Security Analysis of Network Function Virtualization (NFV)

A Thesis submitted in partial fulfillment of the requirements for the degree of

Master of Engineering

in

Computer Science & Engineering

at

Department of Computer Science & Engineering

Jadavpur University, Kolkata

by

Jyotiprana Bhattacharya

Registration Number- 154128 of 2020-21

Class Roll Number- 002010502004

Examination Roll Number- M4CSE22004

Under the Guidance of

Prof. Chandan Mazumdar

Department of Computer Science and Engineering

Faculty of Engineering and Technology

Jadavpur University, Kolkata- 700032, India

July, 2022

JADAVPUR UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

Certificate of Recommendation

This is to certify that the thesis entitled **"Security Analysis of Network Function Virtualization (NFV)"** has been satisfactorilycarried out by **Jyotiprana Bhattacharya (University Registration Number: - 154128 of 2020-21, Class Roll Number: - 002010502004, Examination Roll Number: - M4CSE22004)**. It is a bona-fide record of workcarried out under my guidance and supervision and be accepted in partialfulfillment of the requirement for the degree of **Master of Engineering in Computer Science and Engineering** from the **Department of Computer Science & Engineering, Faculty of Engineering and Technology, Jadavpur University**, Kolkata - 700032.

Prof. Chandan Mazumdar
(Thesis Supervisor)

Department of Computer Science and Engineering
Jadavpur University, Kolkata, West Bengal, India - 700032

Prof. (Dr.) Anupam Sinha

HOD, Department of Computer Science and Engineering Jadavpur University, Kolkata, West Bengal, India - 700032

Prof. Chandan Mazumdar

Dean, Faculty of Engineering and Technology
Jadavpur University, Kolkata, West Bengal, India - 700032

JADAVPUR UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

Certificate of Approval

This is to certify that the thesis entitled **"Security Analysis of Network Function Virtualization (NFV)"** is a bona-fide record of work carried out by **Jyotiprana Bhattacharya (University Registration Number :- 154128 of 2020-21, Class Roll Number :- 002010502004, Examination Roll Number :- M4CSE22004)** in partial fulfillment of the requirements for the award of the degree of **Master of Engineering in Computer Science and Engineering** from the **Department of Computer Science & Engineering, Faculty of Engineering and Technology, Jadavpur University**, Kolkata - 700032, during the period of July 2021 to July 2022. It is understood that by this approval, the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion withdrawn therein, but approve the thesis only for the purpose for which it has been submitted.

_____

Signature of Examiner

Date :-

Name :-

_____

Signature of Supervisor

Date :-

Name :-

JADAVPUR UNIVERSITY

188, Raja S.C. Mallick Rd, Kolkata - 700032

Declaration of Authorship

I, hereby declare that the work described in this thesis titled **"Security Analysis of Network Function Virtualization (NFV)"** is entirely my own and in accordance to the requirements of my Master of Engineering in Computer Science and Engineering studies.

All information in this document have been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials that are not original to this work

_____

Date :-

Name :- JYOTIPRANA BHATTACHARYA

Registration Number :- 154128 of 2020-21

Class Roll Number :- 002010502004

Examination Roll Number :- M4CSE22004

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompanies the successful completion of this task would be incomplete without the mention of the people who made it possible. Their constant guidance and encouragement crowned my effort with success.

It is a great pleasure to express my sincerest thanks to my project supervisor Prof. Chandan Mazumdar, Professor, Department of Computer Science and Engineering,Faculty of Engineering and Technology, Jadavpur University, for his encouragement, valuable suggestion, and constant support during the course of thisproject.

I would like to thank all the professors of the Department of Computer Science andEngineering, Jadavpur University, Kolkata for the guidance they provided methroughout the duration of the Master of Computer Science Engineering course.

A special note of thanks goes to Prof. Anupam Sinha, Head, Department of Computer Science and Engineering, Jadavpur University.

I am also indebted to Srijita Basu, Subrata Sarkar, Anirban Sengupta and the research scholars of CDC JU lab for their seamless cooperation and help in completion of this project. I am thankful to my fellow classmates, my juniors and my family for constant help and support.

Date: **Jyotiprana Bhattacharya**

Place: ME in CSE,

Registration Number: - 154128 of 2020-21

Class Roll Number: - 002010502004

Examination Roll Number: - M4CSE22004

Abstract

As a result of inflexibility and increasing cost of traditional network, Network Function Virtualization came into focus of the industry. NFV decouples network function software implementation from underlying hardware, and it provides an abstraction of network functions, via software components that can run-on general-purpose devices that can be located in a variety of telecom infrastructure, such as data centers, network nodes, and end-user devices. NFV architecture consists of three main components: NFVI, VNF and MANO. There are various use cases of NFV including using NFVI as service, VNF forwarding graphs etc. Benefits of NFV includes reducing cost in purchasing hardware, faster deployment time, flexibility to scale up or down as per the requirement. It can be beneficial to use a software-defined NFV as NFV can help SDN by virtualizing the SDN controller to run on the cloud, allowing for dynamic relocation of the controllers to the best locations. In return, SDN, benefits NFV by enabling programmable network connectivity between VNFs for optimized traffic engineering and steering. OpenStack, an open-source Infrastructure-as-a-Service (IaaS) platform for cloud computing, offers a number of services that, when coordinated, can be used to create and deploy an NFV system that is both affordable and scalable. However, in order to have a safe NFV environment, each of NFV and OpenStack presents its own unique set of security concerns that must be overcome. As NFV is in its early security stages there are many open security issues present which can be dealt by using some available techniques also through other emerging solutions.

In this thesis, we demonstrate how the most recent version of OpenStack, code-named OpenStack Yoga, may be used to build and deploy an all-in-one single node NFV environment. We will also have a look at the typical NFV use-cases and its architecture. Finally, we will assess the security flaws that pose a threat to our deployment and recommend threat mitigation techniques and security best practices to implement in order to make our NFV environment secure.

Contents

List of Figures

List of Tables

Chapter 1:
Introduction

1. INTRODUCTION

1.1. Virtualization

Virtualization in computing refers to the act of constructing a virtual (rather than actual) version of something at the same abstraction level, such as virtual computer hardware platforms, storage devices, and computer network resources. The applications running on top of the virtualized machine may appear to be on their own dedicated computer, with its own operating system, libraries, and other programs that are distinct from the host operating system that sits beneath it. Virtualization employs software to construct an abstraction layer over computer hardware, allowing the hardware components of a single computer—processors, memory, storage, and so on—to be separated into several virtual computers, also known as virtual machines (VMs). Each VM runs its own operating system (OS) and behaves like a separate computer, despite the fact that it only uses a part of the underlying computer hardware.

Virtual Machine (VM): When a physical machine is virtualized, each virtualization instance is a virtual machine (VM). A virtual machine (VM) functions as if it were its own computer, with its own set of processing power, memory, and storage available on the real hardware. Each VM has its own operating system and apps that are unrelated to the other VMs. If the virtual computer is kept on a virtual disk, this is known as a disk image. A disk image may contain the files required for a virtual computer to boot, or it may contain any other special storage requirements. VMs rely on a piece of software called a hypervisor to interface with the underlying hardware in order to coordinate many virtual instances without interference.

Hypervisor: A hypervisor is the software layer that coordinates VMs. It serves as an interface between the VM and the underlying physical hardware, ensuring that each has access to the physical resources it needs to execute. It also ensures that the VMs don't interfere with each other by impinging on each other's memory space or compute cycles. There are two types of hypervisors:

- Type 1 or "bare-metal" hypervisors interact with the underlying physical resources, replacing the traditional operating system altogether. They most commonly appear in virtual server scenarios.
- Type 2 hypervisors run as an application on an existing OS. Most commonly used on endpoint devices to run alternative operating systems, they carry a performance overhead because they must use the host OS to access and coordinate the underlying hardware resources.

Some of the benefits that Virtualization brings data center operators and service providers are:

Reduce expenses: Organizations frequently deploy servers to run applications that require only a small portion of their available resources. Such servers are never used to their full potential. To make matters worse, these servers are completely idle whenever their applications are not operating. In a virtualized environment, we can accurately allot each VM the amount of computing resources it requires to accomplish its function. The remaining resources are subsequently made accessible to additional virtual machines and their applications. The cost of virtualization is almost always less than the cost of purchasing and maintaining extra hardware.

Resiliency: A virtualized server environment, unlike a traditional system, is not hardware-bound. We can quickly backup, copy, and clone virtual machines to new physical hardware. It can take days, weeks, or even months for new hardware to be ready for deployment. Meanwhile, we can set up a VM backup in only a few minutes.

High availability: We can simply set up redundant virtualized environments with extremely high availability since we can clone a VM almost effortlessly. Virtualization delivers an exceptionally stable system with no single point of failure in hardware or software by automatically monitoring VM status and rapidly transitioning to backup VMs in the event of a failure. These "failover" methods allow users to smoothly resume running the VM from its previous operational state. This ensures that service availability is maximized regardless of what goes wrong. We can monitor, configure, and restart the complete virtual environment remotely. This gives developers continuous access, regardless of how far they are from the physical hardware, which helps to prevent any potential downtime.

Some of the types of Virtualizations are:

- Application virtualization
- Network virtualization
- Storage virtualization
- Desktop virtualization
- Server virtualization
- Data virtualization

1.2. Traditional Network Functions

Routing, switching, load balancing, network address translation, and other network functions are traditionally tied to proprietary hardware deployed as network appliances, which means that the implementation of routing, switching, load balancing, and other network functions varies from vendor to vendor.

In the past, network functions revolved all around hardware but now they have all the flexibility of software. A virtualized network should allow you to take an entire network, complete with all its configurations and functions, and duplicate it in software. You should be able to create and run your virtualized network in parallel on top of your existing network hardware. A virtual network can be created, saved, deleted, and restored, just as you would do with VMs, but in this case, you're doing it with the entire network.

1.3. Network Function Virtualization

Network Function Virtualization decouples the network functions like firewall or encryption from the underlying dedicated hardware by using various virtualization techniques. The European Telecommunications Standards Institute (ETSI), which is the Network Functions Virtualization working group, stated that NFV "aims to address… problems by leveraging standard IT virtualization technology to consolidate many network equipment types onto industry standard high-volume servers, switches and storage, which could be located in datacenters, network nodes and in the end user premises." NFV can offer many network functions like network address translation, load balancing, VPN (Virtual Private Network), routing and many more. If a new network function is needed, NFV allows network operator to perform on demand network functions and create a new virtual machine to perform that network function.

For instance, encryption software can be installed on a standardized server or switch that is already part of the network, rather than introducing a new hardware appliance across the network to allow network encryption.

Because of the greater scalability and adaptability across the entire network, network operators are less dependent on specialized hardware appliances as a result of this virtualization of network operations. NFV aims to offload network functions exclusively, as opposed to the full network, unlike a virtualized network. Below Figure 1[41] shows the difference between classic network appliance approach and Network Function Virtualization approach.

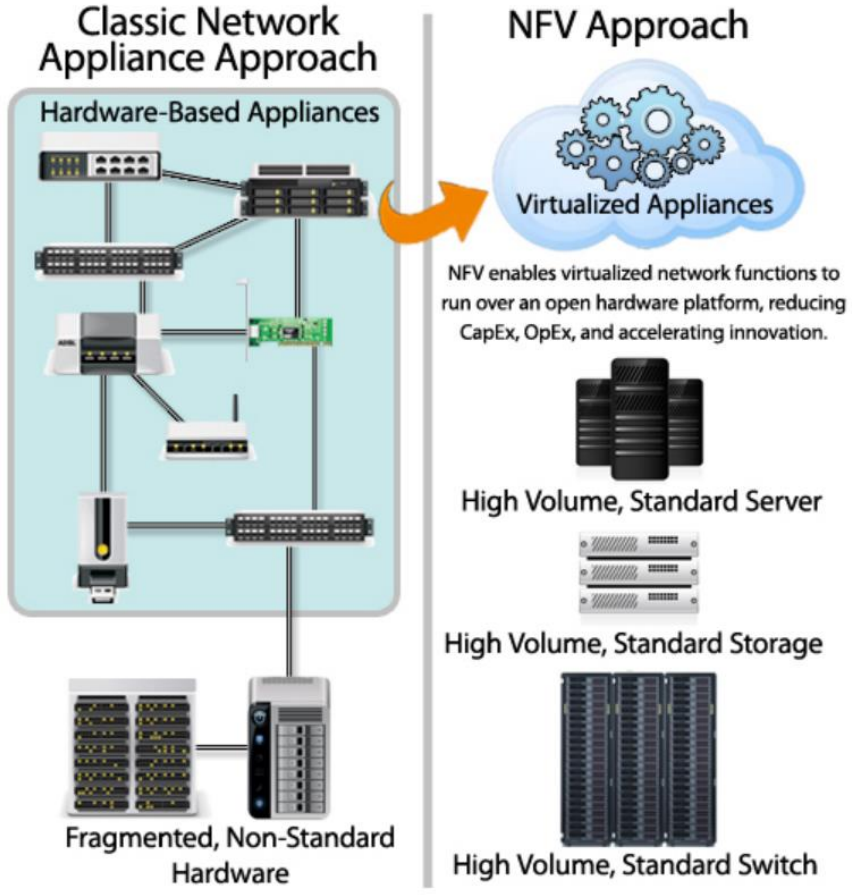

Figure 1: Difference between Classic Network Appliance and NFV approach

1.4. Benefits of Network Function Virtualization (NFV)

By giving service providers the freedom to transfer network operations from specialized appliances to generic servers, network functions virtualization (NFV) aims to lessen the burden. In order to make networks more flexible and effective, NFV intends to consolidate many networks equipment types onto industry-standard, high-volume servers, switches, and storage.

Additionally, NFV is scalable, secure, affordable, and flexible. These advantages help NFV handle various trends affecting service provider networks. Some of the benefits of NFV are as follows:

i. The main driver of NFV is cost efficiency. Consolidating equipment results in lower equipment costs and lower power consumption.
ii. Increases the velocity of Time to Market by shortening the typical network operator innovation cycle. It enables the network operator to reduce the time to deploy new services.
iii. The ability to run production, test, and reference facilities on the same infrastructure allows for far more efficient testing and integration, lowering development costs and speeding up time to market.
iv. Service providers constantly want to make sure they can scale up or down their capacity as their network expands and meet new requirements. When using conventional network equipment, it takes effort, preparation, and money. These worries are overcome by NFV since it provides a method for expanding and contracting resources, which permits capacity modifications. Scalability and automation are made possible, network service provisioning is made more flexible, and the time required to roll out new services is decreased. Furthermore, provisioning services remotely in software improves service deployment speed by eliminating the need for site visits to install new hardware.
v. Supports multi-tenancy, allowing network operators to provide customized services and connections to various users, applications, internal systems or other network operators, all while coexisting on the same hardware with suitable secure administrative domain separation.
vi. Improves operational efficiency by using the physical network platform's higher uniformity and homogeneity with other support platforms.
vii. Reduces energy consumption by utilizing power management features in standard servers and storage, as well as workload consolidation and location optimization. For example, using virtualization techniques, it would be possible to concentrate the workload on a smaller number of servers during off-peak hours (like overnight), allowing all other servers to be turned off or put into an energy-saving mode.
viii. Providing opportunities for a diverse range of eco-systems and encouraging openness. It allows pure software entrants, small players, and universities to enter the virtual appliance market, enabling more innovation to bring new services and income streams to market faster and with less risk.

1.5. Objective

The goals of this thesis are to examine network function virtualization, various approaches to achieve network function virtualization, and how OpenStack and mininet can be used to meet our demands for network function virtualization. Internal communication among two or more OpenStack Virtual Machine instances have also been implemented and tested using an intermediary VNF. Moreover, a controlled sequence of packet flows has been realized by customizing the VNF descriptor.

We are also exploring ways to modify OpenStack's security features to meet our security property needs. Further, we have concentrated on the analysis of OpenStack's setup and evidence-gathering procedures, as well as how we may structure our network function virtualization platform to address current security concerns.

1.6. Structure of the Thesis

The rest of the document is organized as follows: Chapter 2 describes the proprietary and open stacks available for Network Function Virtualization and its working mechanism. Chapter 3 describes What is Network Function Virtualization, its architecture and the use cases of network function virtualization. Chapter 4 details our choice of Network function virtualization implementation i.e., OpenStack and Mininet and their detailed architecture. Chapter 5 discusses the implementation mechanism of NFV using OpenStack and Mininet. It also describes the working of Virtual Network Functions and the way we can modify them as per our requirement. Additionally, we have developed a packet counting VNF and implemented communication between two or more virtual instances using that VNF. Chapter 6 details the Implementation level mechanism of Network Function Virtualization using OpenStack. Chapter 7 describes the configuration methodology and evidence identification in network function virtualization using OpenStack. The overall conclusions drawn on the subject appear in Chapter 8 and it also includes future scope of the work. The references cited in the work are placed at the end.

Chapter 2:
Technology Survey

2. TECHNOLOGY SURVEY

This section will list several proprietary and open-source components available from various manufacturers for deploying an NFV environment. These suppliers either supply all or some of the functional NFV components. An enterprise can employ these functional blocks from the same or various vendors and combine them to construct a working NFV system.

I. Cisco (Proprietary):

NFVO: Cisco Network Services Orchestrator (NSO) is industry-leading software for automating services across traditional and virtualized networks.

Virtualized Infrastructure: Fully integrated Red Hat Enterprise Linux and Red Hat OpenStack Platform runs on top of Cisco Unified Computing System™ (Cisco UCS®). It is open source yet hardened and mature.

SR1000V, XRv, ASAv, WSAv, vFirePower, vSCE, vISE NetScaler 1000v, vNAM, vWaaS, CUCM are some of the VNFs that are provided by cisco.

II. Juniper (Proprietary):

NFVO & VNFM: A programmable cloud reference architecture for MANO that leverages Contrail for a turnkey management and orchestration platform.

Virtualized Infrastructure: A horizontal, pre-validated NFVI stack that leverages Contrail for a turnkey management and orchestration platform.

VNFs are enabled through vSRX and vMX. Customer premises equipment to securely extend VNFs to the end users with the NFX250 Network Services Platform.

III. VMware (Proprietary):

NFVO & VNFM: Cloudify provides infrastructure automation using 'Environment as a Service' technology to deploy and continuously manage any cloud, private data center.

Virtualized Infrastructure: vCloud NFV - vSphere, vSAN, NSX, vRealize Operations, vRealize Logs. vCPE - Antivirus, Firewall, WAN Opti vIMS - IMS Core, Voice Recording vProb - Correlation and Analytics vEPC - SGW, MME, HSS, PGW, PCRF Brocade Virtual Core Mobile (VCM) are some of the VNFs that are provided by VMware.

IV. Redhat (Proprietary):

NFVO & VNFM: The Management layer is mainly composed of:

- Red Hat CloudForms: Red Hat CloudForms is the cloud management tool that presents data from multiple sources like the Virtual Infrastructure Manager (VIM) and the Network Functions Virtualization Infrastructure (NFVI) in a unified display. It allows you to manage hybrid VIM scenario (VNF running on legacy virtualization platform and VNF running on top Openstack platform) to consolidate FCAPS.
- Red Hat OpenStack Platform director: Red Hat CloudForms is the cloud management tool that presents data from multiple sources like the Virtual Infrastructure Manager (VIM) and the Network Functions Virtualization Infrastructure (NFVI) in a unified display. It allows you to manage hybrid VIM scenario (VNF running on legacy virtualization platform and VNF running on top Openstack platform) to consolidate FCAPS.
- FCAPS operational tools: Operational tools provide fault, configuration, accounting, performance, security (FCAPS) management. These tools allow you to monitor the health of the NFVi.

Virtualized Infrastructure: The Infrastructure layer is the foundation of the Red Hat NFV Solution. The main components are: Red Hat Enterprise Linux (RHEL), Software-Defined Storage with Ceph,

Software-Defined Networking (SDN) with Open Daylight or a certified 3rd party controller, Open vSwitch (OVS) standalone, or accelerated with Data Plane Development Kit (DPDK)

vDDoS, vCPE, vEPC, vSBC, vRouter, vLB, vFW, vPCRF, vCDN, vADC, vIMS are some of the VNFs that are provided by Redhat.

V. OpenStack (Open):

NFVO & VNFM: Tacker and Heat services provides the orchestration for NFV.
Virtualized Infrastructure: Common x86 Server Platform and OpenStack provides the needed infrastructure for NFV.
Native Firewall (in Neutron), Native Load Balancer (in Neutron) are some of the VNFs that are provided by OpenStack. 3rd party VNFs can also be used in OpenStack.

VI. Oracle (Proprietary):

NFVO & VNFM: Oracle Network Service Orchestrator serves as NFVO and Oracle Application Orchestrator serves as VNFM for NFV.
Virtualized Infrastructure: Servers Storage Networking provides the needed infrastructure for NFV.
3rd party VNFs can also be used in Oracle NFV solution.

As we can see that OpenStack provides solutions for each of the NFV components, which makes it a suitable for use to deploy NFV environments. Apart from being open-source, below are few reasons as to why OpenStack is synonymous with NFV deployments: -

- The OpenStack platform serves as the cornerstone for NFV architecture, which is essentially a purpose-built cloud for building, coordinating, and maintaining virtual network services.

- The OpenStack project's open, modular, and interoperable platform allows telecoms and organizations to construct their own NFV system, free of superfluous components.

- Standardized connections between infrastructure and NFV components. Pluggable architecture with specified APIs, user interfaces, shared services, operations, and automation options for VNFs and other functionalities including open source and commercial network plug-ins and driver integration.

- Neutron, an OpenStack networking project, is notably beneficial in NFV Orchestration, as well as NFV deployment automation and network function rollout.

Chapter 3:

Network Function Virtualization (NFV)

3. NETWORK FUNCTION VIRTUALIZATION (NFV)

Network Function Virtualization decouples the network functions like firewall or encryption from the underlying dedicated hardware by using various virtualization techniques. The European Telecommunications Standards Institute (ETSI), which is the Network Functions Virtualization working group, stated that NFV "aims to address… problems by leveraging standard IT virtualization technology to consolidate many network equipment types onto industry standard high-volume servers, switches and storage, which could be located in datacenters, network nodes and in the end user premises." NFV can offer many network functions like network address translation, load balancing, VPN (Virtual Private Network), routing and many more. If a new network function is needed, NFV allows network operator to perform on demand network functions and create a new virtual machine to perform that network function.

3.1. NFV Architecture

The NFV Architecture, according to ETSI, is made up of three main components: Network Function Virtualization Infrastructure (NFVI), Virtual Network Functions (VNFs), and NFV Management and Orchestration (MANO). Figure 2[42] illustrates the high-level NFV framework and Figure 3[1] shows the NFV reference architectural framework

- **Virtualized network functions (VNFs):** This is the basic block of NFV architecture. A virtual network function (VNF) is a Network Function implementation that is deployed on virtual resources such as a Virtual Machine (VM). A single VNF can be made up of multiple internal components so, it can be deployed across multiple VMs, with each VM hosting a single component of the VNF. A whole VNF can also be deployed on a single VM as well.

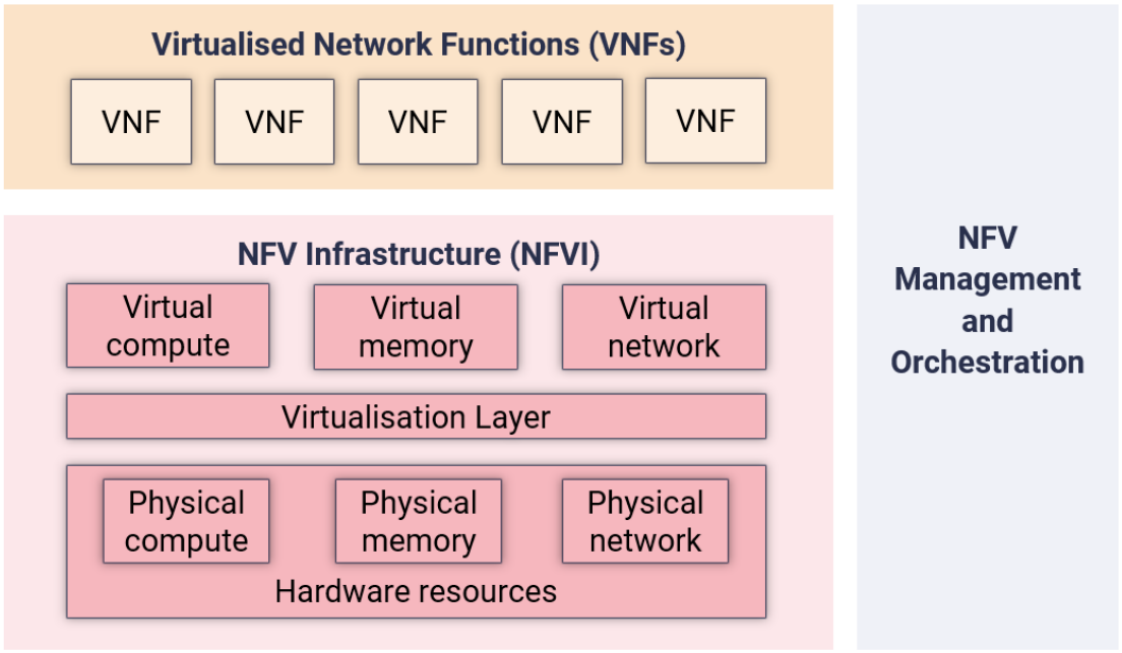

Figure 2. High level NFV framework

- **Network Function Virtualization Infrastructure (NFVI):** It's the environment in which VNFs are deployed, managed, and executed is made up of a combination of hardware and software resources. Abstraction of computing, storage and network resources are called virtual resources. The virtualization layer (based on a hypervisor) achieves the abstraction by decoupling the virtual resources from the underlying physical resources.

- **NFV Management and Orchestration (MANO):** The MANO framework is used to manage and coordinate all of the resources in the NFV environment. It is where the infrastructure layer's resource management occurs, as well as where resources are created, assigned, and the allocation of VNFs is controlled.

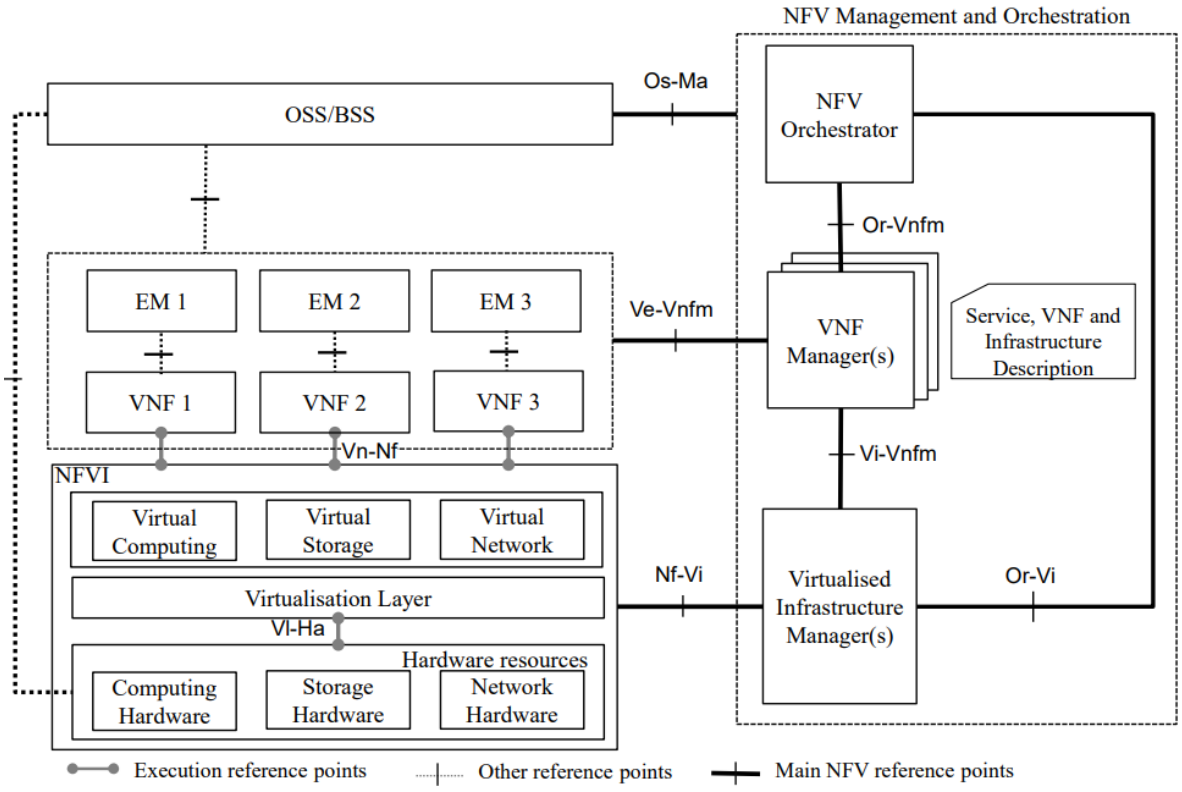

Figure 3: NFV reference architectural framework

3.1.1. Network Function Virtualization Infrastructure (NFVI)

The platform on which VNFs are installed, managed, and executed is called Network Function Virtualization Infrastructure (NFVI), and it encompasses both the hardware resources and the virtual instantiations that make up the infrastructure. The NFVI could be a component of the cloud Infrastructure-as-a-Service (IaaS), which cloud providers utilize to build Virtual Data Centers (VDCs), which include all the virtualized computing, storage, and networking requirements to function as a physical data center. These VDCs are given to NFV service providers, who use them to offer network services to NFV customers.

From the VNF's perspective, the virtualization layer and the hardware resources look like a single entity providing them with desired virtualized resources. NFVI contains three main components: -

- **Hardware Resources:** The virtualization layer connects VNFs to physical hardware resources such as computation, storage, and networks for processing, storing, and connection (e.g. hypervisor). Instead of being custom-built, computing hardware is considered to be commercial-off-the-shelf (COTS). Storage resources can be divided into two categories: storage that is located on the server, and shared network attached storage (NAS). Resources for computing and storage are frequently combined. Switching components, such as routers, and wired or wireless links make up network resources. Network resources can also be spread across numerous domains**.**

- **Virtualization Layer:** The virtualization layer abstracts the hardware resources and decouples the VNF software from the underlying hardware, thus ensuring a hardware independent lifecycle for the VNFs. In short, the virtualization layer is responsible for:
    - Abstracting and logically partitioning physical resources, commonly as a hardware abstraction layer.
    - Enabling the software that implements the VNF to use the underlying virtualized infrastructure.
    - Providing virtualized resources to the VNF, so that the latter can be executed.

  The middle virtualization layer ensures that VNFs are separated from hardware resources so that the software can be installed on various types of actual hardware resources.

- **Virtualized Resources:** The compute, storage, and network resources are decoupled from the hardware layer by the virtualization layer and made available as virtual resources. However, these virtual resources are supported by the same physical infrastructure.

3.1.2. Virtual Network Functions (VNFs)

A Network Function (NF) is an element within a network with well-defined external interfaces and functional behavior e.g., DHCP, Firewall, SAE Gateway, MME, eNodeB. Virtual Network Function (VNF) is a software implementation of a Network Function that can be easily deployed on virtual resources such as a Virtual Machine (VM). A service is an offering provided by a Service Provider that is composed of one or more VNFs. Functional behavior and state of a NF are largely independent of whether the NF is virtualized or not. The functional behavior and the external operational interfaces of a PNF and a VNF are expected to be the same.

VNF can be composed of multiple internal components. For example, one VNF can be deployed over multiple VMs,where each VM hosts a single component of the VNF. However, in other cases, the whole VNF can be deployed in a single VM as well. In order to simplify deployment and management, a single VNF may be made up of a number of internal parts, such as firewalls, residential gateways, packet data network gateways (PGW), etc. However, a VNF may also include just one component in order to maximize scalability and reuse, as well as to have a quicker reaction time due to its simplicity; keep in mind that a single VNF may be deployed and dispersed across multiple VMs [1]. According to the demands of the consumers, TSPs' virtual network services are typically made up of a variety of VNF.

Every VNF is directly connected to an Element Management System (EMS), which is responsible for carrying out the standard management functions for each VNF connected to it. In order to simplify deployment and management, a single VNF may be made up of a number of internal parts, such as firewalls, residential gateways, packet data network gateways (PGW), etc. However, a VNF may also include just one component in order to maximize scalability and reuse, as well as to have a quicker reaction time due to its simplicity; keep in mind that a single VNF may be deployed and dispersed across multiple VMs [1]. According to the demands of the consumers, TSPs' virtual network services are typically made up of a variety of VNF. Every VNF is directly connected to an Element Management System (EMS), which is responsible for carrying out the standard management functions for each VNF connected to it.

3.1.3. NFV Management and Orchestration (NFV MANO):

From merging many services into a single VNF package to mapping this service to the customers upon request, NFV MANO is in charge of managing and orchestrating all the virtualization-specific duties necessary throughout the lifecycle of the VNF. Along with managing potential VNF failures, NFV MANO also maintains status data on each VNF in the service. Additionally, the NFV MANO is in charge of establishing the communications between various VNFs that work together to form the network graph. A network graph is a collection of VNF services that work together to give the customer the service they were looking for, allowing them to construct a virtual network using the VNFs that were made accessible.

The NFV Management and Orchestration contains three main functional blocks which are-

I. **Virtualized Infrastructure Manager (VIM):** NFVI resources including network, computation, and storage are managed and managed by the VIM. By opening up the interfaces to the appropriate resource controllers, VIM can also be modified to manage at least one particular type of NFVI resource (such as network-only, compute-only, or storage-only). Multiple Virtualized Infrastructure Manager instances can be deployed. The Virtualized Infrastructure Manager performs:

- Management of Inventory of software (e.g., hypervisors), computing, storage and network resources dedicated to NFV infrastructure.
- Allocation of virtualization enablers, e.g., VMs onto hypervisors, compute resources, storage, and relevant network connectivity.
- Management of infrastructure resource and allocation, e.g., increase resources to VMs, improve energy efficiency, and resource reclamation.
- Visibility into and management of the NFV infrastructure.
- Root cause analysis of performance issues from the NFV infrastructure perspective.
- Collection of infrastructure fault information.
- Collection of information for capacity planning, monitoring, and optimization.

II. **VNF Manager (VNFM):** A VNF Manager is responsible for VNF lifecycle management (e.g., instantiation, update, query, scaling, termination). The VNFM is responsible for managing multiple VNF instances. One thing should be noted, though: while one VNFM may be tasked with managing

several VNF instances, only one VNF instance is linked with each individual VNFM. The majority of VNFMs are made to support all VNF types as well as management tasks like VNF instantiation, updating, searching, extension, and termination. The functionalities provided by VNFM are: -

- Management of the integrity of the VNF instance through its lifecycle.
- Lifecycle management of each VNF instance.
- Overall coordination and adaptation role for configuration and event reporting between the VIM and the EM.

III. **NFV Orchestrator (NFVO):** The Orchestrator is in charge of the orchestration and management of NFV infrastructure and software resources, and realizing network services on NFVI**.** The NFV Orchestrator has two main responsibilities:

- The orchestration of NFVI resources across multiple VIMs, fulfilling the Resource Orchestration functions
- The lifecycle management of Network Services, fulfilling the Network Service Orchestration functions

Some of the capabilities provided by NFVO via its Network Service Orchestration functions are:

- Management of Network Services deployment templates and VNF Packages
- Network Service instantiation and Network Service instance lifecycle management, e.g. update, query, scaling, collecting performance measurement results, event collection and correlation, termination.
- Management of the instantiation of VNF Managers where applicable.
- Policy management and evaluation for the Network Service instances and VNF instances

Some of the capabilities provided by NFVO via its Resource Service Orchestration function are:

- Validation and authorization of NFVI resource requests from VNF Manager(s),
- Supporting the management of the relationship between the VNF instances and the NFVI resources allocated to those VNF instances by using NFVI Resources repository and information received from the VIMs.

NFV architecture also includes a number of secondary reference points and supporting systems that serve as interfaces to the three primary building pieces. The reference points are: -

I. **Virtualization Layer - Hardware Resources - (Vl-Ha):** This reference point interfaces the virtualisation layer to hardware resources to create an execution environment for VNFs, and collect relevant hardware resource state information for managing the VNFs without being dependent on any hardware platform.

II. **VNF - NFV Infrastructure (Vn-Nf):** This reference point represents the execution environment provided by the NFVI to the VNF. It does not assume any specific control protocol. It is in the scope of NFV in order to guarantee hardware independent lifecycle, performance and portability requirements of the VNF.

III. Orchestrator - VNF Manager (Or-Vnfm): This reference point is used for:

- Resource related requests, e.g. authorization, validation, reservation, allocation, by the VNF Manager(s).
- Sending configuration information to the VNF manager, so that the VNF can be configured appropriately to function within the VNF Forwarding Graph in the network service.
- Collecting state information of the VNF necessary for network service lifecycle management

IV. Virtualized Infrastructure Manager - VNF Manager (Vi-Vnfm): This reference point is used for:

- Resource allocation requests by the VNF Manager.
- Virtualized hardware resource configuration and state information (e.g. events) exchange.

V. Orchestrator - Virtualized Infrastructure Manager (Or-Vi): This reference point is used for:

- Resource reservation and/or allocation requests by the Orchestrator.
- Virtualized hardware resource configuration and state information (e.g. events) exchange.

VI. NFVI - Virtualized Infrastructure Manager (Nf-Vi): This reference point is used for:

- Specific assignment of virtualized resources in response to resource allocation requests.
- Forwarding of virtualized resources state information.
- Hardware resource configuration and state information (e.g. events) exchange.

VII. OSS/BSS - NFV Management and Orchestration (Os-Ma): This reference point is used for:

- Requests for network service lifecycle management.
- Requests for VNF lifecycle management.
- Forwarding of NFV related state information.
- Policy management exchanges.
- Data analytics exchanges.
- Forwarding of NFV related accounting and usage records.
- NFVI capacity and inventory information exchanges.

VIII. VNF/EMS - VNF Manager (Ve-Vnfm): This reference point is used for:

- Requests for VNF lifecycle management.
- Exchanging configuration information.
- Exchanging state information necessary for network service lifecycle management.

IX. Service, VNF and Infrastructure Description: NFV Management and Orchestration (Se-Ma): This reference point is used for retrieving information regarding the VNF deployment template, VNF Forwarding Graph, service-related information, and NFV infrastructure information models. The information provided is used by NFV management and orchestration.

X. Operations Support Systems and Business Support Systems (OSS/BSS): Operation

Support System (OSS), as name suggests, is a software tool that is generally used to organizations to manage their operation system or communication networks. Business Support System (BSS), is a software tool that is generally used by organization to manage all business activities such as processing, financial issues, etc.

3.2. NFV Use Cases

In the initial phase of work, ETSI proposed nine typical NFV use cases that considerably promote NFV standards and related products. Some of the use cases are as follows-

3.2.1. NFV Infrastructure as a service (NFVIaaS):

In general, it is nearly impossible for any service provider to build and maintain its own physical infrastructure throughout the world, despite the fact that user demands are worldwide. As a result of this inconsistency, the concept of NFVI as a Service (NFVIaaS) emerges. NFVIaaS stands for "NFVI as a Service," which means that NFVI can be offered and sold from one service provider to another. In this case, it is very convenient and cost-effective for one service provider to serve clients who live far away from its own NFVI sites by running VNF instances remotely on the NFVI platforms of other service providers. From the standpoint of NFVI owners, they must ensure that only authorized customers are able to run and deploy VNF instances on their platforms. Meanwhile, customer-isolated and resource-constrained methods are required to prevent customers from impacting one another. As a result, a well-designed NFVI is very crucial for NFVIaaS implementation.

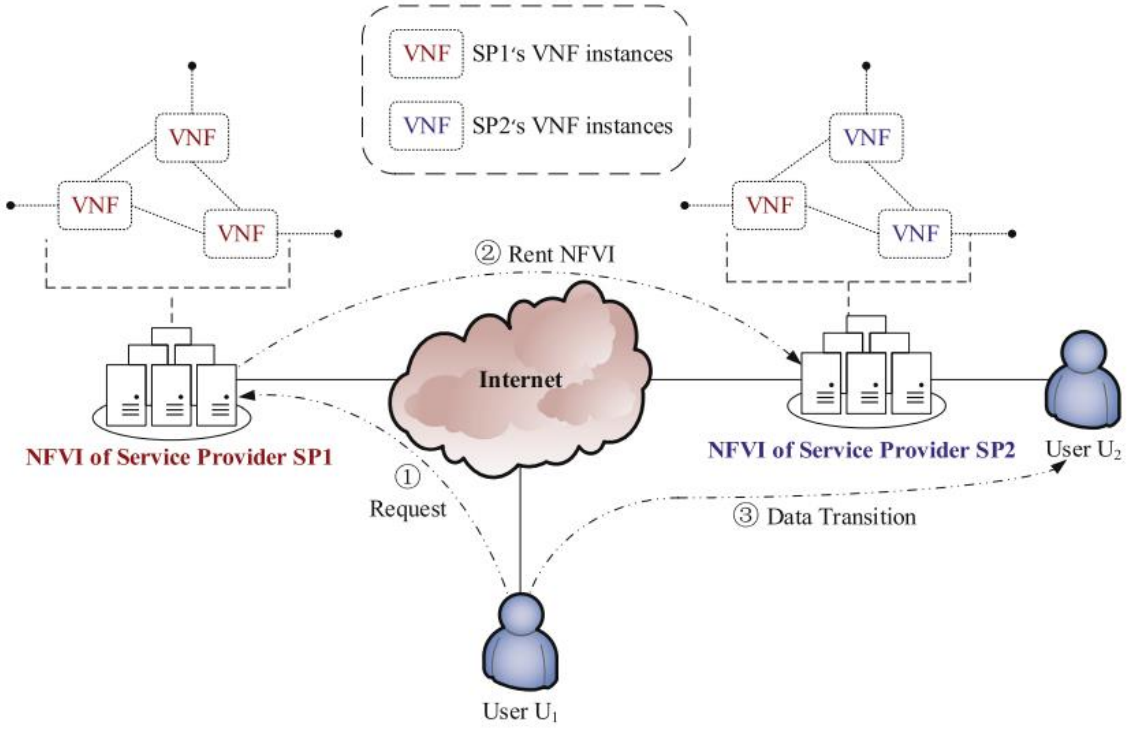

Figure 4: NFV Infrastructure as a service example

Figure 4[6] shows an example in which service provider SP1 operates its VNF instances on service provider SP2's NFVI infrastructure. The process can be broken down in 3 simple steps- At first (Step 1), before reaching the destination user U2, the user U1 asks SP1 to prepare a specific VNF instance. However, SP1 recognizes that the user's destination is close to SP2, and compulsively creating the requisite VNF instance on SP1's NFVI would be too expensive. A better option would be, renting SP2's NFVI and deploying the appropriate VNF instance on it. (Step 2). When the deployment is complete, traffic between U1 and U2 is routed through the VNF instance on SP2. (Step 3). Here, Step 2 reflects the NFVIaaS.

3.2.2. VNF forwarding graph (VNF FG):

Any data center will install a large number of service nodes at various points along the network architecture, on which a range of layer 4 through layer 7 service functions (VNFs) will be deployed in both physical and virtual forms. VNFs hosted at one service node may overlap with those hosted at other service nodes in this way, and this condition can occur between any two data centers. For example, four different data centers owned by different service providers are shown in Fig. 5, where the second data center and the third data center offer the same VNF (i.e., VNF-A). The service delivery in virtualized network environment is much more complex than the traditional network traffic. The network functions provided by data centers are abstracted in the form of VNFs, with their interconnection preserved. All these abstracted VNFs constitute to the graph which is referred to as VNF Forwarding Graph (VNF FG). VNF FG can be thought of as an analogue of a physical network forwarding graph, which connects physical appliances via bidirectional cables and actually connects VNFs via virtual links to describe traffic among these VNFs. VNF chaining algorithms can handle traffic routing logic among these connected VNFs. After that, depending on the control logic, the concrete traffic steering technologies are used to insert headers that carry service path information into packets, thereby completing the traffic steering process in the physical network. The advantage of adopting VNF FG is that when building services, service providers do not have to worry about the underlying infrastructure. The service providers are offering pay-per-use VNF instances, and they may provide the same VNFs. Users can choose VNF providers based on a variety of parameters, including pricing and proximity, which not only helps users but also encourages competition among service providers.

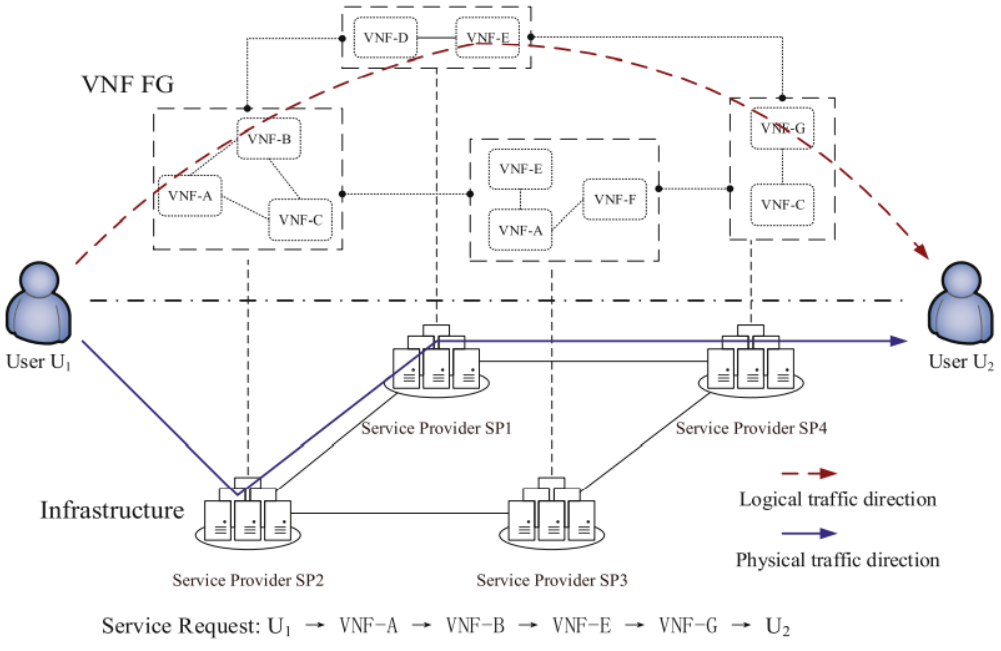

Figure 5: VNF forwarding graph example

Figure 5[6] shows a case of unicast service function chaining where the VNF FG is used to direct the service chaining process and four data centers provide the execution environment for VNF instances.
It appears that four separate service companies own the data centers. They each offer several VNF instance types. It must travel through four separate VNF instances, VNF-A, VNFB, VNF-E, and VNF-G, in accordance with the request from the user U1. From the perspective of VNF FG located upside Fig. 4, the red dotted line represents the logical path of traffic from U1 to U2. The logical path is consequently mapped onto the underlying infrastructure, which is SP2 (supporting VNF-A and VNF-B), SP1 (supporting VNF-E), and SP4 (supporting VNF-G).

3.2.3. Mobile core network virtualization:

By using NFV technology, many mobile network functions can be isolated from the hardware and implemented as software. Because of this decoupling, NFV provides a complete virtual environment with some programmability to the mobile network. As a result, a growing number of third-party creative applications are being developed to meet the ever-increasing demands of users, bringing several benefits to network management and operation. Because functions are implemented on virtual platforms, we may easily close idle virtual machines or re-open those that are required, resulting in increased energy efficiency. In addition to the decoupling between hardware and network function, the underlying proprietary infrastructure can be replaced with cheap and general-purpose devices.

3.2.4. Content delivery network virtualization:

Carrier networks are experiencing numerous issues as a result of the vast and ever-increasing video traffic generated by end users, with content delivery being one of them. Traditional infrastructure-based networks can provide content delivery services as demanded by customers. But the service quality offered by traditional networks cannot be guaranteed. As a result of the clear conflict between high demand and inadequate quality, user experience suffers. As a result, integration of Content Delivery Network (CDN) cache nodes into networks started, which can be a cost-effective and effective way to manage such huge traffic requests while maintaining a high level of service quality. Obtaining material from CDN-nodes near the consumer rather than remote source nodes can save a lot of network resources and money while also allowing high-bandwidth and high-quality data streams to be sent. The CDN has two parts which are distributed cache nodes and the centralized controller. The cache nodes, restore some required content, while the controller determines which cache node should be used to supply the needed content. In this approach, CDN virtualization has two components: controller virtualization and cache node virtualization. The public focus is concentrated on cache node virtualization in order to achieve better performance in areas such as network latency and throughput.

3.2.5. Virtualization of home network:

Through dedicated customer premises equipment (CPE) and network-based back-end systems, network service providers provide home services. Residential gateways (RGs) for Internet access and set-top boxes (STBs) for multimedia services are examples of common CPE hardware. Due to the interactive stream management features, such as rewind and fast forward, it is well recognized that under this architecture providing time-shifted Internet Protocol television (IPTV) services can be challenging. With high-throughput last-mile connection available, developing NFV technology makes it easier to virtualize the home network and

reduces the complexity of IPTV services. In Figure 6[7], the architecture of virtualized home networks is shown. STBs and a variety of RG parts, including the firewall, DHCP server, VPN gateway, and NAT router, are the virtualization targets. The three grey boxes in the bottom left corner of Fig. 4 show how network and service operators can reduce their costs by only offering low-cost, low-maintenance devices to clients by transferring them to data centers. As the layer 3 and above functionalities of RGs are moved into the operators' network, these devices just need to provide layer 2 capability for Internet access. It is observed that with this virtual architecture, customers can share some RG and STB features.

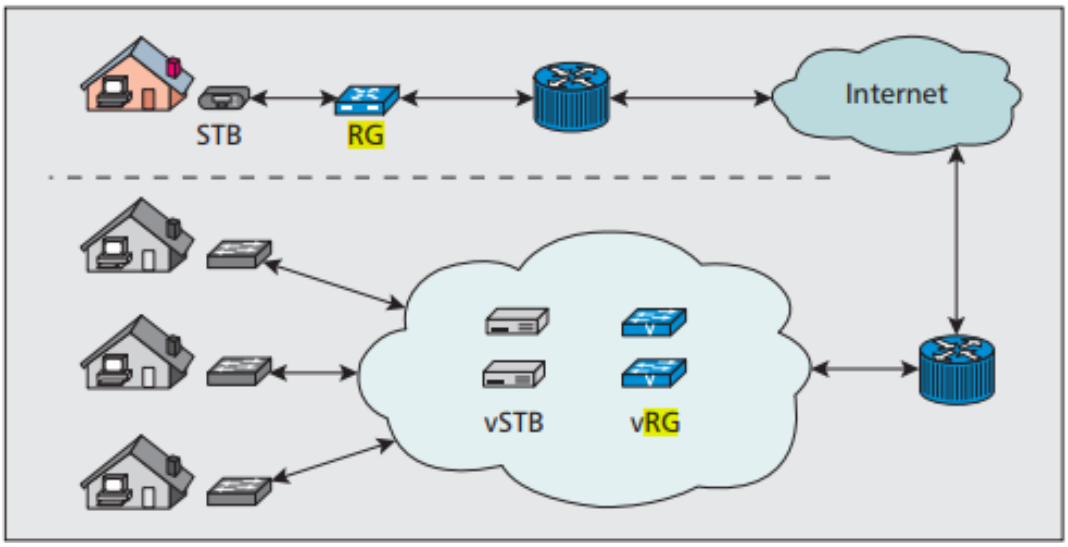

Figure 6. Virtualization of home network

Chapter 4:
Methodologies used for NFV

4. METHODOLOGIES USED FOR NFV

4.1. OpenStack

It is an open-source, free cloud computing platform that was created on July 21st, 2010. To make cloud computing more pervasive, Rackspace Hosting and NASA collaborated on this initiative. When consumers are provided with virtual resources through public or private clouds, it is implemented as infrastructure-as-a-service (IaaS). The software platform consists of interconnected parts that operate data center-based multi-vendor hardware pools of computing, storage, and networking resources.

The building blocks for this platform are referred to in OpenStack as "projects." Numerous services, including computing, networking, and storage services, are handled by these projects. In contrast to virtualization, where resources like RAM, CPU, etc. are detached from the hardware using hypervisors, OpenStack uses a number of APIs to do so. This enables users and administrators to communicate directly with the cloud services. Some of the benefits of using OpenStack are:

- **Easy scalability:** Physical servers can easily run out of resources when they are most needed, while cloud configurations can keep up with demand and give their virtual servers more resources as and when they are required.
- **Easy automation:** One of OpenStack's primary selling features as compared to other solutions is how simple it is to automate operations. This makes OpenStack particularly potent. The software includes built-in technologies that make cloud management quite simple.
- **Fast development:** Therefore, OpenStack is comparable to Linux in that there are numerous distributions, each with a variety of features but all sharing the same fundamental building blocks. This encourages innovation because no one entity holds exclusive rights to the software. Since the fundamental code is available for free, suppliers and resellers must all start from the same point and exercise ingenuity to develop a spin-off product that introduces something novel to the market.
- **Strong Community:** There are a lot of users and developers of OpenStack who enjoy getting together to discuss the product. Over 4,000 developers have reportedly contributed to OpenStack, according to latest data.
- **Easy-to-Manage Panel:** An easy-to-use control panel for OpenStack's power management tools offers visibility, control, and quick access. Consequently, customers will find it very simple to administer and monitor their cloud services, and users and administrators will be able to see clearly how resources are managed and how many active VM instances there are.

In addition to the standard infrastructure-as-a-service (IaaS) features, other OpenStack components include orchestration, fault management, and service management, among other services, to guarantee the high availability of user applications. Numerous plug-ins, modules, and services offered by OpenStack are frequently used in NFV implementations. The management of virtual or bare metal servers is done using the OpenStack Compute service (Nova). Virtual storage is provided by the OpenStack Block Storage service (Cinder), and virtual networking is done so by the OpenStack Networking service (Neutron). These three services regulate the administration of physical and virtual resources to meet the resource needs of the Virtual Network Functions, coupled with OpenStack Authentication service (Keystone), which offers authentication options for administrators (VNFs). Templates that explain the infrastructure required to handle the whole lifetime of the VNF are provided by the OpenStack Orchestration Service (Heat). In an NFV system, Heat functions as a component of the NFVO to push the infrastructure requirements for a network service as specified by the VNF to the VIM in order to enable the creation of OpenStack resource types (such as instances, floating IPs, volumes,

security groups, and users) for that network service. The Tacker service/project in OpenStack consists of an NFVO and a VNFM that aid in the onboarding, deployment, and management of Network Services and Virtual Network Functions (VNFs).

4.1.1. OpenStack Architecture

One illustration of the most typical integrated services found in OpenStack, along with how they communicate with one another, is provided by the OpenStack Logical Architecture (Figure 7[38]). The dashboard, CLIs, and APIs all allow interaction with end users. All services use a single Identity service for authentication, and all communication between services is done through open APIs, with the exception of situations where privileged administrator commands are required.

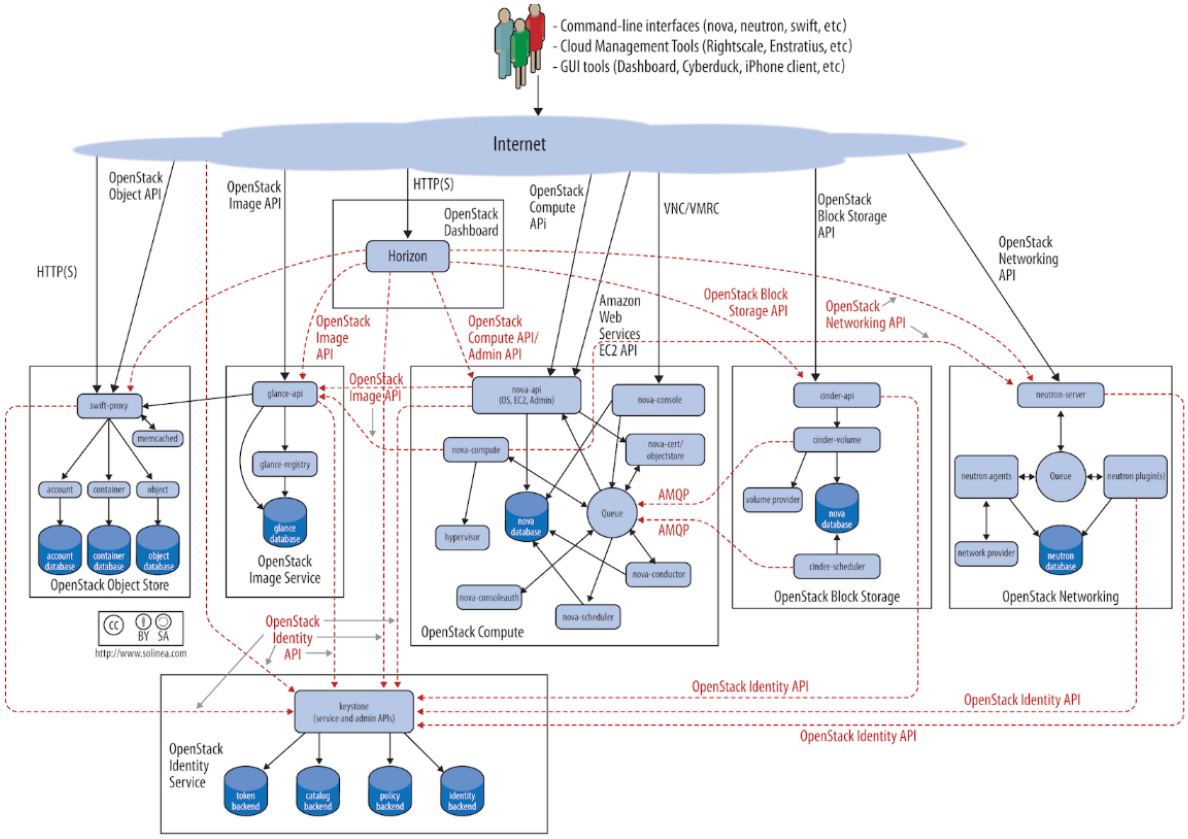

Figure 7. OpenStack Logical Architecture

The fundamental components are divided into five areas based on their features, including computing, networking, storage, sharing, and supporting services. Although not native to the core of OpenStack, the supporting services are crucial for its operation. Services can be installed in line with specifications; therefore, all of them be can install or just a selected handful. In Figure 8[43] we can see high-level overview of some OpenStack core services and their relationship with each other. Some of the services provided by OpenStack are as follows:

Nova (Compute Service): It also goes by the names OpenStack Compute or Nova Compute. It offers virtual servers on demand that interact with several hypervisors, including KVM, Xen, VMware, and Hyper-V. Virtual machines and bare-metal servers are supported by Nova. It offers limited support for different system containers. It runs as a daemon that is installed on the top of the current Linux server to provide that service. It is a variety of software that offers cloud resource management services via its APIs and is able to coordinate active instances, networks, and access control. It is a necessary service for the creation of a simple cloud architecture.

Neutron (Networking Service): One way to describe Neutron is as an OpenStack project. It provides "network connectivity as a service" across different interface devices (such vNICs), which are managed by some other types of OpenStack services (such as Nova). It manages OpenStack's Networking API. It manages all networking aspects for PNI (Physical Networking Infrastructure) and VNI (Virtual Networking Infrastructure) on an OpenStack platform, as well as many authorization layer aspects. Projects can create sophisticated virtual network topologies using OpenStack networking. It may include some services like a firewall and a virtual private network (VPN). Its architecture enables customers to benefit from frameworks (such as intrusion detection systems, load balancing, and virtual private networks) from supported vendors.

Cinder (Block Storage Service): OpenStack's Cinder service uses block storage to offer volumes to Nova VMs, containers, ironic bare-metal hosts, and more. Cinder volumes facilitate persistent storage for guest VMs which are called instances. OpenStack compute software handles these. Additionally, Cinder can be utilized as software-defined stand-alone storage independently of other OpenStack services. The detaching, attaching, replication, creation, and snapshot management of several block devices to the servers is handled by this block storage system. This service was a crucial component of Nova in early OpenStack releases under the name nova-volume. Through its API, Cinder allows the manipulation of volumes, volume kinds, and volume snapshots, primarily interacting with Nova to provide volumes for its instances.

Keystone (Identity Service): For the purposes of implementing OpenStack policy, catalogue, token, and authentication to user and service interactions, the OpenStack identity serves as a single point of integration. Without Keystone, there would be no service compliance, and as a result, end users would not have access to cloud products. It is a necessary service for the creation of a simple cloud architecture.

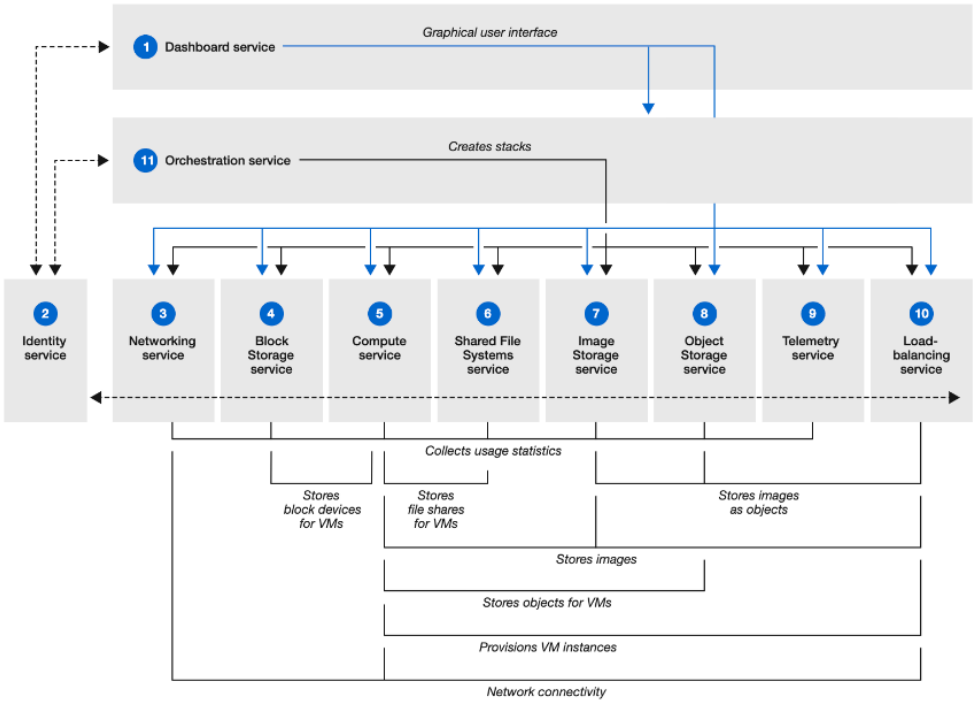

Figure 8. high-level overview of some OpenStack core services and their relationship.

Glance (Image Service): It is the OpenStack Image service, a method for searching and locating virtual machine (VM) images. It offers services for finding, logging in, and obtaining virtual images via an API that enables querying of VM image metadata, cataloguing, and managing enormous libraries of server images. For the construction of a fundamental cloud architecture, it is a necessary service.

Swift (Object Storage Service): Swift is a distributed, eventually consistent blob/object store. Swift is the name of the OpenStack object store project, which offers cloud storage software and a standard API to enable us to retrieve and store massive amounts of data. It was built for scalability and has since been updated for concurrency, availability, and durability throughout the entire data set. Unstructured data that can grow unrestrictedly is best stored in object storage.

Horizon (Dashboard Service): The OpenStack dashboard is a modular web application that interacts with all other services' public APIs to provide a user interface for managing cloud infrastructure. It is a necessary service for the creation of a simple cloud architecture. The API abstraction set for various Core OpenStack projects is used by the Horizon application to allow a stable and dependable collection of reusable methods for developers. The OpenStack Horizon developers don't need to be experts in all of the OpenStack project's APIs because of these abstractions.

Mistral (Workflow Service): The OpenStack service that manages workflows is called Mistral. Usually, the user creates the workflow using YAML as the language. It uses the REST API to upload the workflow definition to Mistral. Following that, the user can manually start the workflow using a similar API. Additionally, it sets up the trigger so that a select few events can initiate the workflow.

Ceilometer (Telemetry Service): For various billing systems, OpenStack Ceilometer (Telemetry) provides a Single Point of Contact, facilitating each counter needed to create consumer billing around every present and future OpenStack component. The counter delivery can be tracked and audited. The counter ought to be easily expandable to support additional initiatives. Additionally, the agents responsible for data collecting must be kept apart from the system as a whole.

Heat (Orchestration Service): Heat can be expressed as a service for orchestrating more than one fusion cloud application with templates by CloudFormation adaptable Query API and OpenStack-native REST API. HEAT orchestrates the infrastructure resources for an application or virtualized network function based on templates in the form of text files that can be treated like code. HEAT has a Query API that works with the Cloud as well as a REST API that is native to OpenStack. In order to incorporate a scaling group as a resource in a template, HEAT additionally offers an auto-scaling service that interfaces with the OpenStack Telemetry services.

Sahara (Elastic map-reduce Service): Sahara is a part that makes it simple and quick to provision Hadoop clusters. Numerous users will specify different characteristics, including the Hadoop version number, node flavour information (RAM and CPU settings, storage space specification), cluster topology type, and more. Sahara increases the cluster more quickly after each parameter has been offered by any user. Sahara also provides a way to scale an existing Hadoop cluster by removing and adding worker nodes as needed.

Barbican (Key Manager Service): Barbican is the REST API developed for the management, provisioning, and secure storage of secrets. Barbican is focused on being helpful for each environment including huge ephemeral Clouds.

Placement (Shared Placement Service): PLACEMENT is an OpenStack service that offers an HTTP API for tracking cloud resource inventories and usages to assist other OpenStack services with efficient resource management and allocation. The Placement service is provided with Nova packages in the majority of OpenStack releases.

Aodh (Rule-based Alarm Actions): This alarming service enables the ability to trigger actions based on defined rules against metric or event data collected by Ceilometer or Gnocchi.

Ironic (Bare Metal Service): Another one of OpenStack's projects is Ironic. In place of virtual machines, it plans bare-metal machines. Ironic began as an independent project that was forked through the driver of Nova Bare Metal. The plugin set and bare-metal hypervisor API that worked with different bare-metal hypervisors was the finest concept. Ironic supports and might be developed with vendor-specific plugins for providing extra functionality, however it will use IPMI and PXE in combination for turning machines on and off as well as for provisioning them.

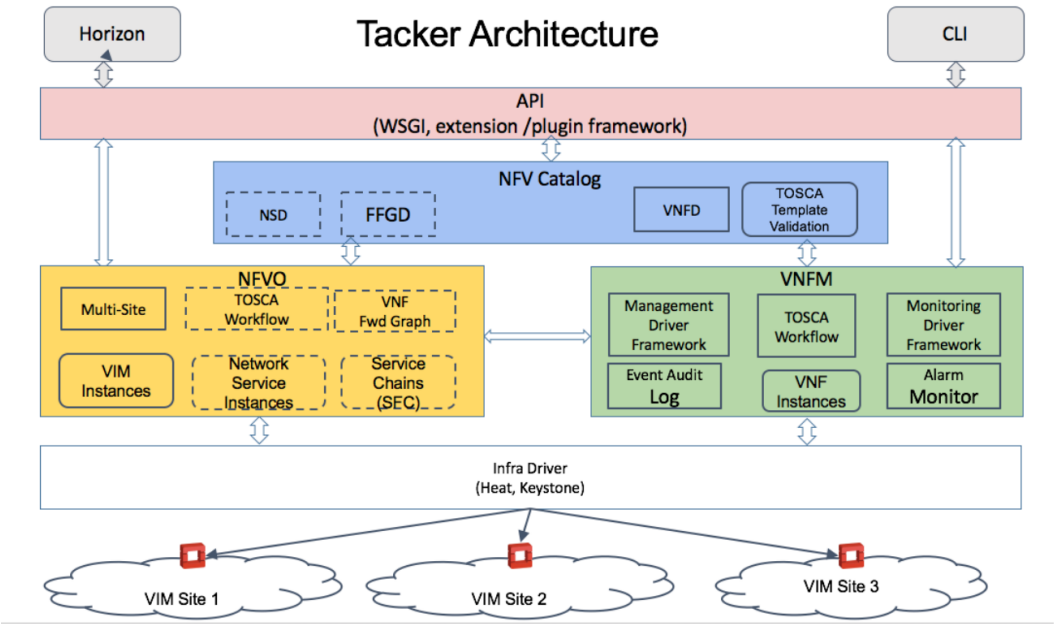

Figure 9. Tacker Architecture

Tacker (NFV Orchestration Service): Tacker is an official OpenStack project building a Generic VNF Manager (VNFM) and an NFV Orchestrator (NFVO) to deploy and operate Network Services and Virtual Network Functions (VNFs) on an NFV infrastructure platform like OpenStack. It is based on ETSI MANO Architectural Framework and provides a functional stack to Orchestrate Network Services end-to-end using VNFs. Figure 9[44] shows the architecture of tacker.
Three main components of Tacker are:

- **NFV Catalog:** VNF Descriptors
  Network Services Descriptors
  VNF Forwarding Graph Descriptors
- **VNFM:** Basic life-cycle of VNF (create/update/delete)
  Enhanced platform-aware (EPA) placement of high-performance NFV workloads
  Health monitoring of deployed VNFs
  Auto Healing / Auto Scaling VNFs based on Policy
  Facilitate initial configuration of VNF
- **NFVO**: Templatized end-to-end Network Service deployment using decomposed VNFs
  VNF placement policy – ensure efficient placement of VNFs
  VNFs connected using an SFC - described in a VNF Forwarding Graph Descriptor
  VIM Resource Checks and Resource Allocation
  Orchestrate VNFs across Multiple VIMs and Multiple Sites (POPs)

4.2. Mininet

A tool for software-defined networks is called Mininet. It is used to visualize the switches and usage of software-defined networks in a virtualized environment and is an emulator of a network. Additionally, it is used to evaluate OpenFlow-compatible and software-defined network hardware. OpenFlow switches are utilized in Mininet. Standard Linux network software is used by mininet hosts, and its switches support OpenFlow for Software-Defined Networking and very flexible custom routing.

In order to test, experiment with, and learn about software-defined networks, Mininet is primarily used as a learning tool. Mininet enables all activities that would benefit from having a full experimental network on a laptop or other PC, including research, development, learning, prototyping, testing, and debugging. Mininet is used because it is quick and enables us to design topologies that are specific to our needs. Using it is also pretty simple.
Process-based virtualization and network namespaces, capabilities present in modern Linux kernels, are used by Mininet to build virtual networks. Any code that would typically run on a Linux server (such as a web server or client program) should work just fine within a Mininet "Host" since hosts in Mininet are mimicked as bash processes operating in a network namespace. With its own private network interface, the Mininet "Host" will only be able to observe its own processes. Software-based switches, such as Open vSwitch or the OpenFlow reference switch, are used in Mininet. Links are virtual ethernet pairs that connect our emulated switches to emulated hosts and reside in the Linux kernel (processes).

Some of the key features of mininet are:

- offers a straightforward and affordable network testbed for creating OpenFlow applications.
- allows many developers to independently work on the same topology at the same time.
- allows for complicated topology testing without requiring the physical wiring of a network
- Contains a topology-aware and OpenFlow-aware CLI for troubleshooting and executing network-wide tests.
- supports reproducible and simple-to-package system-level regression testing.
- provides a basic set of parametrized topologies and supports arbitrary custom topologies.
- is programming-free and usable right out of the box, yet
- also offers a simple and expandable Python API for building and testing networks.

Mininet offers a simple method for obtaining proper system behavior (and, to the extent supported by your hardware, performance) and for topology experimentation.
Real code is executed on mininet networks, including the genuine Linux kernel and network stack as well as common Unix/Linux network applications (including any kernel extensions which is may be available, as long as they are compatible with network namespaces.) Because of this, the code developed and tested on Mininet, for an OpenFlow controller, modified switch, or host, can move to a real system with minimal changes, for real-world testing, performance evaluation, and deployment. Importantly this means that a design that works in Mininet can usually move directly to hardware switches for line-rate packet forwarding.

Chapter 5:

Implementation of NFV

5. IMPLEMENTATION OF NFV

In lab we have implemented Network Function Virtualization using both OpenStack and mininet.

5.1. NFV using OpenStack

From the following picture we can see how the OpenStack services work in concert to facilitate the deployment of an NFV environment in accordance with the functions played by the functional elements of the ETSI NFV architecture. OpenStack service tacker is used to address NFV Orchestration and VNF Manager. Figure 10 shows the architecture of NFV mapped with OpenStack services

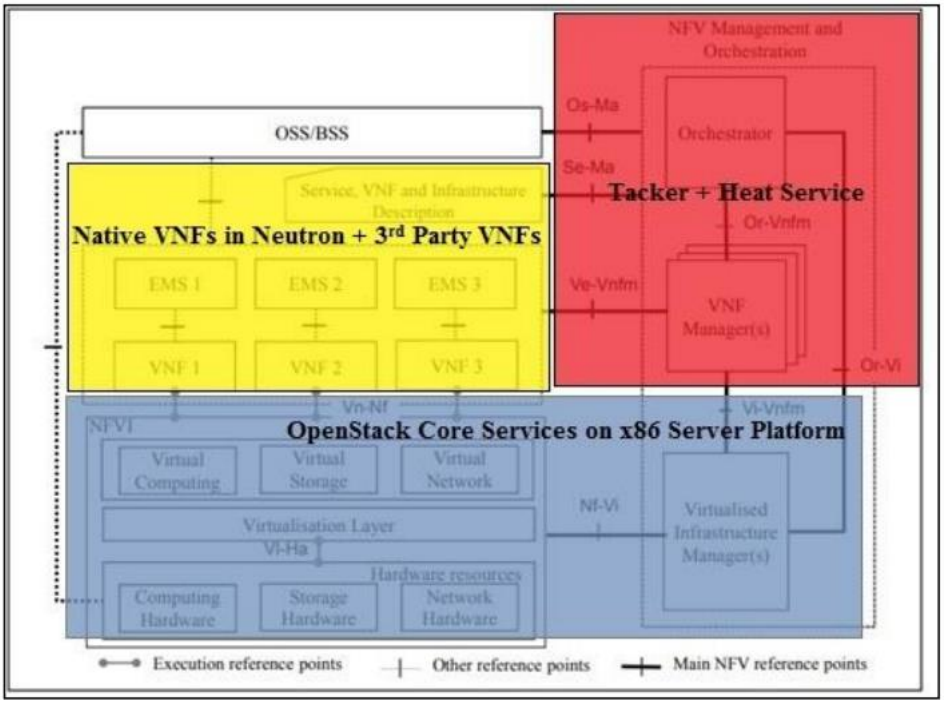

Figure 10. OpenStack Services for NFV

The OpenStack Core services, including Nova, Neutron, Cinder, Keystone, Glance, and Placement, function as the VIM, while the NFVI is made up of regular x86 servers whose physical resources, including computing hardware, storage hardware, and network hardware, can be virtualized into virtual resources using a virtualization tool like proxmox or KVM.

In order to assist with VNF onboarding, VNF lifecycle management, global resource management and orchestration, and validation and authorization of NFVI resource requests, the OpenStack services Tacker and Heat serve as the NFV-Orchestrator (NFVO) and VNF-Manager (VNFM), respectively.

Native VNFs such as firewall and load balancer in OpenStack Neutron service and other 3rd party VNFs provide the required network function required by consumers on their networks, without the NFV provider having to assign separate physical devices at each consumer site.

Figure 11 shows the deployed stack we have used to build the NFV environment using OpenStack.

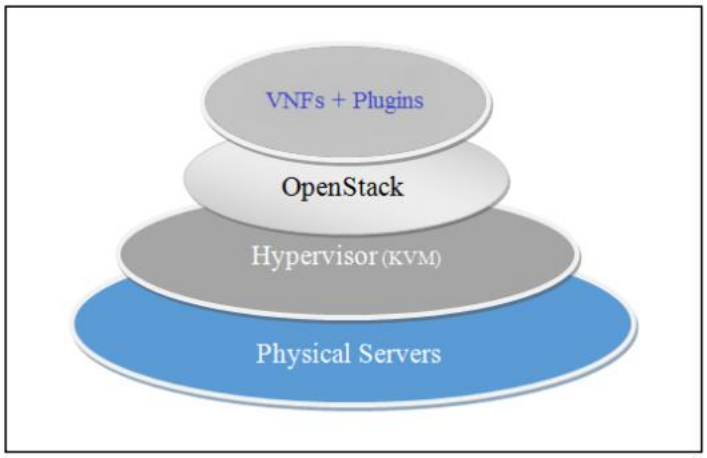

Figure 11: NFV Environment setup

- **VNF + Plug-in provider:** Native VNFs (Userdata, Hello world and Firewall) in Neutron Service of OpenStack and some other 3rd party VNFs onboarded using VNFM template.

  Open-source and easily available for Linux Environments.

  Other available VNFS: Frontinet VNFs, Nokie VNFs

- **OpenStack:** Selected Version: Yoga

  Reason - It's free and open-source. It is easily available on Ubuntu and offers a large range of services for configuring an NFV environment. The OpenStack Tacker and Heat Services serve as the NFV Orchestrator and VNF Manager, respectively, while the Nova Service serves as the VIM and Neutron service serves the network management.

  Other NFV Platforms – Juniper, VMware, Cisco, AWS.

- **Hypervisor:** Selected Hypervisor: proxmox (KVM-Based) Reason - Free, Open-Source. Other available Hypervisors – VirtualBox, VMware.

- **Physical Servers:** HP Proliant G8 Server with 16-core processor, 16 GB RAM and 100 GB HDD as minimum system requirements

  A single server as the controller node is sufficient for an all-in-one or single node arrangement.

  Additional Compute and Storage Servers are required for Cluster/Multi-node Deployment based on business requirements.

5.1.1. Implementation of OpenStack with devstack

In lab we have implemented NFV using a single all-in-one node which acts as the controller, compute and storage node.

- To implement OpenStack at first, we have created a VM in proxmox with 16GB RAM and 100GB memory, with 16cores and Bridge network adapter. Proxmox Virtual Environment (Proxmox VE or PVE) is an open-source software server for managing virtualization. It is a hosted Type-2 hypervisor that supports the use of Linux and Windows operating systems on x64 hardware. It is a Linux distribution based on Debian that uses an Ubuntu LTS kernel that has been tweaked, and it enables the deployment and management of virtual machines and containers.

- Installed Ubuntu 20.04 LTS DESKTOP OS on the said VM.

- ***sudo apt-get update*** it fetches the latest version of the package list from your distro's software repository, and any third-party repositories you may have configured. In other words, it'll figure out what the latest version of each package and dependency is, but will not actually download or install any of those updates.

- ***sudo apt-get install build-essential*** The build-essential package actually belongs to Debian. It is not a piece of software in itself. It contains a list of packages that are required to create a Debian package (deb). These packages are libc, gcc, g++, make, dpkg-dev etc. The build-essential package contains those required packages as dependencies, so when we install build-essential, we install all those packages in one single command.

- ***sudo apt-get install net-tools*** this package includes the important tools for controlling the network subsystem of the Linux kernel. This includes arp, ifconfig, netstat, rarp, nameif and route. Additionally, this package contains utilities relating to particular network hardware types (plipconfig, slattach, mii-tool) and advanced aspects of IP configuration (iptunnel, ipmaddr).

- ***sudo apt-get install python3-pip*** Pip is a package management system that simplifies installation and management of software packages written in Python such as those found in the Python Package Index (PyPI).

- ***pip install -U os-testr*** A testr wrapper to provide functionality for OpenStack projects.

- ***sudo apt-get install autoconf automake gdb git libffi-dev zlib1g-dev libssl-dev -y*** Autoconf is a GPLv2-licensed package of macros that produces a shell script "configure" to generate a Makefile and config header for a project. Automake is a tool for automatically generating `Makefile.in's from files called `Makefile.am'. Tool for generating GNU Standards-compliant Makefiles. GDB is a source-level debugger, capable of breaking programs at any specific line, displaying variable values, and determining where errors occurred. Currently, gdb supports C, C++, D, Objective-C,

Fortran, Java, OpenCL C, Pascal, assembly, Modula-2, and Ada. Git is popular version control system designed to handle very large projects with speed and efficiency; it is used for many high-profile open-source projects, most notably the Linux kernel. fast, scalable, distributed revision control system. libffi-dev package contains the headers and static library files necessary for building programs which use libffi. zlib is a library implementing the deflate compression method found in gzip and PKZIP. This package includes the development support files. libssl-dev package is part of the OpenSSL project's implementation of the SSL and TLS cryptographic protocols for secure communication over the Internet.

- ***sudo useradd -s /bin/bash -d /opt/stack -m*** *stack* This command creates a new user named stack to the system. The user will access the root opt directory.

- ***sudo chmod +x /opt/stack*** It provides the execute permission to the directory.

- ***echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack*** No root password will be needed for the user stack.

- ***sudo su - stack*** This gives stack user as sudo user privilege also all the environment variables will be sourced to the user, stack.

- ***git clone https://opendev.org/openstack/devstack -b stable/yoga*** This will clone the devstack folder from the link and it will clone the stable yoga branch.

- ***cd devstack*** Change the current directory to the devstack folder.

- ***git branch*** OpenStack version can be checked by this command.

- ***nano local.conf*** Create local.conf file with all the credentials and the services that are needed to be installed.

The following local.conf file contains has been used in lab to install OpenStack via devstack

```
[[local|localrc]]
############################################################
# Customize the following HOST_IP based on your installation
############################################################
IP_VERSION=4
HOST_IP=192.168.0.150
Q_FLOATING_ALLOCATION_POOL=start=192.168.0.151,end=192.168.0.199
FLOATING_RANGE=192.168.0.0/24
PUBLIC_NETWORK_GATEWAY=192.168.0.1
ADMIN_PASSWORD=devstack
MYSQL_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
```

```
SERVICE_PASSWORD=$ADMIN_PASSWORD
SERVICE_TOKEN=$ADMIN_PASSWORD

##################################################################
# Customize the following section based on your installation
##################################################################

# Pip
PIP_USE_MIRRORS=False
USE_GET_PIP=1

#OFFLINE=False
#RECLONE=True

# Logging
LOGFILE=$DEST/logs/stack.sh.log
VERBOSE=True
ENABLE_DEBUG_LOG_LEVEL=True
ENABLE_VERBOSE_LOG_LEVEL=True

# Neutron ML2 with OpenVSwitch
Q_PLUGIN=ml2
Q_AGENT=ovn

# Disable security groups
LIBVIRT_FIREWALL_DRIVER=nova.virt.firewall.NoopFirewallDriver

# Enable heat, networking-sfc, barbican and mistral
enable_plugin heat https://opendev.org/openstack/heat master
enable_plugin networking-sfc https://opendev.org/openstack/networking-sfc master
enable_plugin barbican https://opendev.org/openstack/barbican master
enable_plugin mistral https://opendev.org/openstack/mistral master

# Ceilometer
#CEILOMETER_PIPELINE_INTERVAL=300
CEILOMETER_EVENT_ALARM=True
enable_plugin ceilometer https://opendev.org/openstack/ceilometer master
enable_plugin aodh https://opendev.org/openstack/aodh master

# Blazar
enable_plugin blazar https://github.com/openstack/blazar.git master
```

```
# Fenix
enable_plugin fenix https://opendev.org/x/fenix.git master

# Tacker
enable_plugin tacker https://opendev.org/openstack/tacker master

enable_service n-novnc
enable_service n-cauth

disable_service tempest
disable_service etcd3

[[post-config|/etc/neutron/dhcp_agent.ini]]
[DEFAULT]
enable_isolated_metadata = True
```

- ***./stack.sh*** This executes stack.sh file. ``stack.sh`` is a shell script and an opinionated OpenStack developer installation. It installs and configures various combinations of OpenStack services like Ceilometer, Cinder, Glance, Heat, Horizon, Keystone, Nova. local.conf file is sources by stack.sh

- This script takes 45-50mins to run and may end in error due to unavailability of net issue. If it ends in error, we need to run ***./unstack.sh*** which stops all the process that started by stack.sh and then run **./stack.sh** again.

- After the successful installation we can access dashboard by going to "***http://<host_ip_address>/dashboard".*** Figure 12 shows the OpenStack dashboard.

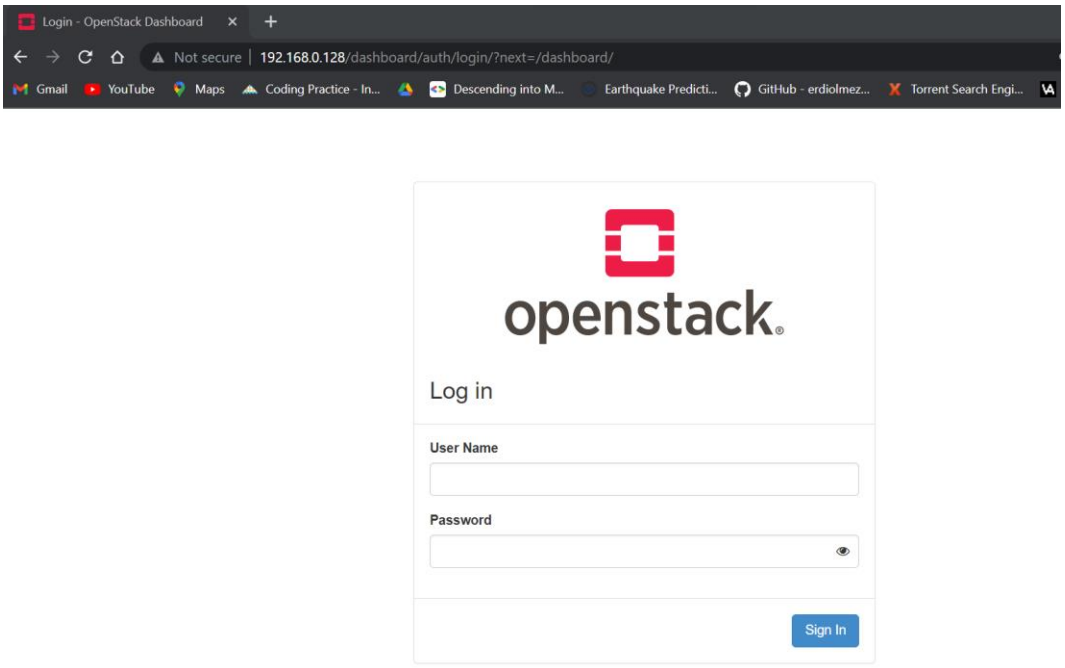

Figure 12. OpenStack dashboard login page

Now we can login with the password given in local.conf file. Figure 13 shows the dashboard after logging in.

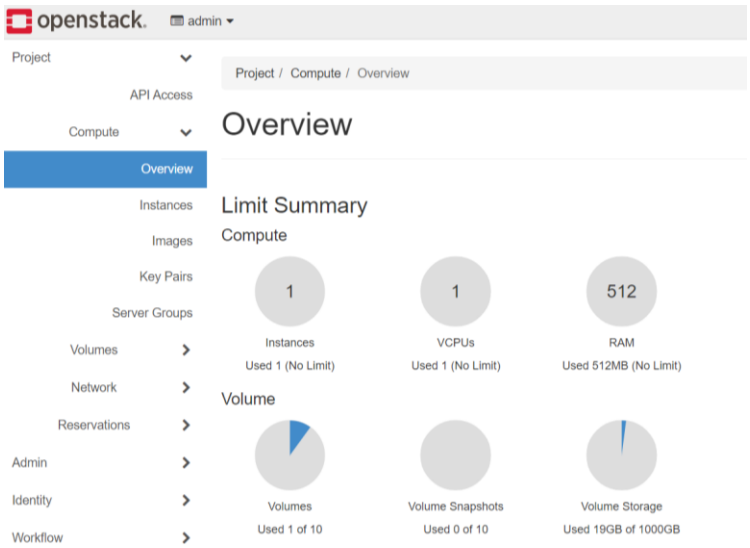

Figure 13: OpenStack dashboard

To start openstack after system restart, we need to use the following commands:

- ***sudo su - stack*** This gives stack user as sudo user privilege also all the environment variables will be sourced to the user, stack.
- ***cd devstack*** Change the current directory to the devstack folder.
- ***source openrc admin admin*** it configures login credentials and sources admin project

5.1.2. Creating an instance in OpenStack

At first, we need to go to Project → Compute → Instances. Figure 14 shows the OpenStack instance page.

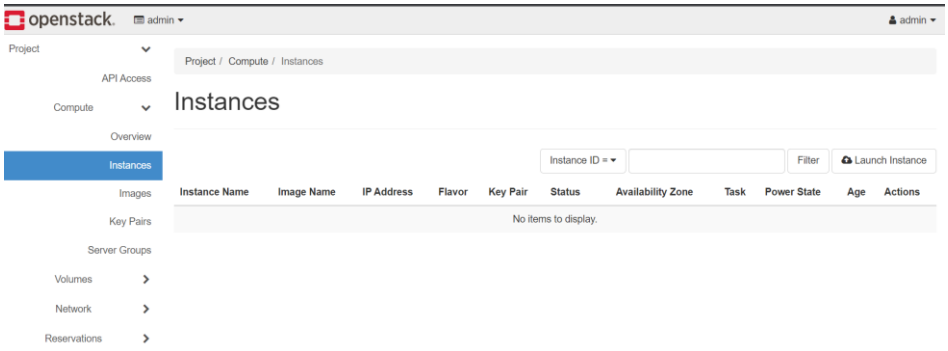

Figure 14: OpenStack Instance page

Click "Launch Instance". Then insert the name of the Instance (eg. "test") and click Next button. Figure 15 shows the OpenStack launch instance page.

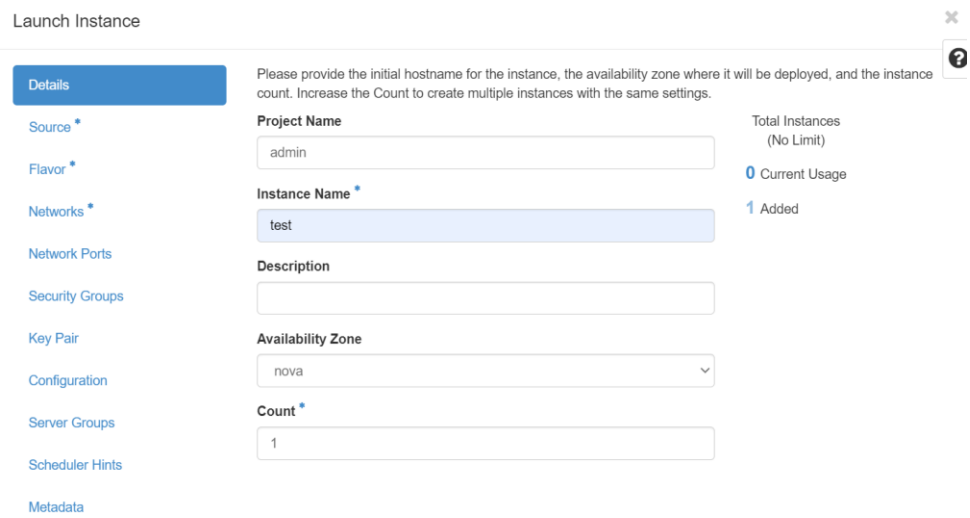

Figure 15: OpenStack launch instance page

In the next pages, Select the desired image (eg. "Ubuntu 16.04 LTS"), flavour and networks. Then launch the instance. Figure 16 shows the launched instance details.

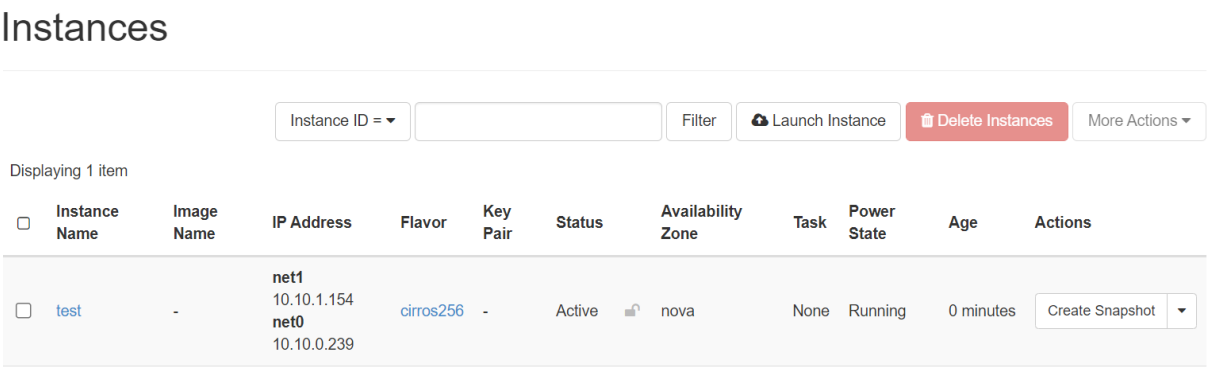

Figure 16: Created instance details

Now we can go to console and access the console of the newly created Instance. Figure 17 shows the console of the newly created instance.

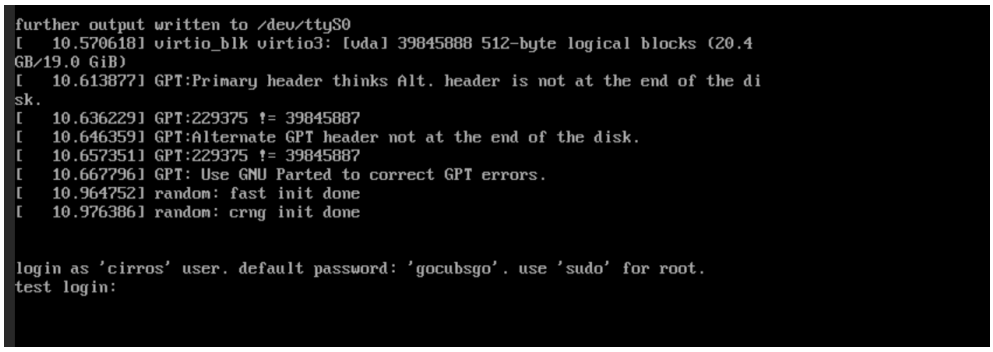

Figure 17: Console of the newly created instance

The internal flow of creating a VM using OpenStack

- Step 1: Using the REST API, the Horizon Dashboard or OpenStack CLI obtains user credentials and authenticates with the identity service. The identity service (Keystone) authenticates the user using the user credentials and then generates and returns an auth-token, which is used to submit requests to other components via REST-Call.
- Step 2: The Dashboard or OpenStack CLI translates the launch instance or nova boot command's new instance request to a REST API request and sends it to nova-api.
- Step 3: The nova-api service then receives the request and forwards it to the identity service (Keystone) for validation of the auth-token and access permission. The Keystone service validates the token and forwards the revised authentication headers with roles and permissions.
- Step 4: After receiving the response from keystone, nova-api checks for incompatibilities with nova-database before creating the initial database entry for the new instance or VM.
- Step 5: Nova-api sends a rpc.call request to nova-scheduler, expecting to get an updated instance entry with the supplied host id.
- Step 6: Nova-scheduler now selects the request from the queue.
- Step 7: Nova-scheduler communicates with nova-database to locate an appropriate host using the filtering and weighing mechanism.
- after filtering and weighing, nova-scheduler returns the updated instance entry with the appropriate host ID. Then nova-scheduler sends the rpc.cast request to nova compute to launch an instance on

the appropriate host.

- Step 8: Nova-compute selects the request from the queue and sends the rpc.call request to nova-conductor to obtain information about the VM or instance such as the host id and flavor (RAM,CPU and Disk)
- Step 9: Nova-conductor retrieves the request from the queue and communicates with the nova-database; nova-conductor now obtains the instance information. The instance information is selected from the queue by nova-compute.
- Step 10: Nova-compute connects to glance-api through REST using the auth-token, and then uses the image id to obtain the image URI from image service and load the image from image storage.
- Step 11: After glance-api validates the auth-token with keystone, nova-compute retrieves the image metadata.

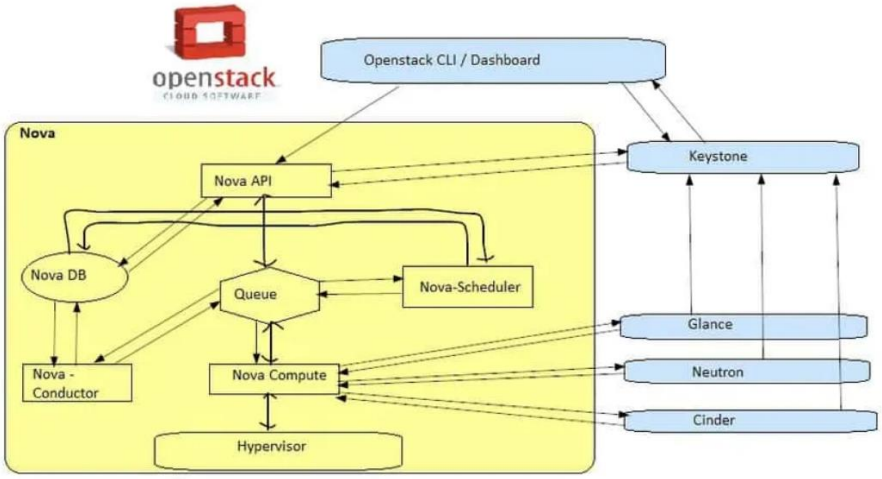

Figure 18: Internal flow of creating an instance

- Step 12: Nova-compute makes the REST-call by giving the authentication token to the Network API (Neutron) in order to allocate and configure the network so that the vm receives an IP address.
- Step 13: The auth-token is validated with keystone by the neutron-server, and the network information is then retrieved by the nova-compute.
- Step 14: Nova-Compute calls the Volume API with the auth-token to attach the volume to the instance or VM.
- Step 15: The auth-token is validated with keystone by cinder-api, and the block storage information is obtained by nova-compute.
- Step 16: nova-compute generates data for the hypervisor driver, executes the request on the hypervisor via libvirt or API, and eventually creates a virtual machine on the hypervisor. We can see that VM in Dashboard as well as using the "nova list" command.

Figure 18[45] shows the full internal flow of creating a instance.

5.1.3. Creating VNFs using OpenStack

Before creating VNFs we need to create a Virtual Infrastructure Manager (VIM)

- ***cd tacker/tools*** We change the directory to tool folder in tacker directory.
- ***./gen_vim_config.sh*** gen_vim_config.sh generates the vim_config.yaml file in the same path

and initializes the environment variables. In Figure 19 we can see the output of executing gen_vim_config.sh file.

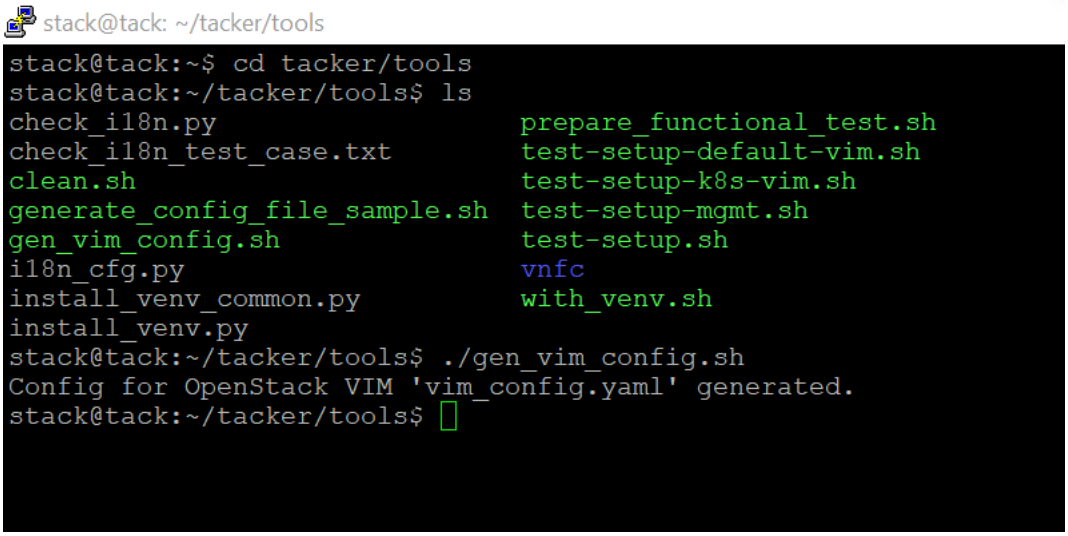

Figure 19: vim_config.yaml created

- ***cp vim_config.sh ~/devstack*** This command copies the vim_config.sh file to the devstack directory.
- ***cd devstack*** Changes the directory to devstack
- ***openstack vim register --config-file ./vim_config.yaml --is-default --description 'vim for nfv_user in nfv' openstack-nfv-vim*** This registers default OpenStack VIM. In Figure 20 we can see the details of the created VIM.

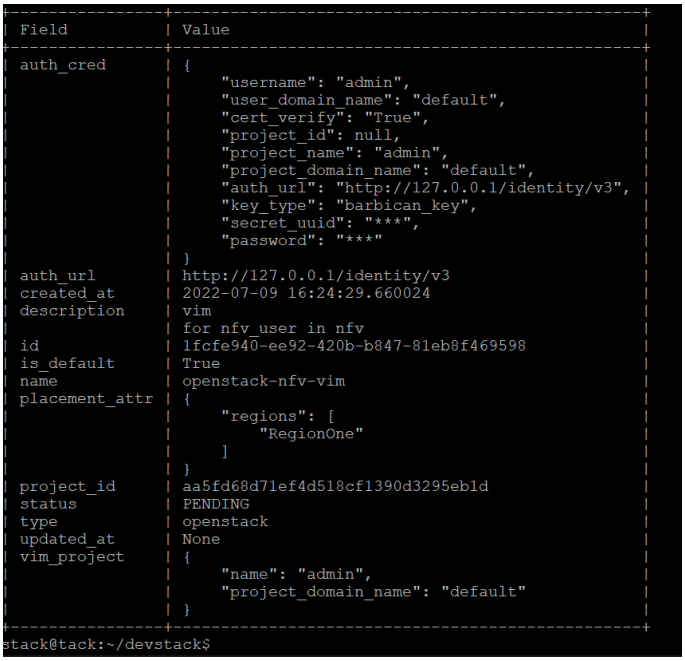

Figure 20: VIM created

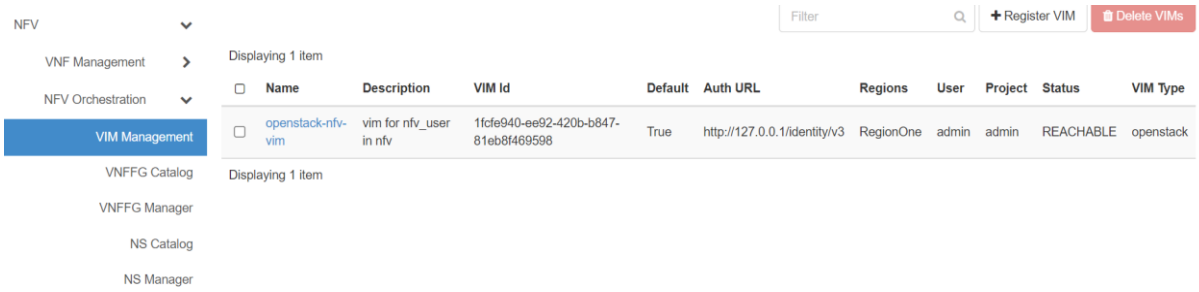

Figure 21: VIM can be seen in the dashboard

Now as VIM has been registered, we can deploy VNF. In Figure 21 we can see the deployed VIM in the dashboard. To deploy VNF we need to on board the VNF descriptor (VNFD) file first.

- ***cd tacker/samples/tosca-templates/vnfd*** Changes the directory to the folder containing vnfd files.
- ***cp tosca_vnfd_userdata.yaml ~/devstack*** We have taken tosca_vnfd_userdata.yaml file and copied it to devstack folder
- ***nano tosca-vnfd-userdata.yaml*** Open the vnfd file and change the network name to net0/net1 as these are accessible
- ***tacker vnfd-create --vnfd-file tosca-vnfd-userdata.yaml ud*** TOSCA VNFD templates can be onboarded to Tacker VNFD Catalog using this command. (tacker vnfd-create --vnfd-file <yaml file path> <VNFD-NAME>). In Figure 22 we can see that the VNFD has been created successfully.

```
Created a new vnfd:
+-----------------+--------------------------------------+
| Field           | Value                                |
+-----------------+--------------------------------------+
| created_at      | 2022-07-09 16:30:02.682106           |
| description     | Demo with user-data                  |
| id              | f8937cb2-67b0-4490-9249-8a1c7d8e9f03 |
| name            | ud                                   |
| service_types   | vnfd                                 |
| template_source | onboarded                            |
| tenant_id       | aa5fd68d71ef4d518cf1390d3295eb1d     |
| updated_at      |                                      |
+-----------------+--------------------------------------+
stack@tack:~/devstack$
```

Figure 22: VNFD created

- ***tacker vnf-create --vnfd-name ud udvnf*** This command creates the vnf. In Figure 23 we can see that the VNFD has been created successfully.

```
Created a new vnf:
+-----------------+--------------------------------------+
| Field           | Value                                |
+-----------------+--------------------------------------+
| created_at      | 2022-07-09 16:31:26.772692           |
| description     | Demo with user-data                  |
| error_reason    |                                      |
| id              | 0f747183-a2d8-4ca9-ba78-9c1811e82bc1 |
| instance_id     | e0ccf85a-f8a5-4249-be82-ae9bd2be6df7 |
| mgmt_ip_address |                                      |
| name            | udvnf                                |
| placement_attr  | {"vim_name": "openstack-nfv-vim"}    |
| status          | PENDING_CREATE                       |
| tenant_id       | aa5fd68d71ef4d518cf1390d3295eb1d     |
| updated_at      |                                      |
| vim_id          | 1fcfe940-ee92-420b-b847-81eb8f469598 |
| vnfd_id         | f8937cb2-67b0-4490-9249-8a1c7d8e9f03 |
+-----------------+--------------------------------------+
stack@tack:~/devstack$
```

Figure 23: VNF created

In Figure 24 we can see the instance created from the VNF in the dashboard.

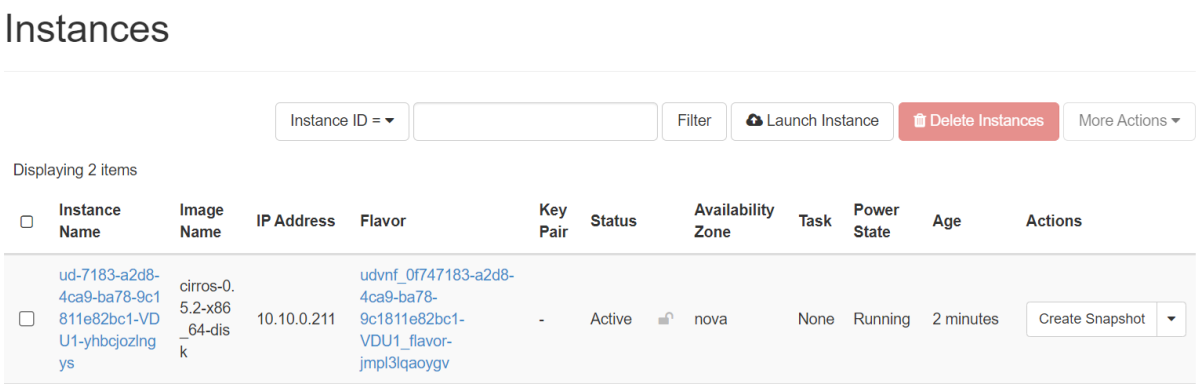

Figure 24: Instance from VNF got created

Figure 25 shows that we can check the VNF functionality through the VNF instance console.

```
sk.
[    8.616175] GPT:229375 != 2097151
[    8.623618] GPT:Alternate GPT header not at the end of the disk.
[    8.633479] GPT:229375 != 2097151
[    8.638336] GPT: Use GNU Parted to correct GPT errors.
[    8.878275] random: fast init done
[    8.891459] random: crng init done


login as 'cirros' user. default password: 'gocubsgo'. use 'sudo' for root.
ud-7183-a2d8-4ca9-ba78-9c1811e82bc1-vdu1-yhbcjozlngys login: cirros
Password:
$ cat /tmp/hostname
my hostname is ud-7183-a2d8-4ca9-ba78-9c1811e82bc1-vdu1-yhbcjozlngys
$ _
```

Figure 25: vnf name got printed in the hostname file

5.2. NFV using mininet: Mini-nfv

Mini-nfv is an NFV Orchestration framework with a general purpose VNF Manager for deploying and managing Virtual Network Functions (VNFs) and Network Services on Mininet. It is built using the ETSI MANO Architectural Framework. Mini-nfv maintains a Virtual Network Function's life cycle (VNF). Mini-nfv manages VNF setup, monitoring, scaling, and removal on Mininet.

Mini-nfv's purpose is to take the time-consuming effort of setting up a comprehensive service chaining environment or complicated infrastructure environments (for example, OpenStack) and focus on designing a specific VNF, prototyping, implementing an orchestration algorithm, or bespoke traffic steering. By utilizing mini-nfv and its parametrized OASIS TOSCA or ETSI NVF profiles, the NFV experimental world gains some amount of standardization, reproducibility, and replicability.

5.2.1. Architecture of Mini-nfv

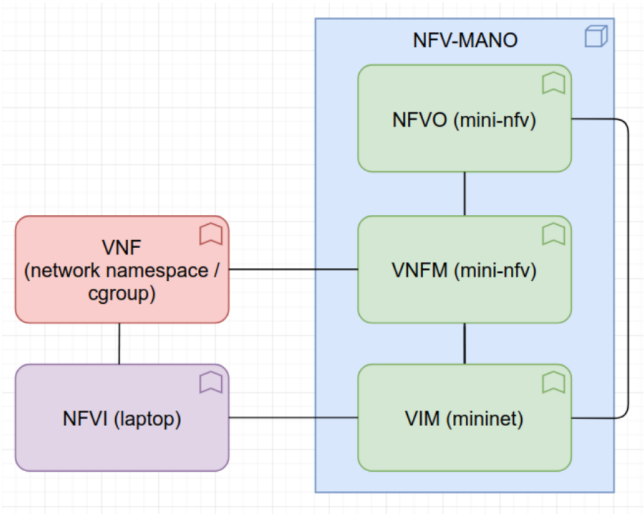

Figure 26: Functional architecture of Mini-nfv

VNF (network namespace/cgroup): VNF are deployed on top of virtual userspace in mini-nfv. In those individual userspaces, individual network stacks are created using network namespaces. Network namespaces allows for creation of virtual networking domain with its own set of interfaces, ip addresses, routing table etc. Network namespaces allows for isolating tenants as well as to have overlapping IP addresses.

NFVI (laptop): Network functions virtualization infrastructure (NFVI) is the layer who is responsible to handle hardware. This hosts all Storage, Compute & Network hardware & abstract same as virtual resources for consumption of virtual machines. Since we have deployed mini-nfv on top of our laptop so it will actually virtualise the underlying laptop hardware resources. Figure 26 illustrates the architecture of mini-nfv

Mini-nfv is in charge of controlling an NFV-based solution's virtualized infrastructure. Its operations include the following:

- Keeping track of the distribution of virtual resources to physical resources. This enables mininfv to manage and optimise the allocation, upgrade, release, and reclamation of NFVI resources.

- Assisting with VNF forwarding graph administration by organising virtual links, networks, subnets, and ports.

The following capabilities are expressly supported:

• In terms of the NFV catalogue:

- VNF Descriptor (VNFD): A template for defining a VNF's behavior and deployment metadata.
- VNF Forwarding Graph Descriptor (VNFFGD): Describe the chain and classifier that will be used to establish a path through a set of VNFs. Policy (traffic matching policy to flow through the path) and path are two of its properties (chain of VNFs and Connection Points).
- Network Services Descriptor: Specify the network service to be established, which may include VNFs and VNF forwarding graphs describing connections between NFs.
- Network Services Descriptor: Specify the network service to be established, which may include VNFs and VNF forwarding graphs describing connections between NFs.

• In terms of the VNF Manager:

- Basic VNF lifecycle (create/update/delete)
- Initial VNF configuration via cloud-init scripts

• In terms of the NFV Orchestrator:

- Decomposed VNFs were used to template end-to-end Network Service deployment.
- VNF placement policy, assuring optimal VNF placement.
- VNFs linked together by a Service Function Chain (SFC), as stated in a VNFFGD.
- Traffic from and to VNFs can be symmetrical or asymmetrical (the Tacker/OpenStack VIM driver currently only supports asymmetrical unidirectional traffic).
- A novel mirroring mode for monitoring-like VNFs in which traffic is duplicated rather than diverted.

5.2.2. Implementation of Mini-nfv

We have implemented mini-nfv on a VM with ubuntu 18.04 OS

- ***git clone*** ***https://github.com/josecastillolema/mini-nfv*** Clones mini-nfv repository to our system
- ***cd mini-nfv*** Changes the current directory to mini-nfv.
- ***$ pip install -r ./requirements.txt*** This command installs all the pre-requisites which are mentioned in the requirements.txt file.
- ***$ sudo systemctl stop openvswitch-testcontroller.****service* openvswitch-testcontroller must be stopped before running mini-nfv in standalone mode.
- **sudo ./mininfv.py** executes mininfv.py script, Figure 27 shows the output of executing

mininfv.py

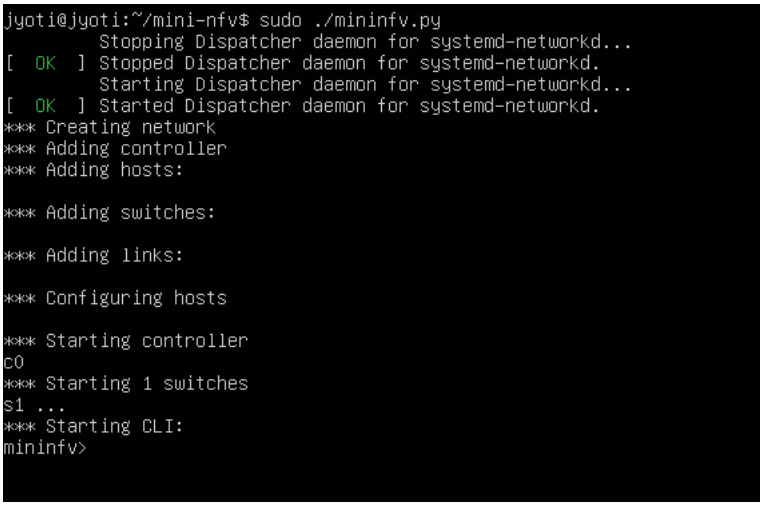

Figure 27: mini-nfv

- ***vnfd_create --vnfd-file samples/vnfd/tosca-vnfd-userdata.yaml vnfd-userdata*** This creates vnf descriptor from the vnf sample file tosca-vnfd-userdata.yaml

- ***vnf_create --vnfd-file samples/vnfd/tosca-vnfd-userdata.yaml vnfUD*** This creates the vnf names vnfUD

- ***nodes*** This shows the existing nodes in the network

- ***add_host h1 10.0.0.11/24*** This creates host h1 with the ip 10.0.0.11.

- ***add_host h2 10.0.0.12/24*** This creates host h1 with the ip 10.0.0.12.

- ***switch s1 start*** Start the switch s1. Figure 28 shows the process of VNF creation.

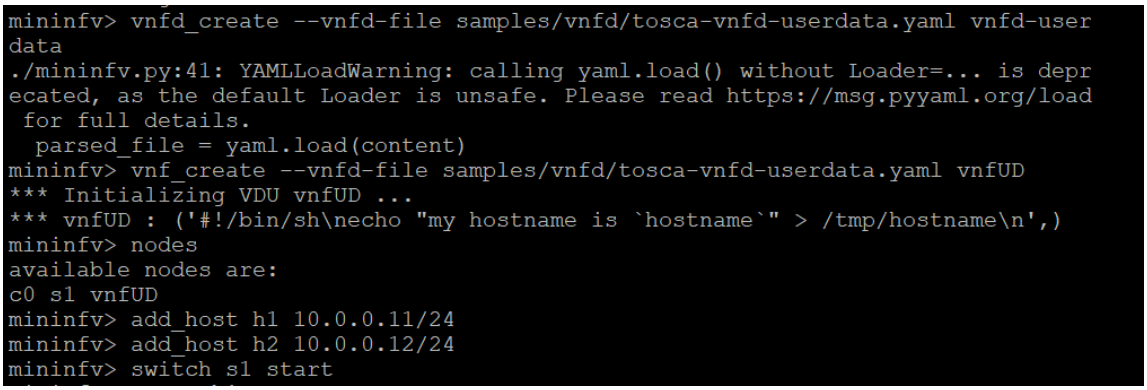

Figure 28: VNF creation in mini-nfv

- ***xterm vnfUD*** Now we can access the terminal of the VNF

Figure 29 shows the commands and steps to create VNF using mini-nfv.

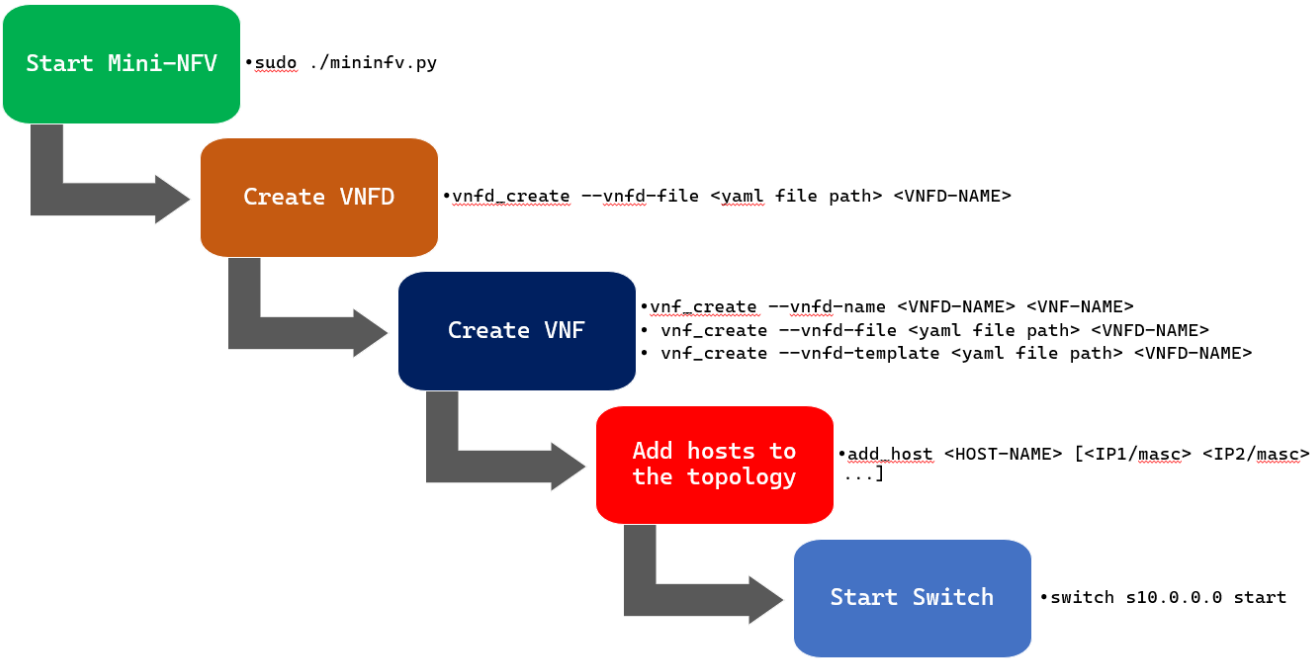

Figure 29: mini-nfv flow to create VNF

5.3. VNF Descriptors (VNFDs)

A VNFD is a deployment template that outlines the deployment and operating behavior requirements of a VNF. It also includes ETSI-defined Virtualized Deployment Units (VDUs), internal virtual link descriptors, external connection point descriptors, software image descriptors, and deployment flavor descriptors.

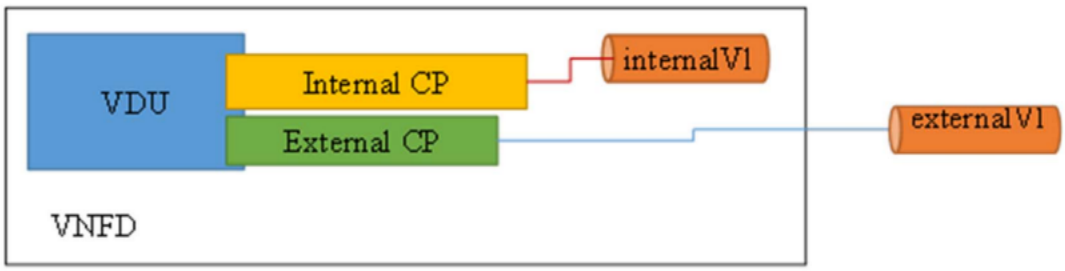

Figure 30: Overview of VNF descriptor

Figure 30 illustrates the overview of a VNFD.

A VNFD contains the following key bits of information:

- A Virtualisation Deployment Unit (VDU) is a construct that aids in the description of a VNF Component's deployment and operating behaviour (VNFC). A VNFC instance based on the VDU corresponds to a single virtualisation container (e.g. a VM). The term AVDU refers to the resources required to deploy and manage a VNFC's lifespan. A VDU contains internal Connection Point Descriptors (CPDs) that specify internal connection points that can be used to connect a VNFC to an internal virtual link or re-exposed as external connection points outside the VNF.
- External CPD: a VNF external connection point where either an internal connection point of a VDU is exposed as an external connection point or the external connection point is directly connected to an internal virtual link.
- • Internal VLD: specifies the resource needs for deploying and maintaining the lifecycle of virtual connections generated based on one or more VDUs.

Each VNFD template will have below fields:

- **tosca_definitions_version:** This defines the TOSCA definition version on which the template is based. The current version being tosca_simple_profile_for_nfv_1_0_0.
- **tosca_default_namespace:** This is optional. It mentions default namespace which includes schema, types version etc.
- **description:** A short description about the template.
- **metadata:** A name to be given to the template.
- **topology_template:** Describes the topology of the VNF under node_template field.
    - **node_template:** Describes node types of a VNF.
    - **VDU**: Describes properties and capabilities of Virtual Deployment Unit.
    - **CP**: Describes properties and capabilities of Connection Point.
    - **VL**: Describes properties and capabilities of Virtual Link.

Virtual Deployment Unit (VDU): Virtual Deployment Unit is a basic part of VNF. It is the VM that hosts the network function. A VDU can contain the following attributes:

- **Type:** tosca.nodes.nfv.VDU.Tacker
- **properties:** Describes the properties like image to be used in VDU, availability zone in which VDU is to be spawned, management driver to be used to manage the VDU, flavor describing physical properties for the VDU to be spawned, monitoring policies for the VDU, providing user data in form of custom commands to the VDU.
- **artifacts:** Specifies an image via a file or an external link. An image URL can be specified as artifacts. Tacker will specify the image location in HOT (Heat Template) and pass it to heat-api. Heat will then spawn the VDU with that image.
- **Capabilities:** Computational properties of a VDU are described as its capabilities. Allocated RAM size, allocated disk size, memory page size, number of CPUs, number of cores per CPU, number of threads per core can be specified.
- **user_data:** Custom commands to be run on VDU once it is spawned can be specified in a VNFD template as user data.

```
VDU1:
  type: tosca.nodes.nfv.VDU.Tacker
  artifacts:
    VNFImage:
      type: tosca.artifacts.Deployment.Image.VM
      file: http://download.cirros-cloud.net/0.5.2/cirros-0.5.2-x86_64-disk.img
  capabilities:
    nfv_compute:
      properties:
        disk_size: 10 GB
        mem_size: 2048 MB
        num_cpus: 2
        mem_page_size: small
        cpu_allocation:
          cpu_affinity: dedicated
          thread_count: 4
          core_count: 2
  properties:
    user_data_format: RAW
    user_data: |
      #!/bin/sh
      echo "Adding this line to demofile" > /tmp/demofile
```

Sample VDU of a VNFD

Connection Points (CP): Connection point is used to connect the internal virtual link or outside virtual link. Each connection point has to bind to a VDU. A CP always requires a virtual link and a virtual binding associated with it. A Connection Point can contain the following attributes:

- **Type:** tosca.nodes.nfv.CP.Tacker
- **properties:** Describes the properties like
  - anti spoofing protection: Indicates whether anti_spoof rule is enabled for the VNF or not. Applicable only when CP type is virtual NIC
  - management: Specifies whether the CP is accessible by the user or not.
  - Order: Uniquely numbered order of CP within a VDU. Must be provided when binding more than one CP to a VDU and ordering is required.
  - security_groups: List of security groups associated with the CP
  - mac_address
  - ip _address
- **virtualLink:** This is a required field. It states the VL node to connect to.
- **Virtualbinding:** This is a required field. It states the VDU node to connect to.

```
CP1:
  type: tosca.nodes.nfv.CP.Tacker
  properties:
```

```
      mac_address: fa:40:08:a0:de:0a
      ip_address: 10.10.1.12
      type: vnic
      anti_spoofing_protection: false
      management: true
      order: 0
      security_groups:
        - secgroup1
        - secgroup2
    requirements:
      - virtualLink:
          node: VL1
      - virtualBinding:
          node: VDU1
  CP2:
    type: tosca.nodes.nfv.CP.Tacker
    properties:
      type: vnic
      anti_spoofing_protection: false
      management: true
      order: 1
    requirements:
      - virtualLink:
          node: VL2
      - virtualBinding:
          node: VDU1
```

Sample CP of a VNFD

Virtual Links (VL): Virtual link provides connectivity between VDUs. It represents the logical virtual link entity. A Virtual Link can contain the following attributes:

- Type: tosca.nodes.nfv.VL
- properties: Describes the properties like
    - vendor: This is a required field. The vendor generating this VL
    - network_name: This is a required field. Name of the network to which VL is to be attached.

```
  VL1:
    type: tosca.nodes.nfv.VL
    properties:
      vendor: Tacker
      network_name: net-01
```

Sample VL of a VNFD

Parameterization allows for the ability to use a single VNFD to be deployed multiple times with different values for the VDU parameters provided at deploy time. In contrast, a non-parameterized VNFD has static values for the parameters that might limit the number of concurrent VNFs that can be deployed using a single VNFD. For example, deploying an instance of a non-parameterized template that has fixed IP addresses specified for network interface a second time without deleting the first instance of VNF would lead to an error.

Below is a general VNFD template which mentions all node types with all available options.

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0
description: Sample VNFD template mentioning possible values for each node.
metadata:
 template_name: sample-tosca-vnfd-template-guide
topology_template:
 node_templates:
  VDU:
    type: tosca.nodes.nfv.VDU.Tacker
    capabilities:
      nfv_compute:
        properties:
          mem_page_size: [small, large, any, custom]
          cpu_allocation:
            cpu_affinity: [shared, dedicated]
            thread_allocation: [avoid, separate, isolate, prefer]
            socket_count: any integer
            core_count: any integer
            thread_count: any integer
          numa_node_count: any integer
          numa_nodes:
            node0: [ id: >=0, vcpus: [host CPU numbers], mem_size: >= 0MB]
    properties:
      image: Image to be used in VDU
      flavor: Nova supported flavors
      availability_zone: available availability zone
      mem_size: in MB
      disk_size: in GB
      num_cpus: any integer
      metadata:
        entry_schema:
      config_drive: [true, false]
      monitoring_policy:
        name: [ping, noop, http-ping]
        parameters:
          monitoring_delay: delay time
```

```
        count: any integer
        interval: time to wait between monitoring
        timeout: monitoring timeout time
        actions:
          [failure: respawn, failure: terminate, failure: log]
        retry: Number of retries
        port: specific port number if any
    config: Configuring the VDU as per the network function requirements
    mgmt_driver: [default=noop]
    service_type: type of network service to be done by VDU
    user_data: custom commands to be executed on VDU
    user_data_format: format of the commands
    key_name: user key
  artifacts:
    VNFImage:
      type: tosca.artifacts.Deployment.Image.VM
      file: file to be used for image
CP:
  type: tosca.nodes.nfv.CP.Tacker
  properties:
    management: [true, false]
    anti_spoofing_protection: [true, false]
    type: [ sriov, vnic ]
    order: order of CP within a VDU
    security_groups: list of security groups
  requirements:
    - virtualLink:
        node: VL to link to
    - virtualBinding:
        node: VDU to bind to
VL:
  type: tosca.nodes.nfv.VL
  properties:
    network_name: name of network to attach to
    vendor: Tacker
```

Below is the sample VNFD file to get user data

```
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
  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
```

We have changed the userdata section to check if we can execute a python file through the VNF. As the userdata section contains shell script we have used ./filepath/abc.sh to execute a shell script which executes the python file. This will allow us to run any desired python script through the VNF.

Contains of VNFD file after the modification:

```
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
          sh /home/cdcju/Downloads/mininfv/samples/vnfd/abc.sh


    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
```

In the abc.sh file we have executed a python program named sum.

```
#!/bin/sh
 #echo "abc my hostname is `hostname`" > /tmp/hostname
python3 /home/mininfv/mini-nfv/samples/vnfd/sum2.py
```

In the python file we have done summation of two variables and saved the output in a file named text.txt

```
with open("/home/mininfv/mini-nfv/samples/vnfd/tese.txt","w") as file:
        a=5
        b=5
        c=a+b
        file.write(f'The sum is {c}')
```

We can also parse the VNFD file using yaml parser and select the desired attribute

```
#!/usr/bin/python3
import yaml
with open("/home/mininfv/mini-nfv/samples/vnfd/tosca-vnfd-userdata.yaml","r")
as file:
        prime_service = yaml.safe_load(file)
print (prime_service)
with open("/home/mininfv/mini-nfv/samples/vnfd/text.txt","w+") as file:
        #file.write(str(prime_service))
        file.write(str(prime_service['topology_template']['node_templates']
        ['VDU1']['type']))
```

In figure 31, we can see text.txt got generated with the VDU1 type.

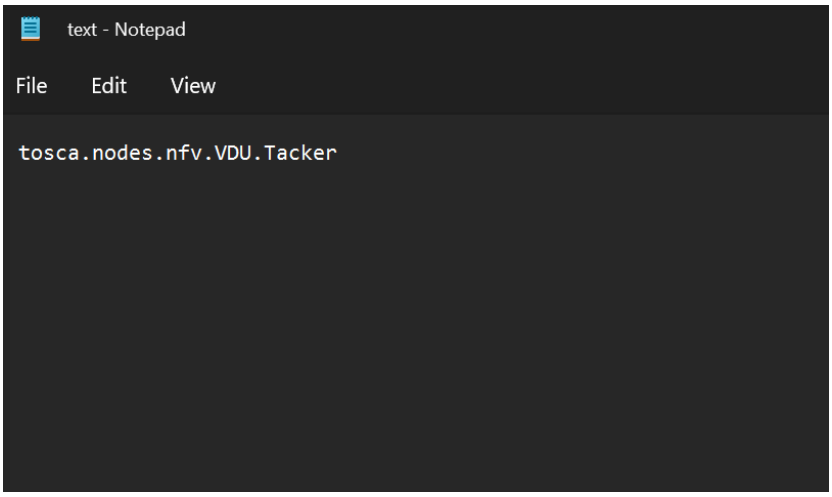

Figure 31: Contains of text.txt

5.4. Deployment of Packet Counting VNF

A VNF has been developed which will count the packets exchanged between the server and the client. To make the VNF work, at first a server instance and a client instance have been deployed in separate subnets. Server has been deployed at net0 and Client has been deployed at net1 subnet. Then the VNF has been deployed with two net interfaces, one for the subnet net0 and another for the subnet net1. Now a linear architecture has been created between client, VNF and the server. When the client pings the server, it passes through the VNF. Figure 32 shows the implemented linear architecture.

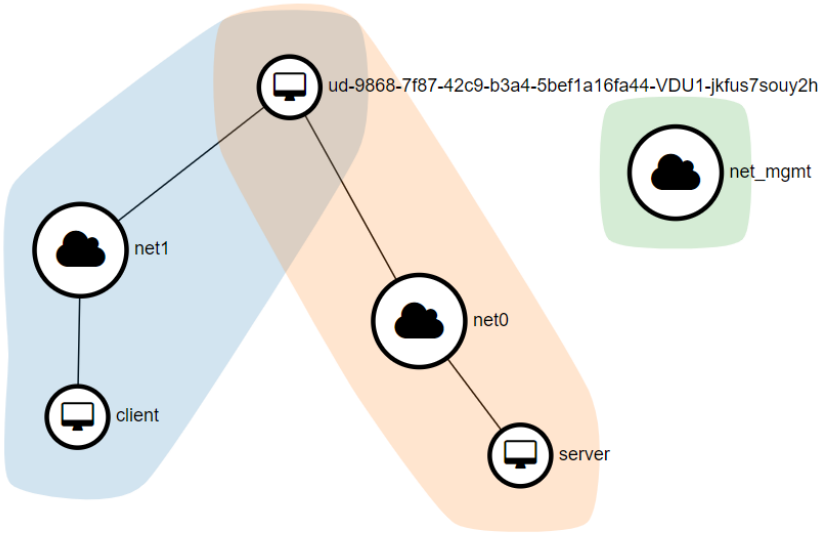

Figure 32: The implemented linear architecture between client, VNF and server

The VNF actually counts the packets coming from the client. When the packet count crosses a threshold, the VNF shuts down the port for a certain time period. After the said time period is over then the VNF turns the port up and sets the count to 0 so that the client is now able to ping the server again.

Steps to implement the packet counting VNF:

- Create two instances, server and client in separate subnets.

- Launch the VNF with two interfaces both assigned to different subnets. One interface is assigned to the client subnet and another interface is assigned to the server subnet. The VNFD configuration file is given below-

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0

description: Packet-counting

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: 16 GB
      properties:
        image: ubuntu_s
        config: |
          param0: key1
          param1: key2
        mgmt_driver: noop
        user_data_format: RAW
        user_data: |
          #!/bin/sh
          sh /tmp/packet.sh

    CP1:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
        management: true
        ip_address: 10.10.0.1
        order: 0
        anti_spoofing_protection: false
      requirements:
        - virtualLink:
            node: VL1
        - virtualBinding:
            node: VDU1
```

```
VL1:
  type: tosca.nodes.nfv.VL
  properties:
    network_name: net0
    vendor: Tacker

CP2:
  type: tosca.nodes.nfv.CP.Tacker
  properties:
    management: true
    ip_address: 10.10.1.1
    order: 0
    anti_spoofing_protection: false
  requirements:
    - virtualLink:
        node: VL2
    - virtualBinding:
        node: VDU1

VL2:
  type: tosca.nodes.nfv.VL
  properties:
    network_name: net1
    vendor: Tacker
```

The three instances can be seen in figure 33.

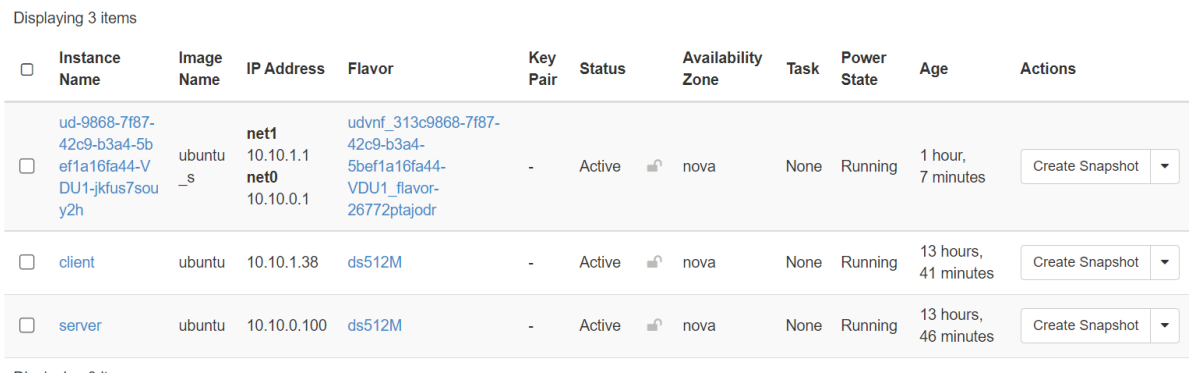

Figure 33: Three launched instances

This VNF calls the *packet.sh* file which in turn calls the *packet_counter.py* python file. The later contains the code to count the packets and does the needful. The *packet_counter.py* file is given below-

```
import os
import time
import datetime
while True:
    file = open("/sys/class/net/ens3/statistics/rx_packets", 'r')
    value = file.read()
    set_start=int(value)
    timestamp=datetime.datetime.now()
    file.close()
```

```
threshold=100
counter=0

os.system("echo fox | sudo -S ifconfig ens3 up")
while True:
    filex = open("/sys/class/net/ens32/statistics/rx_packets", 'r')
    c_value = filex.read()
    current_value=int(c_value)
    counter=current_value-set_start
    os.system("clear")
    print("\nPACKET COUNTING STARTED AT: {}".format(timestamp))
    print("\n\nPACKET RECEIVED: {}".format(counter))
    if counter > threshold:
        print("COUNTER HAS CROSSED {} PACKETS...".format(threshold))
        print("System is going to turn down the port for 10 secs...")
        os.system("echo fox | sudo -S ifconfig ens3 down ")
        time.sleep(10)
        break
```

- Check inside the VNF instance console if both the interfaces are up with the ***ifconfig*** command. If not, turn up the interface by using ***ifconfig ens4 up*** command.

- The command ***dhclient ens4*** will assign dhcp protocol to ens4. Now both the interfaces are up and can be seen using ***ifconfig*** command.

- In the client instance: ***ping 10.0.0.100*** This will ping the server from the client. Figure 34 shows the console for the client instance.
  In the server instance: use the ***tcpdump*** command. This will show the received packet details. Figure 36 shows the console for the server instance.
  Packet count can be seen in the VNF instance console. Figure 35 shows the VNF console.

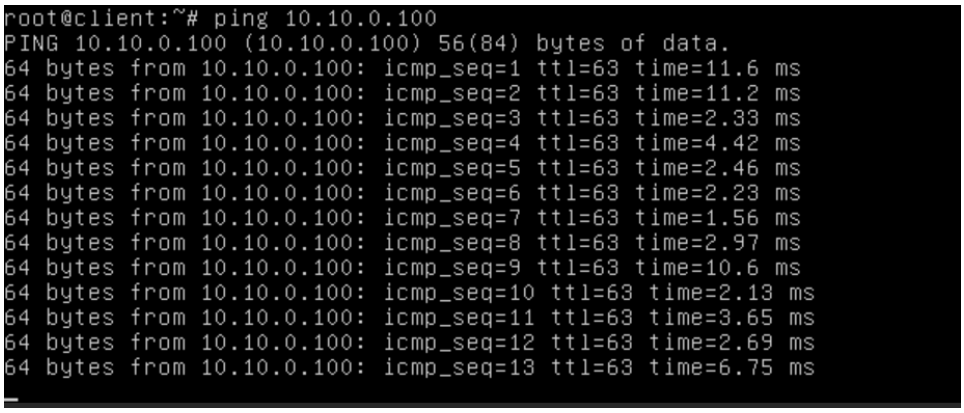

Figure 34: Client instance after ping started

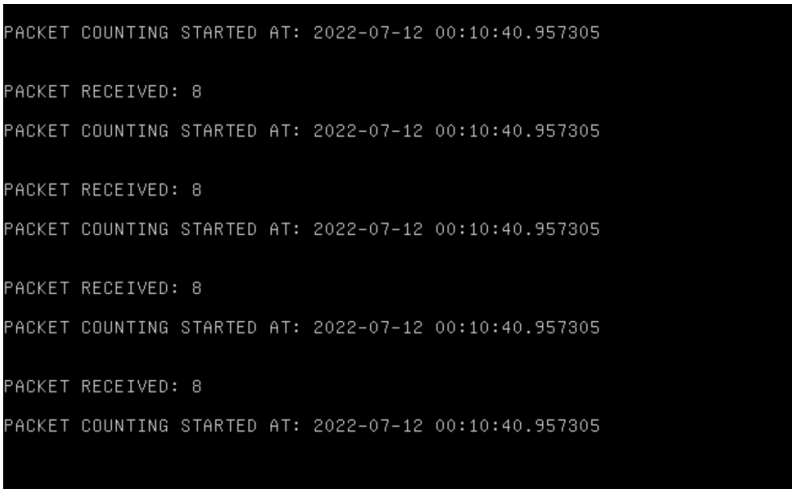

Figure 35: VNF instance showing packet count

Figure 36: Server instance showing packet details.

- When 100 packets have been transmitted, the VNF turns down the port which stops the transmission. We can see this in Figure 38. The Client gets the massage, "Destination host is unreachable" shown in Figure 37 and the server stops showing ICMP echo requests shown in figure 39.

Figure 37: Host unreachable massage in client instance

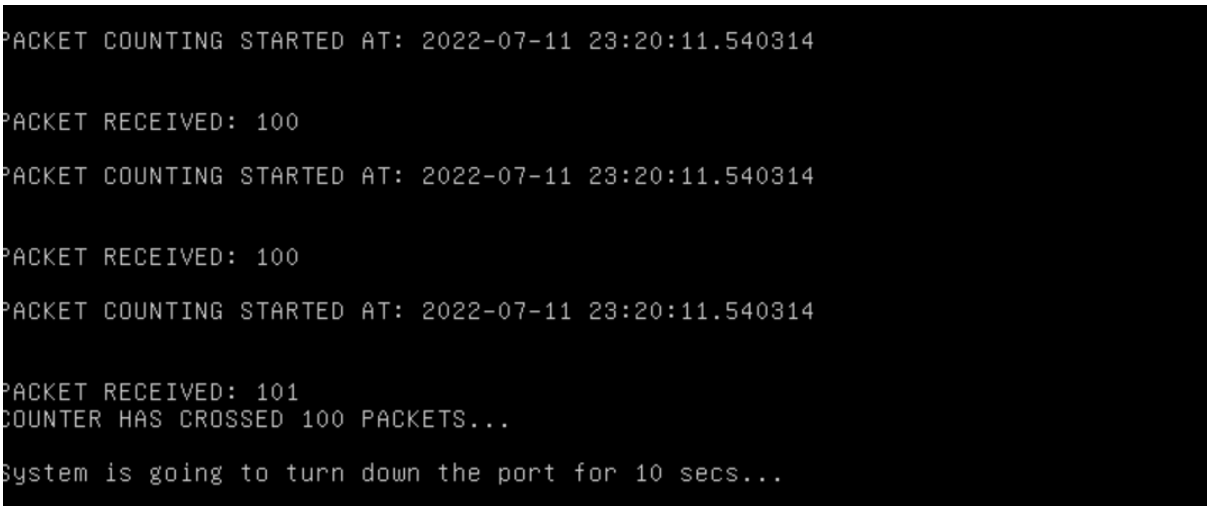

Figure 38: VNF instance after 100 packet count

Figure 39: Server instance

- After 10 seconds, the VNF turns up the port again and the packets resume transmission. In figure 40 we can see that the client is able to ping the server again. In figure 41 we can see the VNF packet count has been reset to 0 again. In figure 42 we can see that the server is showing the received icmp packet details.

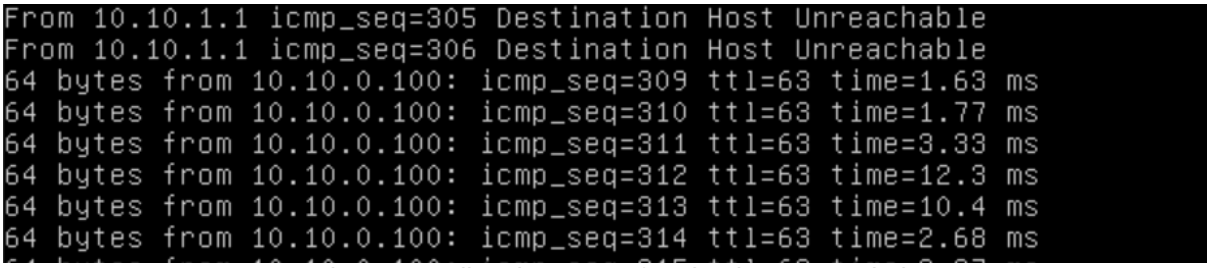

Figure 40: Client instance after the time-out period

```
PACKET RECEIVED: 1

PACKET COUNTING STARTED AT: 2022-07-11 23:26:21.686250


PACKET RECEIVED: 3

PACKET COUNTING STARTED AT: 2022-07-11 23:26:21.686250


PACKET RECEIVED: 3

PACKET COUNTING STARTED AT: 2022-07-11 23:26:21.686250


PACKET RECEIVED: 3

PACKET COUNTING STARTED AT: 2022-07-11 23:26:21.686250


PACKET RECEIVED: 3
```

Figure 41: The count has been reset in the VNF instance

```
15:25:07.600494 IP 10.10.1.38 > server: ICMP echo request, id 1616, seq 358, len
gth 64
15:25:07.600541 IP server > 10.10.1.38: ICMP echo reply, id 1616, seq 358, lengt
h 64
15:25:08.605320 IP 10.10.1.38 > server: ICMP echo request, id 1616, seq 359, len
gth 64
15:25:08.605363 IP server > 10.10.1.38: ICMP echo reply, id 1616, seq 359, lengt
h 64
15:25:09.604651 IP 10.10.1.38 > server: ICMP echo request, id 1616, seq 360, len
gth 64
15:25:09.604695 IP server > 10.10.1.38: ICMP echo reply, id 1616, seq 360, lengt
h 64
15:25:10.607837 IP 10.10.1.38 > server: ICMP echo request, id 1616, seq 361, len
gth 64
15:25:10.607880 IP server > 10.10.1.38: ICMP echo reply, id 1616, seq 361, lengt
h 64
15:25:11.609451 IP 10.10.1.38 > server: ICMP echo request, id 1616, seq 362, len
gth 64
15:25:11.609502 IP server > 10.10.1.38: ICMP echo reply, id 1616, seq 362, lengt
h 64
15:25:12.612449 IP 10.10.1.38 > server: ICMP echo request, id 1616, seq 363, len
gth 64
15:25:12.612498 IP server > 10.10.1.38: ICMP echo reply, id 1616, seq 363, lengt
h 64
```

Figure 42: Server instance again showing the received packet details after the time-out period

So, we have seen the steps involved in setting up a communication between two Virtual Machine instances via a VNF. We have also designed a python file which is getting triggered form the VNF in order to for controlling the packet transmission. The same process can also be implemented on more than two virtual machine instances.

Chapter 6:

Implementation-level Security Mechanisms for Network Function Virtualization (NFV)

6. IMPLEMENTATION-LEVEL SECEURITY MECHANISMS FOR NETWORK FUNCTION VIRTUALIZATION (NFV)

A security service refers to a subset of AAA (Authentication, Authorization, and Accounting) services. It is a processing or communication service given by a system to provide a specified level of protection to the resources, which may or may not reside with the system in question. A security mechanism is a means to provide the security service (e.g. Use of VPN i.e. Virtual Private Network to guarantee the Confidentiality and Integrity of the communication channel in order to address man-in-the-middle attack). The security services are used to implement various aspects of security policies utilizing different security mechanisms.

6.1. Security Services

- **Authentication:** Authentication is the process of identifying whether or not someone or something is who or what it claims to be. A user is often recognized by a Login ID, and authentication occurs when the user offers a credential, such as a password that matches that user ID. Authentication is crucial because it allows organizations to keep their networks safe by allowing only authenticated users (or processes) to access protected resources such as computer systems, networks, databases, websites, and other network-based applications or services.

- **Access Control:** Access control is a critical component of security that governs who has access to and uses business information and resources. Access control policies ensure that users are who they say they are and have proper access to company data through authentication and permission. Physical access to campuses, buildings, rooms, and data centers can also be restricted using access control. Access control verifies various login credentials, such as usernames and passwords, PINs, biometric scans, and security tokens, to identify users. Once a user has been authenticated, access control grants the appropriate level of access and authorizes actions connected with that user's credentials and IP address.

- **Data Confidentiality:** Data confidentiality is concerned with preventing the disclosure of information by guaranteeing that access to the data is restricted to those who are authorized or by portraying the data in such a way that its semantics are only available to those who possess certain crucial knowledge. Data confidentiality is a fundamental data security service.

- **Data Integrity:** The assurance that the data received is exactly what an authorized entity submitted (i.e., contain no modification, insertion, deletion, or replay). It relates to data's dependability and trustworthiness throughout its existence. It can define the condition of your data—for example, whether it is valid or invalid—or the process of assuring and preserving data validity and accuracy. For example, error checking and validation are standard ways for guaranteeing data integrity as part of a process. When you ensure good data correctness and data safety, data performance and stability improve. Maintaining data integrity and guaranteeing data completeness are critical.

- **Non-Repudiation:** The certainty that someone cannot deny the legitimacy of anything is known as non-repudiation. Non-repudiation is a legal notion commonly used in information security that refers

to a service that gives verification of the origin and integrity of data. In other words, non-repudiation makes it extremely impossible to convincingly refute who/where a message originated, as well as its legitimacy and integrity. When it comes to online transactions, digital signatures (when combined with other measures) can provide non-repudiation, which is critical to ensure that a party to a contract or communication cannot deny the authenticity of their signature on a document or sending the communication in the first place. Non-repudiation in this sense refers to the ability to ensure that a party to a contract or communication must recognize the authenticity of their signature on a document or the transmission of a message. It protects against one of the entities involved in a communication denying participation in all or part of the communication.

6.2. Security Mechanisms

The implementation of the security services is provided through security mechanisms. It exists to provide and support security services. It can be divided in two classes as follows:

- Specific security mechanisms
- Pervasive security mechanisms

6.2.1. Specific Security Mechanisms

- **Encipherment:** This security mechanism deals with data concealing and covering, which aids in data confidentiality. It is accomplished by the use of mathematical computations or algorithms that convert information into non-readable form. Encryption, also known as encipherment, is the primary use of cryptography that renders data incomprehensible in order to ensure its confidentiality. It is the process of disguising data or information so that it appears random and can only be accessed by authorized individuals, while unauthorized users cannot. The algorithm employed for encipherment determines the level of data encryption. Encipherment is used to safeguard the confidentiality of data units and traffic flow information, as well as to supplement or assist other security techniques.
- **Digital signature:** For electronic documents, digital signature technologies are employed to provide an electronic equivalent of handwritten signatures. Digital signatures, like handwritten signatures, must not be forgeable; a recipient must be able to authenticate it, and the signer must not be able to later disavow it. However, unlike handwritten signatures, digital signatures include the data (or hash of the data) being signed. Distinct data results in different signatures, even though the signatory remains the same.
- **Access control mechanisms:** To define and enforce access privileges, access control systems use authenticated identities of principals, information about these principals, or capabilities. If a principal tries to use an unauthorized resource or an authorized resource with an incorrect kind of access, the access control function denies the attempt and may also report the incident to generate an alarm and record it as part of a security audit trail.
- **Data integrity mechanisms:** The reliability and trustworthiness of data throughout its lifecycle is referred to as data integrity. It can define the condition of your data—for example, whether it is valid or invalid—or the process of assuring and preserving data validity and accuracy. To protect the integrity of a sequence of data units and fields inside these data units, some type of explicit ordering

is usually required, such as sequence numbering, time stamping, or cryptographic chaining.

- **Authentication exchanges:** Authentication exchange techniques are used to validate principals' stated identities. We use the term "strong" to refer to an authentication exchange method that uses cryptographic techniques to protect the messages that are exchanged, and "weak" to refer to an authentication exchange mechanism that does not. Weak authentication exchange techniques, in general, are subject to passive wiretapping and replay attacks. This can be accomplished at the TCP/IP layer, where a two-way/three-way handshaking technique is utilized to ensure whether or not data is transmitted.
- **Traffic padding:** To protect against traffic analysis attacks, traffic padding measures are used. The development of bogus instances of communication, spurious data units, and artificial data within data units is referred to as traffic padding. The goal is not to determine if the data being transferred represents and encodes information. As a result, traffic padding procedures can be effective only if they are covered by a data secrecy service. Even in the absence of plaintext, traffic padding generates ciphertext output on a continuous basis. A random data stream is generated indefinitely. Plaintext is encrypted and transferred when it is available. In the absence of input plaintext, random data is encrypted and transferred. This makes it impossible for an attacker to tell the difference between genuine data flow and padding, and hence impossible to calculate the quantity of traffic. A handshaking method is used to determine whether or not data is transmitted.
- **Routing control:** Routing control mechanisms can be used to select specific routes for data transmission either dynamically or by prearrangement. On detection of persistent passive or active attacks, communicating systems may seek to tell the network service provider to establish a connection via a new route. Similarly, data with specific security labels may be prohibited from passing via certain networks or linkages by a security policy.
- **Notarization:** Notarization procedures can be used to ensure the integrity, origin, timing, and destination of data exchanged between two or more entities. Notarization is the process of appointing a third trusted party to oversee communication between two entities. To prevent the sender from subsequently denying that she made a request, the receiver can enlist the help of a trustworthy third party to store the sender's request. The assurance is delivered in a testable manner by a trusted third party (TTP).
- **High-Availability Clusters:** High-availability clusters are groups of computers that handle server applications that may be used reliably with little downtime. They work by harnessing redundant computers in groups or clusters utilizing high availability software to provide continuous service when system components fail. If a server running a specific application crash without clustering, the program will be unavailable until the crashed server is repaired.

6.2.2. Pervasive security mechanisms

- **Trusted functionality:** The overall concept of trusted functionality can be utilized to either broaden the scope or validate the efficacy of other security techniques. Any feature that provides or accesses a security mechanism must be reliable.
- **Security labels:** Security labels may be assigned to system resources, for example, to signify sensitivity levels. It is frequently required to provide the appropriate security label along with data in transit. A security label might be explicit or implicit, and it can include additional data connected with the data delivered (e.g., implied by the use of a specific key to encipher data or implied by the context of the data such as the source

address or route).

- **Event detection:** Security-relevant event detection can be used to discover obvious security breaches.
- **Security audit trail:** A security audit is an independent evaluation and inspection of system records and activities to verify the adequacy of system controls, ensure compliance with established policy and operational processes, detect security breaches, and recommend improvements in control, policy, and procedures. As a result, a security audit trail refers to data gathered and potentially used to aid in a security audit.
- **Security recovery:** Security recovery handles requests from mechanisms including event handling and management functions, and performs recovery actions as a result of following a set of rules.

6.3. Security Properties

- **Network Segregation Property:** The separation of vital networks from the internet as well as other internal networks is referred to as network segmentation. It also enforces communication restrictions between hosts and services. This method can reduce the danger of ransomware attacks while also improving cybersecurity safeguards throughout an entire organization, regardless of size. It also improves the performance of IT professionals by improving auditing and alerting capabilities, both of which provide vital knowledge when spotting a cyber-threat and deploying a suitable response. After a first part of the system is compromised, network segregation is critical to limiting network spread or lateral movement.
- **Mutual Authentication Property:** In an authentication protocol, mutual authentication or two-way authentication refers to two parties authenticating each other at the same time. Some protocols (IKE, SSH) make it the default form of authentication, while others make it optional (TLS). It is a desirable feature in verification schemes that transfer sensitive data, as it ensures data security with two forms of credentials: usernames:passwords and public key certificates. Mutual authentication is a critical security step that can protect against a wide range of adversarial attacks, which can have serious effects if sensitive systems such as IoT involving e-Healthcare servers are compromised. The server and client authenticate each other using mutual authentication.
- **Data Protection Property:** Data can exist in multiple states, and it can quickly change states based on an organization's needs. There are three states in which data can exist in an organization
    - Data at Rest - Inactive data that is not flowing between devices or networks is referred to as data at rest. This information is less sensitive than data in transit because it is often saved or preserved. Any information that businesses keep close to their chests is considered more valuable by hackers, making it a target for external attacks.
    - Data in Transit - Data in transit is information that is being moved from one point to another. This includes email, collaborative platforms like instant messengers, and nearly any public communication channel. Given its accessibility across the internet or private business network as it travels from one location to another, this data is often less secure than inactive data. As a result, data in transit is a potential target for hackers.
    - Data in Use - When data is accessed or consumed by an employee or business program, it is said to be in use. Data is most vulnerable in this condition, whether it is being read, processed, or manipulated, because it is immediately available to an individual, leaving it subject to attack or human error - all of which can have serious implications.

- **Source Identification Property**: The source identification property is the attribute that allows the sender of data or information in any form to be identified by the receiving end and also assures that the sender cannot deny sending the data at any moment. This property is also ensured by combining or using the mutual authentication and non-repudiation properties.
- **Configuration Property:** A system's configuration refers to the organization of each of its functional elements based on its nature, number, and primary characteristics. Configuration frequently refers to the selection of hardware, software, firmware, and documentation. A system's configuration, like its architecture, influences both its function and performance. The operating system can provide information about a computer's settings.
- **Resource Allocation Property:** Resource allocation is the process of assigning and managing assets in a manner that supports an organization's strategic goals. Resource allocation includes managing tangible assets such as hardware to make the best use of softer assets such as human capital. Resource allocation is an important feature in a heterogeneous network meant to ensure its high efficiency as well as its maintenance as a cost-benefit network. Proper resource allocation improves the performances of both the associated system and the network, and also helps in avoiding the different kinds of transient bottlenecks encountered in a system.
- **Redundancy Property:** Redundancy refers to the availability of additional or duplicate resources to support the primary system. The process of adding additional instances of network devices and lines of communication to assist assure network availability and reduce the chance of failure along the crucial data path is known as network redundancy. It is a backup system that can take over if the primary system fails. Most of the time, reserve resources are unnecessary because they are not required if everything is working smoothly.
- **User Management Property:** Administrators' capacity to manage user access to various IT resources such as systems, devices, applications, storage systems, networks, SaaS services, and more is referred to as user management. In its most basic form, user administration is the process of creating, removing, and maintaining your user store. Any solution meant to service many users must have some form of user management system, whether it is a proprietary tool integrated into the product or a connection to an existing system like Active Directory/LDAP or another identity provider.
- **Password Management Property:** It is critical to use proper passwords to ensure a company's security and productivity. As a result, password management is a vital activity.
- **Network Admission Control Property:** Network Admission Control (NAC) limits network access based on identity or security posture. When a network equipment is setup for NAC, it can require human or machine authentication before giving network access. Noncompliant devices can be banned or granted limited access to computational resources by network admission control systems, preventing insecure nodes from infecting the network.
- **Backup Property:** A backup is a copy of data that can be used to recreate the original data if needed. There are nine essential properties that a backup must have. The properties are Independent of source platform, Independent of source location, Redundant, sufficiently consistent, Recoverable, sufficiently secure, Repeatable, Testable, Observable.

6.4. Mapping of Security Property and Security Mechanism with respect to Network Function Virtualization

Table I. Security Properties v/S Mechanisms (NFV)

Sl. No.	Security Property Type	Security Mechanism
1	Network Segregation Property	VM traffic and administration traffic should be kept separate to prevent a VM from affecting other VMs or hosts. This will protect the management infrastructure from VM attacks. Separating VLAN traffic into groups and disabling any VLANs that aren't in use is also a possible solution.
2	Mutual Authentication Property	Public Key Infrastructures (PKI) can be used for the purpose of distributing Public Key Certificates (PKC) as applicable to the ETSI (European Telecommunication Standards Institute) ISG (Industry Specification Group) NFV for the support of Public Key Cryptography in authenticating, authorising and encrypting links between objects in NFV.
3	Data Protection Property	The best practice to secure the data involving NFV is by encrypting them and storing the cryptographic keys at safe locations. OpenStack service Barbican can be used for this task, as it encrypts the data using well known cryptographic algorithms such as AES, DES, and stores the keys in a secured location.
4	Source Identification Property	The keystone services of Openstack can be used to guarantee this property
5	Configuration Property	In the context of NFV, admins can provide customised configurations as per required security protocols. Each component of the Openstack platform owns its individual .conf file. Eg. *neutron.conf, nova.conf etc.* inside the /etc directory. These .conf files can be modified as per the security requirement
6	Resource Allocation Property	Resource orchestration in NFV is important to ensure there are adequate compute, storage, and network resources available to provide a network service. To meet the objective of Resource Orchestration, the NFV Orchestrator can work either with the Virtualized Infrastructure Manager (VIM) or directly with NFV Infrastructure (NFVI) resources, depending on the requirements of the network service. In Openstack, *heat* acts as the main module responsible for this property
7	Redundancy Property	For Redundancy, a VNF descriptor can replicate multiple copies of VNFs incase of failure
8	User Management Policy	OpenStack provides native support for user management through the OpenStack Keystone component. In the Keystone realm, Users and User Groups are the entities given access to resources that are isolated in Domains and Projects.

9	Password Management Property	Admin accounts in OpenStack for an organization deploying a NFV system, should have strong password policies, which must be strongly adhered to by the system administrators. Frequent changing of account and OpenStack service passwords should be undertaken so that any form credential leakage attacks could be prevented.
10	Network Admission Control Property	Network Admission Control Property can be implemented by customizing the different network interface related configuration files
11	Backup Property	For NFV, there are vital workloads on NFVs that aren't restorable via redeployment. Even stateless workloads generate logs and those logs need to be backed up. Log files are crucial to maintaining compliance and adhering to governance rules.

Chapter 7:

Configuration Methodology and Evidence Identification in Network Function Virtualization (NFV)

7. CONFIGURATION METHODOLOGY AND EVIDENCE IDENTIFICATION IN NETWORK FUNCTION VIRTUALIZATION (NFV)

7.1. Configuration Methodology of Network Function Virtualization

It is critical that we be able to modify and adapt our network virtualization technology to tackle security challenges in today's world. The ability to customize a system to handle various difficulties based on the situation is an excellent illustration of flexibility and adaptability. NFV includes several configuration files that can be used to handle the scenario.

System Configuration is primarily defined as a set of pairings that can define, alter, and fine-tune the required system functionalities. A configuration file, often known as a config file, defines the parameters, options, settings, and preferences that are applied to a system. In general. Configuration files are specified using the CFG,.CONF, and.CONFIG extensions. Aside from this, there are technically stated configuration file formats such as.XML,.YAML,.JSON,.INI,.TOML,.HCL, and so on. All of these files have a similar pattern in that they detail specific couples. Certain structural/logical differences between configuration file formats may be encountered, such as the representation of pairs (may use: or =), the representation of segments in a long configuration file (generally done using [], the presence of nested configuration, strongly typed/loosely typed, allow/disallow comments, and so on. E.g. On most cases, the firewall configuration in a UBUNTU system is saved in /etc/default/ufw and /etc/ufw/sysctl.conf. In general, packet forwarding should be enabled, and the following pairs should be added or updated to do so.

DEFAULT_FORWARD_POLICY="ACCEPT"
net/ipv4/ip_forward=1

7.1.1. Mapping of Security Properties and Configuration Methodologies - with respect to Network Function Virtualization (NFV)

This section presents a detailed study on the Configuration methodologies that should be implemented to achieve the security properties of NFV. Table II presents the different configuration files related to the NFV implementation platform i.e. OpenStack. Table III. depicts the mapping of Security Properties against the Security Mechanisms implemented to achieve the concerned property. Finally, the right most column specifies the configuration methodology that is required to implement the corresponding security mechanism.

Table II: Configuration files of different OpenStack services

Sl. No.	Openstack Services	Configuration file
1	mistral	mistral.conf
2	cinder	cinder.conf
3	neutron	neutron.conf
4	nova	nova.conf
5	placement	placement.conf
6	keystone	keystone.conf
7	tacker	tacker.conf
8	barbican	barbican.conf
9	glance	glance-api.conf
10	heat	heat.conf

Table III. Security Properties v/S Configuration Methodology (NFV)

Sl. No.	Security Property Type	Security Mechanism	Configuration Methodology
1	Network Segregation Property	VM traffic and administration traffic should be kept separate to prevent a VM from affecting other VMs or hosts. This is done by using two separate network interfaces viz. Provider and Management interface. This will protect the management infrastructure from VM attacks. Separating VLAN traffic into groups and disabling any VLANs that aren't in use is also a possible solution. The network segmentation service is administered via the Neutron API. A set of default network segment ranges are created out of the values defined in the ML2 config file.	A set of default network segment ranges are created out of the values defined in the ML2 config file: network_vlan_ranges for ml2_type_vlan, vni_ranges for ml2_type_vxlan, tunnel_id_ranges for ml2_type_gre and vni_ranges for ml2_type_geneve. They will be reloaded when the Neutron server starts or restarts. To enable the network segment range service plugin by appending network_segment_range to the list of service_plugins in the neutron.conf file on all nodes running the neutron-server service: *[DEFAULT]* *# ...* *service_plugins = ...,network_segment_range,...* The existing network segment ranges can be displayed using: *$ openstack network segment range list* Output is depicted in Fig. 44. The network segment ranges with Default as True are the ranges specified by the operators in the ML2 config file. Besides, there are also shared and tenant specific network segment ranges created by the admin previously. The admin is also able to check/show the detailed information (e.g. availability and usage statistics) of a network segment range: *$ openstack network segment range show test_range_1* Output is depicted in Fig. 45.
2	Mutual Authentication Property	Public Key Infrastructures (PKI) can be used for the purpose of distributing Public Key Certificates (PKC) as applicable to the ETSI (European Telecommunication Standards Institute) ISG (Industry Specification Group) NFV for the support of Public Key Cryptography in authenticating, authorizing and encrypting links between objects in NFV. The OpenStack Identity service (keystone) supports multiple methods of authentication, including username & password, LDAP, and external authentication methods. Kerberos may be used as an external authentication method. A mutual authentication network protocol using 'tickets' is used in this case to initiate a secure communication between client and server.	The Multi-factor authentication (MFA) rules allow an admin to force a user to use specific forms of authentication or combinations of forms of authentication to get a token. The rules are specified as follows via the user option multi_factor_auth_rules: *[["password", "totp"], ["password", "custom-auth-method"]]* Default methods specified in Keystone.conf in the [auth] methods option are: *#methods = external,password,token,oauth1,mapped* TOTP is not enabled in Keystone by default. To enable it add the totp authentication method to the [auth] section in keystone.conf: *methods = external,password,token,oauth1,totp* When Keystone is executed in a web server like

			Apache HTTPD, it is possible to have the web server also handle authentication. This enables support for additional methods of authentication that are not provided by the identity store backend and the authentication plugins that Keystone supports. In order to activate the external authentication mechanism for Identity the external method must be in the list of enabled authentication methods. By default it is enabled, so if we don't want to use external authentication, we need to remove it from the methods option in the auth section.
3	Data Protection Property	The best practice to secure the data involving NFV is by encrypting them and storing the cryptographic keys at safe locations. OpenStack service Barbican can be used for this task, as it encrypts the data using well known cryptographic algorithms such as AES (Advanced Encryption Standard), DES (The Data Encryption Standard), and stores the keys in a secured location.	Barbican is a REST API designed for the secure storage, provisioning and management of secrets such as passwords, encryption keys and X.509 certificates. Plugins are enabled and configured with settings in the */etc/barbican/barbican.conf configuration file.* There are two flavors of storage plugins currently available: the Simple Crypto plugin and the PKCS#11 crypto plugin. The simple crypto plugin is configured by default in barbican.conf and this plugin uses a single symmetric key (KEK - or 'Key Encryption Key'). The PKCS#11 crypto plugin can be used to interface with a Hardware Security Module (HSM) using the PKCS#11 protocol. Secrets are encrypted (and decrypted on retrieval) by a project specific Key Encryption Key (KEK). The KEK is protected (encrypted) with a Master KEK (MKEK). *–/etc/barbican/barbican.conf* *enabled_certificate_plugins* *Type* *multi-valued* *Default* *Simple_certificate* For simple crypto plugin the configuration is: *# Key encryption key to be used by Simple Crypto Plugin (string value)* *#kek=* *dGhpcnR5X3R3b19ieXRlX2tleWJsYWhibGFoYmxhaGg=* For PKCS#11 crypto plugin refer to Fig 43
4	Source Identification Property	The keystone services of Openstack can be used to guarantee this property	In Keystone the Identity service provides auth credential validation and data about users and groups. Users represent an individual API consumer. Groups are a container representing a collection of users. The Keystone api modules used for identity are: Keystone.api.groups Keystone.api.users Roles dictate the level of authorization the end user can obtain. Roles can be granted at either the domain or project level. A role can be assigned at the individual user or group level. Role names are unique within the owning domain.

5	Configuration Property	In the context of NFV, admins can provide customized configurations as per required security protocols. Each component of the Openstack platform owns its individual .conf file. Eg. *neutron.conf, nova.conf etc.* inside the /etc directory. These .conf files can be modified as per the security requirement.	The virtual network function configurations are stored in the vnf descriptor. Fig. 48 is part of a sample vnfd yaml file. The configuration files of the openstack are stored in etc/service/ location.
6	Resource Allocation Property	Resource orchestration in NFV is important to ensure there are adequate compute, storage, and network resources available to provide a network service. To meet the objective of Resource Orchestration, the NFV Orchestrator can work either with the Virtualized Infrastructure Manager (VIM) or directly with NFV Infrastructure (NFVI) resources, depending on the requirements of the network service. In Openstack, *heat* acts as the main module responsible for this property. The default authorization settings allow administrative users only to create resources on behalf of a different project. OpenStack handles two kinds of authorization policies: **Operation based** Policies specify access criteria for specific operations, possibly with fine-grained control over specific attributes. **Resource based** Whether access to a specific resource might be granted or not according to the permissions configured for the resource (currently available only for the network resource). The actual authorization policies enforced in an OpenStack service vary from deployment to deployment.	Each OpenStack service, Identity, Compute, Networking, and so on, has its policy.json file. This config file is mainly responsible for controlling resource access. In some cases, some operations are restricted to administrators only. The example (Figure 46) shows you how to modify a policy file to permit projects to define networks, see their resources, and permit administrative users to perform all other operations.

7	Redundancy Property	For Redundancy, a VNF descriptor can replicate multiple copies of VNFs incase of failure. Two types of redundant services are maintained by OpenStack in order to guarantee high availability. **1. Active/Passive** OpenStack maintains a redundant instance of certain services that can be brought online when the active service fails. For example, OpenStack writes to the main database while maintaining a disaster recovery database that can be brought online if the main database fails. A typical active/passive installation for a stateful service maintains a replacement resource that can be brought online when required. **2. Active/Active** Each service also has a backup but manages both the main and redundant systems concurrently. This way, if there is a failure, the user is unlikely to notice. The backup system is already online and takes on increased load while the main system is fixed and brought back online. A typical active/active installation for a stateful service includes redundant services, with all instances having an identical state. In other words, updates to one instance of a database update all other instances.	To enable redundant services the following configurations can be changed in OpenStack The Neutron API has been updated to allow administrators to set the *--ha=True/False* flag when creating a router, which overrides the default configuration of *l3_ha in /var/lib/config-data/neutron/etc/neutron/neutron.conf.* Configure Layer 3 HA in the */var/lib/config-data/neutron/etc/neutron/neutron.conf* file by enabling L3 HA and defining the number of L3 agent nodes that you want to protect each virtual router: *l3_ha = True* *max_l3_agents_per_router = 2* *min_l3_agents_per_router = 2*
8	User Management Policy	OpenStack provides native support for user management through the OpenStack Keystone component. In the Keystone realm, Users and User Groups are the entities given access to resources that are isolated in Domains and Projects. As an administrator, one has to manage projects, users, and roles. Projects are organizational units in the cloud to which we can assign users. Projects are also known as *projects* or *accounts*. Users can be members of one or more projects. Roles define which actions users can perform. An administrator, assigns roles to user-project pairs.	Each OpenStack service, Identity, Compute, Networking, and so on, has its own role-based access policies. They determine which user can access which objects in which way, and are defined in the service's policy.json file. Whenever an API call to an OpenStack service is made, the service's policy engine uses the appropriate policy definitions to determine if the call can be accepted. Any changes to policy.json are effective immediately, which allows new policies to be implemented while the service is running. A policy.json file is a text file in JSON (Javascript Object Notation) format. Each policy is defined by a one-line statement in the form *"<target>" : "<rule>"*. The policy target, also named "action", represents an API call like "start an instance" or "attach a volume". Action names are usually qualified. For example, the Compute service features API calls to list instances, volumes, and networks. In /etc/nova/policy.json, these APIs are represented by compute:get_all, volume:get_all, and network:get_all, respectively. The following policy ensures that only administrators can create new users in the Identity database: *"identity:create_user" : "role:admin"* You can limit APIs to any role. For example, the Orchestration service defines a role named heat_stack_user. Whoever has this role is not

			allowed to create stacks: *"stacks:create": "not role:heat_stack_user"*
9	Password Management Property	Admin accounts in OpenStack for an organization deploying a NFV system, should have strong password policies, which must be strongly adhered to by the system administrators. Frequent changing of account and OpenStack service passwords should be undertaken so that any form credential leakage attacks could be prevented. Some external authentication methods are often used by the OpenStack platform in order to guarantee proper management of passwords. E.g. **Password policy enforcement** Requires user passwords to conform to minimum standards for length, diversity of characters, expiration, or failed login attempts. In an external authentication scenario this would be the password policy on the original identity store.	Password compliance features are disabled by default but we can enable these features by changing the configuration settings under the [security_compliance] section in keystone.conf. Passwords can be configured to expire within a certain number of days by setting the password_expires_days: *[security_compliance]* *password_expires_days = 90* We can set password strength requirements, such as requiring numbers in passwords or setting a minimum password length, by adding a regular expression to the password_regex setting: *[security_compliance]* *password_regex = ^(?=.*\d)(?=.*[a-zA-Z]).{7,}$* The above example is a regular expression that requires a password to have:1 letter, 1 digit, and minimum length of seven (7) characters We should also provide text that describes the password strength requirements. We can do this by setting the password_regex_description: *[security_compliance]* *password_regex_description = Passwords must contain at least 1 letter, 1 digit, and be a minimum length of 7* *characters.*
10	Network Admission Control Property	Network Admission Control Property can be implemented by customizing the different network interface related configuration files. The OpenStack Networking service provides security group functionality. Security groups and their rules allow administrators and projects the ability to specify the type of traffic and direction (ingress/egress) that is allowed to pass through a virtual interface port. Quota and ARP Spoofing prevention components can be further modified in the existing configuration files to fine tune different networking components in order to guarantee proper network admission control.	The OpenStack Networking service provides security group functionality. We can view the existing security groups using the below command: *$ openstack security group list* Also we can create new security group using the following command: $ *openstack security group create GroupName --description Description* Quotas provide the ability to limit the number of network resources available to projects. We can enforce default quotas for all projects. The /etc/neutron/neutron.conf includes some options for quotas, refer to Figure 49 for the configuration options.
11	Backup Property	For NFV, there are vital workloads on NFVs that aren't restorable via redeployment. Even stateless workloads generate logs and those logs need to be backed up. Log files are crucial to maintaining compliance and adhering to governance rules. OpenStack maintains Active/Passive backups as already mentioned in **7. Redundancy Property**	The openstack command-line interface provides the tools for creating a volume backup. *$ openstack volume backup create [--incremental] [--force] VOLUME* A new configure option *backup_swift_block_size* is introduced into cinder.conf for the default Swift backup driver. This is the size in bytes that changes for each incremental backup. The existing backup_swift_object_size option, the size in bytes

			of Swift backup objects, has to be a multiple of backup_swift_block_size. The default is 32768 for backup_swift_block_size, and the default is 52428800 for backup_swift_object_size. By default, the swift object store is used for the backup repository. Refer to Figure 47 for the backup part of cinder.conf file If instead we want to use an NFS export as the backup repository, we need to add the following configuration options to the [DEFAULT] section of the cinder.conf file and restart the Block Storage services: *backup_driver = cinder.backup.drivers.nfs* *backup_share = HOST:EXPORT_PATH*

```
# Password to login to PKCS11 session (string value)
#login = <None>

# Master KEK label (as stored in the HSM) (string value)
#mkek_label = <None>

# Master KEK length in bytes. (integer value)
#mkek_length = <None>

# Master HMAC Key label (as stored in the HSM) (string value)
#hmac_label = <None>

# (Optional) HSM Slot ID that contains the token device to be used.
# (integer value)
#slot_id = 1

# Flag for Read/Write Sessions (boolean value)
#rw_session = true

# Project KEK length in bytes. (integer value)
#pkek_length = 32

# Project KEK Cache Time To Live, in seconds (integer value)
#pkek_cache_ttl = 900

# Project KEK Cache Item Limit (integer value)
#pkek_cache_limit = 100
```

Figure 43: Barbican.conf

```
+--------------------------------------+--------------------+---------+---------+
| ID                                   | Name               | Default | Shared  |
+--------------------------------------+--------------------+---------+---------+
| 20ce94e1-4e51-4aa0-a5f1-26bdfb5bd90e |                    | True    | True    |
| 4b7af684-ec97-422d-ba38-8b9c2919ae67 | test_range_3       | False   | False   |
| a021e582-6b0f-49f5-90cb-79a670c61973 |                    | True    | True    |
| a3373630-969b-4ce9-bae7-dff0f8fa2f92 | test_range_2       | False   | True    |
| a5707a8f-76f0-4f90-9aa7-c42bf54e94b5 |                    | True    | True    |
| aad1b55b-43f1-46f9-8c35-85f270863ed6 |                    | True    | True    |
| e3233178-2866-4f40-b794-7c6fecdc8655 | test_range_1       | False   | False   |
+--------------------------------------+--------------------+---------+---------+
```

```
+----------------------------------+--------------+------------------+------------+------------+
| Project ID                       | Network Type | Physical Network | Minimum ID | Maximum ID |
+----------------------------------+--------------+------------------+------------+------------+
| None                             | vxlan        | None             |          1 |        200 |
| 7011dc7fccac4efda89dc3b7f0d0975a | gre          | None             |        100 |        120 |
| None                             | vlan         | default          |          1 |        100 |
| None                             | vxlan        | None             |        501 |        505 |
| None                             | gre          | None             |          1 |        150 |
| None                             | geneve       | None             |          1 |        120 |
| 7011dc7fccac4efda89dc3b7f0d0975a | vlan         | group0-data0     |         11 |         11 |
+----------------------------------+--------------+------------------+------------+------------+
```

Figure 44: Openstack Network Segment Range List

```
+------------------+-----------------------------------------------------+
| Field            | Value                                               |
+------------------+-----------------------------------------------------+
| available        | []                                                  |
| default          | False                                               |
| id               | e3233178-2866-4f40-b794-7c6fecdc8655                |
| location         | None                                                |
| maximum          | 11                                                  |
| minimum          | 11                                                  |
| name             | test_range_1                                        |
| network_type     | vlan                                                |
| physical_network | group0-data0                                        |
| project_id       | 7011dc7fccac4efda89dc3b7f0d0975a                    |
| shared           | False                                               |
| used             | {u'7011dc7fccac4efda89dc3b7f0d0975a': ['11']}       |
+------------------+-----------------------------------------------------+
```

Figure 45: Details of a particular Network Segment Range

```
{
        "admin_or_owner": "role:admin", "tenant_id:%(tenant_id)s",
        "admin_only": "role:admin", "regular_user": "",
        "default": "rule:admin_only",
        "create_subnet": "rule:admin_only",
        "get_subnet": "rule:admin_or_owner",
        "update_subnet": "rule:admin_only",
        "delete_subnet": "rule:admin_only",
        "create_network": "",
        "get_network": "rule:admin_or_owner",
        "create_network:shared": "rule:admin_only",
        "update_network": "rule:admin_or_owner",
        "delete_network": "rule:admin_or_owner",
        "create_port": "rule:admin_only",
        "get_port": "rule:admin_or_owner",
        "update_port": "rule:admin_only",
        "delete_port": "rule:admin_only"
}
```

Figure 46: policy.json file (Example)

```
#backup_swift_user = <None>

# Swift key for authentication (string value)
#backup_swift_key = <None>

# The default Swift container to use (string value)
#backup_swift_container = volumebackups

# The size in bytes of Swift backup objects (integer value)
#backup_swift_object_size = 52428800

# The size in bytes that changes are tracked for incremental backups.
# backup_swift_object_size has to be multiple of backup_swift_block_size.
# (integer value)
#backup_swift_block_size = 32768

# The number of retries to make for Swift operations (integer value)
#backup_swift_retry_attempts = 3

# The backoff time in seconds between Swift retries (integer value)
#backup_swift_retry_backoff = 2

# Enable or Disable the timer to send the periodic progress notifications to
# Ceilometer when backing up the volume to the Swift backend storage. The
# default value is True to enable the timer. (boolean value)
#backup_swift_enable_progress_timer = true
```

*Figure 47:**cinder.conf** file for backup property*

```
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
          echo 1 > /proc/sys/net/ipv4/ip_forward
          ip route add default via 192.168.120.10

    CP1:
```

Figure 48: Sample VNF file

```
[QUOTAS]
# resource name(s) that are supported in quota features
quota_items = network,subnet,port

# default number of resource allowed per tenant, minus for unlimited
#default_quota = -1

# number of networks allowed per tenant, and minus means unlimited
quota_network = 10

# number of subnets allowed per tenant, and minus means unlimited
quota_subnet = 10

# number of ports allowed per tenant, and minus means unlimited
quota_port = 50

# number of security groups allowed per tenant, and minus means unlimited
quota_security_group = 10

# number of security group rules allowed per tenant, and minus means unlimited
quota_security_group_rule = 100

# default driver to use for quota checks
quota_driver = neutron.quota.ConfDriver
```

*Figure 49: **etc/neutron/neutron.conf** file for quotas*

7.2. Evidence Collection of Network Function Virtualization

Log files are a common tool for system engineers and administrators. They keep track of "what happened when and by whom" in the system. This data can be used to record defects and aid in their diagnosis. It can detect security breaches and other types of computer fraud. It is suitable for auditing. It is suitable for accounting needs. The information recorded is only available for further analysis if it is stored in an examined format. For analysis, this data can be arranged in a variety of ways. Storing it in a relational database, for example, would force the data into a queryable structure. However, it would make retrieval more difficult if the machine crashed, and logging would be unavailable until the database was available. A plain text format reduces reliance on other system operations and facilitates logging at all stages of computer operation, including startup and shutdown, where such programmes may be unavailable. A user/program can enter the contents of a log file as evidence at any moment for security investigation. Logging can produce technical information usable for the maintenance of applications or websites.

7.2.1. Mapping of Security Properties and Evidence Collection - with respect to Network Function Virtualization (NFV)

This section presents a detailed study on the Evidence collection methodologies that should be implemented to achieve the security properties of NFV. Table IV presents the different log files related to the NFV implementation platform i.e. OpenStack. Table V. depicts the mapping of Security Properties against the evidence collection mechanism of the concerned property. Finally, the right most column specifies the configuration methodology that is required to implement the corresponding security mechanism.

Table IV: Log Files of different OpenStack services

Openstack Services	Log Files
mistral	N/A
cinder	cinder-scheduler.log, cinder-volume.log, cinder-manage.log, api.log
neutron	neutron-dhcp-agent.log, neutron-l3-agent.log, neutron-linuxbridge-agent.log, neutron-linuxbridge-cleanup.log, neutron-metadata-agent.log, neutron-server.log
nova	nova-api.log, nova-compute.log, nova-conductor.log, mova-manage.log, nova-novncproxy.log, nova-scheduler.log, privsep-helper.log
placement	N/A
keystone	keystone-manage.log, keystone-wsgi-public.log, keystone.log
tacker	
barbican	api.log, barbican-api.log
glance	glance-api.log
heat	heat-api.log, heat-api-cfn.log, heat-engine.log

Table V. Security Properties v/s Logging Mechanism (NFV)

Sl. No.	Security Property Type	Security Mechanism	Logging mechanism
1	Network Segregation Property	VM traffic and administration traffic should be kept separate to prevent a VM from affecting other VMs or hosts. This is done by using two separate network interfaces viz. Provider and Management interface. This will protect the management infrastructure from VM attacks. Separating VLAN traffic into groups and disabling any VLANs that aren't in use is also a possible solution. The network segmentation service is administered via the Neutron API. A set of default network segment ranges are created out of the values defined in the ML2 config file.	linuxbridge-agent.log is the file responsible for logging all the events related to neutron-linuxbridge-agent service/interface. The network segment ranges created out of the values defined in the ML2 config file includes, network_vlan_ranges for ml2_type_vlan, vni_ranges for ml2_type_vxlan, tunnel_id_ranges for ml2_type_gre and vni_ranges for ml2_type_geneve. Since linuxbridge-agent.log is mainly concerned with the L2 events logging, any event occurring with respect to the above mentioned network segment ranges are reflected in this log file. A sample of linuxbridge-agent.log file has been presented in Fig. 53.
2	Mutual Authentication Property	Public Key Infrastructures (PKI) can be used for the purpose of distributing Public Key Certificates (PKC) as applicable to the ETSI (European Telecommunication Standards Institute) ISG (Industry Specification Group) NFV for the support of Public Key Cryptography in authenticating, authorizing and encrypting links between objects in NFV. The OpenStack Identity service (keystone) supports multiple methods of authentication, including username & password, LDAP, and external authentication methods. Kerberos	The Keystone log files contain messages that are produced by the keystone service which takes care of the authentication property through Key management and multi-factor authentication. Keystone-manage is only used for operations that cannot be accomplished with the HTTP API, such as data import/export and database migrations. Log for keystone-manage is stored at keystone-manage.log file. This log file contains details of creation of temporary and primary keys. Refer to Fig 50 for keystone-manage.log file sample.

		may be used as an external authentication method. A mutual authentication network protocol using 'tickets' is used in this case to initiate a secure communication between client and server.	In keystone.conf we need to change debug = true- for getting debugging information in a log file. verbose = true - for getting Information messages in the keystone.log log file. The Identity service log should show the access of the certificate files after turning up the logging levels. The below message from the keystone.log file shows the access of the certificate files. (keystone.common.wsgi):2013-07-24 12:18:11,461 DEBUG wsgi __call__arg_dict: {} (access): 2013-07-24 12:18:11,462 INFO core __call__ 127.0.0.1 - - [24/Jul/2013:16:18:11 +0000] "GET http://localhost:35357/v2.0/certificates/signing HTTP/1.0" 200 4518
3	Data Protection Property	The best practice to secure the data involving NFV is by encrypting them and storing the cryptographic keys at safe locations. OpenStack service Barbican can be used for this task, as it encrypts the data using well known cryptographic algorithms such as AES (Advanced Encryption Standard), DES (The Data Encryption Standard), and stores the keys in a secured location.	Barbican log files are stored in the path var/log/barbican. The details of the crypto plugins are stored in the api.log, barbican-api.log files. The barbican service supports auditing. When barbican auditing is enabled, it writes audit messages to an audit log file that is separate from the barbican internal logging.
4	Source Identification Property	The keystone services of Openstack can be used to guarantee this property	Identity log is stored in /var/log/keystone/keystone.log file. The Identity service log should show the access of the certificate files after turning up the logging levels. We need to set debug = True in the Identity configuration file and restart the Identity server. The below message from the keystone.log file shows the access of the certificate files. (keystone.common.wsgi): 2013-07-24 12:18:11,461 DEBUG wsgi __call__ arg_dict: {} (access): 2013-07-24 12:18:11,462 INFO core __call__ 127.0.0.1 - - [24/Jul/2013:16:18:11 +0000] "GET http://localhost:35357/v2.0/certificates/signing HTTP/1.0" 200 4518 The typical content of keystone.log is shown in Fig. 51.
5	Configuration Property	In the context of NFV, admins can provide customized configurations as per required security protocols. Each component of the Openstack platform owns its individual .conf file. E.g. *neutron.conf, nova.conf etc.* inside the /etc directory. These .conf files can be modified as per the security requirement.	Each component of the Openstack platform has their own log files. The log files are stored in var/log/service/ location. Each component and their respective log files are shown in Table IV.
6	Resource Allocation Property	Resource orchestration in NFV is important to ensure there are adequate compute, storage, and network resources available to provide a network service. To meet the objective of Resource Orchestration, the NFV Orchestrator can work either with the Virtualized Infrastructure Manager (VIM) or directly with NFV Infrastructure (NFVI) resources, depending	Since the heat module is mainly responsible for resource allocation in OpenStack, the log files for heat are heat-api.log, heat-api-cfn.log, heat-engine.log. A sample of the above-mentioned files have been presented in Fig. 58, Fig. 59 and Fig. 60.

		on the requirements of the network service. In Openstack, *heat* acts as the main module responsible for this property. The default authorization settings allow administrative users only to create resources on behalf of a different project. OpenStack handles two kinds of authorization policies: **Operation based** Policies specify access criteria for specific operations, possibly with fine-grained control over specific attributes. **Resource based** Whether access to a specific resource might be granted or not according to the permissions configured for the resource (currently available only for the network resource). The actual authorization policies enforced in an OpenStack service vary from deployment to deployment.	
7	Redundancy Property	For Redundancy, a VNF descriptor can replicate multiple copies of VNFs in case of failure. Two types of redundant services are maintained by OpenStack in order to guarantee high availability. **1. Active/Passive** OpenStack maintains a redundant instance of certain services that can be brought online when the active service fails. For example, OpenStack writes to the main database while maintaining a disaster recovery database that can be brought online if the main database fails. A typical active/passive installation for a stateful service maintains a replacement resource that can be brought online when required. **2. Active/Active** Each service also has a backup but manages both the main and redundant systems concurrently. This way, if there is a failure, the user is unlikely to notice. The backup system is already online and takes on increased load while the main system is fixed and brought back online. A typical active/active installation for a stateful service includes redundant services, with all instances having an identical state. In other words, updates to one instance of a database update all other instances.	To enable redundant services the following configurations can be changed in OpenStack The Neutron API has been updated to allow administrators to set the --ha=True/False flag when creating a router, which overrides the default configuration of l3_ha in /var/lib/config-data/neutron/etc/neutron/neutron.conf. The related logs can be found in neutron-l3-agent.log (Fig. 57)
8	User Management Policy	OpenStack provides native support for user management through the OpenStack Keystone component. In the Keystone realm, Users and User Groups are the entities given access to resources that are isolated in Domains and Projects. As an administrator, one has to manage projects, users, and roles. Projects are organizational units in the cloud to which we can assign users. Projects are also	Any issues related to user management are reflected in the keystone-manage.log, keystone-wsgi-public.log and keystone.log files. A sample of the above-mentioned files have been presented in Fig.50, Fig. 51 and Fig. 52.

		known as *projects* or *accounts*. Users can be members of one or more projects. Roles define which actions users can perform. An administrator, assigns roles to user-project pairs.	
9	Password Management Property	Admin accounts in OpenStack for an organization deploying a NFV system, should have strong password policies, which must be strongly adhered to by the system administrators. Frequent changing of account and OpenStack service passwords should be undertaken so that any form credential leakage attacks could be prevented. Some external authentication methods are often used by the OpenStack platform in order to guarantee proper management of passwords. E.g. **Password policy enforcement** Requires user passwords to conform to minimum standards for length, diversity of characters, expiration, or failed login attempts. In an external authentication scenario this would be the password policy on the original identity store.	Any issues related to password management are reflected in the keystone-manage.log, keystone-wsgi-public.log and keystone.log files. A sample of the above-mentioned files have been presented in Fig.50, Fig. 51 and Fig. 52.
10	Network Admission Control Property	Network Admission Control Property can be implemented by customizing the different network interface related configuration files. The OpenStack Networking service provides security group functionality. Security groups and their rules allow administrators and projects the ability to specify the type of traffic and direction (ingress/egress) that is allowed to pass through a virtual interface port. Quota and ARP Spoofing prevention components can be further modified in the existing configuration files to fine tune different networking components in order to guarantee proper network admission control.	Packet logging service is designed as a Neutron plug-in that captures network packets for relevant resources (e.g., security group or firewall group) when the registered events occur. On the Neutron controller node, we have to add log to service_plugins settings in /etc/neutron/neutron.conf file. service_plugins = router,metering,log To enable logging service for security_group in Layer 2, add log to option extensions in section [agent] in /etc/neutron/plugins/ml2/ml2_conf.ini for controller node and in /etc/neutron/plugins/ml2/openvswitch_agent.ini for compute/network nodes. For example: [agent] extensions = log A logging resource with an appropriate resource type can be created as follows: openstack network log create --resource-type security_group \ --description "Collecting all security events" \ --event ALL Log_Created The output for the same has been shown in Fig. 54.
11	Backup Property	For NFV, there are vital workloads on NFVs that aren't restorable via redeployment. Even stateless workloads generate logs and those logs need to be backed up. Log files are crucial to maintaining compliance and adhering to governance rules. OpenStack maintains Active/Passive backups as already	The cinder log files contain messages that are produced by the cinder service which takes care of the backup property of Openstack. The corresponding log file of each Block Storage service is stored in the /var/log/cinder/ directory of the host on which each service runs. cinder-scheduler.log, cinder-volume.log, cinder-manage.log, api.log are

		mentioned in **7. Redundancy Property**	the four log files for cinder. cinder-api Accepts API requests, and routes them to the cinder-volume for action. cinder-manage provides control of cinder database migration, and provides an interface to get information about the current state of cinder cinder-scheduler selects the optimal storage provider node on which to create the volume. Refer to Fig 55 for sample cinder-scheduler log cinder-volume interacts directly with the Block Storage service, and processes such as the cinder-scheduler. It also interacts with these processes through a message queue. The cinder-volume service responds to read and write requests sent to the Block Storage service to maintain state. Refer to Fig 56 for sample cinder-volume log

```
558 2021-07-18 20:35:55.912 46756 INFO keystone.common.utils [-] /etc/keystone/fernet-keys/ does not appear to exist;
    attempting to create it
559 2021-07-18 20:35:55.913 46756 INFO keystone.common.fernet_utils [-] Created a new temporary key: /etc/keystone/fernet-
    keys/0.tmp
560 2021-07-18 20:35:55.913 46756 INFO keystone.common.fernet_utils [-] Become a valid new key: /etc/keystone/fernet-keys/0
561 2021-07-18 20:35:55.913 46756 INFO keystone.common.fernet_utils [-] Starting key rotation with 1 key files: ['/etc/-
    keystone/fernet-keys/0']
562 2021-07-18 20:35:55.913 46756 INFO keystone.common.fernet_utils [-] Created a new temporary key: /etc/keystone/fernet-
    keys/0.tmp
563 2021-07-18 20:35:55.913 46756 INFO keystone.common.fernet_utils [-] Current primary key is: 0
564 2021-07-18 20:35:55.913 46756 INFO keystone.common.fernet_utils [-] Next primary key will be: 1
565 2021-07-18 20:35:55.913 46756 INFO keystone.common.fernet_utils [-] Promoted key 0 to be the primary: 1
566 2021-07-18 20:35:55.913 46756 INFO keystone.common.fernet_utils [-] Become a valid new key: /etc/keystone/fernet-keys/0
567 2021-07-18 20:36:15.833 46765 INFO keystone.common.utils [-] /etc/keystone/credential-keys/ does not appear to exist;
    attempting to create it
568 2021-07-18 20:36:15.833 46765 INFO keystone.common.fernet_utils [-] Created a new temporary key: /etc/keystone/-
    credential-keys/0.tmp
569 2021-07-18 20:36:15.833 46765 INFO keystone.common.fernet_utils [-] Become a valid new key: /etc/keystone/credential-
    keys/0
570 2021-07-18 20:36:15.833 46765 INFO keystone.common.fernet_utils [-] Starting key rotation with 1 key files: ['/etc/-
    keystone/credential-keys/0']
571 2021-07-18 20:36:15.833 46765 INFO keystone.common.fernet_utils [-] Created a new temporary key: /etc/keystone/-
    credential-keys/0.tmp
```

Figure 50: keystone-manage.log file

```
2016-11-09 14:31:46.834 818 INFO migrate.versioning.api [-] done
2016-11-09 14:31:46.834 818 INFO migrate.versioning.api [-] 3 -> 4...
2016-11-09 14:31:46.872 818 INFO migrate.versioning.api [-] done
2016-11-09 14:31:48.435 1082 WARNING keystone.assignment.core [-] Deprecated: Use of the identity driver config to automatically
configure the same assignment driver has been deprecated, in the "O" release, the assignment driver will need to be expicitly
configured if different than the default (SQL).
2016-11-09 14:31:48.648 1082 WARNING oslo_config.cfg [-] Option "rpc_backend" from group "DEFAULT" is deprecated for removal.  Its
value may be silently ignored in the future.
2016-11-09 14:31:48.680 1082 WARNING oslo_config.cfg [-] Option "rabbit_hosts" from group "oslo_messaging_rabbit" is deprecated for
removal.  Its value may be silently ignored in the future.
2016-11-09 14:31:48.681 1082 WARNING oslo_config.cfg [-] Option "rabbit_userid" from group "oslo_messaging_rabbit" is deprecated for
removal.  Its value may be silently ignored in the future.
2016-11-09 14:31:48.681 1082 WARNING oslo_config.cfg [-] Option "rabbit_password" from group "oslo_messaging_rabbit" is deprecated
for removal.  Its value may be silently ignored in the future.
2016-11-09 14:31:48.774 1082 INFO keystone.cmd.cli [-] Created domain default
2016-11-09 14:31:48.802 1082 INFO keystone.cmd.cli [req-ace08b7c-d0d2-4b18-b792-1ec3402575b1 - - - - -] Created project admin
2016-11-09 14:31:48.886 1082 INFO keystone.cmd.cli [req-ace08b7c-d0d2-4b18-b792-1ec3402575b1 - - - - -] Created user admin
2016-11-09 14:31:48.895 1082 INFO keystone.cmd.cli [req-ace08b7c-d0d2-4b18-b792-1ec3402575b1 - - - - -] Created role admin
2016-11-09 14:31:48.916 1082 INFO keystone.cmd.cli [req-ace08b7c-d0d2-4b18-b792-1ec3402575b1 - - - - -] Granted admin on admin to
user admin.
2016-11-09 14:31:48.986 1082 INFO keystone.assignment.core [req-ace08b7c-d0d2-4b18-b792-1ec3402575b1 - - - - -] Creating the default
role 9fe2ff9ee4384b1894a90878d3e92bab because it does not exist.
2016-11-09 14:32:09.175 1988 WARNING keystone.assignment.core [-] Deprecated: Use of the identity driver config to automatically
configure the same assignment driver has been deprecated, in the "O" release, the assignment driver will need to be expicitly
configured if different than the default (SQL).
```

Figure 51: keystone.log file

```
1 2022-02-17 13:04:47.705 2645 WARNING keystone.common.rbac_enforcer.enforcer [req-03feb272-70f4-46b3-beb7-7fd1ec99f49f
  31d2b01615cb4c388904f75b14e24e6c b6fb59f2104e40b3b877de1bb3b8a001 - default default] Deprecated policy rules found. Use
  oslopolicy-policy-generator and oslopolicy-policy-upgrade to detect and resolve deprecated policies in your configuration.
2 2022-02-17 13:10:33.934 2643 WARNING keystone.common.rbac_enforcer.enforcer [req-7ecfa214-38e7-41bd-8c75-fc220b949c1c
  31d2b01615cb4c388904f75b14e24e6c b6fb59f2104e40b3b877de1bb3b8a001 - default default] Deprecated policy rules found. Use
  oslopolicy-policy-generator and oslopolicy-policy-upgrade to detect and resolve deprecated policies in your configuration.
3 2022-02-17 13:10:36.319 2644 WARNING keystone.common.rbac_enforcer.enforcer [req-0deb20e3-1938-4942-89f9-6a8a7375c026
  06ac50ba2a4840778a10685c1c51ad98 b6fb59f2104e40b3b877de1bb3b8a001 - default default] Deprecated policy rules found. Use
  oslopolicy-policy-generator and oslopolicy-policy-upgrade to detect and resolve deprecated policies in your configuration.
4 2022-02-17 13:15:42.321 2642 WARNING keystone.common.rbac_enforcer.enforcer [req-6cb42b74-db1d-4e0a-bab1-b845e74399b8
  06ac50ba2a4840778a10685c1c51ad98 b6fb59f2104e40b3b877de1bb3b8a001 - default default] Deprecated policy rules found. Use
  oslopolicy-policy-generator and oslopolicy-policy-upgrade to detect and resolve deprecated policies in your configuration.
5 2022-02-17 13:30:53.852 2646 WARNING keystone.common.rbac_enforcer.enforcer [req-b146eb43-40ea-4b9f-961d-15dd905fe59b
  31d2b01615cb4c388904f75b14e24e6c b6fb59f2104e40b3b877de1bb3b8a001 - default default] Deprecated policy rules found. Use
  oslopolicy-policy-generator and oslopolicy-policy-upgrade to detect and resolve deprecated policies in your configuration.
```

Figure 52: keystone-wsgi-public.log file

```
2022-02-18 12:59:56.455 1119 INFO neutron.common.config [-] Logging enabled!
2022-02-18 12:59:56.456 1119 INFO neutron.common.config [-] /usr/bin/neutron-linuxbridge-cleanup version 17.1.2
2022-02-18 12:59:56.456 1119 INFO neutron.cmd.linuxbridge_cleanup [-] Interface mappings: {'provider': 'enp0s8'}.
2022-02-18 12:59:56.456 1119 INFO neutron.cmd.linuxbridge_cleanup [-] Bridge mappings: {}.
2022-02-18 12:59:56.457 1119 INFO oslo.privsep.daemon [-] Running privsep helper: ['sudo', '/usr/bin/neutron-rootwrap', '/et
c/neutron/rootwrap.conf', 'privsep-helper', '--config-file', '/etc/neutron/neutron.conf', '--config-file', '/etc/neutron/plu
gins/ml2/linuxbridge_agent.ini', '--privsep_context', 'neutron.privileged.default', '--privsep_sock_path', '/tmp/tmpgis8fp22
/privsep.sock']
2022-02-18 12:59:57.447 1119 INFO oslo.privsep.daemon [-] Spawned new privsep daemon via rootwrap
2022-02-18 12:59:57.311 2375 INFO oslo.privsep.daemon [-] privsep daemon starting
2022-02-18 12:59:57.324 2375 INFO oslo.privsep.daemon [-] privsep process running with uid/gid: 0/0
2022-02-18 12:59:57.328 2375 INFO oslo.privsep.daemon [-] privsep process running with capabilities (eff/prm/inh): CAP_DAC_O
VERRIDE|CAP_DAC_READ_SEARCH|CAP_NET_ADMIN|CAP_SYS_ADMIN|CAP_SYS_PTRACE/CAP_DAC_OVERRIDE|CAP_DAC_READ_SEARCH|CAP_NET_ADMIN|CA
P_SYS_ADMIN|CAP_SYS_PTRACE/none
2022-02-18 12:59:57.329 2375 INFO oslo.privsep.daemon [-] privsep daemon running as pid 2375
2022-02-18 12:59:58.064 1119 INFO neutron.cmd.linuxbridge_cleanup [-] Linux bridge cleanup completed successfully
```

Figure 53:linuxbridge-agent.log

Field	Value
Description	Collecting all security events
Enabled	True
Event	ALL
ID	8085c3e6-0fa2-4954-b5ce-ff6207931b6d
Name	Log_Created
Project	02568bd62b414221956f15dbe9527d16
Resource	None
Target	None
Type	security_group
created_at	2017-07-05T02:56:43Z
revision_number	0
tenant_id	02568bd62b414221956f15dbe9527d16
updated_at	2017-07-05T02:56:43Z

Figure 54: Log Resource for Security Group in OpenStack

```
2022-02-17 13:04:29.456 2512 INFO cinder.rpc [req-b321d7e8-6af4-42ac-b22b-6b32547b44cf - - - - -] Automatically selected
 cinder-volume objects version 1.38 as minimum service version.
2022-02-17 13:04:29.462 2512 INFO cinder.rpc [req-b321d7e8-6af4-42ac-b22b-6b32547b44cf - - - - -] Automatically selected
 cinder-volume RPC version 3.16 as minimum service version.
2022-02-17 13:04:29.476 2512 INFO cinder.rpc [req-b321d7e8-6af4-42ac-b22b-6b32547b44cf - - - - -] Automatically selected
 cinder-backup objects version 1.38 as minimum service version.
2022-02-17 13:04:29.481 2512 INFO cinder.rpc [req-b321d7e8-6af4-42ac-b22b-6b32547b44cf - - - - -] Automatically selected
 cinder-backup RPC version 2.2 as minimum service version.
2022-02-17 13:04:29.486 2512 INFO cinder.rpc [req-b321d7e8-6af4-42ac-b22b-6b32547b44cf - - - - -] Automatically selected
 cinder-scheduler objects version 1.38 as minimum service version.
2022-02-17 13:04:29.491 2512 INFO cinder.rpc [req-b321d7e8-6af4-42ac-b22b-6b32547b44cf - - - - -] Automatically selected
 cinder-scheduler RPC version 3.12 as minimum service version.
2022-02-17 13:04:29.529 2512 INFO cinder.service [-] Starting cinder-scheduler node (version 17.1.0)
2022-02-17 13:04:29.538 2512 INFO cinder.manager [req-09c6de45-da7a-404a-b069-e194841049b4 - - - - -] Initiating service 1
 cleanup
2022-02-17 13:04:29.546 2512 INFO cinder.manager [req-09c6de45-da7a-404a-b069-e194841049b4 - - - - -] Service 1 cleanup
 completed.
2022-02-17 13:04:43.723 2512 INFO cinder.message.api [req-71f5eed1-4375-4389-9e15-263b0019ac0a - - - - -] Deleted 0
 expired messages.
2022-02-17 17:48:25.599 2512 WARNING amqp [req-b321d7e8-6af4-42ac-b22b-6b32547b44cf - - - - -] Received method (60, 30)
 during closing channel 1. This method will be ignored
```

Figure 55:cinder-scheduler.log

```
2022-02-17 13:04:29.300 2515 INFO cinder.rpc [req-5d49251e-99b7-4096-8d2a-5f9b0c5cffab - - - - -] Automatically selected
 cinder-scheduler objects version 1.38 as minimum service version.
2022-02-17 13:04:29.305 2515 INFO cinder.rpc [req-5d49251e-99b7-4096-8d2a-5f9b0c5cffab - - - - -] Automatically selected
 cinder-scheduler RPC version 3.12 as minimum service version.
2022-02-17 13:04:29.364 2515 INFO cinder.volume.manager [req-5d49251e-99b7-4096-8d2a-5f9b0c5cffab - - - - -] Determined
 volume DB was not empty at startup.
2022-02-17 13:04:29.384 2515 INFO cinder.volume.manager [req-5d49251e-99b7-4096-8d2a-5f9b0c5cffab - - - - -] Image-
 volume cache disabled for host openstack-VirtualBox@lvm.
2022-02-17 13:04:29.418 2515 INFO oslo_service.service [req-5d49251e-99b7-4096-8d2a-5f9b0c5cffab - - - - -] Starting 1
 workers
2022-02-17 13:04:29.441 3291 INFO cinder.service [-] Starting cinder-volume node (version 17.1.0)
2022-02-17 13:04:29.478 3291 INFO cinder.volume.manager [req-276e59ef-6d01-40db-9f96-b84b6630ff2b - - - - -] Starting
 volume driver LVMVolumeDriver (3.0.0)
2022-02-17 13:04:35.099 3291 INFO cinder.volume.drivers.lvm [req-276e59ef-6d01-40db-9f96-b84b6630ff2b - - - - -]
 Enabling LVM thin provisioning by default because a thin pool exists.
2022-02-17 13:04:37.136 3291 INFO cinder.volume.driver [req-276e59ef-6d01-40db-9f96-b84b6630ff2b - - - - -] Driver
 hasn't implemented _init_vendor_properties()
2022-02-17 13:04:37.699 3291 INFO oslo.privsep.daemon [req-276e59ef-6d01-40db-9f96-b84b6630ff2b - - - - -] Running
 privsep helper: ['sudo', 'cinder-rootwrap', '/etc/cinder/rootwrap.conf', 'privsep-helper', '--config-file', '/etc/cinder/
 cinder.conf', '--privsep_context', 'cinder.privsep.sys_admin_pctxt', '--privsep_sock_path', '/tmp/tmpg_a79vj3/-
 privsep.sock']
2022-02-17 13:04:38.727 3291 INFO oslo.privsep.daemon [req-276e59ef-6d01-40db-9f96-b84b6630ff2b - - - - -] Spawned new
 privsep daemon via rootwrap
```

Figure 56: cinder-volume.log

```
2022-02-18 13:00:11.456 2554 INFO neutron.common.config [-] Logging enabled!
2022-02-18 13:00:11.457 2554 INFO neutron.common.config [-] /usr/bin/neutron-l3-agent version 17.1.2
2022-02-18 13:01:12.377 2554 ERROR neutron_lib.rpc [req-f3182785-61e8-4044-878e-1aa45d784865 - - - - -] Timeout in RPC metho
d get_host_ha_router_count. Waiting for 28 seconds before next attempt. If the server is not down, consider increasing the r
pc_response_timeout option as Neutron server(s) may be overloaded and unable to respond quickly enough.: oslo_messaging.exce
ptions.MessagingTimeout: Timed out waiting for a reply to message ID 7dc49a0a15a24ed4b6c5e709be3a1dc0
2022-02-18 13:01:12.381 2554 WARNING neutron_lib.rpc [req-f3182785-61e8-4044-878e-1aa45d784865 - - - - -] Increasing timeout
 for get_host_ha_router_count calls to 120 seconds. Restart the agent to restore it to the default value.: oslo_messaging.ex
ceptions.MessagingTimeout: Timed out waiting for a reply to message ID 7dc49a0a15a24ed4b6c5e709be3a1dc0
2022-02-18 13:01:39.964 2554 WARNING neutron.agent.l3.agent [req-f3182785-61e8-4044-878e-1aa45d784865 - - - - -] l3-agent ca
nnot contact neutron server to retrieve HA router count. Check connectivity to neutron server. Retrying... Detailed message:
 Timed out waiting for a reply to message ID 7dc49a0a15a24ed4b6c5e709be3a1dc0.: oslo_messaging.exceptions.MessagingTimeout:
Timed out waiting for a reply to message ID 7dc49a0a15a24ed4b6c5e709be3a1dc0
2022-02-18 13:01:40.088 2554 INFO neutron.agent.l3.agent [req-0f74a6c9-9cee-435a-b837-e020ebe71870 - - - - -] Agent HA route
rs count 0
2022-02-18 13:01:40.090 2554 INFO neutron.agent.agent_extensions_manager [req-0f74a6c9-9cee-435a-b837-e020ebe71870 - - - - -
] Loaded agent extensions: []
2022-02-18 13:01:40.118 2554 INFO eventlet.wsgi.server [-] (2554) wsgi starting up on http:/var/lib/neutron/keepalived-state
-change
2022-02-18 13:01:40.160 2554 INFO neutron.agent.l3.agent [-] L3 agent started
2022-02-18 13:01:40.219 2554 INFO neutron.agent.l3.agent [-] Agent has just been revived. Doing a full sync.
2022-02-18 13:01:42.166 2554 INFO oslo.privsep.daemon [req-20f665d5-6d6c-4e4b-98d7-8b0d1f00637c - - - - -] Running privsep h
elper: ['sudo', '/usr/bin/neutron-rootwrap', '/etc/neutron/rootwrap.conf', 'privsep-helper', '--config-file', '/etc/neutron/
neutron.conf', '--config-file', '/etc/neutron/l3_agent.ini', '--config-file', '/etc/neutron/fwaas_driver.ini', '--privsep_co
ntext', 'neutron.privileged.default', '--privsep_sock_path', '/tmp/tmpulx7hehq/privsep.sock']
2022-02-18 13:01:43.045 2554 INFO oslo.privsep.daemon [req-20f665d5-6d6c-4e4b-98d7-8b0d1f00637c - - - - -] Spawned new privs
ep daemon via rootwrap
2022-02-18 13:01:42.918 4112 INFO oslo.privsep.daemon [-] privsep daemon starting
2022-02-18 13:01:42.931 4112 INFO oslo.privsep.daemon [-] privsep process running with uid/gid: 0/0
2022-02-18 13:01:42.933 4112 INFO oslo.privsep.daemon [-] privsep process running with capabilities (eff/prm/inh): CAP_DAC_O
VERRIDE|CAP_DAC_READ_SEARCH|CAP_NET_ADMIN|CAP_SYS_ADMIN|CAP_SYS_PTRACE/CAP_DAC_OVERRIDE|CAP_DAC_READ_SEARCH|CAP_NET_ADMIN|CA
P_SYS_ADMIN|CAP_SYS_PTRACE/none
2022-02-18 13:01:42.934 4112 INFO oslo.privsep.daemon [-] privsep daemon running as pid 4112
```

Figure 57: neutron-l3-agent.log

```
16 2022-02-18 13:00:11.673 3285 INFO eventlet.wsgi.server [-] (3285) wsgi starting up on http://0.0.0.0:8004
17 2022-02-18 13:00:11.663 2546 INFO heat.common.wsgi [-] Started child 3288
18 2022-02-18 13:00:11.690 2546 INFO heat.common.wsgi [-] Started child 3291
19 2022-02-18 13:00:11.719 2546 INFO heat.common.wsgi [-] Started child 3296
20 2022-02-18 13:00:11.732 2546 INFO heat.common.wsgi [-] Started child 3297
21 2022-02-18 13:00:11.763 2546 INFO heat.common.wsgi [-] Started child 3308
22 2022-02-18 13:00:11.803 2546 INFO heat.common.wsgi [-] Started child 3309
23 2022-02-18 13:00:11.842 2546 INFO heat.common.wsgi [-] Started child 3310
24 2022-02-18 13:00:11.868 2546 INFO heat.common.wsgi [-] Started child 3311
25 2022-02-18 13:00:11.897 2546 INFO heat.common.wsgi [-] Started child 3312
26 2022-02-18 13:00:11.941 2546 INFO heat.common.wsgi [-] Started child 3313
27 2022-02-18 13:00:11.993 2546 INFO heat.common.wsgi [-] Started child 3314
28 2022-02-18 13:00:12.397 3311 INFO eventlet.wsgi.server [-] (3311) wsgi starting up on http://0.0.0.0:8004
29 2022-02-18 13:00:11.702 3288 INFO eventlet.wsgi.server [-] (3288) wsgi starting up on http://0.0.0.0:8004
30 2022-02-18 13:00:12.448 3308 INFO eventlet.wsgi.server [-] (3308) wsgi starting up on http://0.0.0.0:8004
31 2022-02-18 13:00:11.745 3286 INFO eventlet.wsgi.server [-] (3286) wsgi starting up on http://0.0.0.0:8004
32 2022-02-18 13:00:12.500 3309 INFO eventlet.wsgi.server [-] (3309) wsgi starting up on http://0.0.0.0:8004
33 2022-02-18 13:00:11.680 3287 INFO eventlet.wsgi.server [-] (3287) wsgi starting up on http://0.0.0.0:8004
34 2022-02-18 13:00:12.514 3291 INFO eventlet.wsgi.server [-] (3291) wsgi starting up on http://0.0.0.0:8004
35 2022-02-18 13:00:12.527 3296 INFO eventlet.wsgi.server [-] (3296) wsgi starting up on http://0.0.0.0:8004
36 2022-02-18 13:00:12.540 3314 INFO eventlet.wsgi.server [-] (3314) wsgi starting up on http://0.0.0.0:8004
37 2022-02-18 13:00:12.549 3312 INFO eventlet.wsgi.server [-] (3312) wsgi starting up on http://0.0.0.0:8004
38 2022-02-18 13:00:12.566 3313 INFO eventlet.wsgi.server [-] (3313) wsgi starting up on http://0.0.0.0:8004
39 2022-02-18 13:00:12.586 3310 INFO eventlet.wsgi.server [-] (3310) wsgi starting up on http://0.0.0.0:8004
40 2022-02-18 13:00:12.598 3297 INFO eventlet.wsgi.server [-] (3297) wsgi starting up on http://0.0.0.0:8004
41 2022-02-18 13:29:09.630 3308 INFO heat.common.wsgi [req-1dd7ec54-03e9-42d2-acd8-89756ce9087e admin admin - default
   default] Processing request: GET /v1/bc689f9ce76540b2b6252f67a1c84efd/stacks
42 2022-02-18 13:29:09.706 3308 INFO eventlet.wsgi.server [req-1dd7ec54-03e9-42d2-acd8-89756ce9087e admin admin - default
   default] 192.168.0.103 - - [18/Feb/2022 13:29:09] "GET /v1/bc689f9ce76540b2b6252f67a1c84efd/stacks?-
   limit=21&sort_dir=desc&sort_key=created_at HTTP/1.1" 200 212 0.123391
43 2022-02-18 13:41:28.199 3284 INFO heat.common.wsgi [req-79a2749b-b07d-4ae7-8636-d702aa55b1b8 admin admin - default
   default] Processing request: GET /v1/bc689f9ce76540b2b6252f67a1c84efd/services
44 2022-02-18 13:41:28.269 3284 INFO eventlet.wsgi.server [req-79a2749b-b07d-4ae7-8636-d702aa55b1b8 admin admin - default
   default] 192.168.0.103 - - [18/Feb/2022 13:41:28] "GET /v1/bc689f9ce76540b2b6252f67a1c84efd/services HTTP/1.1" 200 5974
   0.825750
```

Figure 58:heat-api.log file

```
1 2022-02-18 13:00:11.415 2543 INFO keyring.backend [-] Loading KWallet
2 2022-02-18 13:00:11.423 2543 INFO keyring.backend [-] Loading SecretService
3 2022-02-18 13:00:11.426 2543 INFO keyring.backend [-] Loading Windows
4 2022-02-18 13:00:11.427 2543 INFO keyring.backend [-] Loading chainer
5 2022-02-18 13:00:11.429 2543 INFO keyring.backend [-] Loading macOS
6 2022-02-18 13:00:11.478 2543 WARNING keystonemiddleware.auth_token [-] AuthToken middleware is set with
  keystone_authtoken.service_token_roles_required set to False. This is backwards compatible but deprecated behaviour.
  Please set this to True.
7 2022-02-18 13:00:11.583 2543 INFO heat-api-cfn [-] Starting Heat API on 0.0.0.0:8000
8 2022-02-18 13:00:11.584 2543 INFO heat.common.wsgi [-] Starting single process server
9 2022-02-18 13:00:11.585 2543 INFO eventlet.wsgi.server [-] (2543) wsgi starting up on http://0.0.0.0:8000
```

Figure 59: heat-api-cfn.log file

```
215 2022-02-21 17:07:38.161 2511 INFO heat.engine.environment [-] Registered: [Plugin](User:False) OS::Zaqar::Queue ->
    <class 'heat.engine.resources.openstack.zaqar.queue.ZaqarQueue'>
216 2022-02-21 17:07:38.161 2511 INFO heat.engine.environment [-] Registered: [Plugin](User:False)
    OS::Zaqar::SignedQueueURL -> <class 'heat.engine.resources.openstack.zaqar.queue.ZaqarSignedQueueURL'>
217 2022-02-21 17:07:38.161 2511 INFO heat.engine.environment [-] Registered: [Plugin](User:False) OS::Zaqar::Subscription -
    > <class 'heat.engine.resources.openstack.zaqar.subscription.ZaqarSubscription'>
218 2022-02-21 17:07:38.161 2511 INFO heat.engine.environment [-] Registered: [Plugin](User:False)
    OS::Zaqar::MistralTrigger -> <class 'heat.engine.resources.openstack.zaqar.subscription.MistralTrigger'>
219 2022-02-21 17:07:38.161 2511 INFO heat.engine.environment [-] Registered: [Plugin](User:False) OS::Zun::Container ->
    <class 'heat.engine.resources.openstack.zun.container.Container'>
220 2022-02-21 17:07:38.161 2511 INFO heat.engine.environment [-] Registered: [Wildcard Mapping](User:False) OS::Quantum* -
    > OS::Neutron*
221 2022-02-21 17:07:38.162 2511 INFO heat.engine.environment [-] Registered: [Template](User:False) AWS::CloudWatch::Alarm
    -> file:///etc/heat/templates/AWS_CloudWatch_Alarm.yaml
222 2022-02-21 17:07:38.162 2511 INFO heat.engine.environment [-] Registered: [Mapping](User:False) OS::Metering::Alarm ->
    OS::Aodh::Alarm
223 2022-02-21 17:07:38.162 2511 INFO heat.engine.environment [-] Registered: [Template](User:False) AWS::RDS::DBInstance -
    > file:///etc/heat/templates/AWS_RDS_DBInstance.yaml
224 2022-02-21 17:07:38.162 2511 INFO heat.engine.environment [-] Registered: [Mapping](User:False) OS::Ceilometer::Alarm -
    > OS::Aodh::Alarm
225 2022-02-21 17:07:38.162 2511 INFO heat.engine.environment [-] Registered: [Mapping](User:False)
    OS::Ceilometer::GnocchiResourcesAlarm -> OS::Aodh::GnocchiResourcesAlarm
226 2022-02-21 17:07:38.162 2511 INFO heat.engine.environment [-] Registered: [Mapping](User:False)
    OS::Ceilometer::GnocchiAggregationByMetricsAlarm -> OS::Aodh::GnocchiAggregationByMetricsAlarm
227 2022-02-21 17:07:38.162 2511 INFO heat.engine.environment [-] Registered: [Mapping](User:False)
    OS::Ceilometer::GnocchiAggregationByResourcesAlarm -> OS::Aodh::GnocchiAggregationByResourcesAlarm
228 2022-02-21 17:07:38.162 2511 INFO heat.engine.environment [-] Registered: [Mapping](User:False)
    OS::Ceilometer::CombinationAlarm -> OS::Aodh::CombinationAlarm
229 2022-02-21 17:07:38.167 2511 WARNING heat.engine.service [-] The default value of "trusts_delegated_roles" option in
    heat.conf is changed to [] in Kilo and heat will delegate all roles of trustor. Please keep the same if you do not want
    to delegate subset roles when upgrading.
230 2022-02-21 17:07:38.168 2511 INFO oslo_service.service [-] Starting 16 workers
231 2022-02-21 17:07:38.276 2511 WARNING oslo_config.cfg [-] Deprecated: Option "deferred_auth_method" from group "DEFAULT"
    is deprecated for removal (Stored password based deferred auth is broken when used with keystone v3 and is not
    supported.).  Its value may be silently ignored in the future.
232 2022-02-21 17:07:38.283 2511 WARNING oslo_config.cfg [-] Deprecated: Option "heat_watch_server_url" from group
    "DEFAULT" is deprecated for removal (Heat CloudWatch Service has been removed.).  Its value may be silently ignored in
    the future.
233 2022-02-21 17:07:38.301 2511 WARNING oslo_config.cfg [-] Deprecated: Option "auth_plugin" from group "trustee" is
    deprecated. Use option "auth_type" from group "trustee".
234 2022-02-21 17:07:38.376 3320 INFO heat.engine.worker [-] Starting engine_worker (1.4) in engine e765539b-e512-4800-99bc-
    be87dc0d7029.
```

Figure 60: heat-engine.log file

Chapter 8:

Conclusion and Future work

8. CONCLUSION AND FUTURE WORK

NFV abstracts network functions from the hardware on which they run, making network equipment more open. These virtual functions can be executed in virtual machines, allowing operators and users to offer more agile and flexible current and new network services, providing a wide range of options and promoting innovation. In the above document we discussed Network Function Virtualization, it's architecture and use cases, technologies available to implement Network Function Virtualization. Following that we have chosen OpenStack to implement network function virtualization and went on a detailed analysis about the architecture, working mechanism, security mechanism, configuration methodologies and evidence identification and collection mechanisms related to it. We have developed a linear network architecture using a packet counting VNF in between a server and a client instance. We find that NFV using OpenStack provides answers a lot of questions that a vulnerability analysis asks.

NFVI security is still in its early stage and there are still many open security concerns to address. Taking care of the open security challenges is a crucial future research direction. Future work will involve an investigation on how to improve final user experience with the platform and administration activities regarding to measures and monitoring the cloud environment. Future research directions will also include improving the performance of NFV, placements of VNFs, installation and migration of VNF, VNF outsourcing, etc. Another Future research direction is OpenStack security and access improvement. Patching the open vulnerabilities in the newer OpenStack versions. Formulating better guidelines for accessing NFV systems deployed using OpenStack. Resolving NFV performance issues in mobile networks and addressing backward compatibility with legacy systems can be another direction for future work.

References

[1] ETSI GS NFV 002 V1.2.1 (2014-12) Network Functions Virtualisation (NFV); Architectural Framework

[2] A. M. Alwakeel, A. K. Alnaim and E. B. Fernandez, "Toward a Reference Architecture for NFV," *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, 2019, pp. 1-6, doi: 10.1109/CAIS.2019.8769449.

[3] ETSI GS NFV-MAN 001 V1.1.1 (2014-12) Network Functions Virtualisation (NFV); Management and Orchestration

[4] J. Gil Herrera and J. F. Botero, "Resource Allocation in NFV: A Comprehensive Survey," in IEEE Transactions on Network and Service Management, vol. 13, no. 3, pp. 518-532, Sept. 2016, doi: 10.1109/TNSM.2016.2598420.

[5] ETSI, NFV: Use Cases, 2017, https://www.etsi.org/deliver/etsi_gs/nfv/001_099/001/01.01.01_60/gs_nfv001v010101p.pdf

[6] Yi, B., Wang, X., Li, K., Das, S.K., & Huang, M. (2018). A comprehensive survey of Network Function Virtualization. *Comput. Networks*, *133*, 212-262.

[7] B. Han, V. Gopalakrishnan, L. Ji and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," in *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90-97, Feb. 2015, doi: 10.1109/MCOM.2015.7045396.

[8] Tiago Rosado and Jorge Bernardino. 2014. An overview of openstack architecture. In Proceedings of the 18th International Database Engineering & Applications Symposium (IDEAS '14). Association for Computing Machinery, New York, NY, USA, 366–367. https://doi.org/10.1145/2628194.2628195

[9] R. L. S. de Oliveira, C. M. Schweitzer, A. A. Shinoda and Ligia Rodrigues Prete, "Using Mininet for emulation and prototyping Software-Defined Networks," 2014 IEEE Colombian Conference on Communications and Computing (COLCOM), 2014, pp. 1-6, doi: 10.1109/ColComCon.2014.6860404.

[10] Kaur, Karamjeet & Singh, Japinder & Ghumman, Navtej. (2014). Mininet as Software Defined Networking Testing Platform.

[11] J. Chen, Y. Chen, S. -C. Tsai and Y. -B. Lin, "Implementing NFV system with OpenStack," *2017 IEEE Conference on Dependable and Secure Computing*, 2017, pp. 188-194, doi: 10.1109/DESEC.2017.8073806.

[12] J. Castillo-Lema, A. Venâncio Neto, F. de Oliveira and S. Takeo Kofuji, "Mininet-NFV: Evolving Mininet with OASIS TOSCA NVF profiles Towards Reproducible NFV Prototyping," 2019 IEEE Conference on Network Softwarization (NetSoft), 2019, pp. 506-512, doi: 10.1109/NETSOFT.2019.8806686.

[13] ETSI GS NFV-SOL 001 V2.7.1 (2020-01) Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; NFV descriptors based on TOSCA specification

[14] FN Division, Telecommunication centre Study paper on Network Function Virtualization (NFV) & Its impact on Future Telecom Networks

[15] Y. Li and M. Chen, "Software-Defined Network Function Virtualization: A Survey," in IEEE Access, vol. 3, pp. 2542-2553, 2015, doi: 10.1109/ACCESS.2015.2499271.

[16] J. Matias, J. Garay, N. Toledo, J. Unzilla and E. Jacob, "Toward an SDN-enabled NFV architecture," in IEEE Communications Magazine, vol. 53, no. 4, pp. 187-193, April 2015, doi: 10.1109/MCOM.2015.7081093.

[17] S. Lal, T. Taleb and A. Dutta, "NFV: Security Threats and Best Practices," in IEEE Communications Magazine, vol. 55, no. 8, pp. 211-217, Aug. 2017, doi: 10.1109/MCOM.2017.1600899.

[18] Oliveira, Diogo. (2018). FUTURE CHALLENGES ADDRESSING NFV SECURITY AND SURVIVABILITY.

[19] mber-Jacobson et al., "OpenNF: Enabling Innovation in Network Function Control," Proc. SIGCOMM 2014, Aug. 2014, pp. 163–74

[20] S. Rajagopalan et al., "Split/Merge: System Support for Elastic Execution in Virtual Middleboxes," Proc. NSDI '13, Apr. 2013, pp. 227–40.

[21] ETSI, NFV: Use Cases, 2017, https://www.etsi.org/deliver/etsi_gs/nfv/001_099/001/01.01.01_60/gs_nfv001v010101p.pdf

[22] O. vSwitch, 2013, [Online]. Available: http://vswitch.org/

[23] L. KVM. [Online]. Available: http://www.linux-kvm.org/page/Main_Page.

[24] Microsoft, Hyper-v, [online]. Available: http://www.microsoft.com/en-us/ server-cloud/solutions/virtualization.aspx.

[25] Openstack, distributed virtual router, [Online]. Available: https://wiki. openstack.org/wiki/Neutron/DVR.

[26] Data plane development toolkit (DPDK), 2015, [Online]. Available: http:// dpdk.org.

[27] ETSI, Network function virtualisation-white paper1, SDN and openflow world congress, 2012, Darmstadt, Germany, [Online]. Available: http://portal.etsi. org/NFV/NFV_White_Paper.pdf.

[28] ETSI, Network function virtualisation-white paper2, SDN and openflow world congress, 2012, Darmstadt, Germany, [Online]. Available: http://portal.etsi. org/NFV/NFV_White_Paper2.pdf.

[29] ETSI, Network function virtualisation-white paper3, SDN and openflow world congress, 2012, Darmstadt, Germany, [Online]. Available: http://portal.etsi. org/NFV/NFV_White_Paper3.pdf.

[30] https://www.ibm.com/in-en/cloud/learn/virtualization-a-complete-guide

[31] https://opensource.com/resources/virtualization

[32] https://circleci.com/blog/top-6-benefits-of-virtualization/

[33] https://www.geeksforgeeks.org/virtualization-cloud-computing-types/

[34] https://www.blueplanet.com/resources/What-is-NFV-prx.html

[35] https://alepotech.medium.com/network-functions-virtualization-basics-to-benefits-e799cb3a865

[36] https://stlpartners.com/articles/telco-cloud/nfv-architectural-framework/

[37] https://github.com/josecastillolema/mini-nfv

[38] https://docs.openstack.org/arch-design/design.html

[39] https://vexxhost.com/blog/benefits-openstack-public-cloud-enterprises/

[40] https://www.eurovps.com/blog/openstack-cloud-benefits-challenges/

[41] https://www.blueplanet.com/resources/What-is-NFV-prx.html

[42] https://stlpartners.com/articles/telco-cloud/nfv-architectural-framework/

[43] https://computingforgeeks.com/openstack-core-services-explained/

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

[45] https://www.linuxtechi.com/step-by-step-instance-creation-flow-in-openstack/