

OpenStack Installation Guide

for Ubuntu 14.04

juno (May 10, 2015)

Juno



OpenStack Installation Guide for Ubuntu 14.04

juno (2015-05-10)

Copyright © 2012-2014 OpenStack Foundation All rights reserved.

The OpenStack® system consists of several key projects that you install separately. These projects 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 walks through an installation by using packages available through Ubuntu 14.04. Explanations of configuration options and sample configuration files are included.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Table of Contents

Preface	vii
Conventions	vii
Document change history	vii
1. Architecture	1
Overview	1
Conceptual architecture	2
Example architectures	3
2. Basic environment	11
Before you begin	11
Security	12
Networking	13
Network Time Protocol (NTP)	24
OpenStack packages	26
Database	27
Messaging server	28
3. Add the Identity service	30
OpenStack Identity concepts	30
Install and configure	32
Create tenants, users, and roles	34
Create the service entity and API endpoint	37
Verify operation	38
Create OpenStack client environment scripts	40
4. Add the Image Service	42
OpenStack Image Service	42
Install and configure	43
Verify operation	47
5. Add the Compute service	49
OpenStack Compute	49
Install and configure controller node	52
Install and configure a compute node	55
Verify operation	58
6. Add a networking component	60
OpenStack Networking (neutron)	60
Legacy networking (nova-network)	83
Next steps	85
7. Add the dashboard	86
System requirements	86
Install and configure	87
Verify operation	88
Next steps	88
8. Add the Block Storage service	89
OpenStack Block Storage	89
Install and configure controller node	90
Install and configure a storage node	93
Verify operation	97
Next steps	98
9. Add Object Storage	99
OpenStack Object Storage	99

Install and configure the controller node	100
Install and configure the storage nodes	103
Create initial rings	107
Finalize installation	111
Verify operation	112
Next steps	113
10. Add the Orchestration module	114
Orchestration module concepts	114
Install and configure Orchestration	114
Verify operation	118
Next steps	120
11. Add the Telemetry module	121
Telemetry module	121
Install and configure controller node	122
Configure the Compute service	125
Configure the Image Service	127
Configure the Block Storage service	127
Configure the Object Storage service	128
Verify the Telemetry installation	129
Next steps	130
12. Add the Database service	131
Database service overview	131
Install the Database service	132
Verify the Database service installation	135
13. Add the Data processing service	136
Data processing service	136
Install the Data processing service	137
Verify the Data processing service installation	138
14. Launch an instance	139
Launch an instance with OpenStack Networking (neutron)	139
Launch an instance with legacy networking (nova-network)	147
A. Reserved user IDs	154
B. Community support	155
Documentation	155
ask.openstack.org	156
OpenStack mailing lists	156
The OpenStack wiki	157
The Launchpad Bugs area	157
The OpenStack IRC channel	158
Documentation feedback	158
OpenStack distribution packages	158
Glossary	159

List of Figures

1.1. Conceptual architecture	2
1.2. Minimal architecture example with OpenStack Networking (neutron)—Hardware requirements	4
1.3. Minimal architecture example with OpenStack Networking (neutron)—Network layout	5
1.4. Minimal architecture example with OpenStack Networking (neutron)—Service layout	6
1.5. Minimal architecture example with legacy networking (nova-network)—Hardware requirements	8
1.6. Minimal architecture example with legacy networking (nova-network)—Network layout	9
1.7. Minimal architecture example with legacy networking (nova-network)—Service layout	10
2.1. Minimal architecture example with OpenStack Networking (neutron)—Network layout	15
2.2. Minimal architecture example with legacy networking (nova-network)—Network layout	21
6.1. Initial networks	78

List of Tables

1.1. OpenStack services	1
2.1. Passwords	12
A.1. Reserved user IDs	154

Preface

Conventions

The OpenStack documentation uses several typesetting conventions.

Notices

Notices take these forms:



Note

A handy tip or reminder.



Important

Something you must be aware of before proceeding.



Warning

Critical information about the risk of data loss or security issues.

Command prompts

- \$ prompt** Any user, including the `root` user, can run commands that are prefixed with the `$` prompt.
- # prompt** The `root` user must run commands that are prefixed with the `#` prompt. You can also prefix these commands with the `sudo` command, if available, to run them.

Document change history

This version of the guide replaces and obsoletes all earlier versions.

The following table describes the most recent changes:

Revision Date	Summary of Changes
October 15, 2014	<ul style="list-style-type: none">For the Juno release, this guide contains these updates: Replace openstack-config commands with general configuration file editing. Standardize on a single message queue system (RabbitMQ). Reference generic SQL database, enabling MySQL or MariaDB where appropriate. Replace <code>auth_port</code> and <code>auth_protocol</code> with <code>identity_uri</code>, and <code>auth_host</code> with <code>auth_uri</code>. Multiple edits for consistency. It has been updated for Juno and new distribution versions.
June 3, 2014	<ul style="list-style-type: none">Start documentation for Juno.
April 16, 2014	<ul style="list-style-type: none">Update for Icehouse, rework Networking setup to use ML2 as plugin, add new chapter for Database Service setup, improved basic configuration.
October 25, 2013	<ul style="list-style-type: none">Added initial Debian support.
October 17, 2013	<ul style="list-style-type: none">Havana release.
October 16, 2013	<ul style="list-style-type: none">Add support for SUSE Linux Enterprise.

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

1. Architecture

Table of Contents

Overview	1
Conceptual architecture	2
Example architectures	3

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 complementary 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 decommissioning 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 <i>RESTful</i> , 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.
Higher-level services		

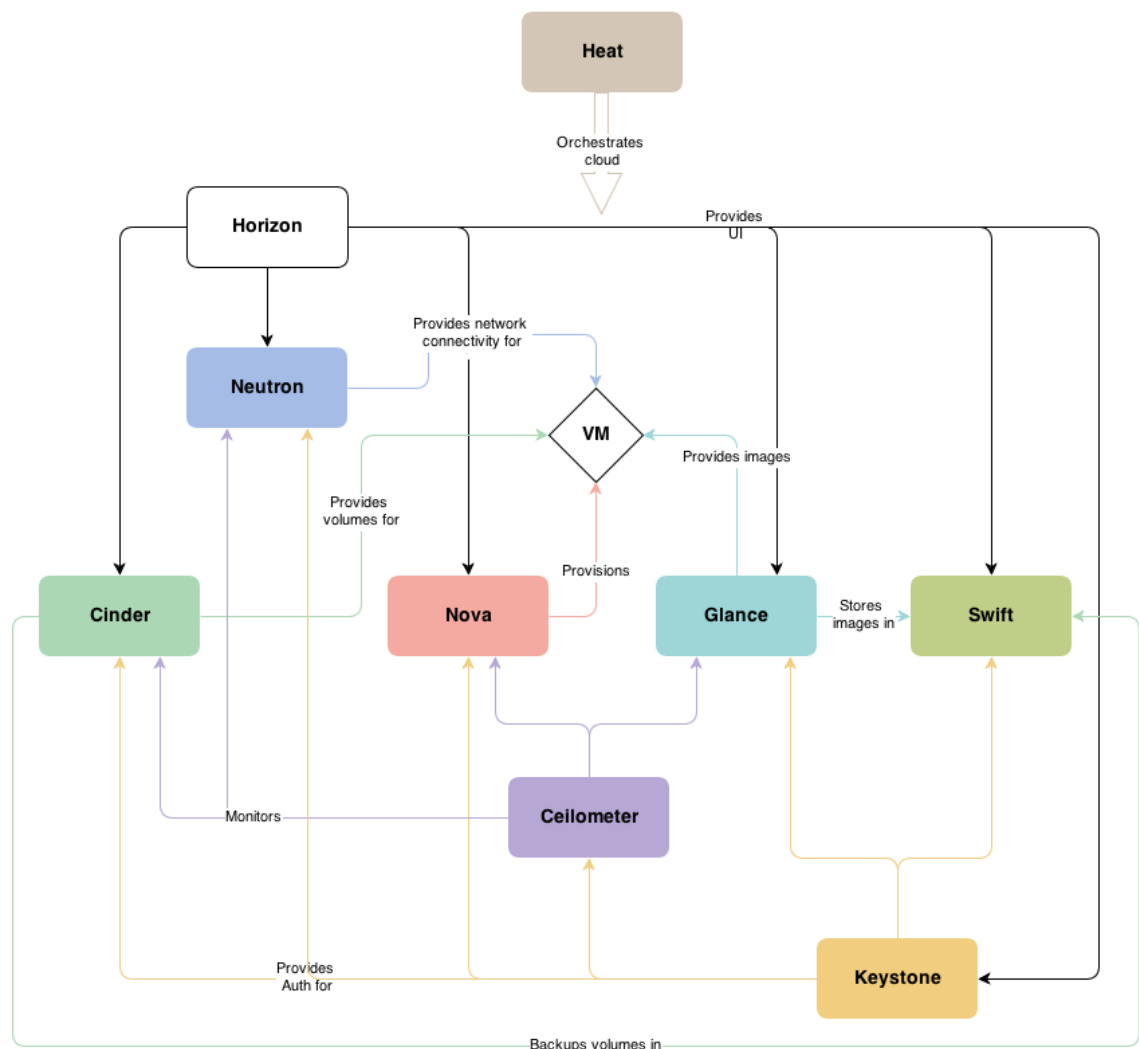
Service	Project name	Description
Orchestration	Heat	Orchestrates multiple composite cloud applications by using either the native <i>HOT</i> template format or the AWS CloudFormation template format, through both an OpenStack-native REST API and a CloudFormation-compatible Query API.
Database Service	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. Realistically, you would use automation tools such as Ansible, Chef, and Puppet to deploy and manage 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 core and optional services. This guide uses the following example architectures:

- Three-node architecture with OpenStack Networking (neutron) and optional nodes for Block Storage and Object Storage services.
- The *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 SQL database, *message queue*, and *Network Time Protocol (NTP)*.

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

- The network node runs the Networking plug-in and several agents that provision tenant networks and provide switching, routing, *NAT*, and *DHCP* services. This node also handles external (Internet) connectivity for tenant virtual machine instances.
- The *compute node* runs the *hypervisor* portion of Compute that operates *tenant virtual machines* or instances. By default, Compute uses *KVM* as the *hypervisor*. The compute node also runs the Networking plug-in and an agent that connect tenant networks to instances and provide firewalling (*security groups*) services. You can run more than one compute node.

Optionally, the compute node runs a Telemetry agent to collect metrics. Also, it can contain a third network interface on a separate storage network to improve performance of storage services.

- The optional Block Storage node contains the disks that the Block Storage service provisions for tenant virtual machine instances. You can run more than one of these nodes.

Optionally, the Block Storage node runs a Telemetry agent to collect metrics. Also, it can contain a second network interface on a separate storage network to improve performance of storage services.

- The optional Object Storage nodes contain the disks that the Object Storage service uses for storing accounts, containers, and objects. You can run more than two of these nodes. However, the minimal architecture example requires two nodes.

Optionally, these nodes can contain a second network interface on a separate storage network to improve performance of storage services.



Note

When you implement this architecture, skip [the section called “Legacy networking \(nova-network\)” \[83\]](#) in Chapter 6, “Add a networking compo-

ment” [60]. Optional services might require additional nodes or additional resources on existing nodes.

Figure 1.2. Minimal architecture example with OpenStack Networking (neutron)—Hardware requirements

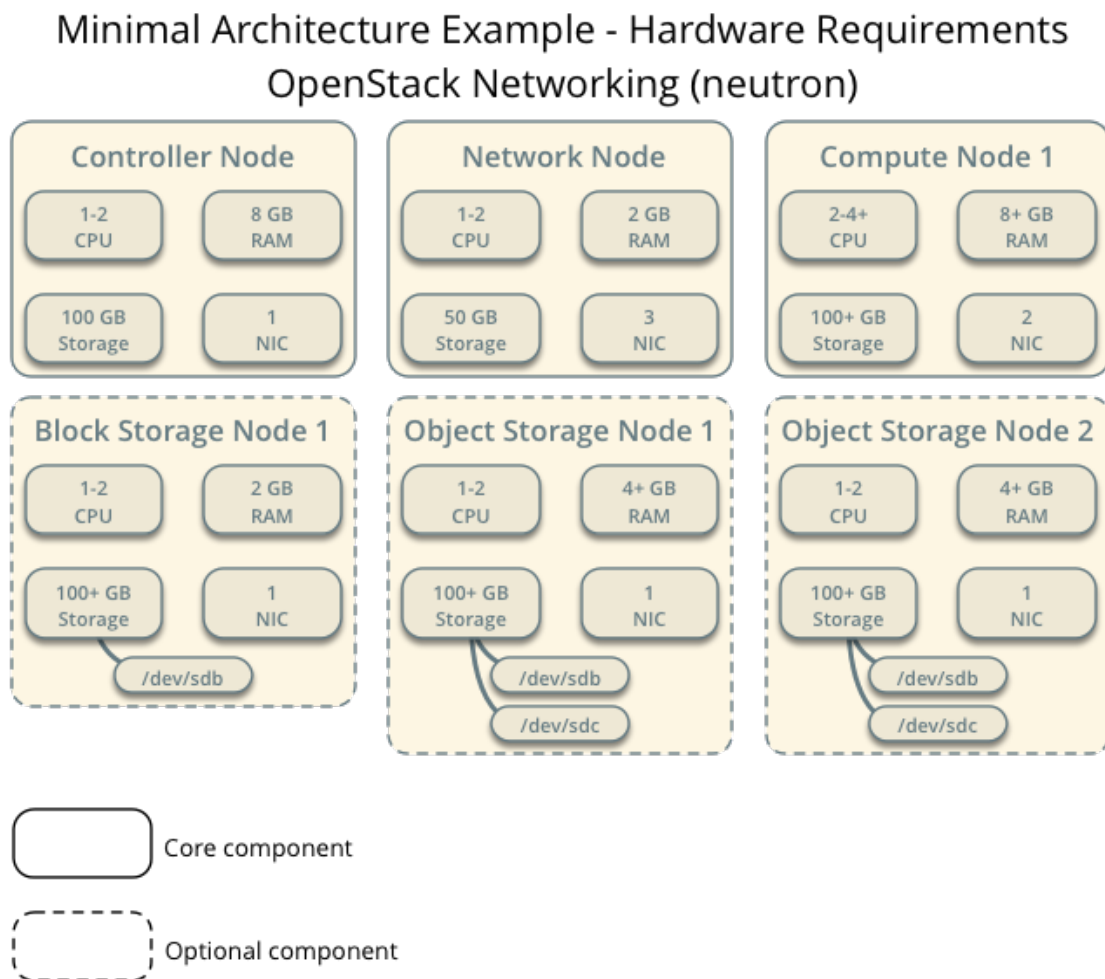


Figure 1.3. Minimal architecture example with OpenStack Networking (neutron)—Network layout

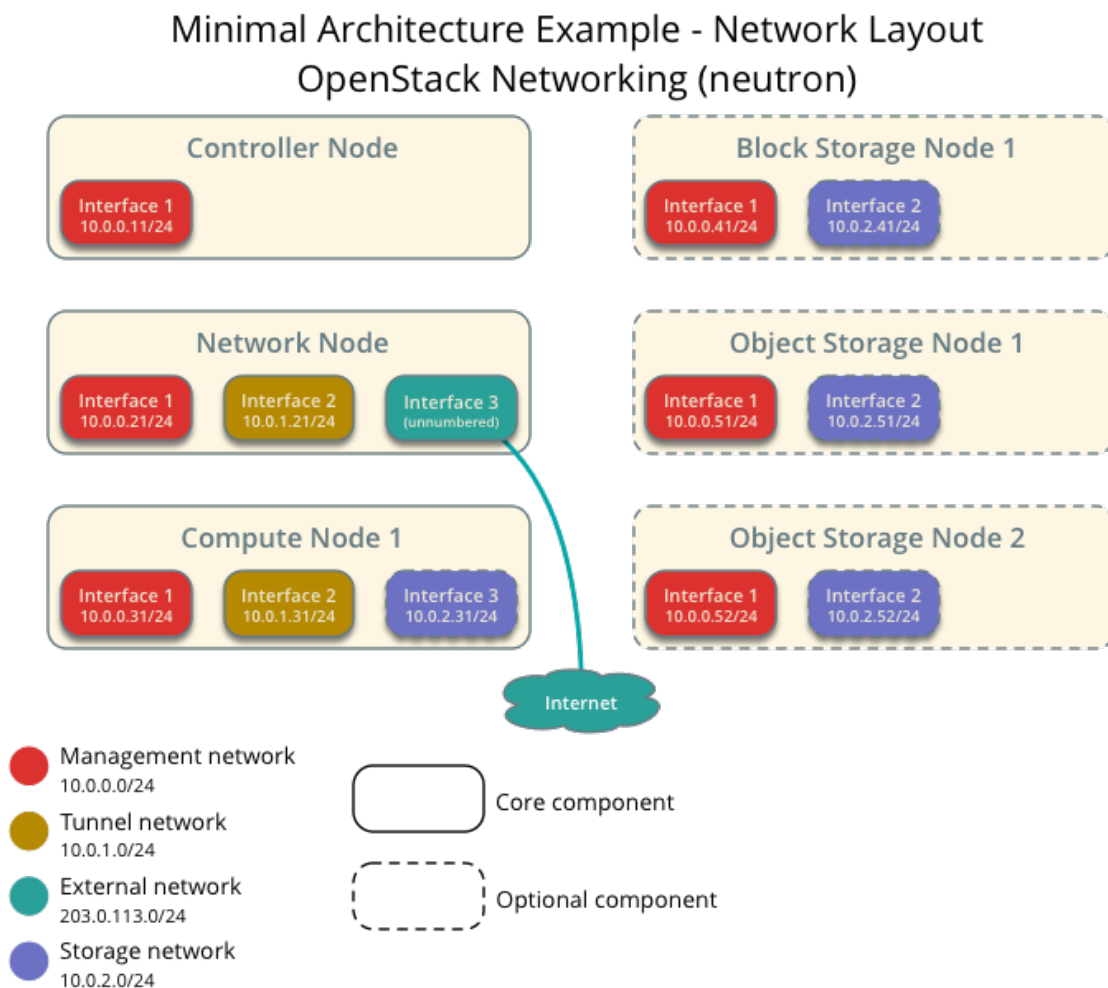
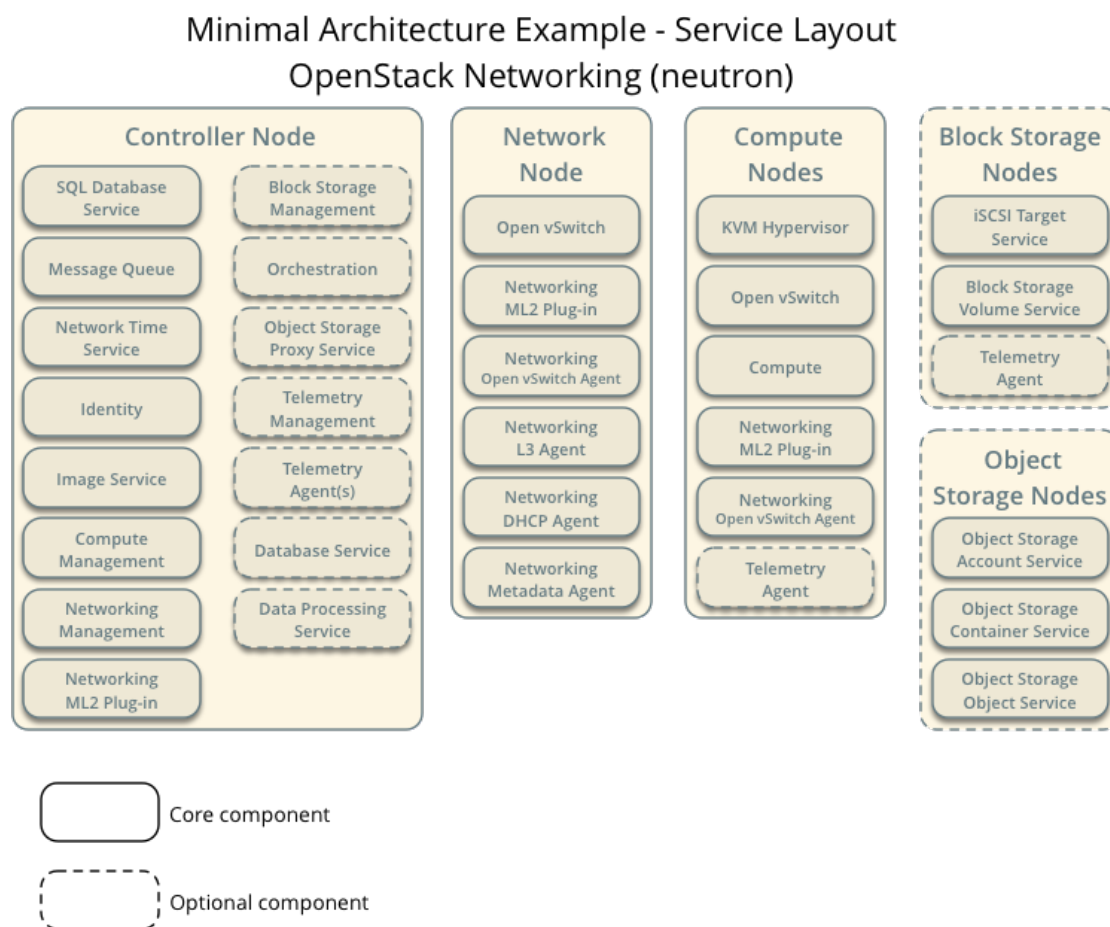


Figure 1.4. Minimal architecture example with OpenStack Networking (neutron)—Service layout

- Two-node architecture with legacy networking (nova-network) and optional nodes for Block Storage and Object Storage services.
- The *controller node* runs the Identity service, Image Service, management portion of Compute, and the dashboard. It also includes supporting services such as a SQL database, message queue, and Network Time Protocol (NTP).

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

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

Optionally, the compute node runs a Telemetry agent to collect metrics. Also, it can contain a third network interface on a separate storage network to improve performance of storage services.

- The optional Block Storage node contains the disks that the Block Storage service provisions for tenant virtual machine instances. You can run more than one of these nodes.

Optionally, the Block Storage node runs a Telemetry agent to collect metrics. Also, it can contain a second network interface on a separate storage network to improve performance of storage services.

- The optional Object Storage nodes contain the disks that the Object Storage service uses for storing accounts, containers, and objects. You can run more than two of these nodes. However, the minimal architecture example requires two nodes.

Optionally, these nodes can contain a second network interface on a separate storage network to improve performance of storage services.



Note

When you implement this architecture, skip [the section called “OpenStack Networking \(neutron\)” \[60\]](#) in [Chapter 6, “Add a networking component” \[60\]](#). To use optional services, you might need to build additional nodes, as described in subsequent chapters.

Figure 1.5. Minimal architecture example with legacy networking (nova-network)—Hardware requirements

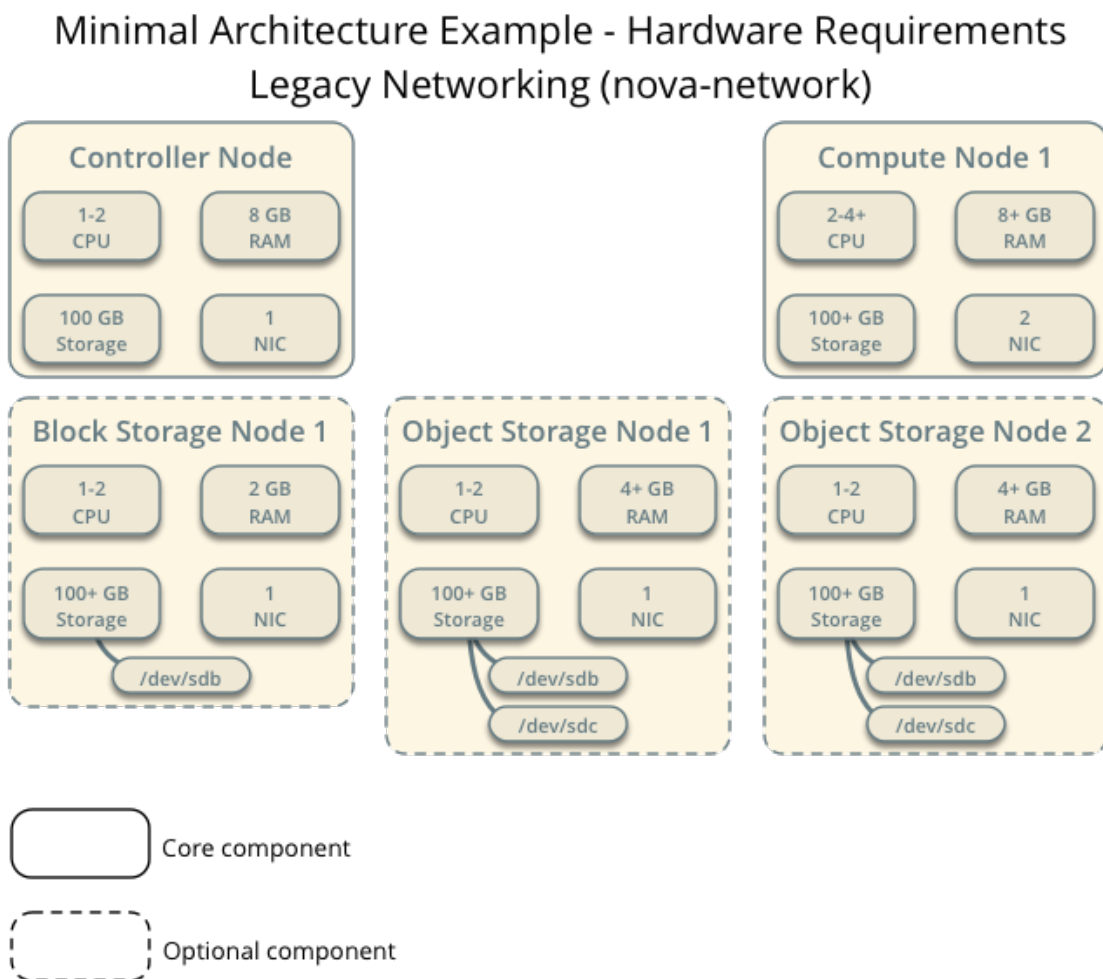


Figure 1.6. Minimal architecture example with legacy networking (nova-network)—Network layout

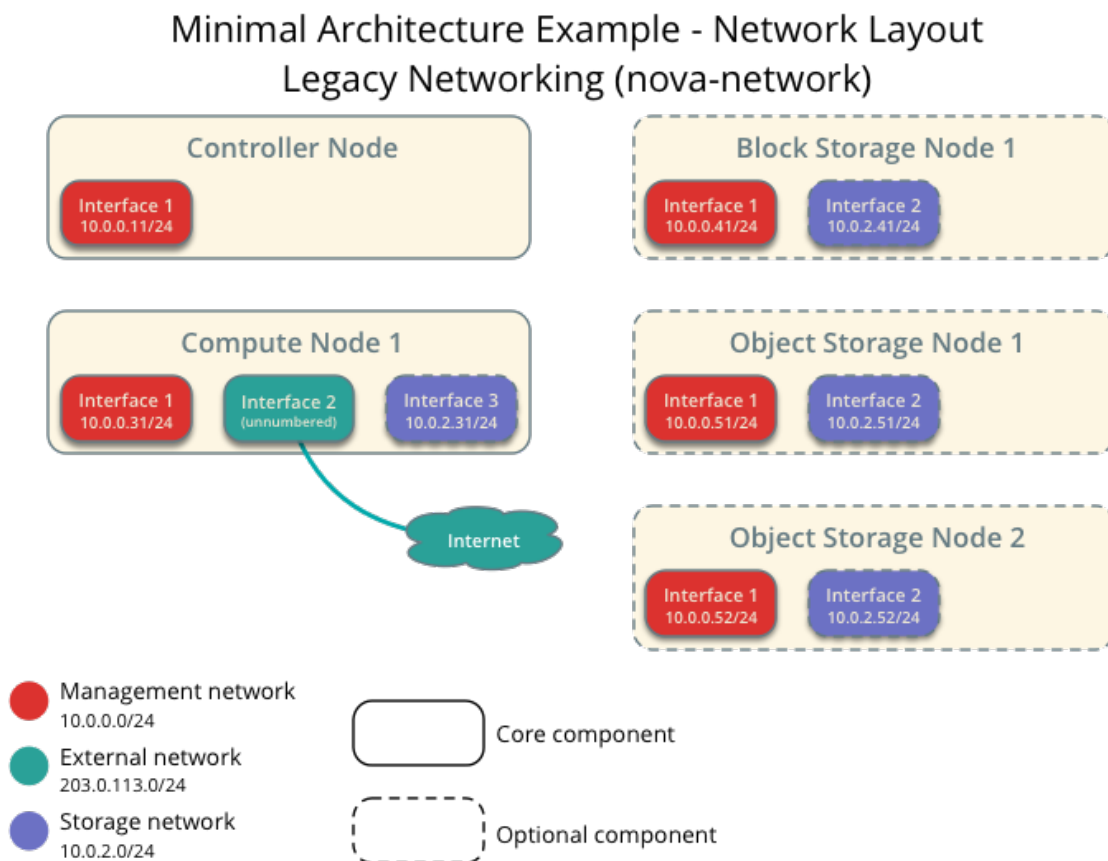
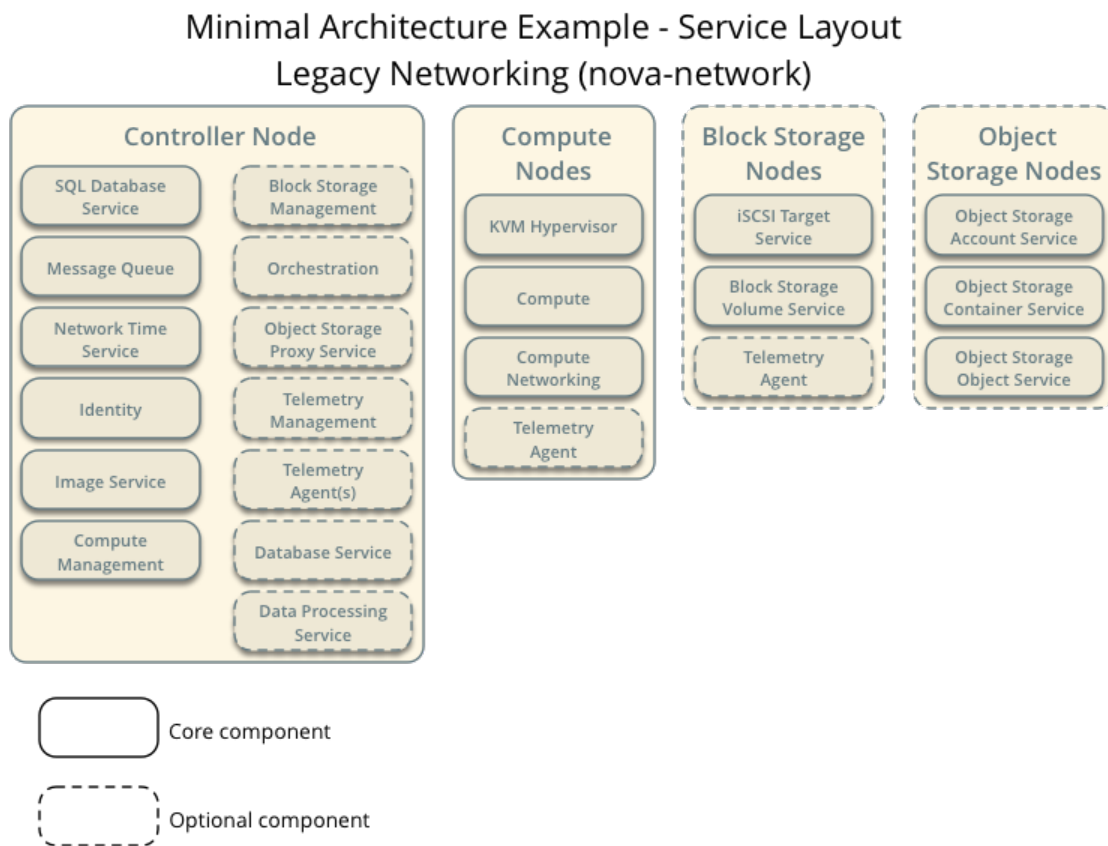


Figure 1.7. Minimal architecture example with legacy networking (nova-network)—Service layout



2. Basic environment

Table of Contents

Before you begin	11
Security	12
Networking	13
Network Time Protocol (NTP)	24
OpenStack packages	26
Database	27
Messaging server	28

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 Identity, Image Service, Compute, at least one networking service, and the dashboard, the Object Storage service can operate independently. If your use case only involves Object Storage, you can skip to [Chapter 9, “Add Object Storage” \[99\]](#) after configuring the appropriate nodes for it. However, the dashboard requires at least the 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.



Note

The `systemctl enable` call on openSUSE outputs a warning message when the service uses SysV Init scripts instead of native systemd files. This warning can be ignored.

Before you begin

For best performance, we recommend that your environment meets or exceeds the hardware requirements in [Figure 1.2, “Minimal architecture example with OpenStack Networking \(neutron\)—Hardware requirements” \[4\]](#) or [Figure 1.5, “Minimal architecture example with legacy networking \(nova-network\)—Hardware requirements” \[8\]](#). However, OpenStack does not require a significant amount of resources and the following minimum requirements should support a proof-of-concept environment with core services and several *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* and disables MAC address filtering on the *external network*.

For more information about system requirements, see the [OpenStack Operations Guide](#).

Security

OpenStack services support various security methods including password, policy, and encryption. Additionally, supporting services including the database server and message broker support at least password security.

To ease the installation process, this guide only covers password security where applicable. You can create secure passwords manually, generate them using a tool such as [pwgen](#), or by running the following command:

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

For OpenStack services, this guide uses `SERVICE_PASS` to reference service account passwords and `SERVICE_DBPASS` to reference database passwords.

The following table provides a list of services that require passwords and their associated references in the guide:

Table 2.1. Passwords

Password name	Description
Database password (no variable used)	Root password for the database

Password name	Description
<code>RABBIT_PASS</code>	Password of user <code>guest</code> of RabbitMQ
<code>KEYSTONE_DBPASS</code>	Database password of Identity service
<code>DEMO_PASS</code>	Password of user <code>demo</code>
<code>ADMIN_PASS</code>	Password of user <code>admin</code>
<code>GLANCE_DBPASS</code>	Database password for Image Service
<code>GLANCE_PASS</code>	Password of Image Service user <code>glance</code>
<code>NOVA_DBPASS</code>	Database password for Compute service
<code>NOVA_PASS</code>	Password of Compute service user <code>nova</code>
<code>DASH_DBPASS</code>	Database password for the dashboard
<code>CINDER_DBPASS</code>	Database password for the Block Storage service
<code>CINDER_PASS</code>	Password of Block Storage service user <code>cinder</code>
<code>NEUTRON_DBPASS</code>	Database password for the Networking service
<code>NEUTRON_PASS</code>	Password of Networking service user <code>neutron</code>
<code>HEAT_DBPASS</code>	Database password for the Orchestration service
<code>HEAT_PASS</code>	Password of Orchestration service user <code>heat</code>
<code>CEILOMETER_DBPASS</code>	Database password for the Telemetry service
<code>CEILOMETER_PASS</code>	Password of Telemetry service user <code>ceilometer</code>
<code>TROVE_DBPASS</code>	Database password of Database service
<code>TROVE_PASS</code>	Password of Database Service user <code>trove</code>

OpenStack and supporting services require administrative privileges during installation and operation. In some cases, services perform modifications to the host that can interfere with deployment automation tools such as Ansible, Chef, and Puppet. For example, some OpenStack services add a root wrapper to `sudo` that can interfere with security policies. See the [Cloud Administrator Guide](#) for more information. Also, the Networking service assumes default values for kernel network parameters and modifies firewall rules. To avoid most issues during your initial installation, we recommend using a stock deployment of a supported distribution on your hosts. However, if you choose to automate deployment of your hosts, review the configuration and policies applied to them before proceeding further.

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 [documentation](#).

Your distribution does not enable a restrictive *firewall* by default. For more information about securing your environment, refer to the [OpenStack Security Guide](#).

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



Note

All nodes require Internet access to install OpenStack packages and perform maintenance tasks such as periodic updates. In most cases, nodes should obtain

Internet access through the management network interface. For simplicity, the network diagrams in this guide only show Internet access for OpenStack network services.

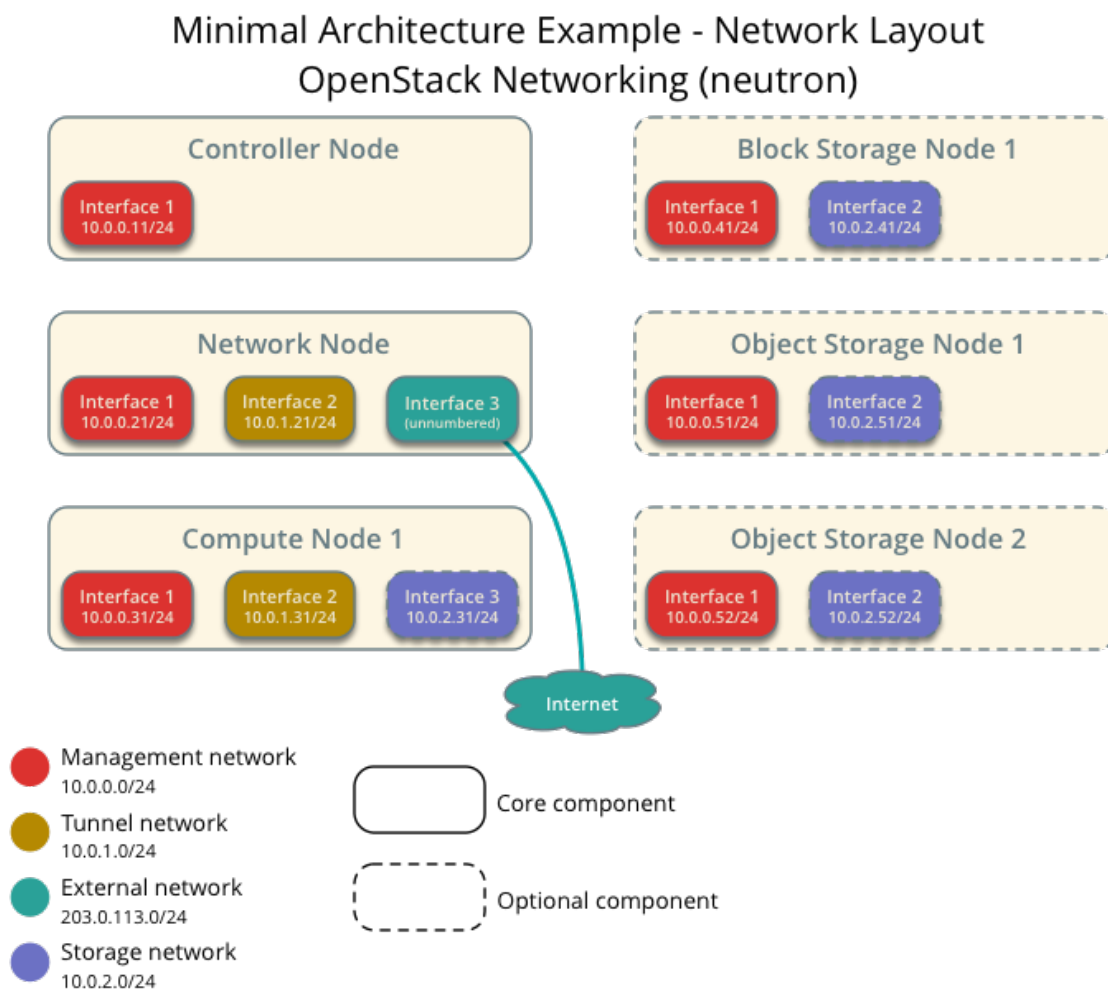
OpenStack Networking (neutron)

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. Minimal architecture example with OpenStack Networking (neutron)—Network layout

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:

1. 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

2. Reboot the system to activate the changes.

To configure name resolution:

1. Set the hostname of the node to *controller*.
2. 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
```



Warning

You must remove or comment the line beginning with `127.0.1.1`.

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/network/interfaces` file to contain the following:


```
# The external network interface
auto INTERFACE_NAME
iface INTERFACE_NAME inet manual
    up ip link set dev $IFACE up
    down ip link set dev $IFACE down
```

4. Reboot the system to activate the changes.

To configure name resolution:

1. Set the hostname of the node to `network`.
2. Edit the `/etc/hosts` file to contain the following:

```
# network
10.0.0.21      network

# controller
10.0.0.11     controller

# computel
10.0.0.31     computel
```



Warning

You must remove or comment the line beginning with `127.0.1.1`.

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.

3. Reboot the system to activate the changes.

To configure name resolution:

1. Set the hostname of the node to `compute1`.
2. 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
```



Warning

You must remove or comment the line beginning with `127.0.1.1`.

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
```

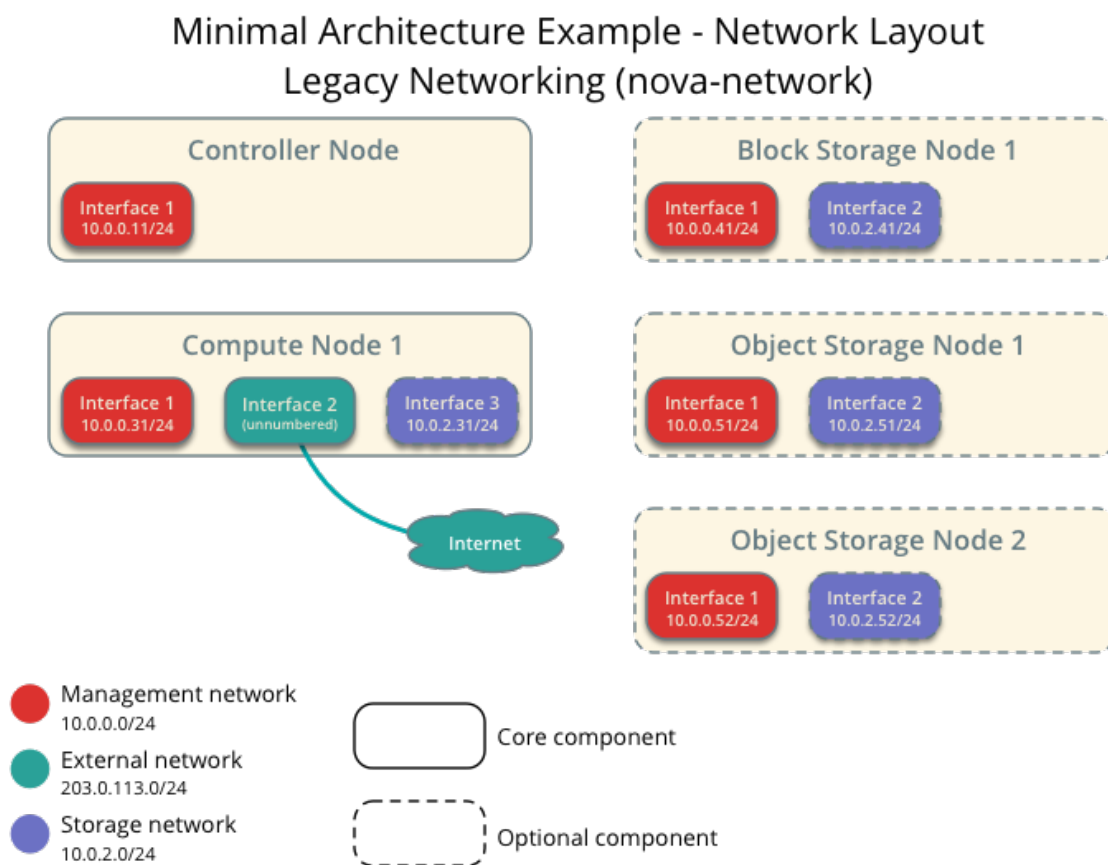
Legacy networking (nova-network)

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. Minimal architecture example with legacy networking (nova-network)—Network layout

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:

1. 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

2. Reboot the system to activate the changes.

To configure name resolution:

1. Set the hostname of the node to *controller*.
2. Edit the `/etc/hosts` file to contain the following:

```
# controller
10.0.0.11    controller

# compute1
10.0.0.31    compute1
```



Warning

You must remove or comment the line beginning with `127.0.1.1`.

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/network/interfaces` file to contain the following:

```
# The external network interface
auto INTERFACE_NAME
iface INTERFACE_NAME inet manual
    up ip link set dev $IFACE up
    down ip link set dev $IFACE down
```

3. Reboot the system to activate the changes.

To configure name resolution:

1. Set the hostname of the node to *compute1*.

2. Edit the `/etc/hosts` file to contain the following:

```
# compute1
10.0.0.31      compute1

# controller
10.0.0.11     controller
```



Warning

You must remove or comment the line beginning with `127.0.1.1`.

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)

You must install *NTP* to properly synchronize services among nodes. We recommend that you configure the controller node to reference more accurate (lower stratum) servers and other nodes to reference the controller node.

Controller node

To install the NTP service

- ```
apt-get install ntp
```

#### To configure the NTP service

By default, the controller node synchronizes the time via a pool of public servers. However, you can optionally edit the `/etc/ntp.conf` file to configure alternative servers such as those provided by your organization.

1. Edit the `/etc/ntp.conf` file and add, change, or remove the following keys as necessary for your environment:

```
server NTP_SERVER iburst
restrict -4 default kod notrap nomodify
restrict -6 default kod notrap nomodify
```

Replace `NTP_SERVER` with the hostname or IP address of a suitable more accurate (lower stratum) NTP server. The configuration supports multiple `server` keys.



#### Note

For the `restrict` keys, you essentially remove the `nopeer` and `noquery` options.



#### Note

Remove the `/var/lib/ntp/ntp.conf.dhcp` file if it exists.

2. Restart the NTP service:

```
service ntp restart
```



## Other nodes

### To install the NTP service

- ```
# apt-get install ntp
```

To configure the NTP service

Configure the network and compute nodes to reference the controller node.

1. Edit the `/etc/ntp.conf` file:

Comment out or remove all but one `server` key and change it to reference the controller node.

```
server controller iburst
```



Note

Remove the `/var/lib/ntp/ntp.conf.dhcp` file if it exists.

2. Restart the NTP service:

```
# service ntp restart
```

Verify operation

We recommend that you verify NTP synchronization before proceeding further. Some nodes, particularly those that reference the controller node, can take several minutes to synchronize.

1. Run this command on the *controller* node:

```
# ntpq -c peers
      remote           refid      st t when poll reach   delay   offset
  jitter
=====
====
*ntp-server1      192.0.2.11           2 u  169 1024   377    1.901   -0.611
  5.483
+ntp-server2      192.0.2.12           2 u  887 1024   377    0.922   -0.246
  2.864
```

Contents in the *remote* column should indicate the hostname or IP address of one or more NTP servers.



Note

Contents in the *refid* column typically reference IP addresses of upstream servers.

2. Run this command on the *controller* node:

```
# ntpq -c assoc
ind assid status  conf reach auth condition  last_event cnt
```

```
=====
1 20487 961a yes yes none sys.peer sys_peer 1
2 20488 941a yes yes none candidate sys_peer 1
=====
```

Contents in the *condition* column should indicate `sys.peer` for at least one server.

3. Run this command on *all other* nodes:

```
# ntpq -c peers
      remote          refid          st t when poll reach  delay  offset
=====
=====
*controller          192.0.2.21          3 u  47   64   37   0.308  -0.251
0.079
```

Contents in the *remote* column should indicate the hostname of the controller node.



Note

Contents in the *refid* column typically reference IP addresses of upstream servers.

4. Run this command on *all other* nodes:

```
# ntpq -c assoc
ind assid status  conf reach auth condition  last_event cnt
=====
1 21181 963a yes yes none sys.peer sys_peer 3
```

Contents in the *condition* column should indicate `sys.peer`.

OpenStack packages

Distributions release OpenStack packages as part of the distribution or using other methods because of differing release schedules. Perform these procedures on all nodes.



Note

Disable or remove any automatic update services because they can impact your OpenStack environment.

To enable the OpenStack repository

- Install the Ubuntu Cloud archive keyring and repository:

```
# apt-get install ubuntu-cloud-keyring
# echo "deb http://ubuntu-cloud.archive.canonical.com/ubuntu" \
"trusty-updates/juno main" > /etc/apt/sources.list.d/cloudarchive-juno.
list
```

To finalize installation

- Upgrade the packages on your system:

```
# apt-get update && apt-get dist-upgrade
```

**Note**

If the upgrade process includes a new kernel, reboot your system to activate it.

Database

Most OpenStack services use an SQL database to store information. The database typically runs on the controller node. The procedures in this guide use MariaDB or MySQL depending on the distribution. OpenStack services also support other SQL databases including [PostgreSQL](#).

To install and configure the database server

1. Install the packages:

**Note**

The Python MySQL library is compatible with MariaDB.

```
# apt-get install mariadb-server python-mysqldb
```

2. Choose a suitable password for the database root account.
3. Edit the `/etc/mysql/my.cnf` file and complete the following actions:
 - a. In 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. In the `[mysqld]` section, set the following keys to enable useful options and the UTF-8 character set:

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

To finalize installation

1. Restart the database service:

```
# service mysql restart
```

2. Secure the database service:

```
# mysql_secure_installation
```

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 RabbitMQ message broker which is 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 RabbitMQ message broker service

- ```
apt-get install rabbitmq-server
```

### To configure the message broker service

1. 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
Changing password for user "guest" ...
...done.
```

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.

2. For RabbitMQ version 3.3.0 or newer, you must enable remote access for the `guest` account.
  - a. Check the RabbitMQ version:

```
rabbitmqctl status | grep rabbit
Status of node 'rabbit@controller' ...
```

```
{running_applications,[{rabbit,"RabbitMQ","3.4.2"},
```

- b. If necessary, edit the `/etc/rabbitmq/rabbitmq.config` file and configure `loopback_users` to reference an empty list:

```
[{rabbit, [{loopback_users, []}]}].
```



### Note

Contents of the original file might vary depending on the source of the RabbitMQ package. In some cases, you might need to create this file.

- c. Restart the message broker service:

```
service rabbitmq-server restart
```

## 3. Add the Identity service

### Table of Contents

|                                                   |    |
|---------------------------------------------------|----|
| OpenStack Identity concepts .....                 | 30 |
| Install and configure .....                       | 32 |
| Create tenants, users, and roles .....            | 34 |
| Create the service entity and API endpoint .....  | 37 |
| Verify operation .....                            | 38 |
| Create OpenStack client environment scripts ..... | 40 |

### OpenStack Identity concepts

The *OpenStack Identity Service* performs the following functions:

- Tracking users and their permissions.
- Providing a catalog of available services with their API endpoints.

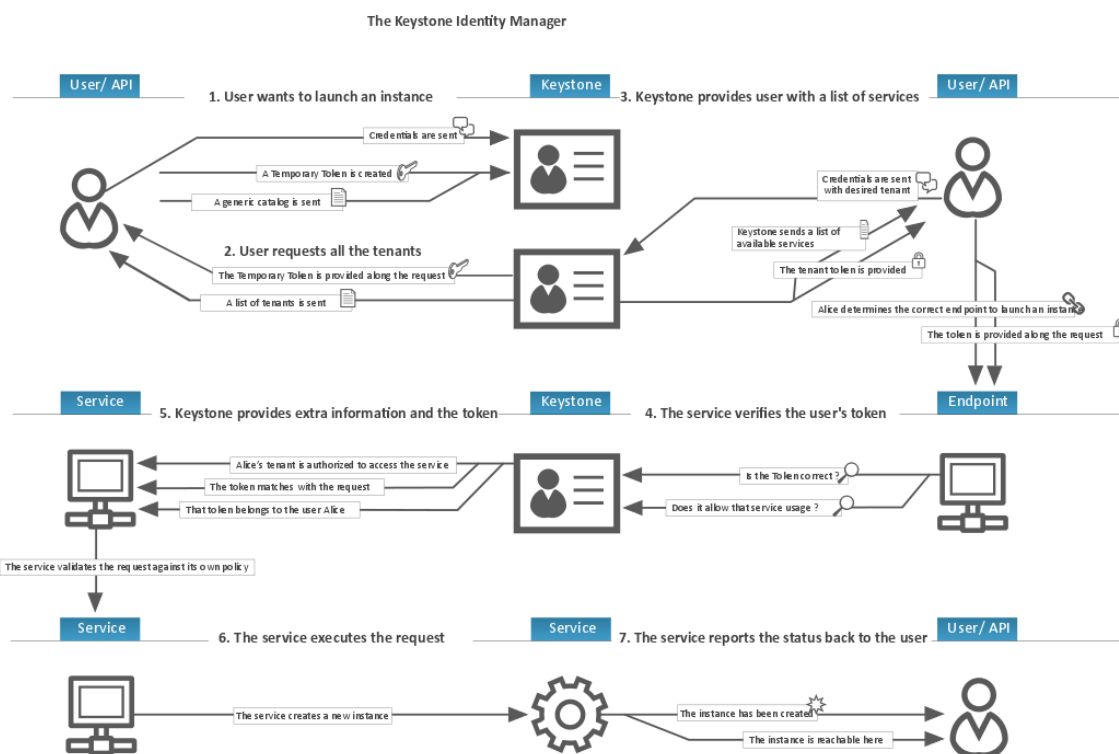
When installing OpenStack Identity service, you must register each service in your OpenStack installation. Identity service can then track which OpenStack services are installed, and where they are located on the network.

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

|                       |                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>User</b>           | 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.                   |
| <b>Credentials</b>    | Data that confirms the user's identity. For example: user name and password, user name and API key, or an authentication token provided by the Identity Service.                                                                                                                                                                                                                                      |
| <b>Authentication</b> | <p>The process of confirming the identity of a user. OpenStack Identity confirms an incoming request by validating a set of credentials supplied by the user.</p> <p>These credentials are initially a user name and password, or a user name and API key. When user credentials are validated, OpenStack Identity issues an authentication token which the user provides in subsequent requests.</p> |
| <b>Token</b>          | An alpha-numeric string of text used to access OpenStack APIs and resources. A token may be revoked at any time and is valid for a finite duration.                                                                                                                                                                                                                                                   |

|                        |                                                                                                                                                                                                                                                                                                                                                                                |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                        | <p>While OpenStack Identity supports token-based authentication in this release, the intention is to support additional protocols in the future. Its main purpose is to be an integration service, and not aspire to be a full-fledged identity store and management solution.</p>                                                                                             |
| <b>Tenant</b>          | <p>A container used to group or isolate resources. Tenants also group or isolate identity objects. Depending on the service operator, a tenant may map to a customer, account, organization, or project.</p>                                                                                                                                                                   |
| <b>Service</b>         | <p>An OpenStack service, such as Compute (nova), Object Storage (swift), or Image Service (glance). It provides one or more endpoints in which users can access resources and perform operations.</p>                                                                                                                                                                          |
| <b>Endpoint</b>        | <p>A network-accessible address where you access a service, usually a URL address. If you are using an extension for templates, an endpoint template can be created, which represents the templates of all the consumable services that are available across the regions.</p>                                                                                                  |
| <b>Role</b>            | <p>A personality with a defined set of user rights and privileges to perform a specific set of operations.</p> <p>In the Identity service, a token that is issued to a user includes the list of roles. 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.</p> |
| <b>Keystone Client</b> | <p>A command line interface for the OpenStack Identity API. For example, users can run the <b>keystone service-create</b> and <b>keystone endpoint-create</b> commands to register services in their OpenStack installations.</p>                                                                                                                                              |

The following diagram shows the OpenStack Identity process flow:



## Install and configure

This section describes how to install and configure the OpenStack Identity service on the controller node.

### To configure prerequisites

Before you configure the OpenStack Identity service, you must create a database and an administration token.

1. To create the database, complete these steps:
  - a. Use the database access client to connect to the database server as the `root` user:
 

```
$ mysql -u root -p
```
  - b. Create the `keystone` database:
 

```
CREATE DATABASE keystone;
```
  - c. Grant proper access to the `keystone` database:
 

```
GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost' \
IDENTIFIED BY 'KEYSTONE_DBPASS';
GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' \
IDENTIFIED BY 'KEYSTONE_DBPASS';
```

Replace `KEYSTONE_DBPASS` with a suitable password.



- d. Exit the database access client.
2. Generate a random value to use as the administration token during initial configuration:

```
openssl rand -hex 10
```

### To install and configure the components

1. Run the following command to install the packages:

```
apt-get install keystone python-keystoneclient
```

2. Edit the `/etc/keystone/keystone.conf` file and complete the following actions:

- a. In the `[DEFAULT]` section, define the value of the initial administration token:

```
[DEFAULT]
...
admin_token = ADMIN_TOKEN
```

Replace `ADMIN_TOKEN` with the random value that you generated in a previous step.

- b. In the `[database]` section, configure database access:

```
[database]
...
connection = mysql://keystone:KEYSTONE_DBPASS@controller/keystone
```

Replace `KEYSTONE_DBPASS` with the password you chose for the database.

- c. In the `[token]` section, configure the UUID token provider and SQL driver:

```
[token]
...
provider = keystone.token.providers.uuid.Provider
driver = keystone.token.persistence.backends.sql.Token
```

- d. In the `[revoke]` section, configure the SQL revocation driver:

```
[revoke]
...
driver = keystone.contrib.revoke.backends.sql.Revoke
```

- e. (Optional) To assist with troubleshooting, enable verbose logging in the `[DEFAULT]` section:

```
[DEFAULT]
...
verbose = True
```

3. Populate the Identity service database:

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

## To finalize installation

1. Restart the Identity service:

```
service keystone restart
```

2. By default, the Ubuntu packages create a SQLite database.

Because this configuration uses a SQL database server, you can remove the SQLite database file:

```
rm -f /var/lib/keystone/keystone.db
```

3. By default, the Identity service stores expired tokens in the database indefinitely. The accumulation of expired tokens considerably increases the database size and might degrade service performance, particularly in environments with limited resources.

We recommend that you use `cron` to configure a periodic task that purges expired tokens hourly:

```
(crontab -l -u keystone 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/crontabs/keystone
```

## Create tenants, users, and roles

After you install the Identity service, create *tenants* (projects), *users*, and *roles* for your environment. You must use the temporary administration token that you created in [the section called “Install and configure” \[32\]](#) and manually configure the location (endpoint) of the Identity service before you run **keystone** commands.

You can pass the value of the administration token to the **keystone** command with the `--os-token` option or set the temporary `OS_SERVICE_TOKEN` environment variable. Similarly, you can pass the location of the Identity service to the **keystone** command with the `--os-endpoint` option or set the temporary `OS_SERVICE_ENDPOINT` environment variable. This guide uses environment variables to reduce command length.

For more information, see the [Operations Guide - Managing Project and Users](#).

### To configure prerequisites

1. Configure the administration token:

```
$ export OS_SERVICE_TOKEN=ADMIN_TOKEN
```

Replace `ADMIN_TOKEN` with the administration token that you generated in [the section called “Install and configure” \[32\]](#). For example:

```
$ export OS_SERVICE_TOKEN=294a4c8a8a475f9b9836
```

2. Configure the endpoint:

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

## To create tenants, users, and roles

1. Create an administrative tenant, user, and role for administrative operations in your environment:

- a. Create the admin tenant:

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

| Property    | Value                            |
|-------------|----------------------------------|
| description | Admin Tenant                     |
| enabled     | True                             |
| id          | 6f4c1e4cbfef4d5a8a1345882fbca110 |
| name        | admin                            |



### Note

OpenStack generates IDs dynamically, so you will see different values from the example command output.

- b. Create the admin user:

```
$ keystone user-create --name admin --pass ADMIN_PASS --
email EMAIL_ADDRESS
```

| Property | Value                            |
|----------|----------------------------------|
| email    | admin@example.com                |
| enabled  | True                             |
| id       | ea8c352d253443118041c9c8b8416040 |
| name     | admin                            |
| username | admin                            |

Replace *ADMIN\_PASS* with a suitable password and *EMAIL\_ADDRESS* with a suitable e-mail address.

- c. Create the admin role:

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

| Property | Value                            |
|----------|----------------------------------|
| id       | bff3a6083b714fa29c9344bf8930d199 |
| name     | admin                            |

- d. Add the admin role to the admin tenant and user:

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



### Note

This command provides no output.

**Note**

Any roles that you create must map to roles specified in the `policy.json` file included with each OpenStack service. The default policy for most services grants administrative access to the `admin` role. For more information, see the [Operations Guide - Managing Projects and Users](#).

2. Create a demo tenant and user for typical operations in your environment:

- a. Create the demo tenant:

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

| Property    | Value                            |
|-------------|----------------------------------|
| description | Demo Tenant                      |
| enabled     | True                             |
| id          | 4aa51bb942be4dd0ac0555d7591f80a6 |
| name        | demo                             |

**Note**

Do not repeat this step when creating additional users for this tenant.

- b. Create the demo user under the demo tenant:

```
$ keystone user-create --name demo --tenant demo --pass DEMO_PASS --
email EMAIL_ADDRESS
```

| Property | Value                            |
|----------|----------------------------------|
| email    | demo@example.com                 |
| enabled  | True                             |
| id       | 7004dfa0dda84d63aef81cf7f100af01 |
| name     | demo                             |
| tenantId | 4aa51bb942be4dd0ac0555d7591f80a6 |
| username | demo                             |

Replace `DEMO_PASS` with a suitable password and `EMAIL_ADDRESS` with a suitable e-mail address.

**Note**

Using the `--tenant` option automatically assigns the `_member_` role to a user. This option will also create the `_member_` role if it does not exist.

**Note**

You can repeat this procedure to create additional tenants and users.

- OpenStack services also require a tenant, user, and role to interact with other services. Each service typically requires creating one or more unique users with the `admin` role under the `service` tenant.

- Create the `service` tenant:

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

| Property    | Value                            |
|-------------|----------------------------------|
| description | Service Tenant                   |
| enabled     | True                             |
| id          | 6b69202e1bf846a4ae50d65bc4789122 |
| name        | service                          |

## Create the service entity and API endpoint

After you create tenants, users, and roles, you must create the *service* entity and *API endpoints* for the Identity service.

### To configure prerequisites

- Set the `OS_SERVICE_TOKEN` and `OS_SERVICE_ENDPOINT` environment variables, as described in [the section called "Create tenants, users, and roles" \[34\]](#).

### To create the service entity and API endpoints

- The Identity service manages a catalog of services in your OpenStack environment. Services use this catalog to locate other services in your environment.

Create the service entity for the Identity service:

```
$ keystone service-create --name keystone --type identity \
--description "OpenStack Identity"
```

| Property    | Value                            |
|-------------|----------------------------------|
| description | OpenStack Identity               |
| enabled     | True                             |
| id          | 15c11a23667e427e91bc31335b45f4bd |
| name        | keystone                         |
| type        | identity                         |



### Note

Because OpenStack generates IDs dynamically, you will see different values from this example command output.

- The Identity service manages a catalog of API endpoints associated with the services in your OpenStack environment. Services use this catalog to determine how to communicate with other services in your environment.

OpenStack provides three API endpoint variations for each service: `admin`, `internal`, and `public`. In a production environment, the variants might reside on separate net-

works that service different types of users for security reasons. Also, OpenStack supports multiple regions for scalability. For simplicity, this configuration uses the management network for all endpoint variations and the `regionOne` region.

Create the Identity service API endpoints:

```
$ 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 \
 --region regionOne
```

| 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

This command references the ID of the service that you created in the previous step.



### Note

Each service that you add to your OpenStack environment requires adding information such as API endpoints to the Identity service. The sections of this guide that cover service installation include steps to add the appropriate information to the Identity service.

## Verify operation

This section describes how to verify operation of the Identity service.

1. Unset the temporary `OS_SERVICE_TOKEN` and `OS_SERVICE_ENDPOINT` environment variables:

```
$ unset OS_SERVICE_TOKEN OS_SERVICE_ENDPOINT
```

2. As the `admin` tenant and user, request an authentication token:

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

| Property  | Value                            |
|-----------|----------------------------------|
| expires   | 2014-10-10T12:50:12Z             |
| id        | 8963eb5ccd864769a894ec316ef8f7d4 |
| tenant_id | 6f4c1e4cbfef4d5a8a1345882fbca110 |
| user_id   | ea8c352d253443118041c9c8b8416040 |

Replace `ADMIN_PASS` with the password you chose for the `admin` user in the Identity service. You might need to use single quotes (') around your password if it includes special characters.

3. As the `admin` tenant and user, list tenants to verify that the `admin` tenant and user can execute admin-only CLI commands and that the Identity service contains the tenants that you created in [the section called "Create tenants, users, and roles" \[34\]](#):

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

| id                               | name    | enabled |
|----------------------------------|---------|---------|
| 6f4c1e4cbfef4d5a8a1345882fbca110 | admin   | True    |
| 4aa51bb942be4dd0ac0555d7591f80a6 | demo    | True    |
| 6b69202e1bf846a4ae50d65bc4789122 | service | True    |



### Note

Because OpenStack generates IDs dynamically, you will see different values from this example command output.

4. As the `admin` tenant and user, list users to verify that the Identity service contains the users that you created in [the section called "Create tenants, users, and roles" \[34\]](#):

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

| id                               | name  | enabled | email             |
|----------------------------------|-------|---------|-------------------|
| ea8c352d253443118041c9c8b8416040 | admin | True    | admin@example.com |
| 7004dfa0dda84d63aef81cf7f100af01 | demo  | True    | demo@example.com  |

5. As the `admin` tenant and user, list roles to verify that the Identity service contains the role that you created in [the section called "Create tenants, users, and roles" \[34\]](#):

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

| id                               | name     |
|----------------------------------|----------|
| 9fe2ff9ee4384b1894a90878d3e92bab | _member_ |
| bff3a6083b714fa29c9344bf8930d199 | admin    |

6. As the `demo` tenant and user, request an authentication token:

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

| Property  | Value                            |
|-----------|----------------------------------|
| expires   | 2014-10-10T12:51:33Z             |
| id        | 1b87ceae9e08411ba4a16e4dada04802 |
| tenant_id | 4aa51bb942be4dd0ac0555d7591f80a6 |
| user_id   | 7004dfa0dda84d63aef81cf7f100af01 |

Replace `DEMO_PASS` with the password you chose for the `demo` user in the Identity service.

7. As the `demo` tenant and user, attempt to list users to verify that you cannot execute admin-only CLI commands:

```
$ keystone --os-tenant-name demo --os-username demo --os-
password DEMO_PASS \
--os-auth-url http://controller:35357/v2.0 user-list
You are not authorized to perform the requested action, admin_required.
(HTTP 403)
```



### Note

Each OpenStack service references a `policy.json` file to determine the operations available to a particular tenant, user, or role. For more information, see the [Operations Guide - Managing Projects and Users](#).

## Create OpenStack client environment scripts

The previous section used a combination of environment variables and command options to interact with the Identity service via the **keystone** client. To increase efficiency of client operations, OpenStack supports simple client environment scripts also known as OpenRC files. These scripts typically contain common options for all clients, but also support unique options. For more information, see the [OpenStack User Guide](#).

### To create the scripts

Create client environment scripts for the `admin` and `demo` tenants and users. Future portions of this guide reference these scripts to load appropriate credentials for client operations.

1. Edit the `admin-openrc.sh` file and add the following content:

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

Replace `ADMIN_PASS` with the password you chose for the `admin` user in the Identity service.

2. Edit the `demo-openrc.sh` file and add the following content:



```
export OS_TENANT_NAME=demo
export OS_USERNAME=demo
export OS_PASSWORD=DEMO_PASS
export OS_AUTH_URL=http://controller:5000/v2.0
```

Replace `DEMO_PASS` with the password you chose for the `demo` user in the Identity service.



## Identity ports

Note the two different ports used above. Port 35357 is used for administrative functions only. Port 5000 is for normal user functions and is the most commonly used.

### To load client environment scripts

- To run clients as a certain tenant and user, you can simply load the associated client environment script prior to running them. For example, to load the location of the Identity service and `admin` tenant and user credentials:

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

## 4. Add the Image Service

### Table of Contents

|                               |    |
|-------------------------------|----|
| OpenStack Image Service ..... | 42 |
| Install and configure .....   | 43 |
| Verify operation .....        | 47 |

The OpenStack Image Service (*glance*) enables users to discover, register, and retrieve virtual machine images. It 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 describes configuring the Image Service to use the *file* back end, which uploads and stores in a directory on the controller node hosting the Image Service. By default, this directory is `/var/lib/glance/images/`.

Before you proceed, ensure that the controller node has at least several gigabytes of space available in this directory.

For information on requirements for other back ends, see [Configuration Reference](#).

### OpenStack Image Service

The OpenStack Image Service is central to Infrastructure-as-a-Service (IaaS) as shown in [Figure 1.1, “Conceptual architecture” \[2\]](#). It accepts API requests for disk or server images, and image metadata from end users or OpenStack Compute components. It also supports the storage of disk or server images on various repository types, including OpenStack Object Storage.

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

The OpenStack Image Service includes the following components:

|                        |                                                                                                        |
|------------------------|--------------------------------------------------------------------------------------------------------|
| <b>glance-api</b>      | Accepts Image API calls for image discovery, retrieval, and storage.                                   |
| <b>glance-registry</b> | 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 for use by OpenStack Image Service. Do not disclose it to users.

### Database

Stores image metadata and you can choose your database depending on your preference. Most deployments use MySQL or SQLite.

### Storage repository for image files

Various repository types are supported including normal file systems, Object Storage, RADOS block devices, HTTP, and Amazon S3. Note that some repositories will only support read-only usage.

## Install and configure

This section describes how to install and configure the Image Service, code-named glance, on the controller node. For simplicity, this configuration stores images on the local file system.



## Note

This section assumes proper installation, configuration, and operation of the Identity service as described in [the section called “Install and configure” \[32\]](#) and [the section called “Verify operation” \[38\]](#).

## To configure prerequisites

Before you install and configure the Image Service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:
  - a. Use the database access client to connect to the database server as the `root` user:

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

- b. Create the `glance` database:

```
CREATE DATABASE glance;
```

- c. Grant proper access to the `glance` database:

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

Replace `GLANCE_DBPASS` with a suitable password.

- d. Exit the database access client.
2. Source the `admin` credentials to gain access to admin-only CLI commands:

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

3. To create the service credentials, complete these steps:

a. Create the glance user:

```
$ keystone user-create --name glance --pass GLANCE_PASS
```

| Property | Value                            |
|----------|----------------------------------|
| email    |                                  |
| enabled  | True                             |
| id       | f89cca5865dc42b18e2421fa5f5cce66 |
| name     | glance                           |
| username | glance                           |

Replace *GLANCE\_PASS* with a suitable password.

b. Add the admin role to the glance user:

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



### Note

This command provides no output.

c. Create the glance service entity:

```
$ keystone service-create --name glance --type image \
 --description "OpenStack Image Service"
```

| Property    | Value                            |
|-------------|----------------------------------|
| description | OpenStack Image Service          |
| enabled     | True                             |
| id          | 23f409c4e79f4c9e9d23d809c50fbacf |
| name        | glance                           |
| type        | image                            |

4. Create the Image Service API endpoints:

```
$ keystone endpoint-create \
 --service-id $(keystone service-list | awk '/ image / {print $2}') \
 --publicurl http://controller:9292 \
 --internalurl http://controller:9292 \
 --adminurl http://controller:9292 \
 --region regionOne
```

| Property    | Value                            |
|-------------|----------------------------------|
| adminurl    | http://controller:9292           |
| id          | a2ee818c69cb475199a1ca108332eb35 |
| internalurl | http://controller:9292           |
| publicurl   | http://controller:9292           |
| region      | regionOne                        |
| service_id  | 23f409c4e79f4c9e9d23d809c50fbacf |

## To install and configure the Image Service components

1. Install the packages:

```
apt-get install glance python-glanceclient
```

2. Edit the `/etc/glance/glance-api.conf` file and complete the following actions:

- a. In the `[database]` section, configure database access:

```
[database]
...
connection = mysql://glance:GLANCE_DBPASS@controller/glance
```

Replace `GLANCE_DBPASS` with the password you chose for the Image Service database.

- b. In the `[keystone_authtoken]` and `[paste_deploy]` sections, configure Identity service access:

```
[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = glance
admin_password = GLANCE_PASS

[paste_deploy]
...
flavor = keystone
```

Replace `GLANCE_PASS` with the password you chose for the `glance` user in the Identity service.



### Note

Comment out any `auth_host`, `auth_port`, and `auth_protocol` options because the `identity_uri` option replaces them.

- c. In the `[glance_store]` section, configure the local file system store and location of image files:

```
[glance_store]
...
default_store = file
filesystem_store_datadir = /var/lib/glance/images/
```

- d. In the `[DEFAULT]` section, configure the `noop` notification driver to disable notifications because they only pertain to the optional Telemetry service:

```
[DEFAULT]
...
notification_driver = noop
```

The Telemetry chapter provides an Image Service configuration that enables notifications.

- e. (Optional) To assist with troubleshooting, enable verbose logging in the [DE-FAULT] section:

```
[DEFAULT]
...
verbose = True
```

3. Edit the `/etc/glance/glance-registry.conf` file and complete the following actions:

- a. In the [database] section, configure database access:

```
[database]
...
connection = mysql://glance:GLANCE_DBPASS@controller/glance
```

Replace `GLANCE_DBPASS` with the password you chose for the Image Service database.

- b. In the [keystone\_authtoken] and [paste\_deploy] sections, configure Identity service access:

```
[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = glance
admin_password = GLANCE_PASS

[paste_deploy]
...
flavor = keystone
```

Replace `GLANCE_PASS` with the password you chose for the `glance` user in the Identity service.



### Note

Comment out any `auth_host`, `auth_port`, and `auth_protocol` options because the `identity_uri` option replaces them.

- c. In the [DEFAULT] section, configure the `noop` notification driver to disable notifications because they only pertain to the optional Telemetry service:

```
[DEFAULT]
...
notification_driver = noop
```

The Telemetry chapter provides an Image Service configuration that enables notifications.

- d. (Optional) To assist with troubleshooting, enable verbose logging in the [DE-FAULT] section:

```
[DEFAULT]
...
verbose = True
```

4. Populate the Image Service database:

```
su -s /bin/sh -c "glance-manage db_sync" glance
```

## To finalize installation

1. Restart the Image Service services:

```
service glance-registry restart
service glance-api restart
```

2. By default, the Ubuntu packages create an SQLite database.

Because this configuration uses a SQL database server, you can remove the SQLite database file:

```
rm -f /var/lib/glance/glance.sqlite
```

## Verify operation

This section describes how to verify operation of the Image Service using [Cirros](#), a small Linux image that helps you test your OpenStack deployment.

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. Create and change into a temporary local directory:

```
$ mkdir /tmp/images
```

2. Download the image to the temporary local directory:

```
$ wget -P /tmp/images http://cdn.download.cirros-cloud.net/0.3.3/cirros-0.3.3-x86_64-disk.img
```

3. Source the admin credentials to gain access to admin-only CLI commands:

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

4. Upload the image to the Image Service:

```
$ glance image-create --name "cirros-0.3.3-x86_64" --file /tmp/images/
cirros-0.3.3-x86_64-disk.img \
 --disk-format qcow2 --container-format bare --is-public True --progress
[=====>] 100%
+-----+-----+
| Property | Value |
+-----+-----+
checksum	133eae9fb1c98f45894a4e60d8736619
container_format	bare
created_at	2014-10-10T13:14:42
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-0.3.3-x86_64                  |
| owner        | ea8c352d253443118041c9c8b8416040     |
| protected    | False                                |
| size         | 13200896                             |
| status       | active                               |
| updated_at   | 2014-10-10T13:14:43                  |
| virtual_size | None                                 |

For information about the parameters for the **glance image-create** command, see [Image Service command-line client](#) in the *OpenStack Command-Line Interface Reference*.

For information about disk and container formats for images, see [Disk and container formats for images](#) in the *OpenStack Virtual Machine Image Guide*.



### Note

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

5. Confirm upload of the image and validate attributes:

```
$ glance image-list
+-----+-----+-----+-----+
| ID | Container Format | Size | Status | Name | Disk Format |
+-----+-----+-----+-----+-----+
| acafc7c0-40aa-4026-9673-b879898e1fc2 | bare | 13200896 | active | cirros-0.3.3-x86_64 | qcow2 |
+-----+-----+-----+-----+-----+
```

6. Remove the temporary local directory:

```
$ rm -r /tmp/images
```



## 5. Add the Compute service

### Table of Contents

|                                             |    |
|---------------------------------------------|----|
| OpenStack Compute .....                     | 49 |
| Install and configure controller node ..... | 52 |
| Install and configure a compute node .....  | 55 |
| Verify operation .....                      | 58 |

### OpenStack Compute

Use OpenStack Compute to host and manage cloud computing systems. OpenStack Compute is a major part of an Infrastructure-as-a-Service (IaaS) system. The main modules are implemented in Python.

OpenStack Compute interacts with OpenStack Identity for authentication, OpenStack Image Service for disk and server images, and OpenStack dashboard for the user and administrative interface. Image access is limited by projects, and by users; quotas are limited per project (the number of instances, for example). OpenStack Compute can scale horizontally on standard hardware, and download images to launch instances.

OpenStack Compute consists of the following areas and their components:

#### API

##### **nova-api service**

Accepts and responds to end user compute API calls. The service supports the OpenStack Compute API, the Amazon EC2 API, and a special Admin API for privileged users to perform administrative actions. It enforces some policies and initiates most orchestration activities, such as running an instance.

##### **nova-api-metadata service**

Accepts metadata requests from instances. The `nova-api-metadata` service is generally used when you run in multi-host mode with `nova-network` installations. For details, see [Metadata service](#) in the *OpenStack Cloud Administrator Guide*.

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

#### Compute core

##### **nova-compute service**

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

Processing is fairly complex. Basically, the daemon accepts actions from the queue and performs a series of system commands such as launching a KVM instance and updating its state in the database.

**nova-scheduler service**

Takes a virtual machine instance request from the queue and determines on which compute server host it runs.

**nova-conductor module**

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

**Networking for VMs****nova-network worker daemon**

Similar to the `nova-compute` service, accepts networking tasks from the queue and manipulates the network. Performs tasks such as setting up bridging interfaces or changing IPtables rules.

## Console interface

|                                          |                                                                                                                                                                                                                                                                                                                                                                            |
|------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>nova-consoleauth daemon</code>     | Authorizes tokens for users that console proxies provide. See <code>nova-novncproxy</code> and <code>nova-xvncproxy</code> . This service must be running for console proxies to work. You can run proxies of either type against a single <code>nova-consoleauth</code> service in a cluster configuration. For information, see <a href="#">About nova-consoleauth</a> . |
| <code>nova-novncproxy daemon</code>      | Provides a proxy for accessing running instances through a VNC connection. Supports browser-based novnc clients.                                                                                                                                                                                                                                                           |
| <code>nova-spicehtml5proxy daemon</code> | Provides a proxy for accessing running instances through a SPICE connection. Supports browser-based HTML5 client.                                                                                                                                                                                                                                                          |
| <code>nova-xvpnvncproxy daemon</code>    | Provides a proxy for accessing running instances through a VNC connection. Supports an OpenStack-specific Java client.                                                                                                                                                                                                                                                     |
| <code>nova-cert daemon</code>            | x509 certificates.                                                                                                                                                                                                                                                                                                                                                         |

## Image management (EC2 scenario)

|                                      |                                                                                                                                                                                                                                                                                                                |
|--------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>nova-objectstore daemon</code> | An S3 interface for registering images with the OpenStack Image Service. Used primarily for installations that must support euca2ools. The euca2ools tools talk to <code>nova-objectstore</code> in <i>S3 language</i> , and <code>nova-objectstore</code> translates S3 requests into Image Service requests. |
| <code>euca2ools client</code>        | A set of command-line interpreter commands for managing cloud resources. Although it is not an OpenStack module, you can configure <code>nova-api</code> to support this EC2 interface. For more information, see the <a href="#">Eucalyptus 3.4 Documentation</a> .                                           |

## Command-line clients and other interfaces

|                          |                                                                         |
|--------------------------|-------------------------------------------------------------------------|
| <code>nova client</code> | Enables users to submit commands as a tenant administrator or end user. |
|--------------------------|-------------------------------------------------------------------------|

## Other components

|                     |                                                                                                                                                                                                                             |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>The queue</b>    | A central hub for passing messages between daemons. Usually implemented with <a href="#">RabbitMQ</a> , but can be implemented with an AMQP message queue, such as <a href="#">Apache Qpid</a> or <a href="#">Zero MQ</a> . |
| <b>SQL database</b> | Stores most build-time and run-time states for a cloud infrastructure, including: <ul style="list-style-type: none"><li>• Available instance types</li><li>• Instances in use</li></ul>                                     |

- Available networks
- Projects

Theoretically, OpenStack Compute can support any database that SQLAlchemy supports. Common databases are SQLite3 for test and development work, MySQL, and PostgreSQL.

## Install and configure controller node

This section describes how to install and configure the Compute service, code-named nova, on the controller node.

### To configure prerequisites

Before you install and configure the Compute service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:
  - a. Use the database access client to connect to the database server as the `root` user:

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

- b. Create the `nova` database:

```
CREATE DATABASE nova;
```

- c. Grant proper access to the `nova` database:

```
GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'localhost' \
 IDENTIFIED BY 'NOVA_DBPASS';
GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%' \
 IDENTIFIED BY 'NOVA_DBPASS';
```

Replace `NOVA_DBPASS` with a suitable password.

- d. Exit the database access client.
2. Source the `admin` credentials to gain access to admin-only CLI commands:

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

3. To create the service credentials, complete these steps:

- a. Create the `nova` user:

```
$ keystone user-create --name nova --pass NOVA_PASS
```

| Property | Value                            |
|----------|----------------------------------|
| email    |                                  |
| enabled  | True                             |
| id       | 387dd4f7e46d4f72965ee99c76ae748c |
| name     | nova                             |
| username | nova                             |

Replace `NOVA_PASS` with a suitable password.

- b. Add the admin role to the nova user:

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



### Note

This command provides no output.

- c. Create the nova service entity:

```
$ keystone service-create --name nova --type compute \
 --description "OpenStack Compute"
```

| Property    | Value                            |
|-------------|----------------------------------|
| description | OpenStack Compute                |
| enabled     | True                             |
| id          | 6c7854f52ce84db795557ebc0373f6b9 |
| name        | nova                             |
| type        | compute                          |

4. Create the Compute service API endpoints:

```
$ 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 \
 --region regionOne
```

| Property    | Value                                   |
|-------------|-----------------------------------------|
| adminurl    | http://controller:8774/v2/%(tenant_id)s |
| id          | c397438bd82c41198ec1a9d85cb7cc74        |
| internalurl | http://controller:8774/v2/%(tenant_id)s |
| publicurl   | http://controller:8774/v2/%(tenant_id)s |
| region      | regionOne                               |
| service_id  | 6c7854f52ce84db795557ebc0373f6b9        |

## To install and configure Compute controller components

1. Install the packages:

```
apt-get install nova-api nova-cert nova-conductor nova-consoleauth \
 nova-novncproxy nova-scheduler python-novaclient
```

2. Edit the `/etc/nova/nova.conf` file and complete the following actions:

- a. In the `[database]` section, configure database access:

```
[database]
...
connection = mysql://nova:NOVA_DBPASS@controller/nova
```

Replace `NOVA_DBPASS` with the password you chose for the Compute database.

- b. In the `[DEFAULT]` section, configure RabbitMQ message broker access:

```
[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

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

- c. In the `[DEFAULT]` and `[keystone_authtoken]` sections, configure Identity service access:

```
[DEFAULT]
...
auth_strategy = keystone

[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = nova
admin_password = NOVA_PASS
```

Replace `NOVA_PASS` with the password you chose for the `nova` user in the Identity service.



### Note

Comment out any `auth_host`, `auth_port`, and `auth_protocol` options because the `identity_uri` option replaces them.

- d. In the `[DEFAULT]` section, configure the `my_ip` option to use the management interface IP address of the controller node:

```
[DEFAULT]
...
my_ip = 10.0.0.11
```

- e. In the `[DEFAULT]` section, configure the VNC proxy to use the management interface IP address of the controller node:

```
[DEFAULT]
...
vncserver_listen = 10.0.0.11
vncserver_proxyclient_address = 10.0.0.11
```

- f. In the `[glance]` section, configure the location of the Image Service:

```
[glance]
...
host = controller
```

- g. (Optional) To assist with troubleshooting, enable verbose logging in the [DEFAULT] section:

```
[DEFAULT]
...
verbose = True
```

3. Populate the Compute database:

```
su -s /bin/sh -c "nova-manage db sync" nova
```

### To finalize installation

1. Restart the Compute services:

```
service nova-api restart
service nova-cert restart
service nova-consoleauth restart
service nova-scheduler restart
service nova-conductor restart
service nova-novncproxy restart
```

2. By default, the Ubuntu packages create an SQLite database.

Because this configuration uses a SQL database server, you can remove the SQLite database file:

```
rm -f /var/lib/nova/nova.sqlite
```

## Install and configure a compute node

This section describes how to install and configure the Compute service on a compute node. The service supports several *hypervisors* to deploy *instances* or *VMs*. For simplicity, this configuration uses the *QEMU* hypervisor with the *KVM* extension on compute nodes that support hardware acceleration for virtual machines. On legacy hardware, this configuration uses the generic QEMU hypervisor. You can follow these instructions with minor modifications to horizontally scale your environment with additional compute nodes.



### Note

This section assumes that you are following the instructions in this guide step-by-step to configure the first compute node. If you want to configure additional compute nodes, prepare them in a similar fashion to the first compute node in the [example architectures](#) section using the same networking service as your existing environment. For either networking service, follow the [NTP configuration](#) and [OpenStack packages](#) instructions. For OpenStack Networking (neutron), also follow the [OpenStack Networking compute node](#) instructions. For legacy networking (nova-network), also follow the [legacy networking compute node](#) instructions. Each additional compute node requires unique IP addresses.

### To install and configure the Compute hypervisor components

1. Install the packages:

```
apt-get install nova-compute sysfsutils
```

2. Edit the `/etc/nova/nova.conf` file and complete the following actions:
  - a. In the `[DEFAULT]` section, configure RabbitMQ message broker access:

```
[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

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

- b. In the `[DEFAULT]` and `[keystone_authtoken]` sections, configure Identity service access:

```
[DEFAULT]
...
auth_strategy = keystone

[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = nova
admin_password = NOVA_PASS
```

Replace `NOVA_PASS` with the password you chose for the `nova` user in the Identity service.



### Note

Comment out any `auth_host`, `auth_port`, and `auth_protocol` options because the `identity_uri` option replaces them.

- c. In the `[DEFAULT]` section, configure the `my_ip` option:

```
[DEFAULT]
...
my_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
```

Replace `MANAGEMENT_INTERFACE_IP_ADDRESS` with the IP address of the management network interface on your compute node, typically `10.0.0.31` for the first node in the [example architecture](#).

- d. In the `[DEFAULT]` section, enable and configure remote console access:

```
[DEFAULT]
...
vnc_enabled = True
vncserver_listen = 0.0.0.0
vncserver_proxyclient_address = MANAGEMENT_INTERFACE_IP_ADDRESS
novncproxy_base_url = http://controller:6080/vnc_auto.html
```



The server component listens on all IP addresses and the proxy component only listens on the management interface IP address of the compute node. The base URL indicates the location where you can use a web browser to access remote consoles of instances on this compute node.

Replace `MANAGEMENT_INTERFACE_IP_ADDRESS` with the IP address of the management network interface on your compute node, typically 10.0.0.31 for the first node in the [example architecture](#).



### Note

If the web browser to access remote consoles resides on a host that cannot resolve the `controller` hostname, you must replace `controller` with the management interface IP address of the controller node.

- e. In the `[glance]` section, configure the location of the Image Service:

```
[glance]
...
host = controller
```

- f. (Optional) To assist with troubleshooting, enable verbose logging in the `[DEFAULT]` section:

```
[DEFAULT]
...
verbose = True
```

## To finalize installation

1. Determine whether your compute node supports hardware acceleration for virtual machines:

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

If this command returns a value of *one or greater*, your compute node supports hardware acceleration which typically requires no additional configuration.

If this command returns a value of *zero*, your compute node does not support hardware acceleration and you must configure `libvirt` to use QEMU instead of KVM.

- Edit the `[libvirt]` section in the `/etc/nova/nova-compute.conf` file as follows:

```
[libvirt]
...
virt_type = qemu
```

2. Restart the Compute service:

```
service nova-compute restart
```

3. By default, the Ubuntu packages create an SQLite database.

Because this configuration uses a SQL database server, you can remove the SQLite database file:

```
rm -f /var/lib/nova/nova.sqlite
```

## Verify operation

This section describes how to verify operation of the Compute service.



### Note

Perform these commands on the controller node.

1. Source the admin credentials to gain access to admin-only CLI commands:

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

2. List service components to verify successful launch of each process:

```
$ nova service-list
```

|   | Id                         | Binary           | Host            | Zone     | Status  | State |
|---|----------------------------|------------------|-----------------|----------|---------|-------|
|   | Updated_at                 |                  | Disabled Reason |          |         |       |
| 1 | 2014-09-16T23:54:02.000000 | nova-conductor   | controller      | internal | enabled | up    |
| 2 | 2014-09-16T23:54:04.000000 | nova-consoleauth | controller      | internal | enabled | up    |
| 3 | 2014-09-16T23:54:07.000000 | nova-scheduler   | controller      | internal | enabled | up    |
| 4 | 2014-09-16T23:54:00.000000 | nova-cert        | controller      | internal | enabled | up    |
| 5 | 2014-09-16T23:54:06.000000 | nova-compute     | compute1        | nova     | enabled | up    |



### Note

This output should indicate four components enabled on the controller node one component enabled on the compute node.

3. List images in the Image Service catalog to verify connectivity with the Identity service and Image Service:

```
$ nova image-list
```

| ID                                   | Name                | Status |
|--------------------------------------|---------------------|--------|
| Server                               |                     |        |
| acafc7c0-40aa-4026-9673-b879898e1fc2 | cirros-0.3.3-x86_64 | ACTIVE |

```
+-----+-----+-----+
+-----+
```

## 6. Add a networking component

### Table of Contents

|                                        |    |
|----------------------------------------|----|
| OpenStack Networking (neutron) .....   | 60 |
| Legacy networking (nova-network) ..... | 83 |
| Next steps .....                       | 85 |

This chapter explains how to install and configure either OpenStack Networking (neutron), or the legacy `nova-network` component. The `nova-network` service enables you to deploy one network type per instance and is suitable for basic network functionality. OpenStack Networking enables you to deploy multiple network types per instance and includes *plug-ins* for a variety of products that support *virtual networking*.

For more information, see the [Networking](#) chapter of the *OpenStack Cloud Administrator Guide*.

## OpenStack Networking (neutron)

### OpenStack Networking

OpenStack Networking allows you to create and attach interface devices managed by other OpenStack services to networks. Plug-ins can be implemented to accommodate different networking equipment and software, providing flexibility to OpenStack architecture and deployment.

It includes the following components:

|                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>neutron-server</b>                           | Accepts and routes API requests to the appropriate OpenStack Networking plug-in for action.                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>OpenStack Networking plug-ins and agents</b> | Plugs and unplugs ports, creates networks or subnets, and provides IP addressing. These plug-ins and agents differ depending on the vendor and technologies used in the particular cloud. OpenStack Networking ships with plug-ins and agents for Cisco virtual and physical switches, NEC OpenFlow products, Open vSwitch, Linux bridging, Ryu Network Operating System, and the VMware NSX product.<br><br>The common agents are L3 (layer 3), DHCP (dynamic host IP addressing), and a plug-in agent. |
| <b>Messaging queue</b>                          | Used by most OpenStack Networking installations to route information between the neutron-server and various agents, as well as a database to store networking state for particular plug-ins.                                                                                                                                                                                                                                                                                                             |

OpenStack Networking mainly interacts with OpenStack Compute to provide networks and connectivity for its instances.

## Networking concepts

OpenStack Networking (neutron) manages all 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 enables tenants to create advanced virtual network topologies including services such as *firewalls*, *load balancers*, and *virtual private networks (VPNs)*.

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

Each router has one gateway that connects to a network, and many interfaces connected to subnets. Subnets can access machines on other subnets connected to the same router.

Any given Networking set up has at least one external network. Unlike the other networks, the external network is not merely a virtually defined network. Instead, it represents a view into a slice of the physical, external network accessible outside the OpenStack installation. IP addresses on the external network are accessible by anybody physically on the outside network. Because the external network merely represents a view into 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 the VNI and OpenStack environment, 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-Balancer-as-a-Service (LBaaS) plug-ins are available.

## Install and configure controller node

### To configure prerequisites

Before you configure the OpenStack Networking (neutron) service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:

- a. Use the database access client to connect to the database server as the `root` user:

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

- b. Create the `neutron` database:

```
CREATE DATABASE neutron;
```

- c. Grant proper access to the `neutron` database:

```
GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'localhost' \
IDENTIFIED BY 'NEUTRON_DBPASS';
GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'%' \
IDENTIFIED BY 'NEUTRON_DBPASS';
```

Replace `NEUTRON_DBPASS` with a suitable password.

- d. Exit the database access client.

2. Source the `admin` credentials to gain access to admin-only CLI commands:

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

3. To create the service credentials, complete these steps:

- a. Create the `neutron` user:

```
$ keystone user-create --name neutron --pass NEUTRON_PASS
```

| Property | Value                            |
|----------|----------------------------------|
| email    |                                  |
| enabled  | True                             |
| id       | 7fd67878dcd04d0393469ef825a7e005 |
| name     | neutron                          |
| username | neutron                          |

Replace `NEUTRON_PASS` with a suitable password.

- b. Add the `admin` role to the `neutron` user:

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



### Note

This command provides no output.

- c. Create the `neutron` service entity:

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

| Property    | Value                            |
|-------------|----------------------------------|
| description | OpenStack Networking             |
| enabled     | True                             |
| id          | 6369ddaf99a447f3a0d41dac5e342161 |
| name        | neutron                          |
| type        | network                          |

#### 4. Create the Networking service API endpoints:

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

| Property    | Value                            |
|-------------|----------------------------------|
| adminurl    | http://controller:9696           |
| id          | fa18b41938a94bf6b35e2c152063ee21 |
| internalurl | http://controller:9696           |
| publicurl   | http://controller:9696           |
| region      | regionOne                        |
| service_id  | 6369ddaf99a447f3a0d41dac5e342161 |

### To install the Networking components

- `# apt-get install neutron-server neutron-plugin-ml2 python-neutronclient`

### To configure the Networking server component

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

- Edit the `/etc/neutron/neutron.conf` file and complete the following actions:
  - In the `[database]` section, configure database access:

```
[database]
...
connection = mysql://neutron:NEUTRON_DBPASS@controller/neutron
```

Replace `NEUTRON_DBPASS` with the password you chose for the database.

- In the `[DEFAULT]` section, configure RabbitMQ message broker access:

```
[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

Replace *RABBIT\_PASS* with the password you chose for the guest account in RabbitMQ.

- c. In the [DEFAULT] and [keystone\_authtoken] sections, configure Identity service access:

```
[DEFAULT]
...
auth_strategy = keystone

[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
```

Replace *NEUTRON\_PASS* with the password you chose or the neutron user in the Identity service.



### Note

Comment out any *auth\_host*, *auth\_port*, and *auth\_protocol* options because the *identity\_uri* option replaces them.

- d. In the [DEFAULT] section, enable the Modular Layer 2 (ML2) plug-in, router service, and overlapping IP addresses:

```
[DEFAULT]
...
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True
```

- e. In the [DEFAULT] section, configure Networking to notify Compute of network topology changes:

```
[DEFAULT]
...
notify_nova_on_port_status_changes = True
notify_nova_on_port_data_changes = True
nova_url = http://controller:8774/v2
nova_admin_auth_url = http://controller:35357/v2.0
nova_region_name = regionOne
nova_admin_username = nova
nova_admin_tenant_id = SERVICE_TENANT_ID
nova_admin_password = NOVA_PASS
```

Replace *SERVICE\_TENANT\_ID* with the service tenant identifier (id) in the Identity service and *NOVA\_PASS* with the password you chose for the nova user in the Identity service.



### Note

To obtain the service tenant identifier (id):



```
$ source admin-openrc.sh
$ keystone tenant-get service
```

| Property    | Value                            |
|-------------|----------------------------------|
| description | Service Tenant                   |
| enabled     | True                             |
| id          | f727b5ec2ceb4d71bad86dfc414449bf |
| name        | service                          |

- f. (Optional) To assist with troubleshooting, enable verbose logging in the [DEFAULT] section:

```
[DEFAULT]
...
verbose = True
```

### 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 components because it does not handle instance network traffic.

- Edit the `/etc/neutron/plugins/ml2/ml2_conf.ini` file and complete the following actions:
  - In the [ml2] section, enable the *flat* and *generic routing encapsulation (GRE)* network type drivers, GRE tenant networks, and the OVS mechanism driver:

```
[ml2]
...
type_drivers = flat,gre
tenant_network_types = gre
mechanism_drivers = openvswitch
```



### Warning

Once you configure the ML2 plug-in, be aware that disabling a network type driver and re-enabling it later can lead to database inconsistency.

- In the [ml2\_type\_gre] section, configure the tunnel identifier (id) range:

```
[ml2_type_gre]
...
tunnel_id_ranges = 1:1000
```

- In the [securitygroup] section, enable security groups, enable *ipset*, and configure the OVS *iptables* firewall driver:

```
[securitygroup]
...
enable_security_group = True
enable_ipset = True
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
```

## To configure Compute to use Networking

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

- Edit the `/etc/nova/nova.conf` file on the controller node and complete the following actions:
  - a. In the `[DEFAULT]` section, configure the *APIs* and drivers:

```
[DEFAULT]
...
network_api_class = nova.network.neutronv2.api.API
security_group_api = neutron
linuxnet_interface_driver = nova.network.linux_net.
LinuxOVSIInterfaceDriver
firewall_driver = nova.virt.firewall.NoopFirewallDriver
```



### 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.

- b. In the `[neutron]` section, configure access parameters:

```
[neutron]
...
url = http://controller:9696
auth_strategy = keystone
admin_auth_url = http://controller:35357/v2.0
admin_tenant_name = service
admin_username = neutron
admin_password = NEUTRON_PASS
```

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

## To finalize installation

1. Populate the database:

```
su -s /bin/sh -c "neutron-db-manage --config-file /etc/neutron/neutron.
conf \
--config-file /etc/neutron/plugins/ml2/ml2_conf.ini upgrade juno"
neutron
```



### Note

Database population occurs later for Networking because the script requires complete server and plug-in configuration files.

2. Restart the Compute services:

```
service nova-api restart
service nova-scheduler restart
```

```
service nova-conductor restart
```

- Restart the Networking service:

```
service neutron-server restart
```

## Verify operation



### Note

Perform these commands on the controller node.

- Source the admin credentials to gain access to admin-only CLI commands:

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

- List loaded extensions to verify successful launch of the neutron-server process:

```
$ neutron ext-list
```

| alias                 | name                                          |
|-----------------------|-----------------------------------------------|
| security-group        | security-group                                |
| l3_agent_scheduler    | L3 Agent Scheduler                            |
| ext-gw-mode           | Neutron L3 Configurable external gateway mode |
| binding               | Port Binding                                  |
| provider              | Provider Network                              |
| agent                 | agent                                         |
| quotas                | Quota management support                      |
| dhcp_agent_scheduler  | DHCP Agent Scheduler                          |
| l3-ha                 | HA Router extension                           |
| multi-provider        | Multi Provider Network                        |
| external-net          | Neutron external network                      |
| router                | Neutron L3 Router                             |
| allowed-address-pairs | Allowed Address Pairs                         |
| extraroute            | Neutron Extra Route                           |
| extra_dhcp_opt        | Neutron Extra DHCP opts                       |
| dvr                   | Distributed Virtual Router                    |

## Install and configure network node

The network node primarily handles internal and external routing and *DHCP* services for virtual networks.

### To configure prerequisites

Before you install and configure OpenStack Networking, you must configure certain kernel networking parameters.

- Edit the `/etc/sysctl.conf` file to contain the following parameters:

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

- Implement the changes:

```
sysctl -p
```

## To install the Networking components

- ```
# apt-get install neutron-plugin-ml2 neutron-plugin-openvswitch-agent \
neutron-l3-agent neutron-dhcp-agent
```

To configure the Networking common components

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

- Edit the `/etc/neutron/neutron.conf` file and complete the following actions:
 - a. In the `[database]` section, comment out any `connection` options because network nodes do not directly access the database.
 - b. In the `[DEFAULT]` section, configure RabbitMQ message broker access:

```
[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

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

- c. In the `[DEFAULT]` and `[keystone_authtoken]` sections, configure Identity service access:

```
[DEFAULT]
...
auth_strategy = keystone

[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
```

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



Note

Comment out any `auth_host`, `auth_port`, and `auth_protocol` options because the `identity_uri` option replaces them.

- d. In the `[DEFAULT]` section, enable the Modular Layer 2 (ML2) plug-in, router service, and overlapping IP addresses:

```
[DEFAULT]
...
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True
```

- e. (Optional) To assist with troubleshooting, enable verbose logging in the [DE-FAULT] section:

```
[DEFAULT]
...
verbose = True
```

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.

- Edit the `/etc/neutron/plugins/ml2/ml2_conf.ini` file and complete the following actions:

- a. In the [ml2] section, enable the *flat* and *generic routing encapsulation (GRE)* network type drivers, GRE tenant networks, and the OVS mechanism driver:

```
[ml2]
...
type_drivers = flat,gre
tenant_network_types = gre
mechanism_drivers = openvswitch
```

- b. In the [ml2_type_flat] section, configure the external flat provider network:

```
[ml2_type_flat]
...
flat_networks = external
```

- c. In the [ml2_type_gre] section, configure the tunnel identifier (id) range:

```
[ml2_type_gre]
...
tunnel_id_ranges = 1:1000
```

- d. In the [securitygroup] section, enable security groups, enable *ipset*, and configure the OVS *iptables* firewall driver:

```
[securitygroup]
...
enable_security_group = True
enable_ipset = True
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
```

- e. In the [ovs] section, enable tunnels, configure the local tunnel endpoint, and map the external flat provider network to the `br-ex` external network bridge:

```
[ovs]
...
local_ip = INSTANCE_TUNNELS_INTERFACE_IP_ADDRESS
enable_tunneling = True
bridge_mappings = external:br-ex
```

Replace `INSTANCE_TUNNELS_INTERFACE_IP_ADDRESS` with the IP address of the instance tunnels network interface on your network node.

- f. In the `[agent]` section, enable GRE tunnels:

```
[agent]
...
tunnel_types = gre
```

To configure the Layer-3 (L3) agent

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

- Edit the `/etc/neutron/l3_agent.ini` file and complete the following actions:
 - a. In the `[DEFAULT]` section, configure the driver, enable *network namespaces*, configure the external network bridge and enable deletion of defunct router namespaces:

```
[DEFAULT]
...
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
use_namespaces = True
external_network_bridge = br-ex
router_delete_namespaces = True
```

- b. (Optional) To assist with troubleshooting, enable verbose logging in the `[DEFAULT]` section:

```
[DEFAULT]
...
verbose = True
```

To configure the DHCP agent

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

1. Edit the `/etc/neutron/dhcp_agent.ini` file and complete the following actions:
 - a. In the `[DEFAULT]` section, configure the drivers, enable namespaces and enable deletion of defunct DHCP namespaces:

```
[DEFAULT]
...
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
use_namespaces = True
dhcp_delete_namespaces = True
```

- b. (Optional) To assist with troubleshooting, enable verbose logging in the `[DEFAULT]` section:

```
[DEFAULT]
...
verbose = True
```

2. (Optional)

Tunneling protocols such as GRE include additional packet headers that increase overhead and decrease space available for the payload or user data. Without knowledge of

the virtual network infrastructure, instances attempt to send packets using the default Ethernet *maximum transmission unit (MTU)* of 1500 bytes. *Internet protocol (IP)* networks contain the *path MTU discovery (PMTUD)* mechanism to detect end-to-end MTU and adjust packet size accordingly. However, some operating systems and networks block or otherwise lack support for PMTUD causing performance degradation or connectivity failure.

Ideally, you can prevent these problems by enabling *jumbo frames* on the physical network that contains your tenant virtual networks. Jumbo frames support MTUs up to approximately 9000 bytes which negates the impact of GRE overhead on virtual networks. However, many network devices lack support for jumbo frames and OpenStack administrators often lack control over network infrastructure. Given the latter complications, you can also prevent MTU problems by reducing the instance MTU to account for GRE overhead. Determining the proper MTU value often takes experimentation, but 1454 bytes works in most environments. You can configure the DHCP server that assigns IP addresses to your instances to also adjust the MTU.



Note

Some cloud images ignore the DHCP MTU option in which case you should configure it using metadata, script, or other suitable method.

- a. Edit the `/etc/neutron/dhcp_agent.ini` file and complete the following action:

- In the `[DEFAULT]` section, enable the `dnsmasq` configuration file:

```
[DEFAULT]
...
dnsmasq_config_file = /etc/neutron/dnsmasq-neutron.conf
```

- b. Create and edit the `/etc/neutron/dnsmasq-neutron.conf` file and complete the following action:

- Enable the DHCP MTU option (26) and configure it to 1454 bytes:

```
dhcp-option-force=26,1454
```

- c. Kill any existing `dnsmasq` processes:

```
# pkill dnsmasq
```

To configure the metadata agent

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

1. Edit the `/etc/neutron/metadata_agent.ini` file and complete the following actions:
 - a. In the `[DEFAULT]` section, configure access parameters:

```
[DEFAULT]
...
auth_url = http://controller:5000/v2.0
auth_region = regionOne
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
```

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

- b. In the `[DEFAULT]` section, configure the metadata host:

```
[DEFAULT]
...
nova_metadata_ip = controller
```

- c. In the `[DEFAULT]` section, configure the metadata proxy shared secret:

```
[DEFAULT]
...
metadata_proxy_shared_secret = METADATA_SECRET
```

Replace *METADATA_SECRET* with a suitable secret for the metadata proxy.

- d. (Optional) To assist with troubleshooting, enable verbose logging in the `[DEFAULT]` section:

```
[DEFAULT]
...
verbose = True
```

2. On the *controller* node, edit the `/etc/nova/nova.conf` file and complete the following action:

- In the `[neutron]` section, enable the metadata proxy and configure the secret:

```
[neutron]
...
service_metadata_proxy = True
metadata_proxy_shared_secret = METADATA_SECRET
```

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

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

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

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-ex` 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 connects the virtual and physical external networks in your environment.

1. Restart the OVS service:

```
# service openvswitch-switch restart
```

2. Add the external bridge:

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

3. 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

- Restart the Networking services:

```
# service neutron-plugin-openvswitch-agent restart
# service neutron-l3-agent restart
# service neutron-dhcp-agent restart
# service neutron-metadata-agent restart
```

Verify operation



Note

Perform these commands on the controller node.

1. Source the `admin` credentials to gain access to admin-only CLI commands:

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

2. List agents to verify successful launch of the neutron agents:

```
$ neutron agent-list
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| id                  | agent_type          | host      |
+-----+-----+-----+-----+
| 30275801-e17a-41e4-8f53-9db63544f689 | Metadata agent      | network
| :- )               | neutron-metadata-agent |
+-----+-----+-----+-----+
```

```

| 4bd8c50e-7bad-4f3b-955d-67658a491a15 | Open vSwitch agent | network
| :- ) | True | neutron-openvswitch-agent |
| 756e5bba-b70f-4715-b80e-e37f59803d20 | L3 agent | network
| :- ) | True | neutron-l3-agent |
| 9c45473c-6d6d-4f94-8df1-ebd0b6838d5f | DHCP agent | network
| :- ) | True | neutron-dhcp-agent |
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+

```

Install and configure compute node

The compute node handles connectivity and *security groups* for instances.

To configure prerequisites

Before you install and configure OpenStack Networking, you must configure certain kernel networking parameters.

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

```
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

- `# apt-get install neutron-plugin-m12 neutron-plugin-openvswitch-agent`

To configure the Networking common components

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

- Edit the `/etc/neutron/neutron.conf` file and complete the following actions:
 - a. In the `[database]` section, comment out any `connection` options because compute nodes do not directly access the database.
 - b. In the `[DEFAULT]` section, configure RabbitMQ message broker access:

```
[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

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

- c. In the `[DEFAULT]` and `[keystone_authtoken]` sections, configure Identity service access:

```
[DEFAULT]
...
auth_strategy = keystone

[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
```

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



Note

Comment out any *auth_host*, *auth_port*, and *auth_protocol* options because the *identity_uri* option replaces them.

- d. In the `[DEFAULT]` section, enable the Modular Layer 2 (ML2) plug-in, router service, and overlapping IP addresses:

```
[DEFAULT]
...
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True
```

- e. (Optional) To assist with troubleshooting, enable verbose logging in the `[DEFAULT]` section:

```
[DEFAULT]
...
verbose = True
```

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.

- Edit the `/etc/neutron/plugins/ml2/ml2_conf.ini` file and complete the following actions:
 - a. In the `[ml2]` section, enable the *flat* and *generic routing encapsulation (GRE)* network type drivers, GRE tenant networks, and the OVS mechanism driver:

```
[ml2]
...
type_drivers = flat,gre
tenant_network_types = gre
mechanism_drivers = openvswitch
```

- b. In the `[ml2_type_gre]` section, configure the tunnel identifier (id) range:

```
[ml2_type_gre]
...
tunnel_id_ranges = 1:1000
```

- c. In the `[securitygroup]` section, enable security groups, enable *ipset*, and configure the OVS *iptables* firewall driver:

```
[securitygroup]
...
enable_security_group = True
enable_ipset = True
firewall_driver = neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver
```

- d. In the `[ovs]` section, enable tunnels and configure the local tunnel endpoint:

```
[ovs]
...
local_ip = INSTANCE_TUNNELS_INTERFACE_IP_ADDRESS
enable_tunneling = True
```

Replace `INSTANCE_TUNNELS_INTERFACE_IP_ADDRESS` with the IP address of the instance tunnels network interface on your compute node.

- e. In the `[agent]` section, enable GRE tunnels:

```
[agent]
...
tunnel_types = gre
```

To configure the Open vSwitch (OVS) service

The OVS service provides the underlying virtual networking framework for instances.

- Restart the OVS service:

```
# service openvswitch-switch restart
```

To configure Compute to use Networking

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

- Edit the `/etc/nova/nova.conf` file and complete the following actions:

- a. In the `[DEFAULT]` section, configure the *APIs* and drivers:

```
[DEFAULT]
...
network_api_class = nova.network.neutronv2.api.API
security_group_api = neutron
linuxnet_interface_driver = nova.network.linux_net.
LinuxOVSIInterfaceDriver
firewall_driver = nova.virt.firewall.NoopFirewallDriver
```

**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.

- b. In the `[neutron]` section, configure access parameters:

```
[neutron]
...
url = http://controller:9696
auth_strategy = keystone
admin_auth_url = http://controller:35357/v2.0
admin_tenant_name = service
admin_username = neutron
admin_password = NEUTRON_PASS
```

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

To finalize the installation

1. Restart the Compute service:

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

2. Restart the Open vSwitch (OVN) agent:

```
# service neutron-plugin-openvswitch-agent restart
```

Verify operation**Note**

Perform these commands on the controller node.

1. Source the admin credentials to gain access to admin-only CLI commands:

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

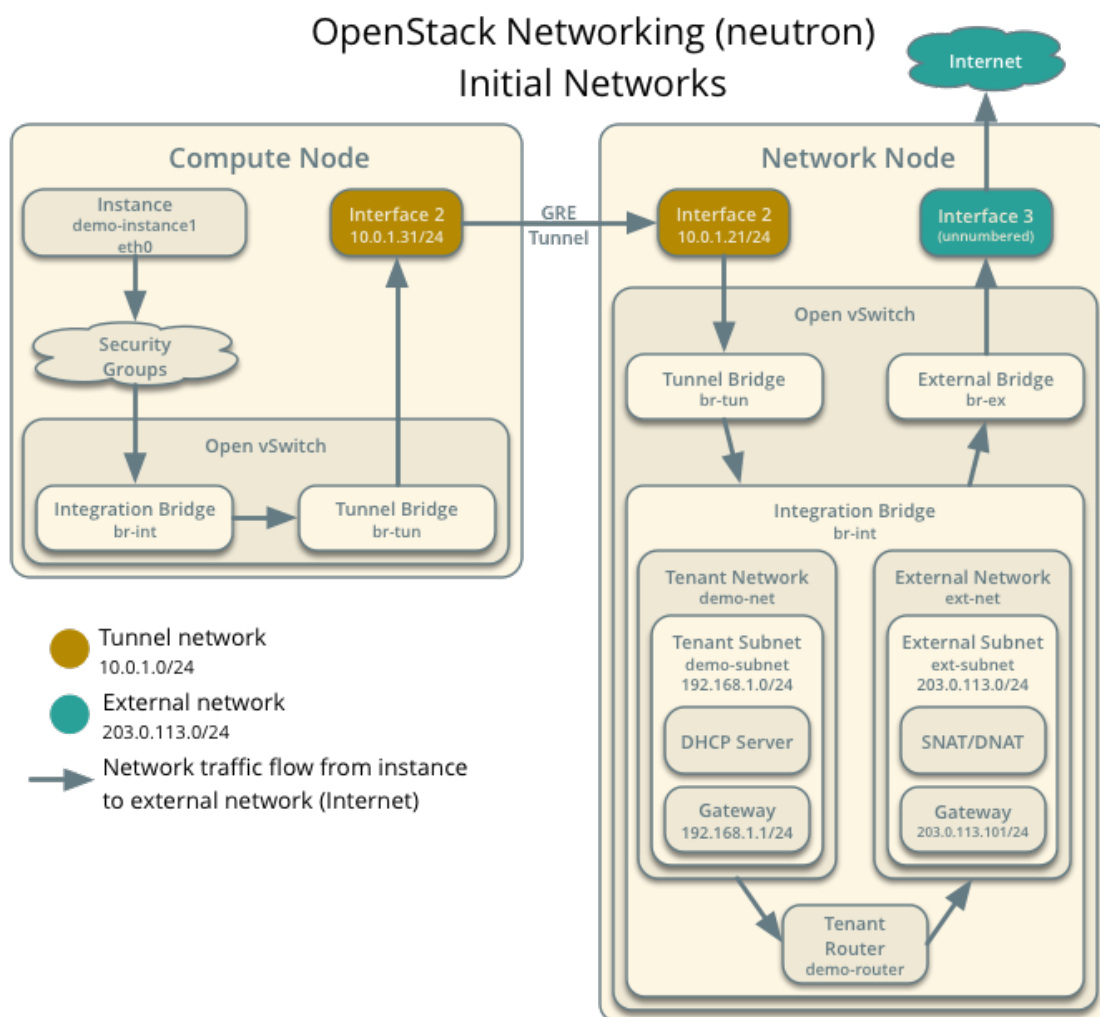
2. List agents to verify successful launch of the neutron agents:

```
$ neutron agent-list
+-----+-----+-----+-----+
| id          | admin_state_up | binary          | agent_type          | host          |
+-----+-----+-----+-----+
| a5a49051-05eb-4b4f-bfc7-d36235fe9131 | True          | neutron-openvswitch-agent | Open vSwitch agent | compute1     |
+-----+-----+-----+-----+
| :- )       |               |                   |                     |              |
+-----+-----+-----+-----+
...
+-----+-----+-----+-----+
| a5a49051-05eb-4b4f-bfc7-d36235fe9131 | True          | neutron-openvswitch-agent | Open vSwitch agent | compute1     |
+-----+-----+-----+-----+
| :- )       |               |                   |                     |              |
+-----+-----+-----+-----+
```

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 6.1, “Initial networks” \[78\]](#). After creating this infrastructure, we recommend that you [verify connectivity](#) and resolve any issues before proceeding further. [Figure 6.1, “Initial networks” \[78\]](#) provides a basic architectural overview of the components that Networking implements for the initial networks and shows how network traffic flows from the instance to the external network or Internet.

Figure 6.1. Initial networks



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.



Note

Perform these commands on the controller node.

To create the external network

1. Source the *admin* credentials to gain access to admin-only CLI commands:

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

2. Create the network:

```
$ neutron net-create ext-net --router:external True \
  --provider:physical_network external --provider:network_type flat
Created a new network:
```

Field	Value
admin_state_up	True
id	893aebb9-1c1e-48be-8908-6b947f3237b3
name	ext-net
provider:network_type	flat
provider:physical_network	external
provider:segmentation_id	
router:external	True
shared	False
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.

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
```

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.

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` credentials to gain access to user-only CLI commands:

```
$ 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
router:external	False
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. 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
```

Replace *TENANT_NETWORK_CIDR* with the subnet you want to associate with the tenant network and *TENANT_NETWORK_GATEWAY* with the gateway you want to associate with it, typically the ".1" IP address.

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
routes	
status	ACTIVE
tenant_id	cdef0071a0194d19ac6bb63802dc9bae

2. Attach the router to the demo tenant subnet:

```
$ neutron router-interface-add demo-router demo-subnet
Added interface bla894fd-ae8-475c-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 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 legacy networking.

To configure legacy networking

1. Edit the `/etc/nova/nova.conf` file and complete the following actions:
 - In the `[DEFAULT]` section, configure the network and security group APIs:

```
[DEFAULT]
...
network_api_class = nova.network.api.API
security_group_api = nova
```

2. Restart the Compute services:

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

- `# apt-get install nova-network nova-api-metadata`

To configure legacy networking

1. Edit the `/etc/nova/nova.conf` file and complete the following actions:

- In the `[DEFAULT]` section, configure the network parameters:

```
[DEFAULT]
...
network_api_class = nova.network.api.API
security_group_api = nova
firewall_driver = nova.virt.libvirt.firewall.IptablesFirewallDriver
network_manager = nova.network.manager.FlatDHCPManager
network_size = 254
allow_same_net_traffic = False
multi_host = True
send_arp_for_ha = True
share_dhcp_address = True
force_dhcp_release = True
flat_network_bridge = br100
flat_interface = INTERFACE_NAME
public_interface = INTERFACE_NAME
```

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

2. Restart the services:

```
# service nova-network restart
# service nova-api-metadata restart
```

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 OpenStack services to your environment.

7. Add the dashboard

Table of Contents

System requirements	86
Install and configure	87
Verify operation	88
Next steps	88

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.

Horizon enables you to customize the brand of the dashboard.

Horizon provides a set of core classes and reusable templates and tools.

This example deployment uses an Apache web server.

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 and configure

This section describes how to install and configure the dashboard on the controller node.

Before you proceed, verify that your system meets the requirements in [the section called "System requirements" \[86\]](#). Also, the dashboard relies on functional core services including Identity, Image Service, Compute, and either Networking (neutron) or legacy networking (nova-network). Environments with stand-alone services such as Object Storage cannot use the dashboard. For more information, see the [developer documentation](#).

To install the dashboard components

- Install the packages:

```
# apt-get install openstack-dashboard apache2 libapache2-mod-wsgi  
memcached python-memcache
```



Note

Ubuntu installs the `openstack-dashboard-ubuntu-theme` package as a dependency. Some users reported issues with this theme in previous releases. If you encounter issues, remove this package to restore the original OpenStack theme.

To configure the dashboard

- Edit the `/etc/openstack-dashboard/local_settings.py` file and complete the following actions:

- a. Configure the dashboard to use OpenStack services on the controller node:

```
OPENSTACK_HOST = "controller"
```

- b. Allow all hosts to access the dashboard:

```
ALLOWED_HOSTS = ['*']
```

- c. Configure the memcached session storage service:

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



Note

Comment out any other session storage configuration.

- d. Optionally, configure the time zone:

```
TIME_ZONE = "TIME_ZONE"
```

Replace *TIME_ZONE* with an appropriate time zone identifier. For more information, see the [list of time zones](#).

To finalize installation

- Restart the web server and session storage service:

```
# service apache2 restart  
# service memcached restart
```

Verify operation

This section describes how to verify operation of the dashboard.

1. Access the dashboard using a web browser: `http://controller/horizon`.
2. Authenticate using `admin` or `demo` user credentials.

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.

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* for information on setting up colors, logos, and site titles.
- Set up session storage. See section [Set up session storage for the dashboard](#) in the *OpenStack Cloud Administrator Guide* for information on user session data.

8. Add the Block Storage service

Table of Contents

OpenStack Block Storage	89
Install and configure controller node	90
Install and configure a storage node	93
Verify operation	97
Next steps	98

The OpenStack Block Storage service provides block storage devices to instances using various backends. The Block Storage API and scheduler services run on the controller node and the volume service runs on one or more storage nodes. Storage nodes provide volumes to instances using local block storage devices or SAN/NAS backends with the appropriate drivers. For more information, see the [Configuration Reference](#).



Note

This chapter omits the backup manager because it depends on the Object Storage service.

OpenStack Block Storage

The OpenStack Block Storage service (cinder) adds persistent storage to a virtual machine. Block Storage provides an infrastructure for managing volumes, and interacts with OpenStack Compute to provide volumes for instances. The service also enables management of volume snapshots, and volume types.

The Block Storage service consists of the following components:

cinder-api	Accepts API requests, and routes them to the <code>cinder-volume</code> for action.
cinder-volume	Interacts directly with the Block Storage service, and processes such as the <code>cinder-scheduler</code> . It also interacts with these processes through a message queue. The <code>cinder-volume</code> service responds to read and write requests sent to the Block Storage service to maintain state. It can interact with a variety of storage providers through a driver architecture.
cinder-scheduler daemon	Selects the optimal storage provider node on which to create the volume. A similar component to the <code>no-va-scheduler</code> .
Messaging queue	Routes information between the Block Storage processes.

Install and configure controller node

This section describes how to install and configure the Block Storage service, code-named `cinder`, on the controller node. This service requires at least one additional storage node that provides volumes to instances.

To configure prerequisites

Before you install and configure the Block Storage service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:

- a. Use the database access client to connect to the database server as the `root` user:

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

- b. Create the `cinder` database:

```
CREATE DATABASE cinder;
```

- c. Grant proper access to the `cinder` database:

```
GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'localhost' \
  IDENTIFIED BY 'CINDER_DBPASS';
GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'%' \
  IDENTIFIED BY 'CINDER_DBPASS';
```

Replace `CINDER_DBPASS` with a suitable password.

- d. Exit the database access client.

2. Source the `admin` credentials to gain access to admin-only CLI commands:

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

3. To create the service credentials, complete these steps:

- a. Create a `cinder` user:

```
$ keystone user-create --name cinder --pass CINDER_PASS
+-----+-----+
| Property | Value |
+-----+-----+
| email    |       |
| enabled  | True  |
| id       | 881ab2de4f7941e79504a759a83308be |
| name     | cinder |
| username | cinder |
+-----+-----+
```

Replace `CINDER_PASS` with a suitable password.

- b. Add the `admin` role to the `cinder` user:

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

**Note**

This command provides no output.

- c. Create the cinder service entities:

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

Property	Value
description	OpenStack Block Storage
enabled	True
id	1e494c3e22a24baafcaf777d4d467eb
name	cinder
type	volume

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

Property	Value
description	OpenStack Block Storage
enabled	True
id	16e038e449c94b40868277f1d801edb5
name	cinderv2
type	volumev2

**Note**

The Block Storage service requires two different service entities to support API versions 1 and 2.

4. Create the Block Storage service API endpoints:

```
$ 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 \
--region regionOne
```

Property	Value
adminurl	http://controller:8776/v1/%(tenant_id)s
id	d1b7291a2d794e26963b322c7f2a55a4
internalurl	http://controller:8776/v1/%(tenant_id)s
publicurl	http://controller:8776/v1/%(tenant_id)s
region	regionOne
service_id	1e494c3e22a24baafcaf777d4d467eb

```
$ 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 \
--region regionOne
```

Property	Value
adminurl	http://controller:8776/v2/(tenant_id)s
id	097b4a6fc8ba44b4b10d4822d2d9e076
internalurl	http://controller:8776/v2/(tenant_id)s
publicurl	http://controller:8776/v2/(tenant_id)s
region	regionOne
service_id	16e038e449c94b40868277f1d801edb5



Note

The Block Storage service requires two different endpoints to support API versions 1 and 2.

To install and configure Block Storage controller components

1. Install the packages:

```
# apt-get install cinder-api cinder-scheduler python-cinderclient
```

2. Edit the `/etc/cinder/cinder.conf` file and complete the following actions:

- a. In the `[database]` section, configure database access:

```
[database]
...
connection = mysql://cinder:CINDER_DBPASS@controller/cinder
```

Replace `CINDER_DBPASS` with the password you chose for the Block Storage database.

- b. In the `[DEFAULT]` section, configure RabbitMQ message broker access:

```
[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

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

- c. In the `[DEFAULT]` and `[keystone_authtoken]` sections, configure Identity service access:

```
[DEFAULT]
...
auth_strategy = keystone

[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = cinder
admin_password = CINDER_PASS
```

Replace `CINDER_PASS` with the password you chose for the `cinder` user in the Identity service.



Note

Comment out any `auth_host`, `auth_port`, and `auth_protocol` options because the `identity_uri` option replaces them.

- d. In the `[DEFAULT]` section, configure the `my_ip` option to use the management interface IP address of the controller node:

```
[DEFAULT]
...
my_ip = 10.0.0.11
```

- e. (Optional) To assist with troubleshooting, enable verbose logging in the `[DEFAULT]` section:

```
[DEFAULT]
...
verbose = True
```

3. Populate the Block Storage database:

```
# su -s /bin/sh -c "cinder-manage db sync" cinder
```

To finalize installation

1. Restart the Block Storage services:

```
# service cinder-scheduler restart
# service cinder-api restart
```

2. By default, the Ubuntu packages create an SQLite database.

Because this configuration uses a SQL database server, you can remove the SQLite database file:

```
# rm -f /var/lib/cinder/cinder.sqlite
```

Install and configure a storage node

This section describes how to install and configure storage nodes for the Block Storage service. For simplicity, this configuration references one storage node with an empty local block storage device `/dev/sdb` that contains a suitable partition table with one partition `/dev/sdb1` occupying the entire device. The service provisions logical volumes on this device using the *LVM* driver and provides them to instances via *iSCSI* transport. You can follow these instructions with minor modifications to horizontally scale your environment with additional storage nodes.

To configure prerequisites

You must configure the storage node before you install and configure the volume service on it. Similar to the controller node, the storage node contains one network interface on the *management network*. The storage node also needs an empty block storage device of

suitable size for your environment. For more information, see [Chapter 2, “Basic environment” \[11\]](#).

1. Configure the management interface:

IP address: 10.0.0.41

Network mask: 255.255.255.0 (or /24)

Default gateway: 10.0.0.1

2. Set the hostname of the node to *block1*.
3. Copy the contents of the `/etc/hosts` file from the controller node to the storage node and add the following to it:

```
# block1
10.0.0.41      block1
```

Also add this content to the `/etc/hosts` file on all other nodes in your environment.

4. Install and configure *NTP* using the instructions in [the section called “Other nodes” \[25\]](#).
5. Install the LVM packages:

```
# apt-get install lvm2
```



Note

Some distributions include LVM by default.

6. Create the LVM physical volume `/dev/sdb1`:

```
# pvcreate /dev/sdb1
Physical volume "/dev/sdb1" successfully created
```



Note

If your system uses a different device name, adjust these steps accordingly.

7. Create the LVM volume group `cinder-volumes`:

```
# vgcreate cinder-volumes /dev/sdb1
Volume group "cinder-volumes" successfully created
```

The Block Storage service creates logical volumes in this volume group.

8. Only instances can access Block Storage volumes. However, the underlying operating system manages the devices associated with the volumes. By default, the LVM volume scanning tool scans the `/dev` directory for block storage devices that contain volumes. If tenants use LVM on their volumes, the scanning tool detects these volumes and attempts to cache them which can cause a variety of problems with both the underlying operating system and tenant volumes. You must reconfigure LVM to scan only the devices that contain the `cinder-volume` volume group. Edit the `/etc/lvm/lvm.conf` file and complete the following actions:

- In the `devices` section, add a filter that accepts the `/dev/sdb` device and rejects all other devices:

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

Each item in the filter array begins with `a` for *accept* or `r` for *reject* and includes a regular expression for the device name. The array must end with `r/.*/` to reject any remaining devices. You can use the `vgs -vvvv` command to test filters.



Warning

If your storage nodes use LVM on the operating system disk, you must also add the associated device to the filter. For example, if the `/dev/sda` device contains the operating system:

```
filter = [ "a/sda/", "a/sdb/", "r/.*/"]
```

Similarly, if your compute nodes use LVM on the operating system disk, you must also modify the filter in the `/etc/lvm/lvm.conf` file on those nodes to include only the operating system disk. For example, if the `/dev/sda` device contains the operating system:

```
filter = [ "a/sda/", "r/.*/"]
```

Install and configure Block Storage volume components

1. Install the packages:

```
# apt-get install cinder-volume python-mysqldb
```

2. Edit the `/etc/cinder/cinder.conf` file and complete the following actions:

- a. In the `[database]` section, configure database access:

```
[database]  
...  
connection = mysql://cinder:CINDER_DBPASS@controller/cinder
```

Replace `CINDER_DBPASS` with the password you chose for the Block Storage database.

- b. In the `[DEFAULT]` section, configure RabbitMQ message broker access:

```
[DEFAULT]  
...  
rpc_backend = rabbit  
rabbit_host = controller  
rabbit_password = RABBIT_PASS
```

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

- c. In the `[DEFAULT]` and `[keystone_authtoken]` sections, configure Identity service access:

```
[DEFAULT]
...
auth_strategy = keystone

[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = cinder
admin_password = CINDER_PASS
```

Replace *CINDER_PASS* with the password you chose for the `cinder` user in the Identity service.



Note

Comment out any `auth_host`, `auth_port`, and `auth_protocol` options because the `identity_uri` option replaces them.

- d. In the `[DEFAULT]` section, configure the `my_ip` option:

```
[DEFAULT]
...
my_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
```

Replace *MANAGEMENT_INTERFACE_IP_ADDRESS* with the IP address of the management network interface on your storage node, typically 10.0.0.41 for the first node in the [example architecture](#).

- e. In the `[DEFAULT]` section, configure the location of the Image Service:

```
[DEFAULT]
...
glance_host = controller
```

- f. (Optional) To assist with troubleshooting, enable verbose logging in the `[DEFAULT]` section:

```
[DEFAULT]
...
verbose = True
```

To finalize installation

1. Restart the Block Storage volume service including its dependencies:

```
# service tgt restart
# service cinder-volume restart
```

2. By default, the Ubuntu packages create an SQLite database. Because this configuration uses a SQL database server, remove the SQLite database file:

```
# rm -f /var/lib/cinder/cinder.sqlite
```


Verify operation

This section describes how to verify operation of the Block Storage service by creating a volume.

For more information about how to manage volumes, see the [OpenStack User Guide](#).



Note

Perform these commands on the controller node.

1. Source the admin credentials to gain access to admin-only CLI commands:

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

2. List service components to verify successful launch of each process:

```
$ cinder service-list
+-----+-----+-----+-----+-----+
+-----+
| Binary          | Host      | Zone | Status | State |
| Updated_at      | Disabled  | Reason |         |       |
+-----+-----+-----+-----+-----+
| cinder-scheduler | controller | nova | enabled | up    |
| 2014-10-18T01:30:54.000000 |      None |         |         |       |
| cinder-volume    | block1    | nova | enabled | up    |
| 2014-10-18T01:30:57.000000 |      None |         |         |       |
+-----+-----+-----+-----+-----+
```

3. Source the demo tenant credentials to perform the following steps as a non-administrative tenant:

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

4. Create a 1 GB volume:

```
$ cinder create --display-name demo-volume1 1
+-----+-----+
| Property          | Value                                     |
+-----+-----+
| attachments       | []                                       |
| availability_zone  | nova                                    |
| bootable          | false                                  |
| created_at        | 2014-10-14T23:11:50.870239             |
| display_description | None                                    |
| display_name       | demo-volume1                           |
| encrypted         | False                                  |
| id                | 158bea89-07db-4ac2-8115-66c0d6a4bb48  |
| metadata          | {}                                       |
| size              | 1                                       |
| snapshot_id       | None                                    |
| source_volid       | None                                    |
| status            | creating                                |
| volume_type       | None                                    |
+-----+-----+
```

5. Verify creation and availability of the volume:

```
$ cinder list
+-----+-----+-----+-----+-----+
|          ID          | Status | Display Name | Size |
| Volume Type | Bootable | Attached to |      |
+-----+-----+-----+-----+-----+
| 158bea89-07db-4ac2-8115-66c0d6a4bb48 | available | demo-volume1 | 1    |
|      None      |    false   |              |      |
+-----+-----+-----+-----+-----+
```

If the status does not indicate `available`, check the logs in the `/var/log/cinder` directory on the controller and volume nodes for more information.

**Note**

The [launch an instance](#) chapter includes instructions for attaching this volume to an instance.

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.

9. Add Object Storage

Table of Contents

OpenStack Object Storage	99
Install and configure the controller node	100
Install and configure the storage nodes	103
Create initial rings	107
Finalize installation	111
Verify operation	112
Next steps	113

The OpenStack Object Storage services (swift) work together to provide object storage and retrieval through a *REST* API. Your environment must at least include the Identity service (keystone) prior to deploying Object Storage.

OpenStack Object Storage

The OpenStack Object Storage is a multi-tenant object storage system. It is highly scalable and can manage large amounts of unstructured data at low cost through a RESTful HTTP API.

It includes the following components:

Proxy servers (<code>swift-proxy-server</code>)	Accepts OpenStack 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 that is usually deployed with memcache.
Account servers (<code>swift-account-server</code>)	Manages accounts defined with Object Storage.
Container servers (<code>swift-container-server</code>)	Manages the mapping of containers or folders, within Object Storage.
Object servers (<code>swift-object-server</code>)	Manages actual objects, such as files, on the storage nodes.
Various 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.
WSGI middleware	Handles authentication and is usually OpenStack Identity.

Install and configure the controller node

This section describes how to install and configure the proxy service that handles requests for the account, container, and object services operating on the storage nodes. For simplicity, this guide installs and configures the proxy service on the controller node. However, you can run the proxy service on any node with network connectivity to the storage nodes. Additionally, you can install and configure the proxy service on multiple nodes to increase performance and redundancy. For more information, see the [Deployment Guide](#).

To configure prerequisites

The proxy service relies on an authentication and authorization mechanism such as the Identity service. However, unlike other services, it also offers an internal mechanism that allows it to operate without any other OpenStack services. However, for simplicity, this guide references the Identity service in [Chapter 3, “Add the Identity service”](#) [30]. Before you configure the Object Storage service, you must create service credentials and API endpoints.



Note

The Object Storage service does not use a SQL database on the controller node.

1. To create the Identity service credentials, complete these steps:

- a. Create the `swift` user:

```
$ keystone user-create --name swift --pass SWIFT_PASS
```

Property	Value
email	
enabled	True
id	d535e5cbd2b74ac7bfb97db9cced3ed6
name	swift
username	swift

Replace *SWIFT_PASS* with a suitable password.

- b. Add the `admin` role to the `swift` user:

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



Note

This command provides no output.

- c. Create the `swift` service entity:

```
$ keystone service-create --name swift --type object-store \
--description "OpenStack Object Storage"
```

Property	Value
description	OpenStack Object Storage
enabled	True

id	75ef509da2c340499d454ae96a2c5c34
name	swift
type	object-store

2. Create the Object Storage service API endpoints:

```
$ 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 \
  --region regionOne
```

Property	Value
adminurl	http://controller:8080/
id	af534fb8b7ff40a6acf725437c586ebe
internalurl	http://controller:8080/v1/AUTH_\$(tenant_id)s
publicurl	http://controller:8080/v1/AUTH_\$(tenant_id)s
region	regionOne
service_id	75ef509da2c340499d454ae96a2c5c34

To install and configure the controller node components

1. Install the packages:



Note

Complete OpenStack environments already include some of these packages.

```
# apt-get install swift swift-proxy python-swiftclient python-
keystoneclient \
  python-keystonemiddleware memcached
```

2. Create the `/etc/swift` directory.
3. Obtain the proxy service configuration file from the Object Storage source repository:

```
# curl -o /etc/swift/proxy-server.conf \
  https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/proxy-
server.conf-sample
```

4. Edit the `/etc/swift/proxy-server.conf` file and complete the following actions:
 - a. In the `[DEFAULT]` section, configure the bind port, user, and configuration directory:

```
[DEFAULT]
...
bind_port = 8080
user = swift
swift_dir = /etc/swift
```

- b. In the `[pipeline:main]` section, enable the appropriate modules:

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

**Note**

For more information on other modules that enable additional features, see the [Deployment Guide](#).

- c. In the `[app:proxy-server]` section, enable account management:

```
[app:proxy-server]
...
allow_account_management = true
account_autocreate = true
```

- d. In the `[filter:keystoneauth]` section, configure the operator roles:

```
[filter:keystoneauth]
use = egg:swift#keystoneauth
...
operator_roles = admin,_member_
```

**Note**

You might need to uncomment this section.

- e. In the `[filter:authtoken]` section, configure Identity service access:

```
[filter:authtoken]
paste.filter_factory = keystonemiddleware.auth_token:filter_factory
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = swift
admin_password = SWIFT_PASS
delay_auth_decision = true
```

Replace *SWIFT_PASS* with the password you chose for the `swift` user in the Identity service.

**Note**

You might need to uncomment this section.

**Note**

Comment out any `auth_host`, `auth_port`, and `auth_protocol` options because the `identity_uri` option replaces them.

- f. In the `[filter:cache]` section, configure the memcached location:

```
[filter:cache]
...
memcache_servers = 127.0.0.1:11211
```

Install and configure the storage nodes

This section describes how to install and configure storage nodes that operate the account, container, and object services. For simplicity, this configuration references two storage nodes, each containing two empty local block storage devices. Each of the devices, `/dev/sdb` and `/dev/sdc`, must contain a suitable partition table with one partition occupying the entire device. Although the Object Storage service supports any file system with *extended attributes* (*xattr*), testing and benchmarking indicate the best performance and reliability on *XFS*. For more information on horizontally scaling your environment, see the [Deployment Guide](#).

To configure prerequisites

You must configure each storage node before you install and configure the Object Storage service on it. Similar to the controller node, each storage node contains one network interface on the *management network*. Optionally, each storage node can contain a second network interface on a separate network for replication. For more information, see [Chapter 2, “Basic environment”](#) [11].

1. Configure unique items on the first storage node:
 - a. Configure the management interface:

IP address: 10.0.0.51

Network mask: 255.255.255.0 (or /24)

Default gateway: 10.0.0.1
 - b. Set the hostname of the node to *object1*.
2. Configure unique items on the second storage node:
 - a. Configure the management interface:

IP address: 10.0.0.52

Network mask: 255.255.255.0 (or /24)

Default gateway: 10.0.0.1
 - b. Set the hostname of the node to *object2*.
3. Configure shared items on both storage nodes:
 - a. Copy the contents of the `/etc/hosts` file from the controller node and add the following to it:

```
# object1
10.0.0.51      object1

# object2
10.0.0.52      object2
```

Also add this content to the `/etc/hosts` file on all other nodes in your environment.

- b. Install and configure *NTP* using the instructions in [the section called “Other nodes” \[25\]](#).
- c. Install the supporting utility packages:

```
# apt-get install xfsprogs rsync
```

- d. Format the `/dev/sdb1` and `/dev/sdc1` partitions as XFS:

```
# mkfs.xfs /dev/sdb1
# mkfs.xfs /dev/sdc1
```

- e. Create the mount point directory structure:

```
# mkdir -p /srv/node/sdb1
# mkdir -p /srv/node/sdc1
```

- f. Edit the `/etc/fstab` file and add the following to it:

```
/dev/sdb1 /srv/node/sdb1 xfs noatime,nodiratime,nobarrier,logbufs=8 0
2
/dev/sdc1 /srv/node/sdc1 xfs noatime,nodiratime,nobarrier,logbufs=8 0
2
```

- g. Mount the devices:

```
# mount /srv/node/sdb1
# mount /srv/node/sdc1
```

4. Edit the `/etc/rsyncd.conf` file and add the following to it:

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

[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
```


Replace `MANAGEMENT_INTERFACE_IP_ADDRESS` with the IP address of the management network on the storage node.



Note

The `rsync` service requires no authentication, so consider running it on a private network.

5. Edit the `/etc/default/rsync` file and enable the `rsync` service:

```
RSYNC_ENABLE=true
```

6. Start the `rsync` service:

```
# service rsync start
```

Install and configure storage node components



Note

Perform these steps on each storage node.

1. Install the packages:

```
# apt-get install swift swift-account swift-container swift-object
```

2. Obtain the accounting, container, and object service configuration files from the Object Storage source repository:

```
# curl -o /etc/swift/account-server.conf \
  https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/
account-server.conf-sample
```

```
# curl -o /etc/swift/container-server.conf \
  https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/
container-server.conf-sample
```

```
# curl -o /etc/swift/object-server.conf \
  https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/
object-server.conf-sample
```

3. Edit the `/etc/swift/account-server.conf` file and complete the following actions:
 - a. In the `[DEFAULT]` section, configure the bind IP address, bind port, user, configuration directory, and mount point directory:

```
[DEFAULT]
...
bind_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
bind_port = 6002
user = swift
swift_dir = /etc/swift
devices = /srv/node
```

Replace `MANAGEMENT_INTERFACE_IP_ADDRESS` with the IP address of the management network on the storage node.

- b. In the `[pipeline:main]` section, enable the appropriate modules:

```
[pipeline:main]
pipeline = healthcheck recon account-server
```



Note

For more information on other modules that enable additional features, see the [Deployment Guide](#).

- c. In the `[filter:recon]` section, configure the recon (metrics) cache directory:

```
[filter:recon]
...
recon_cache_path = /var/cache/swift
```

4. Edit the `/etc/swift/container-server.conf` file and complete the following actions:

- a. In the `[DEFAULT]` section, configure the bind IP address, bind port, user, configuration directory, and mount point directory:

```
[DEFAULT]
...
bind_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
bind_port = 6001
user = swift
swift_dir = /etc/swift
devices = /srv/node
```

Replace `MANAGEMENT_INTERFACE_IP_ADDRESS` with the IP address of the management network on the storage node.

- b. In the `[pipeline:main]` section, enable the appropriate modules:

```
[pipeline:main]
pipeline = healthcheck recon container-server
```



Note

For more information on other modules that enable additional features, see the [Deployment Guide](#).

- c. In the `[filter:recon]` section, configure the recon (metrics) cache directory:

```
[filter:recon]
...
recon_cache_path = /var/cache/swift
```

5. Edit the `/etc/swift/object-server.conf` file and complete the following actions:

- a. In the `[DEFAULT]` section, configure the bind IP address, bind port, user, configuration directory, and mount point directory:

```
[DEFAULT]
...
bind_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
bind_port = 6000
user = swift
swift_dir = /etc/swift
devices = /srv/node
```

Replace `MANAGEMENT_INTERFACE_IP_ADDRESS` with the IP address of the management network on the storage node.

- b. In the `[pipeline:main]` section, enable the appropriate modules:

```
[pipeline:main]
pipeline = healthcheck recon object-server
```



Note

For more information on other modules that enable additional features, see the [Deployment Guide](#).

- c. In the `[filter:recon]` section, configure the recon (metrics) cache directory:

```
[filter:recon]
...
recon_cache_path = /var/cache/swift
```

6. Ensure proper ownership of the mount point directory structure:

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

7. Create the `recon` directory and ensure proper ownership of it:

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

Create initial rings

Before starting the Object Storage services, you must create the initial account, container, and object rings. The ring builder creates configuration files that each node uses to determine and deploy the storage architecture. For simplicity, this guide uses one region and zone with 2^{10} (1024) maximum partitions, 3 replicas of each object, and 1 hour minimum time between moving a partition more than once. For Object Storage, a partition indicates a directory on a storage device rather than a conventional partition table. For more information, see the [Deployment Guide](#).

Account ring

The account server uses the account ring to maintain lists of containers.

To create the ring



Note

Perform these steps on the controller node.

1. Change to the `/etc/swift` directory.
2. Create the base `account.builder` file:

```
# swift-ring-builder account.builder create 10 3 1
```

3. Add each storage node to the ring:

```
# swift-ring-builder account.builder \
  add
  r1z1-STORAGE_NODE_MANAGEMENT_INTERFACE_IP_ADDRESS:6002/DEVICE_NAME DEVICE_WEIGHT
```

Replace `STORAGE_NODE_MANAGEMENT_INTERFACE_IP_ADDRESS` with the IP address of the management network on the storage node. Replace `DEVICE_NAME` with a storage device name on the same storage node. For example, using the first storage node in [the section called “Install and configure the storage nodes” \[103\]](#) with the `/dev/sdb1` storage device and weight of 100:

```
# swift-ring-builder account.builder add r1z1-10.0.0.51:6002/sdb1 100
```

Repeat this command for each storage device on each storage node. The example architecture requires four variations of this command.

4. Verify the ring contents:

```
# swift-ring-builder account.builder
account.builder, build version 4
1024 partitions, 3.000000 replicas, 1 regions, 1 zones, 4 devices, 0.00
balance
The minimum number of hours before a partition can be reassigned is 1
Devices:  id region zone      ip address  port  replication ip
          replication port      name weight partitions balance meta
          0          1      1      10.0.0.51  6002      10.0.0.51
          6002          sdb1 100.00      768      0.00
          1          1      1      10.0.0.51  6002      10.0.0.51
          6002          sdc1 100.00      768      0.00
          2          1      1      10.0.0.52  6002      10.0.0.52
          6002          sdb1 100.00      768      0.00
          3          1      1      10.0.0.52  6002      10.0.0.52
          6002          sdc1 100.00      768      0.00
```

5. Rebalance the ring:

```
# swift-ring-builder account.builder rebalance
```



Note

This process can take a while.

Container ring

The container server uses the container ring to maintain lists of objects. However, it does not track object locations.

To create the ring



Note

Perform these steps on the controller node.

1. Change to the `/etc/swift` directory.
2. Create the base `container.builder` file:

```
# swift-ring-builder container.builder create 10 3 1
```

3. Add each storage node to the ring:

```
# swift-ring-builder container.builder \
  add
  r1z1-STORAGE_NODE_MANAGEMENT_INTERFACE_IP_ADDRESS:6001/DEVICE_NAME DEVICE_WEIGHT
```

Replace `STORAGE_NODE_MANAGEMENT_INTERFACE_IP_ADDRESS` with the IP address of the management network on the storage node. Replace `DEVICE_NAME` with a storage device name on the same storage node. For example, using the first storage node in [the section called “Install and configure the storage nodes” \[103\]](#) with the `/dev/sdb1` storage device and weight of 100:

```
# swift-ring-builder container.builder add r1z1-10.0.0.51:6001/sdb1 100
```

Repeat this command for each storage device on each storage node. The example architecture requires four variations of this command.

4. Verify the ring contents:

```
# swift-ring-builder container.builder
container.builder, build version 4
1024 partitions, 3.000000 replicas, 1 regions, 1 zones, 4 devices, 0.00
balance
The minimum number of hours before a partition can be reassigned is 1
Devices:  id region zone  ip address  port  replication ip
         replication port  name weight partitions balance meta
         0          1    1    10.0.0.51  6001    10.0.0.51
         6001      sdb1 100.00    768    0.00
         1          1    1    10.0.0.51  6001    10.0.0.51
         6001      sdc1 100.00    768    0.00
         2          1    1    10.0.0.52  6001    10.0.0.52
         6001      sdb1 100.00    768    0.00
         3          1    1    10.0.0.52  6001    10.0.0.52
         6001      sdc1 100.00    768    0.00
```

5. Rebalance the ring:

```
# swift-ring-builder container.builder rebalance
```



Note

This process can take a while.

Object ring

The object server uses the object ring to maintain lists of object locations on local devices.

To create the ring



Note

Perform these steps on the controller node.

1. Change to the `/etc/swift` directory.
2. Create the base `object.builder` file:

```
# swift-ring-builder object.builder create 10 3 1
```

3. Add each storage node to the ring:

```
# swift-ring-builder object.builder \
add
r1z1-STORAGE_NODE_MANAGEMENT_INTERFACE_IP_ADDRESS:6000/DEVICE_NAME DEVICE_WEIGHT
```

Replace `STORAGE_NODE_MANAGEMENT_INTERFACE_IP_ADDRESS` with the IP address of the management network on the storage node. Replace `DEVICE_NAME` with a storage device name on the same storage node. For example, using the first storage node in [the section called “Install and configure the storage nodes” \[103\]](#) with the `/dev/sdb1` storage device and weight of 100:

```
# swift-ring-builder object.builder add r1z1-10.0.0.51:6000/sdb1 100
```

Repeat this command for each storage device on each storage node. The example architecture requires four variations of this command.

4. Verify the ring contents:

```
# swift-ring-builder object.builder
object.builder, build version 4
1024 partitions, 3.000000 replicas, 1 regions, 1 zones, 4 devices, 0.00
balance
The minimum number of hours before a partition can be reassigned is 1
Devices:  id region zone      ip address  port  replication ip
          replication port      name weight partitions balance meta
          0          1      1      10.0.0.51  6000      10.0.0.51
          6000      sdb1 100.00      768      0.00
          1          1      1      10.0.0.51  6000      10.0.0.51
          6000      sdc1 100.00      768      0.00
          2          1      1      10.0.0.52  6000      10.0.0.52
          6000      sdb1 100.00      768      0.00
          3          1      1      10.0.0.52  6000      10.0.0.52
          6000      sdc1 100.00      768      0.00
```

5. Rebalance the ring:

```
# swift-ring-builder object.builder rebalance
```

**Note**

This process can take a while.

Distribute ring configuration files

Copy the `account.ring.gz`, `container.ring.gz`, and `object.ring.gz` files to the `/etc/swift` directory on each storage node and any additional nodes running the proxy service.

Finalize installation

Configure hashes and default storage policy

1. Obtain the `/etc/swift/swift.conf` file from the Object Storage source repository:

```
# curl -o /etc/swift/swift.conf \
https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/swift.
conf-sample
```

2. Edit the `/etc/swift/swift.conf` file and complete the following actions:

- a. In the `[swift-hash]` section, configure the hash path prefix and suffix for your environment.

```
[swift-hash]
...
swift_hash_path_suffix = HASH_PATH_PREFIX
swift_hash_path_prefix = HASH_PATH_SUFFIX
```

Replace `HASH_PATH_PREFIX` and `HASH_PATH_SUFFIX` with unique values.

**Warning**

Keep these values secret and do not change or lose them.

- b. In the `[storage-policy:0]` section, configure the default storage policy:

```
[storage-policy:0]
...
name = Policy-0
default = yes
```

3. Copy the `swift.conf` file to the `/etc/swift` directory on each storage node and any additional nodes running the proxy service.
4. On all nodes, ensure proper ownership of the configuration directory:

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

5. On the controller node and any other nodes running the proxy service, restart the Object Storage proxy service including its dependencies:

```
# service memcached restart
# service swift-proxy restart
```

6. On the storage nodes, start the Object Storage services:

```
# swift-init all start
```



Note

The storage node runs many Object Storage services and the **swift-init** command makes them easier to manage. You can ignore errors from services not running on the storage node.

Verify operation

This section describes how to verify operation of the Object Storage service.



Note

Perform these steps on the controller node.

1. Source the demo tenant credentials:

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

2. Show the service status:

```
$ 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. Upload a test file:

```
$ swift upload demo-container1 FILE
```

Replace *FILE* with the name of a local file to upload to the `demo-container1` container.

4. List containers:

```
$ swift list
demo-container1
```

5. Download a test file:

```
$ swift download demo-container1 FILE
```

Replace *FILE* with the name of the file uploaded to the `demo-container1` container.

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.

10. Add the Orchestration module

Table of Contents

Orchestration module concepts	114
Install and configure Orchestration	114
Verify operation	118
Next steps	120

The Orchestration module (heat) uses a heat orchestration template (HOT) to create and manage cloud resources.

Orchestration module concepts

The Orchestration module 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 allow you to create most OpenStack resource types, such as instances, floating IPs, volumes, security groups and users. It also provides advanced functionality, such as instance high availability, instance auto-scaling, and nested stacks. This enables OpenStack core projects to receive a larger user base.

The service enables deployers to integrate with the Orchestration module directly or through custom plug-ins.

The Orchestration module consists of the following components:

heat command-line client	A CLI that communicates with the heat-api to run AWS CloudFormation APIs. End developers can directly use the Orchestration REST API.
heat-api component	An OpenStack-native REST API that processes API requests by sending them to the heat-engine over Remote Procedure Call (RPC).
heat-api-cfn component	An AWS Query API that is compatible with AWS CloudFormation. It 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 and configure Orchestration

This section describes how to install and configure the Orchestration module, code-named heat, on the controller node.

To configure prerequisites

Before you install and configure Orchestration, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:

- a. Use the database access client to connect to the database server as the `root` user:

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

- b. Create the `heat` database:

```
CREATE DATABASE heat;
```

- c. Grant proper access to the `heat` database:

```
GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'localhost' \
  IDENTIFIED BY 'HEAT_DBPASS';
GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'%' \
  IDENTIFIED BY 'HEAT_DBPASS';
```

Replace `HEAT_DBPASS` with a suitable password.

- d. Exit the database access client.

2. Source the `admin` credentials to gain access to admin-only CLI commands:

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

3. To create the service credentials, complete these steps:

- a. Create the `heat` user:

```
$ keystone user-create --name heat --pass HEAT_PASS
```

Property	Value
email	
enabled	True
id	7fd67878dcd04d0393469ef825a7e005
name	heat
username	heat

Replace `HEAT_PASS` with a suitable password.

- b. Add the `admin` role to the `heat` user:

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



Note

This command provides no output.

- c. Create the `heat_stack_owner` role:

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

- d. Add the `heat_stack_owner` role to the demo tenant and user:

```
$ keystone user-role-add --user demo --tenant demo --role
heat_stack_owner
```



Note

You must add the `heat_stack_owner` role to users that manage stacks.

- e. Create the `heat_stack_user` role:

```
$ keystone role-create --name heat_stack_user
```



Note

The Orchestration service automatically assigns the `heat_stack_user` role to users that it creates during stack deployment. By default, this role restricts *API* operations. To avoid conflicts, do not add this role to users with the `heat_stack_owner` role.

- f. Create the `heat` and `heat-cfn` service entities:

```
$ keystone service-create --name heat --type orchestration \
--description "Orchestration"
```

Property	Value
description	Orchestration
enabled	True
id	031112165cad4c2bb23e84603957de29
name	heat
type	orchestration

```
$ keystone service-create --name heat-cfn --type cloudformation \
--description "Orchestration"
```

Property	Value
description	Orchestration
enabled	True
id	297740d74c0a446bbff867acdccb33fa
name	heat-cfn
type	cloudformation

4. Create the Orchestration service API endpoints:

```
$ 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 \
--region regionOne
```

Property	Value

```

+-----+
|  adminurl  | http://controller:8004/v1/%(tenant_id)s |
|    id     | f41225f665694b95a46448e8676b0dc2 |
| internalurl | http://controller:8004/v1/%(tenant_id)s |
| publicurl  | http://controller:8004/v1/%(tenant_id)s |
|    region  | regionOne |
| service_id | 031112165cad4c2bb23e84603957de29 |
+-----+
$ 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 \
  --region regionOne
+-----+
| Property | Value |
+-----+
|  adminurl  | http://controller:8000/v1 |
|    id     | f41225f665694b95a46448e8676b0dc2 |
| internalurl | http://controller:8000/v1 |
| publicurl  | http://controller:8000/v1 |
|    region  | regionOne |
| service_id | 297740d74c0a446bbff867acdccb33fa |
+-----+

```

To install and configure the Orchestration components

1. Run the following commands to install the packages:

```
# apt-get install heat-api heat-api-cfn heat-engine python-heatclient
```

2. Edit the `/etc/heat/heat.conf` file and complete the following actions:

- a. In the `[database]` section, configure database access:

```
[database]
...
connection = mysql://heat:HEAT_DBPASS@controller/heat
```

Replace `HEAT_DBPASS` with the password you chose for the Orchestration database.

- b. In the `[DEFAULT]` section, configure RabbitMQ message broker access:

```
[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

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

- c. In the `[keystone_authtoken]` and `[ec2authtoken]` sections, configure Identity service access:

```
[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = heat
admin_password = HEAT_PASS

[ec2authtoken]
...
auth_uri = http://controller:5000/v2.0
```

Replace *HEAT_PASS* with the password you chose for the *heat* user in the Identity service.



Note

Comment out any `auth_host`, `auth_port`, and `auth_protocol` options because the `identity_uri` option replaces them.

- d. In the `[DEFAULT]` section, configure the metadata and wait condition URLs:

```
[DEFAULT]
...
heat_metadata_server_url = http://controller:8000
heat_waitcondition_server_url = http://controller:8000/v1/
waitcondition
```

- e. (Optional) To assist with troubleshooting, enable verbose logging in the `[DEFAULT]` section:

```
[DEFAULT]
...
verbose = True
```

3. Populate the Orchestration database:

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

To finalize installation

1. Restart the Orchestration services:

```
# service heat-api restart
# service heat-api-cfn restart
# service heat-engine restart
```

2. By default, the Ubuntu packages create a SQLite database.

Because this configuration uses a SQL database server, you can remove the SQLite database file:

```
# rm -f /var/lib/heat/heat.sqlite
```

Verify operation

This section describes how to verify operation of the Orchestration module (*heat*).

1. Source the demo tenant credentials:

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

2. The Orchestration module uses templates to describe stacks. To learn about the template language, 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: 2014-10-16
description: A simple server.

parameters:
  ImageID:
    type: string
    description: Image use to boot a server
  NetID:
    type: string
    description: Network ID for the server

resources:
  server:
    type: OS::Nova::Server
    properties:
      image: { get_param: ImageID }
      flavor: ml.tiny
      networks:
        - network: { get_param: NetID }

outputs:
  private_ip:
    description: IP address of the server in the private network
    value: { get_attr: [ server, first_address ] }
```

3. Use the **heat stack-create** command to create a stack from the template:

```
$ NET_ID=$(nova net-list | awk '/ demo-net / { print $2 }')
$ heat stack-create -f test-stack.yml \
  -P "ImageID=cirros-0.3.3-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 |            |                    |
+-----+-----+-----+
+-----+
```

4. Use the **heat stack-list** command to verify successful creation of the stack:

```
$ 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.

11. Add the Telemetry module

Table of Contents

Telemetry module	121
Install and configure controller node	122
Configure the Compute service	125
Configure the Image Service	127
Configure the Block Storage service	127
Configure the Object Storage service	128
Verify the Telemetry installation	129
Next steps	130

Telemetry provides a framework for monitoring and metering the OpenStack cloud. It is also known as the ceilometer project.

Telemetry module

The Telemetry module performs the following functions:

- Efficiently polls metering data related to OpenStack services.
- Collects event and metering data by monitoring notifications sent from services.
- Publishes collected data to various targets including data stores and message queues.
- Creates alarms when collected data breaks defined rules.

The Telemetry module consists of the following 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 our focus is 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. Multiple agents can be started to scale service horizontally.
A notification agent (ceilometer-agent-notification)	Runs on a central management server(s) and consumes messages from the message queue(s) to build event and metering data.
A collector (ceilometer-collector)	Runs on central management server(s) and dispatches collected telemetry data to a data store or external consumer without modification.

An alarm evaluator (ceilometer-alarm-evaluator)	Runs on one or more central management servers to determine when alarms fire due to the associated statistic trend crossing a threshold over a sliding time window.
An alarm notifier (ceilometer-alarm-notifier)	Runs on one or more central management servers to allow alarms to be set based on the threshold evaluation for a collection of samples.
An API server (ceilometer-api)	Runs on one or more central management servers to provide data access from the data store.

These services communicate by using the OpenStack messaging bus. Only the collector and API server have access to the data store.

Install and configure controller node

This section describes how to install and configure the Telemetry module, code-named ceilometer, on the controller node. The Telemetry module uses separate agents to collect measurements from each OpenStack service in your environment.

To configure prerequisites

Before you install and configure Telemetry, you must install MongoDB, create a MongoDB database, service credentials, and API endpoints.

1. Install the MongoDB package:

```
# apt-get install mongodb-server mongodb-clients python-pymongo
```

2. Edit the `/etc/mongodb.conf` file and complete the following actions:

- a. Configure the `bind_ip` key to use the management interface IP address of the controller node.

```
bind_ip = 10.0.0.11
```

- b. By default, MongoDB creates several 1 GB journal files in the `/var/lib/mongodb/journal` directory. If you want to reduce the size of each journal file to 128 MB and limit total journal space consumption to 512 MB, assert the `smallfiles` key:

```
smallfiles = true
```

If you change the journaling configuration, stop the MongoDB service, remove the initial journal files, and start the service:

```
# service mongodb stop
# rm /var/lib/mongodb/journal/prealloc.*
# service mongodb start
```

You can also disable journaling. For more information, see the [MongoDB manual](#).

- c. Restart the MongoDB service:

```
# service mongodb restart
```

3. Create the ceilometer database:

```
# mongo --host controller --eval '
db = db.getSiblingDB("ceilometer");
db.addUser({user: "ceilometer",
pwd: "CEILOMETER_DBPASS",
roles: [ "readWrite", "dbAdmin" ]})'

MongoDB shell version: X.Y.Z
connecting to: controller:27017/test
{
  "user" : "ceilometer",
  "pwd" : "72f25aeee7ad4be52437d7cd3fc60f6f",
  "roles" : [
    "readWrite",
    "dbAdmin"
  ],
  "_id" : ObjectId("5489c22270d7fad1ba631dc3")
}
```

Replace *CEILOMETER_DBPASS* with a suitable password.

4. Source the admin credentials to gain access to admin-only CLI commands:

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

5. To create the service credentials, complete these steps:

a. Create the ceilometer user:

```
$ keystone user-create --name ceilometer --pass CEILOMETER_PASS
```

Replace *CEILOMETER_PASS* with a suitable password.

b. Add the admin role to the ceilometer user.

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

c. Create the ceilometer service entity:

```
$ keystone service-create --name ceilometer --type metering \
--description "Telemetry"
```

6. Create the Telemetry module API endpoints:

```
$ keystone endpoint-create \
--service-id $(keystone service-list | awk '/ metering / {print $2}') \
--publicurl http://controller:8777 \
--internalurl http://controller:8777 \
--adminurl http://controller:8777 \
--region regionOne
```

To install and configure the Telemetry module components

1. Install the packages:

```
# apt-get install ceilometer-api ceilometer-collector ceilometer-agent-
central \
```

```
ceilometer-agent-notification ceilometer-alarm-evaluator ceilometer-  
alarm-notifier \  
python-ceilometerclient
```

2. Generate a random value to use as the metering secret:

```
# openssl rand -hex 10
```

3. Edit the `/etc/ceilometer/ceilometer.conf` file and complete the following actions:

- a. In the `[database]` section, configure database access:

```
[database]  
...  
connection = mongodb://ceilometer:CEILOMETER_DBPASS@controller:27017/  
ceilometer
```

Replace `CEILOMETER_DBPASS` with the password you chose for the Telemetry module database. MongoDB connection string requires escaping special characters as per RFC2396.

- b. In the `[DEFAULT]` section, configure RabbitMQ message broker access:

```
[DEFAULT]  
...  
rpc_backend = rabbit  
rabbit_host = controller  
rabbit_password = RABBIT_PASS
```

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

- c. In the `[DEFAULT]` and `[keystone_authtoken]` sections, configure Identity service access:

```
[DEFAULT]  
...  
auth_strategy = keystone  
  
[keystone_authtoken]  
...  
auth_uri = http://controller:5000/v2.0  
identity_uri = http://controller:35357  
admin_tenant_name = service  
admin_user = ceilometer  
admin_password = CEILOMETER_PASS
```

Replace `CEILOMETER_PASS` with the password you chose for the ceilometer user in the Identity service.



Note

Comment out any `auth_host`, `auth_port`, and `auth_protocol` options because the `identity_uri` option replaces them.

- d. In the `[service_credentials]` section, configure service credentials:

```
[service_credentials]
...
os_auth_url = http://controller:5000/v2.0
os_username = ceilometer
os_tenant_name = service
os_password = CEILOMETER_PASS
```

Replace *CEILOMETER_PASS* with the password you chose for the *ceilometer* user in the Identity service.

- e. In the `[publisher]` section, configure the metering secret:

```
[publisher]
...
metering_secret = METERING_SECRET
```

Replace *METERING_SECRET* with the random value that you generated in a previous step.

- f. (Optional) To assist with troubleshooting, enable verbose logging in the `[DEFAULT]` section:

```
[DEFAULT]
...
verbose = True
```

To finalize installation

- Restart the Telemetry services:

```
# service ceilometer-agent-central restart
# service ceilometer-agent-notification restart
# service ceilometer-api restart
# service ceilometer-collector restart
# service ceilometer-alarm-evaluator restart
# service ceilometer-alarm-notifier restart
```

Configure the Compute service

Telemetry uses a combination of notifications and an agent to collect Compute metrics. Perform these steps on each compute node.

To install and configure the agent

1. Install the packages:

```
# apt-get install ceilometer-agent-compute
```

2. Edit the `/etc/ceilometer/ceilometer.conf` file and complete the following actions:

- a. In the `[publisher]` section, configure the metering secret:

```
[publisher]
...
metering_secret = METERING_SECRET
```

Replace *METERING_SECRET* with the metering secret you chose for the Telemetry module.

- b. In the [DEFAULT] section, configure RabbitMQ message broker access:

```
[DEFAULT]
...
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

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

- c. In the [keystone_authtoken] section, configure Identity service access:

```
[keystone_authtoken]
...
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = ceilometer
admin_password = CEILOMETER_PASS
```

Replace *CEILOMETER_PASS* with the password you chose for the Telemetry module database.



Note

Comment out any *auth_host*, *auth_port*, and *auth_protocol* options because the *identity_uri* option replaces them.

- d. In the [service_credentials] section, configure service credentials:

```
[service_credentials]
...
os_auth_url = http://controller:5000/v2.0
os_username = ceilometer
os_tenant_name = service
os_password = CEILOMETER_PASS
os_endpoint_type = internalURL
os_region_name = regionOne
```

Replace *CEILOMETER_PASS* with the password you chose for the ceilometer user in the Identity service.

- e. (Optional) To assist with troubleshooting, enable verbose logging in the [DEFAULT] section:

```
[DEFAULT]
...
verbose = True
```

To configure notifications

Configure the Compute service to send notifications to the message bus.

- Edit the `/etc/nova/nova.conf` file and configure notifications in the `[DEFAULT]` section:

```
[DEFAULT]
...
instance_usage_audit = True
instance_usage_audit_period = hour
notify_on_state_change = vm_and_task_state
notification_driver = messagingv2
```

To finalize installation

1. Restart the agent:

```
# service ceilometer-agent-compute restart
```

2. Restart the Compute service:

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

Configure the Image Service

To retrieve image-oriented events and samples, configure the Image Service to send notifications to the message bus. Perform these steps on the controller node.

Edit the `/etc/glance/glance-api.conf` and `/etc/glance/glance-registry.conf` files and complete the following actions:

1. In the `[DEFAULT]` section, configure notifications and RabbitMQ message broker access:

```
[DEFAULT]
...
notification_driver = messagingv2
rpc_backend = rabbit
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

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

2. Restart the Image Service:

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

Configure the Block Storage service

To retrieve volume-oriented events and samples, you must configure the Block Storage service to send notifications to the message bus. Perform these steps on the controller and storage nodes.

To configure prerequisites

Edit the `/etc/cinder/cinder.conf` file and complete the following actions:

1. In the `[DEFAULT]` section, configure notifications:

```
[DEFAULT]
...
control_exchange = cinder
notification_driver = messagingv2
```

- Restart the Block Storage services on the controller node:

```
# service cinder-api restart
# service cinder-scheduler restart
```

- Restart the Block Storage services on the storage nodes:

```
# service cinder-volume restart
```

- Use the `cinder-volume-usage-audit` command to retrieve metrics on demand. For more information, see [Block Storage audit script setup to get notifications](#).

Configure the Object Storage service

To retrieve storage-oriented events and samples, configure the Object Storage service to send notifications to the message bus.

To configure prerequisites

The Telemetry service requires access to the Object Storage service using the ResellerAdmin role. Perform these steps on the controller node.

- Source the admin credentials to gain access to admin-only CLI commands.

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

- Create the ResellerAdmin role:

```
$ keystone role-create --name ResellerAdmin
+-----+-----+
| Property | Value |
+-----+-----+
| id       | 462fa46c13fd4798a95a3bfbe27b5e54 |
| name     | ResellerAdmin |
+-----+-----+
```

- Add the ResellerAdmin role to the service tenant and ceilometer user:

```
$ keystone user-role-add --tenant service --user ceilometer \
  --role 462fa46c13fd4798a95a3bfbe27b5e54
```

To configure notifications

Perform these steps on the controller and any other nodes that run the Object Storage proxy service.

- Edit the `/etc/swift/proxy-server.conf` file and complete the following actions:
 - In the `[filter:keystoneauth]` section, add the ResellerAdmin role:

```
[filter:keystoneauth]
...
operator_roles = admin,_member_,ResellerAdmin
```


- b. In the `[pipeline:main]` section, add `ceilometer`:

```
[pipeline:main]
...
pipeline = authtoken cache healthcheck keystoneauth proxy-logging
ceilometer proxy-server
```

- c. In the `[filter:ceilometer]` section, configure notifications:

```
[filter:ceilometer]
use = egg:ceilometer#swift
log_level = WARN
```

2. Add the `swift` system user to the `ceilometer` system group to permit access to the Telemetry configuration files by the Object Storage service:

```
# usermod -a -G ceilometer swift
```

3. Restart the Object Storage proxy service:

```
# service swift-proxy restart
```

Verify the Telemetry installation

This section describes how to verify operation of the Telemetry module.



Note

Perform these commands on the controller node.

1. Source the `admin` credentials to gain access to admin-only CLI commands:

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

2. List available meters:

```
$ ceilometer meter-list
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
| Name      | Type  | Unit  | Resource ID                               | User
| ID | Project ID |                               |
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
| image     | gauge | image | acafc7c0-40aa-4026-9673-b879898elfc2 | None
| efa984b0a914450e9a47788ad330699d |
| image.size | gauge | B      | acafc7c0-40aa-4026-9673-b879898elfc2 | None
| efa984b0a914450e9a47788ad330699d |
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
```



Note

Some meters might not appear until after you successfully launch at least one instance.

3. Download an image from the Image Service:

```
$ glance image-download "cirros-0.3.3-x86_64" > cirros.img
```

4. List available meters again to validate detection of the image download:

```
$ 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		

5. Retrieve usage statistics from the image.download meter:

```
$ 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.

12. Add the Database service

Table of Contents

Database service overview	131
Install the Database service	132
Verify the Database service installation	135

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 use 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. This example is a high-level process flow for using Database services:

1. The OpenStack Administrator configures the basic infrastructure using the following steps:
 - a. Install the Database service.
 - b. Create an image for each type of database. For example, one for MySQL and one for MongoDB.
 - c. Use the **trove-manage** command to import images and offer them to tenants.
2. The OpenStack end user deploys the Database service using the following steps:
 - a. Create a Database service instance using the **trove create** command.
 - b. Use the **trove list** command to get the ID of the instance, followed by the **trove show** command to get the IP address of it.
 - c. Access the Database service instance using typical database access commands. For example, with MySQL:

```
$ mysql -u myuser -p -h TROVE_IP_ADDRESS mydb
```

The Database service includes the following components:

python-troveclient command-line client	A CLI that communicates with the <code>trove-api</code> 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.

- If you want to do backup and restore, you also need Object Storage.
- If you want to provision datastores on block-storage volumes, you also need Block Storage.

To install the Database module on the controller:

1. Install required packages:

```
# apt-get install python-trove python-troveclient python-glanceclient \
trove-common trove-api trove-taskmanager
```

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

Replace `TROVE_PASS` with a suitable password.

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 = mysql://trove:TROVE_DBPASS@controller/trove
notifier_queue_hostname = controller
```

- b. Configure the Database module to use the RabbitMQ message broker by setting the following options in the [DEFAULT] configuration group of each file:

```
[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = controller
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_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
admin_user = trove
admin_password = ADMIN_PASS
admin_tenant_name = service
signing_dir = /var/cache/trove
```

5. Edit the trove.conf file so it includes appropriate values for the default datastore, network label regex, and API information as shown below:

```
[DEFAULT]
default_datastore = mysql
....
# Config option for showing the IP address that nova does out
add_addresses = True
network_label_regex = ^NETWORK_LABEL$
....
api_paste_config = /etc/trove/api-paste.ini
```

6. Edit the trove-taskmanager.conf file so it includes the required settings 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
taskmanager_manager = trove.taskmanager.manager.Manager
...
```

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 \
--region regionOne
```

12. Restart the Database services:

```
# service trove-api restart
# service trove-taskmanager restart
# service trove-conductor restart
```

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
```

13. Add the Data processing service

Table of Contents

Data processing service	136
Install the Data processing service	137
Verify the Data processing service installation	138

The Data processing service (sahara) enables users to provide a scalable data processing stack and associated management interfaces. This includes provision and operation of data processing clusters as well as scheduling and operation of data processing jobs.



Warning

This chapter is a work in progress. It may contain incorrect information, and will be updated frequently.

Data processing service

The Data processing service for OpenStack (sahara) aims to provide users with simple means to provision data processing (Hadoop, Spark) clusters by specifying several parameters like Hadoop version, cluster topology, nodes hardware details and a few more. After user fills in all the parameters, the Data processing service deploys the cluster in a few minutes. Also sahara provides means to scale already provisioned clusters by adding/removing worker nodes on demand.

The solution addresses the following use cases:

- Fast provisioning of Hadoop clusters on OpenStack for development and QA.
- Utilization of unused compute power from general purpose OpenStack IaaS cloud.
- Analytics-as-a-Service for ad-hoc or bursty analytic workloads.

Key features are:

- Designed as an OpenStack component.
- Managed through REST API with UI available as part of OpenStack dashboard.
- Support for different Hadoop distributions:
 - Pluggable system of Hadoop installation engines.
 - Integration with vendor specific management tools, such as Apache Ambari or Cloudera Management Console.
- Predefined templates of Hadoop configurations with ability to modify parameters.

- User-friendly UI for ad-hoc analytics queries based on Hive or Pig.

Install the Data processing service

This procedure installs the Data processing service (sahara) on the controller node.

To install the Data processing service on the controller:

1.



Warning

You need to install the required packages. For now, sahara doesn't have packages for Ubuntu. Documentation will be updated once the packages are available. The rest of this document assumes that you have the sahara service packages installed on the system.

2. Edit `/etc/sahara/sahara.conf` configuration file

- First, edit `connection` parameter in the `[database]` section. The URL provided here should point to an empty database. For instance, connection string for MySQL database will be:

```
connection = mysql://sahara:SAHARA_DBPASS@controller/sahara
```

- Switch to the `[keystone_auth_token]` section. The `auth_uri` parameter should point to the public Identity API endpoint. `identity_uri` should point to the admin Identity API endpoint. For example:

```
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
```

- Next specify `admin_user`, `admin_password` and `admin_tenant_name`. These parameters must specify a keystone user which has the `admin` role in the given tenant. These credentials allow sahara to authenticate and authorize its users.
- Switch to the `[DEFAULT]` section. Proceed to the networking parameters. If you are using Neutron for networking, then set `use_neutron=true`. Otherwise if you are using nova-network set the given parameter to `false`.
- That should be enough for the first run. If you want to increase logging level for troubleshooting, there are two parameters in the config: `verbose` and `debug`. If the former is set to `true`, sahara will start to write logs of `INFO` level and above. If `debug` is set to `true`, sahara will write all the logs, including the `DEBUG` ones.

3. If you use the Data processing service with a MySQL or MariaDB database, you must configure the maximum number of allowed packets for storing big job binaries in the service internal database.

- Edit the `/etc/mysql/my.cnf` file and change the `max_allowed_packet` option:

```
[mysqld]
max_allowed_packet = 256M
```

- Restart the database service:

```
# service mysql restart
```

4. Create database schema:

```
# sahara-db-manage --config-file /etc/sahara/sahara.conf upgrade head
```

5. You must register the Data processing service with the Identity service so that other OpenStack services can locate it. Register the service and specify the endpoint:

```
$ keystone service-create --name sahara --type data_processing \
--description "Data processing service"
$ keystone endpoint-create \
--service-id $(keystone service-list | awk '/ sahara / {print $2}') \
--publicurl http://controller:8386/v1.1/tenant_id/s \
--internalurl http://controller:8386/v1.1/tenant_id/s \
--adminurl http://controller:8386/v1.1/tenant_id/s \
--region regionOne
```

6. Start the sahara service:

```
# systemctl start openstack-sahara-all
```

7. (Optional) Enable the Data processing service to start on boot

```
# systemctl enable openstack-sahara-all
```

Verify the Data processing service installation

To verify that the Data processing service (sahara) is installed and configured correctly, try requesting clusters list using sahara client.

1. Source the demo tenant credentials:

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

2. Retrieve sahara clusters list:

```
$ sahara cluster-list
```

You should see output similar to this:

```
+-----+-----+-----+-----+
| name | id | status | node_count |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

14. Launch an instance

Table of Contents

Launch an instance with OpenStack Networking (neutron)	139
Launch an instance with legacy networking (nova-network)	147

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 4, “Add the Image Service” \[42\]](#) 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 key pair

Most cloud images support *public key authentication* rather than conventional user name/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 | ml.tiny   | 512       | 1    | 0         |      | 1     | 1.0
|   | True     |
| 2 | ml.small  | 2048      | 20   | 0         |      | 1     | 1.0
|   | True     |
| 3 | ml.medium | 4096      | 40   | 0         |      | 2     | 1.0
|   | True     |
| 4 | ml.large  | 8192      | 80   | 0         |      | 4     | 1.0
|   | True     |
| 5 | ml.xlarge | 16384     | 160  | 0         |      | 8     | 1.0
|   | True     |
+-----+-----+-----+-----+-----+-----+-----+
```

Your first instance uses the `ml.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-0.3.3-x86_64 | ACTIVE |
+-----+-----+-----+
```

Your first instance uses the `cirros-0.3.3-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 |          | b54a8d85-b434-4e85- |
| 9bce64a3-a963-4c05-bfcd-161f708042d1 | ext-net   | 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-0.3.3-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-0.3.3-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          |
+-----+-----+-----+-----+-----+
| 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.

- 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
+-----+
+-----+
+
```



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 user name/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
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.

To attach a Block Storage volume to your instance

If your environment includes the Block Storage service, you can attach a volume to the instance.

1. Source the demo tenant credentials:

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

2. List volumes:

```
$ nova volume-list
+-----+-----+-----+-----+
| ID                                     | Status   | Display Name | Size |
| Volume Type | Attached to |              |      |
+-----+-----+-----+-----+
| 158bea89-07db-4ac2-8115-66c0d6a4bb48 | available | demo-volume1 | 1    |
| None          |              |              |      |
+-----+-----+-----+-----+
```

3. Attach the `demo-volume1` volume to the `demo-instance1` instance:

```
$ nova volume-attach demo-instance1 158bea89-07db-4ac2-8115-66c0d6a4bb48
+-----+-----+-----+-----+
```

Property	Value
device	/dev/vdb
id	158bea89-07db-4ac2-8115-66c0d6a4bb48
serverId	05682b91-81a1-464c-8f40-8b3da7ee92c5
volumeId	158bea89-07db-4ac2-8115-66c0d6a4bb48

**Note**

You must reference volumes using the IDs instead of names.

4. List volumes:

```
$ nova volume-list
```

ID	Status	Display Name	Size
Volume Type	Attached to		
158bea89-07db-4ac2-8115-66c0d6a4bb48	in-use	demo-volume1	1
None	05682b91-81a1-464c-8f40-8b3da7ee92c5		

The demo-volume1 volume status should indicate in-use by the ID of the demo-instance1 instance.

5. Access your instance using SSH from the controller node or any host on the external network and use the **fdisk** command to verify presence of the volume as the `/dev/vdb` block storage device:

```
$ ssh cirros@203.0.113.102
$ sudo fdisk -l
```

```
Disk /dev/vda: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders, total 2097152 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000
```

Device	Boot	Start	End	Blocks	Id	System
/dev/vda1	*	16065	2088449	1036192+	83	Linux

```
Disk /dev/vdb: 1073 MB, 1073741824 bytes
16 heads, 63 sectors/track, 2080 cylinders, total 2097152 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

Disk /dev/vdb doesn't contain a valid partition table
```

**Note**

You must create a partition table and file system to use the volume.

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 (no- va-network)

To generate a key pair

Most cloud images support *public key authentication* rather than conventional user name/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	ml.tiny	512	1	0	1	1.0
	True					
2	ml.small	2048	20	0	1	1.0
	True					
3	ml.medium	4096	40	0	2	1.0
	True					
4	ml.large	8192	80	0	4	1.0
	True					
5	ml.xlarge	16384	160	0	8	1.0
	True					

Your first instance uses the `ml.tiny` flavor.



Note

You can also reference a flavor by ID.

2. List available images:

```
$ nova image-list
```

ID	Name	Status
acafc7c0-40aa-4026-9673-b879898e1fc2	cirros-0.3.3-x86_64	ACTIVE

Your first instance uses the `cirros-0.3.3-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-0.3.3-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-0.3.3-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 user name/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.

To attach a Block Storage volume to your instance

If your environment includes the Block Storage service, you can attach a volume to the instance.

1. Source the demo tenant credentials:

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

2. List volumes:

```
$ nova volume-list
+-----+-----+-----+-----+-----+
| ID                                     | Status      | Display Name | Size |
+-----+-----+-----+-----+-----+
| 158bea89-07db-4ac2-8115-66c0d6a4bb48 | available   | demo-volume1 | 1    |
| None                                |             |              |      |
+-----+-----+-----+-----+-----+
```

3. Attach the `demo-volume1` volume to the `demo-instance1` instance:

```
$ nova volume-attach demo-instance1 158bea89-07db-4ac2-8115-66c0d6a4bb48
+-----+-----+-----+
| Property | Value                                     |
+-----+-----+-----+
| device   | /dev/vdb                                |
| id        | 158bea89-07db-4ac2-8115-66c0d6a4bb48    |
| serverId  | 45ea195c-c469-43eb-83db-1a663bbad2fc    |
| volumeId  | 158bea89-07db-4ac2-8115-66c0d6a4bb48    |
+-----+-----+-----+
```



Note

You must reference volumes using the IDs instead of names.

4. List volumes:

```
$ nova volume-list
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
```


ID	Status	Display Name	Size
Volume Type Attached to			
158bea89-07db-4ac2-8115-66c0d6a4bb48	in-use	demo-volume1	1
None 45ea195c-c469-43eb-83db-1a663bbad2fc			

The demo-volume1 volume status should indicate in-use by the ID of the demo-instance1 instance.

5. Access your instance using SSH from the controller node or any host on the external network and use the **fdisk** command to verify presence of the volume as the `/dev/vdb` block storage device:

```
$ ssh cirros@203.0.113.102
$ sudo fdisk -l

Disk /dev/vda: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders, total 2097152 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/vda1    *        16065      2088449     1036192+   83   Linux

Disk /dev/vdb: 1073 MB, 1073741824 bytes
16 heads, 63 sectors/track, 2080 cylinders, total 2097152 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

Disk /dev/vdb doesn't contain a valid partition table
```



Note

You must create a partition table and file system to use the volume.

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

OpenStack reserves certain user IDs to run specific services and own specific files. These user IDs 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	Assigned during package installation
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

Documentation	155
ask.openstack.org	156
OpenStack mailing lists	156
The OpenStack wiki	157
The Launchpad Bugs area	157
The OpenStack IRC channel	158
Documentation feedback	158
OpenStack distribution packages	158

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](#)
- [Installation Guide for openSUSE 13.1 and SUSE Linux Enterprise Server 11 SP3](#)
- [Installation Guide for Red Hat Enterprise Linux 7, CentOS 7, and Fedora 20](#)
- [Installation Guide for Ubuntu 14.04](#)

The following books explain how to configure and run an OpenStack cloud:

- [Architecture Design Guide](#)
- [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 \(developer.openstack.org\)](#)
- [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 Compute service is nova.

compute node

A node that runs the `nova-compute` daemon that manages VM instances that provide a wide range of services, such as 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.

dnsmasq

Daemon that provides DNS, DHCP, BOOTP, and TFTP services for virtual networks.

extended attributes (xattr)

File system option that enables storage of additional information beyond owner, group, permissions, modification time, and so on. The underlying Object Storage file system must support extended attributes.

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

Virtual network type that uses neither VLANs nor tunnels to segregate tenant traffic. Each flat network typically requires a separate underlying physical interface defined by bridge mappings. However, a flat network can contain multiple subnets.

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.

generic routing encapsulation (GRE)

Protocol that encapsulates a wide variety of network layer protocols inside virtual point-to-point links.

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.

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

A running VM, or a VM in a known state such as suspended, that can be used like a hardware server.

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.

Internet protocol (IP)

Principal communications protocol in the internet protocol suite for relaying datagrams across network boundaries.

ipset

Extension to iptables that allows creation of firewall rules that match entire "sets" of IP addresses simultaneously. These sets reside in indexed data structures to increase efficiency, particularly on systems with a large quantity of rules.

iptables

Used along with arptables and ebtables, iptables create firewalls in Compute. iptables are the tables provided by the Linux kernel firewall (implemented as different Netfilter modules) and the chains and rules it stores. Different kernel modules and programs are currently used for different protocols: iptables applies to IPv4, ip6tables to IPv6, arptables to ARP, and ebtables to Ethernet frames. Requires root privilege to manipulate.

iSCSI

The SCSI disk protocol tunneled within Ethernet, supported by Compute, Object Storage, and Image Service.

jumbo frame

Feature in modern Ethernet networks that supports frames up to approximately 9000 bytes.

kernel-based VM (KVM)

An OpenStack-supported hypervisor. KVM is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V), ARM, IBM Power, and IBM zSeries. It consists of a loadable kernel module, that provides the core virtualization infrastructure and a processor specific module.

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.

maximum transmission unit (MTU)

Maximum frame or packet size for a particular network medium. Typically 1500 bytes for Ethernet networks.

message broker

The software package used to provide AMQP messaging capabilities within Compute. Default package is RabbitMQ.

message queue

Passes requests from clients to the appropriate workers and returns the output to the client after the job completes.

Metadata agent

OpenStack Networking agent that provides metadata services for instances.

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 namespace

Linux kernel feature that provides independent virtual networking instances on a single host with separate routing tables and interfaces. Similar to virtual routing and forwarding (VRF) services on physical network equipment.

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.

Open vSwitch

Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols (for example NetFlow, sFlow, SPAN, RSPAN, CLI, LACP, 802.1ag).

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.

path MTU discovery (PMTUD)

Mechanism in IP networks to detect end-to-end MTU and adjust packet size accordingly.

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.

Quick EMUlator (QEMU)

QEMU is a generic and open source machine emulator and virtualizer.

One of the hypervisors supported by OpenStack, generally used for development purposes.

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 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

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

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.

XFS

High-performance 64-bit file system created by Silicon Graphics. Excels in parallel I/O operations and data consistency.