

OpenStack Installation Guide

for Red Hat Enterprise Linux, CentOS, and Fedora

Grizzly, 2013.1 with Object Storage 1.8.0 (Jun 5, 2013)



OpenStack Installation Guide for Red Hat Enterprise Linux, CentOS, and Fedora

Grizzly, 2013.1 with Object Storage 1.8.0 (2013-06-05)

Copyright © 2012, 2013 OpenStack Foundation All rights reserved.

The OpenStack™ system has several key projects that are separate installations but can work together depending on your cloud needs: OpenStack Compute, OpenStack Object Storage, OpenStack Identity Service, and OpenStack Image Service. You can install any of these projects separately and then configure them either as standalone or connected entities. This guide walks through an installation using packages available through Fedora 17 as well as on RHEL and derivatives through the EPEL repository. It offers explanations for the configuration choices as well as sample configuration files.

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

1. Installing OpenStack Walk-through	1
Scripted Development Installation	2
Example Installation Architectures	2
Service Architecture	4
Compute and Image System Requirements	4
Compute Network Planning	6
Installing Network Time Protocol (NTP)	6
Installing MySQL	7
Installing A Messaging Server	7
2. OpenStack Terminology	9
Version Names and Release Notes	9
Code Names	9
OpenStack Services and Linux Services	10
Storage: objects, blocks, and files	10
Object storage	10
Block storage (SAN)	11
File storage (NAS)	11
3. Underlying Technologies	12
4. Installation Assumptions	15
Co-locating services	16
5. Installing OpenStack Identity Service	17
Basic Concepts	17
User management	19
Service management	23
Installing and Configuring the Identity Service	23
Configuring Services to work with Keystone	25
Defining Services	28
Troubleshooting the Identity Service (Keystone)	33
Verifying the Identity Service Installation	34
6. Installing OpenStack Image Service	36
Installing and Configuring the Image Service	36
Configuring the Image Service database backend	36
Edit the Glance configuration files	37
Troubleshooting the Image Service (Glance)	39
Verifying the Image Service Installation	39
7. Installing OpenStack Compute Service	41
Configuring the Hypervisor	41
KVM	41
Checking for hardware virtualization support	42
Enabling KVM	43
Specifying the CPU model of KVM guests	44
Troubleshooting	45
QEMU	45
Tips and fixes for QEMU on RHEL	46
Xen, XenAPI, XenServer and XCP	46
Xen terminology	47
XenAPI deployment architecture	48
XenAPI pools	49

Installing XenServer and XCP	49
Xen Boot from ISO	53
Further reading	53
Pre-configuring the network	54
Configuration requirements with RHEL	54
Configuring the SQL Database (MySQL) on the Cloud Controller	55
Configuring the SQL Database (PostgreSQL) on the Cloud Controller	56
Installing and configuring Block Storage (Cinder)	57
Installing Compute Services	58
File format for nova.conf	59
Configuring OpenStack Compute	61
Configuring the Database for Compute	63
Creating the Network for Compute VMs	64
Verifying the Compute Installation	64
Defining Compute and Image Service Credentials	65
Installing Additional Compute Nodes	66
Adding Block Storage Nodes	66
8. Registering Virtual Machine Images	68
9. Running Virtual Machine Instances	71
Security groups: Enabling SSH and ICMP (ping)	71
Adding a keypair	71
Confirm all services running	72
Starting an instance	73
Bringing down an instance	76
10. Installing OpenStack Object Storage	77
System Requirements	77
Object Storage Network Planning	78
Example Object Storage Installation Architecture	78
Installing OpenStack Object Storage on Ubuntu	79
Before You Begin	79
General Installation Steps	80
Installing and Configuring the Storage Nodes	80
Installing and Configuring the Proxy Node	82
Start the Storage Nodes Services	85
OpenStack Object Storage Post Installation	85
Verify the Installation	85
Adding an Additional Proxy Server	86
11. Installing the OpenStack Dashboard	88
About the Dashboard	88
System Requirements for the Dashboard	88
Installing the OpenStack Dashboard	89
Configuring the Dashboard	90
Validating the Dashboard Install	93
How To Custom Brand The OpenStack Dashboard (Horizon)	94
OpenStack Dashboard Session Storage	96
Local Memory Cache	97
Memcached	97
Database	97
Cached Database	98
Cookies	99
Overview of VNC Proxy	99

About nova-consoleauth	100
Typical Deployment	100
Frequently asked questions about VNC access to VMs	103
A. Configuration File Examples	105
keystone.conf	105
glance-registry.conf	107
glance-registry-paste.ini	109
glance-api.conf	109
glance-api-paste.ini	115
glance-scrubber.conf	116
nova.conf	117
api-paste.ini	118
Credentials (openrc)	120
cinder.conf	121
Dashboard configuration	121
etc/swift/swift.conf	123
etc/network/interfaces.conf	123
etc/swift/proxy-server.conf	124
etc/swift/account-server.conf	125
etc/swift/account-server/1.conf	125
etc/swift/container-server.conf	125
etc/swift/container-server/1.conf	126
etc/swift/object-server.conf	126
etc/swift/object-server/1.conf	126

List of Figures

3.1. Underlying technologies (Scenario 1)	12
3.2. Underlying technologies (Scenario 2)	13
11.1. NoVNC Process	100

List of Tables

1.1. Hardware Recommendations	5
2.1. OpenStack version names	9
2.2. Code names	9
2.3. Code names	10
3.1. Technologies and supported implementations	13
10.1. Hardware Recommendations	77
11.1. Description of nova.conf file configuration options for VNC access to guest instances	101

1. Installing OpenStack Walk-through

Table of Contents

Scripted Development Installation	2
Example Installation Architectures	2
Service Architecture	4
Compute and Image System Requirements	4
Compute Network Planning	6
Installing Network Time Protocol (NTP)	6
Installing MySQL	7
Installing A Messaging Server	7

The OpenStack Compute and Image services work together to provide access to virtual servers and images through REST APIs. The Identity Service provides a common authorization layer for all OpenStack services. You must use the Identity Service to install the OpenStack Dashboard, which offers a web-based user interface for OpenStack components. The OpenStack Object Storage service provides not only a storage method for virtual images but also a cloud-based object storage system with a REST API to store and retrieve objects such as images or videos. This walk-through starts with Identity, then goes through Image and Compute and also provides deployment information about an Object Storage installation.

If you are interested in how to plan for and operate an OpenStack cloud, refer to the [OpenStack Operations Guide](#).

Here are the overall steps for a manual install:

1. Review the most supported platforms. Red Hat Enterprise Linux, Scientific Linux, CentOS, Fedora, Debian, and Ubuntu are the most tested platforms currently.
2. Install the Identity Service (Keystone).
3. Configure the Identity Service.
4. Install the Image Service (Glance).
5. Configure the Image Service.
6. Install Compute (Nova).
7. Review the assumptions made in this installation for Compute.
8. Configure Compute with FlatDHCP networking using `192.168.100.0/24` as the fixed range for our guest VMs on a bridge named `br100`.
9. Create and initialize the Compute database with MySQL. PostgreSQL is also documented but all examples follow MySQL as an assumed default.
10. Add images.

-
- 11.(optional) Install OpenStack Object Storage (Swift).
 - 12.Install the OpenStack Dashboard.
 - 13.Launch the Dashboard.
 - 14.Add a keypair through the Dashboard.
 - 15.Launch an image through the Dashboard to verify the entire installation.

Scripted Development Installation

You can download a script for a standalone install for proof-of-concept, learning, or for development purposes for Ubuntu 12.04 or Fedora 16 at <https://devstack.org>.

1. Install Ubuntu 12.04 or Fedora 16:

In order to correctly install all the dependencies, we assume a specific version of the OS to make it as easy as possible.

2. Download DevStack:

```
$ git clone git://github.com/openstack-dev/devstack.git
```

The devstack repository contains a script that installs OpenStack Compute, Object Storage, the Image Service, Volumes, the Dashboard and the Identity Service and offers templates for configuration files plus data scripts.

3. Start the install:

```
$ cd devstack; ./stack.sh
```

It takes a few minutes, we recommend [reading the well-documented script](#) while it is building to learn more about what is going on.

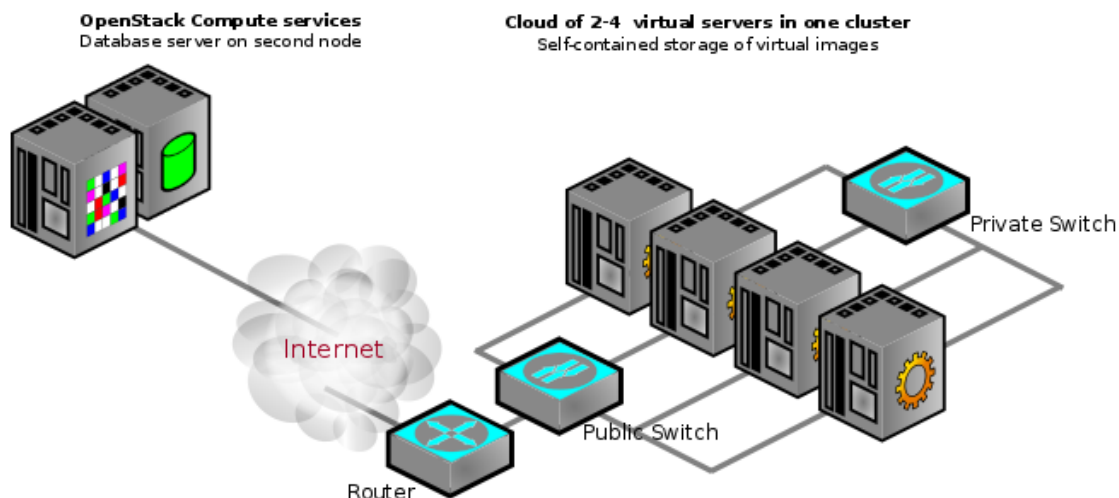
Example Installation Architectures

OpenStack Compute uses a shared-nothing, messaging-based architecture. While very flexible, the fact that you can install each nova- service on an independent server means there are many possible methods for installing OpenStack Compute. Here are the types of installation architectures:

- Single node: Only one server runs all nova- services and also drives all the virtual instances. Use this configuration only for trying out OpenStack Compute, or for development purposes.
- Two nodes: A cloud controller node runs the nova- services except for **nova-compute**, and a compute node runs **nova-compute**. A client computer is likely needed to bundle images and interfacing to the servers, but a client is not required. Use this configuration for proof of concepts or development environments.
- Multiple nodes: You can add more compute nodes to the two node installation by simply installing **nova-compute** on an additional server and copying a `nova.conf` file to the

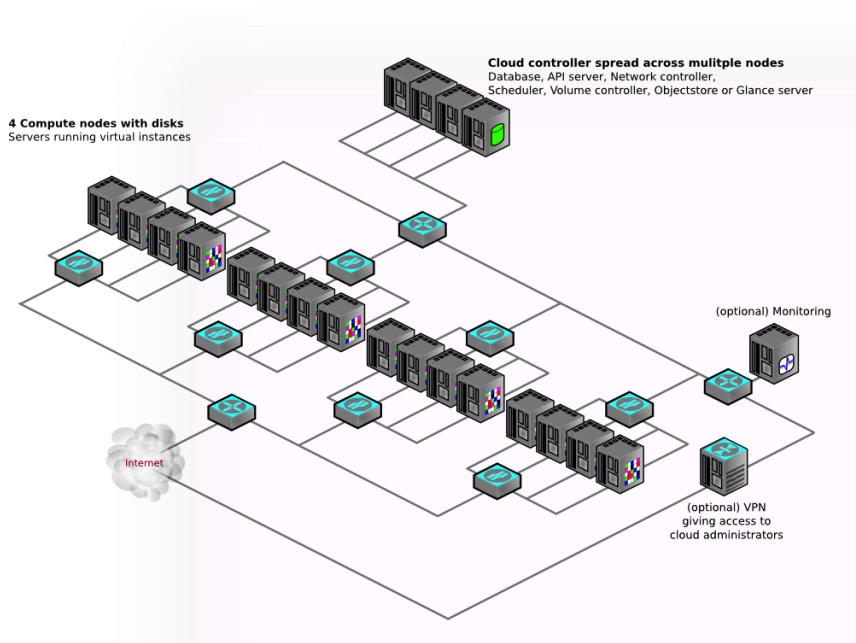
added node. This would result in a multiple node installation. You can also add a volume controller and a network controller as additional nodes in a more complex multiple node installation. A minimum of 4 nodes is best for running multiple virtual instances that require a lot of processing power.

This is an illustration of one possible multiple server installation of OpenStack Compute; virtual server networking in the cluster may vary.



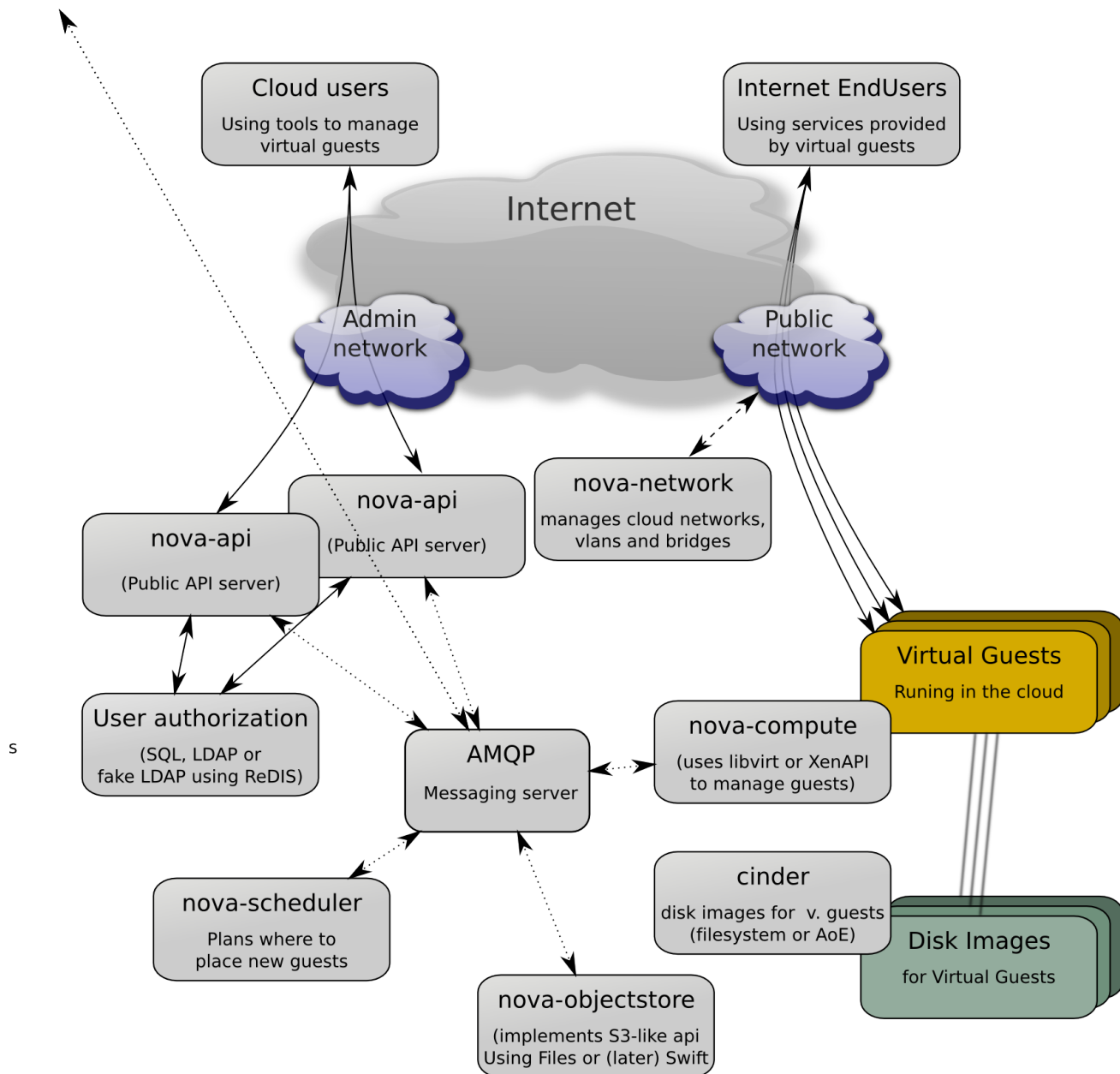
An alternative architecture would be to add more messaging servers if you notice a lot of back up in the messaging queue causing performance problems. In that case you would add an additional messaging server in addition to or instead of scaling up the database server. Your installation can run any nova- service on any server as long as the `nova.conf` is configured to point to the messaging server and the server can send messages to the server.

Multiple installation architectures are possible, here is another example illustration.



Service Architecture

Because Compute has multiple services and many configurations are possible, here is a diagram showing the overall service architecture and communication systems between the services.



Compute and Image System Requirements

Hardware: OpenStack components are intended to run on standard hardware. Recommended hardware configurations for a minimum production deployment are as

follows for the cloud controller nodes and compute nodes for Compute and the Image Service, and object, account, container, and proxy servers for Object Storage.

Table 1.1. Hardware Recommendations

Server	Recommended Hardware	Notes
Cloud Controller node (runs network, volume, API, scheduler and image services)	<p>Processor: 64-bit x86</p> <p>Memory: 12 GB RAM</p> <p>Disk space: 30 GB (SATA, SAS or SSD)</p> <p>Volume storage: two disks with 2 TB (SATA) for volumes attached to the compute nodes</p> <p>Network: one 1 Gbps Network Interface Card (NIC)</p>	Two NICs are recommended but not required. A quad core server with 12 GB RAM would be more than sufficient for a cloud controller node.
Compute nodes (runs virtual instances)	<p>Processor: 64-bit x86</p> <p>Memory: 32 GB RAM</p> <p>Disk space: 30 GB (SATA)</p> <p>Network: two 1 Gbps NICs</p>	<p>With 2 GB RAM you can run one m1.small instance on a node or three m1.tiny instances without memory swapping, so 2 GB RAM would be a minimum for a test-environment compute node. As an example, Rackspace Cloud Builders use 96 GB RAM for compute nodes in OpenStack deployments.</p> <p>Specifically for virtualization on certain hypervisors on the node or nodes running nova-compute, you need a x86 machine with an AMD processor with SVM extensions (also called AMD-V) or an Intel processor with VT (virtualization technology) extensions.</p> <p>For XenServer and XCP refer to the XenServer installation guide and the XenServer hardware compatibility list.</p> <p>For LXC, the VT extensions are not required.</p>



Note

While certain parts of OpenStack are known to work on various operating systems, currently the only feature-complete, production-supported host environment is 64-bit Linux.

Operating System: OpenStack currently has packages for the following distributions: CentOS, Debian, Fedora, RHEL, openSUSE, SLES, and Ubuntu. These packages are maintained by community members, refer to <http://wiki.openstack.org/Packaging> for additional links.



Note

The Grizzly release of OpenStack Compute requires Fedora 16 or later.

Database: For OpenStack Compute, you need access to either a PostgreSQL or MySQL database, or you can install it as part of the OpenStack Compute installation process.

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

Network Time Protocol: You must install a time synchronization program such as NTP. For Compute, time synchronization avoids problems when scheduling VM launches on

compute nodes. For Object Storage, time synchronization ensure the object replications are accurately updating objects when needed so that the freshest content is served.

Compute Network Planning

For both conserving network resources and ensuring that network administrators understand the needs for networks and public IP addresses for accessing the APIs and VMs as necessary, this section offers recommendations and required minimum sizes. Throughput of at least 1000 Mbps is suggested. This walkthrough shows network configurations for a single server.

For OpenStack Compute, networking is configured on multi-node installations between the physical machines on a single subnet. For networking between virtual machine instances, three network options are available: flat, DHCP, and VLAN. Two NICs (Network Interface Cards) are recommended on the server running nova-network.

Management Network (RFC1918 IP Range, not publicly routable): This network is utilized for all inter-server communications within the cloud infrastructure. Recommended size: 255 IPs (CIDR /24)

Public Network (Publicly routable IP range): This network is utilized for providing Public IP accessibility to the API endpoints within the cloud infrastructure. Minimum size: 8 IPs (CIDR /29)

VM Network (RFC1918 IP Range, not publicly routable): This network is utilized for providing primary IP addresses to the cloud instances. Recommended size: 1024 IPs (CIDR /22)

Floating IP network (Publicly routable IP Range): This network is utilized for providing Public IP accessibility to selected cloud instances. Minimum size: 16 IPs (CIDR /28)

Installing Network Time Protocol (NTP)

To keep all the services in sync across multiple machines, you need to install NTP, and if you do a multi-node configuration you will configure one server to be the reference server.

```
$ sudo yum install -y ntp
```

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

```
$ sudo service ntpd start  
$ sudo chkconfig ntpd on
```

Set up the NTP client on your compute node so that the time between controller node and compute node is synchronized. The simplest way to do this is to add a daily cron job that synchronizes the compute node's clock with the controller node. You can accomplish this by adding a file, owned by root, marked executable, at `/etc/cron.daily/ntpdate` that contains the following:

```
ntpdate <hostname or IP address of controller>  
hwclock -w
```

If a large clock skew builds up between the compute nodes and the controller node, then the time change that occurs when the `ntpd` cron job runs may confuse some programs running on the compute nodes. To allow more gradual time updates, install the NTP package on the compute nodes in addition to the API nodes.

Installing MySQL

Install MySQL as root:

```
# yum install mysql mysql-server MySQL-python
```

During the install, you'll be prompted for the `mysql` root password. Enter a password of your choice and verify it.

Set MySQL to start the daemon every time you start and then start the server.

```
# chkconfig --level 2345 mysqld on
# service mysqld start
```

Installing A Messaging Server

Install the messaging queue server. Typically this is Qpid but RabbitMQ and ZeroMQ (0MQ) are also available.

1. All steps in this procedure require the privileges of the `root` user. If you are not logged in as the `root` user then either escalate your privileges using the `su` command or prefix each command with `sudo`, depending on the configuration of your system.
2. Install the Qpid message broker and supporting utilities that will be needed in subsequent procedures.

```
#yum install qpid-cpp-server memcached openstack-utils
```

3. Disable Qpid authentication by setting the value of the `auth` configuration key to `no` in the `/etc/qpid.conf` file.

```
# echo "auth=no" >> /etc/qpid.conf
```

4. Start the Qpid message broker.

```
# service qpid start
```

5. Configure the Qpid message broker to start automatically in future.

```
# chkconfig qpid on
```

6. Configure Nova to use the Qpid message broker for remote procedure calls by setting the value of the `rpc_backend` configuration key in `/etc/nova/nova.conf`.

```
# openstack-config --set /etc/nova/nova.conf \
    DEFAULT rpc_backend nova.rpc.impl_qpid
```

-
7. Configure Nova to use the Qpid message broker by setting the value of the `qpid_hostname` configuration key in `/etc/nova/nova.conf`.

```
# openstack-config --set /etc/nova/nova.conf \  
    DEFAULT qpid_hostname 127.0.0.1
```

Replace `127.0.0.1` with the IP address or host name of your message broker.

2. OpenStack Terminology

Table of Contents

Version Names and Release Notes	9
Code Names	9
OpenStack Services and Linux Services	10
Storage: objects, blocks, and files	10
Object storage	10
Block storage (SAN)	11
File storage (NAS)	11

Version Names and Release Notes

Each OpenStack release has a name, in increasing alphabetical order (e.g., Havana follows Grizzly). There are also version numbers corresponding to these releases, as shown in the table below. Click on a release name in the table (e.g., Grizzly) for the release notes, which are hosted on the OpenStack wiki.

Table 2.1. OpenStack version names

Release name	Release date	OpenStack version number for Block Storage, Compute, Identity, Image, and Networking	OpenStack Object Storage version number
Havana	October 2013	2013.3	unknown
Grizzly	April 2013	2013.1	1.7.6
Folsom	October 2012	2012.2	1.7.2
Essex	April 2012	2012.1	1.4.8
Diablo	October 2011	2011.3	1.4.3
Cactus	April 2011	2011.2	1.3.0
Bexar	March 2011	2011.1	1.2.0
Austin	October 2010	0.9.0	1.0.0

Beginning with the Cactus release, OpenStack adopted a six month release schedule. The Havana release is scheduled for October 2013.

Code Names

Each OpenStack service has a code name. For example, the Image Service is code-named Glance. The full list is shown in the table below:

Table 2.2. Code names

Service name	Code name
Identity	Keystone
Compute	Nova

Service name	Code name
Image	Glance
Dashboard	Horizon
Object Storage	Swift
Volumes	Cinder
Networking	Quantum

These code names are reflected in the names of configuration files and command-line utility programs. For example, the Identity service has a configuration file called `keystone.conf`.

In addition, projects can go through an incubation phase and become integrated with other OpenStack services that release as a group with integrated testing. Two projects went through that process and will be integrated with the Havana release:

Table 2.3. Code names

Service name	Code name
Metering	Ceilometer
Orchestration	Heat

OpenStack Services and Linux Services

In the Linux world, a service (also known as a daemon) refers to a single program that runs in the background and typically listens on a port to respond to service requests. An OpenStack service, on the other hand, refers to a collection of Linux services working in concert.

OpenStack services are implemented by multiple Linux services. For example, **nova-compute** and **nova-scheduler** are two of the Linux services that implement the Compute service. OpenStack also depends on several third-party services, such as a database (typically MySQL) and a message broker (typically RabbitMQ or Qpid).

In this document, we generally use the term "service" to refer both to lower-level Linux services and higher-level OpenStack services. It should be clear from the context whether we are referring to a high-level OpenStack service (e.g., Image), or a low-level Linux service (e.g., **glance-api**).

Storage: objects, blocks, and files

Many cloud computing use cases require persistent remote storage. Storage solutions are often divided into three categories: object storage, block storage, and file storage.

Note that some storage solutions support multiple categories. For example, [NexentaStor](#) supports both block storage and file storage (with announcements for future support for object storage), [GlusterFS](#) supports file storage and object storage, and [Ceph Storage](#) supports object storage, block storage, and file storage.

Object storage

In OpenStack: Object Storage service (Swift)

Related concepts: Amazon S3, Rackspace Cloud Files, Ceph Storage

With *object storage*, files are exposed through an HTTP interface, typically with a REST API. All client data access is done at the user level: the operating system is unaware of the presence of the remote storage system. In OpenStack, the Object Storage service provides this type of functionality. Users access and modify files by making HTTP requests. Because the data access interface provided by an object storage system is at a low level of abstraction, people often build on top of object storage to build file-based applications that provide a higher level of abstraction. For example, the OpenStack Image service can be configured to use the Object Storage service as a backend. Another use for object storage solutions is as a content delivery network (CDN) for hosting static web content (e.g., images, and media files), since object storage already provides an HTTP interface.

Block storage (SAN)

In OpenStack: Block Storage service (Cinder)

Related concepts: Amazon Elastic Block Store (EBS), Ceph RADOS Block Device (RBD), iSCSI

With *block storage*, files are exposed through a low-level computer bus interface such as SCSI or ATA, that is accessible over the network. Block storage is synonymous with SAN (storage area network). Clients access data through the operating system at the device level: users access the data by mounting the remote device in a similar manner to how they would mount a local, physical disk (e.g., using the "mount" command in Linux). In OpenStack, the `cinder-volume` service that forms part of the Compute service provides this type of functionality, and uses iSCSI to expose remote data as a SCSI disk that is attached to the network.

Because the data is exposed as a physical device, the end-user is responsible for creating partitions and formatting the exposed disk device. In addition, in OpenStack Compute a device can only be attached to one server at a time, so block storage cannot be used to share data across virtual machine instances concurrently.

File storage (NAS)

In OpenStack: none

Related concepts: NFS, Samba/CIFS, GlusterFS, Dropbox, Google Drive

With *file storage*, files are exposed through a distributed file system protocol. Filesystem storage is synonymous with NAS (network attached storage). Clients access data through the operating system at the file system level: users access the data by mounting a remote file system. Examples of file storage include NFS and GlusterFS. The operating system needs to have the appropriate client software installed to be able to access the remote file system.

Currently, OpenStack Compute does not have any native support for this type of file storage inside of an instance. However, there is a [Gluster storage connector for OpenStack](#) that enables the use of the GlusterFS file system as a back-end for the Image service.

3. Underlying Technologies

You can think of OpenStack Compute as a toolkit for building a cloud computing environment by stitching together existing Linux technologies.

The figures below shows two examples of how these underlying technologies can be assembled to construct an OpenStack Compute cloud. The circles are Linux services that are part of OpenStack Compute, and the rectangles are external (not maintained by the OpenStack project) components. Solid lines show interactions between OpenStack components and external components, and dashed lines show interactions between external components. All services that are part of OpenStack Compute interact with a queueing backend (e.g., RabbitMQ, Qpid, OMQ) and a database backend (e.g., MySQL, PostgreSQL); these connections are not shown. Also not shown are services that do not explicitly rely on external technologies. For example, the `nova-api` service, the Identity service, and the Image service are not shown.

Figure 3.1. Underlying technologies (Scenario 1)

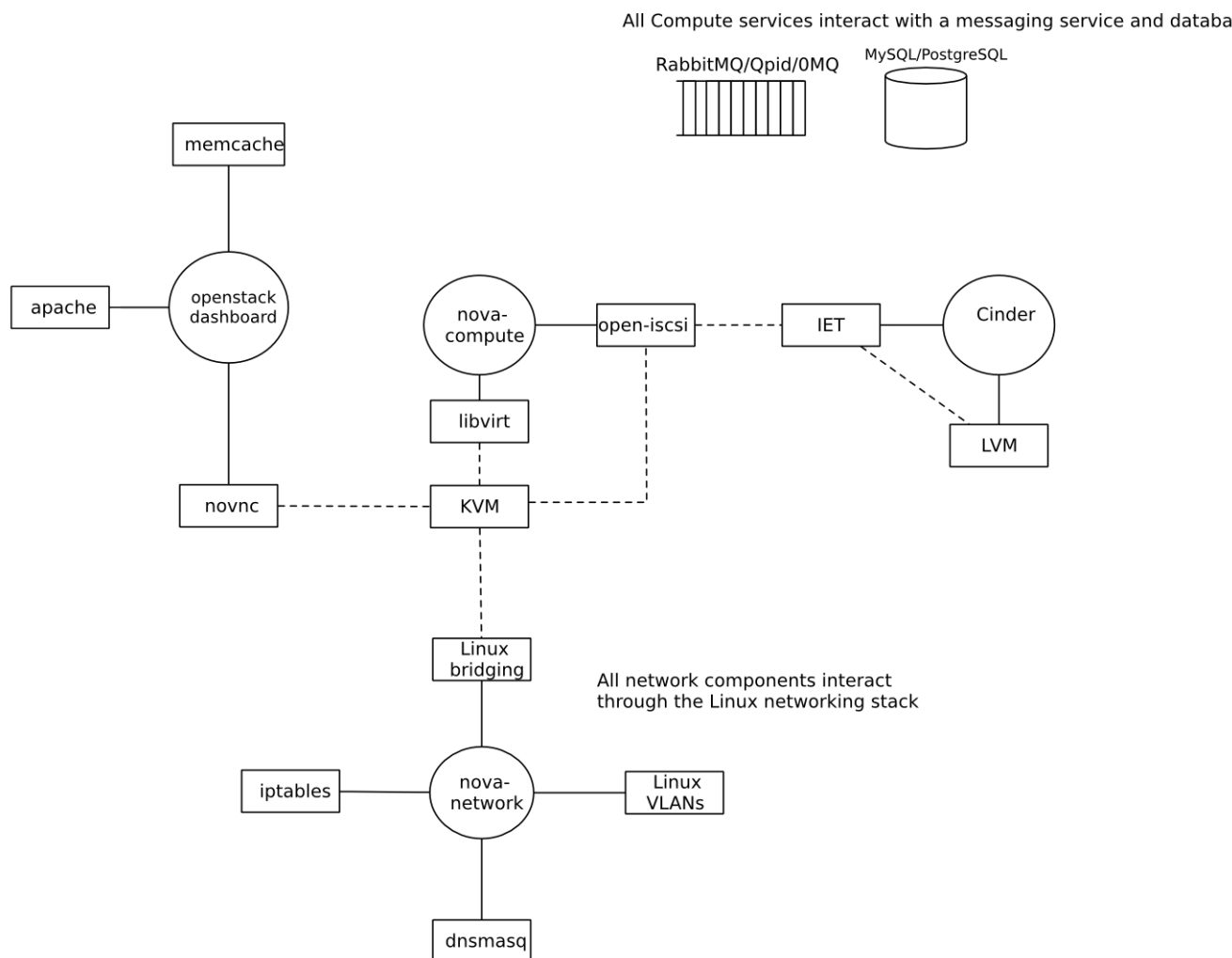
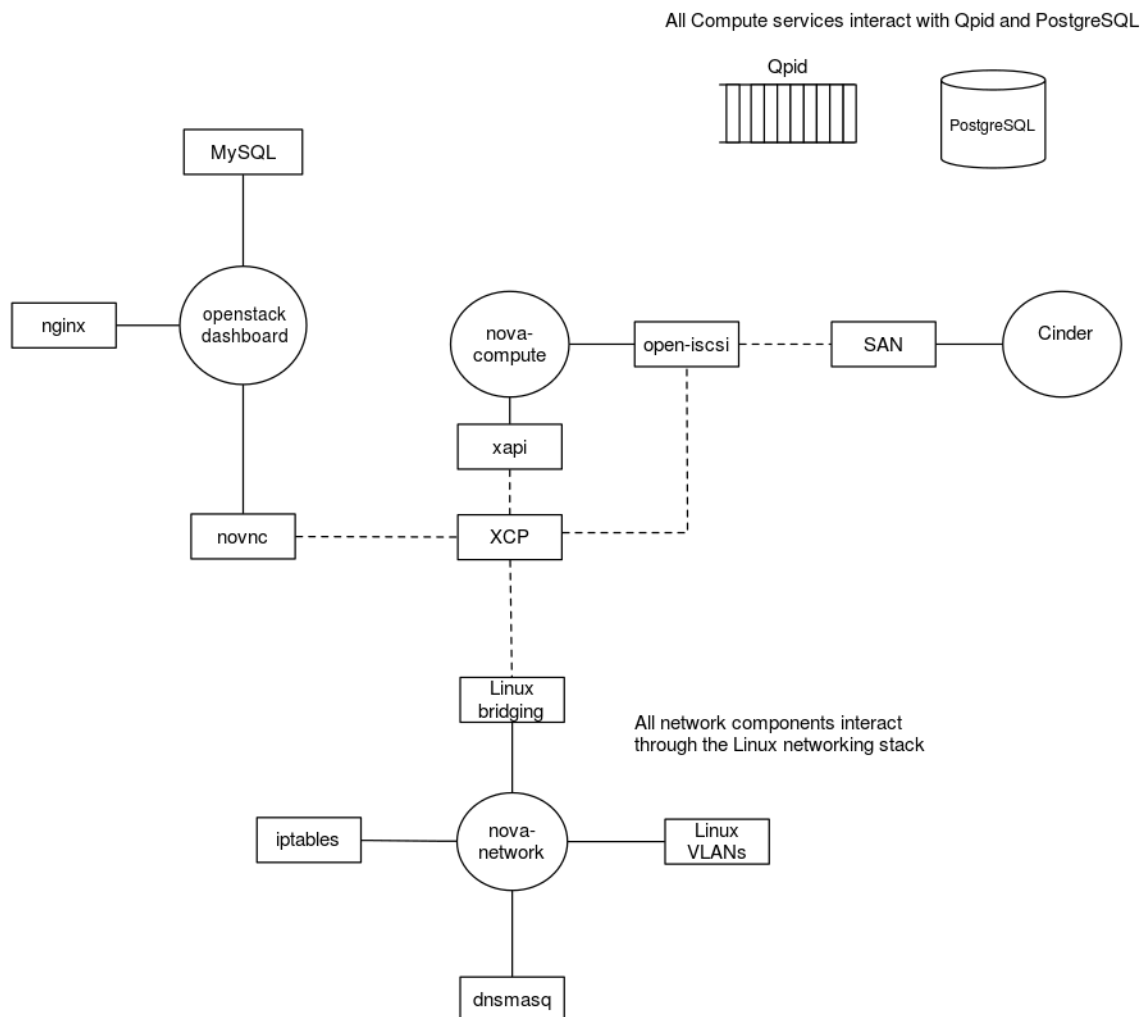


Figure 3.2. Underlying technologies (Scenario 2)



Many of the external technologies can be substituted with other components, as shown in the table below.

Table 3.1. Technologies and supported implementations

Technology	Supported implementations
Message queue	RabbitMQ, Qpid, ZeroMQ
Virtualization	xapi+XCP, xapi+XenServer, libvirt+KVM, libvirt+QEMU, libvirt+LXC, libvirt+VMWare
iSCSI back-end	LVM+IET, LVM+tgt, Xen Storage Manager, SAN (Solaris, HP, SolidFire), NexentaStor, NetApp, Ceph, Sheepdog
Database	MySQL, PostgreSQL, sqlite
Web server	Apache, Nginx
Session cache	memcache, any Django-supported database backend (e.g., MySQL, PostgreSQL, sqlite)

nova-compute

The `nova-compute` service depends on a virtualization driver to manage virtual machines. By default, this driver is *libvirt*, which is used to drive *KVM*. However, the *libvirt* driver can also drive other hypervisor technologies, and there is a separate Xen virtualization driver for driving Xen-based virtual machines if configured to use Xen Cloud Platform (XCP) or XenServer. *Open-iscsi* is used to mount remote block devices, also known as volumes. *Open-iscsi* exposes these remote devices as local device files which can be attached to instances.

nova-network

The `nova-network` service depends on a number of Linux networking technologies. It uses *Linux bridging* to create network bridges to connect virtual machines to the physical networks. These bridges may be associated with VLANs using *Linux networking VLAN support*, if running in the VLAN networking mode. *Iptables* is used to implement security rules and implement NAT functionality, which is used for providing instances with access to the metadata service and for supporting floating IP addresses. *Dnsmasq* is used as a DHCP server to hand out IP addresses to virtual machine instances, as well as a DNS server.

While `nova-network` continues to be available, it is not actively developed, as the OpenStack Networking (code-named Quantum) project is its replacement. Use one or the other in your installation.

OpenStack Networking (Quantum)

The OpenStack Networking service also depends on Linux networking technologies, using a plugin mechanism. Read more about it in the [OpenStack Networking Administration Guide](#).

OpenStack Block Storage (Cinder)

By default, `Cinder` service uses *LVM* to create and manage local volumes, and exports them via iSCSI using *IET* or *tgt*. It can also be configured to use other iSCSI-based storage technologies.

openstack-dashboard (Horizon)

The `openstack-dashboard` is a Django-based application that runs behind an *Apache* web server by default. It uses *memcache* for the session cache by default. A web-based VNC client called *novnc* is used to provide access to the VNC consoles associated with the running *KVM* instances.

4. Installation Assumptions

Table of Contents

Co-locating services	16
----------------------------	----

OpenStack Compute has a large number of configuration options. To simplify this installation guide, we make a number of assumptions about the target installation. If you want a longer conceptual planning guide that discusses considerations of these decisions, refer to the [OpenStack Operations Guide](#).

- You have a collection of compute nodes, each installed with Fedora 18, RHEL 6.4, Scientific Linux 6.1 or CentOS 6 + CR distributions (continuous release (CR) repository).



Note

There is also an [OpenStack Install and Deploy Manual for Ubuntu](#) Debian, openSUSE, and SLES also have OpenStack support, but are not documented here.

- You have designated one of the nodes as the Cloud Controller, which will run all of the services (RabbitMQ or Qpid, MySQL, Identity, Image, nova-api, nova-network, nova-scheduler, nova-volume, nova-conductor) except for nova-compute and possibly networking services depending on the configuration.
- The disk partitions on your cloud controller are being managed by the [Logical Volume Manager](#) (LVM).
- Your Cloud Controller has an LVM volume group named "cinder-volumes" to provide persistent storage to guest VMs. Either create this during the installation or leave some free space to create it prior to installing nova services.
- Ensure that the server can resolve its own hostname, otherwise you may have problems if you are using RabbitMQ as the messaging backend. Qpid is the default messaging backend on Fedora.
- 192.168.206.130 is the primary IP for our host on eth0.
- 192.168.100.0/24 as the fixed range for our guest VMs, connected to the host via br100.
- FlatDHCP with a single network interface.
- KVM or Xen (XenServer or XCP) as the hypervisor.
- Ensure the operating system is up-to-date by running **yum update** prior to the installation.
- On RHEL (and derivatives) enable this testing repo for grizzly.

```
$ wget http://repos.fedorapeople.org/repos/openstack/openstack-grizzly/epel-  
openstack-grizzly.repo
```

This installation process walks through installing a cloud controller node and a compute node using a set of packages that are known to work with each other. The cloud controller node contains all the nova- services including the API server and the database server. The compute node needs to run only the nova-compute service. You only need one nova-network service running in a multi-node install, though if high availability for networks is required, there are additional options.

Co-locating services

While for performance reasons you may want OpenStack project's services to live on different machines, this is not always practical. For example, in small deployments there might be too few machines available, or a limited number of public IP addresses. Components from different OpenStack projects are not necessarily engineered to be able to be co-located, however many users report success with a variety of deployment scenarios.

The following is a series of pointers to be used when co-location of services from different OpenStack projects on the same machine is a must:

- Ensure dependencies aren't in conflict. The OpenStack Continuous Integration team does attempt to ensure there is no conflict - so if you see issues during package installation, consider filing a bug.
- Monitor your systems and ensure they are not overloaded. Some parts of OpenStack use a lot of CPU time (such as Object Storage Proxy Servers), while others are I/O focused (such as Object Storage Object Servers). Try to balance these so they complement each other.
- Beware of security. Different parts of OpenStack assume different security models. For example, OpenStack Object Storage (Swift) assumes the storage nodes will be on a private network and does not provide additional security between nodes in the cluster.
- Ensure the ports you are running the services on don't conflict. Most ports used by OpenStack are configurable.

5. Installing OpenStack Identity Service

Table of Contents

Basic Concepts	17
User management	19
Service management	23
Installing and Configuring the Identity Service	23
Configuring Services to work with Keystone	25
Defining Services	28
Troubleshooting the Identity Service (Keystone)	33
Verifying the Identity Service Installation	34

The OpenStack Identity service manages users, tenants (accounts or projects) and offers a common identity system for all the OpenStack components.

Basic Concepts

The Identity service has two primary functions:

1. User management: keep track of users and what they are permitted to do
2. Service catalog: Provide a catalog of what services are available and where their API endpoints are located

The Identity Service has several definitions which are important to understand.

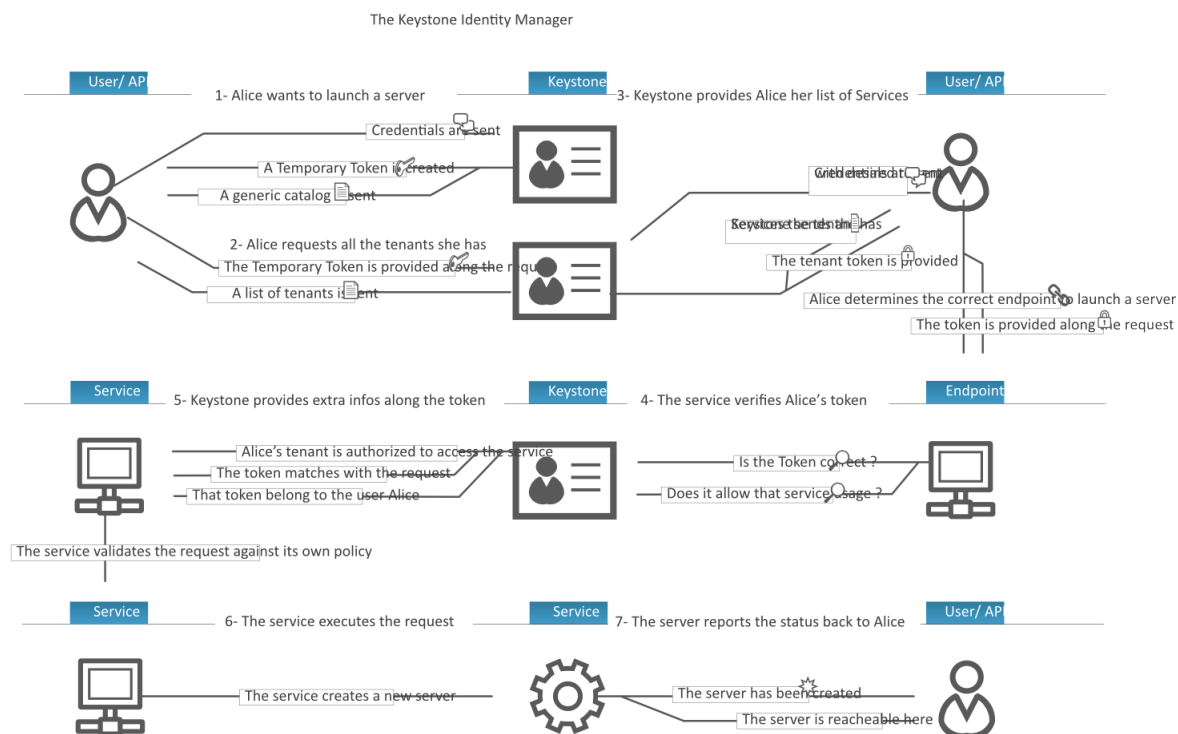
User	A digital representation of a person, system, or service who uses OpenStack cloud services. Identity authentication services will validate that incoming request are being made by the user who claims to be making the call. Users have a login and may be assigned tokens to access resources. Users may be directly assigned to a particular tenant and behave as if they are contained in that tenant.
------	--

Credentials	Data that belongs to, is owned by, and generally only known by a user that the user can present to prove they are who they are (since nobody else should know that data).
-------------	---

Examples are:

- a matching username and password
- a matching username and API key
- yourself and a driver's license with a picture of you
- a token that was issued to you that nobody else knows of

Authentication	<p>In the context of the identity service, authentication is the act of confirming the identity of a user or the truth of a claim. The identity service will confirm that incoming request are being made by the user who claims to be making the call by validating a set of claims that the user is making. These claims are initially in the form of a set of credentials (username & password, or username and API key). After initial confirmation, the identity service will issue the user a token which the user can then provide to demonstrate that their identity has been authenticated when making subsequent requests.</p>
Token	<p>A token is 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 anytime and is valid for a finite duration.</p> <p>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.</p>
Tenant	<p>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.</p>
Service	<p>An OpenStack service, such as Compute (Nova), Object Storage (Swift), or Image Service (Glance). A service provides one or more endpoints through which users can access resources and perform (presumably useful) operations.</p>
Endpoint	<p>An network-accessible address, usually described by URL, where a service may be accessed. 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.</p>
Role	<p>A personality that a user assumes when performing a specific set of operations. A role includes a set of rights and privileges. A user assuming that role inherits those rights and privileges.</p> <p>In the identity service, a token that is issued to a user includes the list of roles that user can assume. Services that are being called by that user determine how they interpret the set of roles a user has and which operations or resources each roles grants access to.</p>



User management

The three main concepts of Identity user management are:

- Users
- Tenants
- Roles

A *user* represents a human user, and has associated information such as username, password and email. This example creates a user named "alice":

```
$ keystone user-create --name=alice --pass=mypassword123 --email=alice@example.com
```

A *tenant* can be thought of as a project, group, or organization. Whenever you make requests to OpenStack services, you must specify a tenant. For example, if you query the Compute service for a list of running instances, you will receive a list of all of the running instances in the tenant you specified in your query. This example creates a tenant named "acme":

```
$ keystone tenant-create --name=acme
```



Note

Because the term *project* was used instead of *tenant* in earlier versions of OpenStack Compute, some command-line tools use `--project_id` instead of `--tenant-id` or `--os-tenant-id` to refer to a tenant ID.

A *role* captures what operations a user is permitted to perform in a given tenant. This example creates a role named "compute-user":

```
$ keystone role-create --name=compute-user
```



Note

It is up to individual services such as the Compute service and Image service to assign meaning to these roles. As far as the Identity service is concerned, a role is simply a name.

The Identity service associates a user with a tenant and a role. To continue with our previous examples, we may wish to assign the "alice" user the "compute-user" role in the "acme" tenant:

```
$ keystone user-list
```

id	enabled	email	name
892585	True	alice@example.com	alice

```
$ keystone role-list
```

id	name
9a764e	compute-user

```
$ keystone tenant-list
```

id	name	enabled
6b8fd2	acme	True

```
$ keystone user-role-add --user=892585 --role=9a764e --tenant-id=6b8fd2
```

A user can be assigned different roles in different tenants: for example, Alice may also have the "admin" role in the "Cyberdyne" tenant. A user can also be assigned multiple roles in the same tenant.

The `/etc/[SERVICE_CODENAME]/policy.json` controls what users are allowed to do for a given service. For example, `/etc/nova/policy.json` specifies the access policy for the Compute service, `/etc/glance/policy.json` specifies the access policy for the Image service, and `/etc/keystone/policy.json` specifies the access policy for the Identity service.

The default `policy.json` files in the Compute, Identity, and Image service recognize only the `admin` role: all operations that do not require the `admin` role will be accessible by any user that has any role in a tenant.

If you wish to restrict users from performing operations in, say, the Compute service, you need to create a role in the Identity service and then modify `/etc/nova/policy.json` so that this role is required for Compute operations.

For example, this line in `/etc/nova/policy.json` specifies that there are no restrictions on which users can create volumes: if the user has any role in a tenant, they will be able to create volumes in that tenant.

```
"volume:create": [],
```

If we wished to restrict creation of volumes to users who had the `compute-user` role in a particular tenant, we would add `"role:compute-user"`, like so:

```
"volume:create": ["role:compute-user"],
```

If we wished to restrict all Compute service requests to require this role, the resulting file would look like:

```
{
  "admin_or_owner": [["role:admin"], ["project_id:
%(project_id)s"]],
  "default": [["rule:admin_or_owner"]],

  "compute:create": ["role:compute-user"],
  "compute:create:attach_network": ["role:compute-user"],
  "compute:create:attach_volume": ["role:compute-user"],
  "compute:get_all": ["role:compute-user"],

  "admin_api": [["rule:admin"]],
  "compute_extension:accounts": [["rule:admin_api"]],
  "compute_extension:admin_actions": [["rule:admin_api"]],
  "compute_extension:admin_actions:pause":
[["rule:admin_or_owner"]],
  "compute_extension:admin_actions:unpause":
[["rule:admin_or_owner"]],
  "compute_extension:admin_actions:suspend":
[["rule:admin_or_owner"]],
  "compute_extension:admin_actions:resume":
[["rule:admin_or_owner"]],
  "compute_extension:admin_actions:lock": [["rule:admin_api"]],
  "compute_extension:admin_actions:unlock":
[["rule:admin_api"]],
  "compute_extension:admin_actions:resetNetwork":
[["rule:admin_api"]],
  "compute_extension:admin_actions:injectNetworkInfo":
[["rule:admin_api"]],
  "compute_extension:admin_actions:createBackup":
[["rule:admin_or_owner"]],
  "compute_extension:admin_actions:migrateLive":
[["rule:admin_api"]],
  "compute_extension:admin_actions:migrate":
[["rule:admin_api"]],
  "compute_extension:aggregates": [["rule:admin_api"]],
  "compute_extension:certificates": ["role:compute-user"],
  "compute_extension:cloudpipe": [["rule:admin_api"]],
```

```

        "compute_extension:console_output": ["role":"compute-user"],
        "compute_extension:consoles": ["role":"compute-user"],
        "compute_extension:createserverext": ["role":"compute-user"],
        "compute_extension:deferred_delete": ["role":"compute-user"],
        "compute_extension:disk_config": ["role":"compute-user"],
        "compute_extension:evacuate": [{"rule:admin_api"}],
        "compute_extension:extended_server_attributes":
[["rule:admin_api"]],
        "compute_extension:extended_status": ["role":"compute-user"],
        "compute_extension:flavorextradata": ["role":"compute-user"],
        "compute_extension:flavorextraspecs": ["role":"compute-user"],
        "compute_extension:flavormanage": [{"rule:admin_api"}],
        "compute_extension:floating_ip_dns": ["role":"compute-user"],
        "compute_extension:floating_ip_pools": ["role":"compute-
user"],
        "compute_extension:floating_ips": ["role":"compute-user"],
        "compute_extension:hosts": [{"rule:admin_api"}],
        "compute_extension:keypairs": ["role":"compute-user"],
        "compute_extension:multinic": ["role":"compute-user"],
        "compute_extension:networks": [{"rule:admin_api"}],
        "compute_extension:quotas": ["role":"compute-user"],
        "compute_extension:rescue": ["role":"compute-user"],
        "compute_extension:security_groups": ["role":"compute-user"],
        "compute_extension:server_action_list": [{"rule:admin_api"}],
        "compute_extension:server_diagnostics": [{"rule:admin_api"}],
        "compute_extension:simple_tenant_usage:show":
[["rule:admin_or_owner"]],
        "compute_extension:simple_tenant_usage:list":
[["rule:admin_api"]],
        "compute_extension:users": [{"rule:admin_api"}],
        "compute_extension:virtual_interfaces": ["role":"compute-
user"],
        "compute_extension:virtual_storage_arrays": ["role":"compute-
user"],
        "compute_extension:volumes": ["role":"compute-user"],
        "compute_extension:volumetypes": ["role":"compute-user"],

        "volume:create": ["role":"compute-user"],
        "volume:get_all": ["role":"compute-user"],
        "volume:get_volume_metadata": ["role":"compute-user"],
        "volume:get_snapshot": ["role":"compute-user"],
        "volume:get_all_snapshots": ["role":"compute-user"],

        "network:get_all_networks": ["role":"compute-user"],
        "network:get_network": ["role":"compute-user"],
        "network:delete_network": ["role":"compute-user"],
        "network:disassociate_network": ["role":"compute-user"],
        "network:get_vifs_by_instance": ["role":"compute-user"],
        "network:allocate_for_instance": ["role":"compute-user"],
        "network:deallocate_for_instance": ["role":"compute-user"],
        "network:validate_networks": ["role":"compute-user"],
        "network:get_instance_uuids_by_ip_filter": ["role":"compute-
user"],

        "network:get_floating_ip": ["role":"compute-user"],
        "network:get_floating_ip_pools": ["role":"compute-user"],
        "network:get_floating_ip_by_address": ["role":"compute-user"],
        "network:get_floating_ips_by_project": ["role":"compute-
user"],

```

```
"network:get_floating_ips_by_fixed_address": ["role":"compute-  
user"],  
    "network:allocate_floating_ip": ["role":"compute-user"],  
    "network:deallocate_floating_ip": ["role":"compute-user"],  
    "network:associate_floating_ip": ["role":"compute-user"],  
    "network:disassociate_floating_ip": ["role":"compute-user"],  
  
    "network:get_fixed_ip": ["role":"compute-user"],  
    "network:add_fixed_ip_to_instance": ["role":"compute-user"],  
    "network:remove_fixed_ip_from_instance": ["role":"compute-  
user"],  
  
    "network:add_network_to_project": ["role":"compute-user"],  
    "network:get_instance_nw_info": ["role":"compute-user"],  
  
    "network:get_dns_domains": ["role":"compute-user"],  
    "network:add_dns_entry": ["role":"compute-user"],  
    "network:modify_dns_entry": ["role":"compute-user"],  
    "network:delete_dns_entry": ["role":"compute-user"],  
    "network:get_dns_entries_by_address": ["role":"compute-user"],  
    "network:get_dns_entries_by_name": ["role":"compute-user"],  
    "network:create_private_dns_domain": ["role":"compute-user"],  
    "network:create_public_dns_domain": ["role":"compute-user"],  
    "network:delete_dns_domain": ["role":"compute-user"]  
}
```

Service management

The two main concepts of Identity service management are:

- Services
- Endpoints

The Identity service also maintains a user that corresponds to each service (e.g., a user named *nova*, for the Compute service) and a special service tenant, which is called *service*.

The commands for creating services and endpoints are described in a later section.

Installing and Configuring the Identity Service

Install the Identity service on any server that is accessible to the other servers you intend to use for OpenStack services, as root:

```
$ yum install openstack-utils openstack-keystone python-keystoneclient
```

After installing, you need to delete the sqlite database it creates, then change the configuration to point to a MySQL database. This configuration enables easier scaling scenarios since you can bring up multiple Keystone front ends when needed, and configure them all to point back to the same database. Plus a database backend has built-in data replication features and documentation surrounding high availability and data redundancy configurations.

Delete the `keystone.db` file created in the `/var/lib/keystone` directory.

```
$ sudo rm /var/lib/keystone/keystone.db
```

Configure the production-ready backend data store rather than using the catalog supplied by default for the ability to backup the service and endpoint data. This example shows MySQL.

The following sequence of commands will create a MySQL database named "keystone" and a MySQL user named "keystone" with full access to the "keystone" MySQL database.

On Fedora, RHEL, and CentOS, you can configure the Keystone database with the **openstack-db** command.

```
$ sudo openstack-db --init --service keystone
```

To manually create the database, start the **mysql** command line client by running:

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

Enter the mysql root user's password when prompted.

To configure the MySQL database, create the keystone database.

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

Create a MySQL user for the newly-created keystone database that has full control of the keystone database.



Note

Choose a secure password for the keystone user and replace all references to `[YOUR_KEYSTONEDB_PASSWORD]` with this password.

```
mysql> GRANT ALL ON keystone.* TO 'keystone'@'%' IDENTIFIED BY  
'[YOUR_KEYSTONEDB_PASSWORD]';  
mysql> GRANT ALL ON keystone.* TO 'keystone'@'localhost' IDENTIFIED BY  
'[YOUR_KEYSTONEDB_PASSWORD]';
```



Note

In the above commands, even though the `'keystone'@'%'` also matches `'keystone'@'localhost'`, you must explicitly specify the `'keystone'@'localhost'` entry.

By default, MySQL will create entries in the user table with `User=''` and `Host='localhost'`. The `User=''` acts as a wildcard, matching all users. If you do not have the `'keystone'@'localhost'` account, and you try to log in as the keystone user, the precedence rules of MySQL will match against the `User='' Host='localhost'` account before it matches against the `User='keystone' Host='%'` account. This will result in an error message that looks like:

```
ERROR 1045 (28000): Access denied for user  
'keystone'@'localhost' (using password: YES)
```

Thus, we create a separate `User='keystone' Host='localhost'` entry that will match with higher precedence.

See the [MySQL documentation on connection verification](#) for more details on how MySQL determines which row in the user table it uses when authenticating connections.

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

```
mysql> quit
```



Reminder

Recall that this document assumes the Cloud Controller node has an IP address of 192.168.206.130.

Once Keystone is installed, it is configured via a primary configuration file (`/etc/keystone/keystone.conf`), and by initializing data into keystone using the command line client. By default, Keystone's data store is `sqlite`. To change the data store to `mysql`, change the line defining connection in `/etc/keystone/keystone.conf` like so:

```
connection = mysql://keystone:[YOUR_KEYSTONEDB_PASSWORD]@192.168.206.130/keystone
```

Also, ensure that the proper *service token* is used in the `keystone.conf` file. An example is provided in the Appendix or you can generate a random string. The sample token is:

```
admin_token = 012345SECRET99TOKEN012345
```

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

By default Keystone will use `ssl` encryption between it and all of the other services. To create the encryption certificates run:

```
# keystone-manage pki_setup
# chown -R keystone:keystone /etc/keystone/*
```

Next, restart the keystone service so that it picks up the new database configuration.

```
$ sudo service openstack-keystone start && sudo chkconfig openstack-keystone on
```

Lastly, initialize the new keystone database, as root:

```
# keystone-manage db_sync
```

Configuring Services to work with Keystone

Once Keystone is installed and running, you set up users and tenants and services to be configured to work with it. You can either follow the [manual steps](#) or [use a script](#).

Setting up tenants, users, and roles - manually

You need to minimally define a tenant, user, and role to link the tenant and user as the most basic set of details to get other services authenticating and authorizing with the Identity service.



Scripted method available

These are the manual, unscripted steps using the keystone client. A scripted method is available at [Setting up tenants, users, and roles - scripted](#).

Typically, you would use a username and password to authenticate with the Identity service. However, at this point in the install, we have not yet created a user. Instead, we use the service token to authenticate against the Identity service. With the **keystone** command-line, you can specify the token and the endpoint as arguments, as follows:

```
$ keystone --token 012345SECRET99TOKEN012345 --endpoint http://192.168.206.130:35357/v2.0 <command parameters>
```

You can also specify the token and endpoint as environment variables, so they do not need to be explicitly specified each time. If you are using the bash shell, the following commands will set these variables in your current session so you don't have to pass them to the client each time.

```
$ export OS_SERVICE_TOKEN=012345SECRET99TOKEN012345
$ export OS_SERVICE_ENDPOINT=http://192.168.206.130:35357/v2.0
```

In the remaining examples, we will assume you have set the above environment variables.

Because it is more secure to use a username and password to authenticate rather than the service token, when you use the token the **keystone** client may output the following warning, depending on the version of python-keystoneclient you are running:

```
WARNING: Bypassing authentication using a token & endpoint (authentication
credentials are being ignored).
```

First, create a default tenant, we'll name it `demo` in this example.

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

+-----+-----+
| Property | Value |
+-----+-----+
| description | Default Tenant |
| enabled | True |
| id | b5815b046cfe47bb891a7b64119e7f80 |
| name | demo |
+-----+-----+
```

Create a default user named `admin`.

```
$ keystone user-create --tenant-id b5815b046cfe47bb891a7b64119e7f80 --name
admin --pass secrete

+-----+-----+
| Property | Value |
+-----+-----+
| email | |
| enabled | True |
| id | a4c2d43f80a549a19864c89d759bb3fe |
| name | admin |
| tenantId | b5815b046cfe47bb891a7b64119e7f80 |
+-----+-----+
```

Create an administrative role based on keystone's default `policy.json` file, `admin`.

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

Property	Value
id	e3d9d157cc95410ea45d23bbbc2e5c10
name	admin

Grant the admin role to the admin user in the demo tenant with "user-role-add".

```
$ keystone user-role-add --user-id a4c2d43f80a549a19864c89d759bb3fe --tenant-id b5815b046cfe47bb891a7b64119e7f80 --role-id e3d9d157cc95410ea45d23bbbc2e5c10
```

Create a Service Tenant. This tenant contains all the services that we make known to the service catalog.

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

Property	Value
description	Service Tenant
enabled	True
id	eb7e0c10a99446cfa14c244374549e9d
name	service

Create a Glance Service User in the Service Tenant. You'll do this for any service you add to be in the Keystone service catalog.

```
$ keystone user-create --tenant-id eb7e0c10a99446cfa14c244374549e9d --name glance --pass glance
```

WARNING: Bypassing authentication using a token & endpoint (authentication credentials are being ignored).

Property	Value
email	
enabled	True
id	46b2667a7807483d983e0b4037a1623b
name	glance
tenantId	eb7e0c10a99446cfa14c244374549e9d

Grant the admin role to the glance user in the service tenant.

```
$ keystone user-role-add --user-id 46b2667a7807483d983e0b4037a1623b --tenant-id eb7e0c10a99446cfa14c244374549e9d --role-id e3d9d157cc95410ea45d23bbbc2e5c10
```

Create a Nova Service User in the Service Tenant.

```
$ keystone user-create --tenant-id eb7e0c10a99446cfa14c244374549e9d --name nova --pass nova
```

WARNING: Bypassing authentication using a token & endpoint (authentication credentials are being ignored).

Property	Value
email	
enabled	True
id	54b3776a8707834d983e0b4037b1345c
name	nova

```
| tenantId | eb7e0c10a99446cfa14c244374549e9d |  
+-----+-----+
```

Grant the admin role to the nova user in the service tenant.

```
$ keystone user-role-add --user-id 54b3776a8707834d983e0b4037b1345c --tenant-  
id eb7e0c10a99446cfa14c244374549e9d --role-id e3d9d157cc95410ea45d23bbbc2e5c10
```

Create an EC2 Service User in the Service Tenant.

```
$ keystone user-create --tenant-id eb7e0c10a99446cfa14c244374549e9d --name ec2  
--pass ec2
```

```
+-----+-----+  
| Property | Value |  
+-----+-----+  
| email |  
| enabled | True |  
| id | 32e7668b8707834d983e0b4037b1345c |  
| name | ec2 |  
| tenantId | eb7e0c10a99446cfa14c244374549e9d |  
+-----+-----+
```

Grant the admin role to the ec2 user in the service tenant.

```
$ keystone user-role-add --user-id 32e7668b8707834d983e0b4037b1345c --tenant-  
id eb7e0c10a99446cfa14c244374549e9d --role-id e3d9d157cc95410ea45d23bbbc2e5c10
```

Create an Object Storage Service User in the Service Tenant.

```
$ keystone user-create --tenant-id eb7e0c10a99446cfa14c244374549e9d --name  
swift --pass swiftpass
```

```
+-----+-----+  
| Property | Value |  
+-----+-----+  
| email |  
| enabled | True |  
| id | 4346677b8909823e389f0b4037b1246e |  
| name | swift |  
| tenantId | eb7e0c10a99446cfa14c244374549e9d |  
+-----+-----+
```

Grant the admin role to the swift user in the service tenant.

```
$ keystone user-role-add --user-id 4346677b8909823e389f0b4037b1246e --tenant-  
id eb7e0c10a99446cfa14c244374549e9d --role-id e3d9d157cc95410ea45d23bbbc2e5c10
```

Next you create definitions for the services.

Defining Services

Keystone also acts as a service catalog to let other OpenStack systems know where relevant API endpoints exist for OpenStack Services. The OpenStack Dashboard, in particular, uses the service catalog heavily - and this **must** be configured for the OpenStack Dashboard to properly function.

There are two alternative ways of defining services with keystone:

1. Using a template file

2. Using a database backend

While using a template file is simpler, it is not recommended except for development environments such as [DevStack](#). The template file does not enable CRUD operations on the service catalog through keystone commands, but you can use the `service-list` command when using the template catalog. A database backend can provide better reliability, availability, and data redundancy. This section describes how to populate the Keystone service catalog using the database backend. Your `/etc/keystone/keystone.conf` file should contain the following lines if it is properly configured to use the database backend.

```
[catalog]
driver = keystone.catalog.backends.sql.Catalog
```

Elements of a Keystone service catalog entry

For each service in the catalog, you must perform two keystone operations:

1. Use the **keystone service-create** command to create a database entry for the service, with the following attributes:

<code>--name</code>	Name of the service (e.g., nova, ec2, glance, keystone)
<code>--type</code>	Type of service (e.g., compute, ec2, image, identity)
<code>--description</code>	A description of the service, (e.g., "Nova Compute Service")

2. Use the **keystone endpoint-create** command to create a database entry that describes how different types of clients can connect to the service, with the following attributes:

<code>--region</code>	the region name you've given to the OpenStack cloud you are deploying (e.g., RegionOne)
<code>--service-id</code>	The ID field returned by the keystone service-create (e.g., 935fd37b6fa74b2f9fba6d907fa95825)
<code>--publicurl</code>	The URL of the public-facing endpoint for the service (e.g., <code>http://192.168.206.130:9292</code> or <code>http://192.168.206.130:8774/v2/\$(tenant_id)s</code>)
<code>--internalurl</code>	The URL of an internal-facing endpoint for the service. This typically has the same value as <code>publicurl</code> .
<code>--adminurl</code>	The URL for the admin endpoint for the service. The Keystone and EC2 services use different endpoints for <code>adminurl</code> and <code>publicurl</code> , but for other services these endpoints will be the same.

Keystone allows some URLs to contain special variables, which are automatically substituted with the correct value at runtime. Some examples in this document employ the `tenant_id` variable, which we use when specifying the Volume and Compute service endpoints. Variables can be specified using either `%(varname)s` or `$(varname)s` notation. In this document, we always use the `%(varname)s` notation (e.g., `%(tenant_id)s`) since `$` is interpreted as a special character by Unix shells.

Creating keystone services and service endpoints

Here we define the services and their endpoints. Recall that you must have the following environment variables set.

```
$ export OS_SERVICE_TOKEN=012345SECRET99TOKEN012345
$ export OS_SERVICE_ENDPOINT=http://192.168.206.130:35357/v2.0
```

Define the Identity service:

```
$ keystone service-create --name=keystone --type=identity --description=
"Identity Service"
```

Property	Value
description	Identity Service
id	15c11a23667e427e91bc31335b45f4bd
name	keystone
type	identity

```
$ keystone endpoint-create \
--region RegionOne \
--service-id=15c11a23667e427e91bc31335b45f4bd \
--publicurl=http://192.168.206.130:5000/v2.0 \
--internalurl=http://192.168.206.130:5000/v2.0 \
--adminurl=http://192.168.206.130:35357/v2.0
```

Property	Value
adminurl	http://192.168.206.130:35357/v2.0
id	11f9c625a3b94a3f8e66bf4e5de2679f
internalurl	http://192.168.206.130:5000/v2.0
publicurl	http://192.168.206.130:5000/v2.0
region	RegionOne
service_id	15c11a23667e427e91bc31335b45f4bd

Define the Compute service, which requires a separate endpoint for each tenant. Here we use the `service` tenant from the previous section.



Note

The `%(tenant_id)s` and single quotes around the `publicurl`, `internalurl`, and `adminurl` must be typed exactly as shown for both the Compute endpoint and the Volume endpoint.

```
$ keystone service-create --name=nova --type=compute --description="Compute
Service"
```

Property	Value
description	Compute Service
id	abc0f03c02904c24abdcc3b7910e2eed
name	nova
type	compute

```
$ keystone endpoint-create \  
--region RegionOne \  
--service-id=abc0f03c02904c24abdcc3b7910e2eed \  
--publicurl='http://192.168.206.130:8774/v2/%(tenant_id)s' \  
--internalurl='http://192.168.206.130:8774/v2/%(tenant_id)s' \  
--adminurl='http://192.168.206.130:8774/v2/%(tenant_id)s'
```

Property	Value
adminurl	http://192.168.206.130:8774/v2/%(tenant_id)s
id	935fd37b6fa74b2f9fba6d907fa95825
internalurl	http://192.168.206.130:8774/v2/%(tenant_id)s
publicurl	http://192.168.206.130:8774/v2/%(tenant_id)s
region	RegionOne
service_id	abc0f03c02904c24abdcc3b7910e2eed

Define the Volume service, which also requires a separate endpoint for each tenant.

```
$ keystone service-create --name=volume --type=volume --description="Volume  
Service"
```

Property	Value
description	Volume Service
id	1ff4ecel3c3e48d8a6461faebd9cd38f
name	volume
type	volume

```
$ keystone endpoint-create \  
--region RegionOne \  
--service-id=1ff4ecel3c3e48d8a6461faebd9cd38f \  
--publicurl='http://192.168.206.130:8776/v1/%(tenant_id)s' \  
--internalurl='http://192.168.206.130:8776/v1/%(tenant_id)s' \  
--adminurl='http://192.168.206.130:8776/v1/%(tenant_id)s'
```

Property	Value
adminurl	http://192.168.206.130:8776/v1/%(tenant_id)s
id	1ff4ecel3c3e48d8a6461faebd9cd38f
internalurl	http://192.168.206.130:8776/v1/%(tenant_id)s
publicurl	http://192.168.206.130:8776/v1/%(tenant_id)s
region	RegionOne
service_id	8a70cd235c7d4a05b43b2dffb9942cc0

Define the Image service:

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

```
+-----+
| Property | Value |
+-----+
| description | Image Service |
| id | 7d5258c490144c8c92505267785327c1 |
| name | glance |
| type | image |
+-----+

$ keystone --token 012345SECRET99TOKEN012345 \
--endpoint http://192.168.206.130:35357/v2.0/ \
endpoint-create \
--region RegionOne \
--service-id=7d5258c490144c8c92505267785327c1 \
--publicurl=http://192.168.206.130:9292 \
--internalurl=http://192.168.206.130:9292 \
--adminurl=http://192.168.206.130:9292

+-----+
| Property | Value |
+-----+
| adminurl | http://192.168.206.130:9292 |
| id | 3c8c0d749f21490b90163bfaed9befef7 |
| internalurl | http://192.168.206.130:9292 |
| publicurl | http://192.168.206.130:9292 |
| region | RegionOne |
| service_id | 7d5258c490144c8c92505267785327c1 |
+-----+
```

Define the EC2 compatibility service:

```
$ keystone service-create --name=ec2 --type=ec2 --description="EC2
Compatibility Layer"

+-----+
| Property | Value |
+-----+
| description | EC2 Compatibility Layer |
| id | 181cdad1d1264387bcc411e1c6a6a5fd |
| name | ec2 |
| type | ec2 |
+-----+

$ keystone --token 012345SECRET99TOKEN012345 \
--endpoint http://192.168.206.130:35357/v2.0/ \
endpoint-create \
--region RegionOne \
--service-id=181cdad1d1264387bcc411e1c6a6a5fd \
--publicurl=http://192.168.206.130:8773/services/Cloud \
--internalurl=http://192.168.206.130:8773/services/Cloud \
--adminurl=http://192.168.206.130:8773/services/Admin

+-----+
| Property | Value |
+-----+
```

adminurl	http://192.168.206.130:8773/services/Admin
id	d2a3d7490c61442f9b2c8c8a2083c4b6
internalurl	http://192.168.206.130:8773/services/Cloud
publicurl	http://192.168.206.130:8773/services/Cloud
region	RegionOne
service_id	181cdad1d1264387bcc411e1c6a6a5fd

Define the Object Storage service:

```
$ keystone service-create --name=swift --type=object-store --description="Object Storage Service"
```

Property	Value
description	Object Storage Service
id	272efad2d1234376cbb911c1e5a5a6ed
name	swift
type	object-store

```
$ keystone endpoint-create \
  --region RegionOne \
  --service-id=272efad2d1234376cbb911c1e5a5a6ed \
  --publicurl 'http://192.168.206.130:8888/v1/AUTH_%(tenant_id)s' \
  --internalurl 'http://192.168.206.130:8888/v1/AUTH_%(tenant_id)s' \
  --adminurl 'http://192.168.206.130:8888/v1'
```

Property	Value
adminurl	http://192.168.206.130:8888/v1
id	e32b3c4780e51332f9c128a8c208a5a4
internalurl	http://192.168.206.130:8888/v1/AUTH_%(tenant_id)s
publicurl	http://192.168.206.130:8888/v1/AUTH_%(tenant_id)s
region	RegionOne
service_id	272efad2d1234376cbb911c1e5a5a6ed

Setting up Tenants, Users, Roles, and Services - Scripted

The Keystone project offers a bash script for populating tenants, users, roles and services at https://github.com/openstack/keystone/blob/master/tools/sample_data.sh with sample data. This script uses 127.0.0.1 for all endpoint IP addresses. This script also defines services for you.

Troubleshooting the Identity Service (Keystone)

To begin troubleshooting, look at the logs in the `/var/log/keystone.log` file (the location of log files is configured in the `/etc/keystone/logging.conf` file). It shows all the components that have come in to the WSGI request, and will ideally have an error in that log that explains why an authorization request failed. If you're not seeing the request at all in those logs, then run keystone with "`--debug`" where `--debug` is passed in directly after the CLI command prior to parameters.

Verifying the Identity Service Installation

Verify that authentication is behaving as expected by using your established username and password to generate an authentication token:

```
$ keystone --os-username=admin --os-password=secrete --os-auth-url=
http://192.168.206.130:35357/v2.0 token-get
```

Property	Value
expires	2012-10-04T16:08:03Z
id	960ad732a0eb4b2a88516f18384c1fba
user_id	a4c2d43f80a549a19864c89d759bb3fe

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=secrete --os-tenant-name=
demo --os-auth-url=http://192.168.206.130:35357/v2.0 token-get
```

Property	Value
expires	2012-10-04T16:10:14Z
id	8787f264d2a34607b37aa8d58d956afa
tenant_id	c1ac0f7f0e55448fa3940fa6b8b54911
user_id	a4c2d43f80a549a19864c89d759bb3fe

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 CLI usage. First, set up a `keystonerc` file with the admin credentials and admin endpoint:

```
export OS_USERNAME=admin
export OS_PASSWORD=secrete
export OS_TENANT_NAME=demo
export OS_AUTH_URL=http://192.168.206.130:35357/v2.0
```

Save and source the file.

```
$ source keystonerc
```

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

```
$ keystone token-get
```

Property	Value
expires	2012-10-04T16:12:38Z
id	03a13f424b56440fb39278b844a776ae
tenant_id	c1ac0f7f0e55448fa3940fa6b8b54911
user_id	a4c2d43f80a549a19864c89d759bb3fe

You should receive a new token in response, reflecting the same tenant and user ID values as above.

This verifies that you have configured your environment variables correctly.

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



Reminder

Unlike basic authentication/authorization, which can be performed against either port 5000 or 35357, administrative commands **MUST** be performed against the admin API port: 35357. This means that you **MUST** use port 35357 in your `OS_AUTH_URL` or `--os-auth-url` setting.

```
$ keystone user-list
```

id	enabled	email	name
318003c9a97342dbab6ff81675d68364	True	None	swift
3a316b32f44941c0b9ebc577feaa5b5c	True	None	nova
ac4dd12ebad84e55a1cd964b356ddf65	True	None	glance
a4c2d43f80a549a19864c89d759bb3fe	True	None	admin
ec47114af7014afd9a8994cbb6057a8b	True	None	ec2

This verifies that your user account has the `admin` role, as defined in keystone's `policy.json` file.

6. Installing OpenStack Image Service

Table of Contents

Installing and Configuring the Image Service	36
Configuring the Image Service database backend	36
Edit the Glance configuration files	37
Troubleshooting the Image Service (Glance)	39
Verifying the Image Service Installation	39

Installing and Configuring the Image Service

Install the Image service, as root:

```
# yum install openstack-glance
# rm /var/lib/glance/glance.sqlite
```

Configuring the Image Service database backend

Configure the backend data store. For MySQL, create a glance MySQL database and a glance MySQL user. Grant the "glance" user full access to the glance MySQL database.

Start the MySQL command line client by running:

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

Enter the MySQL root user's password when prompted.

To configure the MySQL database, create the glance database.

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

Create a MySQL user for the newly-created glance database that has full control of the database.

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



Note

In the above commands, even though the 'glance'@ '%' also matches 'glance'@'localhost', you must explicitly specify the 'glance'@'localhost' entry.

By default, MySQL will create entries in the user table with User= ' ' and Host= 'localhost'. The User= ' ' acts as a wildcard, matching all users. If you do not have the 'glance'@'localhost' account, and you try to log in as the glance user, the precedence rules of MySQL will match against

the `User='' Host='localhost'` account before it matches against the `User='glance' Host='%'` account. This will result in an error message that looks like:

```
ERROR 1045 (28000): Access denied for user  
'glance'@'localhost' (using password: YES)
```

Thus, we create a separate `User='glance' Host='localhost'` entry that will match with higher precedence.

See the [MySQL documentation on connection verification](#) for more details on how MySQL determines which row in the user table it uses when authenticating connections.

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

```
mysql> quit
```

Edit the Glance configuration files

The Image service has a number of options that you can use to configure the Glance API server, optionally the Glance Registry server, and the various storage backends that Glance can use to store images. By default, the storage backend is in file, specified in the `glance-api.conf` config file in the section `[DEFAULT]`.

The `glance-api` service implements versions 1 and 2 of the OpenStack Images API. By default, both are enabled by setting these configuration options to `True` in the `glance-api.conf` file.

```
enable_v1_api=True
```

```
enable_v2_api=True
```

Disable either version of the Images API by setting the option to `False` in the `glance-api.conf` file.



Note

In order to use the v2 API, you must copy the necessary SQL configuration from your `glance-registry` service to your `glance-api` configuration file. The following instructions assume that you want to use the v2 Image API for your installation. The v1 API is implemented on top of the `glance-registry` service while the v2 API is not.

Most configuration is done via configuration files, with the Glance API server (and possibly the Glance Registry server) using separate configuration files. When installing through an operating system package management system, sample configuration files are installed in `/etc/glance`.

This walkthrough installs the image service using a file backend and the Identity service (Keystone) for authentication.

Add the admin and service identifiers and `flavor=keystone` to the end of `/etc/glance/glance-api.conf` as shown below.

```
[keystone_authtoken]
auth_host = 127.0.0.1
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = glance
admin_password = glance

[paste_deploy]
# Name of the paste configuration file that defines the available pipelines
config_file = /etc/glance/glance-api-paste.ini

# Partial name of a pipeline in your paste configuration file with the
# service name removed. For example, if your paste section name is
# [pipeline:glance-api-keystone], you would configure the flavor below
# as 'keystone'.
flavor=keystone
```

Ensure that `/etc/glance/glance-api.conf` points to the MySQL database rather than `sqlite`.

```
sql_connection = mysql://glance:[YOUR_GLANCEDB_PASSWORD]@192.168.206.130/
glance
```

Restart `glance-api` to pick up these changed settings.

```
service glance-api restart
```

Update the last sections of `/etc/glance/glance-registry.conf` to reflect the values you set earlier for admin user and the service tenant, plus enable the Identity service with `flavor=keystone`.

```
[keystone_authtoken]
auth_host = 127.0.0.1
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = glance
admin_password = glance

[paste_deploy]
# Name of the paste configuration file that defines the available pipelines
config_file = /etc/glance/glance-registry-paste.ini

# Partial name of a pipeline in your paste configuration file with the
# service name removed. For example, if your paste section name is
# [pipeline:glance-api-keystone], you would configure the flavor below
# as 'keystone'.
flavor=keystone
```

Update `/etc/glance/glance-registry-paste.ini` by enabling the Identity service, `keystone`:

```
# Use this pipeline for keystone auth
[pipeline:glance-registry-keystone]
pipeline = authtoken context registryapp
```

Ensure that `/etc/glance/glance-registry.conf` points to the MySQL database rather than `sqlite`.

```
sql_connection = mysql://glance:[YOUR_GLANCEDB_PASSWORD]@192.168.206.130/  
glance
```

Restart glance-registry to pick up these changed settings.

```
service glance-registry restart
```



Note

Any time you change the .conf files, restart the corresponding service.

Now you can populate or migrate the database.

```
# glance-manage db_sync
```

Restart glance-registry and glance-api services, as root:

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



Note

This guide does not configure image caching, refer to <http://docs.openstack.org/developer/glance/> for more information.

Troubleshooting the Image Service (Glance)

To begin troubleshooting, look at the logs in the `/var/log/glance/registry.log` or `/var/log/glance/api.log`.

Verifying the Image Service Installation

To validate the Image service client is installed, enter `glance help` at the command line.

Obtain a test image.

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

Upload the kernel.



Note

This example shows inputting `--os-username`, `--os-password`, `--os-tenant-name`, `--os-auth-url` on the command line for reference. You could also use the `OS_*` environment variables by setting them in an `openrc` file:

```
export OS_USERNAME=admin  
export OS_TENANT_NAME=demo  
export OS_PASSWORD=secrete  
export OS_AUTH_URL=http://192.168.206.130:5000/v2.0/  
export OS_REGION_NAME=RegionOne
```

Then you would source these environment variables by running `source openrc`.

```
glance --os-username=admin --os-password=secrete --os-tenant-name=demo --os-  
auth-url=http://192.168.206.130:5000/v2.0 \  
image-create \  
--name="Cirros 0.3.1" \  
--disk-format=qcow2 \  
--container-format bare < cirros-0.3.1-x86_64-disk.img
```

Property	Value
checksum	d972013792949d0d3ba628fbe8685bce
container_format	bare
created_at	2013-05-08T18:59:18
deleted	False
deleted_at	None
disk_format	qcow2
id	acafc7c0-40aa-4026-9673-b879898e1fc2
is_public	False
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

Now a glance image-list should show these image IDs.

```
glance --os-username=admin --os-password=secrete --os-tenant-name=demo --os-  
auth-url=http://192.168.206.130:5000/v2.0 image-list
```

ID	Name		
Disk Format	Container Format	Size	Status
acafc7c0-40aa-4026-9673-b879898e1fc2	Cirros 0.3.1		
qcow2	bare	13147648	active
10ccdf86-e59e-41ac-ab41-65af91ea7a9c	cirros-0.3.0-x86_64-uec	ami	
ami	25165824	active	
8473f43f-cd1f-47cc-8d88-ccd9a62e566f	cirros-0.3.0-x86_64-uec-kernel	aki	
aki	4731440	active	
75c1bb27-a406-462c-a379-913e4e6221c9	cirros-0.3.0-x86_64-uec-ramdisk	ari	
ari	2254249	active	

7. Installing OpenStack Compute Service

Table of Contents

Configuring the Hypervisor	41
KVM	41
Checking for hardware virtualization support	42
Enabling KVM	43
Specifying the CPU model of KVM guests	44
Troubleshooting	45
QEMU	45
Tips and fixes for QEMU on RHEL	46
Xen, XenAPI, XenServer and XCP	46
Xen terminology	47
XenAPI deployment architecture	48
XenAPI pools	49
Installing XenServer and XCP	49
Xen Boot from ISO	53
Further reading	53
Pre-configuring the network	54
Configuration requirements with RHEL	54
Configuring the SQL Database (MySQL) on the Cloud Controller	55
Configuring the SQL Database (PostgreSQL) on the Cloud Controller	56
Installing and configuring Block Storage (Cinder)	57
Installing Compute Services	58
File format for nova.conf	59
Configuring OpenStack Compute	61
Configuring the Database for Compute	63
Creating the Network for Compute VMs	64
Verifying the Compute Installation	64
Defining Compute and Image Service Credentials	65
Installing Additional Compute Nodes	66
Adding Block Storage Nodes	66

Configuring the Hypervisor

For production environments the most tested hypervisors are KVM and Xen-based hypervisors. KVM runs through libvirt, Xen runs best through XenAPI calls. KVM is selected by default and requires the least additional configuration. This guide offers information on both but the specific walkthrough configuration options set up KVM.

KVM

KVM is configured as the default hypervisor for Compute.



Note

There are several sections about hypervisor selection in this document. If you are reading this document linearly, you do not want to load the KVM module prior to installing nova-compute. The nova-compute service depends on qemu-kvm which installs `/lib/udev/rules.d/45-qemu-kvm.rules`, which sets the correct permissions on the `/dev/kvm` device node.

To enable KVM explicitly, add the following configuration options `/etc/nova/nova.conf`:

```
compute_driver=libvirt.LibvirtDriver
libvirt_type=kvm
```

The KVM hypervisor supports the following virtual machine image formats:

- Raw
- QEMU Copy-on-write (qcow2)
- VMWare virtual machine disk format (vmdk)

The rest of this section describes how to enable KVM on your system. You may also wish to consult distribution-specific documentation:

- [Fedora: Getting started with virtualization](#) from the Fedora project wiki.
- [Ubuntu: KVM/Installation](#) from the Community Ubuntu documentation.
- [Debian: Virtualization with KVM](#) from the Debian handbook.
- [RHEL: Installing virtualization packages on an existing Red Hat Enterprise Linux system](#) from the Red Hat Enterprise Linux Virtualization Host Configuration and Guest Installation Guide.
- [openSUSE: Installing KVM](#) from the openSUSE Virtualization with KVM manual.
- [SLES: Installing KVM](#) from the SUSE Linux Enterprise Server Virtualization with KVM manual.

Checking for hardware virtualization support

The processors of your compute host need to support virtualization technology (VT) to use KVM.

If you are running on Ubuntu, install the `cpu` package and use the `kvm-ok` command to check if your processor has VT support, it is enabled in the BIOS, and KVM is installed properly, as root:

```
# apt-get install cpu
# kvm-ok
```

If KVM is enabled, the output should look something like:

```
INFO: /dev/kvm exists
KVM acceleration can be used
```

If KVM is not enabled, the output should look something like:

```
INFO: Your CPU does not support KVM extensions
KVM acceleration can NOT be used
```

In the case that KVM acceleration is not supported, Compute should be configured to use a different hypervisor, such as [QEMU](#) or [Xen](#).

On distributions that don't have **kvm-ok**, you can check if your processor has VT support by looking at the processor flags in the `/proc/cpuinfo` file. For Intel processors, look for the `vmx` flag, and for AMD processors, look for the `svm` flag. A simple way to check is to run the following command and see if there is any output:

```
$ egrep '(vmx|svm)' --color=always /proc/cpuinfo
```

Some systems require that you enable VT support in the system BIOS. If you believe your processor supports hardware acceleration but the above command produced no output, you may need to reboot your machine, enter the system BIOS, and enable the VT option.

Enabling KVM

KVM requires the `kvm` and either `kvm-intel` or `kvm-amd` modules to be loaded. This may have been configured automatically on your distribution when KVM is installed.

You can check that they have been loaded using **lsmod**, as follows, with expected output for Intel-based processors:

```
$ lsmod | grep kvm
kvm_intel      137721  9
kvm            415459  1 kvm_intel
```

The following sections describe how to load the kernel modules for Intel-based and AMD-based processors if they were not loaded automatically by your distribution's KVM installation process.

Intel-based processors

If your compute host is Intel-based, run the following as root to load the kernel modules:

```
# modprobe kvm
# modprobe kvm-intel
```

Add the following lines to `/etc/modules` so that these modules will load on reboot:

```
kvm
kvm-intel
```

AMD-based processors

If your compute host is AMD-based, run the following as root to load the kernel modules:

```
# modprobe kvm
# modprobe kvm-amd
```

Add the following lines to `/etc/modules` so that these modules will load on reboot:

```
kvm
kvm-amd
```

Specifying the CPU model of KVM guests

The Compute service allows you to control the guest CPU model that is exposed to KVM virtual machines. Use cases include:

- To maximise performance of virtual machines by exposing new host CPU features to the guest
- To ensure a consistent default CPU across all machines, removing reliance of variable QEMU defaults

In libvirt, the CPU is specified by providing a base CPU model name (which is a shorthand for a set of feature flags), a set of additional feature flags, and the topology (sockets/cores/threads). The libvirt KVM driver provides a number of standard CPU model names. Examples of model names include:

```
"486", "pentium", "pentium2", "pentiumpro", "coreduo", "n270",
"pentiumpro", "qemu32", "kvm32", "cpu64-rhel5", "cpu64-rhel5",
"kvm64", "pentiumpro", "Conroe", "Penryn", "Nehalem", "Westmere",
"pentiumpro", "cpu64-rhel5", "cpu64-rhel5", "Opteron_G1",
"Opteron_G2", "Opteron_G3", "Opteron_G4"
```

These models are defined in the file `/usr/share/libvirt/cpu_map.xml`. Check this file to determine which models are supported by your local installation.

There are two Compute configuration options that determine the type of CPU model exposed to the hypervisor when using KVM, `libvirt_cpu_mode` and `libvirt_cpu_model`.

The `libvirt_cpu_mode` option can take one of four values: `none`, `host-passthrough`, `host-model` and `custom`.

Host model (default for KVM & QEMU)

If your `nova.conf` contains `libvirt_cpu_mode=host-model`, libvirt will identify the CPU model in `/usr/share/libvirt/cpu_map.xml` which most closely matches the host, and then request additional CPU flags to complete the match. This should give close to maximum functionality/performance, while maintaining good reliability/compatibility if the guest is migrated to another host with slightly different host CPUs.

Host passthrough

If your `nova.conf` contains `libvirt_cpu_mode=host-passthrough`, libvirt will tell KVM to passthrough the host CPU with no modifications. The difference to `host-model`, instead of just matching feature flags, every last detail of the host CPU is matched. This gives absolutely best performance, and can be important to some apps which check low level CPU details, but it comes at a cost with respect to migration: the guest can only be migrated to an exactly matching host CPU.

Custom

If your `nova.conf` contains `libvirt_cpu_mode=custom`, you can explicitly specify one of the supported named model using the `libvirt_cpu_model` configuration option. For example, to configure the KVM guests to expose Nehalem CPUs, your `nova.conf` should contain:

```
libvirt_cpu_mode=custom
libvirt_cpu_model=Nehalem
```

None (default for all libvirt-driven hypervisors other than KVM & QEMU)

If your `nova.conf` contains `libvirt_cpu_mode=none`, then libvirt will not specify any CPU model at all. It will leave it up to the hypervisor to choose the default model. This setting is equivalent to the Compute service behavior prior to the Folsom release.

Troubleshooting

Trying to launch a new virtual machine instance fails with the `ERROR` state, and the following error appears in `/var/log/nova/nova-compute.log`

```
libvirtError: internal error no supported architecture for os type 'hvm'
```

This is a symptom that the KVM kernel modules have not been loaded.

If you cannot start VMs after installation without rebooting, it's possible the permissions are not correct. This can happen if you load the KVM module before you've installed `nova-compute`. To check the permissions, run `ls -l /dev/kvm` to see whether the group is set to `kvm`. If not, run `sudo udevadm trigger`.

QEMU

From the perspective of the Compute service, the QEMU hypervisor is very similar to the KVM hypervisor. Both are controlled through libvirt, both support the same feature set, and all virtual machine images that are compatible with KVM are also compatible with QEMU. The main difference is that QEMU does not support native virtualization. Consequently, QEMU has worse performance than KVM and is a poor choice for a production deployment.

The typical uses cases for QEMU are

- Running on older hardware that lacks virtualization support.
- Running the Compute service inside of a virtual machine for development or testing purposes, where the hypervisor does not support native virtualization for guests.

KVM requires hardware support for acceleration. If hardware support is not available (e.g., if you are running Compute inside of a VM and the hypervisor does not expose the required hardware support), you can use QEMU instead. KVM and QEMU have the same level of support in OpenStack, but KVM will provide better performance. To enable QEMU:

```
compute_driver=libvirt.LibvirtDriver
libvirt_type=qemu
```

For some operations you may also have to install the **guestmount** utility:

```
$> sudo yum install libguestfs-tools
```

The QEMU hypervisor supports the following virtual machine image formats:

- Raw
- QEMU Copy-on-write (qcow2)
- VMWare virtual machine disk format (vmdk)

Tips and fixes for QEMU on RHEL

If you are testing OpenStack in a virtual machine, you need to configure nova to use qemu without KVM and hardware virtualization. The second command relaxes SELinux rules to allow this mode of operation (https://bugzilla.redhat.com/show_bug.cgi?id=753589) The last 2 commands here work around a libvirt issue fixed in RHEL 6.4. Note nested virtualization will be the much slower TCG variety, and you should provide lots of memory to the top level guest, as the OpenStack-created guests default to 2GM RAM with no overcommit.



Note

The second command, **setsebool**, may take a while.

```
$> sudo openstack-config --set /etc/nova/nova.conf DEFAULT libvirt_type qemu
$> setsebool -P virt_use_execmem on
$> sudo ln -s /usr/libexec/qemu-kvm /usr/bin/qemu-system-x86_64
$> sudo service libvirtd restart
```

Xen, XenAPI, XenServer and XCP

The recommended way to use Xen with OpenStack is through the XenAPI driver. To enable the XenAPI driver, add the following configuration options `/etc/nova/nova.conf` and restart the nova-compute service:

```
compute_driver=xenapi.XenAPIDriver
xenapi_connection_url=http://your_xenapi_management_ip_address
xenapi_connection_username=root
xenapi_connection_password=your_password
```

The above connection details are used by the OpenStack Compute service to contact your hypervisor and are the same details you use to connect XenCenter, the XenServer management console, to your XenServer or XCP box. Note these settings are generally unique to each hypervisor host as the use of the host internal management network IP address (169.254.0.1) will cause features such as live-migration to break.

OpenStack with XenAPI supports the following virtual machine image formats:

- Raw
- VHD (in a gzipped tarball)

It is possible to manage Xen using libvirt. This would be necessary for any Xen-based system that isn't using the XCP toolstack, such as SUSE Linux or Oracle Linux. Unfortunately, this is not well-tested or supported. To experiment using Xen through libvirt add the following configuration options `/etc/nova/nova.conf`:

```
compute_driver=libvirt.LibvirtDriver  
libvirt_type=xen
```

The rest of this section describes Xen, XCP, and XenServer, the differences between them, and how to use them with OpenStack. Xen's architecture is different from KVM's in important ways, and we discuss those differences and when each might make sense in your OpenStack cloud.

Xen terminology

Xen is a hypervisor. It provides the fundamental isolation between virtual machines. Xen is open source (GPLv2) and is managed by Xen.org, an cross-industry organization.

Xen is a component of many different products and projects. The hypervisor itself is very similar across all these projects, but the way that it is managed can be different, which can cause confusion if you're not clear which tool stack you are using. Make sure you know what tool stack you want before you get started.

Xen Cloud Platform (XCP) is an open source (GPLv2) tool stack for Xen. It is designed specifically as platform for enterprise and cloud computing, and is well integrated with OpenStack. XCP is available both as a binary distribution, installed from an iso, and from Linux distributions, such as [xcp-xapi](#) in Ubuntu. The current versions of XCP available in Linux distributions do not yet include all the features available in the binary distribution of XCP.

Citrix XenServer is a commercial product. It is based on XCP, and exposes the same tool stack and management API. As an analogy, think of XenServer being based on XCP in the way that Red Hat Enterprise Linux is based on Fedora. XenServer has a free version (which is very similar to XCP) and paid-for versions with additional features enabled. Citrix provides support for XenServer, but as of July 2012, they do not provide any support for XCP. For a comparison between these products see the [XCP Feature Matrix](#).

Both XenServer and XCP include Xen, Linux, and the primary control daemon known as **xapi**.

The API shared between XCP and XenServer is called **XenAPI**. OpenStack usually refers to XenAPI, to indicate that the integration works equally well on XCP and XenServer. Sometimes, a careless person will refer to XenServer specifically, but you can be reasonably confident that anything that works on XenServer will also work on the latest version of XCP. Read the [XenAPI Object Model Overview](#) for definitions of XenAPI specific terms such as SR, VDI, VIF and PIF.

Privileged and unprivileged domains

A Xen host will run a number of virtual machines, VMs, or domains (the terms are synonymous on Xen). One of these is in charge of running the rest of the system, and is known as "domain 0", or "dom0". It is the first domain to boot after Xen, and owns the storage and networking hardware, the device drivers, and the primary control software.

There is an ongoing project to split domain 0 into multiple privileged domains known as **driver domains** and **stub domains**. This would give even better separation between critical components. This technology is what powers Citrix XenClient RT, and is likely to be added into XCP in the next few years. However, the current architecture just has three levels of separation: dom0, the OpenStack domU, and the completely unprivileged customer VMs.

A Xen virtual machine can be **paravirtualized (PV)** or **hardware virtualized (HVM)**. This refers to the interaction between Xen, domain 0, and the guest VM's kernel. PV guests are aware of the fact that they are virtualized and will co-operate with Xen and domain 0; this gives them better performance characteristics. HVM guests are not aware of their environment, and the hardware has to pretend that they are running on an unvirtualized machine. HVM guests have the advantage that there is no need to modify the guest operating system, which is essential when running Windows.

XenAPI deployment architecture

The diagram illustrates the network architecture for OpenStack VMs connected to physical hosts. It shows the following components and connections:

- Physical Host:** Contains network interfaces `xenbr0` and `eth0`.
- OpenStack VM:** Contains network interfaces `eth0`, `eth1`, and `eth2`.
- Tenant VM:** Contains network interface `eth0`.
- Networks:**
 - Management Network (Red):** Connects `xenbr0` to `eth0` in the OpenStack VM.
 - Tenant Network (Blue):** Connects `eth1` in the OpenStack VM to `eth0` in the Tenant VM.
 - Public Network (Green):** Connects `eth2` in the OpenStack VM to the Tenant VM.
- OpenStack Components:**
 - Domain 0:** Includes OpenStack add-ons (xapi plugins, Network isolation rules) and xapi.
 - OpenStack:** Includes nova-compute, nova-network, nova.virt.xenapi, dhcpd, and XenAPI.
- Xen:** The hypervisor layer connecting the OpenStack VM and Tenant VM.
- Storage Repository:** Connected to the Xen layer via blue lines.
- Virtual block devices:** Usually on local disk, connected to the Storage Repository.

Key connections and notes:

- A red dashed line connects `xapi` in Domain 0 to `XenAPI` in OpenStack.
- A note states: "OpenStack is using the XenAPI python module to communicate with dom0 through the management network".
- A note states: "Networks connected to physical interfaces according to the selected configuration."
- A note states: "Virtual block devices, usually on local disk."

48

-
- The hypervisor: Xen
 - Domain 0: runs xapi and some small pieces from OpenStack (some xapi plugins and network isolation rules). The majority of this is provided by XenServer or XCP (or yourself using Kronos).
 - OpenStack VM: The nova-compute code runs in a paravirtualized virtual machine, running on the host under management. Each host runs a local instance of nova-compute. It will often also be running nova-network (depending on your network mode). In this case, nova-network is managing the addresses given to the tenant VMs through DHCP.
 - Nova uses the XenAPI Python library to talk to xapi, and it uses the Management Network to reach from the domU to dom0 without leaving the host.

Some notes on the networking:

- The above diagram assumes FlatDHCP networking (the DevStack default).
- There are three main OpenStack Networks:
 - Management network - RabbitMQ, MySQL, etc. Please note, that the VM images are downloaded by the xenapi plugins, so please make sure, that the images could be downloaded through the management network. It usually means, binding those services to the management interface.
 - Tenant network - controlled by nova-network. The parameters of this network depends on the networking model selected (Flat, Flat DHCP, VLAN)
 - Public network - floating IPs, public API endpoints.
- The networks shown here need to be connected to the corresponding physical networks within the datacenter. In the simplest case, three individual physical network cards could be used. It is also possible to use VLANs to separate these networks. Please note, that the selected configuration must be in line with the networking model selected for the cloud. (in case of VLAN networking, the physical channels have to be able to forward the tagged traffic)

XenAPI pools

The host-aggregates feature allows you to create pools of XenServer hosts (configuring shared storage is still an out of band activity), to enable live migration when using shared storage.

Installing XenServer and XCP

When you want to run OpenStack with XCP or XenServer, you first need to install the software on [an appropriate server](#). Please note, Xen is a type 1 hypervisor. This means when your server starts the first software that runs is Xen. This means the software you install on your compute host is XenServer or XCP, not the operating system you wish to run the OpenStack code on. The OpenStack services will run in a VM you install on top of XenServer.

Before you can install your system you must decide if you want to install Citrix XenServer (either the free edition, or one of the paid editions) or Xen Cloud Platform from Xen.org. You can download the software from the following locations:

- <http://www.citrix.com/XenServer/download>
- <http://www.xen.org/download/xcp/index.html>

When installing many servers, you may find it easier to perform [PXE boot installations of XenServer or XCP](#). You can also package up any post install changes you wish to make to your XenServer by [creating your own XenServer supplemental pack](#).

It is also possible to get XCP by installing the **xcp-xenapi** package on Debian based distributions. However, this is not as mature or feature complete as above distributions. This will modify your boot loader to first boot Xen, then boot your existing OS on top of Xen as Dom0. It is in Dom0 that the xapi daemon will run. You can find more details on the Xen.org wiki: http://wiki.xen.org/wiki/Project_Kronos



Important

Ensure you are using the EXT type of storage repository (SR). Features that require access to VHD files (such as copy on write, snapshot and migration) do not work when using the LVM SR. Storage repository (SR) is a XenAPI specific term relating to the physical storage on which virtual disks are stored.

On the XenServer/XCP installation screen, this is selected by choosing "XenDesktop Optimized" option. In case you are using an answer file, make sure you use `srtype="ext"` within the `installation` tag of the answer file.

Post install steps

You are now ready to install OpenStack onto your XenServer system. This process involves the following steps:

- For resize and migrate functionality, please perform the changes described in the [Configuring Resize](#) section of the OpenStack Compute Administration Manual.
- Install the VIF isolation rules to help prevent mac and ip address spoofing.
- Install the XenAPI plugins - see the next section.
- In order to support AMI type images, you need to set up `/boot/guest symlink/` directory in Dom0. For detailed instructions, see next section.
- To support resize/migration, set up an ssh trust relation between your XenServer hosts, and ensure `/images` is properly set up. See next section for more details.
- Create a Paravirtualised virtual machine that can run the OpenStack compute code.
- Install and configure the nova-compute in the above virtual machine.

For further information on these steps look at how DevStack performs the last three steps when doing developer deployments. For more information on DevStack, take a look at the

[DevStack and XenServer Readme](#). More information on the first step can be found in the [XenServer mutli-tenancy protection doc](#). More information on how to install the XenAPI plugins can be found in the [XenAPI plugins Readme](#).

Installing the XenAPI Plugins

When using Xen as the hypervisor for OpenStack Compute, you can install a Python script (usually, but it can be any executable) on the host side, and then call that through the XenAPI. These scripts are called plugins. The XenAPI plugins live in the nova code repository. These plugins have to be copied to the hypervisor's Dom0, to the appropriate directory, where xapi can find them. There are several options for the installation. The important thing is to ensure that the version of the plugins are in line with the nova installation by only installing plugins from a matching nova repository.

Manual Installation:

- Create temporary files/directories:

```
$ NOVA_ZIPBALL=$(mktemp)
$ NOVA_SOURCES=$(mktemp -d)
```

- Get the source from github. The example assumes the master branch is used, please amend the URL to match the version being used:

```
$ wget -qO "$NOVA_ZIPBALL" https://github.com/openstack/nova/archive/master.zip
$ unzip "$NOVA_ZIPBALL" -d "$NOVA_SOURCES"
```

(Alternatively) Should you wish to use the official Ubuntu packages, use the following commands to get the nova codebase:

```
$ ( cd $NOVA_SOURCES && apt-get source python-nova --download-only )
$ ( cd $NOVA_SOURCES && for ARCHIVE in *.tar.gz; do tar -xzf $ARCHIVE; done )
```

- Copy the plugins to the hypervisor:

```
$ PLUGINPATH=$(find $NOVA_SOURCES -path '*/xapi.d/plugins' -type d -print)
$ tar -czf - -C "$PLUGINPATH" ./ | ssh root@xenserver tar -xozf - -C /etc/xapi.d/plugins/
```

- Remove the temporary files/directories:

```
$ rm "$NOVA_ZIPBALL"
$ rm -rf "$NOVA_SOURCES"
```

Packaged Installation:

Follow these steps to produce a supplemental pack from the nova sources, and package it as a XenServer Supplemental Pack.

- Create RPM packages. Given you have the nova sources (use one of the methods mentioned at Manual Installation):

```
$ cd nova/plugins/xenserver/xenapi/contrib
$ ./build-rpm.sh
```

The above commands should leave an `.rpm` file in the `rpmbuild/RPMS/noarch/` directory.

- Pack the RPM packages to a Supplemental Pack, using the XenServer DDK (the following command should be issued on the XenServer DDK virtual appliance, after the produced rpm file has been copied over):

```
$ /usr/bin/build-supplemental-pack.sh \
> --output=output_directory \
> --vendor-code=novaplugin \
> --vendor-name=openstack \
> --label=novaplugins \
> --text="nova plugins" \
> --version=0 \
> full_path_to_rpmfile
```

The above command should produce an `.iso` file in the output directory specified. Copy that file to the hypervisor.

- Install the Supplemental Pack. Log in to the hypervisor, and issue:

```
# xe-install-supplemental-pack path_to_isofile
```

Prepare for AMI Type Images

In order to support AMI type images within your OpenStack installation, a directory `/boot/guest` needs to be created inside Dom0. The OpenStack VM will put the kernel and ramdisk extracted from the AKI and ARI images to this location.

This directory's content will be maintained by OpenStack, and its size should not increase during normal operation, however in case of power failures or accidental shutdowns, some files might be left over. In order to prevent these files to fill up Dom0's disk, it is recommended to set up this directory as a symlink pointing to a subdirectory of the local SR.

Execute the following commands in Dom0 to achieve the above mentioned setup:

```
# LOCAL_SR=$(xe sr-list name-label="Local storage" --minimal)
# LOCALPATH="/var/run/sr-mount/${LOCAL_SR}/os-guest-kernels"
# mkdir -p "$LOCALPATH"
# ln -s "$LOCALPATH" /boot/guest
```

Dom0 Modifications for Resize/Migration Support

To get resize to work with XenServer (and XCP) you need to:

- Establish a root trust between all hypervisor nodes of your deployment:

You can do so by generating an ssh key-pair (with `ssh-keygen`) and then ensuring that each of your dom0's `authorized_keys` file (located in `/root/.ssh/`)

`authorized_keys`) contains the public key fingerprint (located in `/root/.ssh/id_rsa.pub`).

- Provide an `/images` mount point to your hypervisor's dom0:

Dom0 space is a premium so creating a directory in dom0 is kind of dangerous, and almost surely bound to fail especially when resizing big servers. The least you can do is to symlink `/images` to your local storage SR. The instructions below work for an English-based installation of XenServer (and XCP) and in the case of ext3 based SR (with which the resize functionality is known to work correctly).

```
# LOCAL_SR=$(xe sr-list name-label="Local storage" --minimal)
# IMG_DIR="/var/run/sr-mount/$LOCAL_SR/images"
# mkdir -p "$IMG_DIR"
# ln -s "$IMG_DIR" /images
```

Xen Boot from ISO

XenServer, through the XenAPI integration with OpenStack provides a feature to boot instances from an ISO file. In order to activate the "Boot From ISO" feature, the SR elements on XenServer host must be configured that way.

First, create an ISO-typed SR, such as an NFS ISO library, for instance. For this, using XenCenter is a simple method. You need to export an NFS volume from a remote NFS server. Make sure it is exported in read-write mode.

Second, on the compute host, find the uuid of this ISO SR and write it down.

```
# xe host-list
```

Next, locate the uuid of the NFS ISO library:

```
# xe sr-list content-type=iso
```

Set the uuid and configuration. Even if an NFS mount point isn't local storage, you must specify "local-storage-iso".

```
# xe sr-param-set uuid=[iso sr uuid] other-config:il8n-key=local-storage-iso
```

Now, make sure the host-uuid from "xe pbd-list" equals the uuid of the host you found earlier

```
# xe sr-uuid=[iso sr uuid]
```

You should now be able to add images via the OpenStack Image Registry, with `disk-format=iso`, and boot them in OpenStack Compute.

```
glance image-create --name=fedora_iso --disk-format=iso --container-format=bare < Fedora-16-x86_64-netinst.iso
```

Further reading

Here are some of the resources available to learn more about Xen:

- Citrix XenServer official documentation: <http://docs.vmd.citrix.com/XenServer>.
- What is Xen? by Xen.org: <http://xen.org/files/Marketing/WhatisXen.pdf>.
- Xen Hypervisor project: <http://xen.org/products/xenhyp.html>.
- XCP project: <http://xen.org/products/cloudxen.html>.
- Further XenServer and OpenStack information: <http://wiki.openstack.org/XenServer>.

Pre-configuring the network

These instructions are for using the FlatDHCP networking mode with a single network interface. More complex configurations are described in the networking section, but this configuration is known to work. These configuration options should be set on all compute nodes.

Set your network interface in promiscuous mode so that it can receive packets that are intended for virtual machines. As root:

```
# ip link set eth0 promisc on
```

Here's an example network setup for RHEL, Fedora, or CentOS. Create `/etc/sysconfig/network-scripts/ifcfg-br100`:

```
DEVICE=br100
TYPE=Bridge
ONBOOT=yes
DELAY=0
BOOTPROTO=static
IPADDR=192.168.100.1
NETMASK=255.255.255.0
```

Also install bridge-utils:

```
$sudo yum install bridge-utils
```

Ensure that you set up the bridge, although if you use `flat_network_bridge=br100` in your `nova.conf` file, nova will set up the bridge for you when you run the **nova network-create** command.

```
sudo brctl addbr br100
```

Lastly, restart networking to have these changes take effect. (This method is deprecated but "restart networking" doesn't always work.)

```
sudo /etc/init.d/networking restart
```

Configuration requirements with RHEL

Set selinux in permissive mode:

```
$> sudo setenforce permissive
```

Otherwise you will get issues like https://bugzilla.redhat.com/show_bug.cgi?id=734346 /usr/bin/nova-dhcpbridge: No such file or directory.

If RHEL 6.2 based, use the openstack-config package to turn off force DHCP releases.

```
$> sudo openstack-config --set /etc/nova/nova.conf DEFAULT force_dhcp_release False
```

If RHEL 6.3 based, install dnsmasq utilities.

```
$> sudo yum install dnsmasq-utils
```

If you intend to use guest images that don't have a single partition, then allow libguestfs to inspect the image so that files can be injected, by setting:

```
$> sudo openstack-config --set /etc/nova/nova.conf DEFAULT libvirt_inject_partition -1
```

Configuring the SQL Database (MySQL) on the Cloud Controller

Start the mysql command line client by running:

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

Enter the mysql root user's password when prompted.

To configure the MySQL database, create the nova database.

```
mysql> CREATE DATABASE nova;
```

Create a MySQL user and password for the newly-created nova database that has full control of the database.

```
mysql> GRANT ALL ON nova.* TO 'nova'@'%' IDENTIFIED BY '[YOUR_NOVADB_PASSWORD]';
mysql> GRANT ALL ON nova.* TO 'nova'@'localhost' IDENTIFIED BY '[YOUR_NOVADB_PASSWORD]';
```



Note

In the above commands, even though the 'nova'@'%' also matches 'nova'@'localhost', you must explicitly specify the 'nova'@'localhost' entry.

By default, MySQL will create entries in the user table with User=' ' and Host='localhost'. The User=' ' acts as a wildcard, matching all users. If you do not have the 'nova'@'localhost' account, and you try to log in as the nova user, the precedence rules of MySQL will match against the User=' ' Host='localhost' account before it matches against the User='nova' Host='%' account. This will result in an error message that looks like:

```
ERROR 1045 (28000): Access denied for user 'nova'@'localhost' (using password: YES)
```

Thus, we create a separate `User='nova' Host='localhost'` entry that will match with higher precedence.

See the [MySQL documentation on connection verification](#) for more details on how MySQL determines which row in the user table it uses when authenticating connections.

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

```
mysql> quit
```

The command to populate the database is described later in the documentation, in the Section entitled [Configuring the Database for Compute](#).



Securing MySQL

Additional steps are required to configure MySQL for production mode. In particular, anonymous accounts should be removed. On several distributions, these accounts can be removed by running the following script after installing `mysql: /usr/bin/mysql_secure_installation`

Configuring the SQL Database (PostgreSQL) on the Cloud Controller

Optionally, if you choose not to use MySQL, you can install and configure PostgreSQL for all your databases. Here's a walkthrough for the Nova database:

```
$ sudo yum install postgresql postgresql-client
```

Start the PostgreSQL command line client by running:

```
sudo su - postgres
```

Enter the postgresql root user's password if prompted.

To configure the database, create the nova database.

```
postgres> psql
postgres=# CREATE USER novadbadmin;
postgres=# ALTER USER novadbadmin WITH PASSWORD '[YOUR_NOVADB_PASSWORD]';
postgres=# CREATE DATABASE nova;
postgres=# GRANT ALL PRIVILEGES ON DATABASE nova TO novadbadmin;
postgres=# \q
postgres> exit
```

The database is created and we have a privileged user that controls the database. Now we have to install the packages that will help Nova access the database.

```
$ sudo yum install python-sqlalchemy python-psycopg2
```

Configure the `/etc/nova/nova.conf` file, to ensure it knows to use the PostgreSQL database:

```
sql_connection = postgres://novadbadmin:[YOUR_NOVADB_PASSWORD]@127.0.0.1/  
nova
```

The command to populate the database is described later in the documentation, in the section entitled [Configuring the Database for Compute](#).

Installing and configuring Block Storage (Cinder)

Install the packages for OpenStack Block Storage (cinder) on the cloud controller.

```
# yum install openstack-cinder openstack-cinder-doc \  
iscsi-initiator-utils scsi-target-utils
```

Edit `/etc/cinder/api-paste.ini` (filter authToken).

```
[filter:authtoken]  
paste.filter_factory = keystone.middleware.auth_token:filter_factory  
service_protocol = http  
service_host = 192.168.206.130  
service_port = 5000  
auth_host = 192.168.206.130  
auth_port = 35357  
auth_protocol = http  
admin_tenant_name = service  
admin_user = cinder  
admin_password = openstack
```

Edit `/etc/cinder/cinder.conf` to reflect your settings.

```
[DEFAULT]  
rootwrap_config=/etc/cinder/rootwrap.conf  
sql_connection = mysql://cinder:openstack@192.168.127.130/cinder  
api_paste_config = /etc/cinder/api-paste.ini  
  
iscsi_helper=tgtadm  
volume_name_template = volume-%s  
volume_group = cinder-volumes  
verbose = True  
auth_strategy = keystone  
#osapi_volume_listen_port=5900
```

Configure messaging (RabbitMQ or Qpid) also in `/etc/cinder/cinder.conf`.

```
# Ubuntu  
rabbit_host = 10.10.10.10  
rabbit_port = 5672  
rabbit_userid = rabbit  
rabbit_password = secure_password  
rabbit_virtual_host = /nova
```

```
# Red Hat, Fedora, CentOS  
qpid_hostname=192.168.206.130
```

Verify entries in `nova.conf`. The `volume_api_class` setting is the default setting for grizzly.

```
volume_api_class=nova.volume.cinder.API
```

Set up the cinder database.


```
CREATE DATABASE cinder;
GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'localhost' \
IDENTIFIED BY 'password';
FLUSH PRIVILEGES;
```

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 every physical volume that is needed for LVM on the Cinder host. You can get a list by running `pvdisplay`. Each item in the filter array starts with either an "a" for accept, or an "r" for reject. Physical volumes that are needed on the Cinder host begin with "a". The array must end with "r/. */"

```
devices {
...
filter = [ "a/sda1/", "a/sdb1/", "r/.*/" ]
...
}
```

Setup the target file NOTE: `$state_path=/var/lib/cinder/` and `$volumes_dir=$state_path/volumes` by default and path **MUST** exist!.

```
$ sudo sh -c "echo 'include $volumes_dir/*' >> /etc/tgt/conf.d/cinder.conf"
```

Restart the `tgt` service.

```
$ sudo restart tgt
```

Populate the database.

```
$ sudo cinder-manage db sync
```

Restart the services.

```
$ sudo service cinder-volume restart
$ sudo service cinder-api restart
$ sudo service cinder-scheduler restart
```

Create a 1 GB test volume.

```
$ cinder create --display_name test 1
$ cinder list
```

```
+-----+-----+-----+-----+
+-----+-----+
|          ID          | Status | Display Name | Size |
| Volume Type | Attached to |
+-----+-----+-----+-----+
+-----+-----+
| 5bbad3f9-50ad-42c5-b58c-9b6b63ef3532 | available | test | 1 |
| None | |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

Installing Compute Services

On both the controller and compute nodes install the required nova- packages, and dependencies are automatically installed.

```
sudo yum install openstack-nova
```

File format for nova.conf

Overview

The Compute service supports a large number of configuration options. These options are specified in a configuration file whose default location is `/etc/nova/nova.conf`.

The configuration file is in [INI file format](#), with options specified as `key=value` pairs, grouped into sections. Almost all of the configuration options are in the `DEFAULT` section. Here's a brief example:

```
[DEFAULT]
debug=true
verbose=true

[trusted_computing]
server=10.3.4.2
```

Types of configuration options

Each configuration option has an associated type that indicates what values can be set. The supported option types are as follows:

BoolOpt Boolean option. Value must be either `true` or `false`. Example:

```
debug=false
```

StrOpt String option. Value is an arbitrary string. Example:

```
my_ip=10.0.0.1
```

IntOption Integer option. Value must be an integer. Example:

```
glance_port=9292
```

MultiStrOpt String option. Same as `StrOpt`, except that it can be declared multiple times to indicate multiple values. Example:

```
ldap_dns_servers=dns1.example.org
ldap_dns_servers=dns2.example.org
```

ListOpt List option. Value is a list of arbitrary strings separated by commas. Example:

```
enabled_apis=ec2,osapi_compute,metadata
```

FloatOpt Floating-point option. Value must be a floating-point number. Example:

```
ram_allocation_ratio=1.5
```



Important

Nova options should *not* be quoted.

Sections

Configuration options are grouped by section. The Compute config file supports the following sections.

<code>[DEFAULT]</code>	Almost all of the configuration options are organized into this section. If the documentation for a configuration option does not specify its section, assume that it should be placed in this one.
<code>[cells]</code>	The <code>cells</code> section is used for options for configuring cells functionality. See the Cells section of the OpenStack Compute Admin Manual for more details.
<code>[baremetal]</code>	This section is used for options that relate to the baremetal hypervisor driver.
<code>[conductor]</code>	The <code>conductor</code> section is used for options for configuring the nova-conductor service.
<code>[trusted_computing]</code>	The <code>trusted_computing</code> section is used for options that relate to the trusted computing pools functionality. Options in this section describe how to connect to a remote attestation service.

Variable substitution

The configuration file supports variable substitution. Once a configuration option is set, it can be referenced in later configuration values when preceded by `$`. Consider the following example where `my_ip` is defined and then `$my_ip` is used as a variable.

```
my_ip=10.2.3.4
glance_host=$my_ip
metadata_host=$my_ip
```

If you need a value to contain the `$` symbol, escape it by doing `$$`. For example, if your LDAP DNS password was `$xkj432`, you would do:

```
ldap_dns_password=$$xkj432
```

The Compute code uses Python's `string.Template.safe_substitute()` method to implement variable substitution. For more details on how variable substitution is resolved, see [Python documentation on template strings](#) and [PEP 292](#).

Whitespace

To include whitespace in a configuration value, use a quoted string. For example:

```
ldap_dns_password='a password with spaces'
```

Specifying an alternate location for nova.conf

The configuration file is loaded by all of the nova-* services, as well as the **nova-manage** command-line tool. To specify an alternate location for the configuration file, pass the

--config-file */path/to/nova.conf* argument when starting a nova-* service or calling **nova-manage**.

Configuring OpenStack Compute

This section describes the relevant `nova.conf` settings for getting a minimal install running. Refer to the OpenStack Compute Administration Manual for guidance on more configuration options.

In general, you can use the same `nova.conf` file across the controller and compute nodes. However, the following configuration options need to be changed on each compute host:

- `my_ip`
- `vncserver_listen`
- `vncserver_proxyclient_address`

For the above configuration options, you must use the IP address of the specific compute host, not the cloud controller.

The packages automatically do these steps for a user named `nova`, but if you are installing as another user you should ensure that the `nova.conf` file should have its owner set to `root:nova`, and mode set to `0640`, since the file contains your MySQL server's username and password.



Note

If you are installing as another user, you should set permissions correctly. This packaged install ensures that the `nova` user belongs to the `nova` group and that the `.conf` file permissions are set, but here are the manual commands, which should be run as root:

```
# groupadd nova
# usermod -g nova nova
# chown -R nova:nova /etc/nova
# chmod 640 /etc/nova/nova.conf
```

The hypervisor is set by editing `/etc/nova/nova.conf`. The hypervisor defaults to `kvm`, but if you are working within a VM already, switch to `qemu` on the `libvirt_type=` line. To use Xen, refer to the overview in this book for where to install nova components.



Note

You can also configure the `nova-compute` service (and, for example configure a hypervisor-per-compute-node) with a separate `nova-compute.conf` file and then referring to `nova-compute.conf` in the `nova.conf` file.

Ensure the database connection defines your backend data store by adding a `sql_connection` line to `nova.conf`:

```
sql_connection=mysql://[user]:[pass]@[primary IP]/[db name], such as
sql_connection=mysql://nova:yourpassword@192.168.206.130/nova.
```

Add these settings to `/etc/nova/nova.conf` for the network configuration assumptions made for this installation scenario. You can place comments in the `nova.conf` file by entering a new line with a `#` sign at the beginning of the line. To see a listing of all possible configuration option settings, see [the reference in the OpenStack Compute Administration Manual](#).

```
auth_strategy=keystone
network_manager=nova.network.manager.FlatDHCPManager
fixed_range=192.168.100.0/24
public_interface=eth0
flat_interface=eth0
flat_network_bridge=br100
```

Here is an example `nova.conf` with commented sections:

```
[DEFAULT]

# LOGS/STATE
verbose=True
logdir=/var/log/nova
state_path=/var/lib/nova
lock_path=/var/lock/nova
rootwrap_config=/etc/nova/rootwrap.conf

# SCHEDULER
compute_scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler

# VOLUMES
# configured in cinder.conf

# DATABASE
sql_connection=mysql://nova:yourpassword@192.168.206.130/nova

# COMPUTE
libvirt_type=qemu
compute_driver=libvirt.LibvirtDriver
instance_name_template=instance-%08x
api_paste_config=/etc/nova/api-paste.ini

# COMPUTE/APIS: if you have separate configs for separate services
# this flag is required for both nova-api and nova-compute
allow_resize_to_same_host=True

# APIS
osapi_compute_extension=nova.api.openstack.compute.contrib.standard_extensions
ec2_dmz_host=192.168.206.130
s3_host=192.168.206.130

# QPID
rpc_backend=nova.rpc.impl_qpid
qpid_hostname=192.168.206.130

# GLANCE
image_service=nova.image.glance.GlanceImageService
glance_api_servers=192.168.206.130:9292

# NETWORK
network_manager=nova.network.manager.FlatDHCPManager
force_dhcp_release=True
```

```
dhcpbridge_flagfile=/etc/nova/nova.conf
firewall_driver=nova.virt.libvirt.firewall.IptablesFirewallDriver
# Change my_ip to match each host
my_ip=192.168.206.130
public_interface=eth100
vlan_interface=eth0
flat_network_bridge=br100
flat_interface=eth0
fixed_range=192.168.100.0/24

# NOVNC CONSOLE
novncproxy_base_url=http://192.168.206.130:6080/vnc_auto.html
# Change vncserver_proxyclient_address and vncserver_listen to match each
compute host
vncserver_proxyclient_address=192.168.206.130
vncserver_listen=192.168.206.130

# AUTHENTICATION
auth_strategy=keystone
[keystone_authtoken]
auth_host = 127.0.0.1
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = nova
admin_password = nova
signing_dirname = /tmp/keystone-signing-nova
```



Note

The `my_ip` configuration option will be different for each host, edit it accordingly.

The controller node will run the `nova-api`, `nova-scheduler`, `nova-cert`, `nova-consoleauth`, `nova-conductor` and optionally `nova-network` services. The compute node will run the `nova-compute` and `nova-network` services. Stop the `nova-` services prior to running `db sync`, by running `stop` commands as root. Otherwise your logs show errors because the database has not yet been populated. On the controller node run:

```
$> for svc in api objectstore conductor network volume scheduler cert;
do sudo service openstack-nova-$svc stop ; sudo chkconfig openstack-nova-$svc
on ; done
```

On the compute node run:

```
$> for svc in api compute network; do sudo service openstack-nova-$svc
stop ; sudo chkconfig openstack-nova-$svc on ; done
```

Configuring the Database for Compute

Create the tables in your backend data store by running the following command:

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

If you see any response, you can look in `/var/log/nova/nova-manage.log` to see the problem. No response means the command completed correctly and your nova database is now populated.



Deprecation warnings

Note that if while running this command you see warnings such as `SADeprecationWarning: The 'listeners' argument to Pool (and create_engine()) is deprecated. Use event.listen().`, these will be fixed in future version of the libraries and can be safely ignored.

Restart all services in total, just to cover the entire spectrum. On the controller node run:

```
# for svc in api objectstore conductor network volume scheduler cert;  
do sudo service openstack-nova-$svc start; done
```

On the compute node run:

```
# for svc in compute network; do sudo service openstack-nova-$svc  
start; done
```

All nova services are now installed and started. If the "start" command doesn't work, your services may not be running correctly (or not at all). Review the logs in `/var/log/nova` to look for clues.

Creating the Network for Compute VMs

You must run the command that creates the network and the bridge using the `br100` specified in the `nova.conf` file to create the network that the virtual machines use. This example shows the network range using `192.168.100.0/24` as the fixed range for our guest VMs, but you can substitute the range for the network you have available. We're labeling it with `private` in this case.

```
nova network-create private --fixed-range-v4=192.168.100.0/24 --bridge-  
interface=br100
```



Note

You can find out more about the `nova network-create` command with `nova help network-create`.

Verifying the Compute Installation

You can ensure all the Compute services are running by using the `nova-manage` command, as root:

```
# nova-manage service list
```

In return you should see "smiley faces" rather than three X symbols. Here's an example.

Binary	Host	Zone	Status	State	Updated_At
nova-compute	myhost	nova	enabled	: -)	2012-04-02 14:06:15
nova-cert	myhost	nova	enabled	: -)	2012-04-02 14:06:16
nova-scheduler	myhost	nova	enabled	: -)	2012-04-02 14:06:11
nova-network	myhost	nova	enabled	: -)	2012-04-02 14:06:13
nova-consoleauth	myhost	nova	enabled	: -)	2012-04-02 14:06:10



Note

If you see three X symbols and are running services on separate hosts, ensure that ntp is synchronizing time correctly and that all servers match their time. Out-of-sync time stamps are the most common cause of the XXX state.

You can find the version of the installation by using the **nova-manage** command, as root:

```
# nova-manage version
```

The version number 2013.1 corresponds with the Grizzly release of Compute.

```
2013.1
```

Defining Compute and Image Service Credentials

The commands in this section can be run on any machine that can access the cloud controller node over the network. You can run commands directly on the cloud controller, if you like, but it isn't required.

Create an `openrc` file that can contain these variables that are used by the **nova** (Compute) and **glance** (Image) command-line interface clients. These commands can be run by any user, and the `openrc` file can be stored anywhere. In this document, we store the `openrc` file in the `~/creds` directory:

```
$ mkdir ~/creds
$ nano ~/creds/openrc
```

In this example, we are going to create an `openrc` file with credentials associated with a user who is not an administrator. Because the user is not an administrator, the credential file will use the URL associated with the keystone service API, which runs on port 5000. If we wanted to use the **keystone** command-line tool to perform administrative commands, we would use the URL associated with the keystone admin API, which runs on port 35357.

In the `openrc` file you create, paste these values:

```
export OS_USERNAME=admin
export OS_TENANT_NAME=demo
export OS_PASSWORD=secrete
export OS_AUTH_URL=http://192.168.206.130:5000/v2.0/
export OS_REGION_NAME=RegionOne
```

Next, ensure these are used in your environment. If you see 401 Not Authorized errors on commands using tokens, ensure that you have properly sourced your credentials and that all the pipelines are accurate in the configuration files.


```
$ source ~/creds/openrc
```

Verify your credentials are working by using the **nova** client to list the available images:

```
$ nova image-list
```

ID	Name	Status	Server
21b421e5-44d4-4903-9db0-4f134fdd0793	tty-linux	ACTIVE	
7d9f0378-1640-4e43-8959-701f248d999d	tty-linux-ramdisk	ACTIVE	
599907ff-296d-4042-a671-d015e34317d2	tty-linux-kernel	ACTIVE	

Note that the ID values on your installation will be different.

Installing Additional Compute Nodes

There are many different ways to perform a multinode install of Compute in order to scale out your deployment and run more compute nodes, enabling more virtual machines to run simultaneously.

Ensure that the networking on each node is configured as documented in the [Pre-configuring the network](#) section.

In this case, you can install all the nova- packages and dependencies as you did for the Cloud Controller node, or just install nova-compute. Your installation can run any nova-services anywhere, so long as the service can access `nova.conf` so it knows where the Rabbitmq or Qpid messaging server is installed.

When running in a high-availability mode for networking, the compute node is where you configure the compute network, the networking between your instances. Learn more about high-availability for networking in the Compute Administration manual.

Because you may need to query the database from the compute node and learn more information about instances, the nova client and MySQL client or PostgreSQL client packages should be installed on any additional compute nodes.

Copy the `nova.conf` from your controller node to all additional compute nodes. As mentioned in the section entitled [Configuring OpenStack Compute](#), modify the following configuration options so that they match the IP address of the compute host:

- `my_ip`
- `vncserver_listen`
- `vncserver_proxyclient_address`

Adding Block Storage Nodes

To offer more storage to your tenant's VMs, add another volume node running cinder services by following these steps.

-
1. Install the required packages for cinder.
 2. Create a volume group called cinder-volumes (configurable using the `cinder_volume` parameter in `cinder.conf`).
 3. Configure `tgtd` with its `targets.conf` file and start the `tgtd` service.
 4. Connect the node to the Block Storage (cinder) database by configuring the `cinder.conf` file with the connection information.
 5. Make sure the `iscsi_ip_address` setting in `cinder.conf` matches the public IP of the node you're installing, then restart the cinder services.

When you issue a **cinder-manage host list** command you should see the new volume node listed. If not, look at the logs in `/var/log/cinder/volume.log` for issues.

8. Registering Virtual Machine Images

To test your deployment, download some virtual machine images that are known to work with OpenStack. CirrOS is a small test image that is often used for testing OpenStack deployments. You can find the most recent CirrOS image on the [CirrOS Launchpad home page](#) under "Downloads". As of this writing the most recent image is version 0.3.0. A 64-bit version in QCOW2 format (compatible with KVM or QEMU hypervisors) can be downloaded at https://launchpad.net/cirros/trunk/0.3.0/+download/cirros-0.3.0-x86_64-disk.img. Once you launch the image, log in with the following credentials:

- Username: `cirros`
- Password: `cubswin:)`

The 64-bit CirrOS QCOW2 image is the image we'll use for this walkthrough. More detailed information about how to obtain and create images can be found in the [OpenStack Compute Administration Guide](#) in the "Image Management" chapter.

Create a directory called `stackimages` to house your image files:

```
$ mkdir stackimages
```

Download the CirrOS image into your `stackimages` directory.

```
$ wget -c https://launchpad.net/cirros/trunk/0.3.0/+download/cirros-0.3.0-x86_64-disk.img -O stackimages/cirros.img
```

Verify that your **glance** client can access the Image service by requesting a list of installed images:

```
$ glance index
```

ID	Name	Disk
Format	Container Format	Size
-----	-----	-----
-----	-----	-----

If you get the following error, make sure that the environment variables set in `~/openrc`

```
Failed to show index. Got error:  
You are not authenticated.  
Details: 401 Unauthorized
```

```
This server could not verify that you are authorized to access the document  
you requested. Either you supplied the wrong credentials (e.g., bad  
password), or your browser does not understand how to supply the credentials  
required.
```

```
Authentication required
```

Add the CirrOS image to the Image service using the **glance image-create** command, passing the image file through standard input:

```
$ glance image-create --name=cirros-0.3.0-x86_64 --disk-format=qcow2 --  
container-format=bare < stackimages/cirros.img  
Added new image with ID: f4add24-4e8a-46bb-b15d-fae2591f1a35
```

**Note**

The returned image ID is generated dynamically, and therefore will be different on your deployment than in this example.

The rationale for the arguments is:

`name=cirros-0.3.0-x86_64` The `name` field is an arbitrary label. In this example the name encodes the distribution, version, and architecture: `cirros-0.3.0-x86_64`.

`disk-format=qcow2` The `disk-format` field specifies the format of the image file. In this case, the image file format is QCOW2, which can be verified using the **file** command:

```
$ file stackimages/cirros.  
img  
stackimages/cirros.img: QEMU QCOW Image (v2),  
41126400 bytes
```

Other valid formats are `raw`, `vhd`, `vmdk`, `vdi`, `iso`, `aki`, `ari` and `ami`.

`container-format=bare` The `container-format` field is required by the **glance image-create** command but isn't actually used by any of the OpenStack services, so the value specified here has no effect on system behavior. We specify `bare` to indicate that the image file is not in a file format that contains metadata about the virtual machine.

Because the value is not used anywhere, it safe to always specify `bare` as the container format, although the command will accept other formats: `ovf`, `aki`, `ari`, `ami`.

Confirm it was uploaded by listing the images in the Image service:

```
$ glance index  
ID                                     Name                                     Disk  
Format                               Container Format                               Size  
-----  
f4addd24-4e8a-46bb-b15d-fae2591f1a35 cirros-0.3.0-x86_64                               qcow2  
                                     bare                               9761280
```

The **nova image-list** command will also list the images in the Image service:

```
$ nova image-list  
+-----+  
+-----+-----+-----+  
+-----+  
|      ID      |      Name      |  
+-----+  
| Status |      Server      |  
+-----+  
+-----+-----+-----+  
+-----+  
+-----+-----+-----+
```

```
| f4addd24-4e8a-46bb-b15d-fae2591f1a35 | cirros-0.3.0-x86_64  
| ACTIVE |  
+-----+  
+-----+  
+-----+
```

9. Running Virtual Machine Instances

Table of Contents

Security groups: Enabling SSH and ICMP (ping)	71
Adding a keypair	71
Confirm all services running	72
Starting an instance	73
Bringing down an instance	76

Security groups: Enabling SSH and ICMP (ping)

The Compute service uses the concept of security groups to control what network protocols (TCP, UDP, ICMP), ports, and IP addresses are permitted to access instances. Each tenant manages its own list of security groups and starts off with a security group called `default`. If no security group is specified upon boot, the virtual machine will be associated with the `default` security group.

Security groups can be listed by the `nova secgroup-list` command.

```
$ nova secgroup-list
+-----+-----+
| Name | Description |
+-----+-----+
| default | default |
+-----+-----+
```

In this example, we will use the `nova secgroup-add-rule` command to enable access to TCP port 22 (so we can SSH to instances) Allow access to port 22 from all IP addresses (specified in CIDR notation as `0.0.0.0/0`) with the following command:

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

When specifying rules for TCP and UDP protocols, you may specify a range of port consecutive addresses in a single rule (e.g., from port 5901 to port 5999). In this case, only a single port is being enabled, so we specify the start port as 22 and the end port as 22.

To be able to ping virtual machine instances, you must specify a rule to allow ICMP traffic. When specifying ICMP rules, instead of specifying a begin and end port, you specify a permitted ICMP code and ICMP type. You can also specify `-1` for the code to enable all codes and `-1` for the type to enable all ICMP types. Allow access to all codes and types of ICMP traffic from all IP addresses with the following command:

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

Adding a keypair

The Compute service can inject an SSH public key into an account on the instance, assuming the virtual machine image being used supports this. To add a keypair to the Compute service, use the `nova keypair-add` command. This command can be used to either generate

a new keypair, or to upload an existing public key. The following example uploads an existing public key, located at `~/.ssh/id_rsa.pub`, and gives the keypair the name `mykey`.

```
$ nova keypair-add --pub_key ~/.ssh/id_rsa.pub mykey
```

List the keypairs by doing:

```
$ nova keypair-list
```

Name	Fingerprint
mykey	c3:d2:b5:d3:ec:4a:29:b0:22:32:6e:34:dd:91:f9:cf

Confirm that the uploaded keypair matches your local key by checking your key's fingerprint with the `ssh-keygen` command:

```
$ ssh-keygen -l -f ~/.ssh/id_rsa.pub
2048 c3:d2:b5:d3:ec:4a:29:b0:22:32:6e:34:dd:91:f9:cf /home/myaccount/.ssh/
id_rsa.pub (RSA)
```

Confirm all services running

Before trying to start an instance, confirm that all of the necessary services are running, in particular:

<code>libvirtd</code>	The <code>libvirtd</code> service must be running because the <code>nova-compute</code> service interacts with it. This only applies when using a hypervisor that is managed by <code>libvirt</code> (e.g., KVM, QEMU, LXC).
<code>nova-api</code>	The <code>nova-api</code> service must be running to respond to the request to boot an instance, as well as to serve as the metadata server so that the instance can retrieve the public key uploaded in a previous section. If the <code>nova</code> commands in the previous section succeeded, then the service is running.
<code>nova-scheduler</code>	The <code>nova-scheduler</code> service must be running in order to dispatch requests for a new virtual machine instance to a host running the <code>nova-compute</code> service that has sufficient resources.
<code>nova-compute</code>	The <code>nova-compute</code> service must be running in order to interact with the hypervisor to bring up a virtual machine instance.
<code>nova-network</code>	The <code>nova-network</code> service must be running in order to perform networking tasks such as assigning an IP address to the virtual machine instance and implementing the security group rules.

The `nova-manage service list` command can be used to confirm that these services are running properly.



Note

The `nova-manage service list` command does not indicate whether the `nova-api` service is running.

As root:

# nova-manage service list			
Binary	Host	Zone	Status
State	Updated_At		
nova-compute	myhost-1	nova	enabled
:-)	2012-05-27 12:36:35		
nova-network	myhost-1	nova	enabled
:-)	2012-05-27 12:36:28		
nova-scheduler	myhost-1	nova	enabled
:-)	2012-05-27 12:36:33		

If any of the services are missing in your configuration, or the `State` column does not show a smiley face, then your `Compute` service will not be able to launch an instance.

Starting an instance

To start an instance, we need to specify a *flavor*, also known as an *instance type*, which indicates the size of an instance. Use the **nova flavor-list** command to view the list of available flavors:

\$ nova flavor-list							
ID	Name	Memory_MB	Disk	Ephemeral	Swap	VCPUs	RXTX_Factor
1	m1.tiny	512	0	0		1	1.0
2	m1.small	2048	15	15		1	1.0
3	m1.medium	4096	25	25		2	1.0
4	m1.large	8192	45	45		4	1.0
5	m1.xlarge	16384	85	85		8	1.0

We also need to specify the image. Use the **nova image-list** to retrieve the ID of the CirrOS image.

```
$ nova image-list
+-----+
+-----+-----+
+-----+
|          ID          |          Name          |
| Status |          Server          |
+-----+-----+
+-----+-----+
| f4addd24-4e8a-46bb-b15d-fae2591f1a35 | cirros-0.3.0-x86_64 |
| ACTIVE |          |
+-----+-----+
+-----+-----+
+-----+-----+
+-----+-----+
```

Use the `nova boot` command to launch a new virtual machine instance. We'll use an `m1.small` instance in this example, using the CirrOS image, and the `mykey` keypair we added. We also need to give this virtual machine instance a name, we'll call it `cirros`. We will explicitly specify the `default` security group in this example, although this isn't strictly necessary since the `default` group will be used if no security group is specified.

```
$ nova boot --flavor 2 --image f4add24-4e8a-46bb-b15d-fae2591f1a35 --key_name
mykey --security_group default cirros
+-----+
+-----+
```


Property	Value
OS-DCF:diskConfig	MANUAL
OS-EXT-SRV-ATTR:host	host-1
OS-EXT-SRV-ATTR:hypervisor_hostname	None
OS-EXT-SRV-ATTR:instance_name	instance-00000001
OS-EXT-STS:power_state	0
OS-EXT-STS:task_state	scheduling
OS-EXT-STS:vm_state	building
accessIPv4	
accessIPv6	
adminPass	RG3W2bpZDbCo
config_drive	
created	2012-05-27T13:00:33Z
flavor	m1.small
hostId	
a2fd457e034c030506bac5c790c38d9519ea7a03b6861474a712c6b7	
id	c6bbbf26-b40a-47e7-8d5c-eb17bf65c485
image	cirros-0.3.0-x86_64
key_name	mykey
metadata	{}
name	cirros
progress	0
status	BUILD
tenant_id	b5815b046cfe47bb891a7b64119e7f80
updated	2012-05-27T13:00:33Z
user_id	a4c2d43f80a549a19864c89d759bb3fe

Check the progress of the instance with the **nova list** command. When the instance has booted, the command output will look something like:

```
$ nova list
```

ID	Name	Status
Networks		
c6bbbf26-b40a-47e7-8d5c-eb17bf65c485 private=192.168.100.5	cirros	ACTIVE

You can view the boot messages of the instances using the **nova console-log** command:

```
$ nova console-log
...
Starting network...
udhcpd (v1.18.5) started
Sending discover...
Sending select for 192.168.100.5...
Lease of 192.168.100.5 obtained, lease time 120
deleting routers
route: SIOCDELRT: No such process
adding dns 192.168.100.4
cloud-setup: checking http://169.254.169.254/2009-04-04/meta-data/instance-id
cloud-setup: successful after 1/30 tries: up 1.45. iid=i-00000001
Starting dropbear sshd: generating rsa key... generating dsa key... OK
===== cloud-final: system completely up in 1.77 seconds =====
  instance-id: i-00000001
  public-ipv4:
  local-ipv4 : 192.168.100.5
cloud-userdata: user data not a script

  / _/  _ _ _ _ _ _ _ _ _ _ \ _/ \ _/
 / _/  / / _ _ _ _ _ _ _ _ _ _ \ _/ \ _/
 \ _/  / / _ _ _ _ _ _ _ _ _ _ \ _/ \ _/
      http://launchpad.net/cirros

login as 'cirros' user. default password: 'cubswin:'). use 'sudo' for root.
cirros login:
```

You should be able to ping your instance:

```
$ ping -c5 192.168.100.5
PING 192.168.100.5 (192.168.100.5) 56(84) bytes of data.
64 bytes from 192.168.100.5: icmp_req=1 ttl=64 time=0.270 ms
64 bytes from 192.168.100.5: icmp_req=2 ttl=64 time=0.228 ms
64 bytes from 192.168.100.5: icmp_req=3 ttl=64 time=0.244 ms
64 bytes from 192.168.100.5: icmp_req=4 ttl=64 time=0.203 ms
64 bytes from 192.168.100.5: icmp_req=5 ttl=64 time=0.210 ms

--- 192.168.100.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.203/0.231/0.270/0.024 ms
```

You should be able to ssh to your instance as the **cirros** user, using either the ssh keypair you uploaded or using the password **cubswin:**)

```
$ ssh cirros@192.168.100.5
The authenticity of host '192.168.100.5 (192.168.100.5)' can't be established.
```

```
RSA key fingerprint is c2:0a:95:d4:e7:e1:a6:a2:6a:99:4d:b8:f9:66:13:64.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added '192.168.100.5' (RSA) to the list of known hosts.  
cirros@192.168.100.5's password: cubswin:)  
$
```

Bringing down an instance

Bring down your instance using the **nova delete** command:

```
$ nova delete c6bbbf26-b40a-47e7-8d5c-eb17bf65c485
```

10. Installing OpenStack Object Storage

Table of Contents

System Requirements	77
Object Storage Network Planning	78
Example Object Storage Installation Architecture	78
Installing OpenStack Object Storage on Ubuntu	79
Before You Begin	79
General Installation Steps	80
Installing and Configuring the Storage Nodes	80
Installing and Configuring the Proxy Node	82
Start the Storage Nodes Services	85
OpenStack Object Storage Post Installation	85
Verify the Installation	85
Adding an Additional Proxy Server	86

The OpenStack Object Storage services work together to provide object storage and retrieval through a REST API.

System Requirements

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

Table 10.1. Hardware Recommendations

Server	Recommended Hardware	Notes
Object Storage object servers	<p>Processor: dual quad core</p> <p>Memory: 8 or 12 GB RAM</p> <p>Disk space: optimized for cost per GB</p> <p>Network: one 1 GB Network Interface Card (NIC)</p>	<p>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.</p> <p>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.</p>
Object Storage container/account servers	<p>Processor: dual quad core</p> <p>Memory: 8 or 12 GB RAM</p> <p>Network: one 1 GB Network Interface Card (NIC)</p>	Optimized for IOPS due to tracking with SQLite databases.
Object Storage proxy server	<p>Processor: dual quad core</p> <p>Network: one 1 GB Network Interface Card (NIC)</p>	<p>Higher network throughput offers better performance for supporting many API requests.</p> <p>Optimize your proxy servers for best CPU performance. The Proxy Services are more CPU and network I/O intensive. If you are using</p>

Server	Recommended Hardware	Notes
		10g networking to the proxy, or are terminating SSL traffic at the proxy, greater CPU power will be required.

Operating System: OpenStack Object Storage currently runs on Ubuntu, RHEL, CentOS, or Fedora and the large scale deployment at Rackspace runs on Ubuntu 10.04 LTS.

Networking: 1000 Mbps are suggested. 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.

Object Storage Network Planning

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 document refers to two networks. One is a Public Network for connecting to the Proxy server, and the second is a Storage Network that is not accessible from outside the cluster, to which all of the nodes are connected. 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.

Public Network (Publicly routable IP range): This network is utilized for providing Public IP accessibility to the API endpoints within the cloud infrastructure.

Minimum size: 8 IPs (CIDR /29)

Storage Network (RFC1918 IP Range, not publicly routable): This network is utilized for all inter-server communications within the Object Storage infrastructure.

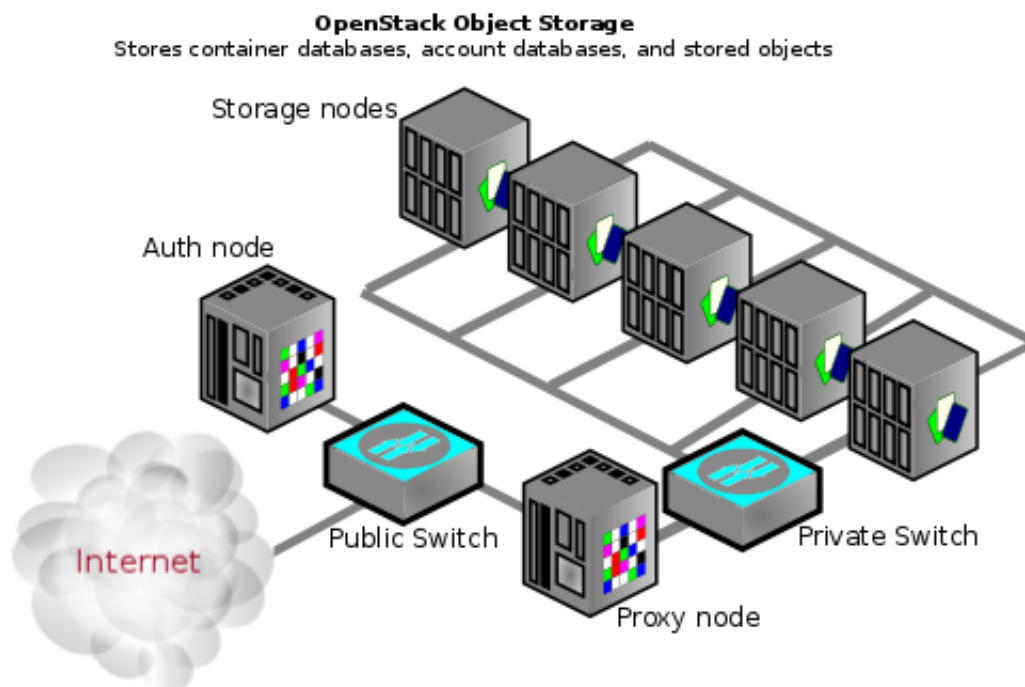
Recommended size: 255 IPs (CIDR /24)

Example Object Storage Installation Architecture

- node - a host machine running one or more OpenStack Object Storage services
- Proxy node - node that runs Proxy services
- Auth node - an optionally separate node that runs the Auth service separately from the Proxy services
- Storage node - node that runs Account, Container, and Object services
- Ring - a set of mappings of OpenStack Object Storage data to physical devices

To increase reliability, you may want to add additional Proxy servers for performance.

This document describes each Storage node as a separate zone in the ring. It is recommended to have a minimum of 5 zones. 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.



Installing OpenStack Object Storage on Ubuntu

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

If you would like to perform a single node installation on Ubuntu for development purposes from source code, use the Swift All In One instructions or DevStack. See http://swift.openstack.org/development_saio.html for manual instructions or <http://devstack.org> for all-in-one including authentication and a dashboard.

Before You Begin

Have a copy of the Ubuntu Server installation media on hand if you are installing on a new server.

This document demonstrates installing a cluster using the following types of nodes:

- One Proxy node which runs the swift-proxy-server processes and may also run the optional swauth or tempauth services, this walkthrough uses the Identity service code-

named Keystone. The proxy server serves proxy 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 5 is recommended for a production cluster.

General Installation Steps

1. Install the baseline operating system, such as Ubuntu Server (12.04) or RHEL, CentOS, or Fedora, on all nodes.
2. Install core Swift files and openSSH.

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

3. Create and populate configuration directories on all nodes:

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

4. Create /etc/swift/swift.conf:

```
[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, proxy node, and an auth node, in this walkthrough we'll use the OpenStack Identity Service, Keystone, for the common auth piece.

Installing and Configuring the Storage Nodes



Note

OpenStack Object Storage should work on any modern filesystem that supports Extended Attributes (XATTRS). We currently recommend XFS as it demonstrated the best overall performance for the swift use case after considerable testing and benchmarking at Rackspace. It is also the only filesystem that has been thoroughly tested.

1. Install Storage node packages:

```
# yum install openstack-swift-account openstack-swift-container openstack-  
swift-object xfsprogs
```

2. For every device on the node you wish to use for storage, setup 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 -i size=1024 /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. Edit the following line in `/etc/default/rsync`:

```
RSYNC_ENABLE = true
```

5. Start rsync daemon:

```
# service rsync start
```

**Note**

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

-
6. Create the swift recon cache directory and set its permissions.

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

Installing and Configuring 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

It is assumed that all commands are run as the root user.

1. Install swift-proxy service:

```
# yum install openstack-swift-proxy memcached openstack-utils python-
swiftclient python-keystone-auth-token
```

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. Preferably this should be on a local, non-public network. Edit the following line in `/etc/memcached.conf`, changing:

```
-l 127.0.0.1
to
-l <PROXY_LOCAL_NET_IP>
```

4. Restart the memcached server:

```
# service memcached restart
```

5. RHEL/CentOS/Fedora Only: To set up Object Storage to authenticate tokens we need to set the keystone Admin token in the swift proxy file with the `openstack-config` command.

```
# openstack-config --set /etc/swift/proxy-server.conf filter:authtoken
admin_token $ADMIN_TOKEN
# sudo openstack-config --set /etc/swift/proxy-server.conf filter:authtoken
auth_token $ADMIN_TOKEN
```

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 like:

```
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 is basically creating a builder file with a few parameters. The parameter with the value of 18 represents 2^{18} , the value that the partition will be sized to. Set this “partition power” value based on the total amount of storage you expect your entire ring to use.

The value of 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/<DEVICE> 100
# swift-ring-builder container.builder add z<ZONE>-
<STORAGE_LOCAL_NET_IP_1>:6001/<DEVICE> 100
# swift-ring-builder object.builder add z<ZONE>-
<STORAGE_LOCAL_NET_IP_1>:6000/<DEVICE> 100
```

For example, if you were setting up a storage node with a partition in Zone 1 on IP 10.0.0.1. 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 would look like:

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



Note

Assuming there are 5 zones with 1 node per zone, ZONE should start at 1 and increment by one for each additional node.

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 all the config files are owned by the swift user:

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

14 Start Proxy services:

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

Start the Storage Nodes Services

Now that the ring files are on each storage node the services can be started. On each storage node run the following:

```
# 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-updater start
# service swift-account-auditor start
```

OpenStack Object Storage Post Installation

Verify the Installation

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

1. First, export the swift admin password (setup previously) in a variable so it can be re-used.

```
$ export ADMINPASS=secrete
```



Note

If you do not wish to have the swift admin password stored in your shell's history, you may perform the following:

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

2. Run the swift CLI, swift, with the correct Identity service URL. Export the information for ADMINPASS using `$ export ADMINPASS=secrete`.

```
$ swift -V 2.0 -A http://<AUTH_HOSTNAME>:5000/v2.0 -U demo:admin -K
$ADMINPASS stat
```

3. Get an X-Storage-Url and X-Auth-Token:

```
$ curl -k -v -H 'X-Storage-User: demo:admin' -H 'X-Storage-Pass: $ADMINPASS'
http://<AUTH_HOSTNAME>:5000/auth/v2.0
```

4. Check that you can HEAD the account:

```
$ curl -k -v -H 'X-Auth-Token: <token-from-x-auth-token-above>' <url-from-x-
storage-url-above>
```

5. Use swift to upload a few files named 'bigfile[1-2].tgz' to a container named 'myfiles':

```
$ swift -A http://<AUTH_HOSTNAME>:5000/v2.0 -U demo:admin -K $ADMINPASS
upload myfiles bigfile1.tgz
$ swift -A http://<AUTH_HOSTNAME>:5000/v2.0 -U demo:admin -K $ADMINPASS
upload myfiles bigfile2.tgz
```

6. Use swift to download all files from the 'myfiles' container:

```
$ swift -A http://<AUTH_HOSTNAME>:5000/v2.0 -U demo:admin -K $ADMINPASS
download myfiles
```



Note

If you are using swauth in preference to the OpenStack Identity service, you should use the `default_swift_cluster` variable to connect to your swift cluster. Please follow the [swauth documentation](#) to verify your installation.

Adding an Additional Proxy Server

For reliability's sake you may want to have more than one proxy server. You can set up the additional proxy node in the same manner that you set up the first proxy node but with additional configuration steps.

Once you have more than two proxies, you also want to load balance between the two, which means 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 for the initial setup, and then follow these additional steps for more proxy servers.

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

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

in each proxy server's conf file.:

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

2. Next, copy all the ring information to all the nodes, including your new proxy nodes, and ensure the ring info gets to all the storage nodes as well.
3. After you sync all the nodes, make sure the admin has the keys in `/etc/swift` and the ownership for the ring file is correct.



Note

If you are using `swauth` in preference to the OpenStack Identity service, there are additional steps to follow for the addition of a second proxy server. Please follow the [swauth documentation](#) Installation section, paying close attention to the `default_swift_cluster` variable.

11. Installing the OpenStack Dashboard

Table of Contents

About the Dashboard	88
System Requirements for the Dashboard	88
Installing the OpenStack Dashboard	89
Configuring the Dashboard	90
Validating the Dashboard Install	93
How To Custom Brand The OpenStack Dashboard (Horizon)	94
OpenStack Dashboard Session Storage	96
Local Memory Cache	97
Memcached	97
Database	97
Cached Database	98
Cookies	99
Overview of VNC Proxy	99
About nova-consoleauth	100
Typical Deployment	100
Frequently asked questions about VNC access to VMs	103

OpenStack has components that provide a view of the OpenStack installation such as a Django-built website that serves as a dashboard.

About the Dashboard

You can use a dashboard interface with an OpenStack Compute installation with a web-based console provided by the Openstack-Dashboard project. It provides web-based interactions with the OpenStack Compute cloud controller through the OpenStack APIs. For more information about the Openstack-Dashboard project, please visit: <https://github.com/openstack/horizon/>. These instructions are for an example deployment configured with an Apache web server.

System Requirements for the Dashboard

Because Apache does not serve content from a root user, you must use another user with sudo privileges and run as that user.

You should have a running OpenStack Compute installation with the Identity Service, Keystone, enabled for identity management.

The dashboard needs to be installed on the node that can contact the Identity Service.

You should know the URL of your Identity endpoint and the Compute endpoint.

You must know the credentials of a valid Identity service user.

You must have git installed. It's straightforward to install it with `sudo apt-get install git-core`.

Python 2.6 is required, and these instructions have been tested with Ubuntu 10.10. It should run on any system with Python 2.6 or 2.7 that is capable of running Django including Mac OS X (installing prerequisites may differ depending on platform).

Optional components:

- An Image Store (*Glance*) endpoint.
- An Object Store (*Swift*) endpoint.
- A [Quantum](#) (networking) endpoint.

Installing the OpenStack Dashboard

Here are the overall steps for creating the OpenStack dashboard. Details about deployment are available at [Deploying Horizon](#).

1. Install the OpenStack Dashboard framework including Apache and related modules.
2. Configure the Dashboard.
3. Restart and run the Apache server.

Install the OpenStack Dashboard, as root:

```
# yum install -y memcached python-memcached mod_wsgi openstack-dashboard
```

Next, modify the variable `CACHE_BACKEND` in `/etc/openstack-dashboard/local_settings` to match the ones set in `/etc/sysconfig/memcached.conf`. Open `/etc/openstack-dashboard/local_settings` and look for this line:

```
CACHE_BACKEND = 'memcached://127.0.0.1:11211/'
```



Note

The address and port in the new value need to be equal to the ones set in `/etc/sysconfig/memcached`.

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



Note

This guide has selected memcache as a session store for OpenStack Dashboard. There are other options available, each with benefits and drawbacks. Refer to the OpenStack Dashboard Session Storage section for more information.



Note

In order to change the timezone you can use either `dashboard` or inside `/etc/openstack-dashboard/local_settings` you can change below mentioned parameter.

```
TIME_ZONE = "UTC"
```


Configuring the Dashboard

1. Simple deployment (HTTP)

Specify the host for your OpenStack Identity Service endpoint in the `/etc/openstack-dashboard/local_settings.py` file with the `OPENSTACK_HOST` setting. An example is included:

```
import os

from django.utils.translation import ugettext_lazy as _

DEBUG = False
TEMPLATE_DEBUG = DEBUG
PROD = True
USE_SSL = False

SITE_BRANDING = 'OpenStack Dashboard'

# Ubuntu-specific: Enables an extra panel in the 'Settings' section
# that easily generates a Juju environments.yaml for download,
# preconfigured with endpoints and credentials required for bootstrap
# and service deployment.
ENABLE_JUJU_PANEL = True

# Note: You should change this value
SECRET_KEY = 'eljlIWlLoWHgryYxFT6j7cM5fGOOxWY0'

# Specify a regular expression to validate user passwords.
# HORIZON_CONFIG = {
#     "password_validator": {
#         "regex": '.*',
#         "help_text": _("Your password does not meet the requirements.")
#     }
# }

LOCAL_PATH = os.path.dirname(os.path.abspath(__file__))

CACHE_BACKEND = 'memcached://127.0.0.1:11211/'

# Send email to the console by default
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
# Or send them to /dev/null
#EMAIL_BACKEND = 'django.core.mail.backends.dummy.EmailBackend'

# Configure these for your outgoing email host
# EMAIL_HOST = 'smtp.my-company.com'
# EMAIL_PORT = 25
# EMAIL_HOST_USER = 'djangomail'
# EMAIL_HOST_PASSWORD = 'top-secret!'

# For multiple regions uncomment this configuration, and add (endpoint,
# title).
# AVAILABLE_REGIONS = [
#     ('http://cluster1.example.com:5000/v2.0', 'cluster1'),
#     ('http://cluster2.example.com:5000/v2.0', 'cluster2'),
# ]
```

```

OPENSTACK_HOST = "127.0.0.1"
OPENSTACK_KEYSTONE_URL = "http://%s:5000/v2.0" % OPENSTACK_HOST
OPENSTACK_KEYSTONE_DEFAULT_ROLE = "Member"

# The OPENSTACK_KEYSTONE_BACKEND settings can be used to identify the
# capabilities of the auth backend for Keystone.
# If Keystone has been configured to use LDAP as the auth backend then set
# can_edit_user to False and name to 'ldap'.
#
# TODO(tres): Remove these once Keystone has an API to identify auth
# backend.
OPENSTACK_KEYSTONE_BACKEND = {
    'name': 'native',
    'can_edit_user': True
}

# OPENSTACK_ENDPOINT_TYPE specifies the endpoint type to use for the
# endpoints
# in the Keystone service catalog. Use this setting when Horizon is running
# external to the OpenStack environment. The default is 'internalURL'.
OPENSTACK_ENDPOINT_TYPE = "publicURL"

# The number of Swift containers and objects to display on a single page
# before
# providing a paging element (a "more" link) to paginate results.
API_RESULT_LIMIT = 1000

# If you have external monitoring links, eg:
# EXTERNAL_MONITORING = [
#     ['Nagios', 'http://foo.com'],
#     ['Ganglia', 'http://bar.com'],
# ]

LOGGING = {
    'version': 1,
    # When set to True this will disable all logging except
    # for loggers specified in this configuration dictionary. Note that
    # if nothing is specified here and disable_existing_loggers is True,
    # django.db.backends will still log unless it is disabled
    # explicitly.
    'disable_existing_loggers': False,
    'handlers': {
        'null': {
            'level': 'DEBUG',
            'class': 'django.utils.log.NullHandler',
        },
        'console': {
            # Set the level to "DEBUG" for verbose output logging.
            'level': 'INFO',
            'class': 'logging.StreamHandler',
        },
    },
    'loggers': {
        # Logging from django.db.backends is VERY verbose, send to null
        # by default.
        'django.db.backends': {
            'handlers': ['null'],
            'propagate': False,
        },
        'horizon': {

```

```
        'handlers': ['console'],
        'propagate': False,
    },
    'novaclient': {
        'handlers': ['console'],
        'propagate': False,
    },
    'keystoneclient': {
        'handlers': ['console'],
        'propagate': False,
    },
    'nose.plugins.manager': {
        'handlers': ['console'],
        'propagate': False,
    }
}
```

The `HORIZON_CONFIG` dictionary contains all the settings for the Dashboard. Whether or not a service is in the Dashboard depends on the Service Catalog configuration in the Identity service. Refer to [Horizon Settings and Configuration](#) for the full listing.

2. Secured deployment (HTTPS)

While the standard installation uses a non-encrypted channel (HTTP), it is possible to enable the SSL support for the OpenStack Dashboard. In the following example, we use the domain "http://openstack.example.com", make sure to use one that fits your current setup.

- In `/etc/openstack-dashboard/local_settings.py` update the following directive:

```
USE_SSL = True
```

- Edit `/etc/apache2/ports.conf` and add the following line:

```
NameVirtualHost *:443
```

- Edit `/etc/apache2/conf.d/openstack-dashboard.conf`:

Before:

```
WSGIScriptAlias / /usr/share/openstack-dashboard/openstack_dashboard/wsgi/
django.wsgi
WSGIDaemonProcess horizon user=www-data group=www-data processes=3
    threads=10
Alias /static /usr/share/openstack-dashboard/openstack_dashboard/static/
<Directory /usr/share/openstack-dashboard/openstack_dashboard/wsgi>
    Order allow,deny
    Allow from all
</Directory>
```

After:

```
<VirtualHost *:80>
    ServerName openstack.example.com
    RedirectPermanent / https://openstack.example.com
</VirtualHost>
```

```
<VirtualHost *:443>
    ServerName openstack.example.com

    SSLEngine On
    SSLCertificateFile /etc/apache2/SSL/openstack.example.com.crt
    SSLCACertificateFile /etc/apache2/SSL/openstack.example.com.crt
    SSLCertificateKeyFile /etc/apache2/SSL/openstack.example.com.key
    SetEnvIf User-Agent ".*MSIE.*" nokeepalive ssl-unclean-shutdown

    WSGIScriptAlias / /usr/share/openstack-dashboard/openstack_dashboard/
wsgi/django.wsgi
    WSGIDaemonProcess horizon user=www-data group=www-data processes=3
    threads=10
    Alias /static /usr/share/openstack-dashboard/openstack_dashboard/
static/
    <Directory /usr/share/openstack-dashboard/openstack_dashboard/wsgi>
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>
```

In this configuration, we instruct Apache to listen on the port 443 and to redirect all the hits to the HTTPs protocol for all the non-secured requests. In the secured section, we define as well the private key, the public key, and the certificate to use.

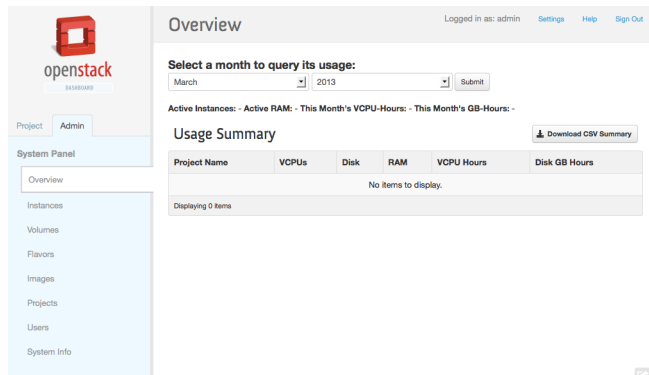
- Finally restart Apache and memcached:

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

You should now be redirected to the HTTPs page if you call the HTTP version of the dashboard via your browser.

Validating the Dashboard Install

To validate the Dashboard installation, point your browser at <http://192.168.206.130>. Once you connect to the Dashboard with the URL, you should see a login window. Enter the credentials for users you created with the Identity Service, Keystone. For example, enter "admin" for the username and "secrete" as the password.

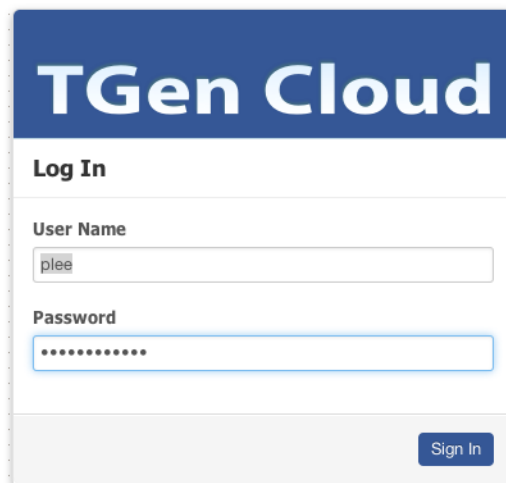


How To Custom Brand The OpenStack Dashboard (Horizon)

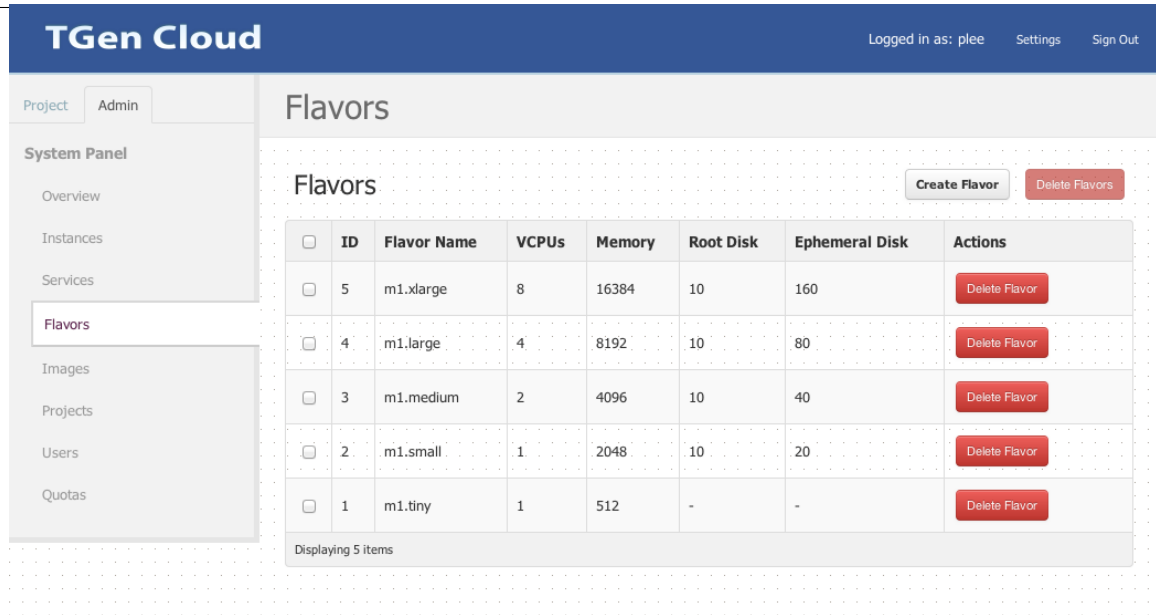
Adapted from a [blog post by Preston Lee](#).

When deploying OpenStack on [Ubuntu Server 12.04](#), you can have the `openstack-dashboard` package installed to provide the web-based “Horizon” GUI component. Canonical also provides an `openstack-dashboard-ubuntu-theme` package that brands the Python-based Django GUI.

The [Horizon documents](#) briefly mention [branding customization](#) to give you a head start, but here are more specific steps. Here’s a custom-branded Horizon dashboard with custom colors, logo, and site title:



The image shows a web-based login interface for a custom-branded OpenStack Horizon dashboard. The header is a dark blue bar with the text "TGen Cloud" in white. Below the header is a white box with the title "Log In". Inside this box, there are two input fields: "User Name" with the text "plee" and "Password" with masked characters "*****". A blue "Sign In" button is located at the bottom right of the white box. The entire interface is set against a light gray background with a subtle dot pattern.



Once you know where to make the appropriate changes, it's super simple. Step-by-step:

1. Create a graphical logo with a transparent background. The text "TGen Cloud" in this example is actually rendered via .png files of multiple sizes created with a graphics program. Use a 200×27 for the logged-in banner graphic, and 365×50 for the login screen graphic.
2. Set the HTML title (shown at the top of the browser window) by adding the following line to `/etc/openstack-dashboard/local_settings.py`: `SITE_BRANDING = "Example, Inc. Cloud"`
3. Upload your new graphic files to:

```
/usr/share/openstack-dashboard/openstack_dashboard/static/dashboard/img/
```

4. Create a new CSS stylesheet — we'll call ours `custom.css` — in the directory:

```
/usr/share/openstack-dashboard/openstack_dashboard/static/dashboard/css/
```

5. Edit your CSS file using the following as a starting point for customization, which simply overrides the Ubuntu customizations made in the `ubuntu.css` file.

Change the colors and image file names as appropriate, though the relative directory paths should be the same.

```
/*
 * New theme colors for dashboard that override the defaults:
 * dark blue: #355796 / rgb(53, 87, 150)
 * light blue: #BAD3E1 / rgb(186, 211, 225)
 *
 * By Preston Lee <plee@tgen.org>
 */
h1.brand {
background: #355796 repeat-x top left;
border-bottom: 2px solid #BAD3E1;
```

```
}
h1.brand a {
background: url(../img/my_cloud_logo_small.png) top left no-repeat;
}
#splash .login {
background: #355796 url(../img/my_cloud_logo_medium.png) no-repeat center
35px;
}
#splash .login .modal-header {
border-top: 1px solid #BAD3E1;
}
.btn-primary {
background-image: none !important;
background-color: #355796 !important;
border: none !important;
box-shadow: none;
}
.btn-primary:hover,
.btn-primary:active {
border: none;
box-shadow: none;
background-color: #BAD3E1 !important;
text-decoration: none;
}
```

6. Open the following HTML template in an editor:

```
/usr/share/openstack-dashboard/openstack_dashboard/templates/_stylesheets.
html
```

7. Add a line to include your new stylesheet pointing to custom.css: (I've highlighted the new line in *bold*.)

```
...
<link href='{{ STATIC_URL }}bootstrap/css/bootstrap.min.css' media='screen'
rel='stylesheet' />
<link href='{{ STATIC_URL }}dashboard/css/{% choose_css %}' media='screen'
rel='stylesheet' />
<link href='{{ STATIC_URL }}dashboard/css/custom.css' media='screen' rel=
'stylesheet' />
...
```

8. Restart apache just for good measure: `sudo service httpd restart`
9. Reload the dashboard in your browser and fine tune your CSS appropriate.

You're done!

OpenStack Dashboard Session Storage

Horizon uses [Django's sessions framework](#) for handling user session data; however that's not the end of the story. There are numerous session backends available, which are controlled through the `SESSION_ENGINE` setting in your `/etc/openstack-dashboard/local_settings` file. What follows is a quick discussion of the pros and cons of each of the common options as they pertain to deploying Horizon specifically.

Local Memory Cache

Local memory storage is the quickest and easiest session backend to set up, as it has no external dependencies whatsoever. However, it has two significant drawbacks:

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

The local memory backend 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'
}
```

Memcached

External caching using an application such as memcached offers persistence and shared storage, and can be very useful for small-scale deployment and/or development. However, for distributed and high-availability scenarios memcached has inherent problems which are beyond the scope of this documentation.

Memcached is an extremely fast and efficient cache backend for cases where it fits the deployment need, but it's not appropriate for all scenarios.

Requirements:

1. Memcached service running and accessible.
2. Python memcached module installed.

Enabled by:

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

Database

Database-backed sessions are scalable (using an appropriate database strategy), persistent, and can be made high-concurrency and highly-available.

The downside to this approach is that database-backed sessions are one of the slower session storages, and incur a high overhead under heavy usage. Proper configuration of your database deployment can also be a substantial undertaking and is far beyond the scope of this documentation. To enable, follow the below steps to initialise the database and configure it for use

Start the mysql command line client by running:

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

Enter the MySQL root user's password when prompted.

To configure the MySQL database, create the dash database.

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

Create a MySQL user for the newly-created dash database that has full control of the database.

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

Enter quit at the mysql> prompt to exit MySQL.

In the `/etc/openstack-dashboard/local_settings` file, 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': 'yourpassword',  
        'HOST': 'localhost',  
        'default-character-set': 'utf8'  
    }  
}
```

After configuring the `/etc/openstack-dashboard/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, you should see the following at the end of what returns:

```
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.
```

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

```
# service httpd restart
```

Cached Database

To mitigate the performance issues of database queries, you can also consider using Django's `cached_db` session backend which utilizes both your database and caching infrastructure to perform write-through caching and efficient retrieval. You can enable this hybrid setting by configuring both your database and cache as discussed above and then using:

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

Cookies

If you're using Django 1.4 or later, a new session backend is available to you which avoids server load and scaling problems: the `signed_cookies` backend!

This backend stores session data in a cookie which is stored by the user's browser. The backend 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 session engine are that it doesn't require any 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 which can be stored.

For a thorough discussion of the security implications of this session backend, please read the Django documentation on [cookie-based sessions](#).

Overview of VNC Proxy

The VNC Proxy is an OpenStack component that allows users of the Compute service to access their instances through VNC clients.

The VNC console connection works as follows:

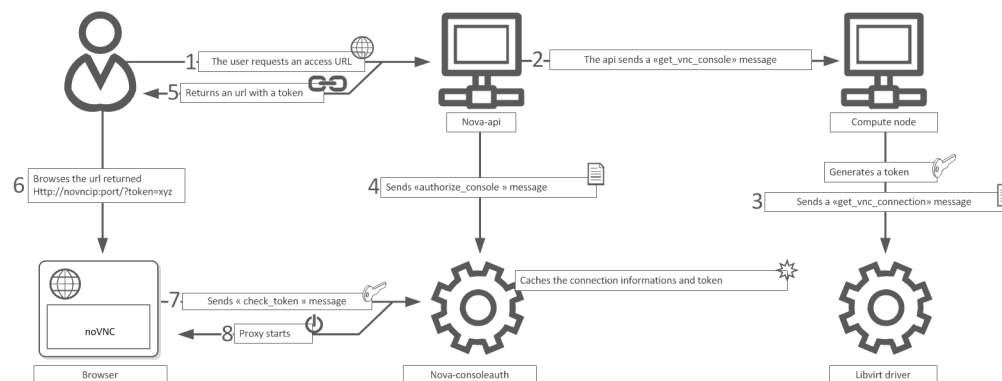
1. User connects to API and gets an `access_url` like `http://ip:port/?token=xyz`.
2. User pastes URL in browser or as client parameter.
3. Browser/Client connects to proxy.
4. Proxy talks to `nova-consoleauth` to authorize the user's token, and then maps the token to the *private* host and port of an instance's VNC server. The compute host specifies the address the proxy should use to connect via the `nova.conf` option `vncserver_proxyclient_address`. In this way, the vnc proxy works as a bridge between the public network, and the private host network.
5. Proxy initiates connection to VNC server, and continues proxying until the session ends.

The proxy also performs the required function of tunneling the VNC protocol over Websockets so that the noVNC client has a way to talk VNC. Note that in general, the VNC proxy performs multiple functions:

- Bridges between public network (where clients live) and private network (where vncservers live).
- Mediates token authentication.

- Transparently deals with hypervisor-specific connection details to provide a uniform client experience.

Figure 11.1. NoVNC Process



About nova-consoleauth

Both client proxies leverage a shared service to manage token auth called nova-consoleauth. This service must be running in order for either proxy to work. Many proxies of either type can be run against a single nova-consoleauth service in a cluster configuration.

The nova-consoleauth shared service should not be confused with nova-console, which is a XenAPI-specific service that is not used by the most recent VNC proxy architecture.

Typical Deployment

A typical deployment will consist of the following components:

- One nova-consoleauth process. Typically this runs on the controller host.
- One or more nova-novncproxy services. This supports browser-based novnc clients. For simple deployments, this service typically will run on the same machine as nova-api, since it proxies between the public network and the private compute host network.
- One or more nova-xvpvncproxy services. This supports the special Java client discussed in this document. For simple deployments, this service typically will run on the same machine as nova-api, since it proxies between the public network and the private compute host network.
- One or more compute hosts. These compute hosts must have correctly configured configuration options, as described below.

Getting an Access URL

Nova provides the ability to create access_urls through the os-consoles extension. Support for accessing this URL is provided by novaclient:

```
$ nova get-vnc-console [server_id] [novnc/xvpvnc]
```

Specify 'novnc' to retrieve a URL suitable for pasting into a web browser. Specify 'xvpvnc' for a URL suitable for pasting into the Java client.

So to request a web browser URL:

```
$ nova get-vnc-console [server_id] novnc
```

VNC Configuration Options

Table 11.1. Description of nova.conf file configuration options for VNC access to guest instances

Configuration option=Default value	(Type) Description
novncproxy_base_url=http://127.0.0.1:6080/vnc_auto.html	(StrOpt) location of VNC console proxy, in the form "http://127.0.0.1:6080/vnc_auto.html"
vnc_enabled=true	(BoolOpt) enable VNC related features
vnc_keymap=en-us	(StrOpt) keymap for vnc
vnc_require_instance_uuid_as_password=false	(BoolOpt) When set to true, secure VNC connections by requiring the user to enter the instance uuid as the password. This ensures the user is connecting to the correct VNC console.
vncserver_listen=127.0.0.1	(StrOpt) IP address on which instance VNC servers should listen
vncserver_proxyclient_address=127.0.0.1	(StrOpt) the address to which proxy clients (like nova-xvpvncproxy) should connect
xvpvncproxy_base_url=http://127.0.0.1:6081/console	(StrOpt) location of nova XCP VNC console proxy, in the form "http://127.0.0.1:6081/console"
xvpvncproxy_host=0.0.0.0	(StrOpt) Address that the XCP VNC proxy should bind to
xvpvncproxy_port=6081	(IntOpt) Port that the XCP VNC proxy should bind to



Note

If you intend to support [live migration](#), you cannot specify a specific IP address for `vncserver_listen`, because that IP address will not exist on the destination host.



Note

`vncserver_proxyclient_address` - Defaults to `127.0.0.1`. This is the address of the compute host that nova will instruct proxies to use when connecting to instance servers. For all-in-one XenServer domU deployments this can be set to `169.254.0.1`. For multi-host XenServer domU deployments this can be set to a dom0 management ip on the same network as the proxies. For multi-host libvirt deployments this can be set to a host management IP on the same network as the proxies.

Accessing VNC Consoles with a Java client

To enable support for the OpenStack Java VNC client in Compute, we provide the `nova-xvpvncproxy` service, which you should run to enable this feature.

- `xvpvncproxy_port=[port]` - port to bind (defaults to 6081)

-
- `xvpngproxy_host=[host]` - host to bind (defaults to 0.0.0.0)

As a client, you will need a special Java client, which is a version of TightVNC slightly modified to support our token auth:

```
$ git clone https://github.com/cloudbuilders/nova-xvpngviewer
$ cd nova-xvpngviewer
$ make
```

Then, to create a session, first request an access URL using **python-novaclient** and then run the client like so. To retrieve access URL:

```
$ nova get-vnc-console [server_id] xvpng
```

To run client:

```
$ java -jar VncViewer.jar [access_url]
```

nova-novncproxy (novnc)

You will need the novnc package installed, which contains the nova-novncproxy service. As root:

```
# apt-get install novnc
```

The service should start automatically on install. To restart it:

```
# service novnc restart
```

The configuration option parameter should point to your `nova.conf` configuration file that includes the message queue server address and credentials.

By default, **nova-novncproxy** binds on `0.0.0.0:6080`.

In order to connect the service to your nova deployment, add the two following configuration options into your `nova.conf` file :

- `vncserver_listen=0.0.0.0`

This configuration option allow you to specify the address for the vnc service to bind on, make sure it is assigned one of the compute node interfaces. This address will be the one used by your domain file.

```
<graphics type="vnc" autoport="yes" keymap="en-us"
listen="0.0.0.0"/>
```



Note

In order to have the live migration working, make sure to use the `0.0.0.0` address.

- `vncserver_proxyclient_address =127.0.0.1`

This is the address of the compute host that nova will instruct proxies to use when connecting to instance vncservers.

Accessing a VNC console through a web browser

Retrieving an `access_url` for a web browser is similar to the flow for the Java client. To retrieve the access URL:

```
$ nova get-vnc-console [server_id] novnc
```

Then, paste the URL into your web browser.

Additionally, you can use the OpenStack Dashboard (codenamed Horizon), to access browser-based VNC consoles for instances.

Frequently asked questions about VNC access to VMs

- **Q: What is the difference between `nova-xvpvncproxy` and `nova-novncproxy`?**

A: `nova-xvpvncproxy` which ships with `nova`, is a new proxy that supports a simple Java client. `nova-novncproxy` uses `noVNC` to provide vnc support through a web browser.

- **Q: I want VNC support in the Dashboard. What services do I need?**

A: You need `nova-novncproxy`, `nova-consoleauth`, and correctly configured compute hosts.

- **Q: When I use `nova get-vnc-console` or click on the VNC tab of the Dashboard, it hangs. Why?**

A: Make sure you are running `nova-consoleauth` (in addition to `nova-novncproxy`). The proxies rely on `nova-consoleauth` to validate tokens, and will wait for a reply from them until a timeout is reached.

- **Q: My vnc proxy worked fine during my All-In-One test, but now it doesn't work on multi host. Why?**

A: The default options work for an All-In-One install, but changes must be made on your compute hosts once you start to build a cluster. As an example, suppose you have two servers:

```
PROXYSERVER (public_ip=172.24.1.1, management_ip=192.168.1.1)
COMPUTESERVER (management_ip=192.168.1.2)
```

Your `nova-compute` configuration file would need the following values:

```
# These flags help construct a connection data structure
vncserver_proxyclient_address=192.168.1.2
novncproxy_base_url=http://172.24.1.1:6080/vnc_auto.html
xvpvncproxy_base_url=http://172.24.1.1:6081/console

# This is the address where the underlying vncserver (not the proxy)
# will listen for connections.
vncserver_listen=192.168.1.2
```

Note that `novncproxy_base_url` and `xvpvncproxy_base_url` use a public ip; this is the url that is ultimately returned to clients, who generally will not have access to your private network. Your PROXYSERVER must be able to reach `vncserver_proxycient_address`, as that is the address over which the vnc connection will be proxied.

See "Important nova-compute Options" for more information.

- **Q: My noVNC does not work with recent versions of web browsers. Why?**

A: Make sure you have `python-numpy` installed, which is required to support a newer version of the WebSocket protocol (HyBi-07+).

- **Q: How do I adjust the dimensions of the VNC window image in horizon?**

A: These values are hard-coded in a Django HTML template. To alter them, you must edit the template file `_detail_vnc.html`. The location of this file will vary based on Linux distribution. On Ubuntu 12.04, the file can be found at `/usr/share/pyshared/horizon/dashboards/nova/instances/templates/instances/_detail_vnc.html`.

Modify the `width` and `height` parameters:

```
<iframe src="{ { vnc_url } }" width="720" height="430"></iframe>
```

Appendix A. Configuration File Examples

Table of Contents

keystone.conf	105
glance-registry.conf	107
glance-registry-paste.ini	109
glance-api.conf	109
glance-api-paste.ini	115
glance-scrubber.conf	116
nova.conf	117
api-paste.ini	118
Credentials (openrc)	120
cinder.conf	121
Dashboard configuration	121
etc/swift/swift.conf	123
etc/network/interfaces.conf	123
etc/swift/proxy-server.conf	124
etc/swift/account-server.conf	125
etc/swift/account-server/1.conf	125
etc/swift/container-server.conf	125
etc/swift/container-server/1.conf	126
etc/swift/object-server.conf	126
etc/swift/object-server/1.conf	126

keystone.conf

The configuration file for the Identity Service is `/etc/keystone/keystone.conf`.

After you install the Identity Service, modify this file to use SQL for endpoint data and to replace the ADMIN key with the one created that was created during the installation.

```
[DEFAULT]
bind_host = 0.0.0.0
public_port = 5000
admin_port = 35357
admin_token = 012345SECRET99TOKEN012345
compute_port = 8774
verbose = True
debug = True
log_config = /etc/keystone/logging.conf

# ===== Syslog Options =====
# Send logs to syslog (/dev/log) instead of to file specified
# by `log-file`
use_syslog = False

# Facility to use. If unset defaults to LOG_USER.
# syslog_log_facility = LOG_LOCAL0

[sql]
connection = mysql://keystone:yourpassword@192.168.127.130/keystone
```



```
idle_timeout = 200
min_pool_size = 5
max_pool_size = 10
pool_timeout = 200

[ldap]
#url = ldap://localhost
#tree_dn = dc=example,dc=com
#user_tree_dn = ou=Users,dc=example,dc=com
#role_tree_dn = ou=Roles,dc=example,dc=com
#tenant_tree_dn = ou=Groups,dc=example,dc=com
#user = dc=Manager,dc=example,dc=com
#password = freeipa4all
#suffix = cn=example,cn=com

[identity]
driver = keystone.identity.backends.sql.Identity

[catalog]
driver = keystone.catalog.backends.sql.Catalog

[token]
driver = keystone.token.backends.sql.Token

# Amount of time a token should remain valid (in seconds)
expiration = 86400

[policy]
driver = keystone.policy.backends.rules.Policy

[ec2]
driver = keystone.contrib.ec2.backends.sql.Ec2

[filter:debug]
paste.filter_factory = keystone.common.wsgi:Debug.factory

[filter:token_auth]
paste.filter_factory = keystone.middleware:TokenAuthMiddleware.factory

[filter:admin_token_auth]
paste.filter_factory = keystone.middleware:AdminTokenAuthMiddleware.factory

[filter:xml_body]
paste.filter_factory = keystone.middleware:XmlBodyMiddleware.factory

[filter:json_body]
paste.filter_factory = keystone.middleware:JsonBodyMiddleware.factory

[filter:crud_extension]
paste.filter_factory = keystone.contrib.admin_crud:CrudExtension.factory

[filter:ec2_extension]
paste.filter_factory = keystone.contrib.ec2:Ec2Extension.factory

[app:public_service]
paste.app_factory = keystone.service:public_app_factory

[app:admin_service]
paste.app_factory = keystone.service:admin_app_factory
```

```
[pipeline:public_api]
pipeline = token_auth admin_token_auth xml_body json_body debug ec2_extension
public_service

[pipeline:admin_api]
pipeline = token_auth admin_token_auth xml_body json_body debug ec2_extension
crud_extension admin_service

[app:public_version_service]
paste.app_factory = keystone.service:public_version_app_factory

[app:admin_version_service]
paste.app_factory = keystone.service:admin_version_app_factory

[pipeline:public_version_api]
pipeline = xml_body public_version_service

[pipeline:admin_version_api]
pipeline = xml_body admin_version_service

[composite:main]
use = egg:Paste#urlmap
/v2.0 = public_api
/ = public_version_api

[composite:admin]
use = egg:Paste#urlmap
/v2.0 = admin_api
/ = admin_version_api
```

glance-registry.conf

The Image Service registry configuration file is `/etc/glance/glance-registry.conf`.

This file stores metadata for images.

After you install the Image Service, modify this configuration file.

```
[DEFAULT]
# Show more verbose log output (sets INFO log level output)
verbose = True

# Show debugging output in logs (sets DEBUG log level output)
debug = False

# Address to bind the registry server
bind_host = 0.0.0.0

# Port the bind the registry server to
bind_port = 9191

# Log to this file. Make sure you do not set the same log
# file for both the API and registry servers!
log_file = /var/log/glance/registry.log

# Backlog requests when creating socket
backlog = 4096

# TCP_KEEPIDLE value in seconds when creating socket.
```

```
# Not supported on OS X.
#tcp_keepidle = 600

# 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:YOUR_GLANCEDB_PASSWORD@192.168.206.130/glance

# Period in seconds after which SQLAlchemy should reestablish its connection
# to the database.
#
# MySQL uses a default `wait_timeout` of 8 hours, after which it will drop
# idle connections. This can result in 'MySQL Gone Away' exceptions. If you
# notice this, you can lower this value to ensure that SQLAlchemy reconnects
# before MySQL can drop the connection.
sql_idle_timeout = 3600

# Limit the api to return `param_limit_max` items in a call to a container. If
# a larger `limit` query param is provided, it will be reduced to this value.
api_limit_max = 1000

# If a `limit` query param is not provided in an api request, it will
# default to `limit_param_default`
limit_param_default = 25

# Role used to identify an authenticated user as administrator
#admin_role = admin

# ===== Syslog Options =====

# Send logs to syslog (/dev/log) instead of to file specified
# by `log_file`
use_syslog = False

# Facility to use. If unset defaults to LOG_USER.
#syslog_log_facility = LOG_LOCAL1

# ===== SSL Options =====

# Certificate file to use when starting registry server securely
#cert_file = /path/to/certfile

# Private key file to use when starting registry server securely
#key_file = /path/to/keyfile

# CA certificate file to use to verify connecting clients
#ca_file = /path/to/cafile

[keystone_authtoken]
auth_host = 127.0.0.1
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = admin
admin_password = secrete

[paste_deploy]
# Name of the paste configuration file that defines the available pipelines
config_file = /etc/glance/glance-registry-paste.ini
```

```
# Partial name of a pipeline in your paste configuration file with the
# service name removed. For example, if your paste section name is
# [pipeline:glance-api-keystone], you would configure the flavor below
# as 'keystone'.
flavor=keystone
```

glance-registry-paste.ini

The Image Service API middleware pipeline configuration file is `/etc/glance/glance-registry-paste.ini`.

After you install the Image Service, modify this configuration file.

```
# Use this pipeline for no auth - DEFAULT
# [pipeline:glance-registry]
# pipeline = unauthenticated-context registryapp

# Use this pipeline for keystone auth
[pipeline:glance-registry-keystone]
pipeline = authtoken context registryapp

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

[app:registryapp]
paste.app_factory = glance.registry.api.v1:API.factory

[filter:context]
paste.filter_factory = glance.api.middleware.context:ContextMiddleware.factory

[filter:unauthenticated-context]
paste.filter_factory =
    glance.api.middleware.context:UnauthenticatedContextMiddleware.factory

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

glance-api.conf

The configuration file for the Image API is `/etc/glance/glance-api.conf`.

After you install the Image Service API, update this file as shown in the following example.

```
[DEFAULT]
# Show more verbose log output (sets INFO log level output)
verbose = True

# Show debugging output in logs (sets DEBUG log level output)
debug = False

# Which backend scheme should Glance use by default is not specified
# in a request to add a new image to Glance? Known schemes are determined
# by the known_stores option below.
# Default: 'file'
```

```
default_store = file

# List of which store classes and store class locations are
# currently known to glance at startup.
#known_stores = glance.store.filesystem.Store,
#
#                 glance.store.http.Store,
#                 glance.store.rbd.Store,
#                 glance.store.s3.Store,
#                 glance.store.swift.Store,

# Maximum image size (in bytes) that may be uploaded through the
# Glance API server. Defaults to 1 TB.
# WARNING: this value should only be increased after careful consideration
# and must be set to a value under 8 EB (9223372036854775808).
#image_size_cap = 1099511627776

# Address to bind the API server
bind_host = 0.0.0.0

# Port the bind the API server to
bind_port = 9292

# Log to this file. Make sure you do not set the same log
# file for both the API and registry servers!
log_file = /var/log/glance/api.log

# Backlog requests when creating socket
backlog = 4096

# TCP_KEEPIDLE value in seconds when creating socket.
# Not supported on OS X.
#tcp_keeppidle = 600

# 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 = sqlite:///glance.sqlite
# sql_connection = sql_connection = mysql://
# glance:YOUR_GLANCEDB_PASSWORD@192.168.206.130/glance

# Period in seconds after which SQLAlchemy should reestablish its connection
# to the database.
#
# MySQL uses a default `wait_timeout` of 8 hours, after which it will drop
# idle connections. This can result in 'MySQL Gone Away' exceptions. If you
# notice this, you can lower this value to ensure that SQLAlchemy reconnects
# before MySQL can drop the connection.
sql_idle_timeout = 3600

# Number of Glance API worker processes to start.
# On machines with more than one CPU increasing this value
# may improve performance (especially if using SSL with
# compression turned on). It is typically recommended to set
# this value to the number of CPUs present on your machine.
workers = 1

# Role used to identify an authenticated user as administrator
#admin_role = admin
```

```
# Allow unauthenticated users to access the API with read-only
# privileges. This only applies when using ContextMiddleware.
#allow_anonymous_access = False

# Allow access to version 1 of glance api
#enable_v1_api = True

# Allow access to version 2 of glance api
#enable_v2_api = True

# ===== Syslog Options =====

# Send logs to syslog (/dev/log) instead of to file specified
# by `log_file`
use_syslog = False

# Facility to use. If unset defaults to LOG_USER.
#syslog_log_facility = LOG_LOCAL0

# ===== SSL Options =====

# Certificate file to use when starting API server securely
#cert_file = /path/to/certfile

# Private key file to use when starting API server securely
#key_file = /path/to/keyfile

# CA certificate file to use to verify connecting clients
#ca_file = /path/to/cafile

# ===== Security Options =====

# AES key for encrypting store 'location' metadata, including
# -- if used -- Swift or S3 credentials
# Should be set to a random string of length 16, 24 or 32 bytes
#metadata_encryption_key = <16, 24 or 32 char registry metadata key>

# ===== Registry Options =====

# Address to find the registry server
registry_host = 0.0.0.0

# Port the registry server is listening on
registry_port = 9191

# What protocol to use when connecting to the registry server?
# Set to https for secure HTTP communication
registry_client_protocol = http

# The path to the key file to use in SSL connections to the
# registry server, if any. Alternately, you may set the
# GLANCE_CLIENT_KEY_FILE environ variable to a filepath of the key file
#registry_client_key_file = /path/to/key/file

# The path to the cert file to use in SSL connections to the
# registry server, if any. Alternately, you may set the
# GLANCE_CLIENT_CERT_FILE environ variable to a filepath of the cert file
#registry_client_cert_file = /path/to/cert/file
```

```
# The path to the certifying authority cert file to use in SSL connections
# to the registry server, if any. Alternately, you may set the
# GLANCE_CLIENT_CA_FILE environ variable to a filepath of the CA cert file
#registry_client_ca_file = /path/to/ca/file

# ===== Notification System Options =====

# Notifications can be sent when images are create, updated or deleted.
# There are three methods of sending notifications, logging (via the
# log_file directive), rabbit (via a rabbitmq queue), qpid (via a Qpid
# message queue), or noop (no notifications sent, the default)
notifier_strategy = noop

# Configuration options if sending notifications via rabbitmq (these are
# the defaults)
rabbit_host = localhost
rabbit_port = 5672
rabbit_use_ssl = false
rabbit_userid = guest
rabbit_password = guest
rabbit_virtual_host = /
rabbit_notification_exchange = glance
rabbit_notification_topic = glance_notifications
rabbit_durable_queues = False

# Configuration options if sending notifications via Qpid (these are
# the defaults)
qpid_notification_exchange = glance
qpid_notification_topic = glance_notifications
qpid_host = localhost
qpid_port = 5672
qpid_username =
qpid_password =
qpid_sasl_mechanisms =
qpid_reconnect_timeout = 0
qpid_reconnect_limit = 0
qpid_reconnect_interval_min = 0
qpid_reconnect_interval_max = 0
qpid_reconnect_interval = 0
qpid_heartbeat = 5
# Set to 'ssl' to enable SSL
qpid_protocol = tcp
qpid_tcp_nodelay = True

# ===== Filesystem Store Options =====

# Directory that the Filesystem backend store
# writes image data to
filesystem_store_datadir = /var/lib/glance/images/

# ===== Swift Store Options =====

# Version of the authentication service to use
# Valid versions are '2' for keystone and '1' for swauth and rackspace
swift_store_auth_version = 2

# Address where the Swift authentication service lives
# Valid schemes are 'http://' and 'https://'
# If no scheme specified, default to 'https://'
# For swauth, use something like '127.0.0.1:8080/v1.0/'
```

```

swift_store_auth_address = 127.0.0.1:5000/v2.0/

# User to authenticate against the Swift authentication service
# If you use Swift authentication service, set it to 'account':'user'
# where 'account' is a Swift storage account and 'user'
# is a user in that account
swift_store_user = jdoe:jdoe

# Auth key for the user authenticating against the
# Swift authentication service
swift_store_key = a86850deb2742ec3cb41518e26aa2d89

# Container within the account that the account should use
# for storing images in Swift
swift_store_container = glance

# Do we create the container if it does not exist?
swift_store_create_container_on_put = False

# What size, in MB, should Glance start chunking image files
# and do a large object manifest in Swift? By default, this is
# the maximum object size in Swift, which is 5GB
swift_store_large_object_size = 5120

# When doing a large object manifest, what size, in MB, should
# Glance write chunks to Swift? This amount of data is written
# to a temporary disk buffer during the process of chunking
# the image file, and the default is 200MB
swift_store_large_object_chunk_size = 200

# Whether to use ServiceNET to communicate with the Swift storage servers.
# (If you aren't RACKSPACE, leave this False!)
#
# To use ServiceNET for authentication, prefix hostname of
# `swift_store_auth_address` with 'snet-'.
# Ex. https://example.com/v1.0/ -> https://snet-example.com/v1.0/
swift_enable_snet = False

# If set to True enables multi-tenant storage mode which causes Glance images
# to be stored in tenant specific Swift accounts.
#swift_store_multi_tenant = False

# A list of tenants that will be granted read/write access on all Swift
# containers created by Glance in multi-tenant mode.
#swift_store_admin_tenants = []

# The region of the swift endpoint to be used for single tenant. This setting
# is only necessary if the tenant has multiple swift endpoints.
#swift_store_region =

# ===== S3 Store Options =====

# Address where the S3 authentication service lives
# Valid schemes are 'http://' and 'https://'
# If no scheme specified, default to 'http://'
s3_store_host = 127.0.0.1:8080/v1.0/

# User to authenticate against the S3 authentication service
s3_store_access_key = <20-char AWS access key>

```



```
# Auth key for the user authenticating against the
# S3 authentication service
s3_store_secret_key = <40-char AWS secret key>

# Container within the account that the account should use
# for storing images in S3. Note that S3 has a flat namespace,
# so you need a unique bucket name for your glance images. An
# easy way to do this is append your AWS access key to "glance".
# S3 buckets in AWS *must* be lowercased, so remember to lowercase
# your AWS access key if you use it in your bucket name below!
s3_store_bucket = <lowercased 20-char aws access key>glance

# Do we create the bucket if it does not exist?
s3_store_create_bucket_on_put = False

# When sending images to S3, the data will first be written to a
# temporary buffer on disk. By default the platform's temporary directory
# will be used. If required, an alternative directory can be specified here.
#s3_store_object_buffer_dir = /path/to/dir

# When forming a bucket url, boto will either set the bucket name as the
# subdomain or as the first token of the path. Amazon's S3 service will
# accept it as the subdomain, but Swift's S3 middleware requires it be
# in the path. Set this to 'path' or 'subdomain' - defaults to 'subdomain'.
#s3_store_bucket_url_format = subdomain

# ===== RBD Store Options =====

# Ceph configuration file path
# If using cephx authentication, this file should
# include a reference to the right keyring
# in a client.<USER> section
rbd_store_ceph_conf = /etc/ceph/ceph.conf

# RADOS user to authenticate as (only applicable if using cephx)
rbd_store_user = glance

# RADOS pool in which images are stored
rbd_store_pool = images

# Images will be chunked into objects of this size (in megabytes).
# For best performance, this should be a power of two
rbd_store_chunk_size = 8

# ===== Delayed Delete Options =====

# Turn on/off delayed delete
delayed_delete = False

# Delayed delete time in seconds
scrub_time = 43200

# Directory that the scrubber will use to remind itself of what to delete
# Make sure this is also set in glance-scrubber.conf
scrubber_datadir = /var/lib/glance/scrubber

# ===== Image Cache Options =====

# Base directory that the Image Cache uses
image_cache_dir = /var/lib/glance/image-cache/
```

```
[keystone_authtoken]
auth_host = 127.0.0.1
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = admin
admin_password = secrete

[paste_deploy]
# Name of the paste configuration file that defines the available pipelines
config_file = /etc/glance/glance-api-paste.ini

# Partial name of a pipeline in your paste configuration file with the
# service name removed. For example, if your paste section name is
# [pipeline:glance-api-keystone], you would configure the flavor below
# as 'keystone'.
flavor=keystone
```

glance-api-paste.ini

The Image ServiceAPI middleware pipeline configuration file is `/etc/glance/glance-api-paste.ini`.

You should not need to modify this file.

```
# Use this pipeline for no auth or image caching - DEFAULT
# [pipeline:glance-api]
# pipeline = versionnegotiation unauthenticated-context rootapp

# Use this pipeline for image caching and no auth
# [pipeline:glance-api-caching]
# pipeline = versionnegotiation unauthenticated-context cache rootapp

# Use this pipeline for caching w/ management interface but no auth
# [pipeline:glance-api-cachemanagement]
# pipeline = versionnegotiation unauthenticated-context cache cachemanage
rootapp

# Use this pipeline for keystone auth
[pipeline:glance-api-keystone]
pipeline = versionnegotiation authtoken context rootapp

# Use this pipeline for keystone auth with image caching
# [pipeline:glance-api-keystone+caching]
# pipeline = versionnegotiation authtoken context cache rootapp

# Use this pipeline for keystone auth with caching and cache management
# [pipeline:glance-api-keystone+cachemanagement]
# pipeline = versionnegotiation authtoken context cache cachemanage rootapp

[composite:rootapp]
paste.composite_factory = glance.api:root_app_factory
/: apiversions
/v1: apiv1app
/v2: apiv2app

[app:apiversions]
paste.app_factory = glance.api.versions:create_resource
```

```
[app:apiv1app]
paste.app_factory = glance.api.v1.router:API.factory

[app:apiv2app]
paste.app_factory = glance.api.v2.router:API.factory

[filter:versionnegotiation]
paste.filter_factory =
    glance.api.middleware.version_negotiation:VersionNegotiationFilter.factory

[filter:cache]
paste.filter_factory = glance.api.middleware.cache:CacheFilter.factory

[filter:cachemanager]
paste.filter_factory =
    glance.api.middleware.cache_manage:CacheManageFilter.factory

[filter:context]
paste.filter_factory = glance.api.middleware.context:ContextMiddleware.factory

[filter:unauthenticated-context]
paste.filter_factory =
    glance.api.middleware.context:UnauthenticatedContextMiddleware.factory

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

glance-scrubber.conf

An additional configuration file for the Image Service is `/etc/glance/glance-scrubber.conf`.

The scrubber is a utility that cleans up images that have been deleted.

```
[DEFAULT]
# Show more verbose log output (sets INFO log level output)
verbose = True

# Show debugging output in logs (sets DEBUG log level output)
debug = False

# Log to this file. Make sure you do not set the same log
# file for both the API and registry servers!
log_file = /var/log/glance/scrubber.log

# Send logs to syslog (/dev/log) instead of to file specified by `log_file`
use_syslog = False

# Delayed delete time in seconds
scrub_time = 43200

# Should we run our own loop or rely on cron/scheduler to run us
daemon = False

# Loop time between checking the registry for new items to schedule for delete
wakeup_time = 300
```

```
[app:glance-scrubber]  
paste.app_factory = glance.store.scrubber:app_factory
```

nova.conf

The Compute configuration file is `/etc/nova/nova.conf`.

For a list of configuration options for this file, see the List of Tables on this page: [OpenStack Compute Administration Manual](#).

This guide assumes that the IP address of the machine that runs the Identity Service is 192.168.206.130. If the IP address of the machine on your setup is different, change the following configuration options:

- `service_host`
- `auth_host`
- `auth_uri`

```
[DEFAULT]  
  
# LOGS/STATE  
verbose=True  
logdir=/var/log/nova  
state_path=/var/lib/nova  
lock_path=/var/lock/nova  
rootwrap_config=/etc/nova/rootwrap.conf  
  
# SCHEDULER  
compute_scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler  
  
# VOLUMES  
# configured in cinder.conf  
  
# DATABASE  
sql_connection=mysql://nova:yourpassword@192.168.206.130/nova  
  
# COMPUTE  
libvirt_type=qemu  
compute_driver=libvirt.LibvirtDriver  
instance_name_template=instance-%08x  
api_paste_config=/etc/nova/api-paste.ini  
  
# COMPUTE/APIS: if you have separate configs for separate services  
# this flag is required for both nova-api and nova-compute  
allow_resize_to_same_host=True  
  
# APIS  
osapi_compute_extension=nova.api.openstack.compute.contrib.standard_extensions  
ec2_dmz_host=192.168.206.130  
s3_host=192.168.206.130  
  
# QPID  
rpc_backend=nova.rpc.impl_qpid  
qpid_hostname=192.168.206.130  
  
# GLANCE  
image_service=nova.image.glance.GlanceImageService
```

```
glance_api_servers=192.168.206.130:9292

# NETWORK
network_manager=nova.network.manager.FlatDHCPManager
force_dhcp_release=True
dhcpbridge_flagfile=/etc/nova/nova.conf
firewall_driver=nova.virt.libvirt.firewall.IptablesFirewallDriver
# Change my_ip to match each host
my_ip=192.168.206.130
public_interface=eth100
vlan_interface=eth0
flat_network_bridge=br100
flat_interface=eth0
fixed_range=192.168.100.0/24

# NOVNC CONSOLE
novncproxy_base_url=http://192.168.206.130:6080/vnc_auto.html
# Change vncserver_proxyclient_address and vncserver_listen to match each
compute host
vncserver_proxyclient_address=192.168.206.130
vncserver_listen=192.168.206.130

# AUTHENTICATION
auth_strategy=keystone
[keystone_auth_token]
auth_host = 127.0.0.1
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = nova
admin_password = nova
signing_dirname = /tmp/keystone-signing-nova
```

api-paste.ini

The middleware configuration file used by the EC2 API and OpenStack Compute API is /etc/nova/api-paste.ini.

You should not need to edit it.

```
#####
# Metadata #
#####
[composite:metadata]
use = egg:Paste#urlmap
/: meta

[pipeline:meta]
pipeline = ec2faultwrap logrequest metaapp

[app:metaapp]
paste.app_factory = nova.api.metadata.handler:MetadataRequestHandler.factory

#####
# EC2 #
#####

[composite:ec2]
use = egg:Paste#urlmap
```

```
/services/Cloud: ec2cloud

[composite:ec2cloud]
use = call:nova.api.auth:pipeline_factory
noauth = ec2faultwrap logrequest ec2noauth cloudrequest validator ec2executor
keystone = ec2faultwrap logrequest ec2keystoneauth cloudrequest validator
          ec2executor

[filter:ec2faultwrap]
paste.filter_factory = nova.api.ec2:FaultWrapper.factory

[filter:logrequest]
paste.filter_factory = nova.api.ec2:RequestLogging.factory

[filter:ec2lockout]
paste.filter_factory = nova.api.ec2:Lockout.factory

[filter:ec2keystoneauth]
paste.filter_factory = nova.api.ec2:EC2KeystoneAuth.factory

[filter:ec2noauth]
paste.filter_factory = nova.api.ec2:NoAuth.factory

[filter:cloudrequest]
controller = nova.api.ec2.cloud.CloudController
paste.filter_factory = nova.api.ec2:Requestify.factory

[filter:authorizer]
paste.filter_factory = nova.api.ec2:Authorizer.factory

[filter:validator]
paste.filter_factory = nova.api.ec2:Validator.factory

[app:ec2executor]
paste.app_factory = nova.api.ec2:Executor.factory

#####
# Openstack #
#####

[composite:osapi_compute]
use = call:nova.api.openstack.urlmap:urlmap_factory
/: oscomputeversions
/v1.1: openstack_compute_api_v2
/v2: openstack_compute_api_v2

[composite:osapi_volume]
use = call:nova.api.openstack.urlmap:urlmap_factory
/: osvolumeverSIONS
/v1: openstack_volume_api_v1

[composite:openstack_compute_api_v2]
use = call:nova.api.auth:pipeline_factory
noauth = faultwrap sizelimit noauth ratelimit osapi_compute_app_v2
keystone = faultwrap sizelimit authToken keystonecontext ratelimit
          osapi_compute_app_v2
keystone_nolimit = faultwrap sizelimit authToken keystonecontext
          osapi_compute_app_v2

[composite:openstack_volume_api_v1]
```

```
use = call:nova.api.auth:pipeline_factory
noauth = faultwrap sizelimit noauth ratelimit osapi_volume_app_v1
keystone = faultwrap sizelimit authtoken keystonecontext ratelimit
osapi_volume_app_v1
keystone_nolimit = faultwrap sizelimit authtoken keystonecontext
osapi_volume_app_v1

[filter:faultwrap]
paste.filter_factory = nova.api.openstack:FaultWrapper.factory

[filter:noauth]
paste.filter_factory = nova.api.openstack.auth:NoAuthMiddleware.factory

[filter:ratelimit]
paste.filter_factory =
    nova.api.openstack.compute.limits:RateLimitingMiddleware.factory

[filter:sizelimit]
paste.filter_factory = nova.api.sizelimit:RequestBodySizeLimiter.factory

[app:osapi_compute_app_v2]
paste.app_factory = nova.api.openstack.compute:APIRouter.factory

[pipeline:oscomputeversions]
pipeline = faultwrap oscomputeversionapp

[app:osapi_volume_app_v1]
paste.app_factory = nova.api.openstack.volume:APIRouter.factory

[app:oscomputeversionapp]
paste.app_factory = nova.api.openstack.compute.versions:Versions.factory

[pipeline:osvolumeversions]
pipeline = faultwrap osvolumeverversionapp
#####
# Shared #
#####

[filter:keystonecontext]
paste.filter_factory = nova.api.auth:NovaKeystoneContext.factory

[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
# Workaround for https://bugs.launchpad.net/nova/+bug/1154809
auth_version = v2.0
```

Credentials (openrc)

This file contains the credentials used by Compute, Image, and Identity services. You can optionally store this file in `/home/openrc`.

Do not source this file in the environment from where you issue commands.

Run `"env | grep OS_"` or `"env | grep NOVA_"` to view what is being used in your environment.

```
export OS_USERNAME=admin
export OS_TENANT_NAME=demo
export OS_PASSWORD=secrete
```

```
export OS_AUTH_URL=http://192.168.206.130:5000/v2.0/  
export OS_REGION_NAME=RegionOne
```

cinder.conf

```
[DEFAULT]  
rootwrap_config=/etc/cinder/rootwrap.conf  
sql_connection = mysql://cinder:openstack@192.168.127.130/cinder  
api_paste_config = /etc/cinder/api-paste.ini  
  
iscsi_helper=tgtadm  
volume_name_template = volume-%s  
volume_group = cinder-volumes  
verbose = True  
auth_strategy = keystone  
#osapi_volume_listen_port=5900  
  
# Add these when not using the defaults.  
rabbit_host = 10.10.10.10  
rabbit_port = 5672  
rabbit_userid = rabbit  
rabbit_password = secure_password  
rabbit_virtual_host = /nova
```

Dashboard configuration

This file contains the database and configuration settings for the OpenStack Dashboard.

```
import os  
  
from django.utils.translation import ugettext_lazy as _  
  
DEBUG = False  
TEMPLATE_DEBUG = DEBUG  
PROD = True  
USE_SSL = False  
  
SITE_BRANDING = 'OpenStack Dashboard'  
  
# Ubuntu-specific: Enables an extra panel in the 'Settings' section  
# that easily generates a Juju environments.yaml for download,  
# preconfigured with endpoints and credentials required for bootstrap  
# and service deployment.  
ENABLE_JUJU_PANEL = True  
  
# Note: You should change this value  
SECRET_KEY = 'eljlIWlLoWHgryYxFT6j7cM5fGOOxWY0'  
  
# Specify a regular expression to validate user passwords.  
# HORIZON_CONFIG = {  
#     "password_validator": {  
#         "regex": '.*',  
#         "help_text": _("Your password does not meet the requirements.")  
#     }  
# }
```



```

LOCAL_PATH = os.path.dirname(os.path.abspath(__file__))

CACHE_BACKEND = 'memcached://127.0.0.1:11211/'

# Send email to the console by default
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
# Or send them to /dev/null
#EMAIL_BACKEND = 'django.core.mail.backends.dummy.EmailBackend'

# Configure these for your outgoing email host
# EMAIL_HOST = 'smtp.my-company.com'
# EMAIL_PORT = 25
# EMAIL_HOST_USER = 'djangomail'
# EMAIL_HOST_PASSWORD = 'top-secret!'

# For multiple regions uncomment this configuration, and add (endpoint,
# title).
# AVAILABLE_REGIONS = [
#     ('http://cluster1.example.com:5000/v2.0', 'cluster1'),
#     ('http://cluster2.example.com:5000/v2.0', 'cluster2'),
# ]

OPENSTACK_HOST = "127.0.0.1"
OPENSTACK_KEYSTONE_URL = "http://%s:5000/v2.0" % OPENSTACK_HOST
OPENSTACK_KEYSTONE_DEFAULT_ROLE = "Member"

# The OPENSTACK_KEYSTONE_BACKEND settings can be used to identify the
# capabilities of the auth backend for Keystone.
# If Keystone has been configured to use LDAP as the auth backend then set
# can_edit_user to False and name to 'ldap'.
#
# TODO(tres): Remove these once Keystone has an API to identify auth backend.
OPENSTACK_KEYSTONE_BACKEND = {
    'name': 'native',
    'can_edit_user': True
}

# OPENSTACK_ENDPOINT_TYPE specifies the endpoint type to use for the endpoints
# in the Keystone service catalog. Use this setting when Horizon is running
# external to the OpenStack environment. The default is 'internalURL'.
#OPENSTACK_ENDPOINT_TYPE = "publicURL"

# The number of Swift containers and objects to display on a single page
# before
# providing a paging element (a "more" link) to paginate results.
API_RESULT_LIMIT = 1000

# If you have external monitoring links, eg:
# EXTERNAL_MONITORING = [
#     ['Nagios', 'http://foo.com'],
#     ['Ganglia', 'http://bar.com'],
# ]

LOGGING = {
    'version': 1,
    # When set to True this will disable all logging except
    # for loggers specified in this configuration dictionary. Note that
    # if nothing is specified here and disable_existing_loggers is True,
    # django.db.backends will still log unless it is disabled explicitly.

```

```
'disable_existing_loggers': False,
'handlers': {
    'null': {
        'level': 'DEBUG',
        'class': 'django.utils.log.NullHandler',
    },
    'console': {
        # Set the level to "DEBUG" for verbose output logging.
        'level': 'INFO',
        'class': 'logging.StreamHandler',
    },
},
'loggers': {
    # Logging from django.db.backends is VERY verbose, send to null
    # by default.
    'django.db.backends': {
        'handlers': ['null'],
        'propagate': False,
    },
    'horizon': {
        'handlers': ['console'],
        'propagate': False,
    },
    'novaclient': {
        'handlers': ['console'],
        'propagate': False,
    },
    'keystoneclient': {
        'handlers': ['console'],
        'propagate': False,
    },
    'nose.plugins.manager': {
        'handlers': ['console'],
        'propagate': False,
    }
}
```

etc/swift/swift.conf

This file contains the settings to randomize the hash for the ring for Object Storage, code-named swift.

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

etc/network/interfaces.conf

These instructions are for using the FlatDHCP networking mode with a single network interface.

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
```

```
iface eth0 inet dhcp

# Bridge network interface for VM networks
auto br100
iface br100 inet static
address 192.168.100.1
netmask 255.255.255.0
bridge_stp off
bridge_fd 0
```

etc/swift/proxy-server.conf

This file contains the settings for the Object Storage proxy server, which contains the Identity service settings.

```
[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
```

etc/swift/account-server.conf

```
[DEFAULT]
bind_ip = 0.0.0.0
workers = 2

[pipeline:main]
pipeline = account-server

[app:account-server]
use = egg:swift#account

[account-replicator]

[account-auditor]

[account-reaper]
```

etc/swift/account-server/1.conf

```
[DEFAULT]
devices = /srv/1/node
mount_check = false
bind_port = 6012
user = swift
log_facility = LOG_LOCAL2

[pipeline:main]
pipeline = account-server

[app:account-server]
use = egg:swift#account

[account-replicator]
vm_test_mode = yes

[account-auditor]

[account-reaper]
```

etc/swift/container-server.conf

```
[DEFAULT]
bind_ip = 0.0.0.0
workers = 2

[pipeline:main]
pipeline = container-server

[app:container-server]
use = egg:swift#container
```

```
[container-replicator]

[container-updater]

[container-auditor]

[container-sync]
```

etc/swift/container-server/1.conf

```
[DEFAULT]
devices = /srv/1/node
mount_check = false
bind_port = 6011
user = swift
log_facility = LOG_LOCAL2

[pipeline:main]
pipeline = container-server

[app:container-server]
use = egg:swift#container

[container-replicator]
vm_test_mode = yes

[container-updater]

[container-auditor]
[container-sync]
```

etc/swift/object-server.conf

```
[DEFAULT]
bind_ip = 0.0.0.0
workers = 2

[pipeline:main]
pipeline = object-server

[app:object-server]
use = egg:swift#object

[object-replicator]

[object-updater]

[object-auditor]

[object-expirer]
```

etc/swift/object-server/1.conf

```
[DEFAULT]
devices = /srv/1/node
mount_check = false
```

```
bind_port = 6010
user = swift
log_facility = LOG_LOCAL2

[pipeline:main]
pipeline = object-server

[app:object-server]
use = egg:swift#object

[object-replicator]
vm_test_mode = yes

[object-updater]

[object-auditor]

[object-expirer]
```