NFV Orchestration using OpenStack

by

Simar Arora
B.Tech., DAV College of Engineering & Technology, 2013

A Project Submitted in Partial Fulfillment
of the Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

NFV Orchestration using OpenStack

by

Simar Arora
B.Tech., DAV College of Engineering & Technology, 2013

Supervisory Committee

Dr. Sudhakar Ganti (Department of Computer Science)

Co-Supervisor

Dr. Daniela Damian (Department of Computer Science)

Co-supervisor

Abstract

Supervisory Committee

Dr. Sudhakar Ganti (Department of Computer Science)

Co-Supervisor

Dr. Daniela Damian (Department of Computer Science)

Co-supervisor

This work demonstrates the study and experimentation of network function virtualization on OpenStack environment. OpenStack is a cloud computing platform which provides shared resources to perform computations and manage data with the help of virtual machines. Additionally, OpenStack serves as network function virtualization infrastructure platform to deploy and operate Virtual Network Functions such as firewall, router or a load balancer which can replace propriety hardware and orchestrate end-to-end network services.

Table of Contents

List of Figures

Acknowledgments

I sincerely thank my supervisor Dr. Sudhakar Ganti for his expert guidance and encouragement.

Also, I would like to thank my girlfriend Sakshi, for providing an invaluable support throughout this degree and beyond.

Dedication

To my parents, my pillar of strength.

Chapter 1

Introduction

Cloud computing has brought a dynamic shift in IT industry in the recent years. Corporate giants like Netflix and Dropbox use cloud computing as their backbone. Cloud computing is based on the model of sharing resources including standard high-volume servers, networking, storage, and switches. The key features of cloud computing include on-demand self-service, rapid elasticity, resource pooling.

Companies benefit from this model by paying as per use and hence deploying applications on those resources. The implications of having such environment are huge and one of the benefits is the origination of Network Function Virtualization.

1.1 Network Function Virtualization

Background

Provisioning network services within the telecommunication industry up until now has followed the approach where network operators install proprietary hardware for specific network function. This approach brings out issues of compatibility and chaining when different hardware components are brought into the equation.

Couple these issues with quality control and scalability requirements and we are stuck with long product cycles and ever-increasing dependence on brand-named hardware. The figure [2] below describes the traditional approach of network architecture.

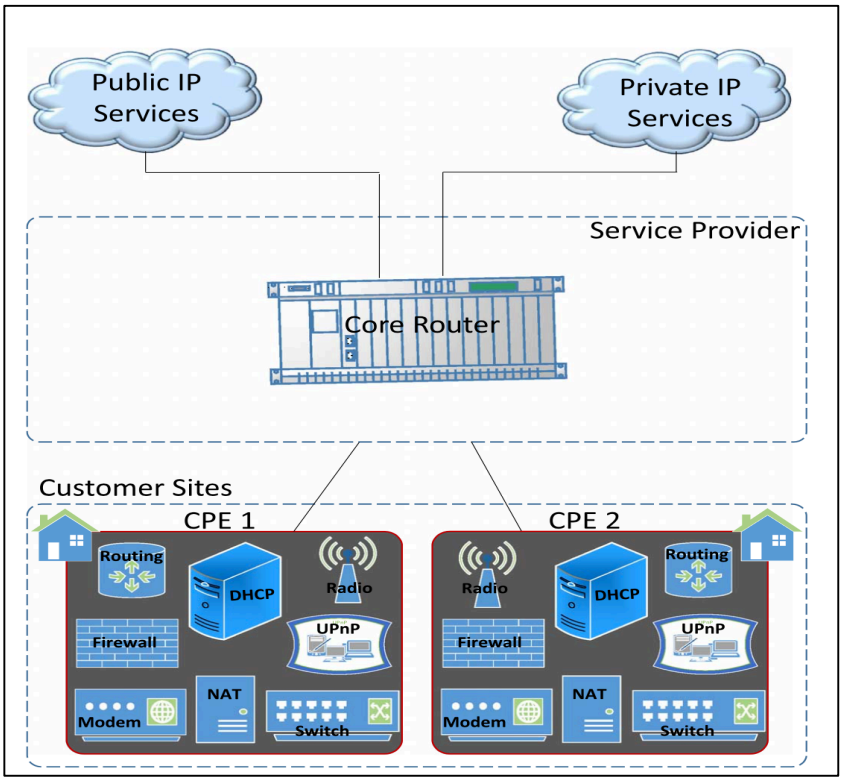

Figure 1.1 Traditional Network Services Architecture

Another challenge in front of the network service operators is the rapidly growing user requirements which push the companies to procure more equipment in order to fulfill those needs which further leads to increase in operational costs. Network Function Virtualization (NFV) seems to be the best solution to resolve these issues.

Network Virtualized Function is a concept to decouple the network services such as router, firewall, load balancer, session border controller etc. from proprietary hardware and deploy them in cloud computing environment on virtual instances [1]. In this approach, a service can be split up into various Virtual Network Functions(VNFs) and can be deployed on standard hardware.

NFV specifies a new architecture for the intended functions to be treated as completely independent logical entities of the physical layer. Although still in inception, NFV promises to bring revolution to the telecom sector as it paves the way for dynamic scaling. The figure [2] below illustrates the network architecture with NFV.

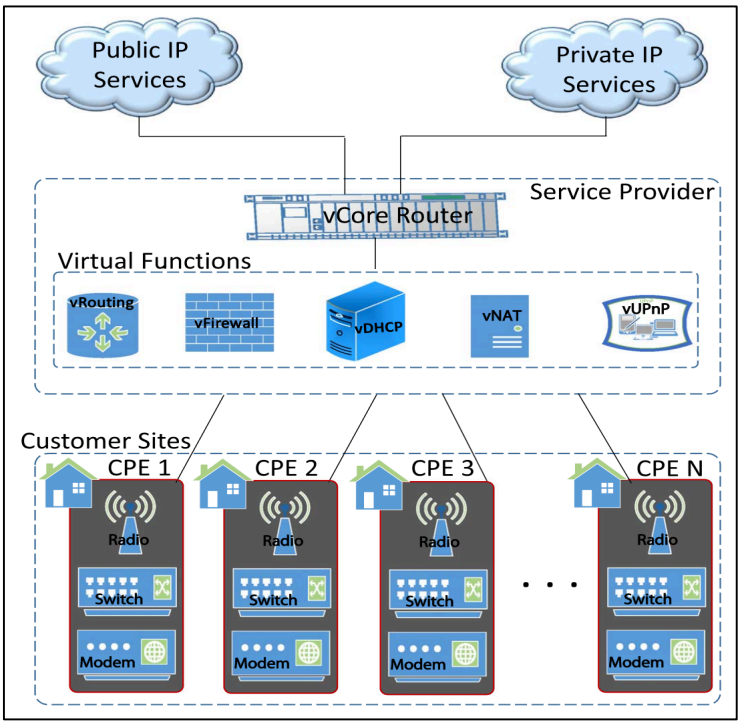

Figure 1.2 Network Services Architecture with NFV

1.2 Project Framework

This project is aimed at using OpenStack as the cloud computing platform to demonstrate NFV.

The report is organized as follows:

Chapter 2 gives in-depth information regarding OpenStack and Network Function Virtualization.

Chapter 3 presents the approaches taken in order to deploy NFV.

Chapter 4 describes the setup and experimentation for this project.

Chapter 5 contains the concluding remarks and provides directions for future work.

Chapter 2
OpenStack and NFV

2.1 OpenStack

OpenStack [3] is an open source cloud-computing software platform. OpenStack was developed by NASA and Rackspace [4]. OpenStack serves as Infrastructure as a service (IaaS) i.e., OpenStack hosts on-demand hardware, software, servers, storage, and networks. These resources can be adjusted on the fly hence it works great with varying amount of workloads. The figure [5] below gives a high-level view of OpenStack services.

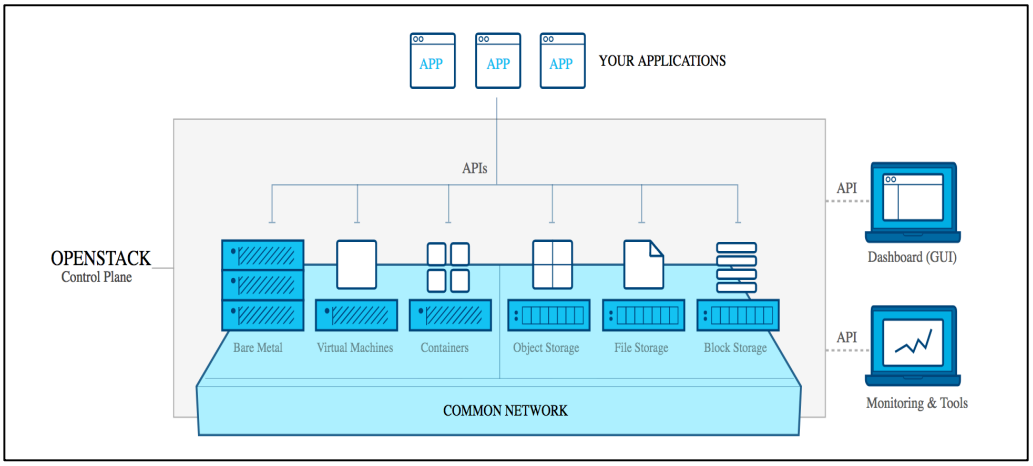

Figure 2.1 OpenStack High-Level View

OpenStack aids users to launch virtual machines. The virtual machines handle tasks assigned by the end-user. Spinning up more virtual instances can distribute the work as the workload increases. OpenStack relies on programming interfaces (Application Programming Interface, API) that hide hypervisors and specific details to applications and users.

2.2 OpenStack Components

OpenStack comprises of key components which are as follows:

Nova [6]: -

Nova is one of the most integral components of OpenStack, Nova serves as compute platform on which all the virtual machines run. Nova provides a control plane for hypervisor residing beneath it. Nova is compatible with all types of hypervisors; bare-metal, KVM, VMware etc. The Nova module is written in Python and uses SQLAlchemy for database access. [3]

Keystone: -

Keystone provides means of authentication and authorization in the OpenStack environment. Integration of LDAP and Keystone provide a directory for role-based access. Keystone supports both token-based and traditional username-password login. Different users can have different roles and access OpenStack services. Any request to Nova components navigate through Keystone to gain authentication.

Glance: -

Glance stores and provides virtual copies of disk and server images. These images can further be used as templates. Whenever a virtual instance is spun Nova talks to Glance and retrieves the specific image requested by the user.

Swift: -

Swift is the storage system for the OpenStack environment. Swift is responsible for data integrity and replication across the cluster. As the data on disks increase, Swift makes it possible for increasing the storage horizontally by adding the servers. Glance uses Swift to store the images. Files in Swift are stored as chunks or segment file and a manifest file helps to track all these files.

Cinder: -

Cinder is another storage component of OpenStack. The difference is that Swift is object storage and Cinder is block storage. Cinder can be thought of as an external hard drive and is slower when compared to Swift. Cinder sustains the data even after the virtual instance is terminated. Cinder stores backup images and snapshots.

Neutron: -

Neutron manages networks in the OpenStack environment. Neutron offers a wide array of configuration settings; Virtual LAN, flat networks. Additionally, it also manages IP addresses which can either be static or derived from DHCP. Neutron acts as Software Defined Network agent in OpenStack, and give users complete control over the network. Users can connect the virtual instances with the internet and with each other with help of Neutron.

Horizon: -

Horizon is the dashboard for OpenStack which provides graphical access to users to deploy and interact with the virtual instances. Horizon also gives users access to all networking, storage components of OpenStack. It also serves as the monitoring tool for the cloud environment and helps administrators troubleshoot the errors.

Heat (Orchestration): -

Heat serves as Orchestration program in the OpenStack environment. Heat provides orchestration engine which takes the human-readable templates and contains specifications to configure cloud resources. Heat extends its support to Amazon Web services template format. It's a useful tool as the scalability increases. The figure [7] below describes the functioning architecture of OpenStack.

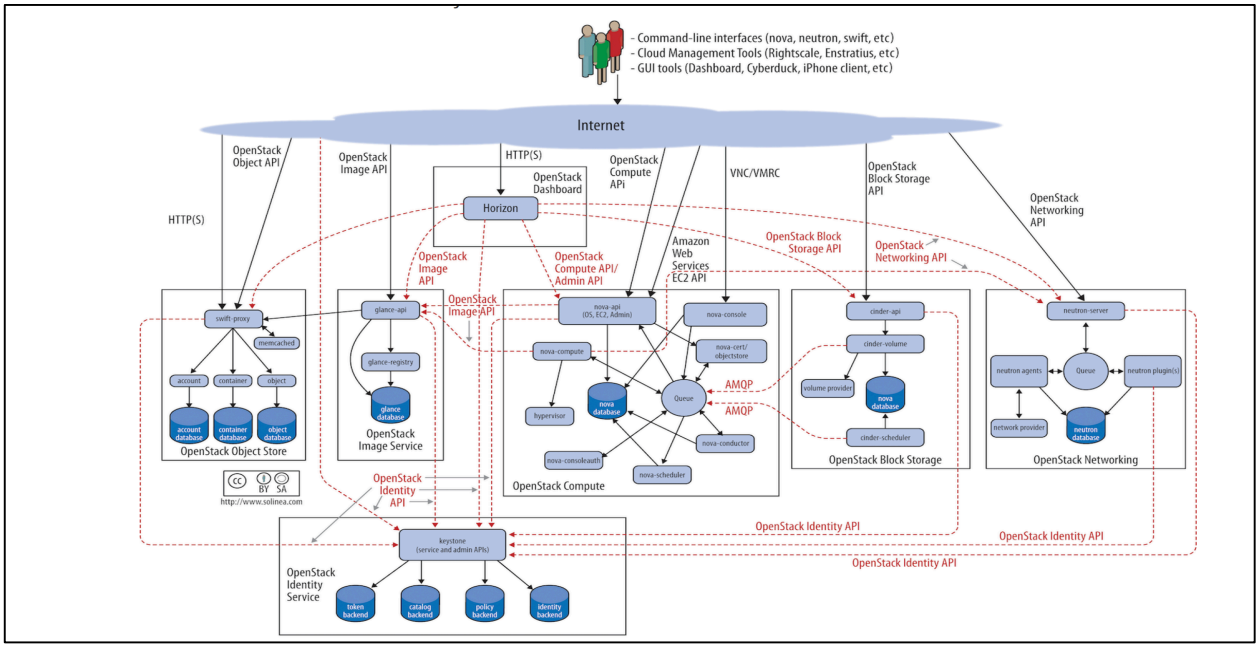

Figure 2.2 OpenStack Architecture

The traditional OpenStack has three nodes:

1) Controller

2) Compute

3) Storage

Controller node is the element responsible for the implementation of the identity service (Keystone), the Image Service (Glance). In particular, it must be able to perform some functions of networks (Neutron). It must also host SQL databases, the bus messages and the NTP (Network Time Protocol) to ensure synchronization between nodes.

A Compute node is responsible for running the hypervisors. Once initiated the virtual instances run on Compute node. It can also run some network functions (Neutron). Storage nodes, regardless of a storage node per block or object storage node, are optional. The figure below illustrates the many functions that exist on each component.

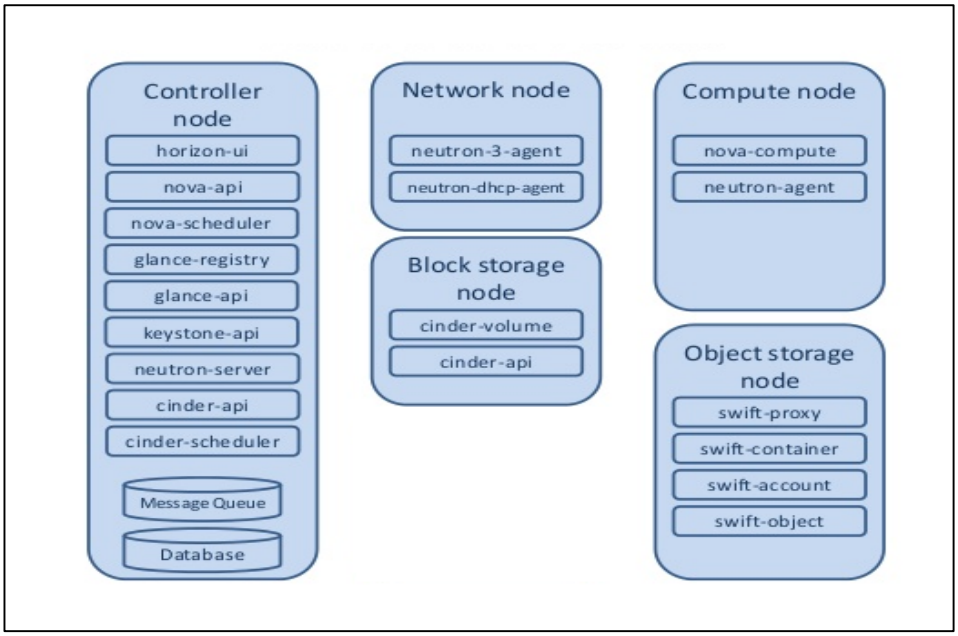

Figure 2.3 OpenStack Nodes.

2.3 What is NFV

The virtualization of network functions redefines the way to deploy proprietary network entities. NFV [1] specifies a new architecture for the intended functions to be treated as completely independent logical entities of the physical layer. This concept allows the execution of Virtual Network Function (VNF) on servers in data centers.

2.4 Use Cases of NFV:

Since this new concept was launched mainly by telecom operators, the group of interested people believed NFV deals only with functions such as telecom Subsystem IP Multimedia (IMS) or Evolved Packet Core (EPC). NFV still has more features such as Content Delivery Networks (CDN), which manages the content at the edge of the network.

Under the NFV umbrella the various use cases are:

- **VCPE (Virtual Customer Premise Equipment):**

VCPE [7] is virtualizing multiple functions like firewall, router, VPN, NAT, DHCP, IPS / IDS, PBX, etc. They are used to connect a single user to the Internet or set of agencies at headquarters. By virtualizing these functions, operators and enterprises can rapidly deploy new services and avoid all the manual processes.

- **VEPC (Virtual Evolved Packet Core):**

EPC is the heart of 4G networks. It is a packet network that provides all IP services including those that were previously provided by the field circuit such as voice. VEPC [8] allows operators to use a virtual infrastructure to accommodate voice and data. It can replace an infrastructure-based function attached directly to the physical layer.

- **VIMS (Virtual IP Multimedia Subsystem):**

IMS is a multi service, multi-access IP network. This concept represents a service-oriented network architecture. Virtualization IMS [9] offers operators the agility and scalability essential to remain competitive in the market.

2.5 NFV at ETSI

The institute "European Telecommunications Standards Institute" (ETSI) [10] is an organization consisting of several service providers. These providers work together to develop new solutions for their networks. In 2012 model for standardization of NFV was put forward by ETSI, the body responsible for outlining present and future of NFV. The researchers at ETSI took care of the standardization process. The result of the effort of 300 companies was the publication of several reference documents for companies that want to migrate to an NFV architecture. One of the documents deals with the reference architecture to follow. As shown in figure 2.4, the recommended architecture is composed of several components:

2.5.1 Components of NFV Architecture [11]

Network Function Virtualization Infrastructure (NFVI)

NFVI is the infrastructure on which the VNFs will be deployed. NFVI comprises of three parts:

- Physical resources: physical resources are computing capacity, storage, and network. They are shared by all VNFs.
- The virtualization layer: The virtualization layer is the hypervisor. This separates between physical resources and NFVI VNFs. Industry Specification Group (ISG) did not recommend a very special solution for this role.
- Virtual resources: The hypervisor provides an abstraction layer between the physical level and the virtual level. These virtual resources are available as independent entities for each VNF.

Virtual Network Function (VNF) is an application entity running on NFVI. VNF is a virtual version of traditional network functions. These virtual functions can be implemented on a single virtual machine or multiple machines for high availability.

Element Management System (EMS)

EMS is the entity's network management elements. This component ensures the configuration of these VNFs.

NFV Management and Orchestration (MANO)

MANO shows a distributed management layer and is composed of several components: Virtual Infrastructure Management (VIM) is the system of management and control of computing resources, storage, and network part of NFVI.

VNF Manager (VNFM):

This part of MANO is responsible for managing VNFs. EMS and VNFM work together via the vi-Vnfm interface to provide overall control of VNFs.

NFV Orchestration:

Orchestration NFV is the part responsible for lifecycle of network services.

Operations Support System / Business Support System (OSS / BSS)

OSS provides operators network management tasks. In return, BSS is business oriented since it is responsible for billing, invoicing and Customer Relationship Management (CRM).

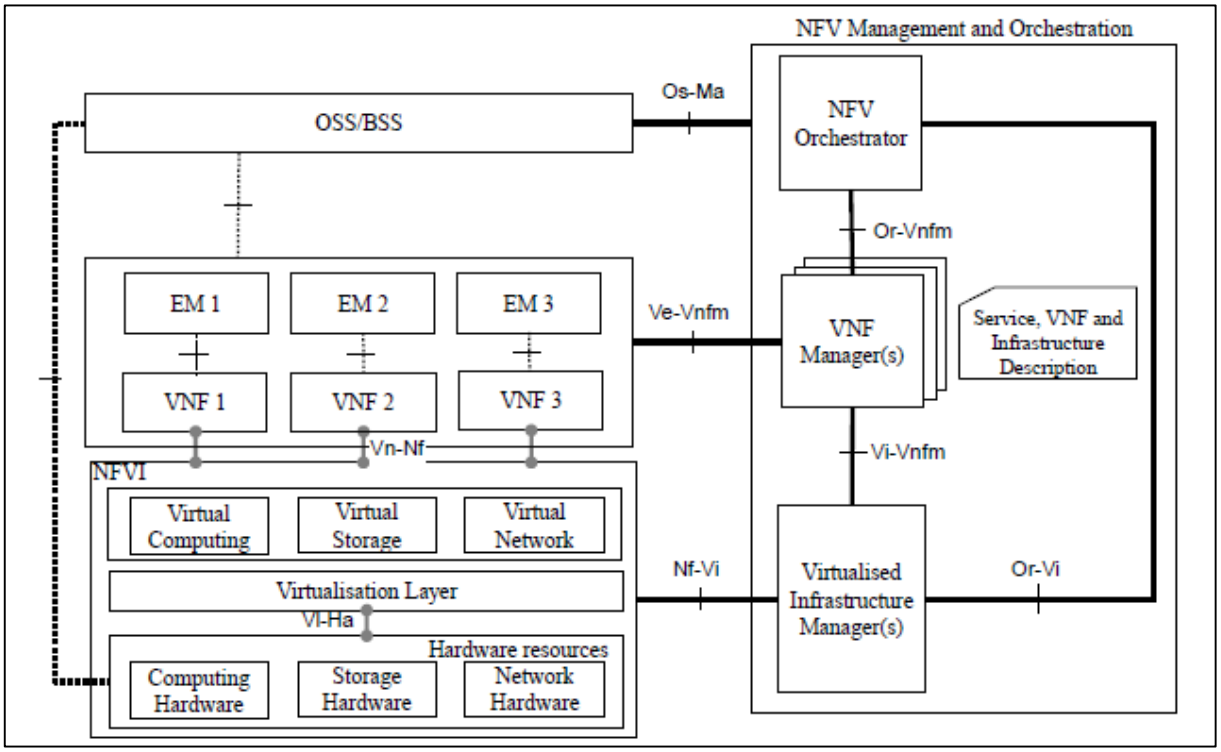

Figure 2.4 NFV Architecture

The figure displays the architecture of NFV proposed at ETSI [12]. To meet all recommendations of this architecture, companies must use a set of solutions as there is no single solution that can unite all these components.

Indeed, a wide range of open source products can meet the needs of every customer. OpenStack can serve as IAAS providing the functionality of Virtual Infrastructure Manager(VIM) and NFVI hardware components. Software Defined Networking tools such as OpenDayLight can be used to customize networking features.

One of the problems with integration of all these tools is the interoperability among them. Generally, open source projects face several bugs which make especially early unstable versions. Thus, the improved versions provide methods for testing the performance of platforms. But this is only a local test that does not extend over the entire NFV architecture. Hence, to combat these problems the idea of the new platform called OPNFV was born.

2.6 OPNFV

OPNFV (Open Platform for Network Function Virtualization) [13] is a project launched in September 2014 by the Linux Foundation. Mainly, OPNFV goals to be achieved are:

- Develop an Open Source test platform which can be used to deploy NFV functions
- Accelerate the deployment of other open source projects like OpenStack and Opendaylight [14].
- Contribute to and participate in various Open Source projects to be undertaken by OPNFV
- Establish an environment for NFV solutions based on open source standards
- Promote OPNFV as the reference platform for NFV

The OPNFV project is the collective work of 51 companies and more than 340 contributors. To ensure continued involvement of end users, the OPNFV community includes more than 30 members from several operators around the goal to populate its advisory group.

Chapter 3

Related Study and Approaches

3.1 Juniper Contrail:

Juniper Contrail [15] is a Software Defined Networking (SDN) platform. This platform is used to automate and orchestrate virtual network services. Contrail offers a lot of advantages:

- Abstraction of Network Rules and Policies.
- Segmentation of Network Tenants.
- Connectivity between Heterogeneous Environments.

The figure [16] below describes the higher level view of Juniper Contrail.

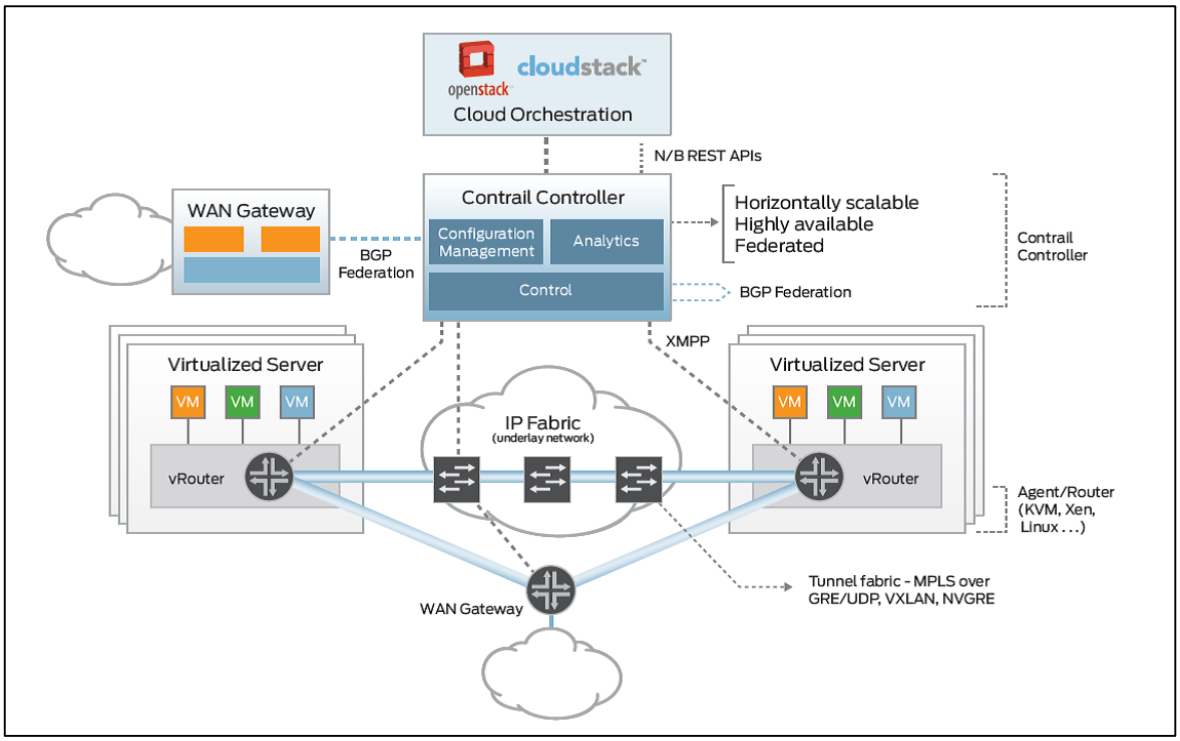

Figure 3.1 Juniper Contrail

The Juniper Contrail NFV framework solution includes:

- Programmability to get total control and visibility over the system with SDN as the integral component.
- Virtual Network Functions and Service Chaining are supported with vSRX and vMX platforms.
- Interactive Graphical User Interface, to simplify network, switching and security functions.

3.1.1 Problems faced with Juniper Contrail:

- Although Juniper Contrail is great networking platform, it is a proprietary product of Juniper Networks which brings a licensing fee in the equation in order to use Contrail.

- Contrail works in stand-alone mode and additionally on top of OpenStack Environment. Working with OpenStack environment brings incompatibility as OpenStack is open source in nature.

3.2 Tacker

Tacker [17] is an OpenStack incubator project which serves as Generic VNF manager(VNFM) and NFV Orchestrator to deploy and operate network functions and services. Tacker manages the lifecycle of Virtual Network Functions. Additionally, it plays other vitals roles like monitoring VNF, scaling the network functions and healing them if a network function goes into a bad state. Tacker is based on the blueprint derived from ETSI MANO which was discussed in the previous chapter. The figure [18] below represents the architecture of Tacker.

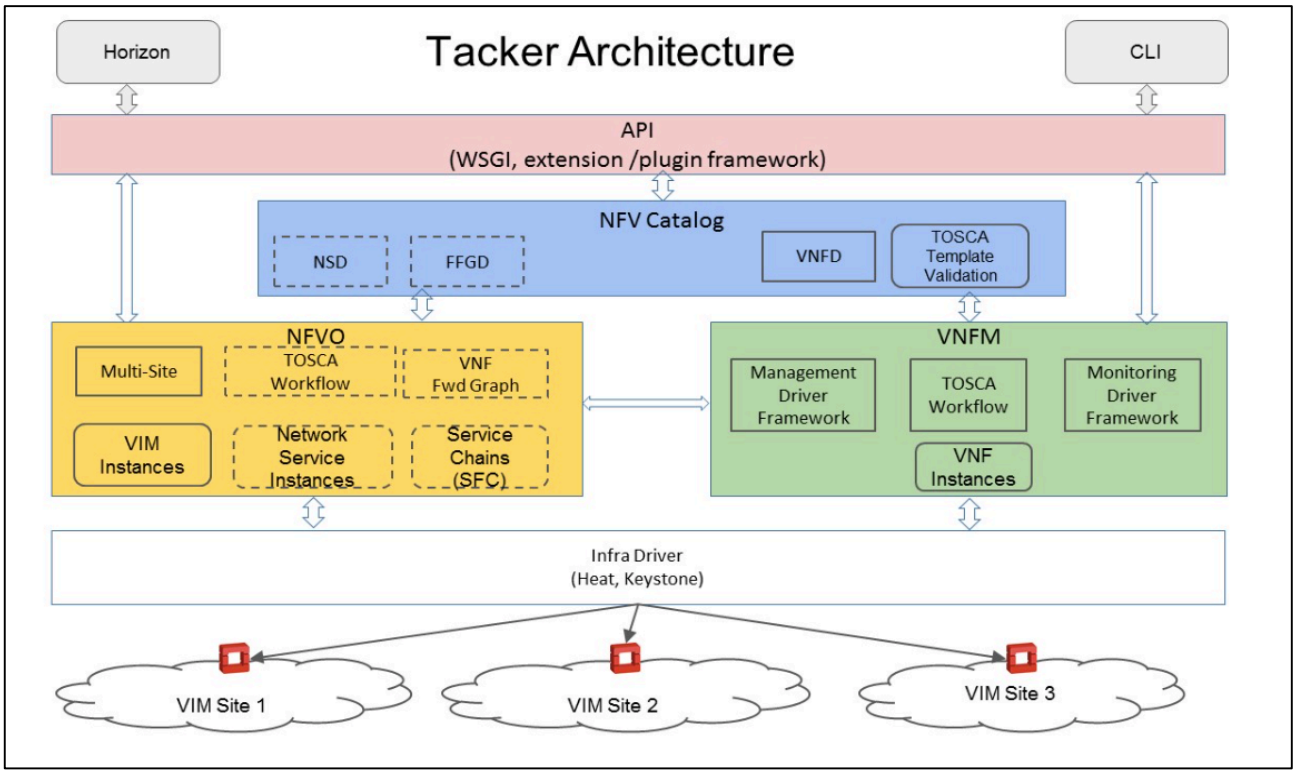

Figure 3.2 Tacker Architecture

3.2.1 Tacker Components:

There are 3 components of Tacker:

1) Network Function Virtualization Catalog:

NFV catalog is the collection of templates can be used to deploy various network services. The standard template format is TOSCA (Topology and Orchestration Specification for Cloud Applications) [19]. The templates that can be on-boarded in Tacker are:

i) Virtual Network Function catalog is a repository of Virtual Network Function Descriptor(VNFD) which in turn is a template file that specifies the deployment and behavior requirements of a VNF. The figure [20] below shows a sample VNFD template.

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: Demo with user-data

metadata:
  template_name: sample-vnfd-userdata

topology_template:
  node_templates:
    VDU1:
      type: tosca.nodes.nfv.VDU.Tacker
      capabilities:
        nfv_compute:
          properties:
            num_cpus: 1
            mem_size: 512 MB
            disk_size: 1 GB
      properties:
        image: cirros-0.3.5-x86_64-disk
        config: |
          param0: key1
          param1: key2
        mgmt_driver: noop
        user_data_format: RAW
        user_data: |
          #!/bin/sh
          echo "my hostname is `hostname`" > /tmp/hostname
          df -h > /home/openwrt/diskinfo

    CP1:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
        management: true
        order: 0
        anti_spoofing_protection: false
      requirements:
        - virtualLink:
            node: VL1
        - virtualBinding:
            node: VDU1

    VL1:
      type: tosca.nodes.nfv.VL
      properties:
        network_name: net_mgmt
        vendor: ACME
```

Figure 3.3 VNFD Sample

ii) Virtual Network Function Forwarding Graph(VNFFG) is a template on-boarded in Tacker. The VNFFG template describes the policy to manage network traffic through the deployed VNFs. This functionality is also known as Service Function Chaining(SFC). The figure [20] below displays a sample template for VNFFG.

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: Sample VNFFG template

topology_template:
  description: Sample VNFFG template

  node_templates:

    Forwarding_path1:
      type: tosca.nodes.nfv.FP.Tacker
      description: creates path (CP12->CP22)
      properties:
        id: 51
        policy:
          type: ACL
          criteria:
            - network_src_port_id: 640dfd77-c92b-45a3-b8fc-22712de480e1
            - destination_port_range: 80-1024
            - ip_proto: 6
            - ip_dst_prefix: 192.168.1.2/24
        path:
          - forwarder: VNFD1
            capability: CP12
          - forwarder: VNFD2
            capability: CP22

  groups:
    VNFFG1:
      type: tosca.groups.nfv.VNFFG
      description: HTTP to Corporate Net
      properties:
        vendor: tacker
        version: 1.0
        number_of_endpoints: 2
        dependent_virtual_link: [VL12,VL22]
        connection_point: [CP12,CP22]
        constituent_vnfs: [VNFD1,VNFD2]
      members: [Forwarding_path1]
```

Figure 3.4 VNFFG Sample

iii) Network Services Descriptor(NSD) is a template that is used to dynamically compose a complete network service. Using a single NSD, multiple VNFs can be created. The figure [20] below represents sample NSD template.

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: VNFDs in import section should be already on-boarded
imports:
    - sample-tosca-vnfd1
    - sample-tosca-vnfd2

topology_template:
  inputs:
    vl1_name:
      type: string
      description: name of VL1 virtuallink
      default: net_mgmt
    vl2_name:
      type: string
      description: name of VL2 virtuallink
      default: net0
  node_templates:
      VNF1:
        type: tosca.nodes.nfv.VNF1
        requirements:
          - virtualLink1: VL1
          - virtualLink2: VL2

      VNF2:
        type: tosca.nodes.nfv.VNF2

      VL1:
        type: tosca.nodes.nfv.VL
        properties:
          network_name: {get_input: vl1_name}
          vendor: tacker

      VL2:
        type: tosca.nodes.nfv.VL
        properties:
          network_name: {get_input: vl2_name}
          vendor: tacker
```

Figure 3.5 NSD Sample

2) Virtual Network Function Manager: -

The role of Virtual Network Function Manager (VNFM) is to manage the life cycle of VNFs. A VNF can be deployed with the help of VNFM with the catalog provided by VNF catalog. Additionally, VNFM monitors the health of deployed VNFs. Tacker works on ETSI MANO architecture and aims at coordinating the components shown in the red box in the figure [21] below.

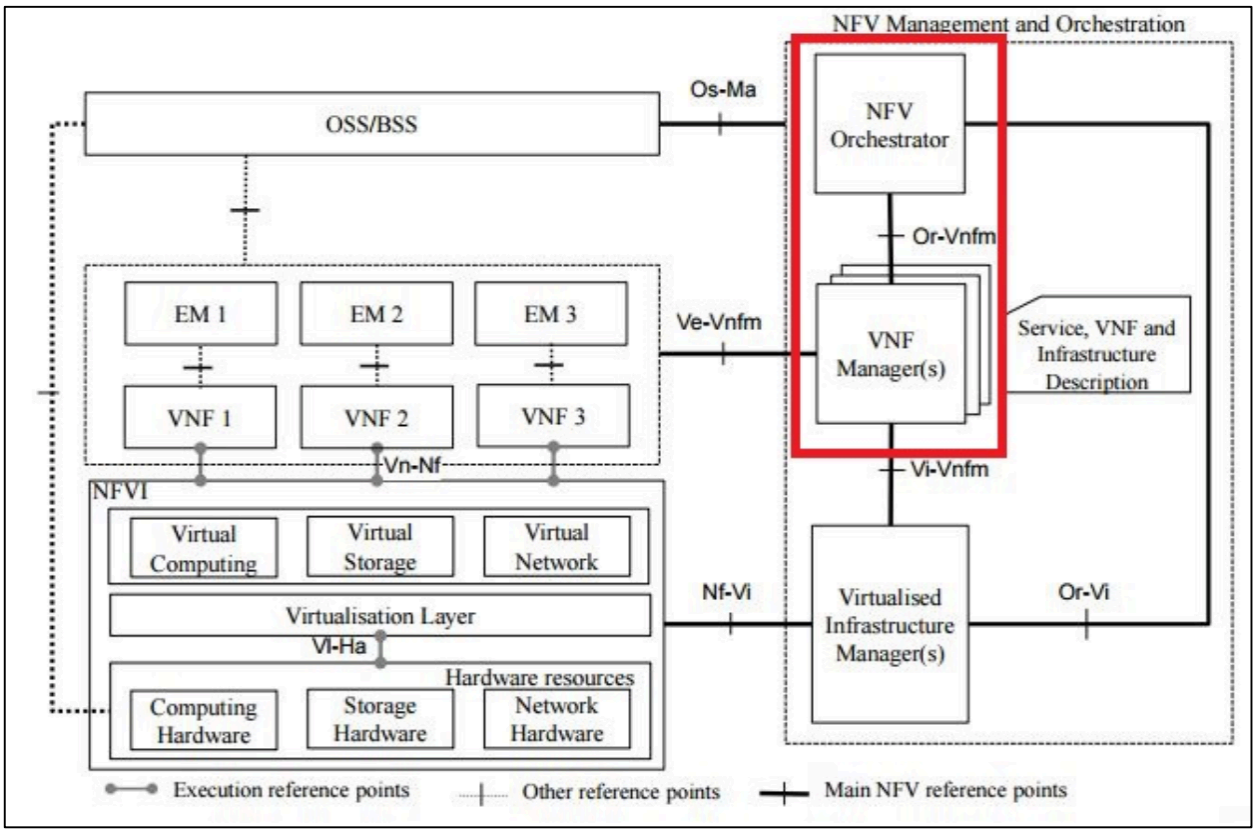

Figure 3.6 Tacker in NFV Architecture

3) Network Function Virtualization Orchestrator:

NFVO helps provide efficient placement of VNFs. NFVO helps connect individual VNFs using Service Function Channing using a VNF forwarding Graph Descriptor. In a Service function changing model decomposed VNFs are joined together to provide end-to-end network solution. The figure [22] below represents the functioning elements of tacker and how they interact with OpenStack.

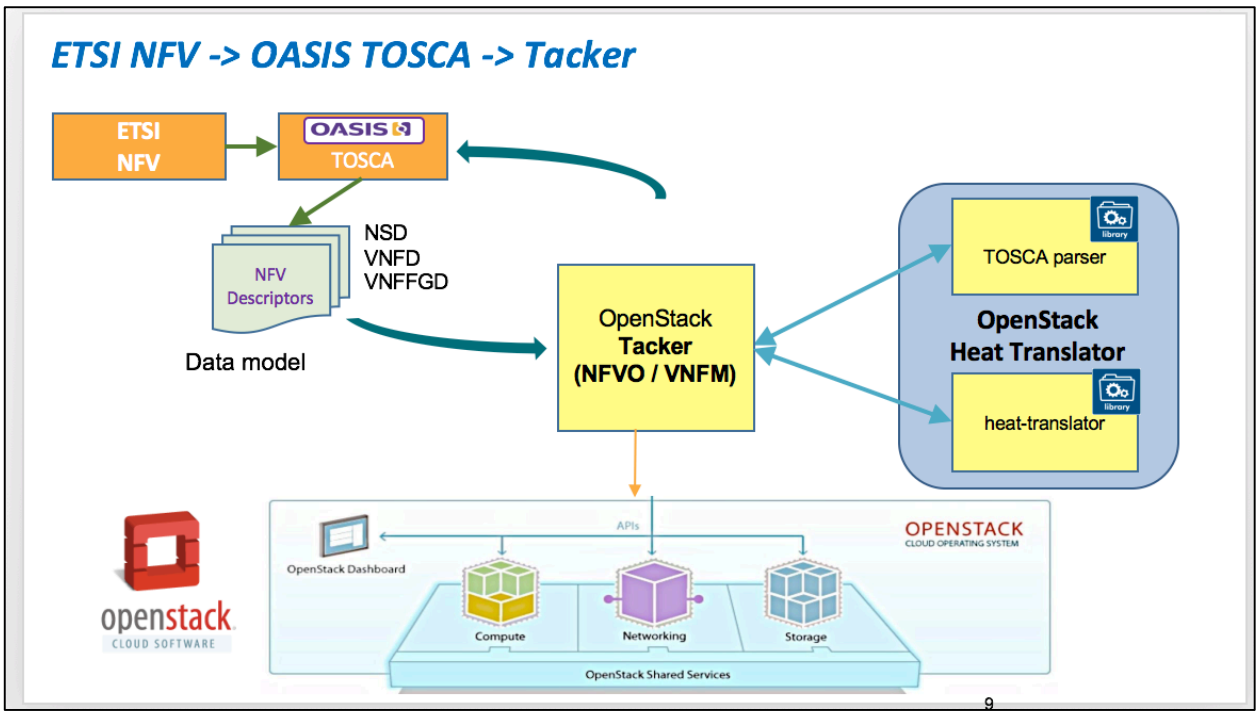

Figure 3.7 Functioning elements in Tacker

Chapter 4

Setup & Experimentation

4.1 Setup

In this chapter, we will write all the steps necessary to create an end-to-end virtual network service for a single as well as multiple tenants. The first embodiment is the composition of a cloud architecture with three physical nodes using OpenStack and the second part is deploying Network Function Virtualization on top of OpenStack.

Creating a setup for NFV Orchestration has three elements:

- Hardware setup
- OpenStack Setup
- NFV Setup

4.1.1 Hardware setup:

Depending upon the requirements of the project the hardware is setup. The bare minimum requirements are

Controller Node: 1 processor, 4 GB memory, and 5 GB storage

Compute Node: 1 processor, 2 GB memory, and 10 GB storage

These requirements are capable of providing proof-of-concept requirements for CirrOS virtual instances. As the requirements increase the hardware needs to be upgraded to support new environment. The figure [23] below represents hardware requirements for the traditional OpenStack environment.

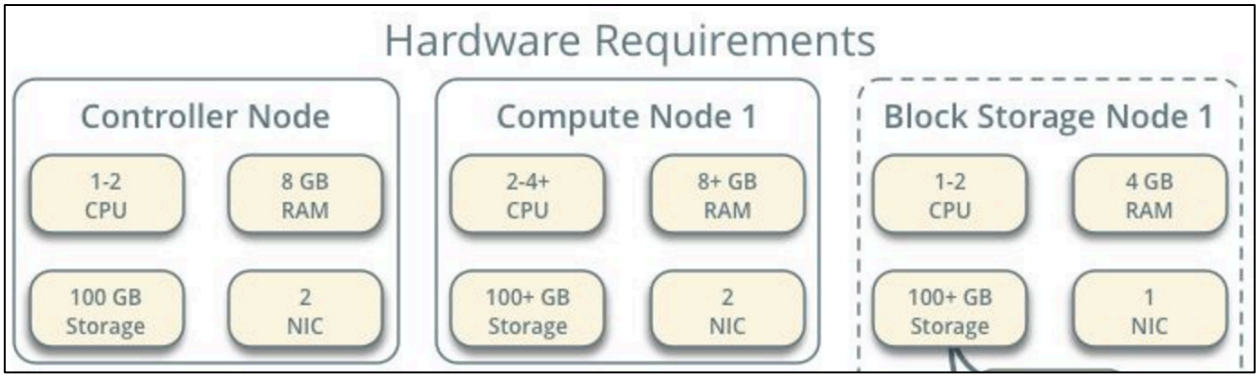

Figure 4.1 Hardware requirements for OpenStack

We start with setting up 3 nodes:

Controller node- 2 processors, 8 GB of memory and 500 GB storage

Compute node- 2 processors, 8 GB of memory and 500 GB storage

Storage node- 2 processors, 8 GB of memory and 500 GB storage

It is important to ensure that the processor in use supports hardware virtualization so that the virtual instances can use virtual processors.

4.1.2 OpenStack setup:

The OpenStack setup comes with two configurations:

i. Devstack

ii. Manual

Devstack is a bunch of extensible scripts to create a small OpenStack environment. Devstack is used as a development environment and tweaks can be made to configuration files. This serves as the purpose for functional testing.

Manual OpenStack setup is a long process which requires setting up the controller, compute and storage node individually. The components discussed in Chapter 2 are deployed individually. The figure below represents the services working on the controller node after successful installation of the OpenStack environment.

```
third@controller:~$ openstack service list
+----------------------------------+-------------+-------------------+
| ID                               | Name        | Type              |
+----------------------------------+-------------+-------------------+
| 10ca22f17f80410d9a3542703ee5cc7c | cinderv3    | volumev3          |
| 1427ba293efc4eff84fe6a6d7f75f4cc | heat        | orchestration     |
| 15bf45fb5a4d4d429f3b6beade07757b | glance      | image             |
| 19be63454ef24da7a634da3838f0c57c | placement   | placement         |
| 6cc386c7a74e4aeca2cd003a6825faf4 | cinderv2    | volumev2          |
| 8ad2e2353f3f4d80a0216d746c1d4638 | heat-cfn    | cloudformation    |
| 8c72d43a21154308bd22f87f3010b5c5 | cinder      | volume            |
| a0545e1bc5724e60bc24ba1cf24aa324 | neutron     | network           |
| acd7f5238e8c436a844e8d626b3728e0 | tacker      | nfv-orchestration |
| b56a95c0d0ec4732b03444e95af6f47b | nova        | compute           |
| ecbfd37f52e4401784b56a8460906f4c | nova_legacy | compute_legacy    |
| fc5fd42c3ffd42fda1a6d2a817125ce4 | keystone    | identity          |
+----------------------------------+-------------+-------------------+
```

Figure 4.2 OpenStack Command Line Services

The figure below represents the Horizon, which is the Graphical User Interface(GUI) for the OpenStack environment. Horizon provides lot of control options to the users in order to launch instances, add images and managing projects. The figure below shows the OpenStack dashboard after complete setup.

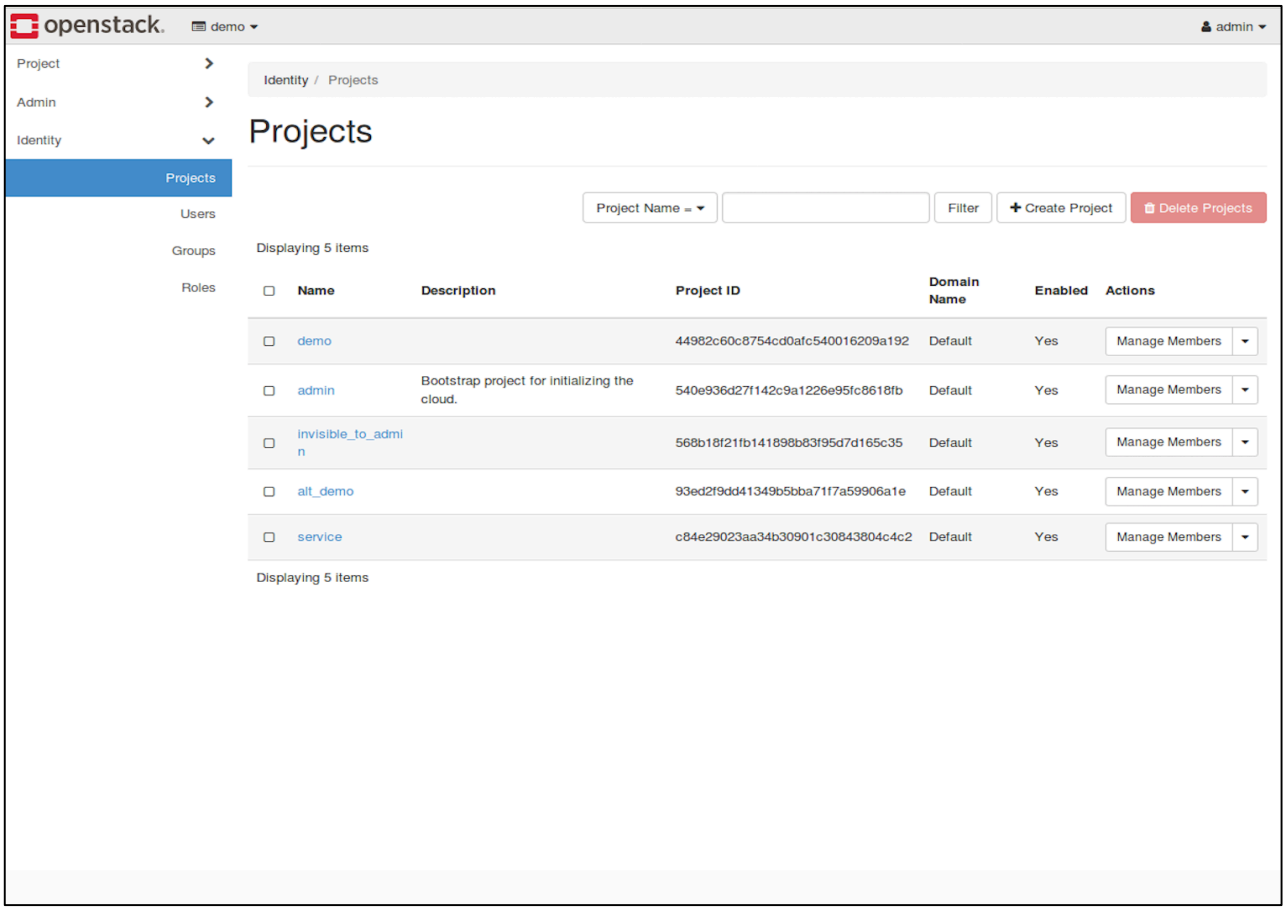

Figure 4.3 OpenStack Dashboard

4.1.3 NFV setup:

Once the OpenStack is setup, the next step is to setup NFV on top of the OpenStack environment. There are various packages that need to be installed to deploy NFV. The NFV component comes with the command line tools as well as a separate entity in Horizon. The next figure shows the commands available with Tacker.

```
vim-delete              Delete given VIM(s).
vim-events-list         List events of VIMs.
vim-list                List VIMs that belong to a given tenant.
vim-register            Create a VIM.
vim-show                Show information of a given VIM.
vim-update              Update a given VIM.
vnf-create              Create a VNF.
vnf-delete              Delete given VNF(s).
vnf-events-list         List events of VNFs.
vnf-list                List VNF that belong to a given tenant.
vnf-resource-list       List resources of a VNF like VDU, CP, etc.
vnf-scale               Scale a VNF.
vnf-show                Show information of a given VNF.
vnf-update              Update a given VNF.
vnfd-create             Create a VNFD.
vnfd-delete             Delete given VNFD(s).
vnfd-events-list        List events of VNFDs.
vnfd-list               List VNFD that belong to a given tenant.
vnfd-show               Show information of a given VNFD.
vnfd-template-show      Show template of a given VNFD.
vnffg-create            Create a VNFFG.
vnffg-delete            Delete a given VNFFG.
vnffg-list              List VNFFGs that belong to a given tenant.
vnffg-show              Show information of a given VNFFG.
vnffg-update            Update a given VNFFG.
vnffgd-create           Create a VNFFGD.
vnffgd-delete           Delete a given VNFFGD.
vnffgd-list             List VNFFGDs that belong to a given tenant.
vnffgd-show             Show information of a given VNFFGD.
vnffgd-template-show    Show template of a given VNFFGD.
```

Figure 4.4 Tacker Command line

The figure below shows Horizon with NFV added to it.

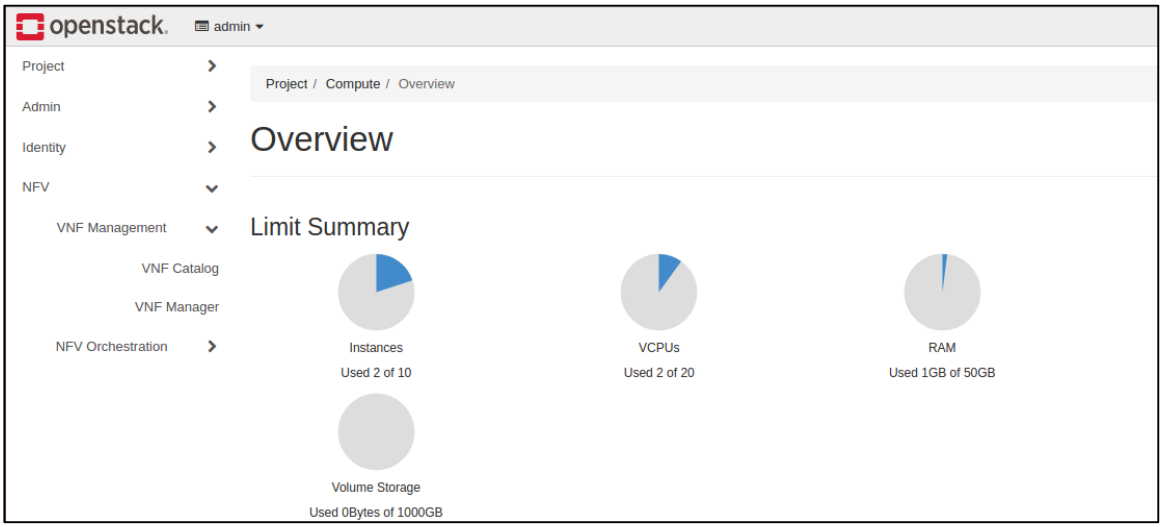

Figure 4.5 OpenStack Dashboard with NFV

4.2 Challenges faced in the setup

There were various challenges faced during the complete setup. OpenStack is an open source platform, hence there are a lot of bugs which need to be handled before the system is completely up and running. The ambiguity of the error messages makes it harder to pinpoint a problem and then troubleshoot it.

4.3 Experimentation

4.3.1 Experiment 1

OpenWRT [24] is an open source project to bring routing functionality to Linux systems. Traditionally, most of the routers come with firmware installed on the proprietary hardware.

This approach outsources the routing control to the company that manufactures the hardware. OpenWRT aims at providing the routing firmware and putting the routing control in user's hands. OpenWRT currently runs on some hardware systems and is Linux compatible.

The OpenWRT image is obtained from the official website and is added to Glance which handles the virtual copies for OpenStack.

```
third@controller:~$ openstack image create OpenWRT --disk-format qcow2 --container-format bare
 --file /openstack/images/openwrt-x86-kvm_guest-combined-ext4.img \
> 
```

Figure 4.6 Adding OpenWRT image

The openness of OpenWRT brings a lot of applications on board, such as:

- Traffic-Shaping and QOS
- Capture and Analyze Network Traffic

4.3.1.1 Creating Subnets in OpenStack:

The next step in the experimentation is to create subnets

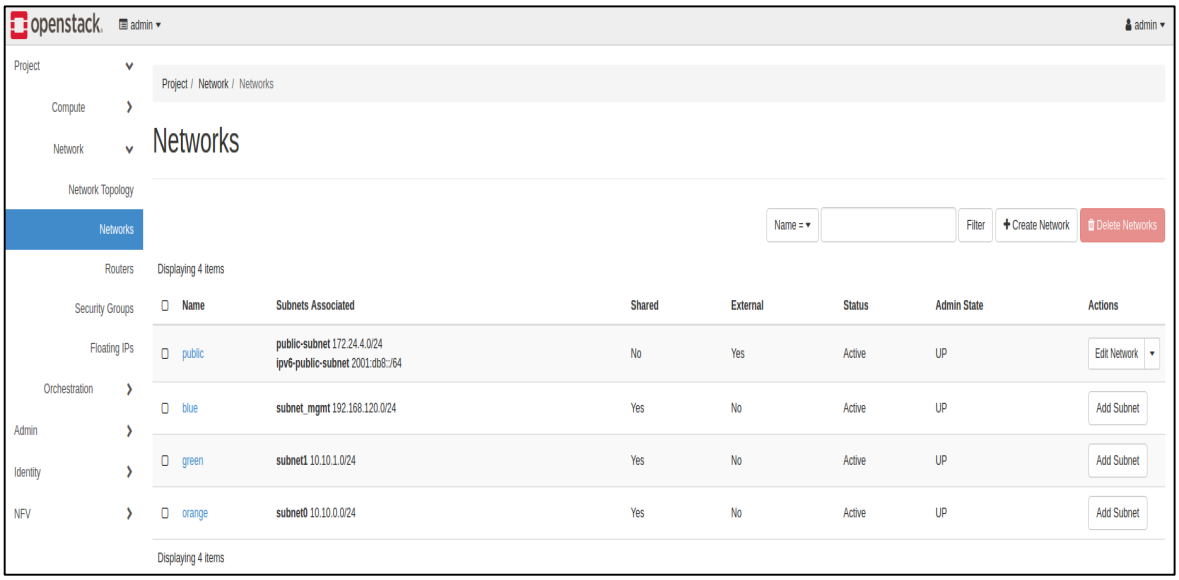

Figure 4.7 Creating Multiple Subnets in OpenStack

4.3.1.2 Creating a VNF catalog:

Deploying a VNF requires a VNFD. The VNFD template consists of the configuration required to setup a VNF. We provide the following VNFD to setup VNF with OpenWRT.

The VNFD has 3 sections:

- Virtual Deployment Unit (VDU): VDU highlights Compute instance, Image to be used, flavor to be deployed. VDU also incorporates the health management configuration for a VNF. The setup of the experiment included setting up an OpenWRT VNF.

The VDU properties for VNF were 512 MB of memory and 1 GB of storage. The image name is specified as OpenWRT. Other VDU configuration properties can be seen in the figure below.

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: OpenWRT with services

metadata:
 template_name: OpenWRT

topology_template:
 node_templates:

  VDU1:
   type: tosca.nodes.nfv.VDU.Tacker
   capabilities:
    nfv_compute:
     properties:
      num_cpus: 1
      mem_size: 512 MB
      disk_size: 1 GB
   properties:
    image: OpenWRT
    config: |
     param0: key1
     param1: key2
    mgmt_driver: openwrt
    monitoring_policy:
     name: ping
     parameters:
      count: 3
      interval: 10
     actions:
      failure: respawn
```

Figure 4.8 VDU Configuration in VNFD

- Virtual Link(VL): The Virtual Link describes the link that would be setup with the VNF. The network name is described in this section. This setup consists of three subnets blue, green and orange which can be seen in the figure below.

```
VL1:
  type: tosca.nodes.nfv.VL
  properties:
    network_name: green
    vendor: Tacker

VL2:
  type: tosca.nodes.nfv.VL
  properties:
    network_name: blue
    vendor: Tacker

VL3:
  type: tosca.nodes.nfv.VL
  properties:
    network_name: orange
    vendor: Tacker
```

Figure 4.9 VL Configuration in VNFD

- Connection Point(CP): this is the section where virtual links and VDU are attached together. We can have multiple connection points in a template. For this setup, we have 3 connection points which bind three virtual links to VDU.

```
CP1:
  type: tosca.nodes.nfv.CP.Tacker
  properties:
    management: true
    order: 0
    anti_spoofing_protection: false
  requirements:
    - virtualLink:
        node: VL1
    - virtualBinding:
        node: VDU1

CP2:
  type: tosca.nodes.nfv.CP.Tacker
  properties:
    order: 1
    anti_spoofing_protection: false
  requirements:
    - virtualLink:
        node: VL2
    - virtualBinding:
        node: VDU1

CP3:
  type: tosca.nodes.nfv.CP.Tacker
  properties:
    order: 2
    anti_spoofing_protection: false
  requirements:
    - virtualLink:
        node: VL3
    - virtualBinding:
        node: VDU1
```

Figure 4.10 CP configuration in VNFD

4.3.1.3 Deploying the VNF

The next part of this experimentation is deploying the VNF based on the on-boarded VNF template discussed above. When deploying a VNF we need to provide Virtual Infrastructure Manager (VIM). OpenStack serves as VIM for Tacker. The figure below describes step to deploy VNF.

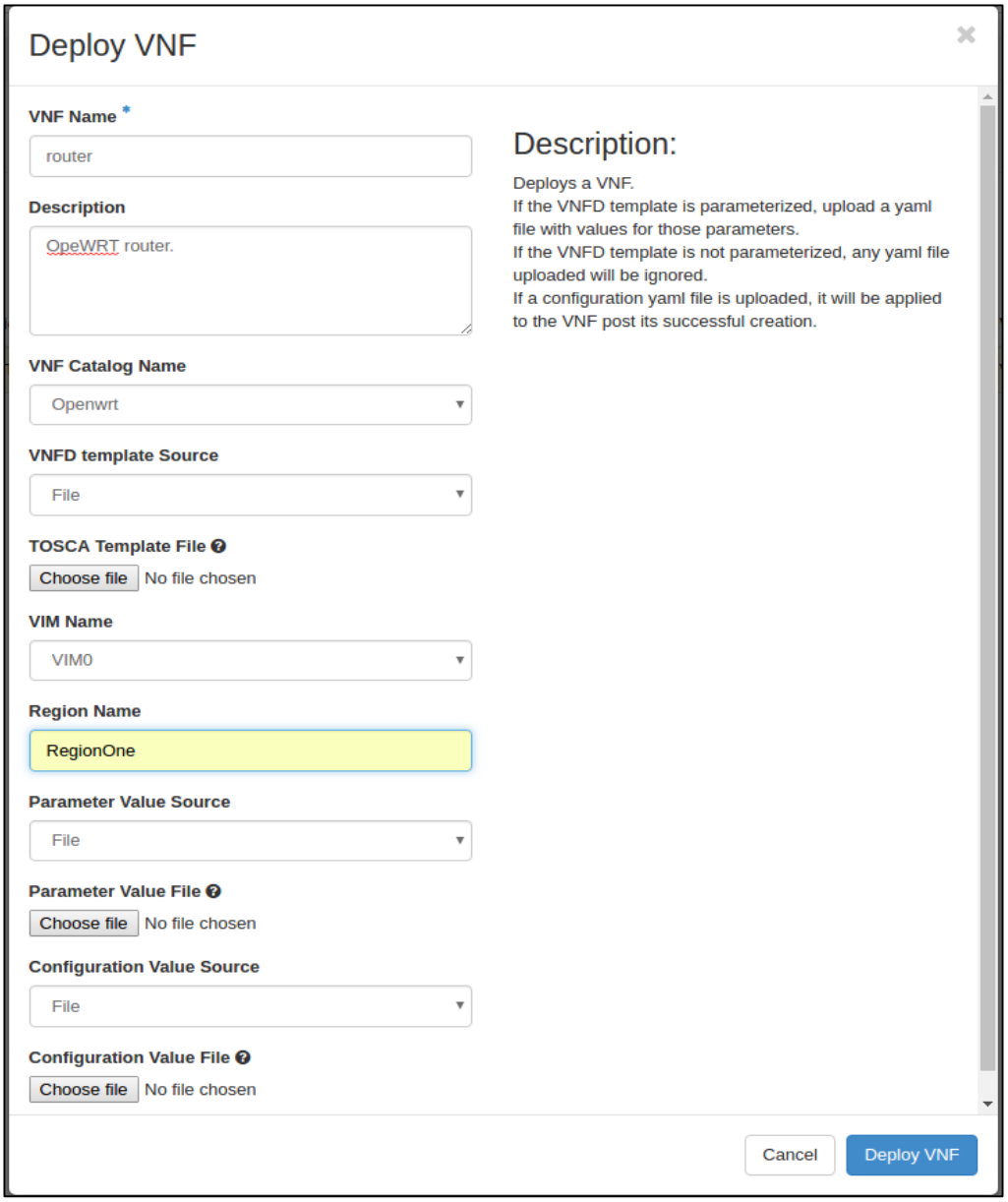

Figure 4.11 Deploying VNF

Once the VNF is deployed, the TOSCA template provided at the time of deploying VNF is transcribed into Heat Orchestration Template(HOT). As described earlier in Chapter 2, HEAT is the Orchestration engine for OpenStack. HEAT deploys the virtual instance with the transcribed template.

The figure below shows the overview of virtual instance launched. The instance launched gets three neutron ports and one compute node.

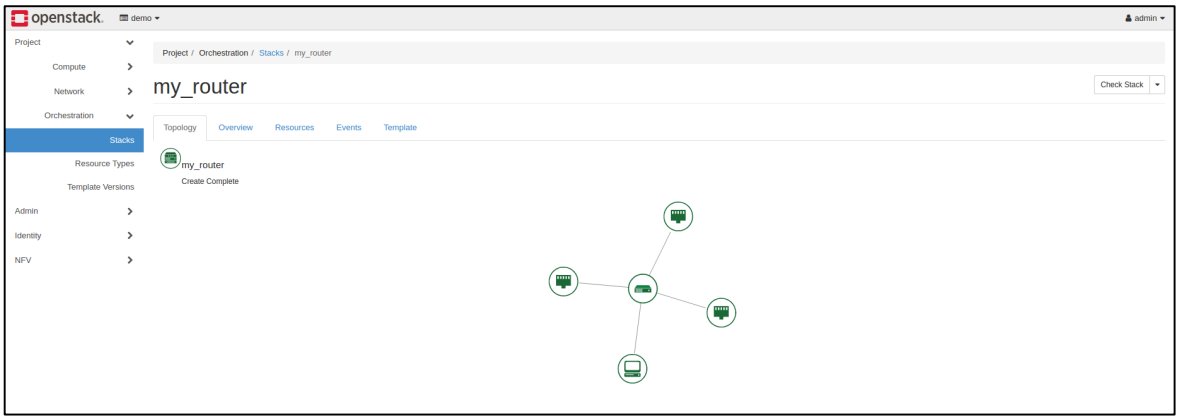

Figure 4.12 Overview of virtual OpenWRT instance

Apart from the OpenWRT instance, we launched three virtual instances explicitly on different subnets in order to create an end-to-end network. Once the VNF is deployed it comes up with three interfaces and is assigned IP address from all three subnets.

The figure below illustrates the instances running on compute node.

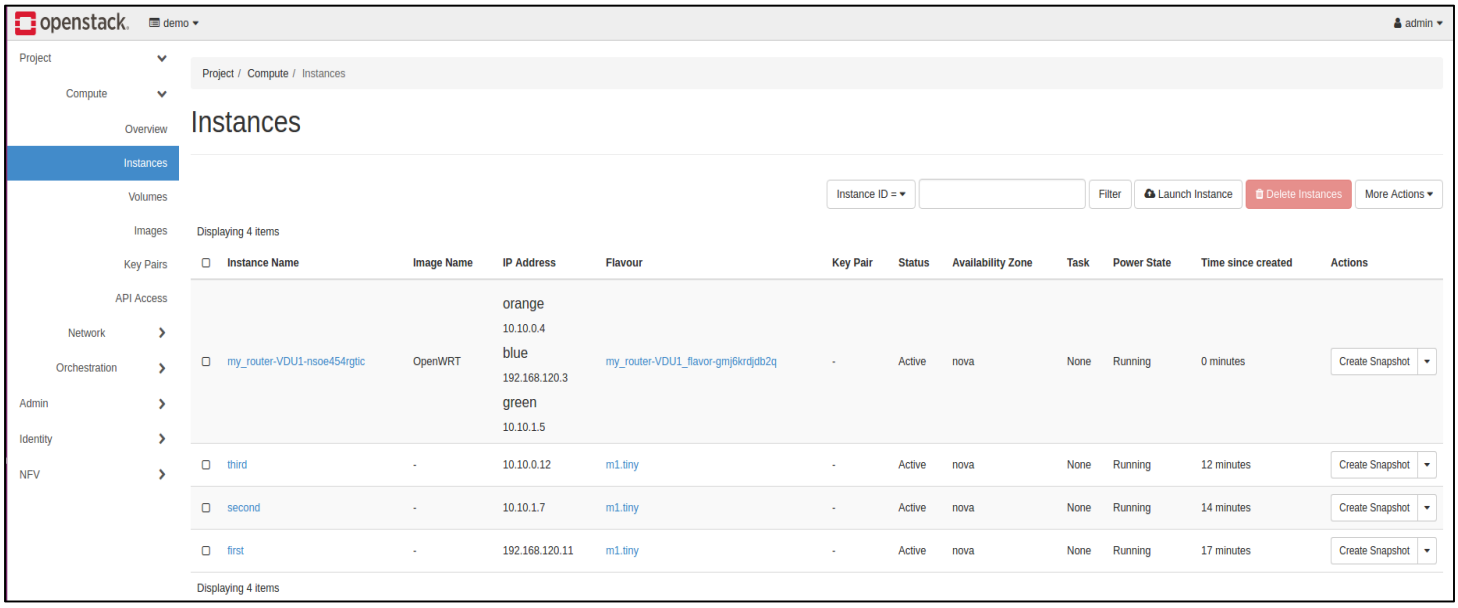

Figure 4.13 VNF instance

The figure below describes shows the network topology, the instance in the middle being common to all subnets is the OpenWRT instance. Hence we successfully carried out the first experiment.

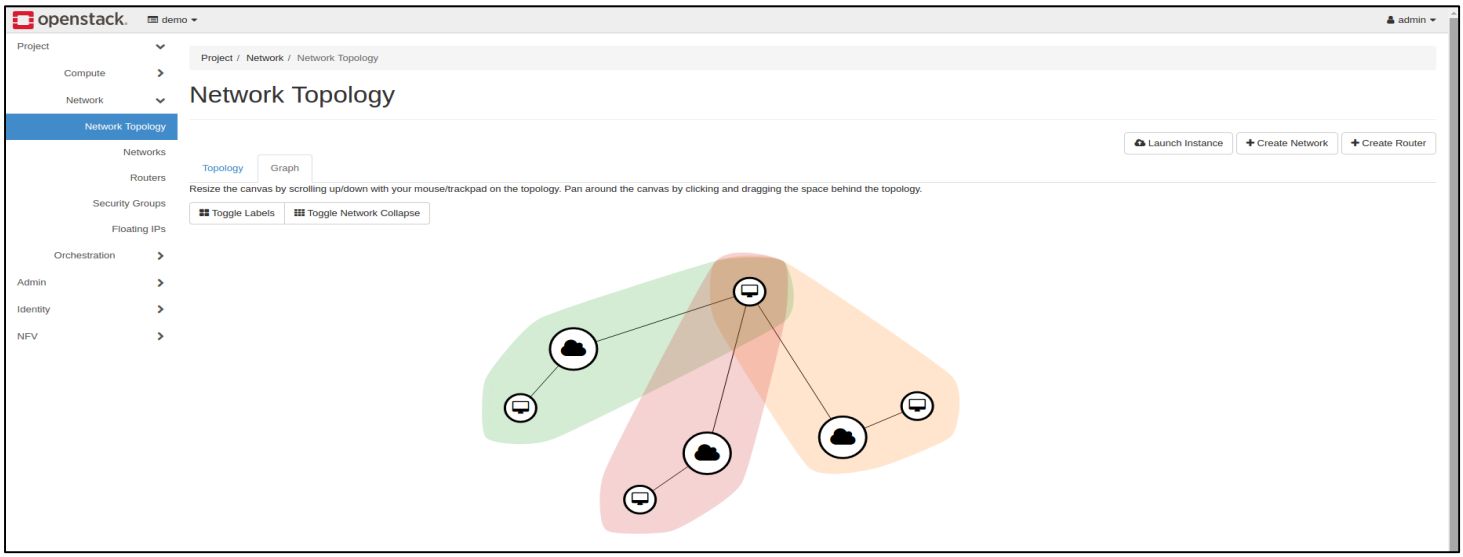

Figure 4.14 Network Topology VNF

4.3.1.4 Health monitoring of VNF

Tacker not only just launches Virtual Network Functions but also does health monitoring of those VNFs. The different status displayed with the related state of a VNF are discussed below.

- PENDING_CREATE: When the VNF is launched, the status of the VNF changes to PENDING_CREATE. This implies that Tacker has launched the VNF and at this state the Topology & Orchestration Standard for Cloud Application (TOSCA) template is being translated into Heat Orchestration Template (HOT).
- ACTIVE: This status appears when the virtual instance is accepted by Orchestration engine(HEAT).
- DEAD: This status appears when Tacker is unable to reach VNF for some reason.

The figure below describes the status for different VNFs.

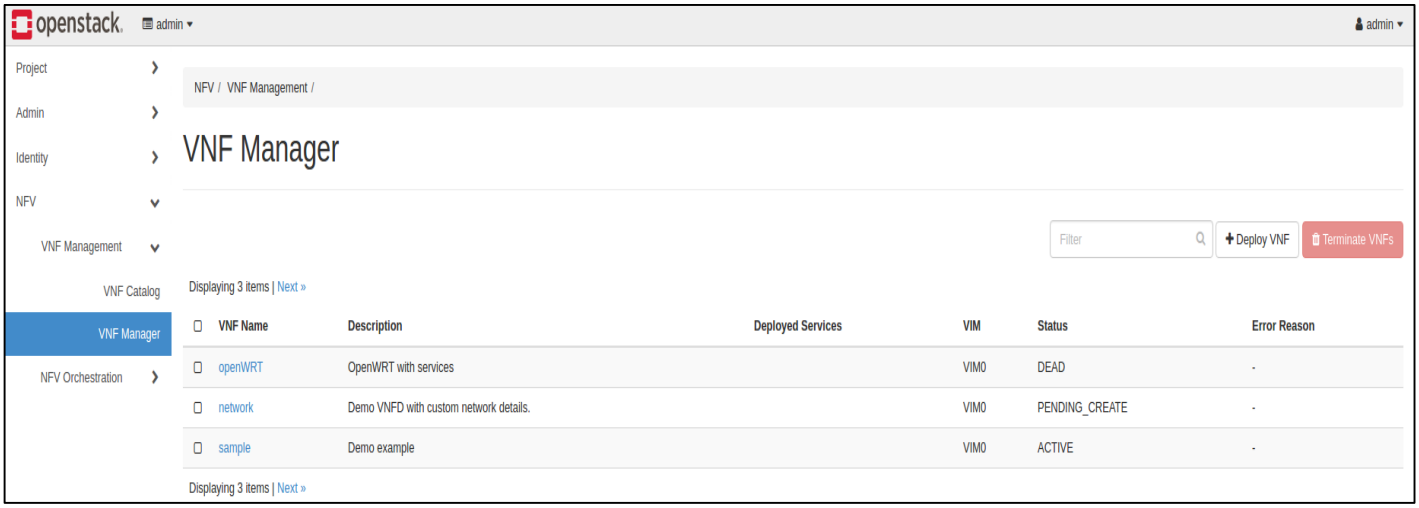

Figure 4.15 Health Monitoring of VNF

4.3.2 Experiment 2

The above experiment is carried out with single tenant, so for the next experiment we setup two tenants to carry out the procedure. The templates and configurations are updated accordingly. The figure below shows the multiple instances running for experiment 2.

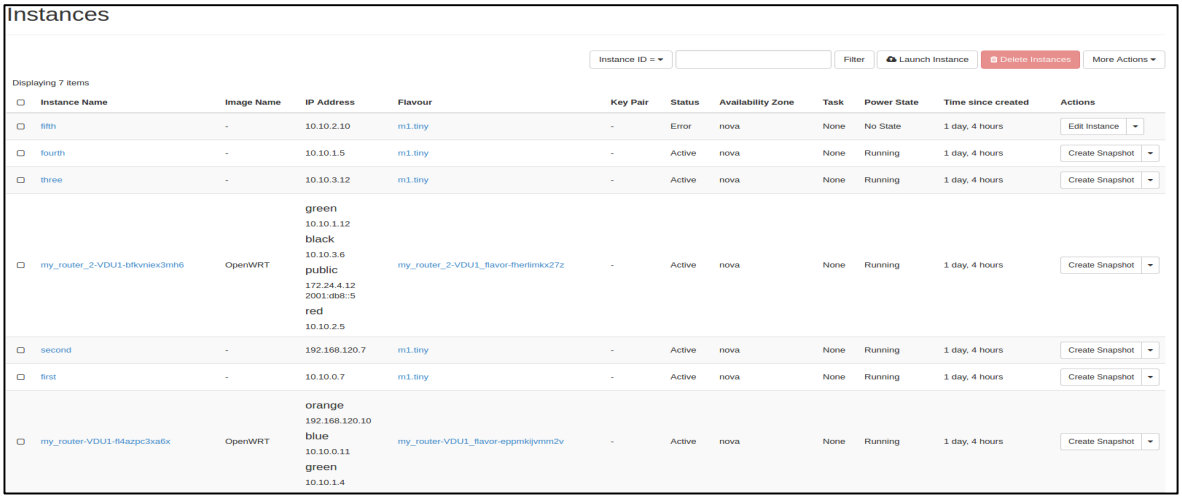

Figure 4.16 Multiple VNF instances

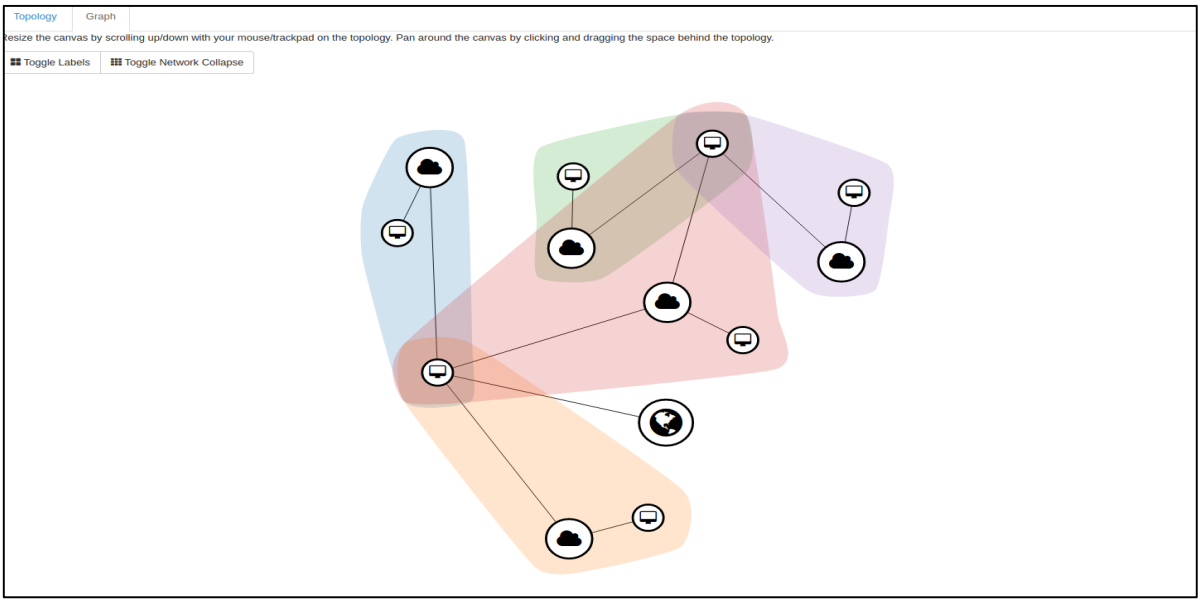

Figure 4.17 Network Topology Multiple VNFs

Chapter 5

Conclusion and Future work

This chapter summarizes the report by presenting the contributions and shedding light on future work.

5.1 Summary

This project provided a study and experimentation on NFV Orchestration on OpenStack. The first chapter focussed on providing introduction to cloud computing and background of Network Function Virtualization. In chapter 2 we talked about cloud computing platform OpenStack, it's components, architecture, on which the NFV was deployed. Additionally, we discussed in-depth about NFV, its architecture, standardization and as well as about its components. The use case discussed further provided enough evidence to establish the importance of NFV and its inevitable use in forthcoming future. Chapter 3 was about discussing the approaches taken to achieve the goal of this project. We looked into Juniper Contrail as a platform for deploying Virtual Network Functions, its architecture and problems. Next we discussed about Tacker, which is an incubator project of OpenStack. We studied in-depth about Tacker, and how it fits in the blueprint of NFV proposed by ETSI. The samples of templates to deploy VNF were discussed. Components and architecture of Tacker were also discussed in detail. Chapter 4 was about setting the environment and carrying out experiments on that platform. We considered the challenges faced during the setup.

The experiments conducted were also discussed in detail in chapter 4, we were able to setup an end-to-end network service with virtual instances.

5.2 Future work

This section outlines the scope of future work. The concept of Network Function Virtualization is fairly new, although there are no doubts that it will be a popular concept in coming years. Tacker was launched in 2015 and is still in its infancy. Thus, stability becomes a big issue for a new project. The next milestone for this project will be to achieve service function chaining, in order to connect the Virtual Network Functions in a single chain.

Additionally, Software Defined Networking(SDN) is another concept which is gaining popularity and is complimentary to Network Function Virtualization. When these projects are combined they can provide a complete control over the network. Hence, the future work would also include deploying NFV with project like OpenDayLight which is open source platform for SDN.

Bibliography

[1] http://www.exfo.com/corporate/blog/2016/what-is-nfv

[2] Bouten, N., Boutaba, R., Gorricho, J., Mijumbi, R., Serrat, J., & Turck, F.D. (2016). Network Function Virtualization: State-of-the-Art and Research Challenges. IEEE Communications Surveys and Tutorials, 18, 236-262.

[3] https://en.wikipedia.org/wiki/OpenStack

[4] Tiago Rosado and Jorge Bernardino. An overview of openstack architecture. In Proceedings of the 18th International Database Engineering & Applications Symposium, pages 366–367. ACM, 2014.

[5] https://www.bvoip.com/blog/bvoip-is-deploying-openstack-onto-of-rackspace

[6] http://docs.openstack.org/developer/nova/runnova/.

[7] http://searchsdn.techtarget.com/definition/vCPE-virtual-customer-premise-equipment

[8] .http://www.lightreading.com/carrier-sdn/ nfv-(network-functions-virtualization)/the-rise-of-virtual-epc/a/d-id/708394

[9] https://f5.com/resources/use-cases/virtual-ip-multimedia-subsystem-ims-network

[10] http://www.etsi.org/technologies-clusters/technologies/nfv

[11] https://www.ietf.org/proceedings/88/slides/slides-88-opsawg-6.pdf

[12] https://sreeninet.wordpress.com/2014/07/06/nfv-deepdive/

[13] https://www.opnfv.org/

[14] https://www.opendaylight.org/

[15] https://www.juniper.net/us/en/products-services/sdn/contrail/contrail-cloud/

[16] http://www.networkcomputing.com/networking/juniper-launches-contrail-sdn-software-goes-open-source/726094841

[17] https://wiki.openstack.org/wiki/Tacker

[18] https://wiki.openstack.org/wiki/File:Tacker-Architecture.jpg

[19] https://www.oasis-open.org/committees/tosca/

[20] https://github.com/openstack/tacker/tree/master/samples/tosca-templates

[21] https://assets.sdxcentral.com/openstack-tacker-open-source-project.jpg

[22] https://www.slideshare.net/OPNFV/summit-16-openstack-tacker-open-platform-for-nfv-orchestration

[23] http://blog.csdn.net/YuYunTan/article/details/47816123

[24] https://openwrt.org/