinder --tenant=service --role=admin
```

5. Edit the `/etc/cinder/cinder.conf` configuration file:

```
# openstack-config --set /etc/cinder/cinder.conf DEFAULT \
  auth_strategy keystone
# openstack-config --set /etc/cinder/cinder.conf keystone_authtoken \
  auth_uri http://controller:5000
# openstack-config --set /etc/cinder/cinder.conf keystone_authtoken \
  auth_host controller
# openstack-config --set /etc/cinder/cinder.conf keystone_authtoken \
  auth_protocol http
# openstack-config --set /etc/cinder/cinder.conf keystone_authtoken \
  auth_port 35357
# openstack-config --set /etc/cinder/cinder.conf keystone_authtoken \
  admin_user cinder
# openstack-config --set /etc/cinder/cinder.conf keystone_authtoken \
  admin_tenant_name service
# openstack-config --set /etc/cinder/cinder.conf keystone_authtoken \
  admin_password CINDER_PASS
```

6. Configure Block Storage to use the RabbitMQ message broker.

   Replace *RABBIT_PASS* with the password you chose for RabbitMQ:

```
# openstack-config --set /etc/cinder/cinder.conf \
  DEFAULT rpc_backend cinder.openstack.common.rpc.impl_kombu
# openstack-config --set /etc/cinder/cinder.conf \
  DEFAULT rabbit_host controller
# openstack-config --set /etc/cinder/cinder.conf \
```

```
  DEFAULT rabbit_port 5672
# openstack-config --set /etc/cinder/cinder.conf \
  DEFAULT rabbit_password RABBIT_PASS
```

7. Register the Block Storage service with the Identity service so that other OpenStack services can locate it:

```
$ keystone service-create --name=cinder --type=volume --description=
"OpenStack Block Storage"
```

```
$ keystone endpoint-create \
  --service-id=$(keystone service-list | awk '/ volume / {print $2}') \
  --publicurl=http://controller:8776/v1/%\(tenant_id\)s \
  --internalurl=http://controller:8776/v1/%\(tenant_id\)s \
  --adminurl=http://controller:8776/v1/%\(tenant_id\)s
```

8. Register a service and endpoint for version 2 of the Block Storage service API:

```
$ keystone service-create --name=cinderv2 --type=volumev2 --description=
"OpenStack Block Storage v2"
```

```
$ keystone endpoint-create \
  --service-id=$(keystone service-list | awk '/ volumev2 / {print $2}') \
  --publicurl=http://controller:8776/v2/%\(tenant_id\)s \
  --internalurl=http://controller:8776/v2/%\(tenant_id\)s \
  --adminurl=http://controller:8776/v2/%\(tenant_id\)s
```

9. Start and configure the Block Storage services to start when the system boots:

```
# service openstack-cinder-api start
# service openstack-cinder-scheduler start
# chkconfig openstack-cinder-api on
# chkconfig openstack-cinder-scheduler on
```

Configure a Block Storage service node

After you configure the services on the controller node, configure a second system to be a Block Storage service node. This node contains the disk that serves volumes.

You can configure OpenStack to use various storage systems. This example uses LVM.

1. Use the instructions in Chapter 2, "Basic environment configuration" [6] to configure the system. Note the following differences from the installation instructions for the controller node:

   - Set the host name to `block1` and use `10.0.0.41` as IP address on the management network interface. Ensure that the IP addresses and host names for both controller node and Block Storage service node are listed in the `/etc/hosts` file on each system.

   - Follow the instructions in the section called "Network Time Protocol (NTP)" [17] to synchronize from the controller node.

2. Create the LVM physical and logical volumes. This guide assumes a second disk `/dev/sdb` that is used for this purpose:

```
# pvcreate /dev/sdb
```

```
# vgcreate cinder-volumes /dev/sdb
```

3. Add a filter entry to the `devices` section in the `/etc/lvm/lvm.conf` file to keep LVM from scanning devices used by virtual machines:

```
devices {
...
filter = [ "a/sda1/", "a/sdb/", "r/.*/"]
...
}
```

Note

You must add required physical volumes for LVM on the Block Storage host. Run the **pvdisplay** command to get a list or required volumes.

Each item in the filter array starts with either an `a` for accept, or an `r` for reject. The physical volumes that are required on the Block Storage host have names that begin with `a`. The array must end with "`r/.*/`" to reject any device not listed.

In this example, `/dev/sda1` is the volume where the volumes for the operating system for the node reside, while `/dev/sdb` is the volume reserved for `cinder-volumes`.

4. After you configure the operating system, install the appropriate packages for the Block Storage service:

```
# zypper install openstack-cinder-volume tgt
```

5. Copy the `/etc/cinder/cinder.conf` configuration file from the controller, or perform the following steps to set the keystone credentials:

```
# openstack-config --set /etc/cinder/cinder.conf DEFAULT \
  auth_strategy keystone
# openstack-config --set /etc/cinder/cinder.conf keystone_authtoken \
  auth_uri http://controller:5000
# openstack-config --set /etc/cinder/cinder.conf keystone_authtoken \
  auth_host controller
# openstack-config --set /etc/cinder/cinder.conf keystone_authtoken \
  auth_protocol http
# openstack-config --set /etc/cinder/cinder.conf keystone_authtoken \
  auth_port 35357
# openstack-config --set /etc/cinder/cinder.conf keystone_authtoken \
  admin_user cinder
# openstack-config --set /etc/cinder/cinder.conf keystone_authtoken \
  admin_tenant_name service
# openstack-config --set /etc/cinder/cinder.conf keystone_authtoken \
  admin_password CINDER_PASS
```

6. Configure Block Storage to use the RabbitMQ message broker. Replace *RABBIT_PASS* with the password you chose for RabbitMQ:

```
# openstack-config --set /etc/cinder/cinder.conf \
  DEFAULT rpc_backend cinder.openstack.common.rpc.impl_kombu
# openstack-config --set /etc/cinder/cinder.conf \
  DEFAULT rabbit_host controller
# openstack-config --set /etc/cinder/cinder.conf \
  DEFAULT rabbit_port 5672
```

```
# openstack-config --set /etc/cinder/cinder.conf \
  DEFAULT rabbit_password RABBIT_PASS
```

7. Configure Block Storage to use your MySQL database. Edit the `/etc/cinder/cinder.conf` file and add the following key to the `[database]` section. Replace *`CINDER_DBPASS`* with the password you chose for the Block Storage database:

```
# openstack-config --set /etc/cinder/cinder.conf \
  database connection mysql://cinder:CINDER_DBPASS@controller/cinder
```

8. Configure Block Storage to use the Image Service. Block Storage needs access to images to create bootable volumes. Edit the `/etc/cinder/cinder.conf` file and update the `glance_host` option in the `[DEFAULT]` section:

```
# openstack-config --set /etc/cinder/cinder.conf \
  DEFAULT glance_host controller
```

9. Start and configure the Block Storage services to start when the system boots:

```
# service openstack-cinder-volume start
# service tgtd start
# chkconfig openstack-cinder-volume on
# chkconfig tgtd on
```

Verify the Block Storage installation

To verify that the Block Storage is installed and configured properly, create a new volume.

For more information about how to manage volumes, see the *OpenStack User Guide*.

1. Source the `demo-openrc.sh` file:

```
$ source demo-openrc.sh
```

2. Use the **cinder create** command to create a new volume:

```
$ cinder create --display-name myVolume 1
+---------------------+--------------------------------------+
|       Property      |                Value                 |
+---------------------+--------------------------------------+
|     attachments     |                  []                  |
|  availability_zone  |                nova                  |
|       bootable      |               false                  |
|      created_at     |      2014-04-17T10:28:19.615050      |
| display_description |                None                  |
|     display_name    |              myVolume                |
|      encrypted      |               False                  |
|          id         | 5e691b7b-12e3-40b6-b714-7f17550db5d1 |
|       metadata      |                  {}                  |
|         size        |                  1                   |
|     snapshot_id     |                None                  |
|     source_volid    |                None                  |
|        status       |              creating                |
|     volume_type     |                None                  |
+---------------------+--------------------------------------+
```

3. Make sure that the volume has been correctly created with the **cinder list** command:

```
$ cinder list
+--------------------------------------+-----------+--------------+------
+-------------+----------+-------------+
|                  ID                  |   Status  | Display Name | Size |
 Volume Type | Bootable | Attached to |
+--------------------------------------+-----------+--------------+------
+-------------+----------+-------------+
| 5e691b7b-12e3-40b6-b714-7f17550db5d1 | available |   myVolume   |  1   |
     None    |  false   |             |
+--------------------------------------+-----------+--------------+------
+-------------+----------+-------------+
```

If the status value is not `available`, the volume creation failed. Check the log files in the `/var/log/cinder/` directory on the controller and volume nodes to get information about the failure.

Next steps

Your OpenStack environment now includes Block Storage. You can launch an instance or add more services to your environment in the following chapters.

10. Add Object Storage

Table of Contents

The OpenStack Object Storage services work together to provide object storage and retrieval through a REST API. For this example architecture, you must have already installed the Identity Service, also known as Keystone.

Object Storage service

The Object Storage service is a highly scalable and durable multi-tenant object storage system for large amounts of unstructured data at low cost through a RESTful HTTP API.

It includes the following components:

- Proxy servers (`swift-proxy-server`). Accepts Object Storage API and raw HTTP requests to upload files, modify metadata, and create containers. It also serves file or container listings to web browsers. To improve performance, the proxy server can use an optional cache usually deployed with memcache.

- Account servers (`swift-account-server`). Manage accounts defined with the Object Storage service.

- Container servers (`swift-container-server`). Manage a mapping of containers, or folders, within the Object Storage service.

- Object servers (`swift-object-server`). Manage actual objects, such as files, on the storage nodes.

- A number of periodic processes. Performs housekeeping tasks on the large data store. The replication services ensure consistency and availability through the cluster. Other periodic processes include auditors, updaters, and reapers.

- Configurable WSGI middleware that handles authentication. Usually the Identity Service.

System requirements for Object Storage

Hardware: OpenStack Object Storage is designed to run on commodity hardware.

Note

When you install only the Object Storage and Identity Service, you cannot use the dashboard unless you also install Compute and the Image Service.

Table 10.1. Hardware recommendations

Server	Recommended Hardware	Notes
Object Storage object servers	Processor: dual quad core Memory: 8 or 12 GB RAM Disk space: optimized for cost per GB Network: one 1 GB Network Interface Card (NIC)	The amount of disk space depends on how much you can fit into the rack efficiently. You want to optimize these for best cost per GB while still getting industry-standard failure rates. At Rackspace, our storage servers are currently running fairly generic 4U servers with 24 2T SATA drives and 8 cores of processing power. RAID on the storage drives is not required and not recommended. Swift's disk usage pattern is the worst case possible for RAID, and performance degrades very quickly using RAID 5 or 6. As an example, Rackspace runs Cloud Files storage servers with 24 2T SATA drives and 8 cores of processing power. Most services support either a worker or concurrency value in the settings. This allows the services to make effective use of the cores available.
Object Storage container/account servers	Processor: dual quad core Memory: 8 or 12 GB RAM Network: one 1 GB Network Interface Card (NIC)	Optimized for IOPS due to tracking with SQLite databases.
Object Storage proxy server	Processor: dual quad core Network: one 1 GB Network Interface Card (NIC)	Higher network throughput offers better performance for supporting many API requests. Optimize your proxy servers for best CPU performance. The Proxy Services are more CPU and network I/O intensive. If you are using 10 GB networking to the proxy, or are terminating SSL traffic at the proxy, greater CPU power is required.

Operating system: OpenStack Object Storage currently runs on Ubuntu, RHEL, CentOS, Fedora, openSUSE, or SLES.

Networking: 1 Gbps or 10 Gbps is suggested internally. For OpenStack Object Storage, an external network should connect the outside world to the proxy servers, and the storage network is intended to be isolated on a private network or multiple private networks.

Database: For OpenStack Object Storage, a SQLite database is part of the OpenStack Object Storage container and account management process.

Permissions: You can install OpenStack Object Storage either as root or as a user with sudo permissions if you configure the sudoers file to enable all the permissions.

Plan networking for Object Storage

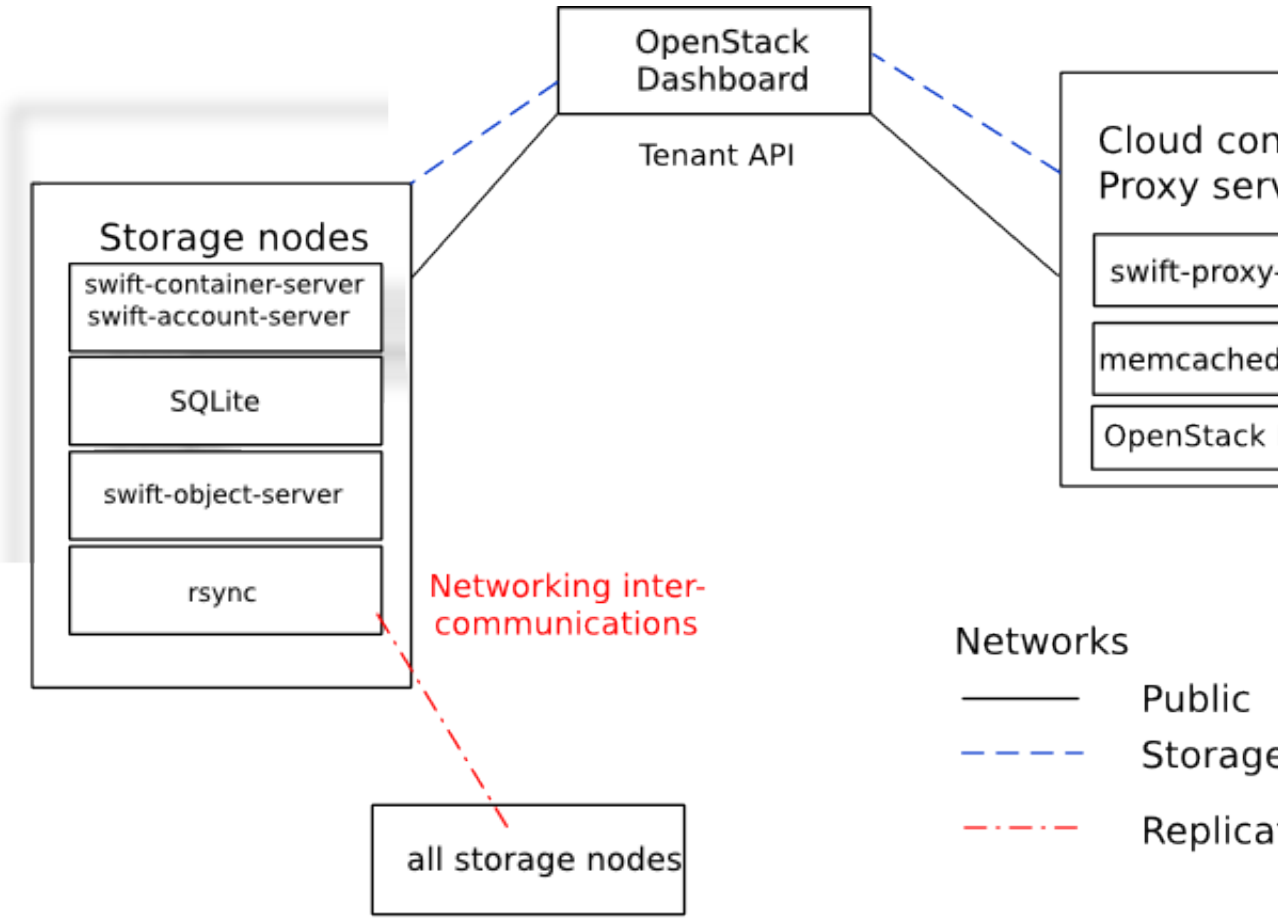
For both conserving network resources and ensuring that network administrators understand the needs for networks and public IP addresses for providing access to the

APIs and storage network as necessary, this section offers recommendations and required minimum sizes. Throughput of at least 1000 Mbps is suggested.

This guide describes the following networks:

- A mandatory public network. Connects to the proxy server.
- A mandatory storage network. Not accessible from outside the cluster. All nodes connect to this network.
- An optional replication network. Not accessible from outside the cluster. Dedicated to replication traffic among storage nodes. Must be configured in the Ring.

This figure shows the basic architecture for the public network, the storage network, and the optional replication network.



By default, all of the OpenStack Object Storage services, as well as the rsync daemon on the storage nodes, are configured to listen on their STORAGE_LOCAL_NET IP addresses.

If you configure a replication network in the Ring, the Account, Container and Object servers listen on both the STORAGE_LOCAL_NET and STORAGE_REPLICATION_NET IP addresses. The rsync daemon only listens on the STORAGE_REPLICATION_NET IP address.

Public Network (Publicly routable IP range)	Provides public IP accessibility to the API endpoints within the cloud infrastructure.

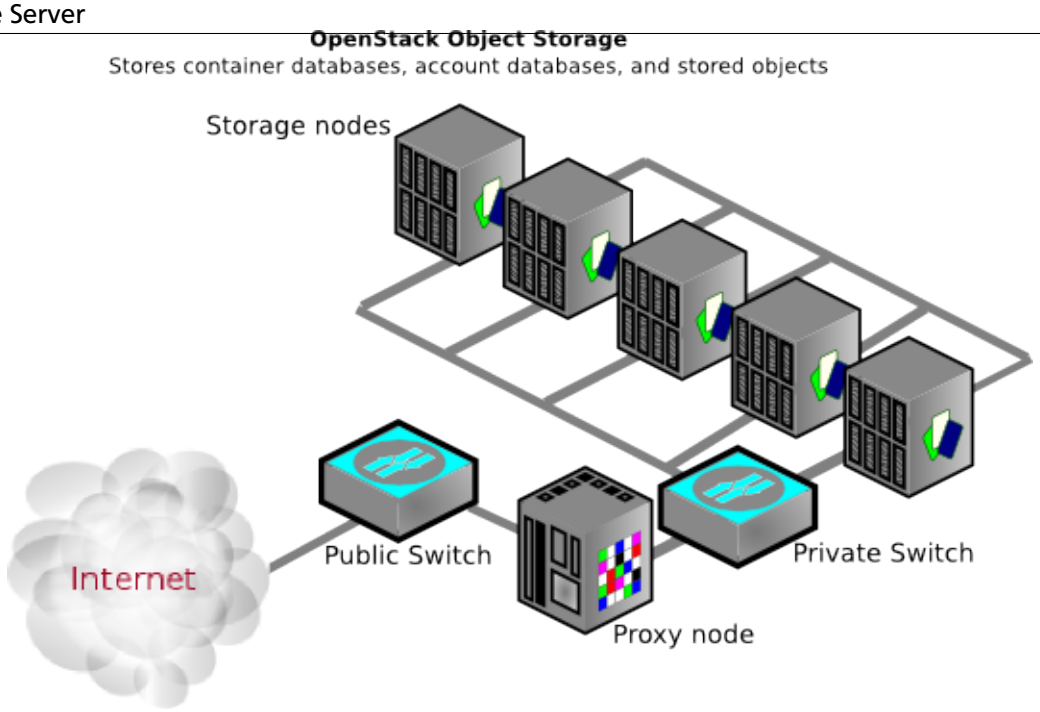
	Minimum size: one IP address for each proxy server.
Storage Network (RFC1918 IP Range, not publicly routable)	Manages all inter-server communications within the Object Storage infrastructure. Minimum size: one IP address for each storage node and proxy server. Recommended size: as above, with room for expansion to the largest your cluster size. For example, 255 or CIDR /24.
Replication Network (RFC1918 IP Range, not publicly routable)	Manages replication-related communications among storage servers within the Object Storage infrastructure. Recommended size: as for `STORAGE_LOCAL_NET`.

Example of Object Storage installation architecture

- Node: A host machine that runs one or more OpenStack Object Storage services.
- Proxy node: Runs proxy services.
- Storage node: Runs account, container, and object services. Contains the SQLite databases.
- Ring: A set of mappings between OpenStack Object Storage data to physical devices.
- Replica: A copy of an object. By default, three copies are maintained in the cluster.
- Zone: A logically separate section of the cluster, related to independent failure characteristics.
- Region (optional): A logically separate section of the cluster, representing distinct physical locations such as cities or countries. Similar to zones but representing physical locations of portions of the cluster rather than logical segments.

To increase reliability and performance, you can add additional proxy servers.

This document describes each storage node as a separate zone in the ring. At a minimum, five zones are recommended. A zone is a group of nodes that are as isolated as possible from other nodes (separate servers, network, power, even geography). The ring guarantees that every replica is stored in a separate zone. This diagram shows one possible configuration for a minimal installation:



Install Object Storage

Though you can install OpenStack Object Storage for development or testing purposes on one server, a multiple-server installation enables the high availability and redundancy you want in a production distributed object storage system.

To perform a single-node installation for development purposes from source code, use the Swift All In One instructions (Ubuntu) or DevStack (multiple distros). See http://swift.openstack.org/development_saio.html for manual instructions or http://devstack.org for all-in-one including authentication with the Identity Service (keystone).

Before you begin

Have a copy of the operating system installation media available if you are installing on a new server.

These steps assume you have set up repositories for packages for your operating system as shown in OpenStack Packages.

This document demonstrates how to install a cluster by using the following types of nodes:

- One proxy node which runs the swift-proxy-server processes. The proxy server proxies requests to the appropriate storage nodes.
- Five storage nodes that run the swift-account-server, swift-container-server, and swift-object-server processes which control storage of the account databases, the container databases, as well as the actual stored objects.

Note

Fewer storage nodes can be used initially, but a minimum of five is recommended for a production cluster.

General installation steps

1. Create a `swift` user that the Object Storage Service can use to authenticate with the Identity Service. Choose a password and specify an email address for the `swift` user. Use the `service` tenant and give the user the `admin` role:

```
$ keystone user-create --name=swift --pass=SWIFT_PASS \
  --email=swift@example.com
$ keystone user-role-add --user=swift --tenant=service --role=admin
```

2. Create a service entry for the Object Storage Service:

```
$ keystone service-create --name=swift --type=object-store \
  --description="OpenStack Object Storage"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |     OpenStack Object Storage     |
|      id     | eede9296683e4b5ebfa13f5166375ef6 |
|     name    |              swift               |
|     type    |           object-store           |
+-------------+----------------------------------+
```

Note

The service ID is randomly generated and is different from the one shown here.

3. Specify an API endpoint for the Object Storage Service by using the returned service ID. When you specify an endpoint, you provide URLs for the public API, internal API, and admin API. In this guide, the `controller` host name is used:

```
$ keystone endpoint-create \
  --service-id=$(keystone service-list | awk '/ object-store / {print
 $2}') \
  --publicurl='http://controller:8080/v1/AUTH_%(tenant_id)s' \
  --internalurl='http://controller:8080/v1/AUTH_%(tenant_id)s' \
  --adminurl=http://controller:8080
+-------------+-----------------------------------------------------+
|   Property  |                        Value                        |
+-------------+-----------------------------------------------------+
|   adminurl  |            http://controller:8080/                  |
|      id     |          9e3ce428f82b40d38922f242c095982e           |
| internalurl | http://controller:8080/v1/AUTH_%(tenant_id)s        |
|  publicurl  | http://controller:8080/v1/AUTH_%(tenant_id)s        |
|    region   |                      regionOne                      |
|  service_id |          eede9296683e4b5ebfa13f5166375ef6           |
+-------------+-----------------------------------------------------+
```

4. Create the configuration directory on all nodes:

```
# mkdir -p /etc/swift
```

5. Create `/etc/swift/swift.conf` on all nodes:

```
[swift-hash]
# random unique string that can never change (DO NOT LOSE)
swift_hash_path_suffix = fLIbertYgibbitZ
```

Note

The suffix value in `/etc/swift/swift.conf` should be set to some random string of text to be used as a salt when hashing to determine mappings in the ring. This file must be the same on every node in the cluster!

Next, set up your storage nodes and proxy node. This example uses the Identity Service for the common authentication piece.

Install and configure storage nodes

Note

Object Storage works on any file system that supports Extended Attributes (XATTRS). XFS shows the best overall performance for the swift use case after considerable testing and benchmarking at Rackspace. It is also the only file system that has been thoroughly tested. See the *OpenStack Configuration Reference* for additional recommendations.

1. Install storage node packages:

```
# zypper install openstack-swift-account openstack-swift-container \
  openstack-swift-object python-xml xfsprogs xinetd
```

2. For each device on the node that you want to use for storage, set up the XFS volume (`/dev/sdb` is used as an example). Use a single partition per drive. For example, in a server with 12 disks you may use one or two disks for the operating system which should not be touched in this step. The other 10 or 11 disks should be partitioned with a single partition, then formatted in XFS.

```
# fdisk /dev/sdb
# mkfs.xfs /dev/sdb1
# echo "/dev/sdb1 /srv/node/sdb1 xfs noatime,nodiratime,nobarrier,logbufs=
8 0 0" >> /etc/fstab
# mkdir -p /srv/node/sdb1
# mount /srv/node/sdb1
# chown -R swift:swift /srv/node
```

3. Replace the content of `/etc/rsyncd.conf` with:

```
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = STORAGE_LOCAL_NET_IP

[account]
max connections = 2
path = /srv/node/
```

```
read only = false
lock file = /var/lock/account.lock

[container]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/container.lock

[object]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/object.lock
```

4. (Optional) If you want to separate rsync and replication traffic to replication network, set STORAGE_REPLICATION_NET_IP instead of STORAGE_LOCAL_NET_IP:

```
address = STORAGE_REPLICATION_NET_IP
```

5. Edit the following line in /etc/xinetd.d/rsync:

```
disable = false
```

6. Start the xinetd service and configure it to start when the system boots:

```
# service xinetd start
# chkconfig xinetd on
```

Note

The rsync service requires no authentication, so run it on a local, private network.

7. Create the swift recon cache directory and set its permissions:

```
# mkdir -p /var/swift/recon
# chown -R swift:swift /var/swift/recon
```

Install and configure the proxy node

The proxy server takes each request and looks up locations for the account, container, or object and routes the requests correctly. The proxy server also handles API requests. You enable account management by configuring it in the /etc/swift/proxy-server.conf file.

Note

The Object Storage processes run under a separate user and group, set by configuration options, and referred to as swift:swift. The default user is swift.

1. Install swift-proxy service:

```
# zypper install openstack-swift-proxy memcached python-swiftclient
 python-keystoneclient python-xml
```

2. Modify memcached to listen on the default interface on a local, non-public network. Edit the `/etc/sysconfig/memcached` file:

```
MEMCACHED_PARAMS="-l PROXY_LOCAL_NET_IP"
```

3. Start the memcached service and configure it to start when the system boots:

```
# service memcached start
# chkconfig memcached on
```

4. Edit `/etc/swift/proxy-server.conf`:

```
[DEFAULT]
bind_port = 8080
user = swift

[pipeline:main]
pipeline = healthcheck cache authtoken keystoneauth proxy-server

[app:proxy-server]
use = egg:swift#proxy
allow_account_management = true
account_autocreate = true

[filter:keystoneauth]
use = egg:swift#keystoneauth
operator_roles = Member,admin,swiftoperator

[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory

# Delaying the auth decision is required to support token-less
# usage for anonymous referrers ('.r:*').
delay_auth_decision = true

# cache directory for signing certificate
signing_dir = /home/swift/keystone-signing

# auth_* settings refer to the Keystone server
auth_protocol = http
auth_host = controller
auth_port = 35357

# the service tenant and swift username and password created in Keystone
admin_tenant_name = service
admin_user = swift
admin_password = SWIFT_PASS

[filter:cache]
use = egg:swift#memcache

[filter:catch_errors]
use = egg:swift#catch_errors

[filter:healthcheck]
use = egg:swift#healthcheck
```

Note

If you run multiple memcache servers, put the multiple IP:port listings in the [filter:cache] section of the `/etc/swift/proxy-server.conf` file:

```
10.1.2.3:11211,10.1.2.4:11211
```

Only the proxy server uses memcache.

5. Create the account, container, and object rings. The builder command creates a builder file with a few parameters. The parameter with the value of 18 represents 2 ^ 18th, the value that the partition is sized to. Set this "partition power" value based on the total amount of storage you expect your entire ring to use. The value 3 represents the number of replicas of each object, with the last value being the number of hours to restrict moving a partition more than once.

```
# cd /etc/swift
# swift-ring-builder account.builder create 18 3 1
# swift-ring-builder container.builder create 18 3 1
# swift-ring-builder object.builder create 18 3 1
```

6. For every storage device on each node add entries to each ring:

```
# swift-ring-builder account.builder add
 zZONE-STORAGE_LOCAL_NET_IP:6002[RSTORAGE_REPLICATION_NET_IP:6005]/DEVICE
 100
# swift-ring-builder container.builder add
 zZONE-STORAGE_LOCAL_NET_IP_1:6001[RSTORAGE_REPLICATION_NET_IP:6004]/DEVICE
 100
# swift-ring-builder object.builder add
 zZONE-STORAGE_LOCAL_NET_IP_1:6000[RSTORAGE_REPLICATION_NET_IP:6003]/DEVICE
 100
```

Note

You must omit the optional *STORAGE_REPLICATION_NET_IP* parameter if you do not want to use dedicated network for replication.

For example, if a storage node has a partition in Zone 1 on IP 10.0.0.1, the storage node has address 10.0.1.1 from replication network. The mount point of this partition is `/srv/node/sdb1`, and the path in `/etc/rsyncd.conf` is `/srv/node/`, the DEVICE would be sdb1 and the commands are:

```
# swift-ring-builder account.builder add z1-10.0.0.1:6002R10.0.1.1:6005/
sdb1 100
# swift-ring-builder container.builder add z1-10.0.0.1:6001R10.0.1.1:6004/
sdb1 100
# swift-ring-builder object.builder add z1-10.0.0.1:6000R10.0.1.1:6003/
sdb1 100
```

Note

If you assume five zones with one node for each zone, start ZONE at 1. For each additional node, increment ZONE by 1.

7. Verify the ring contents for each ring:

```
# swift-ring-builder account.builder
# swift-ring-builder container.builder
# swift-ring-builder object.builder
```

8. Rebalance the rings:

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

Note

Rebalancing rings can take some time.

9. Copy the `account.ring.gz`, `container.ring.gz`, and `object.ring.gz` files to each of the Proxy and Storage nodes in `/etc/swift`.

10. Make sure the swift user owns all configuration files:

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

11. Start the Proxy service and configure it to start when the system boots:

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

Start services on the storage nodes

Now that the ring files are on each storage node, you can start the services. On each storage node, run the following command:

```
# for service in \
  openstack-swift-object openstack-swift-object-replicator openstack-swift-
object-updater openstack-swift-object-auditor \
  openstack-swift-container openstack-swift-container-replicator openstack-
swift-container-updater openstack-swift-container-auditor \
  openstack-swift-account openstack-swift-account-replicator openstack-swift-
account-reaper openstack-swift-account-auditor; do \
    service $service start; chkconfig $service on; done
```

Note

To start all swift services at once, run the command:

```
# swift-init all start
```

To know more about `swift-init` command, run:

```
$ man swift-init
```

Verify the installation

You can run these commands from the proxy server or any server that has access to the Identity Service.

1. Make sure that your credentials are set up correctly in the `admin-openrc.sh` file and source it:

```
$ source admin-openrc.sh
```

2. Run the following **swift** command:

```
$ swift stat
Account: AUTH_11b9758b7049476d9b48f7a91ea11493
Containers: 0
   Objects: 0
     Bytes: 0
Content-Type: text/plain; charset=utf-8
X-Timestamp: 1381434243.83760
X-Trans-Id: txdcdd594565214fb4a2d33-0052570383
X-Put-Timestamp: 1381434243.83760
```

3. Run the following **swift** commands to upload files to a container. Create the `test.txt` and `test2.txt` test files locally if needed.

```
$ swift upload myfiles test.txt
$ swift upload myfiles test2.txt
```

4. Run the following **swift** command to download all files from the `myfiles` container:

```
$ swift download myfiles
test2.txt [headers 0.267s, total 0.267s, 0.000s MB/s]
test.txt [headers 0.271s, total 0.271s, 0.000s MB/s]
```

Add another proxy server

To provide additional reliability and bandwidth to your cluster, you can add proxy servers. You can set up an additional proxy node the same way that you set up the first proxy node but with additional configuration steps.

After you have more than two proxies, you must load balance them; your storage endpoint (what clients use to connect to your storage) also changes. You can select from different strategies for load balancing. For example, you could use round-robin DNS, or a software or hardware load balancer (like pound) in front of the two proxies. You can then point your storage URL to the load balancer, configure an initial proxy node and complete these steps to add proxy servers.

1. Update the list of memcache servers in the `/etc/swift/proxy-server.conf` file for added proxy servers. If you run multiple memcache servers, use this pattern for the multiple IP:port listings in each proxy server configuration file:

```
10.1.2.3:11211,10.1.2.4:11211
```

```
[filter:cache]
use = egg:swift#memcache
memcache_servers = PROXY_LOCAL_NET_IP:11211
```

2. Copy ring information to all nodes, including new proxy nodes. Also, ensure that the ring information gets to all storage nodes.

3. After you sync all nodes, make sure that the admin has keys in `/etc/swift` and the ownership for the ring file is correct.

Next steps

Your OpenStack environment now includes Object Storage. You can launch an instance or add more services to your environment in the following chapters.

11. Add the Orchestration service

Table of Contents

Use the Orchestration module to create cloud resources using a template language called HOT. The integrated project name is Heat.

Orchestration service overview

The Orchestration service provides a template-based orchestration for describing a cloud application by running OpenStack API calls to generate running cloud applications. The software integrates other core components of OpenStack into a one-file template system. The templates enable you to create most OpenStack resource types, such as instances, floating IPs, volumes, security groups, users, and so on. Also, provides some more advanced functionality, such as instance high availability, instance auto-scaling, and nested stacks. By providing very tight integration with other OpenStack core projects, all OpenStack core projects could receive a larger user base.

The service enables deployers to integrate with the Orchestration service directly or through custom plug-ins.

The Orchestration service consists of the following components:

- `heat` command-line client. A CLI that communicates with the heat-api to run AWS CloudFormation APIs. End developers could also use the Orchestration REST API directly.
- `heat-api` component. Provides an OpenStack-native REST API that processes API requests by sending them to the heat-engine over RPC.
- `heat-api-cfn` component. Provides an AWS Query API that is compatible with AWS CloudFormation and processes API requests by sending them to the heat-engine over RPC.
- `heat-engine`. Orchestrates the launching of templates and provides events back to the API consumer.

Install the Orchestration service

1. Install the Orchestration module on the controller node:

```
# zypper install openstack-heat-api openstack-heat-api-cfn \
  openstack-heat-engine
```

2. In the configuration file, specify the location of the database where the Orchestration service stores data. These examples use a MySQL database with a `heat` user on the controller node. Replace *HEAT_DBPASS* with the password for the database user:

```
# openstack-config --set /etc/heat/heat.conf \
  database connection mysql://heat:HEAT_DBPASS@controller/heat
```

3. Use the password that you set previously to log in as `root` and create a `heat` database user:

```
$ mysql -u root -p
mysql> CREATE DATABASE heat;
mysql> GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'localhost' \
IDENTIFIED BY 'HEAT_DBPASS';
mysql> GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'%' \
IDENTIFIED BY 'HEAT_DBPASS';
```

4. Configure the Orchestration Service to use the RabbitMQ message broker.

   Run the following commands:

```
# openstack-config --set /etc/heat/heat.conf DEFAULT
 rabbit_host controller
# openstack-config --set /etc/heat/heat.conf DEFAULT
 rabbit_password RABBIT_PASS
```

5. Create a `heat` user that the Orchestration service can use to authenticate with the Identity Service. Use the `service` tenant and give the user the `admin` role:

```
$ keystone user-create --name=heat --pass=HEAT_PASS \
  --email=heat@example.com
$ keystone user-role-add --user=heat --tenant=service --role=admin
```

6. Edit the `/etc/heat/heat.conf` file to change the `[keystone_authtoken]` and `[ec2authtoken]` sections to add credentials to the Orchestration Service:

```
[keystone_authtoken]
auth_host = controller
auth_port = 35357
auth_protocol = http
auth_uri = http://controller:5000/v2.0
admin_tenant_name = service
admin_user = heat
admin_password = HEAT_PASS

[ec2authtoken]
auth_uri = http://controller:5000/v2.0
```

7. Register the Heat and CloudFormation APIs with the Identity Service so that other OpenStack services can locate these APIs. Register the services and specify the endpoints:

```
$ keystone service-create --name=heat --type=orchestration \
  --description="Orchestration"
$ keystone endpoint-create \
  --service-id=$(keystone service-list | awk '/ orchestration / {print
 $2}') \
  --publicurl=http://controller:8004/v1/%\(tenant_id\)s \
  --internalurl=http://controller:8004/v1/%\(tenant_id\)s \
```

```
  --adminurl=http://controller:8004/v1/%\(tenant_id\)s
$ keystone service-create --name=heat-cfn --type=cloudformation \
  --description="Orchestration CloudFormation"
$ keystone endpoint-create \
  --service-id=$(keystone service-list | awk '/ cloudformation / {print
 $2}') \
  --publicurl=http://controller:8000/v1 \
  --internalurl=http://controller:8000/v1 \
  --adminurl=http://controller:8000/v1
```

8. Start the `heat-api`, `heat-api-cfn` and `heat-engine` services and configure them to start when the system boots:

```
# service openstack-heat-api start
# service openstack-heat-api-cfn start
# service openstack-heat-engine start
# chkconfig openstack-heat-api on
# chkconfig openstack-heat-api-cfn on
# chkconfig openstack-heat-engine on
```

Verify the Orchestration service installation

To verify that the Orchestration service is installed and configured correctly, make sure that your credentials are set up correctly in the `demo-openrc.sh` file. Source the file, as follows:

```
$ source demo-openrc.sh
```

The Orchestration Module uses templates to describe stacks. To learn about the template languages, see the Template Guide in the Heat developer documentation.

Create a test template in the `test-stack.yml` file with the following content:

```
heat_template_version: 2013-05-23

description: Test Template

parameters:
  ImageID:
    type: string
    description: Image use to boot a server
  NetID:
    type: string
    description: Network ID for the server

resources:
  server1:
    type: OS::Nova::Server
    properties:
      name: "Test server"
      image: { get_param: ImageID }
      flavor: "m1.tiny"
      networks:
      - network: { get_param: NetID }

outputs:
  server1_private_ip:
    description: IP address of the server in the private network
```

```
    value: { get_attr: [ server1, first_address ] }
```

Use the **heat stack-create** command to create a stack from this template:

```
$ NET_ID=$(nova net-list | awk '/ demo-net / { print $2 }')
$ heat stack-create -f test-stack.yml \
  -P "ImageID=cirros-0.3.2-x86_64;NetID=$NET_ID" testStack
+--------------------------------------+------------+--------------------
+----------------------+
| id                                   | stack_name | stack_status       |
 creation_time        |
+--------------------------------------+------------+--------------------
+----------------------+
| 477d96b4-d547-4069-938d-32ee990834af | testStack  | CREATE_IN_PROGRESS |
 2014-04-06T15:11:01Z |
+--------------------------------------+------------+--------------------
+----------------------+
```

Verify that the stack was created successfully with the **heat stack-list** command:

```
$ heat stack-list
+--------------------------------------+------------+-----------------
+----------------------+
| id                                   | stack_name | stack_status    |
 creation_time        |
+--------------------------------------+------------+-----------------
+----------------------+
| 477d96b4-d547-4069-938d-32ee990834af | testStack  | CREATE_COMPLETE |
 2014-04-06T15:11:01Z |
+--------------------------------------+------------+-----------------
+----------------------+
```

Next steps

Your OpenStack environment now includes Orchestration. You can launch an instance or add more services to your environment in the following chapters.

12. Add the Telemetry module

Table of Contents

Telemetry provides a framework for monitoring and metering the OpenStack cloud. It is also known as the Ceilometer project.

Telemetry

The Telemetry module:

- Efficiently collects the metering data about the CPU and network costs.
- Collects data by monitoring notifications sent from services or by polling the infrastructure.
- Configures the type of collected data to meet various operating requirements. Accessing and inserting the metering data through the REST API.
- Expands the framework to collect custom usage data by additional plug-ins.
- Produces signed metering messages that cannot be repudiated.

The system consists of the following basic components:

- A compute agent (`ceilometer-agent-compute`). Runs on each compute node and polls for resource utilization statistics. There may be other types of agents in the future, but for now we will focus on creating the compute agent.
- A central agent (`ceilometer-agent-central`). Runs on a central management server to poll for resource utilization statistics for resources not tied to instances or compute nodes.
- A collector (`ceilometer-collector`). Runs on one or more central management servers to monitor the message queues (for notifications and for metering data coming from the agent). Notification messages are processed and turned into metering messages and sent back out onto the message bus using the appropriate topic. Telemetry messages are written to the data store without modification.

- An alarm notifier (`ceilometer-alarm-notifier`). Runs on one or more central management servers to allow settting alarms based on threshold evaluation for a collection of samples.

- A data store. A database capable of handling concurrent writes (from one or more collector instances) and reads (from the API server).

- An API server (`ceilometer-api`). Runs on one or more central management servers to provide access to the data from the data store.

These services communicate by using the standard OpenStack messaging bus. Only the collector and API server have access to the data store.

Install the Telemetry module

Telemetry provides an API service that provides a collector and a range of disparate agents. Before you can install these agents on nodes such as the compute node, you must use this procedure to install the core components on the controller node.

1. Install the Telemetry service on the controller node:

```
# zypper install openstack-ceilometer-api openstack-ceilometer-collector \
  openstack-ceilometer-agent-notification openstack-ceilometer-agent-
central python-ceilometerclient \
  openstack-ceilometer-alarm-evaluator openstack-ceilometer-alarm-notifier
```

2. The Telemetry service uses a database to store information. Specify the location of the database in the configuration file. The examples use a MongoDB database on the controller node:

```
# zypper install mongodb
```

Note

By default MongoDB is configured to create several 1 GB files in the `/var/lib/mongodb/journal/` directory to support database journaling.

If you need to minimize the space allocated to support database journaling then set the `smallfiles` configuration key to `true` in the `/etc/mongodb.conf` configuration file. This configuration reduces the size of each journaling file to 512 MB.

For more information on the `smallfiles` configuration key refer to the MongoDB documentation at http://docs.mongodb.org/manual/reference/configuration-options/#smallfiles.

For instructions detailing the steps to disable database journaling entirely refer to http://docs.mongodb.org/manual/tutorial/manage-journaling/.

3. Configure MongoDB to make it listen on the controller management IP address. Edit the `/etc/mongodb.conf` file and modify the `bind_ip` key:

```
bind_ip = 10.0.0.11
```

4. Start the MongoDB server and configure it to start when the system boots:

```
# service mongodb start
# chkconfig mongodb on
```

5. Create the database and a `ceilometer` database user:

```
# mongo --host controller --eval '
db = db.getSiblingDB("ceilometer");
db.addUser({user: "ceilometer",
            pwd: "CEILOMETER_DBPASS",
            roles: [ "readWrite", "dbAdmin" ]})'
```

6. Configure the Telemetry service to use the database:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  database connection mongodb://
ceilometer:CEILOMETER_DBPASS@controller:27017/ceilometer
```

7. You must define a secret key that is used as a shared secret among Telemetry service nodes. Use **openssl** to generate a random token and store it in the configuration file:

```
# CEILOMETER_TOKEN=$(openssl rand -hex 10)
# echo $CEILOMETER_TOKEN
# openstack-config --set /etc/ceilometer/ceilometer.conf publisher
 metering_secret $CEILOMETER_TOKEN
```

For SUSE Linux Enterprise, run the following command:

```
# CEILOMETER_TOKEN=$(openssl rand 10|hexdump  -e '1/1 "%.2x"')
```

8. Configure the RabbitMQ access:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf DEFAULT
 rabbit_host controller
# openstack-config --set /etc/ceilometer/ceilometer.conf DEFAULT
 rabbit_password RABBIT_PASS
```

9. Configure collector dispatcher:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  collector dispatcher database
```

10. Create a `ceilometer` user that the Telemetry service uses to authenticate with the Identity Service. Use the `service` tenant and give the user the `admin` role:

```
$ keystone user-create --name=ceilometer --pass=CEILOMETER_PASS --
email=ceilometer@example.com
$ keystone user-role-add --user=ceilometer --tenant=service --role=admin
```

11. Add the credentials to the configuration files for the Telemetry service:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  keystone_authtoken auth_host controller
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  keystone_authtoken admin_user ceilometer
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  keystone_authtoken admin_tenant_name service
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  keystone_authtoken auth_protocol http
```

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  keystone_authtoken auth_uri http://controller:5000
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  keystone_authtoken admin_password CEILOMETER_PASS
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  service_credentials os_auth_url http://controller:5000/v2.0
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  service_credentials os_username ceilometer
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  service_credentials os_tenant_name service
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  service_credentials os_password CEILOMETER_PASS
```

12. Register the Telemetry service with the Identity Service so that other OpenStack services can locate it. Use the **keystone** command to register the service and specify the endpoint:

```
$ keystone service-create --name=ceilometer --type=metering \
  --description="Telemetry"
$ keystone endpoint-create \
  --service-id=$(keystone service-list | awk '/ metering / {print $2}') \
  --publicurl=http://controller:8777 \
  --internalurl=http://controller:8777 \
  --adminurl=http://controller:8777
```

13. Start the `openstack-ceilometer-api`, `openstack-ceilometer-agent-central`, `openstack-ceilometer-collector`, `openstack-ceilometer-alarm-evaluator`, and `openstack-ceilometer-alarm-notifier` services and configure them to start when the system boots:

```
# service openstack-ceilometer-api start
# service openstack-ceilometer-agent-notification start
# service openstack-ceilometer-agent-central start
# service openstack-ceilometer-collector start
# service openstack-ceilometer-alarm-evaluator start
# service openstack-ceilometer-alarm-notifier start
# chkconfig openstack-ceilometer-api on
# chkconfig openstack-ceilometer-agent-notification on
# chkconfig openstack-ceilometer-agent-central on
# chkconfig openstack-ceilometer-collector on
# chkconfig openstack-ceilometer-alarm-evaluator on
# chkconfig openstack-ceilometer-alarm-notifier on
```

Install the Compute agent for Telemetry

Telemetry provides an API service that provides a collector and a range of disparate agents. This procedure details how to install the agent that runs on the compute node.

1. Install the Telemetry service on the compute node:

```
# zypper install openstack-ceilometer-agent-compute
```

2. Set the following options in the `/etc/nova/nova.conf` file:

```
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  instance_usage_audit True
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  instance_usage_audit_period hour
```

```
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  notify_on_state_change vm_and_task_state
```

Note

The `notification_driver` option is a multi valued option, which openstack-config cannot set properly. See the section called "OpenStack packages" [19].

Edit the `/etc/nova/nova.conf` file and add the following lines to the `[DEFAULT]` section:

```
[DEFAULT]
...
notification_driver = nova.openstack.common.notifier.rpc_notifier
notification_driver = ceilometer.compute.nova_notifier
```

3. Restart the Compute service:

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

4. You must set the secret key that you defined previously. The Telemetry service nodes share this key as a shared secret:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf publisher \
  metering_secret CEILOMETER_TOKEN
```

5. Configure the RabbitMQ access:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf DEFAULT
 rabbit_host controller
# openstack-config --set /etc/ceilometer/ceilometer.conf DEFAULT
 rabbit_password RABBIT_PASS
```

6. Add the Identity service credentials:

```
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  keystone_authtoken auth_host controller
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  keystone_authtoken admin_user ceilometer
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  keystone_authtoken admin_tenant_name service
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  keystone_authtoken auth_protocol http
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  keystone_authtoken admin_password CEILOMETER_PASS
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  service_credentials os_username ceilometer
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  service_credentials os_tenant_name service
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  service_credentials os_password CEILOMETER_PASS
# openstack-config --set /etc/ceilometer/ceilometer.conf \
  service_credentials os_auth_url http://controller:5000/v2.0
```

7. Start the service and configure it to start when the system boots:

```
# service openstack-ceilometer-agent-compute start
# chkconfig openstack-ceilometer-agent-compute on
```

Configure the Image Service for Telemetry

1. To retrieve image samples, you must configure the Image Service to send notifications to the bus.

   Run the following commands:

```
# openstack-config --set /etc/glance/glance-api.conf DEFAULT
 notification_driver messaging
# openstack-config --set /etc/glance/glance-api.conf DEFAULT rpc_backend
 rabbit
# openstack-config --set /etc/glance/glance-api.conf DEFAULT
 rabbit_host controller
# openstack-config --set /etc/glance/glance-api.conf DEFAULT
 rabbit_password RABBIT_PASS
```

2. Restart the Image Services with their new settings:

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

Add the Block Storage service agent for Telemetry

1. To retrieve volume samples, you must configure the Block Storage service to send notifications to the bus.

   Run the following commands on the controller and volume nodes:

```
# openstack-config --set /etc/cinder/cinder.conf DEFAULT control_exchange
 cinder
# openstack-config --set /etc/cinder/cinder.conf DEFAULT
 notification_driver cinder.openstack.common.notifier.rpc_notifier
```

2. Restart the Block Storage services with their new settings.

   On the controller node:

```
# service openstack-cinder-api restart
# service openstack-cinder-scheduler restart
```

   On the volume node:

```
# service openstack-cinder-volume restart
```

Configure the Object Storage service for Telemetry

1. To retrieve object store statistics, the Telemetry service needs access to Object Storage with the `ResellerAdmin` role. Give this role to your `os_username` user for the `os_tenant_name` tenant:

```
$ keystone role-create --name=ResellerAdmin
+----------+----------------------------------+
| Property |              Value               |
```

```
+----------+----------------------------------+
|    id    | 462fa46c13fd4798a95a3bfbe27b5e54 |
|   name   |          ResellerAdmin           |
+----------+----------------------------------+
```

```
$ keystone user-role-add --tenant service --user ceilometer \
      --role 462fa46c13fd4798a95a3bfbe27b5e54
```

2. You must also add the Telemetry middleware to Object Storage to handle incoming and outgoing traffic. Add these lines to the `/etc/swift/proxy-server.conf` file:

```
[filter:ceilometer]
use = egg:ceilometer#swift
```

3. Add `ceilometer` to the `pipeline` parameter of that same file:

```
[pipeline:main]
pipeline = healthcheck cache authtoken keystoneauth ceilometer proxy-
server
```

4. Restart the service with its new settings:

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

Verify the Telemetry installation

To test the Telemetry installation, download an image from the Image Service, and use the **ceilometer** command to display usage statistics.

1. Use the `ceilometer meter-list` command to test the access to Telemetry:

```
$ ceilometer meter-list
```

```
+------------+-------+-------+--------------------------------------
+---------+----------------------------------+
| Name       | Type  | Unit  | Resource ID                          | User
 ID | Project ID                       |
+------------+-------+-------+--------------------------------------
+---------+----------------------------------+
| image      | gauge | image | acafc7c0-40aa-4026-9673-b879898e1fc2 | None
    | efa984b0a914450e9a47788ad330699d |
| image.size | gauge | B     | acafc7c0-40aa-4026-9673-b879898e1fc2 | None
    | efa984b0a914450e9a47788ad330699d |
+------------+-------+-------+--------------------------------------
+---------+----------------------------------+
```

2. Download an image from the Image Service:

```
$ glance image-download "cirros-0.3.2-x86_64" > cirros.img
```

3. Call the `ceilometer meter-list` command again to validate that the download has been detected and stored by the Telemetry:

```
$ ceilometer meter-list
```

```
+---------------+-------+-------+--------------------------------------
+---------+----------------------------------+
| Name          | Type  | Unit  | Resource ID                          |
 User ID | Project ID                       |
```

```
+----------------+-------+-------+--------------------------------------
+---------+--------------------------------+
| image          | gauge | image | acafc7c0-40aa-4026-9673-b879898e1fc2 |
 None    | efa984b0a914450e9a47788ad330699d |
| image.download | delta | B     | acafc7c0-40aa-4026-9673-b879898e1fc2 |
 None    | efa984b0a914450e9a47788ad330699d |
| image.serve    | delta | B     | acafc7c0-40aa-4026-9673-b879898e1fc2 |
 None    | efa984b0a914450e9a47788ad330699d |
| image.size     | gauge | B     | acafc7c0-40aa-4026-9673-b879898e1fc2 |
 None    | efa984b0a914450e9a47788ad330699d |
+----------------+-------+-------+--------------------------------------
+---------+--------------------------------+
```

4. You can now get usage statistics for the various meters:

```
$ ceilometer statistics -m image.download -p 60
```

```
+--------+---------------------+---------------------+-------
+------------+------------+------------+------------+----------
+----------------------------+----------------------------+
| Period | Period Start        | Period End          | Count | Min
 | Max        | Sum        | Avg        | Duration | Duration Start
      | Duration End               |
+--------+---------------------+---------------------+-------
+------------+------------+------------+------------+----------
+----------------------------+----------------------------+
| 60     | 2013-11-18T18:08:50 | 2013-11-18T18:09:50 | 1     | 13167616.0
 | 13167616.0 | 13167616.0 | 13167616.0 | 0.0      | 2013-11-18T18:09:05.
334000 | 2013-11-18T18:09:05.334000 |
+--------+---------------------+---------------------+-------
+------------+------------+------------+------------+----------
+----------------------------+----------------------------+
```

Next steps

Your OpenStack environment now includes Telemetry. You can launch an instance or add more services to your environment in the previous chapters.

13. Add the Database service

Table of Contents

Use the *Database module* to create cloud database resources. The integrated project name is *trove*.

Warning

This chapter is a work in progress. It may contain incorrect information, and will be updated frequently.

Database service overview

The Database service provides scalable and reliable cloud provisioning functionality for both relational and non-relational database engines. Users can quickly and easily utilize database features without the burden of handling complex administrative tasks. Cloud users and database administrators can provision and manage multiple database instances as needed.

The Database service provides resource isolation at high performance levels, and automates complex administrative tasks such as deployment, configuration, patching, backups, restores, and monitoring.

Process flow example. Here is a high-level process flow example for using Database services:

1. Administrator sets up infrastructure:
    a. OpenStack administrator installs the Database service.
    b. She creates one image for each type of database the administrator wants to have (one for MySQL, one for MongoDB, and so on).
    c. OpenStack administrator updates the datastore to use the new images, using the **trove-manage** command.
2. End user uses database service:
    a. Now that the basic infrastructure is set up, an end user can create a Trove instance (database) whenever the user wants, using the **trove create** command.
    b. The end user gets the IP address of the Trove instance by using the **trove list** command to get the ID of the instance, and then using the **trove show *`instanceID`*** command to get the IP address.
    c. The end user can now access the Trove instance using typical database access commands. MySQL example:

```
$ mysql -u myuser -pmypass -h trove_ip_address mydb
```

Components: The Database service includes the following components:

- `python-troveclient` command-line client. A CLI that communicates with the `trove-api` component.
- `trove-api` component. Provides an OpenStack-native RESTful API that supports JSON to provision and manage Trove instances.
- `trove-conductor` service. Runs on the host, and receives messages from guest instances that want to update information on the host.
- `trove-taskmanager` service. Instruments the complex system flows that support provisioning instances, managing the lifecycle of instances, and performing operations on instances.
- `trove-guestagent` service. Runs within the guest instance. Manages and performs operations on the database itself.

Install the Database service

This procedure installs the Database module on the controller node.

Prerequisites. This chapter assumes that you already have a working OpenStack environment with at least the following components installed: Compute, Image Service, Identity.

To install the Database module on the controller:

1. Install required packages:

```
# zypper install openstack-trove python-troveclient
```

2. Prepare OpenStack:

   a. Source the `admin-openrc.sh` file.

```
$ source ~/admin-openrc.sh
```

   b. Create a `trove` user that Compute uses to authenticate with the Identity service. Use the `service` tenant and give the user the `admin` role:

```
$ keystone user-create --name=trove --pass=TROVE_PASS \
  --email=trove@example.com
$ keystone user-role-add --user=trove --tenant=service --role=admin
```

3. Edit the following configuration files, taking the below actions for each file:

   - `trove.conf`
   - `trove-taskmanager.conf`
   - `trove-conductor.conf`

a. Edit the [DEFAULT] section of each file and set appropriate values for the OpenStack service URLs, logging and messaging configuration, and SQL connections:

```
[DEFAULT]
log_dir = /var/log/trove
trove_auth_url = http://controller:5000/v2.0
nova_compute_url = http://controller:8774/v2
cinder_url = http://controller:8776/v1
swift_url = http://controller:8080/v1/AUTH_
sql_connection = trove:TROVE_DBPASS@controller/trove
notifier_queue_hostname = controller
```

b. Set these configuration keys to configure the Database module to use the RabbitMQ message broker:

```
# openstack-config --set /etc/trove/trove-api.conf \
  DEFAULT rpc_backend rabbit
# openstack-config --set /etc/trove/trove-taskmaster.conf \
  DEFAULT rpc_backend rabbit
# openstack-config --set /etc/trove/trove-conductor.conf \
  DEFAULT rpc_backend rabbit
# openstack-config --set /etc/trove/trove-api.conf DEFAULT \
  rabbit_host controller
# openstack-config --set /etc/trove/trove-taskmaster.conf DEFAULT \
  rabbit_host controller
# openstack-config --set /etc/trove/trove-conductor.conf DEFAULT \
  rabbit_host controller
# openstack-config --set /etc/trove/trove-api.conf DEFAULT \
  rabbit_password RABBIT_PASS
# openstack-config --set /etc/trove/trove-taskmaster.conf DEFAULT \
  rabbit_password RABBIT_PASS
# openstack-config --set /etc/trove/trove-conductor.conf DEFAULT \
  rabbit_password RABBIT_PASS
```

4. Edit the [filter:authtoken] section of the api-paste.ini file so it matches the listing shown below:

```
[filter:authtoken]
auth_host = controller
auth_port = 35357
auth_protocol = http
admin_user = trove
admin_password = ADMIN_PASS
admin_token = ADMIN_TOKEN
admin_tenant_name = service
signing_dir = /var/cache/trove
```

5. Edit the trove.conf file so it includes appropriate values for the default datastore and network label regex as shown below:

```
[DEFAULT]
default_datastore = mysql
....
# Config option for showing the IP address that nova doles out
add_addresses = True
network_label_regex = ^NETWORK_LABEL$
....
```

6. Edit the `trove-taskmanager.conf` file so it includes the appropriate service credentials required to connect to the OpenStack Compute service as shown below:

```
[DEFAULT]
....
# Configuration options for talking to nova via the novaclient.
# These options are for an admin user in your keystone config.
# It proxy's the token received from the user to send to nova via this
 admin users creds,
# basically acting like the client via that proxy token.
nova_proxy_admin_user = admin
nova_proxy_admin_pass = ADMIN_PASS
nova_proxy_admin_tenant_name = service
...
```

7. Prepare the trove admin database:

```
$ mysql -u root -p
mysql> CREATE DATABASE trove;
mysql> GRANT ALL PRIVILEGES ON trove.* TO trove@'localhost' IDENTIFIED BY
 'TROVE_DBPASS';
mysql> GRANT ALL PRIVILEGES ON trove.* TO trove@'%' IDENTIFIED BY
 'TROVE_DBPASS';
```

8. Prepare the Database service:

   a. Initialize the database:

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

   b. Create a datastore. You need to create a separate datastore for each type of database you want to use, for example, MySQL, MongoDB, Cassandra. This example shows you how to create a datastore for a MySQL database:

```
# su -s /bin/sh -c "trove-manage datastore_update mysql '' " trove
```

9. Create a trove image.

   Create an image for the type of database you want to use, for example, MySQL, MongoDB, Cassandra.

   This image must have the trove guest agent installed, and it must have the `trove-guestagent.conf` file configured to connect to your OpenStack environment. To correctly configure the `trove-guestagent.conf` file, follow these steps on the guest instance you are using to build your image:

   - Add the following lines to `trove-guestagent.conf`:

```
rabbit_host = controller
rabbit_password = RABBIT_PASS
nova_proxy_admin_user = admin
nova_proxy_admin_pass = ADMIN_PASS
nova_proxy_admin_tenant_name = service
trove_auth_url = http://controller:35357/v2.0
```

10. Update the datastore to use the new image, using the **trove-manage** command.

    This example shows you how to create a MySQL 5.5 datastore:

```
# trove-manage --config-file=/etc/trove/trove.conf
 datastore_version_update \
  mysql mysql-5.5 mysql glance_image_ID mysql-server-5.5 1
```

11. You must register the Database module with the Identity service so that other OpenStack services can locate it. Register the service and specify the endpoint:

```
$ keystone service-create --name=trove --type=database \
  --description="OpenStack Database Service"
$ keystone endpoint-create \
  --service-id=$(keystone service-list | awk '/ trove / {print $2}') \
  --publicurl=http://controller:8779/v1.0/%\(tenant_id\)s \
  --internalurl=http://controller:8779/v1.0/%\(tenant_id\)s \
  --adminurl=http://controller:8779/v1.0/%\(tenant_id\)s
```

12. Start Database services and configure them to start when the system boots:

```
# service openstack-trove-api start
# service openstack-trove-taskmanager start
# service openstack-trove-conductor start
# chkconfig openstack-trove-api on
# chkconfig openstack-trove-taskmanager on
# chkconfig openstack-trove-conductor on
```

Verify the Database service installation

To verify that the Database service is installed and configured correctly, try executing a Trove command:

1. Source the `demo-openrc.sh` file.

```
$ source ~/demo-openrc.sh
```

2. Retrieve the Trove instances list:

```
$ trove list
```

You should see output similar to this:

```
+----+------+-----------+-------------------+--------+-----------+------+
| id | name | datastore | datastore_version | status | flavor_id | size |
+----+------+-----------+-------------------+--------+-----------+------+
+----+------+-----------+-------------------+--------+-----------+------+
```

3. Assuming you have created an image for the type of database you want, and have updated the datastore to use that image, you can now create a Trove instance (database). To do this, use the trove **create** command.

This example shows you how to create a MySQL 5.5 database:

```
$ trove create name 2 --size=2 --databases=DBNAME \
  --users USER:PASSWORD --datastore_version mysql-5.5 \
  --datastore mysql
```

14. Launch an instance

Table of Contents

An instance is a VM that OpenStack provisions on a compute node. This guide shows you how to launch a minimal instance using the *CirrOS* image that you added to your environment in the Chapter 5, "Configure the Image Service" [35] chapter. In these steps, you use the command-line interface (CLI) on your controller node or any system with the appropriate OpenStack client libraries. To use the dashboard, see the *OpenStack User Guide*.

Launch an instance using OpenStack Networking (neutron) or legacy networking (nova-network) . For more information, see the *OpenStack User Guide*.

Note

These steps reference example components created in previous chapters. You must adjust certain values such as IP addresses to match your environment.

Launch an instance with OpenStack Networking (neutron)

To generate a keypair

Most cloud images support *public key authentication* rather than conventional username/password authentication. Before launching an instance, you must generate a public/private key pair using **ssh-keygen** and add the public key to your OpenStack environment.

1. Source the `demo` tenant credentials:

```
$ source demo-openrc.sh
```

2. Generate a key pair:

```
$ ssh-keygen
```

3. Add the public key to your OpenStack environment:

```
$ nova keypair-add --pub-key ~/.ssh/id_rsa.pub demo-key
```

Note

This command provides no output.

4. Verify addition of the public key:

```
$ nova keypair-list
+----------+-------------------------------------------------+
```

```
| Name     | Fingerprint                                     |
+----------+-------------------------------------------------+
| demo-key | 6c:74:ec:3a:08:05:4e:9e:21:22:a6:dd:b2:62:b8:28 |
+----------+-------------------------------------------------+
```

To launch an instance

To launch an instance, you must at least specify the flavor, image name, network, security group, key, and instance name.

1. A flavor specifies a virtual resource allocation profile which includes processor, memory, and storage.

   List available flavors:

```
$ nova flavor-list
+----+-----------+-----------+------+-----------+------+-------
+-------------+-----------+
| ID | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs |
 RXTX_Factor | Is_Public |
+----+-----------+-----------+------+-----------+------+-------
+-------------+-----------+
| 1  | m1.tiny   | 512       | 1    | 0         |      | 1     | 1.0
   | True      |
| 2  | m1.small  | 2048      | 20   | 0         |      | 1     | 1.0
   | True      |
| 3  | m1.medium | 4096      | 40   | 0         |      | 2     | 1.0
   | True      |
| 4  | m1.large  | 8192      | 80   | 0         |      | 4     | 1.0
   | True      |
| 5  | m1.xlarge | 16384     | 160  | 0         |      | 8     | 1.0
   | True      |
+----+-----------+-----------+------+-----------+------+-------
+-------------+-----------+
```

   Your first instance uses the `m1.tiny` flavor.

   Note

   You can also reference a flavor by ID.

2. List available images:

```
$ nova image-list
+--------------------------------------+---------------------+--------
+--------+
| ID                                   | Name                | Status |
 Server |
+--------------------------------------+---------------------+--------
+--------+
| acafc7c0-40aa-4026-9673-b879898e1fc2 | cirros-0.3.2-x86_64 | ACTIVE |
     |
+--------------------------------------+---------------------+--------
+--------+
```

   Your first instance uses the `cirros-0.3.2-x86_64` image.

3. List available networks:

```
$ neutron net-list
+--------------------------------------+----------
+------------------------------------------------------+
| id                                   | name     | subnets
                                |
+--------------------------------------+----------
+------------------------------------------------------+
| 3c612b5a-d1db-498a-babb-a4c50e344cb1 | demo-net | 20bcd3fd-5785-41fe-
ac42-55ff884e3180 192.168.1.0/24   |
| 9bce64a3-a963-4c05-bfcd-161f708042d1 | ext-net  | b54a8d85-b434-4e85-
a8aa-74873841a90d 203.0.113.0/24   |
+--------------------------------------+----------
+------------------------------------------------------+
```

Your first instance uses the `demo-net` tenant network. However, you must reference this network using the ID instead of the name.

4. List available security groups:

```
$ nova secgroup-list
+--------------------------------------+---------+-------------+
| Id                                   | Name    | Description |
+--------------------------------------+---------+-------------+
| ad8d4ea5-3cad-4f7d-b164-ada67ec59473 | default | default     |
+--------------------------------------+---------+-------------+
```

Your first instance uses the `default` security group. By default, this security group implements a firewall that blocks remote access to instances. If you would like to permit remote access to your instance, launch it and then configure remote access.

5. Launch the instance:

Replace *DEMO_NET_ID* with the ID of the `demo-net` tenant network.

```
$ nova boot --flavor m1.tiny --image cirros-0.3.2-x86_64 --nic net-
id=DEMO_NET_ID \
  --security-group default --key-name demo-key demo-instance1
+--------------------------------------
+------------------------------------------------------------+
| Property                             | Value
                          |
+--------------------------------------
+------------------------------------------------------------+
| OS-DCF:diskConfig                    | MANUAL
                          |
| OS-EXT-AZ:availability_zone          | nova
                          |
| OS-EXT-STS:power_state               | 0
                          |
| OS-EXT-STS:task_state                | scheduling
                          |
| OS-EXT-STS:vm_state                  | building
                          |
| OS-SRV-USG:launched_at               | -
                          |
| OS-SRV-USG:terminated_at             | -
                          |
| accessIPv4                           |
                          |
```

```
| accessIPv6                           |
                          |
| adminPass                            | vFW7Bp8PQGNo
                          |
| config_drive                         |
                          |
| created                              | 2014-04-09T19:24:27Z
                          |
| flavor                               | m1.tiny (1)
                          |
| hostId                               |
                          |
| id                                   |
 05682b91-81a1-464c-8f40-8b3da7ee92c5                       |
| image                                | cirros-0.3.2-x86_64
 (acafc7c0-40aa-4026-9673-b879898e1fc2) |
| key_name                             | demo-key
                          |
| metadata                             | {}
                          |
| name                                 | demo-instance1
                          |
| os-extended-volumes:volumes_attached | []
                          |
| progress                             | 0
                          |
| security_groups                      | default
                          |
| status                               | BUILD
                          |
| tenant_id                            | 7cf50047f8df4824bc76c2fdf66d11ec
                          |
| updated                              | 2014-04-09T19:24:27Z
                          |
| user_id                              | 0e47686e72114d7182f7569d70c519c9
                          |
+--------------------------------------
+-------------------------------------------------------------+
```

6. Check the status of your instance:

```
$ nova list
+--------------------------------------+----------------+--------
+------------+-------------+-------------------------+
| ID                                   | Name           | Status | Task
 State | Power State | Networks                |
+--------------------------------------+----------------+--------
+------------+-------------+-------------------------+
| 05682b91-81a1-464c-8f40-8b3da7ee92c5 | demo-instance1 | ACTIVE | -
    | Running     | demo-net=192.168.1.3    |
+--------------------------------------+----------------+--------
+------------+-------------+-------------------------+
```

The status changes from `BUILD` to `ACTIVE` when your instance finishes the build process.

To access your instance using a virtual console

- Obtain a *Virtual Network Computing (VNC)* session URL for your instance and access it from a web browser:

```
$ nova get-vnc-console demo-instance1 novnc
+-------
+-------------------------------------------------------------------------------
+
| Type  | Url
                  |
+-------
+-------------------------------------------------------------------------------
+
| novnc | http://controller:6080/vnc_auto.html?token=2f6dd985-f906-4bfc-
b566-e87ce656375b    |
+-------
+-------------------------------------------------------------------------------
+
```

Note

If your web browser runs on a host that cannot resolve the `controller` host name, you can replace `controller` with the IP address of the management interface on your controller node.

The CirrOS image includes conventional username/password authentication and provides these credentials at the login prompt. After logging into CirrOS, we recommend that you verify network connectivity using **ping**.

Verify the `demo-net` tenant network gateway:

```
$ ping -c 4 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_req=1 ttl=64 time=0.357 ms
64 bytes from 192.168.1.1: icmp_req=2 ttl=64 time=0.473 ms
64 bytes from 192.168.1.1: icmp_req=3 ttl=64 time=0.504 ms
64 bytes from 192.168.1.1: icmp_req=4 ttl=64 time=0.470 ms

--- 192.168.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2998ms
rtt min/avg/max/mdev = 0.357/0.451/0.504/0.055 ms
```

Verify the `ext-net` external network:

```
$ ping -c 4 openstack.org
PING openstack.org (174.143.194.225) 56(84) bytes of data.
64 bytes from 174.143.194.225: icmp_req=1 ttl=53 time=17.4 ms
64 bytes from 174.143.194.225: icmp_req=2 ttl=53 time=17.5 ms
64 bytes from 174.143.194.225: icmp_req=3 ttl=53 time=17.7 ms
64 bytes from 174.143.194.225: icmp_req=4 ttl=53 time=17.5 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 17.431/17.575/17.734/0.143 ms
```

To access your instance remotely

1. Add rules to the `default` security group:

   a. Permit *ICMP* (ping):

```
$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
+-------------+-----------+---------+-----------+--------------+
| IP Protocol | From Port | To Port | IP Range  | Source Group |
+-------------+-----------+---------+-----------+--------------+
| icmp        | -1        | -1      | 0.0.0.0/0 |              |
+-------------+-----------+---------+-----------+--------------+
```

   b. Permit secure shell (SSH) access:

```
$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
+-------------+-----------+---------+-----------+--------------+
| IP Protocol | From Port | To Port | IP Range  | Source Group |
+-------------+-----------+---------+-----------+--------------+
| tcp         | 22        | 22      | 0.0.0.0/0 |              |
+-------------+-----------+---------+-----------+--------------+
```

2. Create a *floating IP address* on the `ext-net` external network:

```
$ neutron floatingip-create ext-net
Created a new floatingip:
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| fixed_ip_address    |                                      |
| floating_ip_address | 203.0.113.102                        |
| floating_network_id | 9bce64a3-a963-4c05-bfcd-161f708042d1 |
| id                  | 05e36754-e7f3-46bb-9eaa-3521623b3722 |
| port_id             |                                      |
| router_id           |                                      |
| status              | DOWN                                 |
| tenant_id           | 7cf50047f8df4824bc76c2fdf66d11ec     |
+---------------------+--------------------------------------+
```

3. Associate the floating IP address with your instance:

```
$ nova floating-ip-associate demo-instance1 203.0.113.102
```

Note

This command provides no output.

4. Check the status of your floating IP address:

```
$ nova list
+--------------------------------------+----------------+--------
+------------+-------------+-----------------------------------------+
| ID                                   | Name           | Status | Task
 State | Power State | Networks                                |
+--------------------------------------+----------------+--------
+------------+-------------+-----------------------------------------+
| 05682b91-81a1-464c-8f40-8b3da7ee92c5 | demo-instance1 | ACTIVE | -
    | Running     | demo-net=192.168.1.3, 203.0.113.102     |
+--------------------------------------+----------------+--------
+------------+-------------+-----------------------------------------+
```

5. Verify network connectivity using **ping** from the controller node or any host on the external network:

```
$ ping -c 4 203.0.113.102
PING 203.0.113.102 (203.0.113.112) 56(84) bytes of data.
64 bytes from 203.0.113.102: icmp_req=1 ttl=63 time=3.18 ms
64 bytes from 203.0.113.102: icmp_req=2 ttl=63 time=0.981 ms
64 bytes from 203.0.113.102: icmp_req=3 ttl=63 time=1.06 ms
64 bytes from 203.0.113.102: icmp_req=4 ttl=63 time=0.929 ms

--- 203.0.113.102 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 0.929/1.539/3.183/0.951 ms
```

6. Access your instance using SSH from the controller node or any host on the external network:

```
$ ssh cirros@203.0.113.102
The authenticity of host '203.0.113.102 (203.0.113.102)' can't be
 established.
RSA key fingerprint is ed:05:e9:e7:52:a0:ff:83:68:94:c7:d1:f2:f8:e2:e9.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '203.0.113.102' (RSA) to the list of known
 hosts.
$
```

Note

If your host does not contain the public/private key pair created in an earlier step, SSH prompts for the default password associated with the `cirros` user.

If your instance does not launch or seem to work as you expect, see the *OpenStack Operations Guide* for more information or use one of the many other options to seek assistance. We want your environment to work!

Launch an instance with legacy networking (nova-network)

To generate a keypair

Most cloud images support *public key authentication* rather than conventional username/password authentication. Before launching an instance, you must generate a public/private key pair using **ssh-keygen** and add the public key to your OpenStack environment.

1. Source the `demo` tenant credentials:

```
$ source demo-openrc.sh
```

2. Generate a key pair:

```
$ ssh-keygen
```

3. Add the public key to your OpenStack environment:

```
$ nova keypair-add --pub-key ~/.ssh/id_rsa.pub demo-key
```

Note

This command provides no output.

4. Verify addition of the public key:

```
$ nova keypair-list
+----------+-------------------------------------------------+
| Name     | Fingerprint                                     |
+----------+-------------------------------------------------+
| demo-key | 6c:74:ec:3a:08:05:4e:9e:21:22:a6:dd:b2:62:b8:28 |
+----------+-------------------------------------------------+
```

To launch an instance

To launch an instance, you must at least specify the flavor, image name, network, security group, key, and instance name.

1. A flavor specifies a virtual resource allocation profile which includes processor, memory, and storage.

   List available flavors:

```
$ nova flavor-list
+----+-----------+-----------+------+-----------+------+-------
+-------------+-----------+
| ID | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs |
 RXTX_Factor | Is_Public |
+----+-----------+-----------+------+-----------+------+-------
+-------------+-----------+
| 1  | m1.tiny   | 512       | 1    | 0         |      | 1     | 1.0
   | True      |
| 2  | m1.small  | 2048      | 20   | 0         |      | 1     | 1.0
   | True      |
| 3  | m1.medium | 4096      | 40   | 0         |      | 2     | 1.0
   | True      |
| 4  | m1.large  | 8192      | 80   | 0         |      | 4     | 1.0
   | True      |
| 5  | m1.xlarge | 16384     | 160  | 0         |      | 8     | 1.0
   | True      |
+----+-----------+-----------+------+-----------+------+-------
+-------------+-----------+
```

   Your first instance uses the `m1.tiny` flavor.

Note

You can also reference a flavor by ID.

2. List available images:

```
$ nova image-list
+--------------------------------------+---------------------+--------
+--------+
| ID                                   | Name                | Status |
 Server |
```

```
+--------------------------------------+--------------------+--------
+--------+
| acafc7c0-40aa-4026-9673-b879898e1fc2 | cirros-0.3.2-x86_64 | ACTIVE |
    |
+--------------------------------------+--------------------+--------
+--------+
```

Your first instance uses the `cirros-0.3.2-x86_64` image.

3. List available networks:

Note

You must source the `admin` tenant credentials for this step and then source the `demo` tenant credentials for the remaining steps.

```
$ source admin-openrc.sh
```

```
$ nova net-list
+--------------------------------------+----------+------------------+
| ID                                   | Label    | CIDR             |
+--------------------------------------+----------+------------------+
| 7f849be3-4494-495a-95a1-0f99ccb884c4 | demo-net | 203.0.113.24/29  |
+--------------------------------------+----------+------------------+
```

Your first instance uses the `demo-net` tenant network. However, you must reference this network using the ID instead of the name.

4. List available security groups:

```
$ nova secgroup-list
+--------------------------------------+---------+-------------+
| Id                                   | Name    | Description |
+--------------------------------------+---------+-------------+
| ad8d4ea5-3cad-4f7d-b164-ada67ec59473 | default | default     |
+--------------------------------------+---------+-------------+
```

Your first instance uses the `default` security group. By default, this security group implements a firewall that blocks remote access to instances. If you would like to permit remote access to your instance, launch it and then configure remote access.

5. Launch the instance:

Replace *`DEMO_NET_ID`* with the ID of the `demo-net` tenant network.

```
$ nova boot --flavor m1.tiny --image cirros-0.3.2-x86_64 --nic net-
id=DEMO_NET_ID \
  --security-group default --key-name demo-key demo-instance1
+--------------------------------------
+------------------------------------------------------------+
| Property                             | Value
                          |
+--------------------------------------
+------------------------------------------------------------+
| OS-DCF:diskConfig                    | MANUAL
                          |
| OS-EXT-AZ:availability_zone          | nova
                          |
```

```
| OS-EXT-STS:power_state               | 0
                        |
| OS-EXT-STS:task_state                | scheduling
                        |
| OS-EXT-STS:vm_state                  | building
                        |
| OS-SRV-USG:launched_at               | -
                        |
| OS-SRV-USG:terminated_at             | -
                        |
| accessIPv4                           |
                        |
| accessIPv6                           |
                        |
| adminPass                            | ThZqrg7ach78
                        |
| config_drive                         |
                        |
| created                              | 2014-04-10T00:09:16Z
                        |
| flavor                               | m1.tiny (1)
                        |
| hostId                               |
                        |
| id                                   | 45ea195c-
c469-43eb-83db-1a663bbad2fc                        |
| image                                | cirros-0.3.2-x86_64
 (acafc7c0-40aa-4026-9673-b879898e1fc2) |
| key_name                             | demo-key
                        |
| metadata                             | {}
                        |
| name                                 | demo-instance1
                        |
| os-extended-volumes:volumes_attached | []
                        |
| progress                             | 0
                        |
| security_groups                      | default
                        |
| status                               | BUILD
                        |
| tenant_id                            | 93849608fe3d462ca9fa0e5dbfd4d040
                        |
| updated                              | 2014-04-10T00:09:16Z
                        |
| user_id                              | 8397567baf4746cca7a1e608677c3b23
                        |
+--------------------------------------
+-------------------------------------------------------------+
```

6. Check the status of your instance:

```
$ nova list
+--------------------------------------+----------------+--------
+------------+-------------+-----------------------+
| ID                                   | Name           | Status | Task
 State | Power State | Networks              |
+--------------------------------------+----------------+--------
+------------+-------------+-----------------------+
```

```
| 45ea195c-c469-43eb-83db-1a663bbad2fc | demo-instance1 | ACTIVE | -
    | Running     | demo-net=203.0.113.26  |
+--------------------------------------+----------------+--------
+------------+-------------+-----------------------+
```

The status changes from `BUILD` to `ACTIVE` when your instance finishes the build process.

To access your instance using a virtual console

- Obtain a *Virtual Network Computing (VNC)* session URL for your instance and access it from a web browser:

```
$ nova get-vnc-console demo-instance1 novnc
+-------
+-------------------------------------------------------------------------------
+
| Type  | Url
                  |
+-------
+-------------------------------------------------------------------------------
+
| novnc | http://controller:6080/vnc_auto.html?token=2f6dd985-f906-4bfc-
b566-e87ce656375b    |
+-------
+-------------------------------------------------------------------------------
+
```

Note

If your web browser runs on a host that cannot resolve the `controller` host name, you can replace `controller` with the IP address of the management interface on your controller node.

The CirrOS image includes conventional username/password authentication and provides these credentials at the login prompt. After logging into CirrOS, we recommend that you verify network connectivity using **ping**.

Verify the `demo-net` network:

```
$ ping -c 4 openstack.org
PING openstack.org (174.143.194.225) 56(84) bytes of data.
64 bytes from 174.143.194.225: icmp_req=1 ttl=53 time=17.4 ms
64 bytes from 174.143.194.225: icmp_req=2 ttl=53 time=17.5 ms
64 bytes from 174.143.194.225: icmp_req=3 ttl=53 time=17.7 ms
64 bytes from 174.143.194.225: icmp_req=4 ttl=53 time=17.5 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 17.431/17.575/17.734/0.143 ms
```

To access your instance remotely

1. Add rules to the `default` security group:

   a. Permit *ICMP* (ping):

```
$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
```

```
+-------------+-----------+---------+-----------+--------------+
| IP Protocol | From Port | To Port | IP Range  | Source Group |
+-------------+-----------+---------+-----------+--------------+
| icmp        | -1        | -1      | 0.0.0.0/0 |              |
+-------------+-----------+---------+-----------+--------------+
```

b. Permit secure shell (SSH) access:

```
$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
+-------------+-----------+---------+-----------+--------------+
| IP Protocol | From Port | To Port | IP Range  | Source Group |
+-------------+-----------+---------+-----------+--------------+
| tcp         | 22        | 22      | 0.0.0.0/0 |              |
+-------------+-----------+---------+-----------+--------------+
```

2. Verify network connectivity using **ping** from the controller node or any host on the external network:

```
$ ping -c 4 203.0.113.26
PING 203.0.113.26 (203.0.113.26) 56(84) bytes of data.
64 bytes from 203.0.113.26: icmp_req=1 ttl=63 time=3.18 ms
64 bytes from 203.0.113.26: icmp_req=2 ttl=63 time=0.981 ms
64 bytes from 203.0.113.26: icmp_req=3 ttl=63 time=1.06 ms
64 bytes from 203.0.113.26: icmp_req=4 ttl=63 time=0.929 ms

--- 203.0.113.26 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 0.929/1.539/3.183/0.951 ms
```

3. Access your instance using SSH from the controller node or any host on the external network:

```
$ ssh cirros@203.0.113.26
The authenticity of host '203.0.113.26 (203.0.113.26)' can't be
 established.
RSA key fingerprint is ed:05:e9:e7:52:a0:ff:83:68:94:c7:d1:f2:f8:e2:e9.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '203.0.113.26' (RSA) to the list of known
 hosts.
$
```

Note

If your host does not contain the public/private key pair created in an earlier step, SSH prompts for the default password associated with the `cirros` user.

If your instance does not launch or seem to work as you expect, see the *OpenStack Operations Guide* for more information or use one of the many other options to seek assistance. We want your environment to work!

Appendix A. Reserved user IDs

In OpenStack, certain user IDs are reserved and used to run specific OpenStack services and own specific OpenStack files. These users are set up according to the distribution packages. The following table gives an overview.

Note

Some OpenStack packages generate and assign user IDs automatically during package installation. In these cases, the user ID value is not important. The existence of the user ID is what matters.

Table A.1. Reserved user IDs

Name	Description	ID
ceilometer	OpenStack Ceilometer Daemons	Assigned during package installation
cinder	OpenStack Cinder Daemons	Assigned during package installation
glance	OpenStack Glance Daemons	Assigned during package installation
heat	OpenStack Heat Daemons	Assigned during package installation
keystone	OpenStack Keystone Daemons	Assigned during package installation
neutron	OpenStack Neutron Daemons	Assigned during package installation
nova	OpenStack Nova Daemons	96
swift	OpenStack Swift Daemons	Assigned during package installation
trove	OpenStack Trove Daemons	Assigned during package installation

Each user belongs to a user group with the same name as the user.

Appendix B. Community support

Table of Contents

The following resources are available to help you run and use OpenStack. The OpenStack community constantly improves and adds to the main features of OpenStack, but if you have any questions, do not hesitate to ask. Use the following resources to get OpenStack support, and troubleshoot your installations.

Documentation

For the available OpenStack documentation, see docs.openstack.org.

To provide feedback on documentation, join and use the <openstack-docs@lists.openstack.org> mailing list at OpenStack Documentation Mailing List, or report a bug.

The following books explain how to install an OpenStack cloud and its associated components:

- *Installation Guide for Debian 7.0*
- *Installation Guide for openSUSE and SUSE Linux Enterprise Server*
- *Installation Guide for Red Hat Enterprise Linux, CentOS, and Fedora*
- *Installation Guide for Ubuntu 12.04/14.04 (LTS)*

The following books explain how to configure and run an OpenStack cloud:

- *Cloud Administrator Guide*
- *Configuration Reference*
- *Operations Guide*
- *High Availability Guide*
- *Security Guide*

- *Virtual Machine Image Guide*

The following books explain how to use the OpenStack dashboard and command-line clients:

- *API Quick Start*
- *End User Guide*
- *Admin User Guide*
- *Command-Line Interface Reference*

The following documentation provides reference and guidance information for the OpenStack APIs:

- OpenStack API Complete Reference (HTML)
- API Complete Reference (PDF)
- *OpenStack Block Storage Service API v2 Reference*
- *OpenStack Compute API v2 and Extensions Reference*
- *OpenStack Identity Service API v2.0 Reference*
- *OpenStack Image Service API v2 Reference*
- *OpenStack Networking API v2.0 Reference*
- *OpenStack Object Storage API v1 Reference*

The *Training Guides* offer software training for cloud administration and management.

ask.openstack.org

During the set up or testing of OpenStack, you might have questions about how a specific task is completed or be in a situation where a feature does not work correctly. Use the ask.openstack.org site to ask questions and get answers. When you visit the http://ask.openstack.org site, scan the recently asked questions to see whether your question has already been answered. If not, ask a new question. Be sure to give a clear, concise summary in the title and provide as much detail as possible in the description. Paste in your command output or stack traces, links to screen shots, and any other information which might be useful.

OpenStack mailing lists

A great way to get answers and insights is to post your question or problematic scenario to the OpenStack mailing list. You can learn from and help others who might have similar issues. To subscribe or view the archives, go to http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack. You might be interested in the other mailing lists for specific projects or development, which you can find on the wiki. A description of all mailing lists is available at http://wiki.openstack.org/MailingLists.

The OpenStack wiki

The OpenStack wiki contains a broad range of topics but some of the information can be difficult to find or is a few pages deep. Fortunately, the wiki search feature enables you to search by title or content. If you search for specific information, such as about networking or nova, you can find a large amount of relevant material. More is being added all the time, so be sure to check back often. You can find the search box in the upper-right corner of any OpenStack wiki page.

The Launchpad Bugs area

The OpenStack community values your set up and testing efforts and wants your feedback. To log a bug, you must sign up for a Launchpad account at https://launchpad.net/+login. You can view existing bugs and report bugs in the Launchpad Bugs area. Use the search feature to determine whether the bug has already been reported or already been fixed. If it still seems like your bug is unreported, fill out a bug report.

Some tips:

- Give a clear, concise summary.
- Provide as much detail as possible in the description. Paste in your command output or stack traces, links to screen shots, and any other information which might be useful.
- Be sure to include the software and package versions that you are using, especially if you are using a development branch, such as, `"Juno release" vs git commit bc79c3ecc55929bac585d04a03475b72e06a3208`.
- Any deployment-specific information is helpful, such as whether you are using Ubuntu 14.04 or are performing a multi-node installation.

The following Launchpad Bugs areas are available:

- Bugs: OpenStack Block Storage (cinder)
- Bugs: OpenStack Compute (nova)
- Bugs: OpenStack Dashboard (horizon)
- Bugs: OpenStack Identity (keystone)
- Bugs: OpenStack Image Service (glance)
- Bugs: OpenStack Networking (neutron)
- Bugs: OpenStack Object Storage (swift)
- Bugs: Bare Metal (ironic)
- Bugs: Data Processing Service (sahara)
- Bugs: Database Service (trove)

- Bugs: Orchestration (heat)
- Bugs: Telemetry (ceilometer)
- Bugs: Queue Service (marconi)
- Bugs: OpenStack API Documentation (api.openstack.org)
- Bugs: OpenStack Documentation (docs.openstack.org)

The OpenStack IRC channel

The OpenStack community lives in the #openstack IRC channel on the Freenode network. You can hang out, ask questions, or get immediate feedback for urgent and pressing issues. To install an IRC client or use a browser-based client, go to http://webchat.freenode.net/. You can also use Colloquy (Mac OS X, http://colloquy.info/), mIRC (Windows, http://www.mirc.com/), or XChat (Linux). When you are in the IRC channel and want to share code or command output, the generally accepted method is to use a Paste Bin. The OpenStack project has one at http://paste.openstack.org. Just paste your longer amounts of text or logs in the web form and you get a URL that you can paste into the channel. The OpenStack IRC channel is `#openstack` on `irc.freenode.net`. You can find a list of all OpenStack IRC channels at https://wiki.openstack.org/wiki/IRC.

Documentation feedback

To provide feedback on documentation, join and use the `<openstack-docs@lists.openstack.org>` mailing list at OpenStack Documentation Mailing List, or report a bug.

OpenStack distribution packages

The following Linux distributions provide community-supported packages for OpenStack:

- **Debian:** http://wiki.debian.org/OpenStack
- **CentOS, Fedora, and Red Hat Enterprise Linux:** http://openstack.redhat.com/
- **openSUSE and SUSE Linux Enterprise Server:** http://en.opensuse.org/Portal:OpenStack
- **Ubuntu:** https://wiki.ubuntu.com/ServerTeam/CloudArchive

Glossary

API
: Application programming interface.

API endpoint
: The daemon, worker, or service that a client communicates with to access an API. API endpoints can provide any number of services, such as authentication, sales data, performance metrics, Compute VM commands, census data, and so on.

Block Storage
: The OpenStack core project that enables management of volumes, volume snapshots, and volume types. The project name of Block Storage is cinder.

CirrOS
: A minimal Linux distribution designed for use as a test image on clouds such as OpenStack.

cloud controller node
: A node that runs network, volume, API, scheduler, and image services. Each service may be broken out into separate nodes for scalability or availability.

Compute
: The OpenStack core project that provides compute services. The project name of the Compute service is nova.

compute node
: A node that runs the `nova-compute` daemon which manages VM instances that provide a wide range of services such as a web applications and analytics.

controller node
: Alternative term for a cloud controller node.

Database Service
: An integrated project that provide scalable and reliable Cloud Database-as-a-Service functionality for both relational and non-relational database engines. The project name of Database Service is trove.

DHCP
: Dynamic Host Configuration Protocol. A network protocol that configures devices that are connected to a network so that they can communicate on that network by using the Internet Protocol (IP). The protocol is implemented in a client-server model where DHCP clients request configuration data such as, an IP address, a default route, and one or more DNS server addresses from a DHCP server.

DHCP agent
: OpenStack Networking agent that provides DHCP services for virtual networks.

endpoint
: See API endpoint.

external network
: A network segment typically used for instance Internet access.

firewall
: Used to restrict communications between hosts and/or nodes, implemented in Compute using iptables, arptables, ip6tables, and etables.

flat network
: The Network Controller provides virtual networks to enable compute servers to interact with each other and with the public network. All machines must have a public and private network interface. A flat network is a private network interface, which is controlled by the flat_interface option with flat managers.

floating IP address
: An IP address that a project can associate with a VM so that the instance has the same public IP address each time that it boots. You create a pool of floating IP addresses and assign them to instances as they are launched to maintain a consistent IP address for maintaining DNS assignment.

gateway
: An IP address, typically assigned to a router, that passes network traffic between different networks.

Generic Receive Offload (GRO)
: Feature of certain network interface drivers that combines many smaller received packets into a large packet before delivery to the kernel IP stack.

hypervisor
: Software that arbitrates and controls VM access to the actual underlying hardware.

IaaS
: Infrastructure-as-a-Service. IaaS is a provisioning model in which an organization outsources physical components of a data center such as storage, hardware, servers and networking components. A service provider owns the equipment and is responsible for housing, operating and maintaining it. The client typically pays on a per-use basis. IaaS is a model for providing cloud services.

Icehouse
: Project name for the ninth release of OpenStack.

ICMP
: Internet Control Message Protocol, used by network devices for control messages. For example, **ping** uses ICMP to test connectivity.

Identity Service
: The OpenStack core project that provides a central directory of users mapped to the OpenStack services they can access. It also registers endpoints for OpenStack services. It acts as a common authentication system. The project name of the Identity Service is keystone.

Image Service
: An OpenStack core project that provides discovery, registration, and delivery services for disk and server images. The project name of the Image Service is glance.

instance tunnels network
: A network segment used for instance traffic tunnels between compute nodes and the network node.

interface
: A physical or virtual device that provides connectivity to another device or medium.

kernel-based VM (KVM)
: An OpenStack-supported hypervisor.

Layer-3 (L3) agent
: OpenStack Networking agent that provides layer-3 (routing) services for virtual networks.

load balancer
: A load balancer is a logical device that belongs to a cloud account. It is used to distribute workloads between multiple back-end systems or services, based on the criteria defined as part of its configuration.

Logical Volume Manager (LVM)
: Provides a method of allocating space on mass-storage devices that is more flexible than conventional partitioning schemes.

management network
: A network segment used for administration, not accessible to the public Internet.

message broker
: The software package used to provide AMQP messaging capabilities within Compute. Default package is RabbitMQ.

multi-host
: High-availability mode for legacy (nova) networking. Each compute node handles NAT and DHCP and acts as a gateway for all of the VMs on it. A networking failure on one compute node doesn't affect VMs on other compute nodes.

Network Address Translation (NAT)
: The process of modifying IP address information while in-transit. Supported by Compute and Networking.

Network Time Protocol (NTP)
: A method of keeping a clock for a host or node correct through communications with a trusted, accurate time source.

Networking
: A core OpenStack project that provides a network connectivity abstraction layer to OpenStack Compute. The project name of Networking is neutron.

Object Storage
: The OpenStack core project that provides eventually consistent and redundant storage and retrieval of fixed digital content. The project name of OpenStack Object Storage is swift.

OpenStack
: OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a data center, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface. OpenStack is an open source project licensed under the Apache License 2.0.

Orchestration
: An integrated project that orchestrates multiple cloud applications for OpenStack. The project name of Orchestration is heat.

plug-in
: Software component providing the actual implementation for Networking APIs, or for Compute APIs, depending on the context.

promiscuous mode
: Causes the network interface to pass all traffic it receives to the host rather than passing only the frames addressed to it.

public key authentication
: Authentication method that uses keys rather than passwords.

RESTful
: A kind of web service API that uses REST, or Representational State Transfer. REST is the style of architecture for hypermedia systems that is used for the World Wide Web.

role
: A personality that a user assumes that enables them to perform a specific set of operations. A role includes a set of rights and privileges. A user assuming that role inherits those rights and privileges.

router
: A physical or virtual network device that passes network traffic between different networks.

security group
: A set of network traffic filtering rules that are applied to a Compute instance.

service catalog
: Alternative term for the Identity Service catalog.

subnet
: Logical subdivision of an IP network.

Telemetry
: An integrated project that provides metering and measuring facilities for OpenStack. The project name of Telemetry is ceilometer.

tenant
: A group of users, used to isolate access to Compute resources. An alternative term for a project.

trove
: OpenStack project that provides database services to applications.

user
: In Identity Service, each user is associated with one or more tenants, and in Compute can be associated with roles, projects, or both.

virtual machine (VM)
: An operating system instance that runs on top of a hypervisor. Multiple VMs can run at the same time on the same physical host.

virtual networking
: A generic term for virtualization of network functions such as switching, routing, load balancing, and security using a combination of VMs and overlays on physical network infrastructure.

Virtual Network Computing (VNC)
: Open source GUI and CLI tools used for remote console access to VMs. Supported by Compute.

virtual private network (VPN)
: Provided by Compute in the form of cloudpipes, specialized instances that are used to create VPNs on a per-project basis.