CLOUD USAGE BILLING SYSTEM

A PROJECT REPORT

Submitted by

RAJEN PATEL (090110107007)

KALPAN SHAH (090110107016)

In fulfillment for the award of the degree

Of

BACHELOR OF ENGINEERING

In

COMPUTER

G.H PATEL COLLEGE OF ENGINEERING & TECHNOLGY,
V.V.NAGAR

Gujarat Technological University, Ahmadabad

May, 2013

ACKNOWLEDGEMENT

We would like to acknowledge the contribution of certain distinguished people, without their support and guidance this Project work would not have been completed.

We take this opportunity to express our sincere thanks and deep sense of gratitude to our Project Guide **Prof. Hetal Gaudani** for inspiring us by spending her valuable time and efforts and also being supportive at every stage of the project work. We really thank her from the rock bottom of our heart for always being there with her extreme knowledge and kind nature.

We are grateful to **Mr. Nilesh Vaghela** and **Mr. Shyam Arsiwala** Electromech Corporation, Ahmedabad for providing his valuable time and his support to us. We are very thankful to them from bottom of our heart for providing us their support and believe in us during journey of Project Work.

.

We take this opportunity to thank **Prof. Maulika Patel**, Head of Computer Engineering Department, GCET and all Teaching and Non-Teaching staff for their all-time support and help in each and every.

RAJEN PATEL
(090110107007)

KALPAN SHAH
(090110107016)

About Electromech Corporation

OPENSOURCE
SOLUTIONS
SUPPORT
TRAINING
DEVELOPMENT

Electromech Corporation is one of the leading open source company providing open source Solution, Support, Training and Development since 1996, to all size organization starting from small to large corporates, with most experience and certified staff.

Main motto of Electromech Corporation is “to provide solutions based on open source products” to large Corporates, small enterprise and individual users, and make organization feel “Be free and stable".

Electromech Corporation is Redhat Certified Channel and Training partner. As a open source company we provide excellent community VGLUG promoted by our company.

Electromech Corporation is also provide open source free project training to College student.

Abstract

Cloud computing is the new paradigm that has changed traditional computer business schemes. Emerging Cloud computing infrastructures provide computing resources on demand. This project develops an infrastructure capable of delivering elastic capacity that can automatically be increased or decreased according to the customer need. It includes ram, storage, processing power and networking capabilities as requirement. Still, this new scenario poses a number of opportunities to use and novel problems to be faced. Specifically, we focus on the accounting of cloud computing services. For accounting and billing, such infrastructures call for novel approaches to perform accounting for capacity that varies over time and for services (or more precisely virtual machines) that migrate between physical machines or even between data centers. These may include relations between different service providers, user connections to different simultaneous services, and the need for new services to be incorporated into the accounting systems to enable emerging business models, and so on. Classic solutions fail to provide a proper measurement of actual usage of resources as they were specifically design accounting model for hour basis or month basis. Against this background, we put forward a efficient accounting model that allows the deployment of cloud computing services to accomplish real measurement of the usage by evaluating actual ram consumption, processing power and session period .Even though the primary focus for this architecture is accounting and billing between resource consumers and infrastructure provides, future support for inter-site billing is also taken into account.

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

INTRODUCTION CHAPTER-1

Cloud computing has lately become a fashionable term, one of those buzz-words that must appear in every cutting edge tech-talk, covering it with a gaudy, modern, and cool wrapping. Nevertheless, the reality hidden by this recent disguise has somehow always been there. cloud computing is about moving applications that traditionally were locally run to be remotely executed on the Internet .The term "cloud", as used in this project, appears to have its origins in network diagrams that represented the internet, or various parts of it, as schematic clouds. "Cloud computing" was coined for what happens when applications and services are moved into the internet "cloud." Cloud computing is not something that suddenly appeared overnight. In some form it may trace back to a time when computer systems remotely time-shared computing resources and applications. More currently though, cloud computing refers to the many different types of services and applications being delivered in the internet cloud, and the fact that, in many cases, the devices used to access these services and applications do not require any special applications[1].

Many companies are delivering services from the cloud. Some notable examples as of 2011-12 include the following:

• Google — has a private cloud that it uses for delivering many different services to its users including email access, document Applications, text translations, maps, web analytics, and much more.

• Microsoft — Has Microsoft SharePoint online service that allows for content and business intelligence tools to be moved into the cloud, and Microsoft currently makes its office applications available in a cloud.

• Salesforce.com — Runs its application set for its customers in a cloud, and its Force.com and Vmforce.com products provide developers with platforms to build customized cloud services.

But, what is cloud computing? The following sections note cloud and cloud computing characteristics, services models, deployment models, benefits, and challenges.

1.1.1 Characteristics of Cloud Computing

Cloud computing has a variety of characteristics, with the main ones being:
• Shared Infrastructure — Uses a virtualized software model, enabling the sharing of physical services, storage, and networking capabilities. The cloud infrastructure, regardless of deployment model, seeks to make the most of the available infrastructure across a number of users.
• Dynamic Provisioning — Allows for the provision of services based on current demand requirements. This is done automatically using software automation, enabling the expansion and contraction of service capability, as needed. This dynamic scaling needs to be done while maintaining high levels of reliability and security.
• Network Access — Needs to be accessed across the internet from a broad range of devices such as PCs, laptops, and mobile Devices, using standards-based APIs (for example, ones based on HTTP). Deployments of services in the cloud include everything from using business applications to the latest application on the newest smart phones.
• Managed Metering — Uses metering for managing and optimizing the service and to provide reporting and billing information.

In this way, consumers are billed for services according to how much they have actually used during the billing period. In short, cloud computing allows for the sharing and scalable deployment of services, as needed, from almost any location, and for which the customer can be billed based on actual usage.

1.1.2 Service Models of Cloud

Once a cloud is established, how its cloud computing services are deployed in terms of business models can differ depending on requirements. The primary service models being deployed are commonly known as:

- SaaS (Software As A Service):

Is the most widely known and widely used form of cloud computing. It provides all the functions of a sophisticated traditional application to many customers and often thousands of users, but through a Web browser, not a “locally-installed” application. Little or no code is running on the Users local computer and the applications are usually tailored to fulfill specific functions. SaaS eliminates customer worries about application servers, storage, application development and related, common concerns of IT. Highest-profile examples are Salesforce.com, Google's Gmail and Apps, instant messaging from AOL, Yahoo and Google, and VoIP from Vonage and Skype.

- PaaS (Platform as a Service):

It delivers virtualized servers on which customers can run existing applications or develop new ones without having to worry about maintaining the operating systems, server hardware, load balancing or computing capacity. These vendors provide APIs or development platforms to create and run applications in the cloud – e.g. using the Internet. Managed Service providers with application services provided to IT departments to monitor systems and downstream applications such as virus scanning for e-mail are frequently included in this category. Well known providers would include Microsoft's Azure, Sales force's Force.com, Google Maps, ADP Payroll processing, and US Postal Service offerings.

- IaaS (Infrastructure as a Service):

It delivers utility computing capability, typically as raw virtual servers, on demand that customers configure and manage. Here Cloud Computing provides grids or clusters or virtualized servers, networks, storage and systems software, usually (but not always) in a multitenant architecture. IaaS is designed to augment or replace the functions of an entire data center. This saves cost (time and expense) of capital equipment deployment but does not reduce cost of configuration, integration or management and these tasks must be performed remotely. Vendors would include Amazon.com (Elastic Compute Cloud [EC2] and Simple Storage), IBM and other traditional IT vendors.

1.1.3 Benefits

The following are some of the possible benefits for those who offer cloud computing-based services and applications:

• Cost Savings — Companies can reduce their capital expenditures and use operational expenditures for increasing their computing capabilities. This is a lower barrier to entry and also requires fewer in-house IT resources to provide system support.

• Scalability/Flexibility — Companies can start with a small deployment and grow to a large deployment fairly rapidly, and then scale back if necessary. Also, the flexibility of cloud computing allows companies to use extra resources at peak times, enabling them to satisfy consumer demands.

• Reliability — Services using multiple redundant sites can support business continuity and disaster recovery.

• Maintenance — Cloud service providers do the system maintenance, and access is through APIs that do not require application installations onto PCs, thus further reducing maintenance requirements.

• Mobile Accessible — Mobile workers have increased productivity due to systems accessible in an infrastructure available from anywhere.

1.1.4 Challenges

The following are some of the notable challenges associated with cloud computing, and although some of these may cause a slowdown when delivering more services in the cloud, most also can provide opportunities, if resolved with due care and attention in the planning stages.

• Security and Privacy — perhaps two of the more "hot button" issues surrounding cloud computing relate to storing and securing data, and monitoring the use of the cloud by the service providers. These issues are generally attributed to slowing the deployment of cloud services. These challenges can be addressed, for example, by storing the information internal to the organization, but allowing it to be used in the cloud. For this to occur, though, the security mechanisms between organization and the cloud need to be robust and a Hybrid cloud could support such a deployment.

• Lack of Standards — clouds have documented interfaces; however, no standards are associated with these, and thus it is unlikely that most clouds will be interoperable. The Open Grid Forum is developing an Open Cloud Computing Interface to resolve this issue and the Open Cloud Consortium is working on cloud computing standards and practices. The findings of these groups will need to mature, but it is not known whether they will address the needs of the people deploying the services and the specific interfaces these services need. However, keeping up to date on the latest standards as they evolve will allow them to be leveraged, if applicable.

• Continuously Evolving — User requirements are continuously evolving, as are the requirements for interfaces, networking, and storage. This means that a "cloud," especially a public one, does not remain static and is also continuously evolving.

• Compliance Concerns — The Sarbanes-Oxley Act (SOX) in the US and Data Protection directives in the EU are just two among many compliance issues affecting cloud computing, based on the type of data and application for which the cloud is being used. The EU has a legislative backing for data protection across all member states, but in the US data protection is different and can vary from state to state. As with security and privacy mentioned previously, these typically result in Hybrid cloud deployment with one cloud storing the data internal to the organization.

1.2 Motivation

As we were doing our research for project, we were amused by the variations in opinions about Cloud Computing. They seemed to run from one end (Cloud Computing is the future and one day all computing will be done in the Cloud) to the other end (Cloud Computing is just standard technology organized and offered in such a way as to make or save somebody some money). As we dug deeper we found that, to some degree, both views may be valid. How it will all balance out is a bit beyond our ability to predict. We do believe, though, that we should not ignore it. Cloud Computing is not going to “go away”.

Basically, we understand Cloud Computing and according to us it is a service provider somewhere out there on the Internet can supply you with the computer power, storage devices, software and/or the means to build and run your own software on their computer center. It is like buying raw computer power, data storage space, applications or the means to build applications off the Internet in much the same way as you buy natural gas from a gas company. They have a network of gas pipes to get their service/product to you and, for Cloud Computing; the supply device is the Internet, the Web. The Cloud Computing concept is very broad but there is nothing really new in it. Cloud Computing is existing technology and methods put together to supply valuable services to whoever may need them. It might be seen as no more than an evolutionary technology step from the do-it-yourself computer center to a let-somebody-else-do-it computer center. With Cloud Computing, you let someone else get it all together and you buy (or rent) whatever services you need from them. We will refer to them as Cloud service providers.

There are other things that make Cloud Computing attractive as well. For example, if you decide you need a new business application, you may be able to simply buy it from a Cloud service provider or build it yourself. In any case there is no time lost in ordering new equipment or getting things prepared. Most of what you need will be waiting out there in “the Cloud” when you need it. A similar situation comes up if you are running a Cloud Computing solution and your Business grows quickly. You may need more computing power and need it right now. The Cloud service provider can probably give it to you overnight. As a Cloud Computing customer, you can scale up your computer power at the drop of a hat. So, Cloud Computing gives you attractive costs, agility and scalability that may just not be available in your own data center. As that same time we think that though it is used for business ,why it haven’t any proper and efficient resource usage accounting and billing model which precisely work on not only hour basis but also consider ram, processing power ,storage (memory).And we started to work on it to make it commercialize.

1.3 Literature review

Cloud computing is a relatively new concept and its current services are still emerging as of today. Moreover, open source frameworks like Eucalyptus only have the basic user management capabilities on its features while some enterprise versions have a billing capacity.

In 2005, Llorente and Montero developed OpenNebula, an open-source toolkit for building private, public and hybrid clouds. One of its features is user management, wherein it provides functionality for authentication framework, multiple cloud user and administration roles, quota management and secure multitenancy. [3]

In 2009, Keahey et al developed Nimbus, an open-source toolkit for transforming clusters into an IaaS. Nimbus has a feature called Per-client Usage Tracking wherein it makes possible to track the deployment time on a per-client basis. It also has another feature called Per-user Storage Quota wherein enforcement of per-user storage limits is possible through the VM image repository manager of Nimbus, Cumulus. [4]

In 2009, the Abiquo team developed AbiCloud, an open source infrastructure software which can be used to create and manage private and public clouds. AbiCloud also has a user management feature which makes it possible to manage organizations, users, sessions and to implement basic hard and soft limits. [5]

In 2009, the VMware team developed vSphere, an enterprise software which is a platform for virtualization. One of its services is the vNetwork, which enables the administration and management of networking in virtual environments. [6]
In 2007, Wolski et al developed Eucalyptus, an open-source software infrastructure used to implement a private cloud from an information technology (IT) infrastructure. [7]
We checked Dough billing system which having flexible customization of payment policies but does not keep track of tenet's resource usage and does not money from account.[10]

Of the above mentioned tools, it is evident that the open source tools support the basic user management needs of the client, but fail to provide an accounting feature, which is highly necessary for cloud commercialization. Therefore, this study aims to provide an accounting system for adding efficient accounting feature, which may serve as a pioneer study in private cloud billing.

1.4 Organizational Thesis

First, we had presented basic concept cloud computing services and our IaaS model. Second, as a proof of concept, we describe a possible implementation of the previous model running a specific service of cloud computing. The remainder of this paper is structured as follows: chapter 2.Describes Theoretical background for cloud computing IaaS platform.Chapter 3.illustrating how it could be implemented. Finally, chapter 4.concludes and draws the avenues of future work.

THEORETICAL BACKGROUND CHAPTER-2

2.1 Components of Our Project

There are currently three main components: Compute, Object Storage, and Image Service. Let's look at each in turn.

Compute is a cloud fabric controller, used to start up virtual instances for either a user or a group. It's also used to configure networking for each instance or project that contains multiple instances for a particular project.

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

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

The following diagram shows the basic relationships between the projects, how they relate to each other, and how they can fulfill the goals of open source cloud computing.

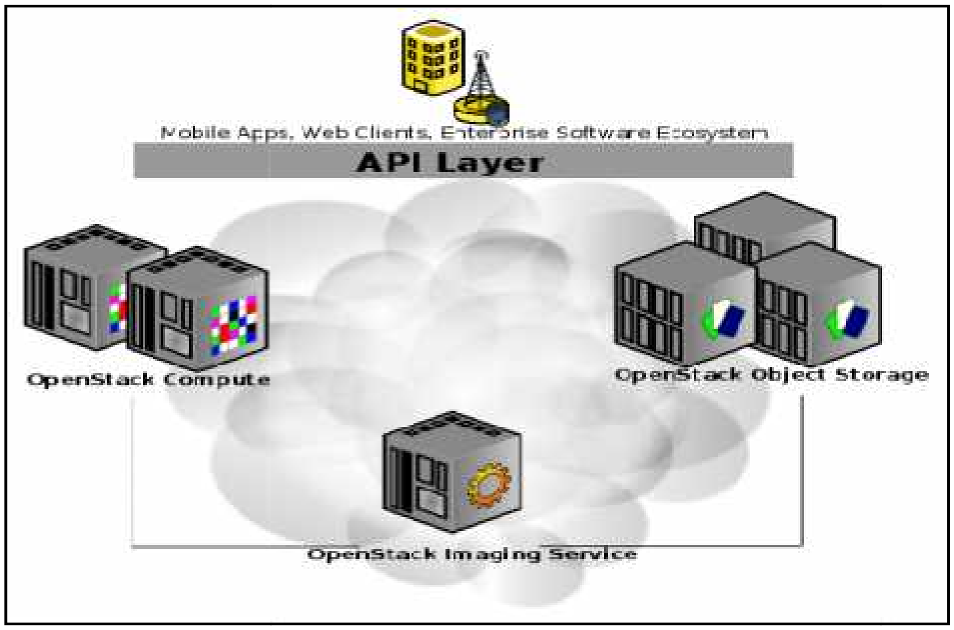

Mobile Apps, Web Clients, Enterprise Software Ecosystem
API Layer
OpenStack Compute
OpenStack Object Storage
OpenStack Imaging Service

Figure: 2.1.1 Relation between compute nova and glance

Because Compute has multiple services and many configurations are possible, here is a diagram showing the overall service architecture and communication systems between the services.

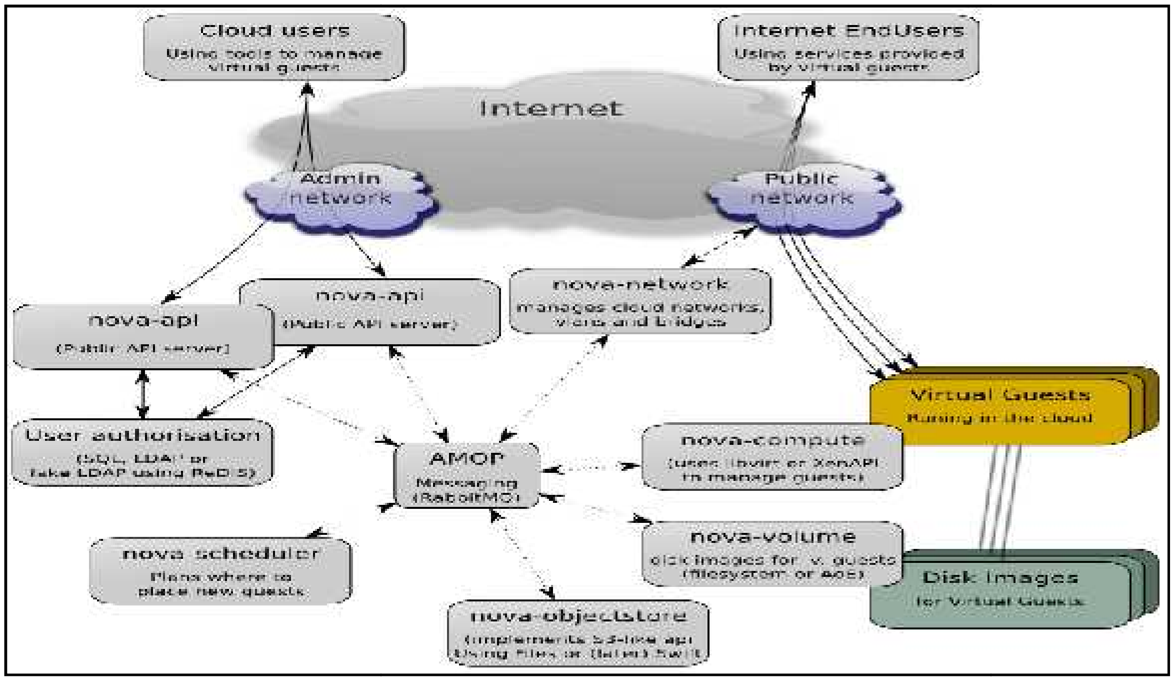

Cloud users
Using tools to manage virtual guests
Internet EndUsers
Using services provided by virtual guests
Internet
Admin network
Public network
nova-api
(Public API server)
nova-api
(Public API server)
nova-network
manages cloud networks, vlans and bridges
Virtual Guests
Running in the cloud
User authorisation
(SQL, LDAP or fake LDAP using ReDIS)
AMQP
Messaging (RabbitMQ)
nova-compute
(uses libvirt or XenAPI to manage guests)
nova-volume
disk images for v. guests (filesystem or AoE)
nova scheduler
Plans where to place new guests
Disk Images
for Virtual Guests
nova-objectstore
(implements S3-like api Using Files or (later) Swift)

Figure: 2.1.2 Relation between all api and cloud user

Three main projects:

- Swift this provides object/blob storage. This is roughly analogous to Rackspace Cloud Files (from which it is derived) or Amazon S3.
- Glance which provides discovery, storage and retrieval of virtual machine images for OpenStack Nova.
- Nova which provides virtual servers upon demand. This is similar to Rackspace Cloud Servers or Amazon EC2.

2.2Cloud Provider Conceptual Architecture

Imagine that we are going to build our own IaaS cloud and offer it to customers. To achieve this, we would need to provide several high level features:

1. Allow application owners to register for our cloud services, view their usage and see their bill (basic customer relations management functionality)
2. Allow Developers/DevOps folks to create and store custom images for their applications (basic build-time functionality)
3. Allow DevOps/Developers to launch, monitor and terminate instances (basic run-time functionality)
4. Allow the Cloud Operator to configure and operate the cloud infrastructure

While there are certainly many, many other features that we would need to offer, these four get to the very heart of providing IaaS. Now assuming that you agree with these four top level features, you might put together a conceptual architecture that looks something like this:

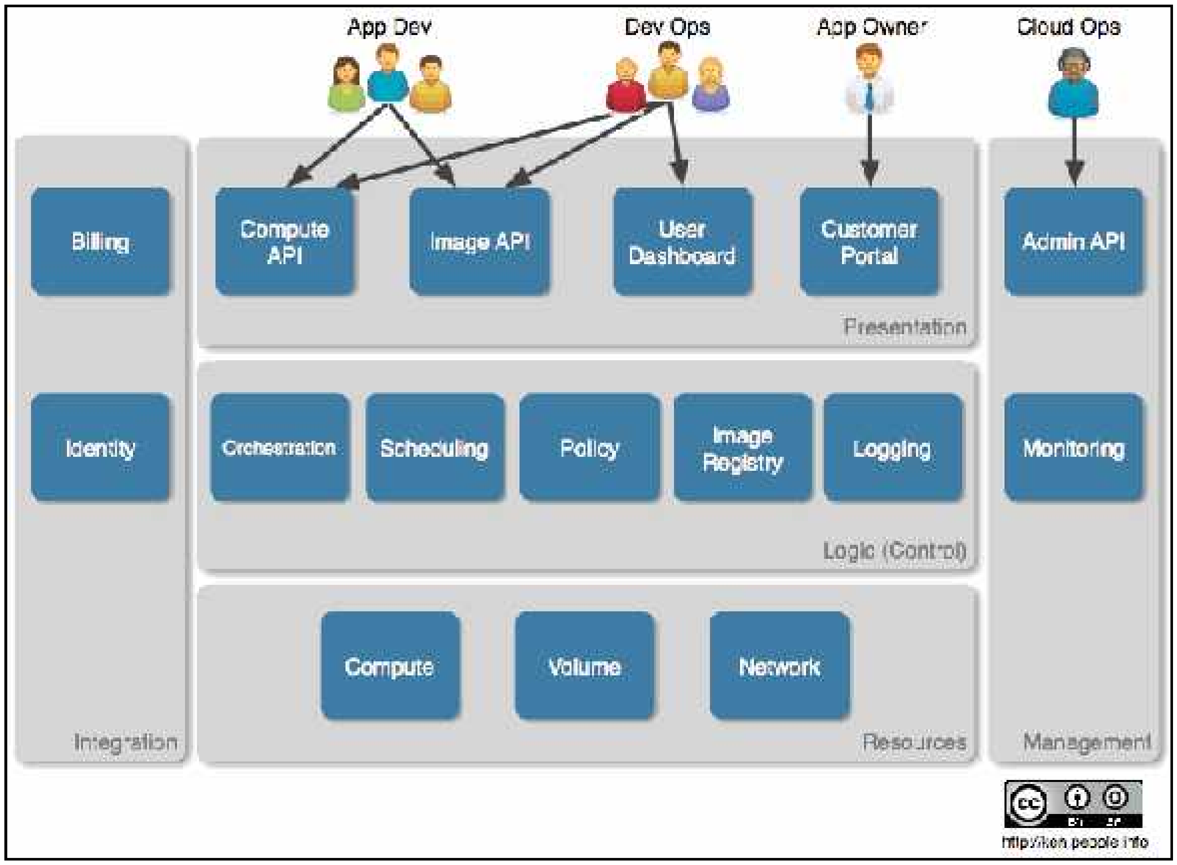


Figure 2.2.1 Cloud conceptual architecture

In this model, we have imagined four sets of users (developers, devops, owners and operators) that need to interact with the cloud and then separated out the functionality needed for each.

From there, we have followed a pretty common tiered approach to the architecture (presentation, logic and resources) with two orthogonal areas (integration and management). Let's explore each a little further:

- As with presentation layers in more typical application architectures, components here interact with users to accept and present information. In this layer, you will find web portals to provide graphical interfaces for non-developers and API endpoints for developers. For more advanced architectures, you might find load balancing, console proxies, security and naming services present here also.
- The logic tier would provide the intelligence and control functionality for our cloud. This tier would house orchestration (workflow for complex tasks), scheduling (determining mapping of jobs to resources), policy (quotas and such) , image registry (metadata about instance images), logging (events and metering).
- There will need to integration functions within the architecture. It is assumed that most service providers will already have a customer identity and billing systems. Any cloud architecture would need to integrate with these systems.
- As with any complex environment, we will need a management tier to operate the environment. This should include an API to access the cloud administration features as well as some forms of monitoring. It is likely that the monitoring functionality will take the form of integration into an existing tool. While I've highlighted monitoring and an admin API for our fictional provider, in a more complete architecture you would see a vast array of operational support functions like provisioning and configuration management.
- Finally, since this is a compute cloud, we will need actual compute, network and storage resources to provide to our customers. This tier provides these services, whether they be servers, network switches, network attached storage or other resources

2.3 OpenStack Compute Logical Architecture

Now that we've looked at a proposed conceptual architecture, let's see how OpenStack Compute is logically architected. There are several logical components of OpenStack Compute architecture but the majority of these components are custom written python daemons of two varieties:

- WSGI applications to receive and mediate API calls (nova-api, glance-api, etc.)
- Worker daemons to carry out orchestration tasks (nova-compute, nova-network, nova-schedule, etc.)

However, there are two essential pieces of the logical architecture are neither custom written nor Python based: the messaging queue and the database. These two components facilitate the asynchronous orchestration of complex tasks through message passing and information sharing. Putting this all together we get a picture like this:

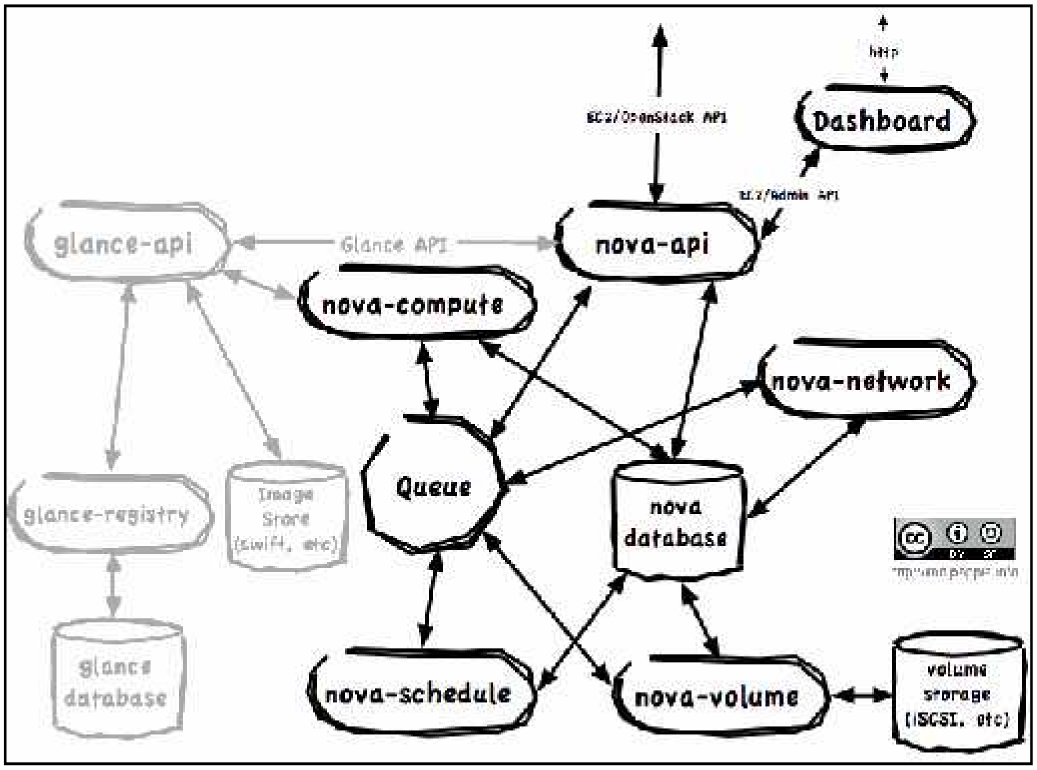


Figure: 2.3.1 Logical architecture of nova api

This complicated, but not overly informative, diagram as it can be summed up in three sentences:

- End users (DevOps, Developers and even other OpenStack components) talk to nova-api to interface with OpenStack Compute
- OpenStack Compute daemons exchange info through the queue (actions) and database (information) to carry out API requests

- OpenStack Glance is basically a completely separate infrastructure which OpenStack Compute interfaces through the Glance API

Now that we see the overview of the processes and their interactions, let’s take a closer look at each component.

- The nova-api daemon is the heart of the OpenStack Compute. You may see it illustrated on many pictures of OpenStack Compute as API and “Cloud Controller”. While this is partly true, cloud controller is really just a class (specifically the Cloud Controller in trunk/nova/api/ec2/cloud.py) within the nova-api daemon. It provides an endpoint for all API queries (either OpenStack API or EC2 API), initiates most of the orchestration activities (such as running an instance) and also enforces some policy (mostly quota checks).
- The nova-schedule process is conceptually the simplest piece of code in OpenStack Compute: take a virtual machine instance request from the queue and determines where it should run (specifically, which compute server host it should run on). In practice however, I am sure this will grow to be the most complex as it needs to factor in current state of the entire cloud infrastructure and apply complicated algorithm to ensure efficient usage. To that end, nova-schedule implements a pluggable architecture that lets you choose (or write) your own algorithm for scheduling. Currently, there are several to choose from (simple, chance, etc.) and it is an area of hot development for the future releases of OpenStack Compute.
- The nova-compute process is primarily a worker daemon that creates and terminates virtual machine instances. The process by which it does so is fairly complex (see this blog post by Laurence Luce for the gritty details) but the basics are simple: accept actions from the queue and then perform a series of system commands (like launching a KVM instance) to carry them out while updating state in the database.
- As you can gather by the name, nova-volume manages the creation, attaching and detaching of persistent volumes to compute instances (similar functionality to Amazon’s Elastic Block Storage). It can use volumes from a variety of providers such as iSCSI or AoE.

- The nova-network worker daemon is very similar to nova-compute and nova-volume. It accepts networking tasks from the queue and then performs tasks to manipulate the network (such as setting up bridging interfaces or changing iptables rules).
- The queue provides a central hub for passing messages between daemons. This is currently implemented with RabbitMQ today, but theoretically could be any AMPQ message queue supported by the python ampqlib.
- The SQL database stores most of the build-time and run-time state for a cloud infrastructure. This includes the instance types that are available for use, instances in use, networks available and projects. Theoretically, OpenStack Compute can support any database supported by SQL-Alchemy but the only databases currently being widely used are sqlite3 (only appropriate for test and development work), MySQL and PostgreSQL.
- OpenStack Glance is a separate project from OpenStack Compute, but as shown above, complimentary. While it is an optional part of the overall compute architecture, I can’t imagine that most OpenStack Compute installations will not be using it (or a complimentary product). There are three pieces to Glance: glance-api, glance-registry and the image store. As you can probably guess, glance-api accepts API calls, much like nova-api, and the actual image blobs are placed in the image store. The glance-registry stores and retrieves metadata about images. The image store can be a number of different object stores, include OpenStack Object Storage (Swift).
- Finally, another optional project that we will need for our fictional service provider is a user dashboard. We had taken OpenStack Dashboard as a reference and designed own Dashboard, but there are also several other web fronts ends available for OpenStack Compute. The OpenStack Dashboard provides a web interface into OpenStack Compute to give application developers and devops staff similar functionality to the API.

This logical architecture represents just one way to architect OpenStack Compute. With its pluggable architecture, we could easily swap out OpenStack Glance with another image service or use another dashboard. In the coming releases of OpenStack, expect to see more modularization of the code especially in the network and volume areas.

2.4 Nova Conceptual Mapping

Now that we've seen a conceptual architecture for a fictional cloud provider and examined the logical architecture of OpenStack Nova, it is fairly easy to map the OpenStack components to the conceptual areas to see what we are lacking:

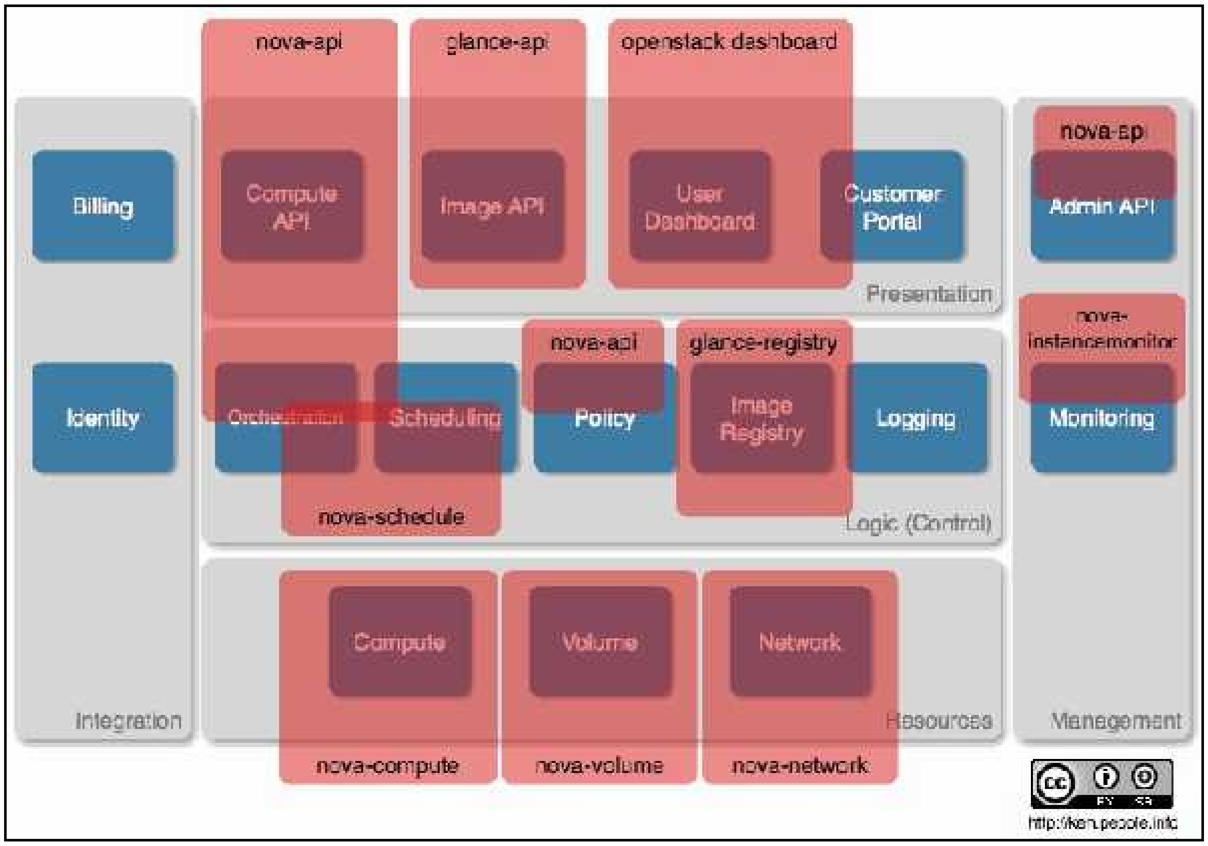


Figure: 2.4.1 Nova conceptual mapping

As you can see from the illustration, I've overlaid logical components of OpenStack Nova, Glance and Dashboard to denote functional coverage. For each of the overlays, I've added the name of the logical component within the project that provides the functionality. While all of these judgments are highly subjective, you can see that we have majority coverage of the functional areas with a few notable exceptions:

- The largest gap in our functional coverage is logging and billing. At the moment, OpenStack Nova doesn't have a billing component that can mediate logging events, rate the logs and create/present bills. That being said, most service providers will already have one (or *many*) of these so the focus is really on the logging and integration with billing. This could be remedied in a variety of ways: augmentations of the code (which should happen in the next release “Diablo”), integration with commercial products or services (perhaps Zuora) or custom log parsing.

- Identity is also a point which will likely need to be augmented. Unless we are running a stock LDAP for our identity system, we will need to integrate our solution with OpenStack Compute. Having said that, this is true of almost all cloud solutions.
- The customer portal will also be an integration point. While OpenStack Compute provides a user dashboard (to see running instance, launch new instances, etc.), it doesn’t provide an interface to allow application owners to sign up for service, track their bills and lodge trouble tickets. Again, this is probably something that it is already in place at our imaginary service provider.
- Ideally, the Admin API would replicate all functionality that we’d be able to do via the command line interface (which in this case is mostly exposed through the nova-manage command). This will get better in the “Diablo” release with the Admin API work.
- Cloud monitoring and operations will be an important area of focus for our service provider. A key to any good operations approach is good tooling. While OpenStack Compute provides nova-instance monitor, which tracks compute node utilization, we’re really going to need a number of third party tools for monitoring.
- Policy is an extremely important area but very provider specific. Everything from quotas (which are supported) to quality of service (QoS) to privacy controls can fall under this. I’ve given OpenStack Nova partial coverage here, but that might vary depending on the intricacies of the provider’s needs. For the record, the Catus release of OpenStack Compute provides quotas for instances (number and cores used, volumes (size and number), floating IP addresses and metadata.
- Scheduling within OpenStack Compute is fairly rudimentary for larger installations today. The pluggable scheduler supports chance (random host assignment), simple (least loaded) and zone (random nodes within an availability zone). As within most areas on this list, this will be greatly augmented in Diablo. In development are distributed schedulers and schedulers that understand heterogeneous hosts (for support of GPUs and differing CPU architectures).

We selected Linux platform for creating our infrastructure and we used Ubuntu 12.04 as our base Operating System. A reason behind selecting Ubuntu is that Ubuntu is most updated operating system of open source world and developers give latest features and greatest security to Ubuntu. And reason for selecting 12.04 is that it is LTS version and latest version of Ubuntu. And its supports and updates available till 2017.Our Hardware requirement are like 12 GB RAM, 5 TB Hard Drive and 5 to 6 GHz speed. But here all requirements are not available with us so we are using a pc which has 4 GB of ram, 500GB of Hard Drive and 3.1 GHz of processing power.
And we followed bellowed command to create our Clod Infrastructure as detailed in charter 2.

Step 3.1: Prepare System for Cloud Infrastructure

Here is basic Ubuntu 12.04 is installed in to pc with very basic packages given by Ubuntu distribution and now we follow below instructions

3.1.1 Creating a Volume Group

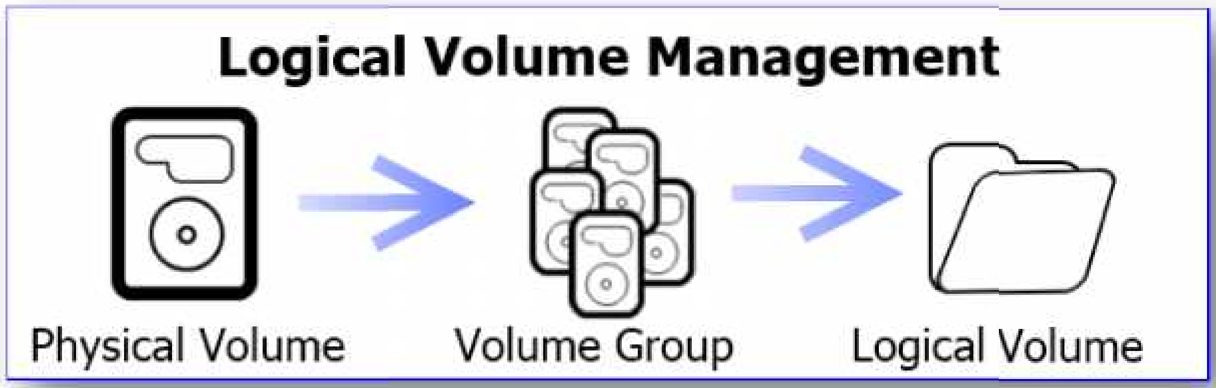


Figure 3.1.1.1 Steps to create logical volume

LVM is an abstraction layer between operating system and physical hard drives. What that means is your physical hard drives and partitions are no longer tied to the hard drives and partitions they reside on. Rather, the hard drives and partitions that your operating system sees can be any number of separate hard drives pooled together or in a software RAID.

Logical volumes are the partitions that operating system uses in LVM. To create a logical volume we first need to have a physical volume and volume group. Here are all of the steps necessary to create a new logical volume.

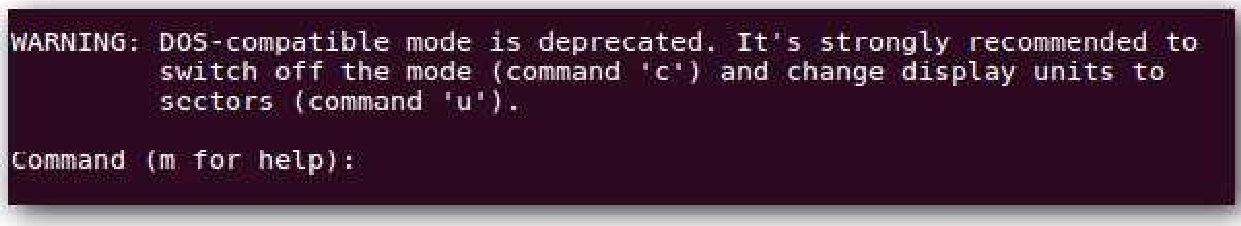
3.1.1.1 Create physical volume

We start from scratch with a brand new hard drive with no partitions or information on it. Start by finding which disk you will be working with (/dev/sda, sdb, etc.).

Our new disk is located at /dev/sdb so let's use fdisk to create a new partition on the drive.From a terminal type the following commands:

```
fdisk /dev/sdb
```

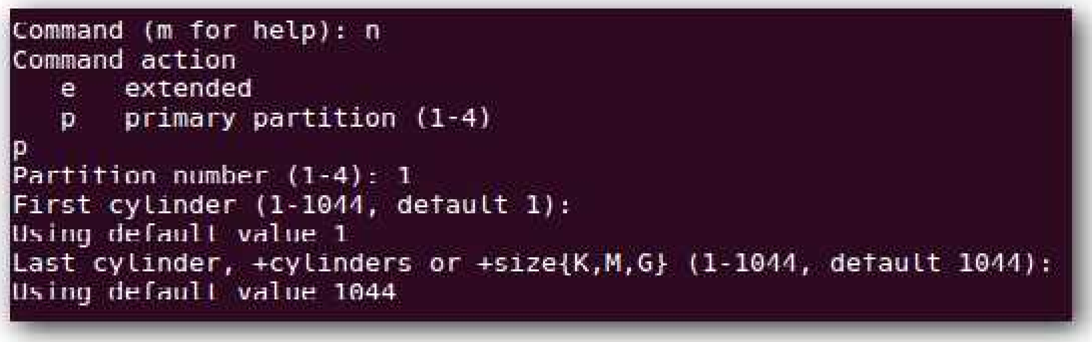
This will put you in a special fdisk prompt:



```
WARNING: DOS-compatible mode is deprecated. It's strongly recommended to
         switch off the mode (command 'c') and change display units to
         sectors (command 'u').

Command (m for help):
```

Figure 3.1.1.1.1 Output of command fdisk /dev/sdb

Push enters twice to accept the default first cylinder and last cylinder.

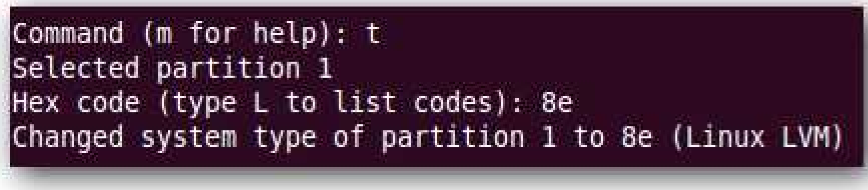


```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-1044, default 1):
Using default value 1
Last cylinder, +cylinders or +size{K,M,G} (1-1044, default 1044):
Using default value 1044
```

Figure 3.1.1.1.2 Partition number and default first and last cylinder

Prepare the partition to be used by LVM use the following two commands:

- t = change partition type
- 8e = changes to LVM partition type

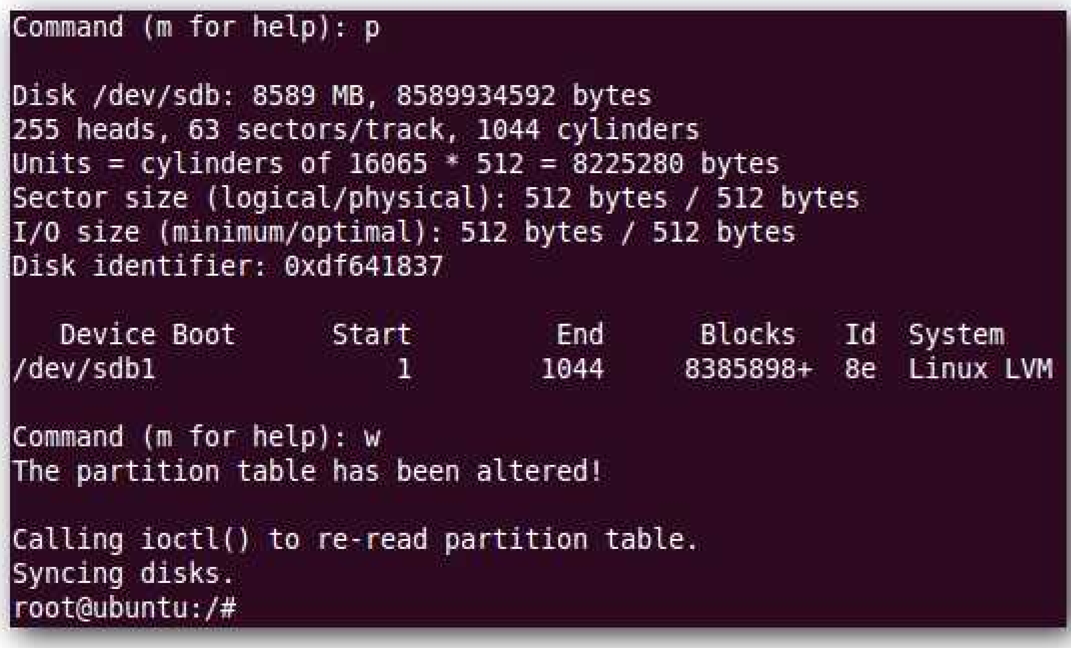


```
Command (m for help): t
Selected partition 1
Hex code (type L to list codes): 8e
Changed system type of partition 1 to 8e (Linux LVM)
```

Figure 3.1.1.1.3 Hex Code 8e for LVM Partition type

Verify and write the information to the hard drive.

- p = view partition setup so we can review before writing changes to disk
- w = write changes to disk

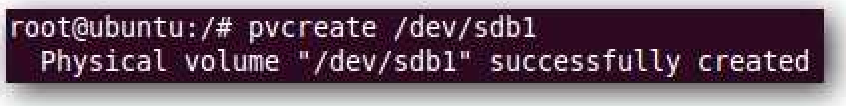


```
Command (m for help): p

Disk /dev/sdb: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xdf641837

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1        1044     8385898+  8e  Linux LVM

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
root@ubuntu:/#
```

Figure 3.1.1.1.4 First Partition named /dev/sdb1

After those commands, the fdisk prompt should exit and you will be back to the bash prompt of your terminal.

Enter pvcreate /dev/sdb1 to create a LVM physical volume on the partition we just created. You may be asking why we didn't format the partition with a file system but don't worry, that step comes later.



```
root@ubuntu:/# pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created
```

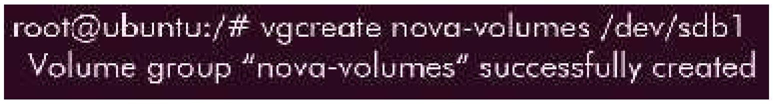
Figure 3.1.1.1.5 Creating LVM Physical volume

Now that we have a partition designated and physical volume created we need to create the volume group. Luckily this only takes one command.

3.1.1.2 Create volume Group

```
vgcreate nova-volumes /dev/sdb1
```

Nova-volumes is the name of the new volume group we created.



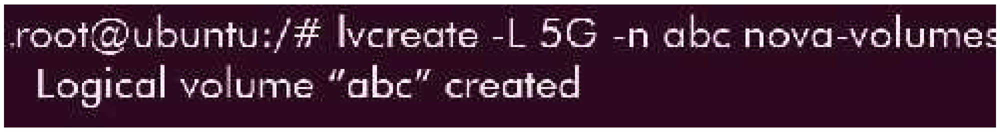
```
root@ubuntu:/# vgcreate nova-volumes /dev/sdb1
  Volume group "nova-volumes" successfully created
```

Figure 3.1.1.2.1 Creating Volume Group named nova-volumes

3.1.1.3Create logical volume

To create the logical volume that LVM will use:

```
lvcreate -L 3G -n openstack nova-volumes
```



```
.root@ubuntu:/# lvcreate -L 5G -n abc nova-volumes
  Logical volume "abc" created
```

Figure 3.1.1.3.1 Create Logical Volume named abc

The -L command designates the size of the logical volume, in this case 5 GB, and the -n command names the volume .nova-volumes is referenced so that the lvcreate command knows what volume to get the space from.

3.1.2 Install NTP server:

```
apt-get install ntp
```

What is ntp? & Why?

“Network Time Protocol “

Protocol designed to synchronize the clocks of computer over a network.

Time is inherently important to the function of routers and networks. It provides the only frame of reference between all devices on the network. This makes synchronized time extremely important. Without synchronized time, accurately correlating information between devices becomes difficult, if not impossible. When it comes to security, if you cannot successfully compare logs between each of your routers and all your network servers, you will find it very hard to develop a reliable picture of an incident. Finally, even if you are able to put the pieces together, unsynchronized times, especially between log files, may give an attacker with a good attorney enough wiggle room to escape prosecution.

Configure files:

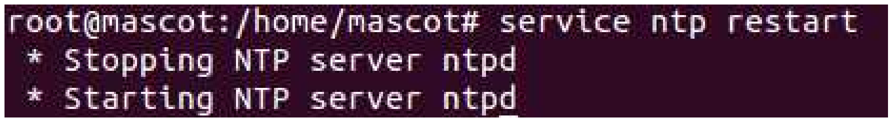
Open */etc/ntp.conf* in your favourite editor and add these lines:

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

Table 3.1.2.1 Configuration in ntp.config file

Restart NTP by issuing the command

```
service ntp restart
```



```
root@mascot:/home/mascot# service ntp restart
 * Stopping NTP server ntpd
 * Starting NTP server ntpd
```

Figure 3.1.2.1 Shows Status of ntp server

3.1.3 Install TGT server:

```
apt-get install tgt
```

What is TGT? & Why?

Target framework (tgt) aims to simplify various SCSI target driver (iSCSI, Fibre Channel, SRP, etc.) creation and maintenance. The key goals are the clean integration into the scsi-mid layer and implementing a great portion of tgt in user space

TGT Consist of kernel modules, user-space daemon and user-space tools.

- TGT supports three target drives
- iSCSI
- Ibmvstgt
- Xenvscsifront/back

ISCSI is Internet SCSI (Small Computer System Interface), an Internet Protocol (IP)-based storage networking standard for linking data storage facilities, developed by the Internet Engineering Task Force (IETF). By carrying SCSI commands over IP networks, ISCSI is used to facilitate data transfers over intranets and to manage storage over long distances. The ISCSI protocol is among the key technologies expected to help bring about rapid development of the storage area network (SAN) market, by increasing the capabilities and performance of storage data transmission. Because of the ubiquity of IP networks, ISCSI can be used to transmit data over local area networks (LANs), wide area networks (WANs), or the Internet and can enable location-independent data storage and retrieval.

When an end user or application sends a request, the operating system generates the appropriate SCSI commands and data request, which then go through encapsulation and, if necessary, encryption procedures.

A packet header is added before the resulting IP packets are transmitted over an Ethernet connection. When a packet is received, it is decrypted (if it was encrypted before transmission), and disassembled, separating the SCSI commands and request. The SCSI commands are sent on to the SCSI controller, and from there to the SCSI storage device. Because iSCSI is bi-directional, the protocol can also be used to return data in response to the original request.Given that we'll be running nova-compute on this machine as well, we'll also need the openiscsi-client. Install it with:

```
apt-get install open-iscsi open-iscsi-utils
```

3.1.4 Network Configuration:

We need to make sure that our network is working as expected. As pointed out earlier, the machine we're doing this on has two network interfaces, *eth0* and *eth1*. *eth0* is the machine's link to the outside world, *eth1* is the interface we'll be using for our virtual machines. We'll also make nova bridge clients via *eth0* into the internet. To achieve this kind of setup, first create the according network configuration in */etc/network/interfaces* (assuming that you are not using Network Manager). An example could look like this:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
address 10.10.13.95
network 10.0.0.0
netmask 255.255.255.255
broadcast 10.255.255.255
gateway 10.10.10.10

auto eth1
iface eth1 inet static
address 192.168.22.1
network 192.168.22.0
netmask 255.255.255.0
broadcast 192.168.22.255
```

Table 3.1.4.1 Network Configuration in config file

Total public host = 16777216

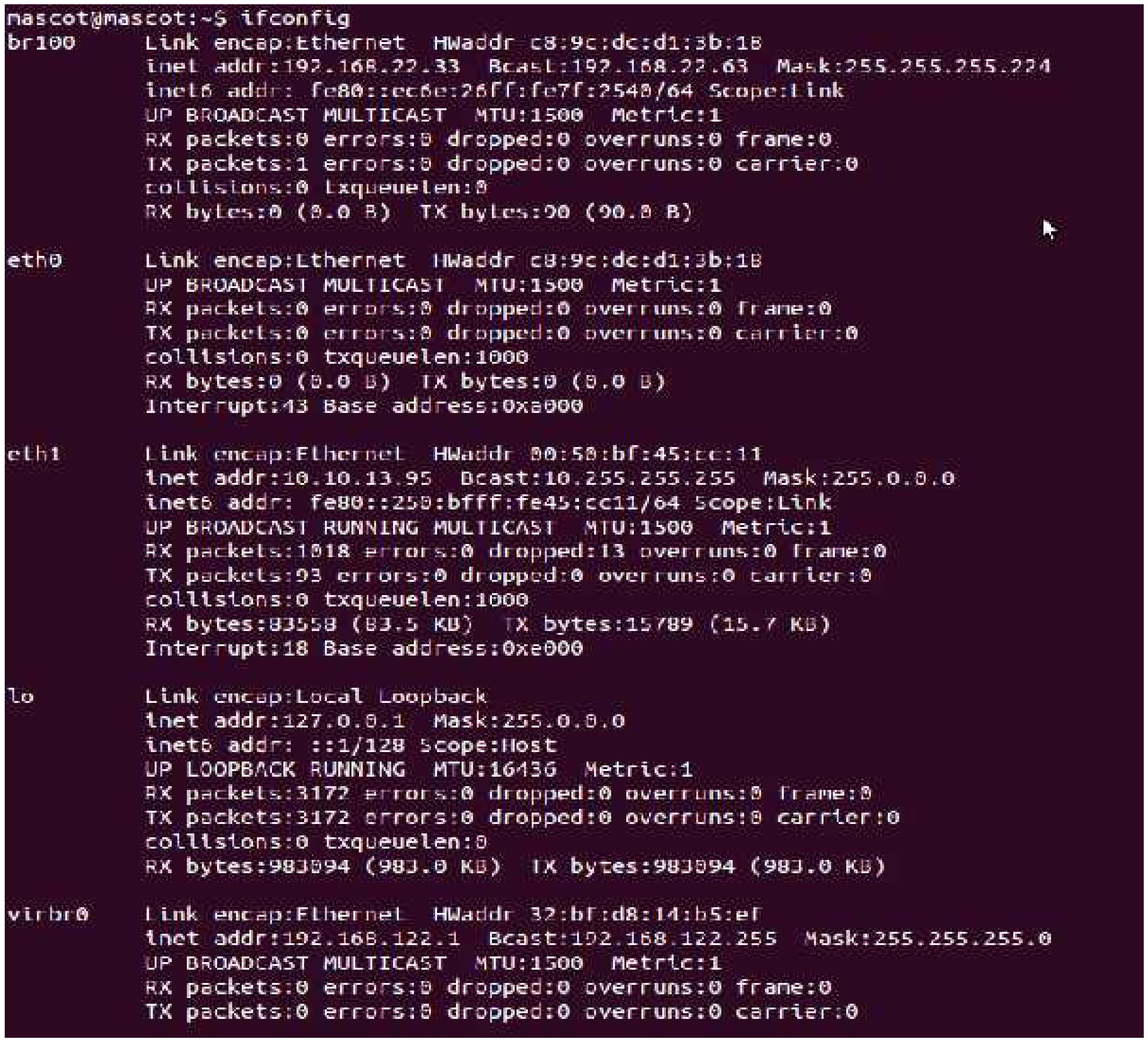
As you can see, the "public" network here is 10.10.0.0/16 while the "private" network (within which our VMs will be residing) is 192.168.22.0/24. This machine's IP address in the public network is 10.10.13.95 and we'll be using this IP in configuration files later on.After changing your network interfaces definition accordingly, make sure that the *bridge-utils* package is installed. Should it be missing on your system, install it with

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

A bridge is a way to connect two Ethernet segments together in a protocol independent way. Packets are forwarded based on Ethernet address, rather than IP address (like a router). Since forwarding is done at Layer 2, all protocols can go transparently through a bridge.

Then, restart your network with

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



```
mascot@mascot:~$ ifconfig
br100     Link encap:Ethernet  HWaddr c8:9c:dc:d1:3b:18
          inet addr:192.168.22.33  Bcast:192.168.22.63  Mask:255.255.255.224
          inet6 addr: fe80::ec6e:26ff:fe7f:2540/64 Scope:Link
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:90 (90.0 B)

eth0      Link encap:Ethernet  HWaddr c8:9c:dc:d1:3b:18
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:43 Base address:0xa000

eth1      Link encap:Ethernet  HWaddr 00:50:bf:45:cc:11
          inet addr:10.10.13.95  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::250:bfff:fe45:cc11/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1018 errors:0 dropped:13 overruns:0 frame:0
          TX packets:93 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:83558 (83.5 KB)  TX bytes:15789 (15.7 KB)
          Interrupt:18 Base address:0xe000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:3172 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3172 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:983094 (983.0 KB)  TX bytes:983094 (983.0 KB)

virbr0    Link encap:Ethernet  HWaddr 32:bf:d8:14:b5:ef
          inet addr:192.168.122.1  Bcast:192.168.122.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
```

Figure 3.1.4.1 Network configuration on system

3.1.5 Install RabbitMQ:

We'll also need RabbitMQ, an AMQP-implementation, as that is what all OpenStack components use to communicate with each other, and memcached

What is RabbitMQ? & Why?

- RabbitMQ is a message broker. The principal idea is pretty simple: it accepts and forwards messages. You can think about it as a post office: when you send mail to the post box you're pretty sure that Mr. Postman will eventually deliver the mail to your recipient. Using this metaphor RabbitMQ is a post box, a post office and a postman.
- The major difference between RabbitMQ and the post office is the fact that it doesn't deal with paper, instead it accepts, stores and forwards binary blobs of data − *messages*.

```
apt-get install rabbitmq-server memcached python-memcache
```

3.1.6 Install Virtual Machine:

As we'll also want to run KVM virtual machines on this very same host, we'll need KVM and libvirt, which OpenStack uses to control virtual machines. Install these packages with:

KVM (for Kernel-based Virtual Machine) is a full virtualization solution for Linux on x86 hardware containing virtualization extensions. It consists of a loadable kernel module, kvm.so, that provides the core virtualization infrastructure and a processor specific module, kvm-intel or kvm-amd KVM also requires a modified QEMU although work is underway to get the required changes upstream.

Using KVM, one can run multiple virtual machines running unmodified Linux or Windows images. Each virtual machine has private virtualized hardware: a network card, disk, graphics adapter, etc.

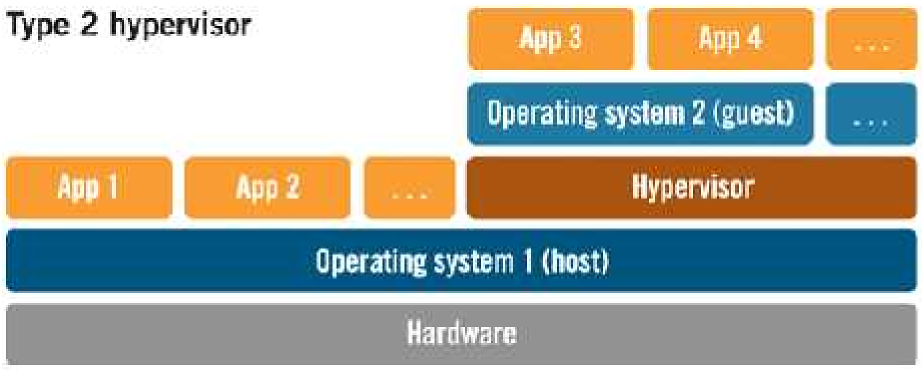


Figure 3.1.6.2 Virtualization or Type 2 hypervisor

Step 3.2 Install MySQL and create the necessary databases and users:-

Nova and glance will use MySQL to store their runtime data. To make sure they can do that, we'll install and set up MySQL. Do this:

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

MySQL
Username : root
Password : gcetdb

Table 3.2.1 MySQL credential

When the package installation is done and you want other machines (read: OpenStack computing nodes) to be able to talk to that MySQL database, too, open up */etc/mysql/my.cnf* in your favourite editor and change this line:

bind-address = 10.10.13.95

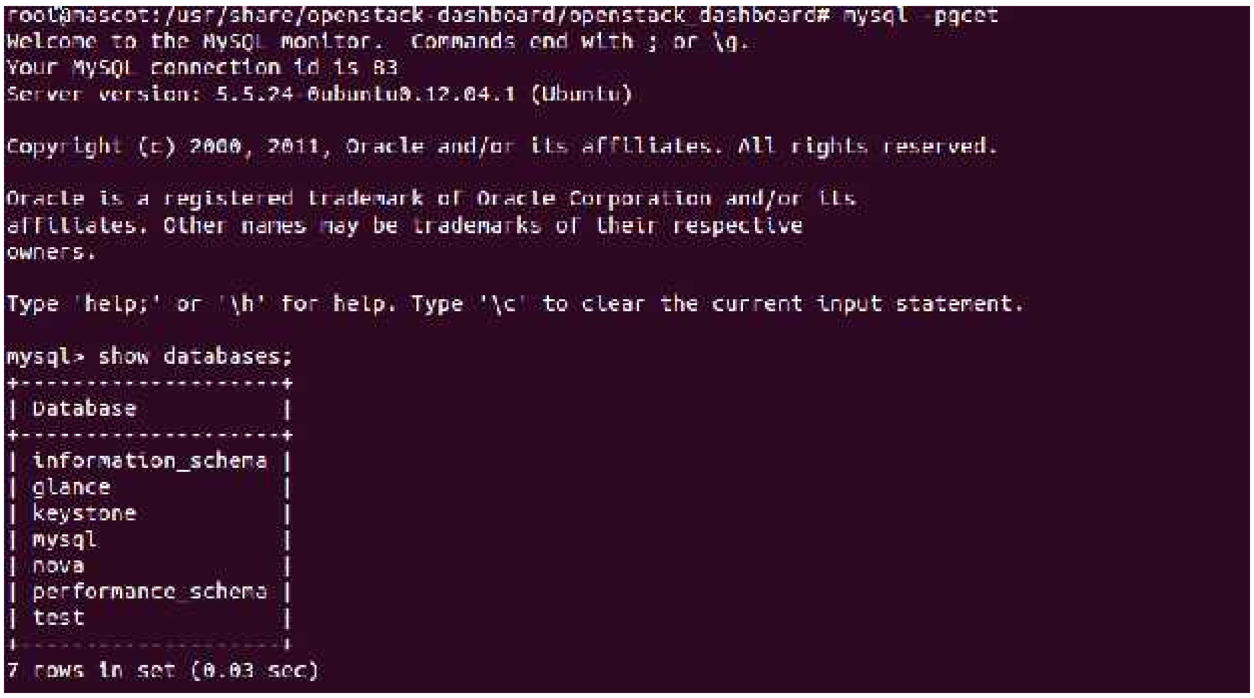
Then, restart MySQL:

Servicemysql restart

Now create the user accounts in mysql and grant them access on the according databases,

```
mysql -u root -pgcetdb<<EOF
CREATE DATABASE nova;
GRANT ALL PRIVILEGES ON nova.* TO 'novadbadmin'@'%'
IDENTIFIED BY 'dieD9Mie';
EOF
mysql -u root -pgcetdb<<EOF
CREATE DATABASE glance;
GRANT ALL PRIVILEGES ON glance.* TO 'glancedbadmin'@'%'
IDENTIFIED BY 'ohC3teiv';
EOF
mysql -u root <<EOF
CREATE DATABASE keystone;
GRANT ALL PRIVILEGES ON keystone.* TO 'keystonedbadmin'@'%'
IDENTIFIED BY 'Ue0Ud7ra';
EOF
```

Table 3.2.2 Command for creating MySQL database.



```
root@mascot:/usr/share/openstack-dashboard/openstack_dashboard# mysql -pgcet
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 83
Server version: 5.5.24-0ubuntu0.12.04.1 (Ubuntu)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| glance             |
| keystone           |
| mysql              |
| nova               |
| performance_schema |
| test               |
+--------------------+
7 rows in set (0.03 sec)
```

Figure 3.2.1 Database exists on system

Step 3.3 Install and configure Keystone:-

We can finally get to Open Stack now and we'll start by installing the Identity component, codenamed Keystone. Install the according packages:

What is keystone? & use of keystone

Keystone is the identity service used by OpenStack for authentication (authN) and high-level authorization (authZ). It currently supports token-based authN and user-service authorization. Keystone is an OpenStack project that provides Identity, Token, Catalogue and Policy services for use specifically by projects in the OpenStack family. It implements OpenStack's Identity API.

```
apt-get install keystone python-keystone python-mysqldb python-keystoneclient
```

Then, open /etc/keystone/keystone.confin an editor and make sure to set a value for admin_token. We'll use "hastexo" in this example.

Admin_token =”hastexo”

Scroll down to the section starting with *[sql]*. Change it to match the database settings that we defined for Keystone in step 2:

```
[sql]
connection = mysql://keystonedbadmin:Ue0Ud7ra@10.10.13.95/keystone
idle_timeout = 200
```

Restart Keystone by issuing this command:

```
service keystone restart
```

Then make Keystone create its tables within the freshly created *keystone* database:

```
keystone-manage db_sync
```

The next step is to fill Keystone with actual data. use the script attached to this blog entry entitled*keystone_data.sh_.txt.* It's courtesy of the Devstack project with some adaptions. Rename the file to *keystone_data.sh.* Be sure to replace the admin password (*ADMIN_PASSWORD* variable) and the value for SERVICE_TOKEN with the entry you specified in keystone.conf for *admin_token* earlier. Then just make the script executable and call it; if everything goes well, it should deliver a return code of 0.

Use the *endpoints.sh._txt* script attached to this text to do that; rename the script to *endpoints.sh* and make sure it's executable. It takes several parameters - a typical call would look like this:

```
./endpoints.sh -m 10.10.13.95 -u keystonedbadmin -D keystone -p Ue0Ud7ra -K 10.10.13.95 -R
RegionOne -E "http://localhost:35357/v2.0" -S 10.10.13.95 -T hastexo
```

The values used have the following meanings:

- -m 10.10.13.95 - the host where your MySQL database is running -u keystonedbadmin - the name of the keystone user that may access the mysql database
- -D keystone - the database that belongs to Keystone in MySQL (as defined in step 2)
- -p Ue0Ud7ra - the password of the keystone MySQL user to access the database (as defined in step 2)
- -K 10.10.13.95 - the host where all your OpenStack services will initially run
- -R RegionOne - the standard region for your endpoints; leave unchanged when following this howto.
- -E "http://localhost:35357/v2.0" - the keystone endpoint for user authentication; leave unchanged when following this howto.
- -S 10.10.13.95 - Should you wish to run Swift at a later point, put in the IP address of the *swift-proxy* server here.
- -T hastexo - the token you put into keystone.conf; use *hastexo* when following this howto.

Step 3.4 Install and configure Glance

The next step on our way to OpenStack is its Image Service, codenamed Glance. First, install the packages necessary for it:

When that is done, open */etc/glance/glance-api-paste.ini* in an editor and scroll down to the end of the document. You'll see these three lines at its very end:

```
admin_tenant_name = service
admin_user = glance
admin_password = hastexo
```

After this, open */etc/glance/glance-registry-paste.ini* and scroll to that file's end, too. Adapt it in the same way you adapted */etc/glance/glance-api-paste.ini* earlier.

Please open */etc/glance/glance-registry.conf* now and scroll down to the line starting with *sql_connection*.

This is where we tell Glance to use MySQL;

according to the MySQL configuration we created earlier, the *sql_connection*-line for this example would look like this:

sql_connection = mysql://glancedbadmin:ohC3teiv@**10.10.13.95**/glance

After this, scroll down until the end of the document and add these two lines:

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

These two lines instruct the Glance Registry to use Keystone for authentication, which is what we want. Now we need to do the same for the Glance API. Open */etc/glance/glance-api.conf* and add these two lines at the end of the document:

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

Afterwards, you need to initially synchronize the Glance database by running these commands:

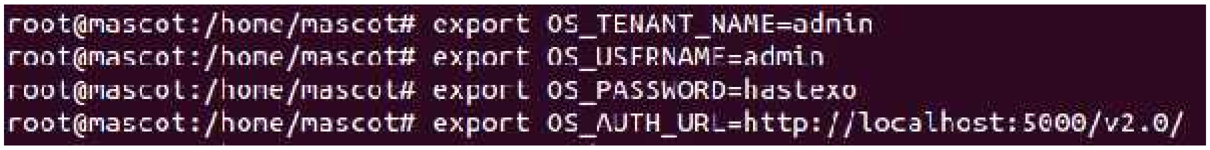
```
glance-manage version_control 0
glance-manage db_sync
```

It's time to restart Glance now:

```
service glance-api restart && service glance-registry restart
```

```
export OS_TENANT_NAME=admin
export OS_USERNAME=admin
export OS_PASSWORD=hastexo
export OS_AUTH_URL=http://localhost:5000/v2.0/
```

Table 3.4.1 Credential for Glance api



```
root@mascot:/home/mascot# export OS_TENANT_NAME=admin
root@mascot:/home/mascot# export OS_USERNAME=admin
root@mascot:/home/mascot# export OS_PASSWORD=hastexo
root@mascot:/home/mascot# export OS_AUTH_URL=http://localhost:5000/v2.0/
```

Figure 3.4.1 Credential for Glance api

The first three entries are identical with what you inserted into Glance's API configuration files earlier and the entry for OS_AUTH_URL is mostly generic and should just work. After exporting these variables, you should be able to do

```
glance index
```

and get no output at all in return (but the return code will be 0; check with *echo $?*). If that's the case, Glance is setup correctly and properly connects with Keystone. Now let's add our first image!

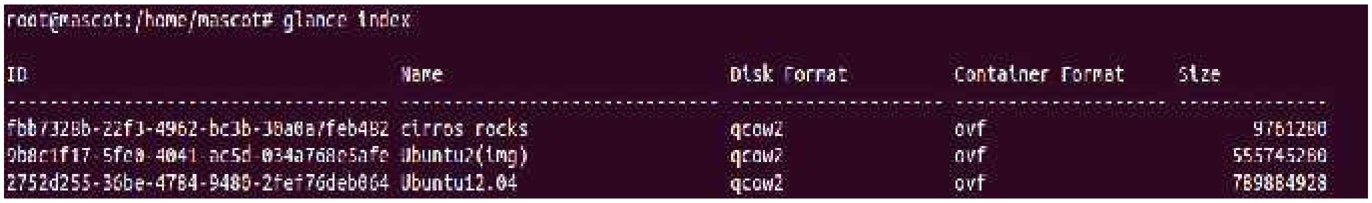
We'll be using a Ubuntu desktop image for this. Download one:

```
wget http://uec-images.ubuntu.com/releases/12.04/release/ubuntu-12.04-Deskt-p.img
```

Then add this image to Glance:

```
glance add name="Ubuntu 12.04 Desktop" is_public=true
container_format=ovfdisk_format=qcow2 < ubuntu-12.04-Desktop.img
```

glance index



```
root@mascot:/home/mascot# glance index

ID                                   Name                           Disk Format          Container Format     Size
------------------------------------ ------------------------------ -------------------- -------------------- --------------
fbb732Bb-22f3-4962-bc3b-38a0a7feb482 cirros rocks                   qcow2                ovf                        9761280
9b8c1f17-5fe0-4041-ac5d-034a768e5afe Ubuntu2(img)                   qcow2                ovf                      555745280
2752d255-36be-4784-9480-2fef76deb064 Ubuntu12.04                    qcow2                ovf                      789884928
```

Figure 3.4.2 Image uploaded on Cloud

Step 3.5 Install and configure Nova

OpenStack Compute, codenamed Nova, is by far the most important and the most substantial OpenStack component. Whatever you do when it comes to managing VMs will be done by Nova in the background. The good news is: Nova is basically controlled by one configuration file, */etc/nova/nova.conf*. Get started by installing all nova-related components:

```
apt-get install nova-api nova-cert nova-common nova-compute nova-compute-kvm nova-
doc nova-network nova-objectstore nova-scheduler nova-volume nova-consoleauth novnc
python-nova python-novaclient
```

Then, open */etc/nova/nova.conf* and replace everything in there with these lines:

```
--dhcpbridge_flagfile=/etc/nova/nova.conf
--dhcpbridge=/usr/bin/nova-dhcpbridge
--logdir=/var/log/nova
--state_path=/var/lib/nova
--lock_path=/var/lock/nova
--allow_admin_api=true
--use_deprecated_auth=false
--auth_strategy=keystone
--scheduler_driver=nova.scheduler.simple.SimpleScheduler
--s3_host=10.10.13.95
--ec2_host=10.10.13.95
--rabbit_host=10.10.13.95
--cc_host=10.10.13.95
--nova_url=http://10.10.13.95:8774/v1.1/
--routing_source_ip=10.10.13.95
--glance_api_servers=10.10.13.95:9292
```

```
--image_service=nova.image.glance.GlanceImageService
--iscsi_ip_prefix=192.168.22
--sql_connection=mysql://novadbadmin:dieD9Mie@10.10.13.95/nova
--ec2_url=http://10.10.13.95:8773/services/Cloud
--keystone_ec2_url=http://10.10.13.95:5000/v2.0/ec2tokens
--api_paste_config=/etc/nova/api-paste.ini
--libvirt_type=kvm
--libvirt_use_virtio_for_bridges=true
--start_guests_on_host_boot=true
--resume_guests_state_on_host_boot=true
--vnc_enabled=true
--vncproxy_url=http://10.10.13.95:6080
--vnc_console_proxy_url=http://10.10.13.95:6080
# network specific settings
--network_manager=nova.network.manager.FlatDHCPManager
--public_interface=eth0
--flat_interface=eth1
--flat_network_bridge=br100
--fixed_range=192.168.22.32/27
--floating_range=10.10.255.255/16
--network_size=65535
--flat_network_dhcp_start=192.168.22.33
--flat_injected=False
--force_dhcp_release
--iscsi_helper=tgtadm
--connection_type=libvirt
--root_helper=sudo nova-rootwrap
--verbose
--libvirt_use_virtio_for_bridges
--ec2_private_dns_show
--novnc_enabled=true
--novncproxy_base_url=http://10.10.13.95:6080/vnc_auto.html
--vncserver_proxyclient_address=10.10.13.95
--vncserver_listen=10.10.13.95
```

Table 3.5.1 nova configuration

As you can see, many of the entries in this file are self-explanatory; the trickiest bit to get done right is the network configuration part, which you can see at the end of the file. We're using Nova's FlatDHCP network mode; 192.168.22.32/27 is the fixed range from which our future VMs will get their IP adresses, starting with 192.168.22.33. Our flat interface is eth1 (nova-network will bridge this into a bridge named *br100*), our public interface is eth0. An additional floating range is defined at 10.10.255.255/16 (for those VMs that we want to have a 'public IP').

After saving *nova.conf*, **open** ***/etc/nova/api-paste.ini*** in an editor and scroll down to the end of the file. **Adapt it** according to the changes you conducted in Glance's paste-files in step 3. Use *service* as tenant name and *nova* as username.

Then, restart all nova services to make the configuration file changes take effect:

```
for a in libvirt-bin nova-network nova-compute nova-cert nova-api nova-objectstore nova-scheduler nova-volume novnc nova-consoleauth; do service "$a" stop; done
```

```
for a in libvirt-bin nova-network nova-compute nova-cert nova-api nova-objectstore nova-scheduler nova-volume novnc nova-consoleauth; do service "$a" start; done
```

The next step will create all databases Nova needs in MySQL. While we are at it, we can also create the network we want to use for our VMs in the Nova databases. Do this:

```
nova-manage db sync
```

```
nova-manage network create private --fixed_range_v4=192.168.22.32/27 --num_networks=1 --bridge=br100 --bridge_interface=eth1 --network_size=32
```

Also, make sure that all files in /etc/nova belong to the nova user and the nova group:

```
chown -R nova:nova /etc/nova
```

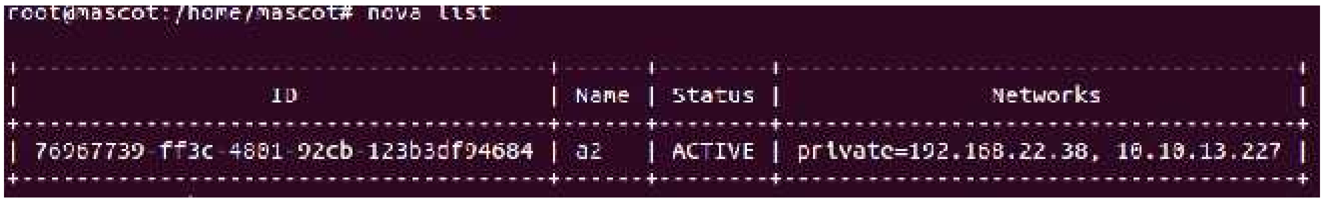
Then, restart all nova-related services again:

```
for a in libvirt-bin nova-network nova-compute nova-cert nova-api nova-objectstore nova-scheduler nova-volume novnc nova-consoleauth; do service "$a" stop; done
```

```
for a in libvirt-bin nova-network nova-compute nova-cert nova-api nova-objectstore nova-scheduler nova-volume novnc nova-consoleauth; do service "$a" start; done
```

You should now see all these nova-* processes when doing *psauxw*. And you should be able to use the numerous nova commands. For example,

```
nova list
```

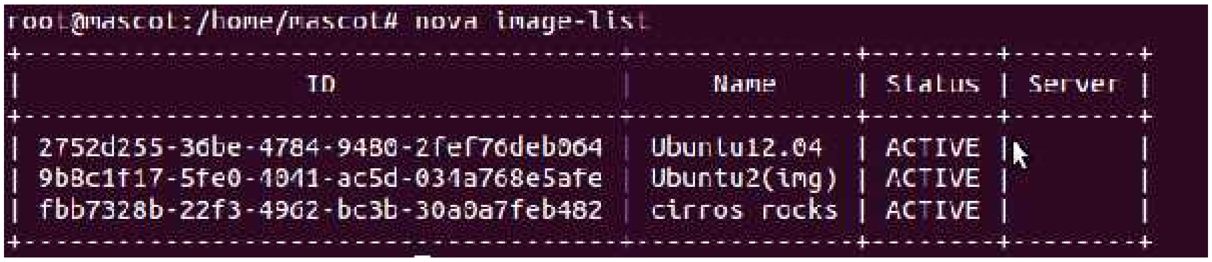


```
root@mascot:/home/mascot# nova list

+--------------------------------------+------+--------+---------------------------------------+
|                  ID                  | Name | Status |               Networks                |
+--------------------------------------+------+--------+---------------------------------------+
| 76967739-ff3c-4801-92cb-123b3df94684 | a2   | ACTIVE | private=192.168.22.38, 10.10.13.227  |
+--------------------------------------+------+--------+---------------------------------------+
```

Figure 3.5.1 nova-list

It should give you a list of all currently running VMs (none, the list should be empty).

```
nova image-list
```



```
root@mascot:/home/mascot# nova image-list
+--------------------------------------+--------------+--------+--------+
|                  ID                  |     Name     | Status | Server |
+--------------------------------------+--------------+--------+--------+
| 2752d255-36be-4784-9480-2fef76deb064 | Ubuntu12.04  | ACTIVE |        |
| 9b8c1f17-5fe0-4041-ac5d-034a768e5afe | Ubuntu2(img) | ACTIVE |        |
| fbb7328b-22f3-4962-bc3b-30a0a7feb482 | cirros rocks | ACTIVE |        |
+--------------------------------------+--------------+--------+--------+
```

Figure 3.5.2 nova image-list

It should show a list of the image you uploaded to Glance in the step before. If that's the case, Nova is working as expected and you can carry on with starting your first VM.

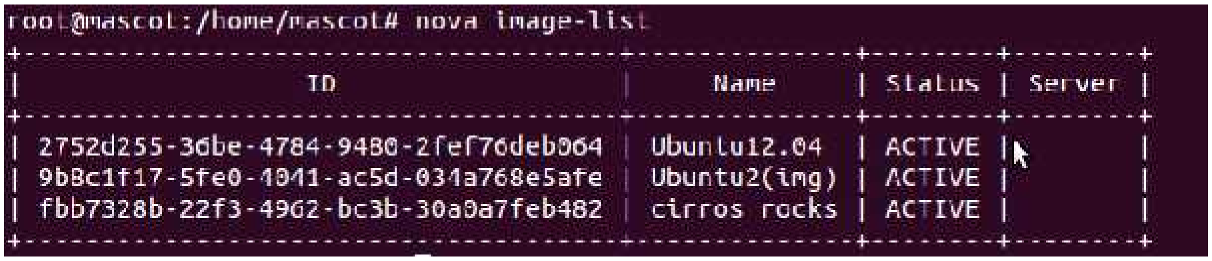
Step 3.6 Start first VM

Once Nova works as desired, starting your first own cloud VM is easy. As we're using a Ubuntu image for this example which allows for SSH-key based login only, we first need to store a public SSH key for our *admin* user in the OpenStack database. Upload the file containing your SSH public key onto the server and do this:

```
novakeypair-add --pub_key id_rsa.pub key1
```

This will add the key to OpenStack Nova and store it with the name "key1". The only thing left to do after this is firing up your VM. Find out what ID your Ubuntu image has, you can do this with:

```
nova image-list
```

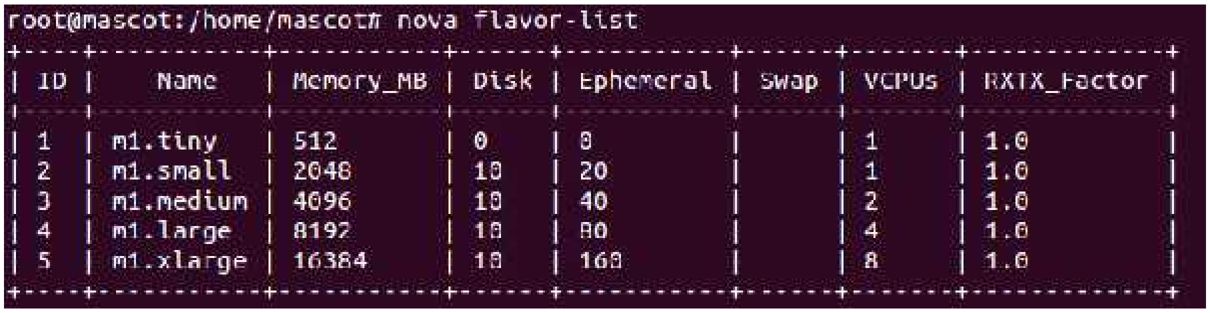


```
root@mascot:/home/mascot# nova image-list
+--------------------------------------+---------------+--------+--------+
| ID                                   | Name          | Status | Server |
+--------------------------------------+---------------+--------+--------+
| 2752d255-36be-4784-9480-2fef76deb064 | Ubuntu12.04   | ACTIVE |        |
| 9b8c1f17-5fe0-4041-ac5d-034a768e5afe | Ubuntu2(img)  | ACTIVE |        |
| fbb7328b-22f3-4962-bc3b-30a0a7feb482 | cirros rocks  | ACTIVE |        |
+--------------------------------------+---------------+--------+--------+
```

Figure 3.6.1 nova image-list

When starting a VM, you also need to define the flavor it is supposed to use. Flavors are pre-defined hardware schemes in OpenStack with which you can define what resources your newly created VM has. OpenStack comes with five pre-defined flavors; you can get an overview over the existing flavors with

```
nova flavor-list
```



```
root@mascot:/home/mascot# nova flavor-list
+----+-----------+-----------+------+-----------+------+-------+-------------+
| ID |    Name   | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor |
+----+-----------+-----------+------+-----------+------+-------+-------------+
| 1  | m1.tiny   | 512       | 0    | 0         |      | 1     | 1.0         |
| 2  | m1.small  | 2048      | 10   | 20        |      | 1     | 1.0         |
| 3  | m1.medium | 4096      | 10   | 40        |      | 2     | 1.0         |
| 4  | m1.large  | 8192      | 10   | 80        |      | 4     | 1.0         |
| 5  | m1.xlarge | 16384     | 10   | 160       |      | 8     | 1.0         |
+----+-----------+-----------+------+-----------+------+-------+-------------+
```

Figure 3.6.2 nova flavor-list

Flavors are referenced by their ID, not by their name. That's important for the actual command to execute to start your VM. That command's syntax basically is this:

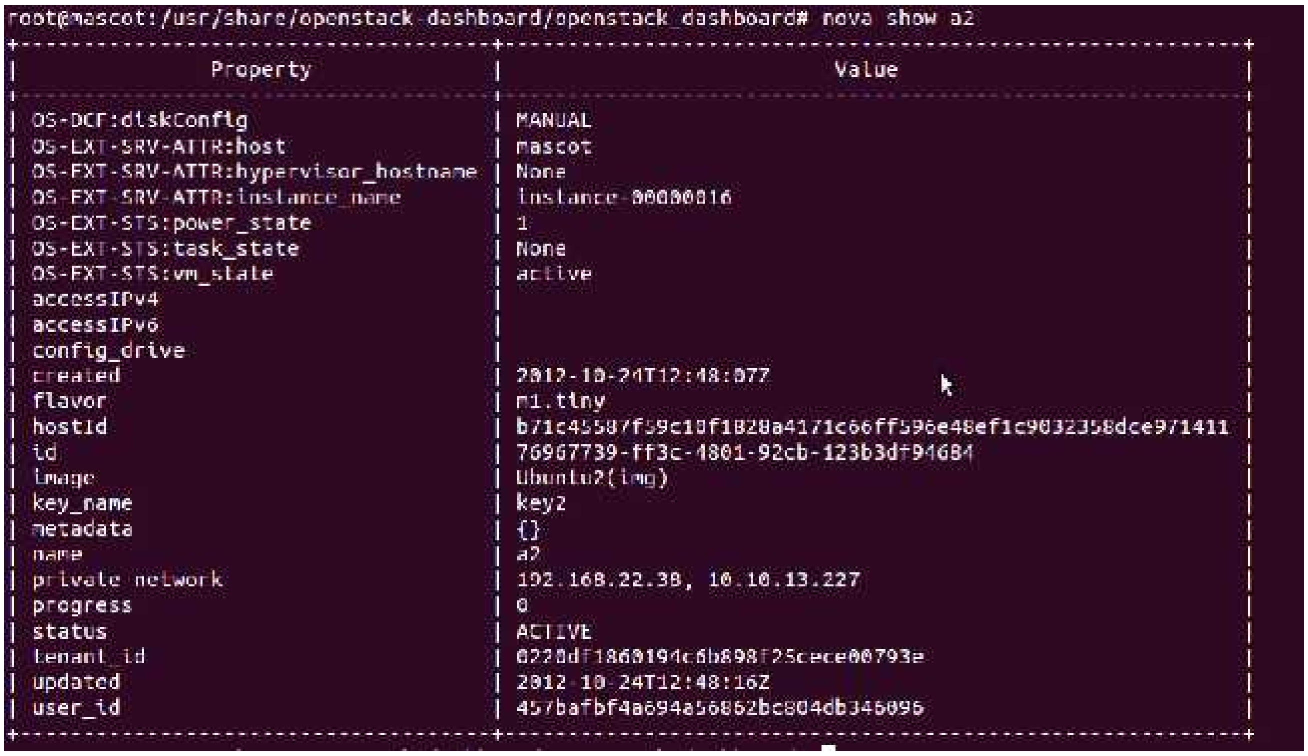
nova boot --flavor *ID* --image *Image-UUID* --key_name*key-name vm_name*

Here's the command you would need to start that particular VM:

```
nova boot --flavor 1 --image 9bab7ce7-7523-4d37-831f-c18fbc5cb543 --key_name key1 a2
```

After hitting the Enter key, Nova will show you a summary with all important details concerning the new VM. After some seconds, issue the command

```
nova show a2
```



```
root@mascot:/usr/share/openstack-dashboard/openstack_dashboard# nova show a2
+-------------------------------------+----------------------------------------------------------+
|              Property               |                          Value                           |
+-------------------------------------+----------------------------------------------------------+
| OS-DCF:diskConfig                   | MANUAL                                                   |
| OS-EXT-SRV-ATTR:host                | mascot                                                   |
| OS-EXT-SRV-ATTR:hypervisor_hostname | None                                                     |
| OS-EXT-SRV-ATTR:instance_name       | instance-00000016                                        |
| OS-EXT-STS:power_state              | 1                                                        |
| OS-EXT-STS:task_state               | None                                                     |
| OS-EXT-STS:vm_state                 | active                                                   |
| accessIPv4                          |                                                          |
| accessIPv6                          |                                                          |
| config_drive                        |                                                          |
| created                             | 2012-10-24T12:48:07Z                                     |
| flavor                              | m1.tiny                                                  |
| hostId                              | b71c45587f59c10f1828a4171c66ff596e48ef1c9032358dce971411 |
| id                                  | 76967739-ff3c-4801-92cb-123b3df94684                     |
| image                               | Ubuntu2(img)                                             |
| key_name                            | key2                                                     |
| metadata                            | {}                                                       |
| name                                | a2                                                       |
| private network                     | 192.168.22.38, 10.10.13.227                              |
| progress                            | 0                                                        |
| status                              | ACTIVE                                                   |
| tenant_id                           | 0220df1860194c6b898f25cece00793e                         |
| updated                             | 2012-10-24T12:48:16Z                                     |
| user_id                             | 457bafbf4a694a56862bc804db346096                         |
+-------------------------------------+----------------------------------------------------------+
```

Figure 3.6.3 Status of project named a2

In the line with the *private_network* keyword, you'll see the IP address that Nova has assigned this particular VM. As soon as the VMs status is *ACTIVE*, you should be able to log into that VM by issuing

```
ssh -i Private-Keyubuntu@IP
```

Of course *Private-Key* needs to be replaced with the path to your SSH private key and *IP* needs to be replaced with the VMs actual IP. If you're using SSH agent forwarding, you can leave out the "-i"-parameter altogether.

Step 3.7 The OpenStack Dashboard

We can use Nova to start and stop virtual machines now, but up to this point, we can only do it on the command line. That's not good, because typically, we'll want users without high-level administrator skills to be able to start new VMs. There's a solution for this on the OpenStack ecosystem called Dashboard, codename Horizon. Horizon is OpenStack's main configuration interface. It's django-based.

Let's get going with it:

```
apt-get install apache2 libapache2-mod-wsgi openstack-dashboard
```

Note: Make sure to install at least the version 2012.1-0ubuntu6 of the openstack-dashboard package, as it contains some changes important for the dashboard to work properly.

Then, open */etc/openstack-dashboard/local_settings.py* in an editor. Go to the line starting with *CACHE_BACKEND* and make sure it looks like this:

```
CACHE_BACKEND = 'memcached://127.0.0.1:11211/'
```

Now restart Apache with

```
service apache2 restart
```

Step 3.8 Multiple Compute Node Configuration

Goal is to split VM load across more than one server by connecting an additional nova-compute node to a cloud controller node. This configuring can be reproduced on multiple compute servers to start building a true multi-node Compute cluster.To build out and scale the Compute platform, it spread out services amongst many servers. While there are additional ways to accomplish the build-out, this section describes adding compute nodes, and the service we are scaling out is called 'nova-compute.'

For a multi-node install we have to only make changes to nova.conf and copy it to additional compute nodes. Ensure each nova.conf file points to the correct IP addresses for the respective services. Customize the nova.conf example below to match your environment.

Configuring your Compute installation involves many configuration files - the nova.conf file, the api-paste.ini file, and related Image and Identity management configuration files.When running in a high-availability mode for networking, the compute node is where you configure the compute network, the networking between your instances. Learn more about high-availability for networking in the Compute Administration manual.Because you may need to query the database from the compute node and learn more information about instances, the nova client and MySQL client or PostgresSQL client packages should be installed on any additional compute nodes.

Copy the nova.conf from your controller node to all additional compute nodes. Modify the following configuration options so that they match the IP address of the compute host:

- my_ip
- vncserver_listen
- vncserver_proxyclient_address

Restart networking:

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

With nova.conf updated and networking set, configuration is nearly complete. First, bounce the relevant services to take the latest updates:

```
restart libvirt-bin; service nova-compute restart
```

To avoid issues with KVM and permissions with Nova, run the following commands to ensure we have VM's that are running optimally:

```
chgrp kvm /dev/kvm
chmod g+rwx /dev/kvm
```

If you want to use the 10.04 Ubuntu Enterprise Cloud images that are readily available at http://uec-images.ubuntu.com/releases/10.04/release/, you may run into delays with booting. Any server that does not have nova-api running on it needs this iptables entry so that UEC images can get metadata info. On compute nodes, configure the iptables with this next step:

```
 # iptables -t nat -A PREROUTING -d 169.254.169.254/32 -p tcp -m tcp --dport 80 -j DNAT --
to-destination $NOVA_API_IP:8773
```

Lastly, confirm that your compute node is talking to your cloud controller. From the cloud controller, run this database query:

```
mysql -u$MYSQL_USER -p$MYSQL_PASS nova -e 'select * from services;'
```

In return, you should see something similar to this:

```
+---------------------+---------------------+------------+---------+----+----------+----------------+-----------
-+--------------+----------+-------------------+
| created_at          | updated_at          | deleted_at | deleted | id | host     | binary         | topic      |
report_count | disabled | availability_zone |
+---------------------+---------------------+------------+---------+----+----------+----------------+-----------
-+--------------+----------+-------------------+
| 2013-01-28 22:52:46 | 2013-02-03 06:55:48 | NULL       |       0 |  1 | osdemo02 | nova-network
| network   |        46064 |        0 | nova              |
| 2013-01-28 22:52:48 | 2013-02-03 06:55:57 | NULL       |       0 |  2 | osdemo02 | nova-compute
| compute   |        46056 |        0 | nova              |
| 2013-01-28 22:52:52 | 2013-02-03 06:55:50 | NULL       |       0 |  3 | osdemo02 | nova-scheduler
| scheduler |        46065 |        0 | nova              |
| 2013-01-29 23:49:29 | 2013-02-03 06:54:26 | NULL       |       0 |  4 | osdemo01 | nova-compute
| compute   |        37050 |        0 | nova              |
| 2013-01-30 23:42:24 | 2013-02-03 06:55:44 | NULL       |       0 |  9 | osdemo04 | nova-compute
| compute   |        28484 |        0 | nova              |
| 2013-01-30 21:27:28 | 2013-02-03 06:54:23 | NULL       |       0 |  8 | osdemo05 | nova-compute
| compute   |        29284 |        0 | nova              |
+---------------------+---------------------+------------+---------+----+----------+----------------+-----------
-+--------------+----------+-------------------+
```

Step 3.9 MooseFS File System

MooseFS is a fault tolerant, network distributed file system. It spreads data over several physical servers which are visible to the user as one resource. For standard file operations MooseFS acts as other Unix-alike file systems:

- A hierarchical structure (directory tree)
- Stores POSIX file attributes (permissions, last access and modification times)
- Supports special files (block and character devices, pipes and sockets)
- Access to the file system can be limited based on IP address and/or password

Distinctive features of MooseFS are:

- High reliability (several copies of the data can be stored across separate computers)
- Capacity is dynamically expandable by attaching new computers/disks
- Deleted files are retained for a configurable period of time (a file system level "trash bin")

Coherent snapshots of files, even while the file is being written/accessed

ARCHITECTURE

MooseFS consists of four components:

- Managing server (master server) – a single machine managing the whole filesystem, storing metadata for every file (information on size, attributes and file location(s), including all information about non-regular files, i.e. directories, sockets, pipes and devices).
- Data servers (chunk servers) - any number of commodity servers storing files data and synchronizing it among themselves (if a certain file is supposed to exist in more than one copy).
- Metadata backup server(s) (metalogger server) - any number of servers, all of which store metadata changelogs and periodically downloading main metadata file; so as to promote these servers to the the role of the Managing server when primary master stops working.
- Client computers that access (mount) the files in MooseFS - any number of machines usingmfsmount process to communicate with the managing server (to receive and modify file metadata) and with chunkservers (to exchange actual file data).

mfsmount is based on the FUSE mechanism (Filesystem in USErspace), so MooseFS is available on every Operating System with a working FUSE implementation (Linux, FreeBSD, MacOS X, etc.)

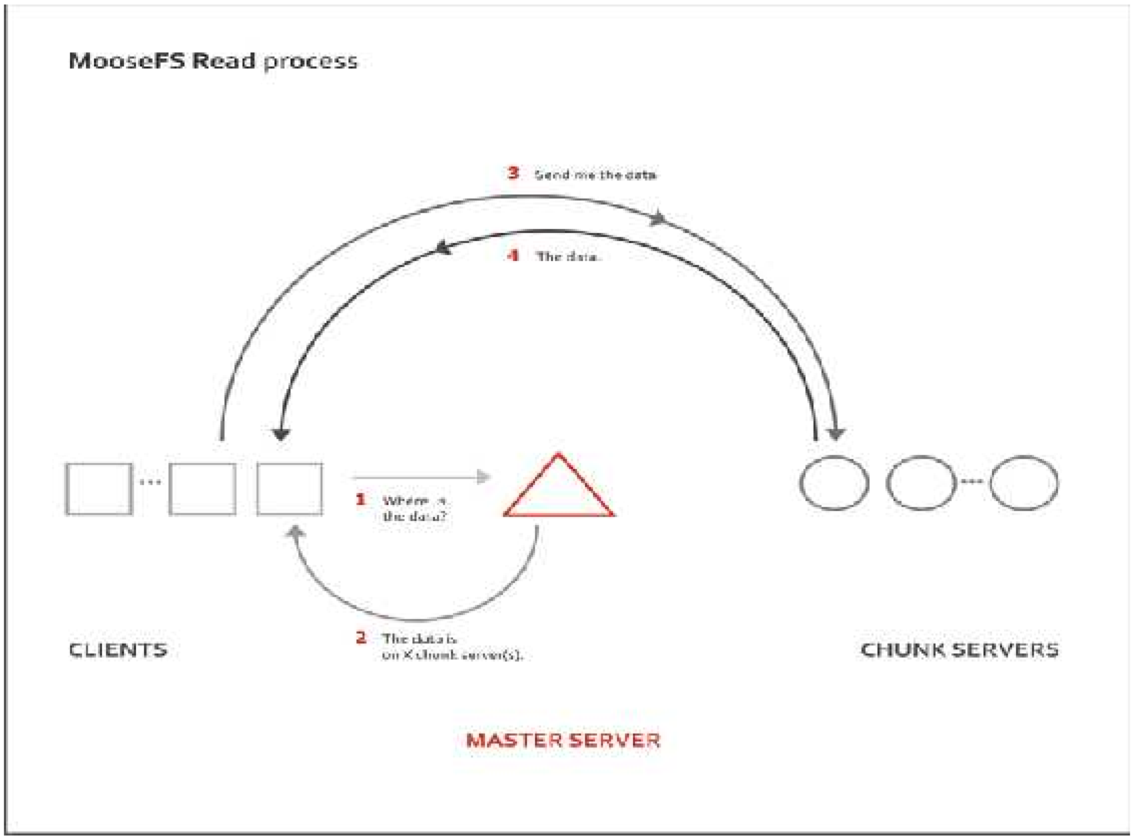


Figure 3.9.1: MooseFS Read Process

HOW THE SYSTEM WORKS:

All file operations on a client computer that has mounted MooseFS are exactly the same as they would be with other file systems. The operating system kernel transfers all file operations to the FUSE module, which communicates with the mfsmount process. The mfsmount process communicates through the network subsequently with the managing server and data servers (chunk servers). This entire process is fully transparent to the user.

mfsmount communicates with the managing server every time an operation on file metadata is required:

- creating files
- deleting files
- reading directories
- reading and changing attributes
- changing file sizes
- at the start of reading or writing data
- on any access to special files on MFSMETA

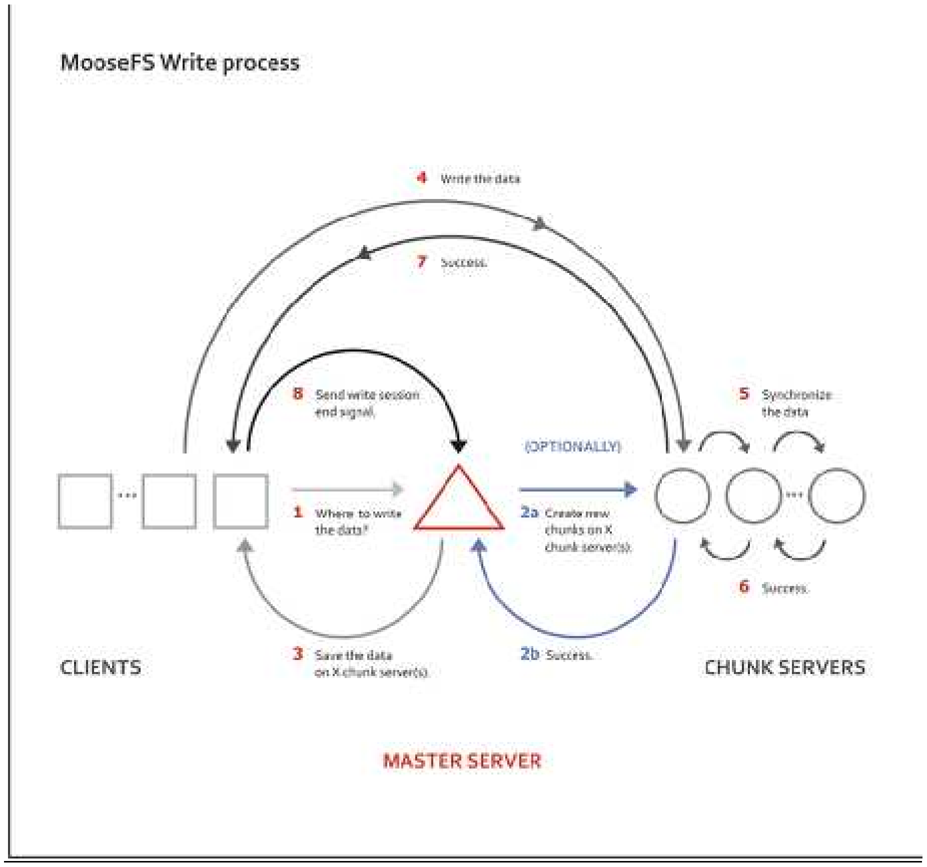

MooseFS Write process
4 Write the data
7 Success.
8 Send write session end signal.
5 Synchronize the data
(OPTIONALLY)
1 Where to write the data?
2a Create new chunks on X chunk server(s).
6 Success.
3 Save the data on X chunk server(s).
2b Success.
CLIENTS
CHUNK SERVERS
MASTER SERVER

Figure: 3.9.2 MooseFs Write Process

FAULT TOLERANCE:

Administrative commands allow the system administrator to specify the "goal", or number of copies that should be maintained, on a per-directory or per-file level. Setting the goal to more than one and having more than one data server will provide fault tolerance. When the file data is stored in many copies (on more than one data server), the system is resistant to failures or temporary network outages of a single data server. This of course does not refer to files with the "goal" set to 1, in which case the file will only exist on a single data server irrespective of how many data servers are deployed in the system. Exceptionally important files may have their goal set to a number higher than two, which will allow these files to be resistant to a breakdown of more than one server at once.

In general the setting for the number of copies available should be one more than the anticipated number of inaccessible or out-of-order servers.

In the case where a single data server experiences a failure or disconnection from the network, the files stored within it that had at least two copies, will remain accessible from another data server. The data that is now 'under its goal' will be replicated on another accessible data server to again provide the required number of copies.

Failures of a client machine (that runs the mfsmount process) will have no influence on the coherence of the file system or on the other client's operations. In the worst case scenario the data that has not yet been sent from the failed client computer may be lost.

- concurrent read operations supported:

All read operations are parallel - there is no problem with concurrent reading of the same data by several clients at the same moment.

- chunkservers and metadata server do their own checksumming:

there is checksumming done by the system itself. We thought it would be CPU consuming but it is not really. Overhead is about 4B per a 64KiB block which is 4KiB per a 64MiB chunk (per goal).

- MooseFS support file locking:

At the moment file locking works only "locally", where if a file is locked on a given client node (e.g. machine A) its kernel remembers it is locked for any other process on the same node. However another client node (e.g. machine B) doesn't know about it.

Step 3.10 Live migration Of Instance across multiple compute node

- To migrate running instances from one Compute server to another Compute server.
- WHY?

-when you need to upgrade or installing patches to hypervisors/BIOS and you need the machines to keep running; for example, when one of HDD volumes RAID or one of bonded NICs is out of order.

-for regular periodic maintenance, you may need to migrate VM instances. When many VM instances are running on a specific physical machine, you can redistribute the high load.

-Sometimes when VM instances are scattered, you can move VM instances to a physical machine to arrange them more logically

The live migration feature is useful when you need to upgrade or installing patches to hypervisors/BIOS and you need the machines to keep running; for example, when one of HDD volumes RAID or one of bonded NICs is out of order. Also for regular periodic maintenance, you may need to migrate VM instances. When many VM instances are running on a specific physical machine, you can redistribute the high load. Sometimes when VM instances are scattered, you can move VM instances to a physical machine to arrange them more logically.

Prerequisites

- **OS:**Ubuntu 12.04
- **Shared storage:** NOVA-INST-DIR/instances/ (eg /var/lib/nova/instances) has to be mounted by shared storage
- **Instances:** Instance can be migrated with ISCSI based volumes
- **Hypervisor:** KVM with libvirt

Example Nova Installation Environment

- Prepare 3 servers at least; for example, HostA, HostB and HostC
- HostA is the "Cloud Controller", and should be running: nova-api, nova-scheduler, nova-network, nova-volume, nova-objectstore, nova-scheduler.
- Host B and Host C are the "compute nodes", running nova-compute.
- Ensure that, NOVA-INST-DIR (set with state_path in nova.conf) is same on all hosts.
- In this example, HostA will be the NFSv4 server which exports NOVA-INST-DIR/instances, and HostB and HostC mount it.

System configuration

1. Configure your DNS or /etc/hosts and ensure it is consistent accross all hosts. Make sure that the three hosts can perform name resolution with each other. As a test, use the **ping** command to ping each host from one another.

```
$ ping Host
 $ ping HostB
$ ping HostC
```

2. configure the NFS server at HostA by adding a line to /etc/exports

```
$ NOVA-INST-DIR/instances HostA/255.255.0.0(rw,sync,fsid=0,no_root_squash)
```

   Change the subnet mask (255.255.0.0) to the appropriate value to include the IP addresses of HostB and HostC. Then restart the NFS server.

```
$ /etc/init.d/nfs-kernel-server restart
$ /etc/init.d/idmapd restart
```

3. Set the 'execute/search' bit on your shared directory

   On both compute nodes, make sure to enable the 'execute/search' bit to allow qemu to be able to use the images within the directories. On all hosts, execute the following command:

```
$ chmod o+x NOVA-INST-DIR/instances
```

4. Configure NFS at HostB and HostC by adding below to /etc/fstab

```
$ HostA:/NOVA-INST-DIR/instances /NOVA-INST-DIR/instances nfs4 defaults 0 0
```

   Then ensure that the exported directory can be mounted.

```
$ mount -a –v
```

Finally, use the **nova live-migration** command to migrate the instances.

```
# nova live-migration bee83dd3-5cc9-47bc-a1bd-6d11186692d0 HostC
Migration of bee83dd3-5cc9-47bc-a1bd-6d11186692d0 initiated.
```

Step 3.11 Billing Module

- How Amazon works:

Amazon EC2 is charging based on instance hours, regardless it's being utilized or not. Given that Amazon EC2 uses virtual hosts, the cost is mostly depend in the CPU time consumed. So actually if a instance is idle, it's odd that it's charged same as others consuming %100 of CPU/Memory resources. At the end of the day, isn't Cloud computing about paying exactly what it is being used.

- How actual usage based our system works:

Better way to charge for cloud resources like CPU, RAM, Disk, Network IO etc., so that it is true fine grain billing without overpaying for non-used resources

Billing of Cloud Resources Usage

- Analysis of top five billing system
- Define usage matrix: Ram(memory),Processor(cpu),Network(bandwidth),Storage and Images.
- Develop separate billing module who updated from each instances detail which running across compute node and measure cost according to usage matrix consumption.

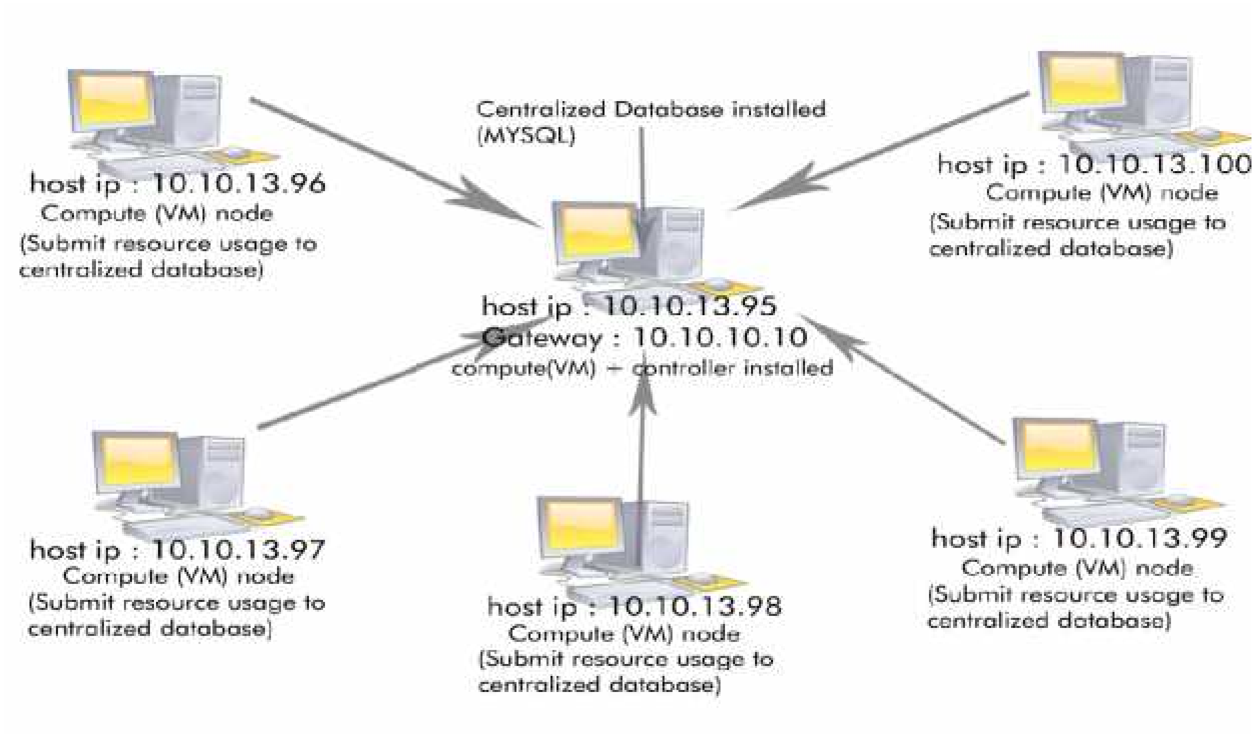

Centralized Database installed (MYSQL)
host ip : 10.10.13.96
Compute (VM) node
(Submit resource usage to centralized database)
host ip : 10.10.13.100
Compute (VM) node
(Submit resource usage to centralized database)
host ip : 10.10.13.95
Gateway : 10.10.10.10
compute(VM) + controller installed
host ip : 10.10.13.97
Compute (VM) node
(Submit resource usage to centralized database)
host ip : 10.10.13.98
Compute (VM) node
(Submit resource usage to centralized database)
host ip : 10.10.13.99
Compute (VM) node
(Submit resource usage to centralized database)

Figure:3.11.1 submit resource usage to centralize database

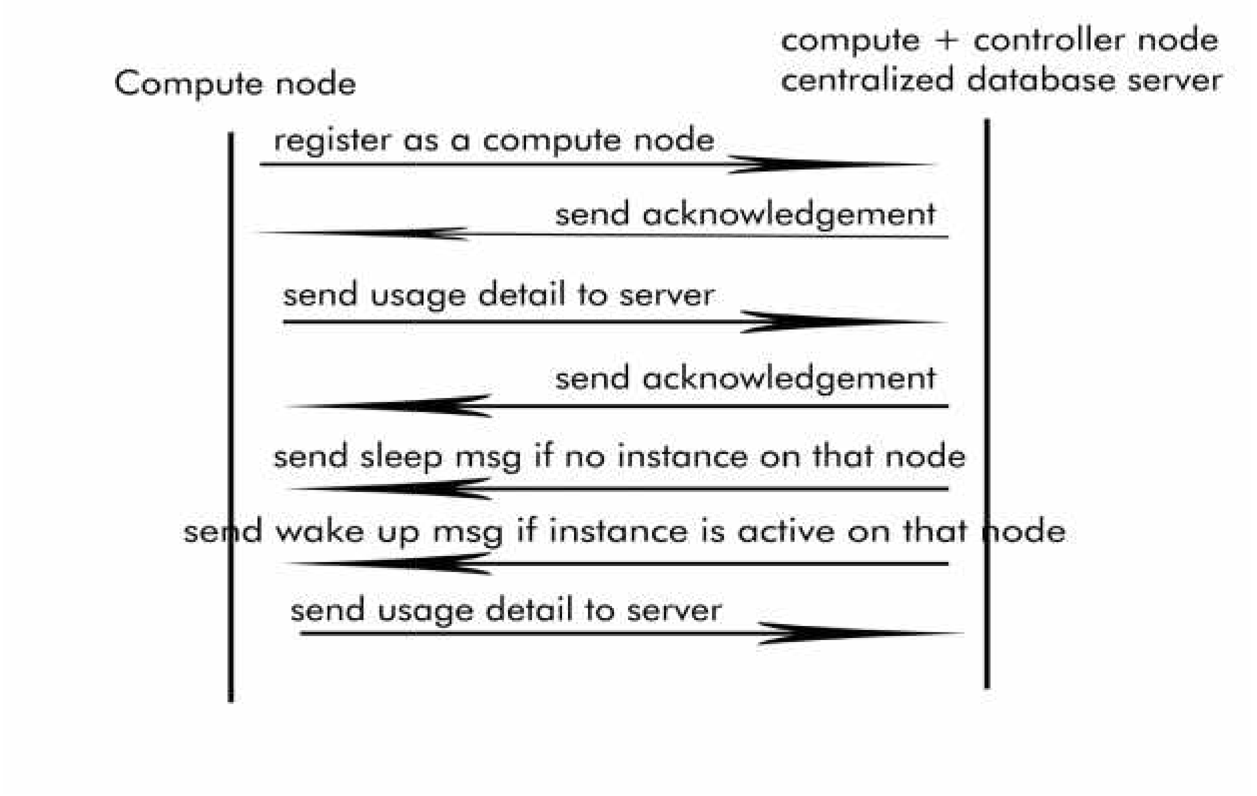

Compute node
compute + controller node
centralized database server
register as a compute node
send acknowledgement
send usage detail to server
send acknowledgement
send sleep msg if no instance on that node
send wake up msg if instance is active on that node
send usage detail to server

Figure 3.11.2 flow diagram for compute node and controller node

We had made our own GUI using django-python project for providing user and admin interface for an IaaS services. It is look like as below images. Have a glance.

4.1 Login page of Dashboard:

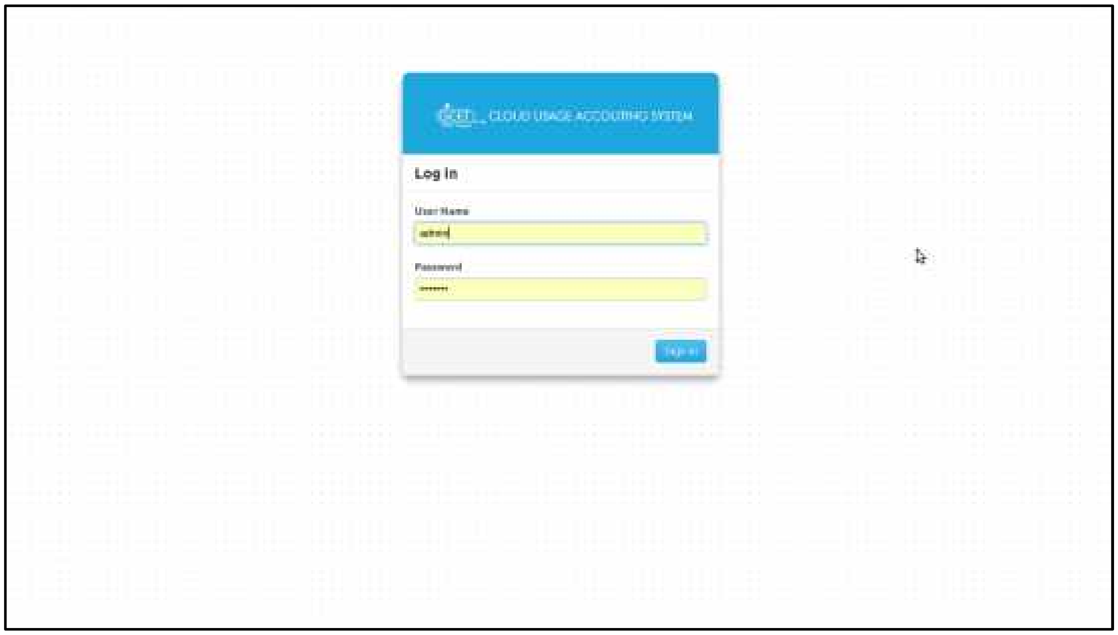
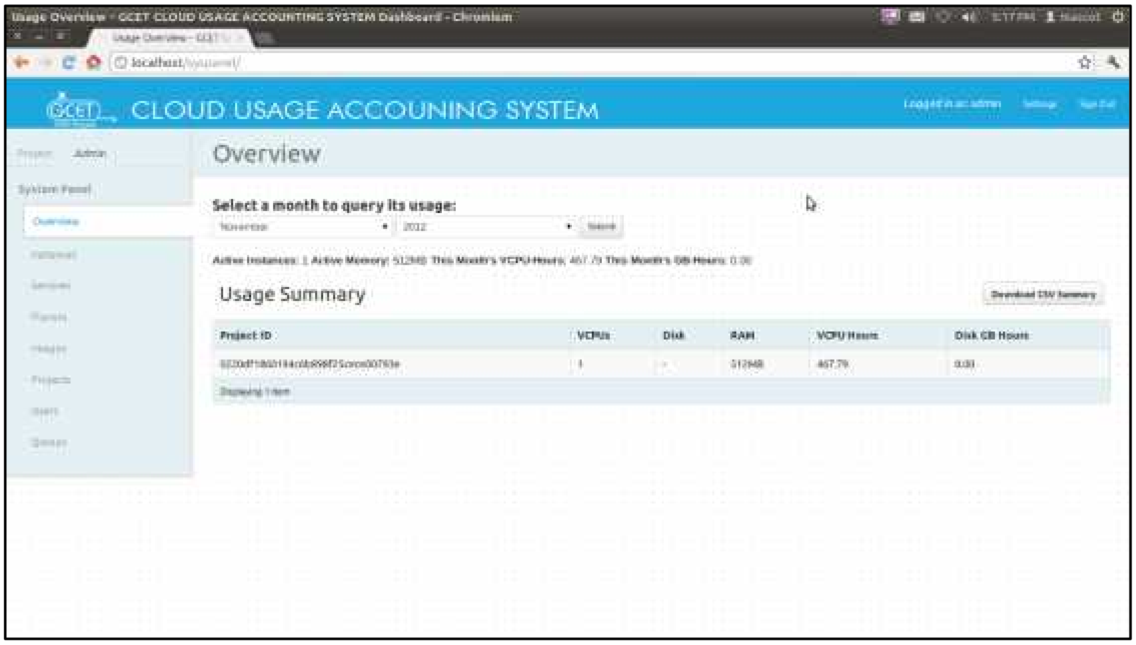


Figure: 4.1.1 Login page of GCET CLOUD

This is a login page for dashboard. We had given two credential for that one for admin one for user .User can select desire operating system as well as configure desire setting as its requirement. For example, image, flavor, memory, processing power etc.

4.2 Overview:

This page is available in only admin module. This page of Dashboard gives total overview of cloud. How many projects are running in cloud, And for that project it displays project id, how much virtual c.p.u is used by it, how much ram and disk is used and for how much time.


Usage Overview - GCET CLOUD USAGE ACCOUNTING SYSTEM Dashboard - Chromium
localhost/syspanel/
CLOUD USAGE ACCOUNING SYSTEM
Project
Admin
System Panel
Overview
Download CSV Summary

Overview

Select a month to query its usage:

November 2012 Submit

Active Instances: 1 Active Memory: 512MB This Month's VCPU-Hours: 467.79 This Month's GB-Hours: 0.00

Usage Summary

Project ID	VCPUs	Disk	RAM	VCPU Hours	Disk GB Hours
[illegible]	1	-	512MB	467.79	0.00

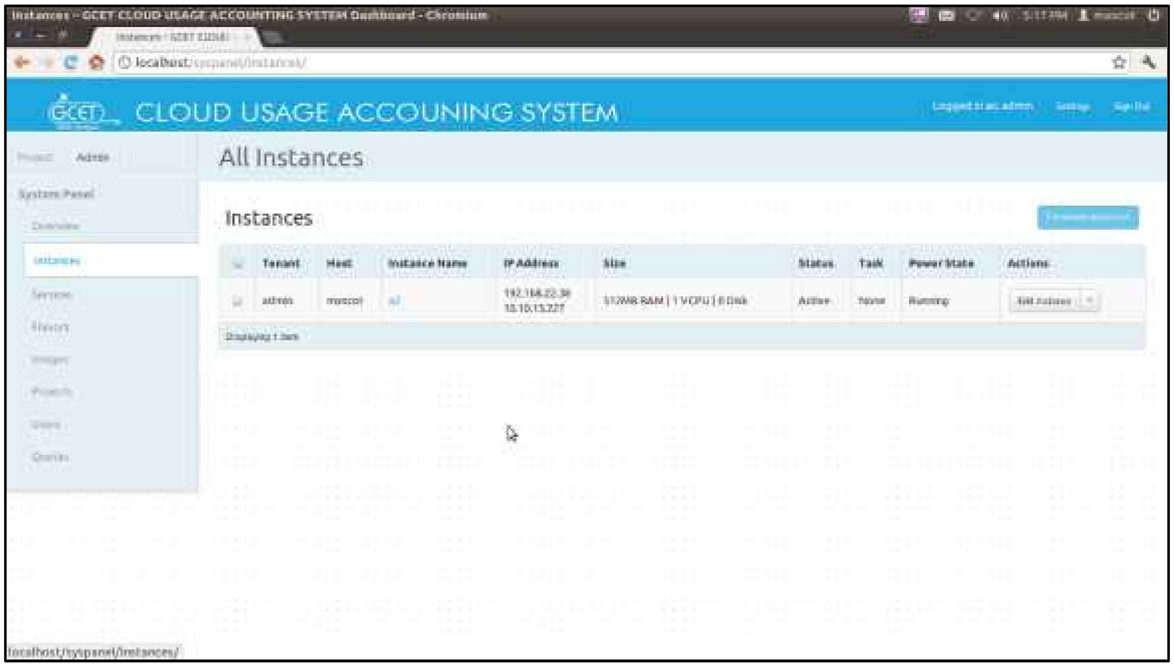
Displaying 1 item

Figure: 4.2.1 overview page of Dashboard

4.3 Instances module of Dashboard:

These pages are also available in only admin module. And which shows running instances in cloud. And display details of instances like tenant name, host name, instance name, ip address (both if available) size of instance, status, task, power state.

And one function for which can edit running instance.

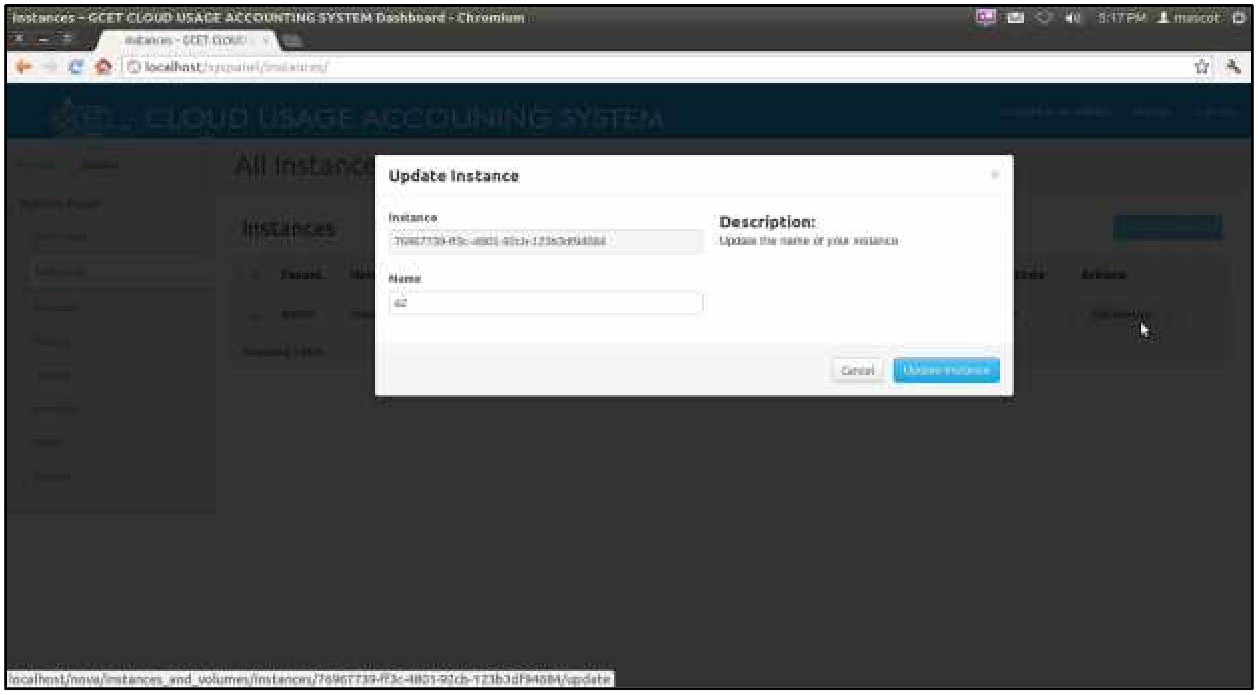


Instances - GCET CLOUD USAGE ACCOUNTING SYSTEM Dashboard - Chromium
localhost/syspanel/instances/
CLOUD USAGE ACCOUNING SYSTEM
All Instances
Instances
Project
Admin
System Panel
Overview
Instances
Services
Flavors
Images
Projects
Users
Quotas

	Tenant	Host	Instance Name	IP Address	Size	Status	Task	Power State	Actions
	admin	mascot	a1	192.168.22.30 10.10.15.227	512MB RAM | 1 VCPU | 0 Disk	Active	None	Running	Edit Instance

Displaying 1 item

Figure: 4.3.1 Instance module of Dashboard



Instances - GCET CLOUD USAGE ACCOUNTING SYSTEM Dashboard - Chromium
localhost/syspanel/instances/
CLOUD USAGE ACCOUNING SYSTEM
All Instances
Instances
Update Instance
Cancel
Update Instance

Instance: 76967739-ff3c-4801-92cb-123b3df94084

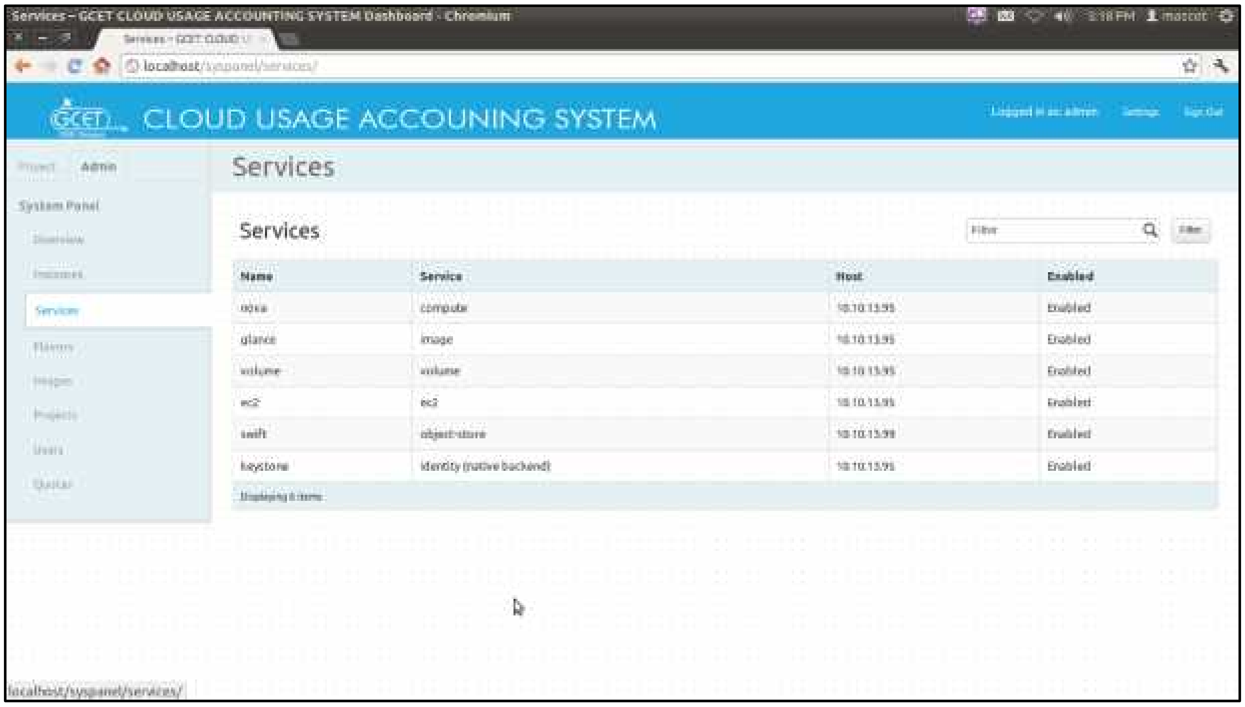
Name: a2

Description:
Update the name of your instance

localhost/nova/instances_and_volumes/instances/76967739-ff3c-4801-92cb-123b3df94084/update

Figure 4.3.2 Update instance module of Dashboard

4.4 Services module:


Services – GCET CLOUD USAGE ACCOUNTING SYSTEM Dashboard - Chromium
localhost/syspanel/services/
CLOUD USAGE ACCOUNING SYSTEM
Project
Admin
System Panel
Overview
Instances
Services
Flavors
Images
Projects
Users
Quotas
Services
Filter

Services

Name	Service	Host	Enabled
nova	compute	10.10.13.95	Enabled
glance	image	10.10.13.95	Enabled
volume	volume	10.10.13.95	Enabled
ec2	ec2	10.10.13.95	Enabled
swift	object-store	10.10.13.99	Enabled
keystone	identity (native backend)	10.10.13.95	Enabled

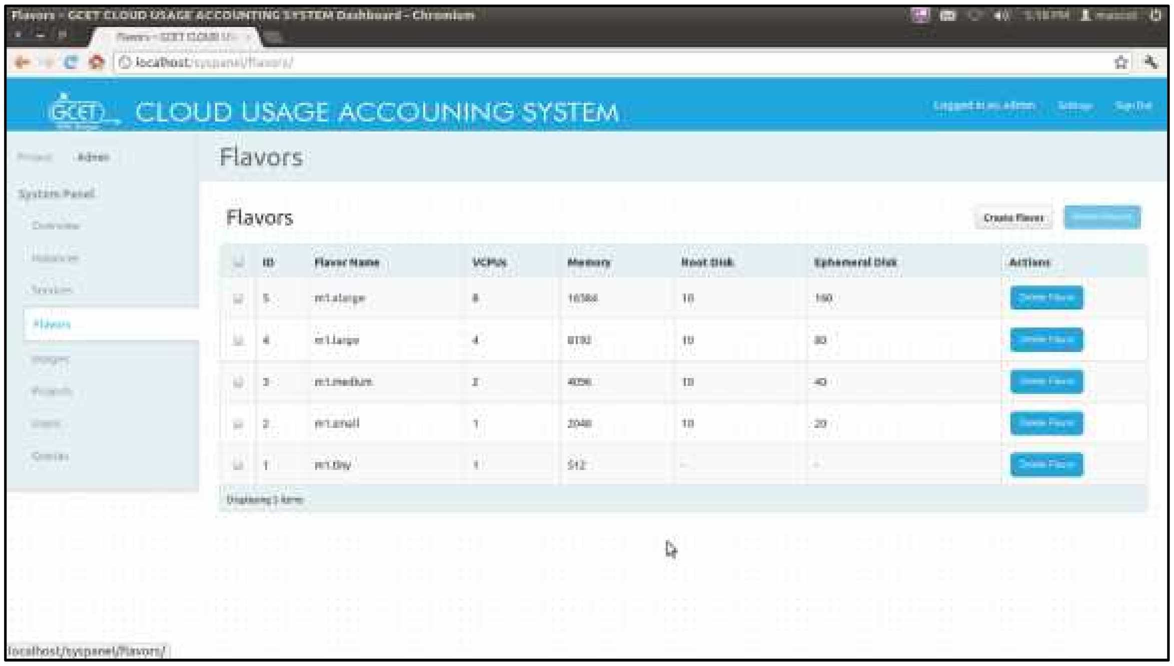
Displaying 6 items

Figure: 4.4.1 Services module of Dashboard

This page is also available in only admin module. This shows running services of cloud. And displays name of service and its service and shows that running on which host. And shows state of that service that it enabled or disabled.

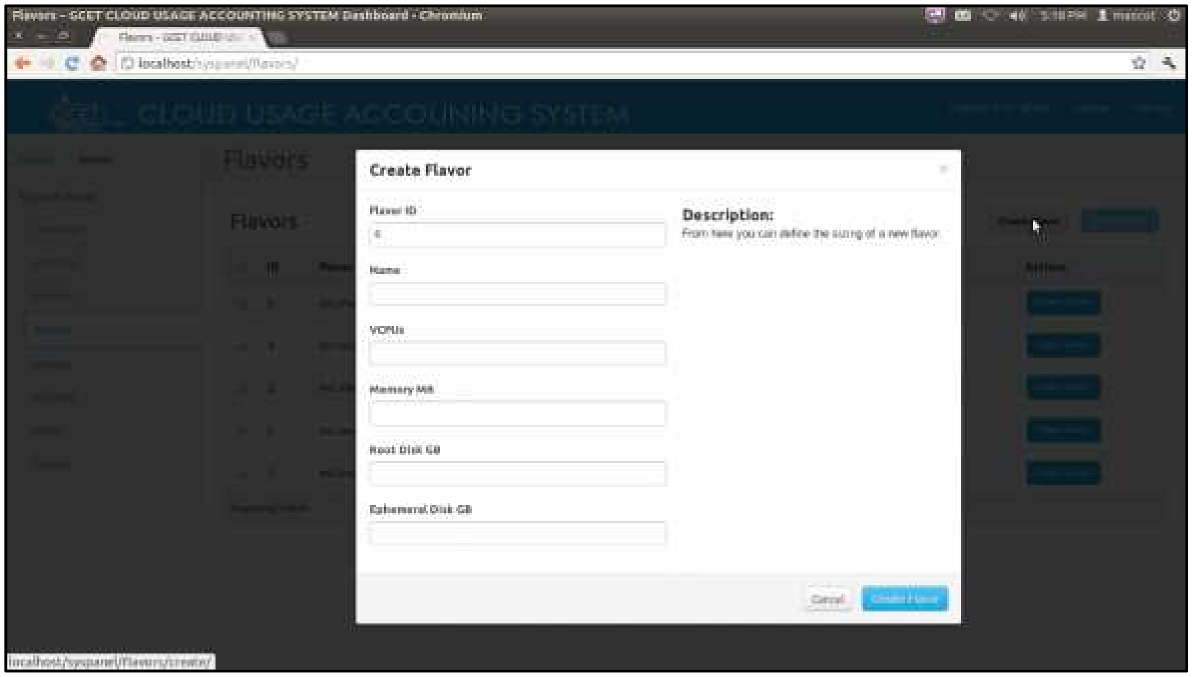
4.6 Flavor module:

This page is also available in only admin module. This page shows name of flavours with it id number and specification of that flavours. In specification it shows that for specific flavour how much virtual CPU required, how much memory required and root disk and ephemeral disk.



ID	Flavor Name	VCPUs	Memory	Root Disk	Ephemeral Disk
5	m1.xlarge	8	16384	10	160
4	m1.large	4	8192	10	80
3	m1.medium	2	4096	10	40
2	m1.small	1	2048	10	20
1	m1.tiny	1	512	-	-

Figure: 4.5.1 Flavors module of Dashboard



Flavor ID: 6

Name:

VCPUs:

Memory MB:

Root Disk GB:

Ephemeral Disk GB:

Description:
From here you can define the sizing of a new flavor.

Figure: 4.5.2 create flavour page of Dashboard

4.6 Images module of cloud:

This page is also available in only admin module. This page shows existing images or OS on cloud. And also shows status of that image and format of that image.Like, we uploaded three images on our cloud and those Ubuntu desktop version, Ubuntu server version and cirros.

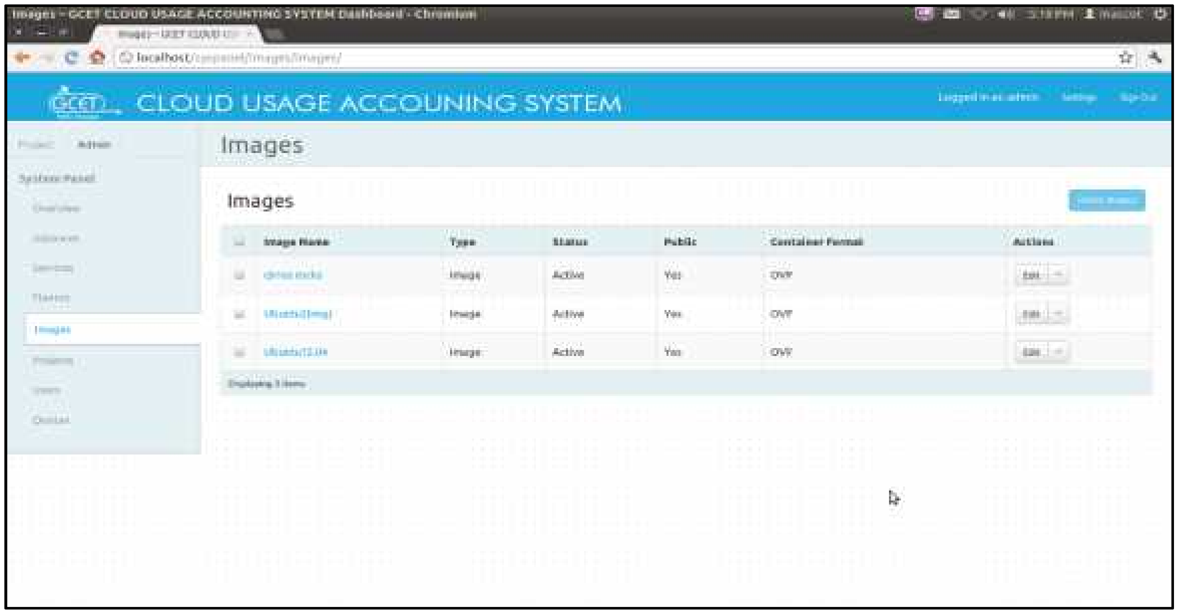
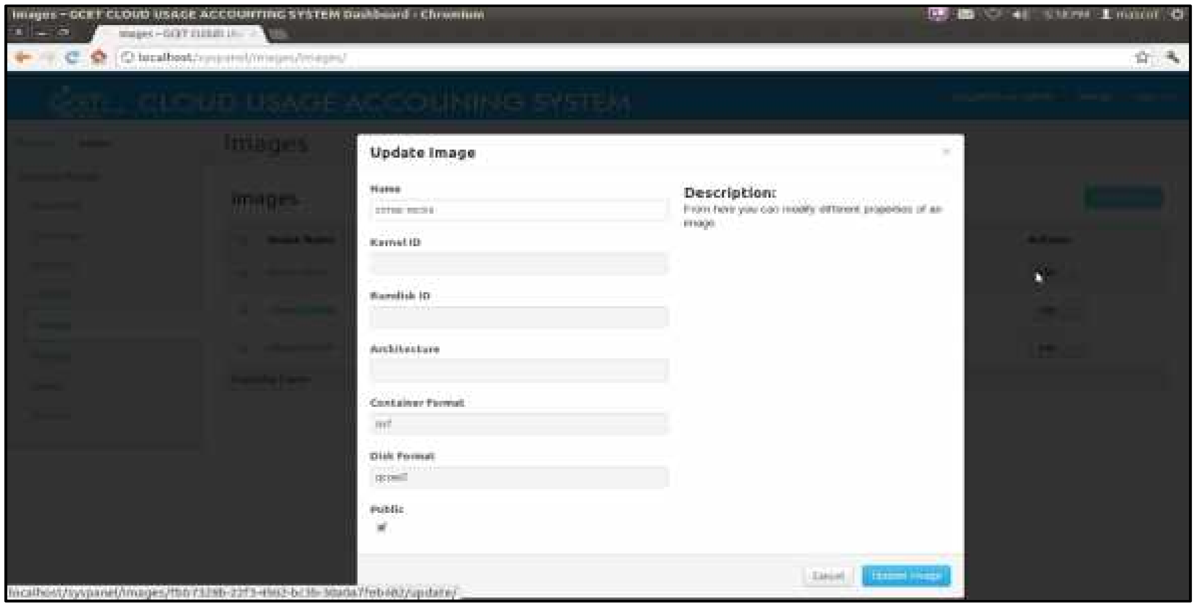

Images – GCET CLOUD USAGE ACCOUNTING SYSTEM Dashboard - Chromium
localhost/syspanel/images/images/
CLOUD USAGE ACCOUNING SYSTEM
Images
Images

	Image Name	Type	Status	Public	Container Format	Actions
	[illegible]	Image	Active	Yes	OVF	Edit
	[illegible]	Image	Active	Yes	OVF	Edit
	[illegible]	Image	Active	Yes	OVF	Edit

Figure: 4.6.1 Image Page of Dashboard

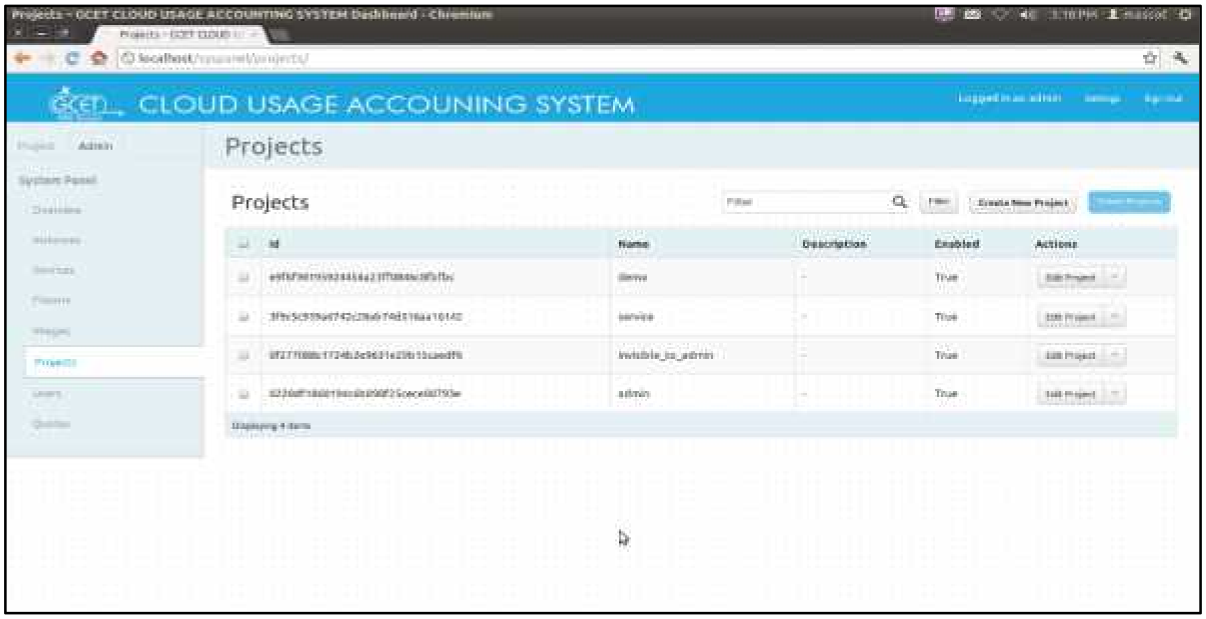

Images – GCET CLOUD USAGE ACCOUNTING SYSTEM Dashboard - Chromium
localhost/syspanel/images/images/
CLOUD USAGE ACCOUNING SYSTEM
Update Image
Cancel
Update Image

Name: [illegible]
Kernel ID:
Ramdisk ID:
Architecture:
Container Format: ovf
Disk Format: qcow2
Public: ✓

Description:
From here you can modify different properties of an image.

Figure: 4.6.2 Update Image module of Dashboard

4.7 Project module of Dashboard:



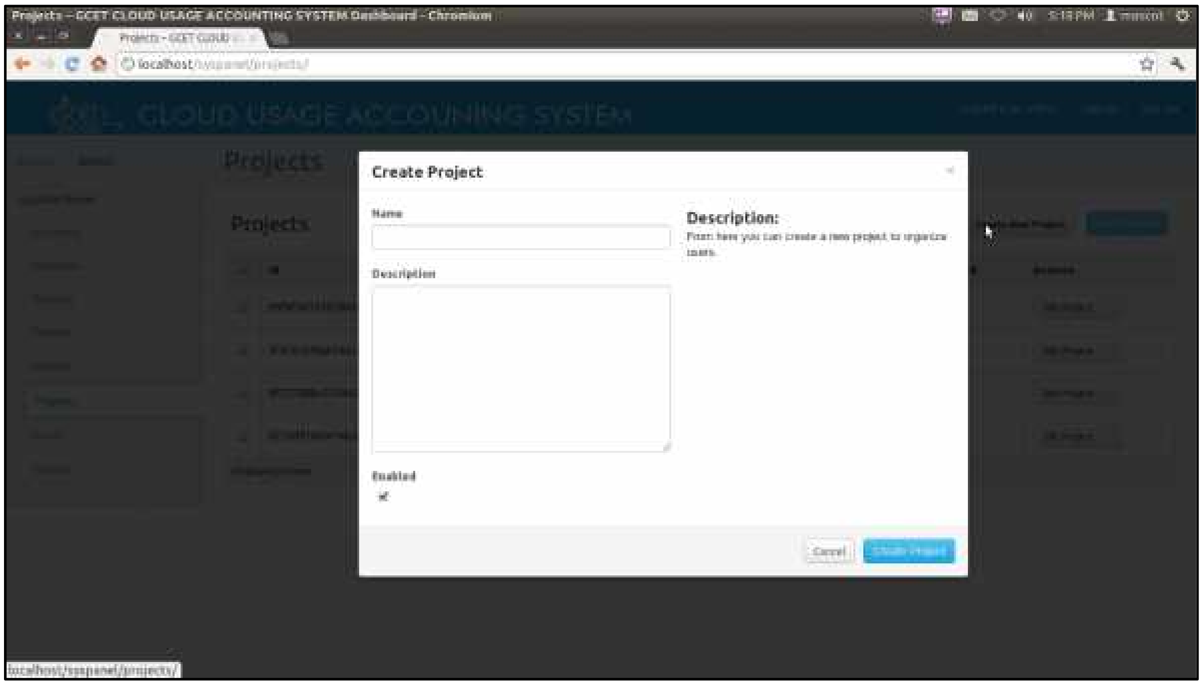
CLOUD USAGE ACCOUNING SYSTEM

Projects

Id	Name	Description	Enabled	Actions
[illegible]	demo	-	True	Edit Project
[illegible]	service	-	True	Edit Project
[illegible]	invisible_to_admin	-	True	Edit Project
[illegible]	admin	-	True	Edit Project

Displaying 4 items

Figure: 4.7.1 Project module of Dashboard



Create Project

Name

Description

Enabled ✓

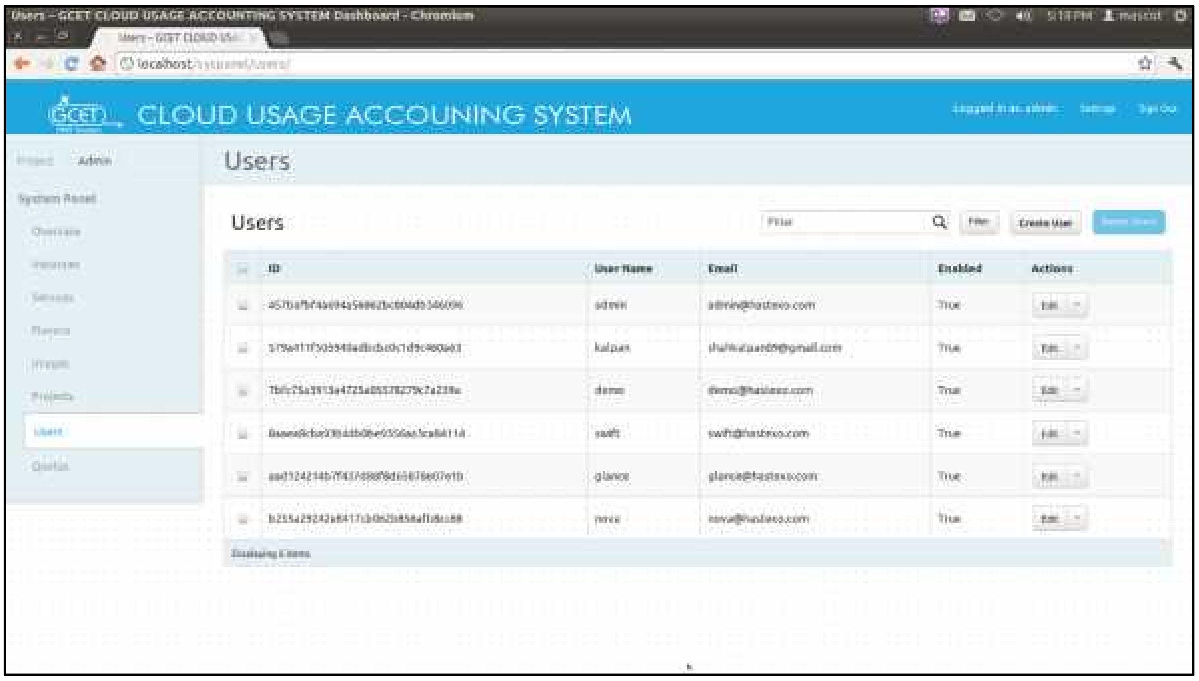
Description:
From here you can create a new project to organize users.

Cancel | Create Project

Figure: 4.7.2 Project module of Dashboard

This page is also available in only admin module. This page shows name of project with it project id and status. Admin can edit project details like can change name of project can add description and can enable and disable.

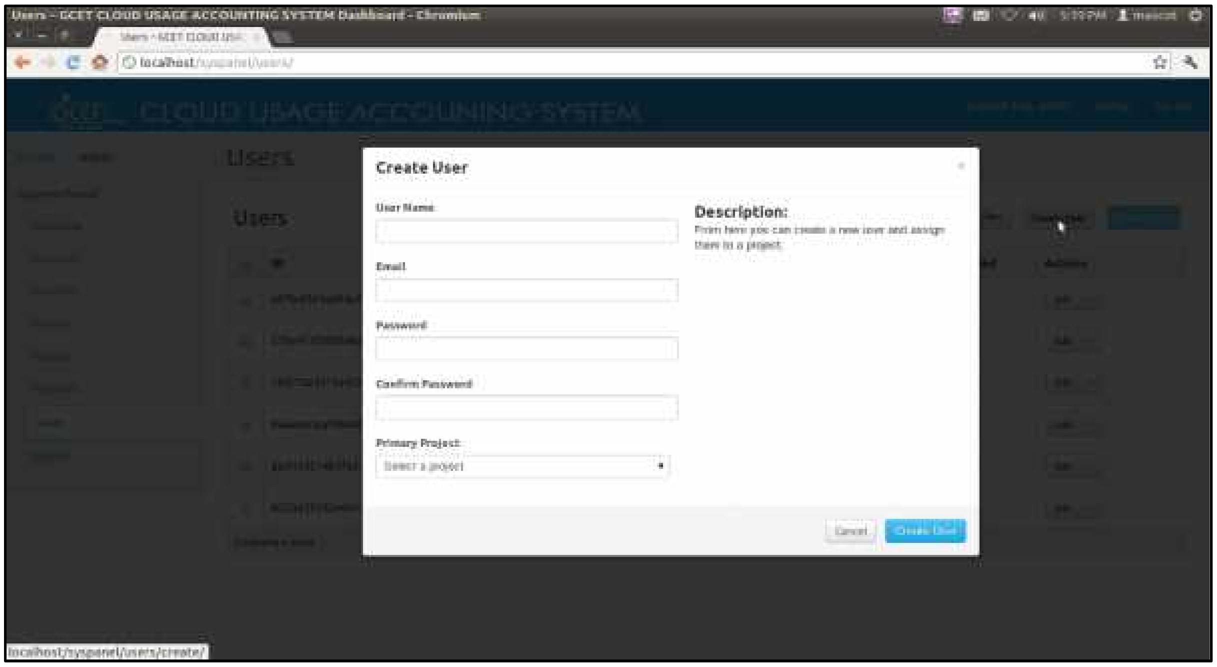
4.8 Users module:



	ID	User Name	Email	Enabled	Actions
	[illegible]	admin	admin@hastexo.com	True	Edit
	[illegible]	kalpan	[illegible]@gmail.com	True	Edit
	[illegible]	demo	demo@hastexo.com	True	Edit
	[illegible]	swift	swift@hastexo.com	True	Edit
	[illegible]	glance	glance@hastexo.com	True	Edit
	[illegible]	nova	nova@hastexo.com	True	Edit

Figure: 4.8.1 User module of Dashboard

This page is also available in only admin module. This page shows user of cloud with their user id, user name and email address. Admin can edit details of users and can also add new users and remove user from cloud.



Create User

User Name:

Email:

Password:

Confirm Password:

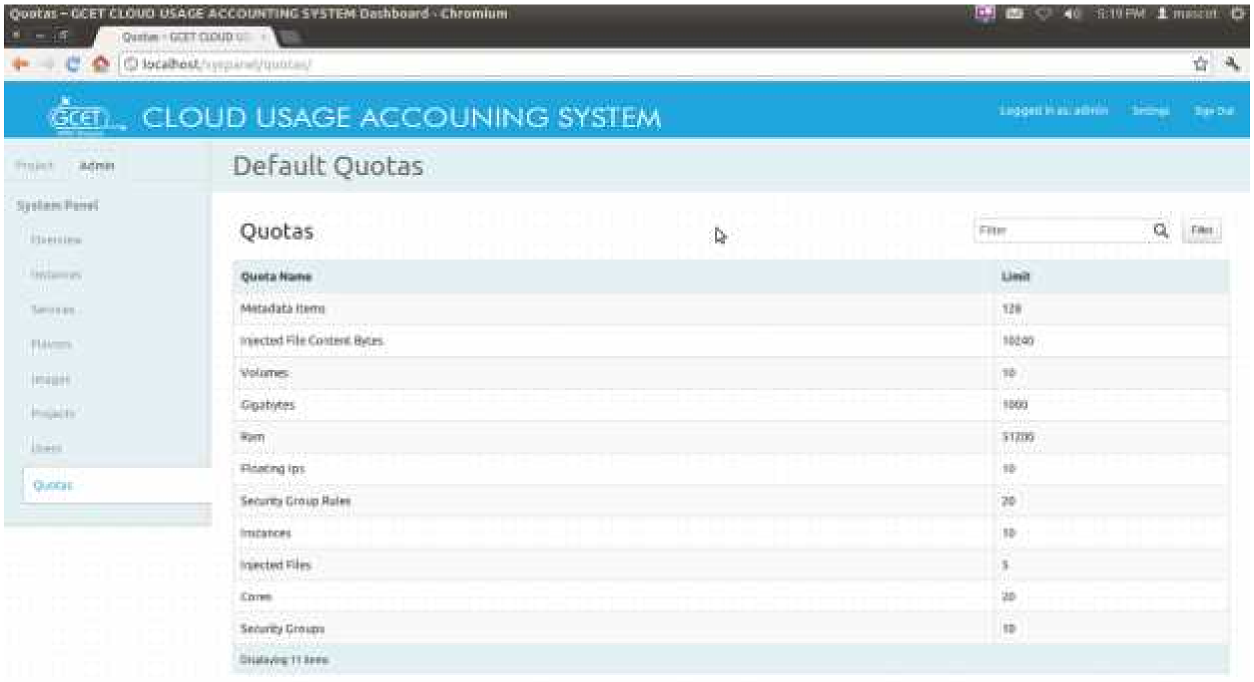
Primary Project: Select a project

Description:
From here you can create a new user and assign them to a project.

Users - GCET CLOUD USAGE ACCOUNTING SYSTEM Dashboard - Chromium
localhost/syspanel/users/
CLOUD USAGE ACCOUNING SYSTEM
Users
Cancel
Create User
localhost/syspanel/users/create/

Figure: 4.8.2 create user module of Dashboard

4.9 Quotas of Cloud:



Default Quotas

Quotas

Quota Name	Limit
Metadata Items	128
Injected File Content Bytes	10240
Volumes	10
Gigabytes	1000
Ram	51200
Floating Ips	10
Security Group Rules	20
Instances	10
Injected Files	5
Cores	20
Security Groups	10

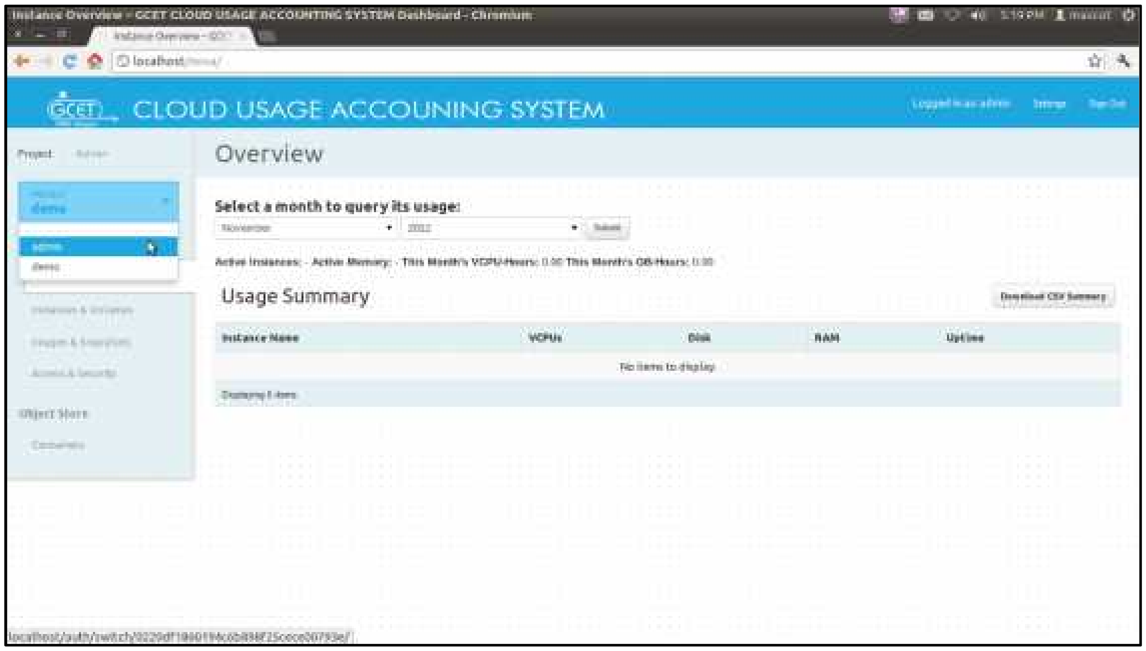
Displaying 11 items

Figure: 4.9.1 create user module of Dashboard

This page is also available in only admin module. This page shows default quotas of cloud like metadata items, volumes, instances, floating ips, security groups, RAM memory etc.

4.10 Project module of Dashboard:

This page is available in admin module and User module. This page is for select project from existing project created by Admin.


Instance Overview - GCET CLOUD USAGE ACCOUNTING SYSTEM Dashboard - Chromium

CLOUD USAGE ACCOUNING SYSTEM

Overview

Select a month to query its usage:

Active Instances: - Active Memory: - This Month's VCPU-Hours: 0.00 This Month's GB-Hours: 0.00

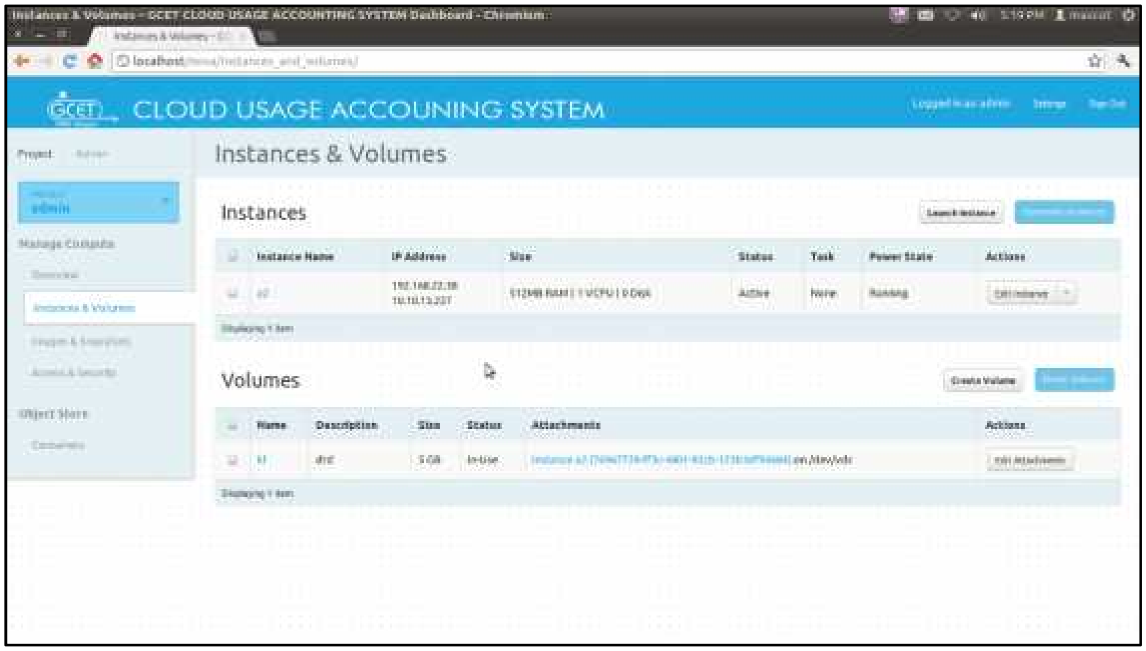
Usage Summary

Instance Name	VCPUs	Disk	RAM	Uptime
No items to display				

Figure: 4.10.1 Project selection module of Dashboard

4.11 Instances & Volumes module of Dashboard:

This page is available in admin module and User module. This page shows available instances in current project with its specification. And user can crate volume and attach that volume with running instances. And user can also launch instance.


Instances & Volumes - GCET CLOUD USAGE ACCOUNTING SYSTEM Dashboard - Chromium

CLOUD USAGE ACCOUNING SYSTEM

Instances & Volumes

Instances

	Instance Name	IP Address	Size	Status	Task	Power State	Actions
	a2	192.168.22.18 10.10.13.227	512MB RAM | 1 VCPU | 0 Disk	Active	None	Running	Edit Instance

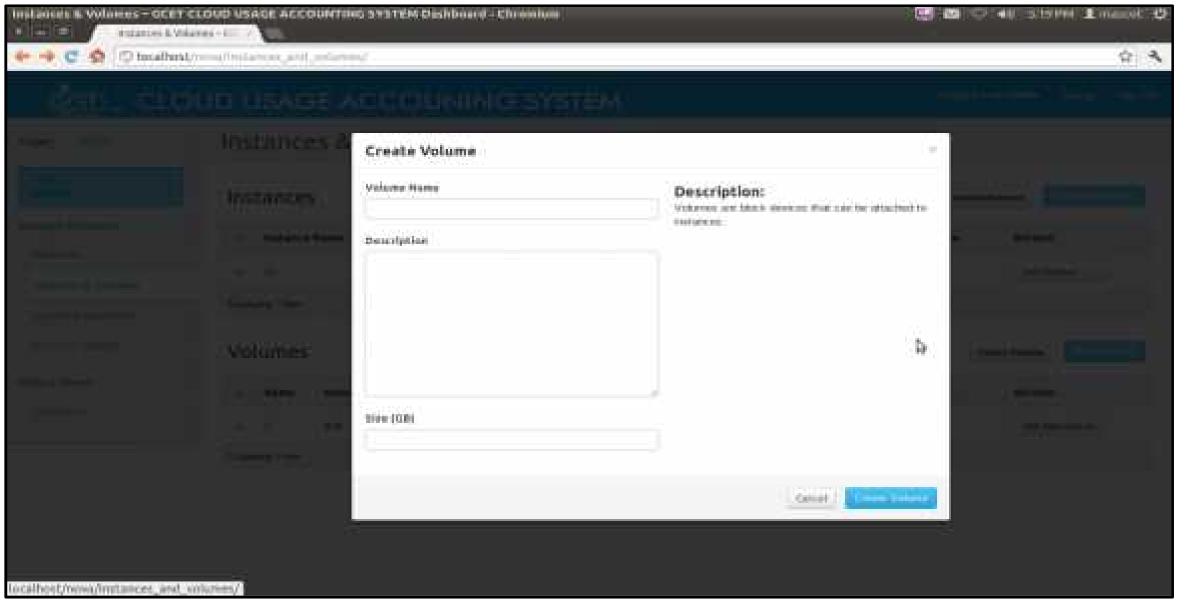
Displaying 1 item

Volumes

	Name	Description	Size	Status	Attachments	Actions
	t1	dnd	5 GB	In-Use	Instance a2 [illegible] on /dev/vdc	Edit Attachments

Displaying 1 item

Figure: 4.11.1 Instance & Volumes module of dashboard


Create Volume

Volume Name

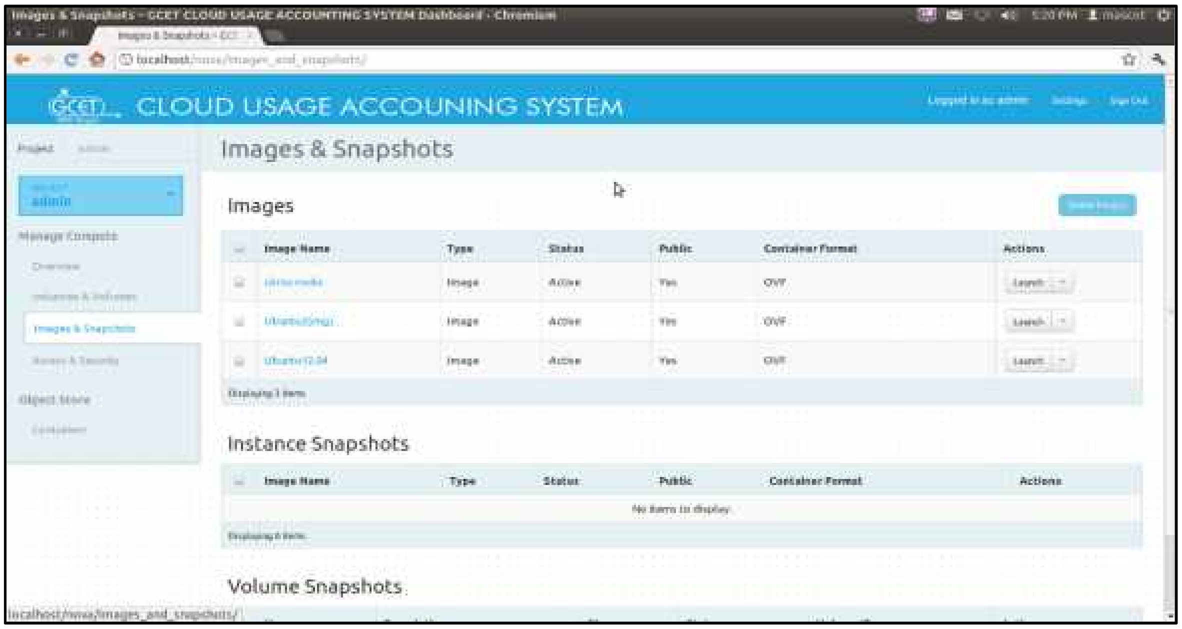
Description

Size (GB)

Description:
Volumes are block devices that can be attached to instances.

Figure: 4.11.2 Create volume module of dashboard

4.12 Images & Snapshots Module of Dashboard:


Images & Snapshots - GCET CLOUD USAGE ACCOUNTING SYSTEM Dashboard - Chromium

localhost/nova/images_and_snapshots/

CLOUD USAGE ACCOUNING SYSTEM

Images & Snapshots

Images

	Image Name	Type	Status	Public	Container Format	Actions
	[illegible]	Image	Active	Yes	OVF	Launch
	[illegible]	Image	Active	Yes	OVF	Launch
	Ubuntu12.04	Image	Active	Yes	OVF	Launch

Displaying 3 items

Instance Snapshots

	Image Name	Type	Status	Public	Container Format	Actions
No items to display						

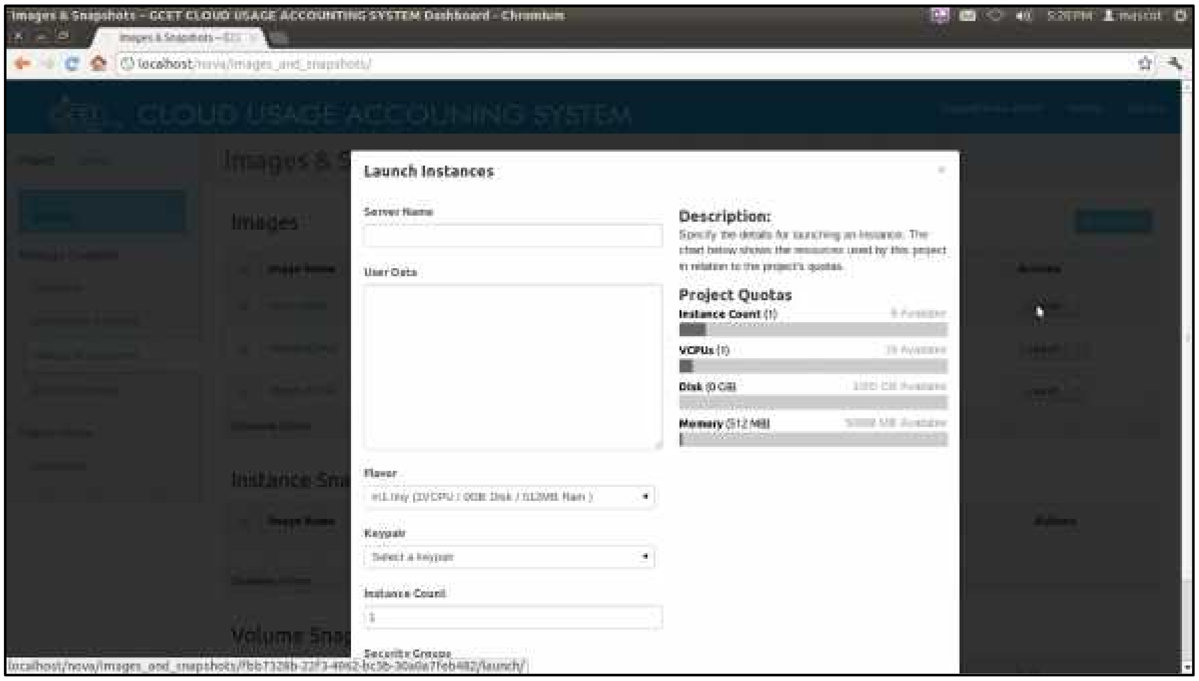
Displaying 0 items

Volume Snapshots

localhost/nova/images_and_snapshots/

Figure: 4.12.1 Images Launch module of Dashboard

This page is also available in Admin module and User module.


Images & Snapshots - GCET CLOUD USAGE ACCOUNTING SYSTEM Dashboard - Chromium

localhost/nova/images_and_snapshots/

Launch Instances

Server Name

User Data

Flavor: m1.tiny (1VCPU / 0GB Disk / 512MB Ram)

Keypair: Select a keypair

Instance Count: 1

Security Groups

Description:
Specify the details for launching an instance. The chart below shows the resources used by this project in relation to the project's quotas.

Project Quotas

Instance Count (1)

VCPUs (1)

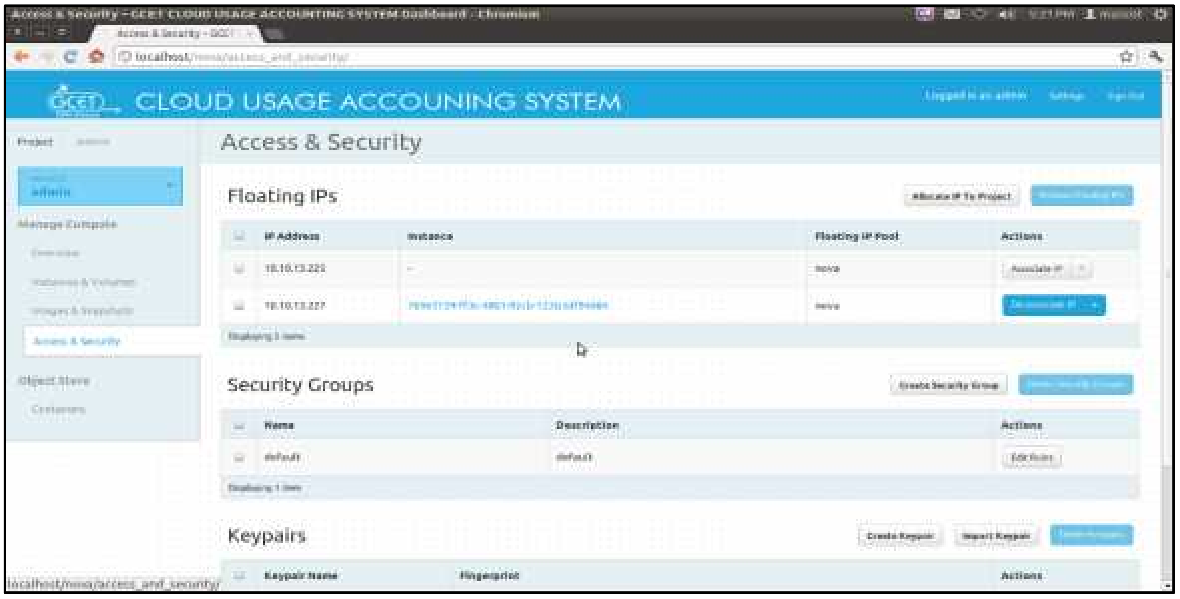
Disk (0 GB)

Memory (512 MB)

localhost/nova/images_and_snapshots/fbb7328b-22f3-4962-bc3b-30a0a7feb482/launch/

Figure: 4.12.2 Launch instance module of Dashboard

4.13. Access & Security module of Dashboard:



Floating IPs

	IP Address	Instance	Floating IP Pool	Actions
	10.10.13.225	-	nova	Associate IP
	10.10.13.227	[illegible]	nova	[illegible]

Security Groups

	Name	Description	Actions
	default	default	Edit Rules

Keypairs

	Keypair Name	Fingerprint	Actions

Figure: 4.13.1 Access & security module of Dashboard

This page is also available in Admin module and User module. This page shows that which keypair is associated with which image. And shows security group of that image.

4.14 Image Details:

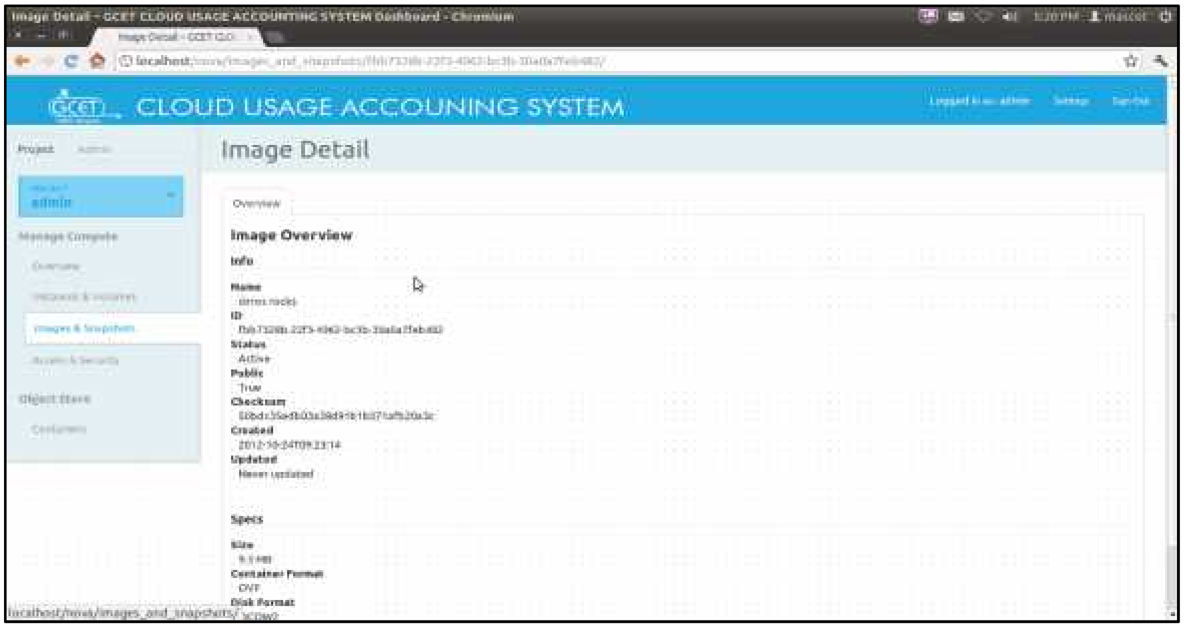


Image Overview

Info

Name
[illegible]
ID
[illegible]
Status
Active
Public
True
Checksum
[illegible]
Created
2012-10-24T09:23:14
Updated
Never updated

Specs

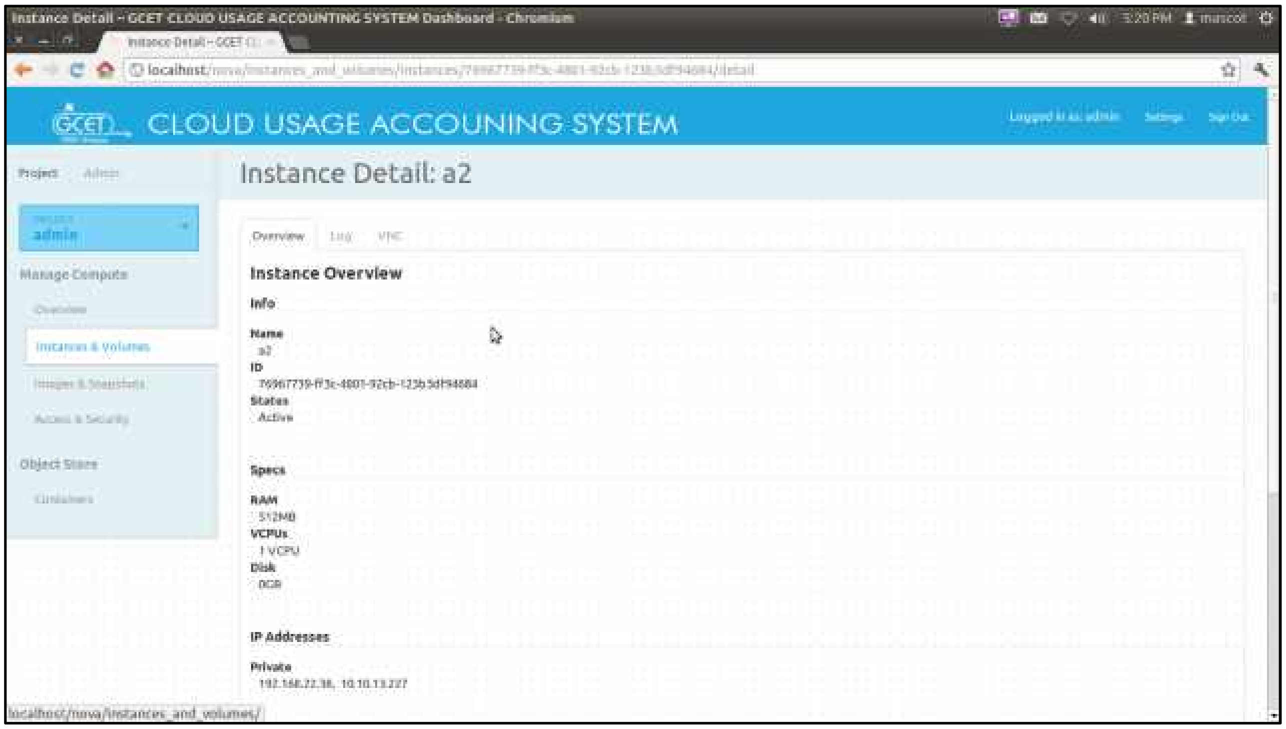
Size
[illegible]
Container Format
OVF
Disk Format
[illegible]

Figure: 3.14.1 Image Details module of Dashboard

This page is also available in Admin module and User module. This page shows full details of image with it name, id, and status, size and disk format. And also shows host id.

4.15 Instance Details:

This page is also available in admin module and User module. This page gives overview of instance, log of instance and VNC. In overview detail of instance is shown. In log it display log generated by Cloud server. And in VNC user can run instance.



Instance Detail: a2

Instance Overview

Info

Name
a2
ID
76967739-ff3c-4801-92cb-123b5df94684
Status
Active

Specs

RAM
512MB
VCPUs
1 VCPU
Disk
0GB

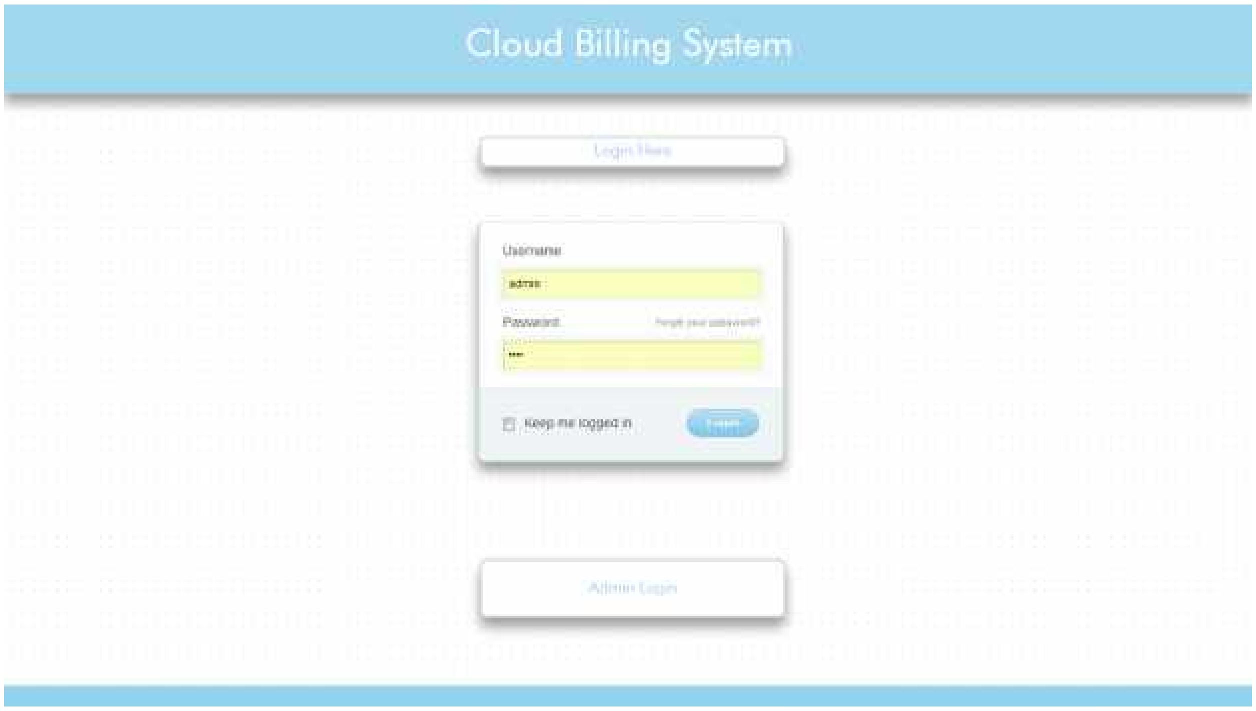
IP Addresses

Private
192.168.22.38, 10.10.13.227

localhost/nova/instances_and_volumes/

Figure: 3.15.1 Instance Details module of Dashboard

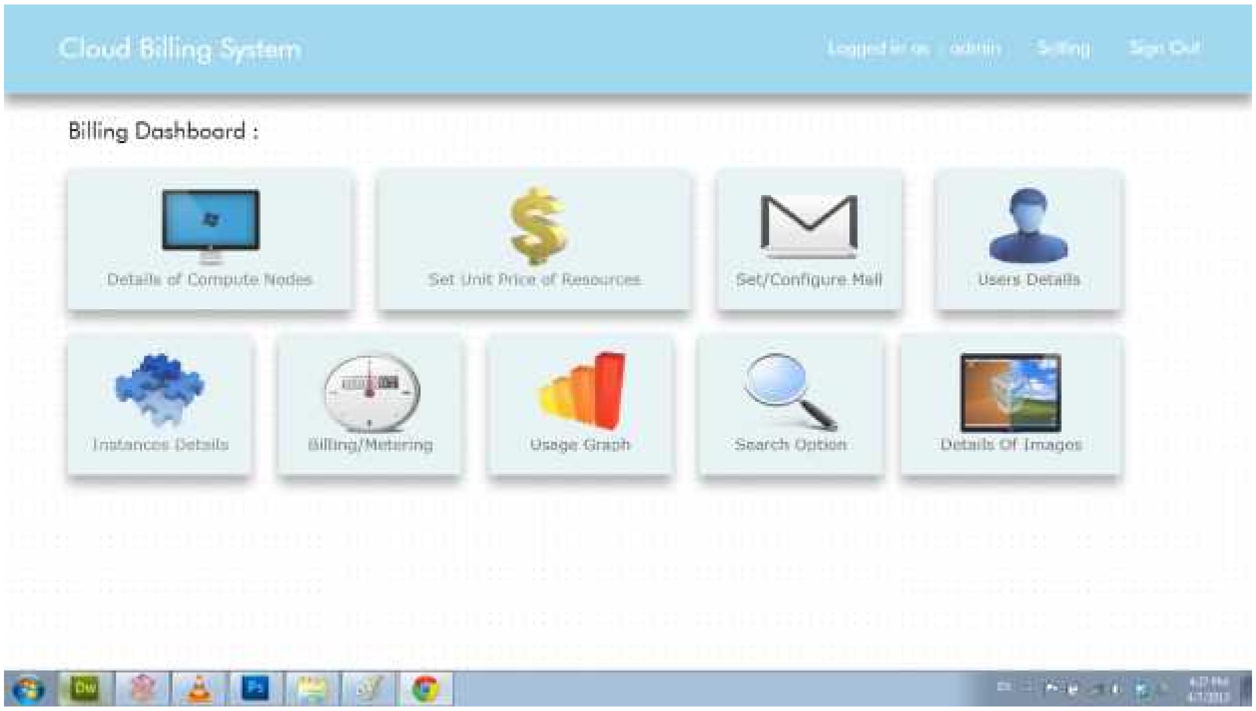
4.16. Billing system login page:



Username
admin
Password
forgot your password?

Figure: 4.16.1 Login page

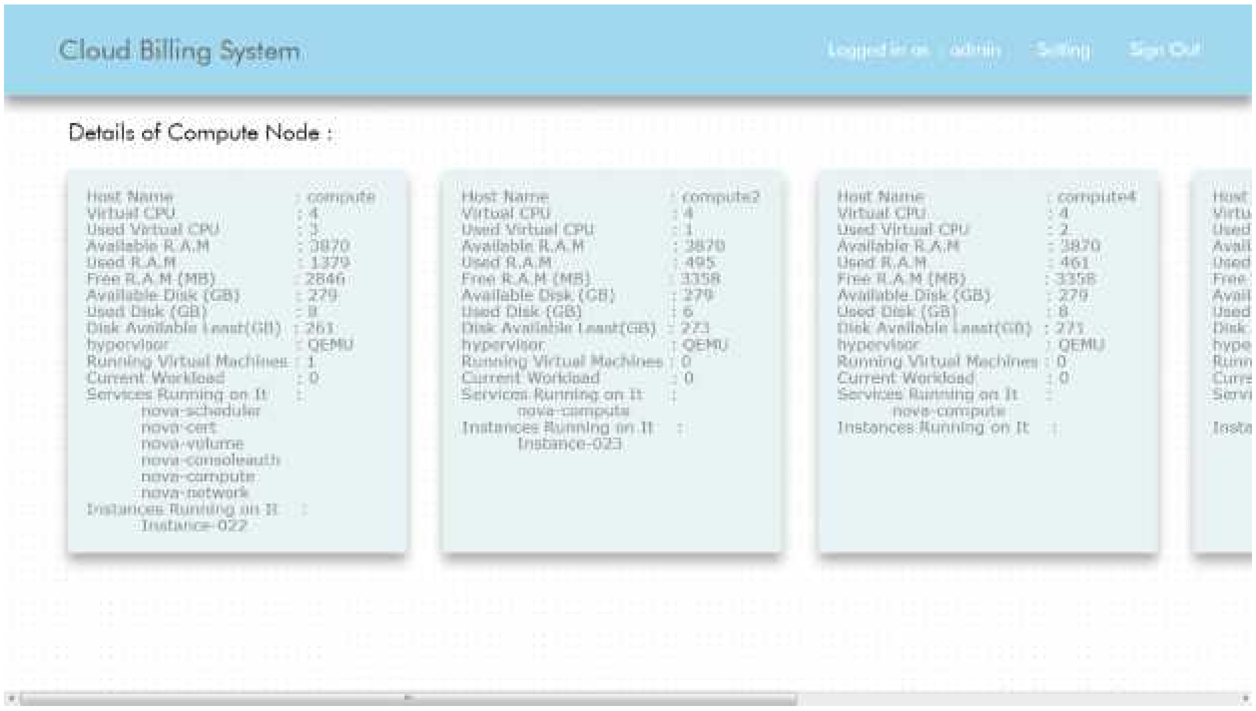
4.17 Billing system home page:



Billing Dashboard :

Figure: 4.17.1 Home page

4.18 Billing system compute node details page:



Details of Compute Node :

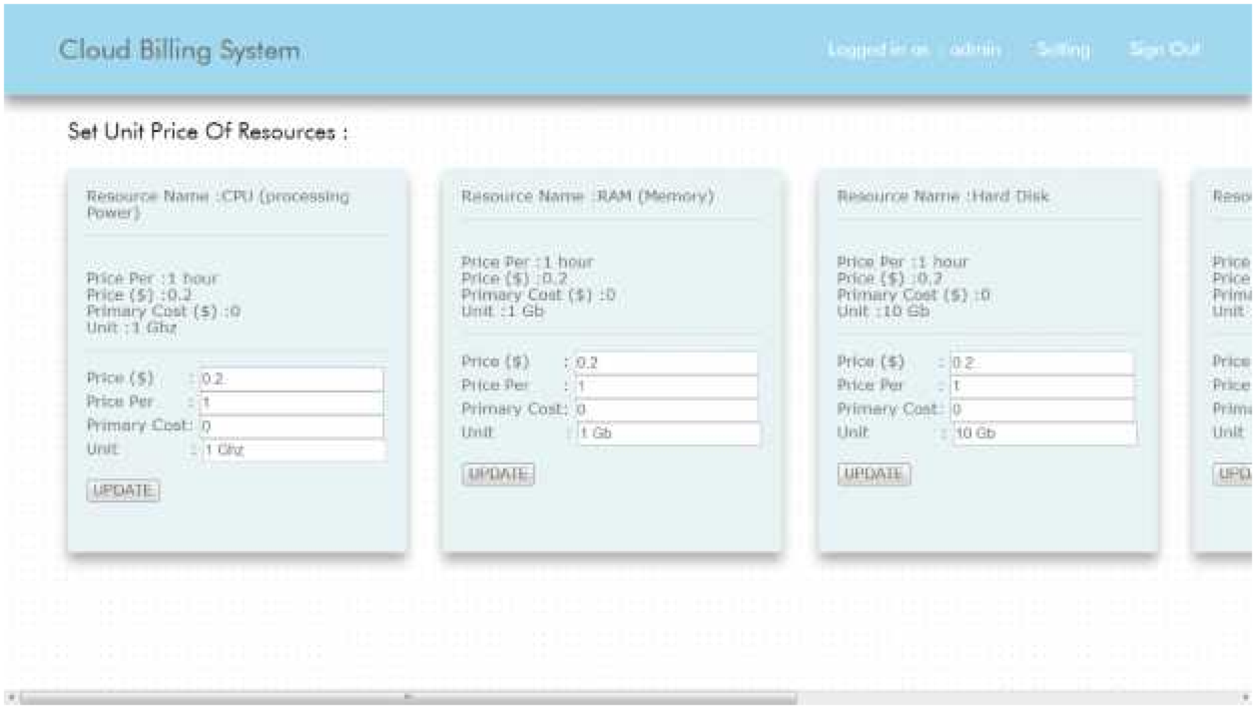
```
Host Name                   : compute
Virtual CPU                 : 4
Used Virtual CPU            : 3
Available R.A.M             : 3870
Used R.A.M                  : 1379
Free R.A.M (MB)             : 2846
Available Disk (GB)         : 279
Used Disk (GB)              : 8
Disk Available Least(GB)    : 261
hypervisor                  : QEMU
Running Virtual Machines    : 1
Current Workload            : 0
Services Running on It      :
        nova-scheduler
        nova-cert
        nova-volume
        nova-consoleauth
        nova-compute
        nova-network
Instances Running on It     :
        Instance-022
```

```
Host Name                   : compute2
Virtual CPU                 : 4
Used Virtual CPU            : 1
Available R.A.M             : 3870
Used R.A.M                  : 495
Free R.A.M (MB)             : 3358
Available Disk (GB)         : 279
Used Disk (GB)              : 6
Disk Available Least(GB)    : 273
hypervisor                  : QEMU
Running Virtual Machines    : 0
Current Workload            : 0
Services Running on It      :
        nova-compute
Instances Running on It     :
        Instance-023
```

```
Host Name                   : compute4
Virtual CPU                 : 4
Used Virtual CPU            : 2
Available R.A.M             : 3870
Used R.A.M                  : 461
Free R.A.M (MB)             : 3358
Available Disk (GB)         : 279
Used Disk (GB)              : 8
Disk Available Least(GB)    : 271
hypervisor                  : QEMU
Running Virtual Machines    : 0
Current Workload            : 0
Services Running on It      :
        nova-compute
Instances Running on It     :
```

Figure: 4.18.1 Details of compute node

4.19 Billing system resource usage prices set:



Set Unit Price Of Resources :

```
Resource Name :CPU (processing
Power)

Price Per :1 hour
Price ($) :0.2
Primary Cost ($) :0
Unit :1 Ghz

Price ($)     : 0.2
Price Per     : 1
Primary Cost: 0
Unit          : 1 Ghz
```

```
Resource Name :RAM (Memory)

Price Per :1 hour
Price ($) :0.2
Primary Cost ($) :0
Unit :1 Gb

Price ($)     : 0.2
Price Per     : 1
Primary Cost: 0
Unit          : 1 Gb
```

```
Resource Name :Hard Disk

Price Per :1 hour
Price ($) :0.2
Primary Cost ($) :0
Unit :10 Gb

Price ($)     : 0.2
Price Per     : 1
Primary Cost: 0
Unit          : 10 Gb
```

Figure: 4.19.1 actual usage calculation

CONCLUSION:

We have introduced IAAS model which provides services in comprehensive and homogenous way .It having multiple compute node and backup mechanism which make it more robust and load sharing system. Efficient billing based on actual usage is the distinctive feature of this system.

FUTURE WORK:

Integrate billing module with cloud dashboard .Furthermore, this model could be expanded by including other business support systems such as client management or network operations.

BIBLIOGRAPHY & REFERENCES

1. Igor Ruiz-Agundez, Yoseba K. Penya and Pablo G. Bringas"A Flexible Accounting Model for Cloud Computing".
2. Marquez, F.G. , Henriksson, D. , Ferrera, D.P "Accounting and Billing for Federated Cloud Infrastructures".

Web:

- Cloud infrastructure Information. http://www.dialogic.com/~/media/products/docs/whitepapers/12023-cloud-computing-wp.pdf
- Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/pricing/
- Openstack docs http://docs.openstack.org/essex/
- Dough billing system http://www.slideshare.net/lzyeval/dough-openstack-billing-project
- MooseFS file system http://www.moosefs.org/
- Ceilometer billing system https://wiki.openstack.org/wiki/Ceilometer
- 6fussion http://www.6fusion.com/
- Google compute cloud https://cloud.google.com/pricing/compute-engine