Ambiente de Desenvolvimento

Lista de Servidores

Educloud00 (LVM) keystone, glance, cinder, nova-*
Eth0 - 00:15:17:DE:B7:24 - 200.145.46.165/24 - educloud00.ncc.unesp.br CNAME cloud.ncc.unesp.br
Eth1 - 00:15:17:DE:B7:25

Educloud01 (TPM01)
Eth0 - 00:1e:67:3c:7c:df - 200.145.46.166/24 - educloud01.local
Eth1 - 00:1e:67:3c:7c:e0
Eth2 - 00:1e:67:3c:7c:e1
Eth3 - 00:1e:67:3c:7c:e2

Educloud02 (TPM02)
Eth0 - 00:1e:67:3c:89:ce - 200.145.46.167/24 - educloud02.local
Eth1 - 00:1e:67:3c:89:cf
Eth2 - 00:1e:67:3c:89:d0
Eth3 - 00:1e:67:3c:89:d1

Educloud03
Eth0 - 00:15:17:da:53:92 - 200.145.46.168/24 - educloud03.local
Eth1 - 00:15:17:da:53:93
Eth2 - 00:15:17:da:53:94
Eth3 - 00:15:17:da:53:95
Eth4 - 00:15:17:da:53:96

Educloud04
Eth0 - 00:15:17:da:55:62 - 200.145.46.169/24 - educloud04.local
Eth1 - 00:15:17:da:55:63
Eth2 - 00:15:17:da:55:64
Eth3 - 00:15:17:da:55:65
Eth4 - 00:15:17:da:55:66

Educloud05
Eth0 - 00:26:18:6d:65:a6 - 200.145.46.170/24 - educloud05.local
Eth1 - 00:26:18:6d:65:a7

Educloud06
Eth0 - 00:26:18:6d:61:64 - 200.145.46.171/24 - educloud06.local
Eth1 - 00:26:18:6d:61:65

Educloud07
Eth0 - 00:26:18:6d:67:ba - 200.145.46.172/24 - educloud07.local
Eth1 - 00:26:18:6d:67:bb

Educloud08
Eth0 - 00:26:18:6d:67:be - 200.145.46.173/24 - educloud08.local
Eth1 - 00:26:18:6d:67:bf

Educloud09
Eth0 - 00:26:18:6d:65:22 - 200.145.46.174/24 - educloud09.local
Eth1 - 00:26:18:6d:65:23

Educloud10
Eth0 - 00:26:18:6d:65:3e - 200.145.46.175/24 - educloud10.local
Eth1 - 00:26:18:6d:65:3f

Introduction

This document refers to the installation manual of the main components of

OpenStack. The version of each component used can be found in the respective sections of the components.

This document was produced by the Center for Scientific Computing, UNESP, with funding from Intel and UNESP.

OpenStack OpenStack is a global collaboration of developers and cloud computing technologists producing the ubiquitous open source cloud computing platform for public and private clouds. The project aims to deliver solutions for all types of clouds by being simple to implement, massively scalable, and feature rich. The technology consists of a series of interrelated projects delivering various components for a cloud infrastructure solution.

The OpenStack project is divided into sub projects that have a specific role in the big picture.

The main components are: Cloud Compute (Nova) Cloud Storage (Swift), Image registration and delivery (Glance), Identity (Keystone) and Dashboard (Horizon).

Each component is independent of each other and can be installed on separate machines, or all in one. In the environment of UNESP, OpenStack was installed in seven machines, described in section "Environment used."

This document is modified version of OpenStack Install and Deploy Manual [1] with minor adjustments and changes to resolve problems encountered during installation and adjustment to the environment in which it was installed.

OpenStack Core Components

OpenStack is a collection of open source technology that provides massively scalable open source cloud computing software. Currently OpenStack develops two related projects: OpenStack Compute, which offers computing power through virtual machine and network management, and OpenStack Object Storage which is software for redundant, scalable object storage capacity. Closely related to the OpenStack Compute project is the Image Service project, named Glance. OpenStack can be used by corporations, service providers, VARS, SMBs, researchers, and global data centers looking to deploy large-scale cloud deployments for private or public clouds.

The main components are: Cloud Compute (Nova) Cloud Storage (Swift), Image registration and delivery (Glance), Identity (Keystone) and Dashboard (Horizon).

They are presented here in an order for educational purposes only to a straightforward procedure for installation. But following a brief description of each component.

Keystone

Keystone is an OpenStack project that provides Identity, Token, Catalog and Policy services for use specifically by projects in the OpenStack family. It implements OpenStack’s Identity API.

Cloud Compute (Nova)

Nova is the project name for OpenStack Compute, a cloud computing fabric controller, the main part of an IaaS system. Individuals and organizations can use Nova to host and manage their own cloud computing systems. Nova originated as a project out of NASA Ames Research Laboratory.

Object Storage (Swift)

OpenStack Object Storage (Swift) is a system to store objects in a massively scalable large capacity system with built-in redundancy and failover. Object

Storage has a variety of applications, such as backing up or archiving data, serving graphics or videos (streaming data to a user's browser), storing secondary or tertiary static data, developing new applications with data storage integration, storing data when predicting storage capacity is difficult, and creating the elasticity and flexibility of cloud-based storage for your web applications.

Image Service (Glance)

OpenStack Image Service (Glance) is a lookup and retrieval system for virtual machine images. It can be configured in three ways: using OpenStack Object Store to store images; using Amazon's Simple Storage Solution (S3) storage directly; or using S3 storage with Object Store as the intermediate for S3 access.

Environment Proposed

This document aims to guide you in setting up a test environment or development. The original documentation suggests many machines. To install OpenStack here, you will need only one or two machines.

You can install all services on one machine only (all-in-one). Or you can install OpenStack controller (all services except the nova-compute) on machine A and a compute cloud controller on machine B.

For a production environment it is advisable to separate the elements to ensure a high availability and redundancy.

Settings used in this document:
- Cloud Controller:
- Cloud Computer Node:

All machines are installed with Ubuntu Server 12.04 LTS.

Initial configuration

Before starting the installation and configuration of each component must first to make some adjustments.

Network Configuration

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.

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)

In this installation we will use eth0 to public interface and eth1 for management interface.

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.

1. 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:

1. 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
2. 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.

1. ntpdate 'controllernode ip'
2. hwclock -w

Storage

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

Here we will use a "object storage" and a "block storage".

Object storage (swift)

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

Block storage (SAN)

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

Because the data is exposed as a physical device, the end-user is responsible for creating partitions and formatting the exposed disk device. In addition, in OpenStack Compute a device can only be attached to one server at a time, so

block storage cannot be used for to share data across virtual machine instances concurrently.

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).
- 192.168.69.1 is the primary IP for our controller host on eth0 and 192.168.69.2 is the primary IP for our computer node on eth0.
- FlatDHCP with a single network interface.
- KVM as the hypervisor.
- Ensure the operating system is up-to-date by running apt-get update and apt-get upgrade prior to the installation.
- Ensure that all hosts are with ntp and correct time.

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.

Keystone

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

The Identity services has two primary functions:

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

User management

The three main concepts of Identity user management are:

- Users
- Tenants
- Roles

A user represents a human user, and has associated information such as username,

password and email.

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

A role captures what operations a user is permitted to perform in a given tenant.

Service management

The two main concepts of Identity service management are:

- Services
- Endpoints

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

Installing Keystone

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

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

```
# rm /var/lib/keystone/keystone.db
```

Install mysql:

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

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
```

To configure the MySQL database, create the keystone database.

```
# mysql -u root -p
mysql> CREATE DATABASE keystone;
mysql> CREATE USER keystone;
mysql> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' IDENTIFIED BY 'keystonepass';
mysql> FLUSH PRIVILEGES;
mysql> quit;
```

Edit ***keystone.conf*** to use mysql:

```
[sql]
connection = mysql://keystone:keystonepass@192.168.69.1/keystone
```

Change in ***keystone.conf*** the ADMIN_TOKEN:

```
admin_token = 012345SECRET99TOKEN012345
```

Restart the keystone service and populate with initial data:

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

Configuring Services to work with Keystone

Create the file ***/root/openstackrc***, with theses lines

```
export SERVICE_ENDPOINT=http://192.168.69.1:35357/v2.0/
export SERVICE_TOKEN=012345SECRET99TOKEN012345
```

Theses environment variables are useful to avoid execute commands with long args, like --token or --endpoint.

Reload the file:

```
# source /root/openstackrc
```

Create the **tenants**: *openstackDemo* and *service*:

```
# keystone tenant-create --name openstackDemo --description "Default Tenant"  --enabled true
# keystone tenant-create --name service --description "Service Tenant"  --enabled true
```

Create the **user***admin*, replace the tenant_id by openStackDemo tenant_id:

```
# keystone user-create --tenant_id 9c401c9d2b9d41bcabda6919d4aec63a --name admin --pass adminpass --enabled true
```

Create the **roles**: *admin* and *user*:

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

Grant *admin***role** to **user***admin*:

```
# keystone user-role-add --user [admin_user_id] --tenant_id [openstackDemo_tenant_id] --role [admin_role]
```

Create an user to each service, replace the tenant_id by the service tenant id:

```
# keystone user-create --tenant_id 92920816c7704a8a95c916d34dc7a05c --name glance --pass glancepass --enabled true
# keystone user-create --tenant_id 92920816c7704a8a95c916d34dc7a05c --name nova --pass novapass --enabled true
# keystone user-create --tenant_id 92920816c7704a8a95c916d34dc7a05c --name swift --pass swiftpass --enabled true
```

Add the users of each service to admin **role**:

```
# keystone user-role-add --user [glance_user_id] --tenant_id [service_tenant_id] --role [admin_role]
# keystone user-role-add --user [nova_user_id] --tenant_id [service_tenant_id] --role [admin_role]
# keystone user-role-add --user [swift_user_id] --tenant_id [service_tenant_id] --role [admin_role]
```

Creating the services:

Until now, we just created: tenants, users and roles. The Keystone manage the services and yours endpoints too. In this way, the OpenStack components query to keystone where each service is located:

Run the follow commands, to create the services:

```
# keystone service-create --name keystone --type identity --description "Keystone Identity Service"
# keystone service-create --name nova --type compute --description "Nova Compute Service"
# keystone service-create --name volume --type volume --description "Nova Volume Service"
# keystone service-create --name glance --type image --description "Glance Image Service"
```

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

Run the follow commands, to create the endpoints of each service:

```
# keystone endpoint-create --region RegionOne --service_id [keystone_service_id] --publicurl http://192.168.69.1:5000/v2.0
--internalurl http://192.168.69.1:5000/v2.0 --adminurl http://192.168.69.1:35357/v2.0
# keystone endpoint-create --region RegionOne --service_id [compute_service_id] --publicurl
'http://192.168.69.1:8774/v2/%(tenant_id)s' --internalurl 'http://192.168.69.1:8774/v2/%(tenant_id)s' --adminurl
'http://192.168.69.1:8774/v2/%(tenant_id)s'
# keystone endpoint-create --region RegionOne --service_id [volume_service_id] --publicurl
'http://192.168.69.1:8776/v1/%(tenant_id)s' --internalurl 'http://192.168.69.1:8776/v1/%(tenant_id)s' --adminurl
'http://192.168.69.1:8776/v1/%(tenant_id)s'
# keystone endpoint-create --region RegionOne --service_id [glance_service_id] --publicurl http://192.168.69.1:9292/v1
--internalurl http://192.168.69.1:9292/v1 --adminurl http://192.168.69.1:9292/v1
# keystone endpoint-create --region RegionOne --service_id [swift_service_id] --publicurl
'http://192.168.69.1:8888/v1/AUTH_%(tenant_id)s' --adminurl 'http://192.168.69.1:8888/' --internalurl
'http://192.168.69.1:8888/v1/AUTH_%(tenant_id)s'
```

Checking the keystone installation:w

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

Do a REST request to keystone-api using curl:

```
# curl -d '{"auth": {"tenantName": "openstackDemo", "passwordCredentials": {"username": "admin", "password": "adminpass"}}}' -H
"Content-type: application/json" http://192.168.69.1:35357/v2.0/tokens  | python -mjson.tool
```

The output should have the services, endpoints and a valid token for 24 hours:


```
{
   "access": {
      "serviceCatalog": [
         {
            "endpoints": [
               {
                  "adminURL": "http://192.168.69.1:8776/v1/9c401c9d2b9d41bcabda6919d4aec63a",
                  "internalURL": "http://192.168.69.1:8776/v1/9c401c9d2b9d41bcabda6919d4aec63a",
                  "publicURL": "http://192.168.69.1:8776/v1/9c401c9d2b9d41bcabda6919d4aec63a",
                  "region": "RegionOne"
               }
            ],
            "endpoints_links": [],
            "name": "volume",
            "type": "volume"
         },
         {
            "endpoints": [
               {
                  "adminURL": "http://192.168.69.1:9292/v1",
                  "internalURL": "http://192.168.69.1:9292/v1",
                  "publicURL": "http://192.168.69.1:9292/v1",
                  "region": "RegionOne"
               }
            ],
            "endpoints_links": [],
            "name": "glance",
            "type": "image"
         },
         {
            "endpoints": [
               {
                  "adminURL": "http://192.168.69.1:8774/v2/9c401c9d2b9d41bcabda6919d4aec63a",
```

```
                "internalURL": "http://192.168.69.1:8774/v2/9c401c9d2b9d41bcabda6919d4aec63a",
                "publicURL": "http://192.168.69.1:8774/v2/9c401c9d2b9d41bcabda6919d4aec63a",
                "region": "RegionOne"
            }
        ],
        "endpoints_links": [],
        "name": "nova",
        "type": "compute"
    },
    {
        "endpoints": [
            {
                "adminURL": "http://192.168.69.1:35357/v2.0",
                "internalURL": "http://192.168.69.1:5000/v2.0",
                "publicURL": "http://192.168.69.1:5000/v2.0",
                "region": "RegionOne"
            }
        ],
        "endpoints_links": [],
        "name": "keystone",
        "type": "identity"
    },
    {
        "endpoints": [
            {
                "adminURL": "http://192.168.69.1:8888/",
                "internalURL": "http://192.168.69.1:8888/v1/AUTH_9c401c9d2b9d41bcabda6919d4aec63a",
                "publicURL": "http://192.168.69.1:8888/v1/AUTH_9c401c9d2b9d41bcabda6919d4aec63a",
                "region": "RegionOne"
            }
        ],
        "endpoints_links": [],
        "name": "swift",
        "type": "object-store"
    }
],
"token": {
    "expires": "2012-06-08T15:31:46Z",
    "id": "05b818ff23564d5599c97ea5e1df7bbe",
    "tenant": {
        "description": "Default Tenant",
        "enabled": true,
        "id": "9c401c9d2b9d41bcabda6919d4aec63a",
        "name": "openstackDemo"
    }
},
"user": {
    "id": "1fcfbe778fc3428aa30db782056b34d7",
    "name": "admin",
    "roles": [
        {
            "id": "a2d6da77c0a54105aab94484e0d2088b",
            "name": "admin"
        }
    ],
    "roles_links": [],
    "username": "admin"
}
}
}
```


You can list users, roles, tenants and services:

```
# keystone user-list
+----------------------------------+---------+-------+--------+
```

```
|                id                | enabled | email |  name  |
+----------------------------------+---------+-------+--------+
| 029c5749548946f7b560b2706d31d488 | True    | None  | glance |
| 1fcfbe778fc3428aa30db782056b34d7 | True    | None  | admin  |
| 4b75079aeb1b408ebdb286bfb85e719f | True    | None  | swift  |
| e25f81d50eb84bdb92907adbd0e41f25 | True    | None  | nova   |
+----------------------------------+---------+-------+--------+
```

```
# keystone role-list
+----------------------------------+-------+
|                id                |  name |
+----------------------------------+-------+
| 845e83e4d9bf4d8d8188b1e261dfe103 | user  |
| a2d6da77c0a54105aab94484e0d2088b | admin |
+----------------------------------+-------+
```

```
# keystone tenant-list
+----------------------------------+---------------+---------+
|                id                |      name     | enabled |
+----------------------------------+---------------+---------+
| 92920816c7704a8a95c916d34dc7a05c | service       | True    |
| 9c401c9d2b9d41bcabda6919d4aec63a | openstackDemo | True    |
+----------------------------------+---------------+---------+
```

```
# keystone service-list
+----------------------------------+----------+---------------+--------------------------+
|                id                |   name   |      type     |       description        |
+----------------------------------+----------+---------------+--------------------------+
| 00486163c7c64c6a86f577627071aafc | nova     | compute       | Nova Compute Service     |
| 69d3a6b21c9348acb1042d7d9386ff81 | volume   | volume        | Nova Volume Service      |
| 7b09782f9faf448781f5a460acabe258 | swift    | object-store  | Object Storage Service   |
| d6b32583a6694e9b828b2a50037623ba | keystone | identity      | Keystone Identity Service |
| f7f964a71e2a45119dcaede68fafbd78 | glance   | image         | Glance Image Service     |
+----------------------------------+----------+---------------+--------------------------+
```

Installing OpenStack Object Storage - Swift

OpenStack Object Storage (Swift) is a system to store objects in a massively scalable large capacity system with built-in redundancy and failover. Object Storage has a variety of applications, such as backing up or archiving data, serving graphics or videos (streaming data to a user's browser), storing secondary or tertiary static data, developing new applications with data storage integration, storing data when predicting storage capacity is difficult, and creating the elasticity and flexibility of cloud-based storage for your web applications.

In this environment we use the same controller node as storage too. So the swift and swift-proxy is installed in the controller-node.

Install the swift and dependencies:

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

Unfortunately at the time of writing this manual swift package not automates some tasks, such as the creation of configuration files. We have to do this manually:

```
# mkdir /etc/swift
```

Create the config file */etc/swift/swift.conf*:

```
[swift-hash]
```

```
# random unique string that can never change (DO NOT LOSE)
swift_hash_path_suffix = fLIbertYgibbitZ
```

Install and configure the "object-nodes"

In this case, the object-nodes (computer when data are stored) is the same cloud-controller.

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

Each "Object node" can have many devices Is a requisite use xfs file system.

```
# fdisk /dev/sdb (set up a single partition)
# mkfs.xfs -i size=1024 /dev/sdb1
```

This device should be mount at /srv/node/

```
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
```

Configure the *rsyncd.conf*

```
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = 192.168.69.1

[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
```

Edit the */etc/default/rsync*:

```
RSYNC_ENABLE = true
```

Restart the rsync:

```
service rsync restart
```

Create the */etc/swift/account-server.conf*:

```
[DEFAULT]
bind_ip = 192.168.69.1
```

```
workers = 2

[pipeline:main]
pipeline = account-server

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

[account-replicator]

[account-auditor]

[account-reaper]
```

Create the */etc/swift/container-server.conf*:

```
[DEFAULT]
bind_ip = 192.168.69.1
workers = 2

[pipeline:main]
pipeline = container-server

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

[container-replicator]

[container-updater]

[container-auditor]
```

Create the */etc/swift/object-server.conf*:

```
[DEFAULT]
bind_ip = 192.168.69.1
workers = 2

[pipeline:main]
pipeline = object-server

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

[object-replicator]

[object-updater]

[object-auditor]
```

Installing and configuring the swift-proxy

The swift-proxy is the component that acts like a proxy, so you can have multiple object-nodes and one swift-proxy. When an user or openStack component need get a object, its ask for swift-proxy.

Install swift-proxy:

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

Create a self-sign SSL certificate:

```
cd /etc/swift
```

```
openssl req -new -x509 -nodes -out cert.crt -keyout cert.key
```

Change the memcached to listen in correct interface, change the /etc/memcached.conf:

```
-l 127.0.0.1
to
-l 192.168.69.1
```

Restart the memcached:

```
service memcached restart
```

Create the file */etc/swift/proxy-server.conf*:

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

[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 = 10
service_port = 5000
service_host = 192.168.69.1
auth_port = 35357
auth_host = 192.168.69.1
auth_token = 012345SECRET99TOKEN012345
admin_token = 012345SECRET99TOKEN012345
auth_protocol = http

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

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

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

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 1 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 1 1
swift-ring-builder container.builder create 18 1 1
swift-ring-builder object.builder create 18 1 1
```

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
```

In our case, we have only one zone in one storage node with one device (sdb1), so the command should be:

```
swift-ring-builder account.builder add z1-192.168.69.1:6002/sdb1 100
swift-ring-builder container.builder add z1-192.168.69.1:6001/sdb1 100
swift-ring-builder object.builder add z1-192.168.69.1:6000/sdb1 100
```

Verify the ring contents for each ring:

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

Rebalance the rings:

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

Change the permissions of config files:

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

Start the swift-proxy:

```
swift-init proxy start
```

Start all swift services:

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

Checking the installation

The file */root/openstackrc* will receive more environment variables:

```
# Used for Keystone
export SERVICE_ENDPOINT=http://192.168.69.1:35357/v2.0/
export SERVICE_TOKEN=012345SECRET99TOKEN012345

# Used for swift
export ST_AUTH=http://192.168.69.1:5000/v2.0
export ST_USER=openstackDemo:admin
export ST_KEY=adminpass
export OS_REGION_NAME=RegionOne
```

Reload the file:

```
# source /root/openstackrc
```

Try to get swift status, with the command:

```
# swift -V 2.0 stat
```

List the containers:

```
# swift -V 2.0 list
```

Do some upload files:

```
# swift -V 2.0 upload myfiles bigfile1.tgz
# swift -V 2.0 upload myfiles bigfile2.tgz
```

Download the container "myfiles":

```
# cd /tmp/
# swift -V 2.0 download myfiles
```

Installing OpenStack Image Service - Glance

OpenStack Image Service (Glance) is a lookup and retrieval system for virtual machine images. It can be configured in three ways: using OpenStack Object Store to store images; using Amazon's Simple Storage Solution (S3) storage directly; or using S3 storage with Object Store as the intermediate for S3 access.

Install the glance on the controller node:

```
# apt-get install glance
```

Remove the sqlite file:

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

Configuring the Image Service database backend

```
# mysql -u root -p
mysql> CREATE DATABASE glance;
mysql> CREATE USER glance;
mysql> GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'%' IDENTIFIED BY 'glancepass';
mysql> FLUSH PRIVILEGES;
mysql> quit;
```

Copy the config files to */etc/glance/*:

/etc/glance/glance-api.conf:

```
[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 = swift

# Address to bind the API server
bind_host = 0.0.0.0
```

```
# Port the bind the API server to
bind_port = 9292

# Log to this file. Make sure you do not set the same log
# file for both the API and registry servers!
log_file = /var/log/glance/api.log

# Backlog requests when creating socket
backlog = 4096

# Number of Glance API worker processes to start.
# On machines with more than one CPU increasing this value
# may improve performance (especially if using SSL with
# compression turned on). It is typically recommended to set
# this value to the number of CPUs present on your machine.
workers = 0

# Role used to identify an authenticated user as administrator
#admin_role = admin

# ================= Syslog Options ============================

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

# Facility to use. If unset defaults to LOG_USER.
# syslog_log_facility = LOG_LOCAL0

# ================= SSL Options ===============================

# Certificate file to use when starting API server securely
# cert_file = /path/to/certfile

# Private key file to use when starting API server securely
# key_file = /path/to/keyfile

# ================= Security Options ==========================

# AES key for encrypting store 'location' metadata, including
# -- if used -- Swift or S3 credentials
# Should be set to a random string of length 16, 24 or 32 bytes
# metadata_encryption_key = <16, 24 or 32 char registry metadata key>

# ============ Registry Options ===============================

# Address to find the registry server
registry_host = 0.0.0.0

# Port the registry server is listening on
registry_port = 9191

# What protocol to use when connecting to the registry server?
# Set to https for secure HTTP communication
registry_client_protocol = http

# The path to the key file to use in SSL connections to the
# registry server, if any. Alternately, you may set the
# GLANCE_CLIENT_KEY_FILE environ variable to a filepath of the key file
# registry_client_key_file = /path/to/key/file

# The path to the cert file to use in SSL connections to the
# registry server, if any. Alternately, you may set the
# GLANCE_CLIENT_CERT_FILE environ variable to a filepath of the cert file
# registry_client_cert_file = /path/to/cert/file

# The path to the certifying authority cert file to use in SSL connections
```

```
# to the registry server, if any. Alternately, you may set the
# GLANCE_CLIENT_CA_FILE environ variable to a filepath of the CA cert file
# registry_client_ca_file = /path/to/ca/file

# ============ Notification System Options =====================

# Notifications can be sent when images are create, updated or deleted.
# There are three methods of sending notifications, logging (via the
# log_file directive), rabbit (via a rabbitmq queue), qpid (via a Qpid
# message queue), or noop (no notifications sent, the default)
notifier_strategy = noop

# Configuration options if sending notifications via rabbitmq (these are
# the defaults)
rabbit_host = localhost
rabbit_port = 5672
rabbit_use_ssl = false
rabbit_userid = guest
rabbit_password = guest
rabbit_virtual_host = /
rabbit_notification_exchange = glance
rabbit_notification_topic = glance_notifications

# Configuration options if sending notifications via Qpid (these are
# the defaults)
qpid_notification_exchange = glance
qpid_notification_topic = glance_notifications
qpid_host = localhost
qpid_port = 5672
qpid_username =
qpid_password =
qpid_sasl_mechanisms =
qpid_reconnect_timeout = 0
qpid_reconnect_limit = 0
qpid_reconnect_interval_min = 0
qpid_reconnect_interval_max = 0
qpid_reconnect_interval = 0
qpid_heartbeat = 5
# Set to 'ssl' to enable SSL
qpid_protocol = tcp
qpid_tcp_nodelay = True

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

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

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

# Version of the authentication service to use
# Valid versions are '2' for keystone and '1' for swauth and rackspace
swift_store_auth_version = 2

# Address where the Swift authentication service lives
# Valid schemes are 'http://' and 'https://'
# If no scheme specified,  default to 'https://'
# For swauth, use something like '127.0.0.1:8080/v1.0/'
swift_store_auth_address = http://192.168.69.1:35357/v2.0/

# User to authenticate against the Swift authentication service
# If you use Swift authentication service, set it to 'account':'user'
# where 'account' is a Swift storage account and 'user'
# is a user in that account
swift_store_user = service:glance

# Auth key for the user authenticating against the
```

```
# Swift authentication service
swift_store_key = glancepass

# 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 = True

# 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
# Valid schemes are 'http://' and 'https://'
# If no scheme specified,  default to 'http://'
s3_store_host = 127.0.0.1:8080/v1.0/

# User to authenticate against the S3 authentication service
s3_store_access_key = <20-char AWS access key>

# Auth key for the user authenticating against the
# S3 authentication service
s3_store_secret_key = <40-char AWS secret key>

# Container within the account that the account should use
# for storing images in S3. Note that S3 has a flat namespace,
# so you need a unique bucket name for your glance images. An
# easy way to do this is append your AWS access key to "glance".
# S3 buckets in AWS *must* be lowercased, so remember to lowercase
# your AWS access key if you use it in your bucket name below!
s3_store_bucket = <lowercased 20-char aws access key>glance

# Do we create the bucket if it does not exist?
s3_store_create_bucket_on_put = False

# When sending images to S3, the data will first be written to a
# temporary buffer on disk. By default the platform's temporary directory
# will be used. If required, an alternative directory can be specified here.
# s3_store_object_buffer_dir = /path/to/dir

# ============ RBD Store Options =============================

# Ceph configuration file path
# If using cephx authentication, this file should
# include a reference to the right keyring
# in a client.<USER> section
rbd_store_ceph_conf = /etc/ceph/ceph.conf
```

```
# RADOS user to authenticate as (only applicable if using cephx)
rbd_store_user = glance

# RADOS pool in which images are stored
rbd_store_pool = images

# Images will be chunked into objects of this size (in megabytes).
# For best performance, this should be a power of two
rbd_store_chunk_size = 8

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

# Turn on/off delayed delete
delayed_delete = False

# Delayed delete time in seconds
scrub_time = 43200

# Directory that the scrubber will use to remind itself of what to delete
# Make sure this is also set in glance-scrubber.conf
scrubber_datadir = /var/lib/glance/scrubber

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

# Base directory that the Image Cache uses
image_cache_dir = /var/lib/glance/image-cache/

[paste_deploy]
flavor = keystone
```

/etc/glance/glance-api-paste.ini:

```
# Default minimal pipeline
[pipeline:glance-api]
pipeline = versionnegotiation context apiv1app authtoken

# Use the following pipeline for keystone auth
# i.e. in glance-api.conf:
#   [paste_deploy]
#   flavor = keystone
#
[pipeline:glance-api-keystone]
pipeline = versionnegotiation authtoken context apiv1app

# Use the following pipeline to enable transparent caching of image files
# i.e. in glance-api.conf:
#   [paste_deploy]
#   flavor = caching
#
[pipeline:glance-api-caching]
pipeline = versionnegotiation context cache apiv1app

# Use the following pipeline for keystone auth with caching
# i.e. in glance-api.conf:
#   [paste_deploy]
#   flavor = keystone+caching
#
[pipeline:glance-api-keystone+caching]
pipeline = versionnegotiation authtoken context cache apiv1app

# Use the following pipeline to enable the Image Cache Management API
# i.e. in glance-api.conf:
#   [paste_deploy]
#   flavor = cachemanagement
#
```

```
[pipeline:glance-api-cachemanagement]
pipeline = versionnegotiation context cache cachemanage apiv1app

# Use the following pipeline for keystone auth with cache management
# i.e. in glance-api.conf:
#   [paste_deploy]
#   flavor = keystone+cachemanagement
#
[pipeline:glance-api-keystone+cachemanagement]
pipeline = versionnegotiation authtoken context cache cachemanage 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 = 192.168.69.1
service_port = 5000
auth_host = 192.168.69.1
auth_port = 35357
auth_protocol = http
auth_uri = http://192.168.69.1:5000/
admin_tenant_name = service
admin_user = glance
admin_password = glancepass
```

/etc/glance/glance-registry.conf:

```
[DEFAULT]
# Show more verbose log output (sets INFO log level output)
verbose = True

# Show debugging output in logs (sets DEBUG log level output)
debug = True

# Address to bind the registry server
bind_host = 0.0.0.0

# Port the bind the registry server to
bind_port = 9191

# Log to this file. Make sure you do not set the same log
# file for both the API and registry servers!
log_file = /var/log/glance/registry.log

# Backlog requests when creating socket
backlog = 4096
```

```
# SQLAlchemy connection string for the reference implementation
# registry server. Any valid SQLAlchemy connection string is fine.
# See: http://www.sqlalchemy.org/docs/05/reference/sqlalchemy/connections.html#sqlalchemy.create_engine
sql_connection = mysql://glance:glancepass@192.168.69.1/glance

# Period in seconds after which SQLAlchemy should reestablish its connection
# to the database.
#
# MySQL uses a default `wait_timeout` of 8 hours, after which it will drop
# idle connections. This can result in 'MySQL Gone Away' exceptions. If you
# notice this, you can lower this value to ensure that SQLAlchemy reconnects
# before MySQL can drop the connection.
sql_idle_timeout = 3600

# Limit the api to return `param_limit_max` items in a call to a container. If
# a larger `limit` query param is provided, it will be reduced to this value.
api_limit_max = 1000

# If a `limit` query param is not provided in an api request, it will
# default to `limit_param_default`
limit_param_default = 25

# Role used to identify an authenticated user as administrator
#admin_role = admin

# ================= Syslog Options ============================

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

# Facility to use. If unset defaults to LOG_USER.
# syslog_log_facility = LOG_LOCAL1

# ================= SSL Options ===============================

# Certificate file to use when starting registry server securely
# cert_file = /path/to/certfile

# Private key file to use when starting registry server securely
# key_file = /path/to/keyfile

[paste_deploy]
flavor = keystone
```

/etc/glance/glance-registry-paste.ini:

```
# Default minimal pipeline
[pipeline:glance-registry]
pipeline = context registryapp authtoken

# Use the following pipeline for keystone auth
# i.e. in glance-registry.conf:
#   [paste_deploy]
#   flavor = keystone
#
[pipeline:glance-registry-keystone]
pipeline = authtoken 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
service_protocol = http
service_host = 192.168.69.1
service_port = 5000
auth_host = 192.168.69.1
auth_port = 35357
auth_protocol = http
auth_uri = http://192.168.69.1:5000/
admin_tenant_name = service
admin_user = glance
admin_password = glancepass
```

/etc/glance/glance-scrubber.conf:

```
[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

# Should we run our own loop or rely on cron/scheduler to run us
daemon = False

# Loop time between checking for new items to schedule for delete
wakeup_time = 300

sql_connection = mysql://glance:glancepass@192.168.69.1/glance

# Directory that the scrubber will use to remind itself of what to delete
# Make sure this is also set in glance-api.conf
scrubber_datadir = /var/lib/glance/scrubber

# Only one server in your deployment should be designated the cleanup host
cleanup_scrubber = False

# pending_delete items older than this time are candidates for cleanup
cleanup_scrubber_time = 86400

# Address to find the registry server for cleanups
registry_host = 0.0.0.0

# Port the registry server is listening on
registry_port = 9191
```

Synchronizing the database and restarting the services

```
# glance-manage version_control 0
# glance-manage db_sync
# service glance-registry restart
# service glance-api restart
```

Testing the Glance installation:

Put the new env vars to */root/openstackrc*:

```
# Used for Keystone
export SERVICE_ENDPOINT=http://192.168.69.1:35357/v2.0/
export SERVICE_TOKEN=012345SECRET99TOKEN012345

# Used for swift
export ST_AUTH=http://192.168.69.1:5000/v2.0
export ST_USER=openstackDemo:admin
export ST_KEY=adminpass
export OS_REGION_NAME=RegionOne

# Used for glance
export OS_USERNAME=admin
export OS_TENANT_NAME=openstackDemo
export OS_PASSWORD=adminpass
export OS_AUTH_URL=http://192.168.69.1:5000/v2.0/
export OS_REGION_NAME=RegionOne
```

Download 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 image:

```
# cd /tmp/images/
# glance add name="tty-linux-kernel" disk_format=aki container_format=aki < ttylinux-uec-amd64-12.1_2.6.35-22_1-vmlinuz
# glance add name="tty-linux-ramdisk" disk_format=ari container_format=ari < ttylinux-uec-amd64-12.1_2.6.35-22_1-loader
# glance add name="tty-linux" disk_format=ami container_format=ami kernel_id=[kernel_image_id]
ramdisk_id=[kernel_ramdisk_image_id] < ttylinux-uec-amd64-12.1_2.6.35-22_1.img
```

List the images:

```
# glance index
```

OpenStack Compute service - nova

Nova is the project name for OpenStack Compute, a cloud computing fabric controller, the main part of an IaaS system. Individuals and organizations can use Nova to host and manage their own cloud computing systems. Nova originated as a project out of NASA Ames Research Laboratory.

OpenStack Compute gives you a tool to orchestrate a cloud, including running instances, managing networks, and controlling access to the cloud through users and projects. The underlying open source project's name is Nova, and it provides the software that can control an Infrastructure as a Service (IaaS) cloud computing platform. It is similar in scope to Amazon EC2 and Rackspace Cloud Servers. OpenStack Compute does not include any virtualization software; rather it defines drivers that interact with underlying virtualization mechanisms that run on your host operating system, and exposes functionality over a web-based API.

AMQP and Nova

AMQP is the messaging technology chosen by the OpenStack cloud. The AMQP broker, either RabbitMQ or Qpid, sits between any two Nova components and allows them to communicate in a loosely coupled fashion. More precisely, Nova components (the compute fabric of OpenStack) use Remote Procedure Calls (RPC hereinafter) to communicate to one another.

For more information about AMQP in Nova, please read the AMQP section in nova manual

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 specific walkthrough configuration options to KVM.

Configuring the kvm

KVM is configured as the default hypervisor for Compute. To enable KVM explicitly, add the following configuration options /etc/nova/nova.conf:

```
connection_type=libvirt
libvirt_type=kvm
```

The KVM hypervisor supports the following virtual machine image formats:

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

Checking for hardware virtualization support

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

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

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

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

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

Enabling KVM

KVM requires the kvm and either kvm-intel or kvm-amd modules to be loaded. This may have been configured automatically on your distribution when KVM is installed. You can check that they have been loaded using lsmod, as follows, with expected output for Intel-based processors:

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

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

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

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

```
kvm
kvm-intel
```

Configuring the Compute Service database backend

To configure the MySQL database, create the nova database on the cloud controller machine:

```
mysql> CREATE DATABASE nova;
mysql> CREATE USER nova;
mysql> GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%' IDENTIFIED BY 'novapass';
mysql> quit
```

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-volume nova-compute nova-vncproxy nova-api  nova-ajax-console-proxy nova-cert nova-consoleauth
nova-doc nova-scheduler nova-network
```

```
nova-manage config convert --infile=/etc/nova/nova.conf --outfile=/etc/nova/nova.conf.new
mv /etc/nova/nova.conf.new /etc/nova/nova.conf
```

h3. Configuring nova
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
```

Modify your nova.conf, nova-compute and api-paste.ini according to the following examples:

nova-compute.conf

```
[DEFAULT]
libvirt_type=kvm
```

nova.conf

```
[DEFAULT]
# LOGS/STATE
verbose=True
use_syslog=False
debug=False

#logging_default_format_string =  '%(asctime)s %(levelname)s %(name)s [-] %(message)s'
logging_default_format_string = '%(asctime)s %(levelname)s %(name)s [-] %(message)s'
```

```
logging_context_format_string = '%(asctime)s %(levelname)s %(name)s [-] %(message)s'

# AUTHENTICATION
auth_strategy=keystone
logdir=/var/log/nova/
state_path=/var/lib/nova
lock_path=/var/run/nova
bindir=/usr/bin/

force_to_raw=False

# SCHEDULER
compute_scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler

scheduler_default_filters=AllHostsFilter

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

# DATABASE
sql_connection=mysql://nova:novapass@192.168.69.1/nova

# COMPUTE
libvirt_type=kvm
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=192.168.69.1
s3_host=192.168.69.1
# RABBITMQ
rabbit_host=192.168.69.1
# GLANCE
image_service=nova.image.glance.GlanceImageService
glance_api_servers=192.168.69.1:9292
# 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=192.168.69.1
public_interface=br100
vlan_interface=eth0
flat_network_bridge=br100
flat_interface=eth0
fixed_range=192.168.70.0/24
# NOVNC CONSOLE - You should change this IP to public ip of current nova-compute machine
novncproxy_base_url=http://200.145.46.167:6080/vnc_auto.html
vncserver_proxyclient_address=200.145.46.167
vncserver_listen=200.145.46.167
xvpvncproxy_base_url=http://200.145.46.167:6081/console
```

api-paste.ini

```
############
# 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 = 192.168.69.1
service_port = 5000
auth_host = 192.168.69.1
auth_port = 35357
auth_protocol = http
auth_uri = http://192.168.69.1:5000/
admin_tenant_name = service
admin_user = nova
admin_password = novapass
```

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
```

Installing additional computer 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.

If you are using a network bridge configuration and the computer nodes will not have external access, no need to install the nova-network computer nodes.

Configuar e atualizar apt (Verificar se estar usando o mirror ubumirror)

```
# apt-get update
# apt-get upgrade
```

Configure bridge

```
# apt-get install bridge-utils
```

Configurar o network/interfaces

Configurar KVM:

```
# modprobe kvm_intel
echo "kvm_intel" >> /etc/modules
```

Configurar ldap para evitar conflito com usuario "nova" (copiar do /etc/nslcd.conf)

Alterar o /etc/sudoers (copiar da 192.168.1.100)

Instalar nova-compute e nova-network:

```
# apt-get install nova-compute nova-network
# cd /etc/nova/
# mkdir back
# mv * back/
# scp 192.168.1.100:/tmp/nova.conf.tgz .
# tar zxvf nova.conf.tgz
# rm nova.conf.tgz
```

Altear os arquivos de configuracao nova.conf (console de vnc e my_ip)

Colocar o openstack.rc no /root/

Configuring the Database for Compute

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

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

Restart all the services:

```
# restart nova-api
# restart nova-compute
# restart nova-network
# restart nova-scheduler
# restart nova-vncproxy
# restart nova-volume
# restart libvirt-bin
# restart nova-cert
# restart nova-consoleauth
# 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 192.168.70.0/24 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=192.168.70.0/24 --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    educloud01                 nova          disabled   XXX   2012-06-03 14:23:45
nova-network    educloud01                nova          enabled    :-)   2012-06-08 12:04:21
nova-scheduler  educloud01                 nova          enabled    :-)   2012-06-08 12:04:14
nova-volume     educloud01                nova          enabled    :-)   2012-06-08 12:04:21
```

```
nova-cert       educloud01                  nova          enabled    :-)   2012-06-08 12:04:21
nova-consoleauth educloud01                   nova          enabled    :-)   2012-06-08 12:04:21
nova-compute    educloud02                  nova          enabled    :-)   2012-06-08 12:04:00
nova-network    educloud02                  nova          enabled    :-)   2012-06-08 12:04:00
```

Creating a multinic instance

In order to create a multinic (multiple network interfaces) the parameter --nic specifying the id of the network created on nova by the nova-manage network create (the net-id parameter).
The user can also specify the ip of the virtual machine (the v4-fixed-ip parameter).
For instance if the user wants to create a virtual machine on two separate network with a specific ips he/she should perform the command

```
# nova boot --flavor 1 --nic net-id=e55b9517-83eb-4377-b7d1-6487529f30f7,v4-fixed-ip=192.168.100.109 --nic
net-id=47a3cef8-b5bf-4880-b621-96d08009d57f,v4-fixed-ip=192.168.200.109
```

Installing and configuring Cinder Volume Manager

Install the packages.

```
$ sudo apt-get install cinder-api cinder-scheduler cinder-volume open-iscsi python-cinderclient tgt
```

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

```
[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
service_protocol = http
service_host = educloud00.local
service_port = 5000
auth_host = educloud00.local
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = cinder
admin_password = openstack
```

Edit /etc/cinder/cinder.conf.

```
[DEFAULT]
rootwrap_config=/etc/cinder/rootwrap.conf
sql_connection = mysql://cinder:CINDER_PASS@educloud00.local/cinder
api_paste_config = /etc/cinder/api-paste.ini

iscsi_helper=tgtadm
volume_name_template = volume-%s
volume_group = cinder-volumes
verbose = True
auth_strategy = keystone
#osapi_volume_listen_port=5900

state_path = /var/lib/cinder
lock_path = /var/lock/cinder
volumes_dir = /var/lib/cinder/volumes
```

Verify entries in nova.conf.

```
volume_api_class=nova.volume.cinder.API
enabled_apis=ec2,osapi_compute,metadata
#MAKE SURE NO ENTRY FOR osapi_volume anywhere in nova.conf!!!
```

```
#Leaving out enabled_apis altogether is NOT sufficient, as it defaults to include osapi_volume
```

Edit the configuration file: /etc/tgt/targets.conf in order to include the cinder_tgt.conf instead of conf.d/*.conf

```
include /etc/tgt/conf.d/cinder_tgt.conf
```

create the file include /etc/tgt/conf.d/cinder_tgt.conf with the following content:

```
include /var/lib/cinder/volumes/*
```

Restart the tgt service.

```
$ sudo restart tgt
```

Populate the database.

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

Restart the services.

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

Create a 1 GB test volume.

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

the result should look like this:

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

Attaching a volume to a virtual machine

First we have to create a virtual machine instance using whatever image and flavor we want:

```
# nova create --image <YOUR IMAGE> --flavor 1 myinstance
```

Then we can use the nova volume-attach tool to attach it to the instance. We need to specify a device identifier within the instance that isn't being used. In
our case we know that /dev/vdb is free, so we will use that:

```
# nova volume-attach myinstance da6ae608-4673-4b24-acd4-75e527b5969a /dev/vdb
```

Booting from a volume

In some cases the users of the virtual machine would like them to be persistent, so changes in kernels, initram would be kept. A solution for these cases is to
use a volume to boot the virtual machine.
After booting from a volume, the user data will be persisted at the file system attached to the virtual machine.

The user can create an image from within a running virtual machine, accessing the device created by the attach command.
Another way to do that is to create a volume from an image. Nova has an utility to do so, the --image-id parameter when using the volume-create utility e.g.:

```
# nova volume-create --image-id <image_id> --display-name my-bootable-vol <size>
```

After creating the volume the user can boot from it using the nova boot command:

```
# nova boot --flavor <favlor_id> --block_device_mapping <dev_name=mapping> boot-from-vol-test
```

where device_mapping} in the format <dev_name$>=<id>:<type>:<size(GB)>:<delete_on_terminate>, with dev_name
being the device name where the volume will be attached in the virtual machine at /dev/dev_name.
Most of the times, this value is vda. id is the id of the volume created with the cinder create command.

type is a string that describes the process used to creat the volume, e.g, it can be snap, which means that the volume was created from a snapshot.
This parameter can be left in blank.

The delete_on_terminate specifies wether the volume should be deleted after the virtual machine is terminated.

Installing the OpenStack frontend - Horizon

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

- 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.
- Python 2.6 is required;
- It should run on any system with Python 2.6 or 2.7 that is capable of running Django;
- Optional components:
  - *An Image Store (Glance) endpoint.*
  - *An Object Store (Swift) endpoint.*

Install the OpenStack Dashboard, as root:

```
# 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/'
```

To configure the MySQL database, create the dash database.

```
mysql> CREATE DATABASE dash;
mysql> GRANT ALL ON dash.* TO 'dash'@'%' IDENTIFIED BY 'dashpass';
mysql> quit;
```

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

```
import os
from django.utils.translation import ugettext_lazy as _
DEBUG = False
TEMPLATE_DEBUG = DEBUG
PROD = False
USE_SSL = False
SWIFT_ENABLED = True
ENABLE_JUJU_PANEL = True
SECRET_KEY = 'elj1IWiLoWHgcyYxFVLj7cM5rGOOxWl0'
LOCAL_PATH = os.path.dirname(os.path.abspath(__file__))
CACHE_BACKEND = 'memcached://192.168.69.1:11211/'
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
OPENSTACK_HOST = "192.168.69.1"
OPENSTACK_KEYSTONE_URL = "http://%s:5000/v2.0" % OPENSTACK_HOST
OPENSTACK_KEYSTONE_DEFAULT_ROLE = "Member"
OPENSTACK_KEYSTONE_BACKEND = {
   'name': 'native',
   'can_edit_user': True
}
API_RESULT_LIMIT = 1000
LOGGING = {
      'version': 1,
      # When set to True this will disable all logging except
      # for loggers specified in this configuration dictionary. Note that
      # if nothing is specified here and disable_existing_loggers is True,
      # django.db.backends will still log unless it is disabled explicitly.
      'disable_existing_loggers': False,
      'handlers': {
         'null': {
            'level': 'DEBUG',
            'class': 'django.utils.log.NullHandler',
            },
         'console': {
            # Set the level to "DEBUG" for verbose output logging.
            'level': 'INFO',
            'class': 'logging.StreamHandler',
            },
         },
      'loggers': {
         # Logging from django.db.backends is VERY verbose, send to null
         # by default.
         'django.db.backends': {
            'handlers': ['null'],
            'propagate': False,
            },
         'horizon': {
            'handlers': ['console'],
            'propagate': False,
         },
         'novaclient': {
            'handlers': ['console'],
            'propagate': False,
         },
```

```
            'keystoneclient': {
                'handlers': ['console'],
                'propagate': False,
            },
            'nose.plugins.manager': {
                'handlers': ['console'],
                'propagate': False,
            }
        }
}
DATABASES = {
'default': {
'ENGINE': 'django.db.backends.mysql',
'NAME': 'dash',
'USER': 'dash',
'PASSWORD': 'dashpass',
'HOST': '192.168.69.1',
'default-character-set': 'utf8'
},
}
```

Run the manage.py syncdb command to initialize the database.

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

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

```
# /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
```