Install Your Own OpenStack Cloud

Essex Edition

Eric Dodémont (dodeeric)
eric.dodemont@skynet.be

Version:

1.03 – 16 May 2012

Electronic version:

http://dodeeric.s3-website-eu-west-1.amazonaws.com/PDF/Install-Your-Own-OpenStack-Cloud-Essex-Edition.pdf

Audience:

This document is for system administrators having experience in Linux and virtualization. Basic knowledge of cloud computing and OpenStack will help.

Table Of Content

Introduction

OpenStack is an open source IaaS cloud computing platform (www.openstack.org) written in the Python programming language. In this document, I will describe in detail the installation, configuration and use of my OpenStack cloud. You can use it to install your own private or public cloud. I try to not use scripts for the installation and configuration to show clearly all the steps to follow. This is for clarity.

Architecture Description

We will install OpenStack on two physical servers (see topology n° 1). We will also give some explanations on how to install OpenStack on more than two physical servers (see topology n° 2).

The node1 will be:

— The **cloud controller node** (running nova-api, nova-scheduler, nova-network, glance, horizon, MySQL, and RabbitMQ).
— A **compute node** (running nova-compute and KVM).
— A **storage node** (running nova-volume and iSCSI).

The node2 will be:

— A **compute node** (running nova-compute and KVM).

It means that an instance can either be created on the node1 or the node2 (the nova-scheduler decides where to create it). It means also that if you deactivate the node2, you can still provision new instances. The node1 can run in stand-alone mode.

If you install nova-volume on the node2 as well, then nova-scheduler will decide where to create a new volume: on node1 or node2.

We will install Horizon (the dashboard web interface) which allows managing almost everything, including:

— In the project panel: instances, volumes, floating IPs, security groups, images, instance and volume snapshots.
— In the admin panel: instances, services, flavors, images, projects, users, and quotas.

In both panels, you also have status and usage information.

Software Versions

— Operating System: Linux Ubuntu Server version 12.04 (Precise), 64 bits.
— Cloud Computing: OpenStack version 2012.1 (Essex) including Nova, Glance, Keystone, and Horizon.

These are the different versions of OpenStack until now:

Code name	Version	Remark
Austin	2010.1	
Bexar	2011.1	
Cactus	2011.2	
Diablo	2011.3.1	
Essex	2012.1	
Folsom	2012.2	In development

Hardware Requirements

Number of hard disks (HDDs) and network cards (NICs) for the two nodes:

	HDD	NIC
node1	2 (sda & sdb)	2 (eth0 & eth1)
node2	1 (sda)	1 (eth1)

This procedure can also be used to install OpenStack on only one physical server with only one NIC and only one HDD.

Network Configuration

Network type: VLAN (VlanNetworkManager)

Eth0 (public/external network):

— Network 1: 192.168.1.0/24

Eth1 (private/internal networks):

— Network 2: 172.16.1.0/24 (OpenStack management and storage network)
— Network 3: 10.0.0.0/8 (OpenStack service network)

The **management network** is used for the communication between the OpenStack components, the MySQL DB server, and the RabbitMQ messaging server.

The **storage network** (volume network) is used for the iSCSI volume traffic between the storage node (volume node) and the compute nodes.

The **service network** is used for the instance fixed IPs and to create all the VLANs/subnets for the tenants. It is completely managed by OpenStack.

For a true public cloud, the **public network** should be a network with internet IP addresses (e.g. 194.78.33.0/28) and no NAT/PAT rules should be needed in the router (see topology n° 2). We do not have a public internet IP address range, and that is why we use NAT/PAT rules in the router to give access from the internet.

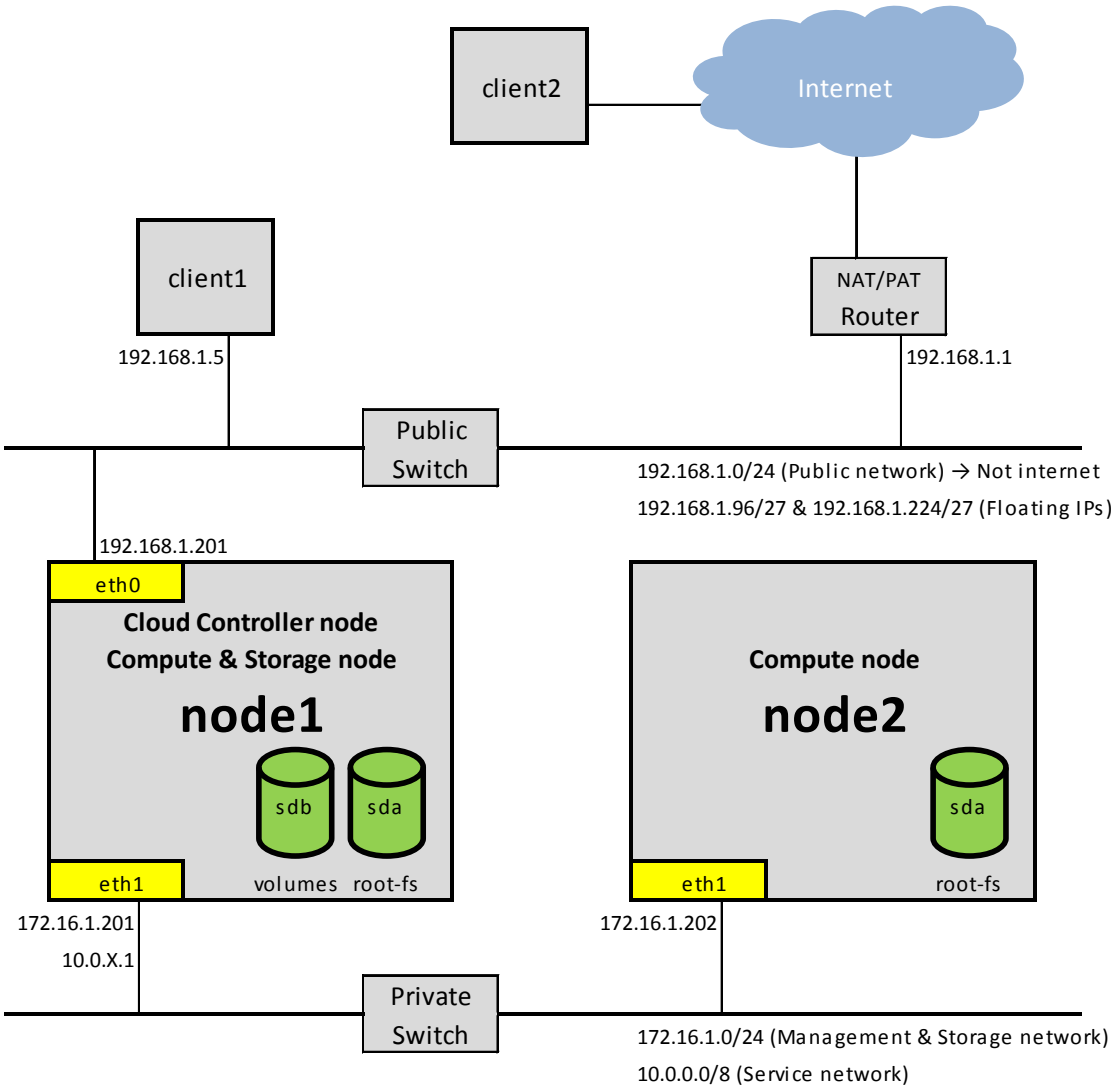
Ideally the **controller node** should not be a compute node (see topology n° 2). In our case, we run nova-compute on node1 only because we have only two physical servers.

The node1 (nova-network) is the **network node**: the floating IPs and the private default gateways IPs (10.0.X.1) are configured on it. The node1 acts as a router to be able to access the instances on the node1 or the node2.

Public and private IPs:

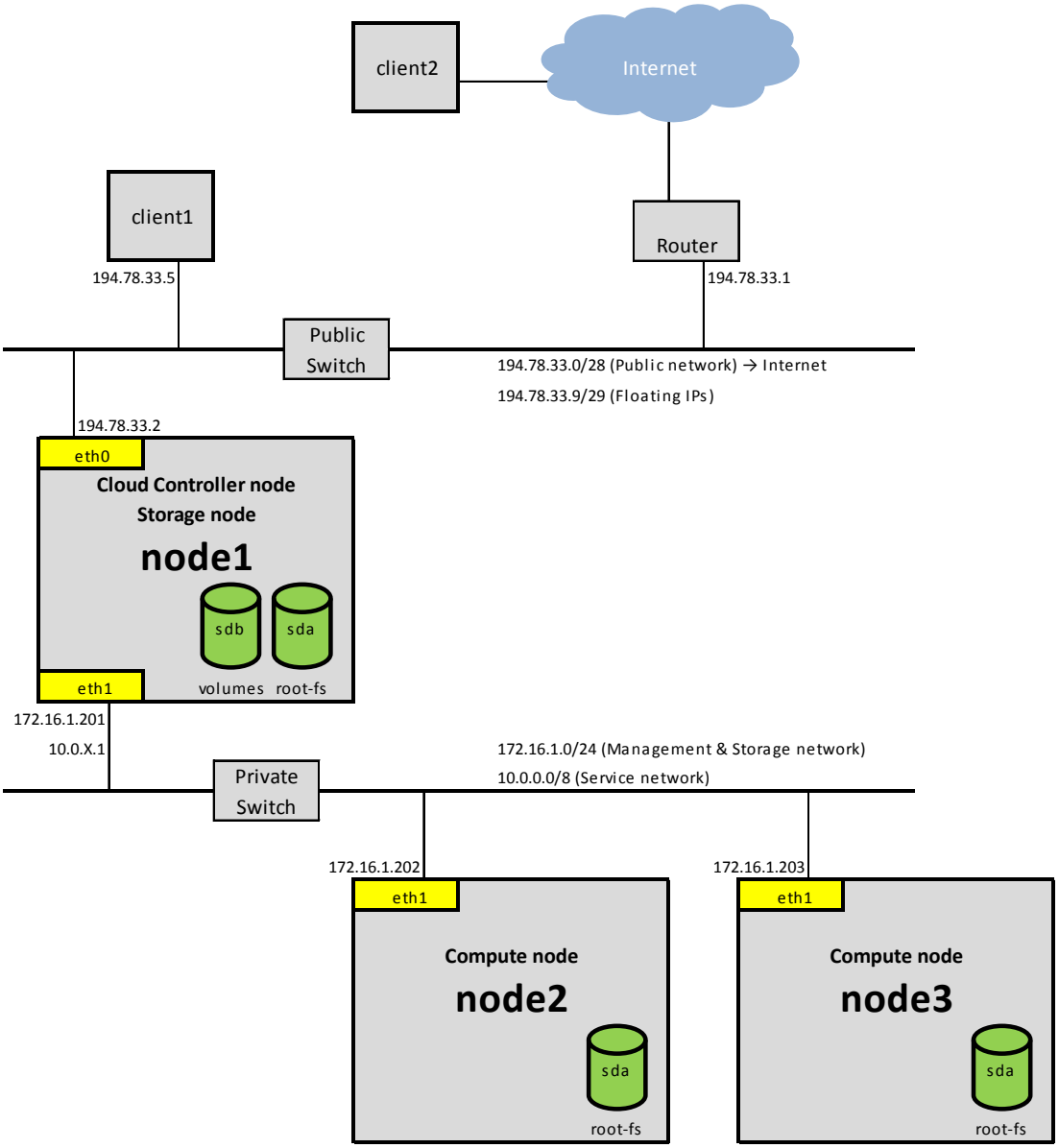
	Eth0	Eth1
node1	192.168.1.201	172.16.1.201
node2	-	172.16.1.202
client1	192.168.1.5	-

Topology n° 1 (the one we will describe in detail in this document):



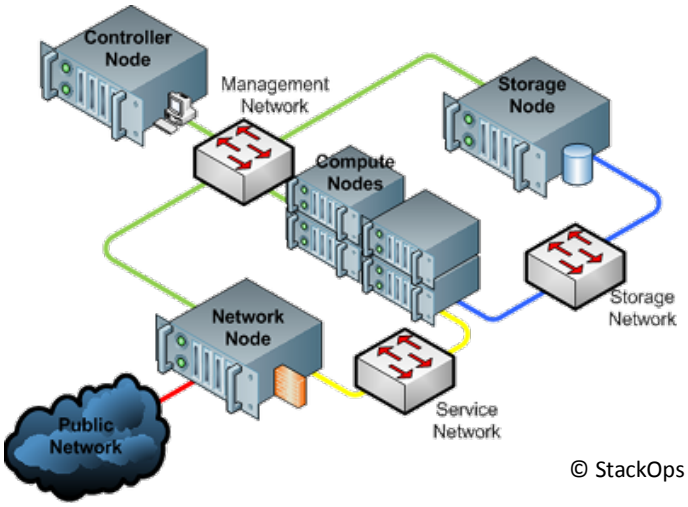
Topology n° 1: Two nodes (both are compute node). The public network does not have internet IP addresses (NAT/PAT rules needed in the router).

Topology n° 2 (we will give some information only on this topology):



Topology n° 2: Three nodes (node1 is not a compute node). The public network has internet IP addresses (NAT/PAT rules not needed in the router).

Please note that a complete private or public cloud installation should look like this:



In this topology, the **management network** and **storage network** are separated. The **controller node**, the **network node**, and the **storage node** are on separated physical hosts.

Such installation and configuration can be done via the **StackOps distribution**. Please visit www.stackops.com or www.stackops.org for more information.

Naming Conventions

Amazon EC2 and **OpenStack** naming conventions are sometime different. For example:

Amazon EC2	OpenStack
Elastic IP (EIP)	Floating IP (FIP)
Elastic Block Storage (EBS)	Volume (VOL)

I will try to use the OpenStack naming conventions as often as possible.

The following terms are considered as synonyms in this document:

- Node / Host / Server
- Instance / Virtual Machine (VM) / Guest / Container (if LXC is used)
- External network / Public network
- Internal network / Private network
- Floating IP / Elastic IP
- Volume / Elastic Block Storage
- Project / Tenant
- Nova components / Nova services

Notation Conventions

Commands to be launched as a:

— Cloud end-user (customer) are written like this:

Outside an instance (e.g. from **client1** or **client2**):

```
pdupond@client1:~$ nova list
```

Inside an instance:

```
pdupond@i1:~$ sudo fdisk -l
```

— Cloud administrator (admin) are written like this (e.g. from **node1** or **node2**):

```
root@node1:~# nova-manage service list
```

If it has to be launched on all nodes:

```
root@nodeX:~# nova-manage service list
```

Install your Cloud

Install node1

- Install RabbitMQ (the messaging/queuing server):

```
root@node1:~# aptitude install rabbitmq-server
```

- Install MySQL (the DB server):

In the following document, my passwords will be **123456**; please chose your own one.

```
root@node1:~# aptitude install mysql-server
```

Change the configuration to make MySQL reachable from the other nodes:

```
root@node1:~# sed -i 's/127.0.0.1/0.0.0.0/g' /etc/mysql/my.cnf
root@node1:~# service mysql restart
```

Nova, Glance, and Keystone will use the same MySQL server, but not the same DBs.

- Install Nova (the compute service):

Create the **nova** DB and **nova** username:

```
root@node1:~# mysql -uroot -p123456 -e 'CREATE DATABASE nova;'
root@node1:~# mysql -uroot -p123456 -e "GRANT ALL PRIVILEGES ON *.* TO 'nova'@'%'
    WITH GRANT OPTION;"
root@node1:~# mysql -uroot -p123456 -e "SET PASSWORD FOR 'nova'@'%' = PASSWORD('123456');"
```

Install all the Nova components:

Controller node:

```
root@node1:~# aptitude install nova-api nova-scheduler nova-network
```

Storage node:

```
root@node1:~# aptitude install nova-volume
```

Compute node:

```
root@node1:~# aptitude install nova-compute
```

KVM/Qemu and **libvirt** are automatically installed when installing nova-compute (see the package dependencies below).

Here is the list of dependencies of all the Nova components as found in the package's metadata. These packages are installed automatically.

- **nova-api**: nova-common, python, upstart-job.
- **nova-scheduler**: nova-common, python, upstart-job.
- **nova-network**: netcat, vlan, bridge-utils, dnsmasq-base, iputils-arping, dnsmasq-utils, nova-common, python2.7, upstart-job.
- **nova-compute**: lsb-base, nova-common, qemu-utils, kpartx, curl, parted, vlan, ebtables, gawk, iptables, open-iscsi, nova-compute-kvm, python2.7, upstart-job.
- **nova-compute-kvm**: nova-compute, python-libvirt, libvirt-bin, kvm.
- **nova-compute-lxc**: nova-compute, python-libvirt, libvirt-bin.
- **nova-volume**: nova-common, lvm2, tgt, python2.7, upstart-job.
- **python-nova**: python2.7, openssh-client, openssl, python-boto, python-m2crypto, python-pycurl, python-daemon, python-carrot, python-kombu, python-lockfile, python-gflags, python-libxml2, python-ldap, python-sqlalchemy, python-eventlet, python-routes, python-webob, python-cheetah, python-netaddr, python-paste, python-pastedeploy, python-tempita, python-migrate, python-glance, python-novaclient, python-simplejson, python-lxml, python-feedparser, python-xattr, python-suds, python-iso8601, sudo.
- **nova-common**: python-amqplib, python-nova, python, adduser.

See **appendix A** for a description of some of these packages.

If you want to use **LXC** (still experimental in Essex release) in place of **KVM** do this on all compute nodes:

```
root@nodeX:~# aptitude remove nova-compute-kvm
root@nodeX:~# aptitude install nova-compute-lxc
```

And adapt the configuration file:

```
root@nodeX:~# vi /etc/nova/nova.conf
```

To look like this:

```
libvirt_type=lxc
```

And restart the nova-compute service:

```
root@nodeX:~# service nova-compute restart
```

If you want to disable the compute node (nova-compute component) on the **node1** (see topology n° 2), do this:

```
root@node1:~# nova-manage service disable --host node1 --service nova-compute
```

The nova-compute will still run on node1, but the scheduler will no more try to boot instances on it.

- Install Glance (the image service):

Create the **glance** DB and **glance** username:

```
root@node1:~# mysql -uroot -p123456 -e 'CREATE DATABASE glance;'
root@node1:~# mysql -uroot -p123456 -e "GRANT ALL PRIVILEGES ON *.* TO 'glance'@'%'
    WITH GRANT OPTION;"
root@node1:~# mysql -uroot -p123456 -e "SET PASSWORD
    FOR 'glance'@'%' = PASSWORD('123456');"
```

Install the Glance components (api and registry):

```
root@node1:~# aptitude install glance
```

- Install Keystone (the identity service):

Create the **keystone** DB and **keystone** username:

```
root@node1:~# mysql -uroot -p123456 -e 'CREATE DATABASE keystone;'
root@node1:~# mysql -uroot -p123456 -e "GRANT ALL PRIVILEGES ON *.* TO 'keystone'@'%'
    WITH GRANT OPTION;"
root@node1:~# mysql -uroot -p123456 -e "SET PASSWORD
    FOR 'keystone'@'%' = PASSWORD('123456');"
```

Install the Keystone components:

```
root@node1:~# aptitude install keystone
```

Install node2 (and eventually more nodes)

- Install the Nova component:

```
root@node2:~# aptitude install nova-compute
```

If you have more than two nodes (see topology n° 2), install the other compute nodes like node2.

E.g. for node3:

```
root@node3:~# aptitude install nova-compute
```

Configure node1 and node2

On both nodes:

- Adapt the **hosts** file:

```
root@nodeX:~# vi /etc/hosts
```

To look like this:

```
127.0.0.1       localhost
192.168.1.201   node1pub
172.16.1.201    node1priv node1
172.16.1.202    node2priv node2
```

- Adapt the **interfaces** file:

For node1:

```
root@node1:~# vi /etc/network/interfaces
```

To look like this:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
    address 192.168.1.201
    netmask 255.255.255.0
    network 192.168.1.0
    broadcast 192.168.1.255
    gateway 192.168.1.1
    up iptables -t nat -A POSTROUTING -s 172.16.1.0/24 -o eth0 -j SNAT --to 192.168.1.201

auto eth1
iface eth1 inet static
```

```
    address 172.16.1.201
    netmask 255.255.255.0
    network 172.16.1.0
    broadcast 172.16.1.255
```

No need to configure manually **Ethernet bridges**: these bridges are automatically configured on the network node and the compute nodes when it is needed (there are one or more bridges per tenant).

The **iptables** rule tells node1 to change the source IPs to 192.168.1.201 (-j SNAT --to 192.168.1.201) for all eth0 outgoing packets (-o eth0) coming from network 172.16.1.0/24 (-s 172.16.1.0/24). **This is to allow the router to route packets to internet if coming from the management/storage network** (e.g. to install packages from the Ubuntu public repositories). If the public and management/storage networks would be the same (e.g. 192.168.1.0/24), this iptables rule would not be needed.

For node2:

```
root@node2:~# vi /etc/network/interfaces
```

To look like this:

```
auto lo
iface lo inet loopback

auto eth1
iface eth1 inet static
    address 172.16.1.202
    netmask 255.255.255.0
    network 172.16.1.0
    broadcast 172.16.1.255
    gateway 172.16.1.201
```

The node1 (172.16.1.201) is the default gateway for node2 (172.16.1.202).

On node1 only:

- To allow the node1 to act as a router:

```
root@node1:~# vi /etc/sysctl.conf
```

And uncomment this line:

```
net.ipv4.ip_forward=1
```

Reboot the server to take that change into account.

- Adapt the Nova, Glance, and Keystone configuration files:

Adapt the **xxx-conf** files:

Nova

```
root@node1:~# vi /etc/nova/nova.conf
```

To look like this:

```
[DEFAULT]

##### RabbitMQ #####
rabbit_host=node1

##### MySQL #####
sql_connection=mysql://nova:123456@node1/nova

##### nova-api #####
auth_strategy=keystone
```

```
cc_host=node1

##### nova-network #####
network_manager=nova.network.manager.VlanManager
public_interface=eth0
vlan_interface=eth1
network_host=node1
fixed_range=10.0.0.0/8
network_size=1024
dhcpbridge_flagfile=/etc/nova/nova.conf
dhcpbridge=/usr/bin/nova-dhcpbridge
force_dhcp_release=True
fixed_ip_disassociate_timeout=30
my_ip=172.16.1.201
routing_source_ip=192.168.1.201

##### nova-compute #####
connection_type=libvirt
libvirt_type=kvm
libvirt_use_virtio_for_bridges=True
use_cow_images=True
snapshot_image_format=qcow2

##### nova-volume #####
iscsi_ip_prefix=172.16.1.20
num_targets=100
iscsi_helper=tgtadm

##### glance #####
image_service=nova.image.glance.GlanceImageService
glance_api_servers=node1:9292

##### Misc #####
logdir=/var/log/nova
state_path=/var/lib/nova
lock_path=/var/lock/nova
root_helper=sudo nova-rootwrap
```

- The **my_ip** parameter (IP of the host) has to be specified or else it will be 192.168.1.201 (public IP) in place of 172.16.1.201 (private IP), and node2 cannot communicate with node1 via the public network.
- The **routing_source_ip** (default SNAT on node1) has to be specified or else it will be 172.16.1.201 (private IP) in place of 192.168.1.201 (public IP), and the router will not be able to route packets coming from the service network (10.0.0.0/8).
- If the public and management/storage networks would be the same (e.g. 192.168.1.0/24), these two parameters (**my_ip** and **routing_source_ip**) would not be needed (they both would be 192.168.1.201).

Some parameters in the **nova.conf** file are the default ones and then do not need to be put in the configuration file. But for clarity, I prefer them to be present.

Examples of some default parameters:

Parameter: **network_manager** (type of networking used on the service network used by the instances)

- **FlatManager**: One flat network for all tenants (no DHCP server).
- **FlatDHCPManager**: One flat network for all tenants (with a DHCP server).
- **VlanManager**: Default – The most sophisticated OpenStack network mode (at least one VLAN, one Ethernet bridge, one IP range subnet, and one DHCP server per tenant).

Parameter: **libvirt_type** (type of virtualization used on the compute nodes)

- **kvm**: Default – Linux Kernel-based Virtual Machine. CPU with hardware virtualization technology like Intel VT-x is required.

— **qemu**: You can use it if you install OpenStack in a VM or on a physical server without CPU with hardware virtualization technology like Intel VT-x. It is quite slow but can emulate other architecture than x86; for example the ARM architecture.
— **lxc**: LinuX Container. Linux kernel-based container (uses control groups and name spaces). Virtualization of the OS, not of the HW. It is similar to OpenVZ and Parallels Virtuozzo; in Solaris Unix, this virtualization technology is called "Solaris Zones"; in BSD UNIX, this virtualization technology is called "BSD Jail". Does not require CPU with hardware virtualization technology like Intel VT-x. Very efficient virtualization technology.

Virtualization which are supported by OpenStack:

— KVM (via libvirt)
— Qemu (via libvirt)
— LXC (via libvirt)
— UML (via libvirt)
— XCP (Xen)
— XenServer (from Citrix)
— ESX (from VMware)
— Hyper-V (from Microsoft)

Glance

```
root@node1:~# vi /etc/glance/glance-registry.conf
```

And change this parameter to use MySQL in place of SQLite:

```
sql_connection = mysql://glance:123456@node1/glance
```

Do also this to use Keystone in place of the deprecated identity service:

```
root@node1:~# vi /etc/glance/glance-api.conf
root@node1:~# vi /etc/glance/glance-registry.conf
```

And add this section and parameter:

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

Keystone

```
root@node1:~# vi /etc/keystone/keystone.conf
```

And adapt or add these parameters:

```
connection = mysql://keystone:123456@node1/keystone
driver = keystone.catalog.backends.templated.TemplatedCatalog
template_file = /etc/keystone/default_catalog.templates
admin_token = 999888777666
```

We will use the API endpoints as defined in the catalog template file (**default_catalog.templates**); no need to configure them in the Keystone DB.

To use the **nova** and **glance** CLIs from internet (e.g. client2), I have to adapt the **default_catalog.templates** file by replacing **localhost** (not accessible from internet) by **lc2.louvrex.net** (accessible from internet):

On node1:

```
root@node1:~# sed -i 's/localhost/lc2.louvrex.net/g'
    /etc/keystone/default_catalog.templates
root@node1:~# service keystone restart
```

Remark: Just replace **localhost** by the node1 public IP (192.168.1.201 here) if access from internet is not required.

Adapt the **xxx-paste.ini** files:

Nova and Glance

```
root@node1:~# vi /etc/nova/api-paste.ini
root@node1:~# vi /etc/glance/glance-api-paste.ini
root@node1:~# vi /etc/glance/glance-registry-paste.ini
```

And change or add these parameters:

```
admin_tenant_name = admin
admin_user = admin
admin_password = 123456
admin_token = 999888777666
```

Then reboot node1:

```
root@node1:~# reboot
```

- Create the tables in the DBs:

```
root@node1:~# nova-manage db sync
root@node1:~# glance-manage version_control 0 ; glance-manage db_sync
root@node1:~# keystone-manage db_sync
```

On node2 only:

- Adapt the **nova.conf** file:

Copy **nova.conf** from node1 to node2:

```
root@node2:~# scp root@node1:/etc/nova/nova.conf /etc/nova/
```

And adapt the **my_ip** parameter:

```
root@node2:~# vi /etc/nova/nova.conf
```

To look like this:

```
my_ip=172.16.1.202
```

Then reboot both nodes to take into account all the configurations made:

```
root@nodeX:~# reboot
```

- Check if all services are running:

```
root@node1:~# nova-manage service list
```

You should see something like this:

```
Binary           Host      Zone             Status     State Updated_At
nova-scheduler   node1     nova             enabled    :-)   2012-04-23 17:32:49
nova-network     node1     nova             enabled    :-)   2012-04-23 17:32:49
nova-volume      node1     nova             enabled    :-)   2012-04-23 17:32:49
nova-compute     node1     nova             enabled    :-)   2012-04-23 17:32:49
nova-compute     node2     nova             enabled    :-)   2012-04-23 17:32:50
```

- If you want to check all the configuration parameters:

```
root@node1:~# nova-manage config list
```

In case of problems, check the logs in **/var/log/nova**.

Configure the Networking (nova-network)

In the VLAN network mode, each tenant is given a specific VLAN/subnet. We will configure 10 VLANs/subnets (feel free to create much more).

VLAN	Bridge	Subnet	DGW (1)	VPN (2)	Instance Fixed IPs (3)
1	br1	10.0.1.0/24	10.0.1.1	10.0.1.2	10.0.1.3→10.0.1.254
2	br2	10.0.2.0/24	10.0.2.1	10.0.2.2	10.0.2.3→10.0.2.254
3	br3	10.0.3.0/24	10.0.3.1	10.0.3.2	10.0.3.3→10.0.3.254
4	br4	10.0.4.0/24	10.0.4.1	10.0.4.2	10.0.4.3→10.0.4.254
5	br5	10.0.5.0/24	10.0.5.1	10.0.5.2	10.0.5.3→10.0.5.254
6	br6	10.0.6.0/24	10.0.6.1	10.0.6.2	10.0.6.3→10.0.6.254
7	br7	10.0.7.0/24	10.0.7.1	10.0.7.2	10.0.7.3→10.0.7.254
8	br8	10.0.8.0/24	10.0.8.1	10.0.8.2	10.0.8.3→10.0.8.254
9	br9	10.0.9.0/24	10.0.9.1	10.0.9.2	10.0.9.3→10.0.9.254
10	br10	10.0.10.0/24	10.0.10.1	10.0.10.2	10.0.10.3→10.0.10.254

(1) Default gateway: automatically configured on the network node (node1 here).
(2) Cloudpipe VPN instance: eventually used to access the network via VPN.
(3) Instance fixed IPs: automatically distributed to the instances via DHCP.

Launch these commands to create the networks:

```
root@node1:~# nova-manage network create --label vlan1 --fixed_range_v4 10.0.1.0/24
    --num_networks 1 --network_size 256 --vlan 1

root@node1:~# nova-manage network create --label vlan2 --fixed_range_v4 10.0.2.0/24
    --num_networks 1 --network_size 256 --vlan 2

root@node1:~# nova-manage network create --label vlan3 --fixed_range_v4 10.0.3.0/24
    --num_networks 1 --network_size 256 --vlan 3

root@node1:~# nova-manage network create --label vlan4 --fixed_range_v4 10.0.4.0/24
    --num_networks 1 --network_size 256 --vlan 4

root@node1:~# nova-manage network create --label vlan5 --fixed_range_v4 10.0.5.0/24
    --num_networks 1 --network_size 256 --vlan 5

root@node1:~# nova-manage network create --label vlan6 --fixed_range_v4 10.0.6.0/24
    --num_networks 1 --network_size 256 --vlan 6

root@node1:~# nova-manage network create --label vlan7 --fixed_range_v4 10.0.7.0/24
    --num_networks 1 --network_size 256 --vlan 7

root@node1:~# nova-manage network create --label vlan8 --fixed_range_v4 10.0.8.0/24
    --num_networks 1 --network_size 256 --vlan 8

root@node1:~# nova-manage network create --label vlan9 --fixed_range_v4 10.0.9.0/24
    --num_networks 1 --network_size 256 --vlan 9

root@node1:~# nova-manage network create --label vlan10 --fixed_range_v4 10.0.10.0/24
    --num_networks 1 --network_size 256 --vlan 10
```

In fact, for each tenant, a specific VLAN and subnet is attributed, but also a specific:

— **Ethernet bridge** is configured on the network node and the compute nodes hosting the tenant's instances.
— **DHCP server** (**dnsmasq**) is launched on the network node to serve IPs to the tenant's instances.

The first time you launch an instance in the cloud, let's say an instance for **tenant1**, the **VLAN1** will be chosen and attributed exclusively to **tenant1**. As from that moment, VLAN1 will always be used for instances for tenant1.

If you launch an instance for another tenant, the first VLAN not yet attributed to a tenant will be chosen and attributed to that tenant.

The **ip addr** (show IP addresses) and **brctl show** (show bridge interfaces) commands on the node1 will give a result like this (I made a lot of cleaning):

Command:

```
root@node1:~# ip addr
```

Result:

```
1: lo
inet 127.0.0.1/8
inet 169.254.169.254/32 (1)

(1) Metadata service

2: eth0 (2)
ether 00:24:1d:d3:a1:e6
inet 192.168.1.201/24 (3)

(2) First physical Ethernet interface (connected to the public network)
(3) node1 public IP

3: eth1 (4)
ether 00:10:18:34:c0:e5
inet 172.16.1.201/24 (5)

(4) Second physical Ethernet interface (connected to the private network)
(5) node1 private IP

4: virbr0 (6)
ether ae:4e:3d:1f:97:3b
inet 192.168.122.1/24

(6) Bridge configured by the libvirt API (not used here)
```

Command:

```
root@node1:~# brctl show
```

Result:

```
virbr0 (1)

(1) Bridge configured by the libvirt API (not used here)
```

If you launch some instances for different tenants, and if you associate some floating IPs to them, you could have a result like this:

Command:

```
root@node1:~# ip addr
```

Result:

```
1: lo
inet 127.0.0.1/8
inet 169.254.169.254/32 (1)

(1) Metadata service

2: eth0 (2)
ether 00:24:1d:d3:a1:e6
inet 192.168.1.201/24 (3)
inet 192.168.1.240/32 (4)
inet 192.168.1.241/32 (5)
```

(2) First physical Ethernet interface (connected to the public network)
(3) node1 public IP
(4) Floating IP n° 1 (associated to an instance running on node1 or node2)
(5) Floating IP n° 2 (associated to an instance running on node1 or node2)

```
3: eth1 (6)
ether 00:10:18:34:c0:e5
inet 172.16.1.201/24 (7)
```

(6) Second physical Ethernet interface (connected to the private network)
(7) node1 private IP

```
4: virbr0 (8)
ether ae:4e:3d:1f:97:3b
inet 192.168.122.1/24
```

(8) Bridge configured by the libvirt API (not used here). I will no more show it bellow.

```
5: vlan1@eth1 (9)
ether 00:10:18:34:c0:e5
```

(9) eth1 interface tagged for VLAN1 (can also be written as eth1.1)

```
6: br1 (10)
ether 00:10:18:34:c0:e5
inet 10.0.1.1/24 (11)
```

(10) Bridge connected on the vlan1@eth1 interface
(11) Default gateway of the first VLAN network (e.g. for the 1st tenant)

```
7: vlan2@eth1 (12)
ether 00:10:18:34:c0:e5
```

(12) eth1 interface tagged for VLAN2 (eth1.2)

```
8: br2 (13)
ether 00:10:18:34:c0:e5
inet 10.0.2.1/24 (14)
```

(13) Bridge connected on the vlan2@eth1 interface
(14) Default gateway of the second VLAN network (e.g. for the 2nd tenant)

```
9: vlan3@eth1 (15)
ether 00:10:18:34:c0:e5
```

(15) eth1 interface tagged for VLAN1 (eth1.3)

```
10: br3 (16)
ether 00:10:18:34:c0:e5
inet 10.0.3.1/24 (17)
```

(16) Bridge connected on the vlan3@eth1 interface
(17) Default gateway of the third VLAN network (e.g. for the 3rd tenant)

```
11: vnet0 (18)
ether fe:16:3e:2a:a3:f1
```

(18) Virtual interface for the first instance running on the node1

```
12: vnet1 (19)
ether fe:16:3e:46:07:6b
```

(19) Virtual interface for the second instance running on the node1

```
13: vnet2 (20)
ether fe:16:3e:34:53:06
```

(20) Virtual interface for the third instance running on the node1

Command:

```
root@node1:~# brctl show
```

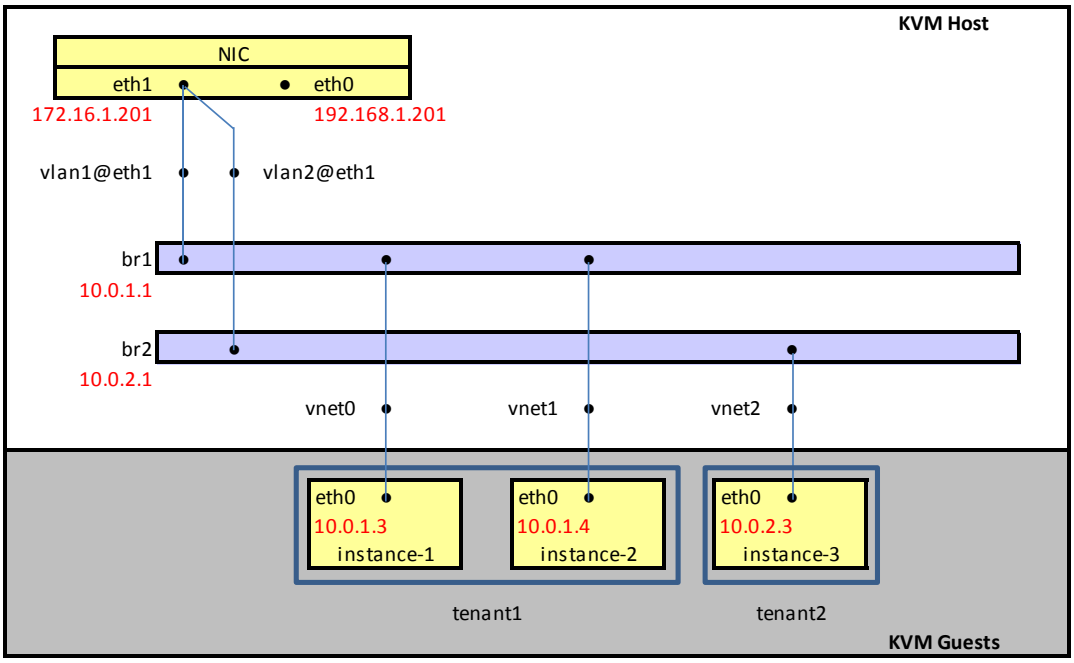
Result:

```
br1   vlan1
   vnet0 (1)
   vnet1 (2)
br2   vlan2
   vnet2 (3)
br3   vlan3

(1) Virtual interface for the 1st instance running on the node1 (VLAN1 / tenant1)
(2) Virtual interface for the 2nd instance running on the node1 (VLAN1 / tenant1)
(3) Virtual interface for the 3rd instance running on the node1 (VLAN2 / tenant2)
```

If you are using LXC, the virtual interfaces will be named veth0, veth1, etc.

Bellow a picture which represents the VLAN networking mode:



- Configure the public floating IPs:

For example two pools:

— **pool1**: 192.168.1.224/27 (192.168.1.225 → 192.168.1.254)
— **pool2**: 192.168.1.96/27 (192.168.1.97 → 192.168.1.126)

Launch these commands:

```
root@node1:~# nova-manage floating create --pool pool1 --ip_range 192.168.1.224/27
root@node1:~# nova-manage floating create --pool pool2 --ip_range 192.168.1.96/27
```

iptables is used to configure the floating IPs on the network node (node1 here). **iptables** is also used to configure the firewall rules (security groups) on the compute nodes (node1 and node2 here).

For the floating IPs, the **nat** table is used. You can see these NATing rules (SNAT and DNAT) on the node1 with this command:

```
root@node1:~# iptables -nL -t nat
```

For the firewall rules (security groups), the **filter** table is used. You can see them on the node1 or the node2 with this command:

```
root@nodeX:~# iptables -nL -t filter
```

Configure the Storage (nova-volume)

On node1:

- Create one LVM primary partition (sdb1) on the second HDD:

```
root@node1:~# cfdisk /dev/sdb
```

- Create one LVM physical volume:

```
root@node1:~# pvcreate /dev/sdb1
```

- Create one LVM volume group called **nova-volumes**:

```
root@node1:~# vgcreate nova-volumes /dev/sdb1
```

- Start the nova-volume service (or reboot the server):

```
root@node1:~# service nova-volume start
```

Please note that in our configuration, the iSCSI traffic will pass on the storage network. This traffic flows between the nova-volume component (the host is the iSCSI target) and the nova-compute component (the host is the iSCSI initiator). The nova-compute component then "exposes" the volumes to the attached instances.

You have the choice of the iSCSI target software you want to use:

— **iet** (iSCSI Enterprise Target / iscsitarget): default until Ubuntu Natty.
— **tgt** (Linux SCSI target framework): default as from Ubuntu Oneiric.

The flag in the **nova.conf** file is:

```
iscsi_helper=ietadm|tgtadm
```

If volumes have to be created on a file backend, and not on a dedicated hard disk partition, you can do this instead for a 50 GB backend file:

```
root@node1:~# truncate -s 50G /var/lib/nova/volume-backing-file
root@node1:~# losetup -f --show /var/lib/nova/volume-backing-file
root@node1:~# vgcreate nova-volumes /dev/loop0
```

Edit the upstart nova-volume script to "losetup" the file before starting nova-volume:

```
root@node1:~# vi /etc/init/nova-volume.conf
```

And adapt the file as follow:

```
pre-start script
        mkdir -p /var/run/nova
        chown nova:root /var/run/nova/
        losetup -f /var/lib/nova/volume-backing-file ← Add this line
end script
```

Install the Dashboard

```
root@node1:~# aptitude install openstack-dashboard
```

If needed, install this dependency:

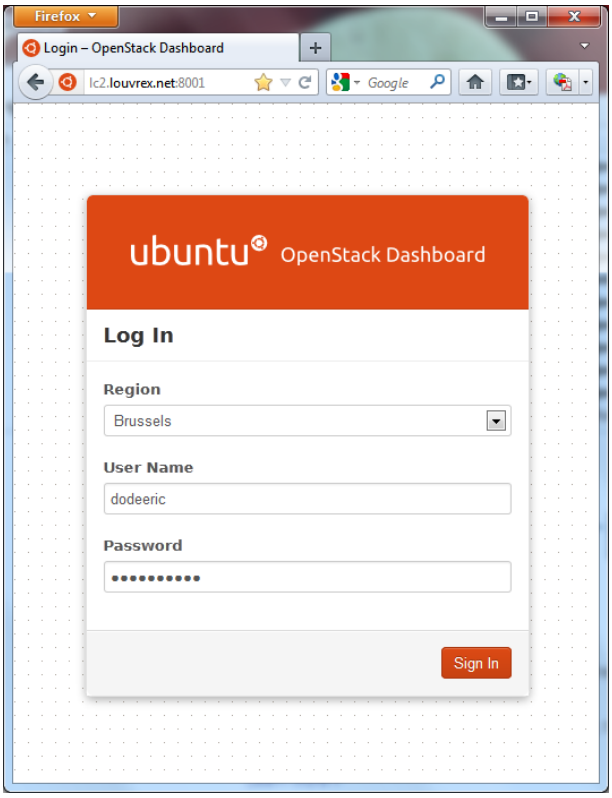
```
root@node1:~# aptitude install memcached
```

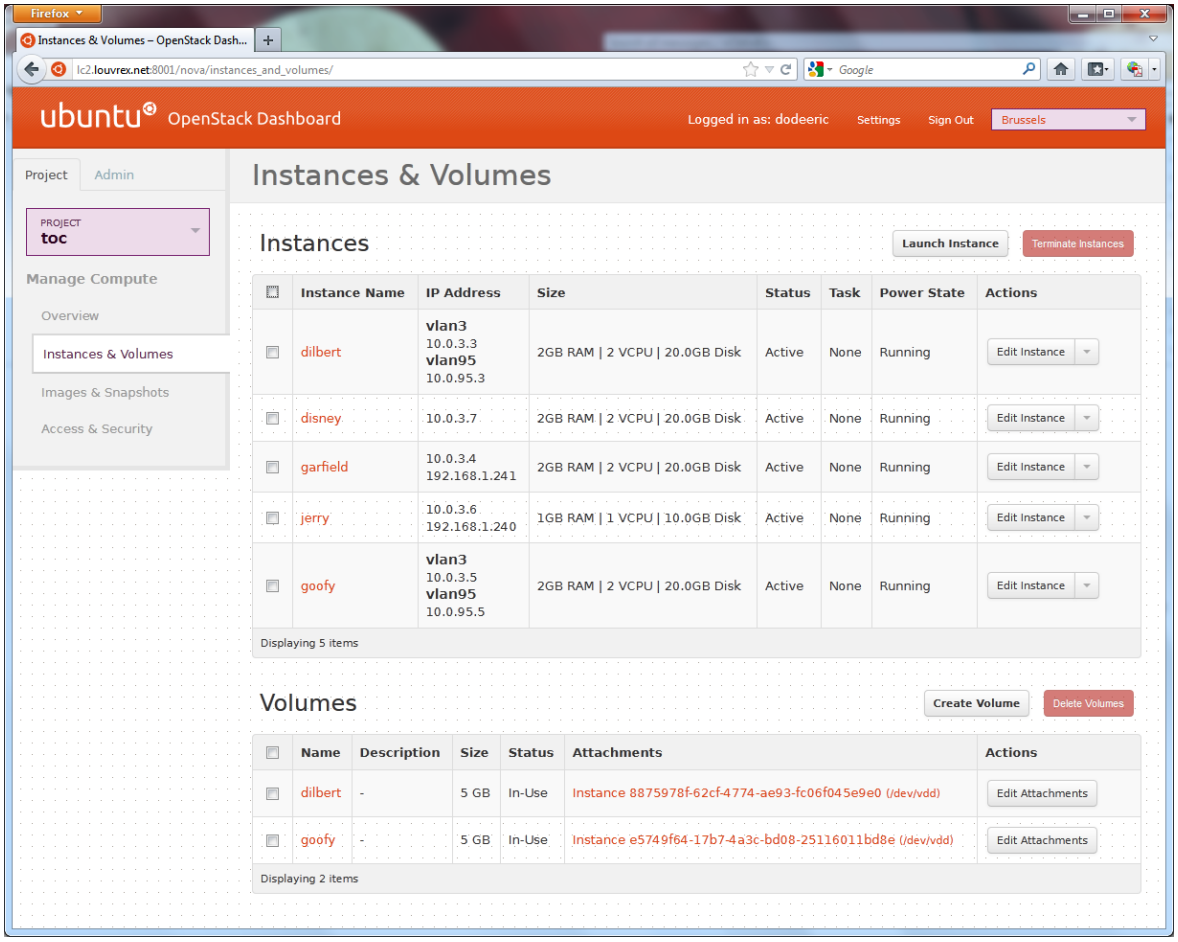
If you want the Ubuntu theme, do this:

```
root@node1:~# aptitude install openstack-dashboard-ubuntu-theme
```

I can access the dashboard with my web browser at the following addresses:

- From my public network (e.g. client1): **http://192.168.1.201**
- From the internet (e.g. client2): **http://lc2.louvrex.net:8001** (see "Internet and DNS" below)





Internet and DNS

DynDNS

My personal home network has only one internet IP address, and it is a dynamic IP changing every 4 days.

My cloud is available from the internet with the DNS name lc2.louvrex.net (LC2 = Louvrex Cloud Computing). The lc2.louvrex.net DNS name is linked to my internet dynamic IP address. I use the DynDNS service with the **ddclient** running on the node1 to update the DNS name automatically.

NAT/PAT

Different NAT/PAT rules are configured in the router to access the nova-api, the nova-ec2-api, the volume-api, the keystone-api, the node (ssh), and the instances via floating IPs (ssh, http, etc.)

Here a sample of these rules:

Rule name	Internet port	LAN port	LAN IP
nova-api	8774	8774	192.168.1.201
nova-ec2-api	8773	8773	192.168.1.201
volume-api	8776	8776	192.168.1.201
glance-api	9292	9292	192.168.1.201
keystone-api	5000	5000	192.168.1.201
keystone-adm-api	35357	35357	192.168.1.201

```
dashboard          8001            80          192.168.1.201

node1-ssh          2201            22          192.168.1.201

fip1-ssh           22              22          192.168.1.240
fip2-ssh           2201            22          192.168.1.241
fip3-ssh           2202            22          192.168.1.242
fip4-ssh           2203            22          192.168.1.243
fip5-ssh           2204            22          192.168.1.244
fip6-ssh           2205            22          192.168.1.245
fip7-ssh           2206            22          192.168.1.246
fip8-ssh           2207            22          192.168.1.247

fip1-http          80              80          192.168.1.240
fip2-http          8041            80          192.168.1.241
fip3-http          8042            80          192.168.1.242
fip4-http          8043            80          192.168.1.243
fip5-http          8044            80          192.168.1.244
fip6-http          8045            80          192.168.1.245
fip7-http          8046            80          192.168.1.246
fip8-http          8047            80          192.168.1.247
```

The generic rules are as follow:

ssh: lc2.louvrex.net:22XX → 192.168.1.2XX:22
http: lc2.louvrex.net:80XX → 192.168.1.2XX:80

For example: to access the web server running in an instance associated to the floating IP 192.168.1.245, the following URL has to be used: http://lc2.louvrex.net:8045.

Configure Keystone

From node1:

- Create the services:

```
root@node1:~# keystone service-create --name nova --type compute
    --description 'OpenStack Compute Service'

root@node1:~# keystone service-create --name nova-volume --type volume
    --description 'OpenStack Nova Volume Service'

root@node1:~# keystone service-create --name glance --type image
    --description 'OpenStack Image Service'

root@node1:~# keystone service-create --name keystone --type identity
    --description 'OpenStack Identity Service'

root@node1:~# keystone service-create --name quantum --type network
    --description 'Openstack Network Service'
```

The volume service (nova-volume) is still included in the Nova code, but has already its own API. In the next OpenStack release (Folsom / 2012.2), it will be a separate project (code name: Cinder).

In our configuration, we are still using the network service (nova-network) included in the Nova code. There is already a separate network project (code name: Quantum) which can be used and which has different network plugins like Linux bridge or Openvswitch (OVS).

- Create the roles (profiles):

```
root@node1:~# keystone role-create --name KeystoneServiceAdmin
root@node1:~# keystone role-create --name Admin
root@node1:~# keystone role-create --name Member
root@node1:~# keystone role-create --name sysadmin
```

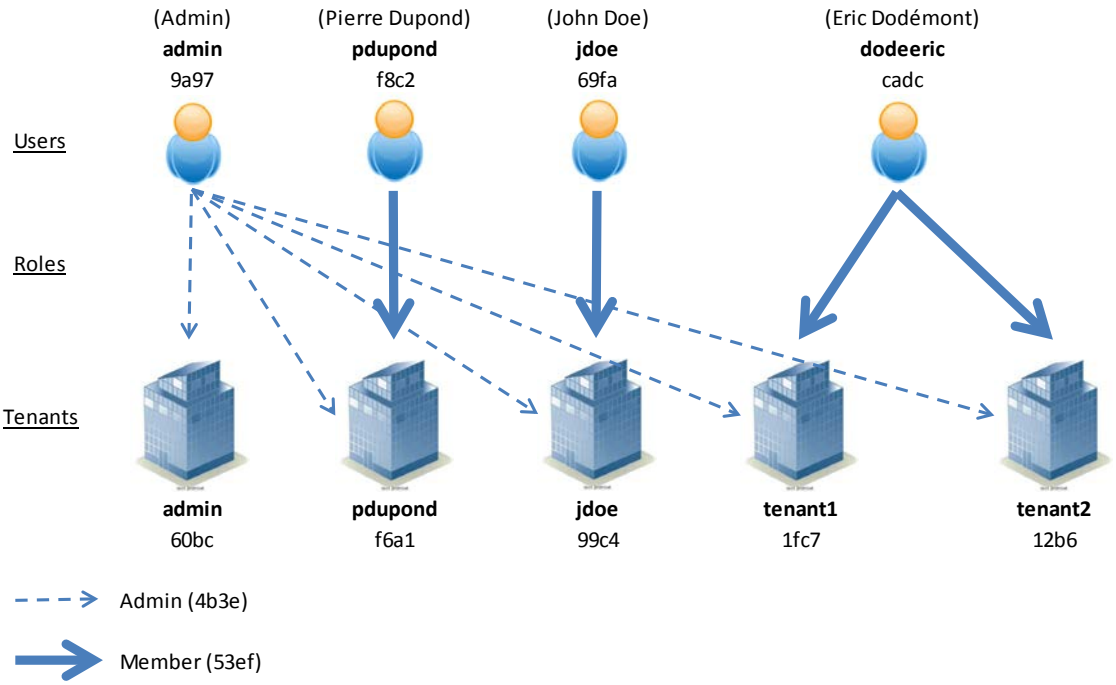
```
root@node1:~# keystone role-create --name netadmin
```

Users and Tenants (keystone)

For clarity, I often represent the IDs (UUIDs) with only the first 4 digits in place of the 32 digits.

Only the **admin** user and tenant need to be created with the keystone CLI. For the other users and tenants, and for the role associations, the dashboard web interface will make you job much more easy.

This picture shows the configuration to do:



- Create the tenants (projects):

```
root@node1:~# keystone tenant-create --name admin
root@node1:~# keystone tenant-create --name tenant1
root@node1:~# keystone tenant-create --name tenant2
root@node1:~# keystone tenant-create --name pdupond
root@node1:~# keystone tenant-create --name jdoe
```

- Create the users:

Command format: keystone user-create --name <user-name> --tenant_id <primary-tenant-id>
--pass <password> --email <email> --enabled true

Use this command to find the tenant IDs:

```
root@node1:~# keystone tenant-list
```

Result:

```
+----------------------------------+---------+---------+
|                id                |   name  | enabled |
+----------------------------------+---------+---------+
| 60bc84ca0ce9429587300cb30c56c71d | admin   | True    |
| 1fc7082857bc412da7418c264eefa28c | tenant1 | True    |
| 12b67b1a806a492f8000f3cb82d912cf | tenant2 | True    |
```

```
| f6a1c12cf10b43e68aa0952470f1bb56 | pdupond | True    |
| 99c4a880dba244d0a2964829a60d167c | jdoe    | True    |
+----------------------------------+---------+---------+
```

That gives this:

```
root@node1:~# keystone user-create --name admin --tenant_id 60bc --pass 123456
    --email admin@lc2.be --enabled true

root@node1:~# keystone user-create --name dodeeric --tenant_id 1fc7 --pass 123456
    --email eric.dodemont@skynet.be --enabled true

root@node1:~# keystone user-create --name pdupond --tenant_id f6a1 --pass 123456
    --email pdupond@skynet.be --enabled true

root@node1:~# keystone user-create --name jdoe --tenant_id 99c4 --pass 123456
    --email jdoe@skynet.be --enabled true
```

- Add the **KeystoneServiceAdmin** role to the **admin** user:

```
root@node1:~# keystone user-role-add --user 9a97 --role e67b
```

- Add the **Admin** role to the **admin** user for each tenant:

Command format: keystone user-role-add --user <admin-user-id> --role <Admin-role-id>
--tenant_id <tenant-id>

Use these commands to find the user and role IDs:

```
root@node1:~# keystone user-list
```

Result:

```
+----------------------------------+---------+-------------------------+----------+
|                id                | enabled |          email          |   name   |
+----------------------------------+---------+-------------------------+----------+
| 9a97f035897a40ba80cffb1ccec43d4b | True    | admin@lc2.be            | admin    |
| cadc85c18c6a4aee928c19653f48dd45 | True    | eric.dodemont@skynet.be | dodeeric |
| f8c27de796914e56b5b4a2bc3f6a2432 | True    | pdupond@skynet.be       | pdupond  |
| 69fa91a0defb4cab967aa089d8209161 | True    | jdoe@skynet.be          | jdoe     |
+----------------------------------+---------+-------------------------+----------+
```

Command:

```
root@node1:~# keystone role-list
```

Result:

```
+----------------------------------+----------------------+
|                id                |         name         |
+----------------------------------+----------------------+
| 4b3e61009ea74a0aab178d6a62c53bf2 | Admin                |
| 53efc9a762d24fe38ac3b8cdb5907120 | Member               |
| 65520c50f6b04b31a69901ba00b32324 | sysadmin             |
| 80a8f4a8fc24402c991fbdd283484638 | netadmin             |
| e67be62bb9144c6cba647c5a4ed5ecdd | KeystoneServiceAdmin |
+----------------------------------+----------------------+
```

That gives this:

```
root@node1:~# keystone user-role-add --user 9a97 --role 4b3e --tenant_id 60bc
root@node1:~# keystone user-role-add --user 9a97 --role 4b3e --tenant_id 1fc7
root@node1:~# keystone user-role-add --user 9a97 --role 4b3e --tenant_id 12b6
root@node1:~# keystone user-role-add --user 9a97 --role 4b3e --tenant_id f6a1
root@node1:~# keystone user-role-add --user 9a97 --role 4b3e --tenant_id 99c4
```

- Add user **dodeeric** as a member of tenant **tenant2** (in addition to its primary tenant):

Command format: keystone user-role-add --user <dodeeric-user-id> --role <Member-role-id>
--tenant_id <tenant2-tenant-id>

That gives this:

```
root@node1:~# keystone user-role-add --user cadc --role 53ef --tenant_id 12b6
```

The user **dodeeric** will be able to manage the tenants **tenant1** and **tenant2**. In the dashboard he will see both projects (tenants) and will have to select on which one he will work.

- Create EC2 credentials (to use the euca commands) for user **dodeeric** and tenants **tenant1** and **tenant2**:

Command format: keystone ec2-credentials-create --user <dodeeric-user-id> --tenant_id <tenant-id>

That gives this:

```
root@node1:~# keystone ec2-credentials-create --user cadc --tenant_id 1fc7
root@node1:~# keystone ec2-credentials-create --user cadc --tenant_id 12b6
```

Flavors (Instance Types)

- Add some personal flavors:

Before this, you can delete all the predefined flavors with the **nova flavor-delete** command.

```
root@node1:~# nova-manage instance_type create --name tiny --memory 512 --cpu 1
    --root_gb 10 --ephemeral_gb 0 --swap 0 --flavor 1

root@node1:~# nova-manage instance_type create --name small --memory 1024 --cpu 1
    --root_gb 10 --ephemeral_gb 10 --swap 1024 --flavor 2

root@node1:~# nova-manage instance_type create --name medium --memory 2048 --cpu 2
    --root_gb 20 --ephemeral_gb 20 --swap 2048 --flavor 3

root@node1:~# nova-manage instance_type create --name large --memory 4096 --cpu 4
    --root_gb 40 --ephemeral_gb 40 --swap 4096 --flavor 4
```

List the available flavors:

```
root@node1:~# nova flavor-list
```

Result:

```
+----+--------+-----------+------+-----------+------+-------+-------------+
| ID |  Name  | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor |
+----+--------+-----------+------+-----------+------+-------+-------------+
| 1  | tiny   | 512       | 10   | 0         |      | 1     | 1.0         |
| 2  | small  | 1024      | 10   | 10        | 1024 | 1     | 1.0         |
| 3  | medium | 2048      | 20   | 20        | 2048 | 2     | 1.0         |
| 4  | large  | 4096      | 40   | 40        | 4096 | 4     | 1.0         |
+----+--------+-----------+------+-----------+------+-------+-------------+
```

Ephemeral is a second local disk in addition to the local root-fs disk.

Use your Cloud

CLI Clients

There are two APIs in OpenStack Nova:

— **Amazon EC2-API**: you will use the euca-XXX commands with that API (package: euca2ools). These commands are compatible with the Amazon public cloud (AWS: Amazon Web Services; EC2: Elastic Compute Cloud).
— **OpenStack OS-API**: you will use the nova XXXX commands with that API (package: python-novaclient).

In this document, I will mainly use the OS-API.

The **nova-manage**, **glance-manage**, **keystone-manage**, and **keystone** commands are reserved for the administrators of the cloud, when **euca**, **nova**, and **glance** commands are for the end-users of the cloud.

On **client1** or **client2**:

- Install python-novaclient (the OpenStack Nova CLI client):

```
pdupond@clientX:~$ sudo aptitude install python-novaclient
```

The python-novaclient package is installed automatically with the nova-common package which is needed by all Nova components. If you want to launch the **nova** commands from the node1 or the node2, then no need to install that package.

- Install Euca2ools (the Amazon EC2 CLI client):

```
pdupond@clientX:~$ sudo aptitude install euca2ools
```

- Install glance-client (the Glance CLI client):

```
pdupond@clientX:~$ sudo aptitude install glance-client
```

Environment Variables

To be able to launch **nova** or **glance** commands, you need to export some environment variables to tell the API who you are and on which tenant you want to work on.

For example, if the user **dodeeric** wants to work with **tenant1**, the following environment variables have to be exported:

```
pdupond@client1:~$ export OS_TENANT_NAME=tenant1
pdupond@client1:~$ export OS_USERNAME=dodeeric
pdupond@client1:~$ export OS_PASSWORD=123456
pdupond@client1:~$ export OS_AUTH_URL=http://192.168.1.201:5000/v2.0
pdupond@client1:~$ export COMPUTE_API_VERSION=1.1
```

Port 5000 is the Keystone API. Keystone has a "Service Catalog" with the different service's API end-points (Nova, Nova-Volume, Glance, etc.)

If you want to use the **euca** commands (EC2 API), export these environment variables:

```
root@node1: ~# keystone ec2-credentials-list
```

Result:

```
+---------+----------------------------------+----------------------------------+
| tenant  |              access              |              secret              |
+---------+----------------------------------+----------------------------------+
| tenant1 | d1b9b28db6894de8b441421e9dd695c2 | 5f0d614e45d04d66be12c3493f03282a |
| tenant2 | 5655c7711e3a43688d328d9eaa147bee | 9c50faff1e0d420f8c8baf4ffac93ff3 |
+---------+----------------------------------+----------------------------------+
```

And export the variables:

```
pdupond@client1:~$ export EC2_ACCESS_KEY="d1b9b28db6894de8b441421e9dd695c2"
pdupond@client1:~$ export EC2_SECRET_KEY="5f0d614e45d04d66be12c3493f03282a"
pdupond@client1:~$ export EC2_URL="http://192.168.1.201:8773/services/Cloud"
```

For the rest of the document, we are user **pdupond** working for tenant **pdupond** unless something else is specified.

Security Groups (Firewall Rules)

- Adapt the **default** security group to allow by default ping (icmp) and ssh (tcp/22) from everywhere (0.0.0.0/0):

```
pdupond@client1:~$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
pdupond@client1:~$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
```

The **default** security group is created automatically and cannot be deleted.

- Create the **web-server** security group to allow ping (icmp), ssh (tcp/22), and http (tcp/80) from everywhere (0.0.0.0/0):

```
pdupond@client1:~$ nova secgroup-create web-server "Web server running on default port"
pdupond@client1:~$ nova secgroup-add-rule web-server icmp -1 -1 0.0.0.0/0
pdupond@client1:~$ nova secgroup-add-rule web-server tcp 22 22 0.0.0.0/0
pdupond@client1:~$ nova secgroup-add-rule web-server tcp 80 80 0.0.0.0/0
```

Key Pairs

Create a SSH key pair to be used to access your instances.

You can store your private key on your personal computer. The public key is automatically stored in the controller node (MySQL DB) and injected in your instance's disk at boot time.

```
pdupond@client1:~$ nova keypair-add pdupond > pdupond.pem
pdupond@client1:~$ chmod 600 *.pem
```

The **pdupond.pem** file is your private key (do not share it).

Key pairs are linked to users. **Images, security groups**, and **floating IPs** are linked to tenants.

You can also get your personal RC files (with the needed environment variables exports) from the dashboard. To download the RC file, go to: Settings ➔ OpenStack Credentials ➔ Select a Project ➔ Download RC File. For EC2, go to "EC2 Credentials".

Then to export the environment variables, do this:

```
pdupond@client1:~$ source openrc.sh
```

Or:

```
pdupond@client1:~$ . openrc.sh
```

Or:

```
pdupond@client1:~$ ./openrc.sh
```

To see your present environment variables:

```
pdupond@client1:~$ env
```

Images

Add at least one image to Glance. Let's use the **glance** CLI for that.

- Download the latest Ubuntu server 12.04 (Precise) 64 bits cloud image:

```
pdupond@client1:~$ wget http://cloud-images.ubuntu.com/precise/current/precise-
    server-cloudimg-amd64-disk1.img
```

Rename it:

```
pdupond@client1:~$ mv precise-server-cloudimg-amd64-disk1.img precise-
    server-cloudimg-amd64.img
```

That image it is in qcow2 format:

```
pdupond@client1:~$ qemu-img info precise-server-cloudimg-amd64.img
```

Result:

```
image: precise-server-cloudimg-amd64.img
file format: qcow2
virtual size: 2.0G (2147483648 bytes)
disk size: 216M
cluster_size: 65536
```

Or:

```
pdupond@client1:~$ file precise-server-cloudimg-amd64.img
```

Result:

```
precise-server-cloudimg-amd64.img: QEMU QCOW Image (v2), 2147483648 bytes
```

This file is only 216 MB big, but is seen as a disk image of 2 GB by the file system.

- Upload that image to Glance:

```
pdupond@client1:~$ glance add name="precise" is_public=True disk_format=qcow2
    container_format=ovf architecture=x86_64 < precise-server-cloudimg-amd64.img
```

Result:

```
Added new image with ID: b6dc7b99-3177-47a0-9010-8095afee902c
```

We gave the name "precise" to our image.

Check the result with **nova**:

```
pdupond@client1:~$ nova image-list
```

Result:

```
+--------------------------------------+---------------------------------+--------+--------+
|                  ID                  |              Name               | Status | Server |
+--------------------------------------+---------------------------------+--------+--------+
| b6dc7b99-3177-47a0-9010-8095afee902c | precise                         | ACTIVE |        |
+--------------------------------------+---------------------------------+--------+--------+
```

Check the result with **glance**:

```
pdupond@client1:~$ glance index
```

Result:

```
ID                                   Name    Disk Format   Container Format   Size
------------------------------------ ------  ------------ ------------------ -----------
b6dc7b99-3177-47a0-9010-8095afee902c precise qcow2         ovf                226689024
```

Show all the details of the image:

```
pdupond@client1:~$ glance show b6dc7b99-3177-47a0-9010-8095afee902c
```

Result:

```
URI: http://localhost:9292/v1/images/b6dc7b99-3177-47a0-9010-8095afee902c
Id: b6dc7b99-3177-47a0-9010-8095afee902c
Public: Yes
Protected: No
Name: precise
Status: active
Size: 226689024
Disk format: qcow2
Container format: ovf
Minimum Ram Required (MB): 0
Minimum Disk Required (GB): 0
Owner: 60bc84ca0ce9429587300cb30c56c71d    → admin
Property 'architecture': x86_64
```

I was working for the tenant **admin** (id: 60bc) when I uploaded that image. That is why the "Owner" of the image is the **admin** tenant. As this image is **public**, it can be used by **all tenants**.

Images are stored in Glance in the **/var/lib/glance/images** directory.

Instances

Boot

Let's start an instance with a **medium** flavor (vCPU: 2, RAM: 2 GB, HDD1: 20 GB, HDD2: 20 GB, swap: 2 GB):

```
pdupond@client1:~$ nova boot --flavor medium --image precise --key_name pdupond
    --security_groups default i1
```

We gave the name "i1" to our instance.

First the instance is in status "BUILD" for some seconds:

```
pdupond@client1:~$ nova list
```

Result:

```
+--------------------------------------+------+--------+----------------+
|                  ID                  | Name | Status |    Networks    |
+--------------------------------------+------+--------+----------------+
| 37125fa1-d89f-4c0f-816d-2855bc01a2c5 | i1   | BUILD  | vlan5=10.0.5.3 |
+--------------------------------------+------+--------+----------------+
```

And after some seconds, the status becomes "ACTIVE":

```
pdupond@client1:~$ nova list
```

Result:

```
+--------------------------------------+------+--------+----------------+
|                  ID                  | Name | Status |    Networks    |
+--------------------------------------+------+--------+----------------+
| 37125fa1-d89f-4c0f-816d-2855bc01a2c5 | i1   | ACTIVE | vlan5=10.0.5.3 |
+--------------------------------------+------+--------+----------------+
```

It will take much longer to boot if the image is used on the compute node for the first time because the image has to be copied (cached) on the host in directory **/var/lib/nova/instances/_base**.

As you can see, this instance received via DHCP the first free IP (10.0.5.3) of network 10.0.5.0/24 on VLAN5. That is the network/VLAN which is reserved for the **pdupond** tenant.

You can check on which compute node the instance has been launched (node1 or node2):

```
root@node1:~# nova-manage vm list
```

Result:

```
Instance   node    type     state    launched   image   project   user   zone
i1         node2   medium   active   08:56:46   a4c5    f6a1      f8c2   None
```

The instance "i1" is running on node2.

To see the details of the instance:

```
pdupond@client1:~$ nova show i1
```

Result:

```
+------------------------+------------------------------------------------+
|        Property        |                  Value                         |
+------------------------+------------------------------------------------+
| created                | 2012-04-27T08:56:30Z                           |
| flavor                 | medium                                         |
| id                     | 37125fa1-d89f-4c0f-816d-2855bc01a2c5           |
| image                  | precise                                        |
| key_name               | pdupond                                        |
| name                   | i1                                             |
| status                 | ACTIVE                                         |
| tenant_id              | f6a1c12cf10b43e68aa0952470f1bb56               | → pdupond
| user_id                | f8c27de796914e56b5b4a2bc3f6a2432               | → pdupond
| vlan5 network          | 10.0.5.3                                       |
+------------------------+------------------------------------------------+
```

Now you can login into your instance from a host connected to VLAN5 (e.g. from node1):

```
pdupond@node1:~$ ssh -i pdupond.pem ubuntu@10.0.5.3
```

Result:

```
ubuntu@i1:~$
```

Please note that the **hostname** is equal to the name given to the instance.

Let's check the network (one NIC) inside the instance:

```
ubuntu@i1:~$ ifconfig
```

Result:

```
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0

eth0      Link encap:Ethernet  HWaddr fa:16:3e:45:97:11
          inet addr:10.0.5.3  Bcast:10.0.5.255  Mask:255.255.255.0
```

Let's check the type of controllers:

```
ubuntu@i1:~$ lspci
```

Result:

```
00:03.0 Ethernet controller: Red Hat, Inc Virtio network device
00:04.0 Ethernet controller: Red Hat, Inc Virtio network device
00:05.0 SCSI storage controller: Red Hat, Inc Virtio block device
00:06.0 SCSI storage controller: Red Hat, Inc Virtio block device
00:07.0 SCSI storage controller: Red Hat, Inc Virtio block device
```

You can see that the disk and network controllers are of type "virtio". This is a generic paravirtulized type of controller very efficient to use with KVM.

Let's check if the number of vCPU, the size of the RAM, and the number and sizes of the HDDs is correct.

The HDDs:

```
ubuntu@i1:~$ df -h
```

Result:

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        20G  670M   19G   4% /
/dev/vdb         20G  173M   19G   1% /mnt
```

The second HDD is automatically mounted on the **/mnt** directory by **cloud-init** (cloud-init is installed automatically in ubuntu-cloudimg).

Or:

```
ubuntu@i1:~$ sudo fdisk -l
```

Result:

```
Disk /dev/vda: 21.5 GB, 21474836480 bytes     → root_gb
Disk /dev/vdb: 21.5 GB, 21474836480 bytes     → ephemeral_gb
Disk /dev/vdc: 2147 MB, 2147483648 bytes      → swap
```

The memory (RAM and disk swap):

```
ubuntu@i1:~$ free –m
```

Result:

```
             total       used       free     shared    buffers     cached
Mem:          2003        249       1753          0         10        170
Swap:         2047          0       2047
```

The vCPU:

```
ubuntu@i1:~$ lscpu
```

Result:

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
CPU(s):                2
On-line CPU(s) list:   0,1
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             2
Vendor ID:             GenuineIntel
Virtualization:        VT-x
Hypervisor vendor:     KVM
Virtualization type:   full
```

Let's check the Linux distribution:

```
ubuntu@i1:~$ cat /etc/issue
```

Result:

```
Ubuntu 12.04 LTS \n \l
```

And the kernel (64 bits):

```
ubuntu@i1:~$ uname -a
```

Result:

```
Linux i1 3.2.0-24-virtual #37-Ubuntu SMP Wed Apr 25 10:17:19 UTC 2012 x86_64 GNU/Linux
```

From node2 (as the instance has been launched on node2), you can check different parameters of the running instance.

The home directory where the instance's files are stored:

```
root@node2:~# cd /var/lib/nova/instances/instance-000000da
```

Command:

```
root@node2:/var/lib/nova/instances/instance-000000da# ls -lh
```

Result:

```
-rw-rw---- 1 libvirt-qemu kvm   22K Apr 27 10:57 console.log
-rw-r--r-- 1 libvirt-qemu kvm  722M Apr 27 11:10 disk
-rw-r--r-- 1 libvirt-qemu kvm   12M Apr 27 10:57 disk.local
-rw-r--r-- 1 libvirt-qemu kvm  6.1M Apr 27 10:56 disk.swap
-rw-rw-r-- 1 nova         nova 1.9K Apr 27 10:56 libvirt.xml
```

This is the virtual machine **libvirt** configuration file:

```
root@node2:/var/lib/nova/instances/instance-000000da# cat libvirt.xml
```

Result:

```
<domain type='kvm'>
    <uuid>37125fa1-d89f-4c0f-816d-2855bc01a2c5</uuid>
```

```
    <name>instance-000000da</name>
    <memory>2097152</memory>
    <os>
            <type>hvm</type>
            <boot dev="hd" />
    </os>
    <features>
        <acpi/>
    </features>
    <vcpu>2</vcpu>
    <devices>
        <disk type='file' device='disk'>
            <driver type='qcow2' cache='none'/>
            <source file='/var/lib/nova/instances/instance-000000da/disk'/>
            <target dev='vda' bus='virtio'/>
        </disk>
            <disk type='file'>
                <driver type='qcow2' cache='none'/>
                <source file='/var/lib/nova/instances/instance-000000da/disk.local'/>
                <target dev='vdb' bus='virtio'/>
            </disk>
            <disk type='file'>
                <driver type='qcow2' cache='none'/>
                <source file='/var/lib/nova/instances/instance-000000da/disk.swap'/>
                <target dev='vdc' bus='virtio'/>
            </disk>
        <interface type='bridge'>
            <source bridge='br5'/>
            <mac address='fa:16:3e:45:97:11'/>
            <model type='virtio'/>
            <filterref filter="nova-instance-instance-000000da-fa163e459711">
                <parameter name="IP" value="10.0.5.3" />
                <parameter name="DHCPSERVER" value="10.0.5.1" />
            </filterref>
        </interface>
        <serial type='file'>
            <source path='/var/lib/nova/instances/instance-000000da/console.log'/>
        </serial>
        <serial type='pty'/>
        <input type='tablet' bus='usb'/>
        <graphics type='vnc' port='-1' autoport='yes' keymap='en-us' listen='127.0.0.1'/>
    </devices>
</domain>
```

The root disk has the format qcow2 (virtual size: 20 GB; real size: 718 MB):

```
root@node2:/var/lib/nova/instances/instance-000000da# qemu-img info disk
```

Result:

```
image: disk
file format: qcow2
virtual size: 20G (21474836480 bytes)
disk size: 718M
backing file: /var/lib/nova/instances/_base/4f74e7b6d2722bc74364949b9c71d9cffc192756_20
```

Summary of the instance's disks:

On the host	Inside the instance		
File	**Device**	**Mountpoint**	**Use**
disk	/dev/vda	/	Root file system
disk.local	/dev/vdb	/mnt	Second disk
disk.swap	/dev/vdc	-	Memory swap

You can also check the instance directly by using the **libvirt** API:

```
root@node2:~# virsh
```

List the running instances on the compute node (node2 here):

```
virsh # list
```

Result:

```
 Id Name                 State
----------------------------------
  5 instance-000000da    running
```

Show information on the instance (domain):

```
virsh # dominfo instance-000000da
```

Result:

```
Id:             5
Name:           instance-000000da
UUID:           37125fa1-d89f-4c0f-816d-2855bc01a2c5
OS Type:        hvm
State:          running
CPU(s):         2
CPU time:       25.2s
Max memory:     2097152 kB
Used memory:    2097152 kB
Persistent:     yes
Autostart:      disable
Managed save:   no
Security model: apparmor
Security DOI:   0
Security label: libvirt-37125fa1-d89f-4c0f-816d-2855bc01a2c5 (enforcing)
```

Let's check the KVM process running on the host:

```
root@node2:~# ps –ef
```

Result:

```
110       6712     1  1 10:56 ?        00:00:25 /usr/bin/kvm -S -M pc-1.0 -enable-kvm -m
2048 -smp 2,sockets=2,cores=1,threads=1 -name instance-000000da -uuid 37125fa1-d89f-4c0f-
816d-2855bc01a2c5 -nodefconfig -nodefaults -chardev sock-
et,id=charmonitor,path=/var/lib/libvirt/qemu/instance-000000da.monitor,server,nowait -mon
chardev=charmonitor,id=monitor,mode=control -rtc base=utc -no-shutdown -drive
file=/var/lib/nova/instances/instance-000000da/disk,if=none,id=drive-virtio-
disk0,format=qcow2,cache=none -device virtio-blk-pci,bus=pci.0,addr=0x4,drive=drive-virtio-
disk0,id=virtio-disk0,bootindex=1 -drive file=/var/lib/nova/instances/instance-
000000da/disk.local,if=none,id=drive-virtio-disk1,format=qcow2,cache=none -device virtio-
blk-pci,bus=pci.0,addr=0x5,drive=drive-virtio-disk1,id=virtio-disk1 -drive
file=/var/lib/nova/instances/instance-000000da/disk.swap,if=none,id=drive-virtio-
disk2,format=qcow2,cache=none -device virtio-blk-pci,bus=pci.0,addr=0x6,drive=drive-virtio-
disk2,id=virtio-disk2 -netdev tap,fd=19,id=hostnet0 -device virtio-net-
pci,netdev=hostnet0,id=net0,mac=fa:16:3e:45:97:11,bus=pci.0,addr=0x3 -chardev
file,id=charserial0,path=/var/lib/nova/instances/instance-000000da/console.log -device isa-
serial,chardev=charserial0,id=serial0 -chardev pty,id=charserial1 -device isa-
serial,chardev=charserial1,id=serial1 -usb -device usb-tablet,id=input0 -vnc 127.0.0.1:0 -k
en-us -vga cirrus -device virtio-balloon-pci,id=balloon0,bus=pci.0,addr=0x7
```

Let's check the instance's network interface on the host:

```
root@node2:~# ip address
```

Result:

```
5: vnet0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master br5
   link/ether fe:16:3e:45:97:11 brd ff:ff:ff:ff:ff:ff
```

Or:

```
root@node2:~# brctl show
```

Result:

```
bridge name     bridge id               STP enabled     interfaces
br5             8000.fa163e5f9ba8       no              vlan5
                                                        vnet0  ← The instance's network interface
```

On the host, the network interface is **vnet0** (connected on brige **br5** which is connected on interface **vlan5** which is connected on physical interface **eth1**). In the guest, the network interface is **eth0** (connected on **vnet0**).

In case of problems, check the logs in **/var/log/nova/nova-compute.log** or **/var/log/libvirt**.

Reboot

If you want to reboot the instance:

```
pdupond@client1:~$ nova reboot i1
```

During some seconds, the status will be "REBOOT".

```
pdupond@client1:~$ nova list
```

Result:

```
+--------------------------------------+------+--------+-------------------------------+
|                  ID                  | Name | Status |            Networks           |
+--------------------------------------+------+--------+-------------------------------+
| 37125fa1-d89f-4c0f-816d-2855bc01a2c5 | i1   | REBOOT | vlan5=10.0.5.3                |
+--------------------------------------+------+--------+-------------------------------+
```

Floating IPs

Reserve

First we have to reserve some floating IPs for our tenant (**pdupond** here). Let's reserve (but not yet associate) 3 floating IPs: 2 from **pool1**, and 1 from **pool2**:

First floating IP:

```
pdupond@client1:~$ nova floating-ip-create pool1
```

Result:

```
+---------------+-------------+----------+-------+
|       Ip      | Instance Id | Fixed Ip |  Pool |
+---------------+-------------+----------+-------+
| 192.168.1.245 | None        | None     | pool1 |
+---------------+-------------+----------+-------+
```

You will receive the first floating IP not yet reserved by a tenant (192.168.1.245 in our case). 192.168.1.225 → 192.168.1.244 have already been reserved by other tenants.

Second floating IP:

```
pdupond@client1:~$ nova floating-ip-create pool1
```

Result:

```
+---------------+-------------+----------+-------+
|       Ip      | Instance Id | Fixed Ip |  Pool |
+---------------+-------------+----------+-------+
| 192.168.1.246 | None        | None     | pool1 |
+---------------+-------------+----------+-------+
```

Third floating IP:

```
pdupond@client1:~$ nova floating-ip-create pool2
```

Result:

```
+--------------+-------------+----------+-------+
|      Ip      | Instance Id | Fixed Ip |  Pool |
+--------------+-------------+----------+-------+
| 192.168.1.97 | None        | None     | pool2 |
+--------------+-------------+----------+-------+
```

Let's list the floating IPs which now are reserved to the tenant **pdupond**:

```
pdupond@client1:~$ nova floating-ip-list
```

Result:

```
+---------------+-------------+----------+-------+
|       Ip      | Instance Id | Fixed Ip |  Pool |
+---------------+-------------+----------+-------+
| 192.168.1.245 | None        | None     | pool1 |
| 192.168.1.246 | None        | None     | pool1 |
| 192.168.1.97  | None        | None     | pool2 |
+---------------+-------------+----------+-------+
```

Associate

Only now we can associate one (or more) floating IP to the instance. Let's associate 192.168.1.245 to our instance:

```
pdupond@client1:~$ nova add-floating-ip i1 192.168.1.245
```

Let's check:

```
pdupond@client1:~$ nova list
```

Result:

```
+--------------------------------------+------+--------+-------------------------------+
|                  ID                  | Name | Status |            Networks           |
+--------------------------------------+------+--------+-------------------------------+
| 37125fa1-d89f-4c0f-816d-2855bc01a2c5 | i1   | ACTIVE | vlan5=10.0.5.3, 192.168.1.245 |
+--------------------------------------+------+--------+-------------------------------+
```

On the network node (node1 here), NAT rules (one SNAT, one DNAT) have been configured linking the floating IP (192.168.1.245) and the fixed IP (10.0.5.3):

```
root@node1:~# iptables -nL -t nat
```

Result:

```
DNAT       all  --  0.0.0.0/0            192.168.1.245       to:10.0.5.3
SNAT       all  --  10.0.5.3             0.0.0.0/0           to:192.168.1.245
```

Now you can login into the instance with the floating IP from any host connected on the public network:

```
pdupond@client1:~$ ssh -i pdupond.pem ubuntu@192.168.1.245
```

Result:

```
ubuntu@i1:~$
```

Or from a computer on the internet (see my NAT/PAT rules in the router):

```
pdupond@client2:~$ ssh -p 2245 -i pdupond.pem ubuntu@lc2.louvrex.net
```

Result:

```
ubuntu@i1:~$
```

If the public network would have had internet IP addresses (see topology n° 2), you could connect to the instance directly with the floating IP.

Later, I can de-associate the floating IP from the instance and eventually associate it to another instance of tenant **pdupond**.

Volumes

Create

Let's create a volume of 2 GB for our tenant **pdupond**:

```
pdupond@client1:~$ nova volume-create --display_name v1 2
```

We gave the name "v1" to our volume.

After some seconds, the volume will be available:

```
pdupond@client1:~$ nova volume-list
```

Result:

```
+----+-----------+--------------+------+-------------+-------------+
| ID |   Status  | Display Name | Size | Volume Type | Attached to |
+----+-----------+--------------+------+-------------+-------------+
| 7  | available | v1           | 2    | None        |             |
+----+-----------+--------------+------+-------------+-------------+
```

On node1 (our volume node), you can now see the new volume as a LVM logical volume (LV) of 2 GB:

```
root@node1:~# lvdisplay
```

Result:

```
  --- Logical volume ---
  LV Name                /dev/nova-volumes/volume-00000007
  VG Name                nova-volumes
  LV UUID                5Q9ev1-X70z-AP7c-gGgp-j0Wx-eQ19-112e5a
  LV Status              available
  LV Size                2.00 GiB
  Block device           252:10
```

In case of problems, check the logs in **/var/log/nova/nova-volume.log**.

Attach

Let's attach the volume (id: 7) to our running instance (name: i1):

```
pdupond@client1:~$ nova volume-attach i1 7 /dev/vdd
```

We chose **vdd** as device because **vda**, **vdb**, and **vdc** are already in use. If you specify **vde** or above, it is still **vdd** which will be seen inside the instance.

During some seconds, the volume is in status "attaching":

```
pdupond@client1:~$ nova volume-list
```

Result:

```
+----+-----------+--------------+------+-------------+-------------+
| ID |   Status  | Display Name | Size | Volume Type | Attached to |
+----+-----------+--------------+------+-------------+-------------+
| 7  | attaching | v1           | 2    | None        |             |
+----+-----------+--------------+------+-------------+-------------+
```

After some seconds, the volume is attached to the instance:

```
pdupond@client1:~$ nova volume-list
```

Result:

```
+----+--------+--------------+------+-------------+--------------------------------------+
| ID | Status | Display Name | Size | Volume Type |             Attached to              |
+----+--------+--------------+------+-------------+--------------------------------------+
| 7  | in-use | v1           | 2    | None        | 37125fa1-d89f-4c0f-816d-2855bc01a2c5 |
+----+--------+--------------+------+-------------+--------------------------------------+
```

There is now an iSCSI session between the compute node (node2 here) and the storage node (node1 here).

From node2:

```
root@node2:~# iscsiadm -m session
```

Result:

```
tcp: [2] 172.16.1.201:3260,1 iqn.2010-10.org.openstack:volume-00000007
```

In the instance, you can see the new disk:

```
ubuntu@i1:~$ sudo fdisk -l
```

Result:

```
Disk /dev/vda: 21.5 GB, 21474836480 bytes
Disk /dev/vdb: 21.5 GB, 21474836480 bytes
Disk /dev/vdc: 2147 MB, 2147483648 bytes
Disk /dev/vdd: 2147 MB, 2147483648 bytes ← The new disk
```

Let's partition, format and mount that new disk.

Partitioning:

```
ubuntu@i1:~$ sudo cfdisk /dev/vdd
```

Display the partition:

```
ubuntu@i1:~$ sudo fdisk -l
```

Result:

```
Disk /dev/vdd: 2147 MB, 2147483648 bytes
Device Boot         Start         End      Blocks   Id  System
/dev/vdd1              63     4194303     2097120+  83  Linux
```

Format:

```
ubuntu@i1:~$ sudo mkfs.ext4 /dev/vdd1
```

Mount on the /ebs directory:

```
ubuntu@i1:~$ sudo mkdir /ebs
ubuntu@i1:~$ mount /dev/vdd1 /ebs
```

Let's check the final result:

```
ubuntu@i1:~$ df -h
```

Result:

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        20G  670M   19G   4% /            → local storage (on the compute node)
/dev/vdb         20G  173M   19G   1% /mnt         → local storage (on the compute node)
/dev/vdd1       2.0G   64M  1.9G   4% /ebs         → remote storage (on the volume node)
```

In case of problems, check the logs in **/var/log/nova/nova-compute.log**.

Detach

To detach the volume:

```
pdupond@client1:~$ nova volume-detach i1 7
```

To terminate the instance:

```
pdupond@client1:~$ nova delete i1
```

The volume disk (on the volume node) will not be deleted, just detached from the instance. But the local disks of the instance (**disk**, **disk.local**, and **disk.swap** on the compute node) will be deleted.

Volume disks are "persistent"; local disks are "ephemeral".

No need to detach a volume from an instance before deleting it. It will be made automatically.

Snapshot

To back up a volume, you can make a snapshot of it:

```
pdupond@client1:~$ nova volume-snapshot-create 7
```

Let's check:

```
pdupond@client1:~$ nova volume-snapshot-list
```

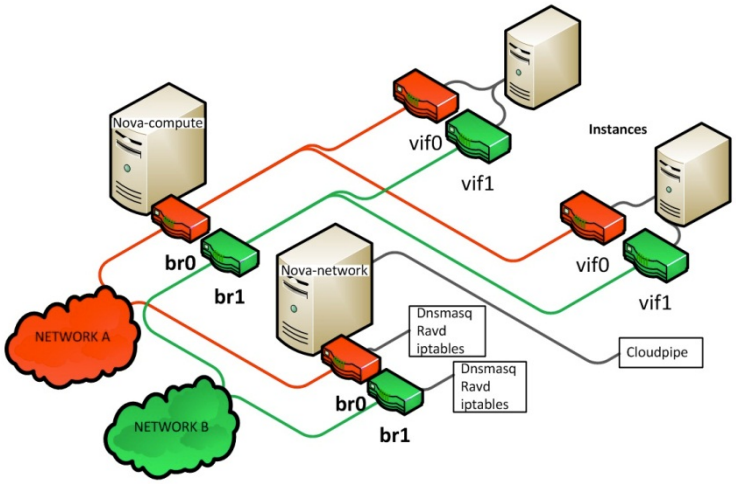
Result:

```
+----+-----------+-----------+--------------+------+
| ID | Volume ID |   Status  | Display Name | Size |
+----+-----------+-----------+--------------+------+
| 1  | 7         | available | None         | 2    |
+----+-----------+-----------+--------------+------+
```

From that volume snapshot, you can create a new volume.

The volume snapshot is also a LV on the storage node.

Instances with Two Network Interfaces (multinic)



© OpenStack

Reserve

Let's reserve a second network/VLAN to our tenant/project (name: pdupond, id: f6a1). Let's pick the first free network/VLAN: 10.0.6.0/24 (VLAN6):

```
root@node1:~# nova-manage network modify --fixed_range 10.0.6.0/24 --project f6a1
    --host node1
```

Remark: Please have a look at bug 952176 (Cannot associate a second network/vlan to a tenant with "nova-manage network modify": https://bugs.launchpad.net/nova/+bug/952176).

Let's check all the networks:

```
root@node1:~# nova-manage network list
```

Result:

```
id   IPv4          IPv6   start address   DNS1   DNS2   VlanID   project   uuid
1    10.0.1.0/24   None   10.0.1.3        None   None   1        1fc7      b0c2  → In-use
2    10.0.2.0/24   None   10.0.2.3        None   None   2        12b6      0dfa  → In-use
3    10.0.3.0/24   None   10.0.3.3        None   None   3        eea2      1810  → In-use
4    10.0.4.0/24   None   10.0.4.3        None   None   4        c1ab      a450  → In-use
5    10.0.5.0/24   None   10.0.5.3        None   None   5        f6a1      301d  → pdupond
6    10.0.6.0/24   None   10.0.6.3        None   None   6        f6a1      f62c  → pdupond
7    10.0.7.0/24   None   10.0.7.3        None   None   7        None      f9c5  → Free
8    10.0.8.0/24   None   10.0.8.3        None   None   8        None      5308  → Free
9    10.0.9.0/24   None   10.0.9.3        None   None   9        None      689b  → Free
10   10.0.10.0/24  None   10.0.10.3       None   None   10       None      d8ce  → Free
```

Boot

Let's start again an instance:

```
pdupond@client1:~$ nova boot --flavor medium --image precise --key_name pdupond
    --security_groups default i2
```

You can see that two IPs in two different networks/VLANs have been distributed to our instance:

```
pdupond@client1:~$ nova list
```

Result:

```
+--------------------------------------+------+--------+--------------------------------+
|                  ID                  | Name | Status |            Networks            |
+--------------------------------------+------+--------+--------------------------------+
| 9c2b77d5-59d5-45cb-8036-07e2b86f9997 | i2   | ACTIVE | vlan5=10.0.5.3; vlan6=10.0.6.3 |
+--------------------------------------+------+--------+--------------------------------+
```

The IP 10.0.5.3 is the same than the one for instance "i1" because we deleted "i1" and its fixed IP became free again.

If we check the interfaces on the host, we can see two **vnet** interfaces:

```
root@node2:~# ifconfig
```

Result:

```
vnet0     Link encap:Ethernet  HWaddr fe:16:3e:3e:e0:e5
          inet6 addr: fe80::fc16:3eff:fe3e:e0e5/64 Scope:Link

vnet1     Link encap:Ethernet  HWaddr fe:16:3e:77:d9:85
          inet6 addr: fe80::fc16:3eff:fe77:d985/64 Scope:Link
```

We can check the bridges as well:

```
root@node2:~# brctl show
```

Result:

```
bridge name     bridge id               STP enabled     interfaces
br5             8000.fa163e5f9ba8       no              vlan5
                                                        vnet0 → first interface for i2
br6             8000.fa163e1badc0       no              vlan6
                                                        vnet1 → second interface for i2
```

For each bridge, there is also a DHCP server (dnsmasq) running on the network node:

```
root@node1:~# ps -ef
```

Result:

```
root   2403  2402  0 /usr/sbin/dnsmasq --pid-file=nova-br1.pid --listen-address=10.0.1.1
root   2566  2565  0 /usr/sbin/dnsmasq --pid-file=nova-br2.pid --listen-address=10.0.2.1
root   2725  2724  0 /usr/sbin/dnsmasq --pid-file=nova-br3.pid --listen-address=10.0.3.1
root   2876  2875  0 /usr/sbin/dnsmasq --pid-file=nova-br4.pid --listen-address=10.0.4.1
root   3035  3034  0 /usr/sbin/dnsmasq --pid-file=nova-br5.pid --listen-address=10.0.5.1
root   3197  3196  0 /usr/sbin/dnsmasq --pid-file=nova-br6.pid --listen-address=10.0.6.1
```

Let's check the interfaces inside the instance:

```
pdupond@node1:~$ ssh -i pdupond.pem ubuntu@10.0.5.3
```

Result:

```
ubuntu@i2:~$ ifconfig
```

Result:

```
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0

eth0      Link encap:Ethernet  HWaddr fa:16:3e:3e:e0:e5
          inet addr:10.0.5.3  Bcast:10.0.5.255  Mask:255.255.255.0
```

Only one interface is configured (eth0), but two are present (eth0 and eth1):

```
ubuntu@i2:~$ ip address
```

Result:

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo

2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether fa:16:3e:3e:e0:e5 brd ff:ff:ff:ff:ff:ff
    inet 10.0.5.3/24 brd 10.0.5.255 scope global eth0

3: eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether fa:16:3e:77:d9:85 brd ff:ff:ff:ff:ff:ff
```

Let's configure the second interface (eth1):

```
ubuntu@i2:~$ cat /etc/network/interfaces
```

Result:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp
```

We will add eth1 with the same configuration as eth0 (DHCP):

```
ubuntu@i2:~$ sudo vi /etc/network/interfaces
```

That gives now:

```
ubuntu@i2:~$ cat /etc/network/interfaces
```

Result:

```
auto lo
iface lo inet loopback

auto eth0                       ← First interface
iface eth0 inet dhcp

auto eth1                       ← Second interface
iface eth1 inet dhcp
```

Let's enable the second interface:

```
ubuntu@i2:~$ sudo ifup eth1
```

And let's check:

```
ubuntu@i2:~$ ifconfig
```

Result:

```
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0

eth0      Link encap:Ethernet  HWaddr fa:16:3e:3e:e0:e5
          inet addr:10.0.5.3  Bcast:10.0.5.255  Mask:255.255.255.0

eth1      Link encap:Ethernet  HWaddr fa:16:3e:77:d9:85
          inet addr:10.0.6.3  Bcast:10.0.6.255  Mask:255.255.255.0
```

Boot with specified IPs

You can also decide which IPs you want for your instance (if not yet distributed). Let's say you want to boot an instance with IP 10.0.5.15 (VLAN5 / id: 301d) and IP 10.0.6.15 (VLAN6 / id: f62c). This is the command to be launched if you have **admin** or **netadmin** rights:

```
pdupond@client1:~$ nova boot --flavor medium --image precise --key_name pdupond
    --security_groups default
    --nic net-id=301d,v4-fixed-ip=10.0.5.15
    --nic net-id=f62c,v4-fixed-ip=10.0.6.15 i3
```

By doing this, you tell the DHCP servers on the network node which IPs to distribute to the instance.

This is of course valid for an instance with only one network interface.

Customize an Image

If you want your own private image customized with specific applications, the easiest is to launch an instance with a generic OS image, then to install your applications inside the instance, then to make a **snapshot of the instance**.

Let's say we want a LAMP (Apache/PHP and MySQL) web server image (appliance).

Step 1: Launch an instance with a generic OS image:

```
pdupond@client1:~$ nova boot --flavor medium --image precise --key_name pdupond
    --security_groups web-server i4
```

We launched it with the "web-server" security group (firewall rules) to be able to access the web server from the internet.

Step 2: Customize the instance:

Upgrade the OS to its latest version:

```
ubuntu@i4:~$ sudo aptitude update
ubuntu@i4:~$ sudo aptitude safe-upgrade
```

- Install the LAMP applications:

```
ubuntu@i4:~$ sudo tasksel
```

And chose "LAMP".

- Install PhpMyAdmin (MySQL Web interface):

```
ubuntu@i4:~$ sudo aptitude install phpmyadmin
```

- Add your own account:

```
ubuntu@i4:~$ sudo adduser pdupond
```

And adapt the configuration for the public SSH key to be used with your account (**pdupond**) in place of the default account (**ubuntu**):

```
ubuntu@i4:~$ sudo vi /etc/cloud/cloud.cfg
```

Replace:

```
user: ubuntu
```

By:

```
user: pdupond
```

Now, when you will launch an instance with your customized image, you will have to login with user **pdupond**.

Add **pdupond** in the **sudoers** file to be able to launch commands with root privileges.

Step 3: Snapshot the instance:

Let's create an image called "pdupond-lamp-v1":

```
pdupond@client1:~$ nova image-create i4 pdupond-lamp-v1
```

During some seconds to some minutes, you will see the status as "SAVING":

```
pdupond@client1:~$ nova image-list
```

Result:

```
+-------+----------------------+-----------+----------------------+
| ID    | Name                 | Status    | Server               |
+-------+----------------------+-----------+----------------------+
| 29da  | pdupond-lamp-v1      | SAVING    | fa59                 |
| 4365  | precise              | ACTIVE    |                      |
+-------+----------------------+-----------+----------------------+
```

When the snapshot is finished, you will see the status has become "ACTIVE":

```
+-------+----------------------+-----------+----------------------+
| ID    | Name                 | Status    | Server               |
+-------+----------------------+-----------+----------------------+
| 29da  | pdupond-lamp-v1      | ACTIVE    | fa59                 |
| 4365  | precise              | ACTIVE    |                      |
+-------+----------------------+-----------+----------------------+
```

That image is now available in Glance:

```
pdupond@client1:~$ glance show 29da730e-9598-4522-9f8b-b0fac8537110
```

Result:

```
URI: http://localhost:9292/v1/images/29da730e-9598-4522-9f8b-b0fac8537110
Id: 29da730e-9598-4522-9f8b-b0fac8537110
Public: No
Protected: No
Name: pdupond-lamp-v1
Status: active
Size: 1200357376
Disk format: qcow2
Container format: ovf
Minimum Ram Required (MB): 0
Minimum Disk Required (GB): 0
Owner: f6a1c12cf10b43e68aa0952470f1bb56
Property 'instance_uuid': fa598439-7f98-4538-b9d0-0f1ed585315a
Property 'image_location': snapshot
Property 'image_state': available
Property 'user_id': f8c27de796914e56b5b4a2bc3f6a2432
Property 'image_type': snapshot
Property 'architecture': x86_64
Property 'owner_id': f6a1c12cf10b43e68aa0952470f1bb56
```

You can notice that the image belongs to tenant **pdupond** (Owner: f6a1), the same tenant as the instance. That image is automatically set as "private" (Public: No). It means it can be used only by tenant **pdupond** (the other tenants will even not see it).

Let's try it:

```
pdupond@client1:~$ nova boot --flavor medium --image pdupond-lamp-v1 --key_name pdupond
    --security_groups web-server i5
```

Log into it with user **pdupond**:

```
pdupond@client1:~$ nova list
```

Result:

```
+--------------------------------------+------+--------+--------------------------------+
|                  ID                  | Name | Status |            Networks            |
+--------------------------------------+------+--------+--------------------------------+
| fa598439-7f98-4538-b9d0-0f1ed585315a | i4   | ACTIVE | vlan5=10.0.5.3; vlan6=10.0.6.3 |
| ce3b060c-9f28-4b24-a0e0-e2de2d3f88da | i5   | ACTIVE | vlan5=10.0.5.4; vlan6=10.0.6.4 |
+--------------------------------------+------+--------+--------------------------------+
```

The instance received IP 10.0.5.4:

```
pdupond@node1:~$ ssh -i pdupond.pem pdupond@10.0.5.4
```

Result:

```
pdupond@i5:~$
```

Let's associate a public IP (floating IP) to access the web server from the internet:

```
pdupond@client1:~$ nova add-floating-ip i5 192.168.1.246
```

Let's check:

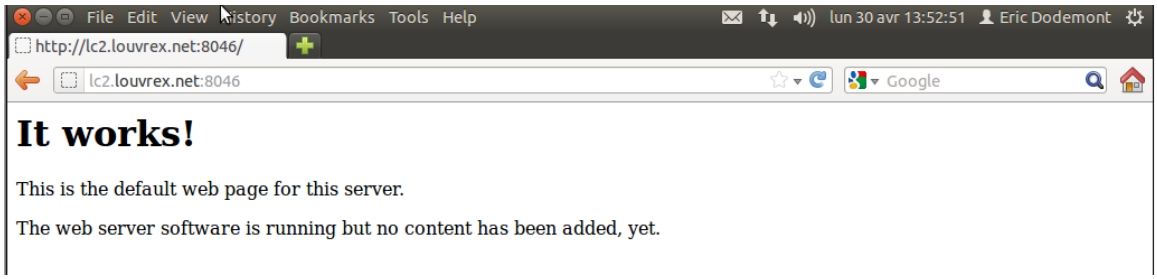
```
pdupond@client1:~$ nova list
```

Result:

```
+------+------+--------+-------------------------------------------------+
| ID   | Name | Status |                    Networks                     |
+------+------+--------+-------------------------------------------------+
| fa59 | i4   | ACTIVE | vlan5=10.0.5.4; vlan6=10.0.6.4                  |
| ce3b | i5   | ACTIVE | vlan5=10.0.5.3, 192.168.1.246; vlan6=10.0.6.3 |
+------+------+--------+-------------------------------------------------+
```

And with a web browser, connect to http://192.168.1.246 (from the public network) or to http://lc2.louvrex.net:8046 (from the internet).

See "Internet and DNS (NAT/PAT rules)": **lc2.louvrex.net:8046** is mapped to **192.168.1.246:80**.



Appendix A: Description of some Nova package dependencies

vlan: User mode programs to enable VLANs on your ethernet devices. This package contains the user mode programs you need to add and remove VLAN devices from your ethernet devices. A typical application for a VLAN enabled box is a single wire firewall, router or load balancer. You need a VLAN Linux kernel for this. Linux kernel versions >= 2.4.14 have VLAN support.

bridge-utils: Utilities for configuring the Linux Ethernet bridge. This package contains utilities for configuring the Linux Ethernet bridge in Linux. The Linux Ethernet bridge can be used for connecting multiple Ethernet devices together. The connecting is fully transparent: hosts connected to one Ethernet device see hosts connected to the other Ethernet devices directly.

dnsmasq-base: Small caching DNS proxy and DHCP/TFTP server. This package contains the dnsmasq executable and documentation, but not the infrastructure required to run it as a system daemon. For that, install the dnsmasq package.

dnsmasq-utils: Utilities for manipulating DHCP leases. Small utilities to query a DHCP server's lease database and remove leases from it. These programs are distributed with dnsmasq and may not work correctly with other DHCP servers.

qemu-utils: Qemu utilities. This package provides some utilities for which full qemu-kvm is not needed, in particular qemu-nbd and qemu-img.

iptables: Administration tools for packet filtering and NAT. These are the user-space administration tools for the Linux kernel's netfilter and iptables. netfilter and iptables provide a framework for stateful and stateless packet filtering, network and port address translation, and other IP packet manipulation. The framework is the successor to ipchains. netfilter and iptables are used in applications such as Internet connection sharing, firewalls, IP accounting, transparent proxying, advanced routing and traffic control.

open-iscsi: High performance, transport independent iSCSI implementation. iSCSI is a network protocol standard that allows the use of the SCSI protocol over TCP/IP networks. This implementation follows RFC3720.

python-libvirt: libvirt Python bindings. Libvirt is a C toolkit to interact with the virtualization capabilities of recent versions of Linux (and other OSes). The library aims at providing a long term stable C API for different virtualization mechanisms. It currently supports QEMU, KVM, XEN, OpenVZ, LXC, and VirtualBox.

libvirt-bin: Programs for the libvirt library. This package contains the supporting binaries to use with libvirt.

lvm2: The Linux Logical Volume Manager. This is LVM2, the rewrite of The Linux Logical Volume Manager. LVM supports enterprise level volume management of disk and disk subsystems by grouping arbitrary disks into volume groups. The total capacity of volume groups can be allocated to logical volumes, which are accessed as regular block devices.

tgt: Linux SCSI target user-space tools. The Linux target framework (tgt) allows a Linux system to provide SCSI devices (targets) over networked SCSI transports. Tgt consists of kernel modules, user-space daemon, and user-space. This package contains the user-space daemon and tools; a recent Linux kernel is required for the modules. This package includes drivers for: FCoE (Fibre Channel over Ethernet), iSCSI (SCSI over IP), and iSER (iSCSI over RDMA, using Infiniband).

python-boto: Python interface to Amazon's Web Services. Boto is a Python interface to the infrastructure services available from Amazon. Boto supports the following services: Elastic Compute

Cloud (EC2), SimpleDB, Simple Storage Service (S3), CloudFront, Simple Queue Service (SQS), Elastic MapReduce, Relational Database Service (RDS).

python-carrot: AMQP messaging queue framework. AMQP is the Advanced Message Queuing Protocol, an open standard protocol for message orientation, queuing, routing, reliability and security. The aim of carrot is to make messaging in Python as easy as possible by providing a high-level interface for producing and consuming messages. At the same time it is a goal to re-use what is already available as much as possible. carrot has pluggable messaging back-ends, so it is possible to support several messaging systems. Currently, there is support for AMQP (py-amqplib) and STOMP (python-stomp). There is also a in-memory backend for testing purposes that uses the Python queue module.

python-kombu: AMQP Messaging Framework for Python. The aim of Kombu is to make messaging in Python as easy as possible by providing an idiomatic high-level interface for the AMQP protocol. It is meant to replace the carrot library by providing a compatibility layer. Features: Allows application authors to support several message server solutions by using pluggable transports; Supports automatic encoding, serialization and compression of message payloads; The ability to ensure that an operation is performed by gracefully handling connection and channel errors.

python-sqlalchemy: SQL toolkit and Object Relational Mapper for Python. SQLAlchemy is an SQL database abstraction library for Python.

python-amqplib: Simple non-threaded Python AMQP client library. Python client for the Advanced Message Queuing Protocol (AMQP) 0-8, featuring basic messaging functionality and SSL support.