OpenStack Folsom Project Report

By:

Suri Samson
Gary Longoria

Presented to:

Sameer Verma, Ph.D.
Professor, Information Systems Department
College of Business
San Francisco State University

Copyright and License

Table of Contents

Executive Summary

The SFSU OpenStack Project is an initiative that was started and is sponsored by Professor Sameer Verma. Seeing the possibilities and the benefits that a cloud computing system could provide to the school, he has initiated this project as a proof of concept (POC), in order to show that such a system is viable and can be undertaken by students as a learning experience.

The end goal is to have an OpenStack system set up in the computer lab at SFSU where Information Systems students that are in need of a server for learning needs, can request an account, login and spin up an instance that they can use as a sandbox. Additionally, if the system is successful it can serve as a model for future IT infrastructure on the campus. This report will provide an overview of cloud computing, our task, our approach and installation steps, issues that came about, as well as where we see the future of the OpenStack project at SFSU.

Introduction

Introduction to Cloud Computing

Cloud computing is a phrase that gets a lot of play today, and for good reason. It represents a new paradigm for computing and the delivery of services, with the ability to scale from a single user to millions, almost instantly, while reducing costs and complexity. Cloud computing, at its simplest, is a service that is delivered to the user over a network (usually the Internet). The most basic type of cloud computing, while being the most flexible and granular in control, is Infrastructure as a Service (IaaS). It provides the basic, but critical, compute and storage services.

This most closely mimics having your own physical machines, except it is a virtual machine along with many other virtual machines running on physical computers orchestrated by software. This makes the most efficient use of physical computing resources, by virtually slicing and dicing them based on customer needs and usage. This is superior to older methods of running one service on single piece of hardware, where average utilization rates were often below 10%. On the other hand, other services might be at or near capacity, and can't expand for technical or organizational reasons.

X-as-a-Service (XaaS)

On top of the compute and storage services comes an array of more specialized services; such as Platform as a Service (PaaS), Software as a Service (SaaS), Desktop as a Service (DaaS), and Database as a Service (DBaaS). PaaS allows application developers to run their software in the cloud without having to configure or manage the operating system and hardware underneath, having to add additional physical or virtual machines, or deal with load balancing. SaaS allows end users to use applications over the Internet, without having to install, maintain, and update them, or worry about backing up their data. Facebook, Dropbox, Pinterest, and Google Apps (Gmail, Drive/Docs, Maps, etc.) are common SaaS that regular users interact with on a daily basis. DaaS offers a full desktop operating system and applications over the Internet, so there is no worrying about maintenance and backups. DBaaS offers high performance databases from various vendors/projects, without having to administer the database and the underlying hardware on your own. Additionally, there are many more as-a-service (*X*aaS) models delivering services over the Internet, with the common denominator between them being abstracting away the layers of complexity below that service layer.

Cloud Storage

There are 3 types of cloud storage: Block storage, File System storage, and Object storage.

Block storage provides "bare metal" access to a physical storage device (tape, HDD, NAND-based SSD flash storage), which is exposed via commonly sized blocks. This is generally the most efficient, flexible, highest performing, and reliable way to access storage; but working directly at the block level is difficult. Usually block storage is only directly accessed by specialized applications like databases due to the database architecture, where records are stored in blocks on a drive, and use cases where performance needs are high. But more commonly, this complexity can be abstracted away by deploying a file system in a volume on the block device, so users rarely interact directly with block storage.

File system storage is what users are most familiar with, and interact with on their traditional personal computer systems. It provides a hierarchical structure, with files and folders/directories, and permissions so users can read, write, and execute to what they have been authorized. They can be shared over a network via Server Message Block/Common Internet File System (SMB/CIFS) on Windows or Samba, or via Network File System (NFS) on *nix-based hosts. These systems are easier to setup and use, but there is some performance penalty associated with them. In cloud services, file system storage is generally used to support legacy applications that are forklifted to the cloud and still dependent on a file system structure, but less for newer cloud native applications.

Object storage is a newer and lesser-known form of storage designed for the cloud. Object storage does not provide block level or file level access. It treats every file as an object, or a blob of data, with every object having a unique object ID. While object storage does not have a hierarchical file structure, a pseudo-hierarchical structure with directories and nested folders can be used for the user's organizational purposes. Objects are made available through an API like S3 or Swift, and objects can't be modified in part, they must be updated in whole. The objects can be made publicly available over the Internet, or restricted to those with appropriate permissions, all that is needed is the object ID, and, if necessary, credentials. This provides a centralized system for the files, with the object ID acting as a receipt for the user, that can be accessed by physical or virtual machines. This provides simplification and scalability, without needing to know the exact location of the files requested, only the object ID is necessary. The object storage system keeps track of where the objects are in the physical storage system, and handles duplication for redundancy and failover. This allows for a storage system that can scale into petabytes and theoretically to infinity.

Public, Private, and Hybrid Clouds

In the cloud computing business, there are different types of clouds: public and private, as well as hybrid clouds that utilize both. Our project will explain the main differences between these different types of clouds. We will take a quick look through the different options for private cloud software. After that, we will go into OpenStack, an open source Infrastructure as a Service (IaaS) cloud computing platform, to implement a private cloud. We will provide a How-To guide, with all the instructions needed, to set up a private cloud with OpenStack.

Public clouds are cloud systems that are available for everyone's use, such as Dropbox and Amazon's EC2. These services can be both free and subscription based, depending on the provider's business strategy and user's needs.

A private cloud can offer the same services as a public cloud; however, its services are meant for in-house enterprise use only. There are a few reasons to choose a private cloud over a public cloud with security being one of the biggest reasons.

There are a couple main reasons why cloud computing has been on the rise. One reason is elasticity and scalability. "This encompasses the idea of computing on demand, and the ability to increase the supply of computing resources as they are needed to deal with spikes in demand for a particular application or service. There's also the idea of turning computing resources into a commodity so more can be added over time, as needed, to ensure systems are almost infinitely scalable" (Rubens, 2010). This allows variable amounts of users on at the same time, or even enabled whenever it is necessary to alleviate bottlenecks.

There is a downside to private clouds, their cost. The capital required to build and operate a private cloud is substantially more than having a public cloud hosted. The need to purchase the hardware, as well as hiring an administrator to manage the private cloud, are the two main disadvantages.

In general, public clouds are highly structured and automated. Usually it's not possible for enterprises to get particular SLA's (Service Level Agreement) for their specific needs. Public clouds usually come with a standard SLA, so you just have the choice to take it or leave it. One area of concern that all companies considering using public clouds should always keep in mind is this: you have to adapt to the cloud, the cloud doesn't adapt to you. The first thing you should think of is how critical the data is that you want to store in a cloud. The more critical the data is, the more important it is to keep it in house.

Open Source Software for Private Clouds

Open source software for private clouds is widely available. OpenStack, Eucalyptus, Ganeti, OpenNebula are the main competitors in the private cloud area. All of these except for Ganeti provide a clean user GUI, so we will just compare OpenStack, Eucalyptus and OpenNebula. Out of the 3 remaining, OpenNebula focuses tremendously on Data Center virtualization, and has fewer options for controlling what we want from a private cloud. Therefore, we are left with Eucalyptus and OpenStack. With Eucalyptus, there are many tools that are comparable to OpenStack's, however, OpenStack has a user-friendlier GUI and the functions seem simpler to use and deploy. After taking a look at the available software, OpenStack fits our needs best.

OpenStack: Open Source Software for Private Clouds

"OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface" (OpenStack, 2013). OpenStack has many features deploying and managing a private cloud simple.

"OpenStack is architected to provide flexibility as you design your cloud, with no proprietary hardware or software requirements and the ability to integrate with legacy systems and third party technologies. It is designed to manage and automate pools of compute resources and can work with widely available virtualization technologies, as well as bare metal and high-performance computing (HPC) configurations" (OpenStack, 2013). We are able to run OpenStack on top of Ubuntu. OpenStack has the capabilities to make the transition from other cloud services with load balancing, which allows it to migrate the services without downtime to the end users. Since OpenStack has met the criteria for this project, we moved forward and installed it on our test servers running Ubuntu 12.04 LTS.

Problem Statement

Our work for this project continues and builds upon the work completed by Brandon Lai and Pascal Schuele in the Fall 2012 semester. Working with Professor Verma, they were able to complete the first major milestone of the overall SFSU OpenStack Project by installing the software on a single server and successfully creating instances. Our project was to install the OpenStack system across four servers of varying hardware and successfully create a POC of a scalable private cloud system on the SFSU campus. In order to attain this goal, our work consisted of learning how to install Ubuntu Server, DevStack, and OpenStack Folsom. Issues that we would have to tackle throughout our project included unfamiliarity with Linux, the use of “hand-me-down” servers, and limited access to the servers.

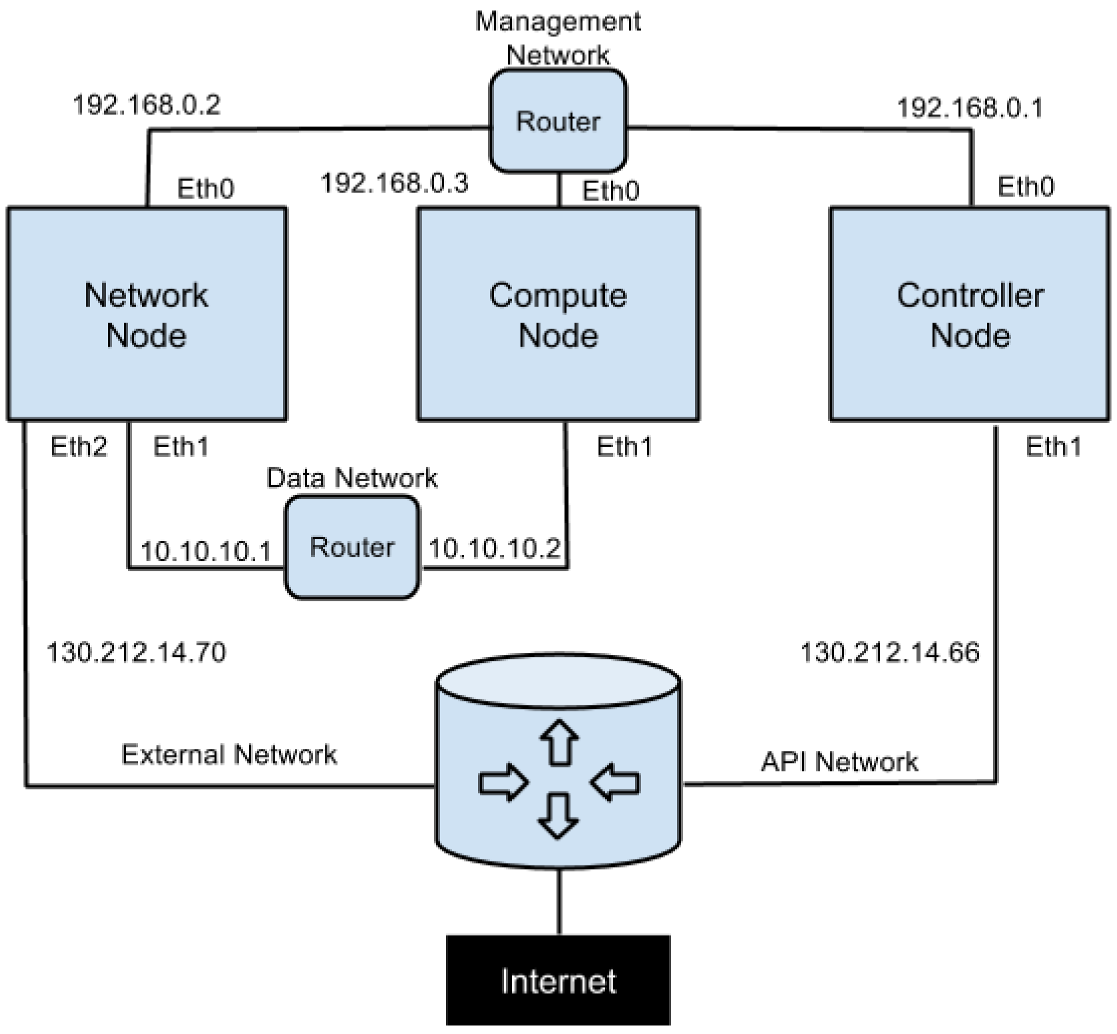
Working with Linux was our largest challenge during the project, as both of us were novices at using the operating system. We were both familiar with the concepts behind command-line and shells, but actually navigating through the system without knowing the commands was difficult. To add to this we also had to learn about security privileges in order to install the various programs within OpenStack and have them interface correctly.

Another challenge we had was the servers that we were working with, as they were not all the same and ranged in age from about 4-8 years. This led to issues with the hardware malfunctioning at times and increased rework when we had to reinstall and reconfigure services.

While we had the entire semester to work on the project, we were limited to working on the servers only a few hours a day, twice a week when we could meet with Professor Verma. This was due to the fact that we did not have a set location in the computer lab for the servers, so they had to be retrieved from the computer lab server room and setup in the open lab each time we met. At the end of each meet, the servers also had to be disconnected and returned to their rack space in the server room.

Approach

We started by learning how to install Ubuntu Server on a single machine and reviewing command line documentation online. Once this step was complete we reviewed the past documentation created by Brandon and Pascal, and their process for installing a development version of OpenStack called DevStack. This non-production version of OpenStack is used to quickly create an OpenStack environment on a single machine and to demonstrate the running of different services (DevStack, 2013). Once these preliminary steps were complete we moved on to installing the Folsom version of OpenStack, which consists of three nodes: Controller, Network, and Compute. For our scalability purposes, we also had a fourth node as an additional compute node. Our architecture for the OpenStack system is detailed in the following diagram:

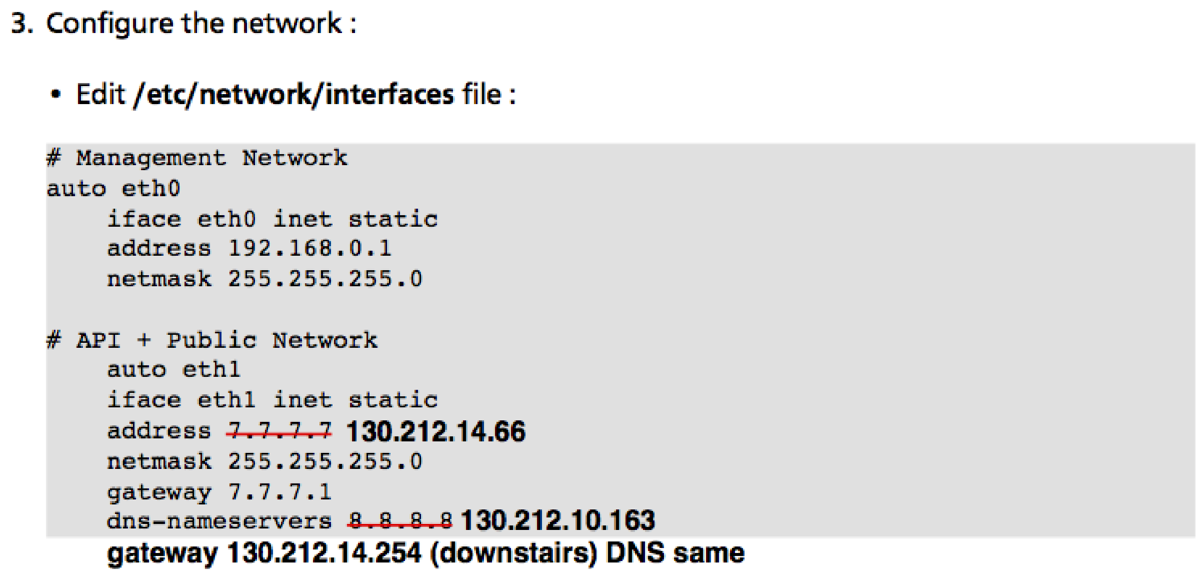


How-To

This section will provide a high-level overview of the steps we took to install OpenStack Folsom and is based on the basic install manual (OpenStack, 2013).

Installing the Controller Node

This step involved configuring the network (shown in the image below) and installing the following components: Databases (with MySQL), Message Queueing (with RabbitMQ), Identity (Keystone), Image Management (Glance), Nova (without nova-compute), Block Storage (Cinder), Networking (Quantum), and Dashboard (Horizon). Of important note is that all installations must be done as root using "sudo su -".



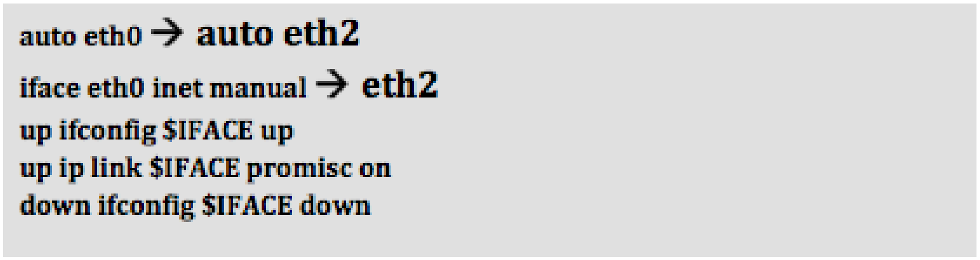
3. Configure the network :

- Edit **/etc/network/interfaces** file :

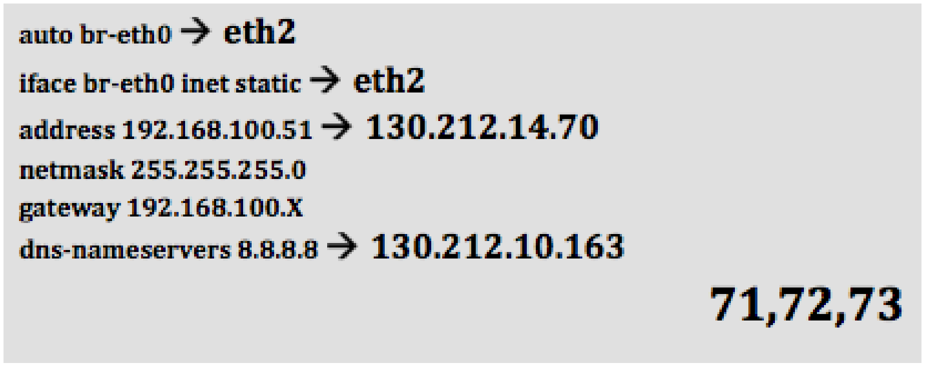
```
# Management Network
auto eth0
    iface eth0 inet static
    address 192.168.0.1
    netmask 255.255.255.0

# API + Public Network
    auto eth1
    iface eth1 inet static
    address ~~7.7.7.7~~ 130.212.14.66
    netmask 255.255.255.0
    gateway 7.7.7.1
    dns-nameservers ~~8.8.8.8~~ 130.212.10.163
    gateway 130.212.14.254 (downstairs) DNS same
```

Installing the Network Node

The network node provides: virtual bridging (Open-vSwitch + Quantum Agent), DHCP Server (Quantum DHCP Agent), and virtual routing (Quantum L3 Agent). As shown in the images below, this step involves configuring Network Services and is where assignment of static IP address to the bridge takes place.



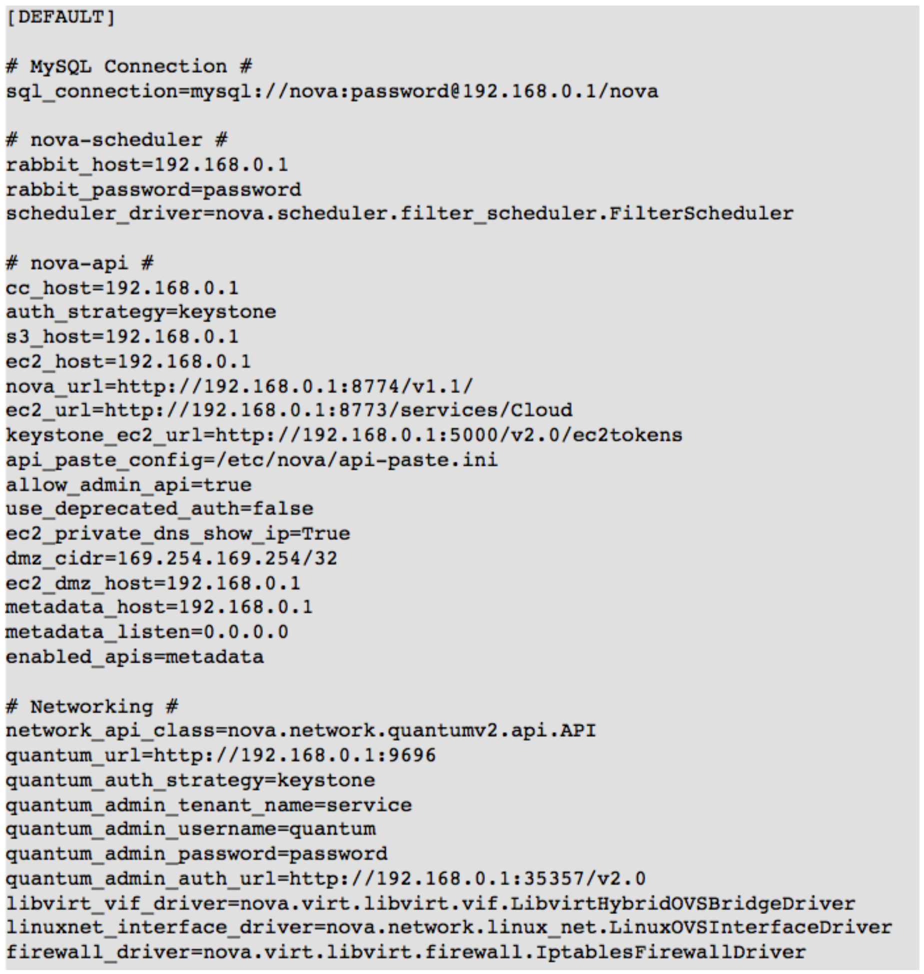
```
auto eth0 → auto eth2
iface eth0 inet manual → eth2
up ifconfig $IFACE up
up ip link $IFACE promisc on
down ifconfig $IFACE down
```



```
auto br-eth0 → eth2
iface br-eth0 inet static → eth2
address 192.168.100.51 → 130.212.14.70
netmask 255.255.255.0
gateway 192.168.100.X
dns-nameservers 8.8.8.8 → 130.212.10.163
                                        71,72,73
```

Installing the Compute Node

The final node to be installed implements the following services: Hypervisor, nova-compute, and Quantum OVS Agent. In the manual it specified to use KVM as the infrastructure for our virtual machines, but because of a lack of supportability, we had to use QEMU instead. The following images show the configuration for Nova that was used.

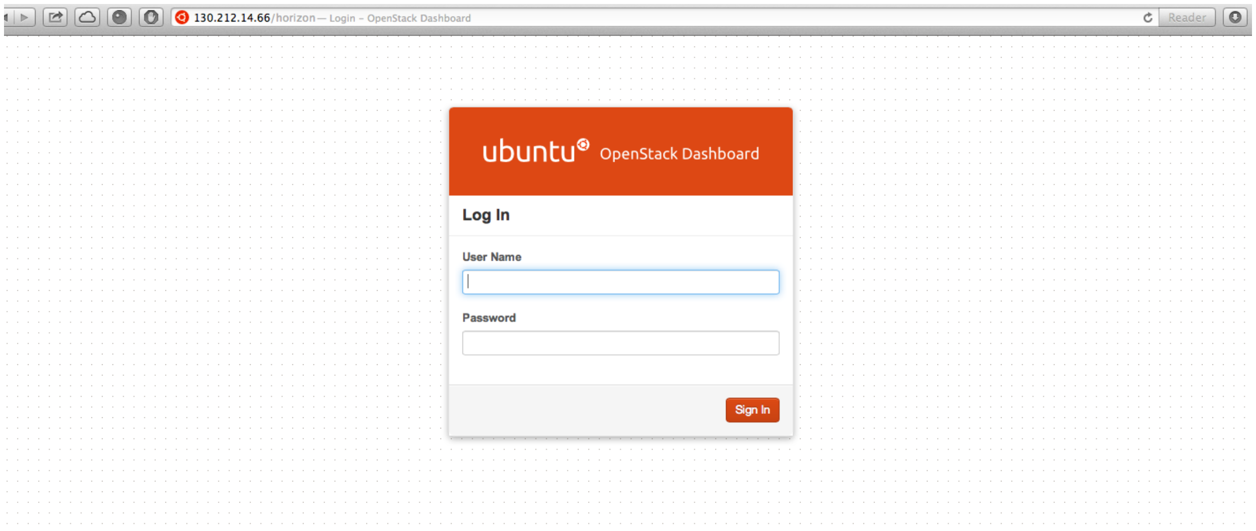


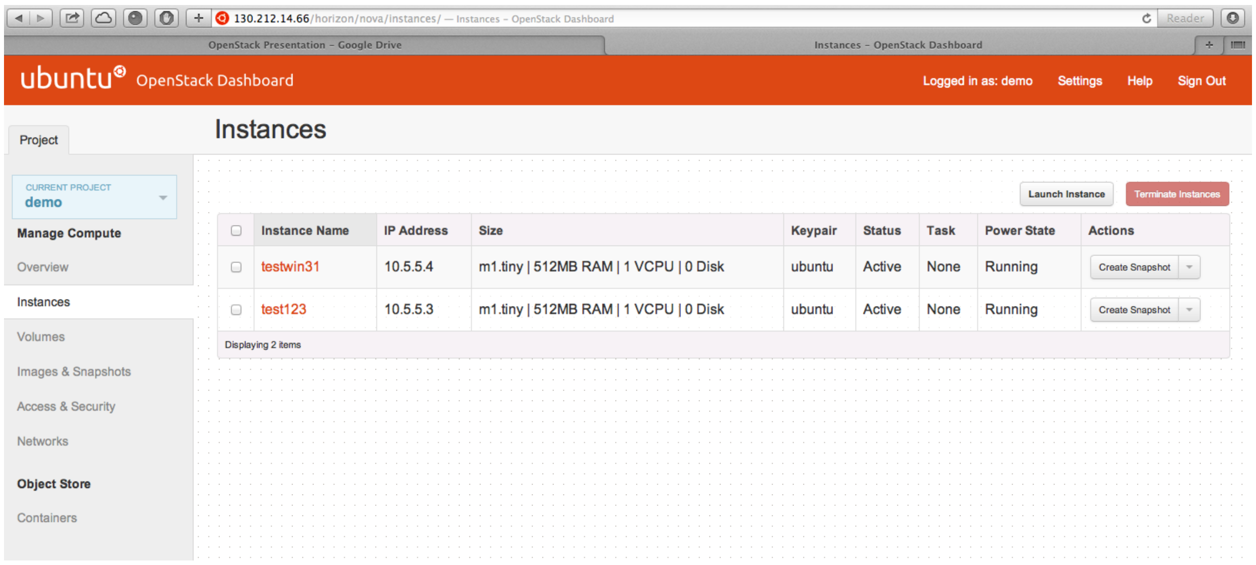
```
[DEFAULT]

# MySQL Connection #
sql_connection=mysql://nova:password@192.168.0.1/nova

# nova-scheduler #
rabbit_host=192.168.0.1
rabbit_password=password
scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler

# nova-api #
cc_host=192.168.0.1
auth_strategy=keystone
s3_host=192.168.0.1
ec2_host=192.168.0.1
nova_url=http://192.168.0.1:8774/v1.1/
ec2_url=http://192.168.0.1:8773/services/Cloud
keystone_ec2_url=http://192.168.0.1:5000/v2.0/ec2tokens
api_paste_config=/etc/nova/api-paste.ini
allow_admin_api=true
use_deprecated_auth=false
ec2_private_dns_show_ip=True
dmz_cidr=169.254.169.254/32
ec2_dmz_host=192.168.0.1
metadata_host=192.168.0.1
metadata_listen=0.0.0.0
enabled_apis=metadata

# Networking #
network_api_class=nova.network.quantumv2.api.API
quantum_url=http://192.168.0.1:9696
quantum_auth_strategy=keystone
quantum_admin_tenant_name=service
quantum_admin_username=quantum
quantum_admin_password=password
quantum_admin_auth_url=http://192.168.0.1:35357/v2.0
libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtHybridOVSBridgeDriver
linuxnet_interface_driver=nova.network.linux_net.LinuxOVSInterfaceDriver
firewall_driver=nova.virt.libvirt.firewall.IptablesFirewallDriver
```

```
# Compute #
compute_driver=libvirt.LibvirtDriver
connection_type=libvirt   QVMU, not KVM

# Cinder #
volume_api_class=nova.volume.cinder.API

# Glance #
glance_api_servers=192.168.0.1:9292
image_service=nova.image.glance.GlanceImageService

# novnc #
novnc_enable=true
novncproxy_base_url=http://7.7.7.7:6080/vnc_auto.html
vncserver_proxyclient_address=192.168.0.3
vncserver_listen=0.0.0.0
```

Logging in Horizon and Creating an Instance

Having completed all the prior steps and configurations, we were then able to log into Horizon and create our first instances.


130.212.14.66/horizon — Login - OpenStack Dashboard
Reader
ubuntu® OpenStack Dashboard
Log In
User Name
Password
Sign In

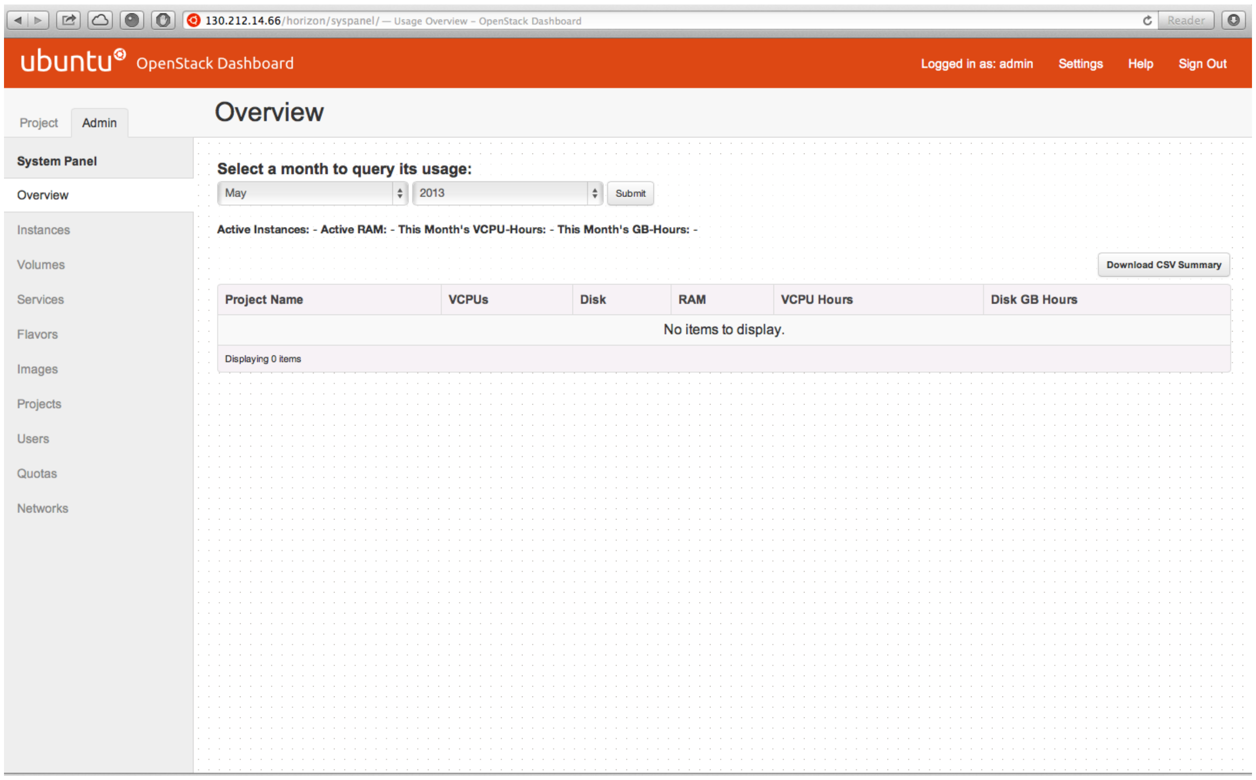


Instances

Instance Name	IP Address	Size	Keypair	Status	Task	Power State	Actions
testwin31	10.5.5.4	m1.tiny | 512MB RAM | 1 VCPU | 0 Disk	ubuntu	Active	None	Running	Create Snapshot
test123	10.5.5.3	m1.tiny | 512MB RAM | 1 VCPU | 0 Disk	ubuntu	Active	None	Running	Create Snapshot

Displaying 2 items

The interface has a different look from the DevStack version, however, once logged in the services and options available are nearly identical.



Overview

Select a month to query its usage:

May 2013

Active Instances: - Active RAM: - This Month's VCPU-Hours: - This Month's GB-Hours: -

Project Name	VCPUs	Disk	RAM	VCPU Hours	Disk GB Hours
No items to display.					

Displaying 0 items

Issues Encountered and Solutions

During the course of our installation of OpenStack Folsom, we encountered numerous issues, but we were able to also find solutions to most of them. The earliest issue we had to solve was devising our network architecture and configuring the routing. We were able to solve this problem by sketching out a diagram and including each of the ethernet ports we were using and the associated IP addresses. The diagram on page 9 is the final architecture that we used to build the system.

Another issue we faced was running apt-get on each server for updates, as they initially would not route through the network node in order to reach the Internet. We tried various methods to pinpoint the ports, but for reasons still unknown, the connection could not be made. We eventually solved this problem by plugging in the nodes directly to the Internet in order to run apt-get, and once the updates were applied, then the routing as depicted in our architecture worked. The current setup uses the 130.212.14.66 node as an update cache (apt-cache-ng) and the other two machines (192.168.0.2 and 192.168.0.3) use this cache for installing and upgrading packages.

We also faced an issue with hypervisor support. The hypervisor is a piece of software that imitates hardware, and creates the virtual environment that hosts the guest virtual machines (VMs). Kernel-based Virtual Machine (KVM), the default hypervisor for OpenStack, failed to work due to the age of the servers we were using, being made before hardware accelerated virtualization was added to Intel and AMD CPUs, complicated our installation of the compute node. We resolved this by using QEMU instead, another hypervisor that is not as fast as KVM, but is not dependent on hardware based virtualization support. After this we saw no issues using QEMU despite the guide specifying the use of KVM.

There were also some issues that persist and require further work in order to be resolved; these include assignment of a public IP address to virtual machines and routing the virtual machines outside of the SFSU network. After our instances were running, we ran into an issue where computers hardwired into the SFSU network could log in to the servers, but when trying to access through WIFI, the location could not be found. We eventually determined that this was due to the security setup of the wireless network, however, we could not find a workaround for this issue.

Future Scope and Phases

There are several areas in which the OpenStack system we created on campus could be enhanced. The future phases of the project could increase the scope by obtaining better servers, adding more compute nodes, adding a storage cluster, implementing additional OpenStack services, implementing a Platform as a Service (PaaS) on top of OpenStack, and running the servers in a data center, either on campus and/or off-campus. Additionally, OpenStack is a fast moving target so the inclusion of a bare metal provisioning system should also be considered.

As shown in the documentation for OpenStack and in our POC, additional compute nodes can be added to our system. Given the setup of our system, these additional nodes would have to be connected to both the management network and the API network. The advantage of this would be to show the horizontal scalability of OpenStack and how it can handle increasing demands in a seamless manner.

Given the age of the servers we were using and the compatibility and hardware issues we encountered using them, an upgrade of the servers should also be included in future phases. This would allow use of KVM hypervisor and also provide more time to work on installation issues, rather than using time for rework. In addition to this, have the servers located and always connected would be a great help. If they could be provided space in a data center or server room on campus, this would also increase time dedicated to the project.

Finally, new versions of OpenStack are developed on a 6-month time based release cycle, so new versions of the system will need to be implemented continually. The way we would do this now would be to wipe all the servers and start over fresh, of course, if this system is to be used by students, this would be less than ideal. A method of continuous integration of new releases into the system would be needed. The future scope of the project should include looking into implementing a Metal as a Service (MaaS) system along with a service tool such as Juju to test such scenarios. The addition of these tools would further show the viability and scalability of the system.

Conclusion

The goal of this project was to show the viability of OpenStack on the SFSU campus by creating a POC system. We started by installing a development version of the system, DevStack, on a single server in order to build our skills and understanding. We then went on to create a scalable system with multiple servers using OpenStack Folsom. We ran into multiple issues during our installation process, but were able to find solutions most of them. The ones that remain will have to be worked on by future students, but in the meantime they do not impose an impediment to usage of the system. Along with solving these issues, there are a number of areas in which the system could be improved upon in the future. Overall, having a private cloud system on campus would be of great benefit both to the students and the school, as it could help control future infrastructure costs while providing a useful resource for students to use and learn from.

Works Cited

DevStack. (2013). *Overview*. Retrieved from DevStack: http://DevStack.org/overview.html

OpenStack. (2013). *Basic Install Intro*. Retrieved from OpenStack: http://docs.openstack.org/folsom/basic-install/content/basic-install_intro.html

OpenStack. (2013). *OpenStack Compute*. Retrieved from OpenStack: http://www.openstack.org/software/openstack-compute/

OpenStack. (2013). *OpenStack: The Open Source Cloud Operating System*. Retrieved from OpenStack: http://www.openstack.org/software/

Rubens, P. (2010, January 27). *Private Cloud, Defined*. Retrieved from ServerWatch: http://www.serverwatch.com/trends/article.php/3861191/Private-Cloud-Defined.htm