University of Alberta

MINT 709
Capstone Project Report
March 2016

OPENSTACK
ORCHESTRATE PUBLIC AND PRIVATE CLOUD USING OPENSTACK/VCENTER INTEGRATION AND PROVIDING TENANT SEPARATION

BIVEK RAJ SINGH
Masters of science in Internetworking

Supervisor: Muhammad Durrani
Brocade Communications Systems

Abstract

Rapid development and advancement in virtualization technology has made it possible to effectively distribute physical resources of a machine across multiple operating systems running simultaneously. With the virtualization technology getting more and more popular the need was felt to centrally control and manage these virtual infrastructures which led to the evolution of cloud computing.

With the concept of cloud computing gaining momentum many solutions emerged which promise to provide complete virtualization and cloud computing package, but most of the solution being proprietary solution followed different standard and architecture making them incompatible with one another forcing a corporation to stick to a single solution. This led to a need of a solution which could support multiple proprietary solution. The solution came in the form of a community supported open source software package "OpenStack".

This project presents a proof of concept implementation of integrating OpenStack with vCenter. It demonstrates the step by step deployment of Mirantis OpenStack using Fuel (Intuitive GUI based tool) and integrate it with the vCenter server to utilize the vSphere ESXi infrastructure as the compute hypervisor providing central control and management using OpenStack dashboard. This project also puts light into OpenStack Heat orchestration to create instances and finally demonstrates the use of Vyatta vRouter as the network management tool for OpenStack cloud.

With OpenStack having the potential to be the future of cloud operation and management system, the knowledge of OpenStack architecture and the ability to implement and administer cloud infrastructure using OpenStack as shown in the project would be a great asset to have for an internetworking graduate who wish to pursuit their career in the field of datacenter or cloud design and administration.

Acknowledgement

I would like to thank Dr. Mike MacGregor (*MINT Director*) and Mr. Shanawaz Mir (*MINT Coordinator*) for providing me the opportunity to work on this project. I would also like to express my gratitude to Mr. Muhammad Durrani for supervising and providing valuable suggestions and guidance during the implementation of this project. Finally, I would like to thank my family and all my friends who have directly or indirectly helped me for the successful completion of this project.

TABLE OF CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

The advancement in compute, storage and networking capacity and an increase in the adaption of virtualization technology has changed the architecture of modern datacenter into a service oriented architecture which utilizes distributed computing resources across the datacenter to provide utility computing. This shift from on demand computing to a service based distributed computing led to the evolution of cloud computing. Offering features such as high performance, scalability and economic service the demand of cloud computing is increasing day by day. This increase in demand created a need for software that provides a complete cloud computing solution package.

There are many solutions developed that provide complete cloud computing solution. But the problem with most of the solutions is they are proprietary solution and have their own hardware, software and virtualization standards. This makes one solution incompatible with other which makes it very difficult to implement these solutions in large environments with multiple underline physical hardware and virtualization technology used. This initiated an open source community effort to develop a ubiquitous cloud computing solution that supports multiple hardware and hypervisor platform called “OpenStack”. Initially developed by NASA and Rackspace, OpenStack is now managed by OpenStack Foundation. Within 5 short years of its development OpenStack has gained a huge popularity in the world of cloud computing with contributions from major player like Cisco, HP, RedHat, IBM etc.

In this project we demonstrate management of cloud on VMware vSphere infrastructure using OpenStack vCenter integration. VMware vSphere is one of the most popular virtualization solution used across enterprise datacenters. This integration of OpenStack with vCenter brings all the benefits of vSphere to OpenStack environment which includes the advance feature like vMotion, High Availability(HA), Fault Tolerance (FT) and Dynamic Resource Scheduling (DRS) making it more scalable, flexible, robust and enterprise friendly. Also, it opens the possibility to integrate other hypervisors (KVM, XEN etc.) to the cloud under the management of OpenStack.

Scope

The scope of this project is to presents the proof of concept demonstration to orchestrate a private and/or public cloud using OpenStack on vSphere infrastructure (OpenStack vCenter integration) and manage the network resources of the cloud using Vyatta vRouter providing tenant separation.

2 TERMINOLOGY AND CONCEPT

2.1 VIRTUALIZATION

Virtualization means creating a virtual version of an actual implementation or process. It is an abstraction layer separating hardware from the software using a software that emulates the underline hardware. Today with the advancement in virtualization technology almost all the physical attribute of a computer can be virtualized which include memory virtualization, storage virtualization, network virtualization etc.

By using virtualization technology, it is possible to capture the state of all the attributes at a point of time (take snapshot) and save it so that user can roll back to that instance at any time. That state can also be copied from one physical machine to another eliminating complete dependency on one physical machine.

2.2 HYPERVISOR

Hypervisor is the software program that makes virtualization possible. It acts as a bridge between the physical hardware and the operating systems running on that hardware. It allows multiple operating systems to run simultaneously on top of a physical hardware by managing and distributing the available physical resources to each operation system. Hypervisors can be classified into two types:

- **Type 1 hypervisors:**
  Type 1 hypervisors are also called bare metal or native hypervisors. They are installed directly on physical hardware and are more efficient. They are used in production grade datacenters and servers across enterprise environment e.g. Cirtix Xen server, VMWare ESX/ESXi etc.

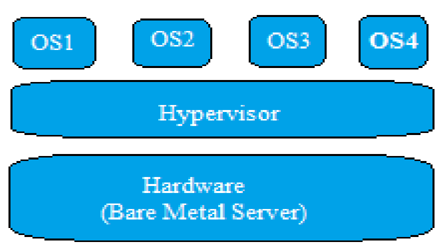


Figure 2.1 type 1 hypervisor

- **Type 2 hypervisors:**
  Type 2 hypervisors are the hypervisors that are installed on an operating system. They run guest operating system inside host operating system. e.g. VMWare Workstation, VMWare player, virtual box etc.

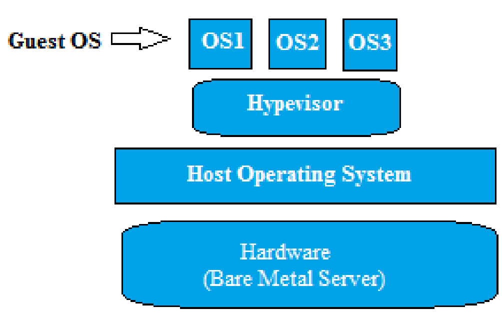


Figure 2.2 type 2 hypervisor

2.3 CLOUD COMPUTING

Cloud computing is the process of utilizing the distributed hardware resources across the network to perform computing task. The unused processing cycle of multiple processors connected across the network are utilized to perform a computing operating which makes cloud computing more efficient in terms of resource utilization. It allows users to share high power computing resources securely and effectively. [14] Cloud computing has hugely benefited small and medium enterprise as it has allowed them to use computing power as per requirement as a service eliminating the upfront cost of implementing whole infrastructure and maintain them.

Deployment model of cloud

- **Private Cloud**:
  Private cloud is the cloud that is entirely owned and managed by a single organization and is tailored as per the organization's requirements. All the infrastructure and services of a private cloud are connected to organization's private network. Creating a private cloud requires the organization to own significant physical resources and is expensive to build and manage. Private cloud is popular among the companies with very high data security and secrecy requirements such as financial institutions and intelligence services.

- **Public Cloud**:
  Public cloud is the cloud which are built and managed by cloud service providers and are connected to public network which can be accessed through internet. The user of the public cloud has no control over the infrastructure and architecture of the cloud and use the cloud as a service. Public cloud is much more efficient and economical from end user prospective. Google, Microsoft, Amazon are some of the large public cloud service providers.

- **Hybrid Cloud**:
  Hybrid cloud is the combination of public and private cloud. In hybrid cloud multiple clouds are integrated as per the requirement of the organization to form a new cloud thus its implementation may be different for different organizations.

2.4 RESTFUL API

RESTful (Representational state transfer) API is the application programming interface following REST software architectural style which communicates over HTTP (Hypertext Transfer Protocol). It is the API supported by most of the cloud based web services including Google, Amazon, Facebook, Twitter etc. The communication is done using the following four methods as defined in HTTP RFC 2616:

- **PUT**
  Put is a idempotent method it is used to change the state or update data
- **GET**
  Get is a nullipotent method used to retrieve data but does not change any data
- **POST**
  Post method is used to create a data
- **DELETE**
  Delete is also an idempotent method used to remove data

In our project we use Restful API to configure firewall on Vyatta vRouter using **cURL** command line.

3 COMPONENTS OF PROJECT

3.1 OPENSTACK

OpenStack is an open source software that provides cloud computing services. It consists of several interrelated components which are responsible for the control of hardware, storage and networking resources packaged together to form a complete cloud operating system. There is an intuitive GUI based web dashboard for management. Users can also use Restful API or command line tools for management. OpenStack supports all the popular open source and enterprise solutions making it a highly ubiquitous cloud computing platform ideal for infrastructure consisting of solutions from multiple vendors.

Components of OpenStack

OpenStack consist of several different projects combined together to form a superior cloud computing platform. Listed below are the core components of OpenStack.

- ***Compute (Nova)***

Nova is the major component of OpenStack which manages the compute instances. It is responsible for spawning, scheduling and decommissioning of machines on demand. It manages and automates compute resources pool and supports all widely available virtualization technologies and bare metal high performance configurations. [1] [11]

- *Object Storage(Swift)*

Swift is the object storage system in OpenStack. It is a highly scalable and redundant system. It uses RESTful API based on HTTP protocol to store and retrieve arbitrary unstructured data objects. It has a scale out architecture and provides data replication making it highly fault tolerant. [1] [11]

- *Block Storage(Cinder)*

Cinder provides persistent block storage to running instances. It has a pluggable driver architecture. Creation, attachment and detachment of the block devices to servers is managed by cinder. [1] [11]

- ***Networking(Neutron)***

Neutron is a system for managing network and IP address. It enables network connectivity as a service for other OpenStack services, such as OpenStack Compute. It ensures the prevention of network bottleneck during cloud deployments. Advance routing services from vendors is support by neutron because of its pluggable backend architecture. [1] [11]

- ***Dashboard(Horizon)***

OpenStack Dashboard (Horizon) is a web based user interface. Administrators and users use this interface to provision and automate cloud-based resources. The design allows for third party products and services, such as billing, monitoring and additional management tools. It can be customized by vendors and services providers with their own brand. [1] [11]

- ***Identity Service(Keystone)***

Keystone provides authentication and authorization service for other OpenStack services. It can be integrated to backend directory services like LDAP and supports multiple forms of authentication. [1] [11]

- ***Image Service(Glance)***

Glance provides discovery, registration and delivery for disk and server images. It stores and retrieves virtual machine disk images. It uses REST API to query information about disk image and allows clients to stream the image to new servers. [1] [11]

-*Source: OpenStack Training Guides* (May 10, 2015)

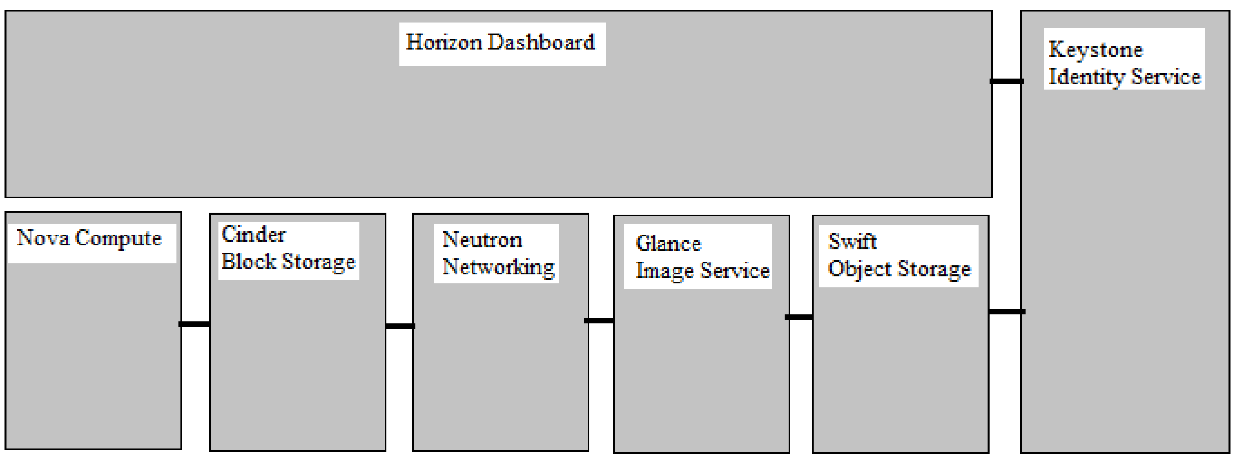


Figure 3.1 OpenStack components overview

Source: http://docs.openstack.org/security-guide/introduction/introduction-to-openstack.html (13/3/2016)

Beside the above listed core components some of the other components of OpenStack are:

- *Orchestration (Heat)*

Heat orchestration is used to launch multiple cloud application based on templates. It automatizes the scaling and addition of compute, storage and network resources across the cloud platform through an OpenStack-native REST API by executing HOT (Heat Orchestration Template) templates written in YAML. [9]

➢ *Telemetry (Ceilometer)*

Ceilometer is to metering tool. It collects and stores data used for automated actions or billing / chargeback purposes in form of samples. [9]

-Source: https://www.OpenStack.org/software/icehouse/

3.2 VMWARE VSPHERE

VMWare vSphere is a cloud computing and virtualizing software package developed by VMware. It is the most popular virtualization and cloud management tool and widely used across enterprise datacenters. It manages large volume of computing, storage and networking resources across the datacenter seamlessly. It offers distributive services such as vMotion, storage vMotion, Distributed Resource Scheduler (DRS), High Availability HA, and fault tolerance which enable very efficient and automated management of resources providing a highly available and reliable virtual infrastructure. [3]

Figure 3.2 VMware vSphere overview

-source: www.vmware.com

-

Components of VMware vSphere

VMware vSphere includes the following components:

➢ *VMware ESXi*

VMware ESXi a hypervisor layer that is installed on a bare metal (physical) servers. It provides effective distribution of processor, memory and storage resources across multiple virtual machines. [3]

➢ *VMware vCenter Server*

VMware vCenter Server is the management tool to manage ESXi hypervisors across the datacenters. It provides central management of compute, storage and network resources of all the ESX/ESXi across the datacenter by aggregating them into a cluster to create a highly reliable, fault tolerant and efficient virtual infrastructure. [3]

➢ *VMware vSphere Client*

VMware vSphere Client is a software that can be installed on any Windows PC that provides an interface to remotely connect and manage ESXi and vCenter server. [3]

- ***VMware vSphere Web Access***

VMware vSphere Web Access is web based GUI interface which provides an intuitive way to manage and administer vCenter server. [3]

-Source: Introduction to VMware vSphere (EN-000102-00)

3.3 OPENSTACK VCENTER INTEGRATION

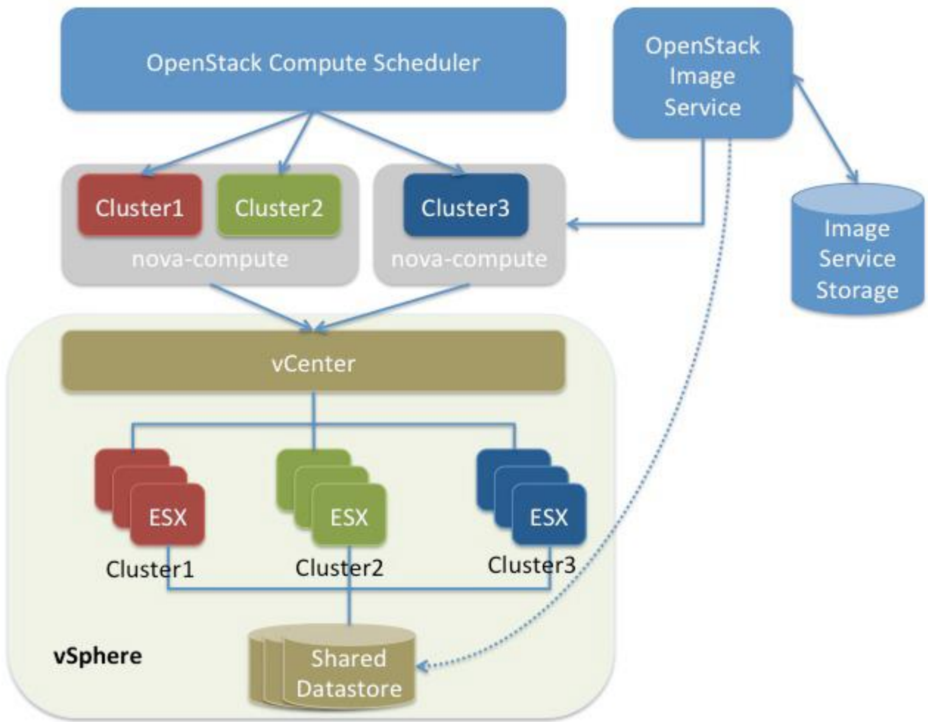


Figure 3.3VMWare driver architecture
-source: OpenStack Configuration Reference icehouse (June 1, 2015)

One of the most important feature of OpenStack is its ability to integrate with vCenter to control and manage the vSphere infrastructure. VMware VSphere is the most popular virtualization and cloud management software used across the enterprise datacenters. By integrating OpenStack with vCenter the advance features offered by vCenter such as vMotion, High Availability (HA), Fault tolerance (FT) and Dynamic Resource Scheduler (DRS) can be leveraged on OpenStack environment.

OpenStack nova-compute communicates with vCenter using VMware vCenter driver. The nova compute service acts as a proxy to translate Nova API calls to vCenter API calls using vCenter driver. Nova scheduler chooses a cluster on vCenter to launch an instance. Nova-compute then makes an API call and hands over the request to vCenter to launch a VM on the cluster chosen by nova scheduler. The actual ESXi host is then chosen by vCenter using Dynamic Resource Scheduler (DRS). [17]

OpenStack Networking on vSphere is done using nova-network service. It can be configured in following ways:

- **Nova-network service with the FlatManager or FlatDHCPManager**
  In this configuration all VM NICs are attached to the same port group. The name of the port group should be same as the flat_network_bridge value defined in nova.conf. It is **br100** by default. [2]

- **Nova-network service with the with VlanManager**
  In this configuration VM NICs are attached to the port groups created automatically by OpenStack compute to handle VLAN-tagged VM traffic. [2]

3.4 BROCADE VYATTA VROUTER

The Brocade Vyatta vRouter is a Debian-Linux distribution based software that provides virtual routing, firewall and VPN services for cloud computing environments. Listed below are some of the features of Vyatta vRouter:

Network Connectivity

- IPv4 and IPv6 dynamic routing protocols support
  - BGP
  - OSPF
  - RIP
  - Multicast
- Policy Based routing (PBR)
- 802.11 wireless
- Serial WAN interfaces
- Ethernet interfaces up to 10Gbps

Firewall Protection

- IPv4/IPv6 stateful packet inspection
- zone- and time-based firewalling
- P2P filtering.

Secure Connectivity

- secure site-to-site VPN tunnels with standards-based IPsec VPN
- secure network access to remote users via Brocade SSL-based OpenVPN functionality
- Dynamic Multipoint VPN (DMVPN)

Administration and Authentication

- network-centric CLI,
- Web-based GUI,
- external management systems using the Brocade Remote Access API.
- securely managed network management sessions using SSHv2, RADIUS, or TACACS+.

-Source: Brocade 5400 vRouter Data sheet

4 NETWORK TOPOLOGY DESIGN CONSIDERATIONS

4.1 BLOCK DIAGRAMMATIC VIEW OF THE PROJECT INFRASTRUCTURE

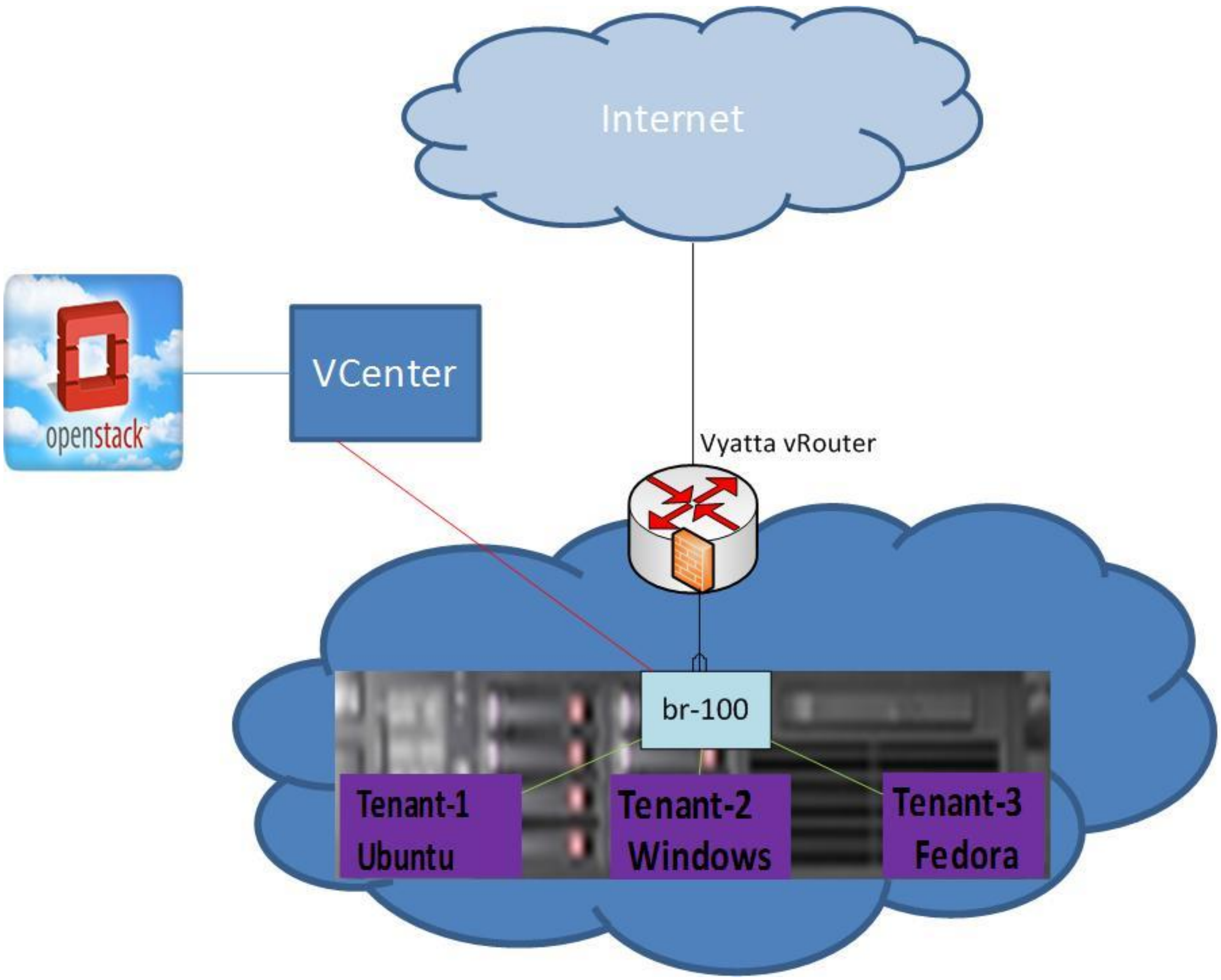


Figure 4.1 Block diagram representation of OpenStack/vCenter integration

4.2 Physical and Logical Network Connection of Mirantis OpenStack with vCenter

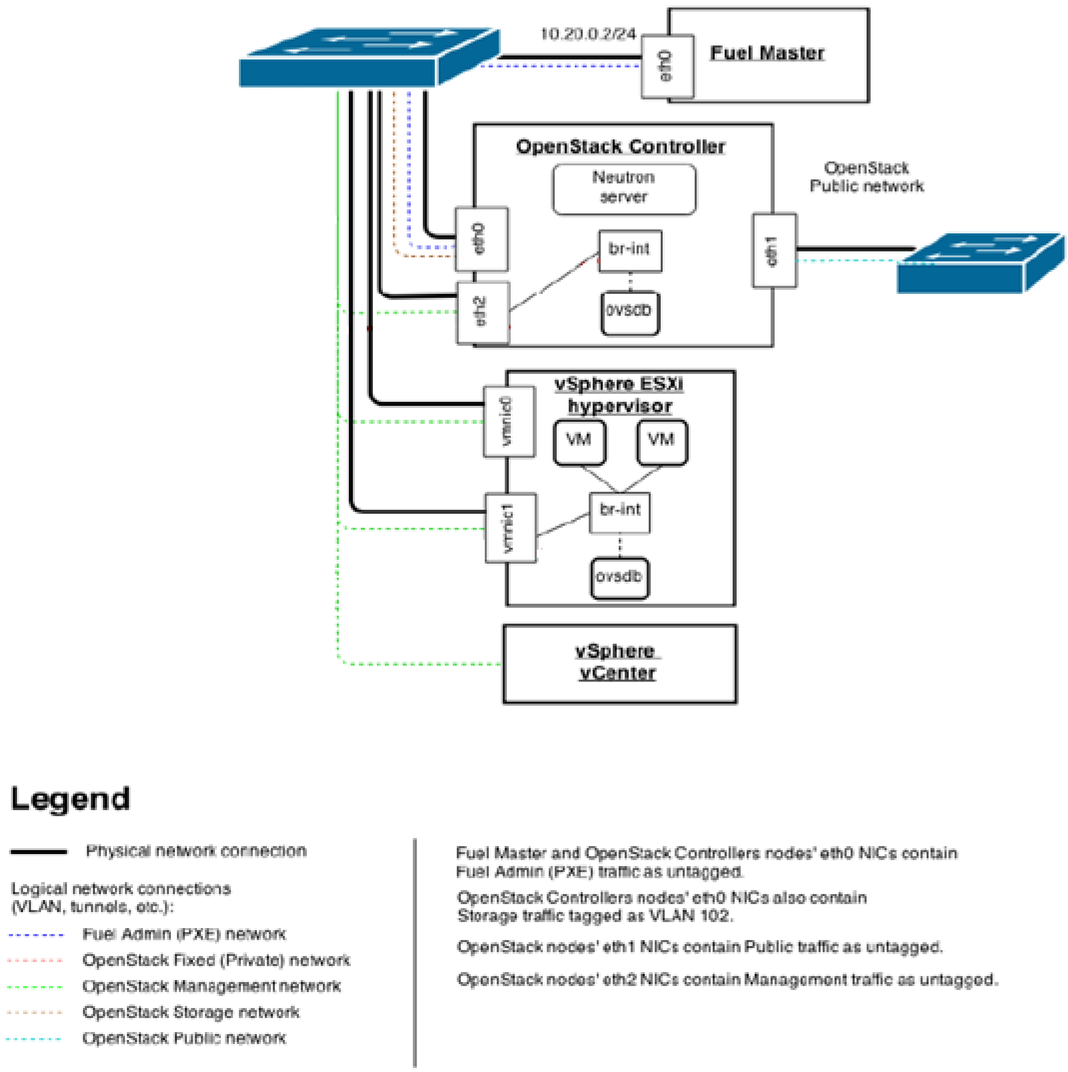


Figure 4.2Mirantis OpenStack physical and logical connection with vCenter

Source: Mirantis Reference Architecture (17 December 2014), page 9

(NSX portion is removed from the diagram as it is not implemented in this project)

4.3 Physical and Logical Network connection view of Project Infrastructure

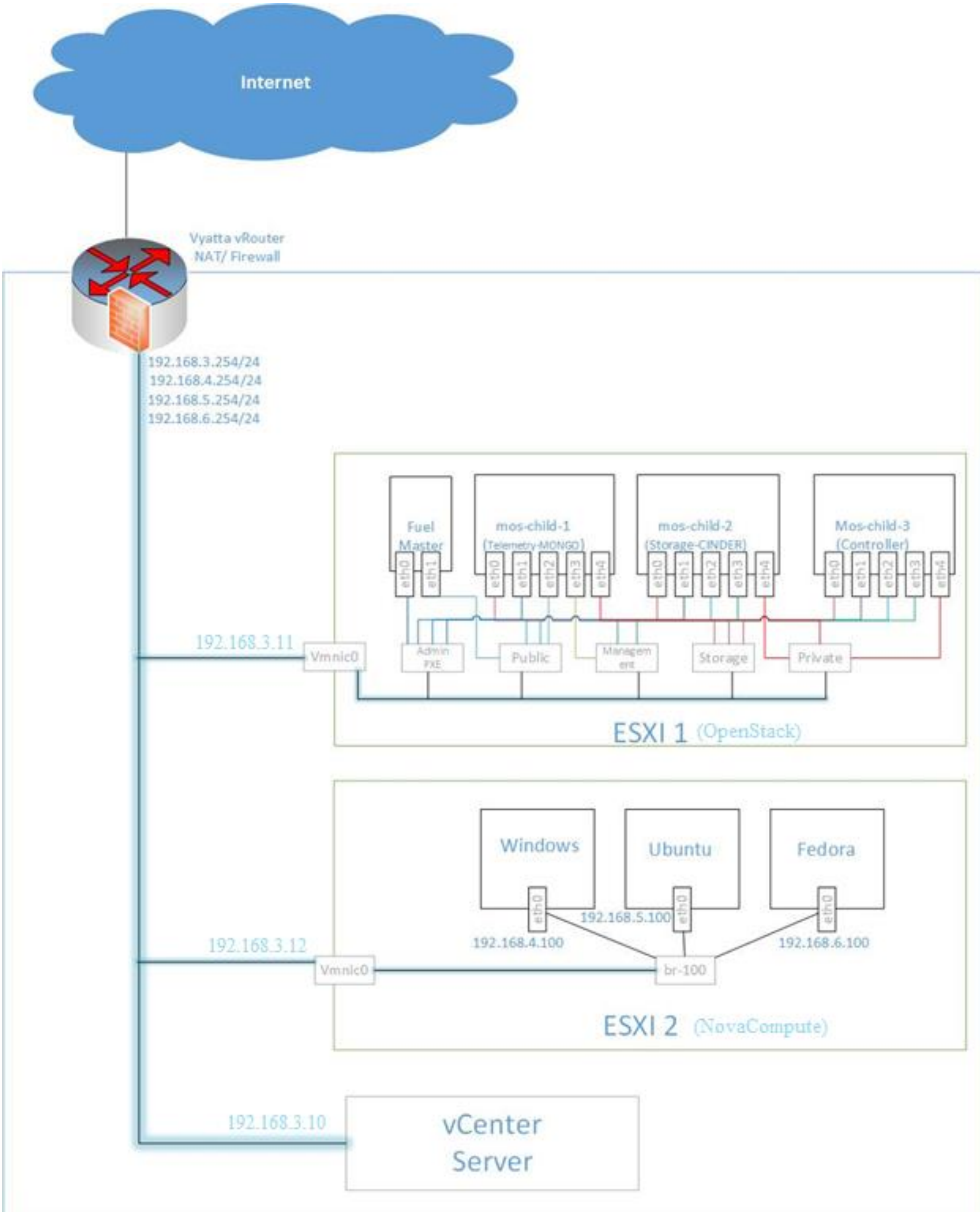


Figure 4.3 Physical and logical network connection diagram

5 IMPLEMENTING VIRTUAL INFRASTRUCTURE

5.1 VMWARE WORKSTATION INFRASTRUCTURE SETUP

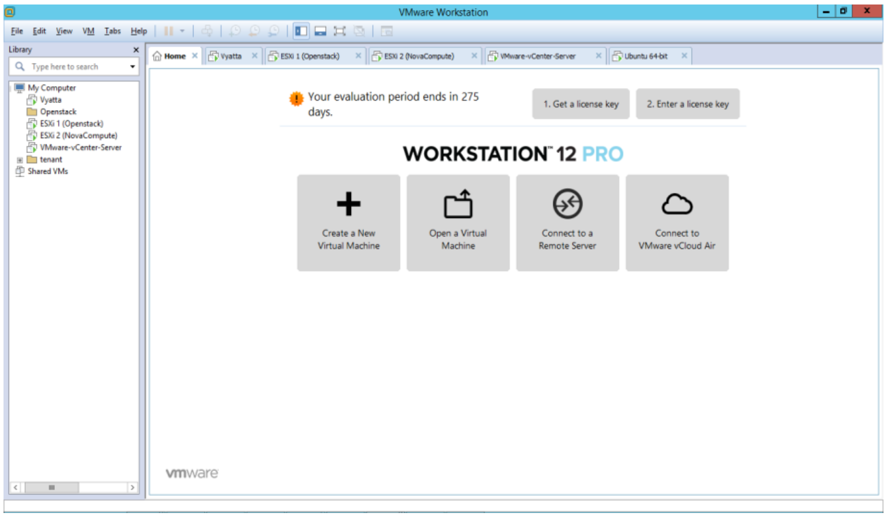


Figure 5.1 View of VMWare Workstation listing all the Virtual Machines

The infrastructure for this project is setup in a VMware Workstation 12 Pro installed on Windows Server 2012 system with 64 GB Ram, 1TB Hard disk drive and Intel Xeon 2.4GHz processor. Following four Virtual Machines are created for the project:

- Vyatta
- ESXi 1
- ESXi 2
- VMware-vCenter-Server

Also a new host only virtual network “vMnet3” is created on VMware Workstation.

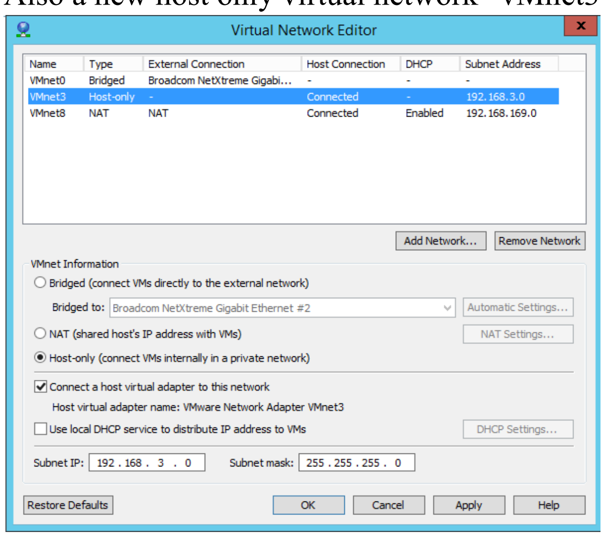


Figure 5.2 View of VMWare Workstation Virtual Networks

5.1.1 Vyatta

Vyatta VM boots using Brocade Vyatta 5410 vRouter ISO. The Vyatta vRouter performs following functions in this project:

- DHCP server (with Static MAC-IP Mapping)
- Network Address Translation (NAT)
- Firewall

Creation and Deployment of Vyatta VRouter

Step 1: Create a new Virtual Machine in VMware Workstation to boot from **Vyatta vRouter 5400 6.7 64-bit** ISO image, the 60-day trial version of which can be downloaded from the following link: http://www1.brocade.com/forms/jsp/Vyatta-download/index.jsp

Step 2: Go to Virtual Machine setting and set the memory for this virtual machine to be 512MB with one processor and 20GB hard disk space.

Step 3: Set Network Adapter to Bridged (Automatic) also add a second Network Adapter "Network Adapter 2" and set the network connection to VMnet3

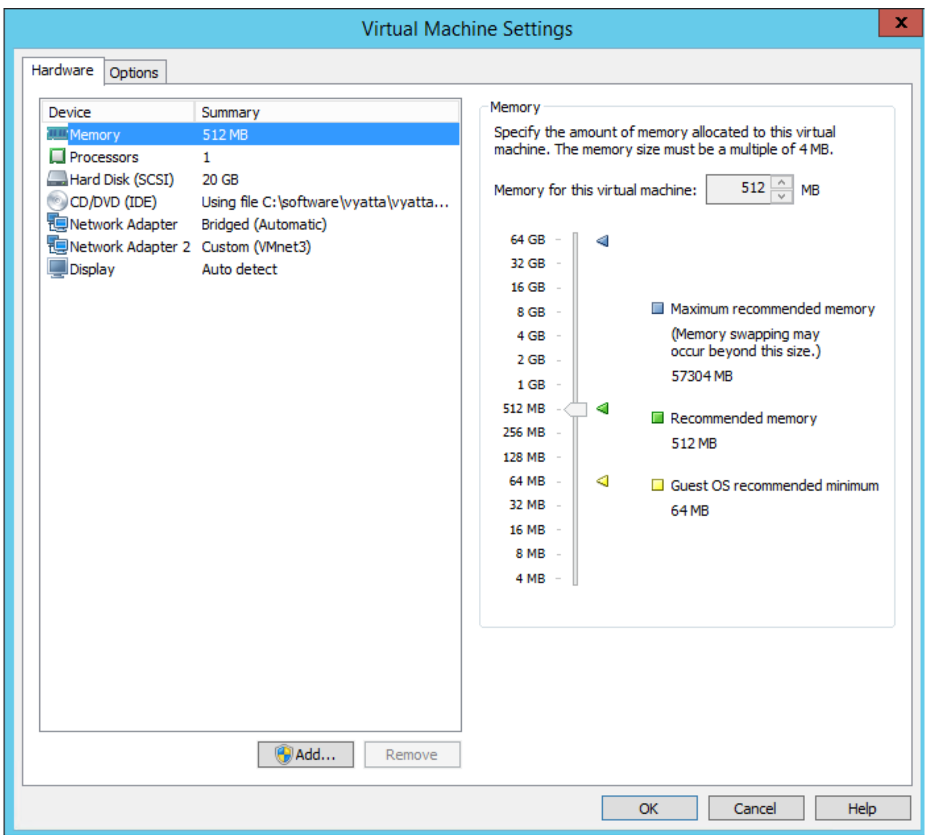


Figure 5.3 Vyatta VM settings on VMware Workstation

Step 4: Power up the virtual machine and after successful boot, install image on local hard drive using **install image** command.

Figure 5.4 View of Vyatta VM deployed on VMware Workstation

Step 5: After successfully installing image on local drive reboot the VM.

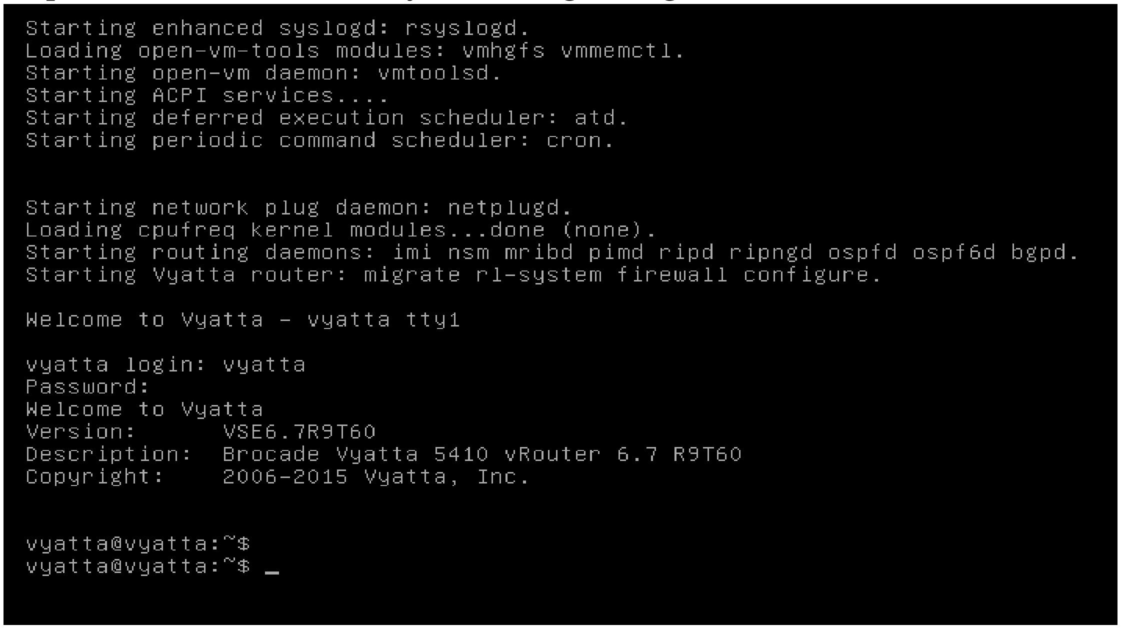


Figure 5.5 View of Vyatta vRouter VM after installing image on local drive

Step 6: Finally configure the interface connected to bridged interface to get IP from DHCP. For the other interface connected to VMnet 3 assign a static IP of 192.168.3.254 and enable DHCP server with default gateway being 192.168.3.254. Also enable NATing on the router setting the bridged interface as outside interface.

5.1.2 ESXi 1

ESXi 1 VM boots using VMware ESXI 5.5 ISO image. This ESXi hypervisor is setup to host the Mirantis OpenStack Fuel master and three child nodes.

Creation and Deployment of ESXi 1

Step 1: Create a new Virtual Machine in VMware Workstation to boot from VMware VSphere 5.5 ISO which can be downloaded from the following link:
https://my.vmware.com/group/vmware/evalcenter?lp=default&p=free-esxi5

Step 2: Go to Virtual Machine Setting and assign 32GB memory to the VM along with 8 processor cores. Also, assign 650GB hard disk space and set Network Adaptor to VMnet3.

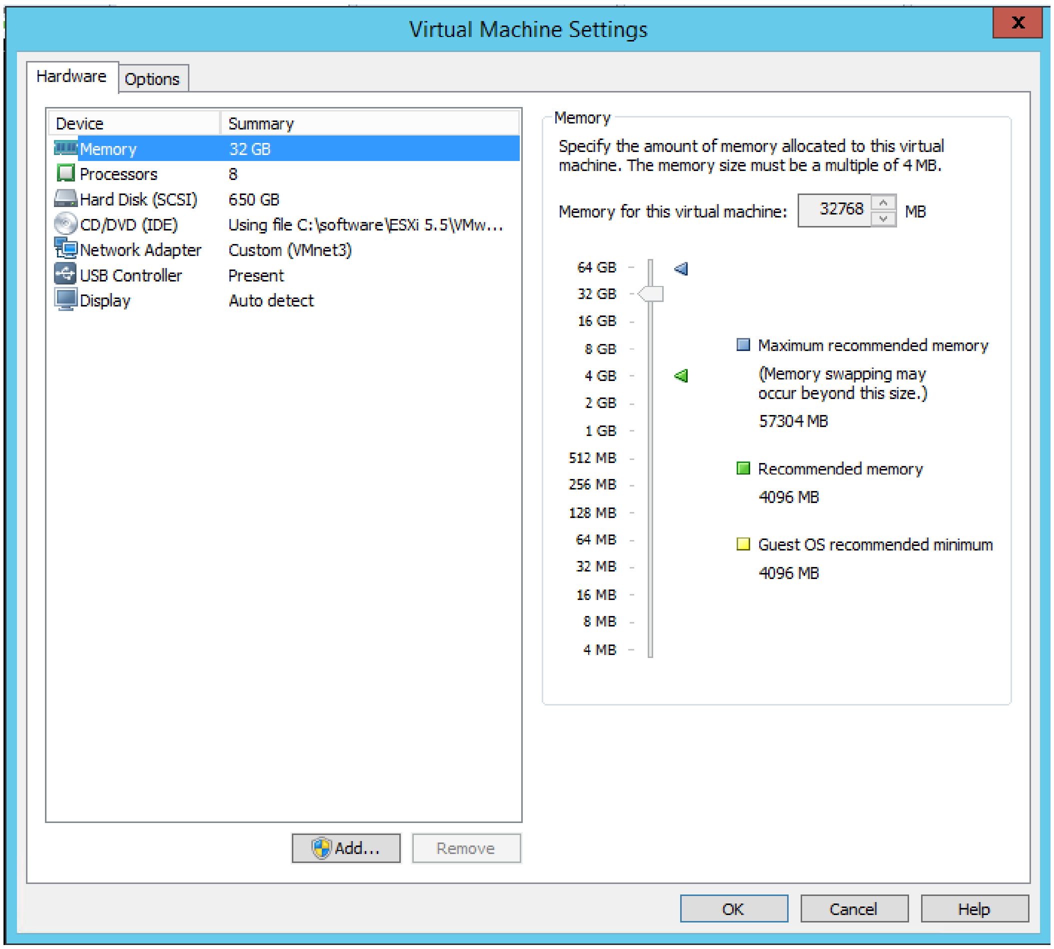


Figure 5.6 ESXI 1 VM setting on VMWare Workstation

Step 3: Power on the Virtual Machine and install ESXi on the local drive.

Step 4: After successful installation assign static IP address of 192.168.3.11 to this ESXi.

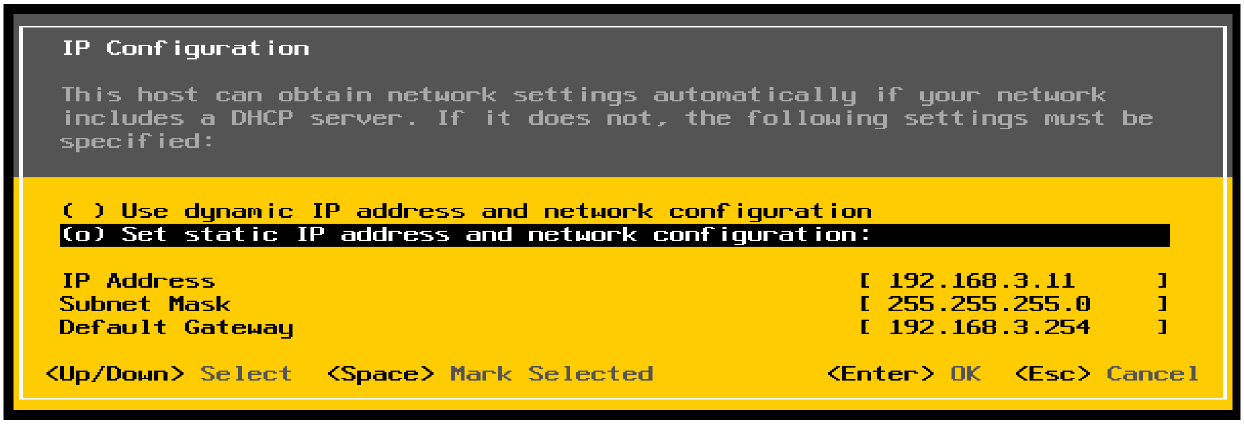


Figure 5.7 Set static IP to ESXi 1

Figure 5.8 View of ESXI 1 on VMWare workstation

5.1.3 ESXi 2

Similar to ESXI 1, ESXi 2 VM boots using VMware ESXI 5.5 ISO image. All the instances created using OpenStack are hosted in this ESXi 2 hypervisor.

Creation and Deployment of ESXi 2

All the steps in creation of ESXi 2 are similar to ESXi 1 except ESXi 2 is assigned with only 20GB memory and 200GB hard disk space.

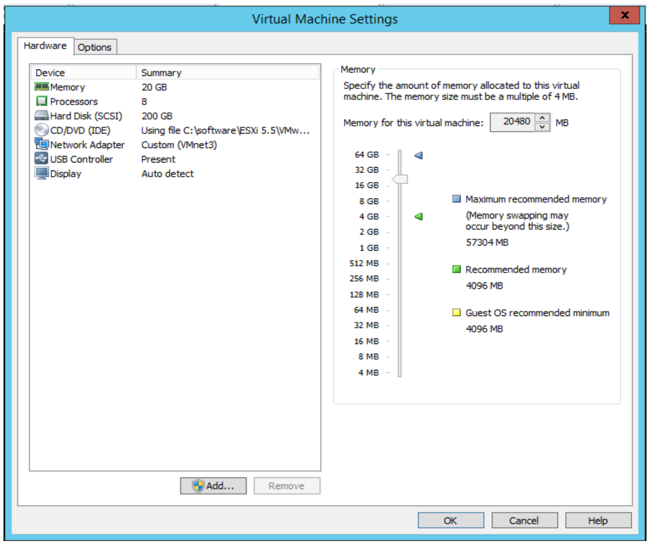


Figure 5.9 ESXI 2 VM setting on VMWare Workstation

Also, ESXi 2 is assigned with static IP 192.168.3.12

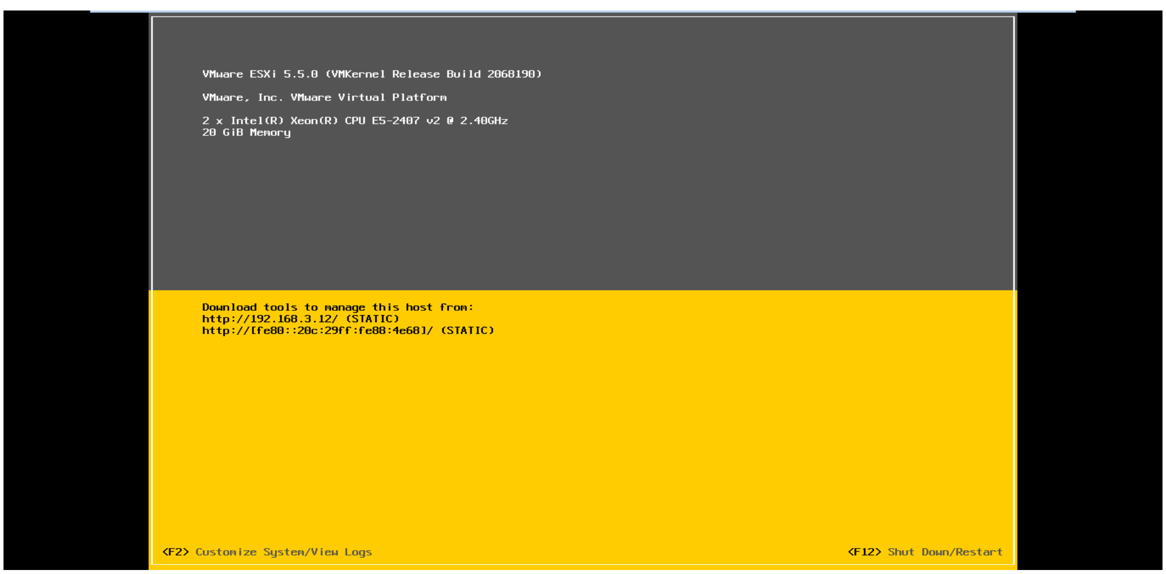


Figure 5.10 View of ESXI 2 on VMWare workstation

5.1.4 vCenter Server

vCenter Server VM boots using VMware-vCenter-Server-Appliance-5.5 OVF file. Both hypervisor ESXi 1 and ESXi 2 are managed using vCenter Server.

Creation and Deployment of vCenter Server

Step 1: Create a new Virtual Machine in VMware Workstation to boot from VMware-vCenter-Server-Appliance-5.5 ovf file.

Step 2: Go to virtual machine setting and set Network Adapter to VMnet 3

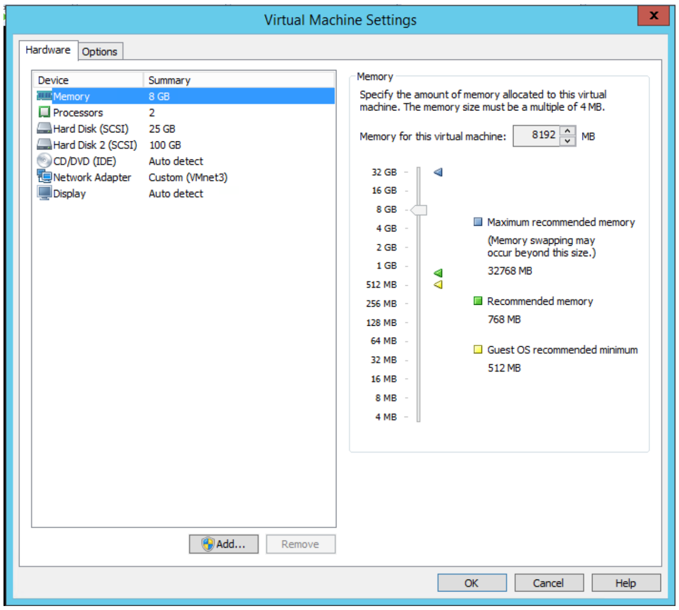


Figure 5.11 VCenter Server VM setting on VMWare Workstation

Step 3: Power on the virtual machine and install the vCenter Server.

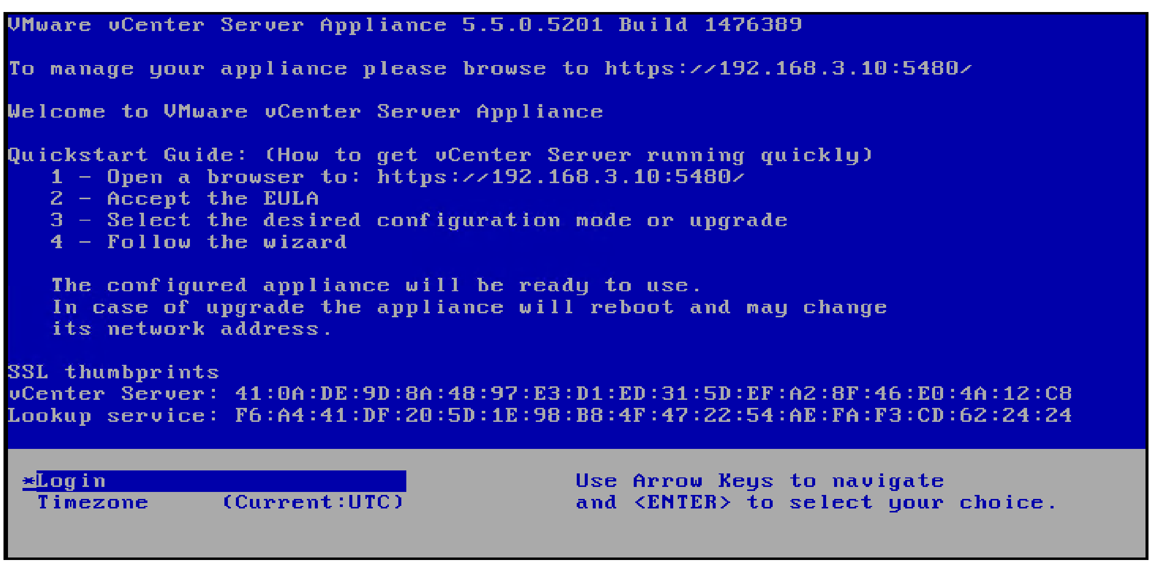


Figure 5.12View of VCenter Server on VMWare workstation

Step 4: After successful installation of vCenter Server on VMware workstation, open a web browser and navigate to vCenter Server Appliance webpage https://192.168.3.10:5480 (as shown in vCenter server console) and login using default username **root** and password **vmware.**

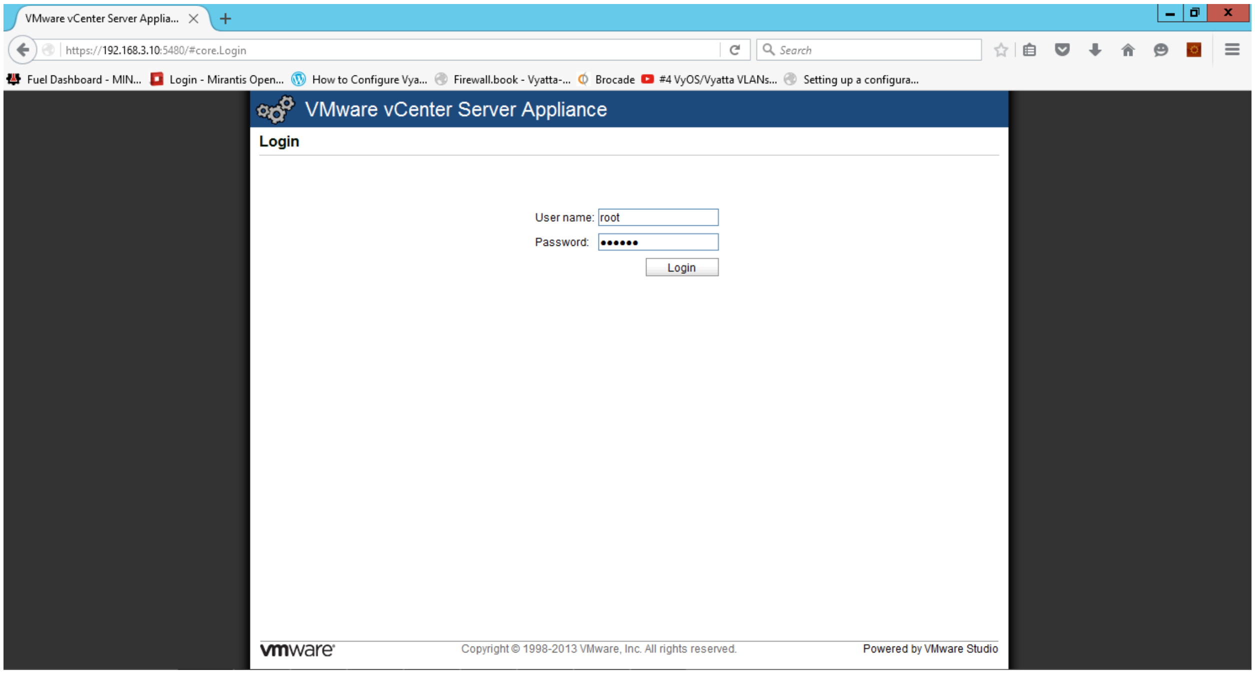


Figure 5.13 vCenter Server Setup login screen

Step 5: Start vCenter Server Setup. Select Set custom configuration as the configuration options, database type embedded, assign new password for user administrator@vSphere.local and set time synchronization as VMware Tools synchronization

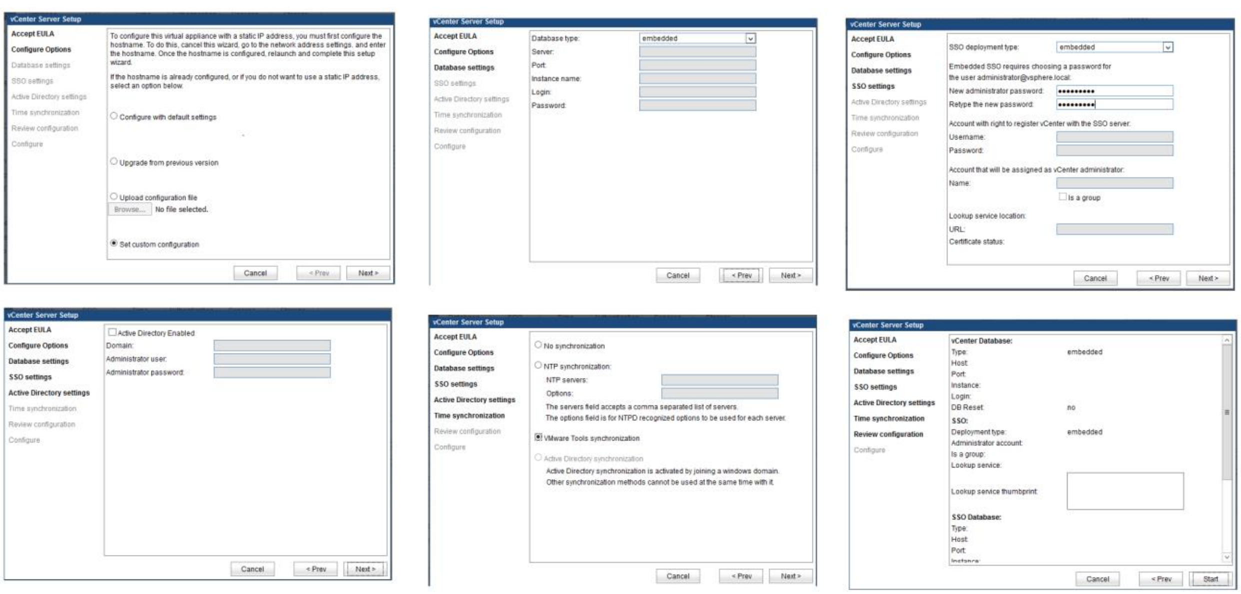


Figure 5.14 step by step vCenter Server Appliance Setup

5.2 VCENTER INFRASTRUCTURE SETUP

Step 1: Open VMware vSphere Client. Enter the vCenter IP address, username and password and click on login.

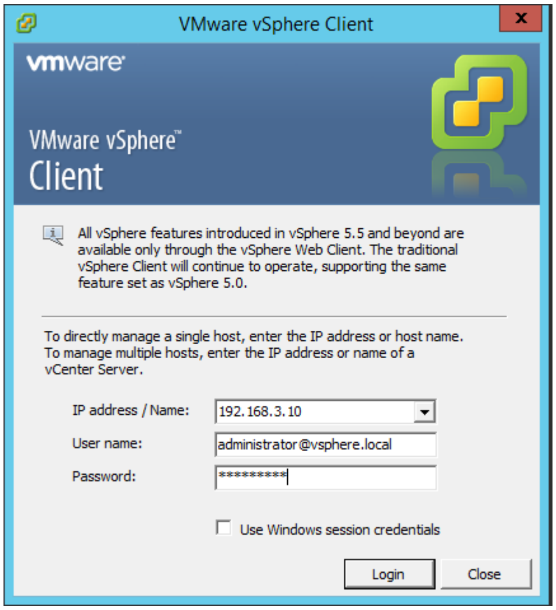


Figure 5.15 VMware vSphere client login

Step 2: Create a new datacenter and name it Datacenter.

Step 3: Create two new clusters OpenStack and NovaCompute and turn on vSphere DRS feature on both the clusters.

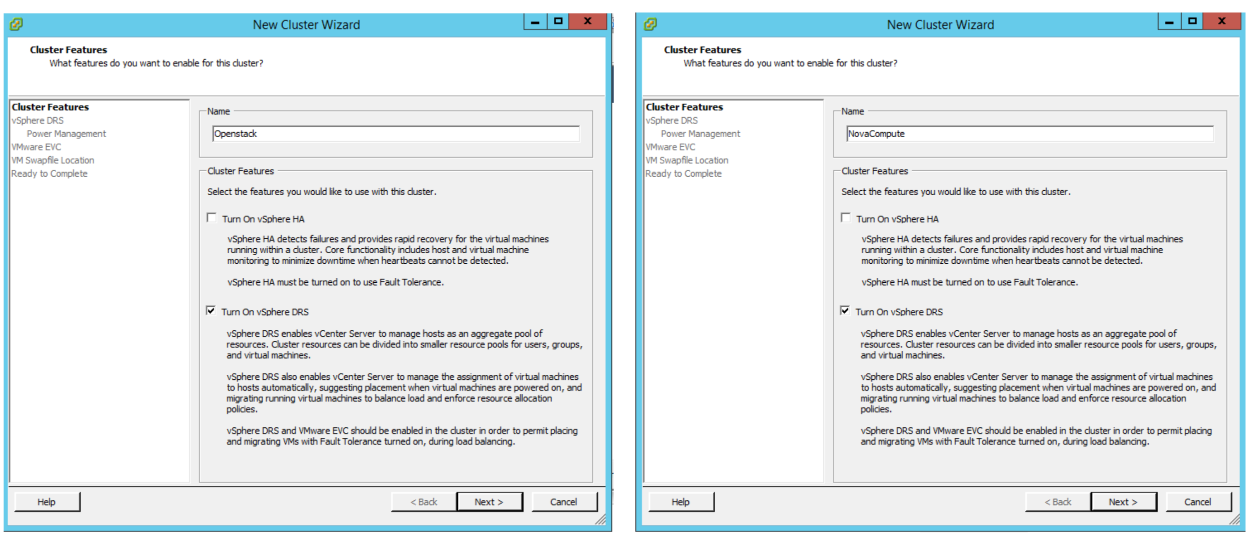


Figure 5.16 Creating clusters OpenStack and NovaCompute

Step 4: Add host ESXi 1 i.e. 192.168.3.11 to OpenStack cluster and ESXi 2 i.e. 192.168.3.12 to NovaCompute Cluster.

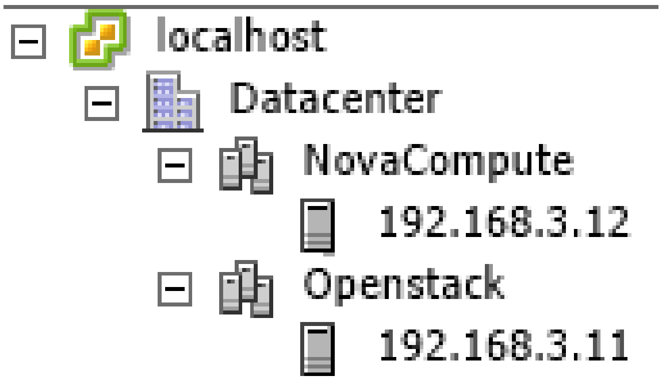


Figure 5.17 view of vCenter inventory after adding hosts to clusters

5.3 DEPLOYING MIRANTIS OPENSTACK 5.0.1 VAPP

A VMware Virtual Appliance (VApp) can be download from Mirantis website and imported to vCenter. This VApp offers a pre-configured deployment of virtual machines for Fuel, and OpenStack. The steps taken to deploy this VApp is described below:

5.3.1 Setting up network

Step 1: Create Virtual Machine Port Groups for deploying Mirantis OpenStack VApp on ESXi 1 and assign them with VLAN IDs as listed below:

Network	VLAN ID
Admin PXE	100
Management	101
Storage	102
Private	103
Public	0 (None)
Default	0 (None)

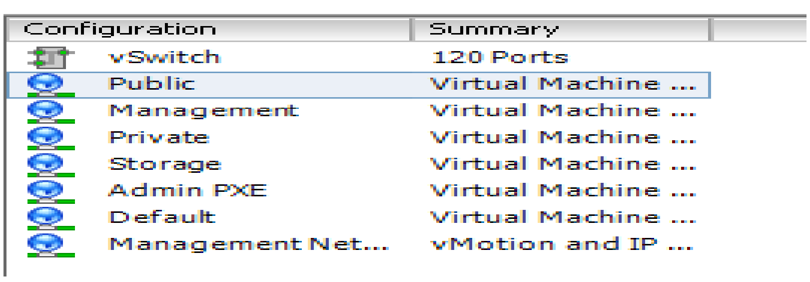


Figure 5.18 Virtual Machine Port Group created to deploy Mirantis OpenStack

Step 2: Configure all the port groups to accept Promiscuous mode, MAC Address Changes and Forged Transmits.

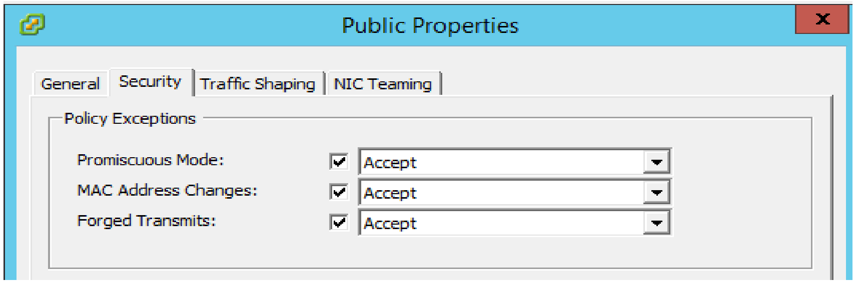


Figure 5.19 VM switch ports security setting

5.3.2 Importing Mirantis OpenStack

Step 1: Download VApp for Mirantis OpenStack from the following link: https://content.mirantis.com/vapp-mirantis-OpenStack-landing-page.html

Step 2: Select host 192.168.3.11 (ESXi 1) and from the file menu click on Deploy OVF template.

Step 3: Specify the location of Mirantis OpenStack VApp ova file and Map the network as shown in the snapshot below:

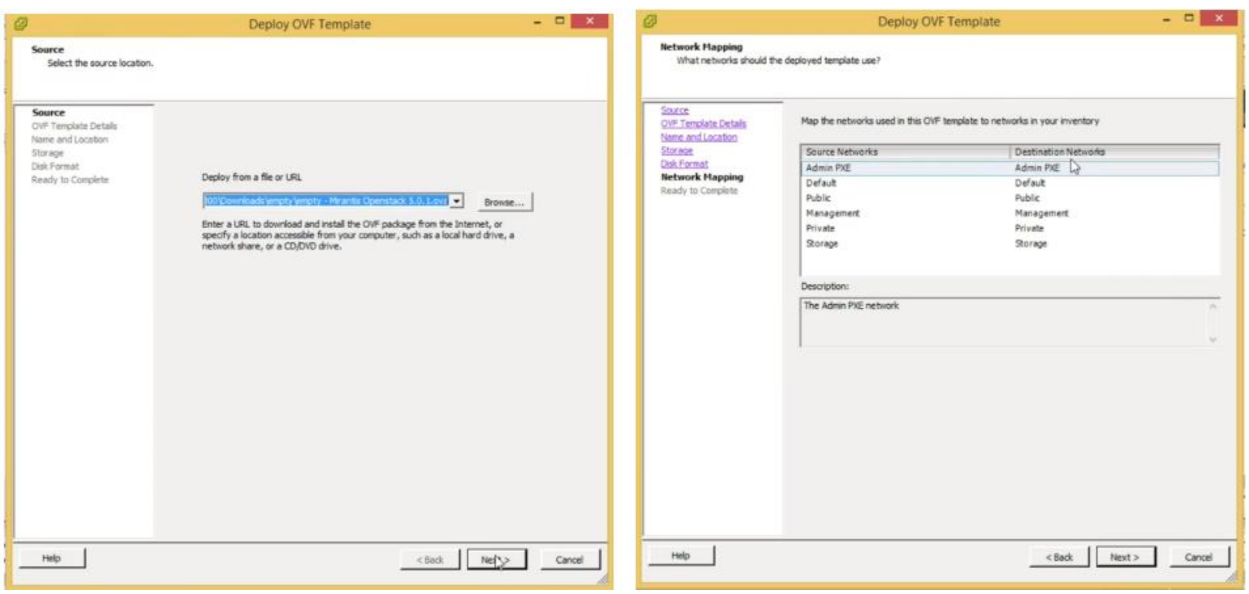


Figure 5.20 Deploy Mirantis OpenStack VApp

Step 4: After successful deployment of Mirantis OpenStack VApp, a Fuel Master VM and three child node VM are created as a part of the VApp as show in snapshot below:

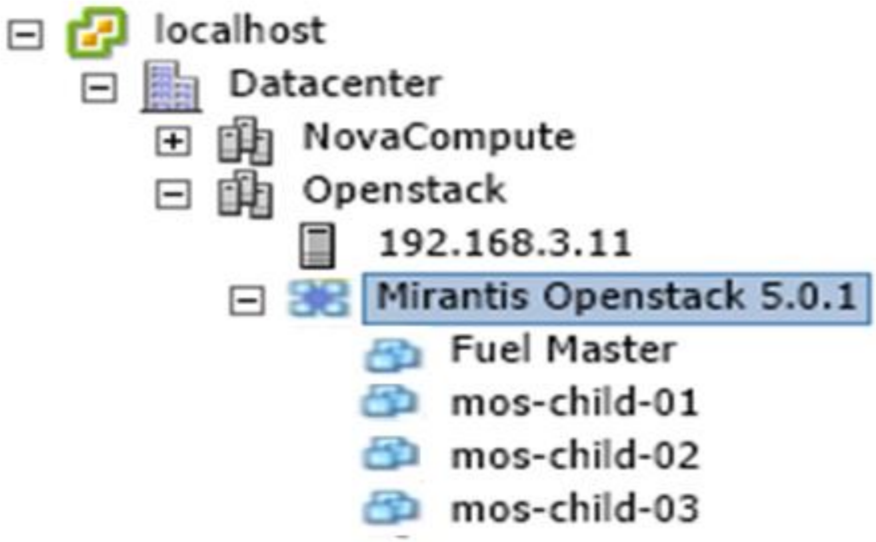


Figure 5.21 vCenter host inventory view after Deployment of Mirantis OpenStack VApp

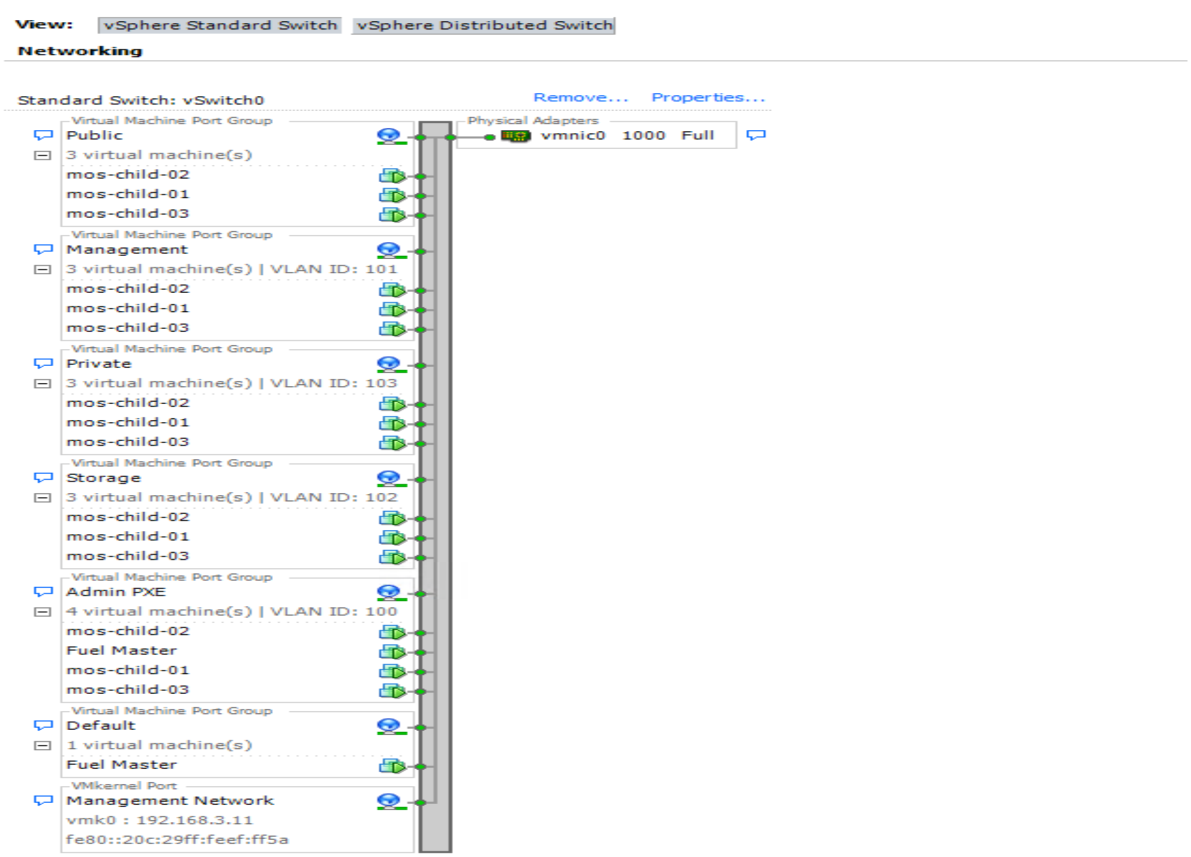


Figure 5.22 ESXi 1 network view after deploying Mirantis OpenStack VApp

5.3.3 Installing Fuel

Step 1: Download Mirantis OpenStack iso from the following link: https://software.mirantis.com/OpenStack-download-form/ In this project we are using Mirantis OpenStack 5.0.1 Icehouse ISO.

Step 2: Upload the ISO into datastore1 (ESXi 1's datastore).

Step 3: Power on Fuel Master VM and connect it to the Mirantis OpenStack 5.0.1 iso stored on the datastore and allow it to boot from the iso.

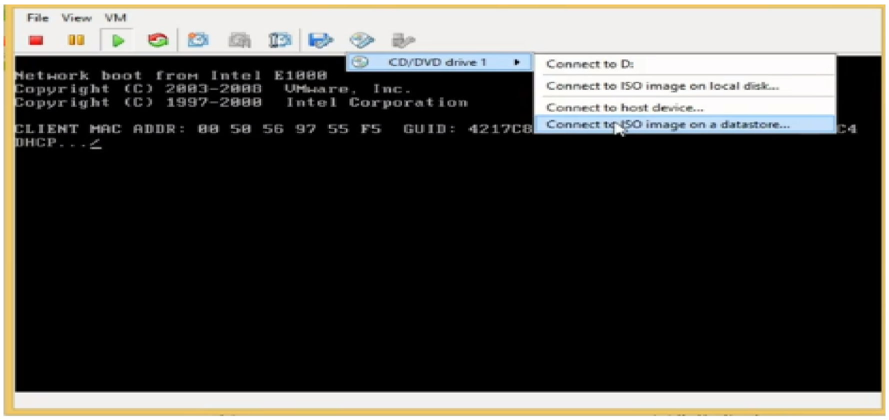


Figure 5.23 View of Fuel Master VM console

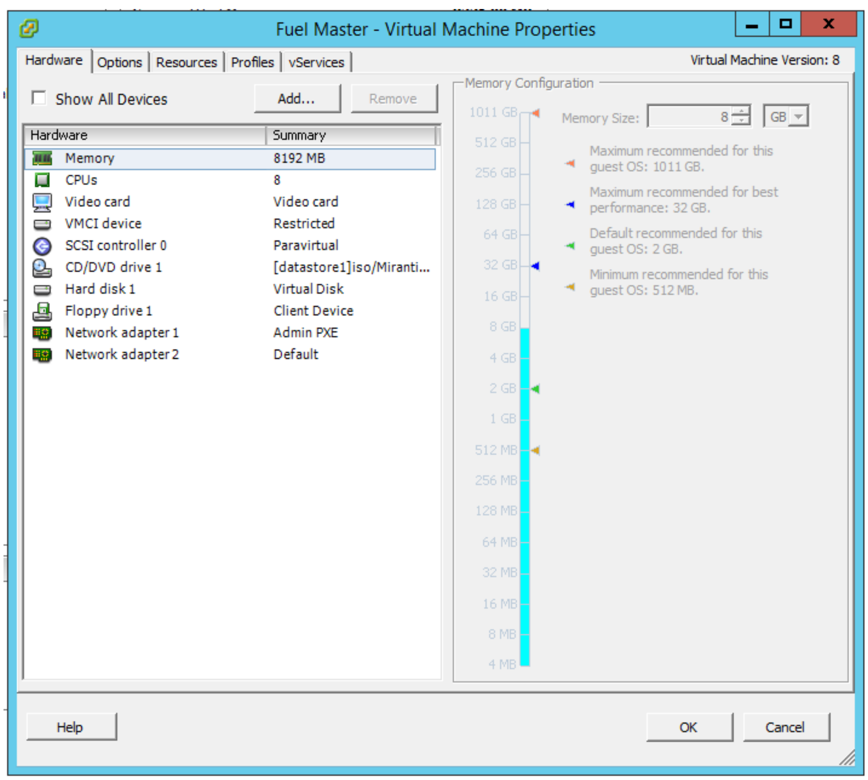


Figure 5.24 view of Fuel Master VM properties

Step 4: Change the **showmenu** option from no to yes and hit enter to continue Fuel Installation.

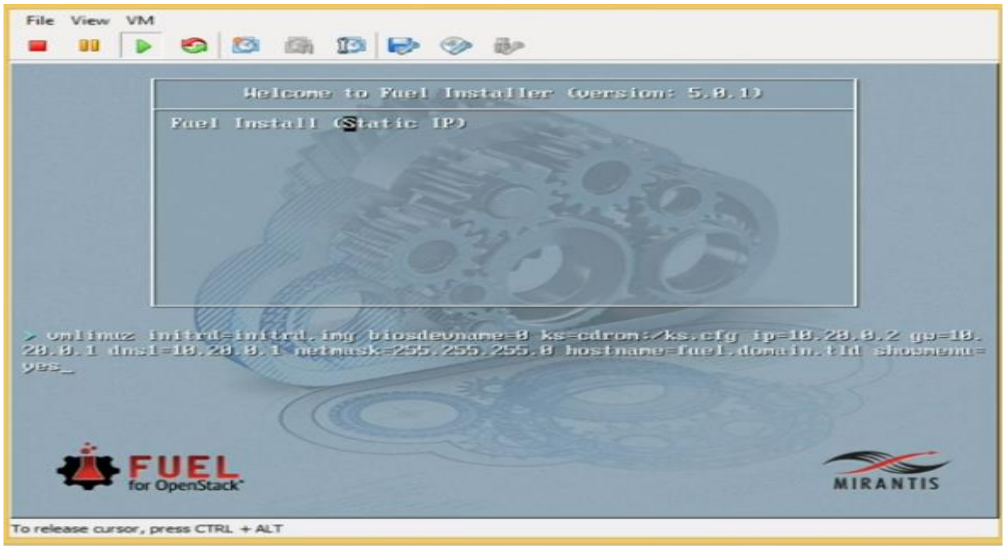


Figure 5.25 view of Fuel Master VM console immediately after booting up with Mirantis OpenStack iso

Step 5: In the configuration menu under the Network setup configure eth0 with default setting and set eth1 to configure using DHCP. Also provide proper DNS and NTP server parameter and then save and quit the configuration menu.

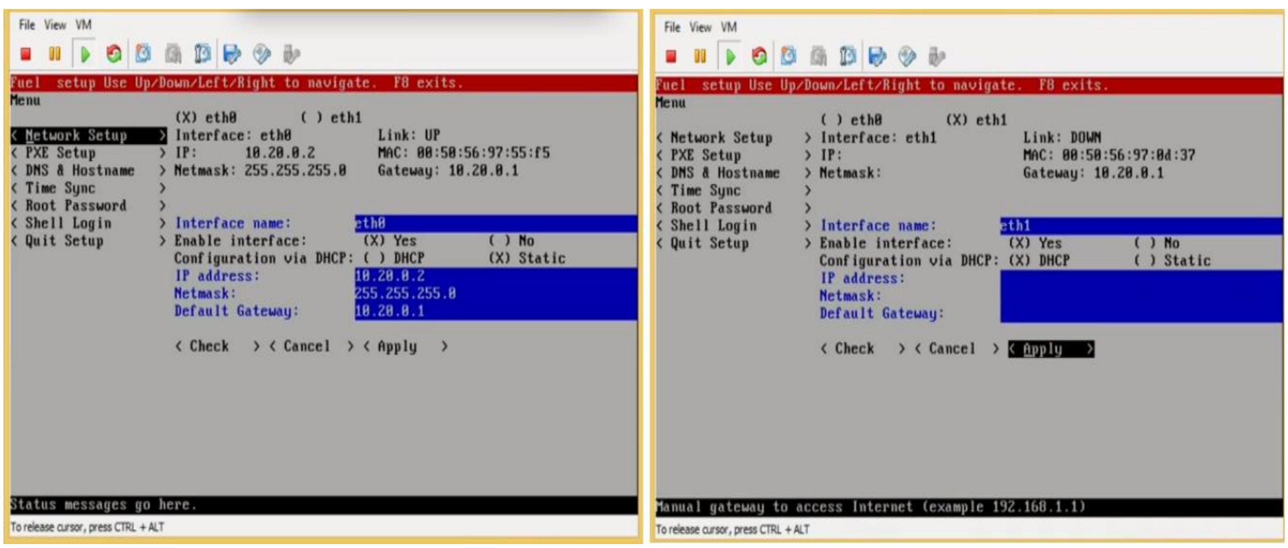


Figure 5.26 view of Fuel Master configuration setting

Setup 6: Wait for some time for the installation to finish. After the installation is completed power on the remaining three child VMs mos-child-1, mos-child-2 and mos-child-3 which will PXE boot from the Fuel Master.

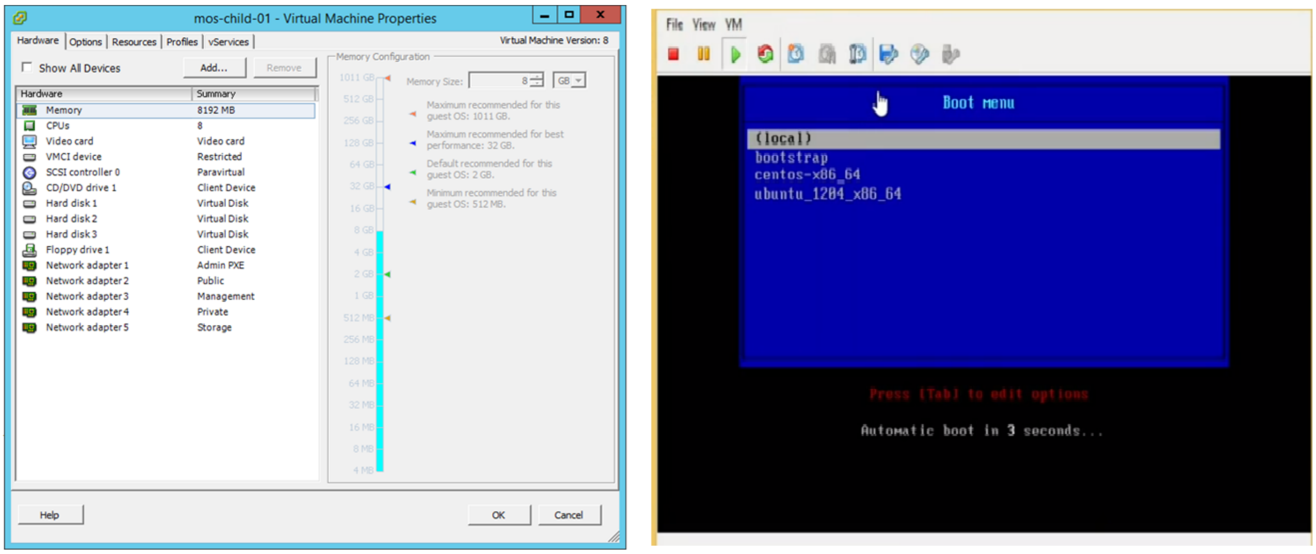


Figure 5.27 view of mos-child-1VM properties and console

5.3.4 Installing OpenStack

Step 1: Open Fuel master console and login as root. Display the ip address of the interfaces using **ip -4 a** command and note the ip address of eth1. In our case it is 192.168.3.15 as highlighted in the snapshot below.

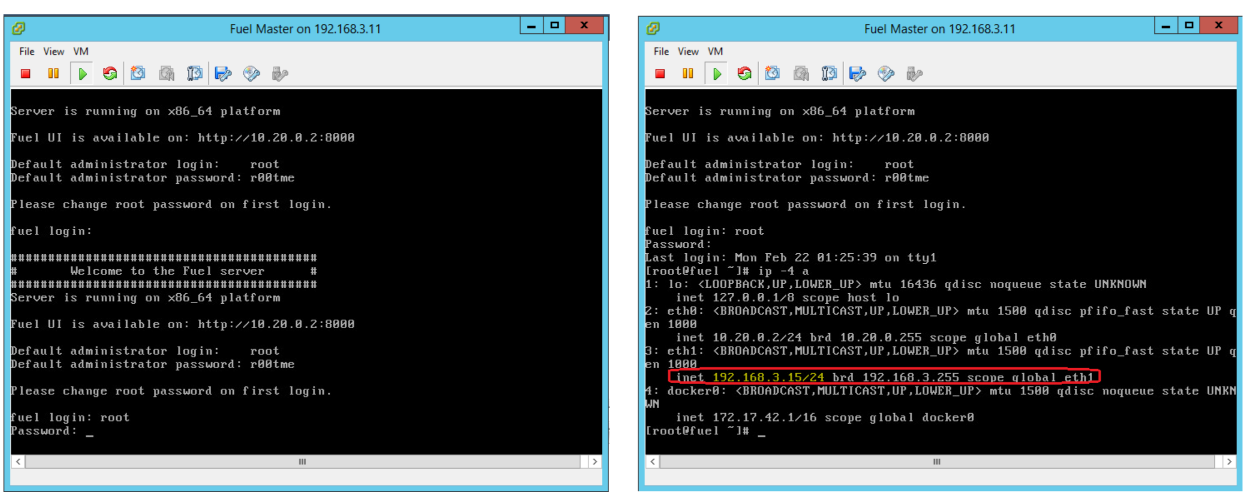


Figure 5.28 view Fuel Master console

Step 2: Open a web browser open link http://192.168.3.15:8000. This will open Fuel Dashboard as shown in snapshot below

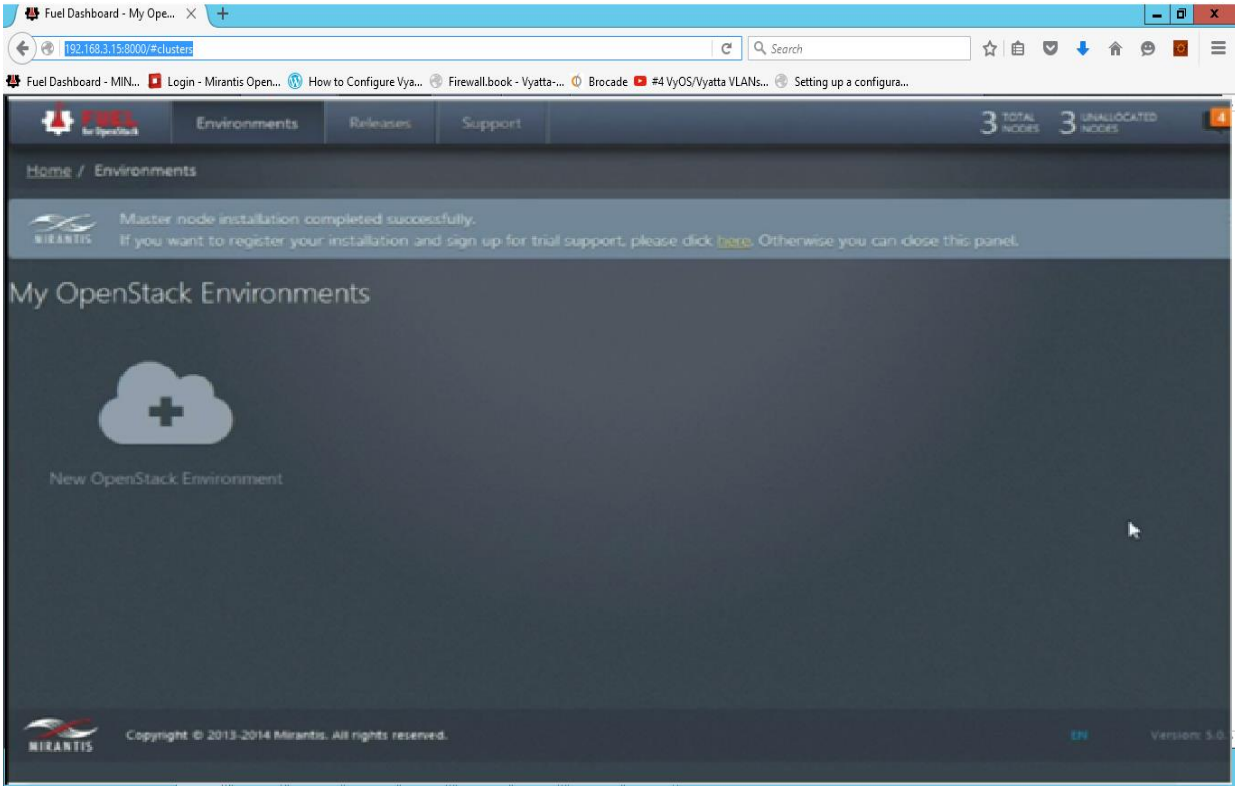
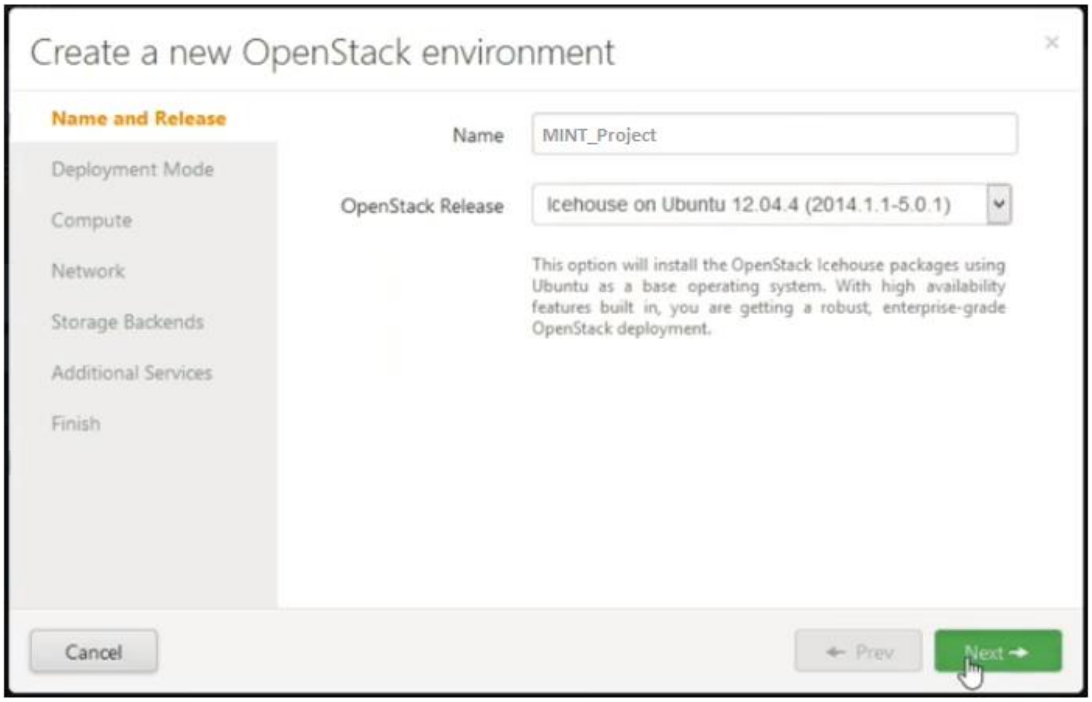


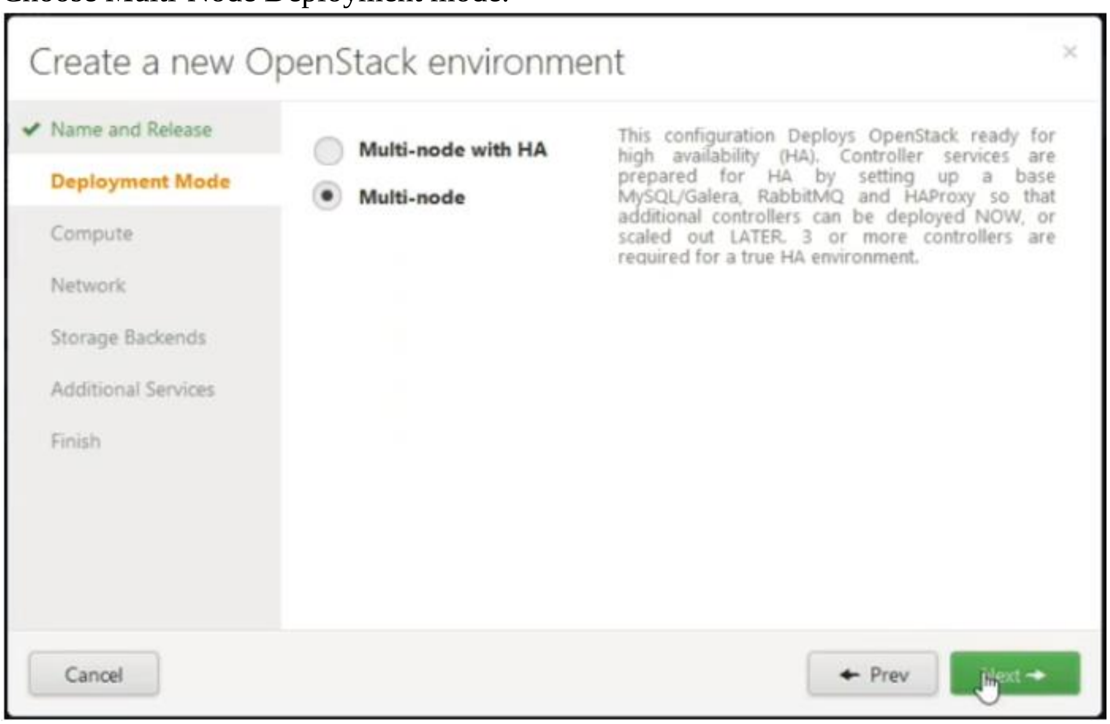
Figure 5.29 view of Fuel Dashboard

Step 3: Create a new OpenStack environment

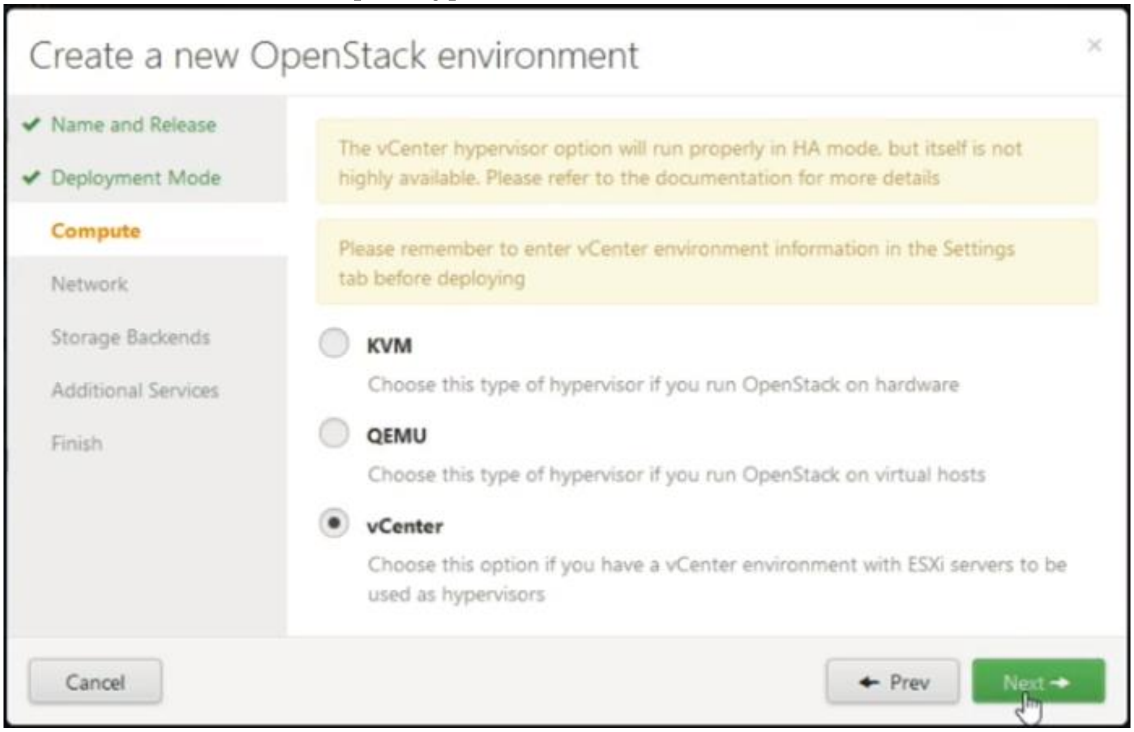
1. Name the new OpenStack environment “MINT_Project” and select Icehouse on Ubuntu 12.04.2 release to be installed.



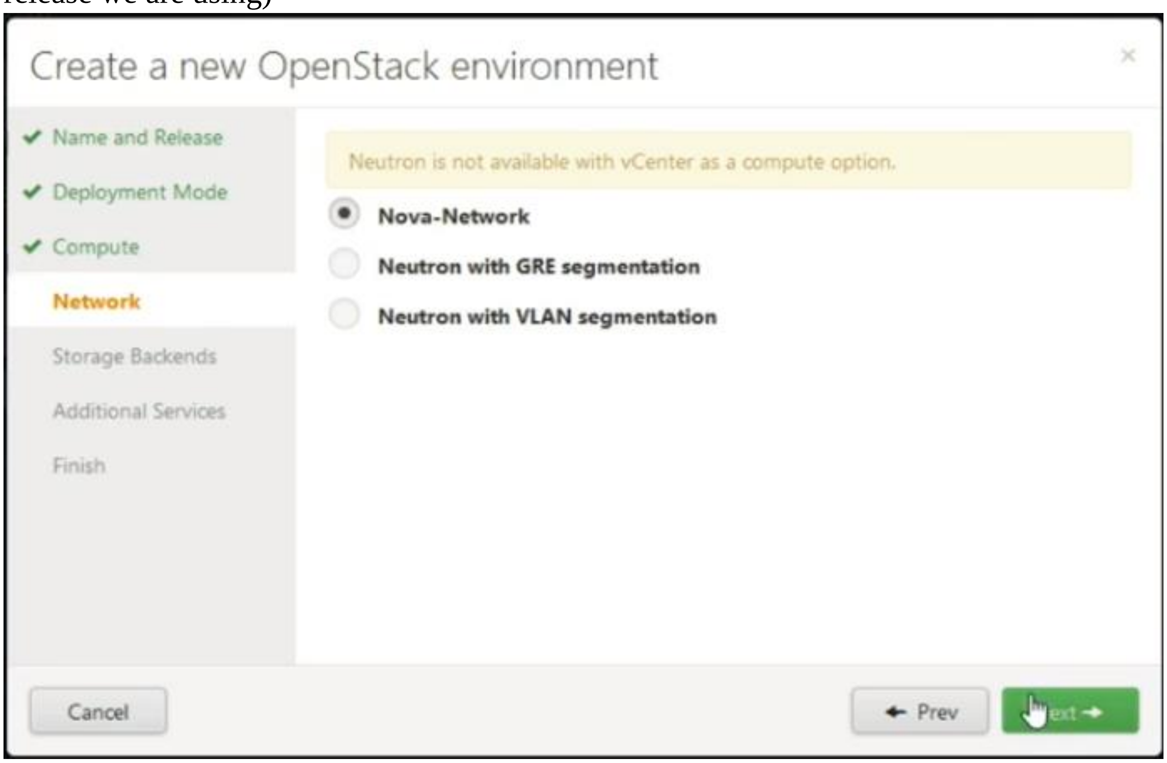
2. Choose Multi-Node Deployment mode.



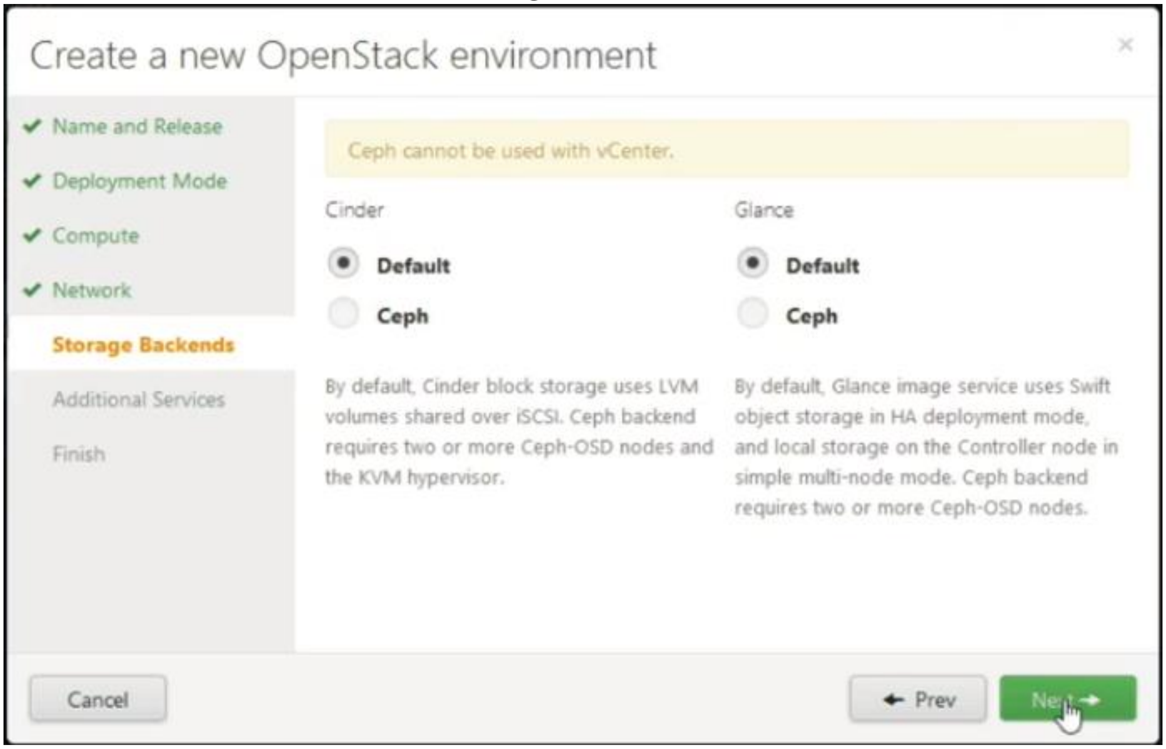
3. Choose vCenter as the compute hypervisor.



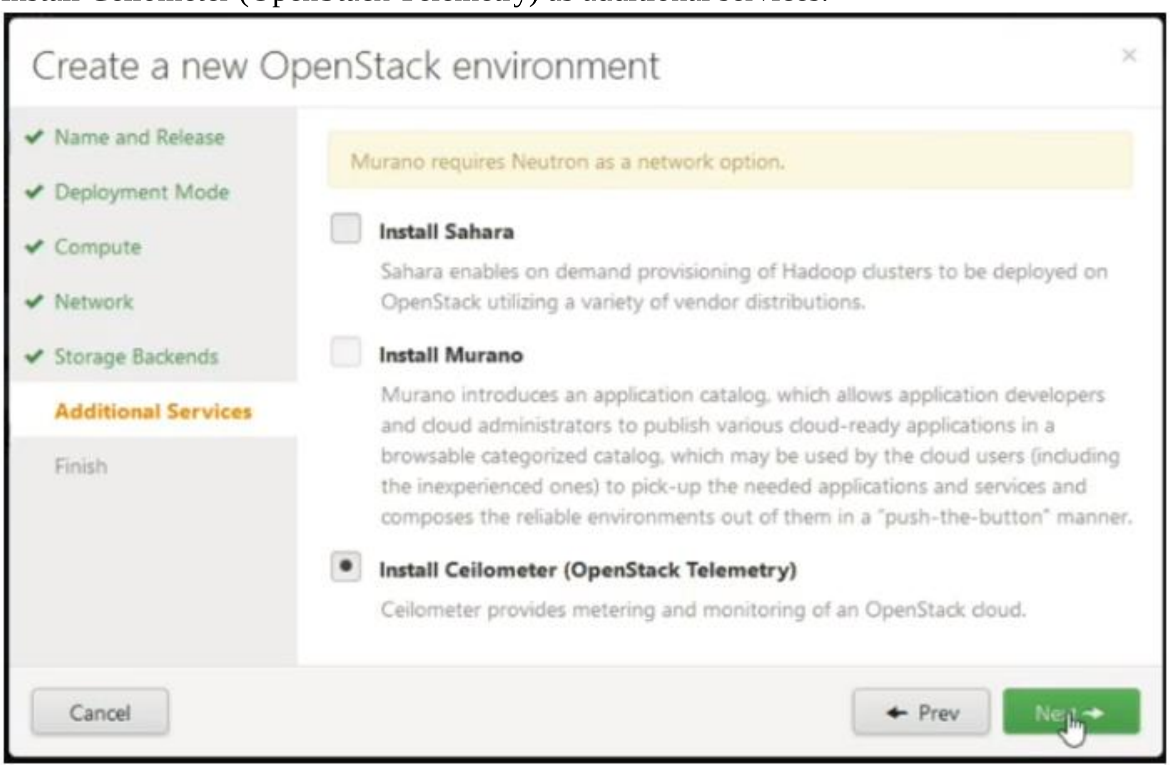
4. Choose Nova network (Neutron is not available with vCenter as compute option in the release we are using)



5. Choose Default Cinder and Glance Storage.



6. Install Celiometer (OpenStack Telemetry) as additional services.



Step 4: Configure OpenStack Environment

1. Add and assign role to the nodes.
    - First node is assigned the role of controller
    - Second node is assigned the role of Storage – Cinder
    - Third node is assigned the role of Telemetry - MongoDB

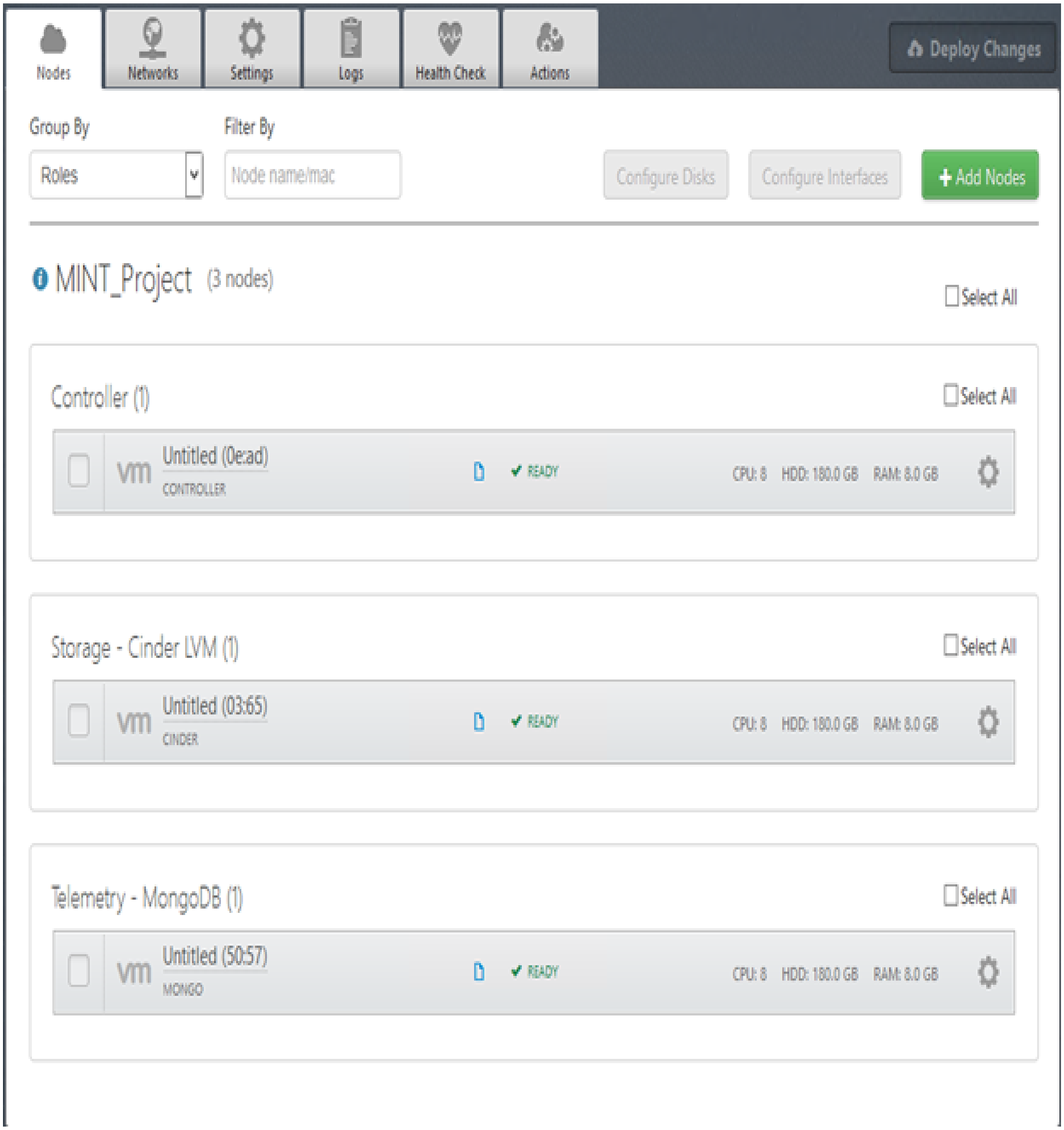


Figure 5.30 Adding and assigning role to nodes

2. Configure interfaces on 3 nodes
   - Assign eth0 to storage network
   - Assign eth1 to Admin(PXE) network
   - Assign eth2 to Public network
   - Assign eth3 to Management network
   - Assign eth4 to VM(Fixed) network

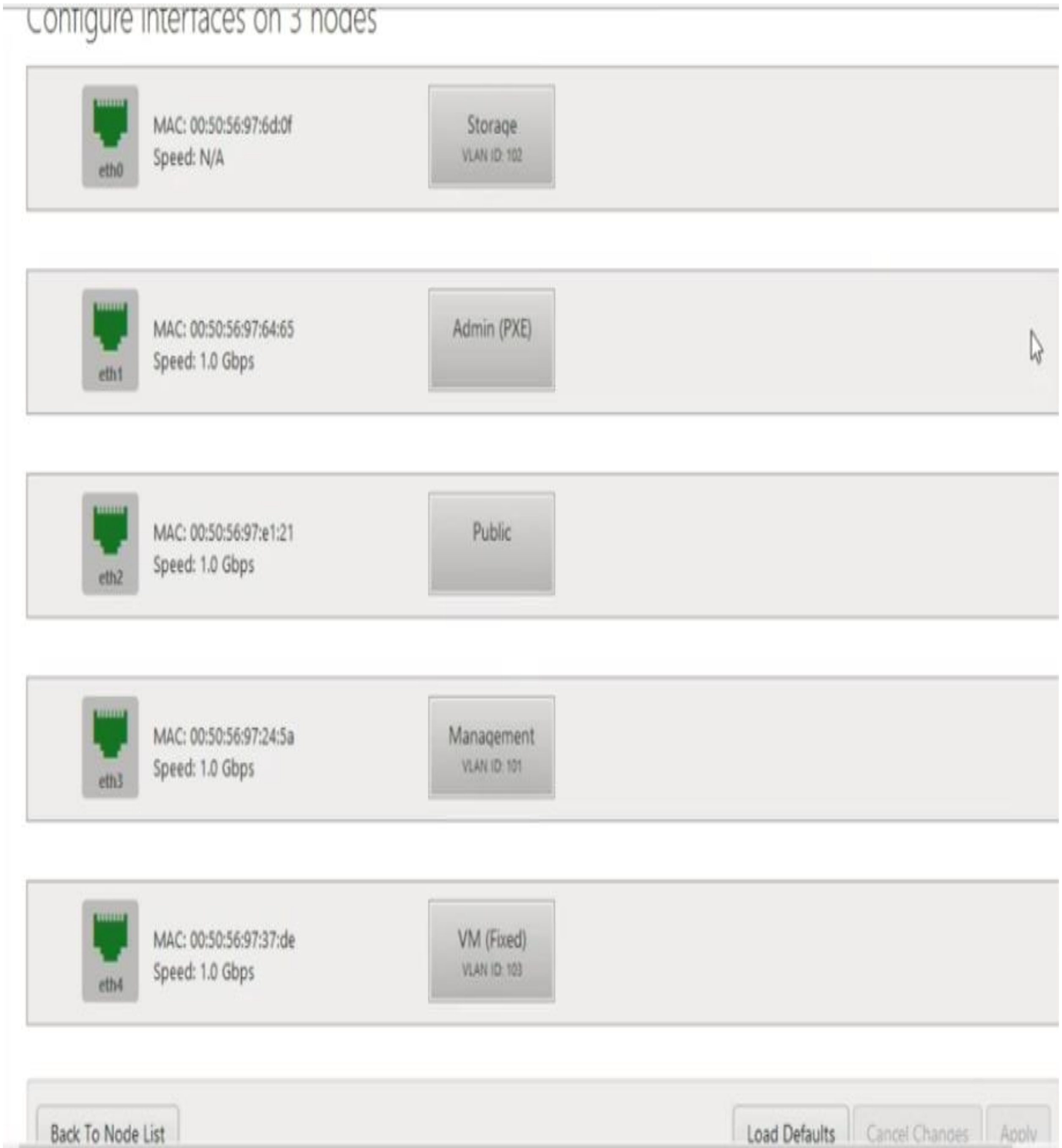


Figure 5.31 Configure interfaces on 3 nodes

3. Configure Network Setting
   - Select FlatDHCP Manager
   - Assign IP address range for Public, Management, Storage and Nova-network Configuration as shown in the snapshot below.
   - Click Verify Networks to check the configuration and
   - Save the network Settings

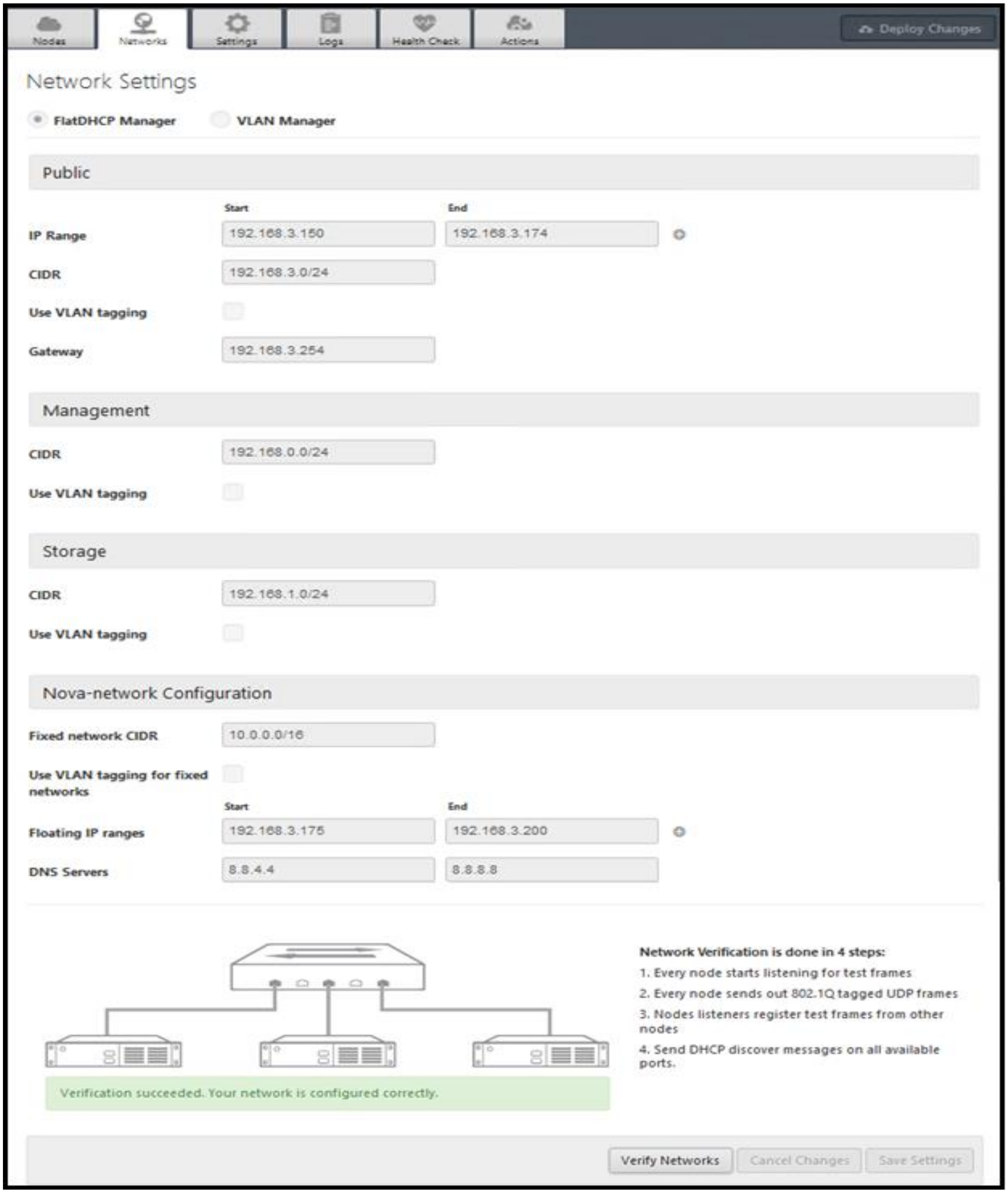


Figure 5.32 OpenStack Network Setting

4. Configure OpenStack Settings
    - Specify access credentials (Administrator username and password)
    - Specify vCenter ip, username, password and cluster
    - Select vCenter as hypervisor type
    - Click on deploy changes

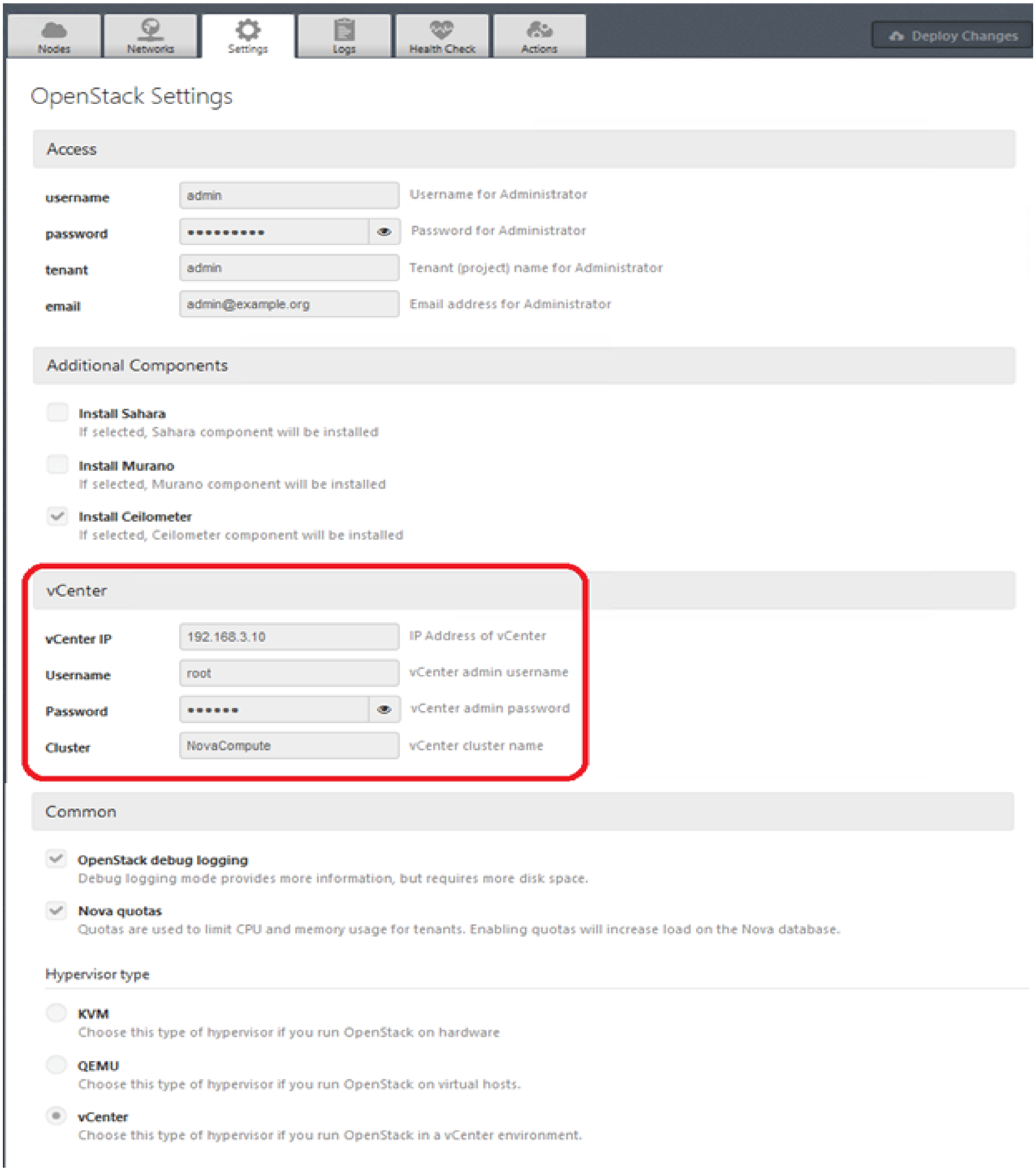


Figure 5.33 OpenStack Setting

5.4 OpenStack Dashboard

OpenStack Horizon Dashboard can be accessed using web URL after the successful deployment of the OpenStack environment.

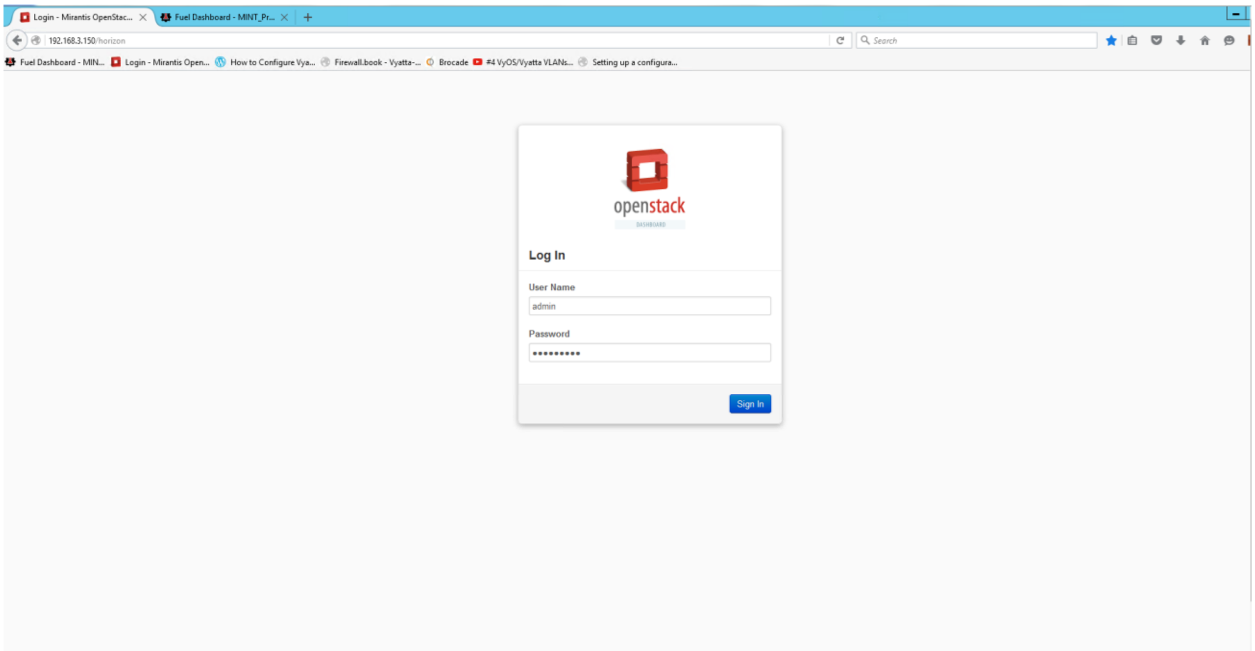


Figure 5.34 view of OpenStack Horizon Dashboard login screen

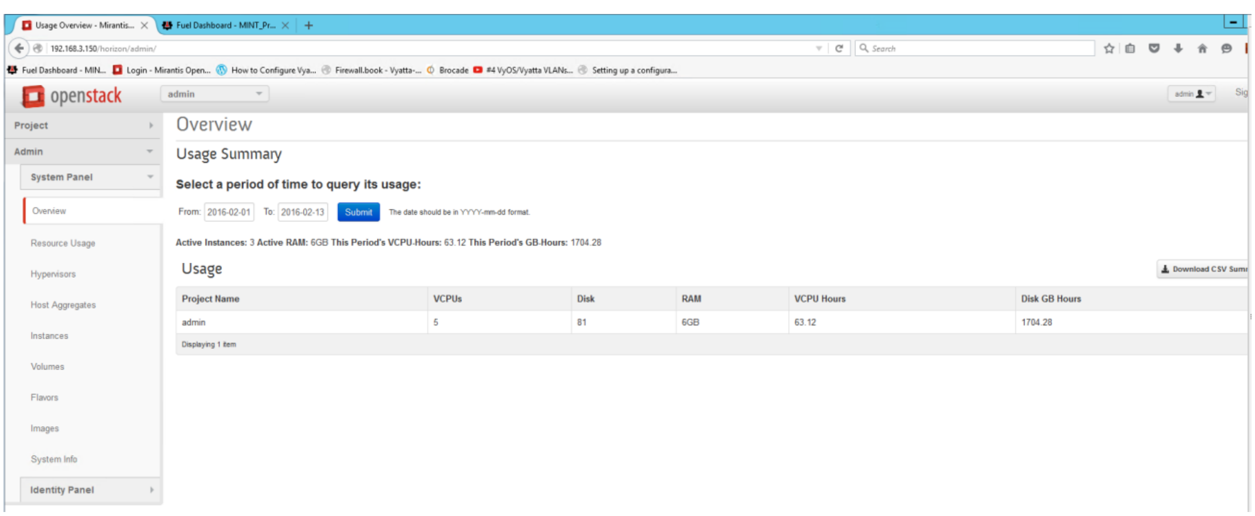


Figure 5.35 OpenStack Horizon Dashboard Overview

6 LAB EXPERIMENT DEMO WITH RESULT

6.1 CREATING INSTANCES USING HEAT ORCHESTRATION

As a part of this project three instances are orchestrated which boots Windows, Ubuntu and Fedora operating system respectively. The process of orchestrating instances is described below:

6.1.1 Creating Image

To orchestrate an instance first of all an image of the operating system should be created. The steps involved in creating an image is explained below:

Step 1: Login to the horizon dashboard
Step 2: Select the appropriate project from the drop down menu at the top left
Step 3: On the Project tab, open the Compute tab and click Images category
Step 4: Click Create Image (The Create an Image dialog box appears)
Step 5: Provide proper name, description of the image and choose image file as the image source.
Step6: point the location of the ISO image of the and choose format as ISO.
Step7: specify the architecture type, minimum disk and RAM required and make this image a publicly available image by checking the public box.

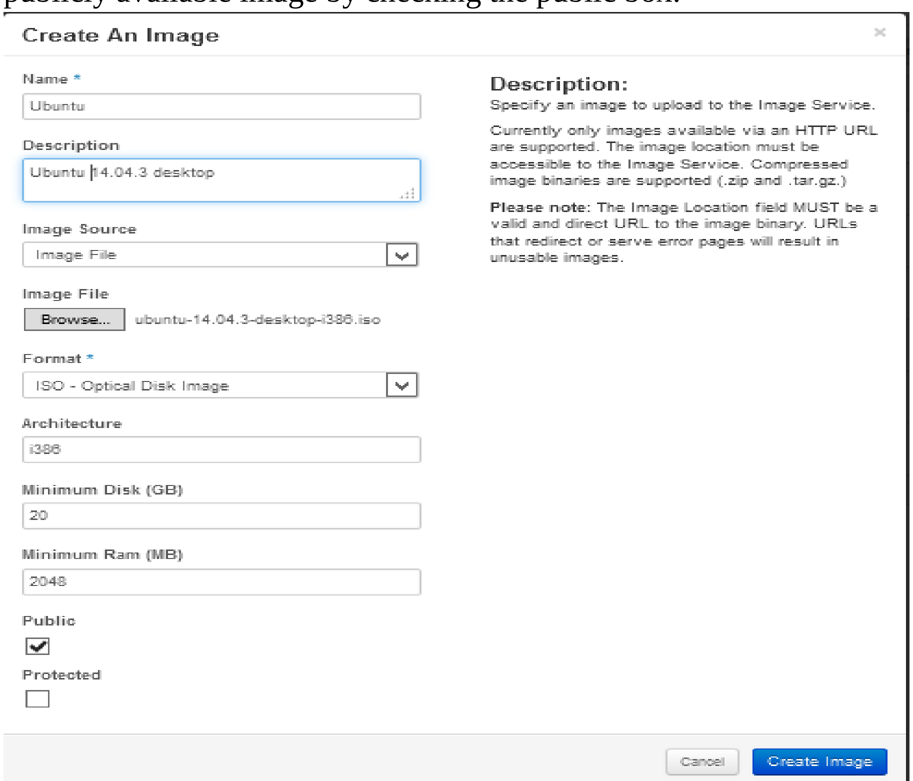


Figure 6.1 Create An Image dialog box

Step 6: Finally click Create Image

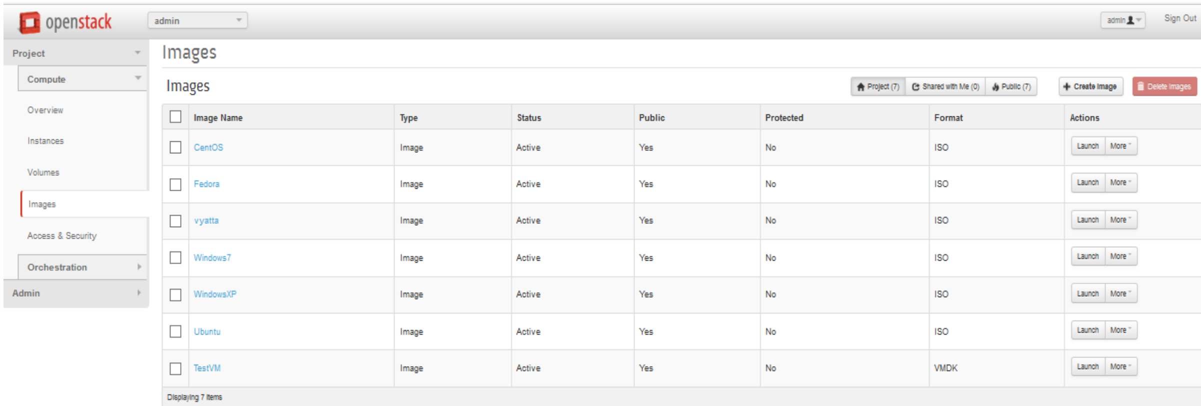


Figure 6.2 View of images tab after creating multiple images

6.1.2 Flavors

Virtual hardware templates are called "flavors" in OpenStack. They define sizes for RAM, disk, number of cores, and so on. The default install provides five flavors as shown in the snapshot below.

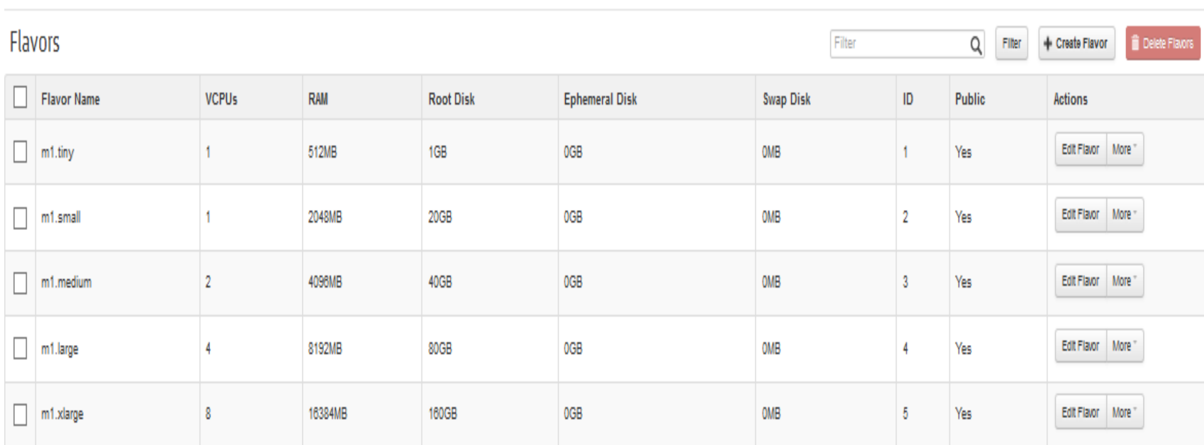


Figure 6.3 view of default OpenStack Flavors

Depending upon our physical hardware constrain we choose m1.small flavor for the three instances we created.

6.1.3 Orchestration

Heat Orchestration used to configure and deploy resource in stack. Heat Orchestration Template (HOT) is used to in this project to define Heat stack.

Step by step creation of stacks using Horizon dashboard:

Step 1: On the Project tab, open the Orchestration tab and click Stacks

Step 2: Click on Launch Stack, we see a dialog that lets us pull in a template by URL, upload it from a file, or simply cut and paste it into an editable dialog.

Step 3: Select Direct Input template source and paste the heat template into the template data field and click next.

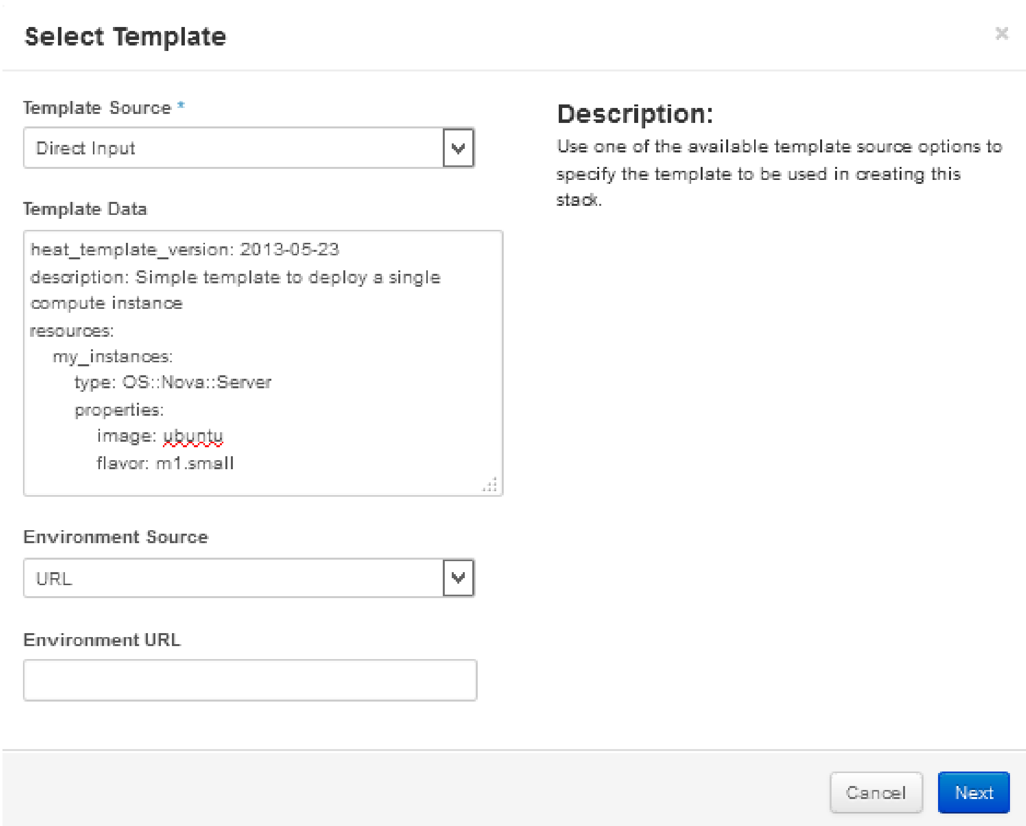


Figure 6.4 Select Template dialogue box

The heat template for creating a single instance is listed below:

Heat template to create an Ubuntu instance:

```
heat_template_version: 2013-05-23
description: Simple template to deploy a single compute instance
resources:
  my_instances:
    type: OS::Nova::Server
    properties:
      image: ubuntu
      flavor: m1.small
```

Step 4: Give a name to the stack and provide admin password and click on launch.

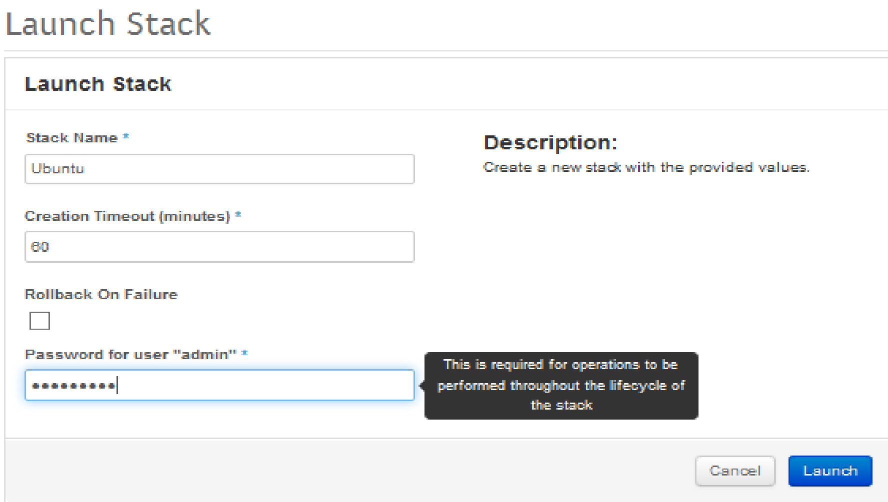


Figure 6.5 Launch Stack dialogue box

Similarly, other two stacks WindowsXP and Fedora are created. The heat template for crating WindowsXP and Fedora are listed below

Windows XP:

```
heat_template_version: 2013-05-23
description: Simple template to deploy a single compute instance
resources:
  my_instances:
    type: OS::Nova::Server
    properties:
      image: WindowsXP
      flavor: m1.small
```

Fedora:

```
heat_template_version: 2013-05-23
description: Simple template to deploy a single compute instance
resources:
  my_instances:
    type: OS::Nova::Server
    properties:
      image: Fedora
      flavor: m1.small
```

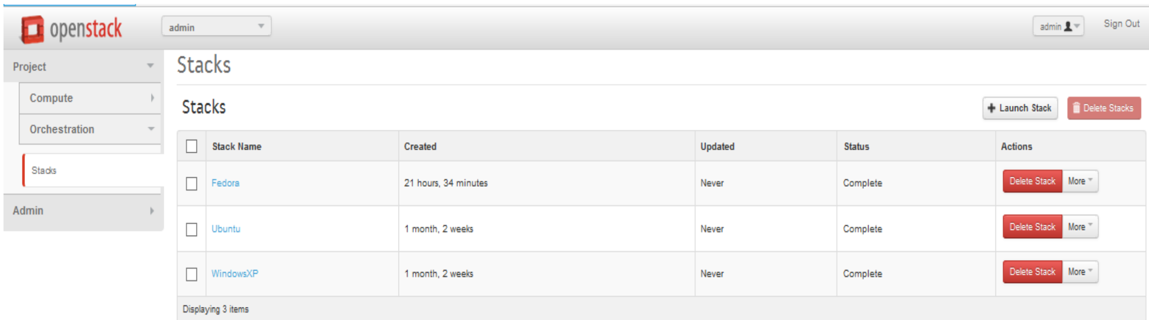
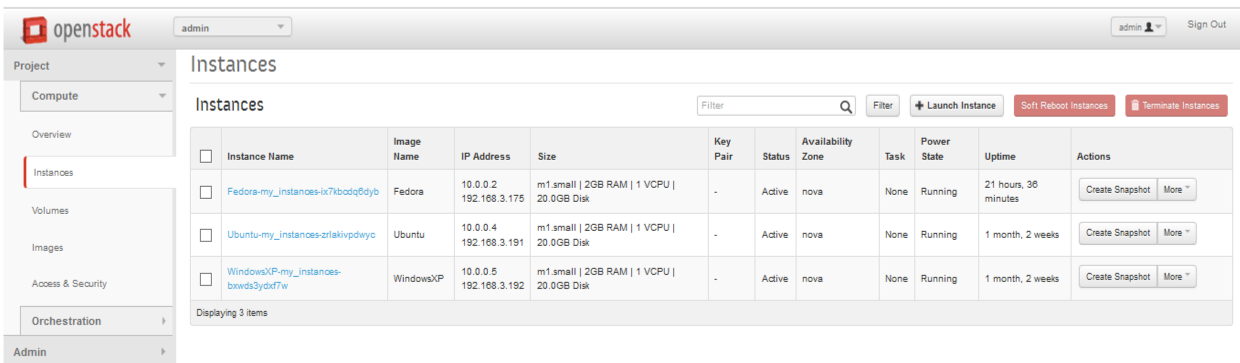


Figure 6.6 view of Stacks

The creation of three new stacks create three new instances which can be seen at the instances tab



WindowsXP Instance:

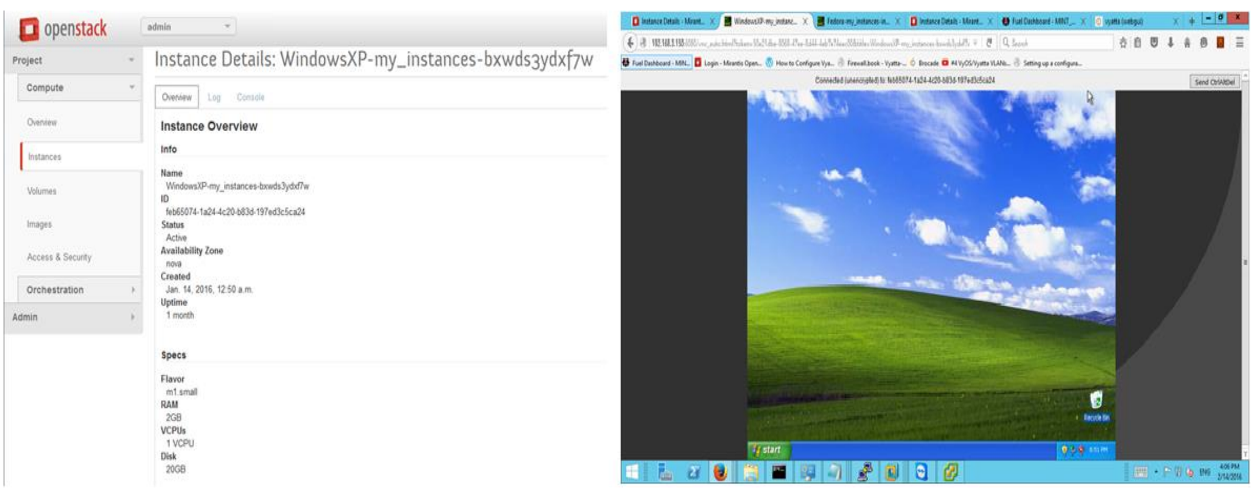


Figure 6.7 view of WindowsXP instance Overview and console

Ubuntu Instance:

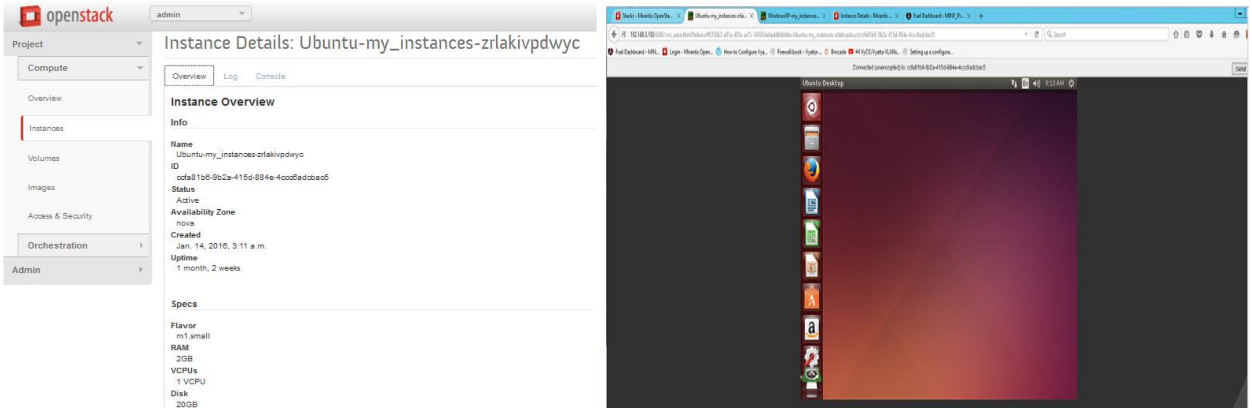


Figure 6.8 view of Ubuntu instance Overview and console

Fedora Instance:

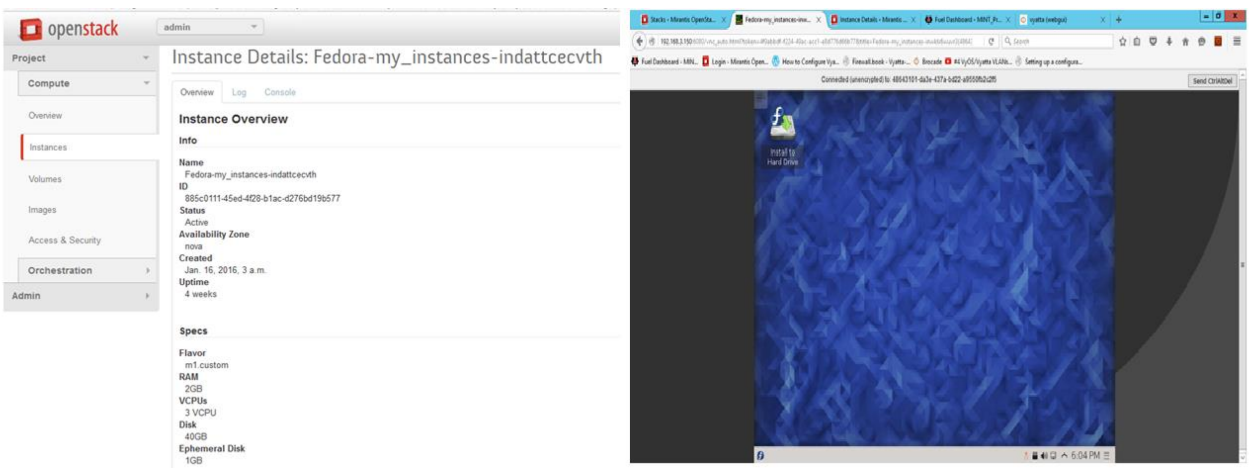


Figure 6.9 view of Fedora instance Overview and console

Since vCenter is selected as the compute node the three instances are actually deployed on the vCenter cluster (NovaCompute) which was specified during fuel deployment with the instance id as the VM name.

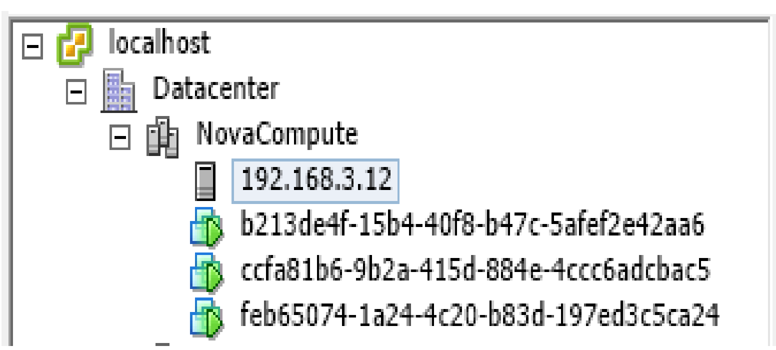


Figure 6.10 view of vCenter NovaCompute cluster after OpenStack instant deplyoment

Also, all the three instances are by default connected to a virtual machine port group **br100.**

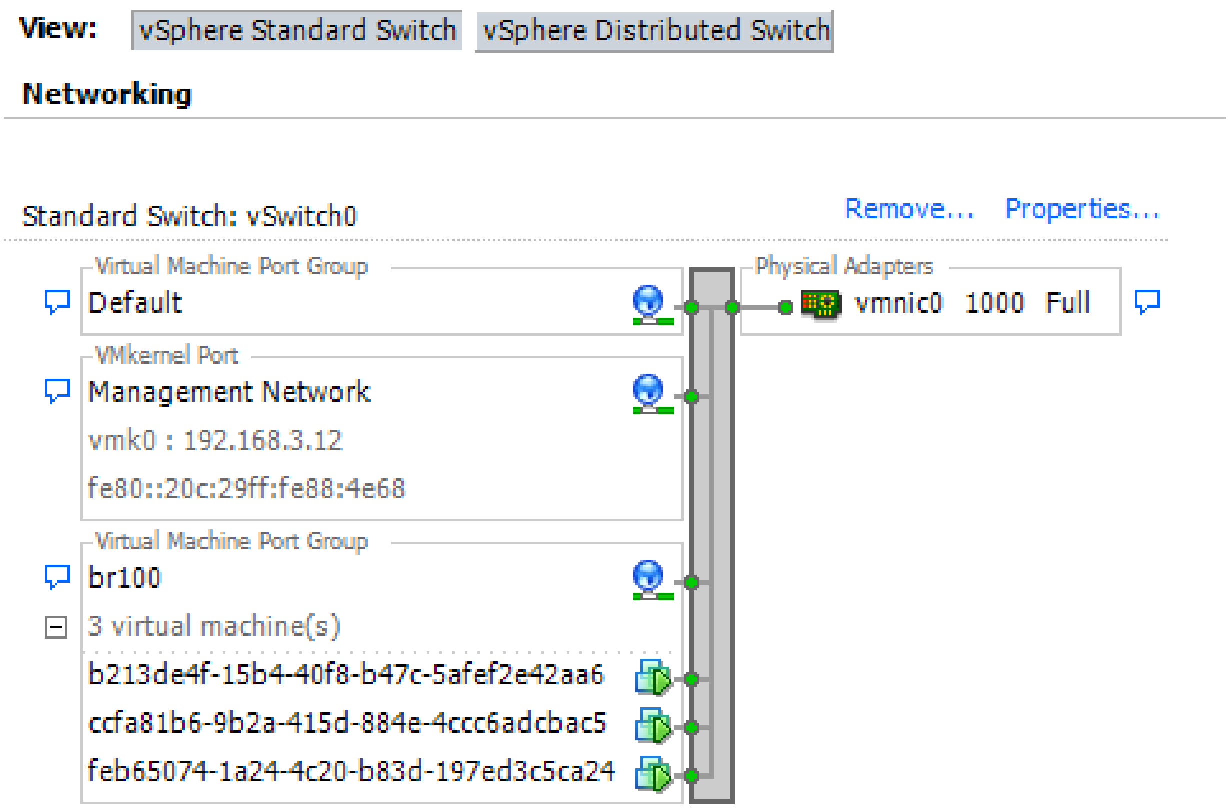


Figure 6.11 view of vCenter Networking

6.2 Tenant Separation Using Vyatta Firewall

Vyatta router is the major part of this project. It is created on VMWare Workstation outside the vCenter and OpenStack environment. Two network adapters are connected to this router, the first one is bridged to the external network and the second is connected to the host only network which connects to the local vCenter and OpenStack environment. The major functions of Vyatta router in the project are as follows:

- DHCP server (with Static MAC-IP Mapping)
- Network Address Translation (NAT)
- Firewall

6.2.1 Vyatta vRouter Network Configuring

The steps involved in configuring Vyatta router are as follows:

Step 1: Configure interface **eth0** connected to the bridged interface to get an IP from the external DHCP server.

Step 2: Configure interface **eth1** connected to the host only interface and assign three more static IPs from different subnet range as follows:
192.168.4.254/24
192.168.5.254/24
192.168.6.254/24

Step 3: Configure DHCP server for the added three different subnet with the default gateway pointed to ip on eth 1 from the same subnet.

Step 4: Configure static mapping such that the three instances (Windows, Ubuntu and Fedora) get IP address from three different subnet as follows:

Instance	MAC	IP
Windows	00:50:56:b7:6e:75	192.168.4.100
Ubuntu	fa:16:3e:a8:b6:26	192.168.5.100
Fedora	fa:16:3e:0c:97:b1	192.168.6.100

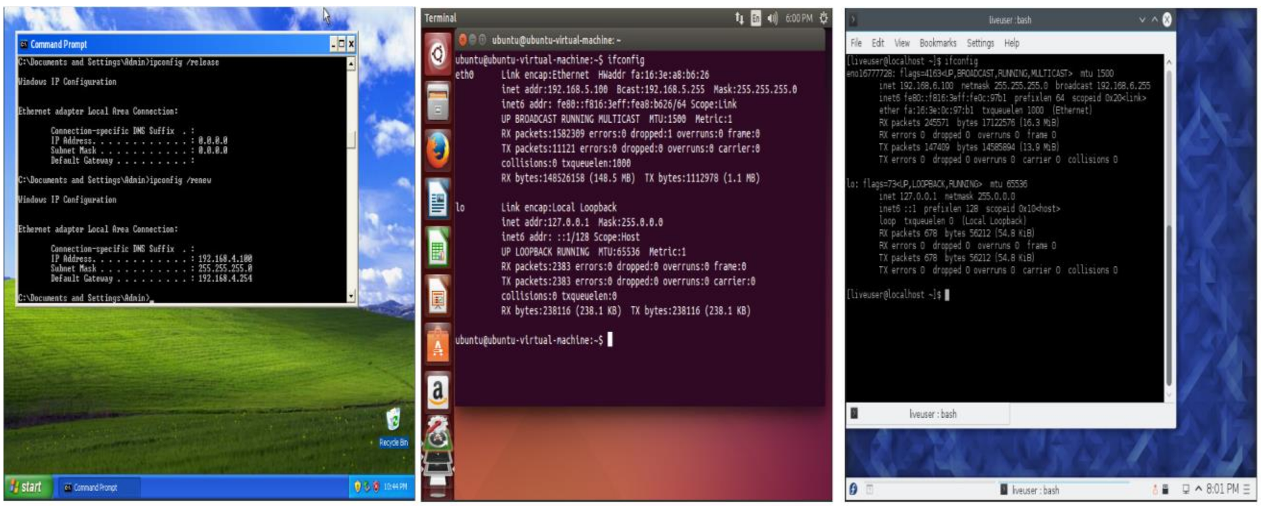


Figure 6.12 view of ip configuration of Windows, Ubuntu and Fedora instances

Step 4: Configure NAT with eth0 as outside interface to translate all the internal address from the subnet 192.168.0.0/16

Output of show configuration on Vyatta router is given below:

```
interfaces {
    ethernet eth0 {
        address dhcp
        duplex auto
        hw-id 00:0c:29:8b:66:39
        smp_affinity auto
        speed auto
    }
    ethernet eth1 {
        address 192.168.3.254/24
        address 192.168.4.254/24
        address 192.168.5.254/24
        address 192.168.6.254/24
        duplex auto
        hw-id 00:0c:29:8b:66:43
        smp_affinity auto
        speed auto
    }
  }
  nat {
    source {
        rule 1 {
            outbound-interface eth0
            source {
                address 192.168.0.0/16
            }
            translation {
                address masquerade
            }
        }
    }
  }
  service {
    dhcp-server {
        disabled false
        shared-network-name MINT {
            authoritative disable
            subnet 192.168.3.0/24 {
                default-router 192.168.3.254
                dns-server 8.8.8.8
                lease 86400
                start 192.168.3.100 {
                    stop 192.168.3.200
                }
            }
            subnet 192.168.4.0/24 {
                default-router 192.168.4.254
                dns-server 8.8.8.8
                start 192.168.4.100 {
                    stop 192.168.4.200
                }
                static-mapping Windows {
                    ip-address 192.168.4.100
                    mac-address 00:50:56:b7:6e:75
                }
```

```
            }
            subnet 192.168.5.0/24 {
                default-router 192.168.5.254
                dns-server 8.8.8.8
                start 192.168.5.100 {
                    stop 192.168.5.200
                }
                static-mapping Ubuntu {
                    ip-address 192.168.5.100
                    mac-address fa:16:3e:a8:b6:26
                }
            }
            subnet 192.168.6.0/24 {
                default-router 192.168.6.254
                dns-server 8.8.8.8
                start 192.168.6.100 {
                    stop 192.168.6.200
                }
                static-mapping Fedora {
                    ip-address 192.168.6.100
                    mac-address fa:16:3e:0c:97:b1
                }
            }
        }
    }
    https {
        http-redirect enable
    }
}
system {
    host-name Vyatta
    login {
        user Vyatta {
            authentication {
                encrypted-password $1$.DjhByrl$.i5PyfOtP34liIYkk1FB//
            }
            level admin
        }
    }
    syslog {
        global {
            facility all {
                level notice
            }
            facility protocols {
                level debug
            }
        }
        user all {
            facility all {
                level emerg
            }
        }
    }
    time-zone GMT
}
```

6.2.2 Vyatta vRouter Firewall configuration using Rest API

Vyatta router is used as a firewall to prevent communication between two instances. As we have performed static mapping of the MAC addresses of the instances with IP address the three instances created always get the same IP address from the DHCP server. Therefore, we setup a Layer 3 firewall rule on the Vyatta router to block the traffic between Windows and Fedora instance and allow the traffic between Windows and Ubuntu instance. This firewall rule is configured on Vyatta router through rest API using curl command.

The steps involved in configuring firewall rule using rest API is as follows:

Step 1: Enable HTTPS on the Vyatta system using the command set services HTTPS

```
vyatta@vyatta# set service https
[edit]
vyatta@vyatta# commit
[ service https ]
Stopping web server: lighttpd.
Starting web server: lighttpd.
Stopping API PAGER server
Starting API PAGER server
spawn-fcgi: child spawned successfully: PID: 48499
```

Figure 6.13 Enabling HTTPS on Vyatta system

Step 2: Start a configuration session and create a unique session ID. To perform this task following curl command is passed through a Ubuntu terminal:

curl -k -s -i -u vyatta:vyatta -H "content-length:0" -H "Accept: application/json" -X POST https://192.168.3.254/rest/conf

```
ubuntu@ubuntu:~$ curl -k -s -i -u vyatta:vyatta -H "content-length:0" -H "Accept
: application/json" -X POST https://192.168.3.254/rest/conf
HTTP/1.1 201 Created
Content-Type: application/json
Location: rest/conf/8271688DAEE87497
Vyatta-Specification-Version: 0.3
Cache-Control: no-cache
Transfer-Encoding: chunked
Date: Sun, 28 Feb 2016 05:37:59 GMT
Server: lighttpd/1.4.28
```

Figure 6.14 start configuration session and create a unique session ID using curl command

Step 3: All the active configuration mode sessions can be listed by using following curl command:

curl -k -s -i -u vyatta:vyatta -H "content-length:0" -H "Accept: application/json" -X GET https://192.168.3.254/rest/conf

```
ubuntu@ubuntu:~$ curl -k -s -i -u vyatta:vyatta -H "content-length:0" -H "Accept
: application/json" -X GET https://192.168.3.254/rest/conf
HTTP/1.1 200 OK
Content-Type: application/json
Vyatta-Specification-Version: 0.3
Cache-Control: no-cache
Content-Length: 226
Date: Sun, 28 Feb 2016 05:38:30 GMT
Server: lighttpd/1.4.28

{
  "session": [
    {
      "id": "8271688DAEE87497",
      "username": "vyatta",
      "description": "",
      "started": "1456637879",
      "modified": "false",
      "updated": "1456637879"
    }
  ],
  "message": " "
}
```

Figure 6.15 List active configuration mode sessions

Step 3: After creating a unique session ID, this session ID is used to reference the session for all the other curl command. Listed below are the curl commands to create a firewall named **MINT-Firewall** with default action accept. Within this firewall two rules are defined, the first one **rule1** is defined to drop all the traffic with source address 192.168.4.100(Windows) and destination address 192.168.6.100(Fedora) while the second rule **rule2** is defined to accept(allow) all the traffic with source address 192.168.4.100(Windows) and the destination address 192.168.5.100(Ubuntu).

curl -k -s -i -u vyatta:vyatta -H "content-length:0" -H "Accept: application/json" -X PUT https://192.168.3.254/rest/conf/8271688DAEE87497/set/firewall/name/MINT-Firewall/default-action/accept

curl -k -s -i -u vyatta:vyatta -H "content-length:0" -H "Accept: application/json" -X PUT https://192.168.3.254/rest/conf/8271688DAEE87497/set/firewall/name/MINT-Firewall/rule/1/action/drop

curl -k -s -i -u vyatta:vyatta -H "content-length:0" -H "Accept: application/json" -X PUT https://192.168.3.254/rest/conf/8271688DAEE87497/set/firewall/name/MINT-Firewall/rule/1/source/address/192.168.4.100

curl -k -s -i -u vyatta:vyatta -H "content-length:0" -H "Accept: application/json" -X PUT https://192.168.3.254/rest/conf/8271688DAEE87497/set/firewall/name/MINT-Firewall/rule/1/destination/address/192.168.6.100

curl -k -s -i -u vyatta:vyatta -H "content-length:0" -H "Accept: application/json" -X PUT https://192.168.3.254/rest/conf/8271688DAEE87497/set/firewall/name/MINT-Firewall/rule/2/action/accept

curl -k -s -i -u vyatta:vyatta -H "content-length:0" -H "Accept: application/json" -X PUT https://192.168.3.254/rest/conf/8271688DAEE87497/set/firewall/name/MINT-Firewall/rule/2/source/address/192.168.4.100

curl -k -s -i -u vyatta:vyatta -H "content-length:0" -H "Accept: application/json" -X PUT https://192.168.3.254/rest/conf/8271688DAEE87497/set/firewall/name/MINT-Firewall/rule/2/destination/address/192.168.5.100

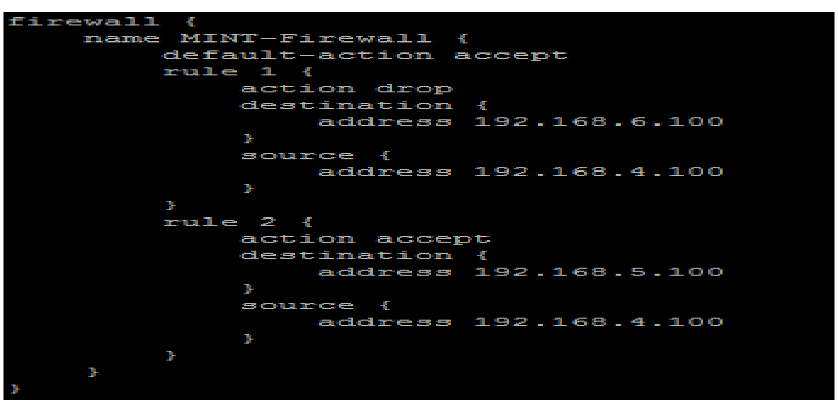
curl -k -s -i -u vyatta:vyatta -H "content-length:0" -H "Accept: application/json" -X PUT https://192.168.3.254/rest/conf/8271688DAEE87497/set/interfaces/ethernet/eth1/firewall/in/name/MINT-Firewall

curl -k -s -i -u vyatta:vyatta -H "content-length:0" -H "Accept: application/json" -X PUT https://192.168.3.254/rest/conf/8271688DAEE87497/set/interfaces/ethernet/eth1/firewall/out/name/MINT-Firewall

curl -k -s -i -u vyatta:vyatta -H "content-length:0" -H "Accept: application/json" -X POST https://192.168.3.254/rest/conf/8271688DAEE87497/commit

curl -k -s -i -u vyatta:vyatta -H "content-length:0" -H "Accept: application/json" -X DELETE https://192.168.3.254/rest/conf/8271688DAEE87497

The new firewall created by using rest API can be verified from Vyatta command prompt



```
firewall {
    name MINT-Firewall {
        default-action accept
        rule 1 {
            action drop
            destination {
                address 192.168.6.100
            }
            source {
                address 192.168.4.100
            }
        }
        rule 2 {
            action accept
            destination {
                address 192.168.5.100
            }
            source {
                address 192.168.4.100
            }
        }
    }
}
```

Figure 6.16 view of Vyatta router firewall configuration

To verify that the firewall rule is working we perform ping test from Windows instance to Ubuntu and Fedora instance.

- Ping response from Windows (192.168.4.100) to Ubuntu (192.168.5.100) and Fedora (192.168.6.100) before configuring firewall.

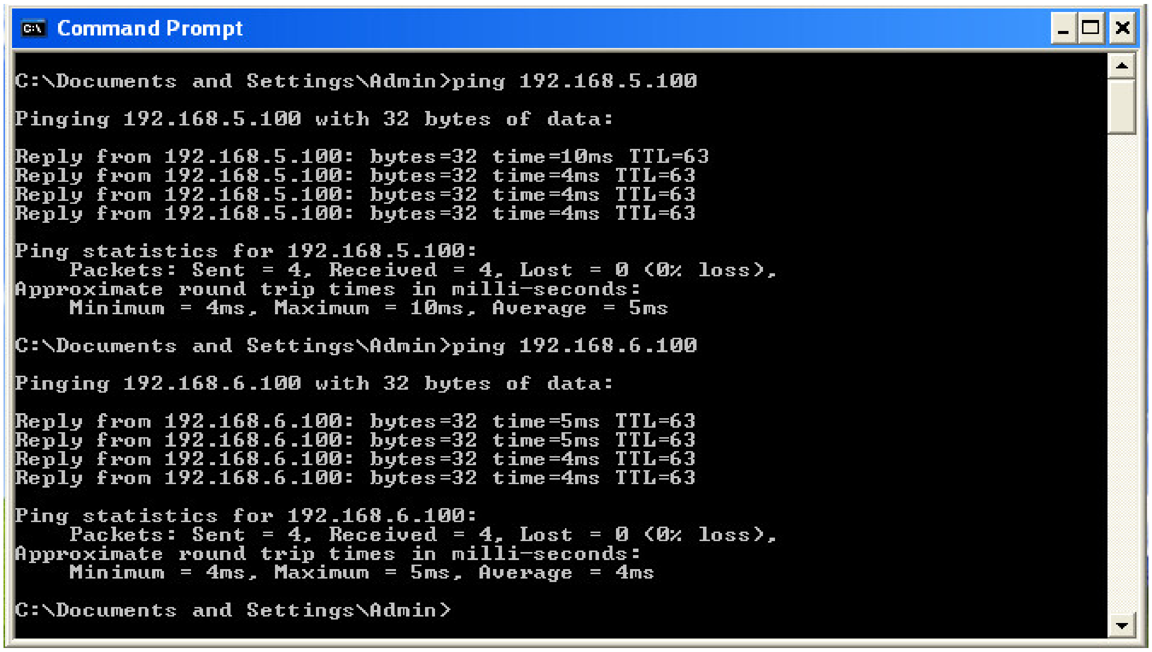


Figure 6.17 ping response before configuring firewall

- Ping response from Windows (192.168.4.100) to Ubuntu (192.168.5.100) and Fedora (192.168.6.100) after configuring firewall.

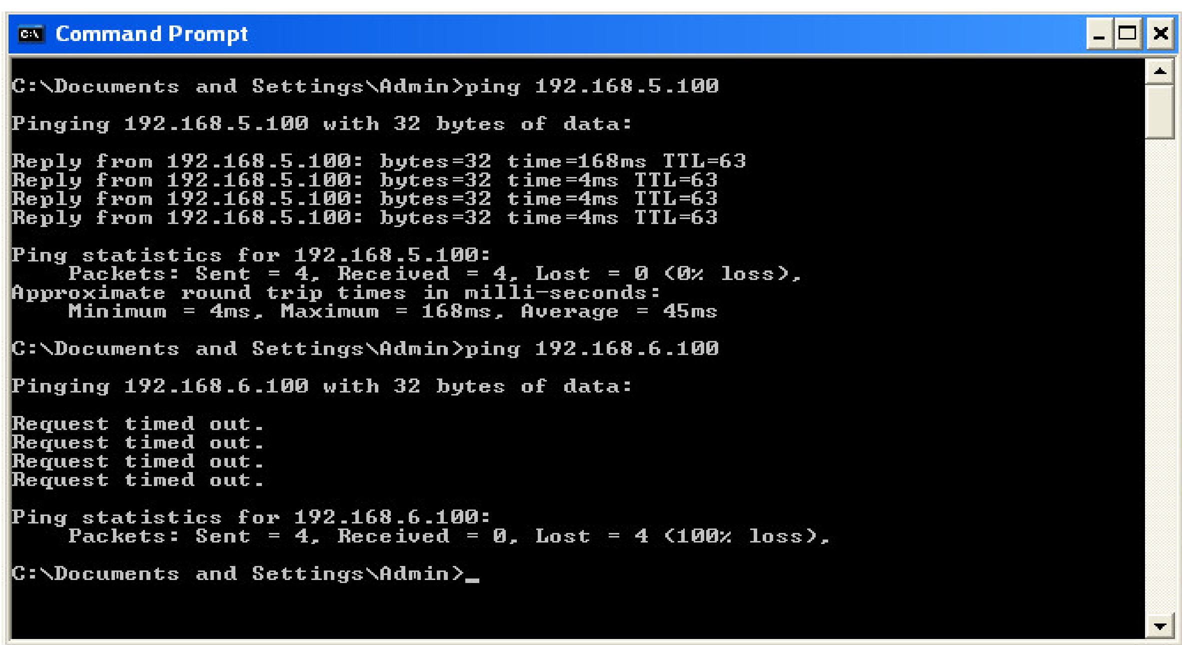


Figure 6.18 ping response after configuring firewall

7 SUMMARY AND CONCLUSION

From this proof of concept project, we were successfully able to demonstrate the integration of OpenStack with vCenter and create instances on vSphere infrastructure (ESXi host) using Heat orchestration. Mirantis OpenStack 5.0.1 Icehouse was deployed using Fuel which provided an intuitive GUI driven web based interface to configure nodes and network setup to connect OpenStack to vCenter. OpenStack Horizon dashboard was then used to orchestrate three different instances (Windows, Ubuntu and Fedora) using Heat template. Also, though this project we were able to demonstrate the use of Vyatta vRouter to manage the network resources of the OpenStack cloud. We successfully configured firewall on Vyatta vRouter using Rest API to perform network separation between the cloud tenant.

This proof of concept demonstration can be scaled up to create an enterprise scale cloud controlled and managed using OpenStack. Due to the limitation on resources this project was performed on a VMWare Workstation environment. The large scale would be installed on high performance servers with high speed network and storage capacity. Each physical server can be added as separate node and can be assigned with different OpenStack services. Also, NSX component can be added to vSphere environment and integrated with OpenStack Neutron. The integration of NSX with Neutron networking will provide full network management of the cloud on vSphere infrastructure using OpenStack.

Bibliography & References:

1. OpenStack Training Guides (May 10, 2015)
2. OpenStack Configuration Reference icehouse (June 1, 2015)
3. Introduction to VMware vSphere (EN-000102-00)
4. Brocade 5400 vRouter DataSheet
5. Mirantis Reference Architecture (17 December 2014)
6. Brocade Vyatta -Remote Access API 2.0 Reference Guide (3.5R3 v01)
7. *http://www.openstack.org/ (4/2/2016)*
8. *http://www.vmware.com/ (4/2/2016)*
9. *https://www.OpenStack.org/software/icehouse/*
10. *https://content.mirantis.com/vapp-mirantis-OpenStack-landing-page.html*
11. *http://www.OpenStack.org/software/project-navigator*
12. *http://docs.OpenStack.org/liberty/config-reference/content/vmware.html*
13. *https://content.mirantis.com/mirantis-vmware-reference-architecture-thank-you.html*
14. *http://www.webopedia.com/TERM/C/cloud_computing.html*
15. *http://blog.rackspace.com/architecting-vmware-vSphere-for-OpenStack/*
16. *http://docs.OpenStack.org/admin-guide-cloud/common/get_started_compute.html*
17. *http://blog.platform9.com/blog/explaining-how-vmware-vSphere-integrates-with-OpenStack/ (11/2/2016)*
18. *http://wikibon.org/wiki/v/The_Data_Center:_Past,_Present_and_Future*
19. *http://getcloudify.org/2014/07/10/what-is-OpenStack-tutorial.html*
20. *https://player.vimeo.com/video/90420485*
21. *http://docs.OpenStack.org/user-guide/dashboard_manage_images.html*
22. *https://developer.rackspace.com/blog/OpenStack-orchestration-in-depth-part-2-single-instance-deployments/*
23. *https://www.mirantis.com/blog/mirantis-OpenStack-express-intro-heat-orchestration/*
24. *https://www.brocade.com/content/dam/common/documents/content-types/api-reference-guide/vyatta-remote-access-api-3.5r3-v01.pdf*
25. *https://curl.haxx.se/docs/manpage.html*
26. *https://wiki.OpenStack.org/wiki/Fuel*
27. *http://www.thoughtsoncloud.com/2014/03/a-brief-history-of-cloud-computing/*
28. *http://www.academia.edu/8605710/Planning_Guide_Introduction_to_Mirantis_OpenStack_and_Fuel_2*
29. *http://pubs.vmware.com/vSphere-55/index.jsp?topic=%2Fcom.vmware.vSphere.install.doc%2FGUID-7C9A1E23-7FCD-4295-9CB1-C932F2423C63.html*
30. *https://dzone.com/articles/what-OpenStack-quick-OpenStack*
31. *http://cloudarchitectmusings.com/2015/08/13/understanding-how-vmware-vSphere-integrates-with-OpenStack/*
32. *http://www.dummies.com/how-to/content/comparing-public-private-and-hybrid-cloud-computin.html*
33. *http://searchcloudstorage.techtarget.com/definition/RESTful-API*
34. *http://docs.openstack.org/security-guide/introduction/introduction-to-openstack.html*