UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

DESCRIPTION OF SERVICE INSTANCES IN A VIRTUALISED NETWORK ENVIRONMENT

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Julia Igual Nevot

In partial fulfilment

of the requirements for the degree in

TELECOMMUNICATIONS ENGINEERING

Director : Franco Callegati

Advisor : Josep Solé Pareta

Bolgona, Italy, June 2018

Abstract

The main goal of this thesis is to learn about the emerging technologies SDN and NFV. Mainly, we will focus on the study of NFV MANO and especially in Open Source MANO. We will explain and define all of them and we will implement some scenarios of network services.

First of all, an overview of SDN and NFV technologies will be given, why did they appear and which is their architecture and main components. We will talk about OpenFlow protocol used by SDN controllers and about the relation between SDN and NFV.

Then, we will define more specifically what NFV MANO and the implementation of Open Source MANO are. We will not only focus in the most important features but also in the architectural framework. Moreover, the main descriptors used to define virtual network functions and network services will be studied and some important features will be explained. We will develop some network service scenarios creating their corresponding descriptors with YAML format and defining their internal components and the connections needed.

Finally, in the conclusions we will value the work done and the knowledge acquired through all the process.

Resum

L'objectiu principal d'aquesta tesi és aprendre les tecnologies emergents SDN i NFV. Principalment ens centrarem en l'estudi de NFV MANO i especialment en Open Source MANO. Explicarem i definirem aquests conceptes i implementarem alguns escenaris de serveis de xarxa.

En primer lloc es proporcionarà una descripció general de les tecnologies SDN i NFV, per què van aparèixer i quina és la seva arquitectura i components principals. Parlarem del protocol OpenFlow utilitzat pels controladors SDN i sobre la relació que hi ha entre SDN i NFV.

Després definirem més específicament què són NFV MANO i la implementació de Open Source MANO. No només explicarem les característiques més importants sinó que també el marc arquitectònic. A més, s'estudiaran els principals descriptors utilitzats per definir les funcions de xarxa virtual i els serveis de xarxa. També s'explicaran algunes característiques importants. Desenvoluparem alguns escenaris de serveis de xarxa amb la creació dels descriptors necessaris en format YAML i descrivint els seus components interns i les connexions necessàries.

Finalment, a les conclusions, valorarem el treball fet i el coneixement adquirit durant tot el procés de realització del projecte.

Resumen

El objetivo principal de esta tesis es aprender las tecnologías emergentes SDN y NFV. Principalmente nos centraremos en el estudio de NFV MANO y especialmente en Open Source MANO. Los explicaremos y definiremos e implementaremos algunos escenarios de servicios de red.

En primer lugar se proporcionará una descripción general de las tecnologías SDN y NFV, por qué aparecieron y cuál es su arquitectura y componentes principales. Hablaremos del protocolo OpenFlow usado por los controladores SDN y sobre la relación entre SDN y NFV.

Después definiremos más específicamente qué son NFV MANO y la implementación de Open Source MANO. No solo explicaremos las características más importantes sino que también el marco arquitectónico. Además, se estudiarán los principales descriptores usados para definir las funciones de red virtual y los servicios de red. También se explicarán algunas características importantes. Desarrollaremos algunos escenarios de servicios de red con la creación de los descriptores necesarios en formato YAML y describiendo sus componentes internos y las conexiones necesarias.

Finalmente, en las conclusiones, valoraremos el trabajo hecho y el conocimiento adquirido a lo largo de la realización del proyecto.

Acknowledgements

First of all I would like to thank my family who have supported me the best way possible even though being far from home. I appreciate the patience of my friends, who have suffered my stress in some points but have done the path easier. Also, thank my teacher Franco Callegati for helping me during these months, introducing me to this new topics and telling me which could be the best approaches for the thesis.

Revision history and approval record

Revision	Date	Purpose
0	31/05/2018	Document creation
1	14/06/2018	Document revision
2	17/06/2018	Document revision
3	19/06/2018	Document revision
4	22/06/2018	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Julia Igual Nevot	julia.igual.nevot@gmail.com
Franco Callegati	franco.callegati@unibo.it
Walter Cerroni	walter.cerroni@unibo.it
Josep Solé Pareta	pareta@ac.upc.edu

Written by:		Reviewed and approved by:	
Date	22/06/2018	Date	25/06/2018
Name	Julia Igual Nevot	Name	Franco Callegati
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

1. Introduction

It is known that current network topologies are in constant evolution. Increasingly, the deployment of new network services requires more flexibility and dynamism. The main problem is the dependence on proprietary hardware which is making development difficult.

These issues have leaded to the use of virtualisation and creation of new paradigms. The solutions offered are Software Defined Networking and Network Function Virtualisation. These new technologies allow faster deployments and much more scalable and agile networks.

In this thesis we will give an explanation of them, introducing the main concepts and architecture. But the main point will be to study the service descriptors and which functionalities can be defined as well as create some network services.

1.1. Statement of purpose

The project is carried out at the University of Bologna, Networking Technologies Laboratory (NetLab) in Bologna.

The objective of this project is to analyse the design and process management of NFV as well as the design of complex network services that can be used as a base for bigger configurations.

The project main goals are:

- ✓ Understand the Software Defined Networking (SDN) technology
- ✓ Get acquainted with Openflow protocol
- ✓ Get acquainted with the OpenStack cloud management platform
- ✓ Understand the concepts of Network Function Virtualization (NFV) and Service Function Chaining
- ✓ Get acquainted with the Open Source Mano (OSM) platform already integrated with OpenStack in an existing test-bed
- ✓ Understand simplest configurations of network services with OSM and develop more complex ones

1.2. Requirements and specifications

Project requirements:

- Understand the SDN main concepts
- Learn the basic architecture of a cloud computing architecture and in particular of the OpenStack case
- Learn how to use the Open Source Mano platform for NFV implementation integrated with an OpenStack cloud computing environment
- Understand Network Service descriptors and how to implement them
- Develop service instances

1.3. Methods and procedures

The project is the continuity of a Master thesis made by Giacomo Cortesi, “Open-source software solutions for management and orchestration of virtualised network infrastructures”.

The main goal is to understand the technologies of SDN and NFV. Moreover, we will focus in the network service descriptors and the features that can be implemented in order to create new useful network service scenarios.

1.4. Workplan

The project has a linear structure. In this section, there is a brief description of the main parts divided in 4 WPs.

WP 1. Training

WP 2. Learn OpenStack

WP 3. Learn Open Source MANO

WP 4. Analysis of descriptors and implementation of NS

Figure shows the Gantt Diagram of the project.

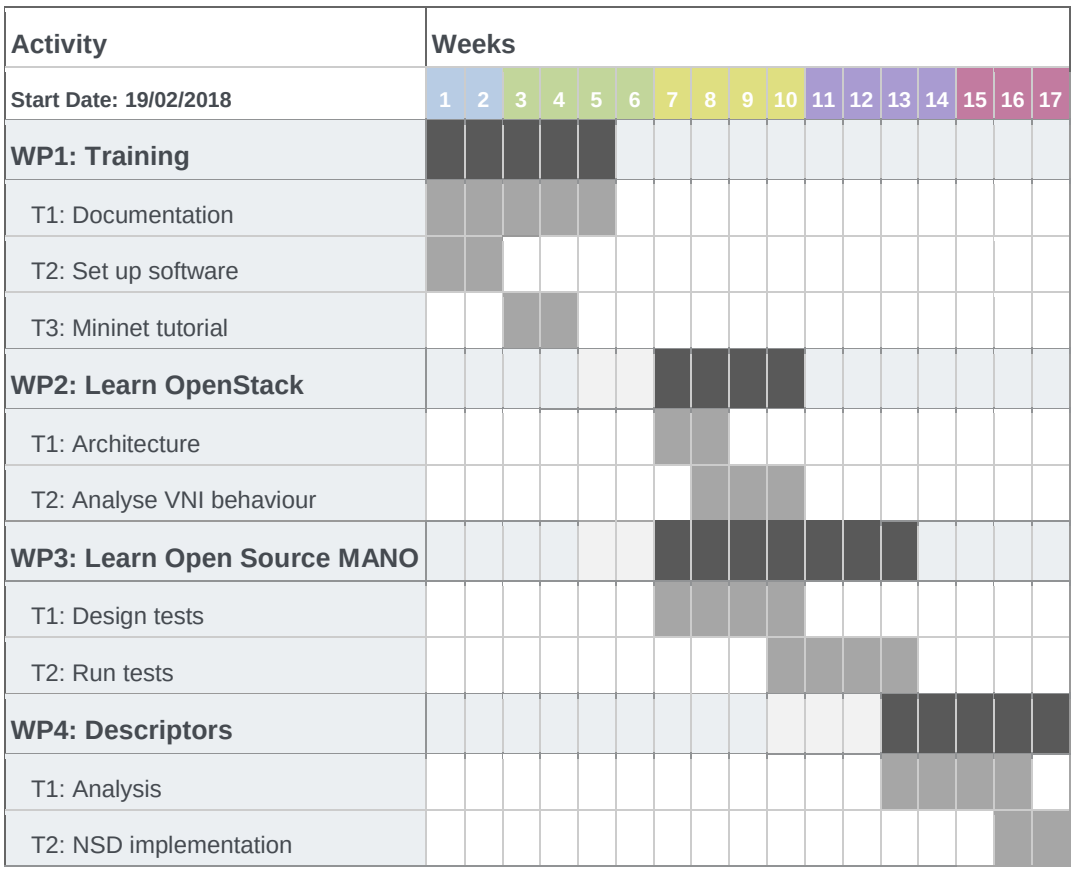

Figure 1. Gantt Diagram of the project

1.5. Incidents and modifications

Through the development of the project we have been modifying the final goal of the project as we have seen that some issues were not possible to accomplish. First of all we wanted to implement the parallel management of two VIMs with OSM. But, due to that we realized that it could not be done and that the hardware and software developments by Giacomo Cortesi were not working, we could not test it. So, finally we focused in the study of possible features that could be defined through the descriptors and the development of services.

2. SDN and NFV

2.1. Software Defined Networking

Software Defined Networking (SDN) is a new approach for network programmability. Through open interfaces it has the capacity to initialize, control, change and manage network behaviour dynamically [1]. In order to achieve it, it is necessary the disassociation of the control plane logic (Control Plane) from the forwarding hardware (Forwarding Plane). The main idea is to centralize the control of the network in one component usually called *Controller*. This enables administrators to respond faster to innovation cycles, changing business requirements and changing networks conditions as the separation allows directly programming the network control and abstracting the hardware infrastructure for applications and network services [2].

Current networks require flexibility and easy troubleshooting and SDN encompasses multiple network technologies to achieve it.

2.1.1. SDN Components

SDN framework is shown in Figure 2. The architecture consists in three layers: Application, Control and Infrastructure. The application layer is on the top and refers to the delivering of services by applications as well as the communication of behaviours and resources needed with the SDN Controller. In the middle there's the control layer with the Controller. The infrastructure layer contains the hardware devices which control de forwarding and data processing capabilities of the network.

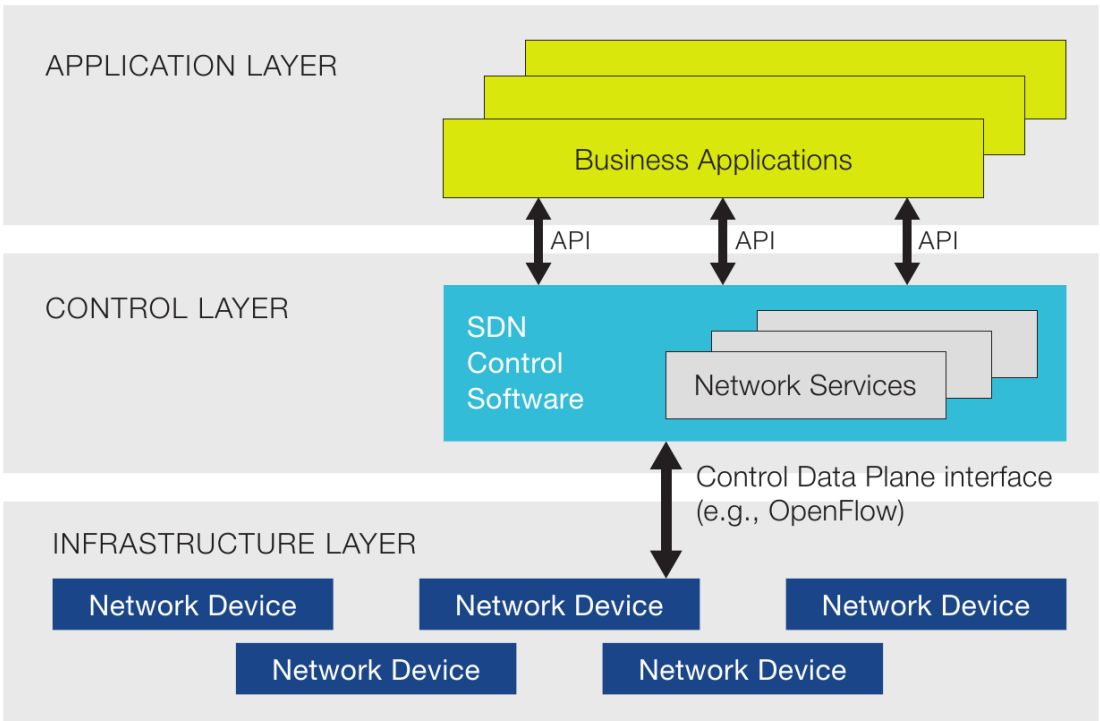

Figure 2. SDN framework [2]

The controller not only transmits to the networking components instructions or requisites received from the top layer but it also gets information about the network (e.g. statistics) from these devices and sends them back to the SDN applications.

In order to accomplish the communication between the three layers the Controller uses APIs. Northbound APIs enable communication between the controller and applications. Southbound APIs are used to relay information to the switches and routers in the infrastructure layer [2, 3].

2.1.2. OpenFlow protocol

OpenFlow [4, 5] is an implementation of the abovementioned southbound API. The concept begun in Standford University and the first version was released in 2009, being one of the first SDN standards. The Open Networking Foundation (ONF), which is a user-led organisation whose purpose is to promote SDN as well as to open standards and SDN adoption, has been managing OpenFlow since its release.

This communication protocol enables the direct interaction between the SDN Controller and the forwarding plane of network devices (Switches, routers…). Through this interface, the controller uses OpenFlow messages to push down changes to the switch or router flow tables (add, update and delete actions).

2.2. Network Function Virtualization

Network Function Virtualization (NFV) [6] started when service providers realised that they could not accelerate the deployment of new network services as hardware-based appliances limited their ability to trial new services and reduce costs. Setting up new services requires greater flexibility and dynamism that hardware-based appliances, which are complex to deploy and manage, do not allow. NFV not only decouples software from hardware but also makes networks agile and capable to reply automatically to the needs of the traffic and services running over it.

In 2012, the NFV Industry Specification Group (ISG) was founded by seven telecom network operators within the European Telecommunication Standard Institute (ETSI). The creation of ETSI NFV ISG leaded to the foundation of NFV's basic requirements and architecture. Nowadays, more than three hundred companies are still working deeply to develop the required standards for NFV as well as sharing their experiences of NFV implementation and testing.

2.2.1. NFV architectural framework

The NFV Architectural framework [7] classifies the functional blocks and the main reference points between those blocks.

The functional blocks, which will be explained in detail later, are:

- Virtualised Network Functions (VNF)
- Element Management (EM)
- NFV Infrastructure
- NFV Management and Orchestration

- Operation and Business Support Systems (OSS/BSS)

Figure 3 represents the general framework at a functional level and it does not specify any detailed implementation.

The solid lines represent the main and execution reference points, which are in the scope of NFV. Instead, the other reference points represented by doted lines are available in present deployments but require extension to handle NFV.

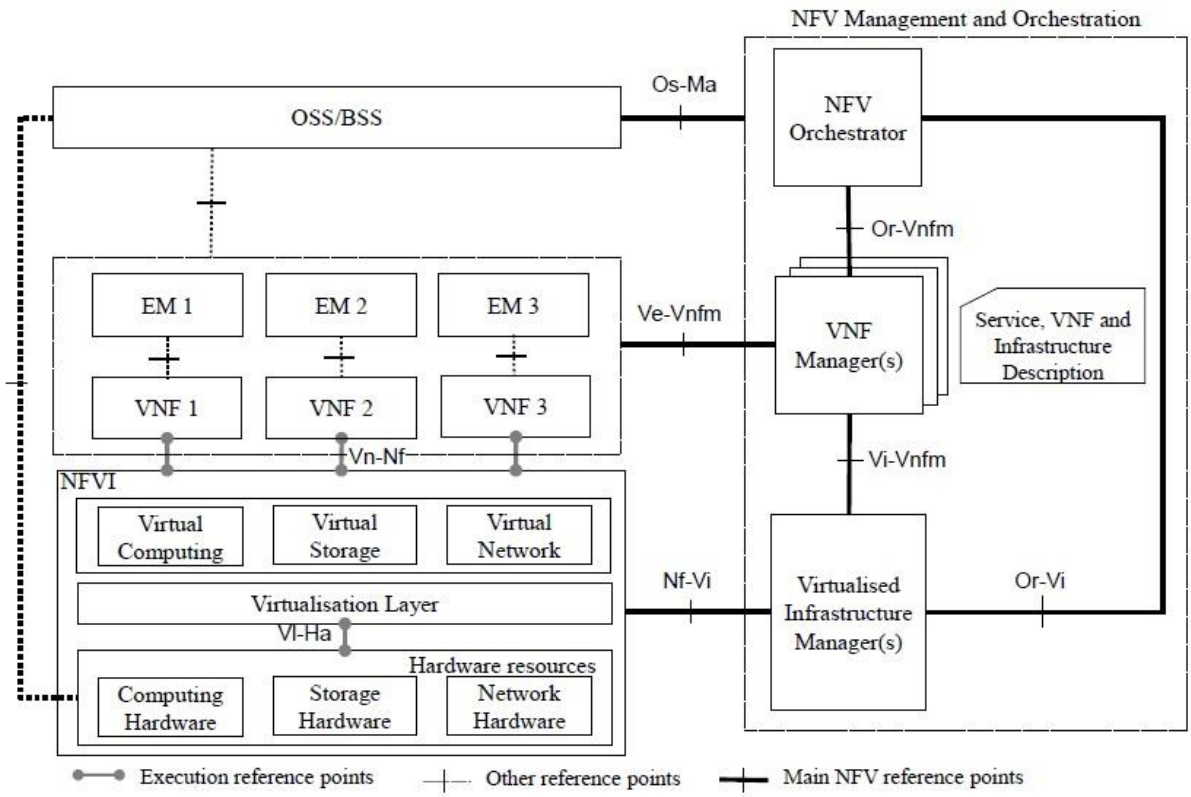

Figure 3. NFV architectural framework [7]

Virtualised Network Functions (VNF)

A VNF is the network function software implementation running on top of the NFVI. It can be deployed in different ways. Multiple VMs can host a single component of the VNF or the whole VNF can be deployed in a single VM. Some examples of VNFs include firewalls, domain name system (DNS), network address translation (NAT), etc.

Element Management

The function of the Element Management is to perform the management of one or numerous VNFs.

NFV Infrastructure

Describes the totality of hardware and software components which build up the environment in which VNFs are deployed. It supports the execution of VNFs and provides the physical platform. A virtualization layer above the physical resources is created to abstract the hardware resources. This helps their partition so that they can be provided to the VNF to perform their functions [8].

NFV Management and Orchestration

Covers the orchestration and lifecycle management of physical and software resources as well as the lifecycle management of VNFs. It also focuses on all virtualization-specific management tasks necessary in the NFV Framework [7].

OSS/BSS

Refers to the Operations Support Systems and Business Support Systems of an Operator .

2.3. SDN and NFV relation

SDN and NFV arose almost at the same time as they build on the rapid evolution of IT and cloud technologies. In Figure 4, it can be observed that NVF and SDN are complementary but not dependent on one another. NFV can be achieved without SDN, although a combination of both can give a greater value [9, 10].

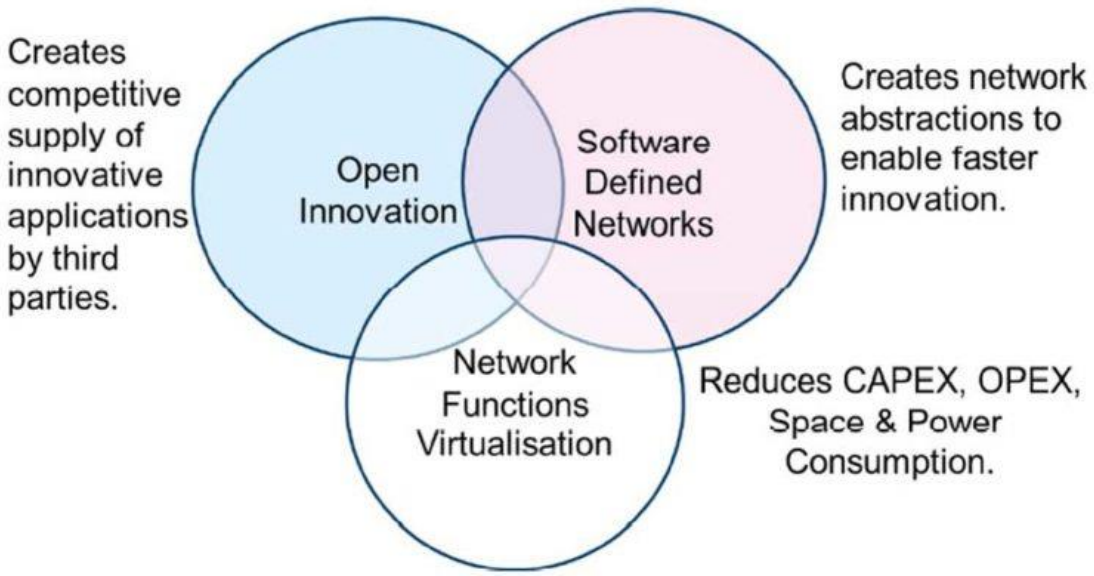

Figure 4. NFV relationship with SDN [10]

On the one hand, NFV [10] goals can be achieved with current techniques used in many datacentres. But methods based on the separation of the control and forwarding planes proposed by SDN can improve performance as well as simplify operation and maintenance procedures. Moreover, SDN can run in an infrastructure provided by NFV.

3. NFV MANO and OSM

3.1. NFV MANO

In NFV [11], the software implementation of Network Functions is decoupled from the computation, storage and networking resources they use. This separation can be done through a virtualisation layer and presents new entities, the Virtualised Network Functions (VNFs), and new relationships between them and the NFVI. The chain of a set of VNFs creates a Network Service (NS).

Since the rise of NFV has added new capabilities, such as the relationship between NFV and NFVI. Their control needs a new set of management and orchestration functions which can be achieved through the Network Functions Virtualisation Management and Orchestration (NFV-MANO). NFV-MANO has the role to manage the NFVI and orchestrate the allocation of resources needed by the NSs and VNFs.

3.1.1. Framework

The NFV-MANO architectural framework is represented on the right side of Figure 5. It is shown at a functional level and it does not specify any implementation.

The NFV MANO architectural framework not only focuses on the new functional blocks brought into operator's network by NFV but also on the reference points between those functional blocs.

New and critical reference points for NFV-MANO are drawn in bold and continuous lines and labelled. They represent potential objectives for development in open source projects and later standardization.

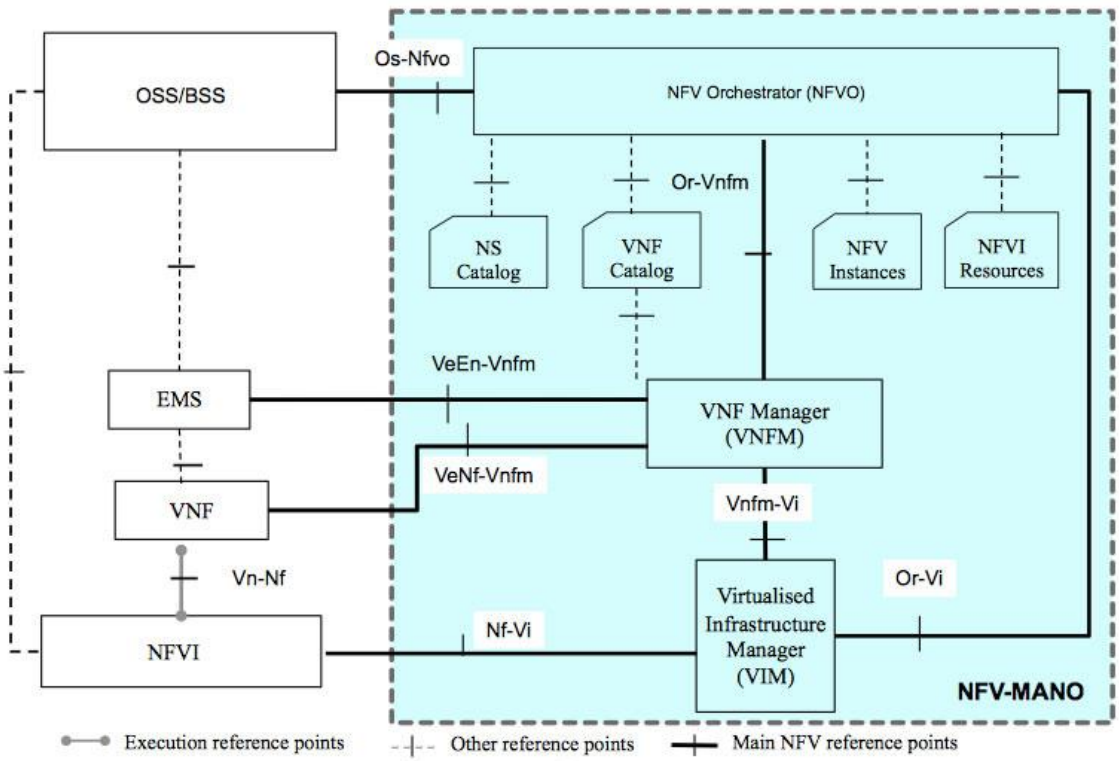

Figure 5. NFV-MANO architectural framework with reference points [12]

3.1.2. Functional blocks

The functional blocks of NFV-MANO are the NFV Orchestrator (NFVO), the VNF Manager (VNFM) and the Virtualised infrastructure manager (VIM). Each of these blocks has a number of responsibilities and works on specific entities. Applies management and orchestration in the functional block itself but it also takes advantage of the services offered by others.

NFV Orchestrator

The NFVO [11, 12] has two main responsibilities.

On the one hand, it fulfils Resource Orchestration (RO) functions which include providing services that support accessing NFVI resources in an abstracted manner independently of any VIMs, as well as governance of VNF instances sharing resources of the NFVI infrastructure.

On the other hand, it fulfils the Network Service Orchestration functions which provide management of the NFV services deployment templates and VNF Packages (e.g. on-boarding of new network services (NS) and virtual network function (VNF) packages); NS lifecycle management; global resource management; validation and authorization of NFVI resource requests; management of the network service instances (e.g. create, update, query, delete VNF Forwarding Graphs).

Additionally, the NFV MANO solution must interact with other components such as the OSS/BSS solutions already in place.

VNF Manager (VNFM)

The VNF Manager [11] is in charge of the lifecycle management of VNF instances. Every VNF instance is supposed to have an associated VNF Manager, but a VNF Manager may be associated to a single or multiple VNFs instances. The VNFs can be of the same or different types.

Even though VNFM functions usually are generic so that they can be applicable to any type of VNF, cases in which a specific functionality is needed should be supported. This functionality may be specified in the VNF Package.

Virtualised infrastructure manager (VIM)

The Virtualised Infrastructure Manager (VIM) [11, 13] is responsible for controlling and managing the NFVI compute, storage and network resources. A VIM may be able to handle a specific type of NFVI resource or multiple types.

Keeping an inventory of the allocation of virtual resources to physical resources is one of the VIM's operations. It also supports the management of VNF Forwarding Graphs as well as handles a repository of NFVI hardware and software resources. Other functions performed by the VIM are managing security group policies to ensure access control and

collecting performance and fault information. Briefly, it could be said that the VIM is the link between hardware and software in the NFV world.

3.2. Open Source MANO

Open Source MANO (OSM) [9] is an operator-led ETSI community that is delivering a production-quality open source Management and Orchestration (MANO). It follows the ETSI NFV information model explained before and meets the requirements of production NFV networks.

The first release was announced in June 2016 and since then they have been following up releases every six months. The last release, number FOUR, has been published in May 2018.

The mapping between OSM component s and ETSI NFV MANO architecture is shown in Figure 6. The functional blocks and reference points are clearly articulated.

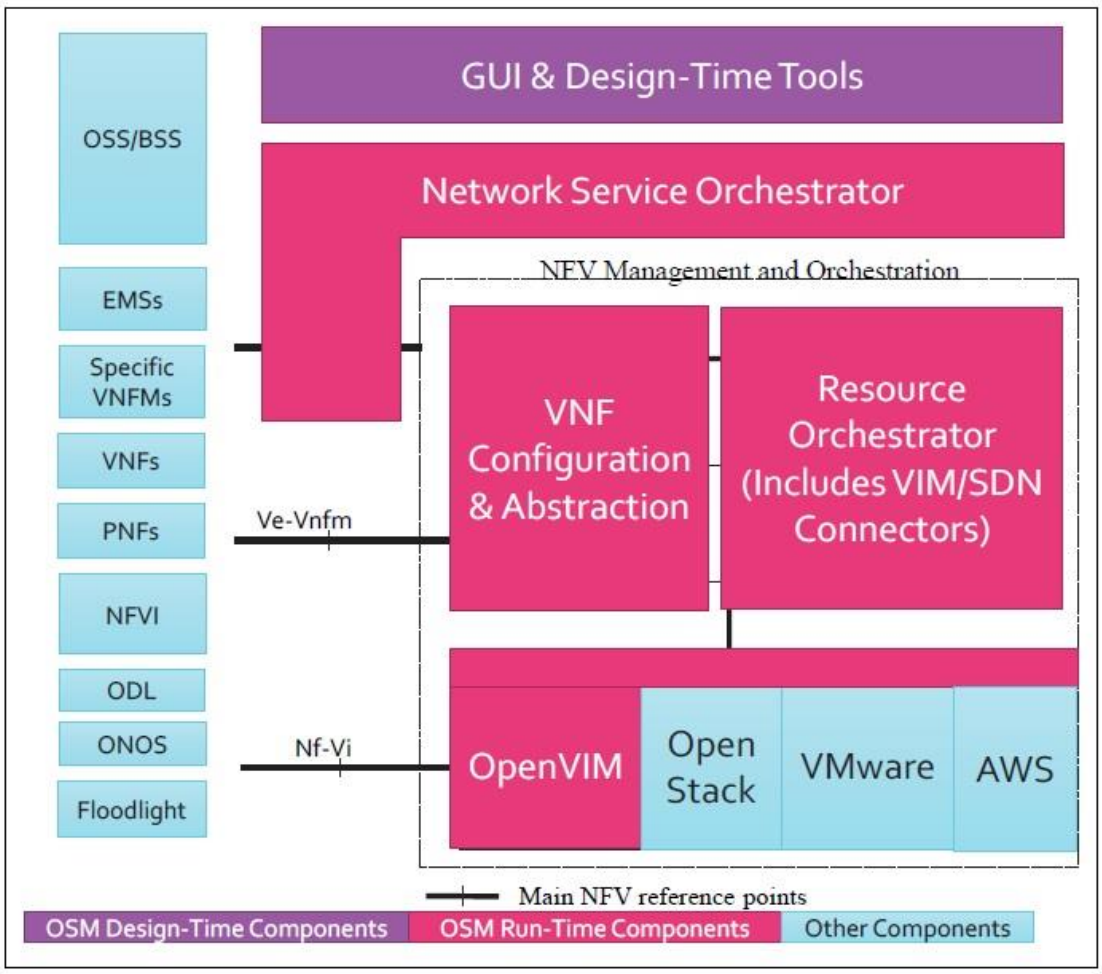

Figure 6. OSM Mapping to ETSI NFV MANO [9]

3.2.1. Features

Every release OSM has been adding new features. In this section some of them will be stated.

Some of the capabilities include the integration of multiple SDN controllers (OpenDayLight (ODL), Open Network Operating System (ONOS)…) and multiple VIMs (OpenVIM, OpenStack, VMWare…). It enables access to features through GUI, CLI, Python based client library and REST interfaces, and it supports multi-site scenarios (NS instances with VNFs running in different datacentres) and multiple monitoring tools for services.

Release four [14] has brought a big number of new functionalities and improvements especially in terms of functionality, user experience and maturity. It has provided a new northbound interface, aligned with NFV work, to facilitate the interoperability with existing and new client applications and a new cloud-native setup that improves user experience and optimization. Monitoring and closed-loop capabilities have also been extended and modelling and networking logic have been improved.

3.2.2. Architecture

Figure 7 shows the logical blocks that represent the functionality offered by OSM. These are colour coded by run-time and design time components.

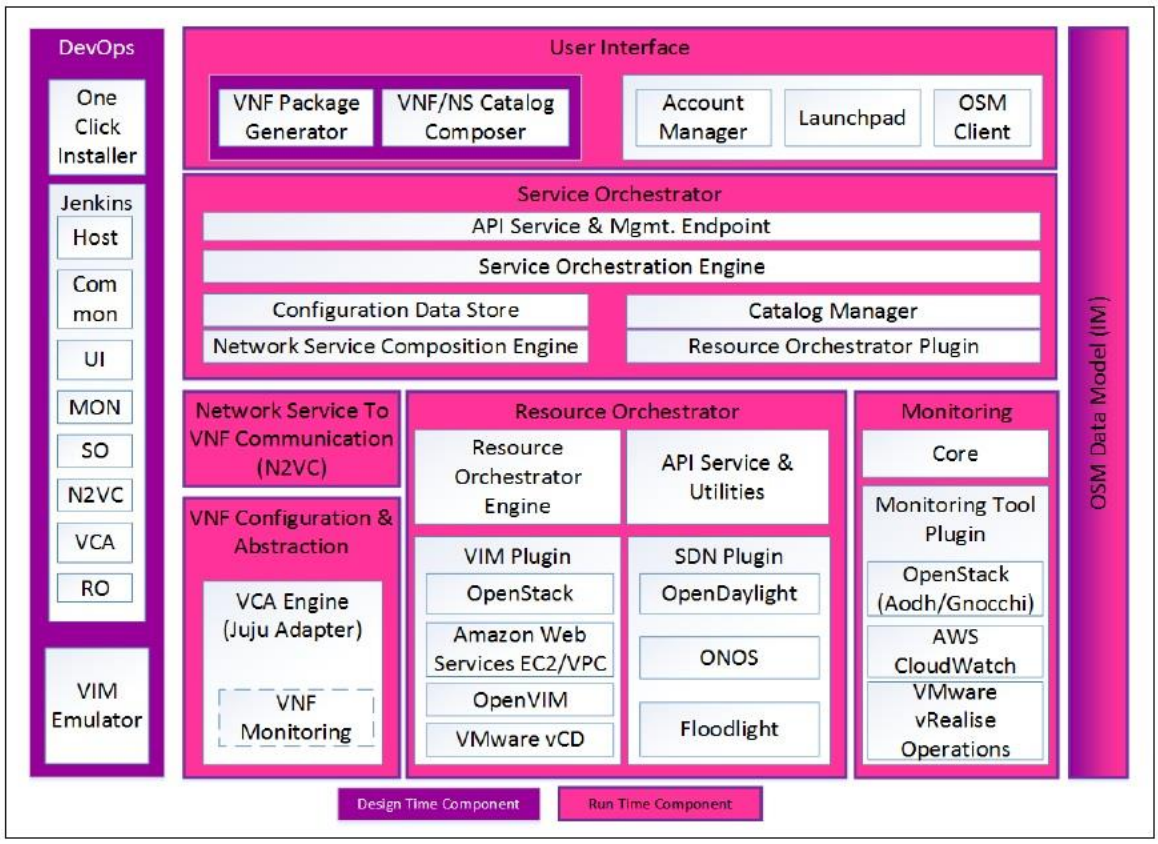

Figure 7. OSM Release three architecture [9]

The main components are [9]:

• *User Interface Module (UI):* It is called Launchpad and is the interactive graphical user interface (GUI) of the system. It allows the management of lifecycle operations on VNFs and NSs. It provides VNFs and NSs statistics and a detailed view of the compute and network topologies. The OSM Client provides a CLI client to remotely interact with OSM's Northbound REST API. It provides a python functional library to programmatically interact with OSM remotely.

• *Service Orchestrator (SO)* which is comprised by the following components:

- Configuration Data Store: stores the SO state, in the context of VNF and NS deployment records.
- API Service & Management Endpoint: provides the primary API endpoint into OSM.
- Service Orchestration Engine: is responsible for all aspects of service orchestration and for supporting the concepts of multi-tenancy, projects, users, and enforcing role-based access controls.
- Network Service Composition Engine: supports NS and VNF descriptor composition and validates that the composed Network Services and Virtual Network Function descriptors conform to the defined YANG schema.
- The Catalog Manager performs create, read, update, delete lifecycle operations on the defined VNF and NS descriptors and packages.
- The Resource Orchestrator Plugin provides an interface to integrate the Resource Orchestrator

• *The Network Service to VNF Communication (N2VC)* module is responsible for the plugin framework between the SO and the VCA layer.

• *The VNF Configuration and Abstraction (VCA*) layer is responsible for enabling configurations, actions and notifications to and from the VNFs and Element Managers.

• *The Resource orchestrator (RO) module* is made of:

- The API Service & Utilities endpoint provides the interface into the RO (for the SO to consume) and provides a number of utilities for internal to RO consumption.
- The Resource Orchestration Engine manages and coordinates resource allocations across multiple geo-distributed VIMs and multiple SDN controllers.
- The VIM and SDN Plugins connect the Resource Orchestration Engine with the specific interface provided by the VIMs and SDN controllers.

• *Monitoring module:* Release 3 includes this experimental module. It is not intended to replicate or compete with the existing or new monitoring systems but to interface with and leverage them. The monitoring tool plugin is responsible for translating alarms and metrics from the innate format of the monitoring tool into the format that OSM interprets.

• *OSM information model module.* OSM is based on a model-driven architecture. 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 to share VNFDs and NSDs using their innate forms between all the components.

3.2.3. OpenStack

OpenStack [15, 16] is a set of open source software tools for building and managing cloud computing platforms for public and private clouds. This cloud computing system controls large pools of compute, storage, and networking resources throughout a datacenter, managed through a dashboard or via the OpenStack API. OpenStack works with popular enterprise and open source technologies making it ideal for heterogeneous infrastructure and provides Infrastructure as a Service (IaaS). It is often used as the VIM in the NFV MANO architecture.

As OSM, Openstack follows a release cycle of 6 months. Queens is the current OpenStack release since February 2018.

As shown in Figure 8, OpenStack is made up of several independent parts, named the OpenStack services. All services authenticate through a common Identity service. Individual services interact with each other through public APIs, except where privileged administrator commands are necessary. These are the nine key components:

- **Nova** is the primary computing engine behind OpenStack. It is used for deploying and managing large numbers of virtual machines and other instances to handle computing tasks.
- **Swift** is a storage system for objects and files.
- **Cinder** provides persistent block storage to running instances.
- **Neutron** provides the networking capability for OpenStack. It helps to ensure that each of the components of an OpenStack deployment can communicate with one another quickly and efficiently.
- **Horizon** is the dashboard behind OpenStack. Provides a web-based self-service portal to interact with underlying OpenStack services, such as launching an instance, assigning IP addresses and configuring access controls.
- **Keystone** provides identity services for OpenStack. Provides an authentication and authorization service for other OpenStack services. Provides a catalog of endpoints for all OpenStack services.
- **Glance** provides image services to OpenStack. Stores and retrieves virtual machine disk images. OpenStack Compute makes use of this during instance provisioning.
- **Ceilometer** provides telemetry services, which allow the cloud to provide billing services to individual users of the cloud.
- **Heat** is the orchestration component of OpenStack, which allows developers to store the requirements of a cloud application in a file that defines what resources are necessary for that application. In this way, it helps to manage the infrastructure needed for a cloud service to run.

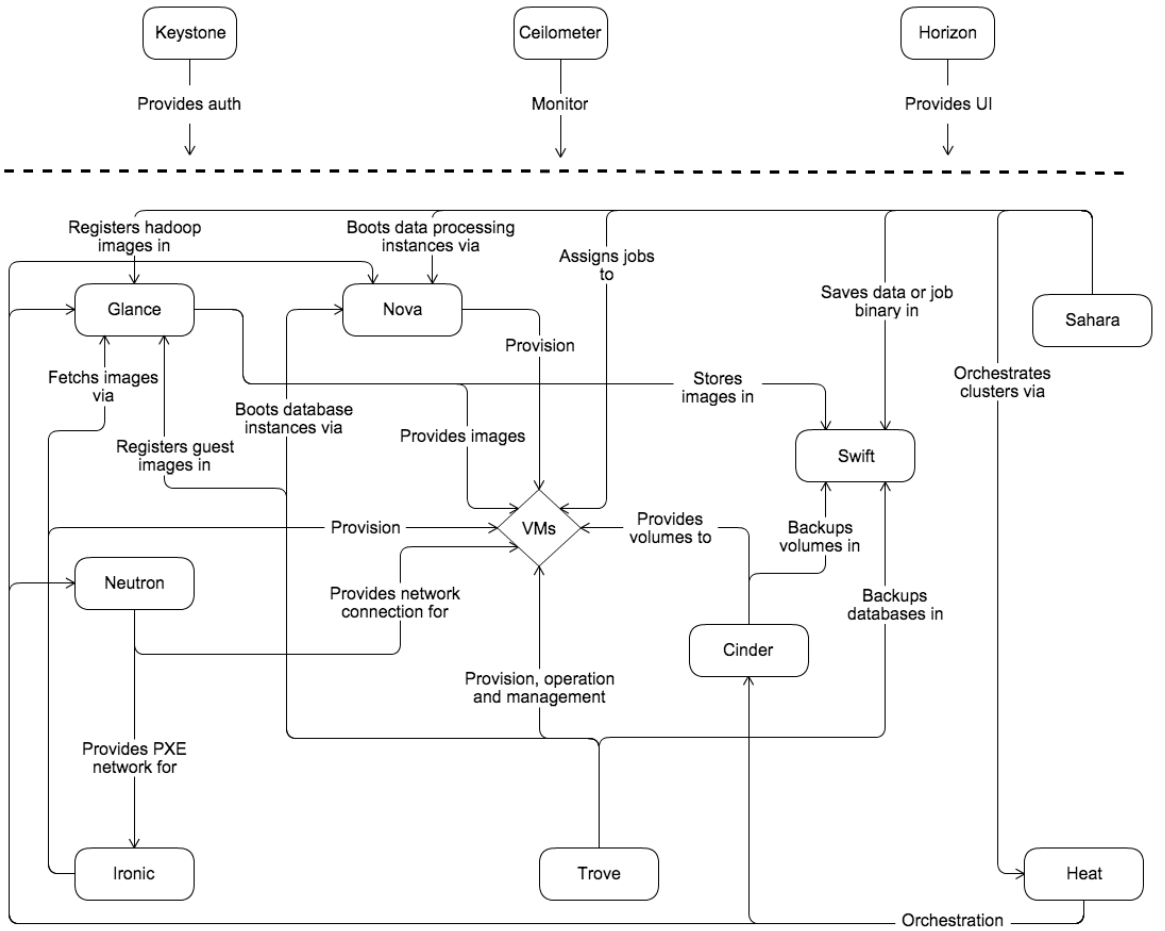

Figure 8. Openstack conceptual architecture [16]

4. Descriptors summary

4.1. Main descriptors

In order to define the main characteristics of the components in the network, VNFs and NSs, it is needed a configuration template. This template is called descriptor and is written in YAML format.

YAML is a human friendly language because is easy to read, implement and understand. It uses native data structures of agile languages so it permits portability between other programming languages.

In the following subsections I will give a high-level vision of the descriptors, a more detailed information about all the data models can be found in the document “OSM Information Model Release Two” from page 14-42 and from 49-112.

4.1.1. VNF Descriptors

The deployment and the operational behavior requirements of a VNF are described in a template called the virtual network function descriptor (VNFD) [17]. This information is used by the VNFM when instantiating VNF and in its lifecycle management. The NFVO also uses it to manage and orchestrate network services and virtualised resources in the NFVI.

The VNFD provides information on VNF images, connectivity (connection points and virtual links), platform resource requirements (CPU, memory, interfaces), Enhanced Platform Awareness (EPA) attributes and scaling properties, among others.

It also defines the Virtual Deployment Unit (VDU) which is a basic part of a VNF. VDUs are deployed as virtual machines (VMs) that host the network function. They specify compute resource and software component, storage, memory and network requirements.

Each VNF has a set of internal and external connection points. The connection points, whether internal or external, are connected using virtual links. Each virtual link has references to two or more 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 network service descriptor (NSD) to chain VNFs.

Figure 9 shows the content of VNFD: lists of VDUs, internal connection points, internal virtual links, and external connection points.

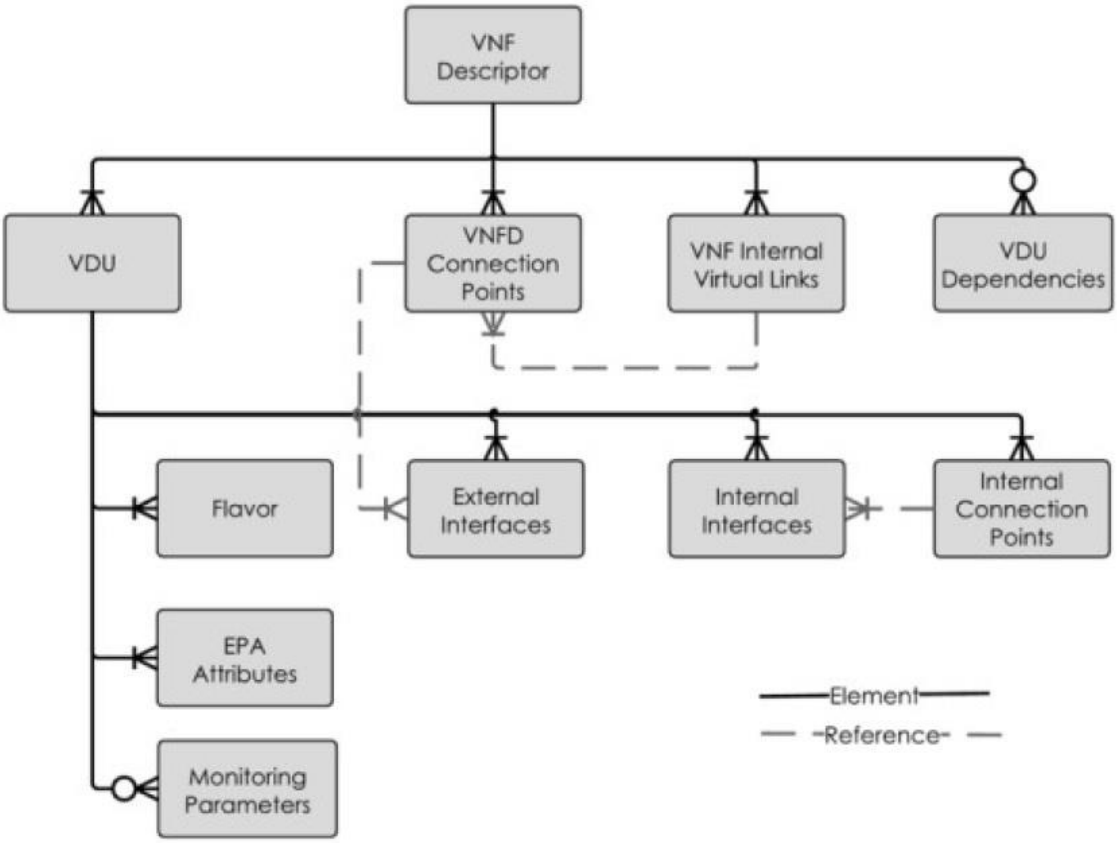

Figure 9. VNFD high-level object model [17]

4.1.2. NS Descriptors

The network service descriptor (NSD) [17] is the top-level construct used for designing the service chains, referencing all other descriptors that describe components that are part of that network service.

The NSD describes deployment flavors of the network service. The NFVO uses it to instantiate a network service.

The NSD defines the VNFs that make up the network service and the virtual links (VL) through which they are connected. The VNF forwarding graphs descriptors (VNFFGD) determine the traffic flow in the service chain. The NSD also exposes a set of connection points to enable connectivity to other network services or to the external world.

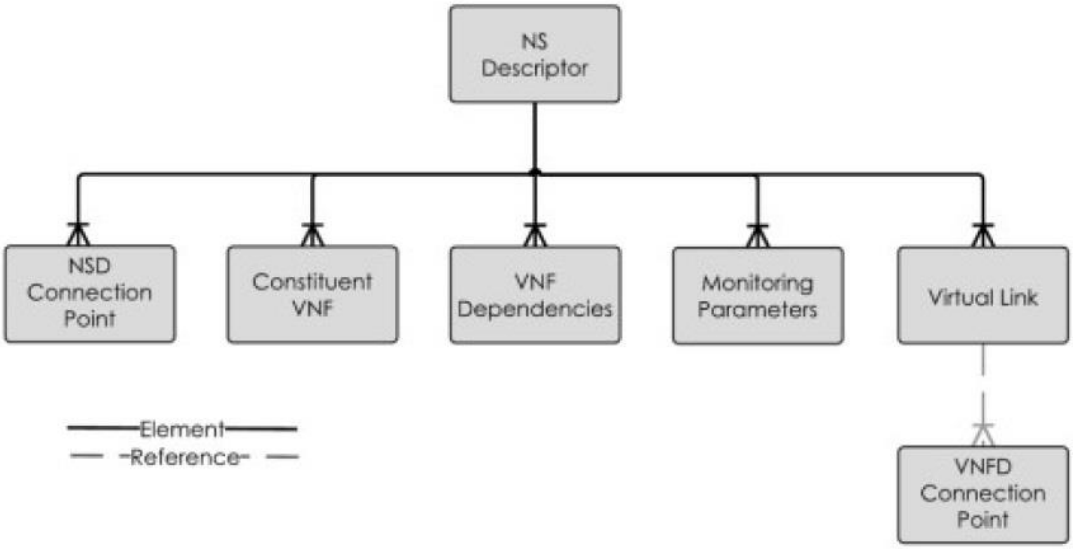

Figure 10. NSD high-level object model [17]

4.2. Special features

With all these fields it is possible to define a lot of different networks services with different characteristics. In this section we will focus in the two that we tried to develop and if we finally could achieve it.

Define the specific datacenter for each VNF

It is usual that in network services, the constituent VNFs belong to multiple datacenters which are geographically located in different places. This feature is called multi-site, and it is supported since the first release [18]. We wanted to prove if given a NS descriptor from the fields we could define where the different VNF are located. After this, do a partition of the information depending on the resources needed and pass them to the relevant VIMs in charge of the underlying infrastructures.

There is one field that could fulfil this feature. Inside the *init-params* field in the NSD, there's one parameter called *vim-network-name*, but it does not define in which VIM the NFV is deployed but the name of the network in the cloud account.

The only option is to define the vim in the moment of the instantiation of the network service. When the NS is instantiated, it is necessary to add the vim-account. The instantiation can be done from the command line (CL) or from the graphical user interface (GUI).

From the CI, the command is the following:

```
Osm ns-create –nsd_name <nsd-name> --ns_name <ns-instance-name> --vim_account <data-center-name>
```

The problem in this way is that only one vim account can be defined. So the only solution is to do it through the GUI.

It can be done in the Launchpad, that is where we instantiate the services. Before instantiating them, some input parameters have to be filled such as the 'Instance Name', 'Resource Orchestrator' and the 'Datacenter' for each VNF. Here is where we define the corresponding VIM that must have been configured before to OSM.

Define VLANs

Another common deployment is the use of Virtual Local Area Networks for network segmentation. In which independent logical networks actually share the bandwidth. The separation of network applications can be done by applying tags to network packets. These tags are handled in the networking system and the packets are sent through the corresponding ports.

VLAN can be configured through software and this allows the grouping of hosts which may not be directly connected to the same network switch.

This implementation can be directly achieved with one descriptor. In the NSD, for each VLD there's a field in which the provider network can be described:

`provider-network:`

- `Physical-network:` name of the physical network on which the provider network is built
- `Overlay-type:` identifies the overlay type network (LOCAL, FLAT, VLAN, VXLAN, GRE)
- `Segmentation-id:` the identifier of the segment

5. Examples with scenarios

In this section the scenarios created after the analysis of the descriptors will be explained. OSM provides some simple scenarios with one VNF and with two VNFs. In this project we have designed two more complex scenarios with 3 VNFs each one. The idea is that these new scenarios, together with the others, are the basic configurations in order to deploy more complex ones.

Both scenarios will define a network service in which the connectivity between the VNFs is segmented in VLANs.

First of all we will explain the VNFs created and then the network services developed with these VNFs. The YAML configurations of each VNF and NS can be found in the Annex A.

• Type1-vnfd

The first VNF has three external connection points, one for the management and the other two for data exchange.

It is composed by three VDUs (mgmt, data1, data2). Each VDU has two interfaces, external and internal. The external interface is connected to the appropriate external connection point that will be used to connect the VNF to another VNF. The internal interface will be connected to its corresponding internal connection point.

There is one internal connection point for each VDU so that all of them merge in the internal virtual link which enables the connectivity between the three VDUs.

Figure 12 shows the diagram of VNF type 1.

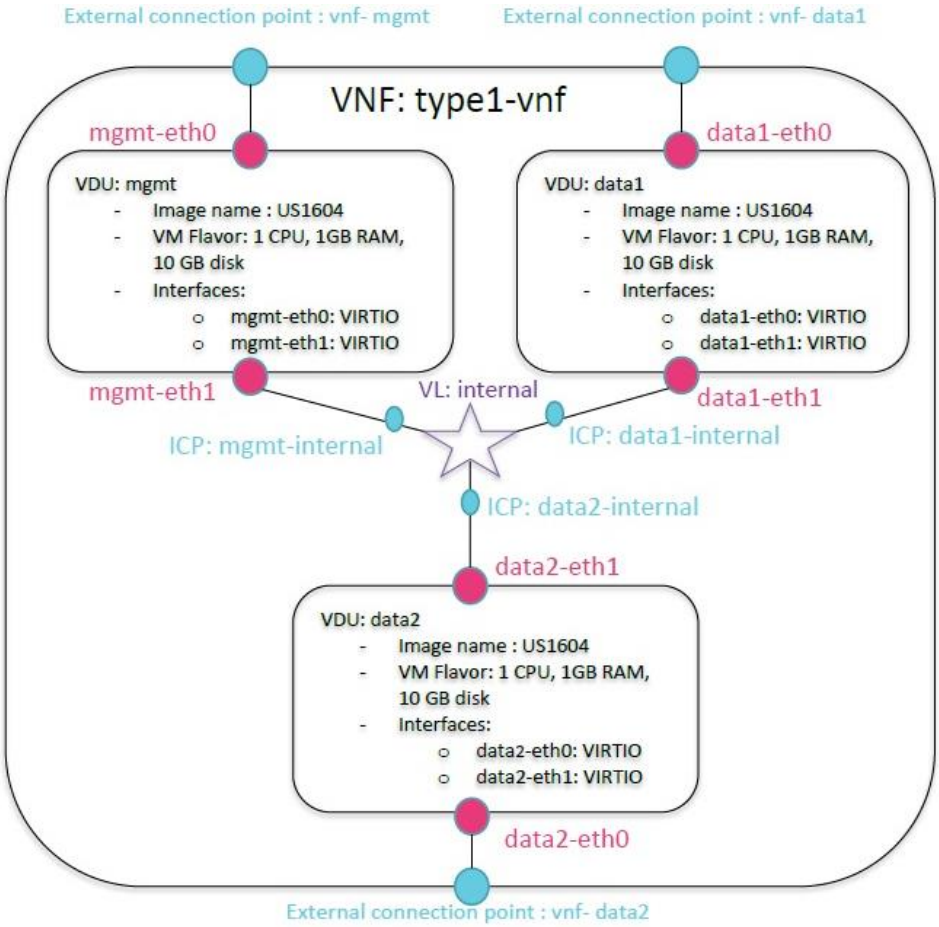

Figure 11. VNF type 1 diagram

• Type2-vnfd

The second VNF has two external connection points, one for management and one for data.

It is constituted by two VDUs that have the same components as the before mentioned. There is one internal virtual link to connect the two VDUs.

Type2-vnf diagram is shown in Figure 13.

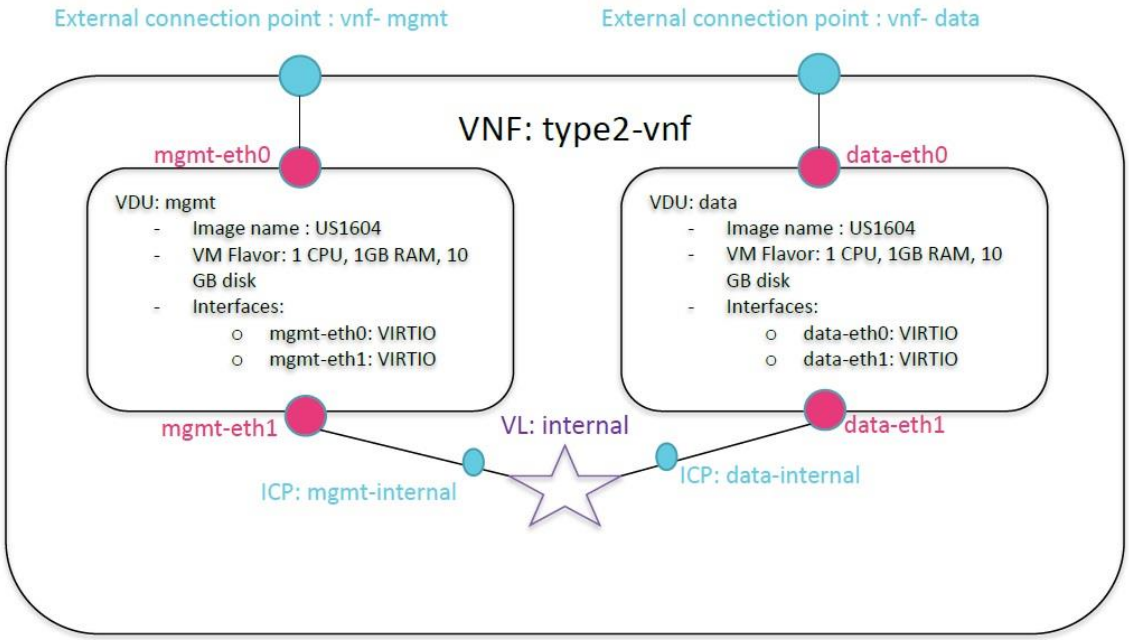

Figure 12. VNF type 2 diagram

5.1. Scenario 1

The first scenario wants to deploy a network service with three VNFs. The connectivity will be between the first VNF and the second, and the first VNF with the third. The second and the third VNFs will not be directly connected.

The VNF connected to the other two will be of type 1 as it will have three connection points (one management, two data). The other two VNFs will be of type 2 as they only need one management and one data connection points.

The NS will have three virtual links, as said before, one for management and two for data exchange. Each virtual link will belongs to a separate VLAN.

It is worth noting that to do a proper deployment it is important to have a static configuration, the management link. It is essential because in case of failure of the other interfaces, we can connect directly to the virtual machines. Every time we turn on the system, the VMs are instantly connected to the management link. Despite this, information data is not supposed to flow through the management link but through the datanet link.

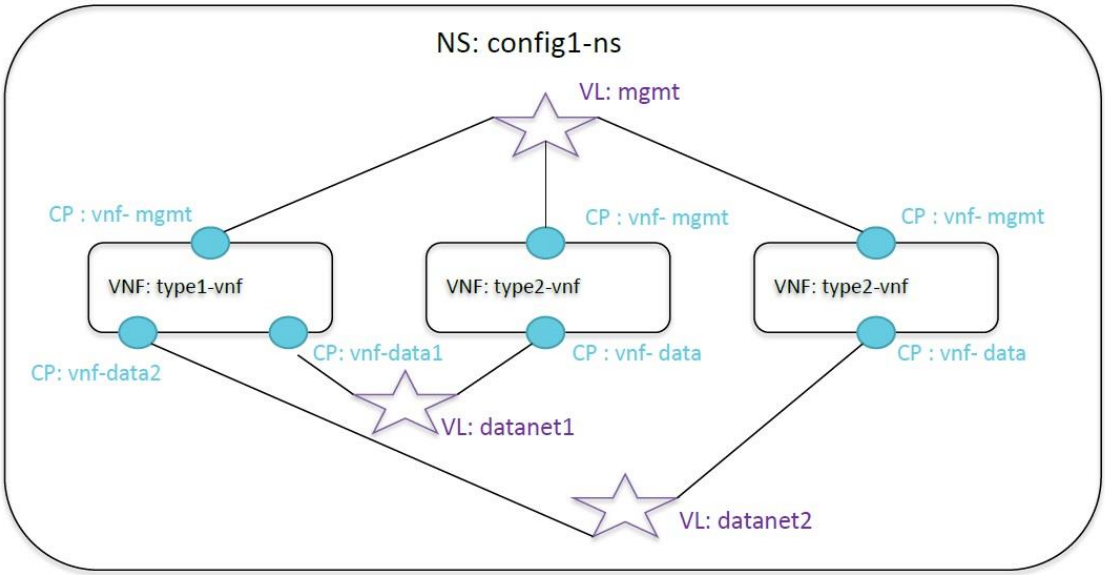

Figure 13. NS configuration 1

5.2. Scenario 2

The second scenario is similar to the first one but with the addition of a new data virtual link. This new VL will allow the connectivity between the second and the third VNF, which weren't directly connected before. In this case, the NS contains three VNFs of type 1, because they all need two data interfaces.

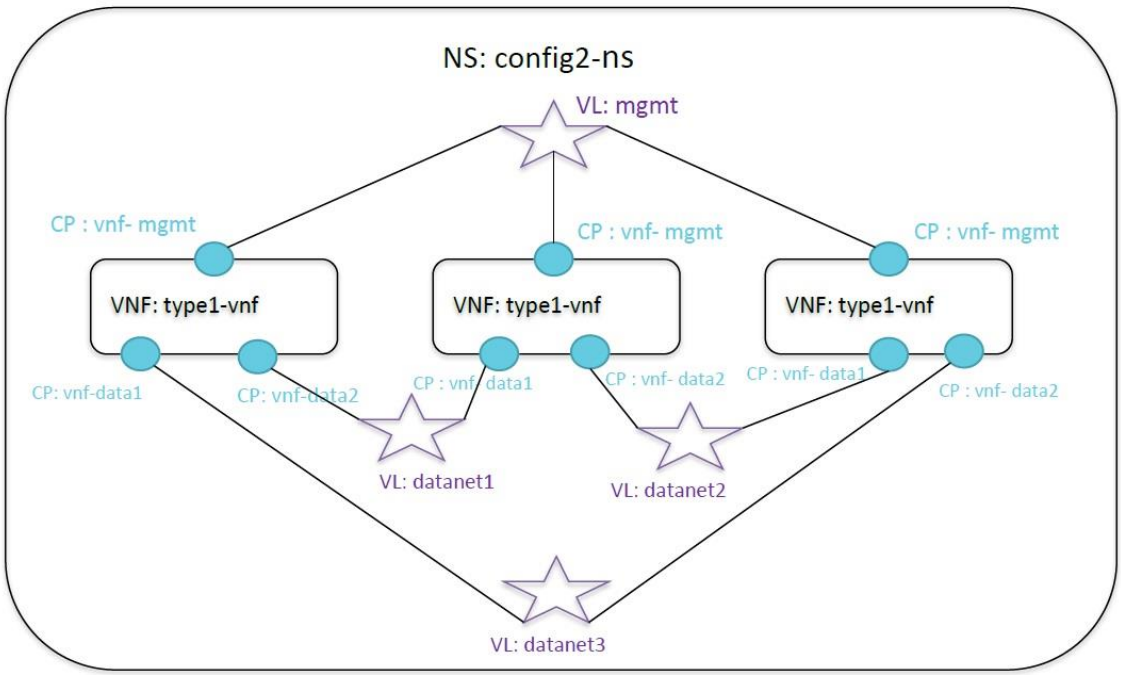

Figure 14. NS configuration 2

5.3. Results

Finally, we could define these network services following the methodology of descriptors. Unfortunately, we just defined the services because deploying them meant that we had to have a real infrastructure implementation. Even though we could not do the deployment, we load them into OSM to prove if the syntax was correct and if the NS could be deployed in a real environment.

6. Conclusions

Taking everything into consideration, it can be said that SDN and NFV have an important role in current telecom networks. They are emerging technologies which have had a great growth in the last years and which are improving day by day. On the other hand, service descriptors have a large number of fields to define exactly the components that are part of it and their characteristics. This allows the creation of very different services and deployments.

Even though it has been difficult, I can say that, personally, I have achieved the objectives of understanding SDN and NFV. After acquiring some basic knowledge, I could focus in Open Source MANO and especially in network service descriptors. This has allowed me to create some network services although we finally could not deploy them.

Bibliography:

[1] “Software-Defined Networking (SDN): Layers and Architecture Terminology”. [Online]. Available: https://tools.ietf.org/html/rfc7426

[2] “Understanding the SDN Architecture – SDN Control Plane & SDN Data Plane”. [Online]. Available: https://www.sdxcentral.com/sdn/definitions/inside-sdn-architecture/

[3] “7 Essentials Of Software-Defined Networking”. [Online]. Available: https://www.networkcomputing.com/cloud-infrastructure/7-essentials-software-defined-networking/1672824201

[4] “An introduction to SDN”. [Online]. Available: https://qmonnet.github.io/whirl-offload/2016/07/08/introduction-to-sdn/

[5] “What is OpenFlow? Definition and How it Relates to SDN”. [Online]. Available: https://www.sdxcentral.com/sdn/definitions/what-is-openflow/

[6] “Network Functions Virtualisation”. [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/nfv

[7] *Network Functions Virtualisation (NFV); Architectural Framework*. ETSI GS NFV 002 V1.2.1, December 2014

[8] “What Is NFV Infrastructure (NFVI)?” . [Online]. Available: https://www.sdxcentral.com/nfv/definitions/nfv-infrastructure-nfvi-definition/

[9] *OSM Release THREE a technical overview.* ETSI OSM Community, October 2017.

[10] nfv introductory white paper

[11] *Network Functions Virtualisation (NFV); Management and Orchestration*. ETSI GS NFV-MAN 001 V1.1.1, December 2014.

[12] “What is an NFV Orchestrator (NFVO)?”. [Online]. Available: https://www.sdxcentral.com/nfv/definitions/nfv-orchestrator-nfvo-definition/

[13] **“**What is Virtualized Infrastructure Manager (VIM)?”. [Online]. Available: https://www.sdxcentral.com/nfv/definitions/virtualized-infrastructure-manager-vim-definition/

[14] *OSM Release FOUR Technical Overview.* ETSI, May 2018.

[15] “What is OpenStack?”. [Online]. Available: https://opensource.com/resources/what-is-openstack

[16] “Openstack Documents”. [Online]. Available: https://docs.openstack.org/

[17] *Open Source MANO Information Model Release TWO.* ETSI, July 2017.

[18] *OSM Release ONE a technical overview.* ETSI OSM Community, October 2016.

Appendix A

VNF type 1 descriptor

```
vnfd:vnfd-catalog:
    vnfd:
    -   id: type1-vnf
        name: type1-vnf
        short-name: type1-vnf
        version: '1.0'
        description: A simple VNF descriptor with three VDU
        logo: osm.png
        connection-point:
        -   name: vnf-mgmt
            id: vnf-mgmt
            type: VPORT
       -   name: vnf-data1
            id: vnf-data1
            type: VPORT
       -   name: vnf-data2
            id: vnf-data2
            type: VPORT
        mgmt-interface:
            cp: vnf-mgmt
        internal-vld:
        -   id: internal
            name: internal
            short-name: internal
            type: ELAN
            internal-connection-point:
            -   id-ref: mgmt-internal
            -   id-ref: data1-internal
            -   id-ref: data2-internal

        vdu:
       ##FIRST VDU
        -   id: mgmt
            name: mgmt
            image: US1604
            count: '1'
            vm-flavor:
                vcpu-count: '1'
                memory-mb: '1024'
                storage-gb: '10'
            interface:
            -   name: mgmt-eth0
                position: '1'
                type: EXTERNAL
                virtual-interface:
                    type: VIRTIO
                external-connection-point-ref: vnf-mgmt
            -   name: mgmt-eth1
                position: '2'
                type: INTERNAL
                virtual-interface:
                    type: VIRTIO
                internal-connection-point-ref: mgmt-internal
            internal-connection-point:
            -   id: mgmt-internal
                name: mgmt-internal
                short-name: mgmt-internal
                type: VPORT

       ##SECOND VDU
```

```
 -   id: data1
     name: data1
     image: US1604
     count: '1'
     vm-flavor:
         vcpu-count: '1'
         memory-mb: '1024'
         storage-gb: '10'
     interface:
     -   name: data1-eth0
         position: '1'
         type: EXTERNAL
         virtual-interface:
             type: VIRTIO
         external-connection-point-ref: vnf-data1
     -   name: data1-eth1
         position: '2'
         type: INTERNAL
         virtual-interface:
             type: VIRTIO
         internal-connection-point-ref: data1-internal
     internal-connection-point:
     -   id: data1-internal
         name: data1-internal
         short-name: data1-internal
         type: VPORT

##THIRD VDU
 -   id: data2
     name: data2
     image: US1604
     count: '1'
     vm-flavor:
         vcpu-count: '1'
         memory-mb: '1024'
         storage-gb: '10'
     interface:
     -   name: data2-eth0
         position: '1'
         type: EXTERNAL
         virtual-interface:
             type: VIRTIO
         external-connection-point-ref: vnf-data2
     -   name: data1-eth1
         position: '2'
         type: INTERNAL
         virtual-interface:
             type: VIRTIO
         internal-connection-point-ref: data2-internal
     internal-connection-point:
     -   id: data2-internal
         name: data2-internal
         short-name: data2-internal
         type: VPORT
```

VNF type 2 descriptor

```
vnfd:vnfd-catalog:
    vnfd:
    -   id: type2-vnf
        name: type2-vnf
        short-name: type2-vnf
        version: '1.0'
        description: A simple VNF descriptor with two VDU
        logo: osm.png
        connection-point:
        -   name: vnf-mgmt
            id: vnf-mgmt
            type: VPORT
       -   name: vnf-data
            id: vnf-data
            type: VPORT
        mgmt-interface:
            cp: vnf-mgmt
        internal-vld:
        -   id: internal
            name: internal
            short-name: internal
            type: ELAN
            internal-connection-point:
            -   id-ref: mgmt-internal
            -   id-ref: data-internal

        vdu:
       ##FIRST VDU
        -   id: mgmt
            name: mgmt
            image: US1604
            count: '1'
            vm-flavor:
                vcpu-count: '1'
                memory-mb: '1024'
                storage-gb: '10'
            interface:
            -   name: mgmt-eth0
                position: '1'
                type: EXTERNAL
                virtual-interface:
                    type: VIRTIO
                external-connection-point-ref: vnf-mgmt
            -   name: mgmt-eth1
                position: '2'
                type: INTERNAL
                virtual-interface:
                    type: VIRTIO
                internal-connection-point-ref: mgmt-internal
            internal-connection-point:
            -   id: mgmt-internal
                name: mgmt-internal
                short-name: mgmt-internal
                type: VPORT

       ##SECOND VDU
        -   id: data
            name: data
            image: US1604
            count: '1'
            vm-flavor:
                vcpu-count: '1'
                memory-mb: '1024'
```

```
            storage-gb: '10'
        interface:
        -   name: data-eth0
            position: '1'
            type: EXTERNAL
            virtual-interface:
                type: VIRTIO
            external-connection-point-ref: vnf-data
        -   name: data-eth1
            position: '2'
            type: INTERNAL
            virtual-interface:
                type: VIRTIO
            internal-connection-point-ref: data-internal
        internal-connection-point:
        -   id: data-internal
            name: data-internal
            short-name: data-internal
            type: VPORT
```

NS configuration 1 descriptor

```
nsd:nsd-catalog:
    nsd:
    -   id: config1-ns
        name: config1-ns
        short-name: config1-ns
        description: NS with 3 VNFs connected by datanet (and mgmtnet) VLs
        version: '1.0'
        logo: osm.png
        constituent-vnfd:
        -   vnfd-id-ref: type1-vnf
            member-vnf-index: '1'
        -   vnfd-id-ref: type2-vnf
            member-vnf-index: '2'
        -   vnfd-id-ref: type2-vnf
            member-vnf-index: '3'

        vld:
       #FIRST VLD
        -   id: mgmtnet
            name: mgmtnet
            short-name: mgmtnet
            type: ELAN
            mgmt-network: 'true'
            vim-network-name: mgmt
           vnfd-connection-point-ref:
            -   vnfd-id-ref: type1-vnf
                member-vnf-index-ref: '1'
                vnfd-connection-point-ref: vnf-mgmt
            -   vnfd-id-ref: type2-vnf
                member-vnf-index-ref: '2'
                vnfd-connection-point-ref: vnf-mgmt
            -   vnfd-id-ref: type2-vnf
                member-vnf-index-ref: '3'
                vnfd-connection-point-ref: vnf-mgmt
           provider-network:
              physical-network:
              overlay-type: VLAN
              segmentation-id: 100
       #SECOND VLD
        -   id: datanet1
            name: datanet1
```

```
            short-name: datanet1
            type: ELAN
            vnfd-connection-point-ref:
            -   vnfd-id-ref: type1-vnf
                member-vnf-index-ref: '1'
                vnfd-connection-point-ref: vnf-data1
            -   vnfd-id-ref: type2-vnf
                member-vnf-index-ref: '2'
                vnfd-connection-point-ref: vnf-data
           provider-network:
              physical-network:
              overlay-type: VLAN
              segmentation-id: 101
        #THIRD VLD
         -   id: datanet2
             name: datanet2
             short-name: datanet2
             type: ELAN
             vnfd-connection-point-ref:
             -   vnfd-id-ref: type1-vnf
                 member-vnf-index-ref: '1'
                 vnfd-connection-point-ref: vnf-data2
             -   vnfd-id-ref: type2-vnf
                 member-vnf-index-ref: '3'
                 vnfd-connection-point-ref: vnf-data
           provider-network:
              physical-network:
              overlay-type: VLAN
              segmentation-id: 102
```

NS configuration 2 descriptor

```
nsd:nsd-catalog:
    nsd:
    -   id: config2-ns
        name: config2-ns
        short-name: config2-ns
        description: NS with 3 VNFs connected (two by two) by datanet (and mgmtnet) VLs
        version: '1.0'
        logo: osm.png
        constituent-vnfd:
        -   vnfd-id-ref: type1-vnf
            member-vnf-index: '1'
        -   vnfd-id-ref: type1-vnf
            member-vnf-index: '2'
        -   vnfd-id-ref: type1-vnf
            member-vnf-index: '3'

        vld:
        #FIRST VLD
         -   id: mgmtnet
             name: mgmtnet
             short-name: mgmtnet
             type: ELAN
             mgmt-network: 'true'
             vim-network-name: mgmt
             vnfd-connection-point-ref:
             -   vnfd-id-ref: type1-vnf
                 member-vnf-index-ref: '1'
                 vnfd-connection-point-ref: vnf-mgmt
             -   vnfd-id-ref: type1-vnf
                 member-vnf-index-ref: '2'
                 vnfd-connection-point-ref: vnf-mgmt
```

```
     -   vnfd-id-ref: type1-vnf
         member-vnf-index-ref: '3'
         vnfd-connection-point-ref: vnf-mgmt
    provider-network:
       physical-network:
       overlay-type: VLAN
       segmentation-id: 100
#SECOND VLD
 -   id: datanet1
     name: datanet1
     short-name: datanet1
     type: ELAN
     vnfd-connection-point-ref:
     -   vnfd-id-ref: type1-vnf
         member-vnf-index-ref: '1'
         vnfd-connection-point-ref: vnf-data2
     -   vnfd-id-ref: type1-vnf
         member-vnf-index-ref: '2'
         vnfd-connection-point-ref: vnf-data1
    provider-network:
       physical-network:
       overlay-type: VLAN
       segmentation-id: 101
#THRID VLD
 -   id: datanet2
     name: datanet2
     short-name: datanet2
     type: ELAN
     vnfd-connection-point-ref:
     -   vnfd-id-ref: type1-vnf
         member-vnf-index-ref: '2'
         vnfd-connection-point-ref: vnf-data2
     -   vnfd-id-ref: type1-vnf
         member-vnf-index-ref: '3'
         vnfd-connection-point-ref: vnf-data1
    provider-network:
       physical-network:
       overlay-type: VLAN
       segmentation-id: 102
#FOURTH VLD
 -   id: datanet3
     name: datanet3
     short-name: datanet3
     type: ELAN
     vnfd-connection-point-ref:
     -   vnfd-id-ref: type1-vnf
         member-vnf-index-ref: '1'
         vnfd-connection-point-ref: vnf-data1
     -   vnfd-id-ref: type1-vnf
         member-vnf-index-ref: '3'
         vnfd-connection-point-ref: vnf-data2
    provider-network:
       physical-network:
       overlay-type: VLAN
       segmentation-id: 103
```

Glossary

A list of all acronyms and the meaning they stand for.

API	Application Programming Interface
CLI	Command-line Interface
DNS	Domain Name System
EM	Element Manager
EPA	Enhanced Platform Awareness
ETSI	European Telecommunications Standards Institute
GUI	Graphical User Interface
IaaS	Infrastructure as a Service
ISG	Industry Specification Group
MANO	Management and Orchestration
NAT	Network Address Translation
NFV	Network Function Virtualisation
NFV MANO	Network Function Virtualisation Management and Orchestration
NFVI	Network Function Virtualisation Infr
NFVO	Network Function Virtualisation nnnnn
NS	Network Service
NSD	Network Service Descriptor
N2VC	Network Service to VNF Communication
ODL	Open DayLight
ONF	Open Networking Foundation
ONOS	Open Network Operating System
OSM	Open Source MANO
OSS/BSS	Operation and Business Support Systems
RO	Resource Orchestrator
SDN	Software Defined Networking
SO	Service Orchestrator
UI	User Interface
VCA	VNF Configuration and Abstraction
VDU	Virtual Deployment Unit
VIM	Virtualized Infrastructure Manager
VLAN	Virtual Local Area Network
VLD	Virtual Link Descriptor

VM	Virtual Machine
VNF	Virtual Network Function
VNFD	Virtual Network Function Descriptor
VNFFGD	Virtual Network Function Forwarding Graph
VNFM	Virtual Network Function Manager
WP	Work Package
YAML	YAML Ain’t Markup Language