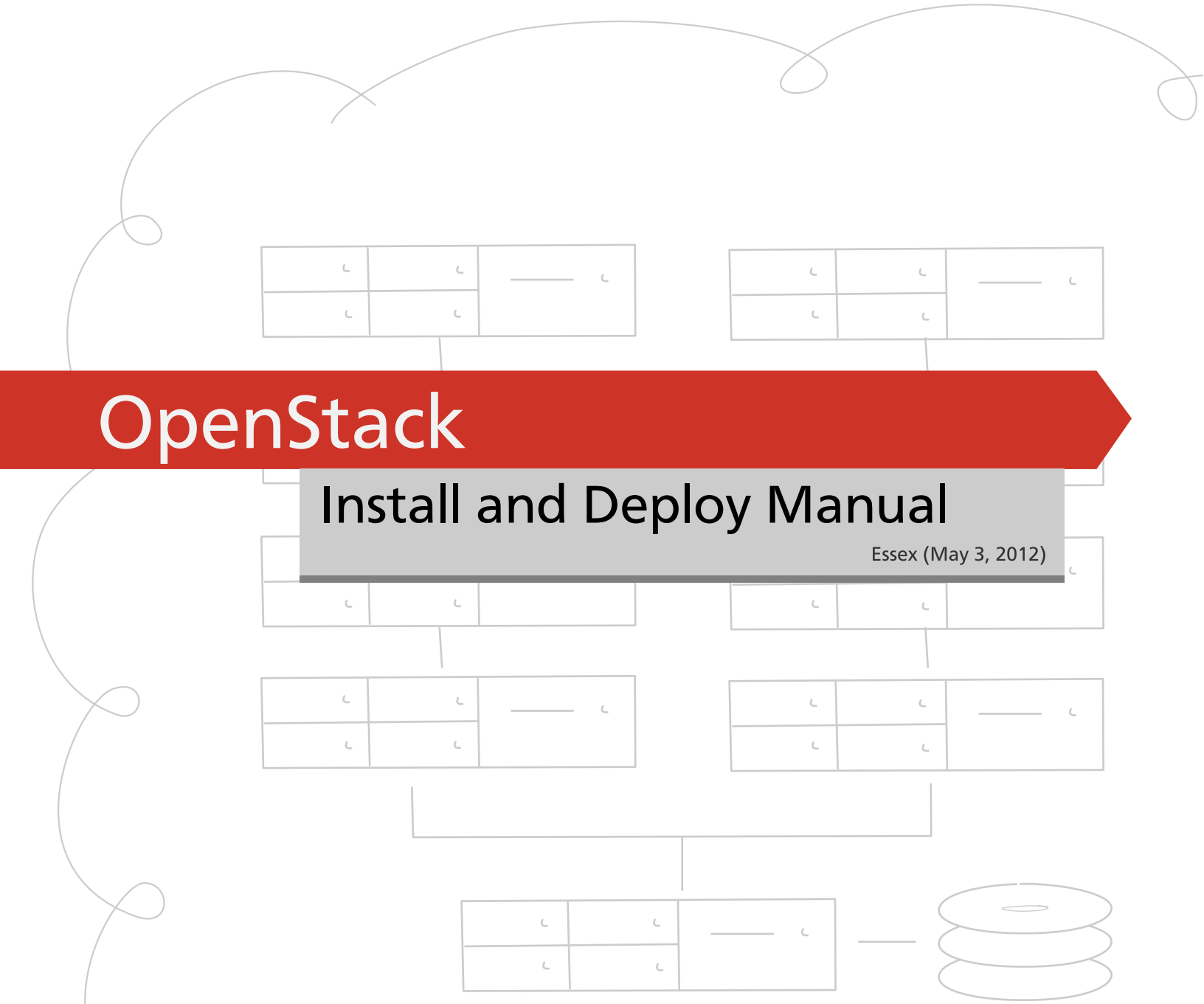
OpenStack

Install and Deploy Manual

Essex (May 3, 2012)



openstack™

docs.openstack.org

OpenStack Install and Deploy Manual

Essex (2012-05-03)

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 Ubuntu 12.04. It offers explanations for the configuration choices as well as sample configuration files.

Table of Contents

List of Tables

1. Installing OpenStack Walk-through

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 Compute and related services and we will add Object Storage at a later date.

Here are the overall steps:

1. Review the most supported platforms.

   This installation walk-through goes through a very specific path for installing OpenStack on Ubuntu 12.04 with root access and specific configuration settings using MySQL for related databases. Fedora 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.

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.

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)	Processor: 64-bit x86 Memory: 12 GB RAM Disk space: 30 GB (SATA or SAS or SSD) Volume storage: two disks with 2 TB (SATA) for volumes attached to the compute nodes Network: one 1 GB Network Interface Card (NIC)	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. 32-bit processors will work for the cloud controller node. The package repositories referred to in this guide do not contain i386 packages.
Compute nodes (runs virtual instances)	Processor: 64-bit x86 Memory: 32 GB RAM Disk space: 30 GB (SATA) Network: two 1 GB NICs	Note that you cannot run 64-bit VM instances on a 32-bit compute node. A 64-bit compute node can run either 32- or 64-bit VMs, however. 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. 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. For Xen-based hypervisors, the Xen wiki contains a list of compatible processors on the HVM Compatible Processors page. For XenServer-compatible Intel processors, refer to the Intel® Virtualization Technology List. For LXC, the VT extensions are not required. The packages referred to in this guide do not contain i386 packages.

Note

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

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

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. For Object Storage, the container and account servers use SQLite, and you can install it as part of the installation process.

Permissions: You can install OpenStack Compute, the Image Service, or Object Storage 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 keeps your cloud controller and compute nodes talking to the same time server to avoid problems 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, you need to install NTP, and if you do a multi-node configuration you will configure one server to be the reference server.

```
# apt-get 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:

```
# sed -i 's/server ntp.ubuntu.com/server ntp.ubuntu.com\nserver 127.127.1.0\
nfudge 127.127.1.0 stratum 10/g' /etc/ntp.conf
# service ntp restart
```

Set up the NTP client on your compute node so that the time between controller node and compute node is synchronized. do the following as a cron job on compute node.

```
ntpdate 'controllernode ip'
hwclock -w
```

2. OpenStack Terminology

Version Names

Each OpenStack release has a name, in increasing alphabetical order (e.g., Essex follows Diablo). There are also version numbers corresponding to these releases, but the numbering schemes are different for OpenStack Compute and OpenStack Object Storage, as shown in the table below:

Table 2.1. OpenStack version names

Release name	Release date	OpenStack Compute version number	OpenStack Object Storage version number
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 Folsom release is scheduled for October 2012.

Code Names

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

Table 2.2. Code names

Service name	Code name
Identity	Keystone
Compute	Nova
Image	Glance
Dashboard	Horizon
Object Storage	Swift

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`.

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**).

3. Installation Assumptions

OpenStack Compute has a large number of configuration options. To simplify this installation guide, we make a number of assumptions about the target installation.

- You have a collection of compute nodes, each installed with Ubuntu Server 12.04
- You have designated one of the nodes as the Cloud Controller, which will run all of the services (RabbitMQ, MySQL, Identity, Image, nova-api, nova-network, nova-scheduler, nova-volume) except for nova-compute.
- 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 "nova-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 (RabbitMQ is the default messaging back-end on Ubuntu, on Fedora the default backend is Qpid).
- `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 **apt-get update** and **apt-get upgrade** prior to the installation.

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.

4. Installing OpenStack Identity Service

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

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:

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

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.

```
# 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.

First, install MySQL as root:

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

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

Use **sed** to edit `/etc/mysql/my.cnf` to change `bind-address` from localhost (127.0.0.1) to any (0.0.0.0) and restart the mysql service, as root:

```
# sed -i 's/127.0.0.1/0.0.0.0/g' /etc/mysql/my.cnf
# service mysql restart
```

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.

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_KEYSTONE_PASSWORD]* with this password.

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

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_KEYSTONE_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.

```
admin_token = 012345SECRET99TOKEN012345
```

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

```
# service keystone restart
```

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.

Setting up tenants, users, and roles

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 keystone.

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

```
$ keystone --token 012345SECRET99TOKEN012345 --endpoint http://192.168.206.
130:35357/v2.0 tenant-create --name openstackDemo --description "Default
 Tenant" --enabled true
```

```
      +-------------+----------------------------------+
      |   Property  |              Value               |
      +-------------+----------------------------------+
      | description | Default Tenant                   |
      | enabled     | true                             |
      | id          | b5815b046cfe47bb891a7b64119e7f80 |
      | name        | openstackDemo                    |
      +-------------+----------------------------------+
```

Create a default user named `adminUser`.

```
$keystone --token 012345SECRET99TOKEN012345 --endpoint http://192.168.206.
130:35357/v2.0 user-create --tenant_id b5815b046cfe47bb891a7b64119e7f80 --name
 adminUser --pass secretword --enabled true
```

```
      +----------
+-------------------------------------------------------------------------------------------
+
      | Property |
 Value                                                          |
      +----------
+-------------------------------------------------------------------------------------------
+
      | email    | None
                                                                |
      | enabled  | true
                                                                |
      | id       | a4c2d43f80a549a19864c89d759bb3fe
                                                                |
      | name     | admin
                                                                |
      | password | $6$rounds=40000$MsFWIgIfbAHnhUH8$vvSK9/Uy3P5BTdH0kn.0MH.
xFHAR2pWQCpTRLTENPs.3w53jb5BbbkIKHnkTbzWW3xVwqsb3W5e./3EIaNPeP0 |
      | tenantId | b5815b046cfe47bb891a7b64119e7f80
                                                                |
      +----------
+-------------------------------------------------------------------------------------------
+
```

Create the default roles, `admin` and `memberRole`.

```
$keystone --token 012345SECRET99TOKEN012345 --endpoint http://192.168.206.
130:35357/v2.0 role-create --name admin
```

```
      +----------+----------------------------------+
      | Property |              Value               |
      +----------+----------------------------------+
      | id       | e3d9d157cc95410ea45d23bbbc2e5c10 |
      | name     | admin                            |
      +----------+----------------------------------+
```

```
$keystone --token 012345SECRET99TOKEN012345 --endpoint http://192.168.206.
130:35357/v2.0 role-create --name memberRole
```

```
      +----------+----------------------------------+
      | Property |              Value               |
      +----------+----------------------------------+
      | id       | cffc2edea9c74b4a8779cc0d7a22fc21 |
      | name     | memberRole                       |
      +----------+----------------------------------+
```

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

```
$keystone --token 012345SECRET99TOKEN012345 --endpoint http://192.168.
206.130:35357/v2.0 user-role-add --user a4c2d43f80a549a19864c89d759bb3fe
 --tenant_id b5815b046cfe47bb891a7b64119e7f80 --role
 e3d9d157cc95410ea45d23bbbc2e5c10
```

There is no output to this command.

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

```
$keystone --token 012345SECRET99TOKEN012345 --endpoint http://192.168.206.
130:35357/v2.0 tenant-create --name service --description "Service Tenant" --
enabled true
```

```
      +-------------+----------------------------------+
      |   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 --token 012345SECRET99TOKEN012345 --endpoint http://192.168.206.
130:35357/v2.0 user-create --tenant_id eb7e0c10a99446cfa14c244374549e9d --name
 glance --pass glance --enabled true
```

```
      +----------
+-------------------------------------------------------------------------------------------
+
      | Property |
 Value                                                                |
      +----------
+-------------------------------------------------------------------------------------------
+
      | email    | None
                                                                     |
      | enabled  | true
                                                                     |
      | id       | 46b2667a7807483d983e0b4037a1623b
                                                                     |
      | name     | glance
                                                                     |
      | password | $6$rounds=40000$kf1ENaCoy7wOfRjx
$LKQtsQbBqSBr2ZH7fwToAut0EYYz6M278N16Xg4Va2vTEOFabvTVXCdCP4hA5ikdCQO8Mh1nJvuFMEvGHaht3/
 |
      | tenantId | eb7e0c10a99446cfa14c244374549e9d
                                                                     |
      +----------
+-------------------------------------------------------------------------------------------
+
```

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

```
$keystone --token 012345SECRET99TOKEN012345 --endpoint http://192.168.
206.130:35357/v2.0 user-role-add --user 46b2667a7807483d983e0b4037a1623b
 --tenant_id eb7e0c10a99446cfa14c244374549e9d --role
 e3d9d157cc95410ea45d23bbbc2e5c10
```

There is no output to this command.

Create a Nova Service User in the Service Tenant.

```
$keystone --token 012345SECRET99TOKEN012345 --endpoint http://192.168.206.
130:35357/v2.0 user-create --tenant_id eb7e0c10a99446cfa14c244374549e9d --name
 nova --pass nova --enabled true
```

```
      +----------
+-------------------------------------------------------------------------------------------
+
      | Property |
 Value                                                                  |
      +----------
+-------------------------------------------------------------------------------------------
+
      | email    | None
                                                                       |
      | enabled  | true
                                                                       |
      | id       | 54b3776a8707834d983e0b4037b1345c
                                                                       |
      | name     | nova
                                                                       |
      | password | $6$rounds=40000$kf1ENaCoy7wOfRjx
$LKQtsQbBqSBr2ZH7fwToAut0EYYz6M278N16Xg4Va2vTEOFabvTVXCdCP4hA5ikdCQO8Mh1nJvuFMEvGHaht3/
 |
      | tenantId | eb7e0c10a99446cfa14c244374549e9d
                                                                       |
      +----------
+-------------------------------------------------------------------------------------------
+
```

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

```
$keystone --token 012345SECRET99TOKEN012345 --endpoint http://192.168.
206.130:35357/v2.0 user-role-add --user 54b3776a8707834d983e0b4037b1345c
 --tenant_id eb7e0c10a99446cfa14c244374549e9d --role
 e3d9d157cc95410ea45d23bbbc2e5c10
```

There is no output to this command.

Create an EC2 Service User in the Service Tenant.

```
$keystone --token 012345SECRET99TOKEN012345 --endpoint http://192.168.206.
130:35357/v2.0 user-create --tenant_id eb7e0c10a99446cfa14c244374549e9d --name
 ec2 --pass ec2 --enabled true
```

```
      +----------
+-------------------------------------------------------------------------------------------
+
      | Property |
 Value                                                              |
      +----------
+-------------------------------------------------------------------------------------------
+
      | email    | None
                                                                    |
      | enabled  | true
                                                                    |
      | id       | 32e7668b8707834d983e0b4037b1345c
                                                                    |
      | name     | ec2
                                                                    |
      | password | $6$rounds=40000$kf1ENaCoy7wOfRjx
$LKQtsQbBqSBr2ZH7fwToAut0EYYz6M278N16Xg4Va2vTEOFabvTVXCdCP4hA5ikdCQO8Mh1nJvuFMEvGHaht3/
 |
      | tenantId | eb7e0c10a99446cfa14c244374549e9d
                                                                    |
      +----------
+-------------------------------------------------------------------------------------------
+
```

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

```
$keystone --token 012345SECRET99TOKEN012345 --endpoint http://192.168.
206.130:35357/v2.0 user-role-add --user 32e7668b8707834d983e0b4037b1345c
 --tenant_id eb7e0c10a99446cfa14c244374549e9d --role
 e3d9d157cc95410ea45d23bbbc2e5c10
```

There is no output to this command.

Create an Object Storage Service User in the Service Tenant.

```
$keystone --token 012345SECRET99TOKEN012345 --endpoint http://192.168.206.
130:35357/v2.0 user-create --tenant_id eb7e0c10a99446cfa14c244374549e9d --name
 swift --pass swiftpass --enabled true
```

```
      +----------
+-------------------------------------------------------------------------------------------
+
      | Property |
 Value                                                              |
      +----------
+-------------------------------------------------------------------------------------------
+
      | email    | None
                                                                    |
      | enabled  | true
                                                                    |
      | id       | 4346677b8909823e389f0b4037b1246e
                                                                    |
      | name     | swift
                                                                    |
      | password | $6$rounds=40000$kf1ENaCoy7wOfRjx
$LKQtsQbBqSBr2ZH7fwToAut0EYYz6M278N16Xg4Va2vTEOFabvTVXCdCP4hA5ikdCQO8Mh1nJvuFMEvGHaht3/
 |
      | tenantId | eb7e0c10a99446cfa14c244374549e9d
                                                                    |
      +----------
+-------------------------------------------------------------------------------------------
+
```

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

```
$keystone --token 012345SECRET99TOKEN012345 --endpoint http://192.168.
206.130:35357/v2.0 user-role-add --user 4346677b8909823e389f0b4037b1246e
 --tenant_id eb7e0c10a99446cfa14c244374549e9d --role
 e3d9d157cc95410ea45d23bbbc2e5c10
```

There is no output to this command.

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:

   `--name` Name of the service (e.g., `nova`, `ec2`, `glance`, `keystone`)

   `--type` Type of service (e.g., `compute`, `ec2`, `image`, `identity`)

   `--description` 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:

   `--region` the region name you've given to the OpenStack cloud you are deploying (e.g., RegionOne)

   `--service_id` The ID field returned by the **keystone service-create** (e.g., `935fd37b6fa74b2f9fba6d907fa95825`)

   `--publicurl` The URL of the public-facing endpoint for the service (e.g., `http://192.168.206.130:9292/v1` or `http://192.168.206.130:8774/v2/%(tenant_id)s`)

   `--internalurl` The URL of an internal-facing endpoint for the service.

   This typically has the same value as `publicurl`.

   `--adminurl` The URL for the admin endpoint for the service. The Keystone and EC2 services use different endpoints for `adminurl` and `publicurl`, 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 interpeted as a special character by Unix shells.

Creating keystone services and service endpoints

Here we define the services and their endpoints.

Define the Identity service:

```
$ keystone --token 012345SECRET99TOKEN012345 \
 --endpoint http://192.168.206.130:35357/v2.0/ \
 service-create \
```

```
 --name=keystone \
 --type=identity \
 --description="Keystone Identity Service"

+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description | Keystone Identity Service        |
| id          | 15c11a23667e427e91bc31335b45f4bd |
| name        | keystone                         |
| type        | identity                         |
+-------------+----------------------------------+
$ keystone --token 012345SECRET99TOKEN012345 \
 --endpoint http://192.168.206.130:35357/v2.0/ \
 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 --token 012345SECRET99TOKEN012345 \
--endpoint http://192.168.206.130:35357/v2.0/ \
service-create \
 --name=nova \
 --type=compute \
 --description="Nova Compute Service"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description | Nova Compute Service             |
| id          | abc0f03c02904c24abdcc3b7910e2eed |
| name        | nova                             |
| type        | compute                          |
+-------------+----------------------------------+

$ keystone --token 012345SECRET99TOKEN012345 \
--endpoint http://192.168.206.130:35357/v2.0/ \
```

```
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 --token 012345SECRET99TOKEN012345 \
 --endpoint http://192.168.206.130:35357/v2.0/ \
service-create \
 --name=volume \
 --type=volume \
 --description="Nova Volume Service"


+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description | Nova Volume Service              |
| id          | 1ff4ece13c3e48d8a6461faebd9cd38f |
| name        | volume                           |
| type        | volume                           |
+-------------+----------------------------------+

$ TENANT=eb7e0c10a99446cfa14c244374549e9d
$ keystone --token 012345SECRET99TOKEN012345 \
 --endpoint http://192.168.206.130:35357/v2.0/ \
 endpoint-create \
 --region RegionOne \
 --service_id=1ff4ece13c3e48d8a6461faebd9cd38f \
 --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/
eb7e0c10a99446cfa14c244374549e9d |
| id          | 1ff4ece13c3e48d8a6461faebd9cd38f
  |
```

```
| internalurl | http://192.168.206.130:8776/v1/
eb7e0c10a99446cfa14c244374549e9d |
| publicurl   | http://192.168.206.130:8776/v1/
eb7e0c10a99446cfa14c244374549e9d |
| region      | RegionOne
  |
| service_id  | 8a70cd235c7d4a05b43b2dffb9942cc0
  |
+-------------
+------------------------------------------------------------------+
```

Define the Image service:

```
$ keystone --token 012345SECRET99TOKEN012345 \
 --endpoint http://192.168.206.130:35357/v2.0/ \
 service-create \
 --name=glance \
 --type=image \
 --description="Glance Image Service"


+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description | Glance 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/v1 \
 --internalurl=http://192.168.206.130:9292/v1 \
 --adminurl=http://192.168.206.130:9292/v1


+-------------+----------------------------------+
|   Property  |             Value                |
+-------------+----------------------------------+
| adminurl    | http://192.168.206.130:9292/v1   |
| id          | 3c8c0d749f21490b90163bfaed9befe7 |
| internalurl | http://192.168.206.130:9292/v1   |
| publicurl   | http://192.168.206.130:9292/v1   |
| region      | RegionOne                        |
| service_id  | 7d5258c490144c8c92505267785327c1 |
+-------------+----------------------------------+
```

Define the EC2 compatibility service:

```
$ keystone  --token 012345SECRET99TOKEN012345 \
 --endpoint http://192.168.206.130:35357/v2.0/ \
 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/Cloud |
| id          | d2a3d7490c61442f9b2c8c8a2083c4b6        |
| internalurl | http://192.168.206.130:8773/services/Cloud |
| publicurl   | http://192.168.206.130:8773/services/Admin |
| region      | RegionOne                               |
| service_id  | 181cdad1d1264387bcc411e1c6a6a5fd        |
+-------------+-----------------------------------------+
```

Define the Object Storage service:

```
$ keystone  --token 012345SECRET99TOKEN012345 \
 --endpoint http://192.168.206.130:35357/v2.0/ \
 service-create \
 --name=swift \
 --type=object-store \
 --description="Object Storage Service"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description | EC2 Compatibility Layer          |
| id          | 272efad2d1234376cbb911c1e5a5a6ed |
| name        | swift                            |
| type        | object-store                     |
+-------------+----------------------------------+

$ keystone --token 012345SECRET99TOKEN012345 \
 --endpoint http://192.168.206.130:35357/v2.0/ \
 endpoint-create \
 --region RegionOne \
 --service_id=272efad2d1234376cbb911c1e5a5a6ed \
 --publicurl 'http://127.0.0.1:8080/v1/AUTH_$(tenant_id)s' \
 --adminurl 'http://127.0.0.1:8080/' \
 --internalurl 'http://127.0.0.1:8080/v1/AUTH_$(tenant_id)s'
```

```
+-------------
+-------------------------------------------------------------------+
|   Property  |                          Value
 |
+-------------
+-------------------------------------------------------------------+
| adminurl    | http://127.0.0.1:8080/
 |
| id          | e32b3c4780e51332f9c128a8c208a5a4
 |
| internalurl | http://192.168.206.130:8080
 |
| publicurl   | http://127.0.0.1:8080/v1/AUTH_eb7e0c10a99446cfa14c244374549e9d
 |
| region      | RegionOne
 |
| service_id  | 272efad2d1234376cbb911c1e5a5a6ed
 |
+-------------
+-------------------------------------------------------------------+
```

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 `keystone.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

Install curl, a command-line tool for running REST API requests along with openssl for meeting a dependency requirement:

```
sudo apt-get install curl openssl
```

Here is a curl command you can use to ensure that the Identity service is working:

```
curl -d '{"auth": {"tenantName": "adminTenant", "passwordCredentials":
{"username": "adminUser", "password": "secretword"}}}' -H "Content-type:
 application/json" http://192.168.206.130:35357/v2.0/tokens | python -
mjson.tool
```

In return, you should receive a token for the adminUser user.

```
{
    "access": {
        "serviceCatalog": {},
        "token": {
            "expires": "2012-04-12T00:40:12Z",
            "id": "cec68088d08747639c682ee5228106d1"
        },
        "user": {
            "id": "6b0141904f09480d810a5949d79ea0f3",
```

```
            "name": "adminUser",
            "roles": [],
            "roles_links": [],
            "username": "adminUser"
        }
    }
}
```

You can also get a token that expires in 24 hours using the adminUser account:

```
curl -d '{"auth": {"tenantName": "openstackDemo", "passwordCredentials":
{"username": "adminUser", "password": "secretword"}}}' -H "Content-type:
 application/json" http://192.168.206.130:35357/v2.0/tokens | python -
mjson.tool
```

In return, you get the token listing shown below.

```
{
    "access": {
        "serviceCatalog": {},
        "token": {
            "expires": "2012-04-12T00:41:21Z",
            "id": "a220bfdf313b404fa5e063fcc7cc1f3e",
            "tenant": {
                "description": "Default Tenant",
                "enabled": true,
                "id": "50af8cc655c24ada96f73010c96b70a2",
                "name": "openstackDemo"
            }
        },
        "user": {
            "id": "6b0141904f09480d810a5949d79ea0f3",
            "name": "adminUser",
            "roles": [],
            "roles_links": [],
            "username": "adminUser"
        }
    }
}
```

5. Installing OpenStack Compute and Image Service

The OpenStack Compute and Image services work together to provide access to virtual servers and images through REST APIs.

Installing and Configuring the Image Service

Install the Image service.

```
sudo apt-get install glance
```

After installing, you need to delete the sqlite database it creates, then change the configuration to point to the mysql database.

Delete the `glance.sqlite` file created in the /var/lib/glance/ directory.

```
sudo 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 'yourpassword';
```

Enter quit at the mysql> prompt to exit MySQL.

```
mysql> quit
```

Edit the Glance configuration files and paste ini middleware files

Update `/etc/glance/glance-api-paste.ini` and configure the admin_* values under [filter:authtoken].

```
[filter:authtoken]
admin_tenant_name = service
admin_user = glance
```

```
admin_password = glance
```

Ensure that the glance-api pipeline section includes authtoken:

```
[pipeline:glance-api]
pipeline = versionnegotiation authtoken auth-context apiv1app
```

Add this to the end of `/etc/glance/glance-api.conf`.

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

Restart glance-api to pick up these changed settings.

```
service glance-api restart
```

Add this to the end of `/etc/glance/glance-registry.conf`.

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

Update `/etc/glance/glance-registry-paste.ini`, configure the admin_* vaules under [filter:authtoken]:

```
[filter:authtoken]
admin_tenant_name = service
admin_user = glance
admin_password = glance
```

Ensure that the glance-registry pipeline section includes authtoken:

```
[pipeline:glance-registry]
#pipeline = context registryapp
# NOTE: use the following pipeline for keystone
pipeline = authtoken auth-context context registryapp
```

Ensure that the glance-registry and glance-scrubber conf files point to the MySQL database rather than sqlite.

```
sql_connection = mysql://glance:yourpassword@192.168.206.130/glance
```

Restart glance-registry and glance-api services.

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

Note

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

On Ubuntu 12.04, the database tables are under version control and you must do these steps on a new install to prevent the Image service from breaking possible upgrades.

```
glance-manage version_control 0
glance-manage db_sync
```

Note

Also note that this guide does not configure image caching, refer to http://glance.openstack.org for more information.

Verifying the Image Service Installation

You can find the version of the installation by using the glance –version command:

```
glance --version
```

The version number 2012.1 corresponds with the Essex release.

```
glance 2012.1
```

Obtain a test image.

```
mkdir /tmp/images
    cd /tmp/images/
    wget http://smoser.brickies.net/ubuntu/ttylinux-uec/ttylinux-uec-amd64-12.
1_2.6.35-22_1.tar.gz
    tar -zxvf ttylinux-uec-amd64-12.1_2.6.35-22_1.tar.gz
```

Upload the kernel.

```
glance --os_username=adminUser --os_password=secretword --os_tenant=
openstackDemo --os_auth_url=http://127.0.0.1:5000/v2.0 add name="tty-linux-
kernel" disk_format=aki container_format=aki < ttylinux-uec-amd64-12.1_2.6.
35-22_1-vmlinuz
```

```
Uploading image 'tty-linux-kernel'
    ================================================================================
================================================================================
=========[100%] 41.8M/s, ETA  0h  0m  0s
    Added new image with ID: 599907ff-296d-4042-a671-d015e34317d2
```

Upload the initrd.

```
glance --os_username=adminUser --os_password=secretword --os_tenant=
openstackDemo --os_auth_url=http://127.0.0.1:5000/v2.0 add name="tty-linux-
ramdisk" disk_format=ari container_format=ari < ttylinux-uec-amd64-12.1_2.6.
35-22_1-loader
```

```
Uploading image 'tty-linux-ramdisk'
    ================================================================================
================================================================================
===[100%] 937.483441K/s, ETA  0h  0m  0s
    Added new image with ID: 7d9f0378-1640-4e43-8959-701f248d999d
```

Upload the image.

```
glance --os_username=adminUser --os_password=secretword --os_tenant=
openstackDemo --os_auth_url=http://127.0.0.1:5000/v2.0 add name="tty-linux"
 disk_format=ami container_format=ami kernel_id=599907ff-296d-4042-a671-
d015e34317d2 ramdisk_id=7d9f0378-1640-4e43-8959-701f248d999d < ttylinux-uec-
amd64-12.1_2.6.35-22_1.img
```

```
Uploading image 'tty-linux'
    ================================================================================
================================================================================
===[100%] 118.480514M/s, ETA  0h  0m  0s
    Added new image with ID: 21b421e5-44d4-4903-9db0-4f134fdd0793
```

Now a glance index should show a legitimate image.

```
glance --os_username=adminUser --os_password=secretword --os_tenant=
openstackDemo --os_auth_url=http://127.0.0.1:5000/v2.0 index
    ID                                   Name                           Disk
 Format          Container Format     Size
    ------------------------------------ ------------------------------
 -------------------- -------------------- --------------
    21b421e5-44d4-4903-9db0-4f134fdd0793 tty-linux                      ami
                ami                       25165824
    7d9f0378-1640-4e43-8959-701f248d999d tty-linux-ramdisk              ari
                ari                          96629
    599907ff-296d-4042-a671-d015e34317d2 tty-linux-kernel               aki
                aki                        4404752
```

Note

This example shows inputting –os_username, –os_password, –os_tenant, –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=adminUser
export OS_TENANT_NAME=openstackDemo
export OS_PASSWORD=secretword
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`**.

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.

Introduction to using Xen, XCP and XenServer with OpenStack

In this section we will describe 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.

Basic 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 toolstack you are using. Make sure you know what toolstack you want before you get started.

XCP is an open source (GPLv2) toolstack for Xen. It is designed specifically as platform for enterprise and cloud computing, and is well integrated with OpenStack.

Citrix XenServer is a commercial product. It is based on XCP, and exposes the same toolstack and managment 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.

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.

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. Any other VM is unprivileged, and are known as a "domU" or "guest". All customer VMs are unprivileged of course, but you should note that on Xen the OpenStack control software (nova-compute) also runs in a domU. This gives a level of security isolation between the privileged system software and the OpenStack sofware (much of which is customer-facing). This architecture is described in more detail later.

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 is an ongoing research project and you shouldn't worry about it right now. The current architecture just has three levels of separation: dom0, the OpenStack domU, and the completely unprivileged customer VMs.

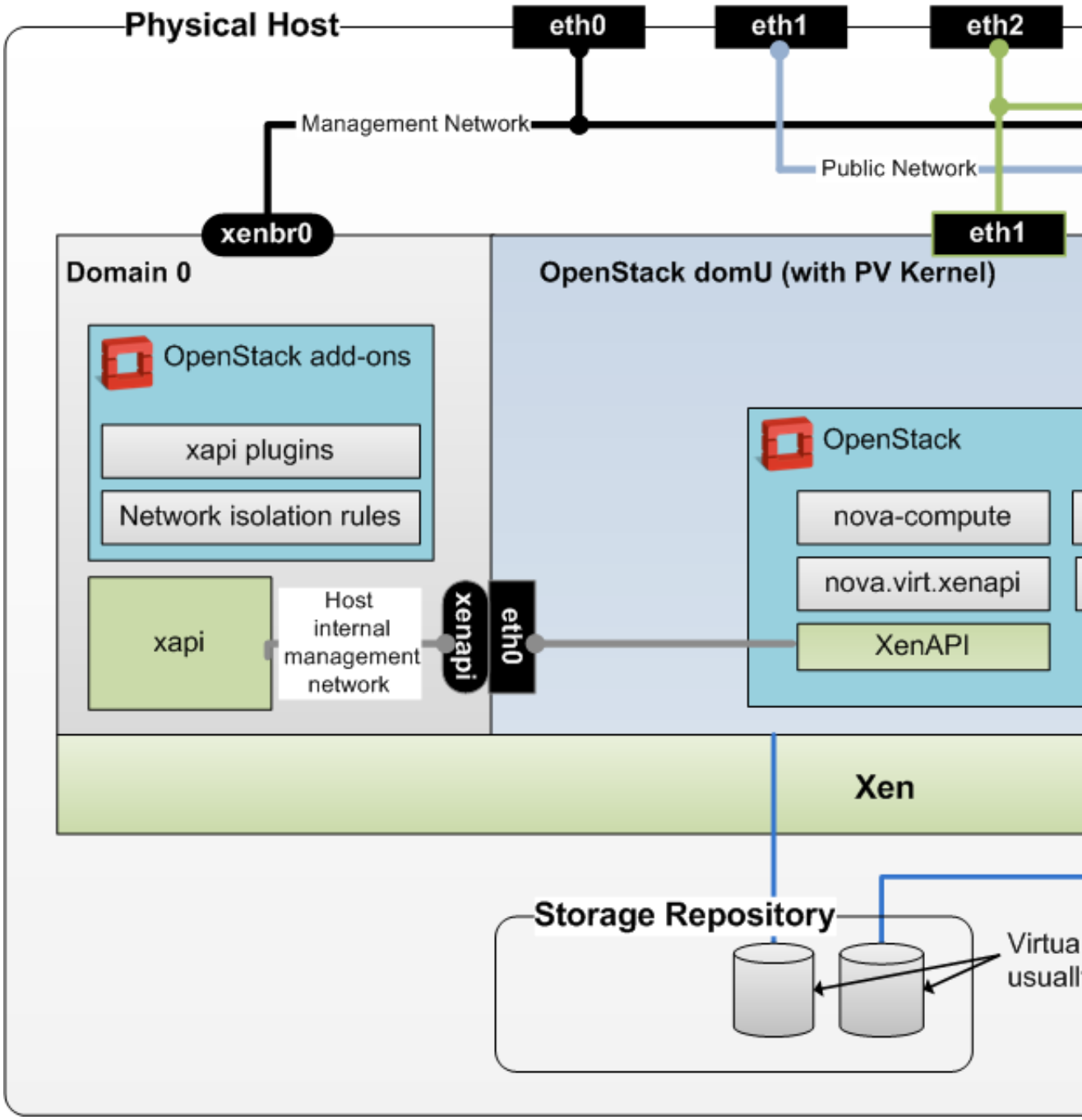
Paravirtualized versus hardware virtualized domains

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.

In OpenStack, customer VMs may run in either PV or HVM mode. However, the OpenStack domU (that's the one running nova-compute) **must** be running in PV mode.

Deployment architecture

When you deploy OpenStack on XCP or XenServer you will get something similar to this:



Key things to note:

- 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 domU: 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 Host Internal 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 traffic (RabbitMQ, MySQL, etc), Tenant network traffic (controlled by nova-network) and Public traffic (floating IPs, public API end points).
- Each network that leaves the host has been put through a separate physical network interface. This is the simplest model, but it's not the only one possible. You may choose to isolate this traffic using VLANs instead, for example.

XenServer pools

Before OpenStack 2012.1 ("Essex"), all XenServer machines used with OpenStack are standalone machines, usually only using local storage.

However in 2012.1 and later, the host-aggregates feature allows you to create pools of XenServer hosts (configuring shared storage is still an out of band activity). This move will enable live migration when using shared storage.

Xen and libvirt

It may 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 today, and using the XCP or XenServer toolstacks with the OpenStack XenAPI backend is the only recommended method.

Further reading

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.

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.

First, setup your /etc/network/interfaces file with these settings:

- eth0: public IP, gateway
- br100: no ports, stp off, fd 0, first address from fixed_range set in nova.conf files.

Here's an example:

```
# 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
```

Also install bridge-utils:

```
sudo apt-get 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-manage network 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
```

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 for the newly-created nova database that has full control of the database.

```
mysql> GRANT ALL ON nova.* TO 'nova'@'%' IDENTIFIED BY
```

```
        'yourpassword';
```

Enter quit at the mysql> prompt to exit MySQL.

```
mysql> quit
```

The database will be populated after running the nova-manage db sync command.

Installing the Cloud Controller

Install the messaging queue server, RabbitMQ. You also have the option of installing Apache Qpid, refer to the Compute Administration Manual for instructions.

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

Install the required nova- packages, and dependencies are automatically installed.

```
sudo apt-get install nova-compute nova-volume nova-vncproxy nova-api nova-
ajax-console-proxy nova-cert nova-consoleauth nova-doc nova-scheduler nova-
network
```

Next set up the database, either MySQL or PostgreSQL, with a database named "nova" and a user named "nova" granting the "nova" user full access to the database. This example goes through MySQL but you can refer to the Compute Admin Manual for PostgreSQL instructions.

Configuring OpenStack Compute

Many of the settings for Compute services are stored in the `/etc/nova/nova.conf` file by default. Here are the relevant settings for getting a minimal install running. Refer to the OpenStack Compute Admin Manual for guidance on more configuration options.

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. 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 root:nova /etc/nova
# chmod 640 /etc/nova/nova.conf
```

The hypervisor is set either by editing `/etc/nova/nova.conf` or referring to `nova-compute.conf` in the `nova.conf` file. 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.

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 http://wiki.openstack.org/NovaConfigOptions.

Note

While Fedora uses the new INI style configuration options, we have found that Ubuntu needs the Diablo-style double-dash (–) in front of each configuration option. This guide is meant to be a generic walk-through eventually so the double-dashes are not included in this guide.

```
auth_strategy=keystone
network_manager=nova.network.manager.FlatDHCPManager
fixed_range=192.168.100.0/24
flat_network_dhcp_start=192.168.100.2
public_interface=eth0
flat_interface=eth0
flat_network_bridge=br100
```

Here is an example nova.conf with commented sections:

```
[DEFAULT]

# LOGS/STATE
verbose=True

# AUTHENTICATION
auth_strategy=keystone

# SCHEDULER
compute_scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler

# VOLUMES
volume_group=nova-volumes
volume_name_template=volume-%08x
iscsi_helper=tgtadm

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

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

# RABBITMQ
rabbit_host=localhost
rabbit_password=yourpassword

# 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
my_ip=192.168.206.130
public_interface=br100
vlan_interface=eth0
flat_network_bridge=br100
flat_interface=eth0
fixed_range=10.0.0.0/24

# NOVNC CONSOLE
novncproxy_base_url=http://192.168.206.130:6080/vnc_auto.html
vncserver_proxyclient_address=192.168.206.130
vncserver_listen=192.168.206.130
```

You also need to configure the api-paste.ini file to enable Keystone as the Identity service. An example api-paste.ini file is included in the Appendix. Update the `/etc/nova/api-paste.ini` file according to the sample file.

Restart the nova- services prior to running db sync, by running as root:

```
# restart nova-api
# restart nova-compute
# restart nova-network
# restart nova-scheduler
# restart nova-vncproxy
# restart nova-volume
```

Configuring the Database for Compute

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

```
sudo 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.

Restart all services in total, just to cover the entire spectrum:

```
sudo restart nova-api
sudo restart nova-compute
sudo restart nova-network
sudo restart nova-scheduler
sudo restart nova-vncproxy
sudo restart nova-volume
sudo restart libvirt-bin
sudo /etc/init.d/rabbitmq-server restart
```

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

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-manage network create private --multi_host=T --
fixed_range_v4=192.168.100.0/24 --bridge_interface=br100 --num_networks=1 --
network_size=256
```

Verifying the Compute Installation

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

```
sudo 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     ubuntu                               nova             enabled
    :-)   2012-04-02 14:06:15
nova-cert        ubuntu                               nova             enabled
    :-)   2012-04-02 14:06:16
nova-volume      ubuntu                               nova             enabled
    :-)   2012-04-02 14:06:14
nova-scheduler   ubuntu                               nova             enabled
    :-)   2012-04-02 14:06:11
nova-network     ubuntu                               nova             enabled
    :-)   2012-04-02 14:06:13
nova-consoleauth ubuntu                               nova             enabled
    :-)   2012-04-02 14:06:10
```

You can find the version of the installation by using the nova-manage command:

```
sudo nova-manage version list
```

The version number 2012.1 corresponds with the Essex release of Compute.

```
2012.1 (2012.1-LOCALBRANCH:LOCALREVISION)
```

Defining Compute and Image Service Credentials

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

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

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

```
export OS_USERNAME=adminUser
export OS_TENANT_NAME=openstackDemo
export OS_PASSWORD=secretword
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 |
+--------------------------------------+------------------+--------+--------+
| 58b71381-1d8a-4cbb-94f2-c67bfced35f5 | tty-linux-kernel | ACTIVE |        |
| 5ca3d2fa-fd89-4edb-af41-ae96136e48ef | tty-linux-ramdisk | ACTIVE |        |
| cc63afc1-9a83-4978-8d75-f19d874fdf94 | tty-linux         | 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 .

In this case, you can install all the nova- packages and dependencies as you did for the Cloud Controller node, or just install nova-network and 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.

The Compute Node is where you configure the Compute network, the networking between your instances.

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 PostgresSQL client packages should be installed on any additional Compute nodes.

Copy the nova.conf from your controller node to all additional compute nodes.

6. Uploading Images

First, download some images that are known to work with OpenStack. The smallest test image is a tty testing image with a username/password of root/password. You can get it from http://images.ansolabs.com/tty.tgz. This is the image we'll use for this walkthrough. Images downloaded from cloud-images.ubuntu.com have a username of ubuntu. 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 to house your image files.

```
sudo mkdir stackimages
```

Download the tty image into your stackimages directory.

```
sudo wget -c http://images.ansolabs.com/tty.tgz -O stackimages/tty.tgz
```

Now expand the image.

```
sudo tar -zxf stackimages/tty.tgz -C stackimages
```

You can get a token in order to upload images using this curl command.

```
curl -d '{"auth": {"tenantName": "openstackDemo", "passwordCredentials":
{"username": "adminUser", "password": "secretword"}}}' -H "Content-type:
 application/json" http://192.168.206.130:35357/v2.0/tokens | python -
mjson.tool
```

Now add the kernel image to the Image Service with glance add commands. In this case we're hard-coding the kernel_id and ramdisk_id values in the third command, but for additional images, you need to get the return values for the first two commands to enter into the third command.

```
glance add -A d1819479-be8b-451d-8682-82c654502ddb name="tty-kernel"
 is_public=true container_format=aki disk_format=aki < stackimages/aki-tty/
image
glance add -A d1819479-be8b-451d-8682-82c654502ddb name="tty-ramdisk"
 is_public=true container_format=ari disk_format=ari < stackimages/ari-tty/
image
glance add -A d1819479-be8b-451d-8682-82c654502ddb name="tty" is_public=true
 container_format=ami disk_format=ami kernel_id=1 ramdisk_id=2 < stackimages/
ami-tty/image
```

For other images, you can get Ubuntu Oneiric (11.10) at http://cloud-images.ubuntu.com/oneiric/current/oneiric-server-cloudimg-amd64-disk1.img.

You can ensure that your Image service has indexed these images by using the glance CLI with an authorization token:

```
glance -A c9640d83-ef4f-4718-a0c2-32437a931196 index
```

In return you should see a listing of images. Here's an example.

```
ID                Name                            Disk Format          Container
 Format     Size
```

```
---------------- ----------------------------- --------------------
 --------------------- --------------
3                tty                           ami                  ami
                  25165824
2                tty-ramdisk                   ari                  ari
                   5882349
1                tty-kernel                    aki                  aki
                   4404752
```

If you see an ECONNREFUSED error or "NotAuthenticated: You are not authenticated" message in return, it means one of your glance-*.conf files is incorrectly configured. Double-check and compare to the files listed in the Appendix. If you get a 401 Unauthorized error, check your environment variables.

7. Installing OpenStack Object Storage

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 7.1. Hardware Recommendations

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

Operating System: OpenStack currently runs on Ubuntu 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.

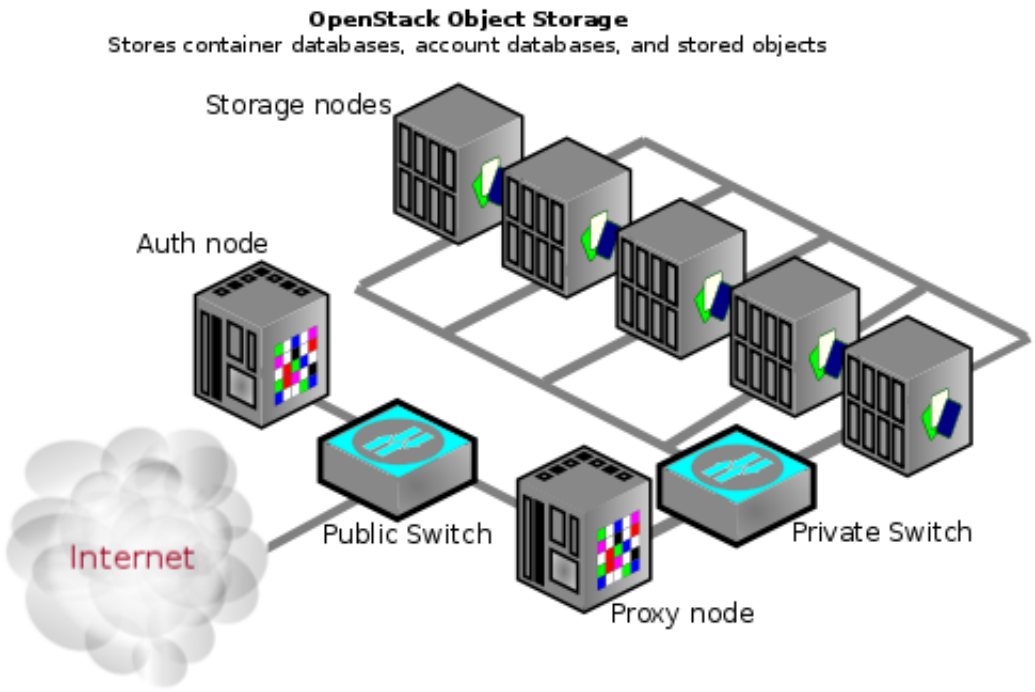
Recommended size: 255 IPs (CIDR /24)

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 Ubuntu Server (10.04 through 12.04) on all nodes.
2. Install the swift service and openSSH.

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

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:

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

2. For every device on the node, setup the XFS volume (/dev/sdb is used as an example):

```
fdisk /dev/sdb  (set up a single partition)
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 /etc/swift/account-server.conf:

```
[DEFAULT]
bind_ip = <STORAGE_LOCAL_NET_IP>
workers = 2

[pipeline:main]
pipeline = account-server

[app:account-server]
use = egg:swift#account

[account-replicator]
```

```
[account-auditor]

[account-reaper]
```

7. Create /etc/swift/container-server.conf:

```
[DEFAULT]
bind_ip = <STORAGE_LOCAL_NET_IP>
workers = 2

[pipeline:main]
pipeline = container-server

[app:container-server]
use = egg:swift#container

[container-replicator]

[container-updater]

[container-auditor]
```

8. Create /etc/swift/object-server.conf:

```
[DEFAULT]
bind_ip = <STORAGE_LOCAL_NET_IP>
workers = 2

[pipeline:main]
pipeline = object-server

[app:object-server]
use = egg:swift#object

[object-replicator]

[object-updater]

[object-auditor]
```

9. Start the storage services:

```
swift-init object-server start
swift-init object-replicator start
swift-init object-updater start
swift-init object-auditor start
swift-init container-server start
swift-init container-replicator start
swift-init container-updater start
swift-init container-auditor start
swift-init account-server start
swift-init account-replicator start
swift-init account-auditor start
```

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:

```
apt-get install swift-proxy memcached
```

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

```
[DEFAULT]
bind_port = 8888
user = <user>

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

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

[filter:keystone]
paste.filter_factory = keystone.middleware.swift_auth:filter_factory
operator_roles = admin, swiftoperator

[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
# Delaying the auth decision is required to support token-less
# usage for anonymous referrers ('.r:*').
delay_auth_decision = true
service_port = 5000
service_host = 127.0.0.1
auth_port = 35357
auth_host = 127.0.0.1
auth_token = 012345SECRET99TOKEN012345
admin_token = 012345SECRET99TOKEN012345

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

[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.

6. 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 ^ 18th, 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
```

7. 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.

8. Verify the ring contents for each ring:

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

9. 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.

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

11. Make sure all the config files are owned by the swift user:

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

12. Start Proxy services:

```
swift-init proxy start
```

Configuring OpenStack Object Storage

Now that the proxy server and storage servers are configured, you want to restart the services.

```
swift-init main start
swift-init rest start
```

You can fine-tune the installation by adding more proxy servers.

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 also changes. You can select from different strategies for load balancing. For example, you could use round robin dns, or an actual 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. Change the default_cluster_url to point to the load balanced url, rather than the first proxy server you created in /etc/swift/proxy-server.conf:

```
[app:auth-server]
use = egg:swift#auth
default_cluster_url = https://<LOAD_BALANCER_HOSTNAME>/v1
# Highly recommended to change this key to something else!
super_admin_key = devauth
```

3. After you change the default_cluster_url setting, you have to delete the auth database and recreate the OpenStack Object Storage users, or manually update the auth database with the correct URL for each account.

4. 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.

5. 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.

Verify the Installation

You can run these commands from the proxy server or any server with access to the Identity Service. Look for the default_swift_cluster setting in the `proxy-server.conf` and match the URLs (including http or https) when issuing auth commands.

1. Run the swift CLI, swift, with the correct Identity service URL.

```
$ swift -V 2 -A http://<AUTH_HOSTNAME>:5000/v2.0 -U adminUser:admin -K
 012345SECRET99TOKEN012345 stat
```

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

```
$ curl -k -v -H 'X-Storage-User: adminUser:admin' -H 'X-Storage-Pass:
 $ADMINPASS' http://<AUTH_HOSTNAME>:5000/auth/v1.0
```

3. 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>
```

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

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

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

Troubleshooting Notes

If you see problems, look in var/log/syslog (or messages on some distros).

Also, at Rackspace we have seen hints at drive failures by looking at error messages in /var/log/kern.log.

8. Installing the OpenStack Dashboard

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.

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:

```
sudo apt-get install -y memcached libapache2-mod-wsgi openstack-dashboard
```

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

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/memcached.conf`.

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

Configuring the Dashboard

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.

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

A full example local_settings.py file is included in the Appendix of the OpenStack Install and Deploy manual.

In the /etc/openstack-dashboard/local_settings.py file, change these options:

- DATABASES: Change the database section to point to the Mysql database named dash:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'dash',
        'USER': 'dash',
        'PASSWORD': 'yourpassword',
        'HOST': 'localhost',
        'default-character-set': 'utf8'
    },
}
```

- SWIFT_ENABLED: If an Object Storage (Swift) endpoint is available and configured in the Identity service catalog, set SWIFT_ENABLED = True.
- QUANTUM_ENABLED: If an Network Connection (Quantum) service is availalbe and configured in the Identity service catalog, set QUANTUM_ENABLED = True. The project is considered core for the Folsom release. You can set also QUANTUM_ENABLED = False.

Run the syncdb command to initialize the 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.
```

If you want to avoid a warning when restarting apache2, create a blackhole directory in the dashboard directory like so:

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

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

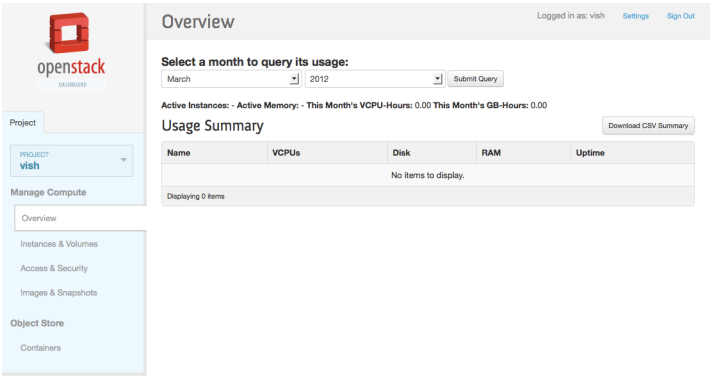
```
sudo /etc/init.d/apache2 restart
```

Restart the nova-api service to ensure the API server can connect to the Dashboard and to avoid an error displayed in the dashboard.

```
sudo restart nova-api
```

Validating the Dashboard Install

To validate the Dashboard installation, point your browser at http://192.168.206.130. Note that you cannot use VNC Console from a Chrome browser. You need both Flash installed and a Firefox browser. 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 "adminUser" for the username and "secretword" as the password.



Overview of VNC Proxy

The VNC Proxy is an OpenStack component that allows users of Nova to access their instances through VNC clients. In Essex and beyond, there is support for for both libvirt and XenServer using both Java and websocket cleints.

The VNC console connection works as follows:

- User connects to API and gets an access_url like `http://ip:port/?token=xyz`.
- User pastes URL in browser or as client parameter.
- Browser/Client connects to proxy.
- 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 what 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.
- 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.

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 for either proxy to work. Many proxies of either type can be run against a single nova-consoleauth service in a cluster configuration.

nova-consoleauth should not be confused with nova-console, which is a Xen-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
```

Important nova-compute Options

To enable vncproxy in your cloud, in addition to running one or both of the proxies and nova-consoleauth, you need to configure the following options in `nova.conf` on your compute hosts.

- `[no]vnc_enabled` - Defaults to enabled. If this option is disabled your instances will launch without VNC support.
- `vncserver_listen` - Defaults to `127.0.0.1`. This is the address that vncservers will bind, and should be overridden in production deployments as a private address. Applies to libvirt only. For multi-host libvirt deployments this should be set to a host management IP on the same network as the proxies.
- `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 vncservers. For all-in-one xen server domU deployments this can be set to 169.254.0.1. For multi-host xen server 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.
- `novncproxy_base_url=`*`[base url for client connections]`* - This is the public base URL to which clients will connect. "?token=abc" will be added to this URL for the purposes of auth. When using the system as described in this document, an appropriate value is "http://$SERVICE_HOST:6080/vnc_auto.html" where *SERVICE_HOST* is a public hostname.
- `xvpvncproxy_base_url=`[base url for client connections] - This is the public base URL to which clients will connect. "?token=abc" will be added to this URL for the purposes of auth. When using the system as described in this document, an appropriate value is "http://$SERVICE_HOST:6081/console" where SERVICE_HOST is a public hostname.

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)
- `xvpvncproxy_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-xvpvncviewer
$ cd nova-xvpvncviewer
$ 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] xvpvnc
```

To run client:

```
$ java -jar VncViewer.jar [access_url]
```

nova-vncproxy replaced with nova-novncproxy

The previous vnc proxy, nova-vncproxy, has been removed from the nova project's source tree and replaced with an improved server that can be found externally at http://github.com/cloudbuilders/noVNC.git

To use this nova-novncproxy:

```
$ git clone http://github.com/cloudbuilders/noVNC.git
$ utils/nova-novncproxy --flagfile=[path to nova.conf options file]
```

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 `0.0.0.0:6080`. This can be configured in `nova.conf` with:

- `novncproxy_port=[port]`
- `novncproxy_host=[host]`

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, Horizon, to access browser-based VNC consoles for instances.

Frequently asked questions about VNC access to VMs

Q: What has changed since Diablo?

A: Previously, VNC support was done differently for libvirt and xen. Now, there is unified multi-hypervisor support. To support this change, configuration options have been added and changed. Also, a new required service called nova-consoleauth has been added. If you are upgrading from Diablo, you will have to take these changes into consideration when upgrading.

If you are using Diablo, please see the documentation that shipped with your code, as this information will not be relevant.

Q: What happened to Diablo's nova-vncproxy?

A: nova-vncproxy was removed from the nova source tree. The Essex analog for this process is nova-novncproxy, which is provided by an external project.

Q: Why is nova-vncproxy no longer part of nova?

A: In Diablo, we shipped a websocket proxy (nova-vncproxy) with nova, but it had poor browser support. This nova-vncproxy code was dependent on external noVNC code, so changes to that system involved updating 2 projects. Due to the rapid evolution of websocket tech, and the tight dependence of the websocket proxy on javscript and html components, we decided to keep that code all in one place.

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 horizon. 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 Horizon, 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 novncproxy_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_proxyclient_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+). Also, if you are using Diablo's nova-vncproxy, note that support for this protocol is not provided.

Appendix A. Appendix: Configuration File Examples

Included for your reference are all configuration files.

keystone.conf

The Identity service's configuration file is found in /etc/keystone/keystone.conf. This file needs to be modified after installing to use SQL for endpoint data and to replace the ADMIN key with the one 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's registry, which stores the metadata about images, is found in /etc/glance/glance-registry.conf. This file needs to be modified after installing.

```
[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

# Send logs to syslog (/dev/log) instead of to file specified by `log_file`
use_syslog = False

# 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:////var/lib/glance/glance.sqlite
sql_connection = mysql://glance:yourpassword@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

[pipeline:glance-registry]
# pipeline = context registryapp
# NOTE: use the following pipeline for keystone
# pipeline = authtoken keystone_shim context registryapp
```

```
pipeline = authtoken keystone_shim registryapp

[app:registryapp]
paste.app_factory = glance.registry.server:app_factory

[filter:context]
context_class = glance.registry.context.RequestContext
paste.filter_factory = glance.common.context:filter_factory

[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
service_protocol = http
service_host = 127.0.0.1
service_port = 5000
auth_host = 127.0.0.1
auth_port = 35357
auth_protocol = http
auth_uri = http://127.0.0.1:5000/
admin_token = 012345SECRET99TOKEN012345

[filter:keystone_shim]
paste.filter_factory = keystone.middleware.glance_auth_token:filter_factory
```

glance-registry-paste.ini

The Identity service's API middleware pipeline is found in /etc/glance/glance-registry-paste.ini. This file needs to be modified after installing.

```
[pipeline:glance-registry]
pipeline = authtoken auth-context context registryapp

[app:registryapp]
paste.app_factory = glance.common.wsgi:app_factory
glance.app_factory = glance.registry.api.v1:API

[filter:context]
context_class = glance.registry.context.RequestContext
paste.filter_factory = glance.common.wsgi:filter_factory
glance.filter_factory = glance.common.context:ContextMiddleware

[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
auth_host = 192.168.206.130
auth_port = 35357
auth_protocol = http
auth_uri = http://192.168.206.130:5000/
admin_tenant_name = service
admin_user = glance
admin_password = glance

[filter:auth-context]
context_class = glance.registry.context.RequestContext
paste.filter_factory = glance.common.wsgi:filter_factory
```

glance-api.conf

The configuration file for the Identity API is found in /etc/glance/glance-api.conf. You need to change this file to look like this example after installing from packages.

```
[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 store should Glance use by default is not specified
# in a request to add a new image to Glance? Default: 'file'
# Available choices are 'file', 'swift', and 's3'
default_store = file

# Address to bind the API server
bind_host = 0.0.0.0

# Port the bind the API server to
bind_port = 9292

# Address to find the registry server
registry_host = 0.0.0.0

# Port the registry server is listening on
registry_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/api.log

# Send logs to syslog (/dev/log) instead of to file specified by `log_file`
use_syslog = False

# ============ 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) 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_topic = glance_notifications

# ============ Filesystem Store Options ========================

# Directory that the Filesystem backend store
# writes image data to
filesystem_store_datadir = /var/lib/glance/images/

# ============ Swift Store Options =============================

# Address where the Swift authentication service lives
swift_store_auth_address = 127.0.0.1:8080/v1.0/
```

```
# User to authenticate against the Swift authentication service
swift_store_user = 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

# ============ S3 Store Options =============================

# Address where the S3 authentication service lives
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

# ============ Image Cache Options ======================

image_cache_enabled = False
```

```
# Directory that the Image Cache writes data to
# Make sure this is also set in glance-pruner.conf
image_cache_datadir = /var/lib/glance/image-cache/

# Number of seconds after which we should consider an incomplete image to be
# stalled and eligible for reaping
image_cache_stall_timeout = 86400

# ============ Delayed Delete Options =============================

# Turn on/off delayed delete
delayed_delete = False

[pipeline:glance-api]
# pipeline = versionnegotiation context apiv1app
# NOTE: use the following pipeline for keystone
# pipeline = versionnegotiation authtoken context apiv1app

# To enable Image Cache Management API replace pipeline with below:
# pipeline = versionnegotiation context imagecache apiv1app
# NOTE: use the following pipeline for keystone auth (with caching)
# pipeline = versionnegotiation authtoken context imagecache apiv1app
pipeline = versionnegotiation authtoken keystone_shim apiv1app

[pipeline:versions]
pipeline = versionsapp

[app:versionsapp]
paste.app_factory = glance.api.versions:app_factory

[app:apiv1app]
paste.app_factory = glance.api.v1:app_factory

[filter:versionnegotiation]
paste.filter_factory =
 glance.api.middleware.version_negotiation:filter_factory

[filter:imagecache]
paste.filter_factory = glance.api.middleware.image_cache:filter_factory

[filter:context]
paste.filter_factory = glance.common.context:filter_factory

[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
service_protocol = http
service_host = 127.0.0.1
service_port = 5000
auth_host = 127.0.0.1
auth_port = 35357
auth_protocol = http
auth_uri = http://127.0.0.1:5000/v2.0
admin_token = 012345SECRET99TOKEN012345

[filter:keystone_shim]
context_class = glance.registry.context.RequestContext
paste.filter_factory = keystone.middleware.glance_auth_token:filter_factory
```

glance-api-paste.ini

The Identity service's API middleware pipeline is found in /etc/glance/glance-api-paste.ini. This file needs to be modified after installing.

```
[pipeline:glance-api]
pipeline = versionnegotiation authtoken auth-context apiv1app

# To enable Image Cache Management API replace pipeline with below:
# pipeline = versionnegotiation context imagecache apiv1app
# NOTE: use the following pipeline for keystone auth (with caching)
# pipeline = versionnegotiation authtoken auth-context imagecache apiv1app

[app:apiv1app]
paste.app_factory = glance.common.wsgi:app_factory
glance.app_factory = glance.api.v1.router:API

[filter:versionnegotiation]
paste.filter_factory = glance.common.wsgi:filter_factory
glance.filter_factory =
 glance.api.middleware.version_negotiation:VersionNegotiationFilter

[filter:cache]
paste.filter_factory = glance.common.wsgi:filter_factory
glance.filter_factory = glance.api.middleware.cache:CacheFilter

[filter:cachemanage]
paste.filter_factory = glance.common.wsgi:filter_factory
glance.filter_factory = glance.api.middleware.cache_manage:CacheManageFilter

[filter:context]
paste.filter_factory = glance.common.wsgi:filter_factory
glance.filter_factory = glance.common.context:ContextMiddleware

[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
service_protocol = http
service_host = 127.0.0.1
service_port = 5000
auth_host = 127.0.0.1
auth_port = 35357
auth_protocol = http
auth_uri = http://127.0.0.1:5000/
admin_tenant_name = service
admin_user = glance
admin_password = glance

[filter:auth-context]
paste.filter_factory = glance.common.wsgi:filter_factory
glance.filter_factory =
 keystone.middleware.glance_auth_token:KeystoneContextMiddleware

[paste_deploy]
flavor = keystone
```

glance-scrubber.conf

An additional configuration file for the Identity service is found in /etc/glance/glance-scrubber.conf. You need to ensure you point to the mysql database in 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

# 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 db for new items to schedule for delete
wakeup_time = 300

# 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#$
sql_connection = mysql://glance:yourpassword@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

[app:glance-scrubber]
paste.app_factory = glance.store.scrubber:app_factory
```

nova.conf

The configuration file for Compute (nova) settings is stored in /etc/nova/nova.conf. To see a list of all possible configuration options for this file, refer to the OpenStack wiki at http://wiki.openstack.org/NovaConfigOptions.

```
[DEFAULT]

# LOGS/STATE
verbose=True
```

```
# AUTHENTICATION
auth_strategy=keystone

# SCHEDULER
compute_scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler

# VOLUMES
volume_group=nova-volumes
volume_name_template=volume-%08x
iscsi_helper=tgtadm

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

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

# RABBITMQ
rabbit_host=localhost
rabbit_password=yourpassword

# 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
my_ip=192.168.206.130
public_interface=br100
vlan_interface=eth0
flat_network_bridge=br100
flat_interface=eth0
fixed_range=10.0.0.0/24

# NOVNC CONSOLE
novncproxy_base_url=http://192.168.206.130:6080/vnc_auto.html
vncserver_proxyclient_address=192.168.206.130
vncserver_listen=192.168.206.130
```

api-paste.ini

The configuration file for Compute (Nova) for the EC2 API and OpenStack Compute API is stored in /etc/nova/api-paste.ini. You should check the admin token in this file to ensure it matches the one you created when setting up the Identity service. You should also add all the Keystone configuration specified in the sample file below.

```
############
```

```
# Metadata #
############
[composite:metadata]
use = egg:Paste#urlmap
/: metaversions
/latest: meta
/1.0: meta
/2007-01-19: meta
/2007-03-01: meta
/2007-08-29: meta
/2007-10-10: meta
/2007-12-15: meta
/2008-02-01: meta
/2008-09-01: meta
/2009-04-04: meta

[pipeline:metaversions]
pipeline = ec2faultwrap logrequest metaverapp

[pipeline:meta]
pipeline = ec2faultwrap logrequest metaapp

[app:metaverapp]
paste.app_factory = nova.api.metadata.handler:Versions.factory

[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
deprecated = ec2faultwrap logrequest authenticate 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:totoken]
paste.filter_factory = nova.api.ec2:EC2Token.factory

[filter:ec2keystoneauth]
paste.filter_factory = nova.api.ec2:EC2KeystoneAuth.factory

[filter:ec2noauth]
```

```
paste.filter_factory = nova.api.ec2:NoAuth.factory

[filter:authenticate]
paste.filter_factory = nova.api.ec2:Authenticate.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 noauth ratelimit osapi_compute_app_v2
deprecated = faultwrap auth ratelimit osapi_compute_app_v2
keystone = faultwrap authtoken keystonecontext ratelimit osapi_compute_app_v2
keystone_nolimit = faultwrap authtoken keystonecontext osapi_compute_app_v2

[composite:openstack_volume_api_v1]
use = call:nova.api.auth:pipeline_factory
noauth = faultwrap noauth ratelimit osapi_volume_app_v1
deprecated = faultwrap auth ratelimit osapi_volume_app_v1
keystone = faultwrap authtoken keystonecontext ratelimit osapi_volume_app_v1
keystone_nolimit = faultwrap authtoken keystonecontext osapi_volume_app_v1

[filter:faultwrap]
paste.filter_factory = nova.api.openstack:FaultWrapper.factory

[filter:auth]
paste.filter_factory = nova.api.openstack.auth:AuthMiddleware.factory

[filter:noauth]
paste.filter_factory = nova.api.openstack.auth:NoAuthMiddleware.factory

[filter:ratelimit]
paste.filter_factory =
 nova.api.openstack.compute.limits:RateLimitingMiddleware.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 osvolumeversionapp

[app:osvolumeversionapp]
paste.app_factory = nova.api.openstack.volume.versions:Versions.factory

##########
# Shared #
##########

[filter:keystonecontext]
paste.filter_factory = nova.api.auth:NovaKeystoneContext.factory

[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
service_protocol = http
service_host = 127.0.0.1
service_port = 5000
auth_host = 127.0.0.1
auth_port = 35357
auth_protocol = http
auth_uri = http://127.0.0.1:5000/
admin_tenant_name = service
admin_user = nova
admin_password = nova
```

Credentials (openrc)

This file contains the credentials used by Compute, Image, and Identity services, you can optionally store in /home/openrc. The important concept to avoid errors is to ensure that it is sourced in the environment from which you issue commands. Run "env | grep OS_" or "env | grep NOVA_" to view what is being used in your environment.

```
export OS_USERNAME=adminUser
export OS_TENANT_NAME=openstackDemo
export OS_PASSWORD=secretword
export OS_AUTH_URL=http://192.168.206.130:5000/v2.0/
export OS_REGION_NAME=RegionOne
```

Dashboard configuration

This file contains the database and configuration settings for the OpenStack Dashboard.

```
import os

DEBUG = True
```

```
TEMPLATE_DEBUG = DEBUG
PROD = False
USE_SSL = False

LOCAL_PATH = os.path.dirname(os.path.abspath(__file__))
#DATABASES = {
#    'default': {
#        'ENGINE': 'django.db.backends.sqlite3',
#        'NAME': os.path.join(LOCAL_PATH, 'dashboard_openstack.sqlite3'),
#    },
#}

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'dash',
        'USER': 'dash',
        'PASSWORD': 'yourpassword',
        'HOST': 'localhost',
        'default-character-set': 'utf8'
    },
}

CACHE_BACKEND = 'dummy://'

SESSION_ENGINE = 'django.contrib.sessions.backends.cached_db'

# 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'

# django-mailer uses a different settings attribute
MAILER_EMAIL_BACKEND = EMAIL_BACKEND

# 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!'


OPENSTACK_KEYSTONE_URL = "http://localhost:5000/v2.0/"
# FIXME: this is only needed until keystone fixes its GET /tenants call
# so that it doesn't return everything for admins
OPENSTACK_KEYSTONE_ADMIN_URL = "http://localhost:35357/v2.0"
OPENSTACK_KEYSTONE_DEFAULT_ROLE = "Member"

# NOTE(tres): Available services should come from the service
#             catalog in Keystone.
SWIFT_ENABLED = False

# Configure quantum connection details for networking
QUANTUM_ENABLED = False
QUANTUM_URL = '127.0.0.1'
QUANTUM_PORT = '9696'
QUANTUM_TENANT = '1234'
QUANTUM_CLIENT_VERSION='0.1'

# If you have external monitoring links
```

```
EXTERNAL_MONITORING = [
    ['Nagios','http://foo.com'],
    ['Ganglia','http://bar.com'],
]

# If you do not have external monitoring links
# EXTERNAL_MONITORING = []

# Uncomment the following segment to silence most logging
# django.db and boto DEBUG logging is extremely verbose.
#LOGGING = {
#        'version': 1,
#        # set to True will disable all logging except that specified, unless
#        # nothing is specified except that django.db.backends will still log,
#        # even when set to True, so disable explicitly
#        'disable_existing_loggers': False,
#        'handlers': {
#            'null': {
#                'level': 'DEBUG',
#                'class': 'django.utils.log.NullHandler',
#                },
#            'console': {
#                'level': 'DEBUG',
#                'class': 'logging.StreamHandler',
#                },
#            },
#        'loggers': {
#            # Comment or Uncomment these to turn on/off logging output
#            'django.db.backends': {
#                'handlers': ['null'],
#                'propagate': False,
#                },
#            'django_openstack': {
#                'handlers': ['null'],
#                'propagate': False,
#            },
#        }
#}

# How much ram on each compute host?
COMPUTE_HOST_RAM_GB = 16
```

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 = fLIbertYgibbitZ
```

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 = <user>

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

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

[filter:keystone]
paste.filter_factory = keystone.middleware.swift_auth:filter_factory
operator_roles = admin, swiftoperator

[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
# Delaying the auth decision is required to support token-less
# usage for anonymous referrers ('.r:*').
delay_auth_decision = true
service_port = 5000
service_host = 127.0.0.1
auth_port = 35357
auth_host = 127.0.0.1
auth_token = 012345SECRET99TOKEN012345
admin_token = 012345SECRET99TOKEN012345

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

[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]
```

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]
```