TELECOM SUDPARIS

OpenStack Essex Understand & Install Guide

Written and tested by Bilel Msekni (bilel.msekni@telecom-sudparis.eu)

Wireless Networks and Multimedia Services Department (RS2M)

05/07/2012

Under the supervision of:

Mr Djamal ZEGHLACHE

Mr Houssem MEDHIOUB

Table of content

What's OpenStack?

OpenStack is a number of open source components that form together a cloud solution. This solution can be used by enterprises/service providers to run their cloud compute and storage infrastructure. NASA and Rackspace were the initiators of this project, which was considered as one of the fastest emerging open source projects. Big names join the efforts of developing this cloud solution like IBM, Dell, Canonical etc.

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 five main service families under OpenStack

- Compute Infrastructure (Nova)
- Storage Infrastructure (Swift)
- Imaging Service (Glance)
- Identity management (Keystone)
- Network management (Quantum)

I have to say that installing an immature Open Source project is always a nasty thing to do, but I never shy away from a challenge. Even if I chose to work with the most stable version of OS but it's still not going to be as easy as you think.

Requirements and results

Let's start with what you should have:

- Ubuntu 11.04 (64bits)
- Kernel 2.6.38-13-generic (*uname -r* will let you know what exactly got)
- Reliable and fast internet connection

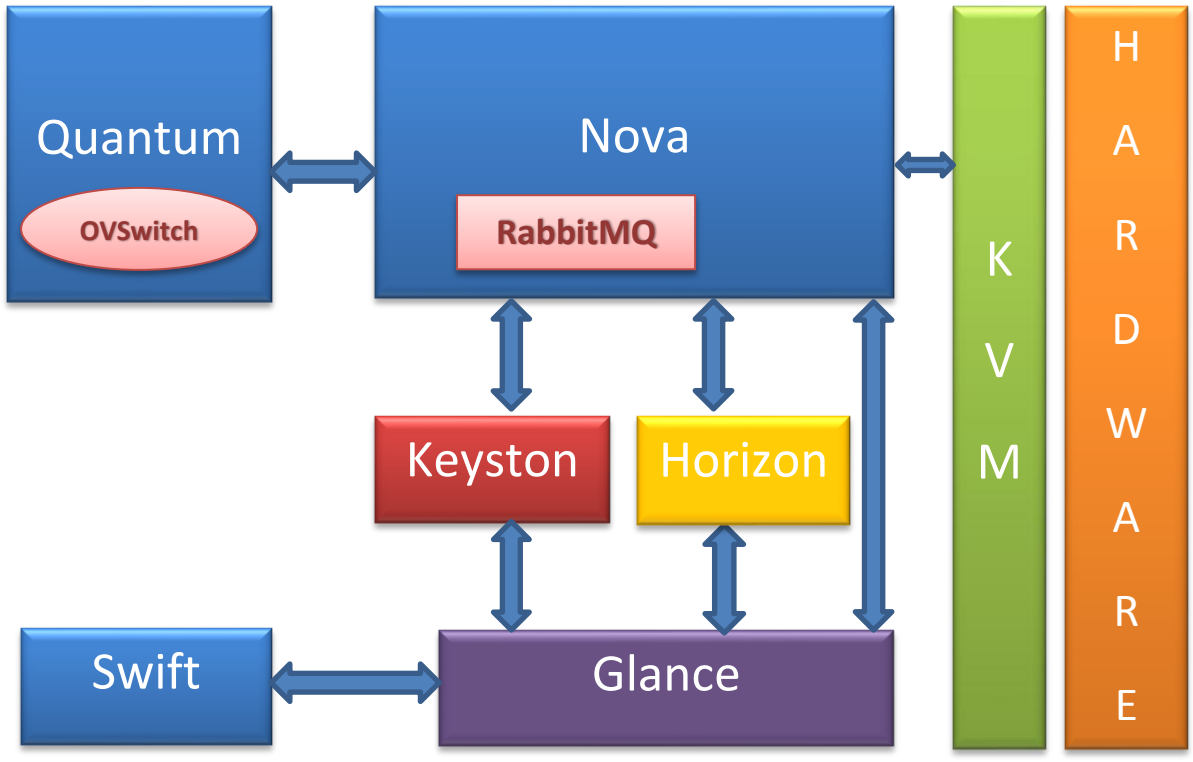
This is a one machine deployment operation so it will put everything from the database server, compute node, controller node etc on one host.

It is very recommended that you also have a good background with Ubuntu command lines tools because you are definitely going to need it.

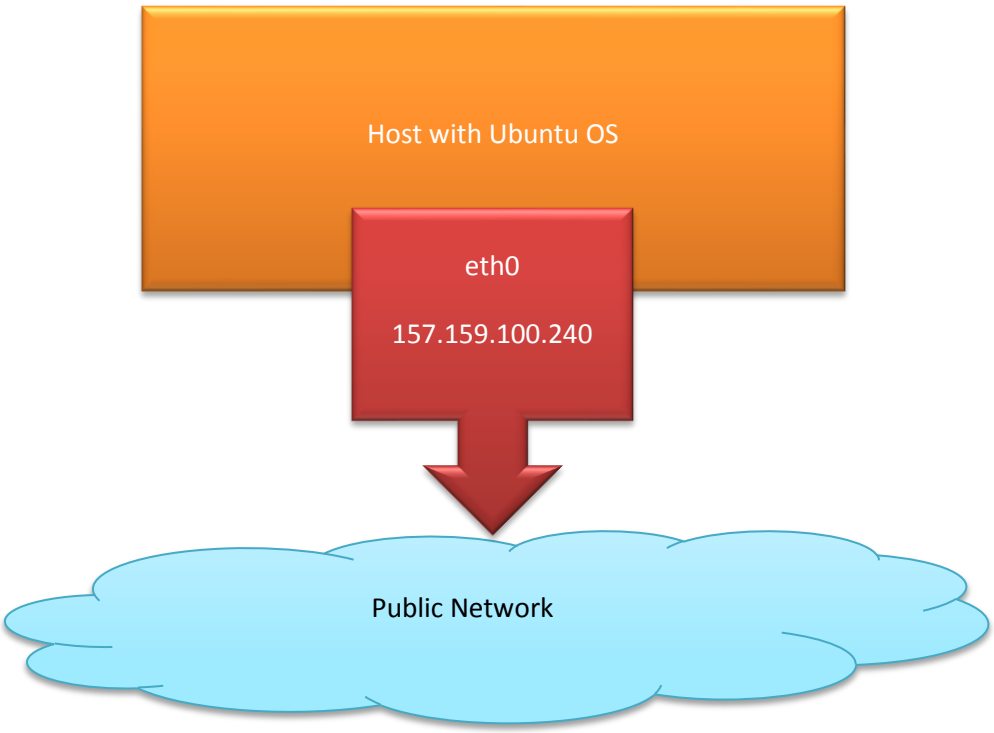
There are many guides now that show you how to install OpenStack but the reason to choose this guide is because it is OS independent since it directly downloads codes and installs it giving you freedom and more control over your future cloud system. Furthermore, most guides focus on OpenStack components and never mention anything about how to install the hypervisor or how to manage the network. Finally, what's the point of copy/paste commands without understanding what are you doing?! This is why, I will brief every step I make so that you can install and understand OpenStack at the same time.

I will try to keep it simple and shed some light on dark corners, it's not easy but what you will get and especially learn is for sure worth it.

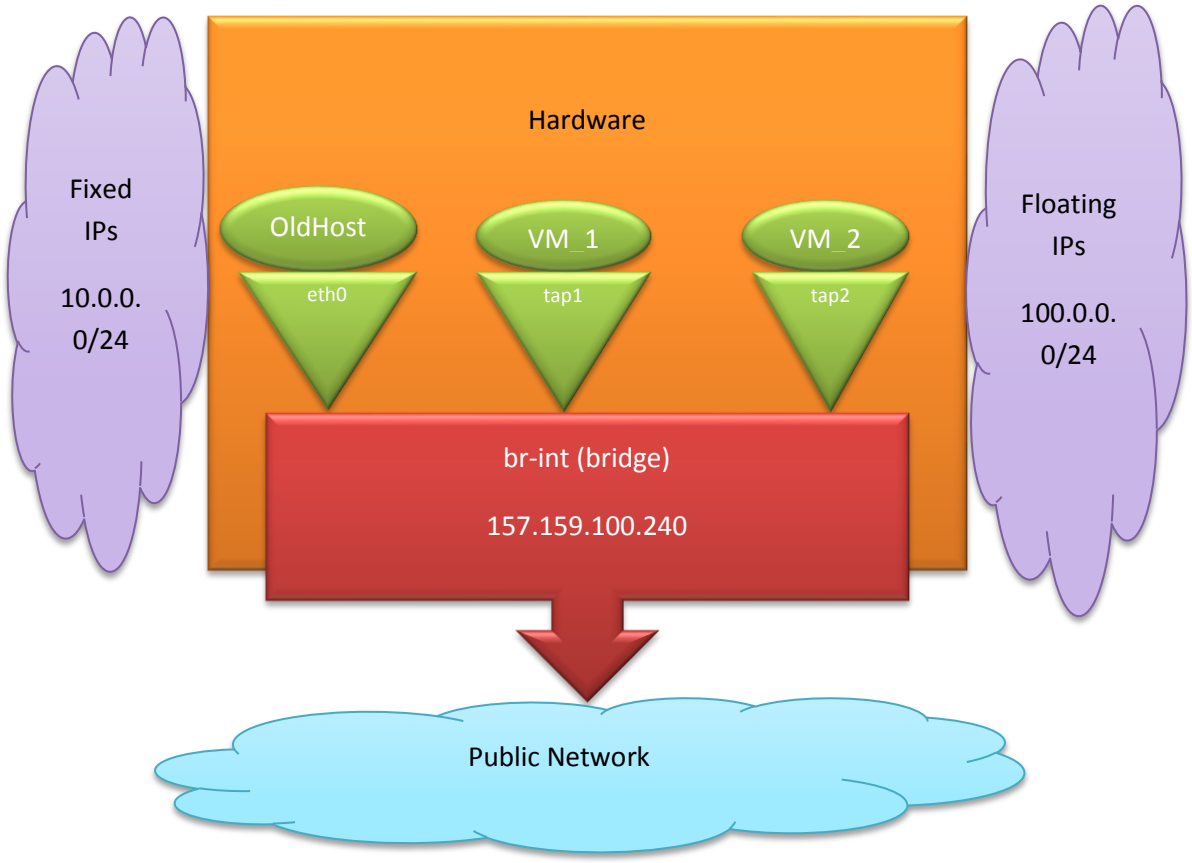
This is what we will end up with:



if we look at the network side, we should be starting with this:



and ending up with this:



Setting up the environment

First thing we will be doing is to get prepared. Start with a full update and don't forget to be a super user ***"sudo"*** before engaging any action:

```
apt-get update
apt-get dist-upgrade
apt-get autoremove
```

Synchronizing between nodes is very important. Install the NTP server on your machine and make sure it is the team leader by replacing two of the *ntp.conf* lines (see 2nd command below).

```
sudo apt-get install -y ntp
sudo 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.conf127.127.1.0\nfudge 127.127.1.0 stratum 10/g' /etc/ntp.conf
```

Finally use this command to link any client node to your controller:

```
ntpdate 'Put your controllernode IP here'
hwclock -w
```

We also need to setup a mysql server: During the install you will be prompted to create a password for the mysql server administrator. Make sure you don't forget it.

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

Now change the bind-address from localhost 127.0.0.1 to any 0.0.0.0 and then restart the service.

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

OpenStack Identity management: Keystone

The first component to install is Keystone. Keystone is a new player introduced to OpenStack team and its role is to provide authentication services and manage services concepts like accounts, endpoints, tokens etc. Think of it as the airport reception booth that will check your passport and then give you the permission to enter.

We have some dependencies to install, I know it is lame but what can you do about it:

```
apt-get install python-setuptools python-dev build-essential
apt-get install libxslt1-dev
apt-get install libxml2-dev
apt-get install python2.*-dev [ change * with your python version]
```

First thing to do is download the *.tar.gz* files from these links then extract them

```
cd Openstack_Essex/Keystone

wget https://launchpadlibrarian.net/100179978/python-keystoneclient-2012.1.tar.gz

wget https://launchpadlibrarian.net/100179831/keystone-2012.1.tar.gz

Tar -xzvf python-keystoneclient-2012.1.tar.gz

Tar -xzvf keystone-2012.1.tar.gz
```

Now we install Keystone the traditional way

```
cd keystone-2012.1

python setup.py install install_log.txt

cd ../python-keystoneclient

python setup.py install install_log.txt
```

To run properly, our keystone will need a database and we shall give it what it wants.

```
mysql -u root -p
#type your mysql password
mysql > CREATE DATABASE KeystoneDB;

mysql > GRANT ALL ON KeystoneDB.* TO 'keystoneUser'@'%' IDENTIFIED BY
'yourpassword';
#keystoneUser will be the administrator of the KeystoneDB
```

Finally we notify keystone of its new toy by replacing the connection string in its configuration file.

```
mysql://keystoneUser:yourpassword@localhost/KeystoneDB
```

We also need to modify the driver attribute under the catalog section:

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

Now, there are two main files in the bin directory of keystone-2012.1
1- keystone-manage: had a lot of potentials when it was in the diablo version but now its functions are limited to synchronizing the database or getting legacy data (See more here)

2- keystone-all: this is the spark to start the identity service, without, it it's like having a car without the key to start it.

Let's see how we can use these bin files:

Our database must be prepared for use so:

```
cd keystone-2012.1

bin/keystone-manage db_sync
```

If you get no response, then you are in the right way. Finally, to start keystone, all we need to type is this command:

```
bin/keystone-all
```

Note that in case a successful start, you will be getting a resume of all the started services.

To be sure, you can type *netstat -ntl* and look for the ports 5000 and 35357, they should be in the listening mode.

Like I said, in the previous version we used to call upon *keystone-manage* to populate our database but now all we need is keystone word and some options. I have written a script that will make your job easier but before the Ctrl+C / Ctrl+V, please try to read the comments so that you can understand what the script is really doing.

```
#!/usr/bin/env bash

#This file is inspired from the original DevStack keystone_data.sh script.

#Copy Rights to Msekni Bilel (bilel.msekni@telecom-sudparis.eu)

#The story here is that two tenants wil be created. AdminTenant will be dedicated to the keystone #adminUser and the other ServiceTenant will be for services such as glance, nova, quantum and #swift.

#SERVICE_ENDPOINT and SERVICE_TOKEN values are found in the DEFAULT section of the #keystone.conf (Don't forget to change to your IP address and put your own passwords).

export SERVICE_ENDPOINT=http://157.159.100.240:35357/v2.0

export SERVICE_TOKEN=ADMIN

#A little function to get the ids of the categories created

function get_id () {echo `"$@" | grep ' id ' | awk '{print $4}'`}

# Tenants

ADMIN_TENANT=$(get_id keystone tenant-create --name adminTenant)

SERVICE_TENANT=$(get_id keystone tenant-create --name serviceTenant)

# Users (Note a user for each service)

ADMIN_USER=$(get_id keystone user-create --name adminUser --pass "****" --email admin@example.com)

GLANCE_USER=$(get_id keystone user-create --name glanceUser --pass "****" --tenant_id $SERVICE_TENANT --email glance@example.com)

NOVA_USER=$(get_id keystone user-create --name novaUser --pass "****" --tenant_id $SERVICE_TENANT --email=nova@example.com)

SWIFT_USER=$(get_id keystone user-create --name swiftUser --pass "****" --tenant_id $SERVICE_TENANT --email swift@example.com)

QUANTUM_USER=$(get_id keystone user-create --name quantumUser --pass "****" --tenant_id $SERVICE_TENANT --email quantum@example.com)
```

```
# Roles (KeystoneAdmin and KeystoneServiceAdmin are very important for safe functioning)

ADMIN_ROLE=$(get_id keystone role-create --name admin)

MEMBER_ROLE=$(get_id keystone role-create --name Member)

KEYSTONEADMIN_ROLE=$(get_id keystone role-create --name KeystoneAdmin)

KEYSTONESERVICE_ROLE=$(get_id keystone role-create --name KeystoneServiceAdmin)

# Add Roles to Users in Tenants

keystone user-role-add --user $ADMIN_USER --role $ADMIN_ROLE --tenant_id $ADMIN_TENANT

keystone user-role-add --user $ADMIN_USER --role $KEYSTONEADMIN_ROLE --tenant_id $ADMIN_TENANT

keystone user-role-add --user $ADMIN_USER --role $KEYSTONESERVICE_ROLE --tenant_id $ADMIN_TENANT

keystone user-role-add --tenant_id $SERVICE_TENANT --user $NOVA_USER --role $ADMIN_ROLE

keystone user-role-add --tenant_id $SERVICE_TENANT --user $GLANCE_USER --role $ADMIN_ROLE

keystone user-role-add --tenant_id $SERVICE_TENANT --user $SWIFT_USER --role $ADMIN_ROLE

keystone user-role-add --tenant_id $SERVICE_TENANT --user $QUANTUM_USER --role $ADMIN_ROLE

# Services (Here we add new services that will be available later like Horizon)

NOVA_SERVICE=$(get_id keystone service-create --name nova --type compute --description "Nova Compute Service")

VOLUME_SERVICE=$(get_id keystone service-create --name "nova-volume" --type volume --description "Nova Volume Service")

GLANCE_SERVICE=$(get_id keystone service-create --name glance --type image --description "Glance Image Service")

KEYSTONE_SERVICE=$(get_id keystone service-create --name keystone --type identity --description "Keystone Identity Service")

EC2_SERVICE=$(get_id keystone service-create --name ec2 --type=ec2 --description=”EC2 Compatibility Layer”)

keystone service-create --name swift --type "object-store" --description "Swift Service"

keystone service-create --name quantum --type network --description "Quantum Service"
```

```
#Endpoints (Swift/Quantum didn't get endpoints, you will have to do it manually)


keystone endpoint-create  --region RegionOne --service_id $NOVA_SERVICE --
publicurl 'http://157.159.100.240:8774/v1.1/$(tenant_id)s' --adminurl
'http://157.159.100.240:8774/v1.1/$(tenant_id)s' --internalurl
'http://157.159.100.240:8774/v1.1/$(tenant_id)s'

keystone endpoint-create  --region RegionOne --service_id $VOLUME_SERVICE --
publicurl 'http://157.159.100.240:8776/v1/$(tenant_id)s' --adminurl
'http://157.159.100.240:8776/v1/$(tenant_id)s' --internalurl
'http://157.159.100.240:8776/v1/$(tenant_id)s'


keystone endpoint-create  --region RegionOne --service_id $GLANCE_SERVICE --
publicurl http://157.159.100.240:9292/v1 --adminurl http://157.159.100.240:9292/v1 --
internalurl http://157.159.100.240:9292/v1


keystone endpoint-create  --region RegionOne --service_id $KEYSTONE_SERVICE --
publicurl http://157.159.100.240:5000/v2.0 --adminurl
http://157.159.100.240:35357/v2.0 --internalurl http://157.159.100.240:35357/v2.0


keystone endpoint-create  --region RegionOne --service_id $EC2_SERVICE --
publicurl http://157.159.100.240:8773/services/Cloud --adminurl
http://157.159.100.240:8773/services/Admin --internalurl http://157.159.100.240
http://157.159.100.240:8773/services/Cloud
```

Well, thanks to this script, a lot of time has been saved. You can still verify the results using the commands like keystone user-list (see more here)

Now, if you are still unsure of the stability of your keystone, you can always count on curl to inspect for you the situation:

```
sudo apt-get install curl

curl -d '{"auth":{"passwordCredentials":{"username": "AminUser", "password": "*****"}}}' -H
"Content-type: application/json" http://localhost:35357/v2.0/tokens
```

This should give a result that looks like this:

```
{
    "version":{
        "id":"v2.0",
        "status":"beta",
        "updated":"2011-11-19T00:00:00Z",
        "links":[
            {
                "rel":"self",
                "href":"http://127.0.0.1:35357/v2.0/"
            },
            {
                "rel":"describedby",
                "type":"text/html",
                "href":"http://docs.openstack.org/api/openstack-identity-service/2.0/content/"
.
..
...
```


Congratulations, you have installed Keystone successfully. It's a very important part because every upcoming service is going to be related to keystone since it is the authentication manager. Don't forget to start it with keystone-all before moving to the next part.

OpenStack Image Store: Glance

Glance is the Image store of OpenStack. It's like those vending machines except that you won't find chips or Pepsis but instead virtual machine images that you can add, register and of course retrieve.

VM images are stored at different locations from simple file systems to object-storage systems giving glance high availability, fault tolerance and many other features.

```
cd Openstack_Essex/Glance
wget https://launchpadlibrarian.net/100179189/glance-2012.1.tar.gz
```

Extract and install:

```
Tar -xzvf glance-2012.1.tar.gz
cd glance-2012.1
python setup.py install
```

Now we prepare a great welcome for glance by installing what it will need:

First, this package dependency

```
easy_install -U iso8601
apt-get install python-boto
```

and then a database:

```
mysql -u root -p
#type your mysql password
mysql > CREATE DATABASE GlanceDB;
#note the confirmation
mysql > GRANT ALL ON GlanceDB.* TO 'glanceUser'@'%' IDENTIFIED BY 'yourpassword';
#glanceUser will be the administrator of the GlanceDB
```

Actually there is two parts here: the first one concerns the glance api and the second one is about the glance registry.

We start with the glance-api where we replace the admin_* values under *[filter:authtoken]* in the *glance-api-paste.ini file*.

```
[filter:authtoken]
admin_tenant_name = serviceTenant
admin_user = glanceUser
admin_password = ******
```

We also modify [pipeline:glance-api] section from :

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

To

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

Now we move to the glance registry part

We are going to do almost the same thing: Update */etc/glance/glance-registry-paste.ini*, configure the admin_* vaules under *[filter:authtoken]*

```
[filter:authtoken]
admin_tenant_name = serviceTenant
admin_user = glanceUser
admin_password = ******
```

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

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

And don't forget to modify the *sql_connection*:

```
mysql://glanceUser:yourpassword@localhost/GlanceDB
```

At this point, Glance is installed. To run it, perform the following:

```
cd Glance/glance-2012.1

glance-manage db_sync

glance-registry

glance-api
```

Hint_1: If somehow the glance-api complains about a missing *api.log*, just create a file in */var/log/glance* and name it *api.log*.

Hint_2: Don't mind the warning upon the glance-registry start, they are so far harmless. Likewise, to be certain that it is working fine check if the ports 9292 (api) and 9191(registry) are in listening mode.

How about we try uploading an image to the store: First download an image:

```
cd OpenStack/Tools

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
```

Now, upload the *kernel* with this command:

```
glance --os_username=adminUser --os_password=adminUser --os_tenant_name=adminTenant --os_auth_url=http://127.0.0.1:5000/v2.0 add name="tty-linux-ramdisk" disk_format=ari container_format=ari < ttylinux-uec-amd64-12.1_2.6.35-22_1-loader
```

Moreover, the *initrd* with this command:

```
glance --os_username=adminUser --os_password=adminUser --os_tenant_name=adminTenant --os_auth_url=http://127.0.0.1:5000/v2.0 add name=tty-linux-kernel disk_format=aki container_format=aki < ttylinux-uec-amd64-12.1_2.6.35-22_1-vmlinuz
```

Finally the image:

```
glance --os_username=adminUser --os_password=adminUser --
os_tenant_name=adminTenant --os_auth_url=http://127.0.0.1:5000/v2.0 add name="tty-linux"
disk_format=ami container_format=ami kernel_id=”put the result of the kernel upload here”
ramdisk_id=”put the result of the initrd upload here”  < ttylinux-uec-amd64-12.1_2.6.35-
22_1.img
```

To list what you have been doing, just use this command line:

```
glance --os_username=adminUser --os_password=adminUser --
os_tenant_name=adminTenant --os_auth_url=http://127.0.0.1:5000/v2.0 index
```

Anyway, I think you know by now how to upload more images to your store but don’t forget to start both services ***glance-api*** and ***glance-registry***. Well, it is two out of six, hung on now because there is still a lot to do.

OpenStack compute infrastructure: Nova

Coming now to the corner stone of OpenStack: It's the compute infrastructure also known as nova.

It manages all the compute resources, networking, authorization, and scalability needs but does not provide any virtualization capabilities by itself; instead, it uses *libvirt* APIs to interact with the supported hypervisors.

In our case, we will be using three tools to support the nova fabric in providing the appropriate service:

- RabbitMQ
- Quantum
- KVM

Let's start with the easiest component which is the RabbitMQ: It's a messaging queue server where the requests filed to the Nova API will be stored.

```
apt-get install erlang #It's a needed dependency

cd OpenStack_Essex/Tools

wget http://www.rabbitmq.com/releases/rabbitmq-server/v2.8.1/rabbitmq-server_2.8.1-1_all.deb

dpkg -i rabbitmq-server_2.8.1-1_all.deb #This will install the newest version of the RabbitMQ Server
```

Depending on your deployment mode, switch the starting address of the rabbitMQ from ':::' to '127.0.0.1'. To do so, proceed with the following:

```
nano /etc/rabbitmq/rabbitmq.config

[{rabbit,  [{tcp_listeners,  [{"127.0.0.1",5672}] }] } ]. #Paste this line

#Exit and SAVE

service start rabbitmq-server
```

Moving now to Quantum which before touching anything, I would like to explain what it really is:

It's the last project announced by OpenStack, it aims to provide network connectivity between devices managed by other OpenStack services. (See more here)

I believe the role of quantum is still unobvious, but as we go further in linking it to Nova, It will get clearer to you why we need such a new project.

First the dependencies, I think you are tired of this story but it is just an endless one:

```
apt-get install python-nose python-webtest python-sqlalchemy python-eventlet
apt-get install lvm2 iscsitarget
```

```
cd Openstack_Essex/Quantum
wget https://launchpadlibrarian.net/100184015/quantum-2012.1.tar.gz
wget https://launchpadlibrarian.net/100184093/python-quantumclient-2012.1.tar.gz
tar -xzvf quantum-2012.tar.gz
cd quantum*
python setup.py install
cd ../
tar -xzvf python-quantumclient-2012.1.tar.gz
cd python-quantum*
python setup.py install
```

By now, your quantum is well installed and can be run through this command line quantum-server from the quantum-2012.1 directory. However, there are always some configurations that have to be done.

By default, the quantum uses a fake plugin which has to be replaced by a real one like cisco, OVswitch and many others. If you have that sharp look, you might have guessed where does quatum's strength comes from: it's the flexibility to change from one technology to another (Cisco, OVswitch or whatever). Moreover, it creates an abstraction that Nova will benefit from when it comes to managing networks for VMs.

In my case, I will be choosing the OVswitch pluging and these are the steps to plug it with the Quantum server.

```
#Create a database for Quantum (precisely OVswitch pluging)
mysql -u root -p
>CREATE DATABASE QuantumDB;
> GRANT ALL ON QuantumDB.* TO 'quantumUser'@'%' IDENTIFIED BY 'yourpassword';
```

Edit the pluging to take under consideration the database:

```
nano etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini

#Alter the connection_sql to :

mysql://novaUser:yourpassword@adresseIP:3306/ovs_QuantumDB #@IP belongs to the node

#containing the database and should be reachable by all compute nodes.
```

Now we plug Quantum with OVS:

```
nano etc/quantum/plugins.ini #and put this line

provider = quantum.plugins.openvswitch.ovs_quantum_plugin.OVSQuantumPlugin
```

Make sure the ***nova.conf*** used when running ***nova-network*** and ***nova-manage*** contains:

```
network_manager=nova.network.quantum.manager.QuantumManager

linuxnet_interface_driver=nova.network.linux_net.LinuxOVSInterfaceDriver
```

I am sure that you are saying it is not as complicated as I though it will be but the funny thing here is that it's going to get a bit nasty. As you have noticed, the linking between OVS and Quantum is easy but have you wondered where will OVS come from? Yep, it won't magically appear, it must be installed too.

```
cd OpenStack_Essex/Tools

wget http://openvswitch.org/releases/openvswitch-1.4.0.tar.gz

tar -xzvf opennvswitch-1.4.0.tar.gz

apt-get install pkg-config linux-libc-dev libtool

apt-get install openvswitch-datapath-dkms openvswitch-common openvswitch-switch
```

Let's install it now:

```
cd openvswitch-1.4.0

./configure --with-l26=/lib/modules/`uname -r`/build

make

make install

rmmod bridge

insmod datapath/linux/openvswitch_mod.ko

insmod datapath/linux/brcompat_mod.ko

modprobe nbd

#if it worked proceed to initialize the configurations with the two following commands
but if #it did not, get more details from the file INSTALL.Linux in the openvswitch
directory

mkdir -p /usr/local/etc/openvswitch

ovsdb-tool create /usr/local/etc/openvswitch/conf.db vswitchd/vswitch.ovsschema

ovsdb-server /usr/local/etc/openvswitch/conf.db --
remote=punix:/usr/local/var/run/openvswitch/db.sock --
remote=db:Open_vSwitch,manager_options --pidfile --detach

ovs-vsctl --no-wait init #Only the first time but it's harmless to run it anytime

ovs-vsctl --pidfile --detach

ovs-vswitchd --pidfile --detach

ovs-brcompatd --pidfile --detach
```

Each time you want to start OVS, check out the script I wrote down here.

Create an OVS bridge, to which all VMs will connect:

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

ovs-vsctl add-port br-int eth0

ifconfig eth0 0.0.0.0

dhclient br-int #So that it can get the IP address of our previous physical eth0.
```

Make sure that ***nova.conf*** used by the nova-compute service contains these three lines:

```
libvirt_ovs_bridge=br-int
libvirt_vif_type=ethernet
libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtOpenVswitchDriver
```

In the Essex version, quantum is still new and was not properly integrated with keystone. Although I have tried a lot but it always ends up with the same: Getting quantum to authenticate through keystone will cause you enormous trouble with nova later. I highly recommend skipping steps A&B but if you insist on configuring quantum to use keystone, edit pipeline: quantumapi_v1_0 and quantumapi_v1_1 sections of the quantum.conf file **[STEP A]**

```
#comment the line
pipeline = extensions quantumapiapp_v1_* # * can be 0 or 1 depending on the section
#uncomment the line
pipeline = authN extensions quantumapiapp_v1_*  #* can be 0 or 1 depending on the section
```

Next, go to the authN section and make it like this: **[STEP B]**

```
[filter:authN]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
auth_host = 127.0.0.1
auth_port = 35357
auth_protocol = http
auth_version = 2.0
auth_uri=http://127.0.0.1:5000/
admin_tenant_name = serviceTenant
admin_user = quantumUser
admin_password = quantumUser
auth_admin_user = adminUser
auth_admin_password = adminUser
```

Finally edit the ***bashrc*** file to take under consideration these variables:

```
export OS_TENANT_NAME=adminTenant
export OS_USERNAME=adminUser
export OS_PASSWORD=*****
export OS_TENANT_ID= %Put adminTenant's ID here
export OS_AUTH_URL=http://157.159.100.240:5000/v2.0
source ~/.bashrc #To load the variables
```

Last touch is to put a quantum agent on each nova compute node, of course agents will talk to ***ovs_database*** so make sure you specify the right bridge and database server address.

```
mkdir OpenStack_Essex/ovs_agent
cd Quantum/quantum-2012.1
cp plugins/openvswitch/agent/ovs_quantum_agent.py ../../Tools /ovs_agent
cd etc/quantum/plugins/openvswitch/ovs_quantum_plugin.ini ../../Tools/ovs_agent
# To start the agent, start the OVS first then type this command line:
python ovs_quantum_agent.py ovs_quantum_plugin.ini
```

Let's continue with the install of the KVM hypervisor. To those who don't know hypervisors, it is a hardware virtualization technique that consists on giving multiple operating systems the possibility to run in a single host thinking they have their own hardware infrastructure or it is actually hardware virtualization in reality.

Installing KVM is not hard but you should probably verify if your hardware supports virtualization:

```
egrep -c '(vmx|svm)' /proc/cpuinfo # 0 means NO and 1 or more means YES
# To install KVM use this command line
apt-get install qemu-kvm libvirt-bin ubuntu-vm-builder bridge-utils kvm-pxe
```

To verify your installation perform the following:

```
virsh -c qemu:///system list
#Getting an error means that you have problems. (Seek help here)
```

Well, KVM creates a virtual bridge called virbr0 but we don't need it so it must go:

```
virsh net-destroy default
virsh net-undefine default
```

Ouf, finally we are done with the tools and we move now to more important things a.K.a NOVA.

Normally, we would download the code from Launchpad but nova had undergone many modifications even after the release of the Essex stable release so we will get it from github this time

```
cd Openstack_Essex/Nova
wget https://nodeload.github.com/openstack/nova/zipball/stable/essex
wget https://launchpadlibrarian.net/100184649/python-novaclient-2012.1.tar.gz
```

Dependencies as always:

```
easy_install -U kombu
apt-get install python-netaddr python-lockfile python-libvirt python-cheetah tgt  python-memcache
mkdir /usr/local/lib/python2.7/dist-packages/nova-2012.1-py2.7.egg/instances
sed -i 's/false/true/g' /etc/default/iscsitarget
service iscsitarget start
pvcreate /dev/sda6
vgcreate nova-volumes /dev/sda6
nano /etc/sysctl.conf # Uncomment the line net.ipv4.ip_forward to enable IP forwarding
```

Install and configure the database, I don't think that you will find any trouble with that:

```
unzip nova-2012.1.tar.gz
cd nova-2012.1
python setup.py install
```

```
Tar -xzvf python-novaclient-2012.1.tar.gz
cd python-novaclient-2012.1
python setup.py install
mysql -u root -p
#type your mysql password
mysql > CREATE DATABASE NovaDB;
#note the confirmation
mysql > GRANT ALL ON NovaDB.* TO 'novaUser'@'%' IDENTIFIED BY 'yourpassword';
#Change the sql connection string in nova.conf to:
 mysql://novaUser:yourpassword@localhost/NovaDB
#novaUser will be the administrator of the NovaDB
nova-manage --config-file=etc/nova/nova.conf db sync
```

The trickiest part about Nova is its configuration file, this is my file. Please don't mind the comments because it was just my way to make the settings clearer.

```
[DEFAULT]
verbose=True
auth_strategy=keystone
allow_resize_to_same_host=True
logdir=/var/log/nova
dhcpbridge_flagfile=/home/rs2m/OpenStack_Essex/Nova/nova-
2012.1/etc/nova/nova.conf
dhcpbridge=/home/rs2m/OpenStack_Essex/Nova/nova-2012.1/bin/nova-dhcpbridge
####Hosts####
quantum_connection_host=0.0.0.0
rpc_backend=nova.rpc.impl_kombu
glance_api_servers=0.0.0.0:9292
```

######Files_and_links######

policy_file=/home/rs2m/OpenStack_Essex/Nova/nova-2012.1/etc/nova/policy.json

sql_connection=mysql://novaUser:novaUser@127.0.0.1/NovaDB

api_paste_config=/home/rs2m/OpenStack_Essex/Nova/nova-2012.1/etc/nova/api-paste.ini

scheduler_driver=nova.scheduler.simple.SimpleScheduler

######VNC######

vnc_enabled=true

vncserver_listen=157.159.100.240

vncserver_proxyclient_address=157.159.100.240

novncproxy_base_url=http://157.159.100.240:6080/vnc_auto.html

xvpvncproxy_base_url=http://157.159.100.240:6081/console

####Volume####

volume_group=nova-volumes

iscsi_ip_prefix=157.159.100

iscsi_helper=tgtadm

#######Network#######

linuxnet_interface_driver=nova.network.linux_net.LinuxOVSInterfaceDriver

network_manager=nova.network.quantum.manager.QuantumManager

quantum_use_dhcp=true

auto_assign_floating_ip=true

force_dhcp_release=True

#####Virtualization######

connection_type=libvirt

libvirt_type=kvm

libvirt_ovs_bridge=br-int

libvirt_vif_type=ethernet

libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtOpenVswitchDriver

libvirt_use_virtio_for_bridges=true

To start nova, you need to start all the components we have already installed:

- Keystone
- Glance (Api & Registry)
- Quantum
- OpenvSwitch
- Ovs_Agent

Once you do that, perform the following:

```
cd Openstack_Essex/Nova/nova-2012.1
nova-all --config-file=etc/nova/nova.conf
```

Verify the starting of all nova-components: compute, network, etc and they are all in good shape with this command line:

```
nova-manage --config-file=etc/nova/nova.conf service list
```

If you get all services [nova-compute, nova-cert, nova-volume, nova-scheduler, nova-network] with a happy smile in the state column then you did a great job otherwise you can check the *nova-all.log* file in the */var/log/nova* directory for more details about the problem.

I think you are pretty excited now to start your first virtual machine instance so what are we waiting for, let's do it:

Starting VM Through command lines:

1. Creating private network

New born instances need IP addresses to be reached so the first thing to do is to provide them with ones:

```
nova-manage --config-file=etc/nova/nova.conf network create --label = %name_for_Network --fixed_range_v4=10.0.0.0/24
```

Note that addresses created here are global but you can make it tenant specific adding the attribute --project_id

2. Creating public addresses

Private network can be reached only from the inside but to communicate with our VM through the internet, it must get a public IP address known as floating IP.

```
nova-manage --config-file=etc/nova/nova.conf floating create --ip_range=100.0.0.0/24 --interface=br-int
```

3. Create a certificate

Certificate will be used to secure the access to VMs. It's not mandatory but we are creating it in order to test the full capacities of nova:

```
nova keypair-add %NameforKeypair > %NameforKeypair.pem
```

4. Enable SSH and Ping

Some modifications have to be done so that we can ping and SSH VMs in the future and altering the security group rules will grant us what we want:

```
nova secgroup-list-rules default #List what rules are predefined already
nova secgroup-add-rule default tcp 80 80 0.0.0.0/0 #enable internet access
nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0 #enable ICMP (ping)
nova secgroup-add-rule default tcp 22 22 0.0.0.0/0 #enable SSH
```

5. Your first VM

We are almost set now to launch our first VM but before that here are some nova commands that you will find helpful:

```
nova list #shows all present VMs no matter what their states are
nova image-list #show all VM images that were uploaded or snapshotted
nova flavor-list #list all flavors
```

Now, the moment that we all have been waiting for:

```
nova boot --flavor %your_Flavor_ID --image %your_Image_ID --key_name %your_Keypair_Name %Name_for_VM
```

In a few moments, type again the **nova list** command to see the result. If the instance spawned

successfully that it will be active otherwise it will have an error state. To see more information about your instance, type this:

```
nova list %your_instance_ID
```

As you can notice, your instance has only a private IP address. To enable access from the internet, a public address must be bonded to the VM:

```
nova floating-ip-create #To create a floating-IP from the range with specified earlier
nova add-floating-ip %Instance_ID %floating_IP_address
```

Check if you can access your machines through ping and SSH:

```
ping %floating_IP_of_yourInstance
ssh -i yourkey.pem % floating_IP_of_yourInstance
```

Congratulations, you have successfully installed Nova and created a VM. I can say that the worst part is now behind us and now we go hunting in the horizon if you know what I mean.

OpenStack compute dashboard: Horizon

This component doesn't add much to the server side of OpenStack but it's a high priority to clients because you can't expect everyone to understand the command line or even bother using it.

The main goal of this feature is to ease the use of OS through an interactive beautiful interface.

We start as usual with dependencies:

```
pip install django-mailer
apt-get install apache2 libapache2-mod-wsgi memcached python-numpy
```

Prepare a database too:

```
mysql -u root -p
create database HorizonDB;
grant all on HorizonDB.* TO 'horizonUser'@'%' IDENTIFIED BY 'yourpassword';
```

Download and compile the code:

```
mkdir Horizon
cd Horizon
wget https://launchpadlibrarian.net/100180408/horizon-2012.1.tar.gz
tar -xzvf horizon-2012.1.tar.gz
cd horizon-2012.1
python setup.py install
```

Good, now with some configuration your dashboard should be up and running in no time.

Make a copy of the local_setting.py file and add then the values concerning the database:

```
cd openstack-dashboard/local/
cp local_settings.py.example local_settings.py
nano local_settings.py
#Paste these values:

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'HorizonDB',
        'USER': 'horizonUser',
        'PASSWORD': 'yourpassword',
        'HOST': 'localhost',
        'default-character-set': 'utf8'
    },}
# This depends on what you have what you want to enable
QUANTUM_ENABLED = False #Since there is still no integration between Horizon and Quantum so far so it's better to put it to False.
SWIFT_ENABLED = True
```

Now, I have detected a problem in the code of Horizon and unfortunately it can't be fixed from the outside because it needs an internal surgery: a.K.a code modification:

```
#The file to modify is named users.py:
nano horizon/users.py
# Look for this line 155
authd = api.tenant_list_for_token(self._request, token)
# Replace with this line
authd = api.tenant_list_for_token(self._request, token, endpoint_type="publicURL")
```

This bug was reported here but the commit of the fix was abandoned. I don't know why but without it your log will be crowded with Not Authorized lines so if you want to do it, it's on your own risk. Finally we synchronize the database and start the server:

```
./manage.py syncdb
./manage.py runserver %Specify the address of the server.
```

Just now open your browser and then type the URL: *http://Dashboard_address* and it will work like a charm. The steps for creating VMs are the same and even easier, but mind that Horizon doesn't have some features like creating networks or uploading images so it's still some time before you ditch the terminal.

By now, I assume that you all had enough of the terminal interface and want to explore more inside your VMs. Hopefully, the dashboard gives us a new toy to log into our instances directly from our browser which is called VNC. I don't want to keep you waiting anymore, let's setup our VNC and dive in.

First thing to do is install a noVNC server which we can get from the github:

```
cd OpenStack_Essex/Tools
mkdir noVNC
cd noVNC
git clone http://github.com/cloudbuilders/noVNC.git #Get inside the noVNC directory
utils/nova-novncproxy --flagfile=[path to nova.conf options file]
```

You can open your browser and type in the value of the novncproxy_base_url from the ***nova.conf*** file to make sure that the server runs fine.

Next, we need to start the nova-consoleauth so proceed with the following:

```
cd OpenStack_Essex/Nova/nova-2012.1
bin/nova-consoleauth --config-file=etc/nova/nova.conf
```

Finally, you have too ways of getting a view inside your machine.

Log into Horizon and go to the VNC console option specific to your VM or you can get a URL using terminal then paste it into your browser with the instructions written below.

```
nova get-vnc-console [VM_id] novnc
```

If somehow you are not satisfied with the dimension of the console, there is no need to worry because you can modify it to suit your screen perfectly.

```
nano horizon-2012.1/horizon/dashboards/nova/templates/nova/instances_and_volumes/instances

#Look for the parameters width and height then change their values.
```

Voila! We have done it, I guess we are still missing the Swing component so it's coming right up.

OpenStack object storage: Swift

The object storage service has been around since the beginning of the openstack dawn. It's stable, reliable and efficient. The role is simple: store any kind of data in huge clusters and assure their full availability as well as their security through replicated data (availability) and HTTPS (security).

Swift installing is not hard especially when you do a one machine install, let's just say it will be a quick walk in the park.

Dependencies:

```
apt-get install openssh-server xfsprogs
apt-get install python-netifaces python-xattr
```

Download and compile the code:

```
cd OpenStack_Essex/Swift
wget https://launchpadlibrarian.net/97851598/swift-1.4.8.tar.gz
cd swift-1.4.8
python setup.py install
```

So far this is well done. Next, we will be using a loopback device for storage so in order to configure it, proceed with the following:

```
mkdir /srv
dd if=/dev/zero of=/srv/swift-disk bs=1024 count=0 seek=1000000 #(modify seek to make a larger
#or smaller partition)
mkfs.xfs -i size=1024 /srv/swift-disk
#Edit /etc/fstab and add
/srv/swift-disk /mnt/sdb1 xfs loop,noatime,nodiratime,nobarrier,logbufs=8 0 0
```

```
mkdir /mnt/sdb1

mount /mnt/sdb1

mkdir /mnt/sdb1/1 /mnt/sdb1/2 /mnt/sdb1/3 /mnt/sdb1/4

chown <your-user-name>:<your-group-name> /mnt/sdb1/*

for x in {1..4}; do ln -s /mnt/sdb1/$x /srv/$x; done

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

chown -R <your-user-name>:<your-group-name> /etc/swift /srv/[1-4]/ /var/run/swift – Make
sure to include the trailing slash after /srv/[1-4]/
```

Add to ***/etc/rc.local*** (before the exit 0):

```
mkdir /var/run/swift

chown <your-user-name>:<your-group-name> /var/run/swift
```

Create the ***rsync.conf*** in the */etc/swift/*:

```
uid = <Your user name>

gid = <Your group name>

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
```

Edit the following line in **/etc/default/rsync**:

```
RSYNC_ENABLE=true
service rsync restart
```

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

```
[DEFAULT]
bind_port = 8080
user = <your-user-name>
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
user_test2_tester2 = testing2 .admin
user_test_tester3 = testing3
[filter:healthcheck]
use = egg:swift#healthcheck
[filter:cache]
use = egg:swift#memcache
```

Create */etc/swift/swift.conf*:

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

Create ***/etc/swift/account-server/x.conf***, mind that x ∈ {1,2,3,4}:

```
[DEFAULT]
devices = /srv/x/node
mount_check = false
bind_port = 60x2
user = <your-user-name>
log_facility = LOG_LOCAL(x+1)
[pipeline:main]
pipeline = account-server
[app:account-server]
use = egg:swift#account
[account-replicator]
vm_test_mode = yes
[account-auditor]
[account-reaper]
```

Create ***/etc/swift/container-server/x.conf***, mind that x ∈ {1, 2, 3, 4}:

```
[DEFAULT]
devices = /srv/x/node
mount_check = false
bind_port = 60x1
```

```
user = <your-user-name>
log_facility = LOG_LOCAL(x+1)
[pipeline:main]
pipeline = container-server
[app:container-server]
use = egg:swift#container
[container-replicator]
vm_test_mode = yes
[container-updater]
[container-auditor]
[container-sync]
```

Finally, create ***/etc/swift/object-server/x.conf***, mind that x ∈ {1, 2, 3, 4}:

```
[DEFAULT]
devices = /srv/x/node
mount_check = false
bind_port = 60x0
user = <your-user-name>
log_facility = LOG_LOCAL(x+1)
[pipeline:main]
pipeline = object-server
[app:object-server]
use = egg:swift#object
[object-replicator]
vm_test_mode = yes
[object-updater]
[object-auditor]
```

To make our job even faster and easier, we will write down some scripts:

Create ***/bin/resetswift***:

```
#!/bin/bash
swift-init all stop
find /var/log/swift -type f -exec rm -f {} \;
sudo umount /mnt/sdb1
sudo mkfs.xfs -f -i size=1024  /srv/swift-disk
sudo mount /mnt/sdb1
sudo mkdir /mnt/sdb1/1 /mnt/sdb1/2 /mnt/sdb1/3 /mnt/sdb1/4
sudo chown <your-user-name>:<your-group-name> /mnt/sdb1/*
mkdir -p /srv/1/node/sdb1 /srv/2/node/sdb2 /srv/3/node/sdb3 /srv/4/node/sdb4
sudo service rsyslog restart
sudo service memcached restart
```

Create ***/bin/remakerings***

```
#!/bin/bash
cd /etc/swift
rm -f *.builder *.ring.gz backups/*.builder backups/*.ring.gz
swift-ring-builder object.builder create 18 3 1
swift-ring-builder object.builder add z1-127.0.0.1:6010/sdb1 1
swift-ring-builder object.builder add z2-127.0.0.1:6020/sdb2 1
swift-ring-builder object.builder add z3-127.0.0.1:6030/sdb3 1
swift-ring-builder object.builder add z4-127.0.0.1:6040/sdb4 1
swift-ring-builder object.builder rebalance
swift-ring-builder container.builder create 18 3 1
swift-ring-builder container.builder add z1-127.0.0.1:6011/sdb1 1
```

```
swift-ring-builder container.builder add z3-127.0.0.1:6031/sdb3 1
swift-ring-builder container.builder add z4-127.0.0.1:6041/sdb4 1
swift-ring-builder container.builder rebalance
swift-ring-builder account.builder create 18 3 1
swift-ring-builder account.builder add z1-127.0.0.1:6012/sdb1 1
swift-ring-builder account.builder add z2-127.0.0.1:6022/sdb2 1
swift-ring-builder account.builder add z3-127.0.0.1:6032/sdb3 1
swift-ring-builder account.builder add z4-127.0.0.1:6042/sdb4 1
swift-ring-builder account.builder rebalance
```

Create ***/bin/startmain***:

```
#!/bin/bash
swift-init main start
```

Create ***/bin/startrest***:

```
#!/bin/bash
swift-init rest start
```

Don't forget to add the execute privilege to our new guests with **chmod +x /bin/***

Now, just type these commands and watch the show:

```
remakerings
startmain
curl -v -H 'X-Storage-User: test:tester' -H 'X-Storage-Pass: testing'
http://127.0.0.1:8080/auth/v1.0 #Get an X-Storage-Url and X-Auth-Token:
#Check that you can GET account:
curl -v -H 'X-Auth-Token: <token-from-x-auth-token-above>' <url-from-x-storage-url-above>
```

Finally check the state of Swift:

```
swift -A http://127.0.0.1:8080/auth/v1.0 -U test:tester -K testing stat
```

Finally, it's about the end. I tried to make this guide as clear as possible and I will work on enhancing it. I hope it would help you find your way into the cloud world as well as a motive to contribute also to the documentation of OpenStack. I want to thank everyone who supported me and helped me out to overcome the numerous bugs I found.

The one node install was fun but now I have to move on for a new guide about installing OpenStack in a multi-nodes environment.

If you like my guide, want to add something or ask a question, please be guests.

References

- http://docs.openstack.org/trunk/openstack-compute/install/content/ch_installing-openstack-overview.html
- http://keystone.openstack.org/
- http://glance.openstack.org/
- http://nova.openstack.org/
- http://swift.openstack.org/
- http://openvswitch.org/openstack/documentation/
- http://openvswitch.org/cgi-bin/gitweb.cgi?p=openvswitch;a=blob_plain;f=INSTALL.Linux;hb=HEAD
- https://help.ubuntu.com/community/KVM/Installation
- http://wiki.openstack.org/NovaConfigOptions