

# OpenStack Compute

## Starter Guide

diablo (Nov 11, 2011)



## OpenStack Compute Starter Guide

diablo (2011-11-11)

Copyright © 2011 CSS Corp Private Limited ( <http://www.csscorp.com/> ) Some rights reserved.

This is a tutorial style beginner's guide for OpenStack™ on Ubuntu 11.10, Oneiric Ocelot. The aim is to help the reader in setting up a minimal installation of OpenStack.



Except where otherwise noted, this document is licensed under  
**Creative Commons Attribution NonCommercial ShareAlike 3.0 License.**  
<http://creativecommons.org/licenses/by-nc-sa/3.0/legalcode>

## List of Tables

2.1. Configuration .....	8
--------------------------	---

# 1. Introduction to OpenStack and Its Components

## Table of Contents

Cloud Computing .....	1
OpenStack .....	1
Open Stack Compute Infrastructure ( Nova ) .....	2
OpenStack Storage Infrastructure (Swift) .....	5
OpenStack Imaging Service ( Glance ) .....	6

## Cloud Computing

Cloud computing is a computing model, where resources such as computing power, storage, network and software are abstracted and provided as services on the Internet in a remotely accessible fashion. Billing models for these services are generally similar to the ones adopted for public utilities. On-demand availability, ease of provisioning, dynamic and virtually infinite scalability are some of the key attributes of cloud computing.

An infrastructure setup using the cloud computing model is generally referred to as the "cloud". The following are the broad categories of services available on the cloud:

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)

Amazon Web Services (AWS) is one of the major players providing IaaS. AWS have two popular services - Elastic Compute Cloud (EC2) and Simple Storage Service (S3), available through web services.

## OpenStack

OpenStack is a collection of open source software projects that enterprises/service providers can use to setup and run their cloud compute and storage infrastructure. Rackspace and NASA are the key initial contributors to the stack. Rackspace contributed their "Cloud Files" platform (code) to power the Object Storage part of the OpenStack, while NASA contributed their "Nebula" platform (code) to power the Compute part. OpenStack consortium has managed to have more than 100 members including Canonical, Dell, Citrix etc. in less than a year.

OpenStack makes its services available through Amazon EC2/S3 compatible APIs and hence the client tools written for AWS can be used with OpenStack as well.

There are 3 main service families under OpenStack

- Compute Infrastructure (Nova)
- Storage Infrastructure (Swift)
- Imaging Service (Glance)

## Open Stack Compute Infrastructure ( Nova )

Nova is the Computing Fabric controller for the OpenStack Cloud. All activities needed to support the life cycle of instances within the OpenStack cloud are handled by Nova. This makes Nova a Management Platform that manages compute resources, networking, authorization, and scalability needs of the OpenStack cloud. But, Nova does not provide any virtualization capabilities by itself; instead, it uses libvirt APIs to interact with the supported hypervisors. Nova exposes all its capabilities through a web services API that is compatible with the EC2 API of Amazon Web Services.

### Functions and Features:

- Instance life cycle management
- Management of compute resources
- Networking and Authorization
- REST-based API
- Asynchronous eventually consistent communication
- Hypervisor agnostic : support for Xen, XenServer/XCP, KVM, UML, VMware vSphere and Hyper-V

### Components of OpenStack Compute

Nova Cloud Fabric is composed of the following major components:

- API Server ( nova-api )
- Message Queue ( rabbit-mq server )
- Compute Workers ( nova-compute )
- Network Controller ( nova-network )
- Volume Worker ( nova-volume )
- Scheduler ( nova-scheduler )



Queue Protocol). Nova uses asynchronous calls for request response, with a call-back that gets triggered once a response is received. Since asynchronous communication is used, none of the user actions get stuck for long in a waiting state. This is especially true since many actions expected by the API calls such as launching an instance or uploading an image are time consuming.

### Compute Worker ( nova-compute )

Compute workers deal with instance management life cycle. they receive the requests for life cycle management via the Message Queue and carry out operations. There are several Compute Workers in a typical production cloud deployment. An instance is deployed on any of the available compute worker based on the scheduling algorithm used.

### Network Controller ( nova-network )

The Network Controller deals with the network configuration of host machines. It does operations like allocating IP addresses, configuring VLANs for projects, implementing security groups and configuring networks for compute nodes.

### Volume Workers ( nova-volume )

Volume workers are used for the management of LVM-based instance volumes. Volume Workers perform volume related functions such as creation, deletion, attaching a volume to an instance, and detaching a volume from an instance. Volumes provide a way of providing persistent storage for use by instances, as the main disk attached to an instance is non-persistent and any changes made to it are lost when the volume is detached or the instance is terminated. When a volume is detached from an instance or when an instance, to which the volume is attached, is terminated, it retains the data that was stored on it when it was attached to an instance earlier. This data can be accessed by reattaching the volume to the same instance or by attaching it to another instances.

Any valuable data that gets accumulated during the life cycle of an instance should be written to a volume, so that it can be accessed later. This typically applies to the storage needs of database servers etc.

### Scheduler ( nova-scheduler )

The scheduler maps the nova-API calls to the appropriate openstack components. It runs as a daemon named nova-schedule and picks up a compute/network/volume server from a pool of available resources depending upon the scheduling algorithm in place. A scheduler can base its decisions on various factors such as load, memory, physical distance of the availability zone, CPU architecture, etc. The nova scheduler implements a pluggable architecture.

Currently the nova-scheduler implements a few basic scheduling algorithms:

- chance: In this method, a compute host is chosen randomly across availability zones.
- availability zone: Similar to chance, but the compute host is chosen randomly from within a specified availability zone.
- simple: In this method, hosts whose load is least are chosen to run the instance. The load information may be fetched from a load balancer.

## OpenStack Storage Infrastructure (Swift)

Swift provides a distributed, eventually consistent virtual object store for OpenStack. It is analogous to Amazon Web Services - Simple Storage Service (S3). Swift is capable of storing billions of objects distributed across nodes. Swift has built-in redundancy and failover management and is capable of archiving and media streaming. It is extremely scalable in terms of both size (Several petabytes) and capacity (Number of objects).

### Functions and Features

- Storage of large number of objects
- Storage of large sized objects
- Data Redundancy
- Archival capabilities - Work with large datasets
- Data container for virtual machines and cloud apps
- Media Streaming capabilities
- Secure storage of objects
- Backup and archival
- Extreme scalability

### Components of Swift

- Swift Account
- Swift Container
- Swift Object
- Swift Proxy
- The RING

### Swift Proxy Server

The consumers interact with the Swift setup through the proxy server using the Swift API. The proxy server acts as a gatekeeper and receives requests from the world. It looks up the location of the appropriate entities and routes the requests to them.

The proxy server also handles failures of entities by rerouting requests to failover entities (handoff entities)

### Swift Object Server

The Object server is a blob store. Its responsibility is to handle storage, retrieval and deletion of objects stored in the local storage. Objects are typically binary files stored in the filesystem with metadata contained as extended file attributes (xattr).

Note: xattr is supported in several filesystems such as ext3, ext4, XFS, Btrfs, JFS and ReiserFS in Linux. But it is known to work best under XFS, JFS, ReiserFS, Reiser4, and ZFS. XFS is considered to be the best option.

## Swift Container server

The container server lists the objects in a container. The lists are stored as SQLite files. The container server also tracks the statistics like the number of objects contained and the storage size occupied by a container.

## Swift Account Server

The account server lists containers the same way a container server lists objects.

## The Ring

The ring contains information about the physical location of the objects stored inside Swift. It is a virtual representation of mapping of names of entities to their real physical location. It is analogous to an indexing service that various processes use to lookup and locate the real physical location of entities within the cluster. Entities like Accounts, Containers, Objects have their own separate rings.

## OpenStack Imaging Service ( Glance )

OpenStack Imaging Service is a lookup and retrieval system for virtual machine images. It can be configured to use any one of the following 3 storage backends:

- OpenStack Object Store to store images
- S3 storage directly
- S3 storage with Object Store as the intermediate for S3 access.

## Functions and Features ( Glance )

- Provides imaging service

## Components of OpenStack Imaging Service ( Glance )

- Glance-control
- Glance-registry

## 2. Installation and Configuration

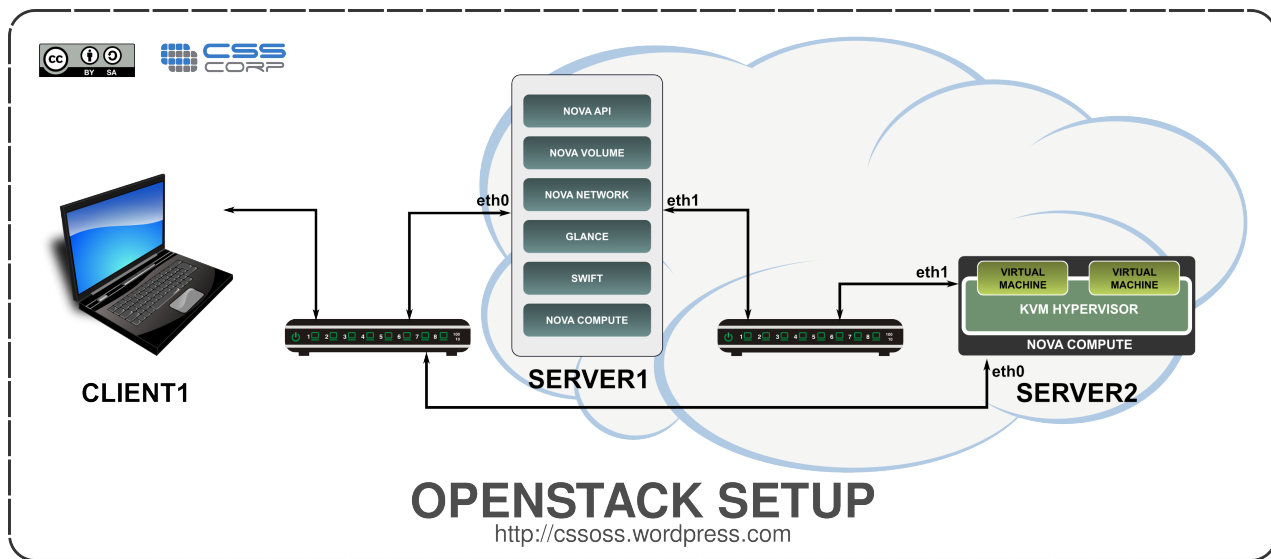
### Table of Contents

Introduction .....	7
Server1 .....	8
Base OS .....	8
Networking Configuration .....	9
NTP Server .....	9
Databases .....	10
Glance .....	11
Nova .....	12
Swift .....	16
Server2 .....	24
BaseOS .....	24
Networking Configuration .....	24
NTP Client .....	24
Nova Components (nova-compute alone) .....	25
Client1 .....	26
BaseOS .....	26
Networking Configuration .....	26
NTP Client .....	26
Client Tools .....	26

### Introduction

The following section describes how to set up a minimal cloud infrastructure based on OpenStack using 3 machines. These machines are referred to in this and subsequent chapters as Server1 and Server2 and Client1. Server1 runs all the components of Nova as well as Glance and Swift. Server2 runs only nova-compute. Since OpenStack components follow a shared-nothing policy, each component or any group of components can be installed on any server.

Client1 is not a required component. In our sample setup, it is used for bundling images, as a client to the web interface, and to run euca commands to manage the infrastructure. Having this client ensures that you do not need to meddle with the servers for tasks such as bundling. Also, bundling of Desktop Systems including Windows will require a GUI and it is better to have a dedicated machine for this purpose. We would recommend this machine to be VT-Enabled so that KVM can be run which allows for Windows VMs during image creation for bundling.



The installation steps use certain specifics such as host names/IP addresses etc. Modify them to suit your environment before using them. The following table summarizes these specifics.

**Table 2.1. Configuration**

	Server1	Server2	Client1
Functionality	All components of OpenStack including nova-compute	nova-compute	Client
Network Interfaces	eth0 - Public N/W, eth1 - Private N/W	eth0 - Public N/W, eth1 - Private N/W	eth0 - Public N/W
IP addresses	eth0 - 10.10.10.2, eth1 - 192.168.3.1	eth0 - 10.10.10.3, eth1 - 192.168.3.2	eth0 - 10.10.10.4
Hostname	server1.example.com	server2.example.com	client.example.com
DNS servers	10.10.10.3	10.10.10.3	10.10.10.3
Gateway IP	10.10.10.1	10.10.10.1	10.10.10.1

## Server1

As shown in the figure above, Server1 contains all nova- services including nova-compute, nova-api, nova-volume, nova-network, as well as the Image Service Glance and Swift. It contains two Network Interface Cards (NICs).

## Base OS

Install 64 bit version of Ubuntu server 11.10 keeping the following configurations in mind.

- Create the first user with the name 'localadmin'.
- Installation lets you setup the IP address for the first interface i.e. eth0. Set the IP address details.
- During installation select only Openssh-server in the packages menu.

We will also be running nova-volume on this server and it is ideal to have a dedicated partition for the use of nova-volume. So, ensure that you choose manual partitioning scheme while installing Ubuntu Server and create a dedicated partition with adequate amount of space for this purpose. We have referred to this partition in the rest of the chapter as /dev/sda6. You can substitute the correct device name of this dedicated partition based on your local setup while following the instructions. Also ensure that the partition type is set as Linux LVM (8e) using fdisk either during install or immediately after installation is over.

Update the machine using the following commands.

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

Install bridge-utils:

```
sudo apt-get install -y bridge-utils
```

Reboot the server and login as the admin user(localadmin) created during the OS installation.

## Networking Configuration

Edit the /etc/network/interfaces file so as to look like this:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
    address 10.10.10.2
    netmask 255.255.255.0
    broadcast 10.10.10.255
    gateway 10.10.10.1
    dns-nameservers 10.10.10.3

auto eth1
iface eth1 inet static
    address 192.168.3.1
    netmask 255.255.255.0
    network 192.168.3.0
    broadcast 192.168.3.255
```

Restart the network now

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

## NTP Server

Install NTP package. This server is going to act as an NTP server for the nodes. The time on all components of OpenStack will have to be in sync. We can run NTP server on this and have other components sync to it.

```
sudo apt-get install -y ntp
```

Open the file `/etc/ntp.conf` and add the following lines to make sure that the time of the server is in sync with an external server and in case Internet connectivity is down, NTP server uses its own hardware clock as the fallback.

```
server ntp.ubuntu.com
server 127.127.1.0
fudge 127.127.1.0 stratum 10
```

Restart NTP service to make the changes effective

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

## Databases

You can use MySQL, PostgreSQL or SQLite for Nova and Glance. Depending upon your choice of database, you will need to install the necessary packages and configure the database server.

### MySQL

Install `mysql-server` package

```
sudo apt-get install -y mysql-server
```

Create the root password for mysql. The password here is "mygreatsecret"

Change the bind address from `127.0.0.1` to `0.0.0.0` in `/etc/mysql/my.cnf` and it will look like this:

```
bind-address = 0.0.0.0
```

Restart MySQL server to ensure that it starts listening on all interfaces.

```
sudo restart mysql
```

### PostgreSQL

Install PostgreSQL and the python driver for PostgreSQL

```
sudo apt-get install -y postgresql python-psycopg2
```

Setup PostgreSQL to listen on all interfaces by editing `/etc/postgresql/9.1/main/postgresql.conf` and uncommenting and modifying the relevant line. Look for the right file if you have a different version of PostgreSQL

```
listen_addresses = '*'
```

Restart PostgreSQL server to ensure that it starts listening on all interfaces.

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

### SQLite

Install SQLite

```
sudo apt-get install -y sqlite
```

## Glance

Nova can use Glance service to manage Operating System images that it needs for bringing up instances. Glance can use several types of storage backends such as filestore, s3 etc.

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

The default config file at `/etc/glance/glance.conf` is good to use for a simple file store as the storage backend. Glance can be configured to use other storage backends such as Swift.

Glance uses sqlite as the default database backend. While sqlite offers a quick and easy way to get started, for production use, you may consider a database such as MySQL or PostgreSQL.

Glance has two components - `glance-api` and `glance-registry`. These can be controlled using the concerned upstart jobs.

## Database Configuration

Glance uses SQLite by default. MySQL and PostgreSQL can also be configured to work with Glance.

### MySQL

Create a database named glance

```
sudo mysql -uroot -pmygreatsecret -e 'CREATE DATABASE glance;'
```

Create a user named glancedbadmin

```
sudo mysql -uroot -pmygreatsecret -e 'CREATE USER glancedbadmin;'
```

Grant all privileges for glancedbadmin on the Database "glance".

```
sudo mysql -uroot -pmygreatsecret -e "GRANT ALL PRIVILEGES ON glance.* TO 'glancedbadmin'@'%' ;"
```

Create a password for the user "glanceadmin"

```
sudo mysql -uroot -pmygreatsecret -e "SET PASSWORD FOR 'glancedbadmin'@'%' = PASSWORD('glancesecret');"
```

Edit the file `/etc/glance/glance-registry.conf` and edit the line which contains the option `"sql_connection ="` to this:

```
sql_connection = mysql://glancedbadmin:glancesecret@10.10.10.2/glance
```

Restart `glance-registry` after making changes to `/etc/glance/glance-registry.conf`

```
sudo restart glance-registry
```

## PostgreSQL

Create a user called 'glancedbadmin' with password 'glancesecret', create the database 'glance' and give 'glancedbadmin' all privileges on that database.

```
sudo su - postgres
psql
CREATE user glancedbadmin;
ALTER user glancedbadmin with password 'glancesecret';
CREATE DATABASE glance;
GRANT ALL PRIVILEGES ON database glance TO glancedbadmin;
\q
exit
```

Edit the file /etc/glance/glance-registry.conf and edit the line which contains the option "sql\_connection =" to this:

```
sql_connection = postgresql://glancedbadmin:glancesecret@10.10.10.2/glance
```

Restart glance-registry after making changes to /etc/glance/glance-registry.conf

```
sudo restart glance-registry
```

## Nova

Install the messaging queue server, RabbitMQ and various nova components.

```
sudo apt-get install -y rabbitmq-server nova-common nova-doc python-nova nova-  
api nova-network nova-volume nova-objectstore nova-scheduler nova-compute
```

Install euca2ools package for command line tools to interact with nova.

```
sudo apt-get install -y euca2ools
```

Install unzip for extracting archives.

```
sudo apt-get install -y unzip
```

## Database Configuration

Nova can use MySQL or PostgreSQL for storing its data. For non-production use, sqlite is also an option.

### MySQL

Create a database named nova.

```
sudo mysql -uroot -pmygreatsecret -e 'CREATE DATABASE nova;'
```

Create a user named novadbadmin which has access to nova related databases.

```
sudo mysql -uroot -pmygreatsecret -e 'CREATE USER novadbadmin;'
```

Grant all privileges for novadbadmin on the Database "nova".

```
sudo mysql -uroot -pmygreatsecret -e "GRANT ALL PRIVILEGES ON nova.* TO
'novadbadmin'@'%' ;"
```

Set password for novadbadmin.

```
sudo mysql -uroot -pmygreatsecret -e "SET PASSWORD FOR 'novadbadmin'@'%' =
PASSWORD('novasecret');"
```

## PostgreSQL

Create a user called 'novadbadmin' with password 'novasecret', create the database 'nova' and give 'novadbadmin' all privileges on that database.

```
sudo su - postgres
psql
CREATE user novadbadmin;
ALTER user novadbadmin with password 'novasecret';
CREATE DATABASE nova;
GRANT ALL PRIVILEGES ON database nova TO novadbadmin;
\q
exit
```

Configure Postgresql to accept connections from users from any machine on 10.10.10.0/24 and 192.168.3.0/24 by adding the following line to /etc/postgresql/9.1/main/pg\_hba.conf.

```
host    all             all             10.10.10.0/24      md5
host    all             all             192.168.3.0/24    md5
```

Restart PostgreSQL.

```
/etc/init.d/postgresql restart
```

## Nova Configuration

Edit the /etc/nova/nova.conf file to look like this.

```
--dhcpbridge_flagfile=/etc/nova/nova.conf
--dhcpbridge=/usr/bin/nova-dhcpbridge
--logdir=/var/log/nova
--state_path=/var/lib/nova
--lock_path=/var/lock/nova
--state_path=/var/lib/nova
--verbose
--s3_host=10.10.10.2
--rabbit_host=10.10.10.2
--cc_host=10.10.10.2
--nova_url=http://10.10.10.2:8774/v1.1/
--fixed_range=192.168.0.0/16
--network_size=8
--routing_source_ip=10.10.10.2
--sql_connection=mysql://novadbadmin:novasecret@10.10.10.2/nova
--glance_api_servers=192.168.3.2:9292
--image_service=nova.image.glance.GlanceImageService
--iscsi_ip_prefix=192.168.
--vlan_interface=br100
```

```
--public_interface=eth0
```

For configuring with PostgreSQL change the line with `--sql_connection` to:

```
--sql_connection=postgresql://novadbadmin:novasecret@10.10.10.2/nova
```

For configuring with SQLite change the line with `--sql_connection` to:

```
--sql_connection=sqlite:///var/lib/nova/nova.sqlite
```

Install `iscsitarget`

```
sudo apt-get -y install iscsitarget iscsitarget-dkms
```

Enable `iscsitarget`.

```
sudo sed -i 's/false/true/g' /etc/default/iscsitarget
```

Restart the `iscsitarget` service

```
sudo service iscsitarget restart
```

Create a Physical Volume.

```
sudo pvcreate /dev/sda6
```

Create a Volume Group named `nova-volumes`.

```
sudo vgcreate nova-volumes /dev/sda6
```

Change the ownership of the `/etc/nova` folder and permissions for `/etc/nova/nova.conf`:

```
sudo chown -R root:nova /etc/nova
sudo chmod 644 /etc/nova/nova.conf
```

Restart all the nova related services.

```
sudo restart libvirt-bin; sudo restart nova-network; sudo restart nova-
compute; sudo restart nova-api; sudo restart nova-objectstore; sudo restart
nova-scheduler; sudo restart nova-volume; sudo restart glance-api; sudo
restart glance-registry
```

Create nova schema in the MySQL Database.

```
sudo nova-manage db sync
```

Provide a range of IPs to be attached to the instances.

```
sudo nova-manage network create private 192.168.4.0/24 1 256
```

Allocate 32 public IP addresses for use with the instances starting from 10.10.10.225.

```
sudo nova-manage floating create --ip_range=10.10.10.224/27
```

Create a user with admin rights on nova.

```
sudo nova-manage user admin novaadmin
```

Create a project named proj.

```
sudo nova-manage project create proj novaadmin
```

Restart all the nova related services.

```
sudo restart libvirt-bin; sudo restart nova-network; sudo restart nova-  
compute; sudo restart nova-api; sudo restart nova-objectstore; sudo restart  
nova-scheduler; sudo restart nova-volume; sudo restart glance-api; sudo  
restart glance-registry
```

Create a directory to download nova credentials and download the zip file.

```
mkdir /home/localadmin/creds
```

Generate and save credentials for accessing/managing the nova cloud.

```
sudo nova-manage project zipfile proj novaadmin /home/localadmin/creds/  
novacreds.zip
```

Contents of novacreds.zip are required to use euca2ools to manage the cloud infrastructure and you will need to transfer this zip file to any machine from where you want to run the commands from euca2ools. We will be using these credentials from client1 as well.

Navigate in to the folder created and extract the files and change their ownership.

```
cd /home/localadmin/creds  
unzip novacreds.zip  
sudo chown localadmin:localadmin /home/localadmin/creds/ -R
```

Here are the files extracted:

ca-cert.pem, cert.pem, novarc, pk.pem

novarc contains several environmental variables including your nova credentials to be set before you can use the commands from euca2ools such euca-describe-images, euca-describe-instances etc. these variables can be set by sourcing novarc file.

In Diablo, by default novarc file contains EC2\_ACCESS\_KEY in a format that is not usable by euca-\* commands. To fix this:

```
sudo nova-manage user exports novaadmin
```

The output will be something like:

```
export EC2_ACCESS_KEY=c043916c-9a0c-4f91-ad6c-4b30908b6c77  
export EC2_SECRET_KEY=d0ac688e-02f3-48f3-b758-96d886461ace
```

Open the novarc file and replace the line

```
export EC2_ACCESS_KEY="novaadmin:proj"
```

with

```
export EC2_ACCESS_KEY="c043916c-9a0c-4f91-ad6c-4b30908b6c77:proj"
```

```
source /home/localadmin/creds/novarc
```

Check if the credentials are working and if nova has been setup properly by running:

```
euca-describe-availability-zones verbose
```

If you see something like the following with all components happy, it means that the set up is ready to be used.

```
AVAILABILITYZONE  nova available
AVAILABILITYZONE  |- server1
AVAILABILITYZONE  | |- nova-compute      enabled :-) 2011-09-29 07:26:04
AVAILABILITYZONE  | |- nova-scheduler   enabled :-) 2011-09-29 07:26:04
AVAILABILITYZONE  | |- nova-network     enabled :-) 2011-09-29 07:26:07
AVAILABILITYZONE  | |- nova-volume     enabled :-) 2011-09-29 07:26:06
```

## Swift

Install Swift via apt-get:

```
sudo apt-get install swift swift-proxy memcached swift-account swift-container
swift-object
```

Install Swift via apt-get:

```
sudo apt-get install xfsprogs curl
```

## Swift Configuration

There are two methods to configure the storage backend for use by Swift.

### Physical Device (Partition) as a storage

A physical device is partitioned and used as storage. Assuming there is a secondary disk / dev/sdb :

```
sudo fdisk /dev/sdb
```

Press m for help; n for new partition; p to view the partition table; w to write changes to disk once you are done. You should now have a partition /dev/sdb1.

### Loopback Device (File) as storage

We create a zero filled file for use as a loopback device for the Swift storage backend. Here we use the disk copy command to create a file named swift-disk and allocate a million 1KiB blocks (976.56 MiB) to it. So we have a loopback disk of approximately 1GiB. We can increase this size by modifying the seek value.

```
sudo dd if=/dev/zero of=/srv/swift-disk bs=1024 count=0 seek=1000000
```

We now create an xfs filesystem out of the partition or the loopback device we just created. For the loopback file, doing "file swift-disk" will give the details of the device. For the partition details, tune2fs utility can be used with "l" parameter.

For the physical partition:

```
sudo mkfs.xfs -i size=1024 /dev/sdb1
sudo tune2fs -l /dev/sdb1 |grep -i inode
```

For the loopback file:

```
sudo mkfs.xfs -i size=1024 /srv/swift-disk
file /srv/swift-disk
swift-disk1: SGI XFS filesystem data (blksz 4096, inosz 1024, v2 dirs)
```

The storage device we created has to be mounted automatically everytime the system starts. Lets create an arbitrary mountpoint /mnt/sdb1.

```
sudo mkdir /mnt/sdb1
```

Edit /etc/fstab and append the following line:

For the physical partiton

```
/dev/sdb1 /mnt/sdb1 xfs noatime,nodiratime,nobarrier,logbufs=8 0 0
```

If you have a loopback file

```
/srv/swift-disk /mnt/sdb1 xfs loop,noatime,nodiratime,nobarrier,logbufs=8 0 0
```

Let's now mount the storage device and create directories (which would act as storage nodes) and provide appropriate permissions and ownerships (user:group format) for them. I have set the ownership to swift:swift for all relevant files.

```
sudo mount /mnt/sdb1
sudo mkdir /mnt/sdb1/1 /mnt/sdb1/2 /mnt/sdb1/3 /mnt/sdb1/4
sudo chown swift:swift /mnt/sdb1/*
sudo ln -s /mnt/sdb1/1 /srv/1
sudo ln -s /mnt/sdb1/2 /srv/2
sudo ln -s /mnt/sdb1/3 /srv/3
sudo ln -s /mnt/sdb1/4 /srv/4
sudo mkdir -p /etc/swift/object-server /etc/swift/container-server /etc/swift/
account-server /srv/1/node/sdb1 /srv/2/node/sdb2 /srv/3/node/sdb3 /srv/4/node/
sdb4 /var/run/swift
sudo chown -R swift:swift /etc/swift /srv/[1-4]/
```

Append the following lines to /etc/rc.local just before the "exit 0":

```
mkdir /var/run/swift
chown swift:swift /var/run/swift
```

## Configuring Rsync

Rsync is responsible for maintaining object replicas. It is used by various swift services to maintain consistency of objects and perform updation operations. It is configured for all the storage nodes.

Create `/etc/rsyncd.conf` file and add the following lines to it:

```
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = 127.0.0.1

[account6012]
max connections = 25
path = /srv/1/node/
read only = false
lock file = /var/lock/account6012.lock

[account6022]
max connections = 25
path = /srv/2/node/
read only = false
lock file = /var/lock/account6022.lock

[account6032]
max connections = 25
path = /srv/3/node/
read only = false
lock file = /var/lock/account6032.lock

[account6042]
max connections = 25
path = /srv/4/node/
read only = false
lock file = /var/lock/account6042.lock

[container6011]
max connections = 25
path = /srv/1/node/
read only = false
lock file = /var/lock/container6011.lock

[container6021]
max connections = 25
path = /srv/2/node/
read only = false
lock file = /var/lock/container6021.lock

[container6031]
max connections = 25
path = /srv/3/node/
read only = false
lock file = /var/lock/container6031.lock

[container6041]
```

```
max connections = 25
path = /srv/4/node/
read only = false
lock file = /var/lock/container6041.lock

[object6010]
max connections = 25
path = /srv/1/node/
read only = false
lock file = /var/lock/object6010.lock

[object6020]
max connections = 25
path = /srv/2/node/
read only = false
lock file = /var/lock/object6020.lock

[object6030]
max connections = 25
path = /srv/3/node/
read only = false
lock file = /var/lock/object6030.lock

[object6040]
max connections = 25
path = /srv/4/node/
read only = false
lock file = /var/lock/object6040.lock
```

Set `RSYNC_ENABLE=true` in `/etc/default/rsync` and then restart the `rsync` service.

```
sudo service rsync restart
```

## Swift Configuration

Create and edit `/etc/swift/swift.conf` and add the following lines to it:

```
[swift-hash]
# random unique (preferably alphanumeric) string that can never change (DO NOT
# LOSE)
swift_hash_path_suffix = {place_random_unique_string_here}
```

You will need the random string when you add more nodes to the setup. So never lose the string.

You can generate a random string by running the following command:

```
od -t x8 -N 8 -A n < /dev/random
```

## Proxy Server Configuration

Create and edit `/etc/swift/proxy-server.conf` and add the following lines to the file:

```
[DEFAULT]
```

```
bind_port = 8080
user = swift
log_facility = LOG_LOCAL1

[pipeline:main]
pipeline = healthcheck cache tempauth proxy-server

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

[filter:tempauth]
use = egg:swift#tempauth
user_admin_admin = admin .admin .reseller_admin
user_test_tester = testing .admin

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

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

Note: You can find sample configuration files at the "etc" directory in the source. If you used apt-get to install swift, the configuration samples can be found at "/usr/share/doc/swift/"

## Account Server Configuration

Create an account server configuration file for node #1: /etc/swift/account-server/1.conf and add the following lines to the file

```
[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]
```

We need to create the configuration for the rest of the three virtual nodes (/srv/2/node, /srv/3/node, /srv/4/node) as well. So we simply make three more copies of 1.conf and set unique bind ports for the rest of the nodes (6022, 6032 and 6042) and different local log values (LOG\_LOCAL3, LOG\_LOCAL4, LOG\_LOCAL5).

```
sudo cp /etc/swift/account-server/1.conf /etc/swift/account-server/2.conf
sudo cp /etc/swift/account-server/1.conf /etc/swift/account-server/3.conf
sudo cp /etc/swift/account-server/1.conf /etc/swift/account-server/4.conf
```

Now we need to edit the files 2.conf, etc. created by the above command and edit the bind port number and local log values.

## Container Server Configuration

Create a container server configuration file for node #1: `/etc/swift/container-server/1.conf` and add the following lines to the file

```
[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]
```

We need to create the configuration for the rest of the three virtual nodes (`/srv/2/node`, `/srv/3/node`, `/srv/4/node`) as well. So we simply make three more copies of 1.conf and set unique bind ports for the rest of the nodes (6021, 6031 and 6041) and different local log values (LOG\_LOCAL3, LOG\_LOCAL4, LOG\_LOCAL5).

```
sudo cp /etc/swift/container-server/1.conf /etc/swift/container-server/2.conf
sudo cp /etc/swift/container-server/1.conf /etc/swift/container-server/3.conf
sudo cp /etc/swift/container-server/1.conf /etc/swift/container-server/4.conf
```

Now we need to edit the files 2.conf, etc. created by the above command and edit the bind port number and local log values.

## Object Server Configuration

Create an object server configuration file for node #1: `/etc/swift/object-server/1.conf` and add the following lines to the file

```
[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]
```

We need to create the configuration for the rest of the three virtual nodes (`/srv/2/node`, `/srv/3/node`, `/srv/4/node`) as well. So we simply make three more copies of `1.conf` and set unique bind ports for the rest of the nodes (6020, 6030 and 6040) and different local log values (`LOG_LOCAL3`, `LOG_LOCAL4`, `LOG_LOCAL5`).

```
sudo cp /etc/swift/object-server/1.conf /etc/swift/object-server/2.conf
sudo cp /etc/swift/object-server/1.conf /etc/swift/object-server/3.conf
sudo cp /etc/swift/object-server/1.conf /etc/swift/object-server/4.conf
```

Now we need to edit the files `2.conf`, etc. created by the above command and edit the bind port number and local log values.

## Building the rings

Rings are important components of Swift. Run the following commands in the same order for building the rings:

```
pushd /etc/swift
sudo swift-ring-builder object.builder create 18 3 1
sudo swift-ring-builder object.builder add z1-127.0.0.1:6010/sdb1 1
sudo swift-ring-builder object.builder add z2-127.0.0.1:6020/sdb2 1
sudo swift-ring-builder object.builder add z3-127.0.0.1:6030/sdb3 1
sudo swift-ring-builder object.builder add z4-127.0.0.1:6040/sdb4 1
sudo swift-ring-builder object.builder rebalance
sudo swift-ring-builder container.builder create 18 3 1
sudo swift-ring-builder container.builder add z1-127.0.0.1:6011/sdb1 1
sudo swift-ring-builder container.builder add z2-127.0.0.1:6021/sdb2 1
sudo swift-ring-builder container.builder add z3-127.0.0.1:6031/sdb3 1
sudo swift-ring-builder container.builder add z4-127.0.0.1:6041/sdb4 1
sudo swift-ring-builder container.builder rebalance
sudo swift-ring-builder account.builder create 18 3 1
sudo swift-ring-builder account.builder add z1-127.0.0.1:6012/sdb1 1
sudo swift-ring-builder account.builder add z2-127.0.0.1:6022/sdb2 1
sudo swift-ring-builder account.builder add z3-127.0.0.1:6032/sdb3 1
sudo swift-ring-builder account.builder add z4-127.0.0.1:6042/sdb4 1
sudo swift-ring-builder account.builder rebalance
```

## Running\_Swift\_Services

```
sudo swift-init main start
```

To start the REST API do the following:

```
sudo swift-init rest start
```

## Testing Swift

As normal user, execute the following command

```
curl -v -H 'X-Storage-User: admin:admin' -H 'X-Storage-Pass: admin' http://127.0.0.1:8080/auth/v1.0
```

The output should look like:

```
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1... connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /auth/v1.0 HTTP/1.1
> User-Agent: curl/7.21.6 (i686-pc-linux-gnu) libcurl/7.21.6 OpenSSL/1.0.0e
zlib/1.2.3.4 libidn/1.22 librtmp/2.3
> Host: 127.0.0.1:8080
> Accept: */*
> X-Storage-User: admin:admin
> X-Storage-Pass: admin
>
< HTTP/1.1 200 OK
< X-Storage-Url: http://127.0.0.1:8080/v1/AUTH_admin
< X-Storage-Token: AUTH_tk441617bc550f4bb7bf51b4dc16800900
< X-Auth-Token: AUTH_tk441617bc550f4bb7bf51b4dc16800900
< Content-Length: 0
< Date: Fri, 14 Oct 2011 15:25:15 GMT
<
* Connection #0 to host 127.0.0.1 left intact
* Closing connection #0
```

If the above command did not throw any error and gave you an output similar to the one above, then the user accounts are being authenticated fine. Make note of the X-Auth-Token and the X-Storage-Url. Then give the following command to check if you can perform a GET operation successfully:

```
curl -v -H 'X-Auth-Token: AUTH_tk441617bc550f4bb7bf51b4dc16800900' http://127.0.0.1:8080/v1/AUTH_admin
```

The output should be similar as follows:

```
* About to connect() to 127.0.0.1 port 8080 (#0)
*   Trying 127.0.0.1... connected
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /v1/AUTH_admin HTTP/1.1
> User-Agent: curl/7.21.6 (i686-pc-linux-gnu) libcurl/7.21.6 OpenSSL/1.0.0e
zlib/1.2.3.4 libidn/1.22 librtmp/2.3
> Host: 127.0.0.1:8080
> Accept: */*
```

```
> X-Auth-Token: AUTH_tk441617bc550f4bb7bf51b4dc16800900
>
< HTTP/1.1 204 No Content
< X-Account-Object-Count: 0
< X-Account-Bytes-Used: 0
< X-Account-Container-Count: 0
< Accept-Ranges: bytes
< Content-Length: 0
< Date: Fri, 14 Oct 2011 15:28:15 GMT
<
* Connection #0 to host 127.0.0.1 left intact
* Closing connection #0
```

## Server2

This server runs nova-compute and all the virtual machines and hypervisor. You can also bundle images on Server2.

## BaseOS

Install 64-bit version of Oneiric Server

## Networking Configuration

Install bridge-utils:

```
sudo apt-get install -y bridge-utils
```

Edit the `/etc/network/interfaces` file so as to look like this:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
    address 10.10.10.3
    netmask 255.255.255.0
    broadcast 10.10.10.255
    gateway 10.10.10.1
    dns-nameservers 10.10.10.3

auto eth1
iface eth1 inet static
    address 192.168.3.2
    netmask 255.255.255.0
    network 192.168.3.0
    broadcast 192.168.3.255
```

Restart the network now

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

## NTP Client

Install NTP package.

```
sudo apt-get install -y ntp
```

Open the file `/etc/ntp.conf` and add the following line to sync to server1.

```
server 10.10.10.2
```

Restart NTP service to make the changes effective

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

## Nova Components (nova-compute alone)

Install the nova-components and dependencies.

```
sudo apt-get install -y nova-common python-nova nova-compute vlan
```

Install euca tools, for command line tools

```
sudo apt-get install -y euca2ools
```

Install unzip for extracting archives

```
sudo apt-get install -y unzip
```

Edit the `/etc/nova/nova.conf` file to look like this. This file is essentially similar to the configuration file (`/etc/nova/nova.conf`) of Server1

```
--dhcpbridge_flagfile=/etc/nova/nova.conf
--dhcpbridge=/usr/bin/nova-dhcpbridge
--logdir=/var/log/nova
--state_path=/var/lib/nova
--lock_path=/var/lock/nova
--state_path=/var/lib/nova
--verbose
--s3_host=10.10.10.2
--rabbit_host=10.10.10.2
--cc_host=10.10.10.2
--network_size=8
--routing_source_ip=10.10.10.2
--sql_connection=mysql://novadbadmin:novasecret@10.10.10.2/nova
--glance_api_servers=192.168.3.1:9292
--image_service=nova.image.glance.GlanceImageService
--iscsi_ip_prefix=192.168.
--vlan_interface=br100
```

Restart nova-compute on Server2.

```
sudo service restart nova-compute
```

On Server1, check if the second compute node (Server2) is detected by running:

```
euca-describe-availability-zones verbose
```

If you see something like the following with all components happy, it means that the set up is ready to be used.

```
AVAILABILITYZONE nova available
AVAILABILITYZONE | - server1
AVAILABILITYZONE | | - nova-compute enabled :-) 2011-09-30 09:26:04
AVAILABILITYZONE | | - nova-scheduler enabled :-) 2011-09-30 09:26:04
AVAILABILITYZONE | | - nova-network enabled :-) 2011-09-30 09:26:07
AVAILABILITYZONE | | - nova-volume enabled :-) 2011-09-30 09:26:06
AVAILABILITYZONE | - server2
AVAILABILITYZONE | | - nova-compute enabled :-) 2011-09-30 09:26:05
```

Sometimes you may have XXX instead of the smiley. The XXX are displayed when the components are not time-synced. Ensure time synchronization across all components of your OpenStack deployment.

## Client1

### BaseOS

Install 64-bit version of Oneiric Desktop

## Networking Configuration

Edit the `/etc/network/interfaces` file so as to look like this:

```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
address 10.10.10.4
netmask 255.255.255.0
broadcast 10.10.10.255
gateway 10.10.10.1
dns-nameservers 10.10.10.3
```

## NTP Client

Install NTP package.

```
sudo apt-get install -y ntp
```

Open the file `/etc/ntp.conf` and add the following line to sync to server1.

```
server 10.10.10.2
```

Restart NTP service to make the changes effective

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

## Client Tools

As mentioned above, this is a desktop installation of Oneiric to be used for tasks such as bundling of images. It will also be used for managing the cloud infrastructure using euca2ools.

### Install euca tools, for command line tools

```
sudo apt-get install -y euca2ools
```

### Install qemu-kvm

```
sudo apt-get install -y qemu-kvm
```

### Download the credentials we created for localadmin to this machine:

```
mkdir /home/localadmin/creds  
cd /home/localadmin/creds  
scp localadmin@10.10.10.2:/home/localadmin/creds/novacreds.zip .  
unzip creds.zip
```

### Source novarc file and see if connectivity to api server is working correctly:

```
source novarc  
euca-describe-availability-zones verbose
```

The output should be similar to what is shown above in the configuration section for Server1.

Note: If you want to avoid manually sourcing the novarc file every time, the user can add the following line to the .profile file in his home directory:

```
source /home/localadmin/creds/novarc
```

## 3. Image Management

### Table of Contents

Introduction .....	28
Creating a Linux Image - Ubuntu & Fedora .....	29
OS Installation .....	29
Extracting the EXT4 partition .....	30
Fetching Metadata in Fedora .....	32
Uploading to OpenStack .....	32
Image Listing .....	33
Creating a Windows Image .....	33

### Introduction

There are several pre-built images for OpenStack available from various sources. You can download such images and use them to get familiar with OpenStack. You can refer to <http://docs.openstack.org/cactus/openstack-compute/admin/content/starting-images.html> for details on using such images.

For any production deployment, you may like to have the ability to bundle custom images, with a custom set of applications or configuration. This chapter will guide you through the process of creating Linux images of Debian and RedHat based distributions from scratch. We have also covered an approach to bundling Windows images.

There are some minor differences in the way you would bundle a Linux image, based on the distribution. Ubuntu makes it very easy by providing cloud-init package, which can be used to take care of the instance configuration at the time of launch. cloud-init handles importing ssh keys for password-less login, setting host name etc. The instance acquires the instance specific configuration from Nova-compute by connecting to a meta data interface running on 169.254.169.254.

While creating the image of a distro that does not have cloud-init or an equivalent package, you may need to take care of importing the keys etc. by running a set of commands at boot time from rc.local.

The process used for Ubuntu and Fedora is largely the same with a few minor differences, which are explained below.

In both cases, the documentation below assumes that you have a working KVM installation to use for creating the images. We are using the machine called 'client1' as explained in the chapter on "Installation and Configuration" for this purpose.

The approach explained below will give you disk images that represent a disk without any partitions. Nova-compute can resize such disks ( including resizing the file system) based on the instance type chosen at the time of launching the instance. These images cannot have 'bootable' flag and hence it is mandatory to have associated kernel and ramdisk images. These kernel and ramdisk images need to be used by nova-compute at the time of launching the instance.

However, we have also added a small section towards the end of the chapter about creating bootable images with multiple partitions that can be used by nova to launch an instance without the need for kernel and ramdisk images. The caveat is that while nova-compute can re-size such disks at the time of launching the instance, the file system size is not altered and hence, for all practical purposes, such disks are not re-sizable.

## Creating a Linux Image - Ubuntu & Fedora

The first step would be to create a raw image on Client1. This will represent the main HDD of the virtual machine, so make sure to give it as much space as you will need.

```
kvm-img create -f raw server.img 5G
```

## OS Installation

Download the iso file of the Linux distribution you want installed in the image. The instructions below are tested on Ubuntu 11.10 Oneiric Ocelot 64-bit server and Fedora 14 64-bit. Most of the instructions refer to Ubuntu. The points of difference between Ubuntu and Fedora are mentioned wherever required.

```
wget http://releases.ubuntu.com/oneiric/ubuntu-11.10-server-amd64.iso
```

Boot a KVM instance with the OS installer ISO in the virtual CD-ROM. This will start the installation process. The command below also sets up a VNC display at port 0

```
sudo kvm -m 256 -cdrom ubuntu-11.10-server-amd64.iso -drive file=server.img,  
if=scsi,index=0 -boot d -net nic -net user -nographic -vnc :0
```

Connect to the VM through VNC (use display number :0) and finish the installation.

For Example, where 10.10.10.4 is the IP address of client1:

```
vncviewer 10.10.10.4 :0
```

During the installation of Ubuntu, create a single ext4 partition mounted on '/'. Do not create a swap partition.

In the case of Fedora 14, the installation will not progress unless you create a swap partition. Please go ahead and create a swap partition.

After finishing the installation, relaunch the VM by executing the following command.

```
sudo kvm -m 256 -drive file=server.img,if=scsi,index=0,boot=on -boot c -net  
nic -net user -nographic -vnc :0
```

At this point, you can add all the packages you want to have installed, update the installation, add users and make any configuration changes you want in your image.

At the minimum, for Ubuntu you may run the following commands

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

```
sudo apt-get install openssh-server cloud-init
```

For Fedora run the following commands as root

```
yum update
```

```
yum install openssh-server
```

```
chkconfig sshd on
```

## Tweaking /etc/fstab

You will need to tweak /etc/fstab to make it suitable for a cloud instance. Nova-compute may resize the disk at the time of launching instances based on the instance type chosen. This can make the UUID of the disk invalid. Hence we have to use file system label as the identifier for the partition instead of the UUID. Edit /etc/fstab and change the following line from

```
UUID=e7f5af8d-5d96-45cc-a0fc-d0d1bde8f31c / ext4 errors=remount-ro 0 1
```

to

```
LABEL=uec-rootfs / ext4 defaults 0 0
```

Also remove the network persistence rules from /etc/udev/rules.d as their presence will result in the network interface in the instance coming up as an interface other than eth0.

```
sudo rm -rf /etc/udev/rules.d/70-persistent-net.rules
```

Shutdown the virtual machine and proceed with the next steps.

## Extracting the EXT4 partition

The image that needs to be uploaded to OpenStack needs to be an ext4 filesystem image. Here are the steps to create a ext4 filesystem image from the raw image i.e server.img

```
sudo losetup -f server.img
```

```
sudo losetup -a
```

You should see an output like this:

```
/dev/loop0: [0801]:16908388 ($filepath)
```

Observe the name of the loop device ( /dev/loop0 in our setup) when \$filepath is the path to the mounted .raw file.

Now we need to find out the starting sector of the partition. Run:

```
sudo fdisk -cul /dev/loop0
```

You should see an output like this:

```
Disk /dev/loop0: 5368 MB, 5368709120 bytes
```

```
149 heads, 8 sectors/track, 8796 cylinders, total 10485760 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00072bd4
Device          Boot  Start      End          Blocks      Id
System
/dev/loop0p1    *           2048    10483711    5240832    83    Linux
```

Make a note of the starting sector of the /dev/loop0p1 partition i.e the partition whose ID is 83. This number should be multiplied by 512 to obtain the correct value. In this case:  $2048 \times 512 = 1048576$

Unmount the loop0 device:

```
sudo losetup -d /dev/loop0
```

Now mount only the partition(/dev/loop0p1) of server.img which we had previously noted down, by adding the -o parameter with value previously calculated value

```
sudo losetup -f -o 1048576 server.img
```

```
sudo losetup -a
```

You'll see a message like this:

```
/dev/loop0: [0801]:16908388 ($filepath) offset 1048576
```

Make a note of the mount point of our device(/dev/loop0 in our setup) when \$filepath is the path to the mounted .raw file.

Copy the entire partition to a new .raw file

```
sudo dd if=/dev/loop0 of=serverfinal.img
```

Now we have our ext4 filesystem image i.e serverfinal.img

Unmount the loop0 device

```
sudo losetup -d /dev/loop0
```

## Fetching Metadata in Fedora

Since, Fedora does not ship with cloud-init or an equivalent, you will need to take a few steps to have the instance fetch the meta data like ssh keys etc.

Edit the `/etc/rc.local` file and add the following lines before the line `"touch /var/lock/subsys/local"`

```
depmod -a
modprobe acpihp
# simple attempt to get the user ssh key using the meta-data service
mkdir -p /root/.ssh
echo >> /root/.ssh/authorized_keys
curl -m 10 -s http://169.254.169.254/latest/meta-data/public-keys/0/openssh-key| grep 'ssh-rsa' >> /root/.ssh/authorized_keys
echo "AUTHORIZED_KEYS:"
echo "*****"
cat /root/.ssh/authorized_keys
echo "*****"
```

Unmount the Loop partition

```
sudo umount /mnt
```

Change the filesystem label of `serverfinal.img` to `'uec-rootfs'`

```
sudo tune2fs -L uec-rootfs serverfinal.img
```

Now, we have all the components of the image ready to be uploaded to OpenStack imaging server.

## Uploading to OpenStack

The last step would be to upload the images to OpenStack Imaging Server glance. The files that need to be uploaded for the above sample setup of Ubuntu are: `vmlinuz-2.6.38-7-server`, `initrd.img-2.6.38-7-server`, `serverfinal.img`

Run the following command

```
uec-publish-image amd64 serverfinal.img bucket1
```

For Fedora, the process will be similar. Make sure that you use the right kernel and `initrd` files extracted above.

`uec-publish-image`, like several other commands from `euca2ools`, returns the prompt back immediately. However, the upload process takes some time and the images will be usable only after the process is complete. You can keep checking the status using the command `'euca-describe-images'` as mentioned below.

You can upload bootable disk images without associating kernel and ramdisk images. When you do not want the flexibility of using the same disk image with different kernel/

ramdisk images, you can go for bootable disk images. This greatly simplifies the process of bundling and uploading the images. However, the caveats mentioned in the introduction to this chapter apply. Please note that the instructions below use `server.img` and you can skip all the cumbersome steps related to extracting the single ext4 partition.

```
euca-bundle-image -i server.img
```

```
euca-upload-bundle -b mybucket -m /tmp/server.img.manifest.xml
```

```
euca-register mybucket/server.img.manifest.xml
```

## Image Listing

The status of the images that have been uploaded can be viewed by using `euca-describe-images` command. The output should like this:

```
localadmin@client1:~$ euca-describe-images
```

```
IMAGE ami-00000003 mybucket9/Fedora14Nova.img.manifest.xml available private  
x86_64 machine instance-store
```

## Creating a Windows Image

The first step would be to create a raw image on Client1, this will represent the main HDD of the virtual machine, so make sure to give it as much space as you will need.

```
kvm-img create -f raw windowsserver.img 20G
```

OpenStack presents the disk using a VIRTIO interface while launching the instance. Hence the OS needs to have drivers for VIRTIO. By default, the Windows Server 2008 ISO does not have the drivers for VIRTIO. Download the virtual floppy drive containing VIRTIO drivers from the following location

<http://alt.fedoraproject.org/pub/alt/virtio-win/latest/images/bin/>

and attach it during the installation

Start the installation by running

```
sudo kvm -m 1024 -cdrom win2k8_dvd.iso -drive file=windowsserver.img,if=virtio,boot=on -fda virtio-win-1.1.16.vfd -boot d -nographic -vnc :0
```

When the installation prompts you to choose a hard disk device you won't see any devices available. Click on "Load drivers" at the bottom left and load the drivers from A: `\i386\Win2008`

After the Installation is over, boot into it once and install any additional applications you need to install and make any configuration changes you need to make. Also ensure that RDP is enabled as that would be the only way you can connect to a running instance of

Windows. Windows firewall needs to be configured to allow incoming ICMP and RDP connections.

For OpenStack to allow incoming RDP Connections, use euca-authorize command to open up port 3389 as described in the chapter on "Security".

Shut-down the VM and upload the image to OpenStack

```
euca-bundle-image -i windowsserver.img
```

```
euca-upload-bundle -b mybucket -m /tmp/windowsserver.img.manifest.xml
```

```
euca-register mybucket/windowsserver.img.manifest.xml
```

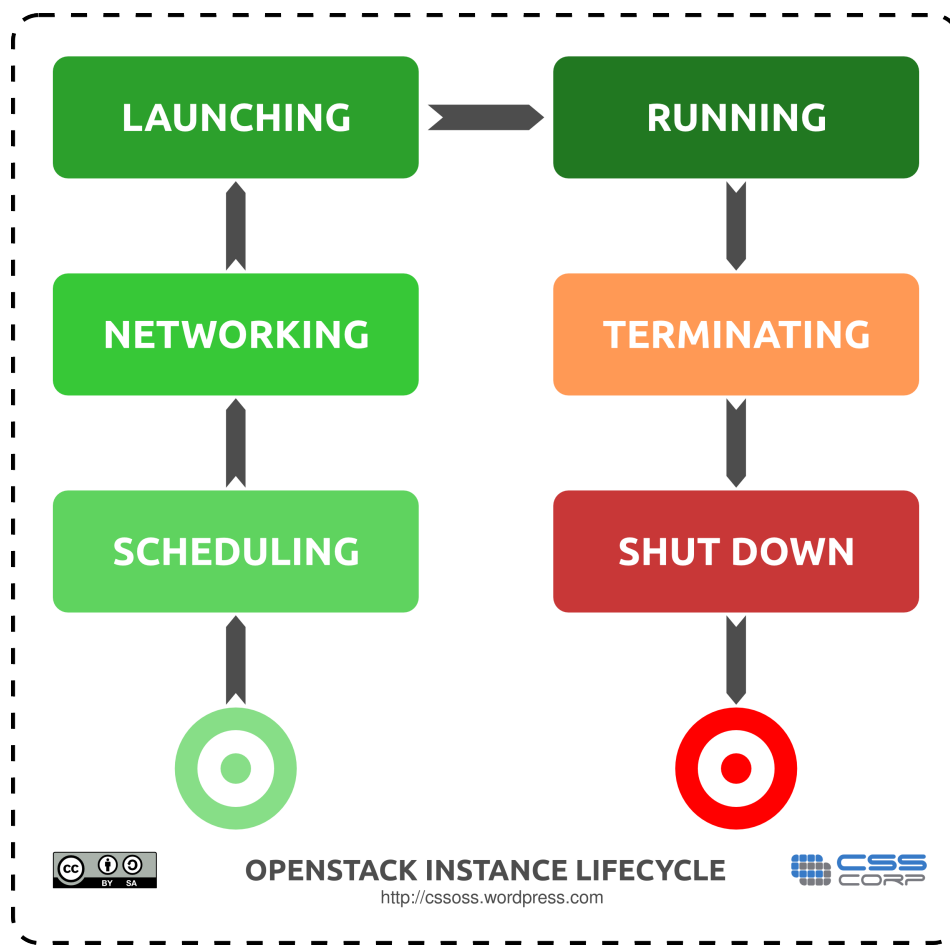
# 4. Instance Management

## Table of Contents

Introduction .....	35
Euca2ools-Command Line Tools .....	36
Installation .....	36
Creation of Key Pairs .....	36
Launch and manage instances .....	37

## Introduction

An instance is a virtual machine provisioned by OpenStack on one of the nova-compute servers. When you launch an instance, a series of steps are triggered on various components of the OpenStack. During the life cycles of an instance, it moves through various stages as shown in the diagram below:



The following interfaces can be used for managing instances in nova.

- Command line tools like euca2ools
- Custom applications developed using EC2 APIs

## Euca2ools-Command Line Tools

Euca2ools from Eucalyptus provide a bunch of command line tools to manage the eucalyptus setup. These commands help you manage images, instances, storage, networking etc. A few commands related to managing the instances are given below.

For a complete list of commands, see Appendix.

## Installation

```
sudo apt-get install euca2ools
```

## Creation of Key Pairs

OpenStack expects the client tools to use 2 kinds of credentials. One set of credentials are called Access Key and Secret Key that all clients would need to use to make any requests to the Cloud Controller. Each user registered on the web interface has this set created for him. You can download it from the web interface as mentioned in the chapter on "Web Interface".

You will also need to generate a keypair consisting of private key/public key to be able to launch instances on Eucalyptus. These keys are injected into the instances to make password-less SSH access to the instance possible. This depends on the way the necessary tools are bundled into the images. Please refer to the chapter on Image Management for more details.

Keypairs can also be generated using the following commands.

```
cd ~/creds
euca-add-keypair mykey > mykey.priv
chmod 600 mykey.priv
```

This creates a new keypair called mykey. The private key mykey.priv is saved locally which can be used to connect to an instance launched with mykey as the keypair. euca-describe-keypairs command to list the available keypairs.

The output should like this:

```
uecadmin@client1:~$ euca-describe-keypairs
KEYPAIR mykey f7:ac:8e:f5:05:19:2b:31:28:8c:9b:d7:b8:07:0c:3c:b6:34:8f:79
KEYPAIR helloworld
12:96:b3:21:34:8d:6a:3f:92:2e:2b:70:23:ff:7f:51:b5:b7:ad:37
KEYPAIR ubuntu f6:af:9a:59:65:35:32:c4:3a:c4:62:0e:e1:44:0f:71:29:03:2d:91
KEYPAIR lucid 74:04:70:33:ed:57:7a:30:36:1f:ca:c6:ec:d5:4f:10:34:1a:52:51
KEYPAIR karmic 01:f9:aa:5f:4d:20:e2:53:d1:29:d0:0f:e2:f3:8c:21:91:70:7e:c8
```

To delete an existing keypair:

```
euca-delete-keypair helloworld
```

The above tasks can be achieved using Hybridfox from the "Keypairs" tab.

## Launch and manage instances

There are several commands that help in managing the instances. Here are a few examples:

```
$ euca-run-instances ami-00000003 -k mykey -t ml.tiny

RESERVATION r-fleklojb proj default
INSTANCE i-00000005 ami-00000003 pending mykey (proj, None) 0 ml.tiny
2011-10-19T12:48:04Z unknown zone ami-00000000 ami-00000000
```

```
$ euca-describe-instances

RESERVATION r-fleklojb proj default
INSTANCE i-00000005 ami-00000003 192.168.4.3 192.168.4.3 running mykey (proj,
Openstackvalidationserver1) 0 ml.tiny 2011-10-19T12:48:04Z nova ami-00000000
ami-00000000
```

```
$ euca-reboot-instances i-00000005
```

```
$ euca-terminate-instances i-00000005
```

```
$ euca-run-instances ami-XXXXXXXX -k mykey
```

```
$ euca-get-console-output i-00000005

i-00000005
2011-10-07T07:22:40.795Z
[ 0.000000] Initializing cgroup subsys cpuset
[ 0.000000] Initializing cgroup subsys cpu
[ 0.000000] Linux version 2.6.32-21-server (buildd@yellow) (gcc version 4.
4.3 (Ubuntu 4.4.3-4ubuntu5) ) #32-Ubuntu SMP Fri Oct 07 09:17:34 UTC 2011 (Ub
untu 2.6.32-21.32-server 2.6.32.11+drm33.2)
.....
```

You can make password less ssh access to the instance as follows:

```
ssh -i mykey.priv user@192.168.4.3
```

VM type has implications for harddisk size, amount of RAM and number of CPUs allocated to the instance. Check the VM types available.

```
sudo nova-manage instance_type list
```

## 5. Storage Management

### Table of Contents

Nova-volume .....	38
Interacting with Storage Controller .....	38
Swift .....	39

### Nova-volume

Nova-volume provides persistent block storage compatible with Amazon's Elastic Block Store. The storage on the instances is non persistent in nature and hence any data that you generate and store on the file system on the first disk of the instance gets lost when the instance is terminated. You will need to use persistent volumes provided by nova-volume if you want any data generated during the life of the instance to persist after the instance is terminated.

Commands from euca2ools package can be used to manage these volumes.

Here are a few examples:

### Interacting with Storage Controller

Make sure that you have sourced novarc before running any of the following commands. The following commands refer to a zone called 'nova', which we created in the chapter on "Installation and Configuration". The project is 'proj' as referred to in the other chapters.

Create a 10 GB volume

```
euca-create-volume -s 10 -z nova
```

You should see an output like:

```
VOLUME    vol-00000002    1    creating (proj, None, None, None)
2011-04-21T07:19:52Z
```

List the volumes

```
euca-describe-volumes
```

You should see an output like this:

```
VOLUME    vol-00000001    1    nova    available (proj, server1, None,
None)    2011-04-21T05:11:22Z
```

```
VOLUME    vol-00000002    1    nova    available (proj, server1, None,
None)    2011-04-21T07:19:52Z
```

Attach a volume to a running instance

```
euca-attach-volume -i i-00000009 -d /dev/vdb vol-00000002
```

A volume can only be attached to one instance at a time. When `euca-describe-volumes` shows the status of a volume as 'available', it means it is not attached to any instance and ready to be used. If you run `euca-describe-volumes`, you can see that the status changes from "available" to "in-use" if it is attached to an instance successfully.

When a volume is attached to an instance, it shows up as an additional SCSI disk on the instance. You can login to the instance and mount the disk, format it and use it.

Detach a volume from an instance.

```
euca-detach-volume vol-00000002
```

The data on the volume persists even after the volume is detached from an instance. You can see the data on reattaching the volume to another instance.

Even though you have indicated `/dev/vdb` as the device on the instance, the actual device name created by the OS running inside the instance may differ. You can find the name of the device by looking at the device nodes in `/dev` or by watching the syslog when the volume is being attached.

## Swift

Swift is a storage service that can be used for storage and archival of objects. Swift provides a REST interface. You can use 'curl' command to get familiar with the service. All requests to Swift service need an authentication token. This authentication token can be obtained by providing your user credentials on Swift. For more details refer to the Swift section in the "Installation & Configuration" chapter.

Execute the following command and make a note of X-Auth-Token. You will need this token to use in all subsequent commands.

```
curl -v -H 'X-Storage-User: admin:admin' -H 'X-Storage-Pass: admin' http://10.10.10.2:8080/auth/v1.0
```

In the following command examples we are using 'AUTH\_tk3bb59eda987446c79160202d4dfbdc8c' as the X-Auth-Token. Replace this with the appropriate token you obtained in the above step.

To create a container:

```
curl -X PUT -H 'X-Auth-Token: AUTH_tk3bb59eda987446c79160202d4dfbdc8c' http://10.10.10.2:8080/v1/AUTH_admin/mycontainer
```

To list all containers in current account:

```
curl -H 'X-Auth-Token: AUTH_tk3bb59eda987446c79160202d4dfbdc8c' http://10.10.10.2:8080/v1/AUTH_admin/
```

Uploading a file "file1" to container "mycontainer"

```
curl -X PUT -T file1 -H 'X-Auth-Token: AUTH_tk3bb59eda987446c79160202d4dfbdc8c' http://10.10.10.2:8080/v1/AUTH_admin/mycontainer/
```

To list all objects in a container:

```
curl -H 'X-Auth-Token: AUTH_tk3bb59eda987446c79160202d4dfbdc8c' http://10.10.10.2:8080/v1/AUTH_admin/mycontainer
```

To list all objects in a container that starts with a particular prefix "fi":

```
curl -H 'X-Auth-Token: AUTH_tk3bb59eda987446c79160202d4dfbdc8c' http://10.10.10.2:8080/v1/AUTH_admin/mycontainer/?prefix=fi
```

To download a particular file "file1" from the container "mycontainer":

```
curl -H 'X-Auth-Token: AUTH_tk3bb59eda987446c79160202d4dfbdc8c' http://10.10.10.2:8080/v1/AUTH_admin/mycontainer/file1
```

To download file1 and save it locally as localfile1:

```
curl -o localfile1 -H 'X-Auth-Token: AUTH_tk3bb59eda987446c79160202d4dfbdc8c' http://10.10.10.2:8080/v1/AUTH_admin/mycontainer/file1
```

To delete a container "mycontainer"

```
curl -X DELETE -H 'X-Auth-Token: AUTH_tk3bb59eda987446c79160202d4dfbdc8c' http://10.10.10.2:8080/v1/AUTH_admin/mycontainer
```

To delete a specific file "file1" from the container:

```
curl -X DELETE -H 'X-Auth-Token: AUTH_tk3bb59eda987446c79160202d4dfbdc8c' http://10.10.10.2:8080/v1/AUTH_admin/mycontainer/file1
```

To get metadata associated with a Swift account:

```
curl -v -X HEAD -H 'X-Auth-Token: AUTH_tk3bb59eda987446c79160202d4dfbdc8c' http://10.10.10.2:8080/v1/AUTH_admin/
```

To get metadata associated with a container:

```
curl -v -X HEAD -H 'X-Auth-Token: AUTH_tk3bb59eda987446c79160202d4dfbdc8c' http://10.10.10.2:8080/v1/AUTH_admin/mycontainer
```

You can request the data from Swift in XML or JSON format by specifying the "format" parameter. This parameter can be applied to any of the above requests. Here are a few examples:

```
curl -H 'X-Auth-Token: AUTH_tk3bb59eda987446c79160202d4dfbdc8c' http://10.10.10.2:8080/v1/AUTH_admin/?format=json
```

```
curl -H 'X-Auth-Token: AUTH_tk3bb59eda987446c79160202d4dfbdc8c' http://10.10.10.2:8080/v1/AUTH_admin/?format=xml
```

The above operations can also be done using 'swift' command. For instructions on using 'swift' command, please refer to "swift -help".

# 6. Network Management

## Table of Contents

Introduction .....	42
--------------------	----

## Introduction

In OpenStack, the networking is managed by a component called "nova-network". This interacts with nova-compute to ensure that the instances have the right kind of networking setup for them to communicate among themselves as well as with the outside world. Just as in Eucalyptus or AWS, each OpenStack instance can have 2 IP addresses attached to it. One is the private IP address and the other called Public IP address. The private IP address is typically used for communication between instances and the public IP is used for communication of instances with the outside world. The so called public IP address need not be a public IP address route-able on the Internet ; it can even be an address on the corporate LAN.

The network configuration inside the instance is done with the private IP address in view. The association between the private IP and the public IP and necessary routing is handled by nova-network and the instances need not be aware of it.

nova-network provides 3 different network management options. Currently you can only choose one of these 3 options for your network management.

- Flat Network
- Flat DHCP Network
- VLAN Network

VLAN Network is the most feature rich and is the idea choice for a production deployment, while the other modes can be used while getting familiar with OpenStack and when you do not have VLAN Enabled switches to connect different components of the OpenStack infrastructure.

The network type is chosen by using one of the following configuration options in nova.conf file. If no network manager is specified explicitly, the default network manager, VLANManager is used.

```
--network_manager = nova.network.manager.FlatManager  
--network_manager = nova.network.manager.FlatDHCPManager  
--network_manager = nova.network.manager.VlanManager
```

In each of these cases, run the following commands to set up private and public IP addresses for use by the instances:

```
sudo nova-manage network create private 192.168.4.0/24 1 256  
sudo nova-manage floating create --ip_range=10.10.10.224/27
```

The public IP which you are going to associate with an instance needs to be allocated first by using "euca-allocate-address" command:

```
euca-allocate-address 10.10.2.225
```

You can then associate a public IP to a running instance by using "euca-associate-address" command:

```
euca-associate-address -i i-00000008 10.10.2.225
```

Please refer to <http://docs.openstack.org/openstack-compute/admin/content/ch04.html> for more details about each of the networking types.

## 7. Security

### Table of Contents

Security Overview .....	44
-------------------------	----

### Security Overview

OpenStack provides ingress filtering for the instances based on the concept of security groups. OpenStack accomplishes ingress filtering by creating suitable IP Tables rules. A Security Group is a named set of rules that get applied to the incoming packets for the instances. You can specify a security group while launching an instance. Each security group can have multiple rules associated with it. Each rule specifies the source IP/network, protocol type, destination ports etc. Any packet matching these parameters specified in a rule is allowed in. Rest of the packets are blocked.

A security group that does not have any rules associated with it causes blocking of all incoming traffic. The mechanism only provides ingress filtering and does not provide any egress filtering. As a result all outbound traffic is allowed. If you need to implement egress filtering, you will need to implement that inside the instance using a firewall.

Tools like Hybridfox let you manage security groups and also let you specify a security group while launching an instance. You can also use command line tools from euca2ools package such as euca-authorize for this purpose.

Here are a few euca commands to manage security groups. Like in our earlier chapters, the project name is "proj"

Create a security group named "myservers".

```
euca-add-group -d "My Servers" myservers
```

Add a rule to the security group "myservers" allowing icmp and tcp traffic from 192.168.1.1.

```
euca-authorize -P tcp -s 192.168.1.1 -p 22 myservers
euca-authorize -P icmp -s 192.168.1.1 -t -1:-1 myservers
```

For a Windows instance, add a rule to accept incoming RDP connections

```
euca-authorize -P tcp -s 192.168.1.1 -p 3389 myservers
```

Rules can be viewed with euca-describe-groups command.

```
$ euca-describe-groups
GROUP      proj  myservers  my servers
PERMISSION proj  myservers  ALLOWS    tcp    22    22    FROM    CIDR
          192.168.1.1
```

---

PERMISSION	proj	myservers	ALLOWS	icmp	-1	-1	FROM	CIDR
192.168.1.1								
PERMISSION	proj	myservers	ALLOWS	tcp	3389	3389	FROM	
CIDR 192.168.1.1								

Remove the rule for ssh traffic from the source ip 192.168.1.1 from the security group "myservers"

```
euca-revoke -P tcp -s 192.168.1.1 -p 22 myservers
```

Delete the security group "myservers"

```
euca-delete-group myservers
```

Launch an instance associated with the security group "myservers".

```
euca-run-instances ami-XXXXXXXX -k mykey -g myservers
```

When you do not specify a security group, the instance gets associated with an inbuilt security group called "default". The rules for this security group can also be modified using euca-add, euca-revoke commands.

## 8. OpenStack Commands

### Table of Contents

Nova Manage Commands .....	46
User/Role Management .....	46
Project Management .....	47
Database Management .....	48
Instance Type Management .....	48
Service Management .....	49
Euca2ools Commands .....	50

### Nova Manage Commands

OpenStack provides commands for administrative tasks such as user/role management, network management etc. In all the examples we will use username as "novadmin" and project name as "proj". All the nova-manage commands will need to be run as "root". Either run them as root or run them under sudo.

#### User/Role Management

Add a new user

```
nova-manage user create --name=novaadmin
```

Add a user along with access and secret keys.

```
nova-manage user create --name=novaadmin --access=myaccess --secret=mysecret
```

Add a user with admin privileges

```
nova-manage user admin --name=novaadmin --access=myaccess --secret=mysecret
```

List existing users

```
nova-manage user list
```

Delete an existing user

```
nova-manage user delete --name=novaadmin
```

Associate a user to a specific existing project

```
nova-manage project add --project=proj --user=novaadmin
```

Remove a user from a specific existing project.

```
nova-manage project remove --project=proj --user=novaadmin
```

View access key and secret keys of particular user.

```
nova-manage user exports --name=novaadmin
```

Adding a user sitewide role.

```
nova-manage role add --user=novaadmin --role=netadmin
```

Remove a sitewide role from a particular user

```
nova-manage role remove --user=novaadmin --role=netadmin
```

Adding a user project specific role.

```
nova-manage role add --user=novaadmin --role=netadmin --project=proj
```

Remove a project specific role from a particular user

```
nova-manage role remove --user=novaadmin --role=netadmin --project=proj
```

With the command below, you can change any or all of access key, secret key and admin role flag for a particular user.

Syntax:

```
nova-manage user modify username new_access_key new_secret_key admin_flag  
<admin flag - T or F>
```

```
nova-manage user modify --name=novaadmin --access=mygreatnewaccesskey " " "
```

```
nova-manage user modify --name=novaadmin " " --secret=mygreatsecretkey " " "
```

```
nova-manage user modify --name=novaadmin " " " " --is_admin=T
```

Check if a particular user has a specific role or not. The role can be either local or global. The output of the command will be True or False

```
nova-manage role has --user=novaadmin --role=cloudadmin  
True
```

```
nova-manage role has --role=novaadmin --role=netadmin --project=proj  
False
```

## Project Management

The following commands help you create and manage projects. "nova-manage account" command is an alias to "nova-manage project" and you can use them interchangeably.

Create a project. It requires you to mention name of the project admin as well. css1 is the name of the project and user5 is the name of the project admin here.

```
nova-manage project create --project=css1 --user=user5 --desc="My new project"
```

List the registered projects.

```
nova-manage project list
```

Download the credentials and associated file for a specific project. Please refer to the chapter on "Installation & Configuration" for more details.

```
nova-manage project zipfile --project=csscorp --user=user5 --file=/home/user5/mysec.zip
```

Delete an existing project.

```
nova-manage project delete --project=css1
```

Check the project wise resource allocation. The output will look like this:

```
nova-manage project quota --project=css1
metadata_items: 128
gigabytes: 1000
floating_ips: 10
instances: 10
volumes: 10
cores: 20
```

## Database Management

Nova stores the data related to the projects, users, resources etc. in a database, by default in a MySQL database.

Print the current database version.

```
nova-manage db version
```

Sync the DB schema to be in sync with the current configuration.

```
nova-manage db sync
```

## Instance Type Management

Nova has the concept of instance types. Each instance type is defined with certain amount of RAM and certain size of the hard disk. When an instance is launched with a particular instance type, Nova resizes the disk image to suit the instance type and allocates the RAM as defined for the instance type chosen. Nova calls instance types as 'flavors' and lets you add to the list of flavors. By default Nova has 5 types - m1.tiny, m1.small, m1.medium, m1.large and m1.xlarge.

List the current instance types

```
nova-manage flavor list
ml.medium: Memory: 4096MB, VCPUS: 2, Storage: 40GB, FlavorID: 3, Swap: 0GB,
RXTX Quota: 0GB, RXTX Cap: 0MB
ml.large: Memory: 8192MB, VCPUS: 4, Storage: 80GB, FlavorID: 4, Swap: 0GB,
RXTX Quota: 0GB, RXTX Cap: 0MB
ml.tiny: Memory: 512MB, VCPUS: 1, Storage: 0GB, FlavorID: 1, Swap: 0GB,
RXTX Quota: 0GB, RXTX Cap: 0MB
ml.xlarge: Memory: 16384MB, VCPUS: 8, Storage: 160GB, FlavorID: 5, Swap: 0GB,
RXTX Quota: 0GB, RXTX Cap: 0MB
ml.small: Memory: 2048MB, VCPUS: 1, Storage: 20GB, FlavorID: 2, Swap: 0GB,
RXTX Quota: 0GB, RXTX Cap: 0MB
```

### Define a new instance type

```
nova-manage flavor create --name=ml.miniscule --memory=128 --cpu=1 --local_gb=
20 --flavor=6 --swap=0 --rxtx_quota=0 --rxtx_cap=0
```

### Remove an existing instance type.

```
nova-manage flavor delete --name=ml.miniscule
ml.miniscule deleted
```

## Service Management

### Check state of available services.

```
nova-manage service list
server1 nova-scheduler enabled :- ) 2011-04-06 17:01:21
server1 nova-network enabled :- ) 2011-04-06 17:01:30
server1 nova-compute enabled :- ) 2011-04-06 17:01:22
server2 nova-compute enabled :- ) 2011-04-06 17:01:28
```

### Disable a running service

```
nova-manage service disable <hostname> <service>
nova-manage service disable --host=server2 --service=nova-compute

nova-manage service list
server1 nova-network enabled :- ) 2011-04-06 17:05:11
server1 nova-compute enabled :- ) 2011-04-06 17:05:13
server1 nova-scheduler enabled :- ) 2011-04-06 17:05:17
server2 nova-compute disabled :- ) 2011-04-06 17:05:19
```

### Re-enable a service that is currently disabled

```
Syntax: nova-manage service enable <hostname> <service>
nova-manage service enable --host=server2 --service=nova-compute

nova-manage service list
server1 nova-scheduler enabled :- ) 2011-04-06 17:08:23
server1 nova-network enabled :- ) 2011-04-06 17:08:22
server1 nova-compute enabled :- ) 2011-04-06 17:08:23
server2 nova-compute enabled :- ) 2011-04-06 17:08:19
```

### Get Information about resource utilization of the OpenStack components

```
Syntax: nova-manage service describe_resource <hostname>

nova-manage service describe_resource --host=server1
HOST PROJECT  cpu mem(mb)  disk(gb)
server1(total)  2 3961 224
server1(used)  1 654 30
server1 proj  2 1024 0
```

## Euca2ools Commands

euca2ools provide a set of commands to communicate with the cloud. All these commands require you to authenticate and this is done by sourcing novarc file as detailed in the chapter on "Installation & Configuration"

Most of the euca2ools command line utilities work with OpenStack, just as they work with EC2 of AWS. There may be some differences due to some of the functionality that is yet to be implemented in OpenStack. Help is available for each of these commands with the switch `-help`.

- euca-add-group
- euca-delete-bundle
- euca-describe-instances
- euca-register
- euca-add-keypair
- euca-delete-group
- euca-describe-keypairs
- euca-release-address
- euca-allocate-address
- euca-delete-keypair
- euca-describe-regions
- euca-reset-image-attribute
- euca-associate-address
- euca-delete-snapshot
- euca-describe-snapshots
- euca-revoke
- euca-attach-volume
- euca-delete-volume

- euca-describe-volumes
- euca-run-instances
- euca-authorize
- euca-deregister
- euca-detach-volume
- euca-terminate-instances
- euca-bundle-image
- euca-describe-addresses
- euca-disassociate-address
- euca-unbundle
- euca-bundle-vol
- euca-describe-availability-zones
- euca-download-bundle
- euca-upload-bundle
- euca-confirm-product-instance
- euca-describe-groups
- euca-get-console-output
- euca-version
- euca-create-snapshot
- euca-describe-image-attribute
- euca-modify-image-attribute
- euca-create-volume
- euca-describe-images
- euca-reboot-instances