University of Alberta

Faculty of Electrical and Computer Engineering

Master of Science in internetworking

Integrating OpenStack and ODL to provide VTN Automation Service

Submitted by Bingzhu Xie

Supervisor: Mr.Gurpreet Nanda

Abstract

With the wide application of IP networks, the modern digital society is in need of a more flexible, fault-tolerant and manageable network. Software-Defined Networking (SDN) is such a kind of network by separating the network's control logic from the routers and switches. This separation makes it easier to introduce new abstractions in networking and simplifies network management. With proper software, the concept of SDN can be implemented to devices.

OpenDaylight and OpenStack can be integrated to support this idea, by connecting the management plane, the control plane and the data plane with one another, via OpenFlow API. In this case, a manager can monitor the network from the manage plane and the operator can simply set parameters in a template designed at the OpenStack side. Besides, the control plane would centralize the intelligent part and leave the data plane with the forwarding part, which makes it much flexible to keep the network up-to-date. In the data plane, we configure protocols such as BGP, OSPF, RIP and so on.

In this project, we will implement a VTN Automation solution using Openstack and OpenDaylight SDN Controller. Specifically, the following will be implemented:

- Implement OpenStack.
- Implement OpenDaylight SDN Controller
- Integrate OpenStack and ODL.

Through this project, we will get to know about what SDN is and the leading controller OpenDaylight as well as the OpenFlow specifications. Besides, a thorough understanding and application of OpenStack are also expected from the project. The project would demonstrate the process of OpenDaylight and OpenStack integration as well as VTN automation as an example.

Acknowledgement

With the completion of my project, I’d like express my special thanks to Mr. Gurpreet Nanda, who has been guiding me through the project. His patient and skillful guidance has helped me so much and been significant to me.

I also want to give my sincere gratitude towards Mr. Shahnawaz Mir, who has offered me a lot of help during my lab setup. Besides, I sincerely appreciate the help from Mr. Mike Macgregor and Ms. Sharon Gannon who have been helping me through the MINT program.

Last but not least, I’d like to thank my families who have been supporting me through the program.

1. Introduction

1.1 Problem Description

In conventional network structure, when it is carried out into the practical world, if a new request is raised, it could be considerably trivet to reconfigure or change the corresponding devices, such as routers, switches, firewalls and so on. With the dramatical change of the internetworking and mobile networking environment, the high stability and idealistic functionality are no longer satisfying, the flexibility and agility are far more critical instead. What SDN does is to separate the control function from forwarding devices, and leave the control function totally to the centralized controller. Therefore, the control plane is independent from the low layer devices, and becomes irrelevant to the variety of the switches, routers, firewalls and all other devices. On the other hand, the access to control is open, and hereby the users can define whatever routing protocols and transmission policies as they want, which makes the network much more flexible and intelligent. In a software defined network, there is no need to reconfigure each and every node in the network over and over again. Since the devices in the same network is connected to each other automatically, the user only needs to define policies when using it. If the protocols built in the routers are no longer desirable, they could be modified by programming, to achieve better data transmission. Another advantage of SDN is that it could easily adjust flow to widen the streaming media, which is to say, the bandwidth and flow is manageable in SDN.

Traditionally, investment in network systems and operating expenses are huge because the network is configured as a silo for each department and system. And hereby, various network appliances must be installed for each tenant and cannot be shared with others. This leads to a heavy burden to design, implement and operate the entire network. However, VTN, the Virtual Tenant Network is an appliance, which can provide various virtual tenant networks on the SDN controller. The uniqueness of VTN is that it has a logical abstraction plane, which enables the complete separation of logical plane from physical plane. Users can design and deploy any desired network not necessarily knowing the physical network topology or bandwidth Restrictions.

However, VTN is not easy to implement with traditional tools as it is basically virtualized and requires significant performance which can hereby simulate the real world. With the limitations of traditional networks and development tools, we, in this project, try to set out from SDN, using VTN tools, to practice a use case.

1.2 Proposed Solution

SDN, the Software Defined Network, is a novel network framework. It is a method of network virtualization. Its core technique is to achieve flexibility in flow control through separating the control plane from data plane, which leaves the data plane only forwarding functions and the control functions are centralized to the control plane. This feature makes the network dramatically intelligent as a tunnel.

VTN can be created and managed by OpenDayLight, ODL has a whole system of modular, pluggable and flexible platform. The platform is on basis of Java development and can operate on any Java supported platforms theoretically. ODL controller uses OSGI framework and SGI framework to be a Java-oriented dynamic modeling system. Bundles don’t need redirection and can be installed, started, updated and uninstalled remotely. It can flexibly load codes and functions via bundles to achieve function separation and resolve the module extension issue, as well as collaborating the modules.

ODL platform introduced SAL and its northbound modules. They provide low layer service in the form of plug-ins. The northbound plug-ins connect various protocols and ignore the differences among the various protocols, providing consistent services for upper functional modules. This could separate the upper modules with lower modules. SAL could adapt different devices automatically and hereby the developers can concentrate on developing the applications. Besides, the ODL controller uses Infinispan, which is a highly scalable, reliable and key-value storage distributed data structure network platform, in order to achieve data storage, look-up and monitoring, to further achieve the clustering of controllers. In a word, OpenDayLight as a controller, is designed on basis of the following principles. Runtime Modularity and Extensibility, Multiprotocol Southbound, Service Abstraction Layer, Open Extensible Northbound API, Support for Multitenancy/Slicing and consistent Clustering.

Another significant application in SDN development is OpenStack. OpenStack is a community and a project. It’s also an open source software devoted to operating virtual computing or cloud storage of enterprises. OpenStack provides users with open source softwares and establishing public cloud as well as private cloud. Public cloud requires its enterprise trust its statistics with the cloud provider’s data centre, which may probably cause data loss with environmental or personnel factors. Many enterprises choose to establish private cloud within a firewall. This leaves more space to the enterprise in aspects of security, and data backup. Back to OpenStack, it provides services including computing, object storage, image, networking and so on. OpenStack make a platform available to users to manage cloud. It can initiate an instance for a user or a group of users. It can also configure a network where there are more than one instance within each and every instance or project. The services provided by OpenStack can be installed independently corresponding to the user's requirements.

OpenStack's core service modules are Nova (computing), Neutron (network), Cinder (block storage), Swift (object storage), Glance (image management), Horizon (the management page), Keystone (access), Heat (function), Ceilometer (monitor). These modules can create an IAAS cloud platform. Neutron in OpenStack itself is an SDN networking control system. It has the ability to have users build their networks and control the flow. It can also connect the server and devices to one or more networks. OpenStack also leaves APIs to integrate other controllers like OpenDayLight to achieve a software defined network.

The concept of SDN can be used to resolve the problems in traditional networks and carried out to create more flexible networks. Tools like OpenDaylight and OpenStack can be applied to achieve such networks. In this project, we establish a network with the concept of SDN, and configure it with related tools mentioned above, to demonstrate a use case of SDN, and explore the application of OpenDaylight and OpenStack.

2. Scope and Preparation

2.1 Technology Description

First of all, I studied the paper of a comprehensive survey.
(https://drive.google.com/drive/u/1/folders/0B4M8Xz4BRpradzFpaDduRzdLRXc)

- The concept of SDN

In traditional IP networks, the control and data planes are tightly coupled, embedded in the same networking devices, and the whole structure is highly decentralized. While in Software Defined Network, the control and data planes are decoupled. The separation of the control plane and the data plane can be realized by means of a well-defined programming interface between the switches and the SDN controller through API.

SDN can be defined as a network architecture with four pillars:

a. The control and data planes are decoupled.
b. Forwarding decisions are flow-based, instead of destination-based.
c. Control logic is moved to an external entity, the so-called SDN controller or Network Operating System (NOS).
d. The network is programmable through software applications running on top

of the NOS that interacts with the underlying data plane devices.

An SDN can be defined by three fundamental abstractions:

a. Forwarding. Ideally, the forwarding abstraction should allow any for- warding behavior desired by the network application (the control program) while hiding details of the underlying hardware, always by Openflow.
b. Distribution. It should shield SDN applications from the vagaries of distributed state, making the distributed control problem a logically centralized one, by a common distribution layer.
c. Specification, which means it allows a network application to express the desired network behavior without being responsible for implementing that behavior itself. By virtualization solutions and programming languages.

● SDN infrastructures

An SDN network infrastructure (from abstract to detail)

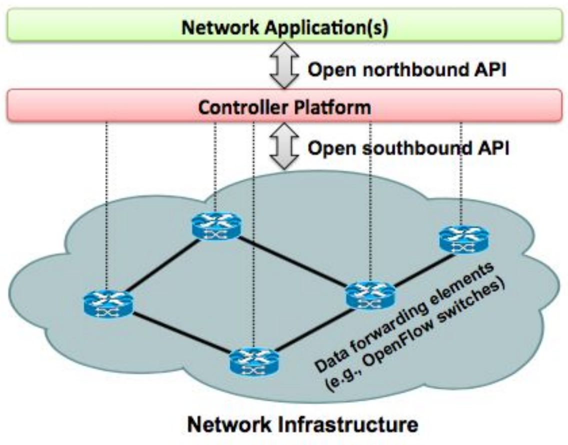


Figure 1. Abstract infrastructure
(quoted from software-defined networking: a comprehensive survey)

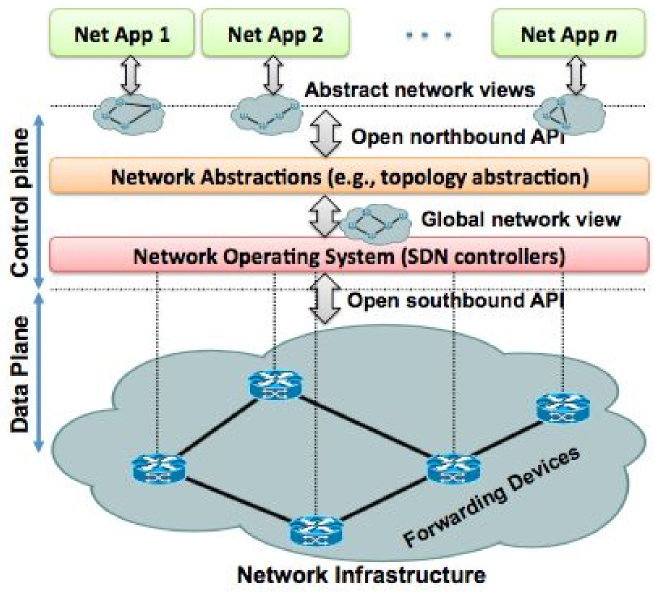


Figure 2. Detailed infrastructure (a)
(quoted from software-defined networking: a comprehensive survey)

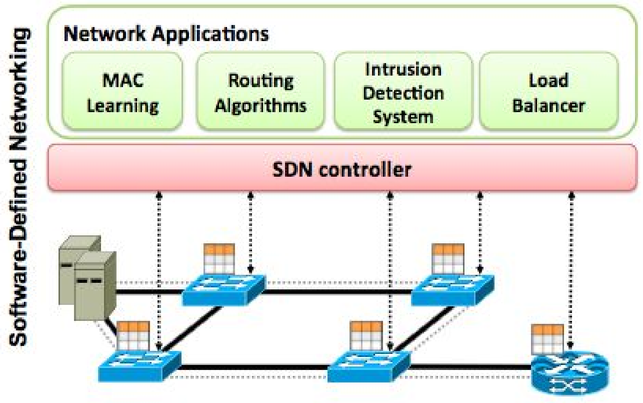


Figure 3. Detailed infrastructure (b)
(quoted from software-defined networking: a comprehensive survey)

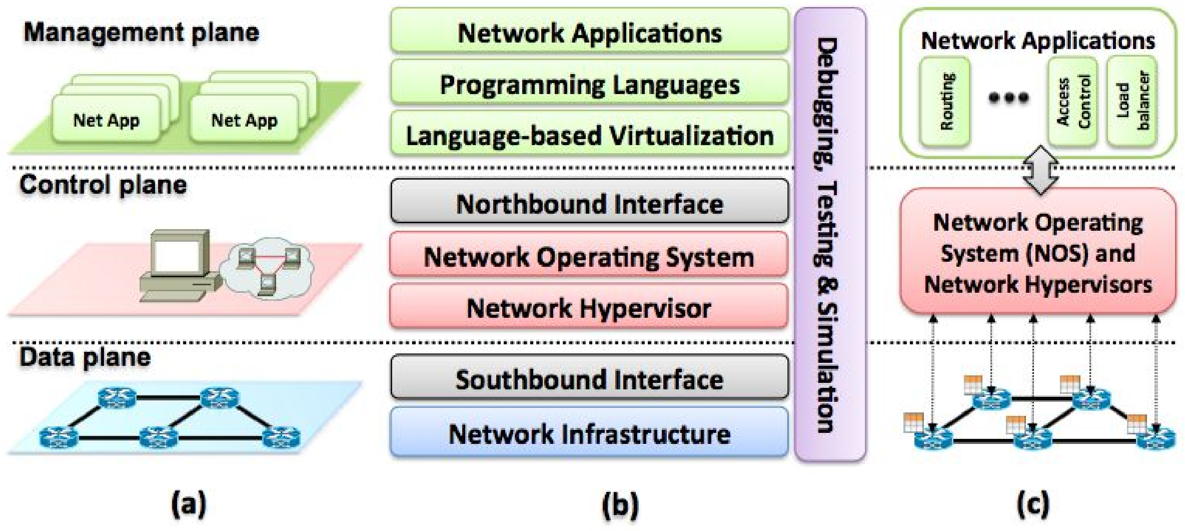


Figure 4. Key building blocks of an SDN infrastructure using a bottom-up, layered approach (quoted from software-defined networking: a comprehensive survey)

layer 1—infrastructure: switches, routers and etc.
layer 2—southbound interface: Openflow
layer 3—network hypervisor: Hypervisors enable distinct virtual machines to share the same hardware resources.
layer 4—network operating system (controller): OpenDaylight.
layer 5—northbound interface: OpenDaylight defines its own API.
layer 6—language-based virtualization.
layer 7—programming language.
layer 8—network applications: implement the control-logic that will be translated into commands to be installed in the data plane.

Then let's take a look at software defined network applications in the practical world.

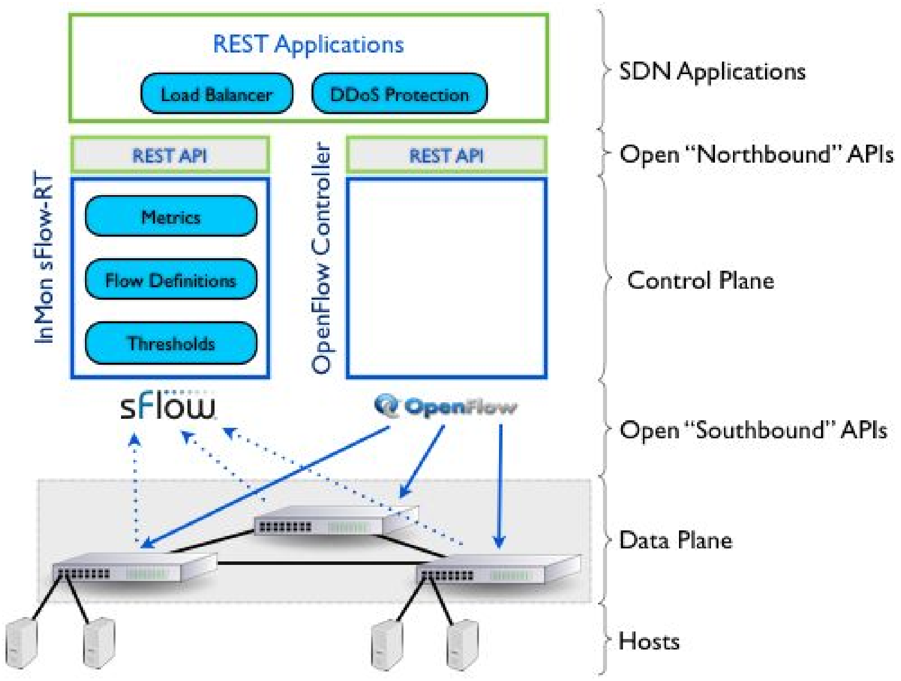


Figure 5. quoted from http://blog.sflow.com/2014/01/large-flow-marking-using-hybrid-openflow.html

The diagram above shows a practical application of SDN using openflow controller. The control plane communicated with data plane through southbound APIs, and communicates with upper layer applications through northbound APIs. The hosts can be virtual machines within the network range.

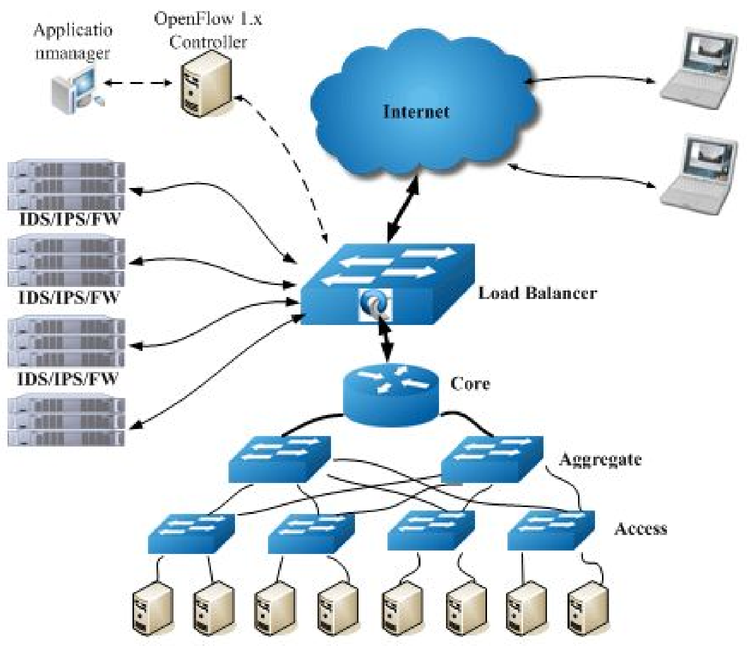


Figure 6. quoted from https://www.sdxcentral.com/products/centec-load-balance/

Sometimes, when there is a higher demand towards flow control or there are too many devices in the same network, we can add a load balancer to the data plane as above. The load balancer can be configured to have access to the internet.

We can conclude a software-defined network as the following picture.

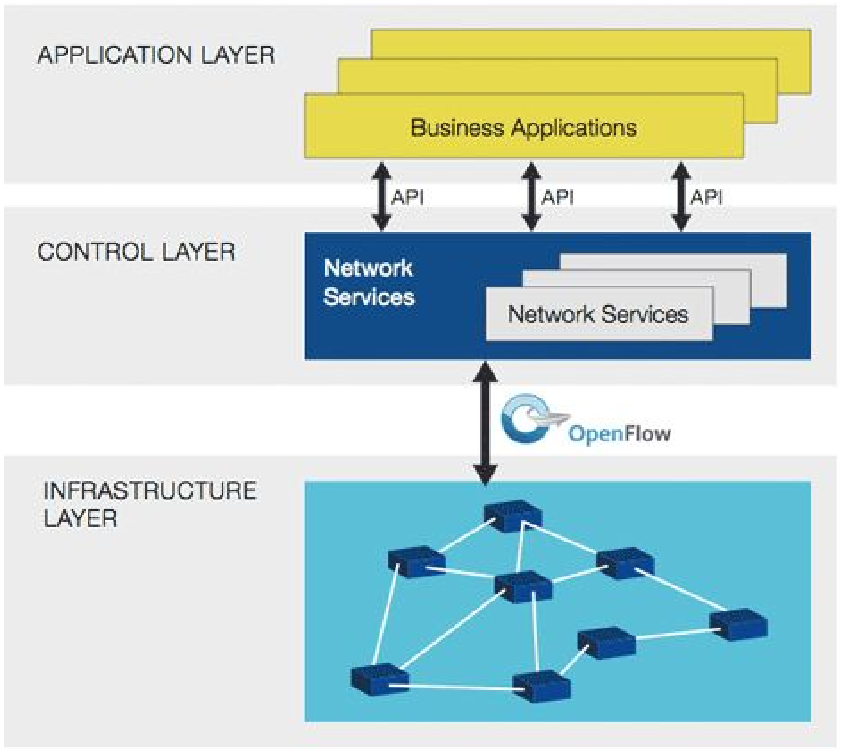


Figure 7. quoted from
https://www.sdxcentral.com/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/

- Difference brought by SDN

Let's first take a look at the former networks we used.

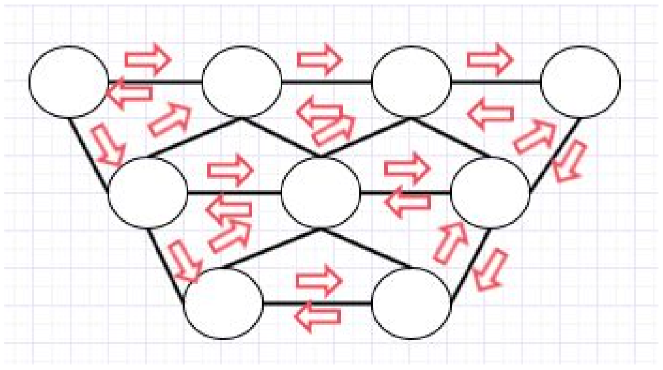


Figure 8.

The communication between devices is mainly through protocols, including switching, routing and security, such as OSPF, BGP, MPLS, MSTP, etc. It's achieved by neighborhood establishment, sharing information and path selection. This is the logic foundation for many protocols. Besides, it is distributed structure that is used by the majority of the networks. Devices transfer local message by overlaying. And then establish database and select the best path by path selection algorithms, the most typical one of which is SPF. During this, each device does its calculation separately and they each have a individual “brain” and forwarding hardwares. So the protocols is like the language for human being, which is the foundation of internetworking. What if a change occurs in the network? How could the devices in the network communicate with each other? When there is a turbulence in the network, the devices will overlay the message down to the next devices then delete the message of the compromised path. It is possible that there would be redundant notice during the convergence. The process is shown as follows.

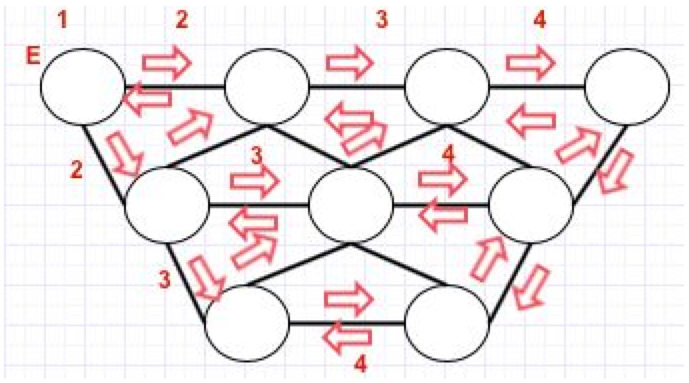


Figure 9.

The rising of cloud computing, big data and mobile internet has brought the doubling of flow, leading to the expansion of low layer networks with growing pressure. The convergence time of the network increases with the expanding of the network. Among all the typical networks, data center and mobile communication networks are under extreme pressure and consequently, a revolution is necessary. Besides, the bottleneck has been exposed in distributed structures.

To find a solution, let’s first think about how to resolve the bandwidth assignment Issue. Flow control is achieved by Qos and etc, generally. Set the priority by classification and labeling. On basis of the various requirements, assign corresponding bandwidth to the service. The quality of flow control directly influence the bandwidth utilization and influence the customer’s investment benefit. Flow control can not only work on switches, routers and other products that support Qos and also on professional flow control products like load balancers. However, no matter how to control the flow, by hardware or software, the process is mostly static bandwidth management. Based on a specific service requirement, make a corresponding regulation on the specific path. The strategy was set in advance and is unable to achieve bandwidth assignment intelligently with response to current network condition. Another problem in flow control is that, the overall network flow control is not visible. In regular flow control products and management systems, only part of the bandwidth assignment of the links and link state monitoring are visible. However, the overall network flow control visibility is the basis intelligent bandwidth Assignment. Gradually, an idea was brought up----can we customize the forwarding strategy?

As we all know, the work mode of traditional network devices are fixed. For instance, switches forward messages according to MAC table and routers forward messages according to the routing table. Therefore, to make the strategies customized is to make the devices programmable. What can be possibly programmable? It’s obvious that the devices like routers and switches cannot be programmable with response to the requirements.

Right here right now, a brand new idea of networking design rose, software defined Network. The very first infrastructure of SDN is as follows.

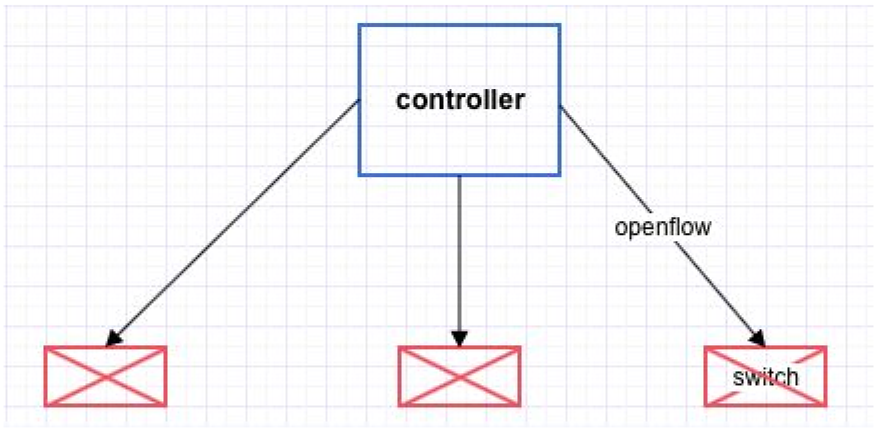


Figure 10.

In the picture above, the switches don’t have their own brains. Instead, all that related to path calculations and security strategies are determined in the controller. The decisions are passed down to the switches through openflow, and the switches simply do the forwarding according to flow table. It achieved the separation between control and forwarding.

A typical example of SDN application is the Google B4 network as the following picture shows.

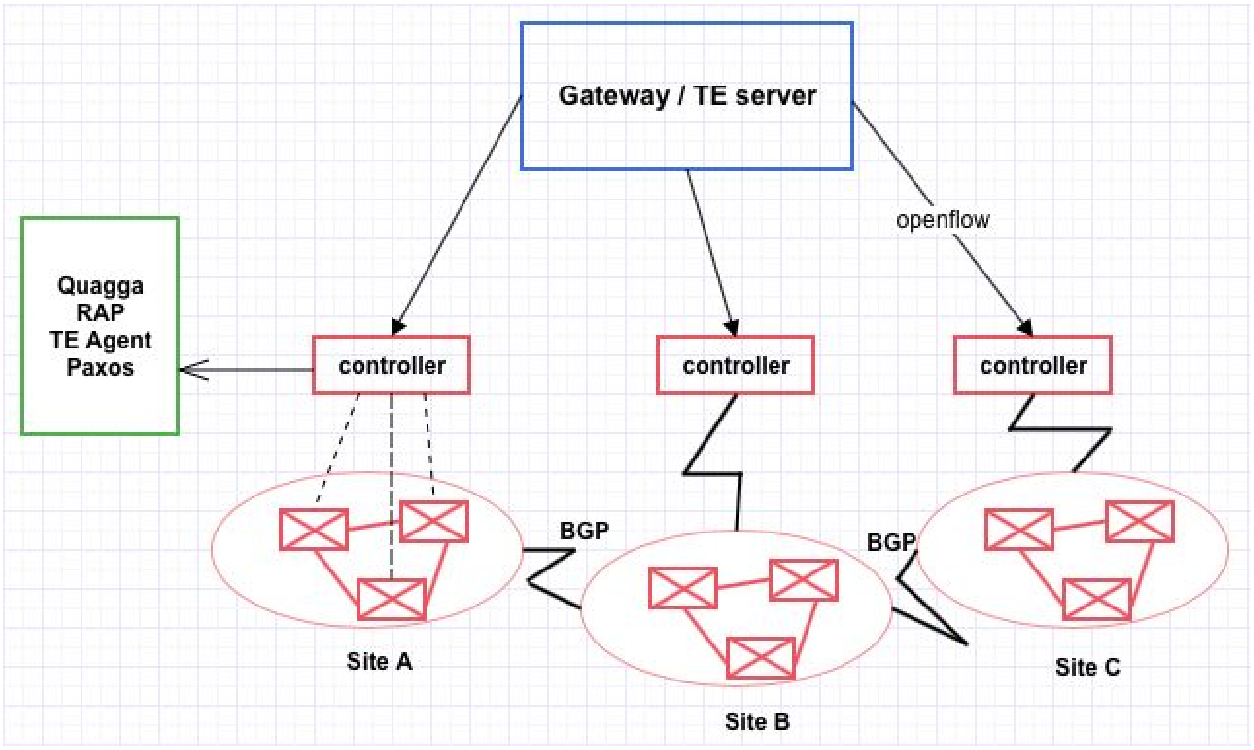


Figure 11.

This network has fundamentally improved the utilization rate of the WAN links from the original thirty or forty percent to one hundred percent. Overall, it applies distributed controllers system and introduces openflow switches to this SDN network. Quagga is a routing protocol stack, which is used to operate BGP and ISIS routing protocols. RAP, as the proxy from switches to quagga, is Routing Application Proxy. For example, when the openflow switches send the link state message to the controller, controller would call RAP to send the message to the protocol stack. Paxos is an election machine. It is used to select the master and slave among the controller cluster. TE agent is used to collect link state and bandwidth information of data centre and then send it to the gateway from the top layer. The gateway summarizes the overall information and send it to TE server to make the path calculation. In return, when TE server completed calculating the paths and bandwidth assignment, it would send this back to gateway. And it is gateway which sends this to openflow switches. The route is like the picture below.

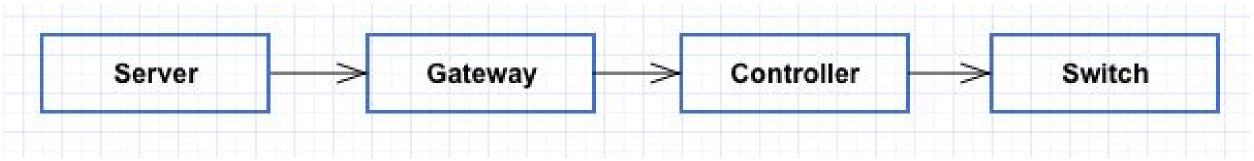
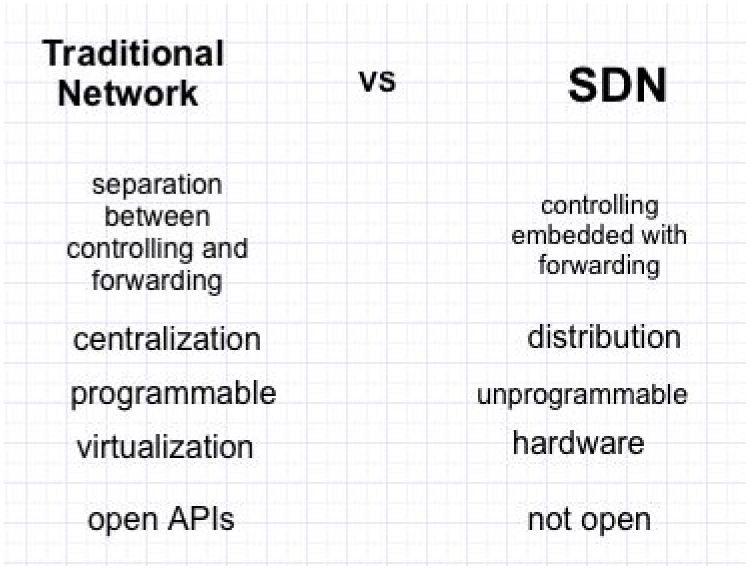


Figure 12.

In conclusion, the major difference between traditional network and SDN is listed as follows.



Traditional Network	vs	SDN
separation between controlling and forwarding		controlling embedded with forwarding
centralization		distribution
programmable		unprogrammable
virtualization		hardware
open APIs		not open

Figure 13.

In the aspect of logic structure,

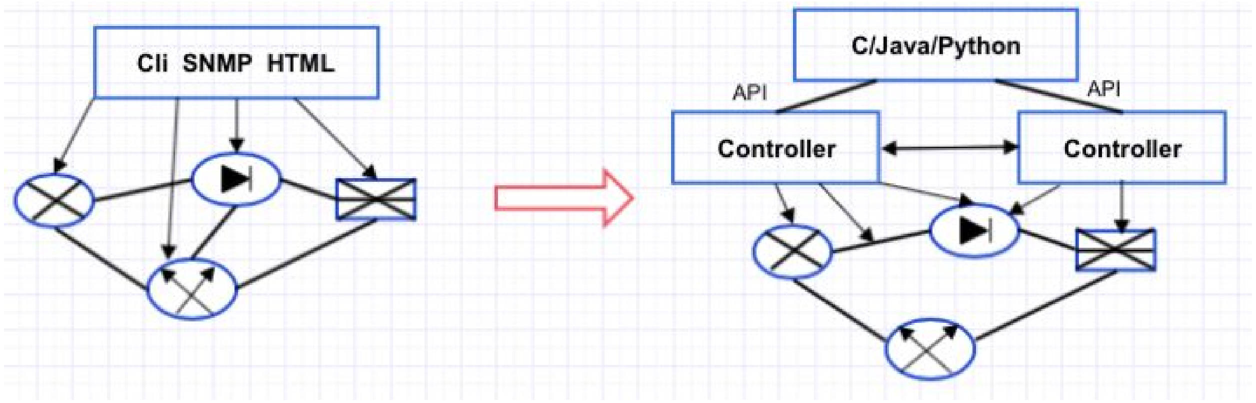


Figure 14.

The change brought by SDN has led to a reform in the industry. And they in return, is pushing SDN moving forward.

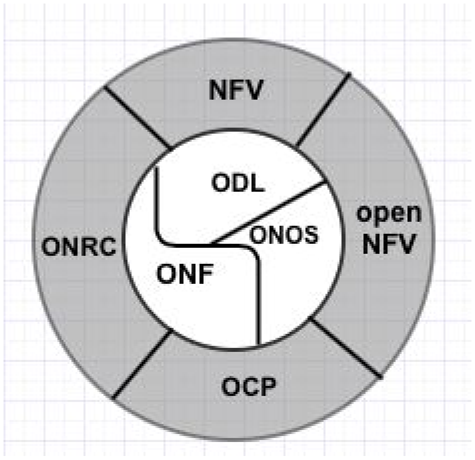


Figure 15.

- The concept of NFV

Let’s go back and take a look at NFV. NFV, network function virtualization, is to centralize network hardware on a server or platform through virtualization technologies. On this standardized server or platform, switches, routers, firewalls, load balancers and security devices can normally function. To be frank, it is to centralize discret network devices to a large box, as the following picture shows, and to use each device as plugins.

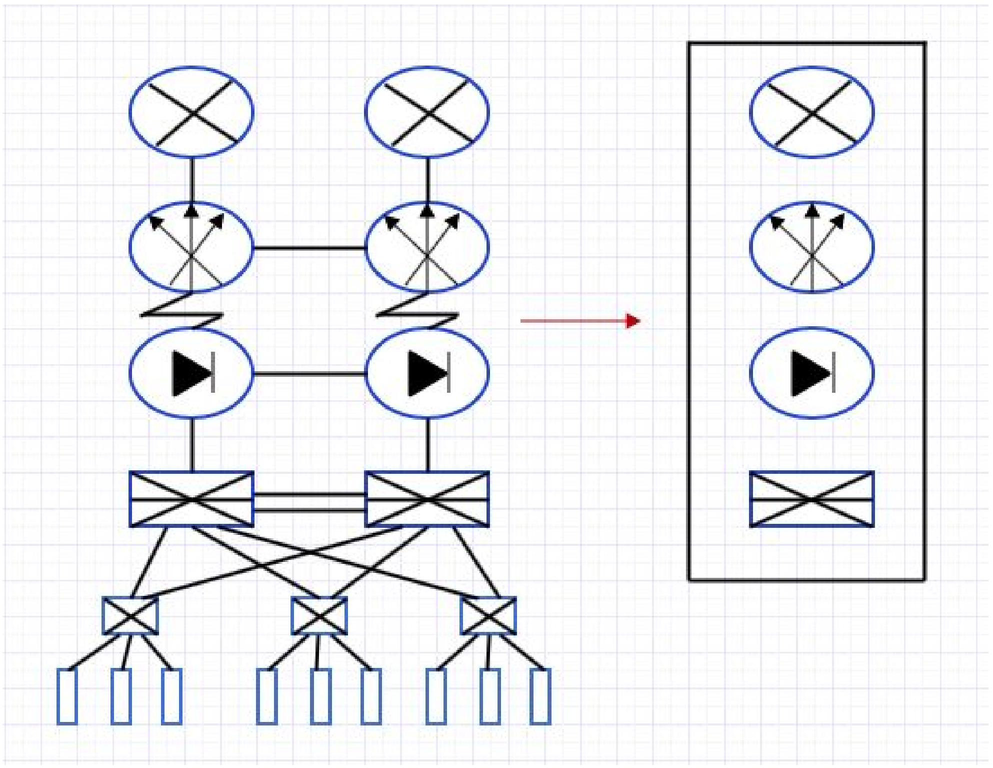


Figure 16.

Even though NFV and SDN are from separate organizations, they share a common purpose, to make the hardware to be software and virtualized, which makes the entire network virtualized, cost-saving, movable, scalable and more manageable.

2.2 Solution Description

SDN (Software Defined Network) is a network design concept. As long as the networking devices can be centralized, is programmable, with the separation between control plane and data plane (forwarding plane), the network is considered as a Software Defined Network. Therefore, SDN is not a specific technology or some protocol. On the other hand, it is an idea and a framework. To push it further, SDN is even involved with software defined security, software defined storage and so on. It’s fair to say that SDN is a trend that will stimulate the whole industry.

VTN is another trend that can map physical network resources. It can save the MAC addresses and VLAN IDs corresponding to the ports of the switches from terminals. The message obtained from terminals will be maintained until the packet from the terminal flows in. If the terminal is not connected to VTN, the timer will be triggered and the message will be maintained until the time is out. The vBridge is able to check the MAC address table. If the destination MAC address is already learnt, the packet will be forwarded to the corresponding virtual port. If the MAC address is not learnt yet, flooding it to all ports. Another function of VTN is Flow Filter Function. Flow Filter can be applied to any port in Vnode with special matching conditions. One significant advantage of VTN is that it can achieve single policy virtual network. Users can easily add a SDN controller to the existing VTN and can cut a SDN controller from the VTN.

For tools, OpenStack as well as OpenDayLight is open to integrate with each other. In OpenStack, a python daemon is the main process of the OpenStack networking that typically runs on the controller node. It exposes APIs, to enforce the network model, and passes the requests to the neutron plugin. So what is plugin? Plugins can be either core or service. Core plugins implement the “core” Neutron API — L2 networking and IP address management. Service plugins provide “additional” services, such as the L3 router, load balancing, VPN, firewall and metering. These network services can also be provided by the core plugins by realizing the relevant API extensions. In short, plugins run on the controller node and implement the networking APIs, which interact with the Neutron server, database and agents. (quoted from openstack.org) There are also plugin agents which are specific to the Neutron plugin being used. They run on compute nodes and communicate with the Neutron plugin to manage virtual switches. These agents are optional in many deployments and perform local virtual switch configurations on each hypervisor.

ML2's plugins are all core plugins. They are either type drivers or mechanism drivers. OVS, adrivers from ODL, Cisco, NEC and some others are mechanism drivers. They respond to actions such as update, establish and delete a network, subnet or port. Type drivers are flat, VLAN, GRE and VXLAN. They define the L2 type.

In short, the user input message to the networking API via the OpenStack horizon and then send to Neutron server. Neutron server receive the message and send it to plugin. Then Neutron server and plugin update their database. Plugin sends the message to the SDN controller via REST API. OpenDayLight could be the controller at this point. The controller receive the message and goes through the southbound plugins or protocols, such as OpenFlow, OVSDB or OF-Config.

Another crucial concept in SDN is Open vSwitch (OVS). In a virtualized platform, OVS is able to provide layer 2 service for dynamic nodes, as well as controlling the policies, network segments and flow control in an NFV. OVS supports OpenFlow. So every controller which supports OpenFlow can use OVS. The following terminologies are significant in OVS. Bridge, it stands for a ethernet switch. A host can establish one or more bridges. Port, it's similar to a port of a physical switch. Each port belongs to a bridge. Interface, it connects to the port. Usually, one port corresponds to one interface, monogally. One port can correspond to more than one interfaces only when the port is configured as the bond mode. Controller, one OVS can concurrently be managed by one or more controllers. Datapath, it's responsible to exchange data, which is to say matching the packets received from the receive port in the flow table and performing as what's matched. Flow table, each datapath is related to a flow table. When the datapath receive the data, OVS will look up the matching flow in flow table and perform the corresponding actions, forwarding data to another port for example.

To sort out the relationship of OpenDayLight, OpenStack and Openflow, let's stop here a little bit. As far as what we have introduced concerned, OpenStack is a Cloud Management System that provides a uniform API for provisioning Compute, Network, Storage in a DataCenter. There are different plugins for each of these areas that can be built into an OpenStack deployment and function underneath. OpenDayLight is a SDN controller that provisions the network policies as specified and sends the message to the Hypervisor. As a controller, it also performs the role of maintaining those policies in spite of the changes happening in the network, recomputing policies and loading to Hypervisors. OpenFlow is the protocol used to program the Hypervisor vSwitches. It's mostly about which traffic to send where and so on. And it's the protocol through which an SDN controller communicates with the Hypervisors. OpenvSwitch is the implementation of a virtual switch in the Hypervisor that exposes OpenFlow protocol for flow message and uses the message through this protocol and make packet forwarding decisions. For virtualization, you need some service to handle bridging between your instances. OpenvSwitch can handle this. OpenFlow is a protocol standard for SDN which facilitates remote management of switches from a centralized control plane with a wide range of support. OpenDaylight is an SDN controller that lets the user to programmably manage OpenFlow capable switches. It is a huge project due to the scale of collaboration with a large set of features and compatible northbound applications. Alternatives to ODL include Floodlight, RYU SDN framework, NOX, POX and so on. OpenStack is a cloud orchestration platform that can work independently without any of these technologies. However, it can also use all the above mentioned to provide the user more programmatic control over the Infrastructure and hence improve the scope for automation. Services like AWS, Google cloud platform, Azure let you orchestrate cloud networks without using OpenStack. So, such services could be considered as alternatives to OpenStack.

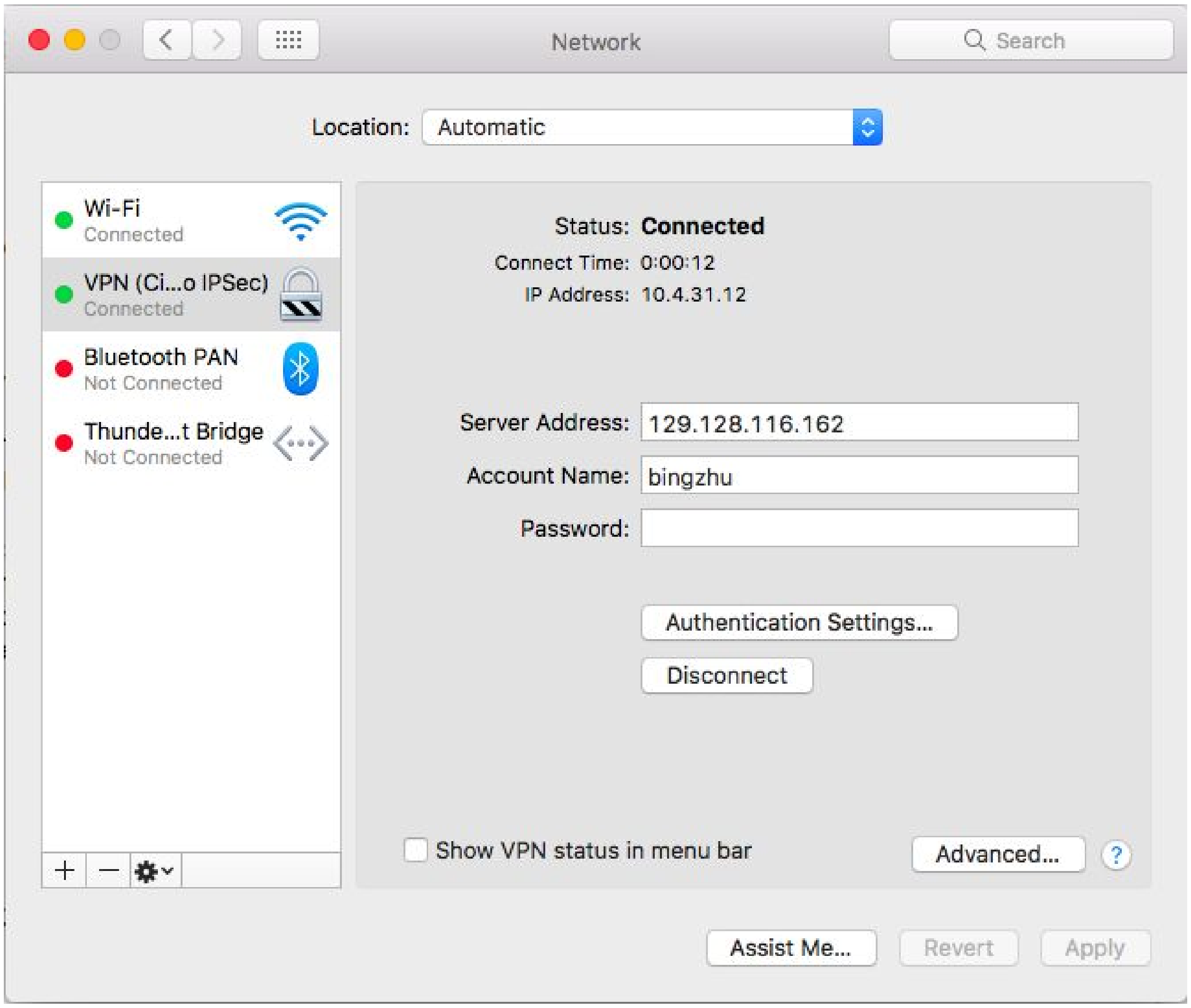
In this project, we use VTN to build a software defined network. The VMs would talk to each other via IP addressing. The controller will be configured on both OpenDaylight and OpenStack, which eventually would be integrated with each other. With application of SDN, the controller will distribute the network whenever there is a new VM is added or changed, and configure the protocols. This process will be achieved by launching instances in OpenStack.

3. Solution Infrastructure Setup

3.1 Hardware Preparation

- VPN

This is to make it possible to connect the server in the lab remotely. The platform we use in this lab is Mac OS. To connect to the MINT lab network, we need to install the VPN plugin, CiscoIPSec on the laptop. And then create user account through it. The details are demonstrated as follows.



Location: Automatic

Wi-Fi Connected
VPN (Ci...o IPSec) Connected
Bluetooth PAN Not Connected
Thunde...t Bridge Not Connected

Status: Connected
Connect Time: 0:00:12
IP Address: 10.4.31.12

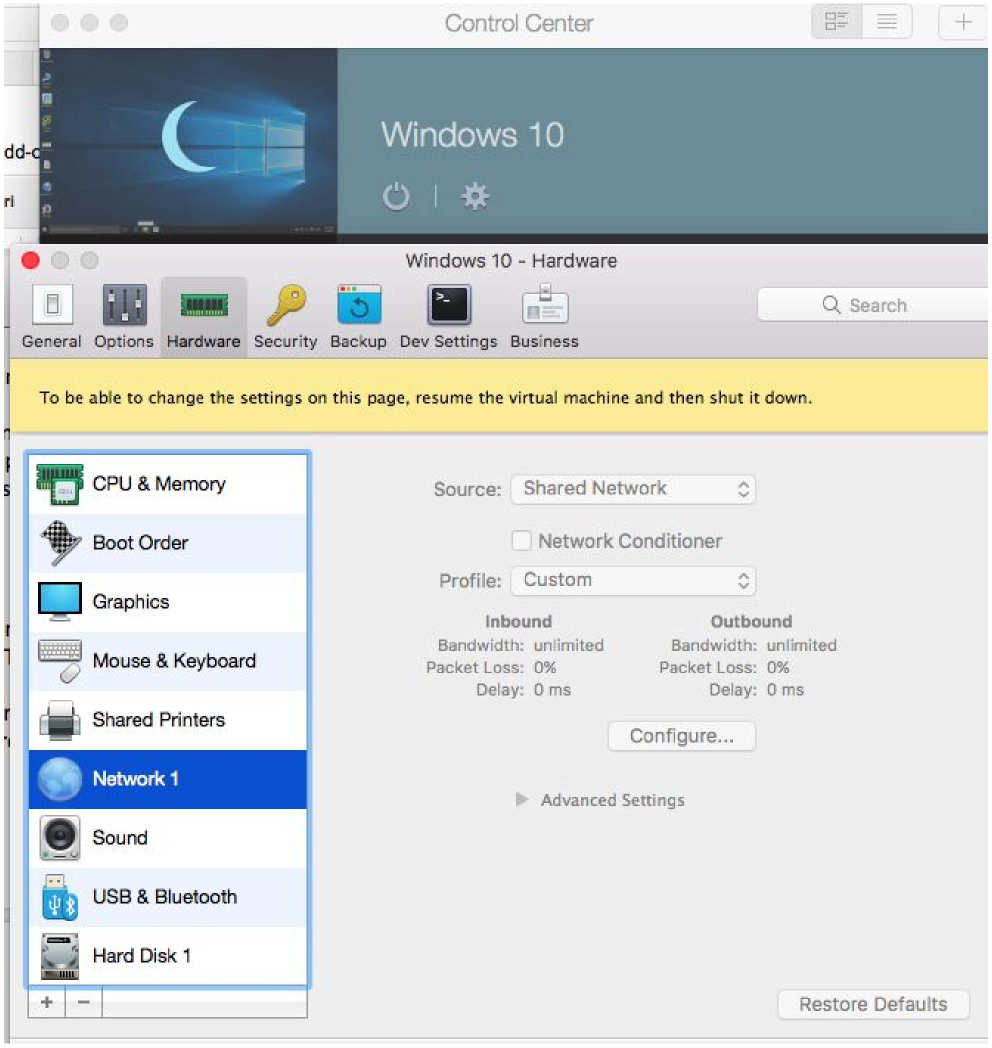
Server Address: 129.128.116.162
Account Name: bingzhu
Password:

☐ Show VPN status in menu bar

Figure 17.

- Operating systems

We need to configure a Windows virtual machine on the laptop which is most compatible to the platform for the remote server. And also, vSphere has no release by June, 2016 compatible on Mac OS. And the similar VMware has limit functions on Mac Os. To do this, a Windows 10 desktop is installed though Parallel Desktop as follows.


Control Center

Windows 10

Windows 10 - Hardware

To be able to change the settings on this page, resume the virtual machine and then shut it down.

Source: Shared Network

Network Conditioner

Profile: Custom

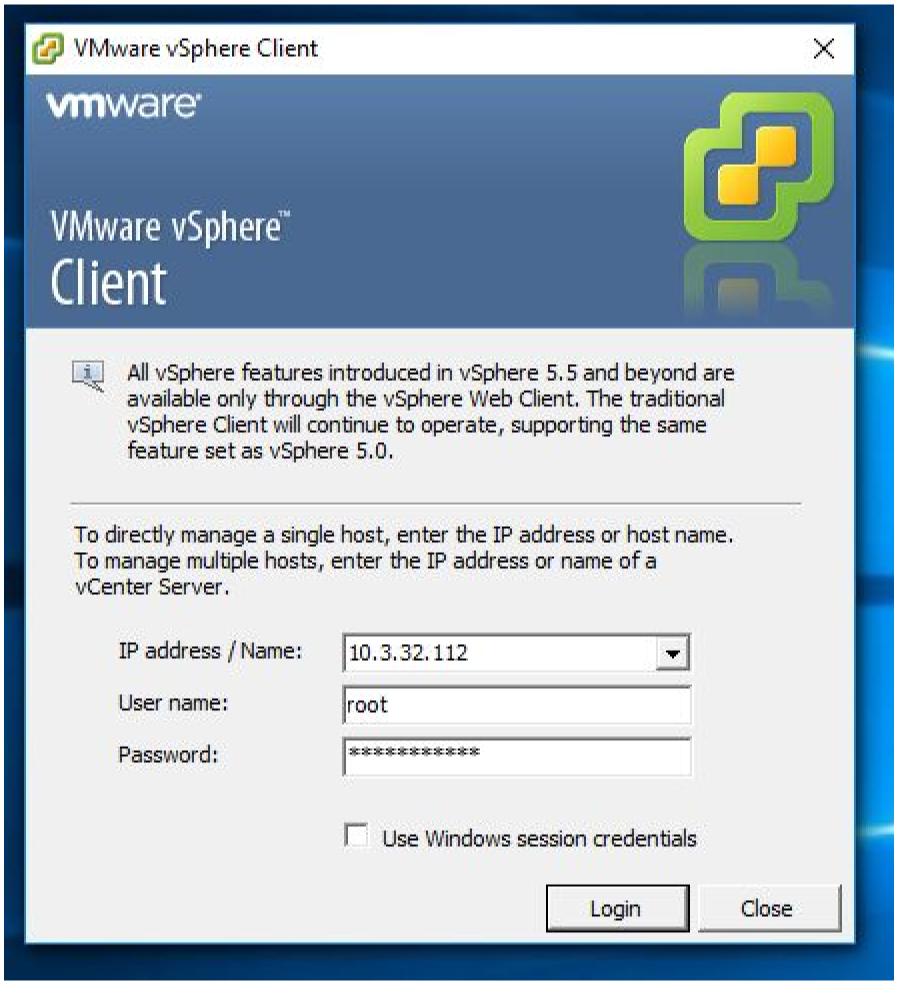
Inbound		Outbound	
Bandwidth:	unlimited	Bandwidth:	unlimited
Packet Loss:	0%	Packet Loss:	0%
Delay:	0 ms	Delay:	0 ms

Figure 18.

As you see, to give the Windows 10 access to the MINT lab network as well as the internet, we configure the guest operating system to share the same network with the host Mac OS.

- Server

The network function virtualization is done on a remote server, located on the MINT lab. The basic parameters and information are as follows.
We use VMware vSphere client as the platform to manage and configure the remote server. We need to install the VMware vSphere client on Windows and then configure
the remote server.
Log in to the server as the following shows. The IP address of the server is 10.3.32.112.


VMware vSphere Client

All vSphere features introduced in vSphere 5.5 and beyond are available only through the vSphere Web Client. The traditional vSphere Client will continue to operate, supporting the same feature set as vSphere 5.0.

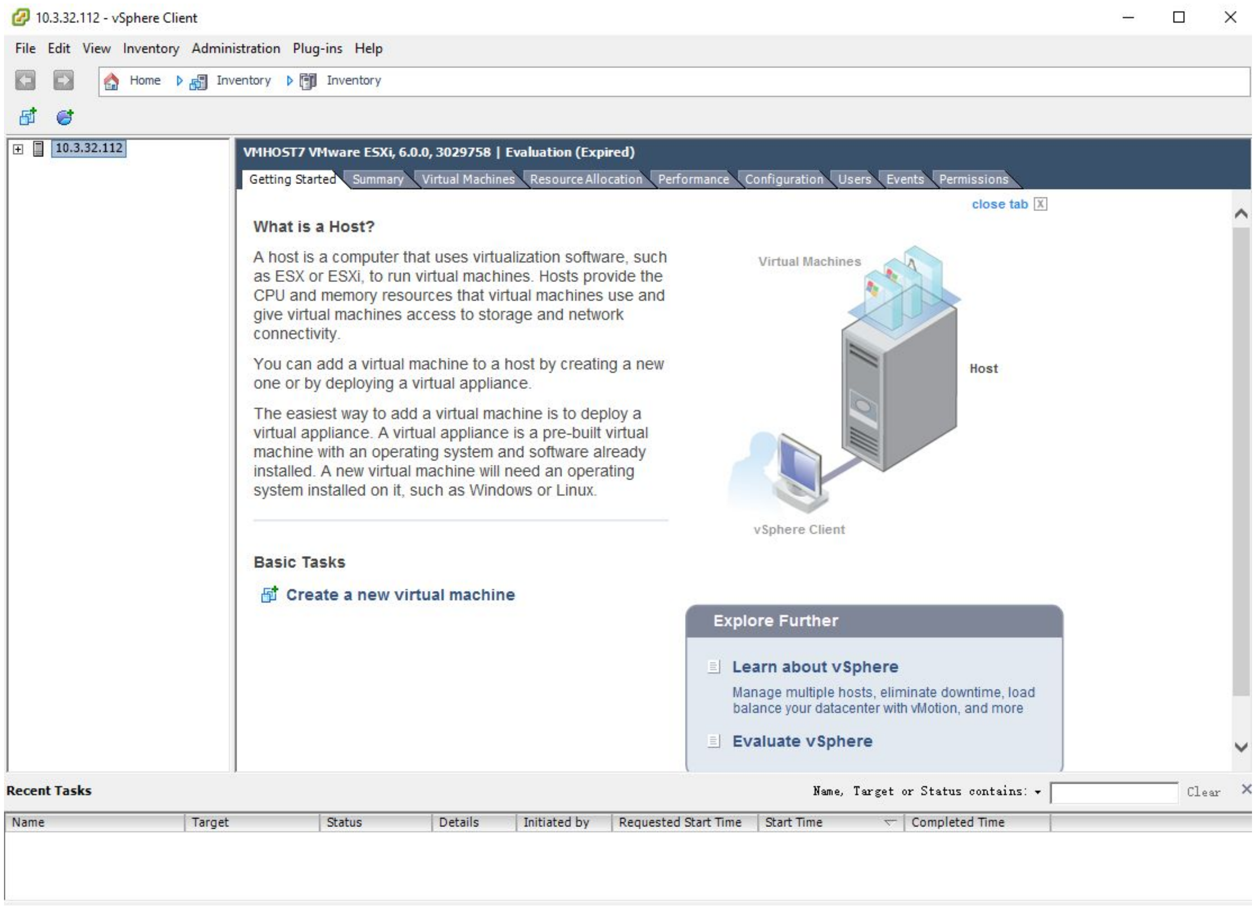
To directly manage a single host, enter the IP address or host name.
To manage multiple hosts, enter the IP address or name of a vCenter Server.

IP address / Name:	10.3.32.112
User name:	root
Password:	************

☐ Use Windows session credentials

Figure 19.

The related parameters are as follows.


10.3.32.112 - vSphere Client

File Edit View Inventory Administration Plug-ins Help

Home ▷ Inventory ▷ Inventory

10.3.32.112

VMHOST7 VMware ESXi, 6.0.0, 3029758 | Evaluation (Expired)

Getting Started | Summary | Virtual Machines | Resource Allocation | Performance | Configuration | Users | Events | Permissions

close tab

What is a Host?

A host is a computer that uses virtualization software, such as ESX or ESXi, to run virtual machines. Hosts provide the CPU and memory resources that virtual machines use and give virtual machines access to storage and network connectivity.

You can add a virtual machine to a host by creating a new one or by deploying a virtual appliance.

The easiest way to add a virtual machine is to deploy a virtual appliance. A virtual appliance is a pre-built virtual machine with an operating system and software already installed. A new virtual machine will need an operating system installed on it, such as Windows or Linux.

Basic Tasks

Create a new virtual machine

Explore Further

Learn about vSphere

Manage multiple hosts, eliminate downtime, load balance your datacenter with vMotion, and more

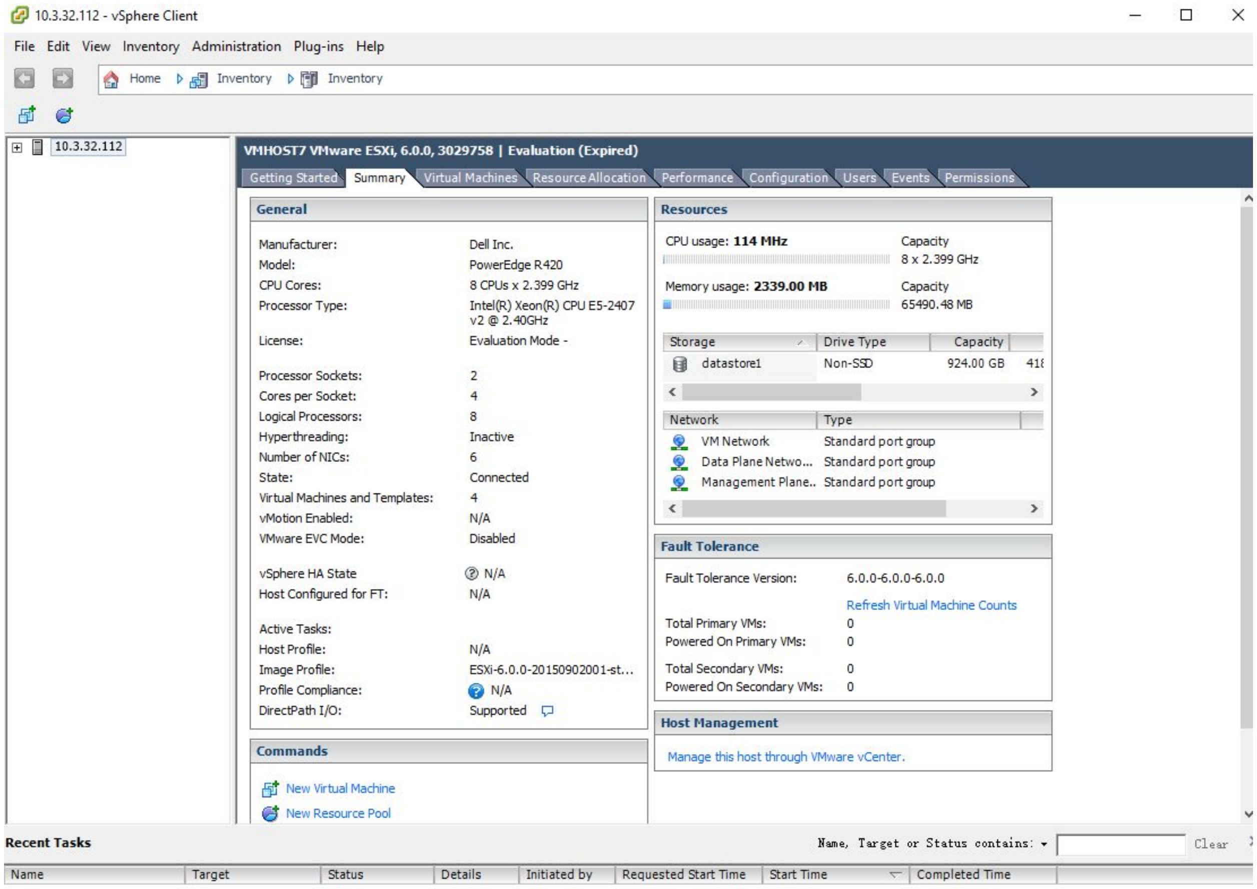
Evaluate vSphere

Recent Tasks

Name, Target or Status contains: ▾ Clear

Name	Target	Status	Details	Initiated by	Requested Start Time	Start Time	Completed Time

Figure 20.



10.3.32.112 - vSphere Client

File Edit View Inventory Administration Plug-ins Help

Home ▷ Inventory ▷ Inventory

10.3.32.112

VMHOST7 VMware ESXi, 6.0.0, 3029758 | Evaluation (Expired)

Getting Started | Summary | Virtual Machines | Resource Allocation | Performance | Configuration | Users | Events | Permissions

General

Manufacturer:	Dell Inc.
Model:	PowerEdge R420
CPU Cores:	8 CPUs x 2.399 GHz
Processor Type:	Intel(R) Xeon(R) CPU E5-2407 v2 @ 2.40GHz
License:	Evaluation Mode -
Processor Sockets:	2
Cores per Socket:	4
Logical Processors:	8
Hyperthreading:	Inactive
Number of NICs:	6
State:	Connected
Virtual Machines and Templates:	4
vMotion Enabled:	N/A
VMware EVC Mode:	Disabled
vSphere HA State	N/A
Host Configured for FT:	N/A
Active Tasks:	
Host Profile:	N/A
Image Profile:	ESXi-6.0.0-20150902001-st...
Profile Compliance:	N/A
DirectPath I/O:	Supported

Commands

New Virtual Machine
New Resource Pool

Resources

CPU usage: **114 MHz** Capacity 8 x 2.399 GHz

Memory usage: **2339.00 MB** Capacity 65490.48 MB

Storage	Drive Type	Capacity	
datastore1	Non-SSD	924.00 GB	41

Network	Type
VM Network	Standard port group
Data Plane Netwo...	Standard port group
Management Plane..	Standard port group

Fault Tolerance

Fault Tolerance Version:	6.0.0-6.0.0-6.0.0
	Refresh Virtual Machine Counts
Total Primary VMs:	0
Powered On Primary VMs:	0
Total Secondary VMs:	0
Powered On Secondary VMs:	0

Host Management

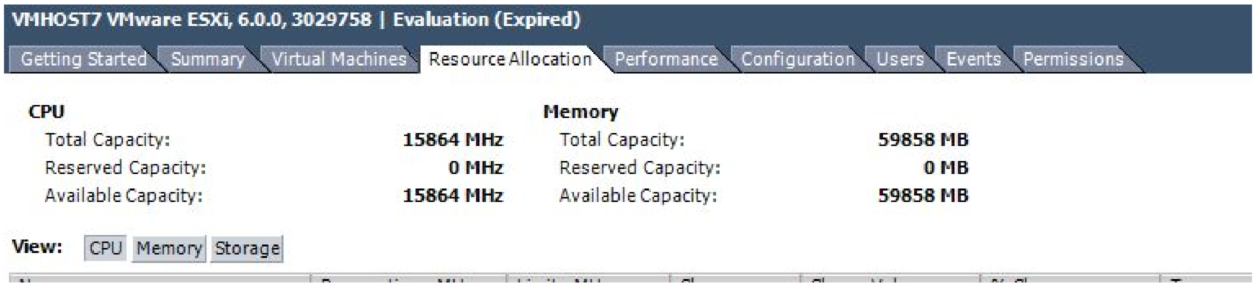
Manage this host through VMware vCenter.

Recent Tasks

Name, Target or Status contains: Clear

Name	Target	Status	Details	Initiated by	Requested Start Time	Start Time	Completed Time

Figure 21.



VMHOST7 VMware ESXi, 6.0.0, 3029758 | Evaluation (Expired)

Getting Started | Summary | Virtual Machines | Resource Allocation | Performance | Configuration | Users | Events | Permissions

CPU		**Memory**	
Total Capacity:	**15864 MHz**	Total Capacity:	**59858 MB**
Reserved Capacity:	**0 MHz**	Reserved Capacity:	**0 MB**
Available Capacity:	**15864 MHz**	Available Capacity:	**59858 MB**

View: CPU Memory Storage

Figure 22.

3.2 Software Preparation

- Local virtualization tools

  The tools we need to prepare and upload to the server are Ubuntu 16.04 server installation package, VM tools and related linux tools.

- Software installation tools

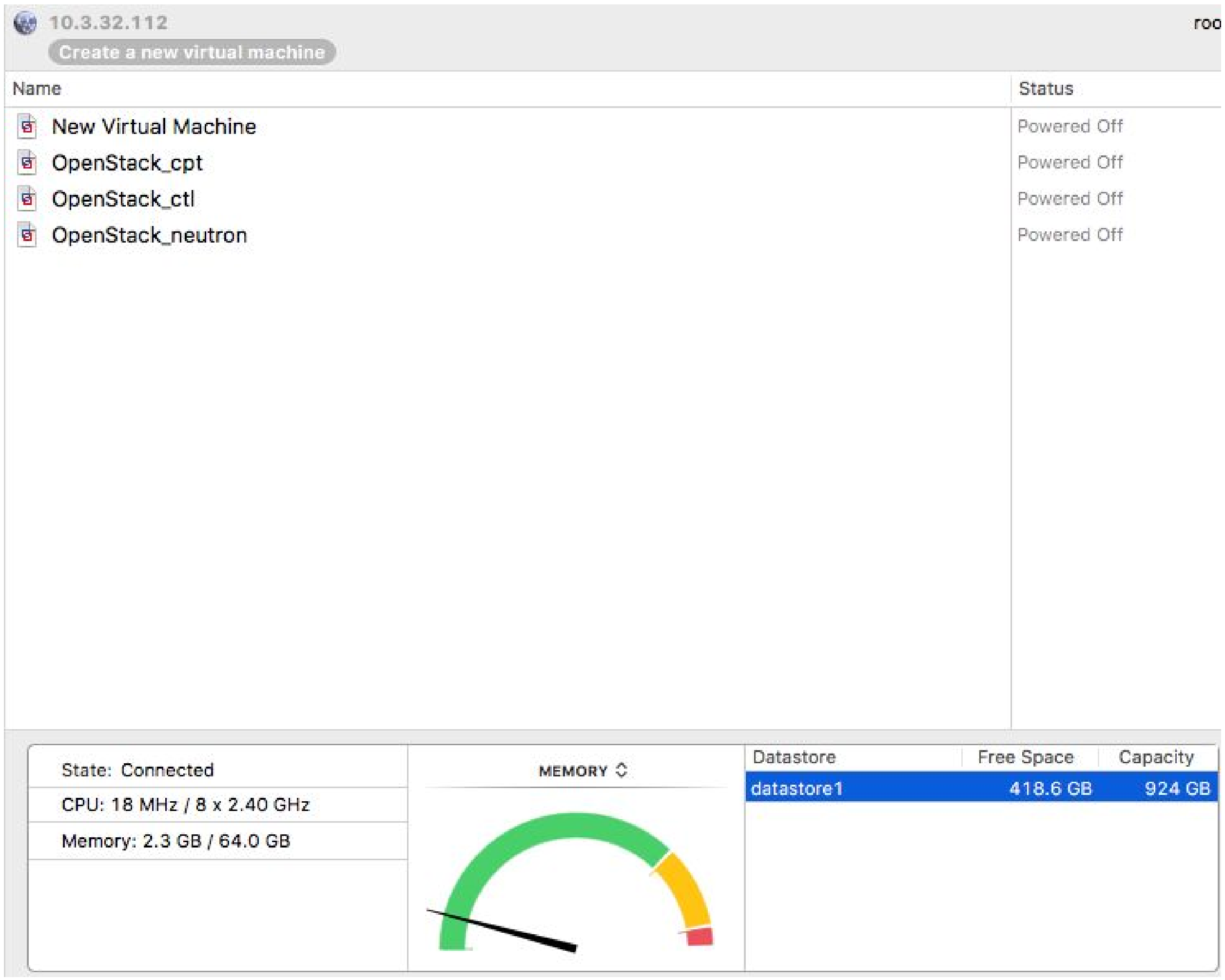
In this lab, we need to install OpenDaylight and OpenStack on several nodes. Therefore, installation packages are needed and installation tools as well as internet access are necessary.
For OpenDayLight, we choose OpenDayLight Beryllium SR2 for Ubuntu 16.04 LTS, the most updated version by August, 2016.
For OpenStack, we choose OpenStack Mitaka for Ubuntu 16.04 LTS, the most updated version by October, 2016.

To set up the remote server, I followed the instructions and set the parameters as follows. It ‘s an ESXi 6 version server.

Type	Linux
Operating systems	Ubuntu (64 bits)
Sockets in total	8 cores
Virtual sockets for each at least	2 cores
Memory	4 GB
Network Interface Cards	6 (located in the back rack physically)
Disk size	Above 20 GB

Table 1.

Then we need to upload packages and files to the datastore in the server, which includes virtual drives such as floppy drives that can be inserted to or ejected from the VMs that we are going to configure.
The overall information of the server is as follows.



10.3.32.112

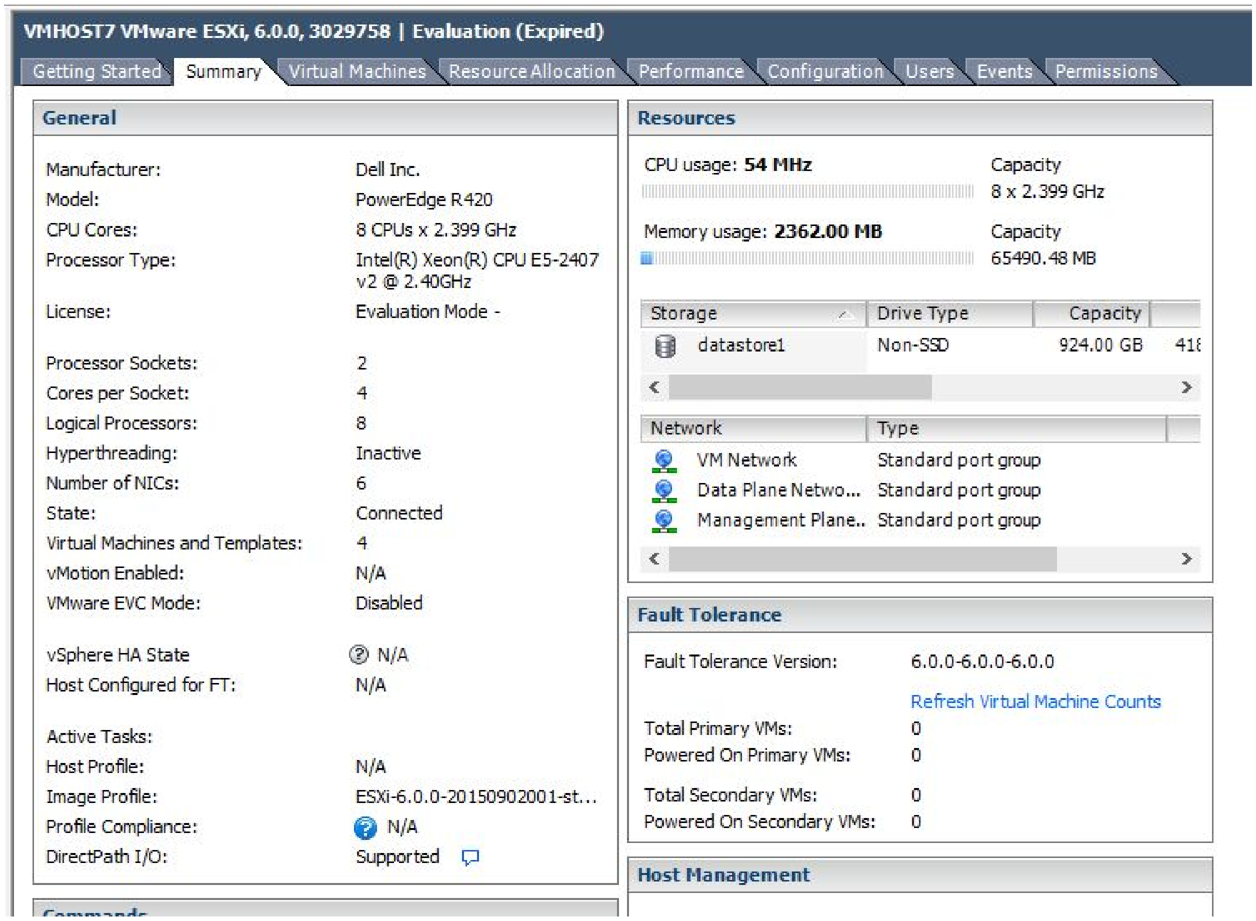
roo

Name	Status
New Virtual Machine	Powered Off
OpenStack_cpt	Powered Off
OpenStack_ctl	Powered Off
OpenStack_neutron	Powered Off

State: Connected
CPU: 18 MHz / 8 x 2.40 GHz
Memory: 2.3 GB / 64.0 GB

Datastore	Free Space	Capacity
datastore1	418.6 GB	924 GB

Figure 23.


VMHOST7 VMware ESXi, 6.0.0, 3029758 | Evaluation (Expired)

Getting Started | Summary | Virtual Machines | Resource Allocation | Performance | Configuration | Users | Events | Permissions

General

Manufacturer:	Dell Inc.
Model:	PowerEdge R420
CPU Cores:	8 CPUs x 2.399 GHz
Processor Type:	Intel(R) Xeon(R) CPU E5-2407 v2 @ 2.40GHz
License:	Evaluation Mode -
Processor Sockets:	2
Cores per Socket:	4
Logical Processors:	8
Hyperthreading:	Inactive
Number of NICs:	6
State:	Connected
Virtual Machines and Templates:	4
vMotion Enabled:	N/A
VMware EVC Mode:	Disabled
vSphere HA State	N/A
Host Configured for FT:	N/A
Active Tasks:	
Host Profile:	N/A
Image Profile:	ESXi-6.0.0-20150902001-st...
Profile Compliance:	N/A
DirectPath I/O:	Supported

Resources

CPU usage: **54 MHz** Capacity 8 x 2.399 GHz

Memory usage: **2362.00 MB** Capacity 65490.48 MB

Storage	Drive Type	Capacity	
datastore1	Non-SSD	924.00 GB	418

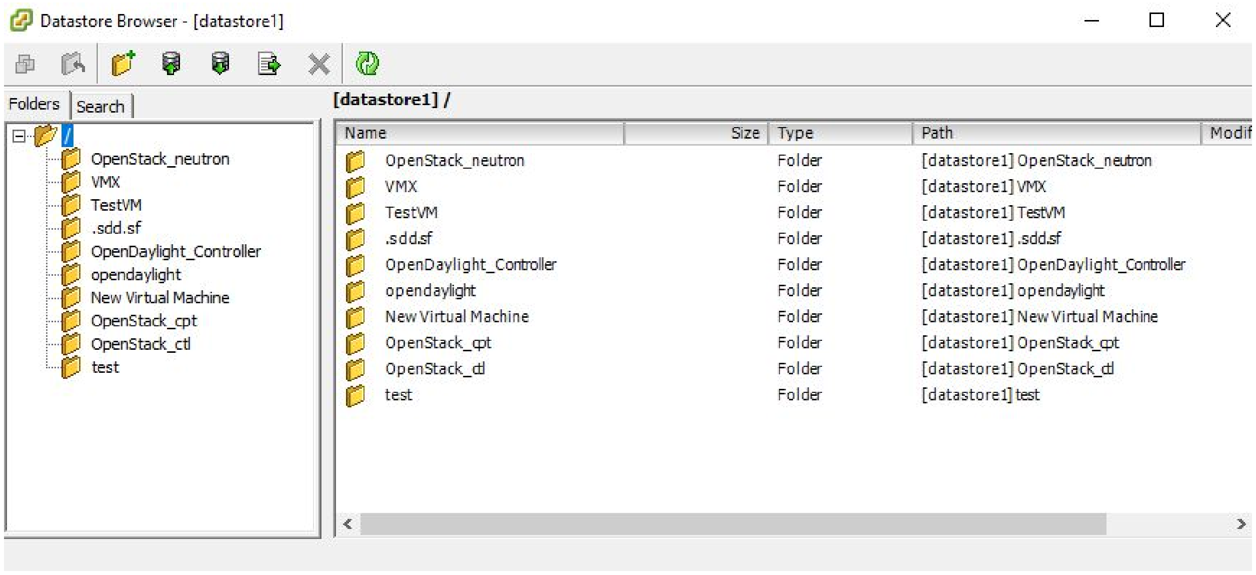
Network	Type
VM Network	Standard port group
Data Plane Netwo...	Standard port group
Management Plane..	Standard port group

Fault Tolerance

Fault Tolerance Version:	6.0.0-6.0.0-6.0.0
	Refresh Virtual Machine Counts
Total Primary VMs:	0
Powered On Primary VMs:	0
Total Secondary VMs:	0
Powered On Secondary VMs:	0

Host Management

Figure 24.

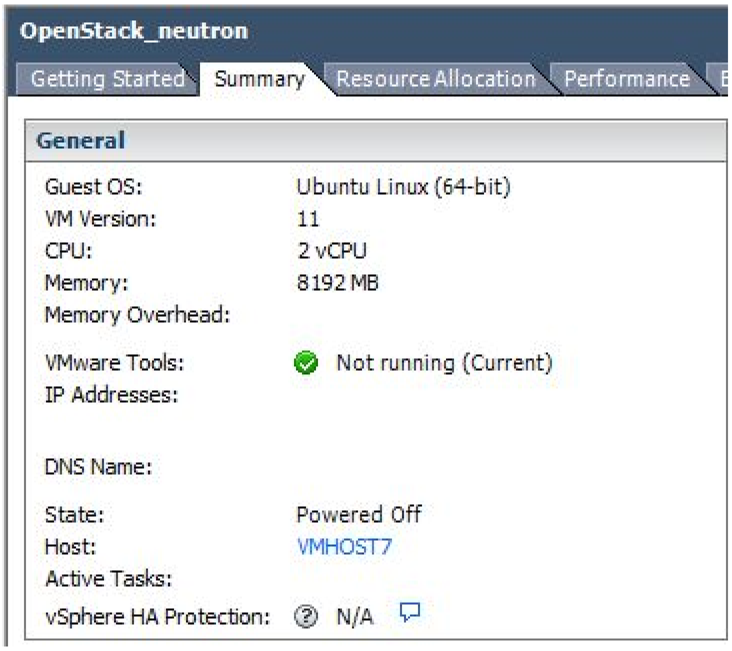

Datastore Browser - [datastore1]

[datastore1] /

Name	Size	Type	Path	Modif
OpenStack_neutron		Folder	[datastore1] OpenStack_neutron	
VMX		Folder	[datastore1] VMX	
TestVM		Folder	[datastore1] TestVM	
.sdd.sf		Folder	[datastore1] .sdd.sf	
OpenDaylight_Controller		Folder	[datastore1] OpenDaylight_Controller	
opendaylight		Folder	[datastore1] opendaylight	
New Virtual Machine		Folder	[datastore1] New Virtual Machine	
OpenStack_cpt		Folder	[datastore1] OpenStack_cpt	
OpenStack_ctl		Folder	[datastore1] OpenStack_ctl	
test		Folder	[datastore1] test	

Figure 25.

We also need to install VM tools to improve operation towards the virtual machines. When the VM tools are successfully installed, there would be icon indicating the VM tools are “running” or “not running”in green.


OpenStack_neutron

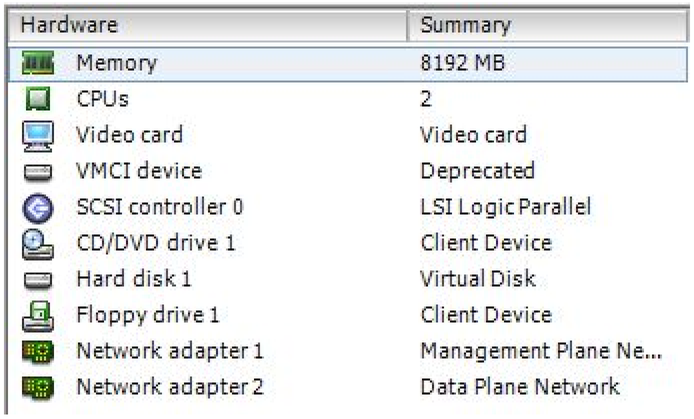
General

Guest OS:	Ubuntu Linux (64-bit)
VM Version:	11
CPU:	2 vCPU
Memory:	8192 MB
Memory Overhead:	
VMware Tools:	✓ Not running (Current)
IP Addresses:	
DNS Name:	
State:	Powered Off
Host:	VMHOST7
Active Tasks:	
vSphere HA Protection:	N/A

Figure 26.

3.2.1 Virtual machines Setup

We need to add corresponding amount of virtual machines to the server, to the network diagram. For each Ubuntu server, the parameters are as follows. (disk size is recommended as 100 GB)



Hardware	Summary
Memory	8192 MB
CPUs	2
Video card	Video card
VMCI device	Deprecated
SCSI controller 0	LSI Logic Parallel
CD/DVD drive 1	Client Device
Hard disk 1	Virtual Disk
Floppy drive 1	Client Device
Network adapter 1	Management Plane Ne...
Network adapter 2	Data Plane Network

Figure 27.

3.2.2 Operating systems Installation

We use Ubuntu server 16.04 LTS and Ubuntu server 14.04 LTS in this lab. The parameters we use in the process of installation is as follows.

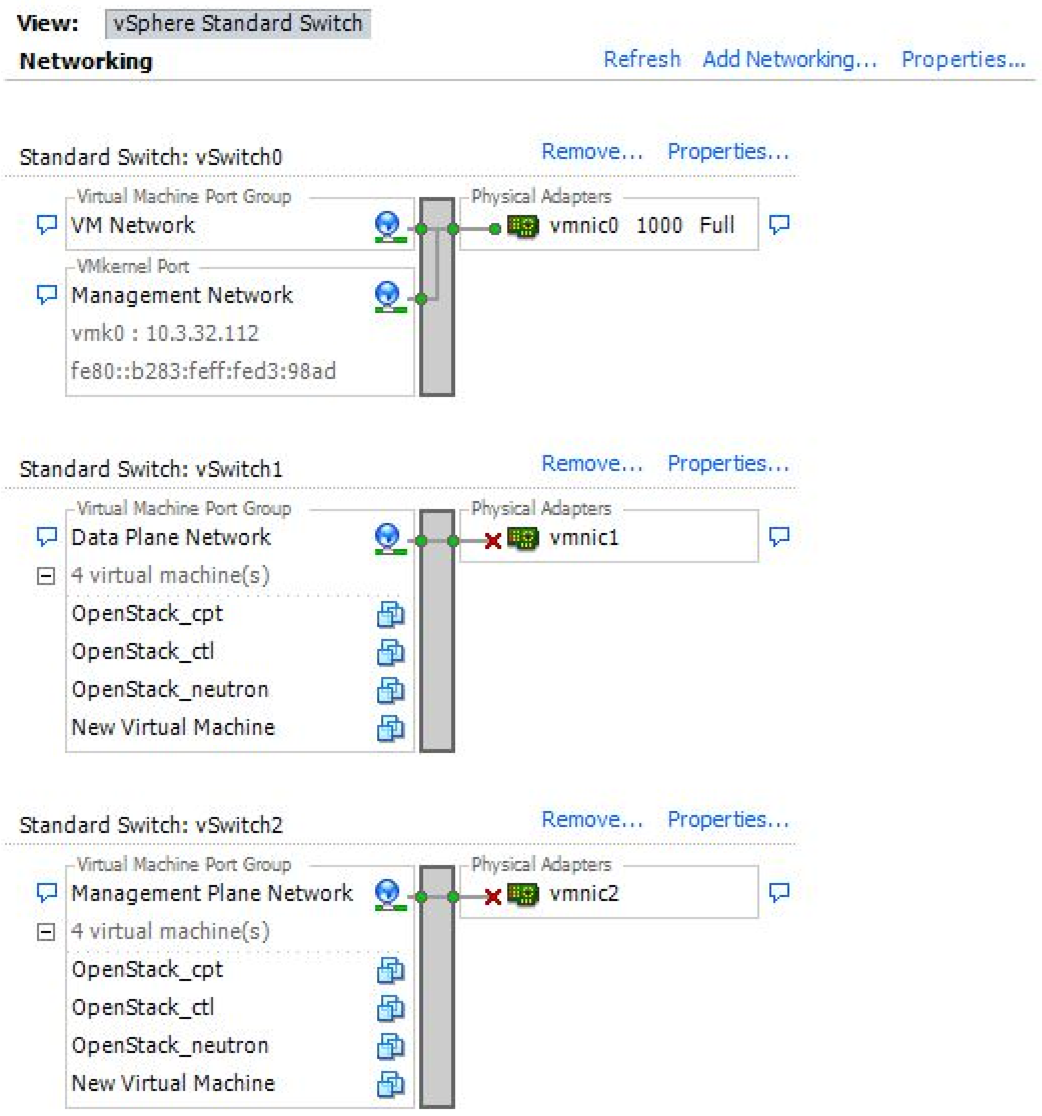
- Create a usable image. I use a ".iso" file in this lab
- In BIOS, select English and enter
- Choose Ubuntu Server and enter
- Select English and Canada in the following pages
- Choose "no" to detection keyboard layout
- Select English in the following pages and enter
- Set hostname and configure network manually
- Set the username and password
- Choose "no" to encrypt the home directory
- Choose "yes" to the time zone correct inquiry
- Select "Guided-use entire disk and set up LVM"
- Choose "yes" to write changes to disk and configure LVM
- Select "no automatic updates"
- Choose "yes" to the GRUB boot loader
- Finish installation

3.2.3 Software Installation

According to the requirements in this lab, we need to install OpenDaylight and OpenStack nodes on VMs separately. The configuration is as follows. This part of configuration will be demonstrated in the next chapter with a use case.

3.2.4 Network Setup

Overall, we need two other networks apart from the existing one which is used to manage the virtual machines. The two networks are shown as follows.


View: vSphere Standard Switch

Networking

Standard Switch: vSwitch0

Virtual Machine Port Group: VM Network

VMkernel Port: Management Network
vmk0 : 10.3.32.112
fe80::b283:feff:fed3:98ad

Physical Adapters: vmnic0 1000 Full

Standard Switch: vSwitch1

Virtual Machine Port Group: Data Plane Network
4 virtual machine(s)
OpenStack_cpt
OpenStack_ctl
OpenStack_neutron
New Virtual Machine

Physical Adapters: vmnic1

Standard Switch: vSwitch2

Virtual Machine Port Group: Management Plane Network
4 virtual machine(s)
OpenStack_cpt
OpenStack_ctl
OpenStack_neutron
New Virtual Machine

Physical Adapters: vmnic2

Figure 28.

4. Solution Setup

4.1 Lab Setup

In this stage, we need to firstly decide on what network topology is going to be configured. Secondly, we need to design the controller’s networks and all the other nodes’ networks, including ones that they use to communicate with each other. Thirdly, we need to install all the features that the OpenDaylight controller requires, which is necessary that we figure out all the features' functionalities. Last but not least, we need to integrate the controller with all the other nodes

4.1.1 Lab Network Topology

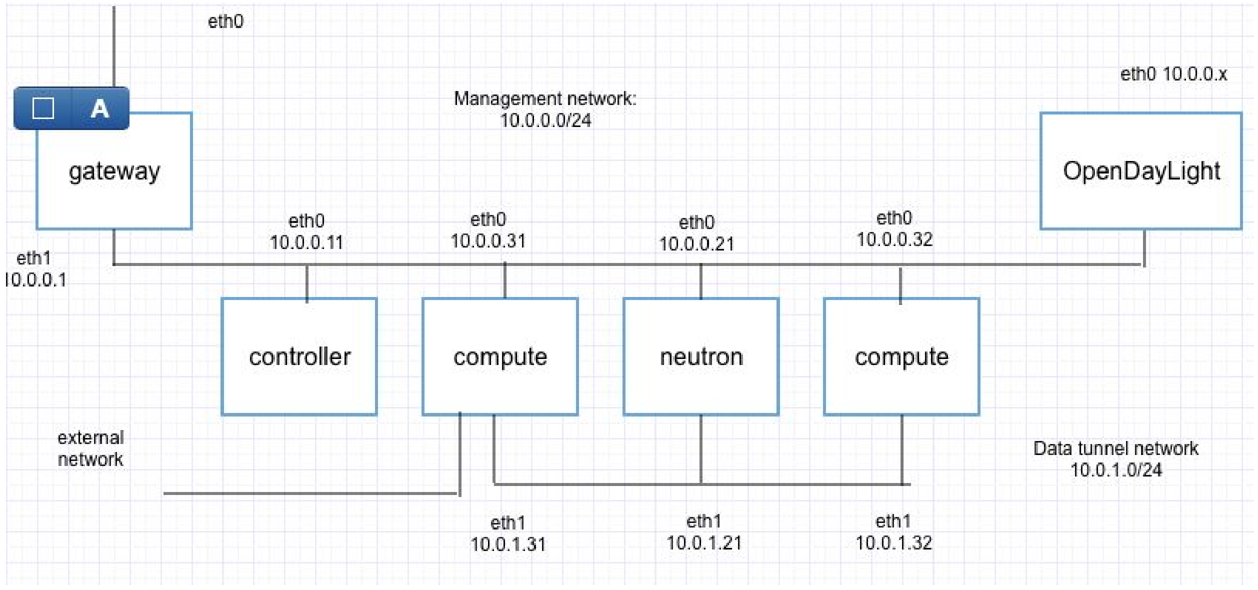


Figure 29. Lab Network Diagram

4.1.2 OpenDayLight features Setup

Before we install OpenDaylight on the Ubuntu server, several things need to be done.

A few tools need to be installed first: Maven, Git,OSGi, JAVA interfaces, and REST APIs.
As for maven, OpenDaylight uses Apache Maven for building all projects. We need to install 3.3.1 or later versions. OpenDayLight maintains all its codes in Git repositories. A Java 7 or later releases is needed for OpenDayLight installation.

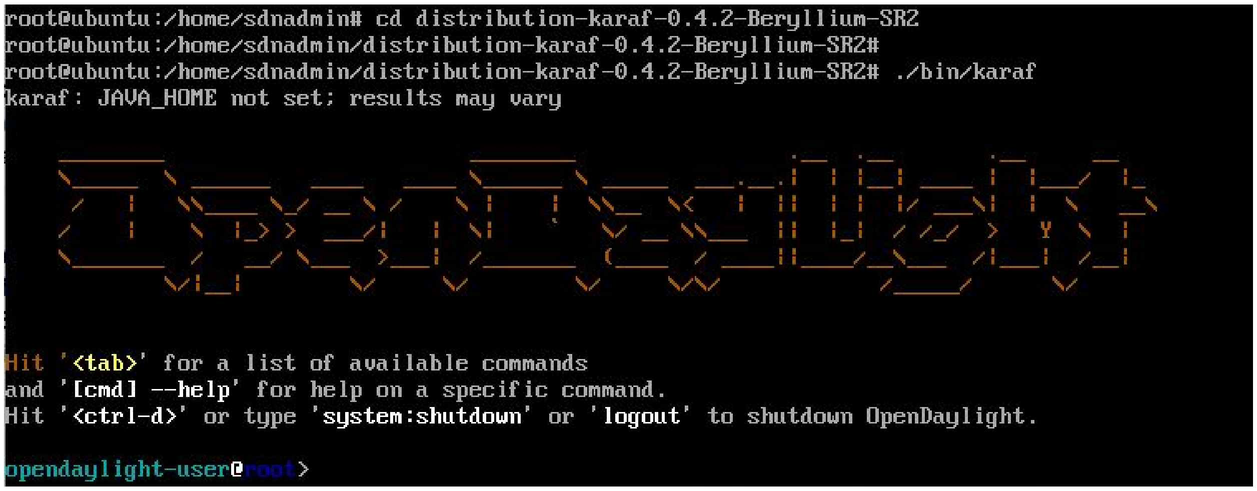
In this project, we choose OpenDayLight Beryllium SR2 as the controller in the network. The hostname of this VM is *ubuntuodlsdnadmin*, username is *sdnadmin*, password is *sdnlabs#112*

The following features are necessary and needs to be installed in OpenDayLight.

Features	Karaf features name
L2SWITCH	odl-l2switch-switch-ui
OF-CONFIG	odl-of-config-rest
PCEP	odl-bgpcep-pcep
OpenFLow Flow Programming	odl-openflowplugin-flow-services-ui
OVSDB Southbound	odl-ovsdb-southbound-impl-ui
OVSDB HWVTEP Southbound	odl-ovsdb-hwvtepsouthbound-ui
OVSDB NetVirt SFC	odl-ovsdb-sfc-ui
OpenFlow Table Type Patterns	odl-ttp-all
DLUX	odl-dlux-all
centinel	odl-centinel-all
REST API	odl-restconf
MD-SAL	odl-mdsal-clustering

Table 2.

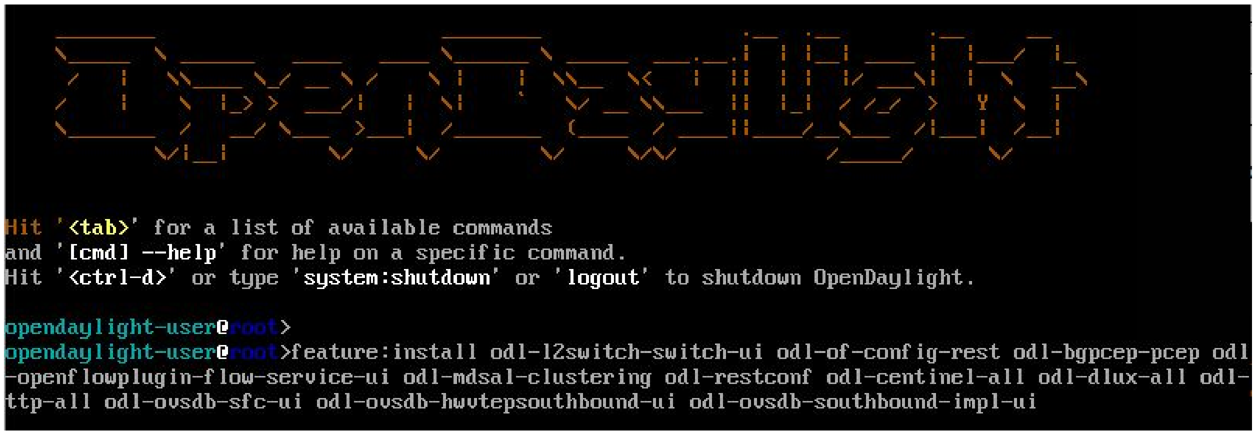
The following shows the successful installation of OpenDayLight and karaf.



```
root@ubuntu:/home/sdnadmin# cd distribution-karaf-0.4.2-Beryllium-SR2
root@ubuntu:/home/sdnadmin/distribution-karaf-0.4.2-Beryllium-SR2#
root@ubuntu:/home/sdnadmin/distribution-karaf-0.4.2-Beryllium-SR2# ./bin/karaf
karaf: JAVA_HOME not set; results may vary

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>
```

Figure 30.

The following shows the installation of OpenDayLight features.



```
Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>
opendaylight-user@root>feature:install odl-l2switch-switch-ui odl-of-config-rest odl-bgpcep-pcep odl
-openflowplugin-flow-service-ui odl-mdsal-clustering odl-restconf odl-centinel-all odl-dlux-all odl-
ttp-all odl-ovsdb-sfc-ui odl-ovsdb-hwvtepsouthbound-ui odl-ovsdb-southbound-impl-ui
```

Figure 31.

Keep this OpenDayLight controller on and continue to configure OpenStack nodes.

4.1.3 OpenStack nodes Setup

The OpenStack project is an open source cloud computing platform that supports all types of cloud environment. It provides an Infrastructure-as-a-Service (Iaas) solution through a variety of complemental services. An Iaas is a provisioning model in which an organization outsources physical components of a data centre, such as storage, hardware, servers, and networking components.

Basically, a completed OpenStack network requires at least two nodes, one controller node for identifying service, image service, management portion of computing and SQL database and one compute node for hypervisor portion of computing. Besides, block node is an optional node. An example architecture is as follows.

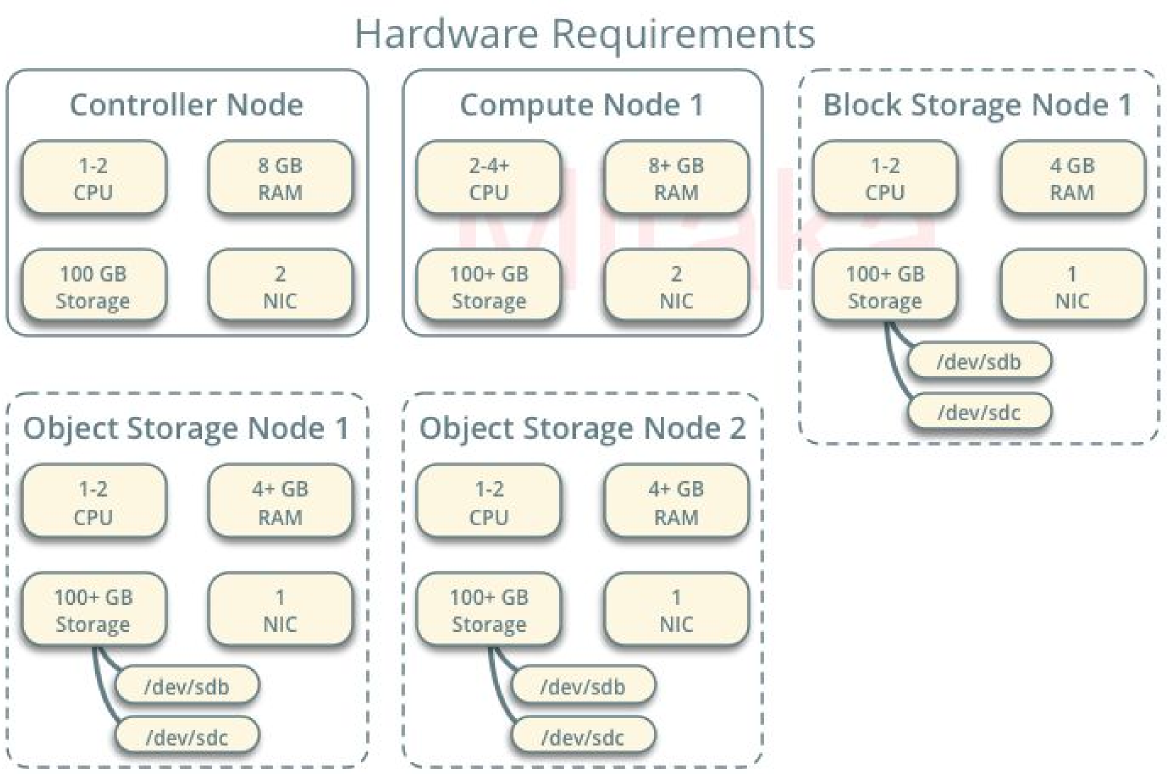


Figure 32. Example architecture
(quoted from http://docs.openstack.org/mitaka/install-guide-ubuntu/overview.html)

There is point that needs to be clear. OpenStack provides several services and some of the service has its nickname which sometimes the nodes in the network could also be called like. The following table shows the services and their nicknames.

Service	Project name
Dashboard	Horizon
Compute	Nova
Networking	Neutron
Object Storage	Swift
Block Storage	Cinder
Identity service	Keystone
Image service	Glance
Telemetry	Ceilometer

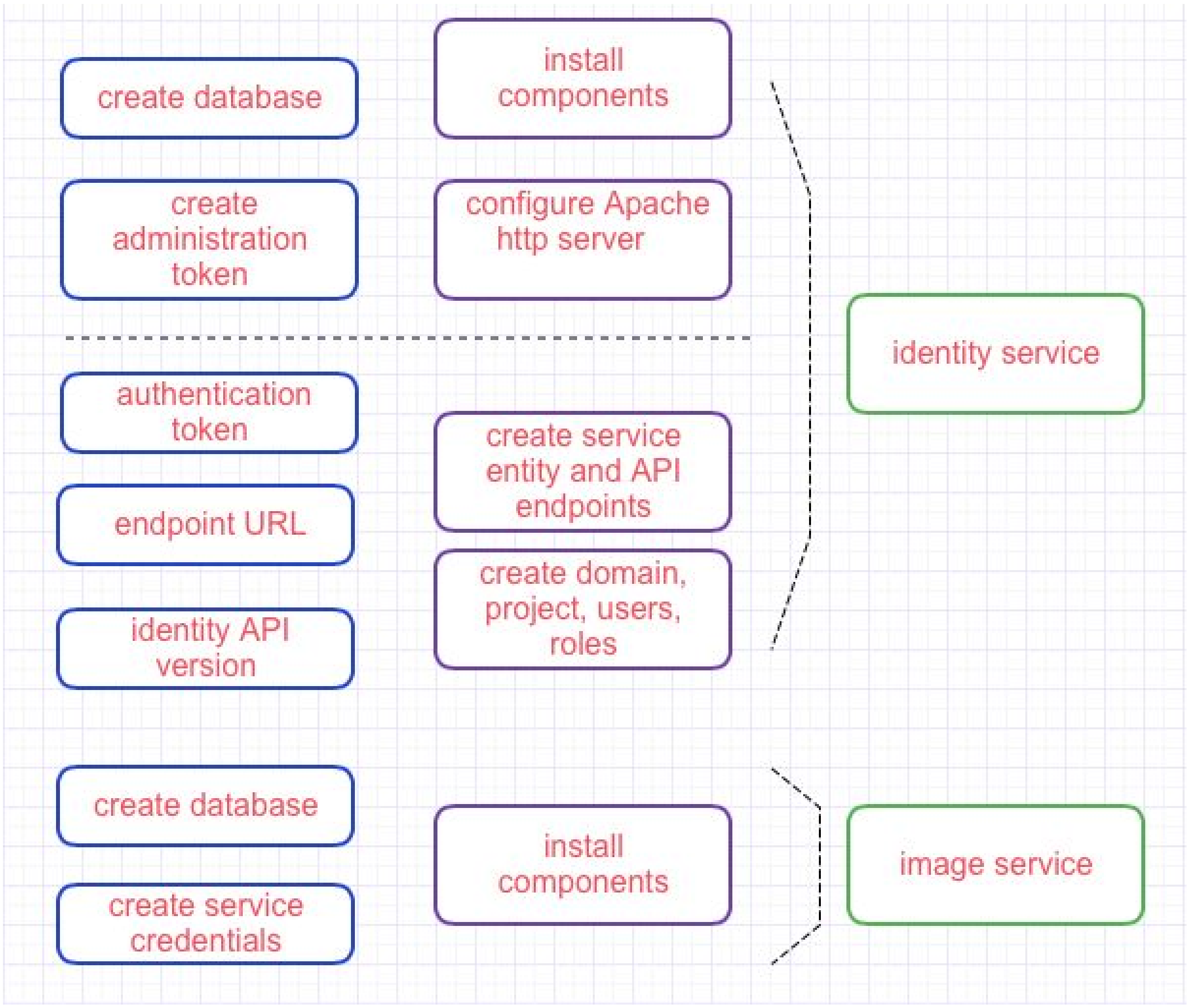
Orchestration	Heat

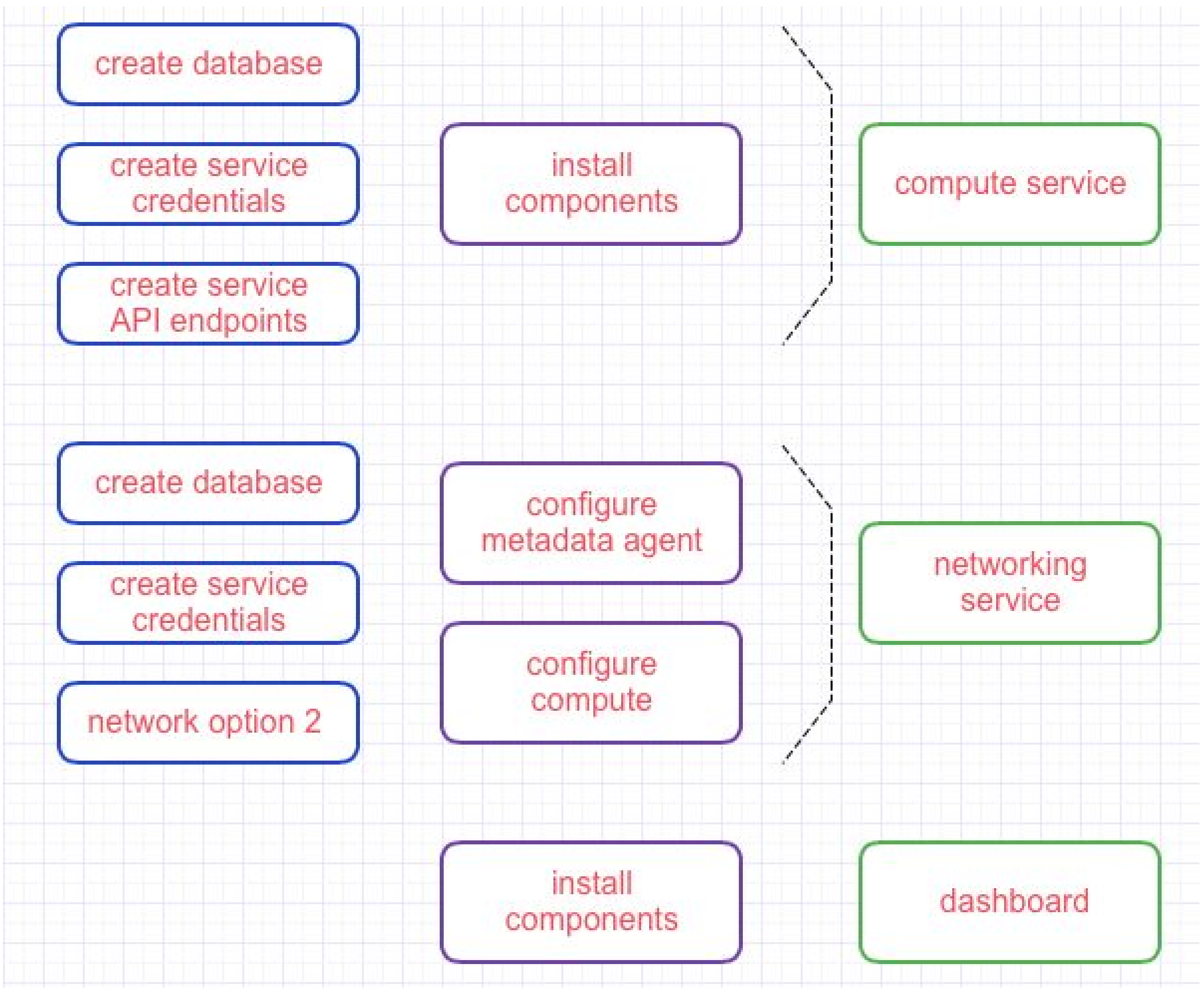
Table 3.

We will start to configure network when we finished configuring these nodes.

- Controller node

  In order to demonstrate the configuration processes clearly, I create the following flow chart.


create database
install components
create administration token
configure Apache http server
identity service
authentication token
create service entity and API endpoints
endpoint URL
identity API version
create domain, project, users, roles
create database
install components
image service
create service credentials


create database
create service credentials
create service API endpoints
install components
compute service
create database
create service credentials
network option 2
configure metadata agent
configure compute
networking service
install components
dashboard

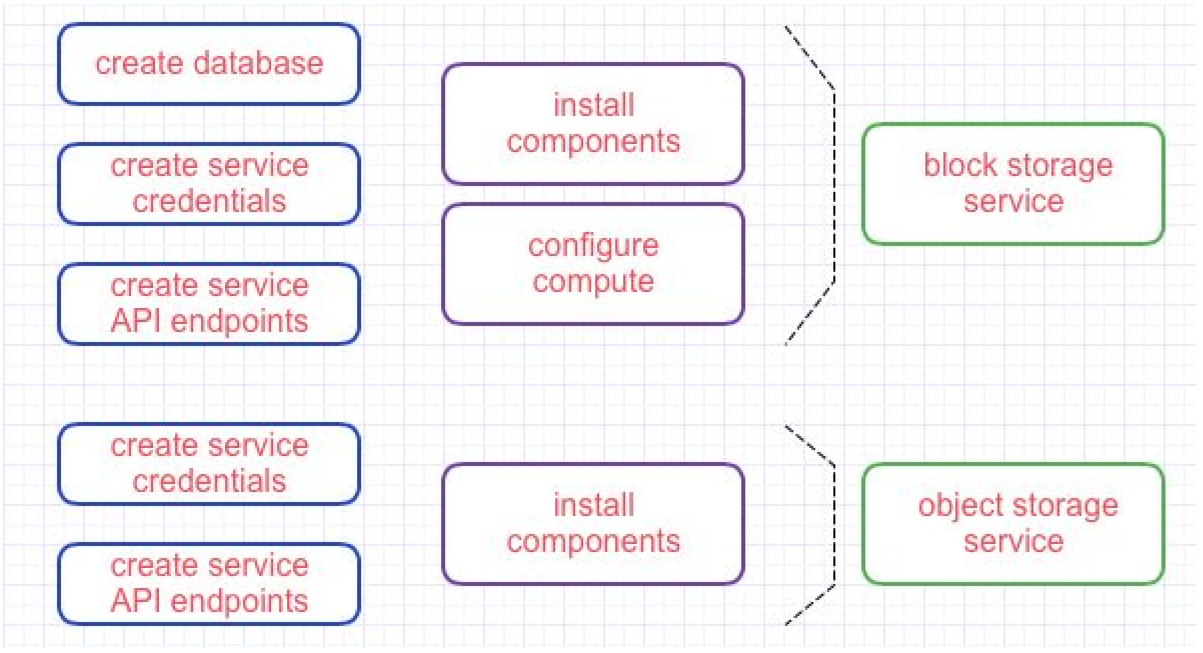

create database
create service credentials
create service API endpoints
install components
configure compute
block storage service
create service credentials
create service API endpoints
install components
object storage service

Figure 33.

The following is the user identifications and passwords used in each of the service. The services and features need to be configured one by one. When it comes to systematically configuration, it requires going into the configuration file to correct or add each and every options and commands. The operation is on the linux operating system and all the commands should be in the console.

Service or database	User ID and password
MariaDB	root / sdnlabs#112
Message queue	Openstack user / RABBIT_PASS
identify	Database keystone / KEYSTONE_DBPASS
Temporary admin_token	ca7eb37670a0352ad2a0
networking	admin / sdnlabs#112
image	GLANCE_DBPASS
glance	sdnlabs#112

Table 4.

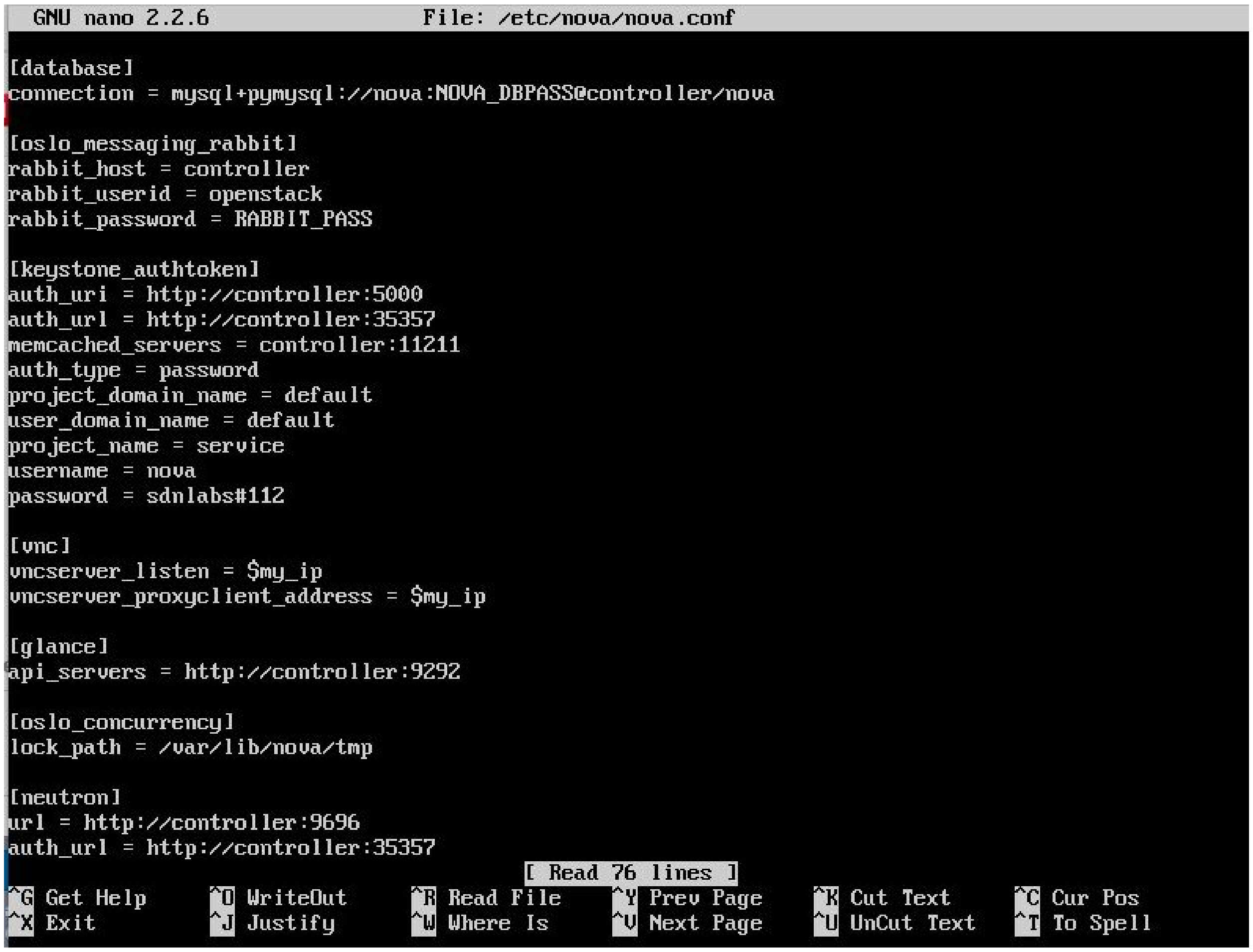
Here are the issues I recorded during configuration. Since the OpenStack Mitaka version is the newest release, there are several issues in the documentation from the official website. The following issue records have also been provided by the author to the website.

issue	reason	solution
Cannot create service entity and API endpoints (HTTP 500)	The keystone/keystone.conf file has wrong typing	Correct the mistaken line and comment those not in use
HTTP 404	Only restart the apache2 server as suggested in the documentation is not enough	Reboot the operating system to make it get the configuration active
Command not found in OpenStack	There are several commands which only active in OpenStack when activated	Type the command:apt-get install python-openstackclient
Cannot verify operation as the admin user (HTTP 5000)	Service not activated	Service apache2 restart
There's no "database" section in	It's normal but ignored by the	Add this part manually to the

/etc/nova/nova.conf.file	website as of October	file
Cannot connect to http://controller horizon	The OS is a server version and the browser opened is the host's	Open the browser in terminal by the command: w3m <http….>

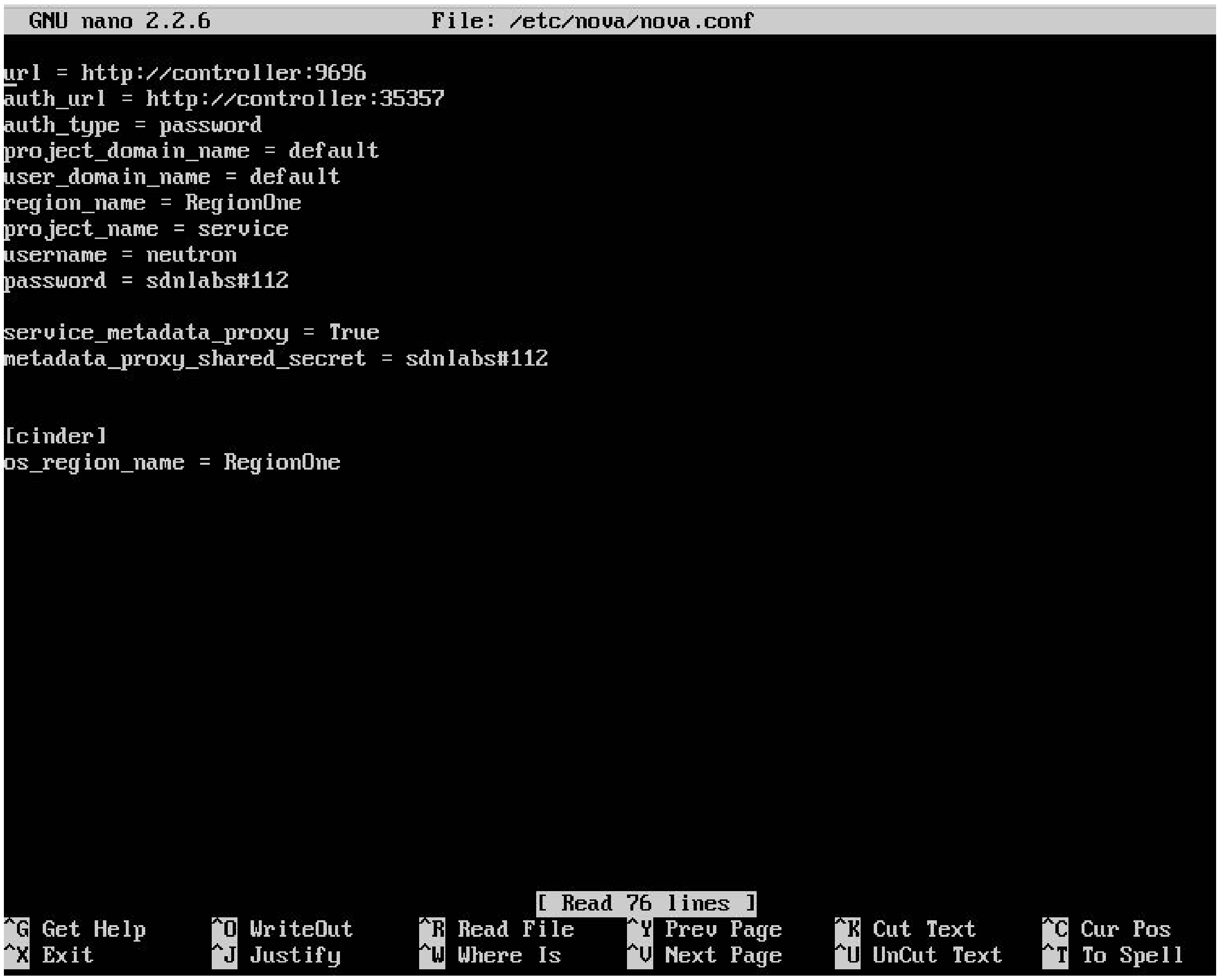
Table 5.

Services configured on controller node requires configuration in each file. For nova service (compute service), /etc/nova/nova.conf needs to be edited as follows. The configuration should includes the proxies, tokens the service use to communicate with other nodes, as well as other supporting services it may require.



```
  GNU nano 2.2.6              File: /etc/nova/nova.conf

[database]
connection = mysql+pymysql://nova:NOVA_DBPASS@controller/nova

[oslo_messaging_rabbit]
rabbit_host = controller
rabbit_userid = openstack
rabbit_password = RABBIT_PASS

[keystone_authtoken]
auth_uri = http://controller:5000
auth_url = http://controller:35357
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = nova
password = sdnlabs#112

[vnc]
vncserver_listen = $my_ip
vncserver_proxyclient_address = $my_ip

[glance]
api_servers = http://controller:9292

[oslo_concurrency]
lock_path = /var/lib/nova/tmp

[neutron]
url = http://controller:9696
auth_url = http://controller:35357
                                        [ Read 76 lines ]
^G Get Help     ^O WriteOut     ^R Read File    ^Y Prev Page    ^K Cut Text     ^C Cur Pos
^X Exit         ^J Justify      ^W Where Is     ^V Next Page    ^U UnCut Text   ^T To Spell
```

Figure 34.

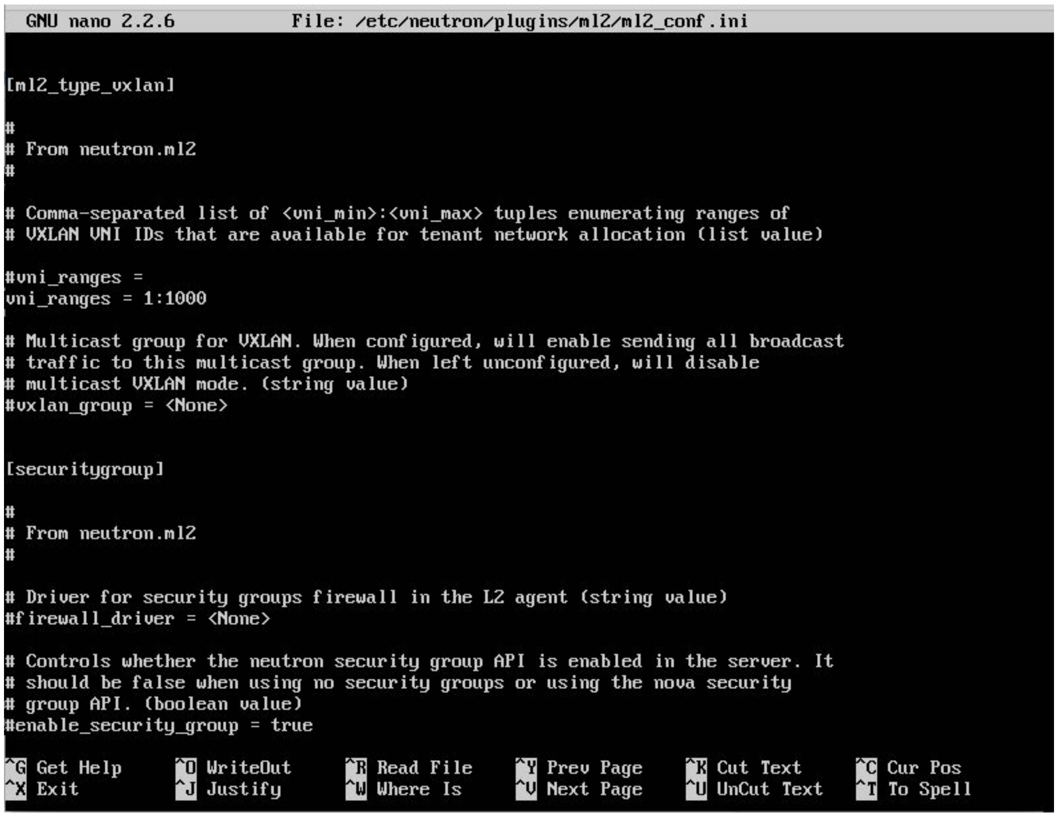


```
  GNU nano 2.2.6                File: /etc/nova/nova.conf

url = http://controller:9696
auth_url = http://controller:35357
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = neutron
password = sdnlabs#112

service_metadata_proxy = True
metadata_proxy_shared_secret = sdnlabs#112


[cinder]
os_region_name = RegionOne
```

```
                                        [ Read 76 lines ]
^G Get Help     ^O WriteOut     ^R Read File    ^Y Prev Page    ^K Cut Text     ^C Cur Pos
^X Exit         ^J Justify      ^W Where Is     ^V Next Page    ^U UnCut Text   ^T To Spell
```

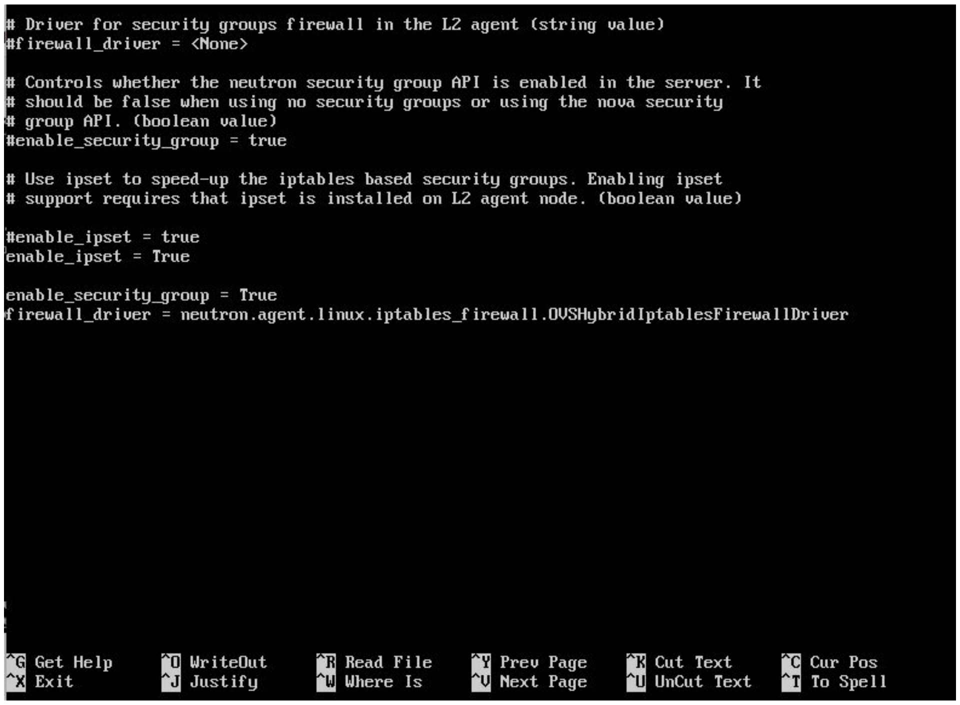
Figure 35.

For network configuration, the vxlan tunnel should not be enabled until an instance is brought up. This will be further talked about in the following section. Therefore, ml2 configuration should be like the following.



```
  GNU nano 2.2.6            File: /etc/neutron/plugins/ml2/ml2_conf.ini


[ml2_type_vxlan]

#
# From neutron.ml2
#

# Comma-separated list of <vni_min>:<vni_max> tuples enumerating ranges of
# VXLAN VNI IDs that are available for tenant network allocation (list value)

#vni_ranges =
vni_ranges = 1:1000

# Multicast group for VXLAN. When configured, will enable sending all broadcast
# traffic to this multicast group. When left unconfigured, will disable
# multicast VXLAN mode. (string value)
#vxlan_group = <None>


[securitygroup]

#
# From neutron.ml2
#

# Driver for security groups firewall in the L2 agent (string value)
#firewall_driver = <None>

# Controls whether the neutron security group API is enabled in the server. It
# should be false when using no security groups or using the nova security
# group API. (boolean value)
#enable_security_group = true

^G Get Help     ^O WriteOut     ^R Read File    ^Y Prev Page    ^K Cut Text     ^C Cur Pos
^X Exit         ^J Justify      ^W Where Is     ^V Next Page    ^U UnCut Text   ^T To Spell
```

Figure 36.

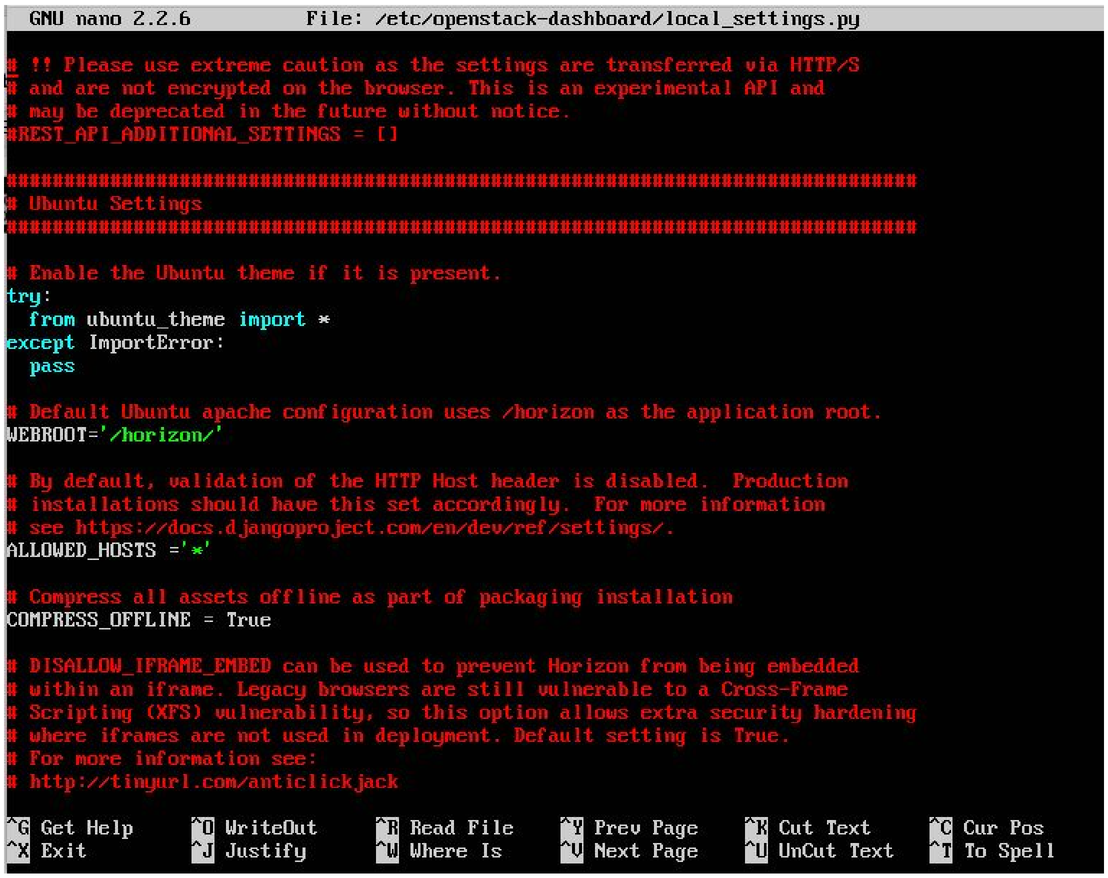


```
# Driver for security groups firewall in the L2 agent (string value)
#firewall_driver = <None>

# Controls whether the neutron security group API is enabled in the server. It
# should be false when using no security groups or using the nova security
# group API. (boolean value)
#enable_security_group = true

# Use ipset to speed-up the iptables based security groups. Enabling ipset
# support requires that ipset is installed on L2 agent node. (boolean value)

#enable_ipset = true
enable_ipset = True

enable_security_group = True
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver










^G Get Help     ^O WriteOut     ^R Read File    ^Y Prev Page    ^K Cut Text     ^C Cur Pos
^X Exit         ^J Justify      ^W Where Is     ^V Next Page    ^U UnCut Text   ^T To Spell
```

Figure 37.

For dashboard service, python code needs to be edited in the following file.

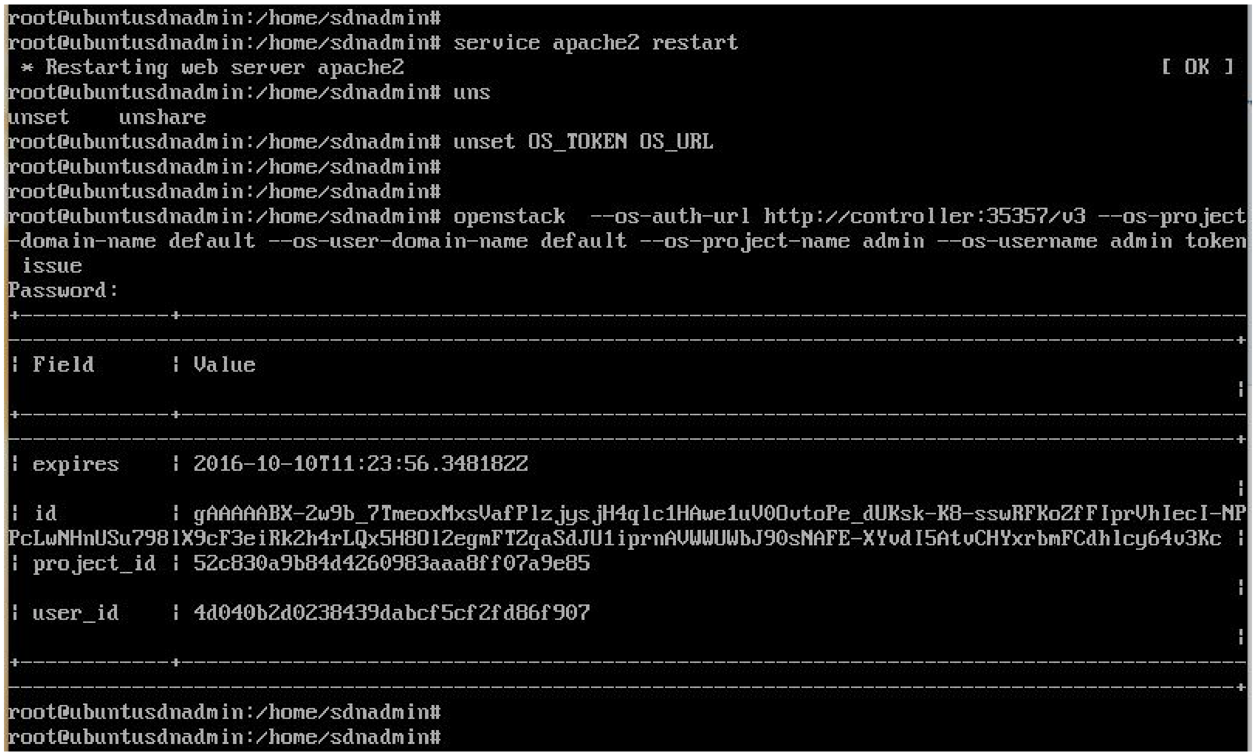
When the features above are configured, the controller is proved to be successfully configured. The following screenshots verify that the services are successfully Configured.



```
  GNU nano 2.2.6          File: /etc/openstack-dashboard/local_settings.py

# !! Please use extreme caution as the settings are transferred via HTTP/S
# and are not encrypted on the browser. This is an experimental API and
# may be deprecated in the future without notice.
#REST_API_ADDITIONAL_SETTINGS = []

###############################################################################
# Ubuntu Settings
###############################################################################

# Enable the Ubuntu theme if it is present.
try:
  from ubuntu_theme import *
except ImportError:
  pass

# Default Ubuntu apache configuration uses /horizon as the application root.
WEBROOT='/horizon/'

# By default, validation of the HTTP Host header is disabled.  Production
# installations should have this set accordingly.  For more information
# see https://docs.djangoproject.com/en/dev/ref/settings/.
ALLOWED_HOSTS ='*'

# Compress all assets offline as part of packaging installation
COMPRESS_OFFLINE = True

# DISALLOW_IFRAME_EMBED can be used to prevent Horizon from being embedded
# within an iframe. Legacy browsers are still vulnerable to a Cross-Frame
# Scripting (XFS) vulnerability, so this option allows extra security hardening
# where iframes are not used in deployment. Default setting is True.
# For more information see:
# http://tinyurl.com/anticlickjack

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Figure 38.

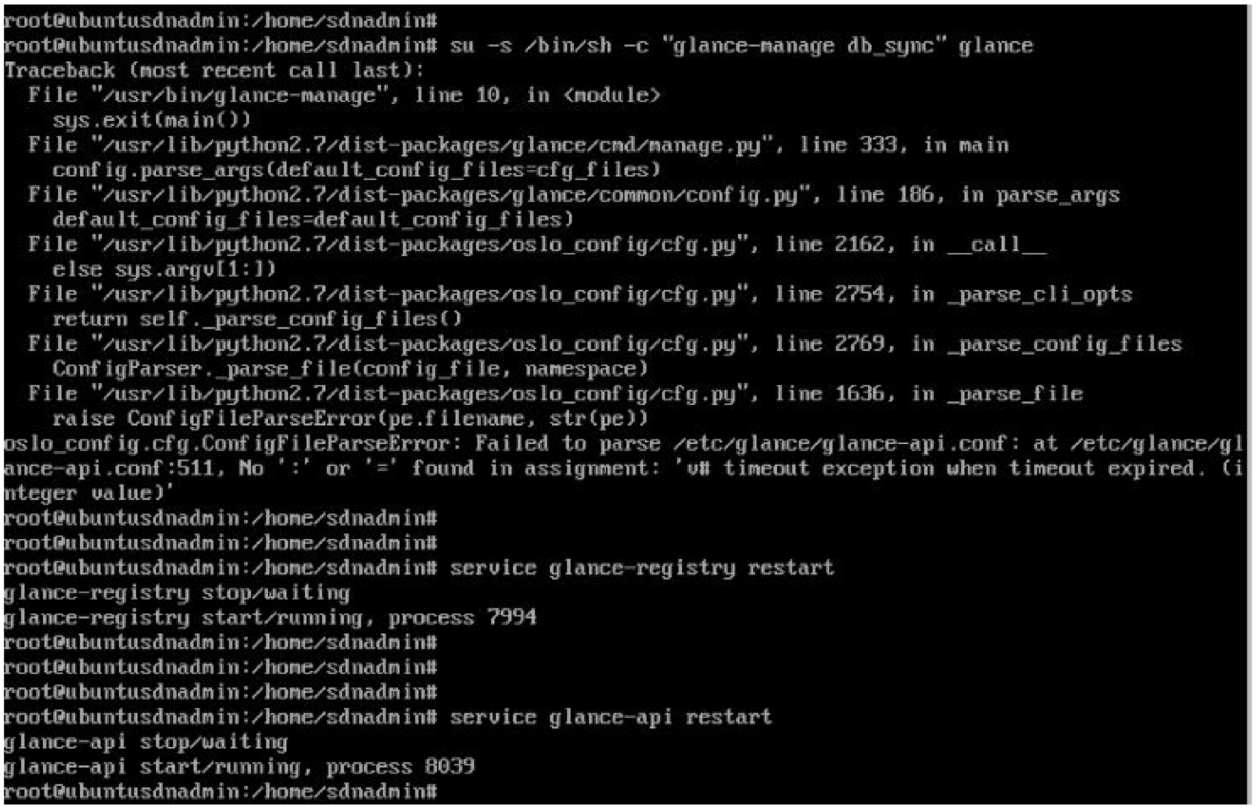
Verification of identity service:



```
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin# service apache2 restart
 * Restarting web server apache2                                                        [ OK ]
root@ubuntusdnadmin:/home/sdnadmin# uns
unset    unshare
root@ubuntusdnadmin:/home/sdnadmin# unset OS_TOKEN OS_URL
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin# openstack  --os-auth-url http://controller:35357/v3 --os-project
-domain-name default --os-user-domain-name default --os-project-name admin --os-username admin token
 issue
Password:
+------------+-------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------+
| Field      | Value
                                                                                                      |
+------------+-------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------+
| expires    | 2016-10-10T11:23:56.348182Z
                                                                                                      |
| id         | gAAAAABX-2w9b_7TmeoxMxsVafPlzjysjH4qlc1HAwe1uV0OvtoPe_dUKsk-K8-sswRFKoZfFIprVhlecI-NP
PcLwNHnUSu798lX9cF3eiRk2h4rLQx5H8Ol2egmFTZqaSdJU1iprnAVWWUWbJ90sNAFE-XYvdI5AtvCHYxrbmFCdhlcy64v3Kc |
| project_id | 52c830a9b84d4260983aaa8ff07a9e85
                                                                                                      |
| user_id    | 4d040b2d0238439dabcf5cf2fd86f907
                                                                                                      |
+------------+-------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------+
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin#
```

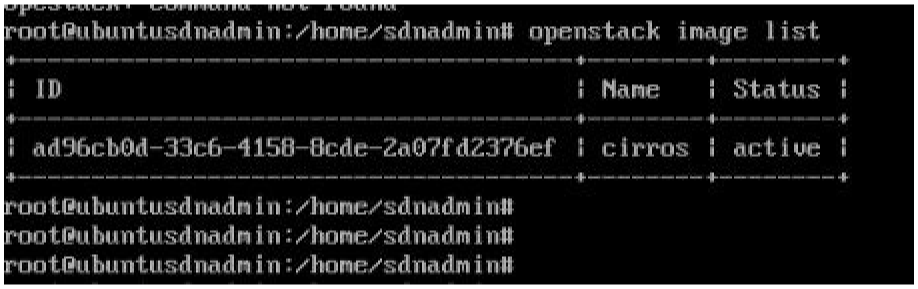
Figure 39.

Verification of image service:



```
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin# su -s /bin/sh -c "glance-manage db_sync" glance
Traceback (most recent call last):
  File "/usr/bin/glance-manage", line 10, in <module>
    sys.exit(main())
  File "/usr/lib/python2.7/dist-packages/glance/cmd/manage.py", line 333, in main
    config.parse_args(default_config_files=cfg_files)
  File "/usr/lib/python2.7/dist-packages/glance/common/config.py", line 186, in parse_args
    default_config_files=default_config_files)
  File "/usr/lib/python2.7/dist-packages/oslo_config/cfg.py", line 2162, in __call__
    else sys.argv[1:])
  File "/usr/lib/python2.7/dist-packages/oslo_config/cfg.py", line 2754, in _parse_cli_opts
    return self._parse_config_files()
  File "/usr/lib/python2.7/dist-packages/oslo_config/cfg.py", line 2769, in _parse_config_files
    ConfigParser._parse_file(config_file, namespace)
  File "/usr/lib/python2.7/dist-packages/oslo_config/cfg.py", line 1636, in _parse_file
    raise ConfigFileParseError(pe.filename, str(pe))
oslo_config.cfg.ConfigFileParseError: Failed to parse /etc/glance/glance-api.conf: at /etc/glance/gl
ance-api.conf:511, No ':' or '=' found in assignment: 'v# timeout exception when timeout expired. (i
nteger value)'
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin# service glance-registry restart
glance-registry stop/waiting
glance-registry start/running, process 7994
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin# service glance-api restart
glance-api stop/waiting
glance-api start/running, process 8039
root@ubuntusdnadmin:/home/sdnadmin#
```

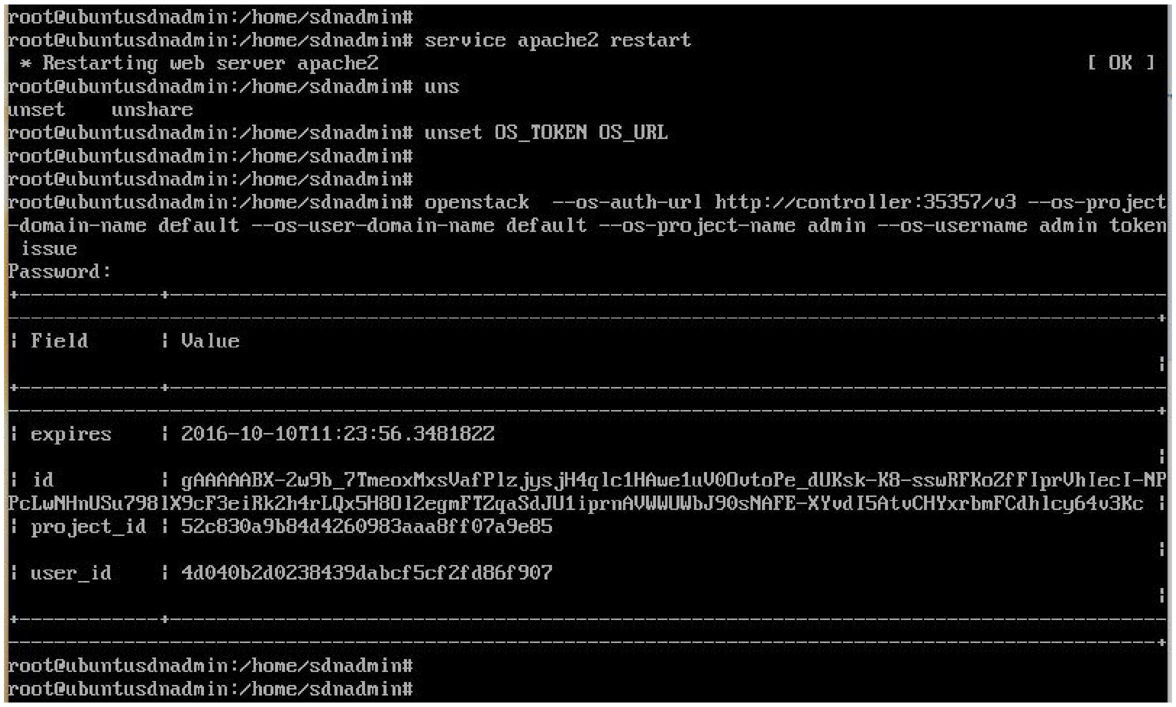
Figure 40.



```
openstack: command not found
root@ubuntusdnadmin:/home/sdnadmin# openstack image list
+--------------------------------------+--------+--------+
| ID                                   | Name   | Status |
+--------------------------------------+--------+--------+
| ad96cb0d-33c6-4158-8cde-2a07fd2376ef | cirros | active |
+--------------------------------------+--------+--------+
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin#
```

Figure 41.

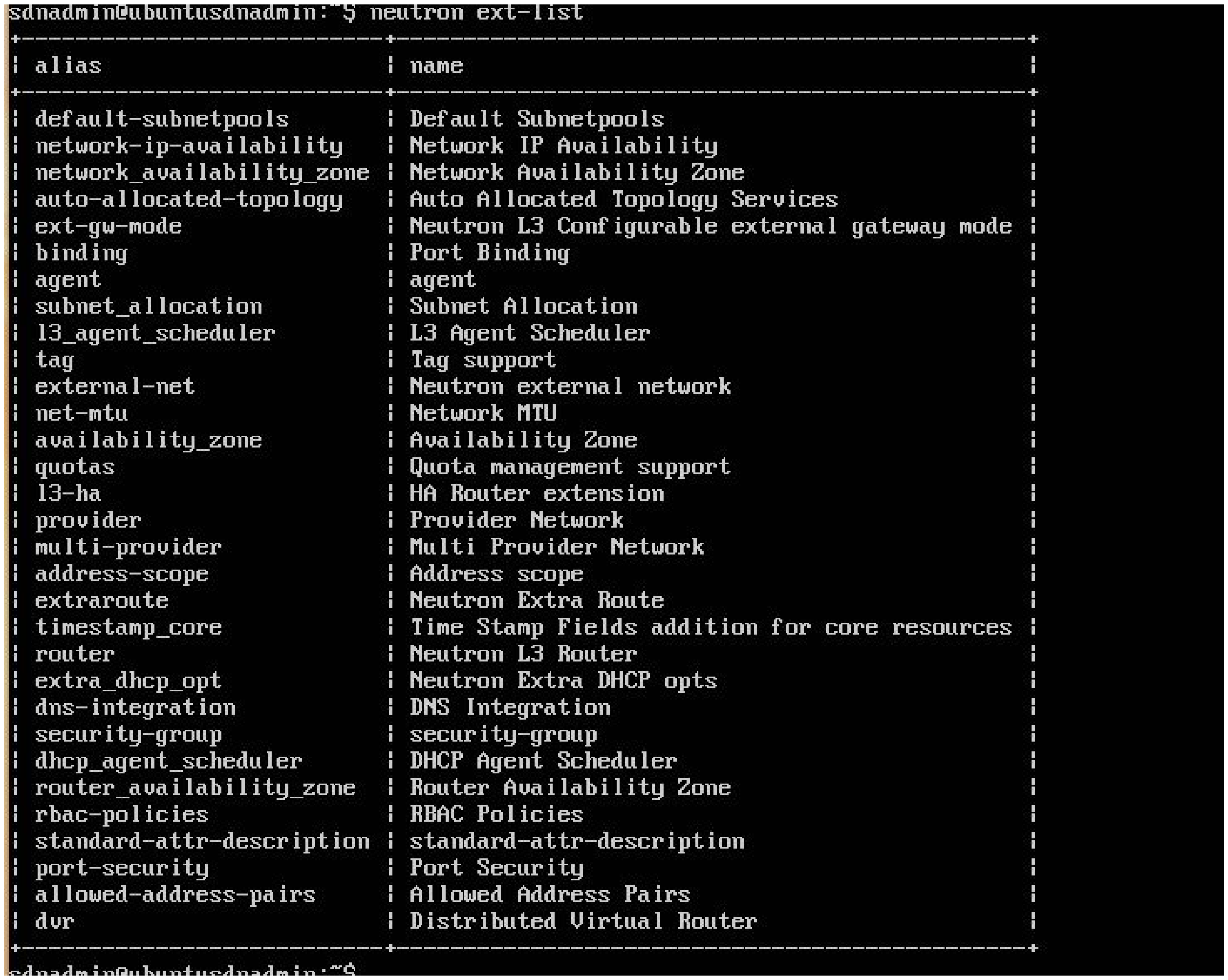
Verification of horizon service:



```
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin# service apache2 restart
 * Restarting web server apache2                                                         [ OK ]
root@ubuntusdnadmin:/home/sdnadmin# uns
unset    unshare
root@ubuntusdnadmin:/home/sdnadmin# unset OS_TOKEN OS_URL
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin# openstack  --os-auth-url http://controller:35357/v3 --os-project
-domain-name default --os-user-domain-name default --os-project-name admin --os-username admin token
 issue
Password:
+------------+------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------+
| Field      | Value
                                                                                                  |
+------------+------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------+
| expires    | 2016-10-10T11:23:56.348182Z
                                                                                                  |
| id         | gAAAAABX-2w9b_7TmeoxMxsVafPlzjysjH4qlc1HAwe1uV0OvtoPe_dUKsk-K8-sswRFKoZfFIprVhlecI-NP
PcLwNHnUSu798lX9cF3eiRk2h4rLQx5H8Ol2egmFTZqaSdJU1iprnAVWWUWbJ90sNAFE-XYvdI5AtvCHYxrbmFCdhlcy64v3Kc |
| project_id | 52c830a9b84d4260983aaa8ff07a9e85
                                                                                                  |
| user_id    | 4d040b2d0238439dabcf5cf2fd86f907
                                                                                                  |
+------------+------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------+
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin#
```

Figure 42.

Verification of network service:



```
sdnadmin@ubuntusdnadmin:~$ neutron ext-list
+----------------------------+-----------------------------------------------+
| alias                      | name                                          |
+----------------------------+-----------------------------------------------+
| default-subnetpools        | Default Subnetpools                           |
| network-ip-availability    | Network IP Availability                       |
| network_availability_zone  | Network Availability Zone                     |
| auto-allocated-topology    | Auto Allocated Topology Services              |
| ext-gw-mode                | Neutron L3 Configurable external gateway mode |
| binding                    | Port Binding                                  |
| agent                      | agent                                         |
| subnet_allocation          | Subnet Allocation                             |
| l3_agent_scheduler         | L3 Agent Scheduler                            |
| tag                        | Tag support                                   |
| external-net               | Neutron external network                      |
| net-mtu                    | Network MTU                                   |
| availability_zone          | Availability Zone                             |
| quotas                     | Quota management support                      |
| l3-ha                      | HA Router extension                           |
| provider                   | Provider Network                              |
| multi-provider             | Multi Provider Network                        |
| address-scope              | Address scope                                 |
| extraroute                 | Neutron Extra Route                           |
| timestamp_core             | Time Stamp Fields addition for core resources |
| router                     | Neutron L3 Router                             |
| extra_dhcp_opt             | Neutron Extra DHCP opts                       |
| dns-integration            | DNS Integration                               |
| security-group             | security-group                                |
| dhcp_agent_scheduler       | DHCP Agent Scheduler                          |
| router_availability_zone   | Router Availability Zone                      |
| rbac-policies              | RBAC Policies                                 |
| standard-attr-description  | standard-attr-description                     |
| port-security              | Port Security                                 |
| allowed-address-pairs      | Allowed Address Pairs                         |
| dvr                        | Distributed Virtual Router                    |
+----------------------------+-----------------------------------------------+
sdnadmin@ubuntusdnadmin:~$
```

Figure 43.

Other services can be verified by successful instances launch later.

- Compute nodes

For compute nodes, there are services needing to be configured cooperating with the controller node. There are also services needing to be configured independently to fulfill the compute service.

The following flowchart shows what configured on compute node.

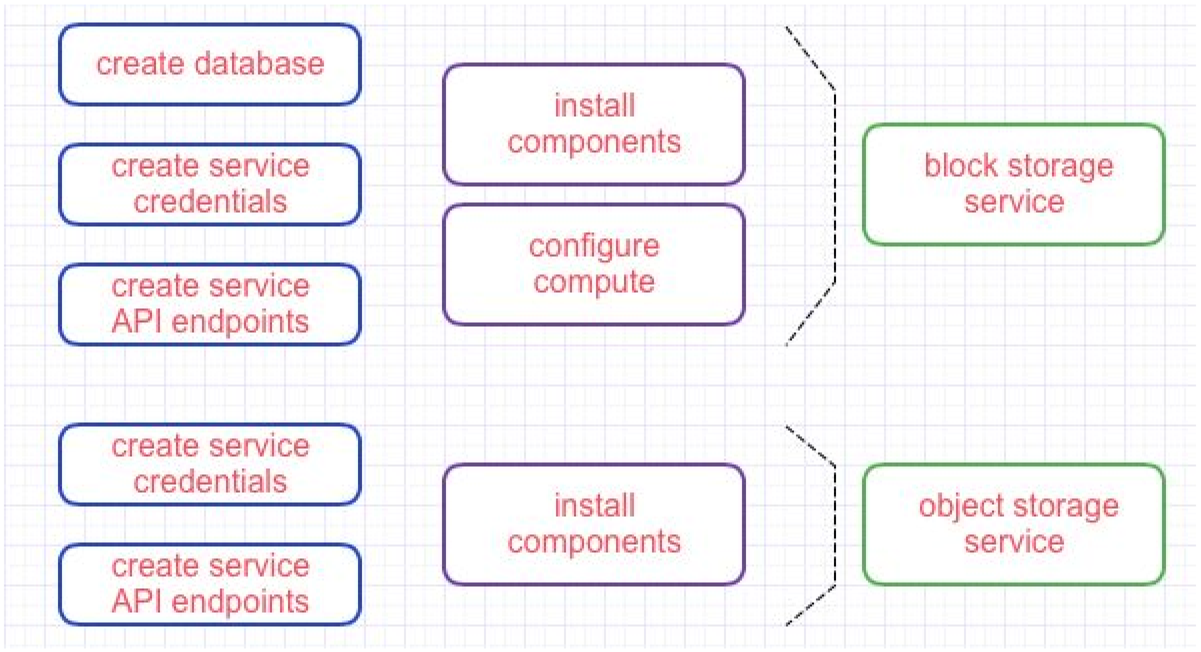
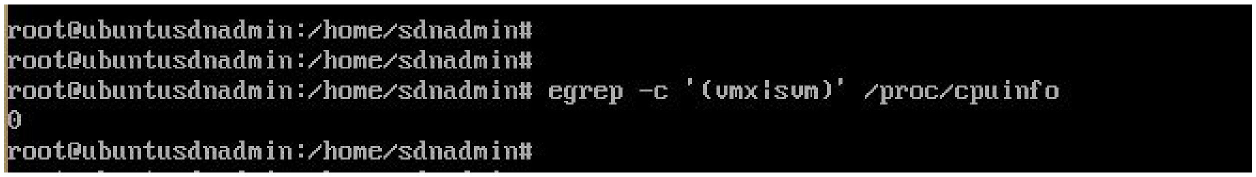


Figure 44.

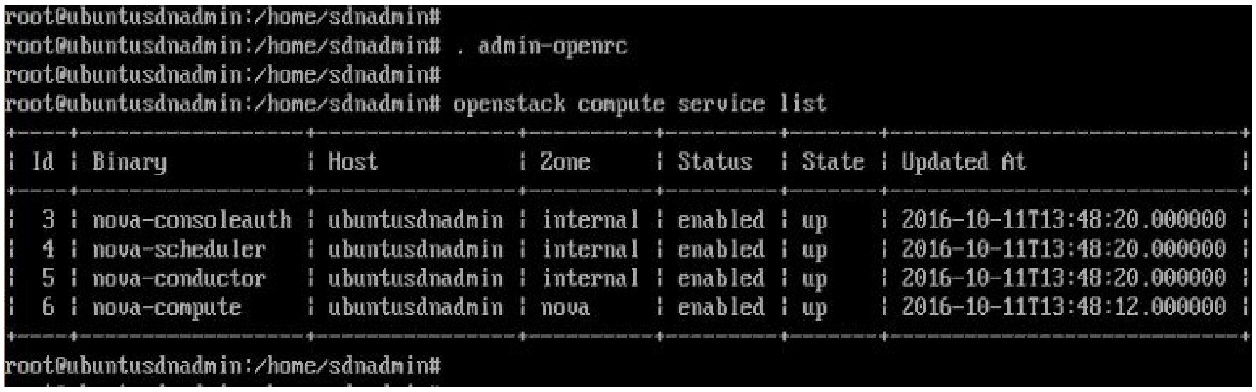
When the features above are configured, the compute node is proved to be successfully configured. The following screenshots verify that the services are successfully configured.

Verification of compute node services:



```
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin# egrep -c '(vmx|svm)' /proc/cpuinfo
0
root@ubuntusdnadmin:/home/sdnadmin#
```

Figure 45.



```
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin# . admin-openrc
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin# openstack compute service list
+----+------------------+----------------+----------+---------+-------+----------------------------+
| Id | Binary           | Host           | Zone     | Status  | State | Updated At                 |
+----+------------------+----------------+----------+---------+-------+----------------------------+
|  3 | nova-consoleauth | ubuntusdnadmin | internal | enabled | up    | 2016-10-11T13:48:20.000000 |
|  4 | nova-scheduler   | ubuntusdnadmin | internal | enabled | up    | 2016-10-11T13:48:20.000000 |
|  5 | nova-conductor   | ubuntusdnadmin | internal | enabled | up    | 2016-10-11T13:48:20.000000 |
|  6 | nova-compute     | ubuntusdnadmin | nova     | enabled | up    | 2016-10-11T13:48:12.000000 |
+----+------------------+----------------+----------+---------+-------+----------------------------+
root@ubuntusdnadmin:/home/sdnadmin#
```

Figure 46.

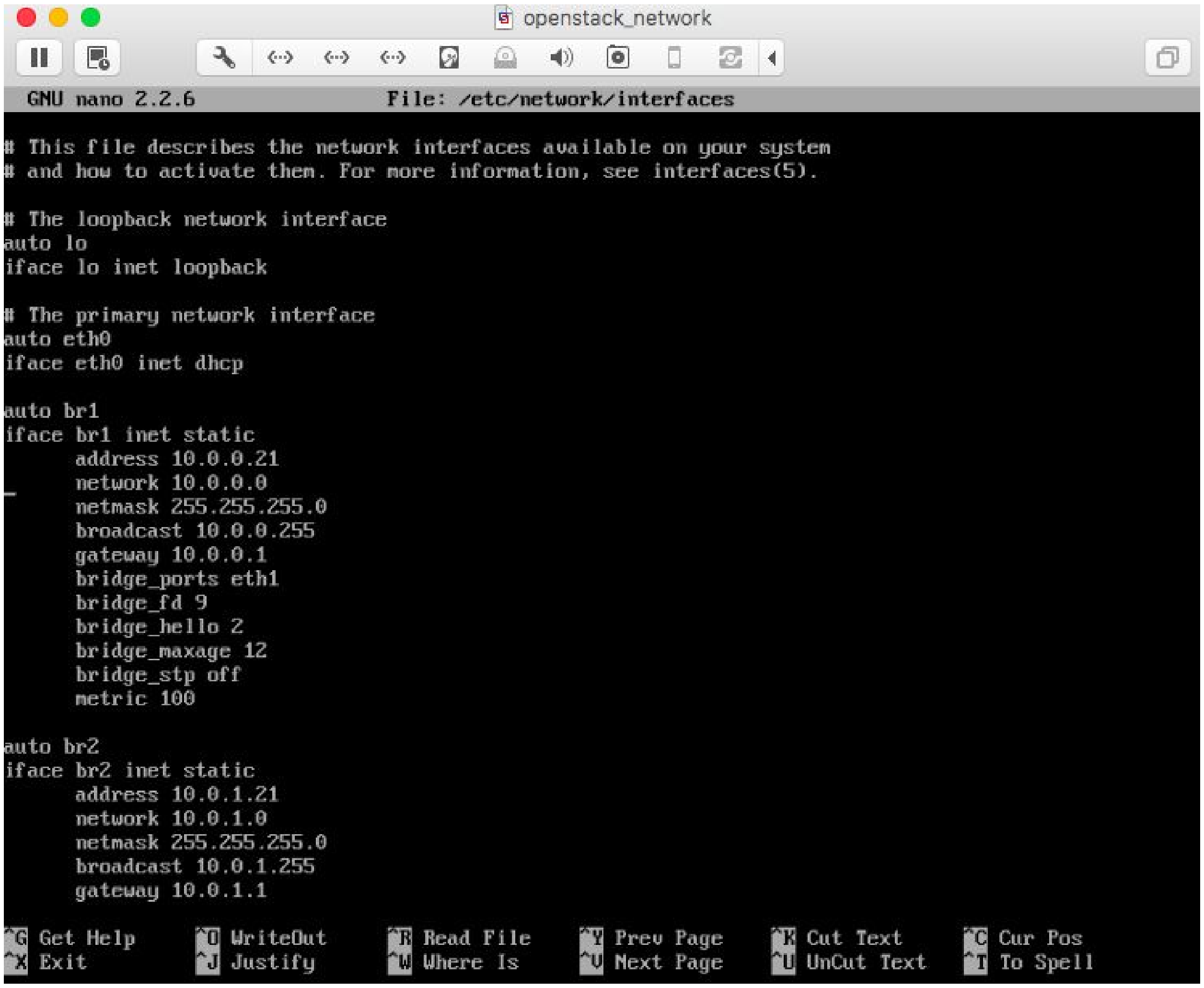
- Network node

In this part, we need to be clear about networks in OpenStack first.

A standard OpenStack Networking setup has up to four distinct physical data center networks:

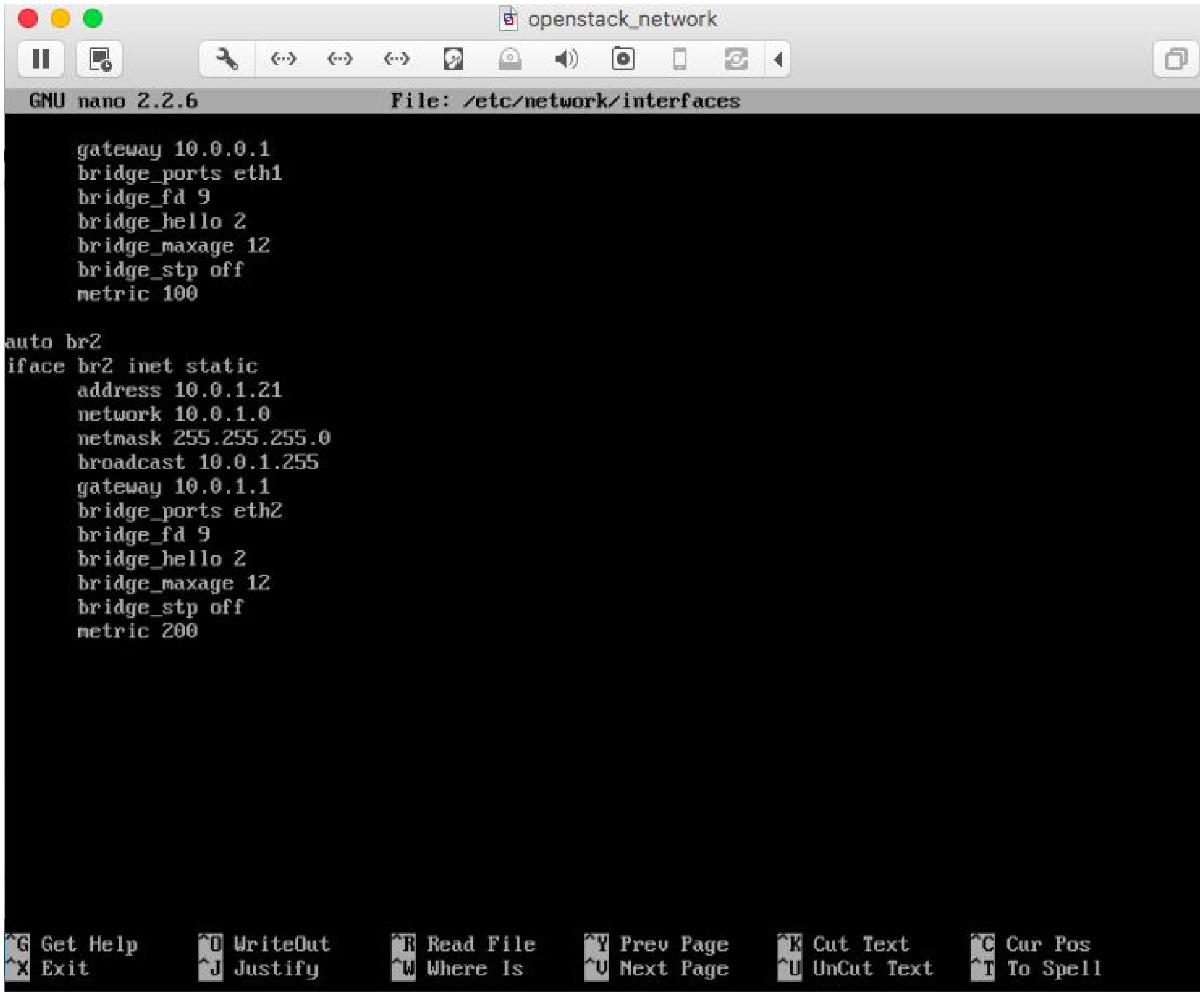
Management network
Used for internal communication between OpenStack Components. The IP addresses on this network should be reachable only within the data center and is considered the Management Security Domain.
Guest network
Used for VM data communication within the cloud deployment. The IP addressing requirements of this network depend on the OpenStack Networking plug-in in use and the network configuration choices of the virtual networks made by the tenant. This network is considered the Guest Security Domain.
External network
Used to provide VMs with Internet access in some deployment scenarios. The IP addresses on this network should be reachable by anyone on the Internet. This network is considered to be in the Public Security Domain.
API network
Exposes all OpenStack APIs, including the OpenStack Networking API, to tenants. The IP addresses on this network should be reachable by anyone on the Internet. This may be the same network as the external network, as it is possible to create a subnet for the external network that uses IP allocation ranges to use only less than the full range of IP addresses in an IP block. This network is considered the Public Security Domain.

Before going any further, we firstly need to configure the networks for nodes connection. This needs to be done after adding network adapters. The network configurations need to follow what's required for linux operating systems. As the following screenshots show, the etc/network/interfaces file for each nodes needs to be configured as the way it shows.



```
  GNU nano 2.2.6              File: /etc/network/interfaces

# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet dhcp

auto br1
iface br1 inet static
      address 10.0.0.21
      network 10.0.0.0
      netmask 255.255.255.0
      broadcast 10.0.0.255
      gateway 10.0.0.1
      bridge_ports eth1
      bridge_fd 9
      bridge_hello 2
      bridge_maxage 12
      bridge_stp off
      metric 100

auto br2
iface br2 inet static
      address 10.0.1.21
      network 10.0.1.0
      netmask 255.255.255.0
      broadcast 10.0.1.255
      gateway 10.0.1.1

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Figure 47.



```
GNU nano 2.2.6                File: /etc/network/interfaces

      gateway 10.0.0.1
      bridge_ports eth1
      bridge_fd 9
      bridge_hello 2
      bridge_maxage 12
      bridge_stp off
      metric 100

auto br2
iface br2 inet static
      address 10.0.1.21
      network 10.0.1.0
      netmask 255.255.255.0
      broadcast 10.0.1.255
      gateway 10.0.1.1
      bridge_ports eth2
      bridge_fd 9
      bridge_hello 2
      bridge_maxage 12
      bridge_stp off
      metric 200

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Figure 48.

For OpenStack, in neutron configuration, I choose Vxlan for tunnel OvSwitch as back-end with Ml2 plug-in. So what is a back-end and plug-in? In OpenStack, plugin configuration is like “core-plugin = ml2”, which is managed by “service-plugin = router” with the agent configuration: “[agent] Tunnel_types = vxvlan”. As for drivers for it, the configurations are like,

“type_drivers = flat,vlan,gre,vxlan
Mechanism_drivers = openvswitch
Firewall_driver = neutron,agent…..OVSHybrid…
Interface_driver = neutron…..OVSInterfaceDriver”

I’ll explain the reason why I choose these two.

For Vxlan, firstly, we need to figure out what’s the difference within flat, local, vlan, gre and vxlan. A local network is a network that can only be realized on a single host. This is only used in proof-of-concept or development environments, because just about any other OpenStack environment will have multiple compute hosts and/or a separate network host. A flat network is a network that does not provide any segmentation options. A traditional L2 ethernet network is a "flat" network. Any servers attached to this network are able to see the same broadcast traffic and can contact each other without requiring a router. flat networks are often used to attach Nova servers to an existing L2 network (this is called a "provider network"). A vlan network is one that uses VLANs for segmentation. When you create a new network in Neutron, it will be assigned a VLAN ID from the range you have configured in your Neutron configuration. Using vlan networks requires that any switches in your environment are configured to trunk the corresponding VLANs. gre and vxlan networks are very similar. They are both "overlay" networks that work by encapsulating network traffic. Like vlan networks, each network you create receives a unique tunnel id. Unlike vlan networks, an overlay network does not require that you synchronize your OpenStack configuration with your L2 switch configuration. Secondly, VXLAN is my a preferred solution, because it provides more entropie on the receiving NIC, which results in a higher performance, because multiple CPU cores are used to process ingress packets. (quoted from (http://www.opencloudblog.com/?p=300))

For Open vSwitch, why would I use Open vSwitch instead of the Linux bridge? Open vSwitch is specially designed to make it easier to manage VM network configuration and monitor state spread across many physical hosts in dynamic virtualized environments. Open vSwitch is targeted at large multi-server virtualization environments, so it is focused on logical abstraction and management; the Linux bridge is fast and reliable, but lacks all fancy control features. Besides, Hypervisors need the ability to bridge traffic between VMs and with the outside world. On Linux-based hypervisors, this used to mean using the built-in L2 switch (the Linux bridge), which is fast and reliable. So, it is reasonable to ask why Open vSwitch is used. Open vSwitch is targeted at multi-server virtualization deployments, a landscape for which the previous stack is not well suited. These environments are often characterized by highly dynamic end-points, the maintenance of logical abstractions, and (sometimes) integration with or offloading to special purpose switching hardware. The following characteristics and design considerations help OpenvSwitch cope with the requirements above. The mobility of state: All network state associated with a network entity (say a virtual machine) should be easily identifiable and migratable between different hosts. This may include traditional "soft state" (such as an entry in an L2 learning table), L3 forwarding state, policy routing state, ACLs, QoS policy, monitoring configuration. Further, Open vSwitch state is typed and backed by a real data-model allowing for the development of structured automation systems. Open vSwitch is also responding to network dynamics: Virtual environments are often characterized by high-rates of change. VMs coming and going, VMs moving backwards and forwards in time, changes to the logical network environments, and so forth.

Open vSwitch supports a number of features that allow a network control system to respond and adapt as the environment changes. This includes simple accounting and visibility support such as NetFlow, IPFIX, and sFlow. But perhaps more useful, Open vSwitch supports a network state database (OVSDB) that supports remote triggers. Open vSwitch also supports OpenFlow as a method of exporting remote access to control traffic. There are a number of uses for this including global network discovery through inspection of discovery or link-state traffic (e.g. LLDP, CDP, OSPF, etc.). In advance, Open vSwitch includes multiple methods for specifying and maintaining tagging rules, all of which are accessible to a remote process for orchestration. In a similar vein, Open vSwitch supports a GRE implementation that can handle thousands of simultaneous GRE tunnels and supports remote configuration for tunnel creation, configuration, and tear-down. This, for example, can be used to connect private VM networks in different data Centers. In the aspect of hardware integration: Open vSwitch's forwarding path (the in-kernel datapath) is designed to be amenable to "offloading" packet processing to hardware chipsets, whether housed in a classic hardware switch chassis or in an end-host NIC. This allows for the Open vSwitch control path to be able to both control a pure software implementation or a hardware switch. In many ways, Open vSwitch targets a different point in the design space than previous hypervisor networking stacks, focusing on the need for automated and dynamic network control in large-scale Linux-based virtualization environments. The goal with Open vSwitch is to keep the in-kernel code as small as possible (as is necessary for performance) and to re-use existing subsystems when applicable (for example Open vSwitch uses the existing QoS stack). As of Linux 3.3, Open vSwitch is included as a part of the kernel and packaging for the userspace utilities are available on most popular distributions. (referred from (http://networkengineering.stackexchange.com/questions/28408/)

4.2 Integration of OpenDayLight and OpenStack

A use case is used to describe this section.

4.2.1 Network Topology Diagram

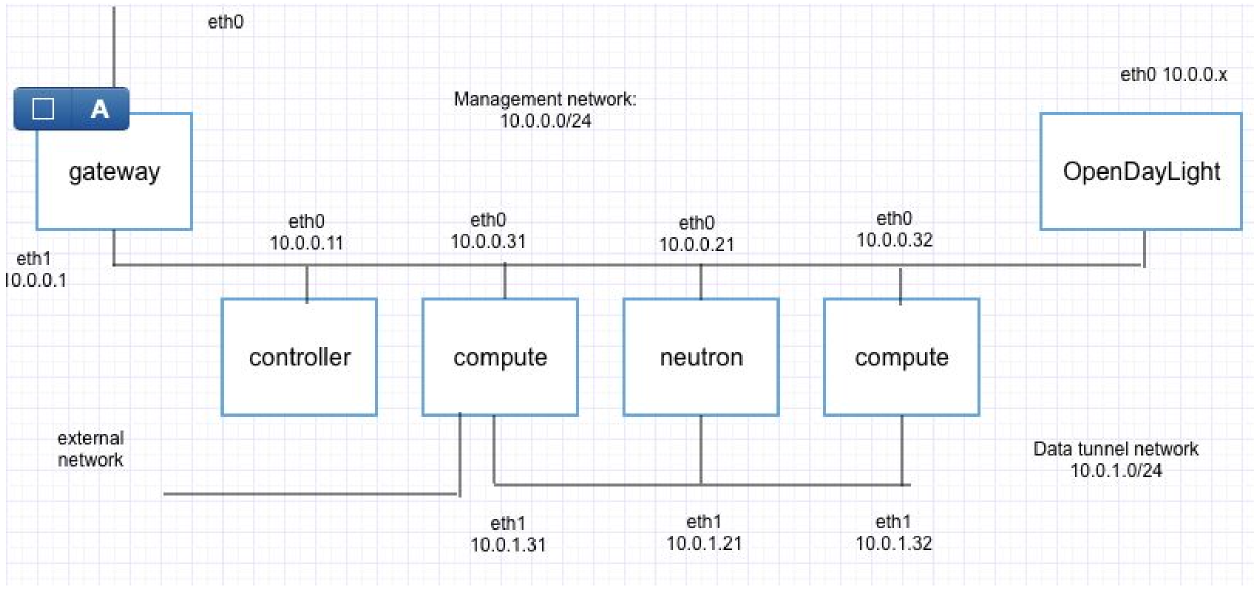


Figure 49.

The diagram is the same one from the last section.

4.2.2 Configuration Process

4.2.2.1 Hardware Requirement

- Gateway:
  Memory: 512 MB
  CPU: 1
  NIC:
  adapter 1: bridged adapter
  adapter 2: internal network (management)
- Controller:
  Memory: 3072 MB
  CPU: 2
  NIC:
  adapter 1: internal network (management)
- Network:
  Memory: 1024 MB
  CPU: 2
  NIC:
  adapter 1: internal network (management)
  adapter 2: internal network (tunnel)
  adapter 3: internal network (vlan)
  adapter 4: internal network (management)
- Compute:

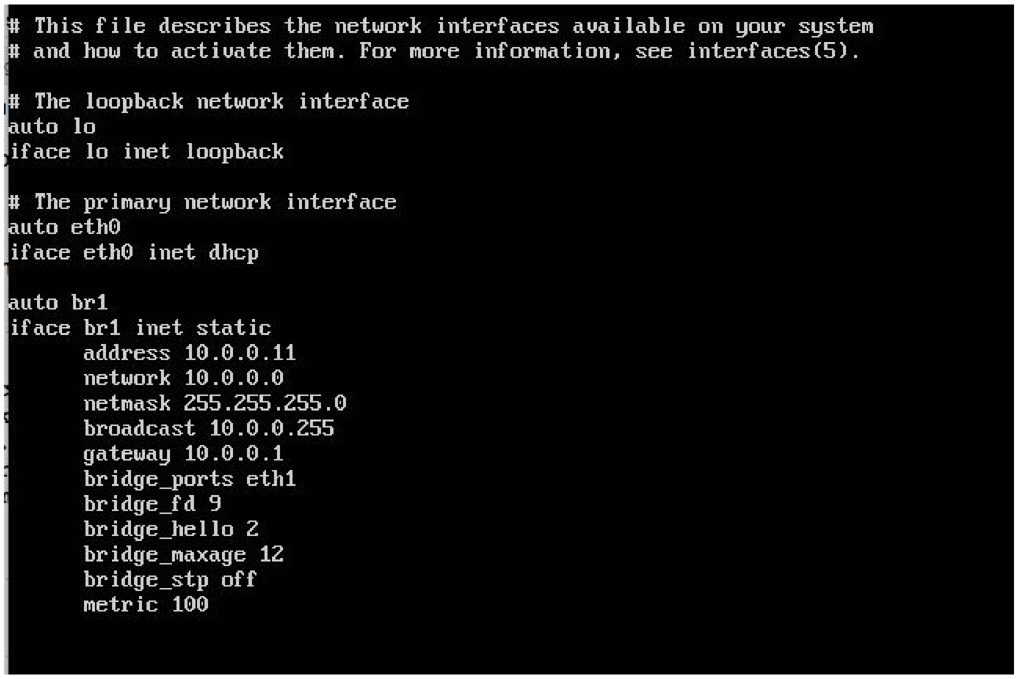
Memory: 2048MB
CPU: 1
NIC:
  adapter 1: internal network (management)
  adapter 2: internal network (tunnel)
  adapter 3: internal network (vlan)
  adapter 4: internal network (management)

- OpenDayLight Controller:
  Memory: 2048 MB
  CPU: 2
  NIC:
    adapter 1: internal network (m_management)

4.2.2.2 Network configuration

Based on the given network topology, we need to configure network adapters and IP addresses on each node.

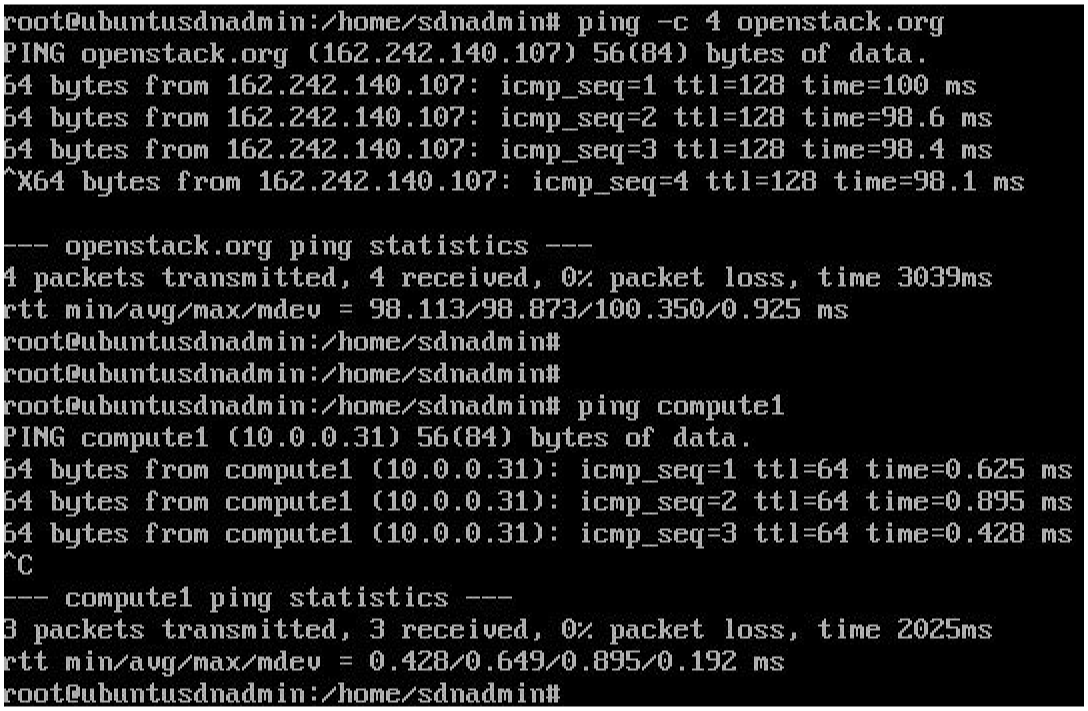
As for IP addresses and network type configuration, the following screenshot shows how they are configured for a single bridged network adapter.



```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet dhcp

auto br1
iface br1 inet static
      address 10.0.0.11
      network 10.0.0.0
      netmask 255.255.255.0
      broadcast 10.0.0.255
      gateway 10.0.0.1
      bridge_ports eth1
      bridge_fd 9
      bridge_hello 2
      bridge_maxage 12
      bridge_stp off
      metric 100
```

Figure 50.

When the IPs are all configured well on each node, we can test connectivity from each of the two nodes.



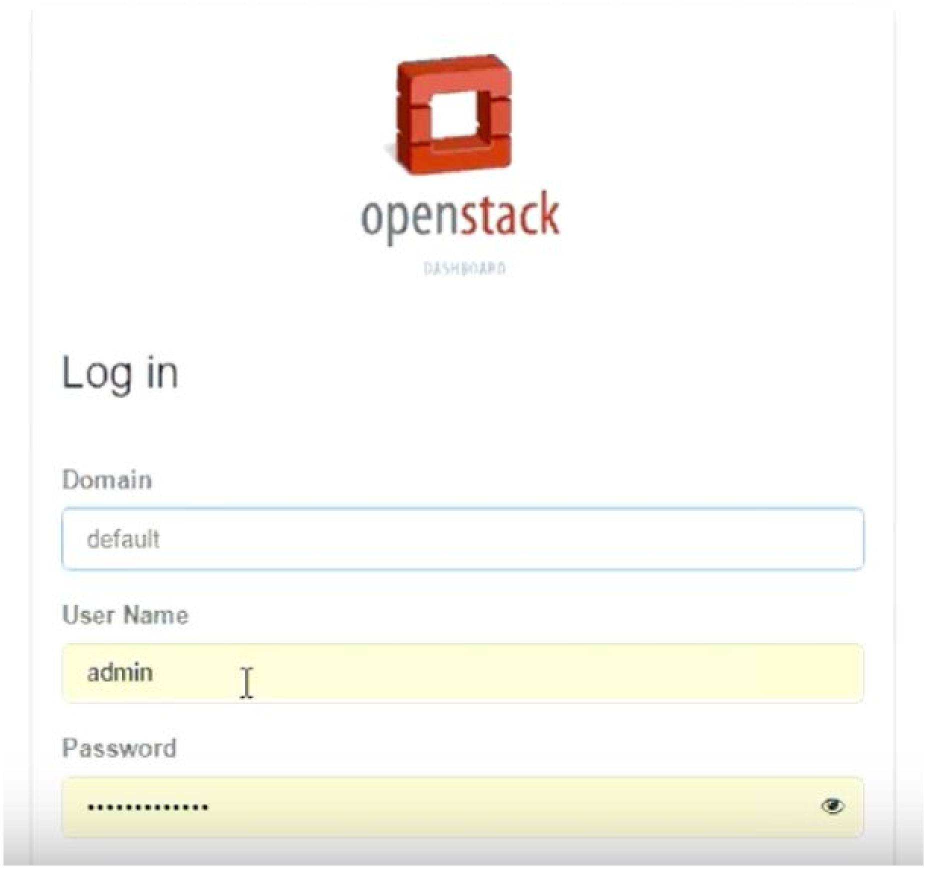
```
root@ubuntusdnadmin:/home/sdnadmin# ping -c 4 openstack.org
PING openstack.org (162.242.140.107) 56(84) bytes of data.
64 bytes from 162.242.140.107: icmp_seq=1 ttl=128 time=100 ms
64 bytes from 162.242.140.107: icmp_seq=2 ttl=128 time=98.6 ms
64 bytes from 162.242.140.107: icmp_seq=3 ttl=128 time=98.4 ms
^X64 bytes from 162.242.140.107: icmp_seq=4 ttl=128 time=98.1 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3039ms
rtt min/avg/max/mdev = 98.113/98.873/100.350/0.925 ms
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin#
root@ubuntusdnadmin:/home/sdnadmin# ping compute1
PING compute1 (10.0.0.31) 56(84) bytes of data.
64 bytes from compute1 (10.0.0.31): icmp_seq=1 ttl=64 time=0.625 ms
64 bytes from compute1 (10.0.0.31): icmp_seq=2 ttl=64 time=0.895 ms
64 bytes from compute1 (10.0.0.31): icmp_seq=3 ttl=64 time=0.428 ms
^C
--- compute1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2025ms
rtt min/avg/max/mdev = 0.428/0.649/0.895/0.192 ms
root@ubuntusdnadmin:/home/sdnadmin#
```

Figure 51.

In this case, in order to give access to the internet, we configure another network adapter as the shared network with the host (virtual machines and the host). The network virtual machines use is through NAT to communicate with internet. As for the bridged network, it appears as an additional computer on the same physical network connection as the host laptop. When configure the bridged network, an error called “ERROR RTNETLINK: File exists” occurs. This is because the interface ”br1” cannot be brought up after configuration the file </etc/network/interfaces>, and can be solved by giving a lower priority to the bridged network like adding the metrics. Besides, “bridge-utils” needs to be installed by “sudo apt install” first.

4.2.2.3 Configuration Process

Firstly, start the gateway node and open its console to configure ntp and check if it’s the most up-to-date version. Then start all nodes of openstack. Open web browser and connect to 10.0.0.11/horizon/ and log in to openstack (domain: default, user name: admin)



Log in

Domain

default

User Name

admin

Password

••••••••••••••

Figure 52.

In the OpenStack page, click project and select instances in the compute drop-down list. Then start the instances and click network and select network topology to see the diagram. click network and select networks to see if there are existing networks. Go back to the instances and check with the floating address assigned to the each node. Now open the terminal of the laptop and type the command : ssh cirros@10.0.0.X (IP of one of the instance's one network). Answer yes and type "sdnlabs#112", the password. Then ping 8.8.8.8 to test google Availability. Repeat this action until having tested all the existing networks. Then clear all the instances by deleting them all in OpenStack.

Click access&security in compute drop-down list and choose floating IP. Select all the IP addresses and click release floating IPs. Click network and choose routers in the drop-down list. Select the router in the page and click clear gateway. Select router in the page again and click delete routers. Click network in the network list and select all the networks in the page. Click delete networks (the ext-net may be not deleted and an error warning would be popped out). When all deleted, it shows as the following screenshot.

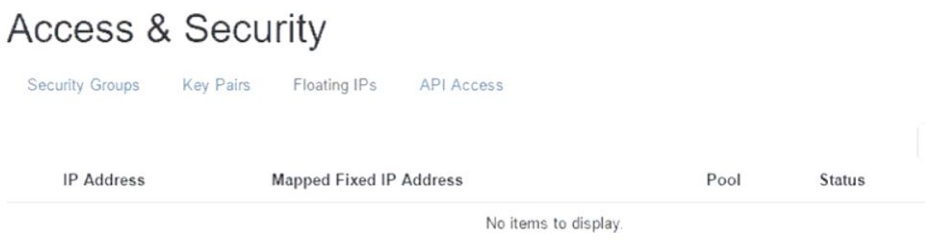


Figure 53.

Then click admin and choose routers in the drop-down list and select the router in the page and click delete routers. Choose networks in the same drop-down list. Select networks in the page and click delete networks. Now turn to network drop-down list and choose network topology to check there's no diagram.

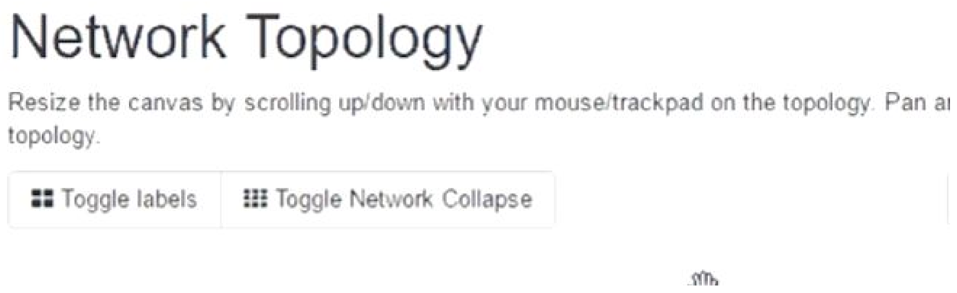

Network Topology
Resize the canvas by scrolling up/down with your mouse/trackpad on the topology. Pan ar
topology.
Toggle labels
Toggle Network Collapse

Figure 54.

Now open a new console of the openstack controller and connect to OpenDayLight by ssh. Start OpenDayLight and karaf. Apart from the features noted from the last section, also install the following features. odl-ovsdb-openstack, odl-dlux-core and Odl-dlux-all. Now we can go to the openstack controller node and check. Type the following commands: curl u admin:admin http://IP of OpenDayLight/. Because it's a server version with GUI, we need to go to the browser on the console. The commands are as follows,

```
Cd OPSInstaller
Cd installer
Cat OSODL-ovs-00-force-set-controller-time.sh
./OSODL-ovs-00-force-set-controller-time.sh
Cat OSODL-ovs-01-force-redo-set-openstack-node.sh
./OSODL-ovs-01-force-redo-set-openstack-node.sh
 Ls OSODL*
 Cat OSODL-ovs-02-stop-neutron.sh
 Ssh openstack@controller cat ./OPSInstaller/controller/exe-
            stage39-SUDO-odl-stop-neutron
Ls OSODL*
./OSODL-ovs-02-stop-neutron.sh
```

```
Ls OSODL*
Cat OSODL-ovs-03-purge-neutron-agent.sh
Ssh openstack@network cat ./OPSInstaller/network/exe-
      stage40-SUDO-odl-purge-neutron-ovs-plugin-network.sh
./OSODL-ovs-03-purge-neutron-agent.sh
Ls OSODL*
Cat OSODL-ovs-04-set-ovs-manager.sh
Ssh openstack@network cat ./OPSInstaller/network/exe-
      stage43-SUDO-odl-set-ovs-management-network.sh
./OSODL-ovs-04-set-ovs-manager.sh
```

Then go to the web browser and visit 10.0.0.X(IP of OpenDayLight):8181/index.html. Sign in to opendaylight (admin/admin)

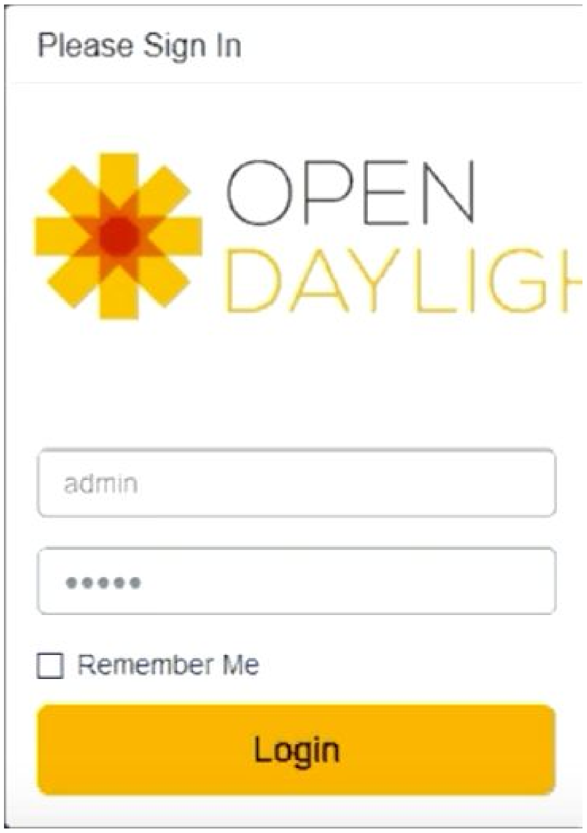


Figure 55.

In the opendaylight web page: click reload under the controls

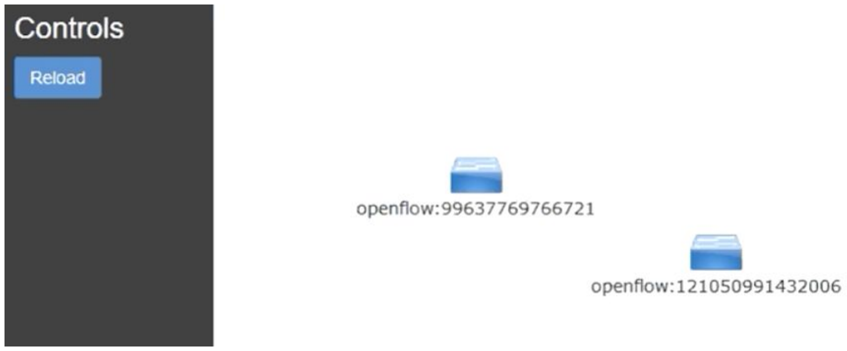


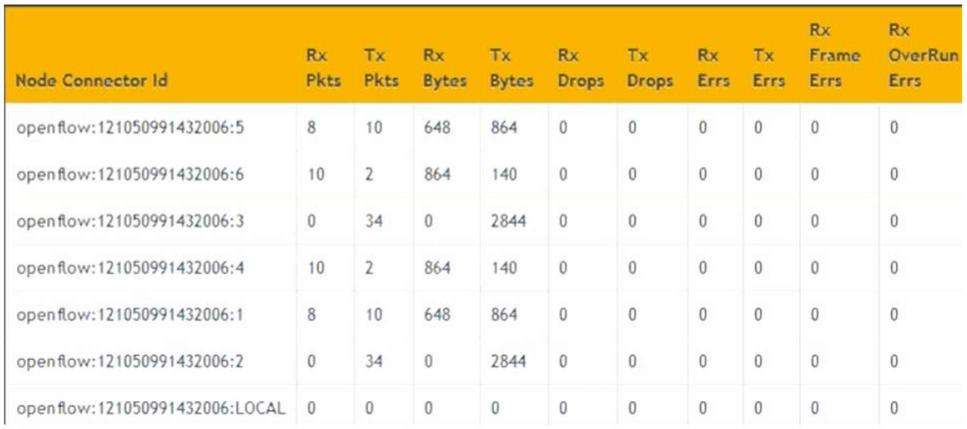
Figure 56.

Now go back to the openstack controller node and configure the integration part. Type the following commands,

```
cat OSODL-ovs-06-set-neutron.sh
ssh openstack@controller cat
    ./OPSInstaller/controller/exe-stage52-USER-odl-neutron-database.sh
ssh openstack@controller cat
    ./OPSInstaller/controller/exe-stage53-USER-SUDO-odl-neutron.sh
    ./OSODL-ovs-06-set-neutron.sh
cat OSODL-ovs-07-pip-init-neutron.sh
ssh openstack@controller cat
    ./OPSInstaller/controller/exe-stage54-USER-SUDO-odl-pip-install.sh
ssh openstack@network
    ./OPSInstaller/network/exe-stage54p2-SUDO-odl-restart-network.sh
ssh openstack@network cat
    ./OPSInstaller/network/exe-stage54p2-SUDO-odl-restart-network.sh
ssh openstack@controller cat
    ./OPSInstaller/controller/exe-stage55-USER-odl-initial-network.sh
    ./OSODL-ovs-07-pip-init-network.sh
```

After monitoring the flows, go to the opendaylight web page and click reload under the controls. Click nodes on the left side and check the nodes. Click node connection on the right side and check. Go to the 10.0.0.X (OpenStack controller)/horizon/auth/login. Log into the openstack (default&admin). Click project -- network -- network topology and check -- networks and check -- routers and check-- compute -- instances and check (shows items to display). Click “launch instance” at the up right corner. Click “details” and edit “instance name” and click “next” at the down left corner. Click “source” and “flavour” and edit, select the “+” button of “m1 tiny” and it gets disappeared. Click “network”, select the “+” button of “admin-net” and it gets disappeared. Click “Network ports” and “security groups” , select the “+” button of “default” of “security groups”and it gets disappeared. Click “keypair” and “configuration” and check in the “configuration” page and click “launch instance”. Go to the main page and click “instance”, we can see the instance is

spawning under “task”. Click “Access&Security” and select “floating IPs”. Click “Allocate IP” and confirm in the pop-up page, when done, click “Associate” under “actions” of the allocated ip address. In the “associate” pop-up page, choose the instance port in the Drop-down list. Click “instances” on the left side and show the page. Open a new console of openstack controller and type the following commands: ssh-keygen -R 10.0.0.102, ssh-cirros@10.0.0.102 to open the opendaylight main page and click topology, check how many nodes are in it
click nodes and click the number under node connection column where the number is consistent with the topology



Node Connector Id	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs
openflow:121050991432006:5	8	10	648	864	0	0	0	0	0	0
openflow:121050991432006:6	10	2	864	140	0	0	0	0	0	0
openflow:121050991432006:3	0	34	0	2844	0	0	0	0	0	0
openflow:121050991432006:4	10	2	864	140	0	0	0	0	0	0
openflow:121050991432006:1	8	10	648	864	0	0	0	0	0	0
openflow:121050991432006:2	0	34	0	2844	0	0	0	0	0	0
openflow:121050991432006:LOCAL	0	0	0	0	0	0	0	0	0	0

Figure 57.

Then go back to openstack main page and click Access&Security. Click “Security Groups” in the Access&Security page and click “manage rules” of the group in the page and click instances on the left.

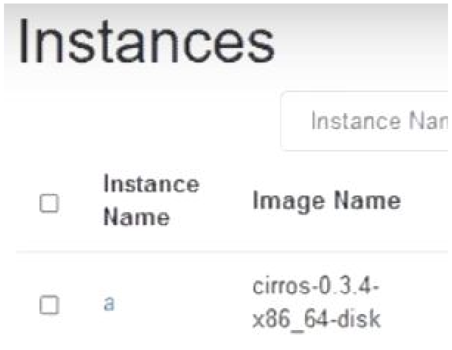


Figure 58.

During the integration, floating IPs are needed. Also, we need to set the router-delete namespace as true. Because the default configuration when installing from Ubuntu's packages was to not delete namespaces after their associated network or router was removed. If you didn't keep tabs on this, you'd soon end up with a lot of redundant namespaces on your network nodes. As a public cloud operator this is especially problematic when you've got public IPv4 address space to manage and you really don't want precious addresses being wasted on gateway interfaces for virtual routers that are no longer in use. (floating IP explanations are quoted from http://dischord.org/2016/01/05/cleaning-up-after-neutron/).

As far as the configuration concerned, the instance is ready to be configured. We can configure several various network requirements through OpenDayLight controller.

4.3. A VTN Design Example

4.3.1. Architecture

The following figure shows how it should be integrated in a cloud application with SDN.

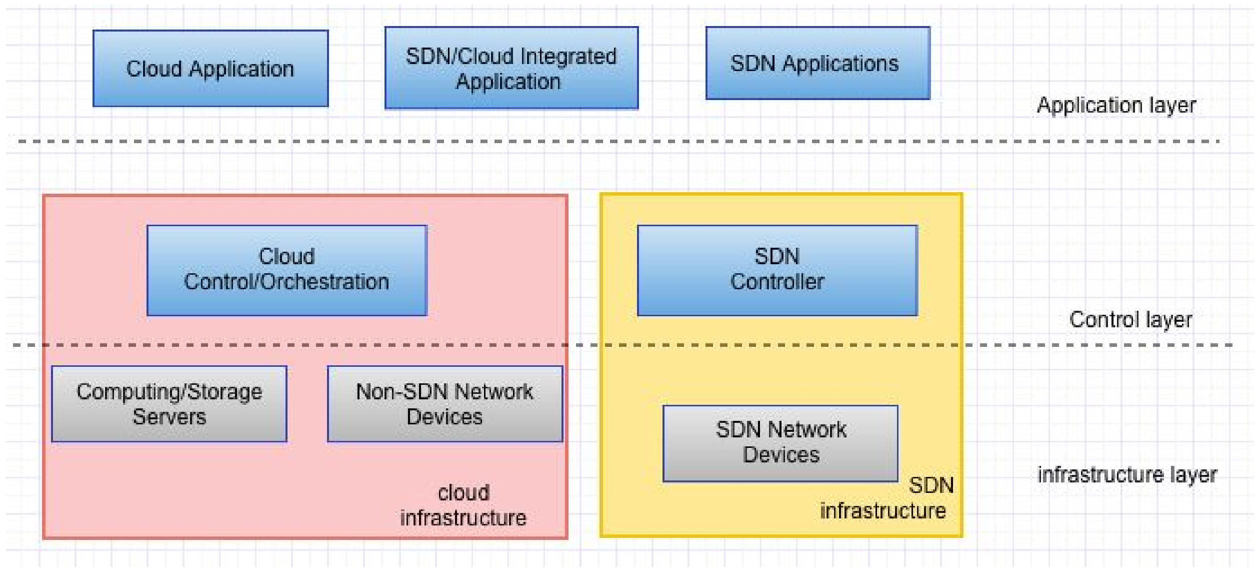


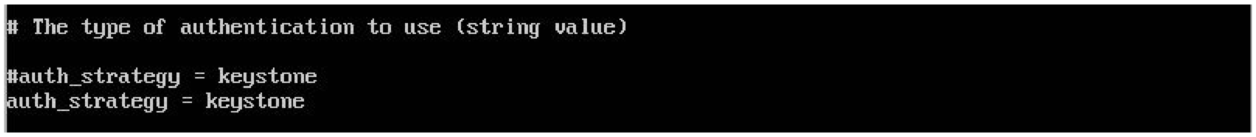
Figure 59.

In this figure, the cloud infrastructure is provided by OpenStack OS, the SDN controller we used in this case is the OpenDaylight controller. The Computing and Storage services are respectively nova and block in OpenStack. Different layers communicate with each other by APIs. For example, like what we mentioned in the former sections, controller communicate with upper layer APIs through northbound APIs like REST API or java, python APIs. The controller communicates with lower layer switched and network devices via southbound APIs, which is openflow in this case.

4.3.2. Design of VTN

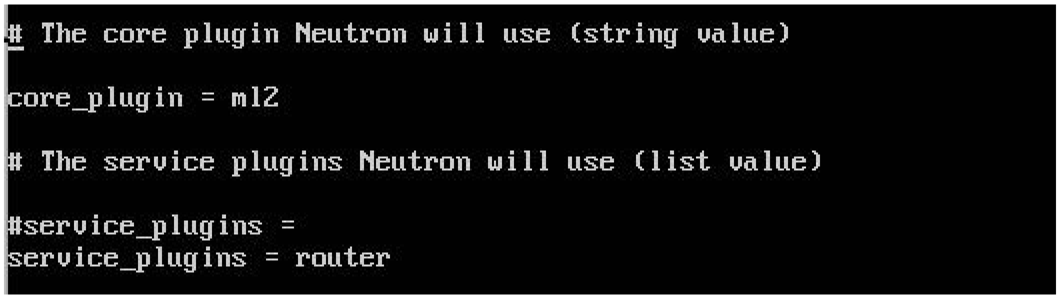
In this example, we use the OpenDaylight controller integrated with the OpenStack control node and configure two compute nodes as well as a network node. Among these nodes, network configuration is necessary. For the controller node, it requires one network interface: management. For the network node, it should include four network interfaces: management, project tunnel networks, VLAN project networks, and external (typically the Internet). The Open vSwitch bridge br-vlan must contain a port on the VLAN interface and Open vSwitch bridge br-ex must contain a port on the external interface. For the compute nodes, they each must have three network interfaces: management, project tunnel networks, and VLAN project networks. The Open vSwitch bridge br-vlan must contain a port on the VLAN Interface. The network and compute nodes should contain a separate network interface for VLAN project networks. VLAN project networks can use any Open vSwitch bridge with access to a network interface. The VLAN network does not require an IP address range because it only handles layer-2 connectivity.
For controller node, we need to configure SQL server, Identity service and message queue service with neutron database in the neutron.conf file. Also, OpenStack Compute controller/management service with appropriate configuration to use neutron in the nova.conf file. For network node, OpenStack Identity service, Open vSwitch service, Open vSwitch agent, L3 agent, DHCP agent, metadata agent, and any dependencies need to be configured in the neutron.conf file. For compute nodes, OpenStack Identity service, OpenStack Compute controller/management service and Open vSwitch service, Open vSwitch agent, and any dependencies need to be configured in the neutron.conf file and nova.conf file.

The following are part of the configuration examples.



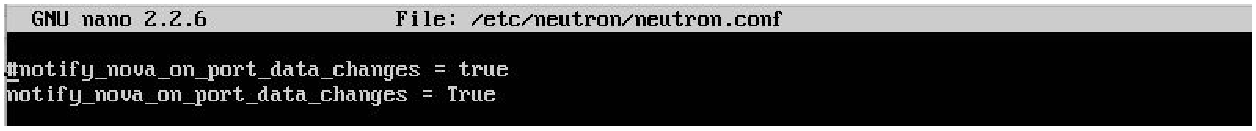
```
# The type of authentication to use (string value)

#auth_strategy = keystone
auth_strategy = keystone
```

Figure 60.



```
# The core plugin Neutron will use (string value)

core_plugin = ml2

# The service plugins Neutron will use (list value)

#service_plugins =
service_plugins = router
```

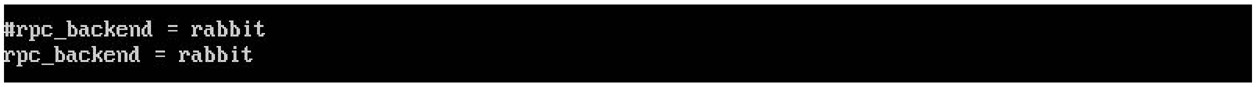
Figure 61.



```
  GNU nano 2.2.6              File: /etc/neutron/neutron.conf

#notify_nova_on_port_data_changes = true
notify_nova_on_port_data_changes = True
```

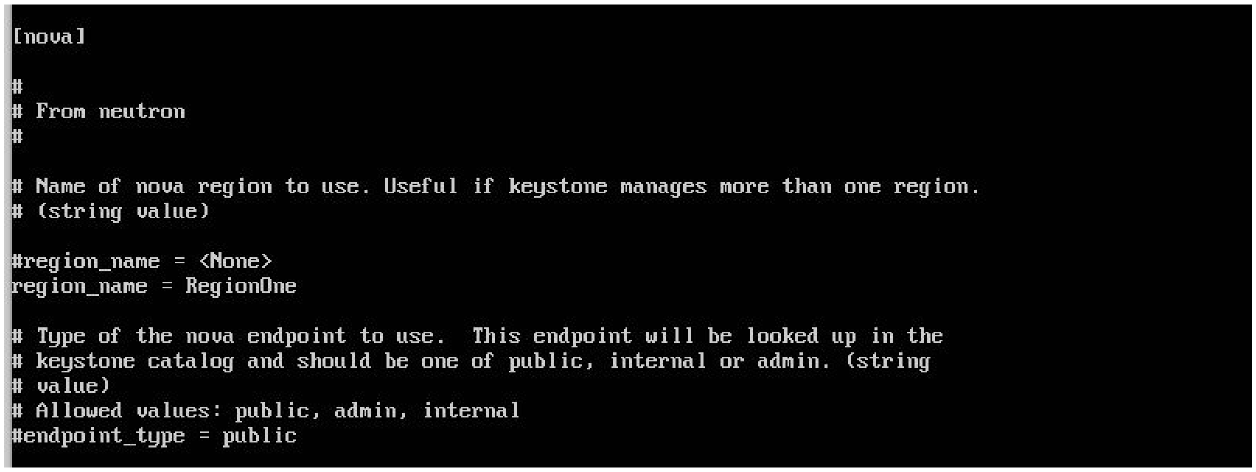
Figure 62

.



```
#rpc_backend = rabbit
rpc_backend = rabbit
```

Figure 63.



```
[nova]

#
# From neutron
#

# Name of nova region to use. Useful if keystone manages more than one region.
# (string value)

#region_name = <None>
region_name = RegionOne

# Type of the nova endpoint to use.  This endpoint will be looked up in the
# keystone catalog and should be one of public, internal or admin. (string
# value)
# Allowed values: public, admin, internal
#endpoint_type = public
```

Figure 64.

Here is the overview of the VTN design.

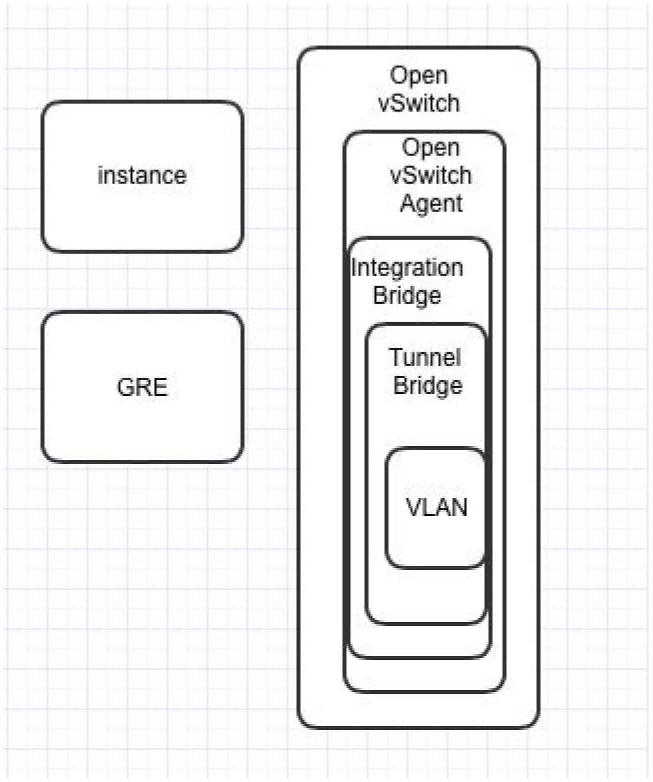


Figure 65.

In this way, virtual tenant networks are established and can hereby be managed through controller. Policies and protocols can also be added or modified through configuration files from controller with corresponding adjustments from each nodes.

5. Conclusion and Future work

In this project, I studied the knowledge of networking basis and SDN as well as VTN and NFV. With the reviewing of OpenDaylight documentations, I implemented the OpenDaylight SDN controller. Studying the cut-edge version of OpenStack, I implemented different instances of OpenStack nodes. Combining the leading applications of SDN, I integrated OpenStack and ODL. At the end of the project, I implemented a VTN Automation solution using Openstack and OpenDaylight SDN controller. Through this project, I get to know about what SDN is and the leading controller OpenDaylight as well as the OpenFlow specifications. Besides, a thorough understanding and application of OpenStack are also obtained from the project.

As far as this project concerned, the application of software defined network and its related development tools are limited due to time range, licence expense and compatibility between tools. In future, with development of this area, I believe there would be more use cases and explorations with relate to corresponding tools. More significant orchestration cases would be taken into consideration.

Reference

http://docs.openstack.org/mitaka/install-guide-ubuntu/keystone-users.html
https://help.ubuntu.com/lts/serverguide/network-configuration.html
https://wiki.debian.org/NetworkConfiguration
http://sciencecloud-community.cs.tu.ac.th/?p=238
http://ualberta.onthehub.com/WebStore/Account/SdmAuthorize.aspx?o=612a6bd0-76cb-e511-9414-b8ca3a5db7a1&ws=687574e5-7a59-e011-bd14-0030487d8897&uid=fe5fc175-76cb-e511-9414-b8ca3a5db7a1&sdm=0
http://blog.csdn.net/midion9/article/details/50748523
https://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=2084505
https://communities.vmware.com/thread/492851?tstart=0
http://quake.iteye.com/blog/1263961
https://help.ubuntu.com/community/Installation/SystemRequirements
https://wiki.opendaylight.org/images/5/55/Integrating-opendaylight-with-openstack-published.pdf
http://docs.openstack.org/mitaka/install-guide-ubuntu/overview.html#figure-hwreqs
http://docs.openstack.org/juno/install-guide/install/apt/content/ch_overview.html
https://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1022525
http://www.linuxidc.com/Linux/2012-04/58485.htm
http://docs.openstack.org/mitaka/install-guide-ubuntu/common/conventions.html
https://www.youtube.com/watch?v=bsaoU254gAc
http://askubuntu.com/questions/293827/error-rtnetlink-answers-file-exists
https://pubs.vmware.com/workstation-9/index.jsp?topic=%2Fcom.vmware.ws.using.doc%2FGUID-3B504F2F-7A0B-415F-AE01-62363A95D052.html
https://help.ubuntu.com/lts/serverguide/network-configuration.html
http://blogging.dragon.org.uk/setting-up-ntp-on-ubuntu-14-04/
https://ask.openstack.org/en/question/89131/creating-keystone-error-openstack-not-found-http-404-entity-liberty-debian/
http://askubuntu.com/questions/460022/using-terminal-as-a-web-browser
http://askubuntu.com/questions/460022/using-terminal-as-a-web-browser
https://www.openstack.org/summit/openstack-summit-atlanta-2014/session-videos/presentation/using-opendaylight-within-an-openstack-environment
https://mitakadesignsummit.sched.org/event/49yI/network-node-is-not-needed-anymore-completed-distributed-virtual-router
http://docs.openstack.org/kilo/install-guide/install/apt/content/neutron-network-node.html
https://ask.openstack.org/en/question/51388/whats-the-difference-between-flat-gre-and-vlan-neutron-network-types/

Appendix

A. The coding and configuration files are listed below. The nodes are edited within this report in the following order.

/etc/neutron/neutron.conf
/etc/nova/nova.conf
/etc/keystone/keystone.conf
/etc/apache/apache2.conf
/etc/glance/glance-api.conf
/etc/glance/glance-registry.conf
/etc/openstack-dashboard/local_settings.py
/etc/cinder/cinder.conf
/etc/heat/heat.conf

B. As the version of OpenStack we use in this project is the latest release by October, 2016. The documentation of OpenStack Mitaka has unavoidably included some bugs or accidentally omitted some details. As one of the first users, I noticed some during this project and revised them. The following are the issues I reported to OpenStack website. From the latest check, these bugs have been revised.

issue	reason	solution
HTTP 404	Only restart the apache2 server as suggested in the documentation is not enough	Reboot the operating system to make it get the configuration active
Command not found in OpenStack	There are several commands which only active in OpenStack when activated	Type the command:apt-get install python-openstackclient
Cannot verify operation as the admin user (HTTP 5000)	Service not activated	Service apache2 restart
There’s no “database” section in /etc/nova/nova.conf.file	It’s normal but ignored by the website as of October	Add this part manually to the file