NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

POSTGRADUATE PROGRAMME
"COMPUTER SCIENCE"

MSc THESIS

Using OSM for real-time redeployment of VNFs based on network status

Alexandros N. Kaltsounidis

Supervisor: **Hadjiefthymiades Stathes,** Professor

ATHENS

DECEMBER 2021

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
"ΠΛΗΡΟΦΟΡΙΚΗ"

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Χρησιμοποιώντας το OSM για τη μετακίνηση VNFs σε πραγματικό χρόνο σύμφωνα με την κατάσταση του δικτύου

Αλέξανδρος Ν. Καλτσουνίδης

Επιβλέπων: **Χατζηευθυμιάδης Ευστάθιος,** Καθηγητής

ΑΘΗΝΑ

ΔΕΚΕΜΒΡΙΟΣ 2021

MSc THESIS

Using OSM for real-time redeployment of VNFs based on network status

Alexandros N. Kaltsounidis
S.N.: CS2190008

SUPERVISOR: **Hadjiefthymiades Stathes,** Professor

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Χρησιμοποιώντας το OSM για τη μετακίνηση VNFs σε πραγματικό χρόνο σύμφωνα με την κατάσταση του δικτύου

Αλέξανδρος Ν. Καλτσουνίδης

Α.Μ.: CS2190008

ΕΠΙΒΛΕΠΩΝ: **Χατζηευθυμιάδης Ευστάθιος,** Καθηγητής

ABSTRACT

In this thesis we will be examining the Network Functions Virtualisation (NFV) framework as a suitable framework for implementing a network appropriate for Internet of Things (IoT), which needs to be flexible and scalable. More precisely, we will be focusing on how Open Source MANO (OSM) can be efficiently utilized in a solution that monitors the network status of Virtual Network Functions (VNFs) and in case of bad network status (e.g. network congestion) triggers the redeployment of affected VNFs to some other Virtual Infrastructure Manager (VIM) to prevent the underperformance of running services.

SUBJECT AREA: Internet of Things

KEYWORDS: IoT, NFV, OSM, MANO, network

ΠΕΡΙΛΗΨΗ

Στην παρούσα διπλωματική εργασία θα εξετάσουμε την Εικονικοποίηση δικτυακών λειτουργιών (Network Functions Virtualisation - NFV) ως την κατάλληλη αρχιτεκτονική για την υλοποίηση ενός δικτύου κατάλληλου για το Διαδίκτυο των Πραγμάτων (Internet of Things - IoT), το οποίο πρέπει να είναι ευέλικτο και επεκτάσιμο. Πιο συγκεκριμένα, θα επικεντρωθούμε στην αποτελεσματική αξιοποίηση του Open Source MANO (OSM) στην υλοποίηση μιας εφαρμογής που παρακολουθεί την κατάσταση του δικτύου των Εικονικοποιημένων δικτυακών λειτουργιών (Virtual Network Functions – VNFs) και σε περίπτωση κακής κατάστασης του δικτύου (π.χ. συμφόρηση του δικτύου) αναλαμβάνει τη μετακίνηση των επηρεαζόμενων VNFs σε κάποιον άλλο Διαχειριστή Εικονικής Υποδομής (Virtual Infrastructure Manager – VIM), για να αποτραπεί η πτώση στην απόδοση των ενεργών υπηρεσιών.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Διαδίκτυο των Πραγμάτων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: IoT, NFV, OSM, MANO, δίκτυο

AKNOWLEDGMENTS

Firstly, I would like to express my gratitude to my supervisor, Prof. Stathes Hadjiefthymiades for his invaluable guidance throughout writing this thesis.

My research would have been impossible without the aid and support of Res. Ass. Charalampos Andreou.

CONTENTS

1. INTRODUCTION 12
2. PROBLEM & STATE OF THE ART 14
3. PROPOSED SOLUTION 15
4. NFV, OSM AND OPENSTACK 16

4.1 Network Functions Virtualization (NFV) 16

4.1.1 Overview: 16

4.1.2 History 16

4.1.3 High level Analysis 17

4.1.4 Architectural Functional Blocks 18

4.1.4.1 Virtualised Network Function (VNF) 19

4.1.4.2 Element Management System (EMS) 19

4.1.4.3 NFV Infrastructure (NFVI) 19

4.1.4.3.1 Hardware Resources 20

4.1.4.3.2 Virtualisation Layer and Virtualised Resources 20

4.1.4.4 Virtualised Infrastructure Manager (VIM) 21

4.1.4.5 Orchestrator and VNF Manager 21

4.1.4.6 Service, VNF and Infrastructure Description 22

4.1.4.7 Operation Support Systems and Business Support Systems 22

4.1.4.8 Reference Points 22

4.2 Open Source MANO (OSM) 23

4.2.1 Overview 23

4.2.2 Scope and Functionality 24

4.2.3 Architecture 26

4.2.3.1 DevOps 27

4.2.3.2 User Interface (UI) Module 27

4.2.3.3 Service Orchestrator Module 28

4.2.3.4 Network Service to VNF Communication (N2VC) Module 28

4.2.3.5 VNF Configuration & Abstraction (VCA) 28

4.2.3.6 Resource Orchestrator (RO) Module 29

4.2.3.7 Monitoring Module (MON) ... 29

4.2.3.8 OSM Information Model (IM) Module ... 29

4.2.4 Descriptor Files ... 29

4.2.4.1 Network Service Descriptor (NSD) ... 30

4.2.4.2 Virtual Network Function Descriptor (VNFD) ... 31

4.2.5 OpenVIM ... 33

4.2.6 OSM through releases ... 35

4.3 OpenStack – MicroStack ... 36

4.3.1 OpenStack Overview ... 36

4.3.2 OpenStack's Software Components ... 37

4.3.3 MicroStack ... 40

5. DETECTION OF BAD NETWORK STATUS AND REDEPLOYMENT OF VNFS . 42

5.1 Outline ... 42

5.2 Extra Software ... 42

5.2.1 vnStat ... 42

5.2.2 iPerf3 ... 43

5.3 Setup of Infrastructure ... 43

5.3.1 Installing OSM ... 43

5.3.2 Installing MicroStack (OpenStack) ... 45

5.3.3 Setting up OpenStack as a VIM for OSM ... 46

5.3.4 Deploying the first service ... 46

5.4 Supervising Agent ... 47

5.4.1 OSM's North Bound REST API ... 47

5.4.2 Implementation of the Agent ... 48

5.4.3 The Agent in action – Test Case ... 49

6. CONCLUSIONS ... 52

7. FUTURE WORK ... 53

ABBREVIATIONS – ACRONYMS ... 54

REFERENCES ... 57

LIST OF FIGURES

Figure 4-1 High-level NFV framework 17

Figure 4-2 NFV reference architectural framework..19

Figure 4-3 OSM Mapping to ETSI NFV MANO..25

Figure 4-4 OSM Architecture 27

Figure 4-5 NSD object model 30

Figure 4-6 Simple and complex NS scenario..31

Figure 4-7 VNFD object model 32

Figure 4-8 Single-VM and Multi-VM VNF 33

Figure 4-9 OpenVIM Operational Model...34

Figure 4-10 OpenStack and OSM architectures together (simplified)...........................37

Figure 4-11 OpenStack Services...39

Figure 4-12 OpenStack Components 40

Figure 5-1 Status check of OSM’s components..44

Figure 5-2 Setup of OSM and OpenStack nodes 50

Figure 5-3 Test case of automatic redeployment..51

LIST OF TABLES

Table 4-1 Modes of OpenVIM operation..34

Table 5-1 Test case of automatic redeployment. *Time has been adjusted to start at 0:00 51

1. INTRODUCTION

In the recent decades, advancement in technology has been more rapid than ever before. Not only have there been numerous innovations in the hardware industry, where the production rates are faster than ever before, but software has evolved to keep up with the challenges of the market. More and more corporations are entering the digital world, which has led to the invention of models such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [1] that allow companies to quickly deploy their systems on the cloud and avoid the tedious task of purchasing and setting up their own hardware and/or software.

The innovation of cloud computing platforms, as well as the availability of low-cost, low-power sensor technology, improved telecommunication technology as is 5G that provisions a fast, low-latency, reliable, and energy-efficient network on a global scale, and growth in scientific fields related to data management have made Internet of Things [2] (IoT) possible. The Internet of Things is a global network and service infrastructure with connectivity and self-configuring capabilities, consisting of heterogeneous “things” that have identities, physical and virtual attributes, and are seamlessly and securely connected into the Internet. The IoT promises that devices will be connected anytime, anyplace, with anything and anyone, ideally using any network. By connecting billions of devices to the Internet, IoT has created a plethora of applications that target every aspect of human life, such as smart homes, smart cities, environmental monitoring, energy management, military, or industrial applications, etc. Although the establishment of IoT promises to create opportunities in a variety of fields and allow for the automation of several tasks, it is inevitable that challenges will arise, and most importantly regarding the network infrastructure.

In the pre-IoT world, network traffic was mainly created by humans and the number of connected devices was limited due to its partial dependency on the population size. However, in the world of IoT, the number of connected devices and consequently generated traffic will significantly increase. To deal with the increased network traffic, present-day networks must be designed to be flexible and scalable. It is worth noting that not all devices in IoT have the same network requirements concerning bandwidth, since one sensor could be transmitting a few bytes per month (e.g., a fire detection sensor) and another one transmitting megabytes per second (e.g., surveillance camera), latency, and other factors.

A typical IoT solution is characterized by many devices (i.e., things) that may use some form of gateway to communicate through a network to an enterprise back-end server that is running an IoT platform that helps integrate the IoT information into the existing enterprise. In some cases, a decentralized version of IoT is proposed where Fog Computing [3] (also referred to as edge computing) is used to minimize network load on the cloud infrastructure and improve responsiveness for latency-sensitive applications. Edge computing means that computation and data storage happen closer to the devices and raw data do not have to be sent to a central server to be processed. Taking the above points into consideration, it is clear that IoT needs a flexible and scalable network that can be reconfigured at will, as well as services which can be redeployed in an automated way. For example, if part of the network is saturated due to a specific service, that could be detrimental to some other latency-dependent service, e.g. one that processes real time medical data of patients.

We believe that Network Functions Virtualization (NFV) [4] is an appropriate framework for the implementation of a suitable IoT network. NFV is a framework created by ETSI

(European Telecommunications Standards Institute) that aims to turn network functions previously implemented using dedicated hardware into software (i.e., Virtual Network Function - VNF) that can be run on top of standard general-purpose hardware. The NFV (Virtual) Infrastructure consists of hardware resources which are provided as virtualized resources with the help of a virtualization layer. A key component of NFV is the Management and Orchestration (MANO) module, composed of three sub-modules, the VNF Orchestrator, the VNF Manager and the Virtualized Infrastructure Manager.

In Chapter 2 of this thesis, we will describe the problem we are trying to deal with and related work. In Chapter 3 we will present an overview of our proposed solution. In Chapter 4 we will focus on the theoretical background relative to the frameworks and software solutions that will be utilized in our solution. Chapter 5 will deal with the setup, implementation, and experimental results of our solution. Finally, in Chapter 6 we will draw the conclusions, while in Chapter 7 we will present ideas for future work on the current subject.

2. PROBLEM & STATE OF THE ART

We have already stated that the Internet of Things has requirements not satisfied by traditional networks and changes will need to take place if the full capabilities of IoT are to be utilized. Although technology has greatly progressed to fulfill some of these requirements and allow the realization of IoT, there still exist many challenging issues to be addressed and the field offers a great number of opportunities for research. These challenges include topics such as network foundation, security and privacy, energy efficiency, as well as managing heterogeneity within the system.

A lot of attention is drawn to evolving current network infrastructure. Current networks have not been designed with IoT in mind and their limited capabilities in terms of scalability, availability, manageability, and mobility/flexibility raise barriers to the establishment of IoT. With an astonishing number of new devices being connected to the network along with services over them, it is obvious that rates of bandwidth utilization will skyrocket. Moreover, in such a heterogenous system it is certain that different applications will have varied network requirements and not all of them are of the same importance (e.g., a medical application and a weather forecasting application).

It could often be the case that applications coexisting in a network are competing for its resources and in some cases the performance of one be undermined due to another. The focus of this thesis is determined around such a scenario where a service is underperforming because of poor network status, which could be the aftermath of unusually high-traffic, faulty hardware, or other causes. We will attempt to provide a complete and efficient solution to this challenge which falls under the framework of Network Functions Virtualization.

It is worth mentioning that in the last decades the research community has shown great attention to IoT, its challenges and its prospects [5-7]. The idea of utilizing NFV and even SDN (Software-Defined networking) [8] to implement a network suitable for IoT has been presented and partially explored in [9] and [10]. The ETSI NFV is a relatively new, but well-established and very promising framework that has attracted the interest of network operators, hardware manufacturers, and researchers alike [11-14].

3. PROPOSED SOLUTION

We have previously expressed our confidence in NFV being a suitable framework for an IoT network, and specifically for the challenge we are dealing with, as it offers network control for the IoT stack.

With the intention of implementing a NFV Management and Orchestration (MANO) stack, we have opted with OpenStack [15] as our Virtual Infrastructure Manager (VIM) and Open Source MANO (OSM) [16] as the Orchestrator and Virtual Network Function (VNF) Manager. These open source platforms allow for effortless setup of Network Services. OpenStack manages and provides the infrastructure on which services will be deployed, while OSM allows for lifecycle management of services, including automations such as auto-scaling of services. OSM’s north bound interface (NBI) offers a simple way to manage services though http requests, and we intend to create a supervising agent with the responsibility of overseeing the network status of “critical” services and if needed communicate with OSM to redeploy that service elsewhere and thus prevent underperformance of the system.

4. NFV, OSM and OpenStack

4.1 Network Functions Virtualization (NFV)

In this section we will present the Network Functions Virtualization Framework, what it is, its main purpose, its benefits, its high-level architectural view of the framework, as well as an analysis of its architecture.

4.1.1 Overview:

In this day and age, networks contain a large and ever-increasing variety of proprietary hardware appliances. Launching a new network service often requires physical reconfiguration on the network, most of the times needing to install new devices which significantly increases costs of operation. If we consider costs of energy, space requirements, capital investment challenges and the resources needed to design, integrate, and operate these complex hardware-based appliances, it is obvious that the current infrastructure model is not in accordance with the rapid rate of changes taking place. Moreover, as technology and services innovation accelerates, hardware lifecycles are becoming shorter and shorter, inhibiting the roll out of new revenue earning network services.

Network Functions Virtualization (NFV) proposes to address the problems mentioned above by evolving standard IT virtualization technology to consolidate many network equipment types onto industry standard high-volume servers, switches, and storage, which could be located in a variety of Network Function Virtualization Infrastructure Points of Presence (NFVI-PoPs) including but not limited to datacenters, network nodes and in end user premises. Contrary to current network architecture management, the virtualized network's architecture introduces a multitude of resources that can be managed and operated in a variety of levels, as well as be interconnected, coordinated, and automated to provide desired network functionality. This diversity of available resources requires a framework that can communicate, manage, and orchestrate them all harmoniously.

4.1.2 History

In November 2012 seven of the world's leading telecoms network operators – namely, AT&T, BT, CenturyLink, China Mobile, Deutsche Telekom, Orange, Telecom Italia, Telefonica, and Verizon - joined forces to form the ETSI Industry Specification Group (ISG) for NFV, to create a standard for NFV, shortly after having published a white paper [17] at a conference in Darmstadt, Germany to shed light onto the potentials of the NFV framework. Since its creation, the ETSI NFV ISG has grown to over 290 organizations, including 38 of the world's biggest service providers, network operators and IT vendors, which provides an unparalleled collective service provider view of business requirements and strategic direction. The intensity of work remains undiminished and the NFV ISG periodically evolves its internal structure to deal with a very high workload and to accommodate the expansion of scope due to greater technical awareness of the topic.

ETSI NFV undertakes work in 2-year phases, which are also referred to as "Releases". Release 1 (2013-2014) aimed to help the industry build a culture and share a common understanding on the important concepts in network virtualization. With Release 2 (2015-2016) [18] and Release 3 (2017-2018) [19] NFV's standard were completed and NFV was marked "ready" for global deployment and operations. Release 4 (2019-2020) [20] as well as the planned Release 5 (2021-2022) [21] focus to enrich the features of NFV framework.

With Phase 3 of its work completed, specifications of ETSI NFV include an infrastructure overview, updated architectural framework and descriptions of the compute, hypervisor, and network domains of the infrastructure, as well as service quality metrics, security and trust, resilience and Management and Orchestration (MANO). ETSI publications are emerging in a swift rate to deal with challenges and opportunities that arise in the world of technology.

4.1.3 High level Analysis

We have already mentioned that NFV is different to current non-virtualised networks, where NFs are implemented as a combination of vendor specific software and hardware – often referred to as network nodes or network elements – in a way that software is decoupled from hardware and the deployment and operation of network functions is much more flexible and dynamic.

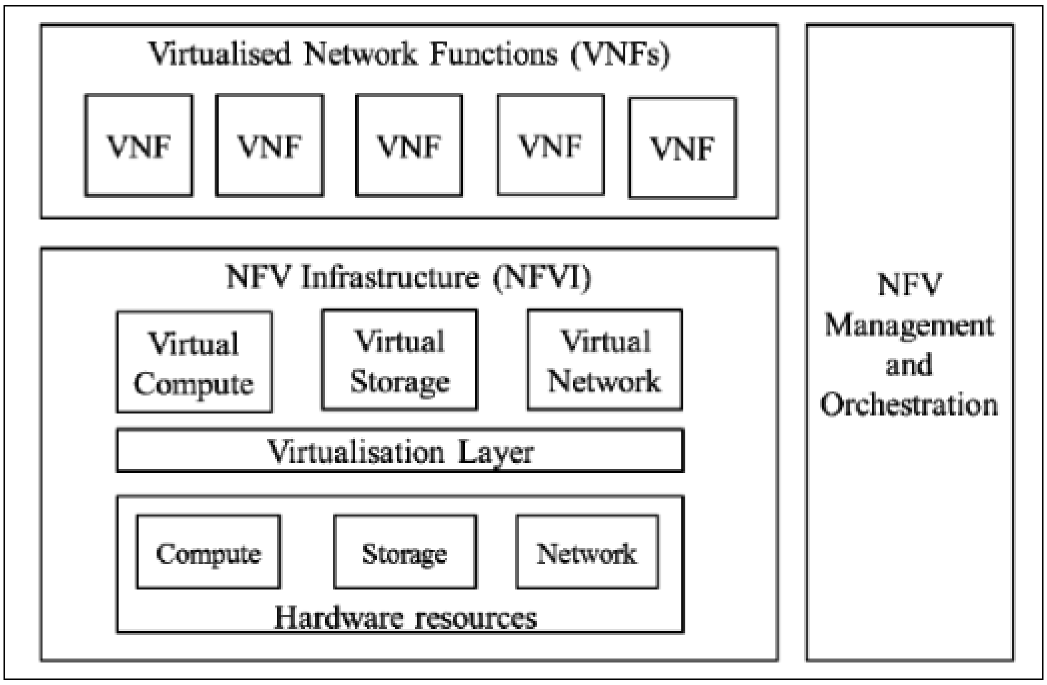


Figure 4-1 High-level NFV framework

Figure 4-1 illustrates the high-level NFV framework. The three key structure units of ETSI NFV Architecture are the following:

1. Virtualised Network Function (VNF) – the software implementation of a network function designed to run over the NFVI.
2. Network Function Virtualisation Infrastructure (NFVI) – includes the various hardware and software components which build up the environment in which VNFs are deployed.

3. Network Function Virtualisation Management and Orchestration (NFV-MANO) – handles the orchestration and lifecycle management of physical and/or software resources that support the infrastructure virtualisation, as well as the lifecycle management of VNFs. NFV-MANO focuses on all virtualisation-specific management tasks in the NFV framework and communicates with the VNF and NFVI blocks to do so.

The NFV framework enables dynamic creation and management of VNF instances and the relationships between them regarding data, control, management, dependencies, and other attributes. Note that multiple VNFs can be chained together to work as a multifunction, e.g. a chain of VNFs in a web server (firewall, NAT, load balancer), or function as standalones. A chain of one or more interconnected Network Functions is called a Network Service.

It is obvious from the architecture that software has been completely detached from hardware appliances. VNFs can be deployed into several hardware machines that have Compute capabilities, Storage and Network interfaces. This allows for a far more flexible and easier to scale infrastructure, as for example new compute units and/or storage, memory, etc. can be added with ease, should it be needed.

4.1.4 Architectural Functional Blocks

The NFV architectural framework can be identified by its functional blocks and the main reference points between them. These functional blocks are:

- Virtualised Network Function (VNF)
- Element Management System (EMS)
- NFV Infrastructure, including hardware, virtualised resources, and virtualisation layer
- Virtualised Infrastructure Manager (VIM)
- Orchestrator
- VNF Manager
- Service, VNF and Infrastructure Description
- Operations and Business Support Systems (OSS/BSS)

The functional blocks and along with their reference points in the NFV framework are depicted in Figure 4-2.

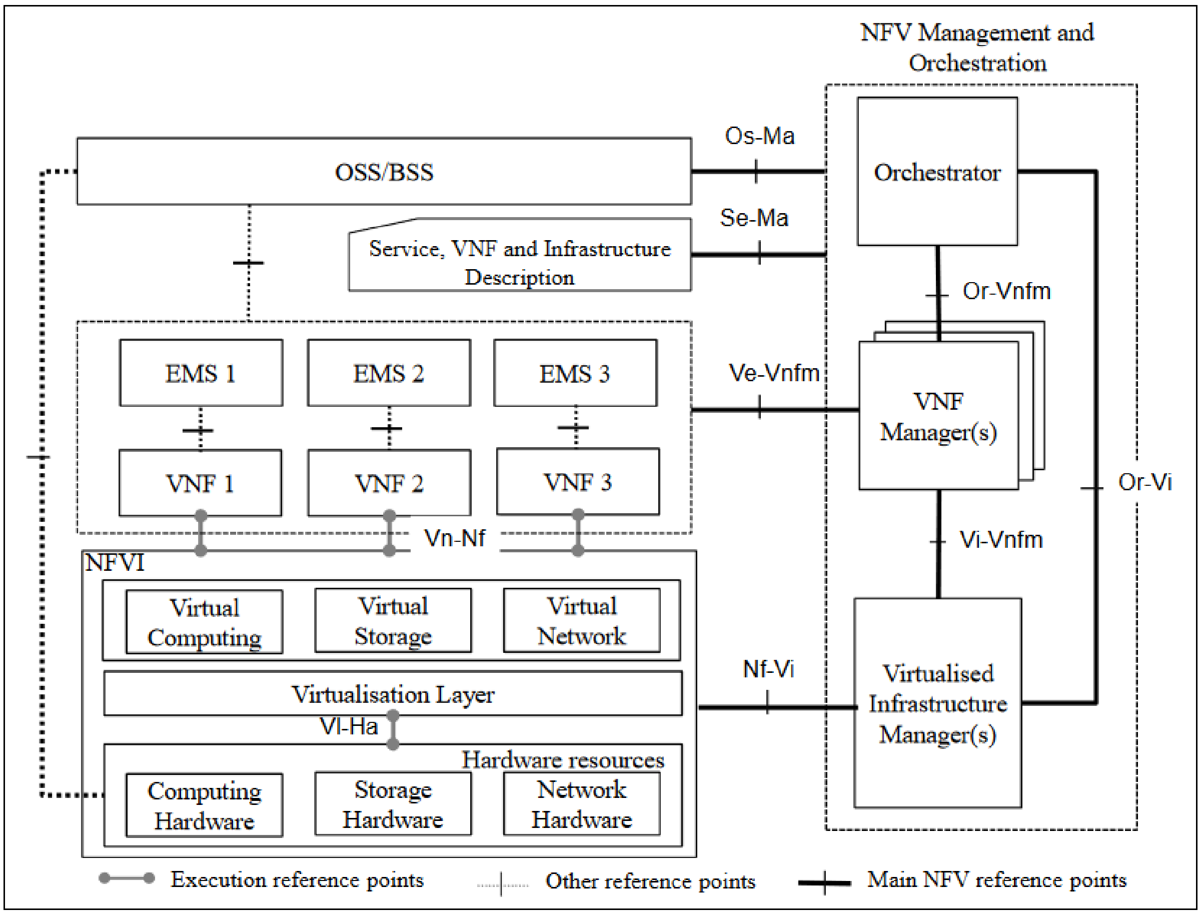


Figure 4-2 NFV reference architectural framework

Each of the functional blocks and their reference points will be briefly described below.

4.1.4.1 Virtualised Network Function (VNF)

A VNF is a virtualisation of a network function in a legacy non-virtualised network, i.e. a Physical Network Function (PNF). Whether a network function is virtualised (VNF) or not (PNF), this has no impact on its functional behavior and external operational interfaces.

A VNF can be composed of multiple internal components, e.g. a VNF can be deployed over multiple Virtual Machines (VMs) where each VM hosts a single component of the network function, or the whole VNF can be deployed in just a single VM.

4.1.4.2 Element Management System (EMS)

The role of the Element Management System is to handle the typical management functionality for one or more VNFs.

4.1.4.3 NFV Infrastructure (NFVI)

The NFVI refers to all the hardware and software components that create the environment in which VNFs are to be deployed, executed, and managed. An installment that is purposed to be part of the NFVI is usually referred to as a NFVI-Point of Presence (NFVI-PoP). The NFVI might consist of multiples NFVI-PoPs, which can exist on several different geographical locations and in that case, the network interconnecting these facilities is also considered as part of the NFVI.

4.1.4.3.1 Hardware Resources

The hardware resources of NFVI include computing, storage and network that provide processing, storage, and connectivity to VNFs through the virtualisation layer. Computing hardware is assumed to be COTS (Commercial Of-The-Self) hardware and not purpose-built. Storage can either be Network-attached Storage (NAS) or storage that resides on the server itself. Network resources are made of switching functions, e.g. routers, and wired or wireless links, and they can span over different domains. The two types of networks in NFVI are the NVFI-PoP network and the Transport network. NFVI-PoP network is the network that interconnects the computing and storage resources in a NFVI-PoP and provides external connectivity to that PoP. On the other hand, Transport network is the network that interconnects NFVI-PoPs, NFVI-PoPs to other networks, or NFVI-PoPs to other network appliances or terminals outside of the NFVI-PoPs.

4.1.4.3.2 Virtualisation Layer and Virtualised Resources

The virtualization layer is in charge of creating an abstraction layer over the hardware resources and thus decoupling the VNF software from the underlying hardware and ensuring a hardware independent lifecycle for the VNFs. 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 virtualised infrastructure.
- Providing the required virtualised resources to the VNF.

The virtualisation layer ensures that VNFs are decoupled from hardware resources and therefore that the software can be deployed on different physical hardware resources. Usually, this functionality is provided for computing and storage resources in the form of hypervisors and virtual machines. A VNF can be deployed on one or more VMs. However, the NFV architectural is not restricted to a specific virtualisation layer solution but expects to use virtualisation layers with standard features and open execution reference points towards VNFs and hardware. In some cases, VMs may have direct access to hardware resources for better performance, e.g. a network interface card or a PCI-passthrough. Virtualisation solutions different to hypervisors may include software running on top of a non-virtualised server by means of an operating system (OS), or VNFs implemented as an application that can run on virtualised infrastructure or on bare metal. Regardless, the operation of the VNF should be independent of its deployment scenario.

As far as network virtualisation is concerned, network hardware is abstracted by the virtualisation layer to realize network paths that provide connectivity between VMs of a

VNF and/or between different VNF instances. Techniques that allow the above functionality, including network abstraction layers that isolate resources via virtual networks and network overlays, include - but are not limited to – the following: Virtual Local Area Network (VLAN), Virtual Private LAN Services (VPLS), Virtual Extensible Local Area Network (VxLAN), and Network Virtualisation using Generic Routing Encapsulation (NVGRE).

4.1.4.4 Virtualised Infrastructure Manager (VIM)

As far as NFV is concerned, a virtualized infrastructure manager’s duty is to provide the functionalities that are used to control and manage the interaction of a VNF with computing, storage and network resources assigned to it, and their virtualisation. Thus, a VIM has all the information needed to manage hardware resources and performance, and performs:

- Resource management, for:
    - Inventory of software (e.g., hypervisors), computing, storage and network resources dedicated to NFVI.
    - Allocation of virtualisation enables (e.g., VMs over hypervisors), compute resources, storage, and relevant network connectivity.
    - Management of infrastructure resource and allocation, e.g. provide more resource to a VM, manage energy consumption, etc.
- Operations, for:
    - Insight into and management of the NFVI
    - Root cause analysis of performance issues from the NFVI perspective
    - Collection of infrastructure fault information
    - Collection of information for capacity planning, monitoring, and optimization

One or more Virtualised Infrastructure Managers may be deployed.

4.1.4.5 Orchestrator and VNF Manager

The Orchestrator oversees the orchestration and management of NFV infrastructure and software resources, as well as realizing network services on NFVI.

A VNF Manager is in charge of VNF lifecycle management, i.e. instantiation, update, query, scaling, termination. Multiple VNF Managers may be deployed. One VNF Manager may serve multiple VNFs, or there may be a VNF Manager for each VNF.

The Orchestrator along with the VNF Manager(s) and the VIM(s) build up the NFV Management and Orchestration layer. This layer is a key component of NFV architecture. It handles the information about resources from VIM’s managing and across multiple VIMs through Resource Orchestration (RO). Moreover, through Service Orchestration (SO) it performs the management of Network Services and VNFs.

It is thanks to the cooperation of its blocks that VNFs are successfully allocated the required resources, instantiated, and then managed throughout their lifecycle with update, query, scaling, or termination operations executed when needed. Furthermore, the organization of this layers into distinct blocks allows it to be flexible and not limited to one solution, for example to a single implementation of a VIM.

4.1.4.6 Service, VNF and Infrastructure Description

This is a collection of information about the VNF deployment template, VNF Forwarding Graph (i.e. how VNFs are chained together), service-related information, as well as NFV infrastructure information models.

4.1.4.7 Operation Support Systems and Business Support Systems

This refers to the Operations/Business support systems i.e., operator’s back-end systems that manage network, services, customers, products, and orders.

4.1.4.8 Reference Points

In the NFV framework the Reference Points play the vital role of interconnecting the various functional blocks. They assure that flow of information between blocks is consistent. Below is a detailed report of those Reference Points and their purpose within the boundaries of the framework.

<u>Virtualisation Layer – Hardware Resources (VI-Ha).</u> This reference point interfaces the virtualisation layer to hardware resource to create an execution environment for VNFs and collect relevant hardware resource state information needed for managing VNFs independently of the underlying hardware platform.

<u>VNF – NFV Infrastructure (Vn-Nf).</u> 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 to guarantee hardware independent lifecycle, performance, and portability requirements of the VNF.

<u>Orchestrator – VNF Manager (Or-Vnfm).</u> This reference point is responsible for information exchange between NFV Orchestrator and VNF Manager. It is used for collecting state information of the VNF essential for network service lifecycle management, forwarding configuration information to the VNF Manager, as well as for resource related requests, e.g. authorization, validation, allocation, etc.

<u>Virtualised Infrastructure Manager – VNF Manager (Vi-Vnfm).</u> This reference point handles resource allocation requests by the VNF Manager and exchange of virtualised hardware resource configuration and state information, e.g. events.

<u>Orchestrator – Virtualised Infrastructure Manager (Or-Vi).</u> This reference point is used for resource reservation and/or allocation requests by the Orchestrator and exchange of virtualised hardware resource configuration and state information.

NFVI – Virtualised Infrastructure Manager (Nf-Vi). This reference point is used for assignment of virtualised resources in response to resource allocation requests, forwarding of virtualised resources state information, and exchange of hardware resources configuration and state information.

OSS/BSS – NFV Management and Orchestration (Os-Ma). This reference point is responsible for information exchange between OSS/BSS and NFVO regarding requests for network service and/or VNF lifecycle management (i.e. instantiation, update, query, scaling, termination), policy management (e.g. authorization, access), NFV related state information, accounting and usage records, NFVI capacity and inventory information, as well as data analytics.

VNF/EMS – VNF Manager (Ve-Vnfm). This reference point is used for requests relative to VNF lifecycle management, exchanging configuration information and state information necessary for network service lifecycle management.

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 NFVI information models. The information provided is used by NFV management and orchestration.

4.2 Open Source MANO (OSM)

4.2.1 Overview

The Open Source MANO [16] project was initiated by ETSI and proposes an implementation of NFV's Management and Orchestration layer. The ETSI organization firmly supported the idea that open source software can facilitate the implementation of an ETSI aligned NFV architecture, provide practical and essential feedback to the ETSI ISG NFV and increase the likelihood of interoperability among NFV implementations and therefore released the OSM project under the Apache Public License 2.0 [22].

ETSI's OSM group has been developing an open source NFV MANO stack using well established open source tools and working procedures. ETSI OSM complements the work of ETSI NFV and vice versa, maximizing innovation, efficiency, and time to market and ensuring a series of reference conformant implementations.

Participation to ETSI OSM is open to members and non-members of ETSI, as well as individual developers and end users from all over the worlds. The founding members of OSM include Telefónica, BT, Canonical, Intel, Mirantis, RIFT.io, Telekom Austria Group, and Telenor, while amongst its current 149 are some of the largest corporations in the field of technology, such as Amazon, Dell, Oracle, SK Telecom, Red Hat, and Whitestack.

First release of OSM (Release ZERO) came out in 2016, and since then there have been another ten releases, with Release TEN being the latest, each improving and adding features to OSM. Furthermore, a significant number of white papers and articles

[23] have been published, while ETSI OSM has held several Plugtests and Hackfests [24].

The open source nature of the project, along with the community model in which all parts are assisting, whether it is with development, testing or bug fixing, gives it the potential to become much more than just another MANO stack. OSM MANO embraces the challenges of real world scenarios, and with features such as Multi-VIM support, Enhanced Platform Awareness (EPA) support, Multi-site Network Services, a uniform deployment model, both a command line interface (CLI) and a web based graphical user interface (GUI), and detailed documentation it is friendly towards network engineers, setting the fundamentals to creating a simple, easy to use and high-performance ready MANO stack.

4.2.2 Scope and Functionality

The goal of ETSI OSM is the development of a community-driven high-quality Edge-to-Edge Network Service Orchestrator (E2E NSO) for telco services, capable of modelling and automating network services. OSM contributes greatly to the advancement of NFV technologies and standards, and provides a way to allow for a broad ecosystem of VNF vendors, test and validate the joint interaction of the orchestrator with the other components it interacts with, i.e. commercial NFV infrastructures (NFVI and VIM) and Network Functions (either VNFs, PNFs, or Hybrids NFs).

OSM is built on four architectural principles, these are:

- Layering: Aligned with ETSI-NFV architecture, there must be clear delineation between the layers and modules.
- Abstraction: Moving up or down the layers should offer clear differentiation in the levels of abstraction/detail presented.
- Modularity: Even within layers, clear modularity enabled thanks to a plugin model allows for painless module replacements as OSM community develops.
- Simplicity: Solution must have the minimal complexity to avoid over-engineering that would result in a complicated final product.

OSM’s approach intends to minimize integration efforts thanks to four key aspects:

1. A well-established Information Model (IM), aligned with ETSI NFV, that is capable of modelling and automating the full lifecycle of Network Functions, Network Services, as well as Network Slices, from their initial deployment (i.e. instantiation, Day-0, and Day-1) to their daily operation and monitoring (Day-2). The IM is created to be infrastructure-agnostic so that the same model can be used to deploy an element (e.g., a VNF) in a variety of VIM types and transport technologies.
2. OSM offers a unified northbound interface (NBI), based on NFV SOL005, which enables the full operation of system and services under its control.
3. The extended concept of “Network Service” in OSM, so that a NS can span across the different domains identified (i.e. virtual, physical and transport), and thus control the lifecycle of a NS consisted of VNFs, PNFs, and HNFs in an undistinguishable manner along with on demand transport connections amongst different sites.

4. OSM can also manage the lifecycle of Network Slices, assuming if required the role of Slice Manager, extending it to also support an integrated operation.

The OSM community has defined an expansive scope for the project covering both design-time and run-time aspects that are related to service delivery for telecommunications service provider environments. The express goal is that OSM code base can be leveraged in such environments as-is in a roll-your-own context, or in whole and/or part of a commercial product.

The figure below demonstrates the approximate mapping of scope between OSM's components and ETSI NFV MANO logical view.

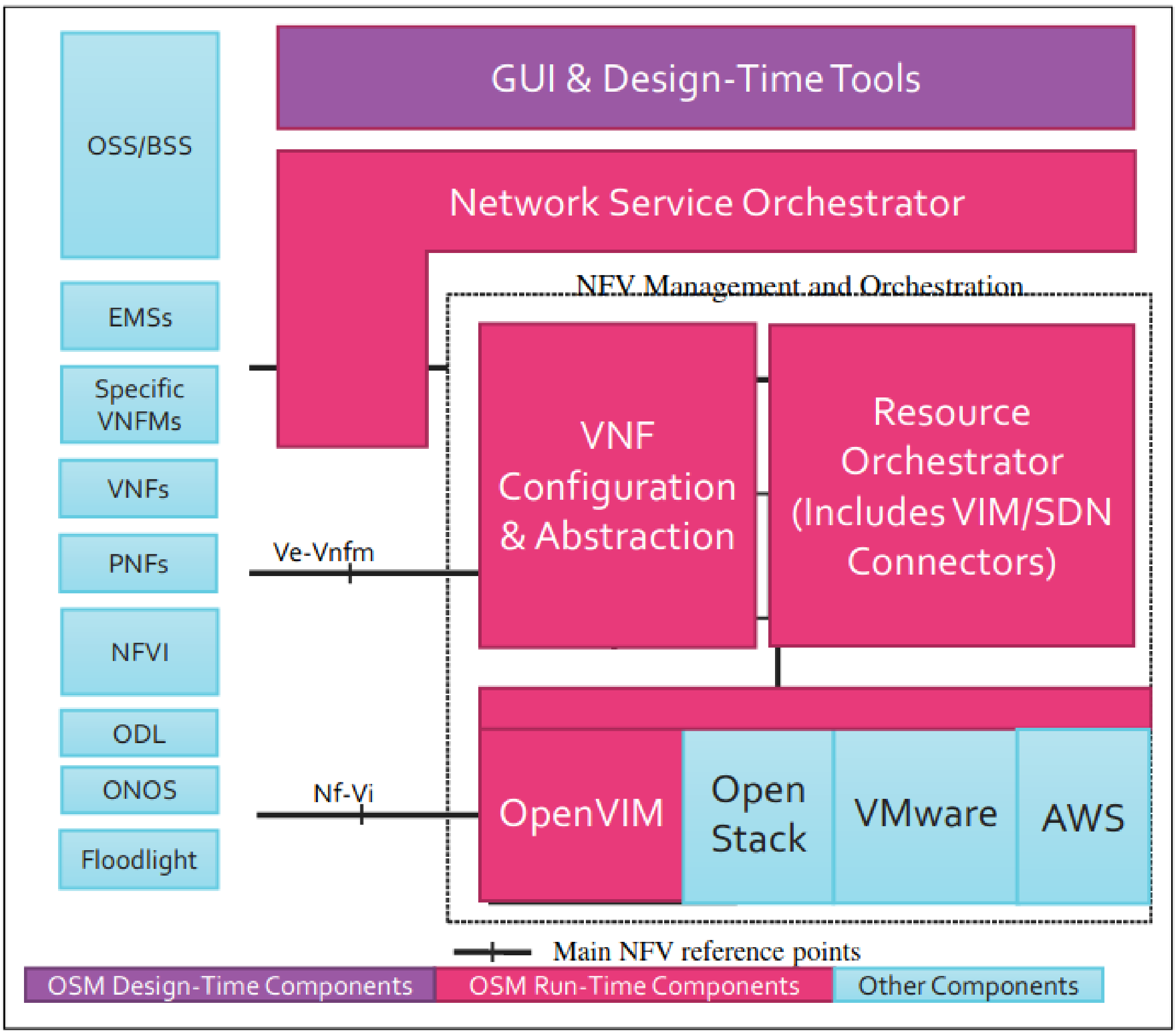

OSS/BSS
EMSs
Specific VNFMs
VNFs
PNFs
NFVI
ODL
ONOS
Floodlight
GUI & Design-Time Tools
Network Service Orchestrator
NFV Management and Orchestration
VNF Configuration & Abstraction
Resource Orchestrator (Includes VIM/SDN Connectors)
Ve-Vnfm
Nf-Vi
OpenVIM
Open Stack
VMware
AWS
Main NFV reference points
OSM Design-Time Components
OSM Run-Time Components
Other Components

Figure 4-3 OSM Mapping to ETSI NFV MANO

Run-Time Scope

The run-time scope of OSM includes:

- An automated Service Orchestration environment that enables and simplifies the operational considerations of the various lifecycle phases involved in running a complex service based on NFV.

- A superset of ETSI NFV MANO where the salient additional area of scope includes Service Orchestration but also explicitly includes provision for SDN control.
- Delivery of a plugin model for integrating multiple SDN controllers.
- Delivery of a plugin model for integrating multiple VIMs, including public cloud-based VIMs.
- Delivery of a plugin model for integrating multiple monitoring tools into the environment.
- One reference VIM that has been optimized for Enhanced Platform Awareness (EPA) to enable high performance VNF deployments.
- An integrated “Generic” VNF Manager (VNFM) with support for integrating “Specific” VNFMs.
- Support to integrate Physical Network Functions into an automated Network Service deployment.
- Being suitable for both Greenfield and Brownfield deployment scenarios.
- GUI, CLI, Python based client library and REST interfaces to enable access to all features.

Design-Time Scope

The design-time scope of OSM includes:

- Support for a model-driven environment with Data Models aligned with ETSI NFV MANO.
- The capability for Create/Read/Update/Delete (CRUD) operations on the Network Service Definition.
- Simplifying VNF Package Generation.
- Supplying a Graphical User Interface (GUI) to accelerate the network service design time phase, VNF on-boarding and deployment.

4.2.3 Architecture

In this section we will present the architectural model of OSM and its components.

It is worth mentioning that the development of OSM did not start from scratch, but three pre-existing elements were used for some key components:

1. OpenMano [25], delivered by Telefónica, as Resource Orchestrator.
2. RIFT.ware [26], by RIFT.io, as Service Orchestrator.
3. Juju [27], by Canonical, for VNF Manager, also referred to as Virtual Network Functions Configuration and Abstraction (VCA) module.

A comprehensive view of OSM’s logical blocks that represent the functionality provided by OSM are shown in Figure 4-4.

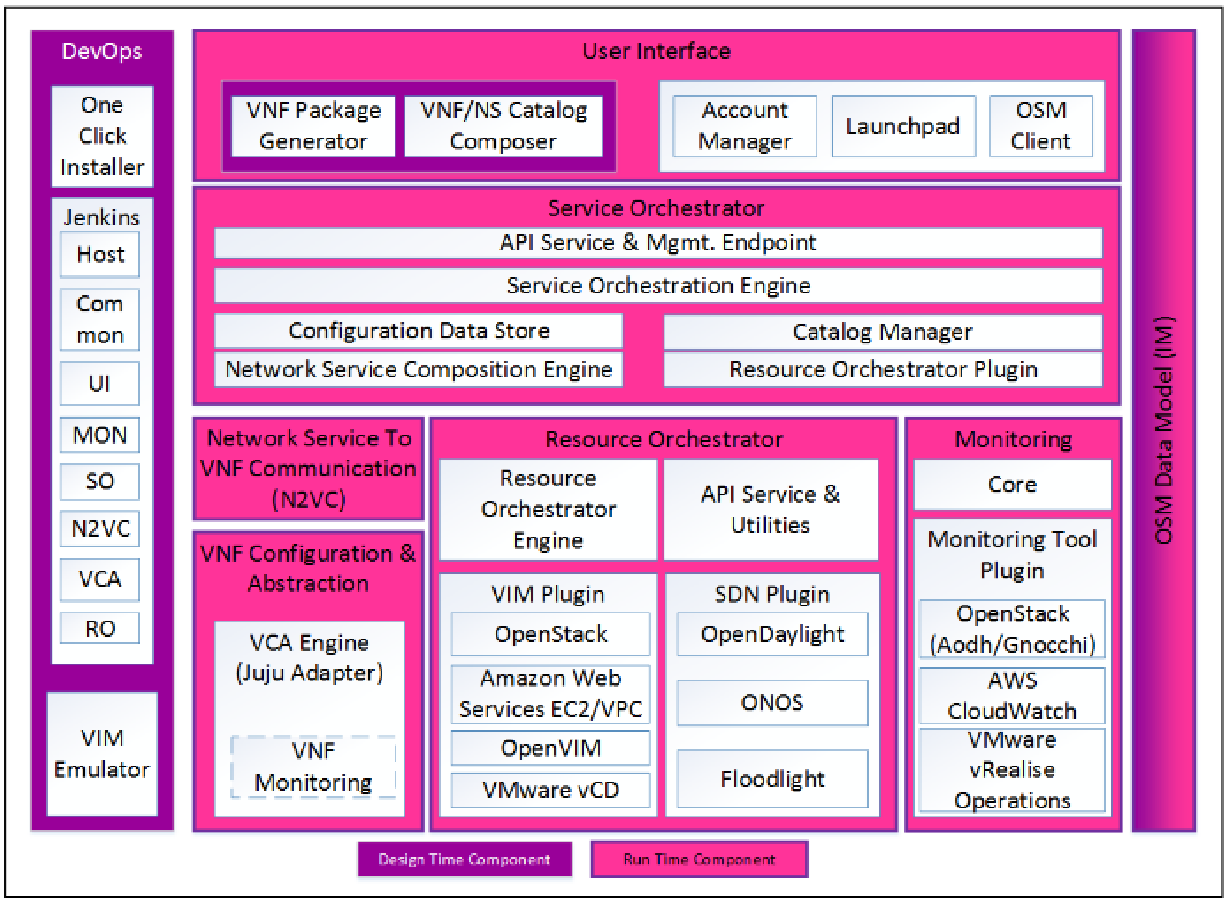


Figure 4-4 OSM Architecture

Each logical block will be briefly described in the following sub-sections in a top-to-bottom fashion.

4.2.3.1 DevOps

Since OSM is an open source projects with an astonishing number of contributors, it is essential that there exists a dedicated module responsible for the Continuous Integration (CI) and Continuous Development (CD) workflow to deliver a world class experience for OSM developers. Since this module is not an essential part of the platform’s workflow (though it is vital for its advancement), it will not be examined in this thesis.

4.2.3.2 User Interface (UI) Module

The User Interface (UI) Module consists of both design-time and run-time elements and is responsible for providing a simple and elegant way for the end user to access and interact with OSM.

The design-time parts of UI are tools built to aid in the creation of properly-formed VNF/NS packages, which contain everything required to allow for successfully deploying a VNF/NS. These tools are the Catalog Composer, and the Package Generator.

The first tool, Catalog Composer, aids to create a descriptor file for a VNF/NS through the GUI, while the former, Package Generator, creates the package, which includes the descriptor file, a README file, a checksum file, and any other files needed by the service.

The run-time parts of UI are the Account Manager, the Launchpad and OSM Client.

Account Manager is in charge of managing and configuring credentials for each part of the platform and regulating access per user and per project.

The Launchpad is a web-based interactive GUI through which the user has access to the core functionality of OSM, such as managing the lifecycle of a NS, uploading VNF/NS packages, adding, or removing VIM accounts, etc. It also visualizes real time statistics regarding network services and a view of the created network topologies.

The OSM Client is a CLI client, written in python, that provides remote interaction with OSM.

4.2.3.3 Service Orchestrator Module

The API Service & Management Endpoint component is providing the primary API endpoint into OSM.

The Service Orchestration Engine is responsible for all aspects of service orchestration including lifecycle management and service primitive execution. It is effectively the “master” orchestration component in the system that rules the workflow throughout OSM, and supports the concepts of multi-tenancy, projects, users, and enforces role-based access controls.

The Configuration Data Store’s role is to persistently store the SO state, especially in the context of VNF and NS deployment records.

The Network Service Composition Engine supports NS and VNF descriptor composition. It checks that the composed descriptors conform to the defined YANG (Yet Another Next Generation) [28] schema.

The Catalog Manager supports CRUD lifecycle operations on the defined VNF/NS descriptors and packages.

The Resource Orchestrator Plugin provides a much needed interface for integrating the Resource Orchestrator.

4.2.3.4 Network Service to VNF Communication (N2VC) Module

This module is responsible for the plugin framework between the SO and the VNF Configuration and Abstraction layer.

4.2.3.5 VNF Configuration & Abstraction (VCA)

The VNF Configuration and Abstraction (VCA) layer is responsible for enabling configurations, actions, and notification to and from the VNFs and/or Element Managers. Then backed by Juju, it provides the facility to create generic or specific indirect-mode VNFMs, via charms that can support the interface that the VNF/EM chooses to export.

4.2.3.6 Resource Orchestrator (RO) Module

The API Service & Utilities endpoint provides the interface into the RO (for SO to consume), as well as several utilities for internal to RO consumption.

The Resource Orchestrator Engine is in charge of managing and coordinating resource allocation across multiple geo-distributed VIMs and multiple SDN controllers.

The VIM and SDN plugins allow the connection between the Resource Orchestration Engine and the specific interface provided by the VIM or SDN controller respectively.

4.2.3.7 Monitoring Module (MON)

The role of the Monitoring Module is to interact with and leverage existing or new monitoring systems, and not replicate or compete with these systems. It mainly drives monitoring configuration updates to the external monitoring tools and forwards actionable events into the SO. These actionable events may be either directly triggered by running NS/VNFs or deduced by the external monitoring tools.

One of the most powerful things OSM is delivering as part of the Monitoring Module is the ability to correlate telemetry related to the VMs and VNFs to the relevant Network Services. Automated correlation is expected to provide a considerable user experience improvement to OSM users and drive up efficiency for operators in a telecommunications environment.

The Monitoring Module's message bus is realized by Apache Kafka [29], a popular fault-tolerant message passing system that supports a publish-subscribe model which fits into the MON's architecture. Moreover, other established tools, such as Prometheus [30] (a monitoring system and time series database) and Grafana [31] (an open source analytics and monitoring solution) are utilized.

4.2.3.8 OSM Information Model (IM) Module

OSM is based on a model-driven architecture. The architectural direction has always been to use the same model as the basis of both the design-time capabilities and the run-time capabilities. The OSM Information Model Module was created to be the single point of authority on the OSM data model that is leveraged by the different components. This helps the move towards a methodology where two of the most important data models in the system, the VNF Descriptor (VNFD) and the Network Service Descriptor (NSD), can be shared in their innate forms between components. OSM modules can act authoritatively on the relevant parts of the VNFD/NSD.

4.2.4 Descriptor Files

OSM's IM uses descriptors, i.e. configuration templates that define the main properties of managed objects in a network. Descriptors define deployment and operational behavior for each component, as well as their dependencies to other elements.

Descriptors are written in YAML language [32], a human-readable and easy to understand markup language oriented for data.

4.2.4.1 Network Service Descriptor (NSD)

The Network Service Descriptor (NSD) is the top-level structure that defines the topology of the network, wrapping all the references to descriptors corresponding to other components.

The NSD consists of static information elements and describes deployment flavors of the network service. The NSD is used by the NFVO to instantiate a NS.

The following four information elements are defined apart from the top-level network service:

- Virtual network function (VNF) information element
- Physical network function (PNF) information element
- Virtual Link (VL) information element.
- VNF forwarding graph (VNFFG) information element

The high-level object model for NSD is shown in the Figure below.

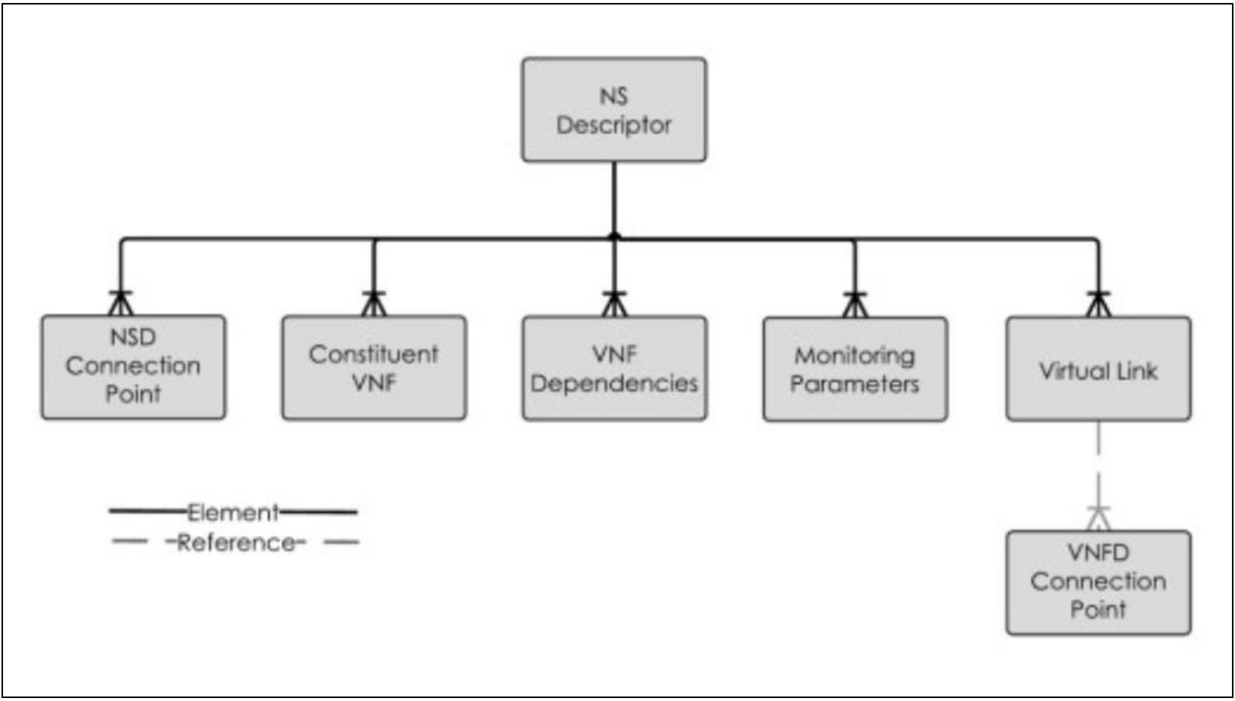


Figure 4-5 NSD object model

The NSD references one or more VNFDs connection points. The VNFs containing these virtual interface are connected with a single of multiple VL) while the VNFFG establishes the data flow around the network. The NSD also exposes a set of connection points to enable connectivity to other network services or the external world.

Here is an example of a simple network service scenario, and a complex one conisted of three nodes and their interconnections.

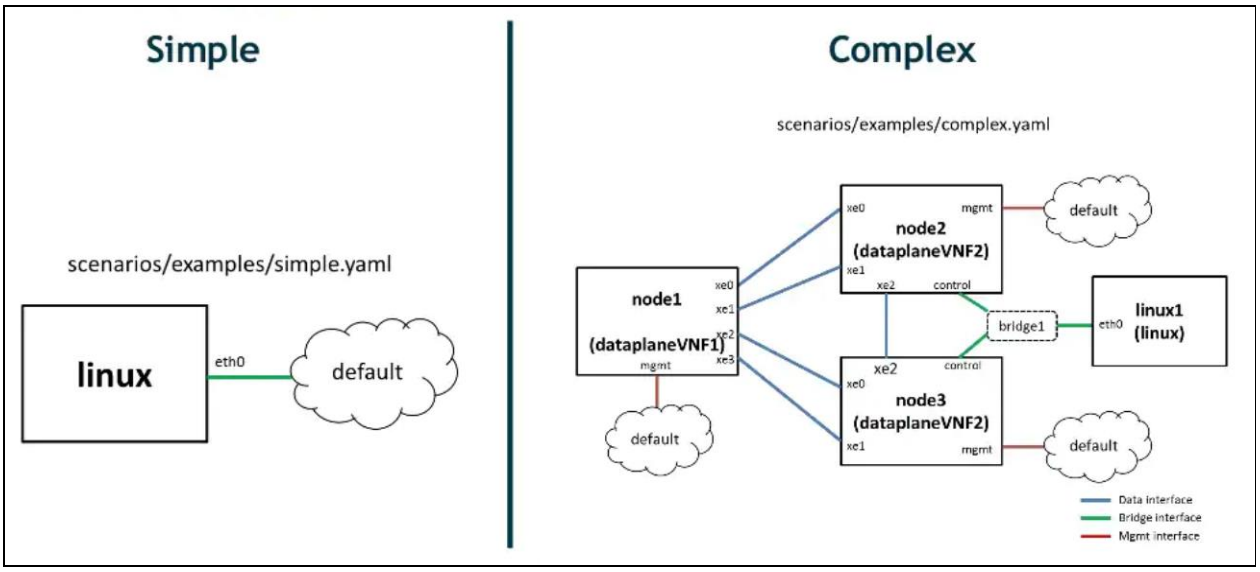


Figure 4-6 Simple and complex NS scenario

4.2.4.2 Virtual Network Function Descriptor (VNFD)

The Virtual Network Function Descriptor (VNFD) is a deployment template that describes the attributes of a single VNF. The VNFD is used primarily by the VNFM in the process of VNF instantiation and lifecycle management of a VNF instance. The information provided in the VNFD is also used by the NFVO to manage and orchestrate network services and virtualised resources on the NFVI.

It is important to highlight the fact that each VNF is constructed from a set of discrete VNF Components (VNFCs), that are pieces of software packaged together to make a more complex architecture. There can be one or more instances of each VNFC in the VNF. The VNFC is realized using a virtualized compute resource from the VIM, which could be either a VM or a container. To run these components a Virtual Deployment Unit (VDU) must be defined, indicating the required information regarding CPU, memory, storage, networking resources, and the software components.

The VNFD also contains:

- VNF images, which contain both the application and the Launchpad.
- Connection points and virtual links that can be used by MANO functional blocks to establish appropriate virtual links between its VNFC instances and between the VNF and the outside network.
- Virtual deployment unit (VDU) that specifies the VM/VNFC compute, storage, and network requirements.
- Platform resource requirements, such as CPU, memory, interfaces, and network.
- EPA characteristics and performance capabilities.
- Scaling properties.

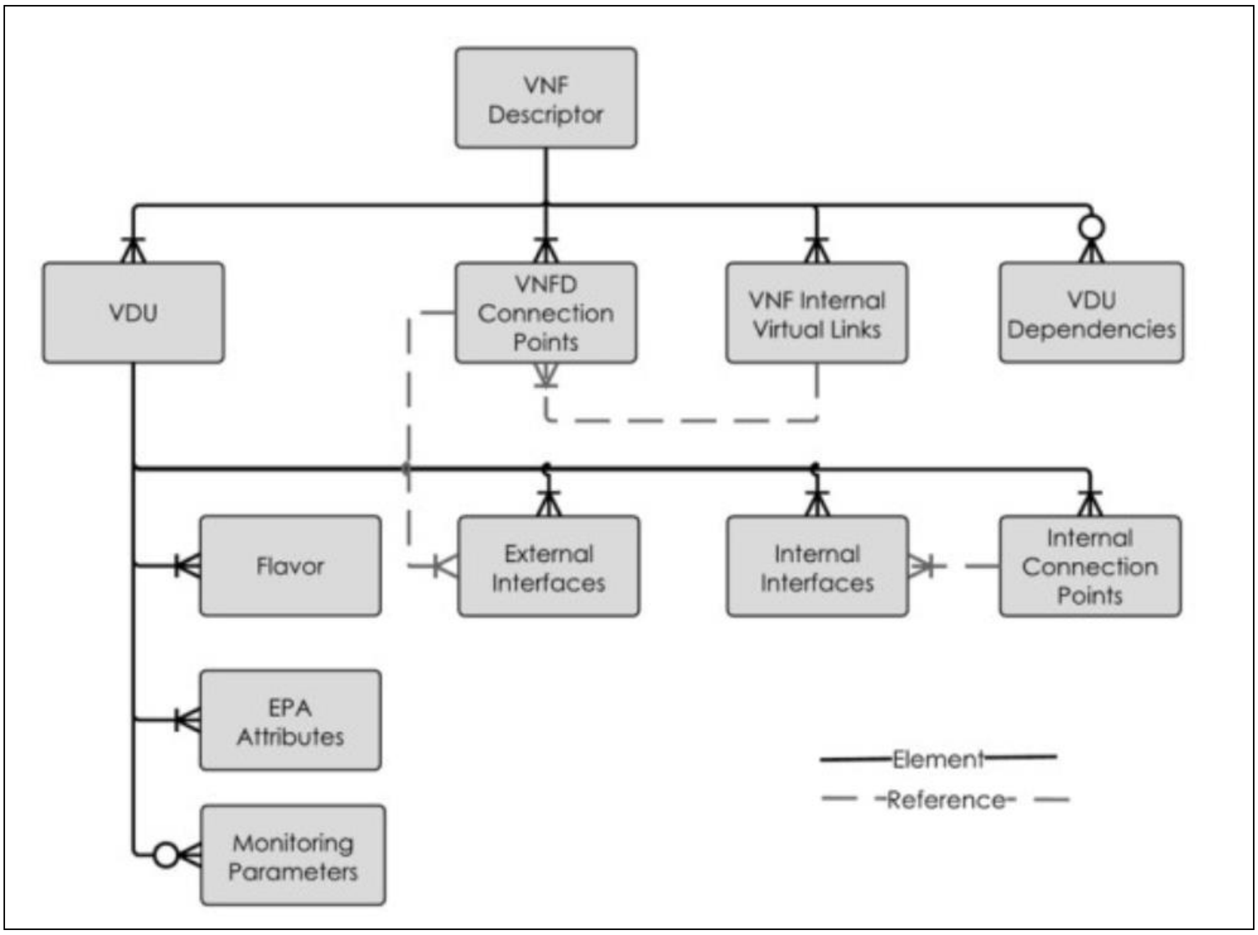


Figure 4-7 VNFD object model

Figure 4-7 is the representation of the high-level object model of the VNFD. This contains lists of VDUs, internal connection points, internal virtual links, and external connection points. The internal connection points and internal virtual links define how the VMs inside the VNF will be connected. The external connection points are used by the NSD to chain VNFs. The VDUs define the individual VNF components and capture information about VM image, VM flavor, and EPA attributes.

Enhanced Platform Awareness (EPA) is designed to improve the performance of guest virtual machines on the hypervisors by enabling fine-grained matching of workload requirements to platform capabilities, before the VM is launched. EPA capabilities captured in the VNFD include:

- Hugepages – that can improve network performance.
- CPU Pinning – pinning the guest system to run on a specific CPU or CPUs and not any available, to avoid latency.
- Guest NUMA (Non-uniform memory access) Awareness – along with CPU Pinning, allows control over how instances run on hypervisor CPUs to help minimize latency and maximize performance.
- PCI Pass-Through – providing direct access to PCI devices when high performance is crucial.
- Data Direct I/O – supporting direct reads/writes from/to storage device to/from user memory space bypassing system page cache, delivering lower latency and increased system I/O.

- Cache Monitoring Technology – allows an operating system, hypervisor, or similar system management agent to determine the usage of L3 cache based on applications running on the platform.
- Cache Allocation Technology – allows an operating system, hypervisor, or similar system management agent to specify the amount of L3 cache space an application can fill.

Below is an example of a single-VM VNF and a multi-VM VNF.

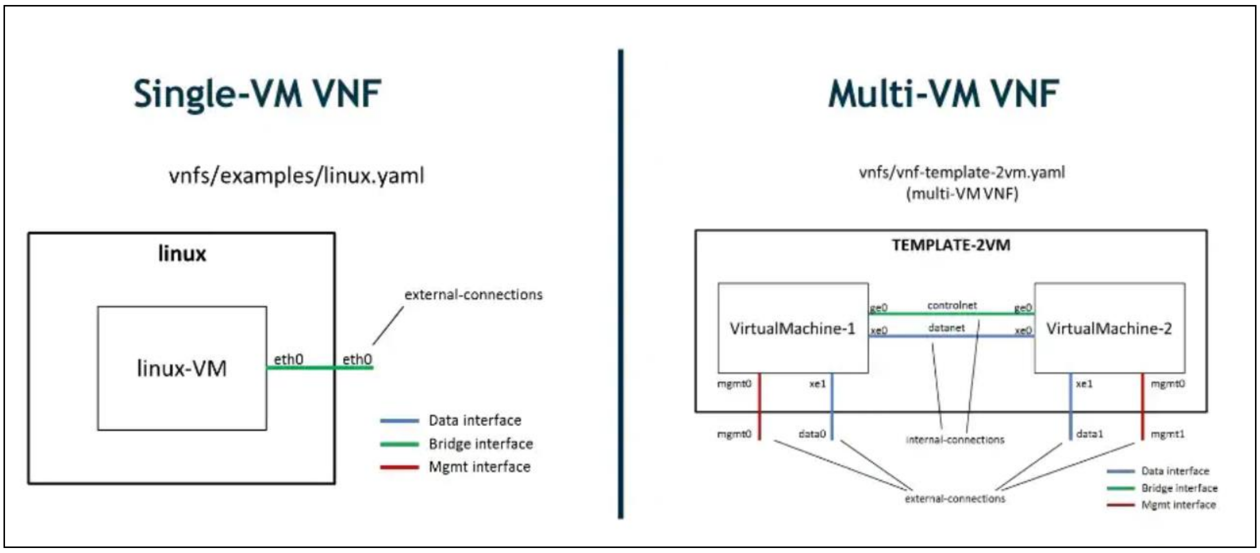


Figure 4-8 Single-VM and Multi-VM VNF

4.2.5 OpenVIM

In its mission to provide a complete ETSI NFV MANO stack, the OSM project also includes a VIM, namely OpenVIM [33]. OpenVIM was originally an open source project that was later contributed to OSM as seed code. OpenVIM is a light implementation of an NFV VIM supporting EPA features and control of an underlay switching infrastructure through an OpenFlow [34] Controller (OFC). OpenVIM interfaces with the compute nodes in the NFVI and an OpenFlow controller in order to provide computing and networking capabilities and to deploy VMs. It offers a northbound interface based on REST, where enhanced cloud services are offered including the creation, deletion and management of images, flavors, instances, and networks. The next Figure shows a datacenter controlled by OpenVIM.

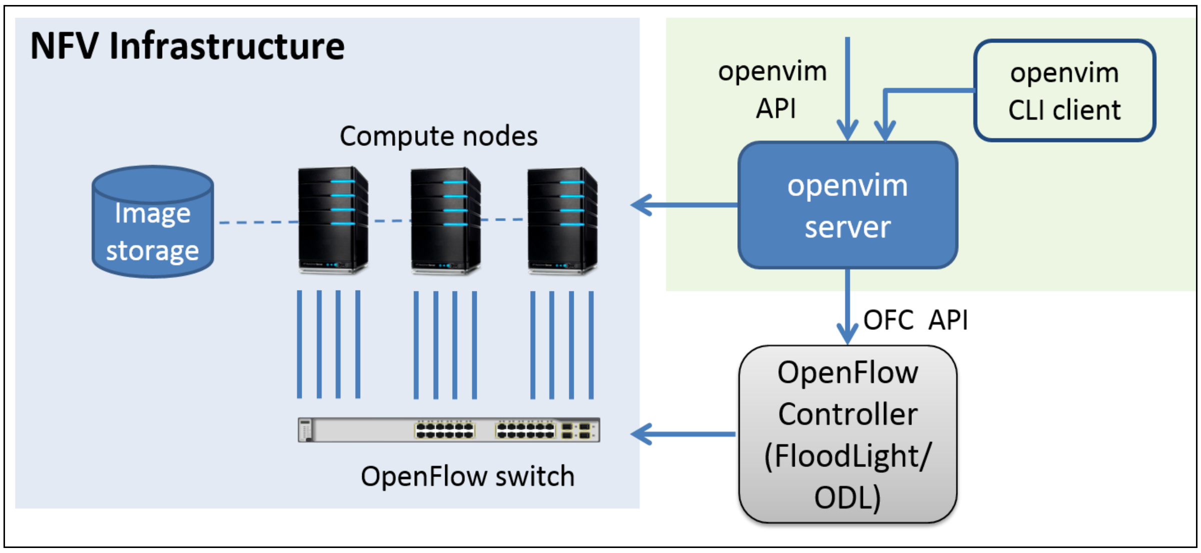


Figure 4-9 OpenVIM Operational Model

OpenVIM supports five modes of operation which are illustrated in the following table.

Table 4-1 Modes of OpenVIM operation

MODE	Purpose	Required Infrastructure
Normal	Regular operation	Compute nodes, OpenFlow switch
Host only	Deploy without OpenFlow switch and controller	Compute nodes
Development	VNF development (deploys without EPA)	Low Performance compute node
Test	Test openMANO installation and API	-
OF only	Test OpenFlow integration	OpenFlow switch

OpenVIM Main Characteristics:

- Host Management: Host addition is done manually through a host descriptor file and hosts can be administratively set up or down.
- Tenant Management: Tenants delimit the property and scope of flavors, images, VMs, networks.
- Network Management: Networks are pure Layer 2 networks. With Precision Time Protocol (PtP) used to create E-line service between two data plane interfaces, data used to create E-LAN service with data plane interfaces, and bridge-data used to create an E-LAN service based on pre-provisioned Linux bridged.
- Port Management: Ports are attached to networks. There are two types of ports. Instance-related ports, with VM interfaces created and deleted as part of the VM life cycle and External Ports, set explicitly by the network administrator in order to

define connections to PNF or external networks physically attached to the OpenFlow switch.

- Image Management: Disk images to be used for a VM.
- Flavor Management: A description of VMs' requirements regarding number of CPUs, memory, and Network Interface Cards (NICs).
- VM instance Management: Besides traditional actions (create, delete, list) allows actions over VMs (shutdown, start, pause, resume, rebuild, reboot).

4.2.6 OSM through releases

OSM first release, Release ZERO, took place in 2016 and since then follows a 6-month release schedule. Each release greatly contributed to the refining of OSM, as well as adding new features.

Release ONE, Release TWO, and Release THREE have set the fundamentals for OSM, urging out of a Release ZERO, which was a test release, and leading to the standardization and the establishment of OSM as it is.

Below is a brief list of some key highlights of each release from Release FOUR onwards.

1. Release FOUR (May 2018):
   a. MON improvements
   b. Support of multi-VDU VNFs
   c. Clean-up of NBI
2. Release FIVE (December 2018):
   a. Support for PNFs, HNFs, and pools of PDUs
   b. Network Slicing for 5G
   c. Monitoring and Policy framework enhancements
   d. Platform and User Experience enhancements
3. Release SIX (June 2019):
   a. Role-based authentication control
   b. Support for full/native charms for enhanced VNF management
   c. Network-Service-level primitives
4. Release SEVEN (December 2019):
   a. Support for Containerized Network Functions (CNFs), also referred to as Kubernetes-based Network Functions (KNFs)
   b. Support for running OSM over Kubernetes
5. Release EIGHT (July 2020):
   a. Kubernetes Proxy Charms
   b. Next Generation User Interface
   c. Subscription API for BSS/OSS
6. Release NINE (December 2020):

   a. Alignment with SOL006
   b. Support of Helm v.3
   c. Centralized VCA for KNFs

7. Release TEN (June 2021):
   a. Improved Scale-in/Scale-out functionality
   b. Support for Microsoft Azure
   c. Distributed VCA
   d. Juju operational dashboard

Along with the improvements through each release, OSM has amassed a significant number of VNF and NS packages which can be found at: https://osm-download.etsi.org/ftp/Packages/ (accessed Dec. 14, 2021).

4.3 OpenStack – MicroStack

4.3.1 OpenStack Overview

OpenStack [15] is a free and open-source cloud computing software platform that enables rapid deployment, management, and development of a cloud infrastructure in a data center. The OpenStack was launched in 2010 as a joint project of Rackspace Hosting and NASA. In 2012 the OpenStack Foundation (later renamed to Open Infrastructure Foundation) was formed, a non-profit organization with the purpose of promoting the development, distribution, and adoption of the software stack. The project has grown rapidly in popularity and is supported by more than 540 companies.

OpenStack platform provides cloud computing services running on standard commodity hardware and is primarily deployed as an Infrastructure-as-a-Service (IaaS) model. The software platform consists of a set of interrelated projects that control diverse, multi-vendor hardware pools of compute, storage, and network resources throughout a data center. The management and control of these pools are exposed through a web-based GUI, a CLI client, or a RESTful API.

OpenStack is fundamental to the Virtualized Infrastructure Manager (VIM) as part of the ETSI NFV MANO stack, as it allows for a NFVI that can host and manage VNFs, including the networking properties of them.

In May 2018 the OpenStack Foundation released the media a presentation [35] expounding the choice of Open Source MANO as the right upper-MANO (i.e., NFV Orchestrator and VNF Manager) into End-to-End NFV with OpenStack.

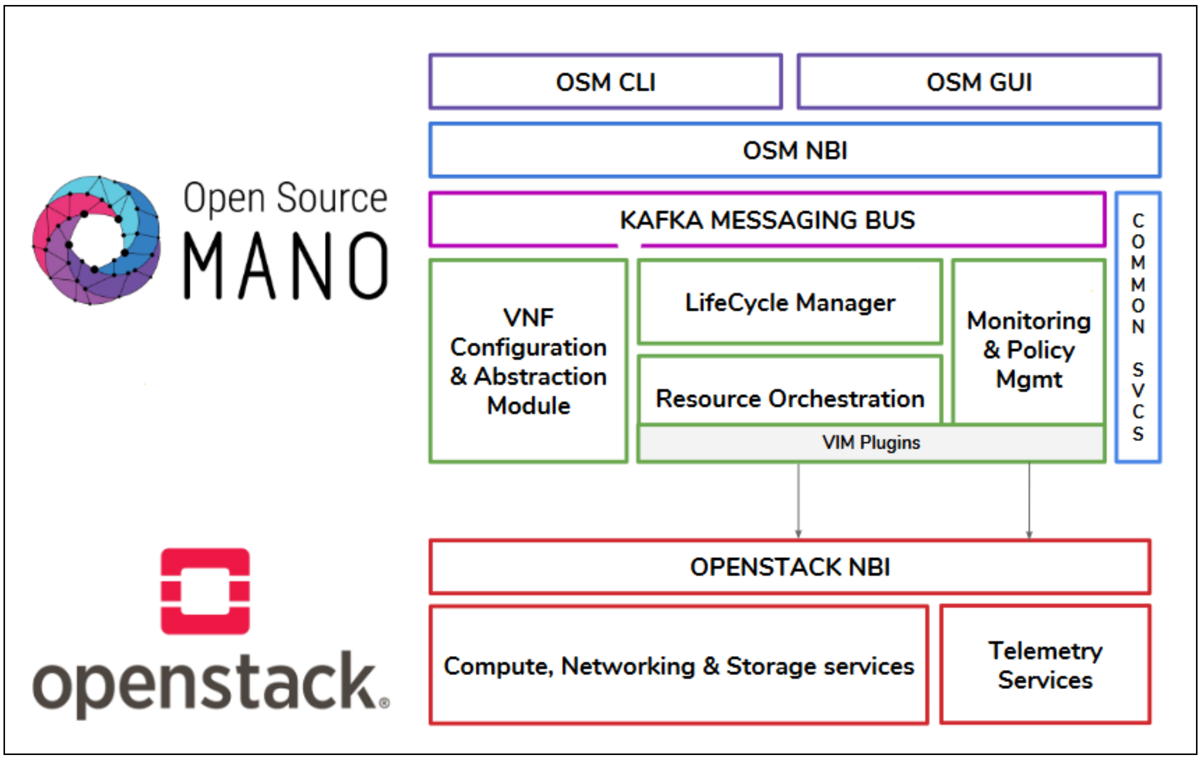


Figure 4-10 OpenStack and OSM architectures together (simplified)

4.3.2 OpenStack's Software Components

The OpenStack projects is composed of several interrelated sub-projects that help to manage different aspects of hardware resources including computing, storage, networking, and other services, each of which offers its own set of APIs to facilitate integration of the whole software stack. This modular architecture of the project allows for easier development and testing of each functionality independently, and provides flexibility to the framework, and a plug-and-play way of using the desired components.

The core components of OpenStack are:

- **Nova**

  Nova is the Compute service and is designed to manage pools of computing resources and provide access to the pools via either gui tools, cli tools, or API calls. Nova works with most popular virtualisation technologies such as KVM (default), VMware, Xen, Hyper-V, and LXC.

  Nova can be considered the main part of an IaaS system, in which cloud users have access to VMs hosted by nodes running Nova service. Within the OpenStack platform, Compute nodes can be added and integrated with existing nodes, making the resource pool horizontally scalable on standard hardware.

- **Neutron**

  Neutron is the Networking service of OpenStack and provides an abstraction of Virtual Network Infrastructure (VNI) (e.g., network, subnets, ports, routers) and services (e.g., firewall, load balancer) within an OpenStack-based cluster.

Neutron essentially provides network-connectivity-as-a-service between interface devices (e.g., vNICs) managed by Nova. It also allows dedicated static IP addresses or DHCP, as well as Floating IP addresses to let traffic be dynamically rerouted.

- **Swift**

  Swift is a distributed object store, that offers storage software so that data can be stored and retrieved through a simple API. It’s built for scale and optimized towards durability, availability, and concurrency across the entire data set.

- **Cinder**

  Cinder is the block storage service for OpenStack. Cinder manages virtualised block storage pools and provides persistent storage (as volumes) to guest virtual machines. Cinder promises to offer high-availability, fault-tolerance, and recoverability.

- **Keystone**

  Keystone is responsible for providing API client authentication, service discovery, and distributed multi-tenant authorization by implementing OpenStack’s Identity API. It is the authentication and authorization system across the whole platform. It supports standard username/password credentials, token-based systems, and AWS-style (i.e., Amazon Web Services) logins.

- **Glance**

  Glance is the Image service that allows creating, storing, and retrieving disk images form VMs, which are used by Compute service during the provisioning of VM instances.

- **Horizon**

  Horizon provides a dashboard which enables users to access and interact with OpenStack services such as Nova, Swift, Neutron, Keystone, etc. in a user-friendly way.

- **Ceilometer**

  Ceilometer provides telemetry services to OpenStack, by tracking and measuring the services used by OpenStack users and provide billing accordingly.

- **Heat**

  Heat is a service to orchestrate the infrastructure resources for a cloud application based on templates. It also provides an autoscaling services that integrates with Telemetry services.

As an IaaS-focused cloud platform, OpenStack has VMs at its center focus, provisioned by the Nova module. Other services, such as Neutron providing network connectivity, Glance storing software images, and Swift and Cinder providing storage services surround the VMs and are key to their successful operation. The rest of the services described above join in to form the core of OpenStack. A schematic of this architecture is presented in the following Figure.

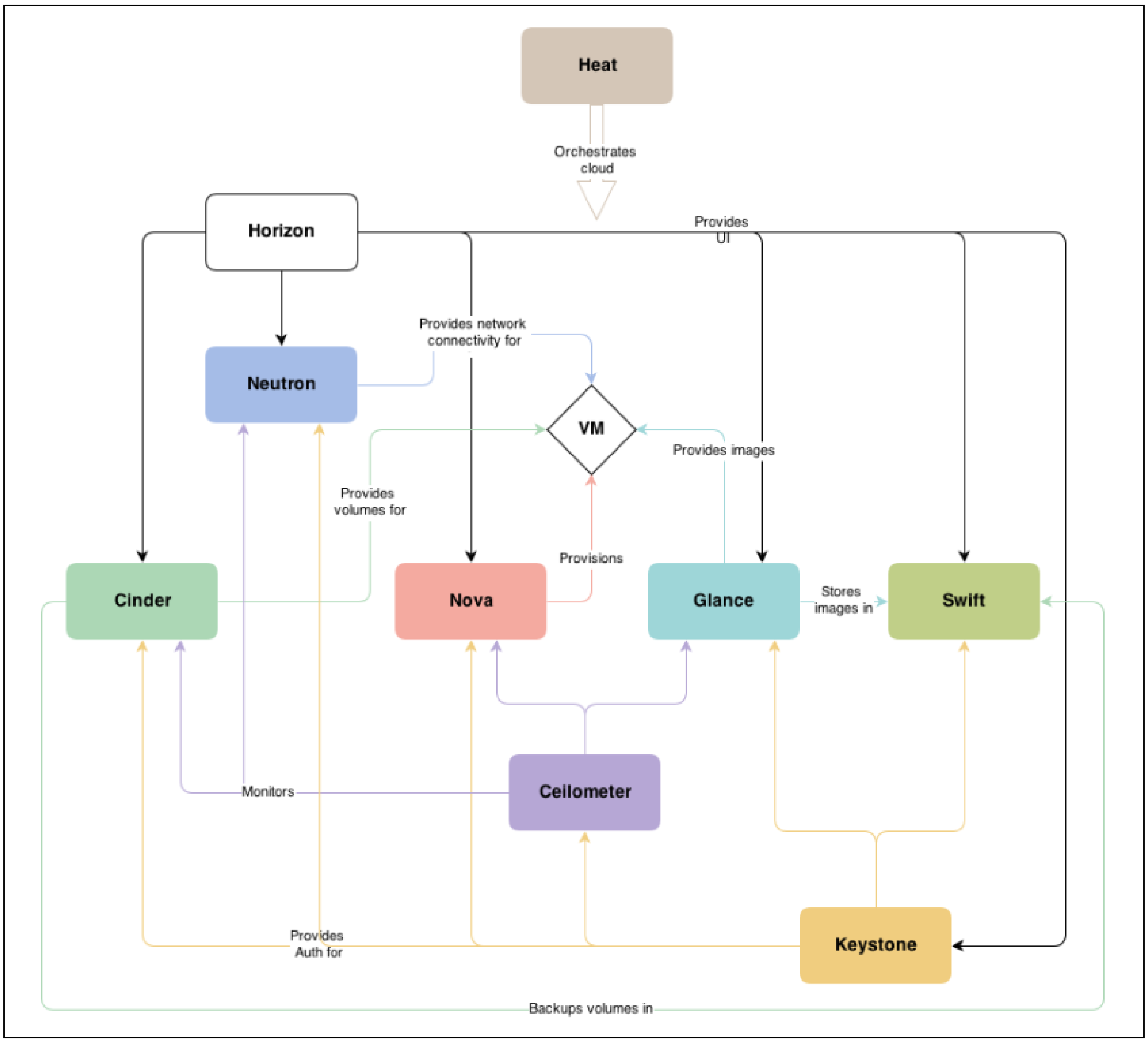


Figure 4-11 OpenStack Services

Of course, OpenStack, being an open source community, is under constant development and always expanding to handle the challenges that emerge. Although the above list of services covers the services responsible for the core functionality of OpenStack, it is not all-inclusive since new services are added with the release of newer versions. Figure 4-12 presents the complete set of OpenStack's components at the time of writing this thesis.

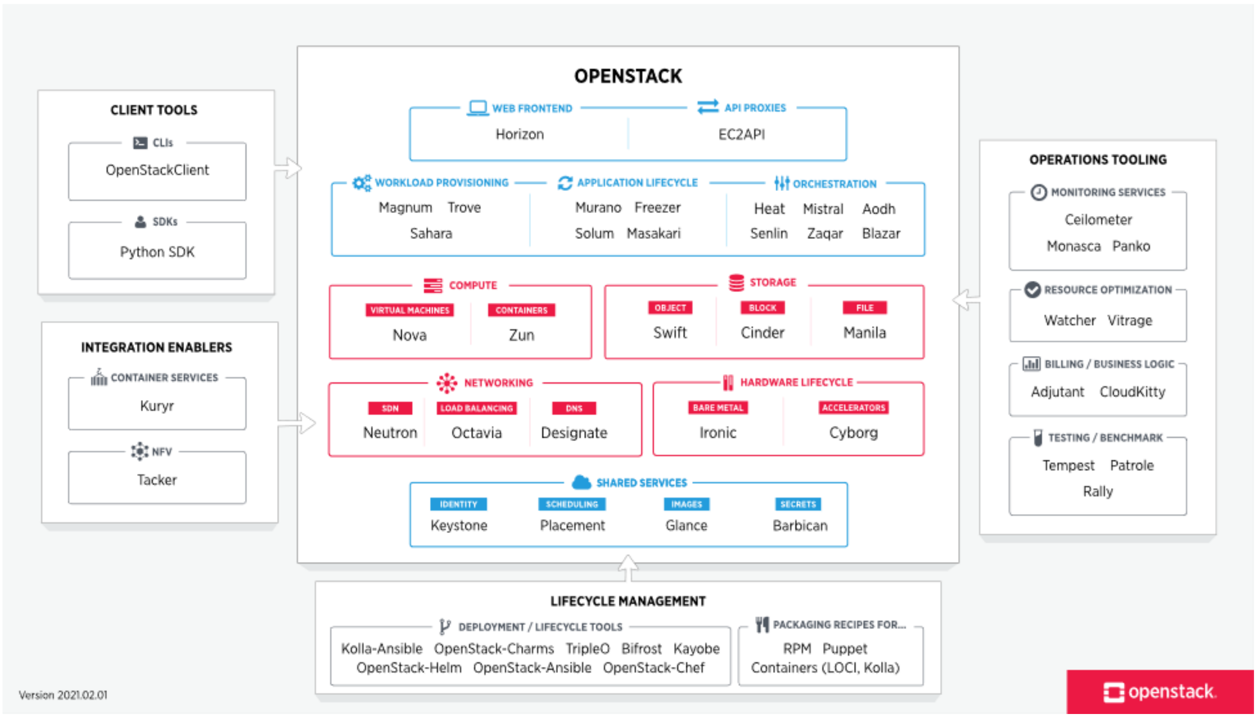


Figure 4-12 OpenStack Components

Each of OpenStack’s modules is designed with a “Share Nothing Architecture” principle in mind and is functionally independent of others. The components communicate with each other through RESTful APIs or Remote Procedure Calls (RPCs), which are backed up by RabbitMQ.

4.3.3 MicroStack

MicroStack [36] is a project launched and maintained by Canonical with the intention to provide a lightweight and easy to setup, yet fully operational distribution of OpenStack. MicroStack is essentially an OpenStack in a snap i.e., a universal Linux package that packs together an application along with its dependencies in a single image that runs in isolation. Snaps allow for single command installation/uninstallation, support automatically updating and due to their embedded nature ensure high-level security.

MicroStack eliminates some of OpenStack’s complexity while keeping its core and most popular services, such as Nova, Neutron, Cinder, Glance, Horizon, and Keystone. This results in a smaller application footprint and renders MicroStack ideal for:

- Edge computing – MicroStack is a great candidate for demanding applications in telecom, industrial, automotive, or other market sectors that require edge infrastructure. It can provide a secure, reliable, and scalable cloud platform with minimal footprint and simplified lifecycle management capabilities.
- Micro Clouds – MicroStack enables corporations to effortlessly deploy small-scale cost-efficient private cloud infrastructure from single-node installations to micro cloud cluster, in contrast to the complexity and scale of standard OpenStack setups.

- Developing and testing of OpenStack workloads – MicroStack can work on devices with minimal hardware resources (e.g., a laptop), which makes it perfectly suitable for developer workstations and CI/CD environments.
- Entry to OpenStack – The reduced complexity of the platform together with the ability to be deployed on a typical Personal Computer, make it ideal for introducing new users to OpenStack.

MicroStack supports both single-node and multi-node (clustering) OpenStack deployments. A fully functional single-node OpenStack can be accomplished with just two commands and in about 20 minutes (depends on network and hardware specifications). However, if needed MicroStack can be installed on several machines and combine them to provide a cluster consisting of a single control node and multiple compute nodes.

Once MicroStack has been installed and the desired setup of OpenStack (i.e., single-node or clustering) has been deployed the user can interact with OpenStack as usual, either through the command line interface client or through the web interface.

5. DETECTION OF BAD NETWORK STATUS AND REDEPLOYMENT OF VNFs

In this chapter we will be focusing on a complete solution to redeploying a Virtual Network Function in real time when it is needed to due to bad network status. We begin by outlining the idea of our proposed solution and how we intend to implement it. Then we will briefly describe complementary software applications that we will use in setting up a test scenario on which we can test and validate the effectiveness of our solution.

5.1 Outline

We plan to implement a complete ETSI NFV stack by using OSM and OpenStack for the MANO layer. A network service (consisting of one or more VNFs) running on this environment could be either a service for the network infrastructure, for example a DHCP server, or a server that collects and analyzes data from IoT devices. For the scope of this thesis the interest lies not on what a NS does, but rather on how we can assure its trouble-free operation.

We will create a program with the role of supervising the network status of a service and if a condition of poor network is found, action will be taken to communicate with OSM and redeploy the service to another VIM, assuring the operation of the service will not be undermined.

In order to know the network status of our VIMs we will use the open source network traffic monitor tool vnStat [37], as well as a lightweight REST web server. Moreover, to simulate network traffic for the sake of testing our solution, iPerf3 [38] will be used, a tool designed for network performance measurement and tuning.

5.2 Extra Software

5.2.1 vnStat

vnStat [37] is a network traffic monitor designed for Linux that keeps a log of network traffic (i.e., received and transmitted bytes/packets) for the selected network interfaces. It uses the network interface statistics provided by the kernel as information source, which assures that vnStat does not sniff traffic and does very light use of system resources regardless of network traffic rate. By default, traffic statistics are stored on a five-minute level for the last 48 hours, on an hourly level for the last 4 days, on a daily level for the last 2 full months and on a yearly level forever, but the data retention durations are fully user configurable.

The tool has a command line interface through which user can configure settings of vnStat, such as which network interfaces to monitor, as well as access the database to retrieve traffic statistics. Output can be provided in a variety of options, such as summary, five-minute, hourly, daily, monthly, and yearly, and can also be given in different formats, such as simple text, json, xml, or even in a png image produced using libgd [39].

Installation of vnStat is quick and simple, and one can opt either for downloading and installing the latest stable release from its git repository, installing an older version of it available on apt, or use a Docker container containing the pre-compiled latest stable release.

Since we would like to remotely monitor network traffic at the VIMs and vnStat is accessible only through its cli client, we implemented a simple RESTful API server. The application is written using the Express framework [40] for node.js, a simple to use framework designed for quick developing of web servers. The REST API listens at port 3030 and accepts GET HTTP requests for retrieving information regarding the available interfaces (at "/interfaces"), as well as network traffic statistics for a specific interface and time duration, for example for the last five minutes (at "/lastfivemins/<interface name>"). Statistics for network traffic are returned in json format and in number of bytes received and transmitted respectively.

5.2.2 iPerf3

iPerf3 [38] is a cross-platform tool created for active measurements of the maximum achievable bandwidth on IP networks. It supports tuning of various parameters related to timing, buffers, and protocols (TCP, UDP, SCTP). The tool can work either as a client or a server; the server handles multiple connections rather than quitting after a single test. By running iPerf3 as a server at one PC and as a client at another, traffic is generated and transferred from the client to the server (default operation) or in the inverse direction for a specified amount of time. This allows the tool to measure the bandwidth of the network, loss, as well as other parameters.

As far as our test case is concerned, we are not interested in measuring the performance of the network, but we will use iPerf3 in a typical client-server setup to simulate heavy network traffic at one VIM.

5.3 Setup of Infrastructure

5.3.1 Installing OSM

Open source MANO can be installed using a single command thanks to an install script provided by OSM's development team. The script can be downloaded and run with the following commands:

```
$ wget https://osm-download.etsi.org/ftp/osm-9.0-nine/install_osm.sh
$ chmod +x install_osm.sh
$ ./install_osm.sh
```

This will install a standalone single-node Kubernetes [41] and deploy OSM on top of it. It is a good idea to save the log of the installation process in case it is later needed for troubleshooting. This can be done at installation time using:

```
$ ./install_osm.sh 2>&1 | tee osm_install_log.txt
```

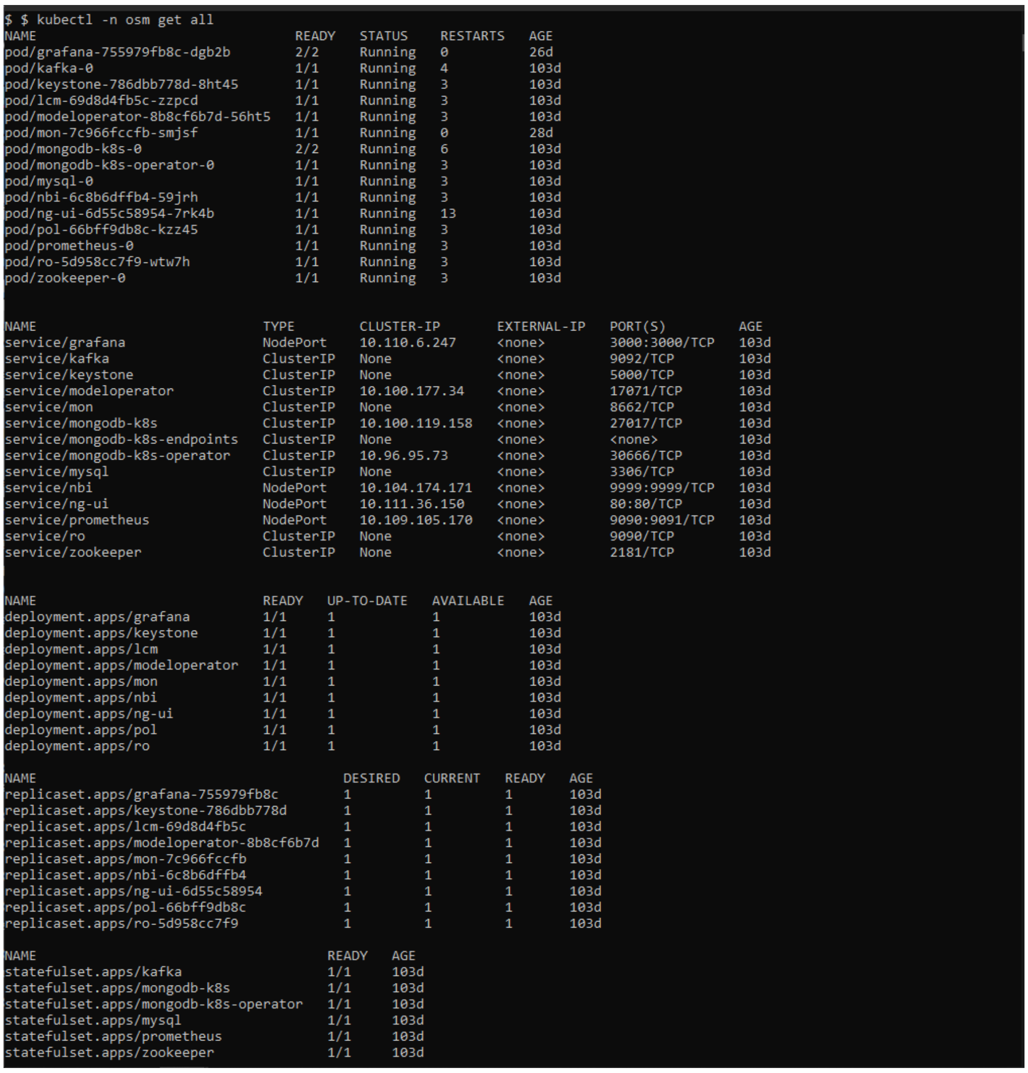
The installation script accepts various options, some of which are:

- -c swarm : to install osm over a single-node docker swarm instead of Kubernetes
- --k8s_monitor : install an add-on to monitor the Kubernetes cluster and OSM running on top of it, through Prometheus and Grafana)
- --pla : install the PLA module for placement support
- --vimemu : additionally deploy the VIM emulator as a docker container

Once the installation is complete, OSM is ready for use. The web interface should be accessible at the url (default credentials: admin/admin): http://X.Y.Z.W, where "X.Y.Z.W" is the IP address of the host, and to validate that everything is running as intended, the status of OSM's components can be retrieved by:

```
$ kubectl -n osm get all
```

Sample output of the command is shown below.



```
$ $ kubectl -n osm get all
NAME                                 READY   STATUS    RESTARTS   AGE
pod/grafana-755979fb8c-dgb2b         2/2     Running   0          26d
pod/kafka-0                          1/1     Running   4          103d
pod/keystone-786dbb778d-8ht45        1/1     Running   3          103d
pod/lcm-69d8d4fb5c-zzpcd             1/1     Running   3          103d
pod/modeloperator-8b8cf6b7d-56ht5    1/1     Running   3          103d
pod/mon-7c966fccfb-smjsf             1/1     Running   0          28d
pod/mongodb-k8s-0                    2/2     Running   6          103d
pod/mongodb-k8s-operator-0           1/1     Running   3          103d
pod/mysql-0                          1/1     Running   3          103d
pod/nbi-6c8b6dffb4-59jrh             1/1     Running   3          103d
pod/ng-ui-6d55c58954-7rk4b           1/1     Running   13         103d
pod/pol-66bff9db8c-kzz45             1/1     Running   3          103d
pod/prometheus-0                     1/1     Running   3          103d
pod/ro-5d958cc7f9-wtw7h              1/1     Running   3          103d
pod/zookeeper-0                      1/1     Running   3          103d


NAME                            TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
service/grafana                 NodePort    10.110.6.247     <none>        3000:3000/TCP    103d
service/kafka                   ClusterIP   None             <none>        9092/TCP         103d
service/keystone                ClusterIP   None             <none>        5000/TCP         103d
service/modeloperator           ClusterIP   10.100.177.34    <none>        17071/TCP        103d
service/mon                     ClusterIP   None             <none>        8662/TCP         103d
service/mongodb-k8s             ClusterIP   10.100.119.158   <none>        27017/TCP        103d
service/mongodb-k8s-endpoints   ClusterIP   None             <none>        <none>           103d
service/mongodb-k8s-operator    ClusterIP   10.96.95.73      <none>        30666/TCP        103d
service/mysql                   ClusterIP   None             <none>        3306/TCP         103d
service/nbi                     NodePort    10.104.174.171   <none>        9999:9999/TCP    103d
service/ng-ui                   NodePort    10.111.36.150    <none>        80:80/TCP        103d
service/prometheus              NodePort    10.109.105.170   <none>        9090:9091/TCP    103d
service/ro                      ClusterIP   None             <none>        9090/TCP         103d
service/zookeeper               ClusterIP   None             <none>        2181/TCP         103d


NAME                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/grafana         1/1     1            1           103d
deployment.apps/keystone        1/1     1            1           103d
deployment.apps/lcm             1/1     1            1           103d
deployment.apps/modeloperator   1/1     1            1           103d
deployment.apps/mon             1/1     1            1           103d
deployment.apps/nbi             1/1     1            1           103d
deployment.apps/ng-ui           1/1     1            1           103d
deployment.apps/pol             1/1     1            1           103d
deployment.apps/ro              1/1     1            1           103d

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/grafana-755979fb8c         1         1         1       103d
replicaset.apps/keystone-786dbb778d        1         1         1       103d
replicaset.apps/lcm-69d8d4fb5c             1         1         1       103d
replicaset.apps/modeloperator-8b8cf6b7d    1         1         1       103d
replicaset.apps/mon-7c966fccfb             1         1         1       103d
replicaset.apps/nbi-6c8b6dffb4             1         1         1       103d
replicaset.apps/ng-ui-6d55c58954           1         1         1       103d
replicaset.apps/pol-66bff9db8c             1         1         1       103d
replicaset.apps/ro-5d958cc7f9              1         1         1       103d

NAME                                   READY   AGE
statefulset.apps/kafka                 1/1     103d
statefulset.apps/mongodb-k8s           1/1     103d
statefulset.apps/mongodb-k8s-operator  1/1     103d
statefulset.apps/mysql                 1/1     103d
statefulset.apps/prometheus            1/1     103d
statefulset.apps/zookeeper             1/1     103d
```

Figure 5-1 Status check of OSM's components

Throughout our work, we faced two major problems regarding the installation of OSM. First, from version 1.17 onwards of kubelet (Kubernetes' primary node agent) the default status of "CSI Migration" was changed to "enabled", which would not allow kubelet to initialize successfully. The solution to this was to disable "CSI Migration" by appending to file "/var/lib/kubelet/config.yaml" the following:

```
featureGates:
    CSIMigration:false
```

Moreover, the hyperlink for downloading and installing Helm had changed but it was not updated in the installation script of OSM Release NINE and thus Helm was not installed.

Both above problems were overcome by installing and using the daily build of OSM, as we were informed by the OSM community at that time. The command for doing so is:

```
$ ./install_osm.sh -R testing-daily -t testing-daily -r testing
```

5.3.2 Installing MicroStack (OpenStack)

We opted for MicroStack as it provides a straightforward and automated way of setting up OpenStack quickly on a single machine. MicroStack can be installed through snap as follows:

```
$ sudo snap install microstack --beta --devmode
```

Once installed, MicroStack can be initialized with the following command:

```
$ sudo microstack init --auto --control
```

The above command will deploy, configure, and start OpenStack services, i.e. create the database, networks, an image with several flavors, ICMP/SSH security groups and an SSH keypair. At this point the standard OpenStack client can be invoked as:

```
$ microstack.openstack <command>
```

OpenStack's web UI provisioned by Horizon can be accessed at the url: http://X.Y.Z.W, where "X.Y.Z.W" is the IP address of the host. The username for connecting is "admin", whereas the password can be obtained with the following command:

```
$ sudo snap get microstack config.credentials.keystone-password
```

5.3.3 Setting up OpenStack as a VIM for OSM

In order for OSM to be able to deploy the desired VNFs and NSs it must be connected to at least one VIM, in our case to an OpenStack. This can be done either through the OSM cli client or through OSM’s web UI.

The command for adding a VIM to OSM through the cli client is:

```
$ osm vim-create –name <VIM-name> --user <username> --password <userpwd> \
  --auth_url <url to VIM> --tenant <tenant name> --account_type openstack
```

where *VIM-name* is an alias name for the added VIM, *username* and *userpwd* are the credentials for accessing OpenStack, *url to VIM* is the full url to OpenStack’s Identity API, and *tenant name* is the name of the tenant/project.

Adding a VIM can also be done through OSM’s web UI, where the required data is filled in an appropriate form.

5.3.4 Deploying the first service

Having successfully installed and configured OSM and OpenStack, we decided to deploy a NS so that we check that everything is working as intended. We chose to go with a very simple NS that consists of one VNF, which requires a single VM. The packages for this setup are: *hackfest_basic_vnf.tar.gz* for the VNF, and *hackfest_basic_ns.tar.gz* for the NS and can be found at https://osm-download.etsi.org/ftp/Packages/examples (accessed Dec. 14, 2021), and a vanilla Ubuntu 16.04 image is needed which can be found at https://cloud-images.ubuntu.com/xenial/current/xenial-server-cloudimg-amd64-disk1.img (accessed Dec. 14, 2021).

First, the Ubuntu image must be uploaded to OpenStack, either through its web UI or through the cli by:

```
$ microstack.openstack image create \
  –file=”./xenial-server-cloudimg-amd64-disk1.img” \
  --container-format=bare --disk-format=qcow2 ubuntu16.04
```

Then, the packages must be uploaded to OSM either through the web UI or the cli. For onboarding the VNF package the following command is to be used:

```
$ osm nfpkg-create hackfest_basic_vnf.tar.gz
```

Whereas the NS package is onboarded with:

```
$ osm nspkg-create hackfest_basic_ns.tar.gz
```

Once the image and the packages have been loaded, the NS can be instantiated by issuing OSM the following command:

```
$ osm ns-create --ns_name <ns-instance-name> --nsd_name \
    hackfest_basic-ns --vim_account <vim-target-name>
```

where *ns-instance-name* is a user defined name for the Network Service, and *vim-target-name* is the alias name of the VIM on which the NS will be deployed. The status of the deployed NS can be checked with:

```
$ osm ns-show <ns-instance-name>
```

If everything is correctly set up, the status of the network service will be “*Instantiated”*.

5.4 Supervising Agent

The supervising agent is responsible for monitoring the operation of a Network Service and assuring that its performance is not undermined due to bad network status. To do so, the agent oversees the network status of the VIM that the NS is running on, and in case of network congestion on that specific VIM interacts with OSM to trigger the redeployment of that NS to another VIM and thus avoid poor performance of the service.

5.4.1 OSM’s North Bound REST API

The realization of our supervising agent is possible thanks to OSM offering a North Bound RESTful API. By default, the API runs on port **9999**, uses SSL (Secure Sockets

Layer) with a self-signed certificate, supports Bearer token authentication, and requests accept a yaml or json body.

The NBI provides endpoints for accessing and managing every element of OSM, and thus allows for easy configuration of the system, as well as for instantiation and lifecycle management of services through HTTP requests. A detailed description of every request the NBI accepts along with their required body parameters and possible responses is hosted at OSM NB API Swagger UIs (accessed Dec. 14, 2021), in a well-organized and easy to read form with Swagger, a toolset created for designing and documenting APIs.

Below we will briefly present some of the available requests, particularly the ones that were vital for our application. The basis url for every endpoint is https://X.Y.Z.W:9999/osm , where "X.Y.Z.W" is the IP address of the OSM host and a suffix is added according to the desired request. For every request presented below we will omit the basis url and only present the suffix.

- /admin/v1/tokens : accepts a **POST** request with login credentials (i.e. username and password) at the body and returns a Bearer Authentication Token which must be sent with every other request or else a "401 Unauthorized" error will be returned.
- /admin/v1/vim_accounts : accepts a **GET** request and returns information about registered VIMs, i.e., VIM name, id, IP address, type, etc.
- /nslcm/v1/ns_descriptors : accepts a **GET** request and returns information of onboarded Network Service Descriptors
- /nslcm/v1/ns_instances : accepts a **GET** request and returns information of running Network Services, such as name, id, VIM on which it is deployed, etc.
- /nslcm/v1/ns_instances_content : accepts a **POST** request with a body containing *nsName* – name of the NS to deploy, *nsdId* – the id of the NS descriptor to be used for deploying the NS, *vimAccountId* – the id of the VIM on which it will be deployed, and *nsDescription* – a short description about the NS, and instantiates a new NS.
- /nslcm/v1/ns_instances_content/<NS ID> : accepts a **DELETE** request and terminates the NS with the specified id

5.4.2 Implementation of the Agent

We chose to implement the supervising agent in python and used packages PyYaml [42] (a yaml parser and emitter) and Requests [43] (for performing http requests). The implementation follows an OOP (Object Oriented Programming) model, with a central class being responsible for retrieving and keeping the required information and offering a method for monitoring a Network Service. The agent can be run as:

```
$ python3 ./monitor.py --url <url for OSM's NBI API> --username \
  <username for OSM> --password <password for OSM> --ns_name \
  <Name of NS to monitor> --nsd_name <Name of NS descriptor to use>\
  [--down <Download rate threshold (Mbps)>] \
  [--up <Upload rate threshold (Mbps)>] \
  [--debug] [--statistics]
```

where:

- --url <url for OSM's NBI API>: the basis url (e.g. https://1.2.3.4:9999/osm) for OSM must be provided
- --username <username for OSM>: username for connecting to OSM
- --password <password for OSM>: password for connecting to OSM
- --ns_name <Name of NS to monitor>: the name of the NS we want to monitor
- --nsd_name <Name of the NS descriptor to use>: the name of the NSD descriptor used to instantiate the NS
- --down <Download rate threshold (Mbps)>: optional parameter for setting the download rate threshold for triggering a bad network status, default is 10
- --up <Upload rate threshold (Mbps)>: optional parameter for setting the upload rate threshold for triggering a bad network status, default is 10
- --debug: optional parameter, if provided the user will be prompted to force a NS redeployment even if network is not congested
- --statistics: optional parameter, if provided the agent will also export a csv file with information such as: time, VIM NS is running on, download and upload rates, if redeployment happened and if so, which is the new VIM

The program initially contacts OSM to get further information, such as available VIMs, details about the monitored NS like its id and the VIM it is running on. Then, every five minutes the agent checks the network status of the VIM that the NS is deployed on, via an http request as we explained earlier in 5.2.1 and if network congestion is detected (by comparing the download/upload rates to a threshold), the agent uses http requests to interact with OSM to force a redeployment of the NS, i.e. terminate the NS from the congested VIM and redeploy it at a different one. Informative messages are displayed for each check and redeployment.

5.4.3 The Agent in action – Test Case

In order to implement and test our idea of the supervising agent, we have created a complete NFV stack. OSM and OpenStack were installed on VMs running Ubuntu Server 18.04 but with varying hardware specifications. For referencing the VMs we have chosen to name them maestroX (since we are dealing with VNF Orchestration), where X is a number. OSM was installed on a VM with 6 CPU cores, 16GB of RAM and 200GB of storage and is named maestro1. One OpenStack installation was done on a VM with 12 CPU cores, 90GB of RAM and 200GB of storage (maestro2), while another installation was done on a VM with 2 CPU cores, 8GB of RAM and 40GB of storage (maestro4). The first OpenStack could represent a regular infrastructure, while the second could be an edge-located one. Of course, the VMs have network interfaces and are connected to a network. Setup of OSM and OpenStack was performed as described earlier in this chapter. The setup is shown in the following figure.

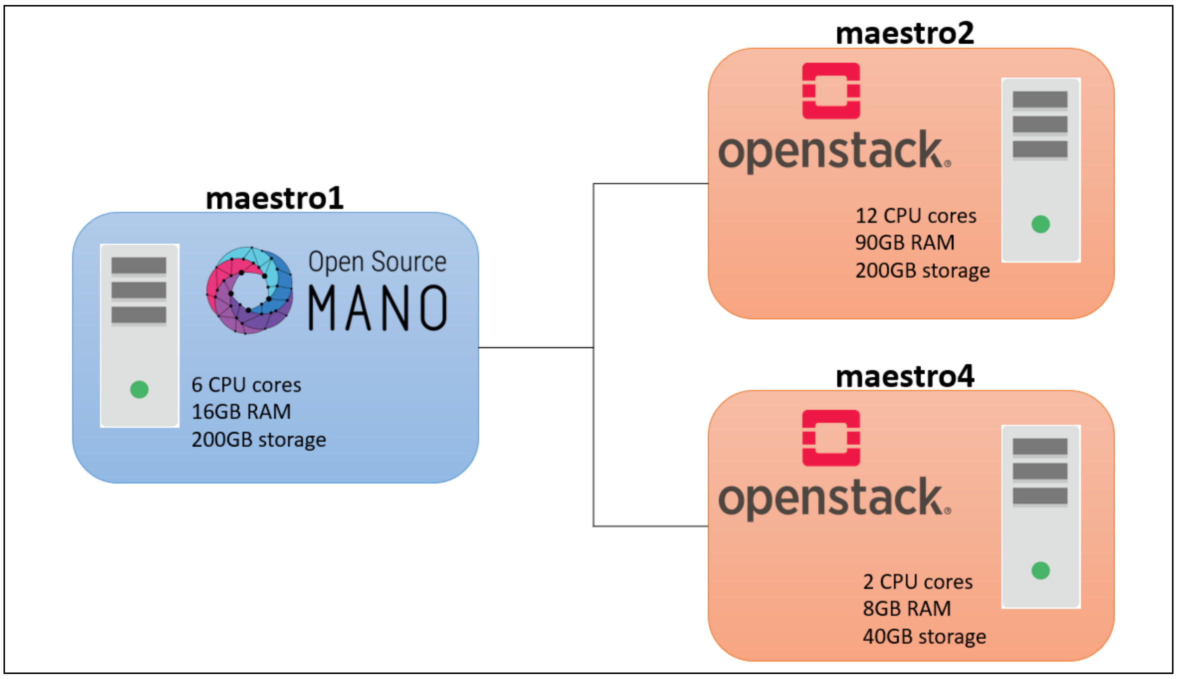


Figure 5-2 Setup of OSM and OpenStack nodes

We proceeded by instantiating a Network Service through OSM on maestro2 and then started our supervising agent in order to monitor the running NS. Note that the agent does not need to be run on the same PC OSM is running, it just needs to have connectivity to the maestros (OSM and OpenStacks) so that it can successfully send and receive http requests and responses respectively.

We allowed the NS to run for some time and then proceeded to simulate heavy network traffic to maestro2. To do so we used iPerf3 and congested maestro2’s network interface card by transmitting data to it at a bandwidth of approximately 120Mbps (Mega bytes per second) which would trigger a redeployment of the NS since it would exceed the set threshold of 50Mbps. Indeed, the agent detected the network congestion and redeployed the NS to the other available VIM, maestro4, updated his information and kept on monitoring it.

Results of this test case are shown in Table 5-1 and Figure 5-3 below, where one can see the point at which the network gets congested, and redeployment takes place. The chart shows the download and upload rates relative to time, as well as the VIM on which the NS had been running until that point.

Table 5-1 Test case of automatic redeployment.
***Time has been adjusted to start at 0:00**

Time*	VIM	Download rate (Mbps)	Upload rate (Mbps)	Redeployment	New VIM
0:00	maestro2	0.0207	0.0893	No	-
0:05	maestro2	0.0201	0.0887	No	-
0:10	maestro2	0.0202	0.0876	No	-
0:15	maestro2	0.0398	0.1003	No	-
0:20	maestro2	119.8236	0.3751	Yes	maestro4
0:25	maestro4	0.0086	0.0203	No	-
0:30	maestro4	0.0096	0.0293	No	-
0:35	maestro4	0.0102	0.0408	No	-
0:40	maestro4	0.0085	0.0242	No	-
0:45	maestro4	0.0091	0.0292	No	-

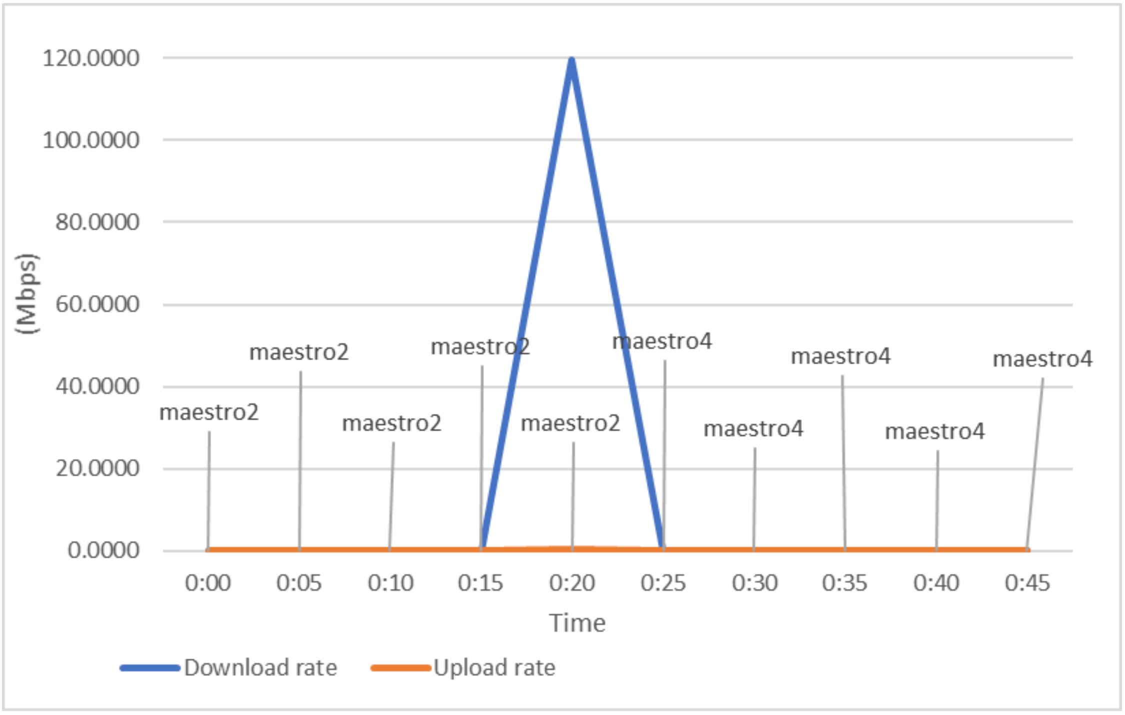


Figure 5-3 Test case of automatic redeployment

6. CONCLUSIONS

Currently, ETSI's Network Functions Virtualization framework is well standardized and mature enough to lead the revolution of virtualization (i.e., use of virtualized infrastructures), provoked by the fact that current physical networks are becoming overcrowded. Technologies such as IoT and 5G have proven that modern networks need to be flexible and scalable, and virtualization can provide that.

All in all, both Open Source MANO and OpenStack have proven to be well established platforms for setting up a NFV stack. OSM VIM-plugin model allows it to support all kinds of VIMs, such as OpenStack [15], OpenVIM [33], VMware vCD [44], as well as popular cloud IaaS platforms like AWS [45] (Amazon Web Services) and Microsoft's Azure [46], makes it appealing to everyone. Moreover, offering a REST NB API, allows for easy realization of extensions and automations like the one proposed in this thesis.

We showed a scenario where we used OSM to redeploy a Network Service in real-time based on its network status. We strongly believe that such an automation will be very useful in the era of Internet of Things, where billions of devices are connected to the internet and network traffic is created at rates higher than ever before. Being able to detect network congestion at one point of the network and redeploy a vital service and prevent it from underperforming can be crucial for modern-era systems.

7. FUTURE WORK

The implementation of the supervising agent that monitors network status of running VNFs and in case of network congestion triggers their redeployment created for the sake of evaluating our proposed solution's potentials in a test case is satisfactory for prototyping. However, should the application be used in real life scenarios and to meet industry standards we believe that the logic behind our idea needs to be integrated into an OSS/BSS solution. At the time of writing this thesis, OSS/BSS solutions compatible with OSM are Mycom OSI [47], Nettracker [48], Wipro RAPIDS [49], and the open source Openslice [50].

Furthermore, although being able to redeploy a service that is underperforming due to bad network status ensures that the quality of service will be undermined only for a short duration of time, i.e. until the service is redeployed, it would be even better if we could predict the spatial and temporal point where network congestion is to occur and take proactive action, redeploying services that would be affected beforehand so that no drop in performance takes place. Thus, the solution integrating the supervising agent, should also employ Artificial Intelligence (AI) in order to predict traffic congestion on the network and act accordingly. The subject of predicting network congestion is one that has drawn a lot of attention in recent years, and various methods to do so have been proposed [51-53].

Both points mentioned above are left open for future research to explore.

ABBREVIATIONS – ACRONYMS

AI	Artificial Intelligence
API	Application Programming Interface
CI/CD	Continuous Integration / Continuous Development
CLI	Command Line Interface
COTS	Commercial Of-The-Self
CRUD	Create/Read/Update/Delete
DHCP	Dynamic Host Configuration Protocol
E2E NSO	Edge-to-Edge Network Service Orchestrator
EMS	Element Management System
EPA	Enhanced Platform Awareness
ETSI	European Telecommunications Standards Institute
GUI	Graphical User Interface
HNF	Hybrid Network Function
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
ICMP	Internet Control Message Protocol
IM	Information Model
IoT	Internet of Things
ISG	Industry Specification Group
IT	Information Technology
MANO	Management and Orchestration
MON	Monitoring Module
N2VC	Network Service to VNF Communication
NAS	Network-attached Storage
NBI	North Bound Interface
NF	Network Function
NFV	Network Functions Virtualization
NFVI	Network Function Virtualization Infrastructure
NFVI-PoPs	Network Function Virtualization Infrastructure Points of Presence
NFVO	Network Functions Virtualization Orchestrator
NIC	Network Interface Card

NS	Network Service
NSD	Network Service Descriptor
NVGRE	Network Virtualisation using Generic Routing Encapsulation
OFC	Openflow Controller
OS	Operating System
OSM	Open Source MANO
OSS/BSS	Operations and Business Support Systems
PaaS	Platform as a Service
PNF	Physical Network Function
PtP	Precision Time Protocol
REST	Representational State Transfer
RO	Resource Orchestration
RPCs	Remote Procedure Calls
SaaS	Software as a Service
SCTP	Stream Control Transmission Protocol
SDN	Software Defined Networking
SO	Service Orchestration
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
VCA	Virtual Network Functions Configuration and Abstraction
VDU	Virtual Deployment Unit
VIM	Virtual Infrastructure Manager
VL	Virtual Link
VLAN	Virtual Local Area Network
VM	Virtual Machine
VNF	Virtual Network Function
VNFCs	Virtual Network Function Components
VNFD	Virtual Network Function Descriptor
VNFFG	Virtual Network Function Forwarding Graph
VNFM	Virtual Network Function Manager
VNI	Virtual Network Infrastructure

VPLS	Virtual Private LAN Services
VxLAN	Virtual Extensible Local Area Network

REFERENCES

[1] IBM Cloud Education, "IaaS vs. PaaS vs. SaaS", IBM, https://www.ibm.com/cloud/learn/iaas-paas-saas (accessed Dec. 14, 2021).

[2] Kramp T., van Kranenburg R., Lange S., "Introduction to the Internet of Things" in *Enabling Things to Talk,* A. Bassi et al, Springer, Berlin, Heidelberg, 2013 pp. 1-10 doi: https://doi.org/10.1007/978-3-642-40403-0_1.

[3] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli, "Fog computing and its role in the internet of things", in *Proc. of the 1st edition of the MCC workshop on Mobile cloud computing (MCC '12),* Association for Computing Machinery, New York, NY, USA, 2012, pp. 13–16. doi: https://doi.org/10.1145/2342509.2342513.

[4] ETSI, "Network Functions Virtualisation (NFV)", ETSI, https://www.etsi.org/technologies/nfv (accessed Dec. 14, 2021).

[5] Øystein L. Andersen, "Security of Internet of Things Protocol Stacks", M.S. thesis, Dept. of Telematics, NTNU, Trondheim, Norway, 2016.

[6] B. Mostafa, "Monitoring Internet of Things Networks," *2019 IEEE 5th World Forum on Internet of Things (WF-IoT),* 2019, pp. 295-298, doi: 10.1109/WF-IoT.2019.8767203.

[7] Bandyopadhyay, D., Sen, J., "Internet of Things: Applications and Challenges" in Technology and Standardization. Wireless Pers Commun vol. 58, pp. 49–69, 2011, https://doi.org/10.1007/s11277-011-0288-5.

[8] ONF, "Software Defined Networking (SDN) Definition)", ONF https://opennetworking.org/sdn-definition/ (accessed Dec. 14, 2021)

[9] G. Fromentoux en N. Omnès, "Network and IT Infrastructure Services for the IoT Store", in *IoT360*, 2014, doi: 10.1007/978-3-319-19656-5_41.

[10] M. Naohisa, T. Kenki, H. Sho, A. Hiroki, A. Aiko, "IoT Network Implemented with NFV" in NEC Technical Journal, vol. 10, no. 3, pp. 35-39, 2016.

[11] X. Wang, C. Wu, F. Le and F. C. M. Lau, "Online Learning-Assisted VNF Service Chain Scaling with Network Uncertainties," *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, 2017, pp. 205-213, doi: 10.1109/CLOUD.2017.34.

[12] M. De Benedictis and A. Lioy, "On the establishment of trust in the cloud-based ETSI NFV framework," *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2017, pp. 280-285, doi: 10.1109/NFV-SDN.2017.8169864.

[13] D. Borsatti, W. Cerroni, G. Davoli and F. Callegati, "Intent-based Service Function Chaining on ETSI NFV Platforms," *2019 10th International Conference on Networks of the Future (NoF)*, 2019, pp. 144-146, doi: 10.1109/NoF47743.2019.9015069.

[14] P. Merle, A. N. Sylla, M. Ouzzif, F. Klamm and K. Guillouard, "A Lightweight Toolchain to Validate, Visualize, Analyze, and Deploy ETSI NFV TopologiesBehaviors," *2019 IEEE Conference on Network Softwarization (NetSoft)*, 2019, pp. 260-262, doi: 10.1109/NETSOFT.2019.8806632.

[15] OpenStack, "Open Source Cloud Computing Infrastructure – OpenStack", OpenStack, https://www.openstack.org/ (accessed Dec. 14, 2021).

[16] Open Source MANO, "OSM", Open Source MANO, https://osm.etsi.org/ (accessed Dec. 14, 2021).

[17] "Network Functions Virtualisation – Introductory White Paper", SDN and OpenFlow World Congres, Germany, White Paper 2012, Available: https://portal.etsi.org/nfv/nfv_white_paper.pdf (accessed Dec. 14, 2021).

[18] ETSI, "NFV Release 2 Description", Available: https://docbox.etsi.org/ISG/NFV/Open/Other/ReleaseDocumentation/NFV(21)000205_NFV_Release_2_Description_v1_12_2.pdf (accessed Dec. 14, 2021).

[19] ETSI, "NFV Release 3 Description", Available https://docbox.etsi.org/ISG/NFV/Open/Other/ReleaseDocumentation/NFV(21)000194r1_NFV_Release_3_Description_v0_8_0.pdf (accessed Dec. 14, 2021).

[20] ETSI, "NFV Release 4 Definition", Available: https://docbox.etsi.org/ISG/NFV/Open/Other/ReleaseDocumentation/NFV(21)000025_NFV_Release_4_Definition_v0_3_0.pdf (accessed Dec. 14, 2021).

[21] ETSI, "ETSI NFV RELEASE 5 KICKS OFF WITH INCREASED SUPPORT FOR CLOUD-ENABLED DEPLOYMENTS", ETSI, https://www.etsi.org/newsroom/press-releases/1992-2021-11-etsi-nfv-release-5-kicks-off-with-increased-support-for-cloud-enabled-deployments (accessed Dec. 14, 2021).

[22] The APACHE Software Foundation, "Apache License, Version 2.0", APACHE, https://www.apache.org/licenses/LICENSE-2.0 (accessed Dec. 14, 2021).

[23] Open Source MANO, "Release notes and whitepapers" Open Source MANO, https://osm.etsi.org/wikipub/index.php/Release_notes_and_whitepapers (accessed Dec. 14, 2021).

[24] Open Source MANO, “OSM workshops and events”, Open Source MANO, https://osm.etsi.org/wikipub/index.php/OSM_workshops_and_events (accessed Dec. 14, 2021).
[25] nfvlabs, (2021), OpenMANO [Source Code], Available: https://github.com/nfvlabs/openmano (accessed Dec. 14, 2021).
[26] RIFTIO, (2017), RIFT.ware [Source Code], Available: https://github.com/RIFTIO/RIFT.ware (accessed Dec. 14, 2021).
[27] Juju, “Operator lifecycle manager for K8s and traditional workloads”, Juju, https://juju.is/ (accessed Dec. 14, 2021).
[28] M. Bjorklund, “YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)”, 2010.
[29] kafka, “Apache Kafka”, kafka, https://kafka.apache.org/ (accessed Dec. 14, 2021).
[30] Prometheus, “Monitoring system & time series database”, Prometheus, https://prometheus.io/ (accessed Dec. 14, 2021).
[31] Grafana Labs, “Grafana: The open observability platform”, Grafana, https://grafana.com/ (accessed Dec. 14, 2021).
[32] YAML, “The Official YAML Web Site”, yaml, https://yaml.org/ (accessed Dec. 14, 2021).
[33] nfvlabs, (2016), OpenVIM [Source Code], Available: https://github.com/nfvlabs/openvim (accessed Dec. 14, 2021).
[34] N. McKeown *et al.*, “OpenFlow: Enabling innovation in campus networks”, *Computer Communication Review*, vol 38, bll 69–74, 04 2008.
[35] Whitestack, “Achieving end-to-end NFV with OpenStack and OpenSource MANO”, Vancouver, BC, Canada, 2018, Available: https://object-storage-ca-ymq-1.vexxhost.net/swift/v1/6e4619c416ff4bd19e1c087f27a43eea/www-assets-prod/presentation-media/Achieving-end-to-end-NFV-with-OpenStack-and-Open-Source-MANO.pdf (accessed Dec. 14, 2021).
[36] MicroStack, “MicroStack – OpenStack in a snap”, microstack, https://microstack.run/ (accessed Dec. 14, 2021).
[37] vnStat, “vnStat- a network traffic monitor for Linux and BDS”, vnStat, https://humdi.net/vnstat/ (accessed Dec. 14, 2021).
[38] iPerf, “iPerf – Download iPerf3 and original iPerf pre-compiled binaries” , iPerf, https://iperf.fr/iperf-download.php (accessed Dec. 14, 2021).
[39] gdLibrary, “GD Graphics Library”, gdLibrary, https://libgd.github.io/ (accessed Dec. 14, 2021).
[40] Express, “Express – Node.js web application framework”, Express, https://expressjs.com/ (accessed Dec. 14, 2021).
[41] kubernetes, “Kubernetes – Production-Grade Container Orchestration” kubernetes, https://kubernetes.io/ (accessed Dec. 14, 2021).
[42] PyYAML, “Welcome to PyYAML” pyyaml, https://pyyaml.org/ (accessed Dec. 14, 2021).
[43] Requests, “Requests: HTTP for Humans – Requests 2.26.0 documentations”, Requests, https://docs.python-requests.org/en/latest/ (accessed Dec. 14, 2021).
[44] vmware, “VMware Cloud Director | Leading Cloud Service Delivery Platform”, vmware, https://www.vmware.com/products/cloud-director.html (accessed Dec. 14, 2021).
[45] aws, “Cloud Services – Amazon Web Services (AWS)”, aws, https://aws.amazon.com/ (accessed Dec. 14, 2021).
[46] Azure, “Cloud Computing Services | Microsoft Azure”, Azure, https://azure.microsoft.com/en-us/ (accessed Dec. 14, 2021).
[47] mycomosi, “Mycom – Assurance, Automation and Analytics”, mycom-osi, https://mycom-osi.com (accessed Dec. 14, 2021).
[48] Netcracker, “Netcracker -Home”, netcracker, https://www.netcracker.com/ (accessed Dec. 14, 2021).
[49] wipro, “Digital BSS Solution | Digital BSS As A Service – Wipro”, wipro, https://www.wipro.com/communications-/rapids/ (accessed Dec. 14, 2021).
[50] C. Tranoris, “Openslice: An opensource OSS for Delivering Network Slice as a Service”, *arXiv [cs.NI]*. 2021.
[51] Li qian-mu, Zhao xue-long, Xu man-wu and Liu feng-yu, "Network congestion prediction based on RFNN," 2005 IEEE International Conference on Systems, Man and Cybernetics, 2005, pp. 2212-2217 Vol. 3, doi: 10.1109/ICSMC.2005.1571477.
[52] Y. V. Sneha, Vimitha, Vishwasini, S. Boloor and N. D. Adesh, "Prediction of Network Congestion at Router using Machine learning Technique," 2020 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER), 2020, pp. 188-193, doi: 10.1109/DISCOVER50404.2020.9278028.
[53] A. Yamamoto *et al.*, “Prediction of Traffic Congestion on Wired and Wireless Networks Using RNN”, in *Proceedings of the 13th International Conference on Ubiquitous Information Management and Communication (IMCOM) 2019*, 2019, bll 315–328.