Automation and Customization of OpenStack

B.E. (CIS) PROJECT REPORT

Project Group:

Danyal Javed	CS-16133
Muhammad Ammar	CS-16126
Muhammad Usman Ghani	CS-16129

BATCH: 2016-17

Project Advisor(s):

Dr. Muhammad Ali Ismail (Internal Advisor)

October 2020

Department of Computer and Information Systems Engineering

NED University of Engg. & Tech., Karachi-75270

Abstract

A user-friendly platform to automate the whole process of cloud development which can eliminate the time of deployment and substantial cost of hiring professional, based on an open-source software OpenStack. The OpenStack provides multiple cloud components for different use cases, and because of its open nature, anyone can add additional components to OpenStack to meet their needs. This makes the installation of OpenStack for the common user very confusing and time-consuming. The idea behind our product is to automate the whole process of cloud development. For this, we need some sort of automation tool that should generate server configuration files on run-time and help us to achieve our goal. Automation tools play an important role in terms of server configuration and management. These tools use provisioning scripts to make the server reach a desirable state. Our product has automated the process of configuring individual components of OpenStack using an automation tool and provide users an effortless installation of the cloud. Different scripts are created to automatically fetch the required components and configure them according to the specification set by the user. These scripts are written in YML language and we have used Ansible as our configuration manager. It detours all barriers and provides users with their own cloud platform running on their machine.

CONTENTS

CHAPTER 1

Introduction

1.1 Brief Introduction

A cloud solution based on OpenStack that allows users to create their own private or public cloud according to the requirement of the user, without the hassle of installing and configuring the individual technologies and services needed to build a complete running cloud environment. This solution eliminates the time of deployment and the substantial cost of hiring cloud architect professionals to build the cloud environment for you.

1.2 Problem Statement

The product is designed upon a free open-source cloud computing platform named OpenStack. The OpenStack is made up of many moving chunks known as components, it provides multiple cloud components for different use cases. Because of its open nature, anyone can add additional components to OpenStack to meet their needs. This makes the installation of OpenStack for the common user very confusing and time-consuming. The individual cloud component requires different configurations and components are usually installed on separate servers or machines to build a completely established cloud environment.

Our product has automated the process of configuring individual components and provides users an effortless installation of the cloud. We have written some scripts that automatically fetches the required components and configures them according to the specification set by the user. It detours all barriers and provides users with their own cloud platform running on their machine.

1.3 Objective

To develop an automated cloud installation setup that can generate a cloud environment consisting of pre-configured virtual machines and customized user-friendly interface for easier VMs management and usability. The generated cloud environment can be customized depending on the users' selected configuration. These cloud configurations or packages are work usage-based like an educational package with minimal installation for learning purposes.

1.4 Scope of Work

Small scale organizations and educational sectors can take advantage of our solution. They can build their cloud platform without spending on public cloud service providers. It can help organizations to focus more on their business rather than wasting time and money on the technical side of cloud development. This solution can also come very handy for DevOps Engineers and others as they can use this platform for learning and training. All in all, this solution provides a cloud platform to everyone without the struggle of configuration and installation.

CHAPTER 2

Project Background

Cloud computing provides users with a shared pool of resources that includes resources related to networking, storage, processing, and servers. There are unlimited use-cases for cloud computing. A tremendous increase has been noticed for computing resources and to deal with the demand people come up with different solutions. There are several types of cloud models such as private cloud, public cloud and hybrid cloud. Our project focuses on automating the process of deploying a private cloud through open-source software. Girish L S. [1] has discussed in detail about building a private cloud using OpenStack. In the open-source market, plenty of software is available that can build a private cloud such as OpenStack, Eucalyptus, CloudStack and OpenNebula.

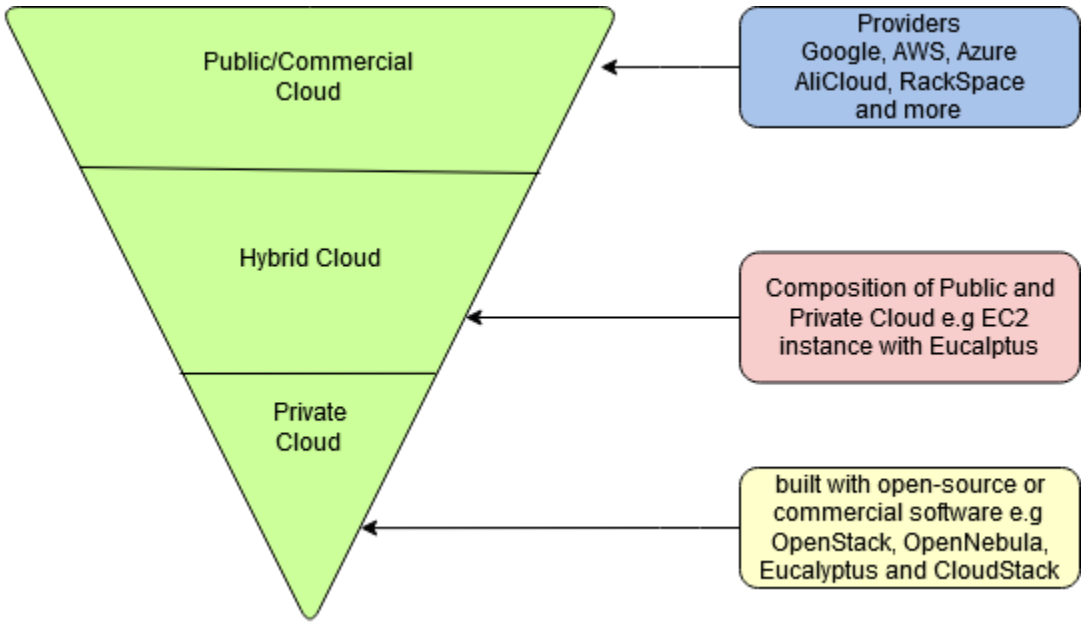

Figure 2.1 Types of Clouds and their provider

We choose OpenStack because of its highly scalable structure and very cooperative community. Also, OpenStack has been used by so many large companies which makes it even more favorable over other Software. OpenStack itself is highly customizable and configurable but at the same time, it can get very confusing for the users to implement it themselves. Our goal is to automate the installation and deployment phase of OpenStack so that small startups, the education sector can set up their private cloud without a hassle.

There are not many products or solutions that directly automate the process of setting up a private cloud with OpenStack. Two such products that are similar to our project are DevStack [2] and PackStack [3], but these products have their shortcomings. DevStack and PackStack are widely supported on different Linux distros but most preferable distros for OpenStack installation is Ubuntu, CentOS or RedHat

DevStack is a set of pre-defined scripts developed by OpenStack that can set up an OpenStack cloud on your machine or any test server. It is installable on the latest Ubuntu version, Fedora, and CentOS. DevStack is an all-in-one installation that is good for people or developers to get acquainted with the OpenStack environment without having in-depth knowledge about the OpenStack lego-like structure. the installation of DevStack is very susceptible to error and it is not meant to be used in production because after a reboot. It is impossible to get back to the recent stable state all the running processes crashed, and data will be lost permanently.

PackStack is similar to DevStack. It is also a utility to install OpenStack. but it is only supported for RedHat based Operating Systems. PackStack is a little mature in terms of the installation process and stability compared to DevStack. PackStack uses puppet under the hood to deploy various services of OpenStack on multiple servers. PackStack is also persistent.

Our solution is comparable to PackStack implementation in a way that we also have a configuration management tool to install and configure all the services. Our installation process is not limited by vendor-specific operating systems like PackStack. It is installable on all OpenStack supported Linux distros. We also enforced persistence so that in case of a power failure we can go back to the last stable state, unlike DevStack. In terms of architecture, we also want to provide flexibility to use the user so that they can build around their needs. PackStack and DevStack do not have any kind of central logging mechanism which can be problematic if anything fails it can be hard to trace it down without proper logs. In the project, we also implemented a centric logging mechanism to provide end-users with complete formatted and GUI friendly logs. Our target is to automate the installation at the same time providing a production-ready solution.

CHAPTER 3

OpenStack

3.1 Brief Overview

OpenStack is an open-source software which is capable of deploying and managing a cloud infrastructure. OpenStack can be used to deploy both public and private cloud. Some of the companies commercially providing public cloud services based on OpenStack so far in 2020 are RackSpace Public Cloud, STS Cloud, OVHCloud, CityCloud, and Open Telekom Cloud (T. Mobile). In terms of private cloud solutions over 100 giant companies are using OpenStack in their technology stack including Paypal, Hubspot, Intel, Mirantis, Canonical, IBM, and much more.

The reason why OpenStack is used by so many powerful companies is that it fulfils two important factors: scalability and flexibility. Lespinasse, F [4] has comprehensively discussed about these factors in his article. It makes OpenStack highly customizable and can be easily built around the consumer's requirements. OpenStack achieves its high configuration ability from its design through the use of different services that are solely responsible for doing one task either be processing, storage, networking etc. More services can be easily added and configured on-the-go based on need. Each service provides an application programmable interface endpoint which allow it to communicate with other services. Another reason for adopting OpenStack is that it supports almost all major hypervisors (Xen, VMware and virtual-based KVM) and also provides support for bare-metal installations.

OpenStack is continually improving due to its open-source community. A new OpenStack release has been coming out every six months. Since its first release in 2010-10-21 named **Austin**.

A total of 21 releases come out as of 2020. Every release is maintained and supported for almost three years. In our Project, the release named **Rocky** is used which came out in 2018-08-30.

3.2 Essential OpenStack Services

OpenStack is configurable in a million ways with its various components. Each component is responsible for providing a different service. Although some essential services are mandatory to set up a cloud infrastructure using OpenStack. All the services used in our project are listed below

3.2.1 Compute

OpenStack uses compute service code-named **NOVA**. it acts as a system controller it is responsible for provisioning and regulating users access to the virtual machines. Nova itself doesn't include any virtualization software but utilizes the underlying hypervisors to deploy VMs. Nova consists of multiple daemons each responsible for a different task. These daemons can reside on the same node or separate nodes depending on the requirement. In our project, these daemons are configured on the same node.

nova-compute	It is primarily responsible for creating and terminating instances. It receives requests from the message and performs it using libvirt library for creating virtual machine instances.
nova-conductor	It deals with all the database related operations. Originally these operations were performed by nova-compute but now they are performed by nova-conductor which also reduces security risks
nova-scheduler	It is responsible for deciding on which node the new virtual machine should run.

All of the above-mentioned services are horizontally scalable, multiple instances of each service can be run on different nodes.

3.2.2 Object Storage

OpenStack has a service code-named **SWIFT** that provides object-storage functionality. Swift provides redundant, scalable data storage. it is capable of handling thousands of petabytes of static data with the ability to retrieve and update. OpenStack ensures data replication and is horizontally scalable. New storage nodes can be added on-the-go and in case of failure, data is replicated on new nodes. It is perfect for archiving, backups and store data that can grow exponentially.

3.2.3 Block Storage

OpenStack uses a service code-named **CINDER** to implement block storage. Cinder responsible for provisioning and managing storage volumes for virtual machines. It is implemented through the use of LVM or other plugin drivers for storage. Cinder provides end-users with a self-service API to easily attach or detach volumes from the virtual machine. Cinder also provides snapshot management and the ability to clone storage volumes.

3.2.4 Authentication

Authentication is essential in the OpenStack environment since each service provides an API endpoint to interact with it. To allow access to only authenticated users OpenStack uses a service code-named KEYSTONE. It securely checks user identity and on successful authentication, a token is assigned to that user. Now, whenever a user requests any service endpoint it needs to send this token in the request body. There are multiple versions of keystone we use v3 in our project due to its reliability and security.

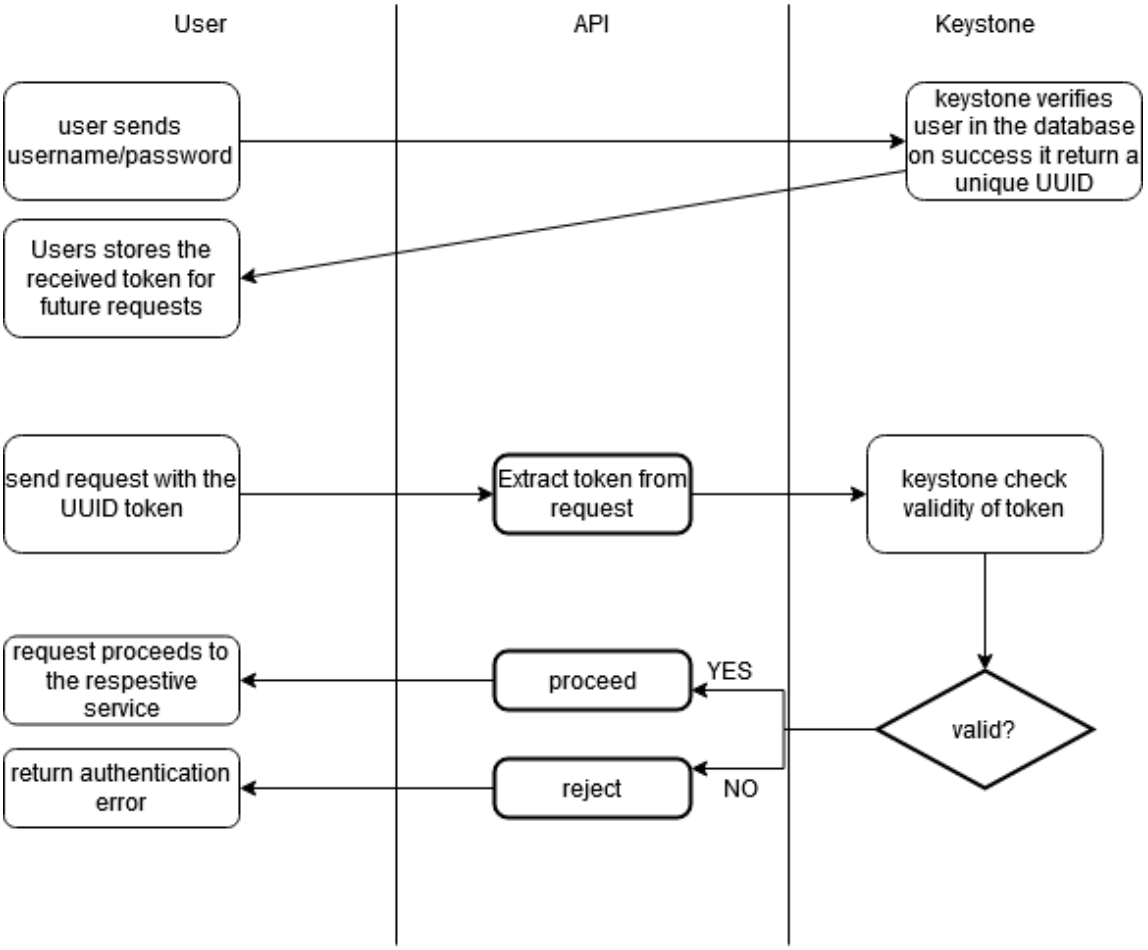

Figure 3.2.4 Workflow of KeyStone

3.2.5 Image Service

OpenStack uses a service code-named Glance to provide image service which is responsible for managing and registering images for disk and servers. Images can be divided into two formats disk and container. Disk formats include ISO, VHD and QEMU image format etc. while container format includes docker images. Glance consists of multiple sub-components

- *glance-API* that deals with API call for the image retrieval, storage or simple discovery,
- *glance-registry* retrieves images, processes image storing,

- *the database* holds complete metadata for each image and
- *glance storage repository* There can be multiple locations available for images repositories typically there are stored in object storage provided by swift or external on amazon s3.

In our project, we use an image of a very lightweight Linux distro named cirrOS. It is the only six megabytes in size and great for testing. Depending on the need we can get any image. These images are available at the time of creating new virtual machines.

3.2.6 Dashboard

OpenStack uses a service code-named **HORIZON** which provides end-users with a web-based user-friendly interface to interact with all the services configured. It provides an overview of all the VMs running with their resource consumption. By default, horizon displays information related to core services only. but new services can also be added to the horizon dashboard by little configuration.

3.3 Other Services

There are a lot more services that can be configured with OpenStack. Few of them will be discussed later with in-depth. These services include Neutron which is responsible for providing networking ability to Virtual machines and the other is Heat which is an orchestration service it helps in implementing stacks according to end-users

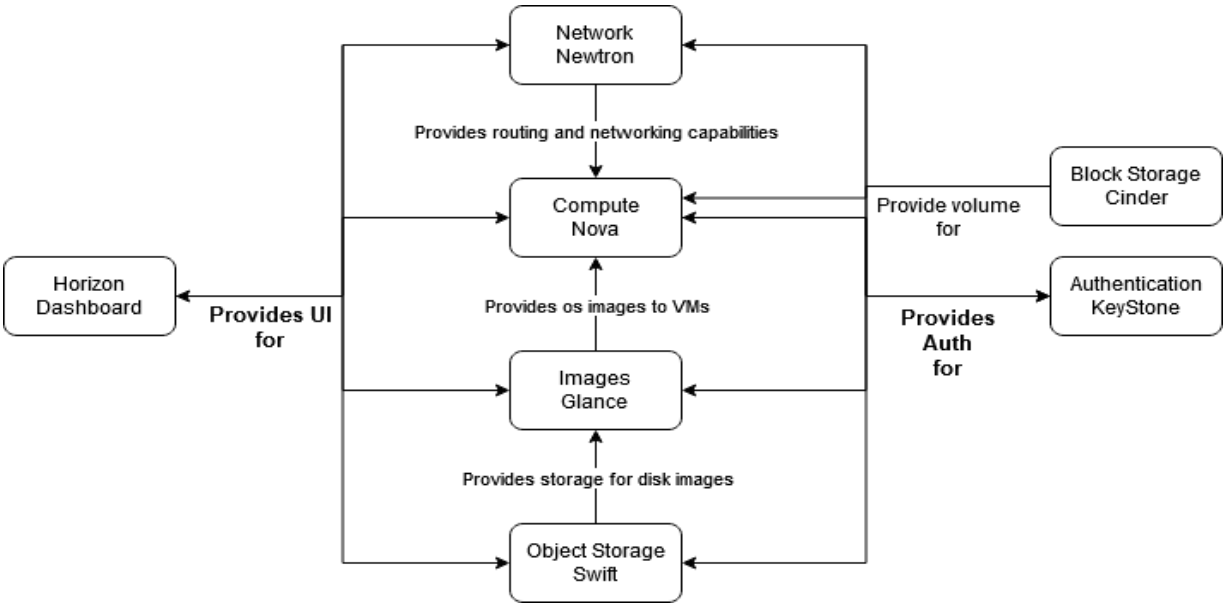

Figure 3.9 Interaction of OpenStack services

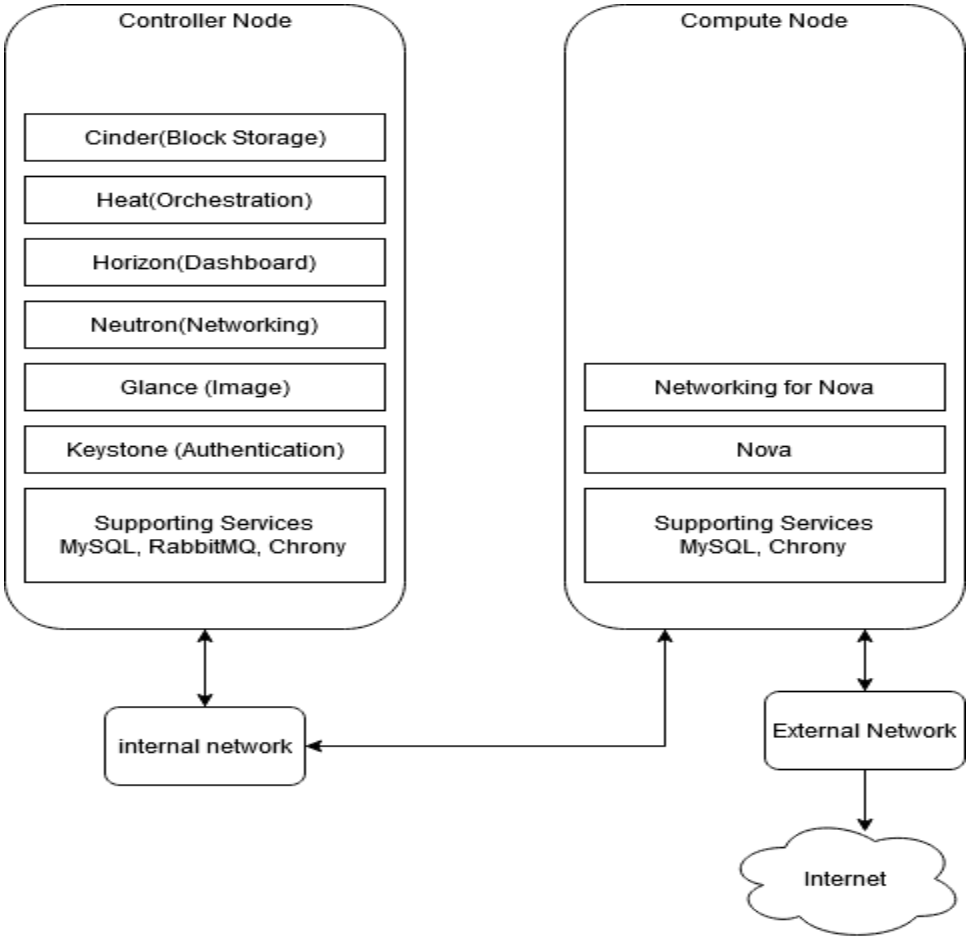

Figure 3.0 Architecture Used

CHAPTER 4

Configuration Management Tool

4.1 Automation Tools

The idea behind this product is to automate the whole process of cloud development. For this, we need some sort of automation tool that can generate server configuration files on run-time and help us to achieve our goal. Automation tools play an important role in terms of server configuration and management. These tools use provisioning scripts to make the server reach a desirable state.

There are multiple configuration management or automation tools available in the market with Puppet, Chef, Salt, and Ansible being the most popular and reliable choice. Each of these tools poses different characteristics and each works differently. Although the purpose of each is the same; make sure the system's state matches the state defined by your provisioning scripts.

In the early stage of our project, we chose Chef [5] as our Automation Tool because it provides a proper learning feature on their website for beginners, which helped us to understand how automation tools work and how we can use it in this product. But then, because of the limited resources, we decided to change our configuration tool and we moved towards Ansible [6].

4.2 Chef

The Chef is an open-source cloud configuration tool that provides a way to define infrastructure-as-code. It simply means defining the infrastructure of the server using code or

script. In the Chef domain, we call this script a **Cookbook**. These configuration scripts are written in a domain-specific language (DSL) and they are stored on the Chef server.

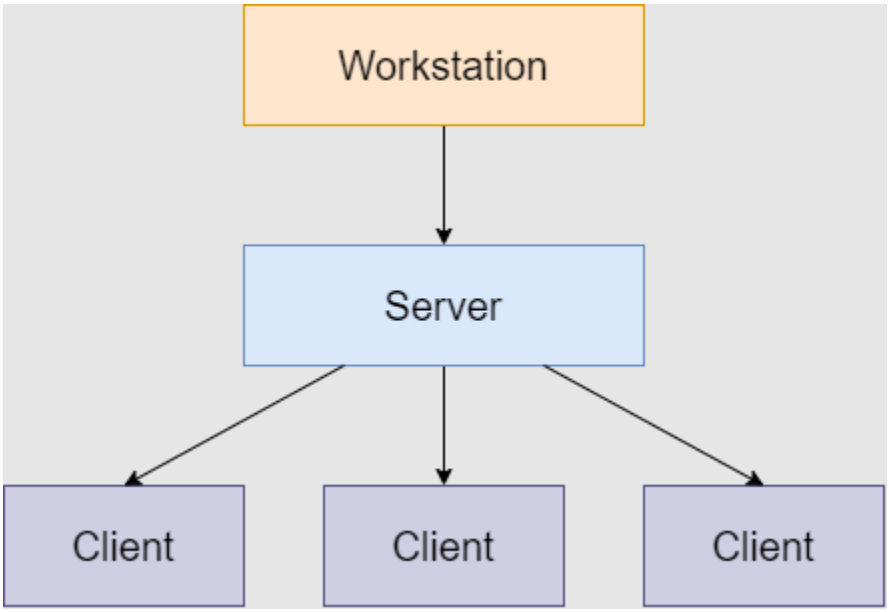

Figure 4.2 Chef Architecture

Chef has a master-client architecture, there exists one master machine with multiple client machines. The server runs on a master machine, while the client part runs as an agent on each client machine. Chef also requires a separate workstation to control the master. This workstation contains all the configurations which are tested and then push to the central Chef server. Clients or agents are also installed through workstation using a command-line tool called Knife. All these configurations to build the proper architecture of Chef makes it difficult to manage when you have a limited number of resources.

4.3 Ansible

Ansible is also an open-source configuration management tool but follows a simpler approach for configuration management work. Ansible is the only automation tool that automates

the entire application with continuous delivery. The architecture model followed by Ansible is different and simpler than Chef as it does not use master-client architecture.

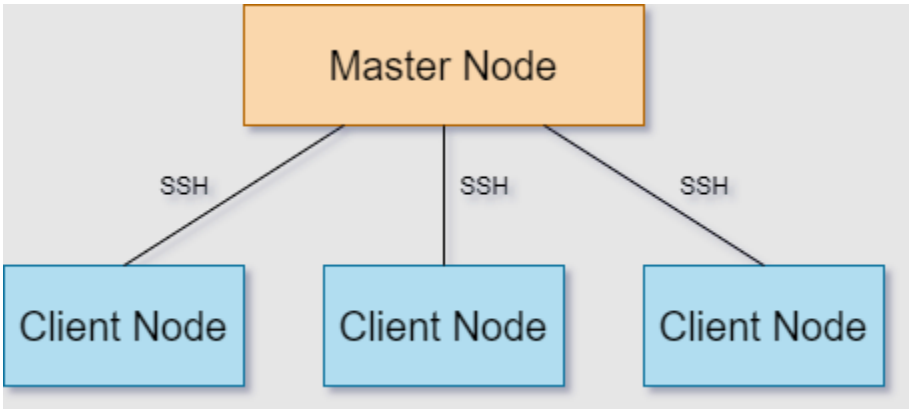

Figure 4.3 Ansible Architecture

Ansible uses an agentless architecture, which means it has a master running on the server machine and it requires no agent on the client machine. To configure each client machine it uses SSH connection and configures each node. This makes Ansible faster and easier to set up as client machines require no special setup. As Chef calls its configuration file, code, or script as Cookbook, here Ansible calls it Playbook. These Playbooks are written in YML (Yet Another Markup Language - Python) which is quite easy to learn and understand. Also, Python is inbuilt in the Linux environment, so there is no need to set up a language environment for each machine. Ansible uses a push-based synchronization approach, the server pushes all the configuration to all client nodes, which is good for real-time based applications.

4.4 Playbook

To configure each client node, Ansible uses a configuration file or script written in YAML language, known as Playbooks. It is a core feature of Ansible, it tells what exactly needed to be done to configure the whole machine environment. It contains several steps that should be executed

sequentially on a particular machine. Each playbook has one or more '*plays*' in it. Execution of the playbook is sequential, and within each play, tasks are also executed sequentially in the top to bottom order. To orchestrate multiple machines deployment, we add multiple plays inside one playbook. One play can be used for configuring the database servers, other play can configure web servers, then another play can take part in network infrastructure configuration, and so on.

To configure the whole cloud environment of our project, we have created one Playbook having 7 plays in it with multiple roles.

```
- hosts: controller
  become: true
  user: ubuntu
  gather_facts: true
  roles:
    - set_env_controller
- hosts: compute
  become: true
  user: ubuntu
  gather_facts: true
  roles:
    - set_env_compute
```

playbook.yml (1-2 plays)

This playbook file contains 7 plays. In the first two plays, by mentioning *set_env_controller* and *set_env_computer*, we are setting up the basic environment of OpenStack in both of the nodes. The first play is for the Controller node and the second play is for the Compute node. Both plays are being used for the same purpose, it will perform a basic operation on each node to build the environment for OpenStack. Basic operations such as DNS setting, general packages installation, database installation, and message queue configuration.

```
- hosts: controller
  become: true
  user: ubuntu
  gather_facts: true
  roles:
    - keystone
```

```
    - glance
    - nova
- hosts: compute
  become: true
  user: ubuntu
  gather_facts: true
  tasks:
    - import_role:
        name: nova
        tasks_from: compute
```

playbook.yml (3-4 plays)

In the third and fourth play, we are installing a very important component of OpenStack in both nodes i.e. a Nova component. But in the Controller node, we first install the Keystone component because it does not depend on any other OpenStack service or component. Also, we installed Glance, a virtual image service required for organizing a large set of virtual disk images. To install Nova, it requires half configuration on Controller and half configuration on Compute, that is the reason, we are installing Nova in the last step of 3rd play and then in the first step of 4th play. We always need to create a new play when we need to change the host.

```
- hosts: controller
  become: true
  user: ubuntu
  gather_facts: true
  roles:
    - horizon
    - neutron
- hosts: compute
  become: true
  user: ubuntu
  gather_facts: true
  tasks:
    - import_role:
        name: neutron
        tasks_from: compute
```

playbook.yml (5-6 plays)

We need Horizon for the virtualization of the whole cloud. The horizon always installs on the Controller node, so in the first step of 5th play, we are installing Horizon on the Controller node. Neutron is another component required for cloud configuration, and it has the same

installation process as Nova i.e. half configuration on the Controller node and half configuration on the Compute node.

```
- hosts: controller
 become: true
 user: ubuntu
 gather_facts: true
 roles:
   - heat
```

playbook.yml (7th play)

The last play is used for the OpenStack orchestration service called Heat. We installed Heat in the last play because to configure Heat properly, it requires a running version of OpenStack as this is service is dependent on other OpenStack services.

CHAPTER 5

Centralized Logging and Monitoring

For monitoring, we used a log management platform that helps in debugging the overall cloud environment state, known as ELK. It is an open-source, most popular log analysis software provided by the elastic company. The ELK stack comprises Elasticsearch, Logstash, and Kibana, each of these products are open source.

Elasticsearch - It is a search engine that powers the stack, used for deep search and data analytics.

Logstash - It processes the data and sends it to Elasticsearch for storage and indexing. It is also used for centralized logging, parsing, and log enrichment.

Kibana - It is a powerful and beautiful user interface of the Elasticsearch cluster where users can visualize log messages.

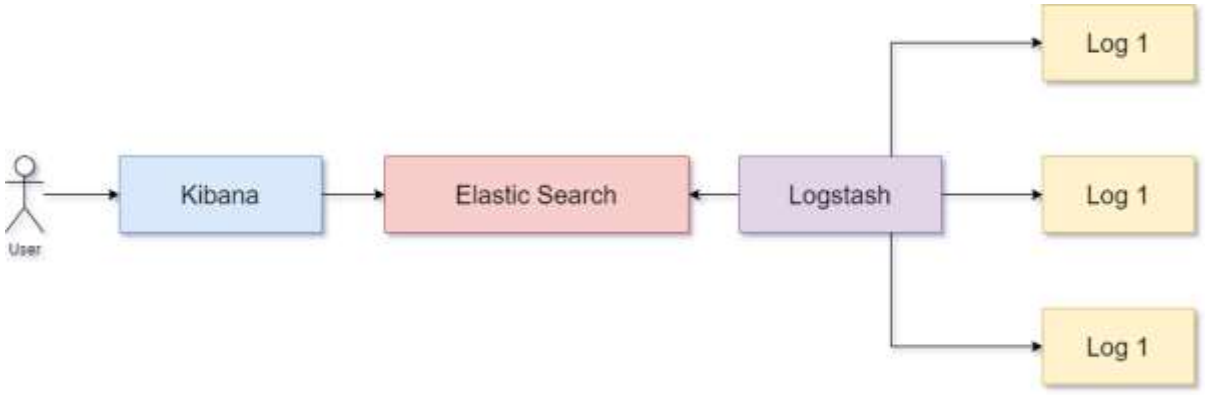

Figure 5.1 ELK Stack Interaction

We have installed this ELK cluster on a single node of our project. With two nodes i.e. controller node and compute node, we are sending logs to Logstash from both of these nodes. To

send these logs, we used a lightweight shipper agent for forwarding and centralizing the log data known as **Filebeat**. It ships all the logs mentioned in configuration to the ELK cluster. So, we have installed this Filebeat agent on both the Controller and the Compute node to send logs details on a Logstash which is kind of a central database for logs. After installing the Filebeat agent on each node, we need to mention in the configuration of the Filebeat input, which logs agent should send to Logstash.

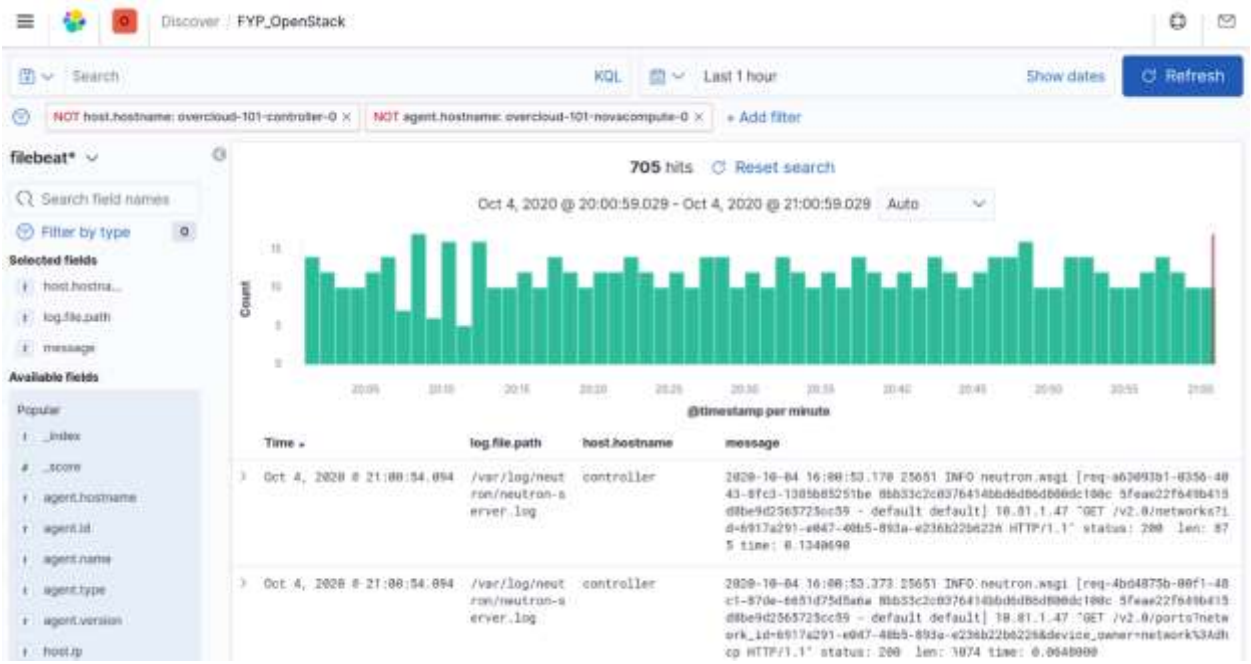

Figure 5.2 Kibana Dashboard

From the controller node, we are sending log files of apache, glance, heat, keystone, database, and some OpenStack component log files.

```
# ================ Filebeat inputs (Controller_Node)====================
# Paths that should be crawled and fetched.
paths:
 - /var/log/apache2/*.log
 - /var/log/glance/*.log
 - /var/log/heat/*.log
 - /var/log/keystone/*.log
```

```
 - /var/log/mysql/*.log
 - /var/log/neutron/*.log
 - /var/log/nova/*.log
 - /var/log/rabbitmq/*.log
```

And from the compute node, the following files are being sent to Logstash.

```
# ================== Filebeat inputs (Compute_Node) ===================
# Paths that should be crawled and fetched.
paths:
 - /var/log/neutron/*.log
 - /var/log/nova/*.log
```

We have only mentioned specific paths in input files of Filebeat. But we also need to mention the destination. Where should Filebeat need to send these logs? For this, we need to create a Filebeat output file on both of the nodes, where we define the IP of the Logstash database cluster with a port number, to correctly specify the destination address.

The output file is the same in both nodes because the destination is the same.

```
# ================== Logstash Output ====================
output.logstash:
 # The Logstash hosts
 hosts: ["10.81.1.39:5044"]
```

Not only Filebeat sends data to the Logstash database cluster, but it also stores information about sending logs in each node. In this logging file, we define where should the agent store sending log information locally.

```
# ================== Logging ====================
#filebeat operations logging destination
logging.level: info
logging.to_files: true
logging.files:
 path: /var/log/filebeat
 name: filebeat.log
 keepfiles: 7
 permissions: 0644
```

CHAPTER 6

Sample Environment

We build a sample environment for web hosting by using Heat templates. The entire environment is built on user's requirements and resources. All a user must do is to write all requirements in a YAML file and Heat build that desired stack for him. Given below is the sample file for the stack requirements:

```
parameters:
  image: nginx
  key_name: mykey
  flavour: m1.medium
  network-ext: "External Network"
  network-pvt: "Private Network"
  server_name:
    - Server-A
    - Server-B
  lb_name:
    - LoadBalancer
```

The figure below depicts a minimal hosting platform in which we create two web servers and on load balancer for http/https traffic load balancing. We use NGINX to serve us both as a web server and load balancer. Load balancer and both servers are connected to a private network. All north-bound traffic from them are forwarded to private subnet. For a load balancer we create an external bridge network to be accessible from the external network. A floating IP (static public IP) is assigned to a load balancing instance and all south bound traffic is forwarded through load balancer to the web servers. Web Servers are kept isolated from external networks and all egress/ingress traffic is targeted to a load balancer. Layer 3 security groups are applied on both web servers to stop any unnecessary ingress traffic from unknown hosts. For management purposes all egress traffic of web servers are DNATed to the external network.

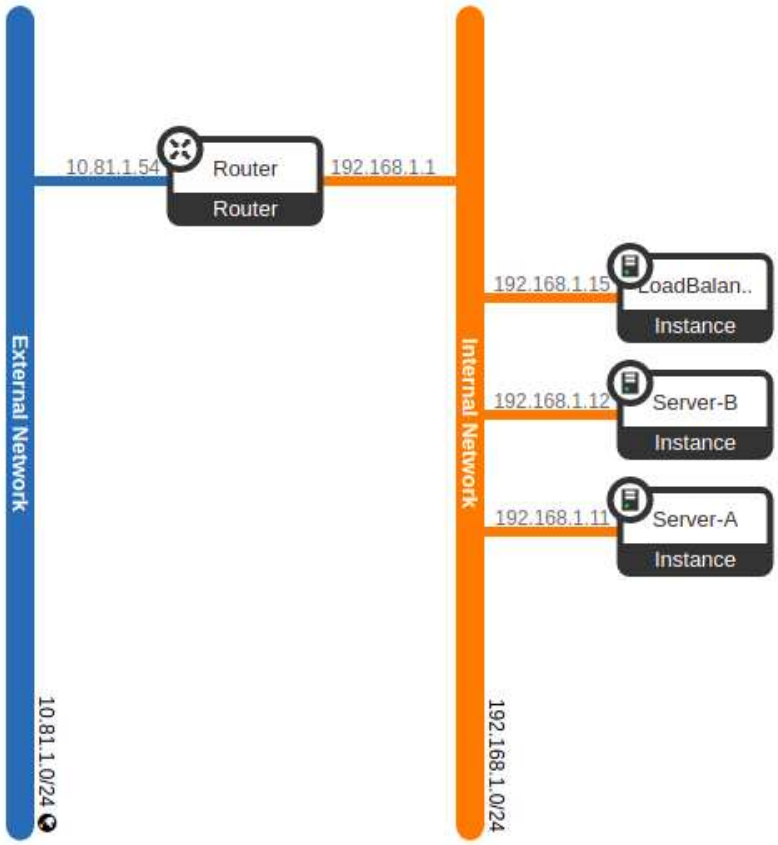

Figure 6.1 Network Topology

CHAPTER 7

Networking Details

This section is exclusive for neutron although we discuss it above in the OpenStack section. The need for this section is that unlike other OpenStack components we use in this project, neutron is a beast. It is too difficult to trace the problem unless you have a deep understanding of it. Before we start, first we have to know five main concepts of Linux networking devices.

- **TAP Device**

  TAP device is a software-only virtual interface that is attached to software program and starts sending ethernet frames to it. Basically, it is used by hypervisor like kvm to build connectivity to VMs.

- **Veth Pair:**

  It connects two different network entities like network namespaces and bridges together. It acts like a virtual wire that connects two virtual NICs.

- **Linux Bridge:**

  It is a virtual switch in Linux. We can add physical and virtual interfaces to it. It operates at layer 2 (MAC Addresses).

- **OpenvSwitch (OVS):**

  OpenvSwitch is more complicated switch than Linux bridge. Like Linux bridge we can also add physical and virtual interfaces to it. The big advantage is to that we can add open flow rules to it which can manipulate layer 2 traffic.

- **Network Namespaces:**

  Network namespaces are isolated network stack in Linux. We can think of it as a

different Linux installation in terms of networking. We can add different interfaces, iptables and routes to them. In OpenStack we have several users having different networks. So, by using network namespaces we can have overlapping IPs. Neutron uses it to implement DHCP-agent and l3-agent.

Neutron has five different sub-components which have different jobs to do:

- **Neutron Server:**

  Allows user to create networks and router

  Provide an API and manages the database

- **L3-Agent:**

  It routes L3 traffic and provides floating IPs.

- **ml2-Agent:**

  It switches layer 2 traffic.

- **DHCP-Agent:**

  Provides DHCP to instances

- **Metadata-Agent:**

  Provides metadata to instances

There are three types of scenarios for VMs to communicate with each other. We discuss each of them in detail how their traffic is forwarded from source host to destination host.

- ## 7.1 Same Network Same Host

  When we create a VM, there is a virtual NIC card inside it. It generates traffic. Traffic goes down to a tap device. Tap device is the first point of entry and exit for VM. So, it's the best place to apply security groups on it. Tap device is connected to a qbr which is a Linux bridge. It connects the tap device and the qvb. qvb and qvo are veth pair. Now the traffic goes to the integration bridge. Int-br is an OpenvSwitch bridge. It is a VLAN tap. Traffic get tagged there and if the traffic is destined to the VM on same network and same host, then this is the endpoint for it.

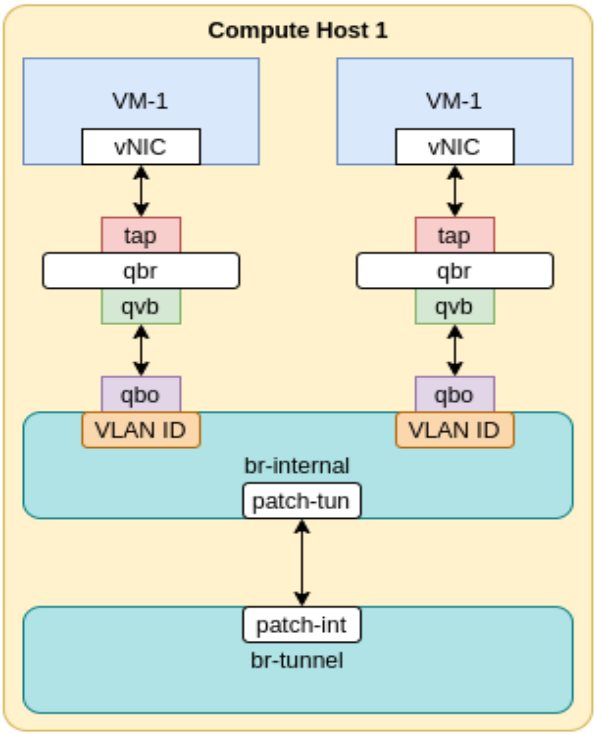

Figure 7.1

- ## 7.2 Same Network Different Hosts

  If VM spawned by nova-scheduler is in different host then we all the steps will follow as its. But in this case traffic has to exit the interface, so it goes to patch interface and from there it goes to overlay network. We can think of it as a highway connecting two cities but, in this case, it connects two compute hosts.

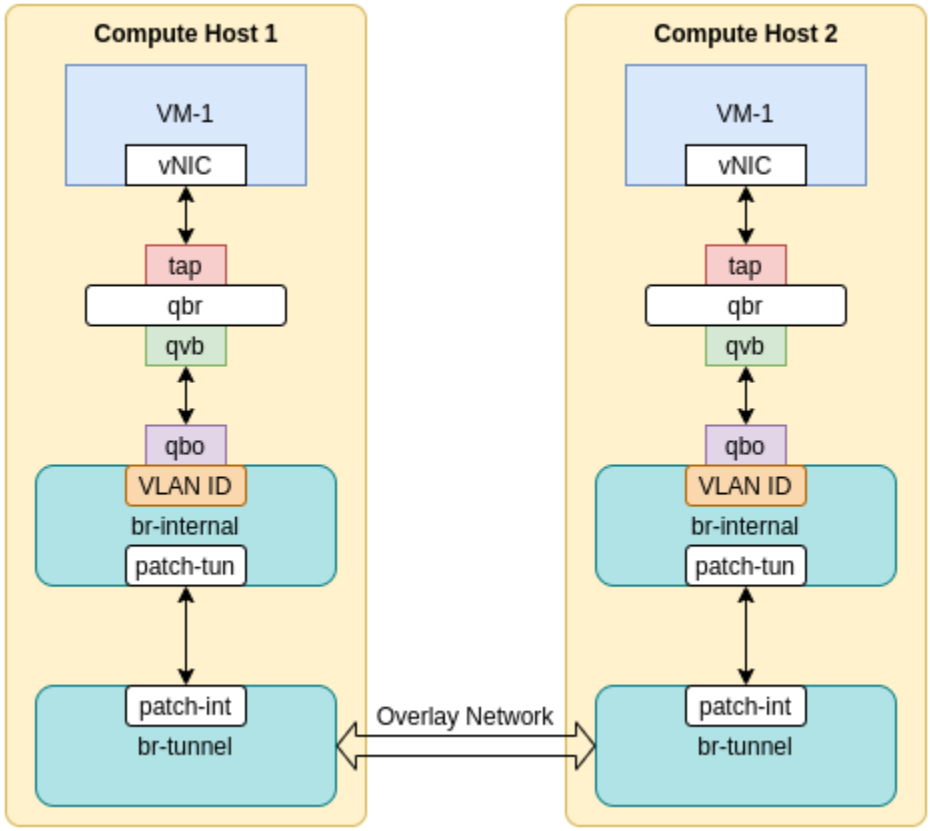

Figure 7.2

- ## 7.3 Different Network Different Hosts

  In this case the traffic follows the same route up to the bridge tunnel. Traffic leaving the patch interface is untagged and then sent to the bridge tunnel. After that, the traffic is routed by router to the destined hosts.

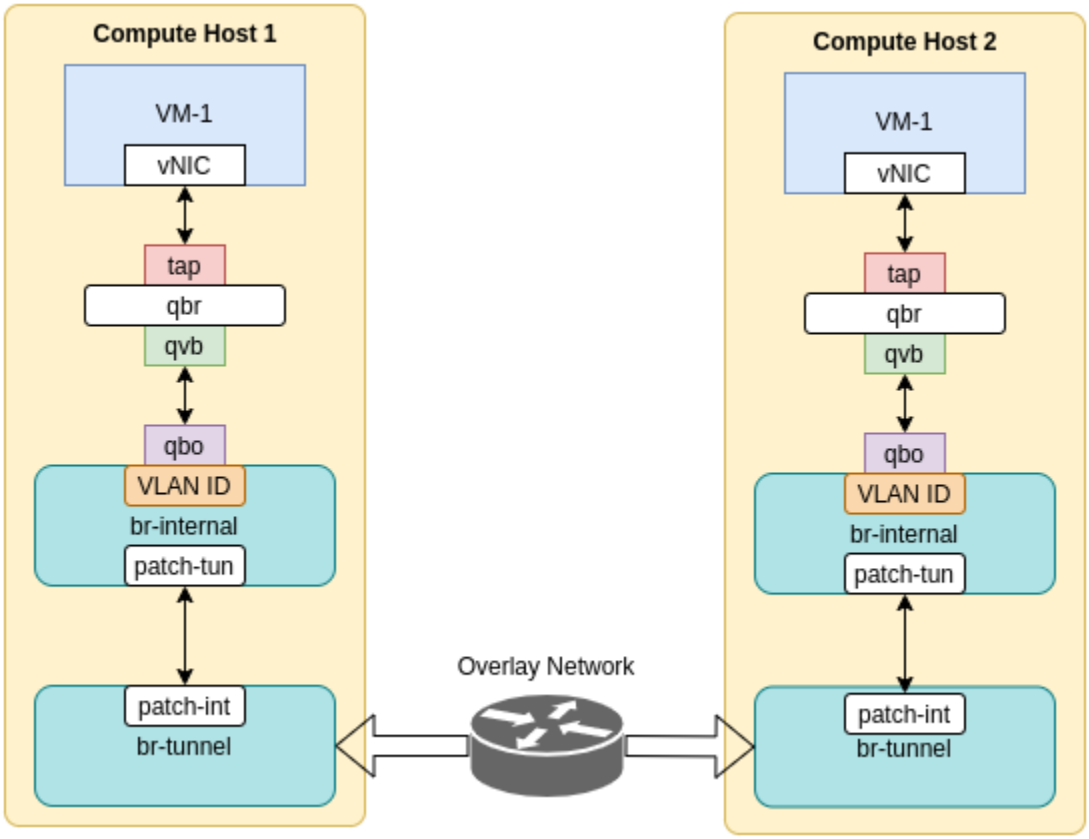

Figure 7.3

CHAPTER 8

Deployment Guide

8.1 mycloud – Command line utility:

we develop a command line utility for ease of deploying openstack. Ita has following functionality:

- `$ mycloud node list`
- `$ mycloud node add [node-name] [ip]`
- `$ mycloud node delete [ip]`
- `$ mycloud node introspect [username]`
- `$ mycloud node deploy`

8.2 Steps to Reproduce:

1. $ git clone https://github.com/M-Ghani97/mycloud.git
2. $ cd mycloud
3. $ ./prep.sh
4. $ mycloud node add <node-name> <ip-address>
5. $ mycloud node introspect <username>
6. $ mycloud deploy -y

CHAPTER 9

Conclusion and Recommendation

The main idea behind this product is to automate the whole process of cloud development using OpenStack, and we have successfully automated the process of configuring OpenStack based on user's requirements. In our current workflow, the requirement is gathered and fed into the script through the command-line. But we wanted to gather these requirements through a GUI and process them directly without the use of command-line. Some more enhancement can also be made, such as additional pre-configured packages can be provided to the user to choose from. Currently, this product is limited to a basic learning purpose environment and a web hosting environment. Integrating more OpenStack services will allow us to make more pre-configured packages e.g.

packages for Big Data processing, Database-as-a-service, and much more. Another enhancement would also be made, is to add some type of self-healing mechanism to deal with crashes. The current central logging system can be improved with better reporting capabilities. Most importantly the project should be licensed and implemented with proper pricing structure before product distribution. These enhancements will allow the project to reach its full potential by providing a reliable automated OpenStack deployment.

Abbreviations

VM – Virtual Machine

API – Application Programmable Interface

LVM – Logical Volume Manager

KVM – Kernel Virtual Machine

DSL – Domain Specific Language

YAML – Yet Another Markup Language

VLAN – Virtual LAN

NIC – Network Interface Card

TAP – Network Test Access Point

DHCP - Dynamic Host Configuration Protocol

VHD - Virtual Hard Disk

ISO - International Organization for Standardization

QEMU - Quick EMUlator

References

1. Girish L S, Dr. H S Guruprasad. (June 2014). Building Private Cloud using OpenStack https://www.ijettcs.org/Volume3Issue3/IJETTCS-2014-06-17-090.pdf

2. DevStack documentation https://docs.openstack.org/devstack/latest/

3. PackStack documentation https://wiki.openstack.org/wiki/Packstack

4. Lespinasse, F. (August 2014). A quick overview of OpenStack technology. Retrieved from https://www.ibm.com/blogs/cloud-computing/2014/08/06/quick-overview-openstack-technology/

5. Chef documentation https://docs.chef.io/

6. Ansible documentation https://docs.ansible.com/ansible/latest/index.html