Implementing open source cloud platform as an enterprise solution and develop a tool to manage the entire ecosystem

Supervisor: Prof. Dr. Mumit Khan

Conducted by:
Md. Imran Hossain
08101025

School of Engineering and Computer Science
BRAC University

Submitted on
April 2012

BRAC University, Dhaka, Bangladesh

DECLARATION

I, Md. Imran Hossain, student of School of Engineering and Computer Science, Brac University represent my thesis work on **"Implementing open source cloud platform as an enterprise solution and develop a tool to manage the entire ecosystem"** as requirement of completion of bachelor degree. This thesis research was performed under supervision of Dr. Mumit Khan, Professor, BRAC University, Dhaka, Bangladesh.

ACKNOWLEDGMENTS

I am thankful to Almighty for blessing me with the patience and knowledge and the opportunity to learn something new.

I am heartily thankful to my thesis supervisor Dr. Mumit Khan for all his belief in me. In my thesis period he has allowed me to work in an organization who are also looking forward to be the first time ever private cloud company in Bangladesh which has been a great opportunity for me to accomplish my project.

I am thankful to Abdur Rahman Adnan, Lecturer, Brac University, he has been a great help from very beginning of my thesis and has helped a lot.

I am thankful to the directors of Amadeyr Cloud Ltd. and all it's employees for giving me the chance to work their, without their help completing my thesis could be harder with such power constraints. They have also provided professional equipment for the cloud deployment.

And last but not the least I thank my family and all my friends for supporting me in times of need.

Abstract

Implementing open source cloud platform as an enterprise solution and develop a tool to manage the entire ecosystem

The fundamental idea behind my thesis work is to use open source technologies and build a fully workable enterprise solution. Another part was to develop a management tool for the entire ecosystem, which can be deployed on web servers for the admins and users to use.

In the very beginning I researched on different open source cloud platforms. Tried deploying them in small scale. I also looked into some professional cloud service providers, like Amazon AWS and Rackspace Cloud. I found Amazon is the pioneer among the cloud providers who are serving infrastructure as a service and Rackspace Cloud in the second position on the number of cloud users basis. So, there most complete open source solution with Amazon AWS cloud is Eucalyptus cloud ecosystem. I used Eucalyptus to deploy the cloud infrastructure.

To make deploying cloud infrastructure easier on small scale I also wrote a script, which will setup a Eucalyptus cloud infrastructure with just few commands.

Another part of my thesis was to develop a cloud management tool. I developed a new tool using a very popular open source python library Boto that is used to manage Amazon EC2 and S3 services.

This report contains all the steps taken to deploy the cloud platform, challenges and the steps taken to resolve issues and also discuss about the cloud management tool.

Table of contents

Technology Survey

The concept private cloud is getting very popular day by day. Sometimes it is because of privacy, sometimes it is for governance, sometimes the data is too big to move to the public cloud etc. There are few open source solutions out there to meet such demand of private cloud. However, I did some research on the available open source technologies. Just when I started looking into most popular Ubuntu Enterprise Cloud (UEC), which used Eucalyptus, Ubuntu declared that they were leaving Eucalyptus and moving to Openstack, the new comer of the cloud technology, backed up by Nasa and Rackspace Cloud along with many other companies and the number is now about 150 giant companies like, Cisco, Dell, HPCloud.

Then I had chosen the latest technology, which was Openstack at that time. I deployed their second release, Openstack Cactus, with two physical machines. At that time Openstack was in very rapid development and sometimes the structure of the platform was changing over night. So, it was really hard to go for production with such a platform. Then I looked at some other platforms, OpenNebula, Cloudstack.

Finally, I started looking into Eucalyptus, had read their reviews, and understood their platform architecture, network architecture and most importantly their APIs, which looked more promising than any other. They follow the De-facto standard APIs of Amazon AWS.

For professional deployment I had to do another research on necessary hardware. The official recommendation is to use at least two physical machines. I chose mid-level hardware for the deployment.

Hardware Configuration

Processor	Intel Quad Core Xeon Processor
Ram	4GB
Hard Disk	500GB
NIC	2 NICs

For API library, I went through the Java Typica; it works with both AWS and Eucalyptus, developed by Eucalyptus employees. But the development stopped for a long time. Then I looked into another Python library, Boto and used it to develop the web management tool.

Chapter 1: Introduction

1.1: Cloud Computing

Cloud computing is a computing model, where resources such as computing power, storage, network and software are abstracted and provided as services on the internet in a remotely accessible fashion. On-demand availability, ease of provisioning, dynamic and virtually infinite scalability are some of the key attributes of cloud computing.

1.2: Types of Cloud Computing

There are three types of cloud computing:

1. Public Cloud
2. Private Cloud
3. Hybrid Cloud

In the public cloud (or external cloud) computing resources are dynamically provisioned over the Internet via Web applications or Web services from an off-site third-party provider. Third parties run public clouds, and applications from different customers are likely to be mixed together on the cloud's servers, storage systems, and networks.

Private cloud (or internal cloud) refers to cloud computing on private networks. Private clouds are built for the exclusive use of one client, providing full control over data, security, and quality of service. Private clouds can be built and managed by a company's own IT organization or by a cloud provider.

A hybrid cloud environment combines multiple public and private cloud models. Hybrid clouds introduce the complexity of determining how to

distribute applications across both a public and private cloud.

1.3: Layers of Cloud Computing

SaaS (Software-as-a-Service) allows users to run applications remotely from the cloud.

Infrastructure-as-a-service (IaaS) refers to computing resources as a service.
This includes virtualized computers with guaranteed processing power and reserved bandwidth for storage and Internet access.

Platform-as-a-Service (PaaS) is similar to IaaS, but also includes operating systems and required services for a particular application. In other words, PaaS is IaaS with a custom software stack for the given application.

1.4: Key technologies

Cloud computing includes virtualization, web service and service oriented architecture. There are other technologies like service flows and workflows, and finally Web 2.0 and mashup.

1.4.1: Virtualization

This is the ability to run multiple operating systems on a single physical system and share the underlying hardware resources. A virtual machine (VM) is a software implementation of a machine that executes programs like a physical machine. Each VM includes its own kernel, operating system, supporting libraries and applications. A hypervisor provides a uniform abstraction of the underlying physical machine.

Multiple VMs can execute simultaneously on a single hypervisor. The decoupling of the VM from the underlying physical hardware allows the same VM to be started on different physical machines. Thus virtualization is seen as an enabler for cloud computing, allowing the cloud-computing provider the necessary flexibility to move and allocate the computing resources requested by the user wherever the physical resources are available.

1.4.2: Web Service and Service Oriented Architecture

Web Services and Service Oriented Architecture (SOA) represent the base technologies for cloud computing. Cloud services are typically designed as Web services, which follow industry standards including WSDL (Web Services Description Language), SOAP (Simple Object Access Protocol), and UDDI (Universal Description, Discovery and Integration). A SOA organizes and manages Web services inside. A SOA also includes a set of cloud services, which are available on various distributed platforms.

1.5 Objective of the thesis

First of all I need to understand the concept of cloud computing getting familiar with new technologies, the usage of those technologies and then to focus on a specific platform. Deploying a cloud platform as an enterprise solution using existing open source technology is the fundamental goal of the thesis.

Another part of the thesis is to develop a cloud management tool for Eucalyptus private cloud. This will also allow me to understand how it works internally, how to communicate with it and understand what kind of solutions can be provide using this technology.

Chapter 2: Eucalyptus

2.1: Introduction

Eucalyptus (Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems) is an open source Linux based software architecture that provides an EC2-compatible cloud computing platform and S3-compatible cloud storage platform. It implements scalable, efficient-enhancing and private and hybrid clouds within and organization's IT infrastructure. It gives an Infrastructure as a Service (IaaS) solution.

Eucalyptus was developed to support the high performance computing (HPC). Eucalyptus can be deployed without modification on all major Linux OS distributions, including Ubuntu, RHEL/CentOS, openSUSE, and Debian.

2.2: Eucalyptus Features

For implementing, managing and maintaining the virtual machines, network and storage Eucalyptus has variety of features.

- SSH Key Management
- Image Management
- Linux-based VM Management
- IP Address Management
- Security Group Management
- Volume and Snapshot Management

2.3: Components of Eucalyptus

2.3.1: Cluster Controller (CC)

Cluster Controller manages the one or more Node controller and responsible for deploying and managing instances on them. It communicates with Node Controller and Cloud Controller simultaneously. CC also manages the

networking for the running instances under certain types of networking modes available in Eucalyptus.

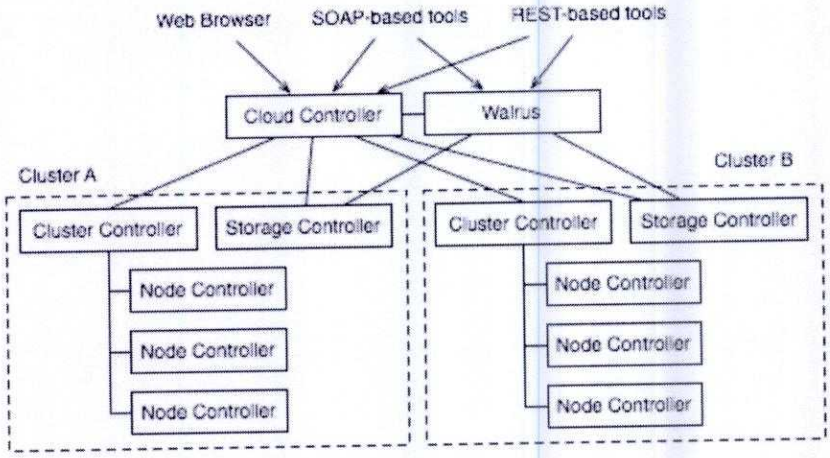


Figure: 2.1

2.3.2: Cloud Controller (CLC)

Cloud Controller is front end for the entire ecosystem. CLC provides an Amazon EC2/S3 compliant web services interface to the client tools on one side and interacts with the rest of the components of the Eucalyptus infrastructure on the other side.

2.3.3: Node Controller (NC)

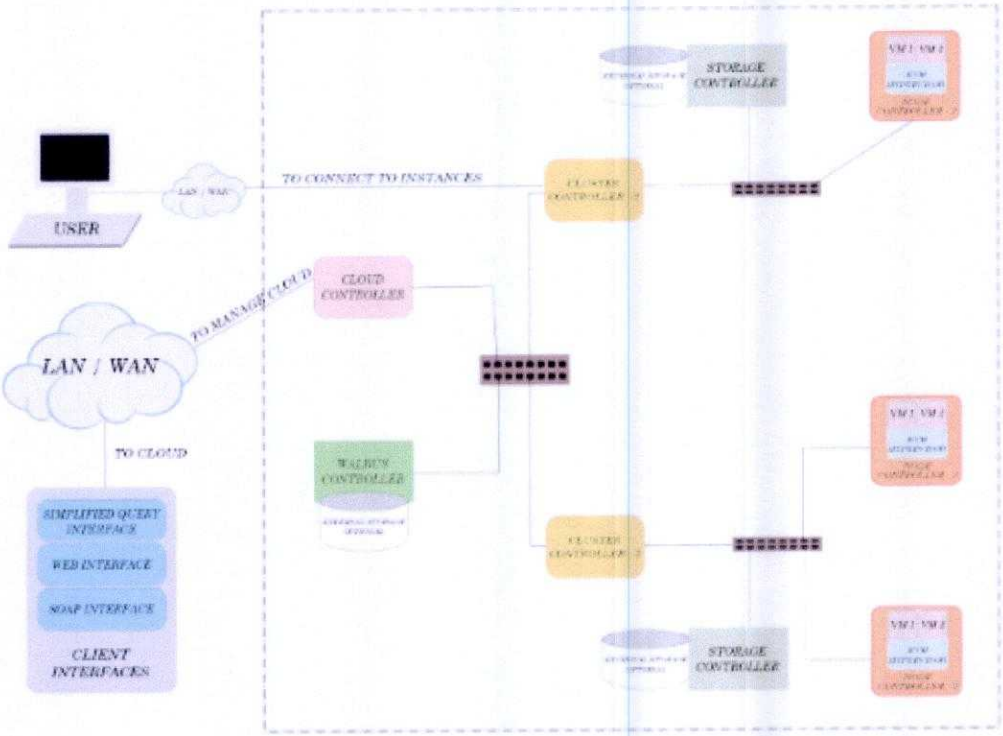
It is the basic component for Nodes. Node controller maintains the life cycle of the instances running on each nodes. Node Controller interacts with the OS, hypervisor and the Cluster Controller simultaneously.

2.3.4: Walrus Storage Controller (WS3)

Walrus Storage Controller is a simple file storage system. WS3 stores the machine images and snapshots. It also stores and serves files using S3 APIs.

2.3.5: Storage Controller (SC)

Allows the creation of snapshots of volumes. It provides persistent block storage over AoE or iSCSI to the instances.



EUCALYPTUS BASED CLOUD

Figure: 2.2

2.4: UEC Installation

For installation two PCs (Server1 and Server2) was needed for the cloud purpose and one PC is for client, which will also be serving for creating KVM images. Server2 and the client machine should have VT enabled, as it would be running all the VMs on Server2 and client PC would be using to create the necessary KVM images.

2.4.1: Required Configuration

	Server1	Server2	Client1
Functionality	CLC, CC, storage controller, and Walrus	NC	Bundling and Web UI Client
No. of NICs	eth0 - (Enterprise N/W) eth1 - (Eucalyptus N/W)	eth0 - (Enterprise N/W) eth1 - (Eucalyptus N/W)	eth0 - (Enterprise N/W) eth1 - (Eucalyptus N/W)
IP Addresses	eth0 - 192.168.1.102 (set during install) eth1 - 192.168.20.1 (set after install)	eth0 – 192.168.1.103 (set during install) eth1 - 192.168.20.2 (set after install)	eth0 – any address within the same same network eth1 – N/A
Hostname	server1.example.com	server2.example.com	client1.example.com
Gateway IP	192.168.1.1	192.168.20.1	192.168.1.1

2.4.2: Server1 Setup

- I booted the Ubuntu 11.04 64 bit Server CD/pen drive and from the graphical menu, selected Ubuntu Enterprise Cloud and follow the installation menu.
- I was using DHCP for the public network, so I just selected eth0 and let the network to be setup automatically. On another setup I had to put the ethernet as mentioned above.
- When the installation asked for the Cloud Controller address just kept it blank.
- For Server1 it installed the 'Cluster Controller', 'Walrus Storage Service', 'Cluster Controller' and 'Storage Controller'.
- Select eth1 for communication with nodes
- Eucalyptus cluster name – Cluster1 (on anything)
- Then selected an IP range to be used for the nodes, i.e. 192.168.1.10-192.168.1.99

2.4.2.1: Post Installation Setup

Then I edited **/etc/network/interfaces** and added static IP for eth1,

```
auto eth1
iface eth1 inet static
address 192.168.20.1
netmask 255.255.255.0
network 192.168.20.0
broadcast 192.168.20.255
```

Executed the following command to restart the networking,

```
localadmin@server1:~$ sudo /etc/init.d/networking restart
```

Updated and upgraded the Eucalyptus to get the latest version of it,

```
localadmin@server1:~$ sudo apt-get update
localadmin@server1:~$ sudo apt-get upgrade eucalyptus
```

Installed NTP package. Server1 was supposed to act as an NTP server for the nodes.

```
localadmin@server1:~$ sudo apt-get install ntp
```

Edited **/etc/ntp.conf** to make sure that the server was to serve time even for no connectivity to the internet. Added the following line to the file so that NTP server use its own clock source.

```
server 127.127.1.0
fudge 127.127.1.0 stratum 10
```

Restarted NTP server to make the changes active

```
localadmin@server1:~$ sudo /etc/init.d/ntp restart
```

Restarted the Cluster Controller

```
localadmin@server1:~$ sudo restart eucalyptus-cc CLEAN=1
```

[NTP stands for Network Time Protocol, and it is an Internet protocol used to synchronize the clocks of computers to some time reference]

2.4.3: Server2 Setup

- Booted Ubuntu 11.04 64 bit Server and selected 'Install Ubuntu Enterprise Cloud' and continued the basic installation process.
- For network setup I selected eth0 and configured it manually. I put the private IP to 192.168.20.2 and the gateway as 192.168.20.1.
- For UEC setup it asked certain configuration option. Sometimes it select the Cluster Controller by itself, but when it didn't do it I put the Cluster Controller address 192.168.20.1
- In cloud installation mode selected 'Node Controller'

2.4.3.1: Post Installation Setup

Changed static IP for eth1 by adding the few lines to **/etc/network/interfaces** so that it looks like following

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet manual

auto br0
iface br0 inet static
    address 192.168.20.2
    netmask 255.255.255.0
    network 192.168.20.0
    broadcast 192.168.20.255
    # gateway 192.168.20.1
    # dns-* options are implemented by the resolvconf
package, if installed
    dns-nameservers 192.168.1.1
    bridge_ports eth0
    bridge_fd 9
    bridge_hello 2
```

```
    bridge_maxage 12
    bridge_stp off

auto eth1
iface eth1 inet static
address 192.168.1.103
netmask 255.255.255.0
network 192.168.1.0
broadcast 192.168.1.255
gateway 192.168.1.1
```

Executed the following command to restart the networking,

```
localadmin@server1:~$ sudo /etc/init.d/networking restart
```

Updated and upgraded the Eucalyptus to get the latest version of it,

```
localadmin@server1:~$ sudo apt-get update
localadmin@server1:~$ sudo apt-get upgrade eucalyptus
```

Installed NTP package.

```
localadmin@server1:~$ sudo apt-get install ntp
```

Edited the file **/etc/ntp.conf** and add the following line

```
server 192.168.20.1
```

Restarted NTP server to make the changes active

```
localadmin@server1:~$ sudo /etc/init.d/ntp restart
```

Edited the file **/etc/eucalyptus/eucalyptus.conf** and made the following changes,

```
VNET_PUBINTERFACE="br0"
VNET_PRIVINTERFACE="br0"
```

```
VNET_BRIDGE="br0"
VNET_DHCPDAEMON="/usr/sbin/dhcpd3"
VNET_DHCPUSER="dhcpd"
VNET_MODE="MANAGED-NOVLAN"
```

Then executed the following command to restart the Node Controller to make all the changes active,

```
localadmin@server2:~$ sudo restart eucalyptus-nc
```

2.4.4: Setup CC's SSH public key to NC

On the Node Controller, set a password for the "eucalyptus" user,

```
localadmin@server2:~$ sudo passwd eucalyptus
```

On the Cluster Controller:

```
localadmin@server1:~$ sudo -u eucalyptus ssh-copy-id -I
~eucalyptus/.ssh/id_rsa.pub eucalyptus@192.168.20.2
```

Removed the password of the "eucalyptus" account from the Node,

```
localadmin@server1:~$ sudo passwd -d eucalyptus
```

2.4.5: Client Setup

Booted the 11.04 32/64 bit Desktop and installed it.

Also installed KVM on the client machine.

```
shaon@client:~$ sudo apt-get install qemu-kvm
```

2.4.5.1: Post Installation Setup

To administrate the cloud installed *euca2ools*

```
shaon@client:~$ sudo apt-get install euca2ools
```

2.4.6: Monitoring

- Logged in to the web interface from https://192.168.10.121:8443, default username is 'admin' and password is 'admin'.
- Downloaded the user credentials from *credential* tab and saved it to ~/.euca directory
- Extracted the credentials and *source* the **eucarc** script so that *euca2ools* can used this as environmental variables.

```
$ cd .euca
$ unzip xxxxxxxxx.zip
$ source eucarc
```

Verify that *euca2ools* can communicate with the UEC properly and all the services are running correctly run the following command,

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

It gave this output,

```
AVAILABILITYZONE    cluster1     192.168.1.102
AVAILABILITYZONE    |- vm types  free / max   cpu   ram
disk
AVAILABILITYZONE    |- m1.small  0001 / 0002   1    192
2
AVAILABILITYZONE    |- c1.medium 0001 / 0002   1    256
5
AVAILABILITYZONE    |- m1.large  0000 / 0001   2    512
10
AVAILABILITYZONE    |- m1.xlarge 0000 / 0001   2   1024
20
AVAILABILITYZONE    |- c1.xlarge 0000 / 0000   4   2048
20
```

If VCPUs are found 0000, then I used the following command on the Server1 to make sure that it finds the Node Controller and approve when prompts 192.168.20.2

```
localadmin@server1:~$ sudo euca_conf --discover-nodes
```

2.5: Creating Image For Eucalyptus

To create image it needed a raw HDD for the virtual machine. So, I set 5GB space for that.

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

Downloaded Ubuntu 11.04 Server from here,

```
wget http://releases.ubuntu.com/natty/ubuntu-11.04-
server-amd64.iso
```

To start the installation process, I booted a KVM instance with the OS installer ISO in the virtual CD-ROM and set up a VNC port at 0

```
sudo kvm -m 256 -cdrom ubuntu-11.04-server-amd64.iso -
drive file=server.img,if=scsi,index=0 -boot d -net nic -
net user -nographic  -vnc :0
```

Connected to the VM using the Client PC's IP with VNC port 0 and then finished the installation i.e. 10.10.10.4 :0

```
vncviewer 10.10.10.4 :0
```

Created a single ext4 partition mounted on '/' during the installation of Ubuntu. For virtual machine images, swap partition is not needed.

After finishing the installation re-launched the VM by executing the following command,

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

Now any packages can be added. So for the test purpose here I updated and upgraded the system.

```
sudo apt-get update
sudo apt-get upgrade
```

Installed the following packages as well

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

Removed the network persistent rules to make sure that the new network interface eth0 without creating any problem.

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

Closed the VM.

To upload the image on Eucalyptus, it needed to be an ext4 filesystem image.

To obtain an ext4 filesystem image, I did the following

```
$ sudo losetup  -f  server.img
$ sudo losetup -a
/dev/loop0: [0801]:16908388 ($filepath)
```

The name of the loop device was (/dev/loop0 in our setup) and $filepath is the path to the mounted .raw file.

Now it is needed to find out the starting sector of the partition

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

It showed an output like this

```
Disk /dev/loop0: 5368 MB, 5368709120 bytes
149 heads, 8 sectors/track, 8796 cylinders, total
10485760 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00072bd4
Device Boot Start End Blocks Id System
/dev/loop0p1 * 2048 10483711 5240832 83 Linux
```

This number should be multiplied by 512 to obtain the correct value. In this case: 2048 x 512 = 1048576

Unmounted the loop0 device:

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

Then mounted only the partition (/dev/loop0) of server.img which we had previously noted down, by adding the -o parameter with value previously calculated value

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

It showed a message like this

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

Copied the entire partition to a new .raw file

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

Now the ext4 filesystem image i.e serverfinal.img is ready.

Unmounted the loop0 device

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

2.5.1: Tweaking /etc/fstab

Then I have tweaked /etc/fstab to make it suitable for a cloud instance.

Loop mount the serverfinal.img

```
sudo mount -o loop serverfinal.img /mnt
```

Edited **/mnt/etc/fstab** and modified the line for mounting root partition (which may look like the following)

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

to

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

2.5.2: Kernel and Initrd

Copied the kernel and the initrd image from **/mnt/boot** to user home directory. These files were used later for creating and uploading a complete virtual image to Eucalyptus.

```
$ sudo cp /mnt/boot/vmlinuz-2.6.38-7-server /home/shaon
$ sudo cp /mnt/boot/initrd.img-2.6.38-7-server
/home/shaon
```

Unmount the Loop partition

```
$ sudo umount  /mnt
```

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

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

Now, all the components of the image got ready to be uploaded to Eucalyptus server.

2.6: Image Management in Eucalyptus

Managing images in Eucalyptus is generally done by Euca2ools distributed by Eucalyptus team.
All users may upload and register images (depending on access granted to them by the Eucalyptus administrator), but only the admin user may ever upload/register kernels or ramdisks.
At first I *source* the '**eucarc**' from **~/.euca** directory.

2.6.1: Registering Kernel Image

I executed the following commands to bundle and register the kernel image (vmlinuz-2.6.35-22-server)

```
shaon@client:~$ euca-bundle-image -i vmlinuz-2.6.32-28-
generic --kernel true
shoan@client:~$ euca-upload-bundle -b mybucket -m
/tmp/vmlinuz-2.6.32-28-generic.manifest.xml
shaon@client:~$ euca-register mybucket/vmlinuz-2.6.32-28
generic.manifest.xml
```

Saved the output produced by the last command above (eki-XXXXXXXX), which will be needed while registering the disk image.

2.6.2: Registering Ramdisk Image

Executed the following commands to bundle and register the ramdisk image (initrd.img-2.6.35-22-server)

```
shaon@client:~$ euca-bundle-image -i initrd.img-2.6.32-
28-generic --ramdisk true
shaon@client:~$ euca-upload-bundle -b mybucket -m
/tmp/initrd.img-2.6.32-28-generic.manifest.xml
shaon@client:~$ euca-register
mybucket/initrd.img-2.6.35-22-server.manifest.xml
```

2.6.3: Registering Disk Image

Executed the following commands to bundle and register the ramdisk image (imagefinal.img)

```
shaon@client:~$ euca-bundle-image -i imagefinal.img --
kernel eki-7A8D1340 --ramdisk eri- B603142C
shaon@client:~$ euca-upload-bundle -b mybucket -m
/tmp/imagefinal.img.manifest.xml
shaon@client:~$ euca-register
mybucket/imagefinal.img.manifest.xml
```

To see the the uploaded images

```
shaon@client:~$ euca-describe-images
IMAGE    eki-7A8D1340    mybucket/vmlinuz-2.6.32-28-
generic.manifest.xml    admin    available    public
x86_64kernel              instance-store
IMAGE    eri-B603142C    mybucket/initrd.img-2.6.32-28-
generic.manifest.xml    admin    available    public
x86_64    ramdisk              instance-store
IMAGE    emi-CF6C10B0
mybucket/imagefinal.img.manifest.xml    admin
available    public        x86_64    machine    eki-
7A8D1340    eri-B603142C    instance-store
```

2.6.4: Running Custom Image

I added a new key-pair to run a new instance with the key.

```
shaon@client:~$ cd ~/.euca/
shaon@client:~/.euca$ euca-add-keypair jssecacerts >
jssecacerts.priv
shaon@client:~/.euca$ chmod 600 jssecacerts.priv
shaon@client:~/.euca$ euca-describe-keypairs
```

Then added a group and started instance with the following command

```
# Check group detail
shaon@client:~/.euca$ euca-describe-groups

# Add new group
shaon@client:~/.euca$ euca-add-group -d "Web Servers"
webservers

# Add tcp and icmp permission to the webservers group
shaon@client:~/.euca$ euca-authorize -P tcp -s 0.0.0.0/0
webservers
shaon@client:~/.euca$ euca-authorize -P icmp -s 0.0.0.0/0
webservers
shaon@client:~/.euca$ euca-authorize -P tcp -s 0.0.0.0/0
default
shaon@client:~/.euca$ euca-authorize -P icmp -s 0.0.0.0/0
default
shaon@client:~/.euca$ euca-authorize -P tcp -p 80 default
```

2.7: Eucalyptus Manual Installation on Ubuntu

Eucalyptus ver. 2.0.2, Ubuntu 11.10, two physical machines (one with two NICs)

First installed the Cluster Controller (CC), Storage Controller (SC), Cloud Controller and Walrus also going on the same box.

```
$ sudo apt-get install eucalyptus-cloud eucalyptus-cc
eucalyptus-walrus eucalyptus-sc
```

Then I installed and configured ntp (Network Time Protocol) for the time sync between two machines.

```
$ sudo apt-get install ntp
```

Modified the **ntp.conf** for this setup, but this may not be a good idea for large-scale installation.

Added the following lines to **ntp.conf**

```
server 127.127.1.0
fudge 127.127.1.0 stratum 10
```

and restarted the ntp service.

Registered cluster, storage controller and walrus.

```
$ sudo euca_conf --register-cluster cluster1 192.168.1.2
$ sudo euca_conf --register-walrus 192.168.1.2
$ sudo euca_conf --register-sc cluster1 192.168.1.2
```

For Node controller I installed few more packages. To be in the safe side, I installed all the recommended and suggested packages.

```
$ sudo apt-get install bridge-utils libcrypt-openssl-
random-perl libcrypt-openssl-rsa-perl libcrypt-openssl-
x509-perl open-iscsi powernap qemu-kvm vlan aoetools
eucalyptus-nc
```

node had to be configured with a bridge as the primary interface

```
auto eth0
iface eth0 inet manual

auto br0
iface br0 inet static
address 10.10.10.3

bridge_ports eth0
bridge_fd 9
bridge_hello 2
bridge_maxage 12
bridge_stp off
```

install and configure ntp by adding the following line

```
server 192.168.1.2
```

Modify the **qemu.conf** file to make sure *libvirt* is configured to run as user "eucalyptus"

```
$ sudo vim /etc/libvirt/qemu.conf
```

search and set: user = "eucalyptus"

modify the libvirt.conf file

```
unix_sock_group = "libvirtd"
unix_sock_ro_perms = "0777"
unix_sock_rw_perms = "0770"
auth_unix_ro = "none"
auth_unix_rw = "none"
```

After the modification I stopped and started **libvirt** for the changes to take place and also executed the following commands to make sure the sockets belong to the correct group,

```
$ sudo /etc/init.d/libvirt-bin stop
$ sudo /etc/init.d/libvirt-bin start
$ chown root:libvirtd /var/run/libvirt/libvirt-sock
$ chown root:libvirtd /var/run/libvirt/libvirt-sock-ro
```

Edited **eucalyptus.conf** and set private and public interface as br0.

Now as NC setup was done, I registered this node from the CC,

```
$ sudo euca_conf --register-nodes 192.168.1.3
```

The manual cloud setup is completed.

2.8: Eucalyptus Installation on CentOS

I did a standard CentOS installation. And before doing anything else, I disabled the SELinux.

On the front-node I installed ntp and configured it. Installed the required packages.

```
yum install -y java-1.6.0-openjdk ant ant-nodeps dhcp
bridge-utils perl-Convert-ASN1.noarch scsi-target-utils
httpd
```

Installed Xen on the Node controller.

```
$ yum install -y xen
$ sed --in-place 's/#(xend-http-server no)/(xend-http-
server yes)/' /etc/xen/xend-config.sxp
$ sed --in-place 's/#(xend-address localhost)/(xend-
address localhost)/' /etc/xen/xend-config.sxp
$ /etc/init.d/xend restart
```

Created and added the lines below to **/etc/yum.repos.d/euca.repo** to download the packages from Eucalyptus repository.

```
[euca]
name=Eucalyptus
baseurl=http://www.eucalyptussoftware.com/downloads/repo/
eucalyptus/2.0.3/yum/centos/x86_64/
gpgcheck=0
```

Downloaded and installed the necessary packages on the front-end or CC

```
yum install eucalyptus-cloud eucalyptus-cc eucalyptus-
walrus eucalyptus-sc
```

Then I downloaded and installed packages on node controller,

```
yum install eucalyptus-nc
```

2.8.1: Post Installation Setup

I have set the network setting of the front-end like below,

```
VNET_MODE="MANAGED-NOVLAN"
VNET_SUBNET="192.168.0.0"
VNET_NETMASK="255.255.0.0"
VNET_DNS="8.8.4.4"
VNET_ADDRSPERNET="32"
VNET_PUBLICIPS="xxx.xxx.xxx.xxx"
```

I edited the **/etc/libvirt/libvirtd.conf** file so the following lines are look like below,

```
unix_sock_group = "libvirt"
unix_sock_ro_perms = "0777"
unix_sock_rw_perms = "0770"
```

Then started the Eucalyptus services on frond-end,

```
/etc/init.d/eucalyptus-cloud start
/etc/init.d/eucalyptus-cc start
```

And on node controller,

```
/etc/init.d/eucalyptus-nc start
```

Registering Eucalyptus components

```
$ sudo euca_conf --register-cluster <clustername> <front
end IP address>
$ sudo euca_conf --register-walrus <walrus IP address>
$ sudo euca_conf --register-sc cluster1 <storage
controller IP address>

$ sudo euca_conf --register-nodes <node IP address>
```

Chapter 3: Automated Installation

Eucalyptus has few other open source projects along with Eucalyptus cloud platform. One of their projects is Faststart. They wrote a bash script for the new comer. This script is to automate the installation process with just few basic commands. But there scripts work only on CentOS.

As I was working on Ubuntu platform in the beginning, I wrote another script for Ubuntu. It works almost like their official one.

3.1: Requirements

- Two physical machines
- Ubuntu 11.10

3.2 How to use

- Using the script is very easy. Download the script from here http://mdshaonimran.github.com/eucastart/
- Make the script executable **chmod +x eucastart.sh**
- Run the script on the front-end **./eucastart.sh**

The script will prompt for confirming the server information and few configurations detail.
And then again follow the same steps on node controller. It will complete the full installation with just this few steps.

Chapter 4: Cloud Management Tool

4.1: Introduction

Managing cloud has never been so easy. To handle a cloud platform like Eucalyptus user has to know the commands and how to use them. Eucalyptus doesn't come with a user interface that meets all the users need. So the idea is to build a very user-friendly interface that communicates with cloud platform.

4.2: Motivation

For thesis, I initially tried Amazon Web Services. There I saw their web interface, which was very slick. Almost everyone with little knowledge can use it. It lets user to use all their services. I found that it was one of the most important things that was missing in the open source cloud platform Eucalyptus.

4.3: Eucalyptus API

I started looking into the APIs and libraries that were available.
Eucalyptus APIs are based on SOAP. It maintains very strict standards. Any language that supports the SOAP communication can be used with Eucalyptus API. Secure SOAP messages use the BinarySecurityToken profile, consisting of an X.509 certificate with an RSA public key.
Here is a simple SOAP request to run an instance

```
<RunInstances xmlns="http://ec2.amazonaws.com/doc/2012-03-01/">
    <instancesSet>
        <item>
            <imageId>ami-60a54009</imageId>
            <minCount>1</minCount>
            <maxCount>3</maxCount>
        </item>
    </instancesSet>
    <groupSet/>
</RunInstances>
```

Figure: 4.1

Eucalyptus also supports REST/Query-based API

```
https://ec2.amazonaws.com/
?Action=RunInstances
&ImageId=ami-60a54009
&MaxCount=3
&MinCount=1
&Placement.AvailabilityZone=us-east-1b
&Monitoring.Enabled=true
&AWSAccessKeyId=0GS7553JW74RRM612K02EXAMPLE
&Version=2012-03-01
&Expires=2010-10-10T12:00:00Z
&Signature=1BP67vCvGlDMBQ1dofZxg8E8SUEXAMPLE
&SignatureVersion=2
&SignatureMethod=HmacSHA256
```

Figure: 4.2

4.4: Decision

I found out, that all I have to do is to send request and use the responses accordingly. Then I looked into Boto python library. Which does all these in a nifty way and also supports REST/Query-based API interface.

For web framework, I looked at web2py and Django. They both serve the purpose. I chose Django as I have minimum experience with it before.
To build the interface I used Twitter's new user interface framework, Bootstrap. It had almost all the components that would need to build an admin panel.
For asynchronous calls, I use jquery ajax.

4.5: Building Web Interface

I setup a development environment with python **virtualenvironment**. Installed Django 1.3 and Boto 2.2.2.

Figure: 4.3

I divided the application into multiple segments. So that maintaining the code base remains easier later on.

Figure: 4.4

With this web interface a user can easily create keypair, delete keypair, create a new security group and edit group preferences.

Figure: 4.5

And then user will be able to launch an instance with preferred group and keypair. User can also choose instance type and the addressing mode.

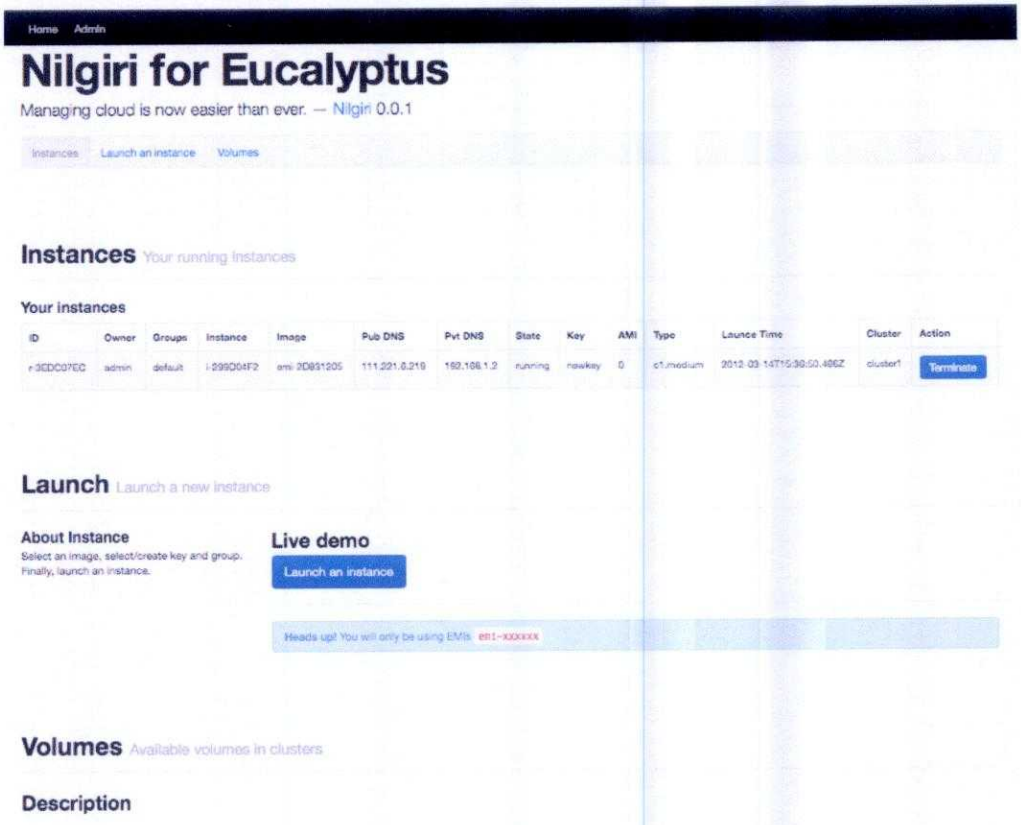


Figure: 4.6

It also has volume management function. New volume can be created, volume attach to an instance and detach volume from an running instance, volume delete all functions are included in volume functionality.

Figure: 4.7

4.6: Architecture of the web console

The project is built on a MVC framework. But in this case I don't need any database, so I have not used model until now.

When a function is called from the web interface, it goes to the views.py of that specific function. Then that particular function calls the commands for that particular task. These commands are intermediate interface with the Boto library. That command calls the common connect function and makes a connection activated with the Eucalyptus interface to do final API call. Then it calls the Boto API to retrieve or send data to Eucalyptus API. The whole system works in the same way.

Code of the main connection mechanism

```
import nilgiri
import getopt
import sys
import os
import textwrap
import urlparse
import boto
from boto.ec2.regioninfo import RegionInfo
from boto.s3.connection import OrdinaryCallingFormat
from boto.ec2.blockdevicemapping import
BlockDeviceMapping, BlockDeviceType
from boto.roboto.param import Param

class NilgiriCommand(object):

    def __init__(self, is_euca=False, debug=False):
        self.region = RegionInfo()
        self.filters = {}
        self.ec2_user_access_key = '<access_key>'
        self.ec2_user_secret_key = '<secret_key>'


    def make_ec2_connection(self):
        self.is_secure = True
        self.debug = 1
        self.region.name = 'eucalyptus'
        self.region.endpoint = 'xxx.xxx.xxx.xxx'
        self.service_path = '/services/Eucalyptus'
        self.port = 8773
```

```
        return
boto.connect_ec2(aws_access_key_id=self.ec2_user_access_k
ey,

aws_secret_access_key=self.ec2_user_secret_key,
                                    is_secure=self.is_secure,
                                    region=self.region,
                                    port=self.port,
                                    path=self.service_path,
                                    debug=self.debug)

    def make_connection(self, conn_type='ec2'):
        if conn_type == 's3':
            conn = self.make_s3_connection()
        elif conn_type == 'ec2':
            conn = self.make_ec2_connection()
        else:
            conn = None
        return conn

    def make_connection_cli(self, conn_type='ec2'):
        conn = self.make_connection(conn_type)
        return conn

    def make_request_cli(self, connection, request_name,
**params):
        method = getattr(connection, request_name)
        return method(**params)



    # done
    def nversion(self):
        return nilgiri.__version__
```

Get the required data from web interface and send it to specific file to do the task,

```
def run_instances(request):
    # run instance
    query_key = request.POST.get('selected_key', '')
    query_image = request.POST.get('selected_image', '')
    query_groups = request.POST.get('selected_group', '')
    query_instance_type =
request.POST.get('instance_type', '')
    query_addressing_type =
request.POST.get('addressing_type', '')
```

```
    groups = []
    groups.append(query_groups)
    nilCmd =
dashboard.nilgiri.commands.euca.runinstances.RunInstances
()
    reservation = nilCmd.main_cli(query_image, query_key,
groups, query_instance_type, query_addressing_type)
    context = { 'reservation': reservation }
    template = "instances/new_instance.html"
    return shortcuts.render_to_response(template,
context, context_instance=RequestContext(request))
```

Codes responsible to run an instance. These codes are to start a new instance with the selected image, keys and group.

```
import nilgiri.commands.nilgiricommand
from boto.roboto.param import Param

class
RunInstances(nilgiri.commands.nilgiricommand.NilgiriComma
nd):

    def main(self, emi_image_id, key, groups, vm_type,
add_type=None):
        conn = self.make_connection_cli()
        return conn.run_instances(emi_image_id,
                                  key_name=key,

instance_type=vm_type,

security_groups=groups,

addressing_type=add_type)

    def main_cli(self, emi_image_id, key, groups,
vm_type, add_type):
        if(add_type == "None"):
            instances = self.main(emi_image_id, key,
groups, vm_type)
        else:
            instances = self.main(emi_image_id, key,
groups, vm_type, add_type)
        return instances
```

4.7: Use Case Diagram

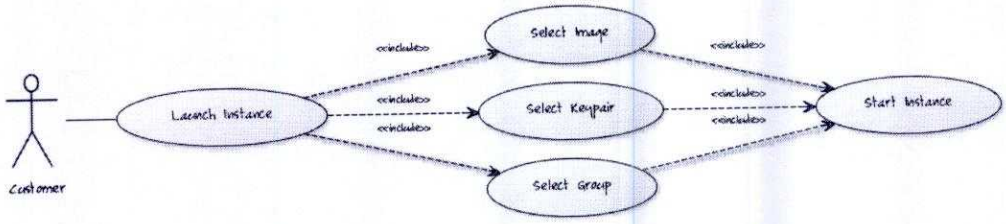


Figure: 4.8

In this project and user can deal with all the features that he needs to run an instance from a single start point. When the user selects the Launch instance button, he is prompted to select an image. As he selects the image then he is prompted to select a keypair and then select a group. So finally when the user takes all the necessary actions then he is shown the start instance view. There the user can select his instance type and addressing mode whether the address will be public address or private address. These options are not mandatory as there default options are there for instances.
Finally, user can launch a new instance from the view.

On the key selection step, the user also can create new pair of keys and delete an existing one. The group management view contains editing group security methods. User can add and revoke new rules to the group and then select that group to use with the instance.

Chapter 5: Challenges

Deploying a cloud platform was completely new to me. First I tried to deploy Openstack. In the very beginning I installed Xen Cloud Platfor (XCP) without knowing what it does. Then installed Openstack, the documentation I was following had a flaw that time. So, I had to debug that, and after that I had Openstack (Cactus) cloud running.

On Ubuntu 11.10 I was having problem with attaching volume. Couldn't solve this issue. The issue can be found here http://osdir.com/ml/ubuntu-bugs/2011-11/msg33174.html

Moving to RPM based distro like CentOS was not too easy in the very begging as I always have been used to with Debian based distros.

I have tried to add the noVNC into this web management tool. But I figured out some issues with it. The current version of noVNC is not working with my setup. So I need to figure that out for further work. Otherwise I have to choose another open source VNC solution for it. I used one deprecated method in Django while I was working with version 1.3, then when I run the application with Django version 1.4, it failed. So, I had to figure that out and solved it.

Chapter 6: Future Plans

Large-scale Implementation

My future plan is to implement the open source cloud platform on larger scale. But for that we first ensure that resources are available for large-scale deployment.

Continuous Feature Integration

I want to implement all the functionalities of Eucalyptus in the management tool.

Add Authentication System

I have also plan to add authentication system, so that it can be used as a hosted solution for the public users.

Add VNC

Another important feature can be added is VNC console. I have also plan to software deployment features if possible. I also want to add user authentication for the management tool, so that it can be used as a hosted solution also.

Chapter 7: Conclusion

If we look back to the objective of the thesis work we can differentiate the points, to setup a enterprise business solution using open source tools, develop a user friendly user interface to manage the cloud ecosystem. Throughout the work phase I tried different solutions to deploy a stable cloud platform. After few tests I got it perfectly working on CentOS 5.7.
Another part of the thesis was to build a user-friendly cloud management system. The complete user management tool is very easy to install and can be used on any platform as it runs on browser. I also tested browser compatibility issues; it runs perfectly on Firefox, Safari and Chrome.

Deploying cloud platform was not too easy, I had to deal with some uncommon problems and had to solve them one by one. Once I mistakenly started front-end with STATIC networking mode and it created an additional virtual ethernet and I had to spend a lot of time figure that out. Understanding the network mode was bit tricky at the very beginning, but later on it has become easy. I spent a lot of time on attaching volume and get it working on running instances both in Ubuntu and in CentOS. But finally it worked on CentOS. Some experts think that, Ubuntu 11.10 was not Eucalyptus ready. Downloading credentials from the front-end was not also working for Ubuntu 11.10 version.
Developing the tool was interesting and I felt very comfortable though I was not too much familiar with Python and Django. I faced another problem during the development. Between that very short period Django released a newer version 1.4 where I was using 1.3 and Boto release another new version 2.3.0. I didn't have any problem with Boto, but in Django I previously used a deprecated function, which didn't work on the newer version. So, I had to fix that as well. I found another problem with Django, the ajax POST method is not working on any browser but Firefox. But then I figured out that it is a browser cookie issue and it only creates problem on development environment.

However, knowing the vast world of cloud computing was very effective. There is no doubt that this new technology is the future. This will be one of the most effective to deal with the fast growing data, better resource utilization, faster server provisioning etc.

For further work, I would like to work on making some Amazon features available on open source technology. Amazon has some fantastic applications built for the ease of computing which are not available in any open source solutions out there. But that will need more time and most importantly resources.

References

- http://en.wikipedia.org/wiki/Cloud_computing
- http://aws.amazon.com
- http://www.eucalyptus.com
- http://www.openstack.org/
- https://github.com/boto/boto