

OpenStack Installation Guide

for Ubuntu 12.04 (LTS)

havana (December 10, 2013)



OpenStack Installation Guide for Ubuntu 12.04 (LTS)

havana (2013-12-10)

Copyright © 2012, 2013 OpenStack Foundation All rights reserved.

The OpenStack® system consists of several key projects that you install separately but that work together depending on your cloud needs. These projects include Compute, Identity Service, Networking, Image Service, Block Storage Service, Object Storage, Telemetry, and Orchestration. You can install any of these projects separately and configure them standalone or as connected entities. This guide walks through an installation by using packages available through Ubuntu 12.04 (LTS). 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	8
Document change history	8
1. Architecture	1
Conceptual architecture	2
Logical architecture	2
Sample architectures	3
2. Basic operating system configuration	5
Before you begin	5
Networking	5
Network Time Protocol (NTP)	7
MySQL database	7
OpenStack packages	8
Messaging server	8
3. Configure the Identity Service	9
Identity Service concepts	9
Install the Identity Service	10
Define users, tenants, and roles	11
Define services and API endpoints	12
Verify the Identity Service installation	13
4. Configure the Image Service	15
Image Service overview	15
Install the Image Service	16
Verify the Image Service installation	18
5. Configure Compute services	20
Compute service	20
Install Compute controller services	23
Configure a Compute node	25
Enable Networking	27
Launch an instance	28
6. Add the dashboard	34
System requirements	34
Install the dashboard	35
Set up session storage for the dashboard	36
7. Add the Block Storage Service	40
Block Storage Service	40
Configure a Block Storage Service controller	40
Configure a Block Storage Service node	42
8. Add Object Storage	45
Object Storage service	45
System requirements	45
Plan networking for Object Storage	46
Example Object Storage installation architecture	47
Install Object Storage	48
Install and configure storage nodes	49
Install and configure the proxy node	51
Start services on the storage nodes	54
Object Storage post-installation tasks	54
9. Install the Networking service	57

Networking considerations	57
Neutron concepts	57
Install Networking services	59
Neutron deployment use cases	72
10. Add the Orchestration service	106
Orchestration service overview	106
Install the Orchestration service	106
Verify the Orchestration service installation	108
11. Add the Telemetry service	112
The Telemetry Service	112
Install the Telemetry service	113
Install the Compute agent for the Telemetry service	115
Add the Image Service agent for the Telemetry service	116
Add the Block Storage Service agent for the Telemetry service	116
Add the Object Storage agent for the Telemetry service	116
Verify the Telemetry Service installation	117
A. Community support	119
Documentation	119
ask.openstack.org	120
OpenStack mailing lists	120
The OpenStack wiki	120
The Launchpad Bugs area	121
The OpenStack IRC channel	121
Documentation feedback	122
OpenStack distribution packages	122

List of Figures

1.1. OpenStack conceptual architecture	2
1.2. Logical architecture	3
1.3. Basic architecture	4
2.1. Basic architecture	6

List of Tables

1.1. OpenStack services	1
8.1. Hardware recommendations	46
9.1. Nodes for use case	82

List of Examples

2.1. /etc/network/interfaces	6
------------------------------------	---

Preface

Document change history

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

Revision Date	Summary of Changes
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.
October 8, 2013	<ul style="list-style-type: none"> Complete reorganization for Havana.
September 9, 2013	<ul style="list-style-type: none"> Build also for openSUSE.
August 1, 2013	<ul style="list-style-type: none"> Fixes to Object Storage verification steps. Fix bug 1207347.
July 25, 2013	<ul style="list-style-type: none"> Adds creation of cinder user and addition to the service tenant. Fix bug 1205057.
May 8, 2013	<ul style="list-style-type: none"> Updated the book title for consistency.
May 2, 2013	<ul style="list-style-type: none"> Updated cover and fixed small errors in appendix.
April 30, 2013	<ul style="list-style-type: none"> Grizzly release.
April 18, 2013	<ul style="list-style-type: none"> Updates and clean up on the Object Storage installation.
April 8, 2013	<ul style="list-style-type: none"> Adds a note about availability of Grizzly packages on Ubuntu and Debian.
April 3, 2013	<ul style="list-style-type: none"> Updates RHEL/CentOS/Fedora information for Grizzly release.
March 26, 2013	<ul style="list-style-type: none"> Updates Dashboard (Horizon) information for Grizzly release.
February 12, 2013	<ul style="list-style-type: none"> Adds chapter about Essex to Folsom upgrade for Compute and related services (excludes OpenStack Object Storage (Swift) and OpenStack Networking (Quantum)).
January 16, 2013	<ul style="list-style-type: none"> Fix file copy issue for figures in the /common/ directory.
November 9, 2012	<ul style="list-style-type: none"> Folsom release of this document.
October 10, 2012	<ul style="list-style-type: none"> Doc bug fixes: 10544591064745
September 26, 2012	<ul style="list-style-type: none"> Adds an all-in-one install section.
July 23, 2012	<ul style="list-style-type: none"> Adds additional detail about installing and configuring nova-volumes. Doc bug fixes: 9785101027230
July 17, 2012	<ul style="list-style-type: none"> Update build process so two uniquely-named PDF files are output.
July 13, 2012	<ul style="list-style-type: none"> Doc bug fixes: 1025840 1025847
June 19, 2012	<ul style="list-style-type: none"> Fix PDF links. Doc bug fixes: 967778 984959, 1002294, 1010163.
May 31, 2012	<ul style="list-style-type: none"> Revise install guide to encompass more Linux distros. Doc bug fixes: 996988, 998116, 999005.
May 3, 2012	<ul style="list-style-type: none"> Fixes problems with <code>glance-api-paste.ini</code> and <code>glance-registry-paste.ini</code> samples and instructions. Removes "DRAFT" designation.
May 2, 2012	<ul style="list-style-type: none"> Essex release.
May 1, 2012	<ul style="list-style-type: none"> Updates the Object Storage and Identity (Keystone) configuration.
April 25, 2012	<ul style="list-style-type: none"> Changes <code>service_id</code> copy/paste error for the EC2 <code>service-create</code> command. Adds verification steps for Object Storage installation. Fixes <code>proxy-server.conf</code> file so it points to keystone not tempauth.
April 23, 2012	<ul style="list-style-type: none"> Adds installation and configuration for multi-node Object Storage service.
April 17, 2012	<ul style="list-style-type: none"> Doc bug fixes: 983417, 984106, 984034

Revision Date	Summary of Changes
April 13, 2012	<ul style="list-style-type: none">• Doc bug fixes: 977905, 980882, 977823, adds additional Glance database preparation steps
April 10, 2012	<ul style="list-style-type: none">• Doc bug fixes: 977831
March 23, 2012	<ul style="list-style-type: none">• Updates for Xen hypervisor.
March 9, 2012	<ul style="list-style-type: none">• Updates for Essex release, includes new Glance config files, new Keystone configuration.
January 24, 2012	<ul style="list-style-type: none">• Initial draft for Essex.<ul style="list-style-type: none">• Assumes use of Ubuntu 12.04 repository.
January 24, 2011	<ul style="list-style-type: none">• Initial draft for Diablo.

1. Architecture

Table of Contents

Conceptual architecture	2
Logical architecture	2
Sample architectures	3

This install guide offers a few of the many ways to install OpenStack components and have them work together. It is meant as a "choose your own adventure" guide, not a comprehensive guide. The *OpenStack Configuration Reference* lists every option in all OpenStack services. Before you begin an installation adventure, here are some things you should know about OpenStack concepts.

The OpenStack project is an open source cloud computing platform for all types of clouds, which aims to be simple to implement, massively scalable, and feature rich. Developers and cloud computing technologists from around the world create the OpenStack project.

OpenStack provides an Infrastructure as a Service (IaaS) solution through a set of interrelated services. Each service offers an application programming interface (API) that facilitates this integration. Depending on your needs, you can install some or all services.

The following table describes the OpenStack services that make up the OpenStack architecture:

Table 1.1. OpenStack services

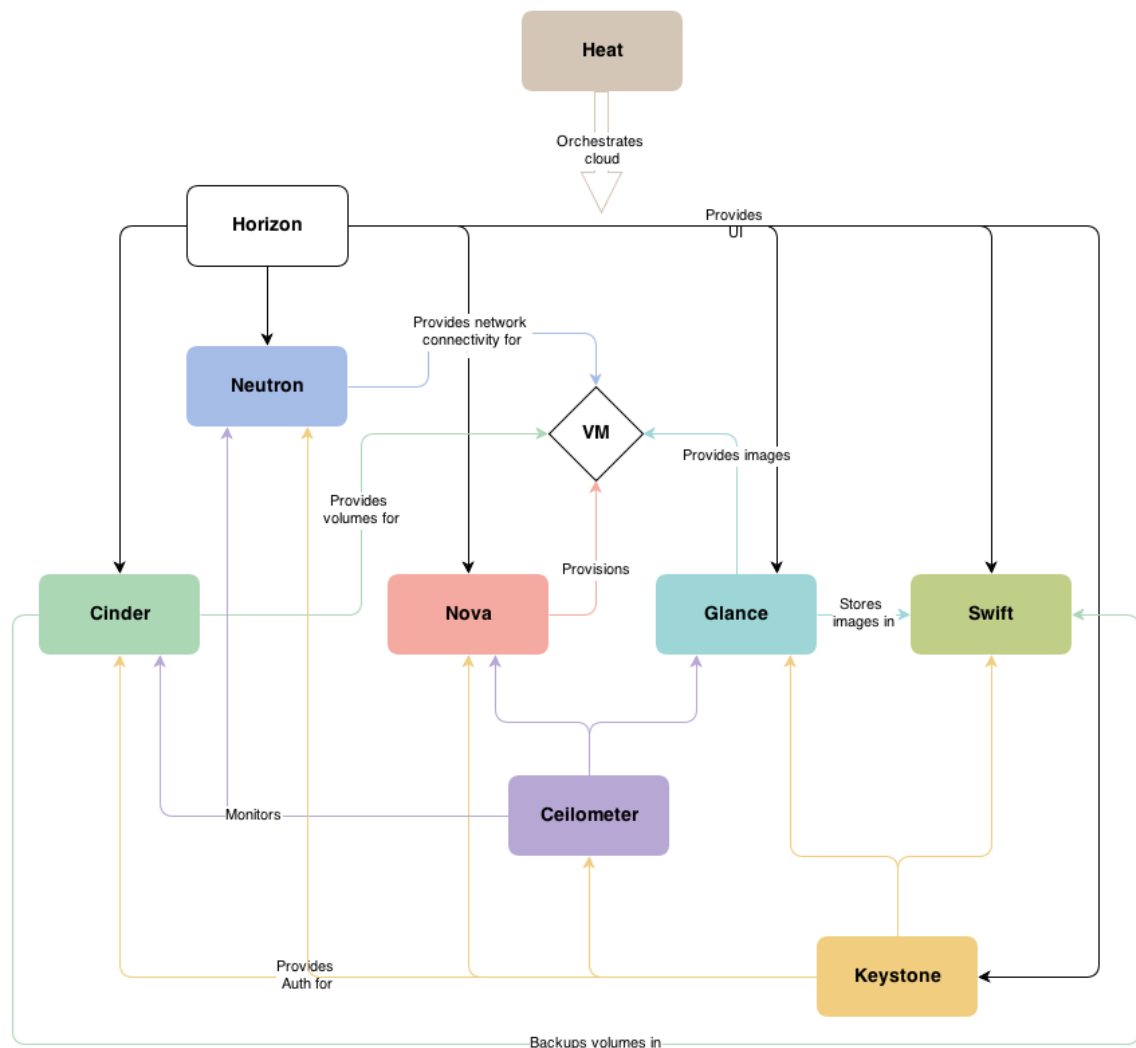
Service	Project name	Description
Dashboard	Horizon	Enables users to interact with OpenStack services to launch an instance, assign IP addresses, set access controls, and so on.
Compute	Nova	Provisions and manages large networks of virtual machines on demand.
Networking	Neutron	Enables network connectivity as a service among interface devices managed by other OpenStack services, usually Compute. Enables users to create and attach interfaces to networks. Has a pluggable architecture that supports many popular networking vendors and technologies.
Storage		
Object Storage	Swift	Stores and gets files. Does not mount directories like a file server.
Block Storage	Cinder	Provides persistent block storage to guest virtual machines.
Shared services		
Identity Service	Keystone	Provides authentication and authorization for the OpenStack services. Also provides a service catalog within a particular OpenStack cloud.
Image Service	Glance	Provides a registry of virtual machine images. Compute uses it to provision instances.
Telemetry Service	Ceilometer	Monitors and meters the OpenStack cloud for billing, benchmarking, scalability, and statistics purposes.
Higher-level services		

Service	Project name	Description
Orchestration Service	Heat	Orchestrates multiple composite cloud applications by using either the native HOT template format or the AWS CloudFormation template format, through both an OpenStack-native REST API and a CloudFormation-compatible Query API.

Conceptual architecture

The following diagram shows the relationships among the OpenStack services:

Figure 1.1. OpenStack conceptual architecture

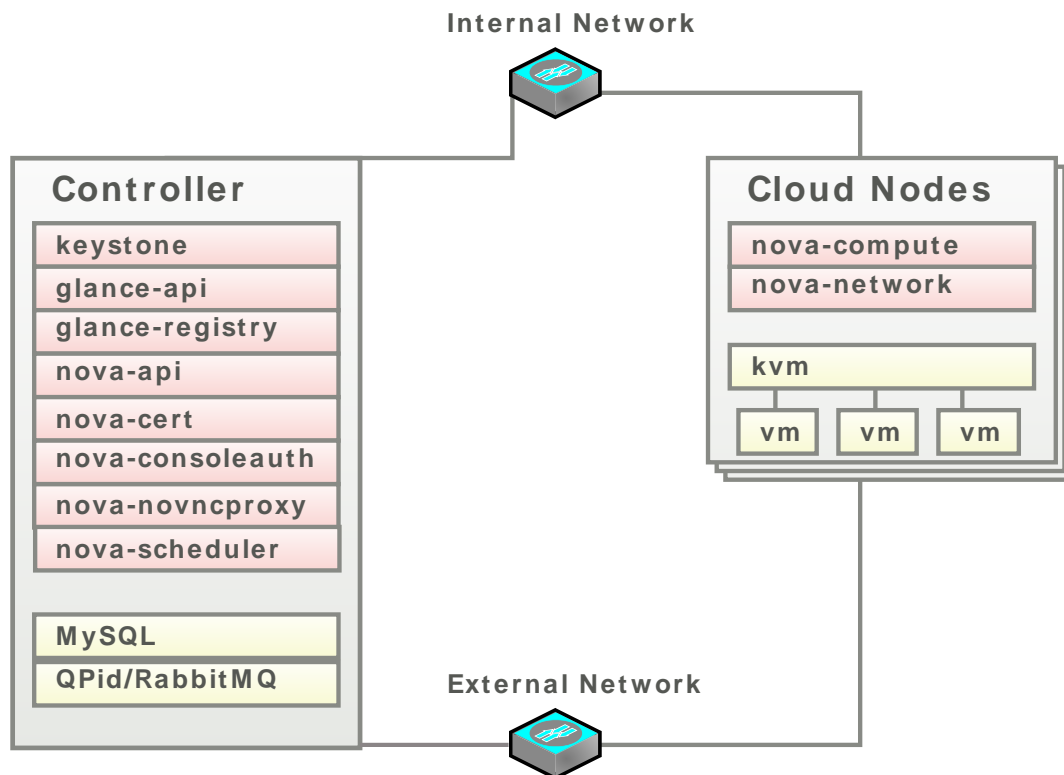


Logical architecture

To design, install, and configure a cloud, cloud administrators must understand the logical architecture.

OpenStack modules are one of the following types:

Figure 1.3. Basic architecture



Technical details: Compute with KVM, local ephemeral storage, nova-network in multi-host flatDHCP mode, MySQL, nova-api, default scheduler, RabbitMQ for messaging, Identity Service with SQL back end, Image Service with local storage, Dashboard (optional extra). Uses as many default options as possible.

- Example architecture from the [OpenStack Operations Guide](#). Same as the basic architecture but with the Block Storage Service LVM/iSCSI back end, nova-network in multi-host with FlatDHCP, Live Migration back end, shared storage with NFS, and Object Storage. One controller node and multiple compute nodes.
- Example architecture with Identity Service and Object Storage: Five-node installation with Identity Service on the proxy node and three replications of object servers. Dashboard does not support this configuration so examples are with CLI.
- Example architecture with OpenStack Networking.

2. Basic operating system configuration

Table of Contents

Before you begin	5
Networking	5
Network Time Protocol (NTP)	7
MySQL database	7
OpenStack packages	8
Messaging server	8

This guide shows you how to create a controller node to host most services and a compute node to run virtual machine instances. Subsequent chapters create additional nodes to run more services. OpenStack is flexible about how and where you run each service, so other configurations are possible. However, you must configure certain operating system settings on each node.

This chapter details a sample configuration for the controller node and any additional nodes. You can configure the operating system in other ways, but this guide assumes that your configuration is compatible with the one described here.

All example commands assume you have administrative privileges. Either run the commands as the root user or prefix them with the `sudo` command.

Before you begin

We strongly recommend that you install a 64-bit operating system on your compute nodes. If you use a 32-bit operating system, attempting to start a virtual machine using a 64-bit image will fail with an error.

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

Networking

For an OpenStack production deployment, most nodes must have these network interface cards:

- One network interface card for external network traffic
- Another card to communicate with other OpenStack nodes.

For simple test cases, you can use machines with a single network interface card.

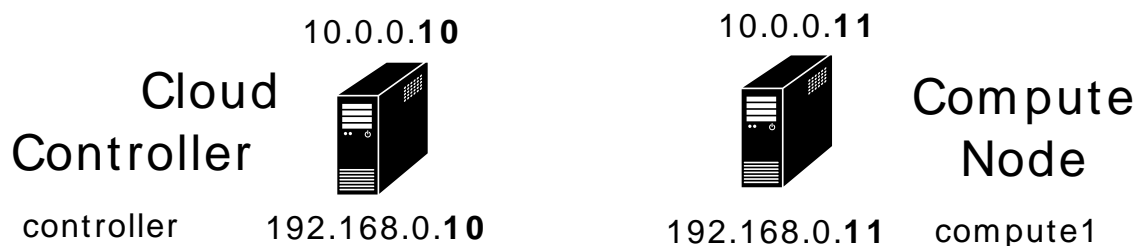
The following example configures Networking on two networks with static IP addresses and manually manages a list of host names on each machine. If you manage a large network, you might already have systems in place to manage this. If so, you can skip this

section but note that the rest of this guide assumes that each node can reach the other nodes on the internal network by using the `controller` and `compute1` host names.

Configure both `eth0` and `eth1`. The examples in this guide use the `192.168.0.x` IP addresses for the internal network and the `10.0.0.x` IP addresses for the external network. Make sure to connect your network devices to the correct network.

In this guide, the controller node uses the `192.168.0.10` and `10.0.0.10` IP addresses. When you create the compute node, use the `192.168.0.11` and `10.0.0.11` addresses instead. Additional nodes that you add in subsequent chapters also follow this pattern.

Figure 2.1. Basic architecture



Example 2.1. `/etc/network/interfaces`

```
# Internal Network
auto eth0
iface eth0 inet static
    address 192.168.0.10
    netmask 255.255.255.0

# External Network
auto eth1
iface eth1 inet static
    address 10.0.0.10
    netmask 255.255.255.0
```

After you configure the network, restart the daemon for changes to take effect:

```
# service networking restart
```

Set the host name of each machine. Name the controller node `controller` and the first compute node `compute1`. The examples in this guide use these host names.

Use the `hostname` command to set the host name:

```
# hostname controller
```

To configure this host name to be available when the system reboots, you must specify it in the `/etc/hostname` file, which contains a single line with the host name.

Finally, ensure that each node can reach the other nodes by using host names. You must manually edit the `/etc/hosts` file on each system. For large-scale deployments, use DNS or a configuration management system like Puppet.

```
127.0.0.1    localhost
192.168.0.10 controller
192.168.0.11 compute1
```

Network Time Protocol (NTP)

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

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

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

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

MySQL database

Most OpenStack services require a database to store information. These examples use a MySQL database that runs on the controller node. You must install the MySQL database on the controller node. You must install MySQL client software on any additional nodes that access MySQL.

- On the controller node, install the MySQL client and server packages, and the Python library.

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



Note

When you install the server package, you are prompted for the root password for the database. Choose a strong password and remember it.

Edit `/etc/mysql/my.cnf` and set the `bind-address` to the internal IP address of the controller, to enable access from outside the controller node.

```
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
bind-address            = 192.168.0.10
```

Restart the MySQL service to apply the changes:

```
# service mysql restart
```

- On nodes other than the controller node, install the MySQL client and the MySQL Python library on any system that does not host a MySQL database.

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

You must delete the anonymous users that are created when the database is first started. Otherwise, database connection problems occur when you follow the instructions in this guide. To do this, use the `mysql_secure_installation` command.

```
# mysql_secure_installation
```


This command presents a number of options for you to secure your database installation. Respond **yes** to all prompts unless you have a good reason to do otherwise.

OpenStack packages

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

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

To use the Ubuntu Cloud Archive for Havana

The [Ubuntu Cloud Archive](#) is a special repository that allows you to install newer releases of OpenStack on the stable supported version of Ubuntu.

1. Install the Ubuntu Cloud Archive for Havana:

```
# apt-get install python-software-properties  
# add-apt-repository cloud-archive:havana
```

2. Upgrade the system (and reboot if you need):

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

Messaging server

On the controller node, install the messaging queue server. Typically this is RabbitMQ but Qpid and ZeroMQ (0MQ) are also available.

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



Important security consideration

The `rabbitmq-server` package configures the RabbitMQ service to start automatically and creates a `guest` user with a default `guest` password. The RabbitMQ examples in this guide use the `guest` account, though it is strongly advised to change its default password, especially if you have IPv6 available: by default the RabbitMQ server enables anyone to connect to it by using `guest` as login and password, and with IPv6, it is reachable from the outside.

To change the default `guest` password of RabbitMQ:

```
# rabbitmqctl change_password guest RABBIT_PASS
```

Congratulations, now you are ready to install OpenStack services!

3. Configure the Identity Service

Table of Contents

Identity Service concepts	9
Install the Identity Service	10
Define users, tenants, and roles	11
Define services and API endpoints	12
Verify the Identity Service installation	13

Identity Service concepts

The Identity Service performs the following functions:

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

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

User Digital representation of a person, system, or service who uses OpenStack cloud services. The Identity Service validates that incoming requests are made by the user who claims to be making the call. Users have a login and may be assigned tokens to access resources. Users can be directly assigned to a particular tenant and behave as if they are contained in that tenant.

Credentials Data that is known only by a user that proves who they are. In the Identity Service, examples are: User name and password, user name and API key, or an authentication token provided by the Identity Service.

Authentication The act of confirming the identity of a user. The Identity Service confirms an incoming request by validating a set of credentials supplied by the user.

These credentials are initially a user name and password or a user name and API key. In response to these credentials, the Identity Service issues an authentication token to the user, which the user provides in subsequent requests.

Token An arbitrary bit of text that is used to access resources. Each token has a scope which describes which resources are accessible with it. A token may be revoked at any time and is valid for a finite duration.

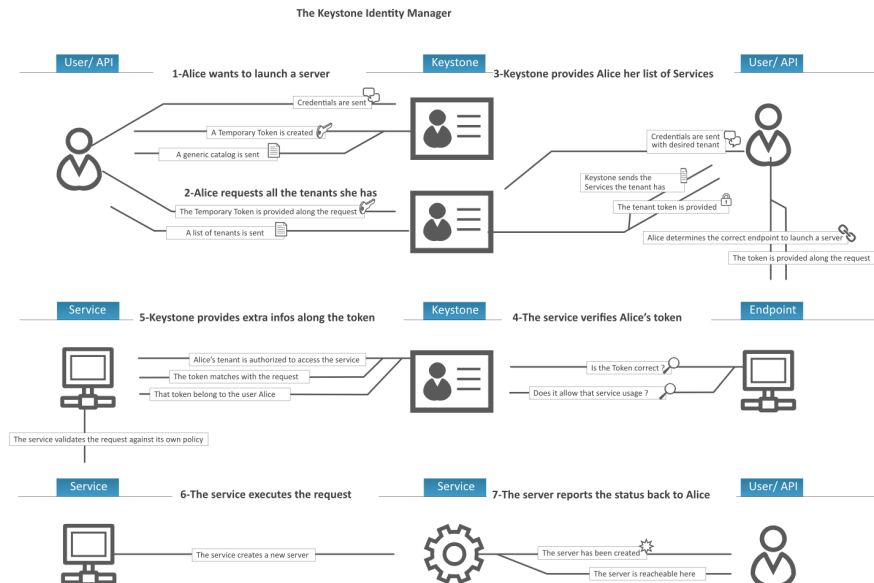
While the Identity Service supports token-based authentication in this release, the intention is for it to support additional protocols in the future. The intent is for it to be an integration service foremost,

and not aspire to be a full-fledged identity store and management solution.

- Tenant** A container used to group or isolate resources and/or identity objects. Depending on the service operator, a tenant may map to a customer, account, organization, or project.
- Service** An OpenStack service, such as Compute (Nova), Object Storage (Swift), or Image Service (Glance). Provides one or more endpoints through which users can access resources and perform operations.
- Endpoint** An network-accessible address, usually described by URL, from where you access a service. If using an extension for templates, you can create an endpoint template, which represents the templates of all the consumable services that are available across the regions.
- Role** A personality that a user assumes that enables them to perform a specific set of operations. A role includes a set of rights and privileges. A user assuming that role inherits those rights and privileges.

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

The following diagram shows the Identity Service process flow:



Install the Identity Service

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

```
# apt-get install keystone
```

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

Edit `/etc/keystone/keystone.conf` and change the `[sql]` section.

```
...
[sql]
# The SQLAlchemy connection string used to connect to the database
connection = mysql://keystone:KEYSTONE_DBPASS@controller/keystone
...
```

3. By default, the Ubuntu packages create an SQLite database. Delete the `keystone.db` file created in the `/var/lib/keystone/` directory so that it does not get used by mistake.
4. Use the password that you set previously to log in as root. Create a keystone database user:

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

5. Start the keystone service and create its tables:

```
# keystone-manage db_sync
# service keystone restart
```

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

```
# openssl rand -hex 10
```

Edit `/etc/keystone/keystone.conf` and change the `[DEFAULT]` section, replacing `ADMIN_TOKEN` with the results of the command.

```
[DEFAULT]
# A "shared secret" between keystone and other openstack services
admin_token = ADMIN_TOKEN
...
```

7. Restart the Identity Service:

```
# service keystone restart
```

Define users, tenants, and roles

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

Typically, you would indicate a user and password to authenticate with the Identity Service. At this point, however, we have not created any users, so we have to use the authorization token created in the previous section. You can pass this with the `--os-token` option to the `keystone` command or set the `OS_SERVICE_TOKEN` environment variable. We'll set `OS_SERVICE_TOKEN`, as well as `OS_SERVICE_ENDPOINT` to specify where the Identity Service is running. Replace `FCAF3E...` with your authorization token.

```
# export OS_SERVICE_TOKEN=FCAF3E...
# export OS_SERVICE_ENDPOINT=http://controller:35357/v2.0
```

First, create a tenant for an administrative user and a tenant for other OpenStack services to use.

```
# keystone tenant-create --name=admin --description="Admin Tenant"
# keystone tenant-create --name=service --description="Service Tenant"
```

Next, create an administrative user called `admin`. Choose a password for the `admin` user and specify an email address for the account.

```
# keystone user-create --name=admin --pass=ADMIN_PASS \
  --email=admin@example.com
```

Create a role for administrative tasks called `admin`. Any roles you create should map to roles specified in the `policy.json` files of the various OpenStack services. The default policy files use the `admin` role to allow access to most services.

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

Finally, you have to add roles to users. Users always log in with a tenant, and roles are assigned to users within tenants. Add the `admin` role to the `admin` user when logging in with the `admin` tenant.

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

Define services and API endpoints

The Identity Service also tracks what OpenStack services are installed and where to locate them on the network. Run these commands for each service in your OpenStack installation:

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

For now, create a service for the Identity Service itself that uses normal authentication instead of the authorization token when you run the `keystone` command in the future.

1. Create a service entry for the Identity Service:

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

type	identity

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

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

```
# keystone endpoint-create \  
--service-id=the_service_id_above \  
--publicurl=http://controller:5000/v2.0 \  
--internalurl=http://controller:5000/v2.0 \  
--adminurl=http://controller:35357/v2.0
```

Property	Value
adminurl	http://controller:35357/v2.0
id	11f9c625a3b94a3f8e66bf4e5de2679f
internalurl	http://controller:5000/v2.0
publicurl	http://controller:5000/v2.0
region	regionOne
service_id	15c11a23667e427e91bc31335b45f4bd

- As you add other services to your OpenStack installation, call these commands to register the services with the Identity Service.

Verify the Identity Service installation

To verify the Identity Service is installed and configured correctly, first unset the `OS_SERVICE_TOKEN` and `OS_SERVICE_ENDPOINT` environment variables. These were only used to bootstrap the administrative user and register the Identity Service.

```
# unset OS_SERVICE_TOKEN OS_SERVICE_ENDPOINT
```

You can now use regular username-based authentication. Request a authentication token using the `admin` user and the password you chose for that user.

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

You should receive a token in response, paired with your user ID. This verifies that keystone is running on the expected endpoint, and that your user account is established with the expected credentials.

Next, verify that authorization is behaving as expected by requesting authorization on a tenant.

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

You should receive a new token in response, this time including the ID of the tenant you specified. This verifies that your user account has an explicitly defined role on the specified tenant, and that the tenant exists as expected.

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

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

You can source this file to read in the environment variables.

```
# source keystonerc
```

Verify that your `keystonerc` is configured correctly by performing the same command as above, but without the `--os-*` arguments.

```
$ keystone token-get
```

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

Finally, verify that your admin account has authorization to perform administrative commands.

```
# keystone user-list
```

```
+-----+-----+-----+-----+
|          id          | enabled | email          | name |
+-----+-----+-----+-----+
| a4c2d43f80a549a19864c89d759bb3fe | True   | admin@example.com | admin |
```

This verifies that your user account has the `admin` role, which matches the role used in the Identity Service `policy.json` file.

4. Configure the Image Service

Table of Contents

Image Service overview	15
Install the Image Service	16
Verify the Image Service installation	18

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



Important

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

Ensure that the system has sufficient space available under this directory to store virtual machine images and snapshots before proceeding. At an absolute minimum several gigabytes of space should be available for use by the Image Service in a proof of concept deployment.

Image Service overview

The Image Service includes the following components:

- `glance-api`. Accepts Image API calls for image discovery, retrieval, and storage.
- `glance-registry`. Stores, processes, and retrieves metadata about images. Metadata includes size, type, and so on.
- Database. Stores image metadata. You can choose your database depending on your preference. Most deployments use MySQL or SQLite.
- Storage repository for image files. In [Figure 1.2, “Logical architecture” \[3\]](#), the Object Storage Service is the image repository. However, you can configure a different repository. The Image Service supports normal file systems, RADOS block devices, Amazon S3, and HTTP. Some choices provide only read-only usage.

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

As shown in [the section called “Conceptual architecture” \[2\]](#), the Image Service is central to the overall IaaS picture. It accepts API requests for images or image metadata from end users or Compute components and can store its disk files in the Object Storage Service.

Install the Image Service

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



Note

This procedure assumes you set the appropriate environment variables to your credentials as described in [the section called “Verify the Identity Service installation” \[13\]](#).

1. Install the Image Service on the controller node:

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

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

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

Edit `/etc/glance/glance-api.conf` and `/etc/glance/glance-registry.conf` and change the `[DEFAULT]` section.

```
...
[DEFAULT]
...
# SQLAlchemy connection string for the reference implementation
# registry server. Any valid SQLAlchemy connection string is fine.
# See: http://www.sqlalchemy.org/docs/05/reference/sqlalchemy/connections.
html#sqlalchemy.create_engine
sql_connection = mysql://glance:GLANCE_DBPASS@controller/glance
...
```

3. By default, the Ubuntu packages create an SQLite database. Delete the `glance.sqlite` file created in the `/var/lib/glance/` directory so that it does not get used by mistake.
4. Use the password you created to log in as root and create a `glance` database user:

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

```
IDENTIFIED BY 'GLANCE_DBPASS';
```

5. Create the database tables for the Image Service:

```
# glance-manage db_sync
```

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

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

7. Add the credentials to the Image Service configuration files:

Edit `/etc/glance/glance-api.conf` and `/etc/glance/glance-registry.conf` and change the `[keystone_authtoken]` section.

```
...  
[keystone_authtoken]  
auth_host = controller  
auth_port = 35357  
auth_protocol = http  
admin_tenant_name = service  
admin_user = glance  
admin_password = GLANCE_PASS  
...
```

8. Add the credentials to the `/etc/glance/glance-api-paste.ini` and `/etc/glance/glance-registry-paste.ini` files.

Edit each file to set the following options in the `[filter:authtoken]` section and leave any other existing option as it is.

```
[filter:authtoken]  
paste.filter_factory=keystoneclient.middleware.auth_token:filter_factory  
auth_host=controller  
admin_user=glance  
admin_tenant_name=service  
admin_password=GLANCE_PASS
```

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

```
# keystone service-create --name=glance --type=image \  
  --description="Glance Image Service"
```

10. Use the `id` property returned for the service to create the endpoint:

```
# keystone endpoint-create \  
  --service-id=the_service_id_above \  
  --publicurl=http://controller:9292 \  
  --internalurl=http://controller:9292 \  
  --adminurl=http://controller:9292
```

11. Restart the glance service with its new settings.

```
# service glance-registry restart
```

```
# service glance-api restart
```

Verify the Image Service installation

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

For more information about how to download and build images, see [OpenStack Virtual Machine Image Guide](#). For information about how to manage images, see the [OpenStack User Guide](#).

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

```
$ mkdir images
$ cd images/
$ wget http://cdn.download.cirros-cloud.net/0.3.1/cirros-0.3.1-x86_64-disk.img
```

2. Upload the image to the Image Service:

```
# glance image-create --name=imageLabel --disk-format=fileFormat \
  --container-format=containerFormat --is-public=accessValue < imageFile
```

Where:

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

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

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

```
$ file cirros-0.3.1-x86_64-disk.img
cirros-0.3.1-x86_64-disk.img: QEMU QCOW Image (v2),
41126400 bytes
```

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

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

accessValue Specifies image access:

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

imageFile Specifies the name of your downloaded image file.

For example:

```
# glance image-create --name="Cirros 0.3.1" --disk-format=qcow2 \  
--container-format=bare --is-public=true < cirros-0.3.1-x86_64-disk.img
```

Property	Value
checksum	d972013792949d0d3ba628fbe8685bce
container_format	bare
created_at	2013-10-08T18:59:18
deleted	False
deleted_at	None
disk_format	qcow2
id	acafc7c0-40aa-4026-9673-b879898e1fc2
is_public	True
min_disk	0
min_ram	0
name	Cirros 0.3.1
owner	efa984b0a914450e9a47788ad330699d
protected	False
size	13147648
status	active
updated_at	2013-05-08T18:59:18



Note

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

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

```
# glance image-list
```

ID	Name	Disk Format
Container Format	Size	Status
acafc7c0-40aa-4026-9673-b879898e1fc2	Cirros 0.3.1	qcow2
bare	13147648	active

5. Configure Compute services

Table of Contents

Compute service	20
Install Compute controller services	23
Configure a Compute node	25
Enable Networking	27
Launch an instance	28

Compute service

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

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

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

API

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

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

Compute core

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

- `nova-scheduler` process. Conceptually the simplest piece of code in Compute. Takes a virtual machine instance request from the queue and determines on which compute server host it should run.
- `nova-conductor` module. Mediates interactions between `nova-compute` and the database. Aims to eliminate direct accesses to the cloud database made by `nova-compute`. The `nova-conductor` module scales horizontally. However, do not deploy it on any nodes where `nova-compute` runs. For more information, see [A new Nova service: nova-conductor](#).

Networking for VMs

- `nova-network` worker daemon. Similar to `nova-compute`, it accepts networking tasks from the queue and performs tasks to manipulate the network, such as setting up bridging interfaces or changing iptables rules. This functionality is being migrated to OpenStack Networking, which is a separate OpenStack service.
- `nova-dhcpbridge` script. Tracks IP address leases and records them in the database by using the `dnsmasq dhcp-script` facility. This functionality is being migrated to OpenStack Networking. OpenStack Networking provides a different script.

Console interface

- `nova-consoleauth` daemon. Authorizes tokens for users that console proxies provide. See `nova-novncproxy` and `nova-xvncproxy`. This service must be running for console proxies to work. Many proxies of either type can be run against a single `nova-consoleauth` service in a cluster configuration. For information, see [About nova-consoleauth](#).
- `nova-novncproxy` daemon. Provides a proxy for accessing running instances through a VNC connection. Supports browser-based novnc clients.
- `nova-console` daemon. Deprecated for use with Grizzly. Instead, the `nova-xvncproxy` is used.
- `nova-xvncproxy` daemon. A proxy for accessing running instances through a VNC connection. Supports a Java client specifically designed for OpenStack.
- `nova-cert` daemon. Manages x509 certificates.

Image management (EC2 scenario)

- `nova-objectstore` daemon. Provides an S3 interface for registering images with the Image Service. Mainly used for installations that must support euca2ools. The euca2ools tools talk to `nova-objectstore` in *S3 language*, and `nova-objectstore` translates S3 requests into Image Service requests.
- euca2ools client. A set of command-line interpreter commands for managing cloud resources. Though not an OpenStack module, you can configure `nova-api` to support this EC2 interface. For more information, see the [Eucalyptus 2.0 Documentation](#).

Command-line clients and other interfaces

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

Other components

- The queue. A central hub for passing messages between daemons. Usually implemented with [RabbitMQ](#), but could be any AMPQ message queue, such as [Apache Qpid](#) or [Zero MQ](#).
- SQL database. Stores most build-time and runtime states for a cloud infrastructure. Includes instance types that are available for use, instances in use, available networks, and projects. Theoretically, OpenStack Compute can support any database that SQL-Alchemy supports, but the only databases widely used are sqlite3 databases (only appropriate for test and development work), MySQL, and PostgreSQL.

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

Install Compute controller services

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

1. Install these Compute packages, which provide the Compute services that run on the controller node.

```
# apt-get install nova-novncproxy novnc nova-api \  
nova-ajax-console-proxy nova-cert nova-conductor \  
nova-consoleauth nova-doc nova-scheduler \  
python-novaclient
```

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

Configure the location of the database. Replace *NOVA_DBPASS* with your Compute service password:

Edit the `/etc/nova/nova.conf` file and add these lines to the `[database]` and `[keystone_authtoken]` sections:

```
...  
[database]  
# The SQLAlchemy connection string used to connect to the database  
connection = mysql://nova:NOVA_DBPASS@controller/nova  
[keystone_authtoken]  
auth_host = controller  
auth_port = 35357  
auth_protocol = http  
admin_tenant_name = service  
admin_user = nova  
admin_password = NOVA_PASS
```

3. Configure the Compute Service to use the RabbitMQ message broker by setting these configuration keys in the `[DEFAULT]` configuration group of the `/etc/nova/nova.conf` file:

```
rpc_backend = nova.rpc.impl_kombu  
rabbit_host = controller  
rabbit_password = RABBIT_PASS
```

4. By default, the Ubuntu packages create an SQLite database. Delete the `nova.sqlite` file created in the `/var/lib/nova/` directory so that it does not get used by mistake.
5. Use the password you created previously to log in as root. Create a nova database user:

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


6. Create the Compute service tables:

```
# nova-manage db sync
```

7. Set the `my_ip`, `vncserver_listen`, and `vncserver_proxyclient_address` configuration options to the internal IP address of the controller node:

Edit the `/etc/nova/nova.conf` file and add these lines to the `[DEFAULT]` section:

```
...
[DEFAULT]
...
my_ip=192.168.0.10
vncserver_listen=192.168.0.10
vncserver_proxyclient_address=192.168.0.10
```

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

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

9. For Compute to use these credentials, you must edit the `nova.conf` configuration file:

Edit the `/etc/nova/nova.conf` file and add these lines to the `[DEFAULT]` section:

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

10. Add the credentials to the `/etc/nova/api-paste.ini` file. Add these options to the `[filter:authtoken]` section:



Use of .ini files

You might sometimes have to edit `.ini` files during initial setup. However, do not edit these files for general configuration tasks.

```
[filter:authtoken]
paste.filter_factory=keystoneclient.middleware.auth_token:filter_factory
auth_host=controller
auth_port=5000
auth_protocol=http
auth_uri=http://controller:5000/v2.0
admin_tenant_name=service
admin_user=nova
admin_password=NOVA_PASS
```



Note

Ensure that the `api_paste_config=/etc/nova/api-paste.ini` option is set in the `/etc/nova/nova.conf` file.

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

```
# keystone service-create --name=nova --type=compute \  
--description="Nova Compute service"
```

12. Use the `id` property that is returned to create the endpoint.

```
# keystone endpoint-create \  
--service-id=the_service_id_above \  
--publicurl=http://controller:8774/v2/%(tenant_id)s \  
--internalurl=http://controller:8774/v2/%(tenant_id)s \  
--adminurl=http://controller:8774/v2/%(tenant_id)s
```

13. Restart 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
```

14. To verify your configuration, list available images:

```
# nova image-list  
+-----+-----+-----+-----+  
+-----+  
| ID | Name | Status | Server |  
+-----+-----+-----+-----+  
+-----+  
| acafc7c0-40aa-4026-9673-b879898e1fc2 | CirrOS 0.3.1 | ACTIVE | |  
+-----+-----+-----+-----+  
+-----+
```

Configure a Compute node

After you configure the Compute service on the controller node, you must configure another system as a Compute node. The Compute node receives requests from the controller node and hosts virtual machine instances. You can run all services on a single node, but the examples in this guide use separate systems. This makes it easy to scale horizontally by adding additional Compute nodes following the instructions in this section.

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

1. Configure the system. Use the instructions in [Chapter 2, "Basic operating system configuration" \[5\]](#), but note the following differences from the controller node:
 - Use different IP addresses when you configure `eth0`. This guide uses `192.168.0.11` for the internal network. Do not configure `eth1` with a static IP address. The networking component of OpenStack assigns and configures an IP address.
 - Set the host name to `compute1`. To verify, use the `uname -n` parameter. Ensure that the IP addresses and host names for both nodes are listed in the `/etc/hosts` file on each system.

- Synchronize from the controller node. Follow the instructions in [the section called “Network Time Protocol \(NTP\)” \[7\]](#).
 - Install the MySQL client libraries. You do not need to install the MySQL database server or start the MySQL service.
 - Enable the OpenStack packages for the distribution that you are using. See [the section called “OpenStack packages” \[8\]](#).
2. After you configure the operating system, install the appropriate packages for the Compute service.

Run this command:

```
# apt-get install nova-compute-kvm python-guestfs
```

When prompted to create a `supermin` appliance, respond **yes**.

3. Due to [this bug](#), which is marked `Won't Fix`, `guestfs` is restricted. Run this command to relax the restriction:

```
# chmod 0644 /boot/vmlinuz*
```

4. Edit the `/etc/nova/nova.conf` configuration file and add these lines to the appropriate sections:

```
...  
[DEFAULT]  
...  
auth_strategy=keystone  
...  
[database]  
# The SQLAlchemy connection string used to connect to the database  
connection = mysql://nova:NOVA_DBPASS@controller/nova
```

5. Configure the Compute Service to use the RabbitMQ message broker by setting these configuration keys in the `[DEFAULT]` configuration group of the `/etc/nova/nova.conf` file:

```
rpc_backend = nova.rpc.impl_kombu  
rabbit_host = controller  
rabbit_password = RABBIT_PASS
```

6. Set the `my_ip`, `vncserver_listen`, and `vncserver_proxyclient_address` configuration keys to the IP address of the compute node on the internal network:

Edit `/etc/nova/nova.conf` and add to the `[DEFAULT]` section.

```
[DEFAULT]  
...  
my_ip=192.168.0.11  
vnc_enabled=True  
vncserver_listen=0.0.0.0  
vncserver_proxyclient_address=192.168.0.11  
novncproxy_base_url=http://controller:6080/vnc_auto.html
```

- Specify the host that runs the Image Service. Edit `/etc/nova/nova.conf` file and add these lines to the `[DEFAULT]` section:

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

- Edit the `/etc/nova/api-paste.ini` file to add the credentials to the `[filter:authtoken]` section:



Use of .ini files

Files with the extension `.ini` sometimes need to be edited during initial setup. However, they should not be used for general configuration tasks.

```
[filter:authtoken]
paste.filter_factory=keystoneclient.middleware.auth_token:filter_factory
auth_host=controller
auth_port = 35357
auth_protocol = http
admin_user=nova
admin_tenant_name=service
admin_password=NOVA_PASS
```

- Restart the Compute service.

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

- Remove the SQLite database created by the packages:

```
# rm /var/lib/nova/nova.sqlite
```

Enable Networking

Configuring Networking can be a bewildering experience. The following example shows the simplest production-ready configuration that is available: the legacy networking in OpenStack Compute, with a flat network, that takes care of DHCP.

This set up uses multi-host functionality. Networking is configured to be highly available by distributing networking functionality across multiple hosts. As a result, no single network controller acts as a single point of failure. This process configures each compute node for networking.



Note

If you need the full software-defined networking stack, see [Chapter 9, “Install the Networking service” \[57\]](#).

- Install the appropriate packages for compute networking:

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

- Edit the `nova.conf` file to define the networking mode:

Edit the `/etc/nova/nova.conf` file and add these lines to the `[DEFAULT]` section:

```
[DEFAULT]
...

network_manager=nova.network.manager.FlatDHCPManager
firewall_driver=nova.virt.libvirt.firewall.IptablesFirewallDriver
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=eth1
public_interface=eth1
```

3. Restart the network service:

```
# service nova-network restart
```

Create a network that virtual machines can use. Do this once for the entire installation and not on each compute node. Run the **nova network-create** command on the controller:

```
# source keystonerc
```

```
# nova network-create vmnet --fixed-range-v4=10.0.0.0/24 \
--bridge-interface=br100 --multi-host=T
```

Launch an instance

After you configure the Compute services, you can launch an instance. An instance is a virtual machine that OpenStack provisions on a Compute servers. This example shows you how to launch a low-resource instance by using a downloaded image.



Note

This procedure assumes you have:

- Installed the nova client library on the machine on which you will run the commands (log on the controller if you are not sure).
- Set environment variables to specify your credentials. See [the section called “Verify the Identity Service installation” \[13\]](#).
- Downloaded an image. See [the section called “Verify the Image Service installation” \[18\]](#).
- Configured networking. See [the section called “Enable Networking” \[27\]](#).

1. Generate a keypair that consists of a private and public key to be able to launch instances on OpenStack. These keys are injected into the instances to make password-less SSH access to the instance. This depends on the way the necessary tools are bundled into the images. For more details, see the [OpenStack Admin User Guide](#).

```
$ ssh-keygen
$ cd .ssh
```

```
$ nova keypair-add --pub_key id_rsa.pub mykey
```

You have just created the `mykey` keypair. The `id_rsa` private key is saved locally in `~/.ssh`, which you can use to connect to an instance launched by using `mykey` as the keypair. To view available keypairs:

```
$ nova keypair-list
+-----+-----+
| Name | Fingerprint |
+-----+-----+
| mykey | b0:18:32:fa:4e:d4:3c:1b:c4:6c:dd:cb:53:29:13:82 |
+-----+-----+
```

- To launch an instance, you must specify the ID for the flavor you want to use for the instance. A flavor is a resource allocation profile. For example, it specifies how many virtual CPUs and how much RAM your instance gets. To see a list of the available profiles:

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

- Get the ID of the image to use for the instance:

```
$ nova image-list
+-----+-----+-----+-----+
| ID | Name | Status | Server |
+-----+-----+-----+-----+
| 9e5c2bee-0373-414c-b4af-b91b0246ad3b | CirrOS 0.3.1 | ACTIVE | |
+-----+-----+-----+-----+
```

- To use SSH and ping, you must configure security group rules. See the [OpenStack User Guide](#).

```
# nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
```

```
# nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
```

- Launch the instance:

```
$ nova boot --flavor flavorType --key_name keypairName --
image ID newInstanceName
```

Create an instance by using flavor 1 or 2. For example:

```
$ nova boot --flavor 1 --key_name mykey --image 9e5c2bee-0373-414c-b4af-
b91b0246ad3b --security_group default cirrOS
-----
+-----+
| Property | Value |
+-----+
| OS-EXT-STS:task_state | scheduling |
| image | CirrOS 0.3.1 |
| OS-EXT-STS:vm_state | building |
| OS-EXT-SRV-ATTR:instance_name | instance-00000001 |
| OS-SRV-USG:launched_at | None |
| flavor | ml.tiny |
| id | 3bdf98a0-c767-4247-
bf41-2d147e4aa043 |
| security_groups | [{u'name': u'default'}] |
| user_id | 530166901fa24d1face95cda82cfae56 |
| OS-DCF:diskConfig | MANUAL |
| accessIPv4 | |
| accessIPv6 | |
| progress | 0 |
| OS-EXT-STS:power_state | 0 |
| OS-EXT-AZ:availability_zone | nova |
| config_drive | |
| status | BUILD |
| updated | 2013-10-10T06:47:26Z |
| hostId | |
| OS-EXT-SRV-ATTR:host | None |
| OS-SRV-USG:terminated_at | None |
| key_name | mykey |
| OS-EXT-SRV-ATTR:hypervisor_hostname | None |
| name | cirrOS |
| adminPass | DWCDW6FnsKNq |
|
```

```
| tenant_id | e66d97ac1b704897853412fc8450f7b9 |
| | |
| created | 2013-10-10T06:47:23Z |
| | |
| os-extended-volumes:volumes_attached | [] |
| | |
| metadata | {} |
| | |
+-----+
+-----+
```



Note

If sufficient RAM is not available for the instance, Compute creates, but does not start, the instance and sets the status for the instance to `ERROR`.

6. After the instance launches, use the `nova list` to view its status. The status changes from `BUILD` to `ACTIVE`:

```
$ nova list
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| ID | Name | Status | Task State |
+-----+-----+-----+-----+
| dcc4a894-869b-479a-a24a-659eef7a54bd | cirrOS | BUILD | spawning |
+-----+-----+-----+-----+
| NOSTATE | vmnet=10.0.0.3 |
+-----+-----+-----+-----+
$ nova list
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| ID | Name | Status | Task State |
+-----+-----+-----+-----+
| dcc4a894-869b-479a-a24a-659eef7a54bd | cirrOS | ACTIVE | None |
+-----+-----+-----+-----+
| Running | vmnet=10.0.0.3 |
+-----+-----+-----+-----+
```



Note

To show details for a specified instance:

```
$ nova show dcc4a894-869b-479a-a24a-659eef7a54bd
+-----+
+-----+-----+-----+
| Property | Value |
+-----+-----+-----+
| status | ACTIVE |
+-----+-----+-----+
| updated | 2013-10-16T21:55:24Z |
+-----+-----+-----+
| OS-EXT-STS:task_state | None |
+-----+-----+-----+
+-----+
```



```
| OS-EXT-SRV-ATTR:host                | compute-node  
| key_name                            | mykey  
| image                               | cirros  
(918a1017-8a1b-41ff-8809-6106ba45366e) |  
| vmnet network                       | 10.0.0.3  
| hostId                              |  
306d7c693911170ad4e5218f626f531cc68caa45f3a0f70f1aeba94d |  
| OS-EXT-STS:vm_state                 | active  
| OS-EXT-SRV-ATTR:instance_name      | instance-0000000a  
| OS-SRV-USG:launched_at             | 2013-10-16T21:55:24.  
000000  
| OS-EXT-SRV-ATTR:hypervisor_hostname | compute-node  
| flavor                              | m1.tiny (1)  
| id                                  | dcc4a894-869b-479a-  
a24a-659eef7a54bd  
| security_groups                     | [{u'name': u'default'}]  
| OS-SRV-USG:terminated_at           | None  
| user_id                             |  
887ac8736b5b473b9dc3c5430a88b15f  
| name                                | cirrOS  
| created                             | 2013-10-16T21:54:52Z  
| tenant_id                           |  
43ab520b2b484578bb6924c0ea926190  
| OS-DCF:diskConfig                   | MANUAL  
| metadata                            | {}  
| os-extended-volumes:volumes_attached | []  
| accessIPv4                           |  
| accessIPv6                           |  
| progress                             | 0  
| OS-EXT-STS:power_state               | 1  
| OS-EXT-AZ:availability_zone          | nova  
| config_drive                         |  
+-----+  
+-----+
```

7. After the instance boots and initializes and you have configured security groups, you can **ssh** into the instance without a password by using the keypair you specified in the **nova boot** command. Use the **nova list** command to get the IP address for the

instance. You do not need to specify the private key because it was stored in the default location, `~/.ssh/.id_rsa`, for the `ssh` client.



Note

If using a CirrOS image to spawn an instance you must log in as the `cirros`, and not the `root`, user.

You can also log in to the `cirros` account without an ssh key by using the `cubswin:) password:`

```
$ ssh cirros@10.0.0.3
```

6. Add the dashboard

Table of Contents

System requirements	34
Install the dashboard	35
Set up session storage for the dashboard	36

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

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

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

After you [install and configure the dashboard](#), you can complete the following tasks:

- Customize your dashboard. See section [Customize the dashboard](#) in the *OpenStack Cloud Administrator Guide*.
- Set up session storage for the dashboard. See [the section called "Set up session storage for the dashboard" \[36\]](#).

System requirements

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

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

Note the URLs of the Identity Service and Compute endpoints.

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

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

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

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

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



Note

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

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

Install the dashboard

Before you can install and configure the dashboard, meet the requirements in [the section called "System requirements" \[34\]](#).



Note

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

For more information about how to deploy the dashboard, see [deployment topics in the developer documentation](#).

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

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



Note for Ubuntu users

Remove the `openstack-dashboard-ubuntu-theme` package. This theme prevents translations, several menus as well as the network map from rendering correctly:

```
# apt-get remove --purge openstack-dashboard-ubuntu-theme
```

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

Open `/etc/openstack-dashboard/local_settings.py` and look for this line:

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



Notes

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

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

- You can use options other than memcached option for session storage. Set the session back-end through the `SESSION_ENGINE` option.
- To change the timezone, use the dashboard or edit the `/etc/openstack-dashboard/local_settings.py` file.

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

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

Edit `/etc/openstack-dashboard/local_settings.py`:

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

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

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

```
OPENSTACK_HOST = "controller"
```

5. Start the Apache web server and memcached:

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

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

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

Set up session storage for the dashboard

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

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

Local memory cache

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

- No shared storage across processes or workers.

- No persistence after a process terminates.

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

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

Key-value stores

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

Memcached

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

Requirements:

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

Enabled by:

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

Redis

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

Requirements:

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

Enabled by:

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

Initialize and configure the database

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

However, database-backed sessions are one of the slower session storages and incur a high overhead under heavy usage. Proper configuration of your database deployment can also be a substantial undertaking and is far beyond the scope of this documentation.

1. Start the `mysql` command line client:

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

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

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

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

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

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

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

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

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

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

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

As a result, the following output is returned:

```
Installing custom SQL ...  
Installing indexes ...  
DEBUG:django.db.backends:(0.008) CREATE INDEX `django_session_c25c2c28` ON  
`django_session` (`expire_date`);; args=()  
No fixtures found.
```

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

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

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

On Ubuntu:

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

On Fedora/RHEL/CentOS:

```
# service httpd restart
```

```
# service apache2 restart
```

On openSUSE:

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

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

```
# sudo restart nova-api
```

Cached database

To mitigate the performance issues of database queries, you can use the Django `cached_db` session back end, which utilizes both your database and caching infrastructure to perform write-through caching and efficient retrieval.

Enable this hybrid setting by configuring both your database and cache, as discussed previously. Then, set the following value:

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

Cookies

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

This back end stores session data in a cookie, which is stored by the user's browser. The back end uses a cryptographic signing technique to ensure session data is not tampered with during transport. This is not the same as encryption; session data is still readable by an attacker.

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

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

See the Django [cookie-based sessions](#) documentation.

7. Add the Block Storage Service

Table of Contents

Block Storage Service	40
Configure a Block Storage Service controller	40
Configure a Block Storage Service node	42

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

Block Storage Service

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

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

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

Configure a Block Storage Service controller

To create the components that control the Block Storage Service, complete the following steps on the controller node.

You can configure OpenStack to use various storage systems. The examples in this guide show you how to configure LVM.

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

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

2. The Block Storage Service stores volume information in a database. The examples in this section use the MySQL database that is used by other OpenStack services.

Configure the Block Storage Service to use the database. Replace `CINDER_DBPASS` with a password of your choosing.

Edit `/etc/cinder/cinder.conf` and change the `[database]` section.

```
[database]
...
# The SQLAlchemy connection string used to connect to the
# database (string value)
connection = mysql://cinder:CINDER_DBPASS@localhost/cinder
...
```

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

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

4. Create the database tables for the Block Storage Service.

```
# cinder-manage db sync
```

5. Create a cinder user. The Block Storage Service uses this user to authenticate with the Identity Service. Use the `service` tenant and give the user the `admin` role.

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

6. Add the credentials to the file `/etc/cinder/api-paste.ini`. Open the file in a text editor and locate the section `[filter:authtoken]`. Set the following options:

```
[filter:authtoken]
paste.filter_factory=keystoneclient.middleware.auth_token:filter_factory
auth_host=controller
auth_port = 35357
auth_protocol = http
admin_tenant_name=service
admin_user=cinder
admin_password=CINDER_PASS
```

7. Configure the Block Storage Service to use the RabbitMQ message broker by setting the following configuration keys. They are found in the `DEFAULT` configuration group of the `/etc/cinder/cinder.conf` file.

```
rpc_backend = cinder.openstack.common.rpc.impl_kombu
rabbit_host = controller
rabbit_port = 5672
rabbit_userid = guest
rabbit_password = RABBIT_PASS
```

8. Register the Block Storage Service with the Identity Service so that other OpenStack services can locate it. Register the service and specify the endpoint using the `keystone` command.

```
# keystone service-create --name=cinder --type=volume \
--description="Cinder Volume Service"
```

Note the `id` property returned and use it to create the endpoint.

```
# keystone endpoint-create \  
--service-id=the_service_id_above \  
--publicurl=http://controller:8776/v1/%(tenant_id)s \  
--internalurl=http://controller:8776/v1/%(tenant_id)s \  
--adminurl=http://controller:8776/v1/%(tenant_id)s
```

9. Also register a service and endpoint for version 2 of the Block Storage Service API.

```
# keystone service-create --name=cinder --type=volumev2 \  
--description="Cinder Volume Service V2"
```

Note the `id` property returned and use it to create the endpoint.

```
# keystone endpoint-create \  
--service-id=the_service_id_above \  
--publicurl=http://controller:8776/v2/%(tenant_id)s \  
--internalurl=http://controller:8776/v2/%(tenant_id)s \  
--adminurl=http://controller:8776/v2/%(tenant_id)s
```

10. Restart the cinder service with its new settings:

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

Configure a Block Storage Service node

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

You can configure OpenStack to use various storage systems. The examples in this guide show you how to configure LVM.

1. Use the instructions in [Chapter 2, “Basic operating system configuration” \[5\]](#) to configure the system. Note the following differences from the installation instructions for the controller node:
 - Set the host name to `block1`. Ensure that the IP addresses and host names for both nodes are listed in the `/etc/hosts` file on each system.
 - Follow the instructions in [the section called “Network Time Protocol \(NTP\)” \[7\]](#) to synchronize from the controller node.
2. Create the LVM physical and logical volumes. This guide assumes a second disk `/dev/sdb` that is used for this purpose.

```
# pvcreate /dev/sdb  
# vgcreate cinder-volumes /dev/sdb
```
3. Add a filter entry to the devices section `/etc/lvm/lvm.conf` to keep LVM from scanning devices used by virtual machines.



Note

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

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

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

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

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

```
# apt-get install cinder-volume lvm2
```

5. Copy the `/etc/cinder/api-paste.ini` file from the controller, or open the file in a text editor and locate the section `[filter:authtoken]`. Make sure the following options are set:

```
[filter:authtoken]  
paste.filter_factory=keystoneclient.middleware.auth_token:filter_factory  
auth_host=controller  
auth_port = 35357  
auth_protocol = http  
admin_tenant_name=service  
admin_user=cinder  
admin_password=CINDER_PASS
```

6. Configure the Block Storage Service to use the RabbitMQ message broker by setting the following configuration keys. They are found in the `DEFAULT` configuration group of the `/etc/cinder/cinder.conf` file.

```
rpc_backend = cinder.openstack.common.rpc.impl_kombu  
rabbit_host = controller  
rabbit_port = 5672  
rabbit_userid = guest  
rabbit_password = RABBIT_PASS
```

7. Configure the Block Storage Service on this node to use the cinder database on the controller node.

Edit `/etc/cinder/cinder.conf` and change the `[database]` section.

```
[database]  
...  
# The SQLAlchemy connection string used to connect to the  
# database (string value)  
connection = mysql://cinder:CINDER_DBPASS@controller/cinder  
...
```

8. Restart the cinder service with its new settings.

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

```
# service tgt restart
```

8. Add Object Storage

Table of Contents

Object Storage service	45
System requirements	45
Plan networking for Object Storage	46
Example Object Storage installation architecture	47
Install Object Storage	48
Install and configure storage nodes	49
Install and configure the proxy node	51
Start services on the storage nodes	54
Object Storage post-installation tasks	54

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

Object Storage service

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

It includes the following components:

- Proxy servers (`swift-proxy-server`). Accepts Object Storage API and raw HTTP requests to upload files, modify metadata, and create containers. It also serves file or container listings to web browsers. To improve performance, the proxy server can use an optional cache usually deployed with memcache.
- Account servers (`swift-account-server`). Manage accounts defined with the Object Storage service.
- Container servers (`swift-container-server`). Manage a mapping of containers, or folders, within the Object Storage service.
- Object servers (`swift-object-server`). Manage actual objects, such as files, on the storage nodes.
- A number of periodic processes. Performs housekeeping tasks on the large data store. The replication services ensure consistency and availability through the cluster. Other periodic processes include auditors, updaters, and reapers.

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

System requirements

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

**Note**

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

Table 8.1. Hardware recommendations

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

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

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

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

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

Plan networking for Object Storage

For both conserving network resources and ensuring that network administrators understand the needs for networks and public IP addresses for providing access to the APIs and storage network as necessary, this section offers recommendations and required minimum sizes. Throughput of at least 1000 Mbps is suggested.

This guide describes the following networks:

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

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

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

Public Network (Publicly
routable IP range)

Provides public IP accessibility to the API endpoints within the cloud infrastructure.

Minimum size: one IP address for each proxy server.

Storage Network (RFC1918 IP
Range, not publicly routable)

Manages all inter-server communications within the Object Storage infrastructure.

Minimum size: one IP address for each storage node and proxy server.

Recommended size: as above, with room for expansion to the largest your cluster size. For example, 255 or CIDR /24.

Replication Network (RFC1918 IP
Range, not publicly routable)

Manages replication-related communications among storage servers within the Object Storage infrastructure.

Recommended size: as for `STORAGE_LOCAL_NET`.

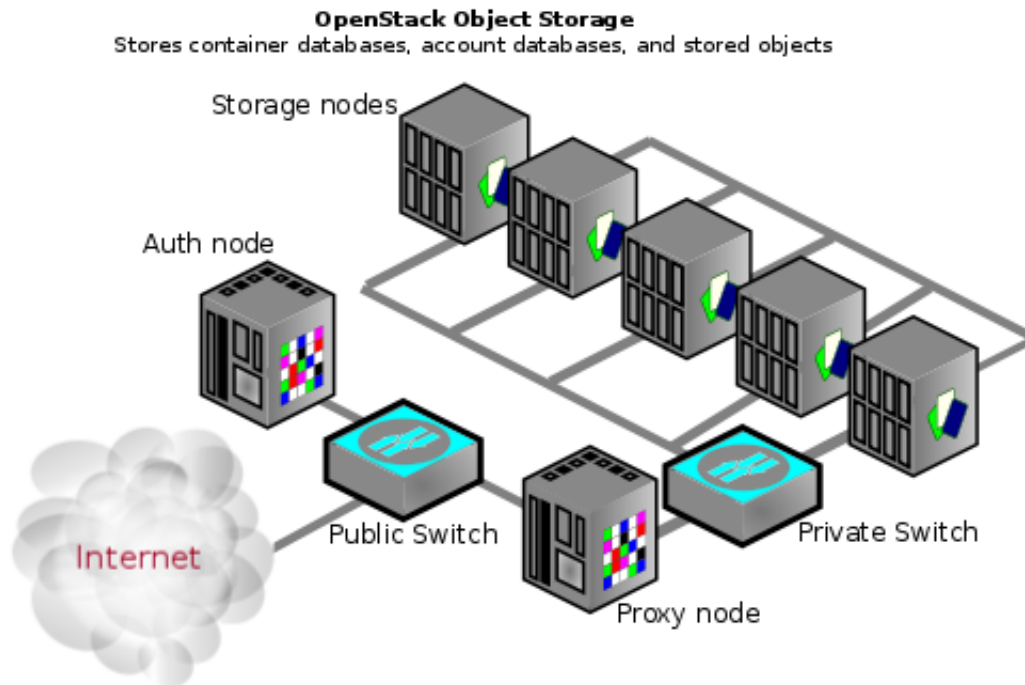
Example Object Storage installation architecture

- node. A host machine that runs one or more OpenStack Object Storage services.
- Proxy node. Runs Proxy services.
- Storage node. Runs Account, Container, and Object services.
- Ring. A set of mappings between OpenStack Object Storage data to physical devices.
- Replica. A copy of an object. By default, three copies are maintained in the cluster.
- Zone. A logically separate section of the cluster, related to independent failure characteristics.

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

This document describes each storage node as a separate zone in the ring. At a minimum, five zones are recommended. A zone is a group of nodes that is as isolated as possible from other nodes (separate servers, network, power, even geography). The ring guarantees that

every replica is stored in a separate zone. This diagram shows one possible configuration for a minimal installation:



Install Object Storage

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

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

Before you begin

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

These steps assume you have set up repositories for packages for your operating system as shown in [OpenStack Packages](#).

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

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



Note

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

General installation steps

1. Install core Swift files and openSSH.

```
# apt-get install swift openssh-server rsync memcached python-netifaces \
python-xattr python-memcache
```

2. Create and populate configuration directories on all nodes:

```
# mkdir -p /etc/swift
# chown -R swift:swift /etc/swift/
```

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

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



Note

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

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

Install and configure storage nodes



Note

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

1. Install Storage node packages:

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

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

```
# fdisk /dev/sdb
```

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

3. Create /etc/rsyncd.conf:

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

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

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

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

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

```
address = <STORAGE_REPLICATION_NET_IP>
```

5. Edit the following line in /etc/default/rsync:

```
RSYNC_ENABLE = true
```

6. Start rsync daemon:

```
# service rsync start
```



Note

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

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

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

Install and configure the proxy node

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



Note

The Object Storage processes run under a separate user and group, set by configuration options, and referred to as `swift:swift`. The default user is `swift`, which may not exist on your system.

1. Install `swift-proxy` service:

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

2. Create self-signed cert for SSL:

```
# cd /etc/swift
# openssl req -new -x509 -nodes -out cert.crt -keyout cert.key
```

3. Modify `memcached` to listen on the default interfaces on a local, non-public network. Edit this line in the `/etc/memcached.conf` file:

```
-l 127.0.0.1
```

Change it to:

```
-l <PROXY_LOCAL_NET_IP>
```

4. Restart the `memcached` server:

```
# service memcached restart
```

5. Ubuntu only: Because the distribution packages do not include a copy of the `keystoneauth` middleware, ensure that the proxy server includes them:

```
$ git clone https://github.com/openstack/swift.git
$ cd swift
$ python setup.py install
$ swift-init proxy start
```

6. Create `/etc/swift/proxy-server.conf`:

```
[DEFAULT]
bind_port = 8888
user = swift

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

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

[filter:keystoneauth]
```

```
use = egg:swift#keystoneauth
operator_roles = Member,admin,swiftoperator

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

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

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

# auth_* settings refer to the Keystone server
auth_protocol = http
auth_host = 192.168.56.3
auth_port = 35357

# the same admin_token as provided in keystone.conf
admin_token = 012345SECRET99TOKEN012345

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

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

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

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



Note

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

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

Only the proxy server uses memcache.

7. Create the `signing_dir` and set its permissions accordingly.

```
# mkdir -p /home/swift/keystone-signing
# chown -R swift:swift /home/swift/keystone-signing
```

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

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

```
# swift-ring-builder object.builder create 18 3 1
```

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

```
# swift-ring-builder account.builder add z<ZONE>-  
<STORAGE_LOCAL_NET_IP>:6002[R<STORAGE_REPLICATION_NET_IP>:6005]/<DEVICE>  
100  
# swift-ring-builder container.builder add z<ZONE>-  
<STORAGE_LOCAL_NET_IP_1>:6001[R<STORAGE_REPLICATION_NET_IP>:6004]/<DEVICE>  
100  
# swift-ring-builder object.builder add z<ZONE>-  
<STORAGE_LOCAL_NET_IP_1>:6000[R<STORAGE_REPLICATION_NET_IP>:6003]/<DEVICE>  
100
```



Note

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

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

```
# swift-ring-builder account.builder add z1-10.0.0.1:6002R10.0.1.1:6005/  
sdb1 100  
# swift-ring-builder container.builder add z1-10.0.0.1:6001R10.0.1.1:6005/  
sdb1 100  
# swift-ring-builder object.builder add z1-10.0.0.1:6000R10.0.1.1:6005/  
sdb1 100
```



Note

If you assume five zones with one node for each zone, start *ZONE* at 1. For each additional node, increment *ZONE* by 1.

10. Verify the ring contents for each ring:

```
# swift-ring-builder account.builder  
# swift-ring-builder container.builder  
# swift-ring-builder object.builder
```

11. Rebalance the rings:

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



Note

Rebalancing rings can take some time.

12. Copy the *account.ring.gz*, *container.ring.gz*, and *object.ring.gz* files to each of the Proxy and Storage nodes in */etc/swift*.

13. Make sure the swift user owns all configuration files:

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

14. Start Proxy services:

```
# service proxy-server start
```

Start services on the storage nodes

Now that the ring files are on each storage node, you can start the services. On each storage node, run the following commands:

```
# service swift-object start
# service swift-object-replicator start
# service swift-object-updater start
# service swift-object-auditor start
# service swift-container start
# service swift-container-replicator start
# service swift-container-updater start
# service swift-container-auditor start
# service swift-account start
# service swift-account-replicator start
# service swift-account-reaper start
# service swift-account-auditor start
```



Note

To start all swift services at once, run the command:

```
# swift-init main start
```

To know more about `swift-init` command, run:

```
# man swift-init
```

```
# service rsyslog restart
# service memcached restart
```

Object Storage post-installation tasks

Verify the installation

You can run these commands from the proxy server or any server that has access to the Identity Service.

1. Export the swift admin password, which you set up as an Identity Service admin and added to the `proxy-server.conf` file to a variable. You can also set up an `openrc` file as described in the [OpenStack User Guide](#) where the variable is `OS_USERNAME`.

```
$ export OS_PASSWORD=ADMIN_PASS
```

```
$ export OS_AUTH_URL=http://controller:5000/v2.0
```



Note

The sample `proxy-server.conf` file uses "swift" for `ADMIN_PASS`. If you do not wish to have the swift admin password stored in your shell's history, you can run the following command:

```
$ export SWIFT_PROXY_CONF=/etc/swift/proxy-server.conf export
OS_PASSWORD=$( grep admin_password ${SWIFT_PROXY_CONF} | awk
' { print $NF }' )
```

2. Run the following swift command with the correct Identity Service URL:

```
$ swift -V 2.0 -A $OS_AUTH_URL -U demo:admin -K $ADMINPASS stat
Account: AUTH_11b9758b7049476d9b48f7a91ea11493
Containers: 0
  Objects: 0
  Bytes: 0
Content-Type: text/plain; charset=utf-8
X-Timestamp: 1381434243.83760
X-Trans-Id: txdcdd594565214fb4a2d33-0052570383
X-Put-Timestamp: 1381434243.83760
```

3. Run the following swift commands to upload files to a container (create a test text files if needed):

```
$ swift -V 2.0 -A $OS_AUTH_URL -U demo:admin -K $OS_PASSWORD upload
myfiles test.txt
$ swift -V 2.0 -A $OS_AUTH_URL -U demo:admin -K $ADMINPASS upload myfiles
test2.txt
```

4. Run the following swift command to download all files from the 'myfiles' container:

```
$ swift -V 2.0 -A $OS_AUTH_URL -U demo:admin -K $ADMINPASS download
myfiles

test2.txt [headers 0.267s, total 0.267s, 0.000s MB/s]
test.txt [headers 0.271s, total 0.271s, 0.000s MB/s]
```

Add a proxy server

For reliability, you add proxy servers. You can set up an additional proxy node the same way that you set up the first proxy node but with additional configuration steps.

After you have more than two proxies, you must load balance them; your storage endpoint (what clients use to connect to your storage) also changes. You can select from different strategies for load balancing. For example, you could use round-robin DNS, or a software or hardware load balancer (like pound) in front of the two proxies, and point your storage URL to the load balancer.

Configure an initial proxy node. Then, complete these steps to add proxy servers.

1. Update the list of memcache servers in the `/etc/swift/proxy-server.conf` file for added proxy servers. If you run multiple memcache servers, use this pattern for the multiple IP:port listings in each proxy server configuration file:

```
10.1.2.3:11211,10.1.2.4:11211

[filter:cache]
use = egg:swift#memcache
memcache_servers = <PROXY_LOCAL_NET_IP>:11211
```

2. Copy ring information to all nodes, including new proxy nodes. Also, ensure that the ring information gets to all storage nodes.

3. After you sync all nodes, make sure that the admin has keys in `/etc/swift` and the ownership for the ring file is correct.

9. Install the Networking service

Table of Contents

Networking considerations	57
Neutron concepts	57
Install Networking services	59
Neutron deployment use cases	72



Warning

This chapter is a bit more adventurous than we would like. We are working on cleanup and improvements to it. Like for the rest of the Installation Guide, feedback through bug reports and patches to improve it are welcome.

Networking considerations

OpenStack Networking drivers range from software bridges to full control of certain switching hardware. This guide focuses on the Open vSwitch driver. However, the theories presented here are mostly applicable to other mechanisms, and the [OpenStack Configuration Reference](#) offers additional information.

To prepare for installation, see [the section called "OpenStack packages" \[8\]](#).



Warning

If you previously set up networking for your compute node by using `nova-network`, this configuration overrides those settings.

Neutron concepts

Like Nova Networking, Neutron manages software-defined networking for your OpenStack installation. However, unlike Nova Networking, you can configure Neutron for advanced virtual network topologies, such as per-tenant private networks and more.

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

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

In addition to external networks, any Neutron 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 an 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.

Neutron 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 Neutron applies the rules in those security groups to block or unblock ports, port ranges, or traffic types for that VM.

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

Open vSwitch concepts

The Open vSwitch plug-in is one of the most popular core plug-ins. Open vSwitch configurations consists of bridges and ports. Ports represent connections to other things, such as physical interfaces and patch cables. Packets from any given port on a bridge are shared with all other ports on that bridge. Bridges can be connected through Open vSwitch virtual patch cables or through Linux virtual Ethernet cables (*veth*). Additionally, bridges appear as network interfaces to Linux, so you can assign IP addresses to them.

In Neutron, the integration bridge, called *br-int*, connects directly to the VMs and associated services. The external bridge, called *br-ex*, connects to the external network. Finally, the VLAN configuration of the Open vSwitch plug-in uses bridges associated with each physical network.

In addition to defining bridges, Open vSwitch has OpenFlow, which enables you to define networking flow rules. Certain configurations use these rules to transfer packets between VLANs.

Finally, some configurations of Open vSwitch use network namespaces that enable Linux to group adapters into unique namespaces that are not visible to other namespaces, which allows the same network node to manage multiple Neutron routers.

With Open vSwitch, you can use two different technologies to create the virtual networks: GRE or VLANs.

Generic Routing Encapsulation (GRE) is the technology used in many VPNs. It wraps IP packets to create entirely new packets with different routing information. When the new packet reaches its destination, it is unwrapped, and the underlying packet is routed. To use GRE with Open vSwitch, Neutron creates GRE tunnels. These tunnels are ports on a bridge

and enable bridges on different systems to act as though they were one bridge, which allows the compute and network nodes to act as one for the purposes of routing.

Virtual LANs (VLANs), on the other hand, use a special modification to the Ethernet header. They add a 4-byte VLAN tag that ranges from 1 to 4094 (the 0 tag is special, and the 4095 tag, made of all ones, is equivalent to an untagged packet). Special NICs, switches, and routers know how to interpret the VLAN tags, as does Open vSwitch. Packets tagged for one VLAN are only shared with other devices configured to be on that VLAN, even though all devices are on the same physical network.

The most common security group driver used with Open vSwitch is the Hybrid IPTables/Open vSwitch plug-in. It uses a combination for IPTables and OpenFlow rules. Use the IPTables tool to create firewalls and set up NATs on Linux. This tool uses a complex rule system and chains of rules to accommodate the complex rules required by Neutron security groups.

Install Networking services

Before you configure individual nodes for Networking, you must create the required OpenStack components: user, service, database, and one or more endpoints. After you complete these steps on the controller node, follow the instructions in this guide to set up OpenStack Networking nodes.

1. Use the password that you set previously to log in as root and create a `neutron` database:

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

2. Create the required user, service, and endpoint so that Networking can interface with the Identity Service.

To list the tenant IDs:

```
# keystone tenant-list
```

To list role IDs:

```
# keystone role-list
```

Create a `neutron` user:

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

Add the user role to the `neutron` user:

```
# keystone user-role-add --user=neutron --tenant=service --role=admin
```

Create the `neutron` service:

```
# keystone service-create --name=neutron --type=network \
```

```
--description="OpenStack Networking Service"
```

Create a Networking endpoint. Use the `id` property for the service that was returned in the previous step to create the endpoint:

```
# keystone endpoint-create \  
  --service-id the_service_id_above \  
  --publicurl http://controller:9696 \  
  --adminurl http://controller:9696 \  
  --internalurl http://controller:9696
```

Install Networking services on a dedicated network node



Note

Before you start, set up a machine as a dedicated network node. Dedicated network nodes have a `MGMT_INTERFACE` NIC, a `DATA_INTERFACE` NIC, and a `EXTERNAL_INTERFACE` NIC.

The management network handles communication among nodes. The data network handles communication coming to and from VMs. The external NIC connects the network node, and optionally to the controller node, so your VMs can connect to the outside world.

All NICs must have static IPs. However, the data and external NICs have a special set up. For details about Networking plug-ins, see [the section called "Install and configure the Networking plug-ins" \[62\]](#).

1. Install the OpenStack Networking service on the network node:

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

2. Enable packet forwarding and disable packet destination filtering so that the network node can coordinate traffic for the VMs. Edit the `/etc/sysctl.conf` file, as follows:

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



Note

With system network-related configurations, you might need to restart the network service to activate configurations, as follows:

```
# service networking restart
```

3. Configure the core networking components. Edit the `/etc/neutron/neutron.conf` file and add these lines to the `keystone_authtoken` section:

```
[keystone_authtoken]  
auth_host = controller  
auth_port = 35357  
auth_protocol = http  
admin_tenant_name = service  
admin_user = neutron  
admin_password = NEUTRON_PASS
```

To activate changes in the `/etc/sysctl.conf` file, run the following command:

```
# sysctl -p
```

4. Configure the RabbitMQ access. Edit the `/etc/neutron/neutron.conf` file to modify the following parameters in the `DEFAULT` section.

```
rabbit_host = controller
rabbit_userid = guest
rabbit_password = RABBIT_PASS
```

5. Configure Networking to connect to the database. Edit the `[database]` section in the same file, as follows:

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

6. Edit the `/etc/neutron/api-paste.ini` file and add these lines to the `[filter:authtoken]` section:

```
[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
auth_host=controller
auth_uri=http://controller:5000
admin_user=neutron
admin_tenant_name=service
admin_password=NEUTRON_PASS
```



Warning

`keystoneclient.middleware.auth_token`: You must configure `auth_uri` to point to the public identity endpoint. Otherwise, clients might not be able to authenticate against an admin endpoint.

7. Install and configure a networking plug-in. OpenStack Networking uses this plug-in to perform software-defined networking. For instructions, see [instructions](#). Then, return here.

Now that you've installed and configured a plug-in (you did do that, right?), it is time to configure the remaining parts of Networking.

1. To perform DHCP on the software-defined networks, Networking supports several different plug-ins. However, in general, you use the `Dnsmasq` plug-in. Edit the `/etc/neutron/dhcp_agent.ini` file:

```
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
```

2. To allow virtual machines to access the Compute metadata information, the Networking metadata agent must be enabled and configured. The agent will act as a proxy for the Compute metadata service.

On the controller, edit the `/etc/nova/nova.conf` file to define a secret key that will be shared between the Compute Service and the Networking metadata agent.

Add to the `[DEFAULT]` section:

```
[DEFAULT]
neutron_metadata_proxy_shared_secret = METADATA_PASS
service_neutron_metadata_proxy = true
```

Restart the nova-api service:

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

On the network node, modify the metadata agent configuration.

Edit the `/etc/neutron/metadata_agent.ini` file and modify the `[DEFAULT]` section:

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

3. Restart Networking:

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

4. After you configure the [compute](#) and [controller](#) nodes, [configure the base networks](#).

Install and configure the Networking plug-ins

Install the Open vSwitch (OVS) plug-in

1. Install the Open vSwitch plug-in and its dependencies:

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



Note

On Ubuntu 12.04 LTS with GRE you must install `openvswitch-datapath-dkms` and restart the service to enable the GRE flow so that OVS 1.10 and higher is used. Make sure you are running the OVS 1.10 kernel module in addition to the OVS 1.10 userspace. Both the kernel module and userspace are required for VXLAN support. The error you see in the `/var/log/openvswitch/ovs-vswitchd.log` log file is "Stderr: 'ovs-ofctl: -1: negative values not supported for in_port\n". If you see this error, make sure `modinfo openvswitch` shows the right version. Also check the output from `dmesg` for the version of the OVS module being loaded.

2. Start Open vSwitch:

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

3. No matter which networking technology you use, you must add the `br-int` integration bridge, which connects to the VMs, and the `br-ex` external bridge, which connects to the outside world.

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

4. Add a *port* (connection) from the `EXTERNAL_INTERFACE` interface to `br-ex` interface:

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

5. Configure the `EXTERNAL_INTERFACE` without an IP address and in promiscuous mode. Additionally, you must set the newly created `br-ex` interface to have the IP address that formerly belonged to `EXTERNAL_INTERFACE`.



Warning

Generic Receive Offload (GRO) should not be enabled on this interface as it can cause severe performance problems. It can be disabled with the `ethtool` utility.

6. You must set some common configuration options no matter which networking technology you choose to use with Open vSwitch. Configure the L3 and DHCP agents to use OVS and namespaces. Edit the `/etc/neutron/l3_agent.ini` and `/etc/neutron/dhcp_agent.ini` files, respectively:

```
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
use_namespaces = True
```

7. Similarly, you must also tell Neutron core to use OVS. Edit the `/etc/neutron/neutron.conf` file:

```
core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.
OVSNeutronPluginV2
```

8. Choose a networking technology to create the virtual networks. Neutron supports GRE tunneling, VLANs, and VXLANs. This guide shows how to configure GRE tunneling and VLANs.

[GRE tunneling](#) is simpler to set up because it does not require any special configuration from any physical network hardware. However, its protocol makes it difficult to filter traffic on the physical network. Additionally, this configuration does not use namespaces. You can have only one router for each network node. However, you can enable namespacing, and potentially veth, as described in the section detailing how to use VLANs with OVS).

On the other hand, [VLAN tagging](#) modifies the ethernet header of packets. You can filter packets on the physical network through normal methods. However, not all NICs handle the increased packet size of VLAN-tagged packets well, and you might need to complete additional configuration on physical network hardware to ensure that your Neutron VLANs do not interfere with any other VLANs on your network and that any physical network hardware between nodes does not strip VLAN tags.



Note

While the examples in this guide enable network namespaces by default, you can disable them if issues occur or your kernel does not support

them. Edit the `/etc/neutron/l3_agent.ini` and `/etc/neutron/dhcp_agent.ini` files, respectively:

```
use_namespaces = False
```

Edit the `/etc/neutron/neutron.conf` file to disable overlapping IP addresses:

```
allow_overlapping_ips = False
```

Note that when network namespaces are disabled, you can have only one router for each network node and overlapping IP addresses are not supported.

You must complete additional steps after you create the initial Neutron virtual networks and router.

9. Configure a firewall plug-in. If you do not wish to enforce firewall rules, called *security groups* by OpenStack, you can use `neutron.agent.firewall.NoopFirewall`. Otherwise, you can choose one of the Networking firewall plug-ins. The most common choice is the Hybrid OVS-IPTables driver, but you can also use the Firewall-as-a-Service driver. Edit the `/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini` file:

```
[securitygroup]
# Firewall driver for realizing neutron security group function.
firewall_driver = neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver
```



Warning

You must use at least the No-Op firewall. Otherwise, Horizon and other OpenStack services cannot get and set required VM boot options.

10. Restart the OVS plug-in and make sure it starts on boot:

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

11. Now, return to the general OVS instructions.

Configure the Neutron OVS plug-in for GRE tunneling

1. Configure the OVS plug-in to use GRE tunneling, the `br-int` integration bridge, the `br-tun` tunneling bridge, and a local IP for the `DATA_INTERFACE` tunnel IP. Edit the `/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini` file:

```
[ovs]
tenant_network_type = gre
tunnel_id_ranges = 1:1000
enable_tunneling = True
integration_bridge = br-int
tunnel_bridge = br-tun
local_ip = DATA_INTERFACE_IP
```

2. Return to the general OVS instructions.

Configure the Neutron OVS plug-in for VLANs

1. Configure OVS to use VLANs. Edit the `/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini` file:

```
[ovs]
tenant_network_type = vlan
network_vlan_ranges = physnet1:1:4094
bridge_mappings = physnet1:br-DATA_INTERFACE
```

2. Create the bridge for `DATA_INTERFACE` and add `DATA_INTERFACE` to it:

```
# ovs-vsctl add-br br-DATA_INTERFACE
# ovs-vsctl add-port br-DATA_INTERFACE DATA_INTERFACE
```

3. Transfer the IP address for `DATA_INTERFACE` to the bridge in the same way that you transferred the `EXTERNAL_INTERFACE` IP address to `br-ex`. However, do not turn on promiscuous mode.
4. Return to the OVS general instruction.

Install networking support on a dedicated compute node



Note

This section details set up for any node that runs the `nova-compute` component but does not run the full network stack.

1. Disable packet destination filtering (route verification) to let the networking services route traffic to the VMs. Edit the `/etc/sysctl.conf` file and run the following command to activate changes:

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

```
# sysctl -p
```

2. Install and configure your networking plug-in components. To install and configure the network plug-in that you chose when you set up your network node, see [the section called "Install and configure Neutron plug-ins on a dedicated compute node" \[66\]](#).
3. Configure the core components of Neutron. Edit the `/etc/neutron/neutron.conf` file:

```
auth_host = controller
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
auth_url = http://controller:35357/v2.0
auth_strategy = keystone
rpc_backend = neutron.openstack.common.rpc.impl_kombu
rabbit_host = controller
rabbit_port = 5672
# Change the following settings if you're not using the default RabbitMQ
configuration
#rabbit_userid = guest
rabbit_password = RABBIT_PASS
```

4. Edit the database URL under the `[database]` section in the above file, to tell Neutron how to connect to the database:

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

5. Edit the `/etc/neutron/api-paste.ini` file and add these lines to the `[filter:authtoken]` section:

```
[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
auth_host=controller
admin_user=neutron
admin_tenant_name=service
admin_password=NEUTRON_PASS
```

6. You must [configure the networking plug-in](#).

Install and configure Neutron plug-ins on a dedicated compute node

Install the Open vSwitch (OVS) plug-in on a dedicated compute node

1. Install the Open vSwitch plug-in and its dependencies:

```
# apt-get install neutron-plugin-openvswitch-agent openvswitch-switch
openvswitch-datapath-dkms
```

2. Restart Open vSwitch:

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

3. You must set some common configuration options no matter which networking technology you choose to use with Open vSwitch. You must add the `br-int` integration bridge, which connects to the VMs.

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

4. You must set some common configuration options. You must configure Networking core to use OVS. Edit the `/etc/neutron/neutron.conf` file:

```
core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.
OVSNeutronPluginV2
```

5. Configure the networking type that you chose when you set up the network node: either [GRE tunneling](#) or [VLANs](#).

6. You must configure a firewall as well. You should use the same firewall plug-in that you chose to use when you set up the network node. To do this, edit `/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini` file and set the `firewall_driver` value under the `securitygroup` to the same value used on the network node. For instance, if you chose to use the Hybrid OVS-IPTables plug-in, your configuration looks like this:

```
[securitygroup]
# Firewall driver for realizing neutron security group function.
firewall_driver = neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver
```



Warning

You must use at least the No-Op firewall. Otherwise, Horizon and other OpenStack services cannot get and set required VM boot options.

7. After you complete OVS configuration *and the core Neutron configuration after this section*, restart the Neutron Open vSwitch agent:

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

8. Now, return to the general OVS instructions.

Configure the Neutron OVS plug-in for GRE tunneling on a dedicated compute node

1. Tell the OVS plug-in to use GRE tunneling with a `br-int` integration bridge, a `br-tun` tunneling bridge, and a local IP for the tunnel of `DATA_INTERFACE`'s IP Edit the `/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini` file:

```
[ovs]
tenant_network_type = gre
tunnel_id_ranges = 1:1000
enable_tunneling = True
integration_bridge = br-int
tunnel_bridge = br-tun
local_ip = DATA_INTERFACE_IP
```

2. Now, return to the general OVS instructions.

Configure the Neutron OVS plug-in for VLANs on a dedicated compute node

1. Tell OVS to use VLANs. Edit the `/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini` file:

```
[ovs]
tenant_network_type = vlan
network_vlan_ranges = physnet1:1:4094
bridge_mappings = physnet1:br-DATA_INTERFACE
```

2. Create the bridge for the `DATA_INTERFACE` and add `DATA_INTERFACE` to it, the same way you did on the network node:

```
# ovs-vsctl add-br br-DATA_INTERFACE
# ovs-vsctl add-port br-DATA_INTERFACE DATA_INTERFACE
```

3. Return to the general OVS instructions.

Install networking support on a dedicated controller node



Note

This is for a node which runs the control components of Neutron, but does not run any of the components that provide the underlying functionality (such as the plug-in agent or the L3 agent). If you wish to have a combined controller/compute node follow these instructions, and then those for the compute node.

1. Install the main Neutron server, Neutron libraries for Python, and the Neutron command-line interface (CLI):
2. Configure the core components of Neutron. Edit the `/etc/neutron/neutron.conf` file:

```
auth_host = controller
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
auth_url = http://controller:35357/v2.0
auth_strategy = keystone
rpc_backend = YOUR_RPC_BACKEND
PUT_YOUR_RPC_BACKEND_SETTINGS_HERE_TOO
```

3. Edit the database URL under the `[database]` section in the above file, to tell Neutron how to connect to the database:

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

4. Configure the Neutron copy of the `api-paste.ini` at `/etc/neutron/api-paste.ini` file:

```
[filter:authtoken]
EXISTING_STUFF_HERE
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
```

5. Configure the plug-in you chose when you set up the network node. Follow the [instructions](#) and return here.
6. Tell Nova about Neutron. Specifically, you must tell Nova that Neutron handles networking and the firewall. Edit the `/etc/nova/nova.conf` file:

```
network_api_class=nova.network.neutronv2.api.API
neutron_url=http://controller:9696
neutron_auth_strategy=keystone
neutron_admin_tenant_name=service
neutron_admin_username=neutron
neutron_admin_password=NEUTRON_PASS
neutron_admin_auth_url=http://controller:35357/v2.0
linuxnet_interface_driver = nova.network.linux_net.LinuxOVSIInterfaceDriver
firewall_driver=nova.virt.firewall.NoopFirewallDriver
security_group_api=neutron
```



Note

Regardless of which firewall driver you chose when you configured the network and compute nodes, set this driver as the No-Op firewall. This firewall is a *Nova* firewall, and because Neutron handles the Firewall, you must tell Nova not to use one.

When Networking handles the firewall, the option `firewall_driver` should be set according to the specified plugin. For example with OVS, edit the `/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini` file:

```
[securitygroup]
# Firewall driver for realizing neutron security group function.
firewall_driver = neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver
```

If you do not want to use a firewall in Compute or Networking, set `firewall_driver=nova.virt.firewall.NoopFirewallDriver` in both config files, and comment out or remove `security_group_api=neutron` in the `/etc/nova/nova.conf` file, otherwise you may encounter `ERROR: The server has either erred or is incapable of performing the requested operation. (HTTP 500)` when issuing `nova list` commands.

7. Restart neutron-server:

```
# service neutron-server restart
```

Install and configure the Neutron plug-ins on a dedicated controller node

Install the Open vSwitch (OVS) plug-in on a dedicated controller node

1. Install the Open vSwitch plug-in:

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

2. You must set some common configuration options no matter which networking technology you choose to use with Open vSwitch. You must configure Networking core to use OVS. Edit the `/etc/neutron/neutron.conf` file:

```
core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.
OVSNeutronPluginV2
```

3. Configure the OVS plug-in for the networking type that you chose when you configured the network node: [GRE tunneling](#) or [VLANs](#).



Note

The dedicated controller node does not need to run Open vSwitch or the Open vSwitch agent.

4. Now, return to the general OVS instructions.

Configure the Neutron OVS plug-in for GRE tunneling on a dedicated controller node

1. Tell the OVS plug-in to use GRE tunneling. Edit the `/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini` file:

```
[ovs]
tenant_network_type = gre
tunnel_id_ranges = 1:1000
enable_tunneling = True
```

2. Return to the general OVS instructions.

Configure the Neutron OVS plug-in for VLANs on a dedicated controller node

1. Tell OVS to use VLANs. Edit the `/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini` file, as follows:

```
[ovs]
tenant_network_type = vlan
network_vlan_ranges = physnet1:1:4094
```

2. Return to the general OVS instructions.

Create the base Neutron networks



Note

In these sections, replace *SPECIAL_OPTIONS* with any options specific to your Networking plug-in choices. See [here](#) to check if your plug-in requires any special options.

1. Create the `ext-net` external network. This network represents a slice of the outside world. VMs are not directly linked to this network; instead, they connect to internal networks. Outgoing traffic is routed by Neutron to the external network. Additionally, floating IP addresses from the subnet for `ext-net` might be assigned to VMs so that the external network can contact them. Neutron routes the traffic appropriately.

```
# neutron net-create ext-net -- --router:external=True SPECIAL_OPTIONS
```

2. Create the associated subnet with the same gateway and CIDR as *EXTERNAL_INTERFACE*. It does not have DHCP because it represents a slice of the external world:

```
# neutron subnet-create ext-net \
  --allocation-pool start=FLOATING_IP_START,end=FLOATING_IP_END \
  --gateway=EXTERNAL_INTERFACE_GATEWAY --enable_dhcp=False \
  EXTERNAL_INTERFACE_CIDR
```

3. Create one or more initial tenants. The following steps use the *DEMO_TENANT* tenant.

Create the router attached to the external network. This router routes traffic to the internal subnets as appropriate. You can create it under the a given tenant: Append `--tenant-id` option with a value of *DEMO_TENANT_ID* to the command.

```
# neutron router-create ext-to-int
```

4. Connect the router to `ext-net` by setting the gateway for the router as `ext-net`:

```
# neutron router-gateway-set EXT_TO_INT_ID EXT_NET_ID
```

5. Create an internal network for *DEMO_TENANT* (and associated subnet over an arbitrary internal IP range, such as, `10.5.5.0/24`), and connect it to the router by setting it as a port:

```
# neutron net-create --tenant-id DEMO_TENANT_ID demo-net SPECIAL_OPTIONS
# neutron subnet-create --tenant-id DEMO_TENANT_ID demo-net 10.5.5.0/24 --
  gateway 10.5.5.1
```

```
# neutron router-interface-add EXT_TO_INT_ID DEMO_NET_SUBNET_ID
```

6. Check the special options page for your plug-in for remaining steps. Now, return to the general OVS instructions.

Plug-in-specific Neutron network options

Open vSwitch Network configuration options

GRE tunneling network options



Note

While this guide currently enables network namespaces by default, you can disable them if you have issues or your kernel does not support them. If you disabled namespaces, you must perform some additional configuration for the L3 agent.

After you create all the networks, tell the L3 agent what the external network ID is, as well as the ID of the router associated with this machine (because you are not using namespaces, there can be only one router for each machine). To do this, edit the `/etc/neutron/l3_agent.ini` file:

```
gateway_external_network_id = EXT_NET_ID  
router_id = EXT_TO_INT_ID
```

Then, restart the L3 agent:

```
# service neutron-l3-agent restart
```

When creating networks, you should use the options:

```
--provider:network_type gre --provider:segmentation_id SEG_ID
```

`SEG_ID` should be 2 for the external network, and just any unique number inside the tunnel range specified before for any other network.



Note

These options are not needed beyond the first network, as Neutron automatically increments the segmentation id and copy the network type option for any additional networks.

Now, return to the general OVS instructions.

VLAN network options

When creating networks, use these options:

```
--provider:network_type vlan --provider:physical_network physnet1 --  
provider:segmentation_id SEG_ID
```

`SEG_ID` should be 2 for the external network, and just any unique number inside the vlan range specified above for any other network.



Note

These options are not needed beyond the first network, as Neutron automatically increments the segmentation ID and copies the network type and physical network options for any additional networks. They are only needed if you wish to modify those values in any way.



Warning

Some NICs have Linux drivers that do not handle VLANs properly. See the `ovs-vlan-bug-workaround` and `ovs-vlan-test` man pages for more information. Additionally, you might try turning off `rx-vlan-offload` and `tx-vlan-offload` by using `ethtool` on the `DATA_INTERFACE`. Another potential caveat to VLAN functionality is that VLAN tags add an additional 4 bytes to the packet size. If your NICs cannot handle large packets, make sure to set the MTU to a value that is 4 bytes less than the normal value on the `DATA_INTERFACE`.

If you run OpenStack inside a virtualized environment (for testing purposes), switching to the `virtio` NIC type (or a similar technology if you are not using KVM/QEMU to run your host VMs) might solve the issue.

Neutron deployment use cases

This section describes how to configure the Networking service and its components for some typical use cases.

Single flat network

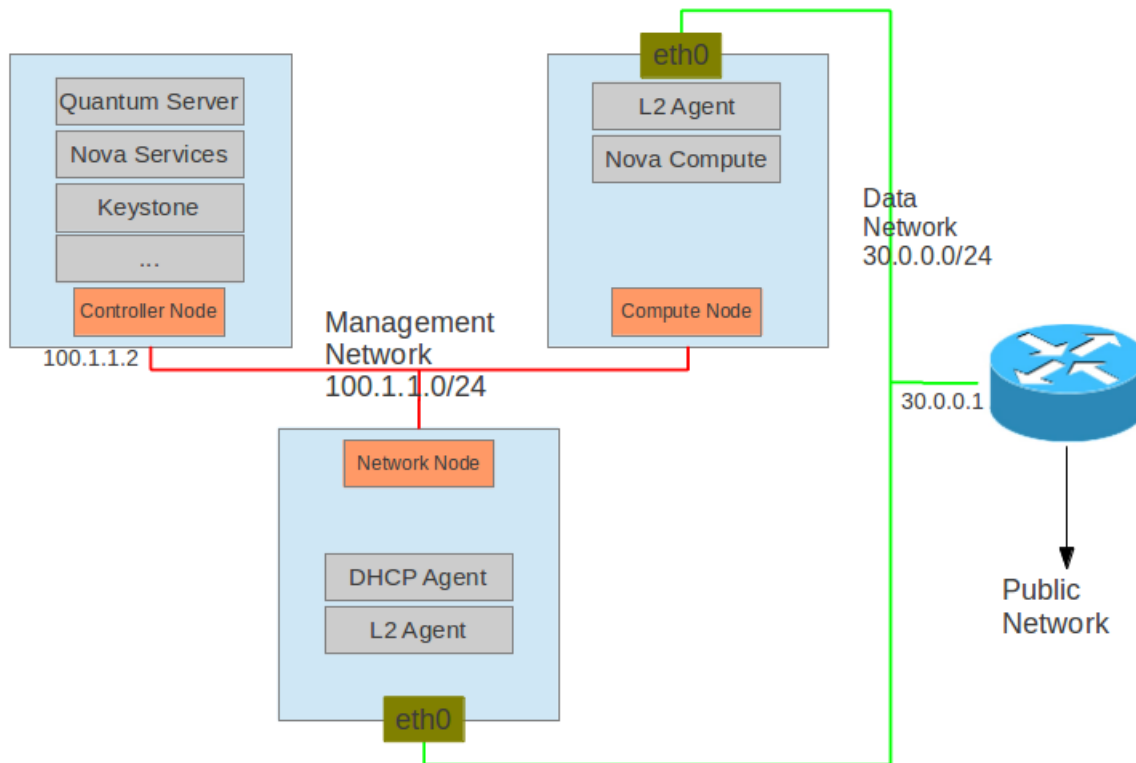
This section describes how to install the OpenStack Networking service and its components for a single flat network use case.

The following diagram shows the set up. For simplicity, all nodes should have one interface for management traffic and one or more interfaces for traffic to and from VMs. The management network is 100.1.1.0/24 with controller node at 100.1.1.2. The example uses the Open vSwitch plugin and agent.




Note

You can modify this set up to make use of another supported plug-in and its agent.



The following table describes some nodes in the set up:

Node	Description
Controller Node	<p>Runs the Networking service, Identity Service, and Compute services that are required to deploy VMs (<code>nova-api</code>, <code>nova-scheduler</code>, for example). The node must have at least one network interface, which is connected to the Management Network. The host name is <code>controller</code>, which every other node resolves to the IP of the controller node.</p> <p> Note</p> <p>The <code>nova-network</code> service should not be running. This is replaced by Networking. To delete a network, use <code>nova-manage network delete</code>:</p> <pre># nova-manage network delete --help Usage: nova-manage network delete <args> [options] Options: -h, --help show this help message and exit --fixed_range=<x.x.x.yy> Network to delete --uuid=<uuid> UUID of network to delete</pre> <p>Note that a network must first be disassociated from a project using the <code>nova network-disassociate</code> command before it can be deleted.</p>
Compute Node	<p>Runs the Networking L2 agent and the Compute services that run VMs (<code>nova-compute</code> specifically, and optionally other <code>nova-*</code> services depending on configuration). The node must have at least two network interfaces. The first communicates with the controller node through the management network. The second interface handles the VM traffic on the data network. The VM can receive its IP address from the DHCP agent on this network.</p>
Network Node	<p>Runs Networking L2 agent and the DHCP agent. The DHCP agent allocates IP addresses to the VMs on the network. The node must have at least two network interfaces. The first communicates with the controller node through the management network. The second interface handles the VM traffic on the data network.</p>

Node	Description
Router	Router has IP 30.0.0.1, which is the default gateway for all VMs. The router must be able to access public networks.

The demo assumes the following prerequisites:

Controller node

1. Relevant Compute services are installed, configured, and running.
2. Glance is installed, configured, and running. Additionally, an image must be available.
3. OpenStack Identity is installed, configured, and running. A Networking user **neutron** is in place on tenant **service** with password *NEUTRON_PASS*.
4. Additional services:
 - RabbitMQ is running with default guest and its password.
 - MySQL server (user is **root** and password is **root**).

Compute node

1. Compute is installed and configured.

Install

• Controller node - Networking server

1. Install the Networking server.
2. Create database **ovs_neutron**.
3. Update the Networking `/etc/neutron/neutron.conf` configuration file to choose a plug-in and Identity Service user as necessary:

```
[DEFAULT]
core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.
OVSNeutronPluginV2
control_exchange = neutron
rabbit_host = controller
rabbit_password = RABBIT_PASS
notification_driver = neutron.openstack.common.notifier.rabbit_notifier

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

[keystone_authtoken]
admin_tenant_name=service
admin_user=neutron
admin_password=NEUTRON_PASS
```

4. Update the plug-in `/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini` configuration file:

```
[ovs]
network_vlan_ranges = physnet1
bridge_mappings = physnet1:br-eth0
```

5. Start the Networking service

• Compute node - Compute

1. Install the nova-compute service.
2. Update the Compute `/etc/nova/nova.conf` configuration file. Make sure the following line is at the end of the file:

```
network_api_class=nova.network.neutronv2.api.API

neutron_admin_username=neutron
neutron_admin_password=NEUTRON_PASS
neutron_admin_auth_url=http://controller:35357/v2.0/
neutron_auth_strategy=keystone
neutron_admin_tenant_name=service
neutron_url=http://controller:9696/

libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtHybridOVSBridgeDriver
```

3. Restart the Compute services

• Compute and Network node#L2 agent

1. Install and start Open vSwitch.
2. Install the L2 agent (Neutron Open vSwitch agent).
3. Add the integration bridge to Open vSwitch:

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

4. Update the Networking `/etc/neutron/neutron.conf` configuration file:

```
[DEFAULT]
core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.
OVSNeutronPluginV2
control_exchange = neutron
rabbit_host = controller
rabbit_password = RABBIT_PASS
notification_driver = neutron.openstack.common.notifier.rabbit_notifier

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

5. Update the plug-in `/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini` configuration file:

```
[ovs]
network_vlan_ranges = physnet1
bridge_mappings = physnet1:br-eth0
```

6. Create the **br-eth0** network bridge to handle communication between nodes using `eth0`:

```
# ovs-vsctl add-br br-eth0
# ovs-vsctl add-port br-eth0 eth0
```

7. Start the OpenStack Networking L2 agent.

- **Network node#DHCP agent**

1. Install the DHCP agent.

2. Update the Networking `/etc/neutron/neutron.conf` configuration file:

```
[DEFAULT]
core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.
OVSNeutronPluginV2
control_exchange = neutron
rabbit_host = controller
rabbit_password = RABBIT_PASS
notification_driver = neutron.openstack.common.notifier.rabbit_notifier
```

3. Update the DHCP `/etc/neutron/dhcp_agent.ini` configuration file:

```
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
```

4. Start the DHCP agent.

Configure logical network

Use the following commands on the network node.



Note

Ensure that the following environment variables are set. Various clients use these variables to access the Identity Service.

```
export OS_USERNAME=admin
export OS_PASSWORD=adminpassword
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://127.0.0.1:5000/v2.0/
```

1. Get the tenant ID (Used as `$TENANT_ID` later):

```
# keystone tenant-list
+-----+-----+-----+
|          id          | name | enabled |
+-----+-----+-----+
| 247e478c599f45b5bd297e8ddb9b6a | TenantA | True |
| 2b4fec24e62e4ff28a8445ad83150f9d | TenantC | True |
| 3719a4940bf24b5a8124b58c9b0a6ee6 | TenantB | True |
| 5fcfbc3283a142a5bb6978b549a511ac | demo | True |
| b7445f221cda4f4a8ac7db6b218b1339 | admin | True |
+-----+-----+-----+
```

2. Get the user information:

```
# keystone user-list
+-----+-----+-----+-----+
|          id          | name | enabled | email |
+-----+-----+-----+-----+
| 5a9149ed991744fa85f71e4aa92eb7ec | demo | True | |
| 5b419c74980d46a1ab184e7571a8154e | admin | True | admin@example.com |
+-----+-----+-----+-----+
```

8e37cb8193cb4873a35802d257348431	UserC	True	
c11f6b09ed3c45c09c21cbbc23e93066	UserB	True	
ca567c4f6c0942bdac0e011e97bddbe3	UserA	True	

3. Create a internal shared network on the demo tenant (\$TENANT_ID is b7445f221cda4f4a8ac7db6b218b1339):

```
$ neutron net-create --tenant-id $TENANT_ID sharednet1 --shared --
provider:network_type flat \
--provider:physical_network physnet1
Created a new network:
```

Field	Value
admin_state_up	True
id	04457b44-e22a-4a5c-be54-a53a9b2818e7
name	sharednet1
provider:network_type	flat
provider:physical_network	physnet1
provider:segmentation_id	
router:external	False
shared	True
status	ACTIVE
subnets	
tenant_id	b7445f221cda4f4a8ac7db6b218b1339

4. Create a subnet on the network:

```
# neutron subnet-create --tenant-id $TENANT_ID sharednet1 30.0.0.0/24
Created a new subnet:
```

Field	Value
allocation_pools	{"start": "30.0.0.2", "end": "30.0.0.254"}
cidr	30.0.0.0/24
dns_nameservers	
enable_dhcp	True
gateway_ip	30.0.0.1
host_routes	
id	b8e9a88e-ded0-4e57-9474-e25fa87c5937
ip_version	4
name	
network_id	04457b44-e22a-4a5c-be54-a53a9b2818e7
tenant_id	5fcfbc3283a142a5bb6978b549a511ac

5. Create a server for tenant A:

```
# nova --os-tenant-name TenantA --os-username UserA --os-password password \
--os-auth-url=http://localhost:5000/v2.0 boot --image tty --flavor 1 \
--nic net-id=04457b44-e22a-4a5c-be54-a53a9b2818e7 TenantA_VM1

# nova --os-tenant-name TenantA --os-username UserA --os-password password \
--os-auth-url=http://localhost:5000/v2.0 list
```

ID	Name	Status	Networks

```
+-----+-----+-----+
+-----+
| 09923b39-050d-4400-99c7-e4b021cdc7c4 | TenantA_VM1 | ACTIVE | sharednet1=
30.0.0.3 |
+-----+-----+-----+
+-----+
```

6. Ping the server of tenant A:

```
# ip addr flush eth0
# ip addr add 30.0.0.201/24 dev br-eth0
$ ping 30.0.0.3
```

7. Ping the public network within the server of tenant A:

```
# ping 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=1.74 ms
64 bytes from 192.168.1.1: icmp_req=2 ttl=64 time=1.50 ms
64 bytes from 192.168.1.1: icmp_req=3 ttl=64 time=1.23 ms
^C
--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.234/1.495/1.745/0.211 ms
```



Note

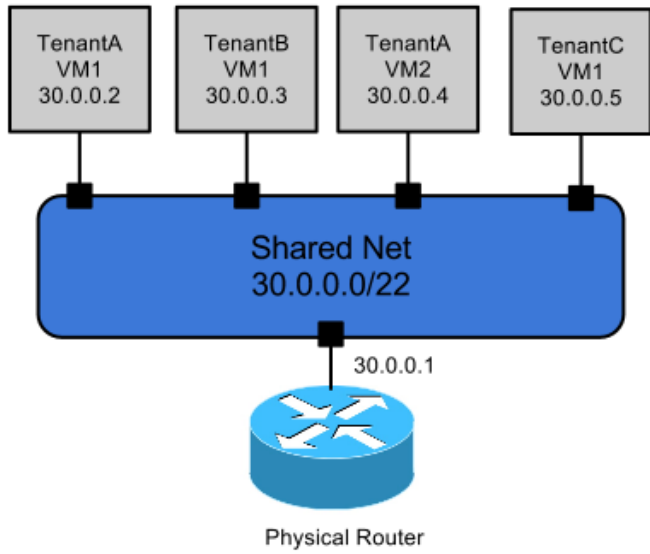
The 192.168.1.1 is an IP on public network to which the router connects.

8. Create servers for other tenants with similar commands. Because all VMs share the same subnet, they can access each other.

Use case: single flat network

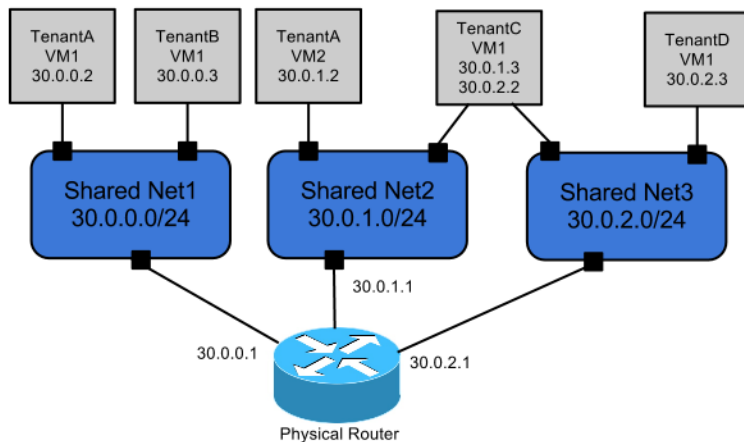
The simplest use case is a single network. This is a "shared" network, meaning it is visible to all tenants via the Networking API. Tenant VMs have a single NIC, and receive a fixed IP address from the subnet(s) associated with that network. This use case essentially maps to the FlatManager and FlatDHCPManager models provided by Compute. Floating IPs are not supported.

This network type is often created by the OpenStack administrator to map directly to an existing physical network in the data center (called a "provider network"). This allows the provider to use a physical router on that data center network as the gateway for VMs to reach the outside world. For each subnet on an external network, the gateway configuration on the physical router must be manually configured outside of OpenStack.



Use case: multiple flat network

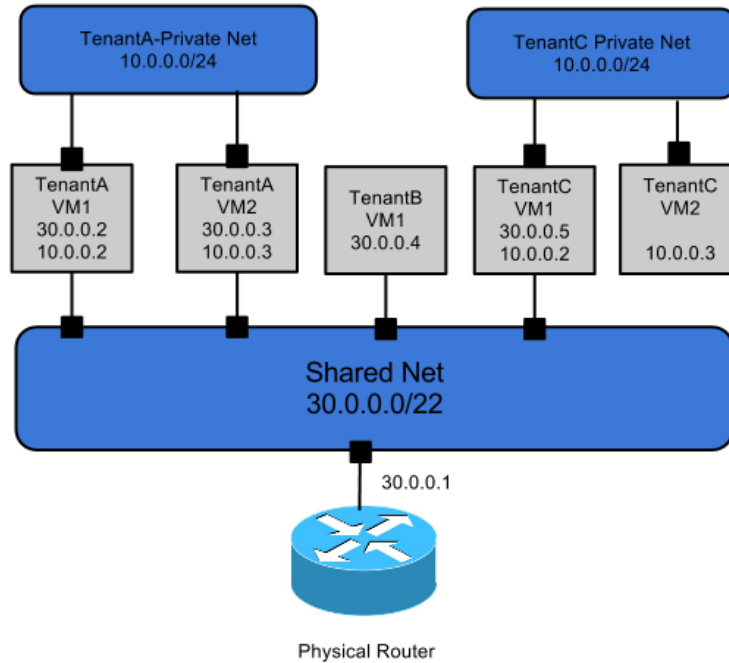
This use case is similar to the above single flat network use case, except that tenants can see multiple shared networks via the Networking API and can choose which network (or networks) to plug into.



Use case: mixed flat and private network

This use case is an extension of the above Flat Network use cases. In addition to being able to see one or more shared networks via the OpenStack Networking API, tenants can also have access to private per-tenant networks (only visible to tenant users).

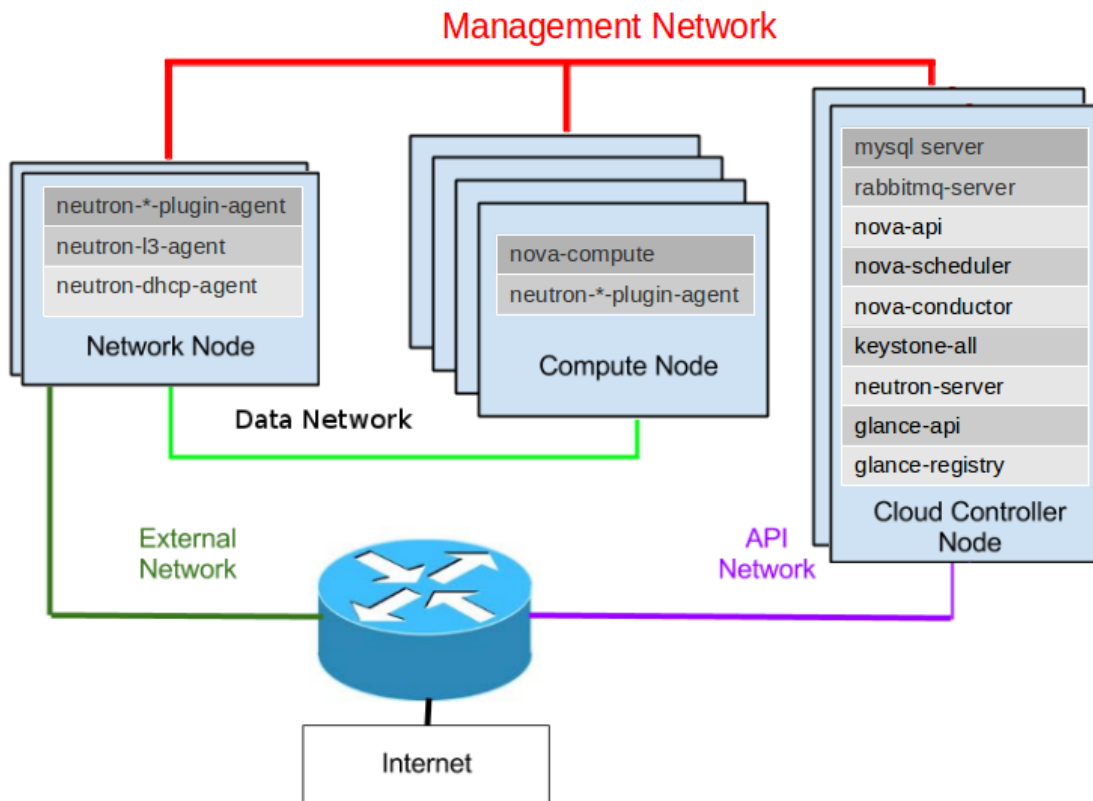
Created VMs can have NICs on any of the shared or private networks that the tenant owns. This enables the creation of multi-tier topologies that use VMs with multiple NICs. It also enables a VM to act as a gateway so that it can provide services such as routing, NAT, and load balancing.



Provider router with private networks

This section describes how to install the OpenStack Networking service and its components for a single router use case: a provider router with private networks.

This figure shows the set up:



Note

Because you run the DHCP agent and L3 agent on one node, you must set `use_namespaces` to `True` (which is the default) in the configuration files for both agents.

The configuration includes these nodes:

Table 9.1. Nodes for use case

Node	Description
Controller	Runs the Networking service, Identity Service, and all Compute services that are required to deploy a VM. The service must have at least two network interfaces. The first should be connected to the Management Network to communicate with the compute and network nodes. The second interface should be connected to the API/public network.
Compute	Runs Compute and the Networking L2 agent. This node does not have access the public network. The node must have a network interface that communicates with the controller node through the management network. The VM receives its IP address from the DHCP agent on this network.
Network	Runs Networking L2 agent, DHCP agent, and L3 agent.

Node	Description
	<p>This node has access to the public network. The DHCP agent allocates IP addresses to the VMs on the network. The L3 agent performs NAT and enables the VMs to access the public network.</p> <p>The node must have:</p> <ul style="list-style-type: none">• A network interface that communicates with the controller node through the management network• A network interface on the data network that manages VM traffic• A network interface that connects to the external gateway on the network

Install

Controller

To install and configure the controller node

1. Run this command:

```
# apt-get install neutron-server
```

2. Configure Networking services:

- Edit the `/etc/neutron/neutron.conf` file and add these lines:

```
core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.  
OVSNeutronPluginV2  
auth_strategy = keystone  
fake_rabbit = False  
rabbit_password = RABBIT_PASS  
  
[database]  
connection = mysql://neutron:NEUTRON_DBPASS@controller/neutron
```

- Edit the `/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini` file and add these lines:

```
[ovs]  
tenant_network_type = vlan  
network_vlan_ranges = physnet1:100:2999
```

- Edit the `/etc/neutron/api-paste.ini` file and add these lines:

```
admin_tenant_name = service  
admin_user = neutron  
admin_password = NEUTRON_PASS
```

3. Start the services:

```
# service neutron-server restart
```

Network node

To install and configure the network node

1. Install the packages:

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

2. Start Open vSwitch:

```
# service openvswitch-switch start
```

3. Add the integration bridge to the Open vSwitch:

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

4. Update the OpenStack Networking `/etc/neutron/neutron.conf` configuration file:

```
rabbit_password = guest  
rabbit_host = controller  
rabbit_password = RABBIT_PASS  
  
[database]  
connection = mysql://neutron:NEUTRON_DBPASS@controller:3306/neutron
```

5. Update the plug-in `/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini` configuration file:

```
[ovs]  
tenant_network_type=vlan  
network_vlan_ranges = physnet1:1:4094  
bridge_mappings = physnet1:br-eth1
```

6. Create the `br-eth1` network bridge. All VM communication between the nodes occurs through `br-eth1`:

```
# ovs-vsctl add-br br-eth1  
# ovs-vsctl add-port br-eth1 eth1
```

7. Create the external network bridge to the Open vSwitch:

```
# ovs-vsctl add-br br-ex  
# ovs-vsctl add-port br-ex eth2
```

8. Edit the `/etc/neutron/l3_agent.ini` file and add these lines:

```
[DEFAULT]  
auth_url = http://controller:35357/v2.0  
admin_tenant_name = service  
admin_user = neutron  
admin_password = NEUTRON_PASS  
metadata_ip = controller  
use_namespaces = True
```

9. Edit the `/etc/neutron/api-paste.ini` file and add these lines:

```
[DEFAULT]
auth_host = controller
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
```

10. Edit the `/etc/neutron/dhcp_agent.ini` file and add this line:

```
use_namespaces = True
```

11. Restart networking services:

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

Compute Node

To install and configure the compute node

1. Install the packages:

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

2. Start the OpenvSwitch service:

```
# service openvswitch-switch start
```

3. Create the integration bridge:

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

4. Create the `br-eth1` network bridge. All VM communication between the nodes occurs through `br-eth1`:

```
# ovs-vsctl add-br br-eth1
# ovs-vsctl add-port br-eth1 eth1
```

5. Edit the OpenStack Networking `/etc/neutron/neutron.conf` configuration file and add this line:

```
rabbit_password = guest
rabbit_host = controller
rabbit_password = RABBIT_PASS

[database]
connection = mysql://neutron:NEUTRON_DBPASS@controller:3306/neutron
```

6. Edit the `/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini` file and add these lines:

```
[ovs]
tenant_network_type = vlan
network_vlan_ranges = physnet1:1:4094
bridge_mappings = physnet1:br-eth1
```

7. Restart the OpenvSwitch Neutron plug-in agent:

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

Logical network configuration



Note

Run these commands on the network node.

Ensure that the following environment variables are set. Various clients use these variables to access the Identity Service.

- Create a `novarc` file:

```
export OS_TENANT_NAME=provider_tenant
export OS_USERNAME=admin
export OS_PASSWORD=password
export OS_AUTH_URL="http://controller:5000/v2.0/"
export SERVICE_ENDPOINT="http://controller:35357/v2.0"
export SERVICE_TOKEN=password
```

- Export the variables:

```
# source novarc echo "source novarc">>.bashrc
```

The admin user creates a network and subnet on behalf of `tenant_A`. A `tenant_A` user can also complete these steps.

To configure internal networking

1. Get the tenant ID (Used as `$TENANT_ID` later).

```
# keystone tenant-list
```

id	name	enabled
48fb81ab2f6b409bafac8961a594980f	provider_tenant	True
cbb574ac1e654a0a992bfc0554237abf	service	True
e371436fe2854ed89cca6c33ae7a83cd	invisible_to_admin	True
e40fa60181524f9f9ee7aa1038748f08	tenant_A	True

2. Create an internal network named `net1` for `tenant_A` (`$TENANT_ID` will be `e40fa60181524f9f9ee7aa1038748f08`):

```
# neutron net-create --tenant-id $TENANT_ID net1
```

Field	Value
admin_state_up	True
id	e99a361c-0af8-4163-9feb-8554d4c37e4f
name	net1
provider:network_type	vlan
provider:physical_network	physnet1
provider:segmentation_id	1024
router:external	False
shared	False
status	ACTIVE

subnets	
tenant_id	e40fa60181524f9f9ee7aa1038748f08

3. Create a subnet on the network **net1** (ID field below is used as `$SUBNET_ID` later):

```
# neutron subnet-create --tenant-id $TENANT_ID net1 10.5.5.0/24
```

Field	Value
allocation_pools	{"start": "10.5.5.2", "end": "10.5.5.254"}
cidr	10.5.5.0/24
dns_nameservers	
enable_dhcp	True
gateway_ip	10.5.5.1
host_routes	
id	c395cb5d-ba03-41ee-8a12-7e792d51a167
ip_version	4
name	
network_id	e99a361c-0af8-4163-9feb-8554d4c37e4f
tenant_id	e40fa60181524f9f9ee7aa1038748f08

A user with the admin role must complete these steps. In this procedure, the user is admin from provider_tenant.

To configure the router and external networking

1. Create a **router1** route. The ID is used as `$ROUTER_ID` later:

```
# neutron router-create router1
```

Field	Value
admin_state_up	True
external_gateway_info	
id	685f64e7-a020-4fdf-a8ad-e41194ae124b
name	router1
status	ACTIVE
tenant_id	48fb81ab2f6b409bafac8961a594980f



Note

The `--tenant-id` parameter is not specified, so this router is assigned to the provider_tenant tenant.

2. Add an interface to the **router1** router and attach it to the subnet from **net1**:

```
# neutron router-interface-add $ROUTER_ID $SUBNET_ID
Added interface to router 685f64e7-a020-4fdf-a8ad-e41194ae124b
```



Note

You can repeat this step to add more interfaces for other networks that belong to other tenants.

3. Create the **ext_net** external network:


```
# neutron net-create ext_net --router:external=True
```

Field	Value
admin_state_up	True
id	8858732b-0400-41f6-8e5c-25590e67ffeb
name	ext_net
provider:network_type	vlan
provider:physical_network	physnet1
provider:segmentation_id	1
router:external	True
shared	False
status	ACTIVE
subnets	
tenant_id	48fb81ab2f6b409bafac8961a594980f

4. Create the subnet for floating IPs.



Note

The DHCP service is disabled for this subnet.

```
# neutron subnet-create ext_net \  
--allocation-pool start=7.7.7.130,end=7.7.7.150 \  
--gateway 7.7.7.1 7.7.7.0/24 --disable-dhcp
```

Field	Value
allocation_pools	{"start": "7.7.7.130", "end": "7.7.7.150"}
cidr	7.7.7.0/24
dns_nameservers	
enable_dhcp	False
gateway_ip	7.7.7.1
host_routes	
id	aef60b55-cbff-405d-a81d-406283ac6cff
ip_version	4
name	
network_id	8858732b-0400-41f6-8e5c-25590e67ffeb
tenant_id	48fb81ab2f6b409bafac8961a594980f

5. Set the gateway for the router to the external network:

```
# neutron router-gateway-set $ROUTER_ID $EXTERNAL_NETWORK_ID  
Set gateway for router 685f64e7-a020-4fdf-a8ad-e41194ae124b
```

A user from tenant_A completes these steps, so the credentials in the environment variables are different than those in the previous procedure.

To allocate floating IP addresses

1. You can associate a floating IP address with a VM after it starts. Find the ID of the port (\$PORT_ID) that was allocated for the VM, as follows:

```
# nova list
```

ID	Name	Status	Networks
----	------	--------	----------

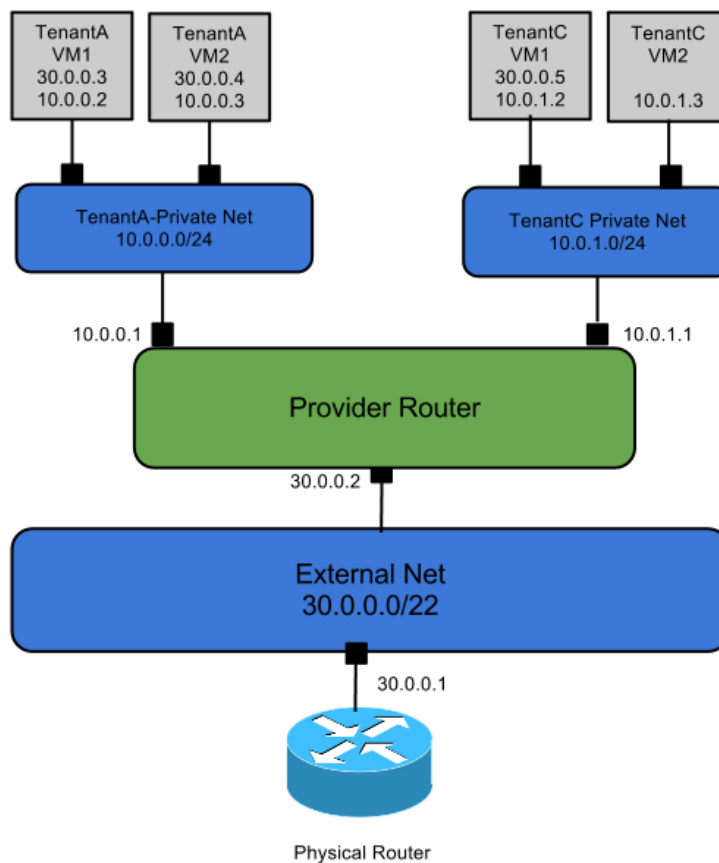

```
64 bytes from 7.7.7.131: icmp_req=2 ttl=64 time=0.152 ms
64 bytes from 7.7.7.131: icmp_req=3 ttl=64 time=0.049 ms
```

Use case: provider router with private networks

This use case provides each tenant with one or more private networks that connect to the outside world through an OpenStack Networking router. When each tenant gets exactly one network, this architecture maps to the same logical topology as the VlanManager in Compute (although of course, Networking does not require VLANs). Using the Networking API, the tenant can only see a network for each private network assigned to that tenant. The router object in the API is created and owned by the cloud administrator.

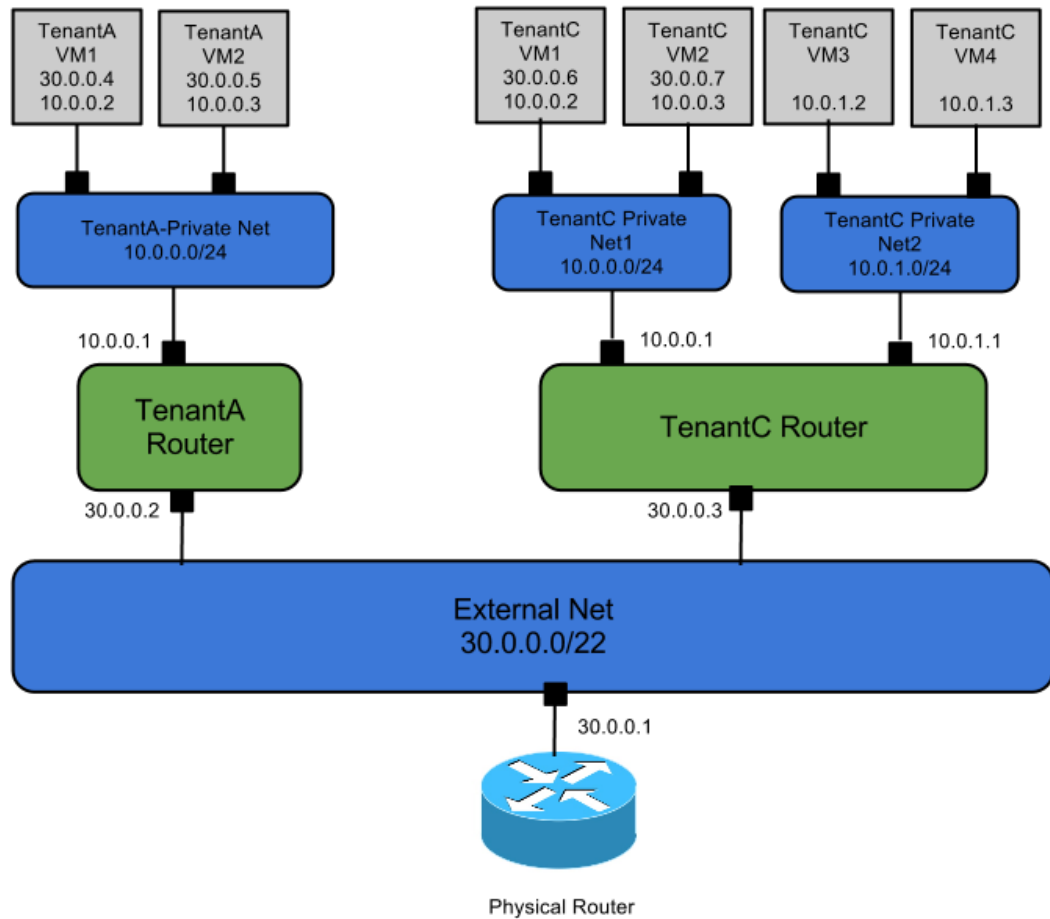
This model supports assigning public addresses to VMs by using *floating IPs*; the router maps public addresses from the external network to fixed IPs on private networks. Hosts without floating IPs can still create outbound connections to the external network because the provider router performs SNAT to the router's external IP. The IP address of the physical router is used as the `gateway_ip` of the external network subnet, so the provider has a default router for Internet traffic.

The router provides L3 connectivity among private networks. Tenants can reach instances for other tenants unless you use additional filtering, such as, security groups). With a single router, tenant networks cannot use overlapping IPs. To resolve this issue, the administrator can create private networks on behalf of the tenants.

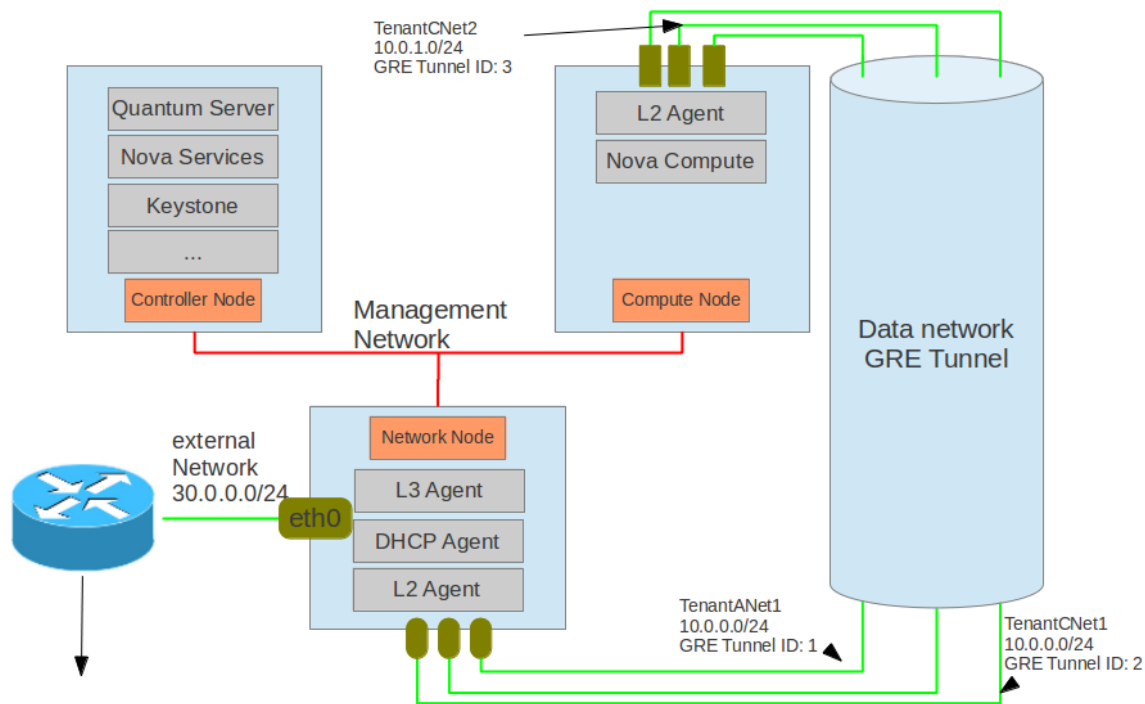


Per-tenant routers with private networks

This section describes how to install the OpenStack Networking service and its components for a use case that has per-tenant routers with private networks.



The following figure shows the setup:



As shown in the figure, the setup includes:

- An interface for management traffic on each node.
- Use of the Open vSwitch plug-in.
- GRE tunnels for data transport on all agents.
- Floating IPs and router gateway ports that are configured in an external network, and a physical router that connects the floating IPs and router gateway ports to the outside world.



Note

Because this example runs a DHCP agent and L3 agent on one node, you must set the `use_namespace` option to `True` in the configuration file for each agent. The default is `True`.

This table describes the nodes:

Node	Description
Controller Node	Runs Networking, Identity Service, and all Compute services that are required to deploy VMs (<code>nova-api</code> , <code>nova-scheduler</code> , for example). The node must have at least one network interface, which connects to the Management Network. The host name is <code>controlnode</code> , which other nodes resolve to the IP of the controller node.

Note

The `nova-network` service should not be running. This is replaced by Networking.

Node	Description
Compute Node	Runs the Networking L2 agent and the Compute services that run VMs (<code>nova-compute</code> specifically, and optionally other <code>nova-*</code> services depending on configuration). The node must have at least two network interfaces. One interface communicates with the controller node through the management network. The other node is used for the VM traffic on the data network. The VM receives its IP address from the DHCP agent on this network.
Network Node	Runs Networking L2 agent, DHCP agent and L3 agent. This node has access to the external network. The DHCP agent allocates IP addresses to the VMs on data network. (Technically, the addresses are allocated by the Networking server, and distributed by the dhcp agent.) The node must have at least two network interfaces. One interface communicates with the controller node through the management network. The other interface is used as external network. GRE tunnels are set up as data networks.
Router	Router has IP 30.0.0.1, which is the default gateway for all VMs. The router must be able to access public networks.

The use case assumes the following:

Controller node

1. Relevant Compute services are installed, configured, and running.
2. Glance is installed, configured, and running. In addition, an image named `ttt` must be present.
3. Identity is installed, configured, and running. A Networking user named **neutron** should be created on tenant **service** with password **NEUTRON_PASS**.
4. Additional services:
 - RabbitMQ is running with default guest and its password.
 - MySQL server (user is **root** and password is **root**).

Compute node

Install and configure Compute.

Install

- **Controller node#Networking server**
 1. Install the Networking server.
 2. Create database **ovs_neutron**.
 3. Update the Networking configuration file, `/etc/neutron/neutron.conf`, with plug-in choice and Identity Service user as necessary:

```
[DEFAULT]
core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.
OVSNeutronPluginV2
control_exchange = neutron
rabbit_host = controller
rabbit_password = RABBIT_PASS
notification_driver = neutron.openstack.common.notifier.rabbit_notifier

[database]
connection = mysql://neutron:NEUTRON_DBPASS@controller:3306/neutron

[keystone_authtoken]
admin_tenant_name=service
admin_user=neutron
admin_password=NEUTRON_PASS
```

4. Update the plug-in configuration file, `/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini`:

```
[ovs]
tenant_network_type = gre
tunnel_id_ranges = 1:1000
enable_tunneling = True
```

5. Start the Networking server.

The Networking server can be a service of the operating system. The command to start the service depends on your operating system. The following command runs the Networking server directly:

```
# neutron-server --config-file /etc/neutron/plugins/openvswitch/
ovs_neutron_plugin.ini \
--config-file /etc/neutron/neutron.conf
```

- **Compute node#Compute**

1. Install Compute services.
2. Update the Compute `/etc/nova/nova.conf` configuration file. Make sure the following line appears at the end of this file:

```
network_api_class=nova.network.neutronv2.api.API

neutron_admin_username=neutron
neutron_admin_password=NEUTRON_PASS
neutron_admin_auth_url=http://controlnode:35357/v2.0/
neutron_auth_strategy=keystone
neutron_admin_tenant_name=service
neutron_url=http://controlnode:9696/

libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtHybridOVSBridgeDriver
```

3. Restart relevant Compute services.
- **Compute and Networking node#L2 agent**

1. Install and start Open vSwitch.

2. Install the L2 agent (Neutron Open vSwitch agent).
3. Add the integration bridge to the Open vSwitch:

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

4. Update the Networking configuration file, `/etc/neutron/neutron.conf`:

```
[DEFAULT]
core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.
OVSNeutronPluginV2
control_exchange = neutron
rabbit_host = controller
rabbit_password = RABBIT_PASS
notification_driver = neutron.openstack.common.notifier.rabbit_notifier

[database]
connection = mysql://neutron:NEUTRON_DBPASS@controller:3306/neutron
```

5. Update the plug-in configuration file, `/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini`.

Compute node:

```
[ovs]
tenant_network_type = gre
tunnel_id_ranges = 1:1000
enable_tunneling = True
local_ip = 9.181.89.202
```

Network node:

```
[ovs]
tenant_network_type = gre
tunnel_id_ranges = 1:1000
enable_tunneling = True
local_ip = 9.181.89.203
```

6. Create the integration bridge `br-int`:

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

7. Start the Networking L2 agent

The Networking Open vSwitch L2 agent can be a service of operating system. The command to start depends on your operating systems. The following command runs the service directly:

```
# neutron-openvswitch-agent --config-file /etc/neutron/plugins/
openvswitch/ovs_neutron_plugin.ini \
--config-file /etc/neutron/neutron.conf
```

- **Network node#DHCP agent**

1. Install the DHCP agent.
2. Update the Networking configuration file, `/etc/neutron/neutron.conf`


```
[DEFAULT]
core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.
OVSNeutronPluginV2
control_exchange = neutron
rabbit_host = controller
rabbit_password = RABBIT_PASS
notification_driver = neutron.openstack.common.notifier.rabbit_notifier
allow_overlapping_ips = True
```

Set `allow_overlapping_ips` because TenantA and TenantC use overlapping subnets.

3. Update the DHCP `/etc/neutron/dhcp_agent.ini` configuration file:

```
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
```

4. Start the DHCP agent.

The Networking DHCP agent can be a service of operating system. The command to start the service depends on your operating system. The following command runs the service directly:

```
# neutron-dhcp-agent --config-file /etc/neutron/neutron.conf \
--config-file /etc/neutron/dhcp_agent.ini
```

- **Network node#L3 agent**

1. Install the L3 agent.
2. Add the external network bridge

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

3. Add the physical interface, for example `eth0`, that is connected to the outside network to this bridge:

```
# ovs-vsctl add-port br-ex eth0
```

4. Update the L3 configuration file `/etc/neutron/l3_agent.ini`:

```
[DEFAULT]
interface_driver=neutron.agent.linux.interface.OVSInterfaceDriver
use_namespaces=True
```

Set the `use_namespaces` option (it is True by default) because TenantA and TenantC have overlapping subnets, and the routers are hosted on one L3 agent network node.

5. Start the L3 agent

The Networking L3 agent can be a service of operating system. The command to start the service depends on your operating system. The following command starts the agent directly:

```
# neutron-l3-agent --config-file /etc/neutron/neutron.conf \
--config-file /etc/neutron/l3_agent.ini
```

Configure logical network

You can run these commands on the network node.



Note

Ensure that the following environment variables are set. Various clients use these to access the Identity Service.

```
export OS_USERNAME=admin
export OS_PASSWORD=adminpassword
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://127.0.0.1:5000/v2.0/
```

1. Get the tenant ID (Used as \$TENANT_ID later):

```
# keystone tenant-list
+-----+-----+-----+
|          id          | name | enabled |
+-----+-----+-----+
| 247e478c599f45b5bd297e8ddbcb9b6a | TenantA | True |
| 2b4fec24e62e4ff28a8445ad83150f9d | TenantC | True |
| 3719a4940bf24b5a8124b58c9b0a6ee6 | TenantB | True |
| 5fcfbc3283a142a5bb6978b549a511ac | demo   | True |
| b7445f221cda4f4a8ac7db6b218b1339 | admin  | True |
+-----+-----+-----+
```

2. Get user information:

```
# keystone user-list
+-----+-----+-----+-----+
|          id          | name | enabled | email |
+-----+-----+-----+-----+
| 5a9149ed991744fa85f71e4aa92eb7ec | demo | True   |      |
| 5b419c74980d46a1ab184e7571a8154e | admin | True   | admin@example.com |
| 8e37cb8193cb4873a35802d257348431 | UserC | True   |      |
| c11f6b09ed3c45c09c21cbbc23e93066 | UserB | True   |      |
| ca567c4f6c0942bdac0e011e97bddbe3 | UserA | True   |      |
+-----+-----+-----+-----+
```

3. Create the external network and its subnet by admin user:

```
# neutron net-create Ext-Net --provider:network_type local --
router:external true
Created a new network:
+-----+-----+
| Field          | Value |
+-----+-----+
| admin_state_up | True  |
| id             | 2c757c9e-d3d6-4154-9a77-336eb99bd573 |
| name          | Ext-Net |
| provider:network_type | local |
| provider:physical_network |      |
| provider:segmentation_id |      |
| router:external | True  |
| shared        | False |
| status        | ACTIVE |
| subnets     |      |
| tenant_id    | b7445f221cda4f4a8ac7db6b218b1339 |
+-----+-----+
```

```
# neutron subnet-create Ext-Net 30.0.0.0/24 --disable-dhcp
Created a new subnet:
+-----+-----+
| Field          | Value                                     |
+-----+-----+
| allocation_pools | {"start": "30.0.0.2", "end": "30.0.0.254"} |
| cidr            | 30.0.0.0/24                             |
| dns_nameservers |                                           |
| enable_dhcp     | False                                    |
| gateway_ip      | 30.0.0.1                                 |
| host_routes     |                                           |
| id              | ba754a55-7ce8-46bb-8d97-aa83f4ffa5f9    |
| ip_version      | 4                                        |
| name            |                                           |
| network_id      | 2c757c9e-d3d6-4154-9a77-336eb99bd573    |
| tenant_id       | b7445f221cda4f4a8ac7db6b218b1339      |
+-----+-----+
```

provider:network_type local means that Networking does not have to realize this network through provider network. **router:external true** means that an external network is created where you can create floating IP and router gateway port.

4. Add an IP on external network to br-ex.

Because br-ex is the external network bridge, add an IP 30.0.0.100/24 to br-ex and ping the floating IP of the VM from our network node.

```
# ip addr add 30.0.0.100/24 dev br-ex
# ip link set br-ex up
```

5. Serve TenantA.

For TenantA, create a private network, subnet, server, router, and floating IP.

- a. Create a network for TenantA:

```
# neutron --os-tenant-name TenantA --os-username UserA --os-password
password \
  --os-auth-url=http://localhost:5000/v2.0 net-create TenantA-Net
Created a new network:
+-----+-----+
| Field          | Value                                     |
+-----+-----+
| admin_state_up | True                                     |
| id              | 7d0e8d5d-c63c-4f13-a117-4dc4e33e7d68    |
| name            | TenantA-Net                             |
| router:external | False                                    |
| shared          | False                                    |
| status          | ACTIVE                                  |
| subnets        |                                           |
| tenant_id       | 247e478c599f45b5bd297e8ddb9b6a         |
+-----+-----+
```

After that, you can use admin user to query the provider network information:

```
# neutron net-show TenantA-Net
```

Field	Value
admin_state_up	True
id	7d0e8d5d-c63c-4f13-a117-4dc4e33e7d68
name	TenantA-Net
provider:network_type	gre
provider:physical_network	
provider:segmentation_id	1
router:external	False
shared	False
status	ACTIVE
subnets	
tenant_id	247e478c599f45b5bd297e8ddb9b6a

The network has GRE tunnel ID (for example, provider:segmentation_id) 1.

- b. Create a subnet on the network TenantA-Net:

```
# neutron --os-tenant-name TenantA --os-username UserA --os-password
password \
  --os-auth-url=http://localhost:5000/v2.0 subnet-create TenantA-Net
10.0.0.0/24
```

Created a new subnet:

Field	Value
allocation_pools	{ "start": "10.0.0.2", "end": "10.0.0.254" }
cidr	10.0.0.0/24
dns_nameservers	
enable_dhcp	True
gateway_ip	10.0.0.1
host_routes	
id	51e2c223-0492-4385-b6e9-83d4e6d10657
ip_version	4
name	
network_id	7d0e8d5d-c63c-4f13-a117-4dc4e33e7d68
tenant_id	247e478c599f45b5bd297e8ddb9b6a

- c. Create a server for TenantA:

```
$ nova --os-tenant-name TenantA --os-username UserA --os-password
password \
  --os-auth-url=http://localhost:5000/v2.0 boot --image tty --flavor 1
\
  --nic net-id=7d0e8d5d-c63c-4f13-a117-4dc4e33e7d68 TenantA_VM1
```

```
$ nova --os-tenant-name TenantA --os-username UserA --os-password
password \
  --os-auth-url=http://localhost:5000/v2.0 list
```

ID	Name	Status
7c5e6499-7ef7-4e36-8216-62c2941d21ff	TenantA_VM1	ACTIVE
TenantA-Net=10.0.0.3		

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



Note

It is important to understand that you should not attach the instance to Ext-Net directly. Instead, you must use a floating IP to make it accessible from the external network.

d. Create and configure a router for TenantA:

```
# neutron --os-tenant-name TenantA --os-username UserA --os-password
password \
  --os-auth-url=http://localhost:5000/v2.0 router-create TenantA-R1
Created a new router:
```

Field	Value
admin_state_up	True
external_gateway_info	
id	59cd02cb-6ee6-41e1-9165-d251214594fd
name	TenantA-R1
status	ACTIVE
tenant_id	247e478c599f45b5bd297e8ddb9b6a

```
# neutron --os-tenant-name TenantA --os-username UserA --os-password
password \
  --os-auth-url=http://localhost:5000/v2.0 router-interface-add \
  TenantA-R1 51e2c223-0492-4385-b6e9-83d4e6d10657
```

Added interface to router TenantA-R1

```
# neutron --os-tenant-name TenantA --os-username UserA --os-password
password \
  --os-auth-url=http://localhost:5000/v2.0 \
  router-gateway-set TenantA-R1 Ext-Net
```

6. Associate a floating IP for TenantA_VM1.

a. Create a floating IP:

```
# neutron --os-tenant-name TenantA --os-username UserA --os-password
password \
  --os-auth-url=http://localhost:5000/v2.0 floatingip-create Ext-Net
Created a new floatingip:
```

Field	Value
fixed_ip_address	
floating_ip_address	30.0.0.2
floating_network_id	2c757c9e-d3d6-4154-9a77-336eb99bd573
id	5a1f90ed-aa3c-4df3-82cb-116556e96bf1
port_id	
router_id	
tenant_id	247e478c599f45b5bd297e8ddb9b6a

b. Get the port ID of the VM with ID 7c5e6499-7ef7-4e36-8216-62c2941d21ff:


```
$ ping 30.0.0.2
PING 30.0.0.2 (30.0.0.2) 56(84) bytes of data.
64 bytes from 30.0.0.2: icmp_req=1 ttl=63 time=45.0 ms
64 bytes from 30.0.0.2: icmp_req=2 ttl=63 time=0.898 ms
64 bytes from 30.0.0.2: icmp_req=3 ttl=63 time=0.940 ms
^C
--- 30.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.898/15.621/45.027/20.793 ms
```

9. Create other servers for TenantA.

You can create more servers for TenantA and add floating IPs for them.

10. Serve TenantC.

For TenantC, you create two private networks with subnet 10.0.0.0/24 and subnet 10.0.1.0/24, some servers, one router to connect to these two subnets and some floating IPs.

a. Create networks and subnets for TenantC:

```
# neutron --os-tenant-name TenantC --os-username UserC --os-password
password \
  --os-auth-url=http://localhost:5000/v2.0 net-create TenantC-Net1
# neutron --os-tenant-name TenantC --os-username UserC --os-password
password \
  --os-auth-url=http://localhost:5000/v2.0 subnet-create TenantC-Net1
\
  10.0.0.0/24 --name TenantC-Subnet1
# neutron --os-tenant-name TenantC --os-username UserC --os-password
password \
  --os-auth-url=http://localhost:5000/v2.0 net-create TenantC-Net2
# neutron --os-tenant-name TenantC --os-username UserC --os-password
password \
  --os-auth-url=http://localhost:5000/v2.0 subnet-create TenantC-Net2
\
  10.0.1.0/24 --name TenantC-Subnet2
```

After that you can use admin user to query the network's provider network information:

```
# neutron net-show TenantC-Net1
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | True |
| id | 91309738-c317-40a3-81bb-bed7a3917a85 |
| name | TenantC-Net1 |
| provider:network_type | gre |
| provider:physical_network | |
| provider:segmentation_id | 2 |
| router:external | False |
| shared | False |
| status | ACTIVE |
| subnets | cf03fd1e-164b-4527-bc87-2b2631634b83 |
| tenant_id | 2b4fec24e62e4ff28a8445ad83150f9d |
+-----+-----+
```

```
# neutron net-show TenantC-Net2
+-----+-----+
| Field                | Value                |
+-----+-----+
| admin_state_up      | True                 |
| id                   | 5b373ad2-7866-44f4-8087-f87148abd623 |
| name                 | TenantC-Net2        |
| provider:network_type | gre                  |
| provider:physical_network |                    |
| provider:segmentation_id | 3                    |
| router:external     | False                |
| shared               | False                |
| status               | ACTIVE               |
| subnets             | 38f0b2f0-9f98-4bf6-9520-f4abede03300 |
| tenant_id            | 2b4fec24e62e4ff28a8445ad83150f9d    |
+-----+-----+
```

You can see GRE tunnel IDs (such as, `provider:segmentation_id`) 2 and 3. And also note the network IDs and subnet IDs because you use them to create VMs and router.

- b. Create a server TenantC-VM1 for TenantC on TenantC-Net1.

```
# nova --os-tenant-name TenantC --os-username UserC --os-password
password \
  --os-auth-url=http://localhost:5000/v2.0 boot --image tty --flavor 1
\
  --nic net-id=91309738-c317-40a3-81bb-bed7a3917a85 TenantC_VM1
```

- c. Create a server TenantC-VM3 for TenantC on TenantC-Net2.

```
# nova --os-tenant-name TenantC --os-username UserC --os-password
password \
  --os-auth-url=http://localhost:5000/v2.0 boot --image tty --flavor 1
\
  --nic net-id=5b373ad2-7866-44f4-8087-f87148abd623 TenantC_VM3
```

- d. List servers of TenantC.

```
# nova --os-tenant-name TenantC --os-username UserC --os-password
password \
  --os-auth-url=http://localhost:5000/v2.0 list
+-----+-----+-----+
+-----+
| ID                | Name                | Status |
+-----+-----+-----+
| b739fa09-902f-4b37-bcb4-06e8a2506823 | TenantC_VM1        | ACTIVE |
| TenantC-Net1=10.0.0.3 |                    |        |
| 17e255b2-b14f-48b3-ab32-5df36566d2e8 | TenantC_VM3        | ACTIVE |
| TenantC-Net2=10.0.1.3 |                    |        |
+-----+-----+-----+
```

Note the server IDs because you use them later.

- e. Make sure servers get their IPs.

You can use VNC to log on the VMs to check if they get IPs. If not, you must make sure that the Networking components are running correctly and the GRE tunnels work.

- f. Create and configure a router for TenantC:

```
# neutron --os-tenant-name TenantC --os-username UserC --os-password  
password \  
--os-auth-url=http://localhost:5000/v2.0 router-create TenantC-R1
```

```
# neutron --os-tenant-name TenantC --os-username UserC --os-password  
password \  
--os-auth-url=http://localhost:5000/v2.0 router-interface-add \  
TenantC-R1 cf03fd1e-164b-4527-bc87-2b2631634b83
```

```
# neutron --os-tenant-name TenantC --os-username UserC --os-password  
password \  
--os-auth-url=http://localhost:5000/v2.0 router-interface-add \  
TenantC-R1 38f0b2f0-9f98-4bf6-9520-f4abede03300
```

```
# neutron --os-tenant-name TenantC --os-username UserC --os-password  
password \  
--os-auth-url=http://localhost:5000/v2.0 \  
router-gateway-set TenantC-R1 Ext-Net
```

- g. Checkpoint: ping from within TenantC's servers.

Because a router connects to two subnets, the VMs on these subnets can ping each other. And because the gateway for the router is set, TenantC's servers can ping external network IPs, such as 192.168.1.1, 30.0.0.1, and so on.

- h. Associate floating IPs for TenantC's servers.

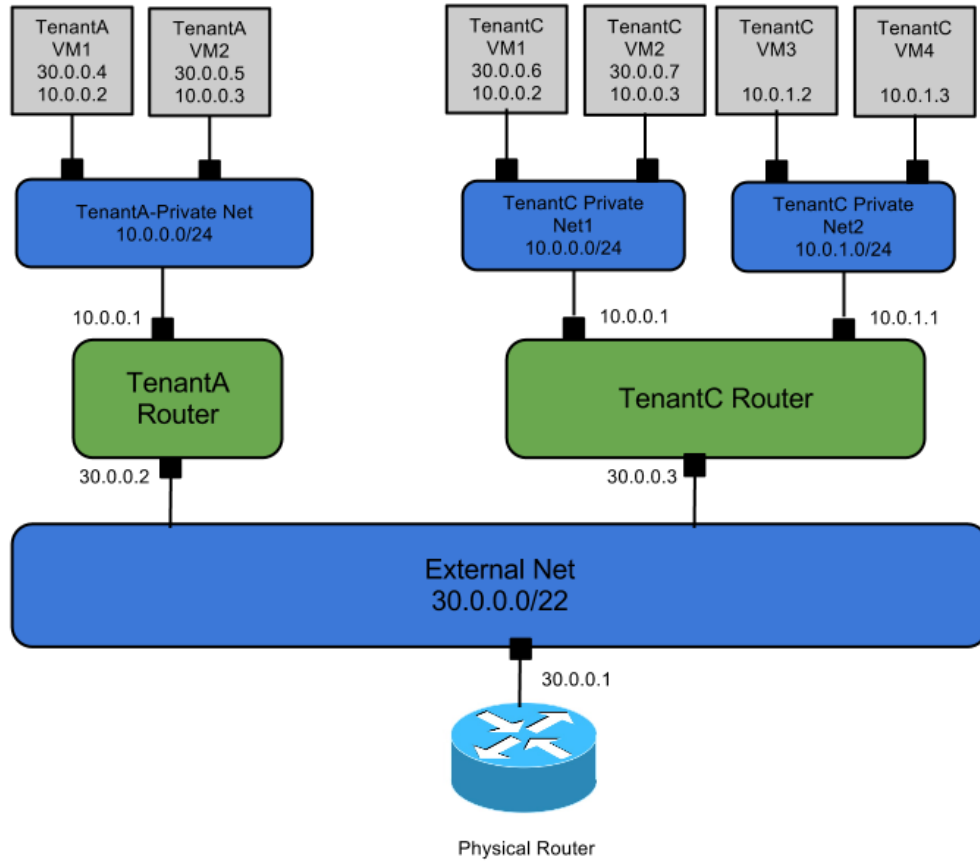
Because a router connects to two subnets, the VMs on these subnets can ping each other. And because the gateway interface for the router is set, TenantC's servers can ping external network IPs, such as 192.168.1.1, 30.0.0.1, and so on.

- i. Associate floating IPs for TenantC's servers.

You can use similar commands to the ones used in the section for TenantA.

Use case: per-tenant routers with private networks

This use case represents a more advanced router scenario in which each tenant gets at least one router, and potentially has access to the Networking API to create additional routers. The tenant can create their own networks, potentially uplinking those networks to a router. This model enables tenant-defined, multi-tier applications, with each tier being a separate network behind the router. Because there are multiple routers, tenant subnets can overlap without conflicting, because access to external networks all happens through SNAT or floating IPs. Each router uplink and floating IP is allocated from the external network subnet.



10. Add the Orchestration service

Table of Contents

Orchestration service overview	106
Install the Orchestration service	106
Verify the Orchestration service installation	108

Use the OpenStack Orchestration service to create cloud resources using a template language called HOT. The integrated project name is Heat.

Orchestration service overview

The Orchestration service provides a template-based orchestration for describing a cloud application by running OpenStack API calls to generate running cloud applications. The software integrates other core components of OpenStack into a one-file template system. The templates enable you to create most OpenStack resource types, such as instances, floating IPs, volumes, security groups, users, and so on. Also, provides some more advanced functionality, such as instance high availability, instance auto-scaling, and nested stacks. By providing very tight integration with other OpenStack core projects, all OpenStack core projects could receive a larger user base.

The service enables deployers to integrate with the Orchestration service directly or through custom plug-ins.

The Orchestration service consists of the following components:

- `heat` command-line client. A CLI that communicates with the `heat-api` to run AWS CloudFormation APIs. End developers could also use the Orchestration REST API directly.
- `heat-api` component. Provides an OpenStack-native REST API that processes API requests by sending them to the `heat-engine` over RPC.
- `heat-api-cfn` component. Provides an AWS Query API that is compatible with AWS CloudFormation and processes API requests by sending them to the `heat-engine` over RPC.
- `heat-engine`. Orchestrates the launching of templates and provides events back to the API consumer.

Install the Orchestration service

1. Install the OpenStack Orchestration service on the controller node:

```
# apt-get install heat-api heat-api-cfn heat-engine
```

2. In the configuration file, specify the location of the database where the Orchestration service stores data. These examples use a MySQL database with a `heat` user on the controller node. Replace `HEAT_DBPASS` with the password for the database user:

Edit `/etc/heat/heat.conf` and change the `[DEFAULT]` section.

```
[database]
# The SQLAlchemy connection string used to connect to the database
connection = mysql://heat:HEAT_DBPASS@controller/heat
...
```

3. By default, the Ubuntu packages create an SQLite database. Delete the `heat.sqlite` file that was created in the `/var/lib/heat/` directory so that it does not get used by mistake.
4. Use the password that you set previously to log in as `root` and create a `heat` database user:

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

5. Create the `heat` service tables:

```
# heat-manage db_sync
```



Note

Ignore `DeprecationWarning` errors.

6. The Ubuntu packages do not correctly set up logging. Edit the `/etc/heat/heat.conf` file and change the `[DEFAULT]` section:

```
[DEFAULT]
...
# Print more verbose output (set logging level to INFO instead
# of default WARNING level). (boolean value)
verbose = True
...
# (Optional) The base directory used for relative --log-file
# paths (string value)
log_dir=/var/log/heat
```

7. Configure the Orchestration Service to use the RabbitMQ message broker.

Edit `/etc/heat/heat.conf` and modify the `[DEFAULT]` section:

```
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

8. Create a `heat` user that the Orchestration service can use to authenticate with the Identity Service. Use the `service` tenant and give the user the `admin` role:

```
# keystone user-create --name=heat --pass=HEAT_PASS --email=heat@example.com
# keystone user-role-add --user=heat --tenant=service --role=admin
```

9. Edit the `/etc/heat/heat.conf` file to change the `[keystone_auth_token]` and `[ec2_auth_token]` sections to add credentials to the Orchestration Service:

```
[keystone_authtoken]
auth_host = controller
auth_port = 35357
auth_protocol = http
auth_uri = http://controller:5000/v2.0
admin_tenant_name = service
admin_user = heat
admin_password = HEAT_PASS
[ec2_authtoken]
auth_uri = http://controller:5000/v2.0
keystone_ec2_uri = http://controller:5000/v2.0/ec2tokens
```

10. Register the Heat and CloudFormation APIs with the Identity Service so that other OpenStack services can locate these APIs. Register the service and specify the endpoint:

```
# keystone service-create --name=heat --type=orchestration \
--description="Heat Orchestration API"
```

11. Use the `id` property that is returned for the service to create the endpoint:

```
# keystone endpoint-create \
--service-id=the_service_id_above \
--publicurl=http://controller:8004/v1/%(tenant_id)s \
--internalurl=http://controller:8004/v1/%(tenant_id)s \
--adminurl=http://controller:8004/v1/%(tenant_id)s
```

```
# keystone service-create --name=heat-cfn --type=cloudformation \
--description="Heat CloudFormation API"
```

12. Use the `id` property that is returned for the service to create the endpoint:

```
# keystone endpoint-create \
--service-id=the_service_id_above \
--publicurl=http://controller:8000/v1 \
--internalurl=http://controller:8000/v1 \
--adminurl=http://controller:8000/v1
```

13. Restart the service with its new settings:

```
# service heat-api restart
# service heat-api-cfn restart
# service heat-engine restart
```

Verify the Orchestration service installation

To verify that the Orchestration service is installed and configured correctly, make sure that your credentials are set up correctly in the `openrc` file. Source the file, as follows:

```
# source openrc
```

Next, create some stacks by using the samples.

Create and manage stacks

Create a stack from an example template file

1. To create a stack, or template, from an [example template file](#), run the following command:

```
$ heat stack-create mystack --template-file=/PATH_TO_HEAT_TEMPLATES/  
WordPress_Single_Instance.template  
--parameters="InstanceType=m1.  
large;DBUsername=USERNAME;DBPassword=PASSWORD;KeyName=HEAT_KEY;LinuxDistribution=  
F17"
```

The `--parameters` values that you specify depend on the parameters that are defined in the template. If a website hosts the template file, you can specify the URL with the `--template-url` parameter instead of the `--template-file` parameter.

The command returns the following output:

```
+-----+-----+-----+-----+  
+-----+-----+-----+-----+  
| id          | stack_name  | stack_status  | creation_time  
|  
+-----+-----+-----+-----+  
+-----+-----+-----+-----+  
| 4c712026-dcd5-4664-90b8-0915494c1332 | mystack      |  
CREATE_IN_PROGRESS | 2013-04-03T23:22:08Z |  
+-----+-----+-----+-----+  
+-----+-----+-----+-----+
```

2. You can also use the `stack-create` command to validate a template file without creating a stack from it.

To do so, run the following command:

```
$ heat stack-create mystack --template-file=/PATH_TO_HEAT_TEMPLATES/  
WordPress_Single_Instance.template
```

If validation fails, the response returns an error message.

Get information about stacks

To explore the state and history of a particular stack, you can run a number of commands.

- To see which stacks are visible to the current user, run the following command:

```
$ heat stack-list  
+-----+-----+-----+-----+  
+-----+-----+-----+-----+  
| id          | stack_name  | stack_status  | creation_time  
+-----+-----+-----+-----+  
+-----+-----+-----+-----+  
| 4c712026-dcd5-4664-90b8-0915494c1332 | mystack      | CREATE_COMPLETE |  
2013-04-03T23:22:08Z |  
| 7edc7480-bda5-4e1c-9d5d-f567d3b6a050 | my-otherstack | CREATE_FAILED   |  
2013-04-03T23:28:20Z |  
+-----+-----+-----+-----+  
+-----+-----+-----+-----+
```

- To show the details of a stack, run the following command:

```
$ heat stack-show mystack
```

- A stack consists of a collection of resources.

To list the resources and their status, run the following command:

```
$ heat resource-list mystack
+-----+-----+-----+-----+
+-----+
| logical_resource_id | resource_type      | resource_status | updated_time
|                   |                   |                 |
+-----+-----+-----+-----+
+-----+
| WikiDatabase       | AWS::EC2::Instance | CREATE_COMPLETE |
| 2013-04-03T23:25:56Z |                   |                 |
+-----+-----+-----+-----+
+-----+
```

- To show the details for the specified resource in a stack, run the following command:

```
$ heat resource-show mystack WikiDatabase
```

Some resources have associated metadata which can change throughout the life-cycle of a resource:

```
$ heat resource-metadata mystack WikiDatabase
```

- A series of events is generated during the life-cycle of a stack.

To display life-cycle events, run::

```
$ heat event-list mystack
+-----+-----+-----+-----+
+-----+
| logical_resource_id | id | resource_status_reason | resource_status |
| event_time         |    |                       |                 |
+-----+-----+-----+-----+
+-----+
| WikiDatabase       | 1  | state changed         | IN_PROGRESS     |
| 2013-04-03T23:22:09Z |    |                       |                 |
| WikiDatabase       | 2  | state changed         | CREATE_COMPLETE |
| 2013-04-03T23:25:56Z |    |                       |                 |
+-----+-----+-----+-----+
+-----+
```

- To show the details for a particular event, run the following command:

```
$ heat event-show WikiDatabase 1
```

Update a stack

- To update an existing stack from a modified template file, run a command like the following command:

```
$ heat stack-update mystack --template-file=/path/to/heat/templates/
WordPress_Single_Instance_v2.template
  --parameters="InstanceType=m1.large;DBUsername=wp;DBPassword=
verybadpassword;KeyName=heat_key;LinuxDistribution=F17"
+-----+-----+-----+-----+
+-----+
| id          | stack_name      | stack_status    | creation_time    |
+-----+-----+-----+-----+
+-----+
```

```
| 4c712026-dcd5-4664-90b8-0915494c1332 | mystack      | UPDATE_COMPLETE |  
2013-04-03T23:22:08Z |  
| 7edc7480-bda5-4e1c-9d5d-f567d3b6a050 | my-otherstack | CREATE_FAILED   |  
2013-04-03T23:28:20Z |  
+-----+-----+-----+  
+-----+
```

Some resources are updated in-place, while others are replaced with new resources.

11. Add the Telemetry service

Table of Contents

The Telemetry Service	112
Install the Telemetry service	113
Install the Compute agent for the Telemetry service	115
Add the Image Service agent for the Telemetry service	116
Add the Block Storage Service agent for the Telemetry service	116
Add the Object Storage agent for the Telemetry service	116
Verify the Telemetry Service installation	117

The OpenStack Telemetry service provides a framework for monitoring and metering the OpenStack cloud. It is also known as the Ceilometer project.

The Telemetry Service

The OpenStack Telemetry service:

- Efficiently collects the metering data about the CPU and network costs.
- Collects data by monitoring notifications sent from services or by polling the infrastructure.
- Configures the type of collected data to meet various operating requirements. Accessing and inserting the metering data through the REST API.
- Expands the framework to collect custom usage data by additional plug-ins.
- Produces signed metering messages that cannot be repudiated.

The system consists of the following basic components:

- A compute agent (`ceilometer-agent-compute`). Runs on each compute node and polls for resource utilization statistics. There may be other types of agents in the future, but for now we will focus on creating the compute agent.
- A central agent (`ceilometer-agent-central`). Runs on a central management server to poll for resource utilization statistics for resources not tied to instances or compute nodes.
- A collector (`ceilometer-collector`). Runs on one or more central management servers to monitor the message queues (for notifications and for metering data coming from the agent). Notification messages are processed and turned into metering messages and sent back out onto the message bus using the appropriate topic. Telemetry messages are written to the data store without modification.

- An alarm notifier (`ceilometer-alarm-notifier`). Runs on one or more central management servers to allow setting alarms based on threshold evaluation for a collection of samples.
- A data store. A database capable of handling concurrent writes (from one or more collector instances) and reads (from the API server).
- An API server (`ceilometer-api`). Runs on one or more central management servers to provide access to the data from the data store. These services communicate using the standard OpenStack messaging bus. Only the collector and API server have access to the data store.

These services communicate by using the standard OpenStack messaging bus. Only the collector and API server have access to the data store.

Install the Telemetry service

OpenStack Telemetry is an API service that provides a collector and a range of disparate agents. Before you can install these agents on nodes such as the compute node, you must use this procedure to install the core components on the controller node.

1. Install the Telemetry Service on the controller node:

```
# apt-get install ceilometer-api ceilometer-collector ceilometer-agent-  
central python-ceilometerclient
```

2. The Telemetry Service uses a database to store information. Specify the location of the database in the configuration file. The examples use a MongoDB database on the controller node:

```
# apt-get install mongodb
```

3. Configure MongoDB to make it listen on the controller public IP address. Edit the `/etc/mongodb.conf` file and modify the `bind_ip` key:

```
bind_ip = 192.168.0.10
```

4. Restart the MongoDB service to apply the configuration change:

```
# service mongodb restart
```

5. Create the database and a `ceilometer` database user:

```
# mongo  
> use ceilometer  
> db.addUser( { user: "ceilometer",  
               pwd: "CEILOMETER_DBPASS",  
               roles: [ "readWrite", "dbAdmin" ]  
             } )
```

6. Configure the Telemetry Service to use the database:

Edit the `/etc/ceilometer/ceilometer.conf` file and change the `[database]` section:

```
...
[database]
...
# The SQLAlchemy connection string used to connect to the
# database (string value)
connection = mongodb://ceilometer:CEILOMETER_DBPASS@controller:27017/
ceilometer
...
```

7. You must define a secret key that is used as a shared secret among Telemetry Service nodes. Use **openssl** to generate a random token and store it in the configuration file:

```
# openssl rand -hex 10
```

Edit the `/etc/ceilometer/ceilometer.conf` file and change the `[publisher_rpc]` section. Replace `ADMIN_TOKEN` with the results of the `openssl` command:

```
...
[publisher_rpc]
...
# Secret value for signing metering messages (string value)
metering_secret = ADMIN_TOKEN
...
```

8. Configure the RabbitMQ access:

Edit the `/etc/ceilometer/ceilometer.conf` file and update the `[DEFAULT]` section.

```
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

9. Create a `ceilometer` user that the Telemetry Service uses to authenticate with the Identity Service. Use the `service` tenant and give the user the `admin` role:

```
# keystone user-create --name=ceilometer --pass=CEILOMETER_PASS --
email=ceilometer@example.com
# keystone user-role-add --user=ceilometer --tenant=service --role=admin
```

10. Add the credentials to the configuration files for the Telemetry Service:

Edit the `/etc/ceilometer/ceilometer.conf` file and change the `[keystone_authtoken]` section:

```
[keystone_authtoken]
auth_host = controller
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = ceilometer
admin_password = CEILOMETER_PASS
```

11. Register the Telemetry Service with the Identity Service so that other OpenStack services can locate it. Use the **keystone** command to register the service and specify the endpoint:

```
# keystone service-create --name=ceilometer --type=metering \
```

```
--description="Ceilometer Telemetry Service"
```

- Note the `id` property that is returned for the service. Use it when you create the endpoint:

```
# keystone endpoint-create \  
--service-id=the_service_id_above \  
--publicurl=http://controller:8777/ \  
--internalurl=http://controller:8777/ \  
--adminurl=http://controller:8777/
```

- Restart the services with their new settings:

```
# service ceilometer-agent-central restart  
# service ceilometer-api restart  
# service ceilometer-collector restart
```

Install the Compute agent for the Telemetry service

OpenStack Telemetry is an API service that provides a collector and a range of disparate agents. This procedure details how to install the agent that runs on the compute node.

- Install the Telemetry service on the Compute node:

```
# apt-get install ceilometer-agent-compute
```

- Edit the `/etc/nova/nova.conf` file and add the following lines to the `[DEFAULT]` section:

```
...  
[DEFAULT]  
...  
instance_usage_audit=True  
instance_usage_audit_period=hour  
notify_on_state_change=vm_and_task_state  
notification_driver=nova.openstack.common.notifier.rpc_notifier  
notification_driver=ceilometer.compute.nova_notifier
```

- You must set the secret key that you defined previously. The Telemetry service nodes share this key as a shared secret:

Edit the `/etc/ceilometer/ceilometer.conf` file and change these lines in the `[DEFAULT]` section. Replace `ADMIN_TOKEN` with the admin token that you created previously:

```
...  
[publisher_rpc]  
# Secret value for signing metering messages (string value)  
metering_secret = ADMIN_TOKEN  
...
```

- Restart the service with its new settings:

```
# service ceilometer-agent-compute restart
```

Add the Image Service agent for the Telemetry service

1. To retrieve image samples, you must configure the Image Service to send notifications to the bus.

Edit `/etc/glance/glance-api.conf` and modify the `[DEFAULT]` section:

```
notifier_strategy = rabbit
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

2. Restart the Image services with their new settings:

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

Add the Block Storage Service agent for the Telemetry service

1. To retrieve volume samples, you must configure the Block Storage Service to send notifications to the bus.

Edit `/etc/cinder/cinder.conf` and add in the `[DEFAULT]` section:

```
control_exchange = cinder
notification_driver = cinder.openstack.common.notifier.rpc_notifier
```

2. Restart the Block Storage services with their new settings:

```
# service cinder-volume restart
# service cinder-api restart
```

Add the Object Storage agent for the Telemetry service

1. To retrieve object store statistics, the Telemetry service needs access to Object Storage with the `ResellerAdmin` role. Give this role to your `os_username` user for the `os_tenant_name` tenant:

```
$ keystone role-create --name=ResellerAdmin
+-----+-----+
| Property |          Value          |
+-----+-----+
|   id    | 462fa46c13fd4798a95a3bfbe27b5e54 |
|   name  |          ResellerAdmin          |
+-----+-----+

$ keystone user-role-add --tenant service --user ceilometer \
  --role 462fa46c13fd4798a95a3bfbe27b5e54
```

2. You must also add the Telemetry middleware to Object Storage to handle incoming and outgoing traffic. Add these lines to the `/etc/swift/proxy-server.conf` file:

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

3. Add `ceilometer` to the `pipeline` parameter of that same file:

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

4. Restart the service with its new settings:

```
# service swift-proxy-server restart
```

Verify the Telemetry Service installation

To test the Telemetry Service installation, download an image from the Image Service, and use the Telemetry Service to display usage statistics.

1. Use the `ceilometer meter-list` command to test the access to the Telemetry Service:

```
$ ceilometer meter-list
```

```
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| Name          | Type  | Unit  | Resource ID          | User
ID | Project ID          |
+-----+-----+-----+-----+
| image         | gauge | image | 9e5c2bee-0373-414c-b4af-b91b0246ad3b | None
| e66d97ac1b704897853412fc8450f7b9 |
| image.size    | gauge | B     | 9e5c2bee-0373-414c-b4af-b91b0246ad3b | None
| e66d97ac1b704897853412fc8450f7b9 |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

2. Download an image from the Image Service:

```
$ glance image-download "Cirros 0.3.1" > cirros.img
```

3. Call the `ceilometer meter-list` command again to validate that the download has been detected and stored by the Telemetry Service:

```
$ ceilometer meter-list
```

```
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| Name          | Type  | Unit  | Resource ID          |
User ID | Project ID          |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| image         | gauge | image | 9e5c2bee-0373-414c-b4af-b91b0246ad3b |
None   | e66d97ac1b704897853412fc8450f7b9 |
| image.download | delta | B     | 9e5c2bee-0373-414c-b4af-b91b0246ad3b |
None   | e66d97ac1b704897853412fc8450f7b9 |
| image.serve    | delta | B     | 9e5c2bee-0373-414c-b4af-b91b0246ad3b |
None   | e66d97ac1b704897853412fc8450f7b9 |
| image.size     | gauge | B     | 9e5c2bee-0373-414c-b4af-b91b0246ad3b |
None   | e66d97ac1b704897853412fc8450f7b9 |
+-----+-----+-----+-----+
```

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

4. You can now get usage statistics for the various meters:

```
$ ceilometer statistics -m image.download -p 60
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Period | Period Start          | Period End          | Count | Min
| Max    | Sum                   | Avg                 | Duration | Duration Start
|        | Duration End         |                     |          |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 60     | 2013-11-18T18:08:50 | 2013-11-18T18:09:50 | 1      | 13147648.0
| 13147648.0 | 13147648.0 | 13147648.0 | 0.0    | 2013-11-18T18:09:05.
334000 | 2013-11-18T18:09:05.334000 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

Appendix A. Community support

Table of Contents

Documentation	119
ask.openstack.org	120
OpenStack mailing lists	120
The OpenStack wiki	120
The Launchpad Bugs area	121
The OpenStack IRC channel	121
Documentation feedback	122
OpenStack distribution packages	122

To help you run and use OpenStack, many resources are available. Many OpenStack community members can answer questions and help with bug suspicions. We are constantly improving and adding to the main features of OpenStack, but if you have any problems, do not hesitate to ask. Use the following resources to get OpenStack support and troubleshoot your existing 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 components:

- [Installation Guide for Debian 7.0](#)
- [Installation Guide for openSUSE and SUSE Linux Enterprise Server](#)
- [Installation Guide for Red Hat Enterprise Linux, CentOS, and Fedora](#)
- [Installation Guide for Ubuntu 12.04 \(LTS\)](#)

The following books explain how to configure and run an OpenStack cloud:

- [Cloud Administrator Guide](#)
- [Configuration Reference](#)
- [Operations Guide](#)
- [High Availability Guide](#)
- [Security Guide](#)
- [Virtual Machine Image Guide](#)

The following books explain how to use the OpenStack dashboard and command-line clients:

- [API Quick Start](#)
- [End User Guide](#)
- [Admin User Guide](#)

The following documentation provides reference and guidance information for the OpenStack APIs:

- [OpenStack API Reference](#)
- [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](#)

ask.openstack.org

During set up or testing, you might have questions about how to do something 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 was already 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, link to screen shots, and so on.

OpenStack mailing lists

A great way to get answers and insights is to post your question or scenario to the OpenStack mailing list. You can learn from and help others who might have the same scenario as you. 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 content on a broad range of topics but some of it sits a bit below the surface. 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 lots of content. 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

So you think you've found a bug. That's great! Seriously, it is. 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 was already reported (or even better, already 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, link to screen shots, and so on.
- Be sure to include the software version that you are using, especially if you are using a development branch, such as, "Grizzly release" vs `git commit bc79c3ecc55929bac585d04a03475b72e06a3208`.
- Any deployment specific information is helpful, such as Ubuntu 12.04 or multi-node install.

The Launchpad Bugs areas are available here:

- [Bugs: OpenStack Compute \(nova\)](#)
- [Bugs : OpenStack Object Storage \(swift\)](#)
- [Bugs : OpenStack Image Service \(glance\)](#)
- [Bugs : OpenStack Identity \(keystone\)](#)
- [Bugs : OpenStack Dashboard \(horizon\)](#)
- [Bugs : OpenStack Networking \(neutron\)](#)
- [Bugs : OpenStack Orchestration \(heat\)](#)
- [Bugs : OpenStack Telemetry \(ceilometer\)](#)

The OpenStack IRC channel

The OpenStack community lives and breathes 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 you can paste into the channel. The OpenStack IRC channel is: #openstack on `irc.freenode.net`. You can find a list of all OpenStack-related 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>