

HELLENIC MEDITERRANEAN UNIVERSITY

Department of Electrical and Computer Engineering

Curriculum Informatics Engineering T.I.



BACHELOR'S THESIS

VNF Validation over OpenMANO Orchestrator

Anastasios Giannoulis

(TP3919)

Supervisor: Dr. Evangelos Markakis

Heraklion, Crete 2021

Acknowledgments

I would first like to thank my supervisor, Dr. Evangelos Markakis for the excellent cooperation and communication all this time. His expertise was invaluable in formulating the research questions and methodology. I want to thank you also for your patient support and for all of the opportunities I was given to further my research.

I would also like to thank my tutors, Dr. Evangelos Pallis and Yannis Nikoloudakis, PhD Candidate, for their valuable guidance during my studies and work at Pasiphae Lab.

In addition, I would like to thank Nikolaos Zotos who believed in me, gave me a chance, and provided me with the hardware material for the installation of my dissertation.

I also take this opportunity to express a deep sense of gratitude to my mentor and colleague MSc George Alexiou, for his cordial support, valuable information, and guidance through all these years of cooperation and studying. You provided me with knowledge and tools that I needed to choose the right direction and successfully complete my dissertation. You are a role model for me, and I really admire you.

I am extremely grateful to my parents for their love, caring and sacrifices for educating and preparing me for my future. Your encouragement when the times got rough are much appreciated and duly noted. I could never ask for more support than you have already given me, and I deeply admire you for that.

I would also like to thank my friends and my little brother who provided stimulating discussions as well as happy distractions to rest my mind outside of my research.

Finally, I could not have completed this dissertation without the support of my life partner Artemisia Symeonidou who was the main reason I managed to complete my thesis. Thank you for giving me strength every day with your smile.

1. Abstract

With the rapid growth of cloud computing these days, the need for better management of hardware resources, network services and orchestration in a cloud infrastructure is necessary. Network Function Virtualization (NFV) concept specializes in Management and Orchestration (MANO) of a Virtual Infrastructure Manager (VIM) by turning network functions from devoted hardware to software known as Virtual Network Functions (VNF). Further research shows that the challenge is to create a solid and valid VNF to offer a full-scale network communication services on the top of the NFV layer and avoid collisions and errors with the hardware layer.

The scope of Thesis elaborates on the study, design, and implementation of a VNF validation service. The aim is to enable VNF vendors to validate their visualized network functions, to make sure a particular VNF can function properly within the exact client environment. Open-Source MANO is the main NFV Management and Orchestration software that we will use, the Openstack software will be our VIM and the validation procedure will be on our custom Angular dashboard.

The platform also provides a VNF Descriptor (VNFD) Generator which is responsible for creating a fully functional VNF based on the European Telecommunications Standards Institute (ETSI) standards, avoiding time-consuming and error-prone tasks that result from creating manually a VNFD. The Validation of the VNF will activated with just a push of a button, and the automated procedure will be in three parts: (a) the first validation of the descriptor build in Yet Another Next Generation (YANG) Yaml Data Model and based on a structure that follows the VNF ETSI standards, for more complex style of checking validity, (b) the second validation of the VNF initialization, deployment and connection with the public network of the VIM, and (c) the validation through the expose and display of the VNF interfaces and connection points of the Network Service (NS). The user also will be able to monitor in real time the successful uploaded Virtual Display Unit (VDU) attributes and the time of its deployment.

Table of Contents

Acknowledgments	2
1. Abstract	3
2. Acronyms	8
3. Introduction	10
4. NFV Fundamentals and Overview of Open Source MANO	11
4.1 Network Function Virtualization	11
4.2 Open Source MANO	12
4.2.1 Network Slices	14
4.2.2 Virtual Network Function	15
4.2.3 Network Services	16
4.2.4 OSM architecture	16
5. Technology Enablers	18
5.1. Open Source MANO	18
5.2. Openstack	19
5.3. VirtualBox	20
5.4. Google Firebase Real-Time Database	21
5.5. NodeJS	22
5.6. Angular	23
5.7. NG ALAIN	23
5.8. Apexcharts	23
5.9. Falcon	24
5.10. GitLab	25
5.11. Cloud-Init	25

6. Architecture and Service Validation Framework	26
6.1. Architecture	26
6.1.1. Cloud	26
6.1.2. Openstack as VIM.....	27
6.1.3. Back-End and Falcon as RESTful API.....	27
6.1.4. Open Source MANO.....	28
6.1.5. Validation Dashboard.....	28
6.2. Implementation.....	29
6.2.1. Openstack Installation.....	29
6.2.2. Openstack Custom Images.....	30
6.2.3. Automation scripts.....	31
6.3. Framework Use Case	33
6.3.1. Failure examples	44
7. Evaluation.....	47
7.1. Test Case 1	48
7.2. Test Case 2	48
7.3. Test Case 3	50
7.4. Test Case 4	50
7.5. Test Case 5	52
7.6. Test Case 6	53
8. Conclusion	54
9. References.....	55

Table of Figures

Figure 1 NFV architecture.....	11
Figure 2 OSM interaction with VIMs and VNFs.....	13
Figure 3 Network Slice descriptor.....	15
Figure 4 Virtual Network Function descriptor.....	15
Figure 5 Network Service descriptor.....	16
Figure 6 OSM architecture.....	17
Figure 7 Open Source Mano Dashboard.....	19
Figure 8 Openstack Logo.....	19
Figure 9 Openstack Dashboard.....	20
Figure 10 VirtualBox Logo.....	20
Figure 11 VirtualBox Menu.....	21
Figure 12 Firebase Logo.....	21
Figure 13 Firebase Dashboard.....	22
Figure 14 NodeJS Logo.....	22
Figure 15 Angular Logo.....	23
Figure 16 NG-ALAIN Logo.....	23
Figure 17 Apexcharts Logo.....	24
Figure 18 Dashboard concept.....	24
Figure 19 Falcon Logo.....	24
Figure 20 GitLab Logo.....	25
Figure 21 GitLab Dashboard.....	25
Figure 22 Architecture diagram.....	Error! Bookmark not defined.
Figure 23 Dashboard Login page.....	33
Figure 24 Dashboard.....	34
Figure 25 Internal VLD and External Connection Point form fields.....	34
Figure 26 VNFD form field.....	35
Figure 27 NSD field form.....	35
Figure 28 VNF Descriptor file.....	36
Figure 29 NS Descriptor file.....	37
Figure 30 Upload and YAML validation process.....	38
Figure 31 Deployment Stage component.....	39
Figure 32 Server Log modal.....	39
Figure 33 VDUs table.....	40
Figure 34 Openstack Network Topology.....	40
Figure 35 Dashboard after the NS deployment.....	41
Figure 36 Dashboard during the Monitoring.....	41
Figure 37 Instances Table.....	42
Figure 38 NS Instance (a).....	42
Figure 39 NS Instance (b).....	43
Figure 40 VNF Instance.....	43
Figure 41 VDU Instance.....	44

Figure 42 Failure Example 1.....	45
Figure 43 Failure Example 2.....	45
Figure 44 Deployment Stage error 1.....	46
Figure 45 Server Log of error 1.....	46
Figure 46 Deployment Stages error 2.....	47
Figure 47 Server Log of error 2.....	47
Figure 48 Test Case 1.....	48
Figure 49 Test Case 2.....	49
Figure 50 Test Case 2 VNF Instance modal.....	49
Figure 51 Test Case 3.....	50
Figure 52 Test case 4.....	51
Figure 53 Test Case 4 cloud init file.....	51
Figure 54 Test Case 4 VDUs Table.....	51
Figure 55 Test Case 4 Apache server running.....	52
Figure 56 Test Case 5.....	52
Figure 57 Test Case 6.....	53

2. Acronyms

API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
BaaS	Backend as a Service
CEP	Complex Event Streaming Systems
CP	Connection Point
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
E2E	End-to-End
ETSI	European Telecommunications Standards Institute
EPC	Evolved Packet Core
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IaaS	Infrastructure as a Service
IM	Information Model
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
LCM	Lightweight Life Cycle Manager
LAN	Local Area Network
MANO	Management and Orchestration
MEC	Mobile Edge Computing
NAT	Network Address Translation
NBI	Northbound Interface
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NFVO	Network Function Virtualization Orchestration
NPM	Node Package Manager
NS	Network Service

NSD	Network Service Descriptor
NST	Network Slices
NSO	Network Service Orchestration
OS	Operating System
OSM	Open Source MANO
RAM	Random Access Memory
REST	REpresentational State Transfer
RO	Resource Orchestrator
PDU	Physical Network Unit
PDUD	Physical Network Unit Descriptor
SDN	Software Defined Network
TAR	Tape Archive
VCA	VNF Configuration and Abstraction
VDU	Virtual Display Unit
VIM	Virtual Infrastructure Manager
VL	Virtual Link
VLD	Virtual Link Descriptor
VM	Virtual Machine
VNF	Virtual Network Functions
VNFD	Virtual Network Functions Descriptor
WAN	Wide Area Network
WSGI	Web Server Gateway Interface
XML	eXtensible Markup Language
YAML	YAML Ain't Markup Language
YANG	Yet Another Next Generation

3. Introduction

The mass demand and adoption of cloud computing and especially the development of products within the telecommunication industry led the companies to search for equipment that specifically meets protocol adherence and strict standards for stability and quality. The classic method of specifically hardware architecture has worked well in the past but soon had problems with producing and finding hardware (e.g., Application-Specific Integrated Circuit [ASIC]) that follows these requirements, resulting in the undoubted creation of large growth cycles and slow progress [13]. Soon the telecommunication companies turned their attention to NFV network architecture as the management and the orchestration of it became more efficient and flexible [7].

Network Function Virtualization (NFV) technology has introduced to IT companies a more flexible and cost effective way to manage and operate their network functions by shifting them from dedicated hardware to fully functional software network functions called VNFs. Virtual Network Functions (VNFs) can run at the top of the virtual infrastructure separate or combined as building blocks to offer a full scale network communication service known as service chaining. Responsible for the lifecycle management and orchestration in the NFV framework is the MANO functional block [2] which consists of the VIM, the VNF Manager and the Orchestrator.

The challenge faced by MANO and the VNF vendors is to create a deployment template for VNFs called VNFD, as well to validate this descriptor and the deployment of the VNF. Virtual Network Function Descriptor (VNFD) contains all the requirements and the information about the operational behavior of the VNF such as storage, virtual CPU cores, memory, external connection points, interfaces, and other network specifications. The whole process of creating a manually VNF descriptor is time consuming and very prone to errors, making it difficult for VNF vendors to deploy a VNF for their customers. From the above, it is understood that the Validation of a solid VNFD and a VNF deployment is necessary.

The thesis presents a solution of this issue by developing a web application in which the VNF vendors can automate create a VNF Descriptor based on the ETSI standards and then ensure the validation of their VNF instances through the Open Source MANO that is our main VNF-manager and orchestrator of the NFV architecture. The rest of this thesis is organized as follows. Section 4 presents the fundamentals of NFV architecture and the overview of Open Source MANO. Section

5 presents all the technology enablers and tools that were used for the purposes of this research. Section 6 presents the architecture of the framework as well the implementation. Section 7 presents the evaluation of the VNFD generator and the VNF Validation. Finally, Section 8 concludes this thesis achievements and discusses possible future updates for the web application and consequently around the whole framework.

4. NFV Fundamentals and Overview of Open Source MANO

In this Chapter we will present and analyze the fundamentals of the NFV Architecture and give an overview of the Open Source MANO (OSM).

4.1 Network Function Virtualization

Virtual Network Function (NFV) as we mentioned earlier is a technology that has aroused great interest of network providers, researchers and made a huge impact in the market. By leveraging virtualization technology to consolidate software network functions running on storage, switches, and servers, NFV managed to significantly reduce Operating Expenses (OPEX) and Capital Expenses (CAPEX) [9].

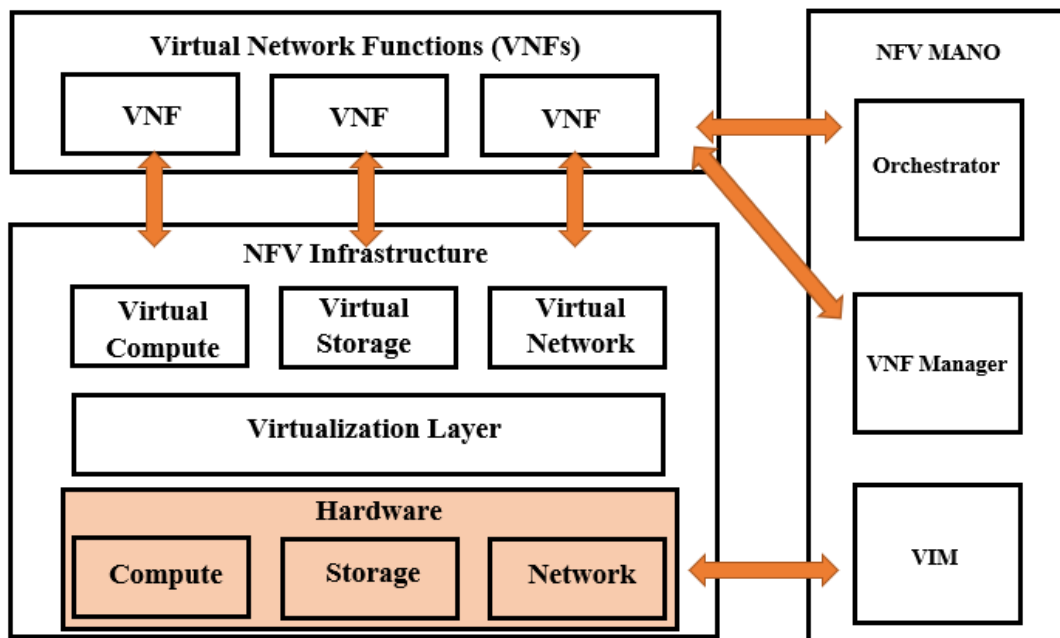


Figure 1 NFV architecture.

As we see from Figure 1, NFV architecture consists of 3 parts: (a) Virtual Network Functions (VNF) is the software blocks which enables the network functions, (b) NFV Infrastructure (NFVI) that is responsible of passing all the data and provides resources for the running network services, (c) NFV Management and Orchestration (MANO) which is responsible for building the connection tunnels between VNFs and orchestrating resources for NFVI.

On the other hand, the NFV MANO consists of 3 parts as well: (a) Virtual Infrastructure Manager (VIM) that manages controls and the interaction of the VNF with the NFV Infrastructure (NFVI) storage, compute, and network resources. It also has necessary monitoring and deployment tools for the virtualization layer, (b) VNF Manager which manages the lifecycle of VNF instances. It is responsible to initialize, query, update and terminate VNF instances, (c) Orchestrator that manages the lifecycle of network services, which includes instantiation, policy management, performance measurement and monitoring.

4.2 Open Source MANO

A lot of popular MANO solutions in recent years have made their appearance like Gohan¹, ONAP², Cloudify³, Tacker⁴ etc. but according to [1] the 2 most mature enough orchestrators that are driven by the ETSI NFV MANO specification are: ONAP and OSM. Both of them are the top contenders in the telecommunication area, because they successfully cover the sectors of the NFV Orchestration (NFVO), VNF Manager (VNFM), VIM, Software Defined Network (SDN) controllers, reference points and security, with ONAP has the leading for bit. Open Source MANO⁵ (OSM) as the name suggests is an ETSI⁶ hosted project to develop an Open Source NFV Management and Orchestration (MANO) software stack aligned with ETSI NFV [10]. Open Source MANO (OSM) aims for the creation of an End-to-End Network Service Orchestrator (E2E NSO) that will have the ability to automate and design real network services beyond the complexity and the difficulty that an environment has [3].

¹ <https://gohan.cloudwan.io/>

² <https://www.onap.org/>

³ <https://cloudify.co/>

⁴ <https://wiki.openstack.org/wiki/Tacker>

⁵ <https://osm.etsi.org/>

⁶ <https://etsi.org/>

The 3 things that must be settle so the OSM be able to work are :(a) each VIM must have an API endpoint so the OSM can reach it, (b) each VIM must have a management network that can provide an IP address to VNFs, (c) OSM must be able to reach this management network.

Figure 2 shows the OSM interaction with VIMs and VNFs:

- For the deployment of a VNF the OSM must talk with the VIM.
- OSM should be able to talk to the VNFs deployed in the VIM to execute so-called day-0, day-1, and day-2 configurations [10].

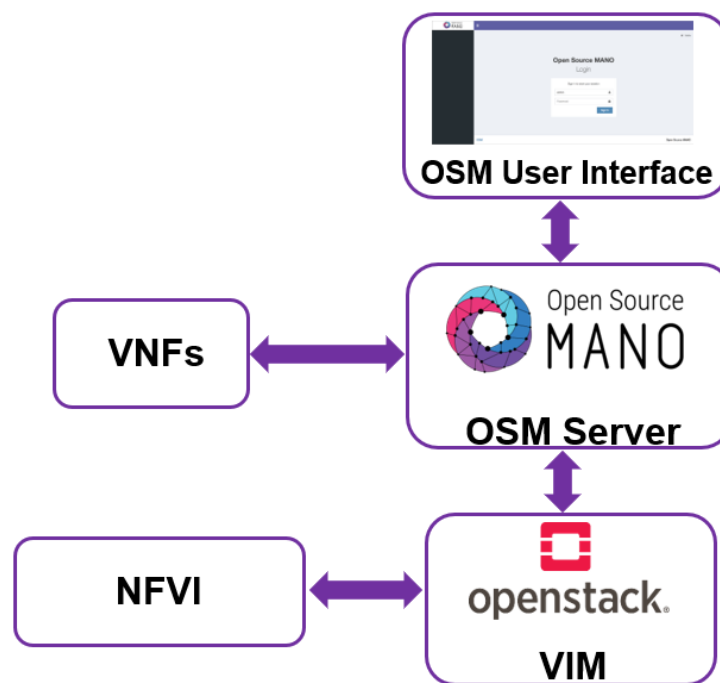


Figure 2 OSM interaction with VIMs and VNFs.

The range of features and requirements it could give to a VNF through OSM is really a big plus for it as a MANO. Thomas Dreibholz in [4] could manage to create the SIMULAMET EPC an opens source VNF for an Evolved Packet Core (EPC) aims to set up a 4G/5G testbed infrastructure easy and fast. The research on Mobile Edge Computing (MEC) instead of dealing with the configuration of the EPC and complex features for setting it up, now she/he can simply combine MEC functionalities with this VNF. But even with an OSM as a mature MANO where a user can easily deploy a VNF, the critical part is how to build it and validate it, which is the topic in this thesis.

Open Source MANO (OSM) provides mature NFV technologies for the VNF vendors so they can easily test and validate their network services, manage, and orchestrate their NFV Infrastructures both at the production level and at the commercial level⁷. Thanks to the four key aspects of the OSM, the minimization of integration efforts is more approachable.

1. A well-structured Information Model⁸ (IM) which provides automation and modeling in the complete lifecycle of Network Slices (NST), Network functions and Network Services from the time that the machine gets ready to be managed (Day-0) to the full configuration and management of the machine (Day-1, Day-2). The IM is aligned with ETSI NFV SOL006.
2. A unified Northbound Interface⁹ (NBI) which puts under control the Network Slices, the Network Services, and enables the full operation of the system. It is the tool that offers the functionality of managing the lifecycle of the NSTs and the NSs. Northbound Interface is based on NFV SOL005.
3. The possibility that the Network Service can extend across the different domains identified such as the physical/virtual ports in the OSM.
4. The management for the lifecycle of Network Slices, taking the role of Slice manager and supporting the operations which are.

4.2.1 Network Slices

Network Slices referred to the overlay of multiple networks on top of a public/shared network in a VIM. The users can create Network Slices inside the VNF and the NS by provide their characteristics in the yaml descriptors. Figure 3 shows how 2 network slice subnets are connected by Virtual Links Descriptors (VLDs) through the connection points of the NSs.

⁷ <https://osm.etsi.org/docs/user-guide/02-osm-architecture-and-functions.html?highlight=architecture>

⁸ <https://osm.etsi.org/docs/user-guide/11-osm-im.html>

⁹ <https://osm.etsi.org/docs/user-guide/12-osm-nbi.html>

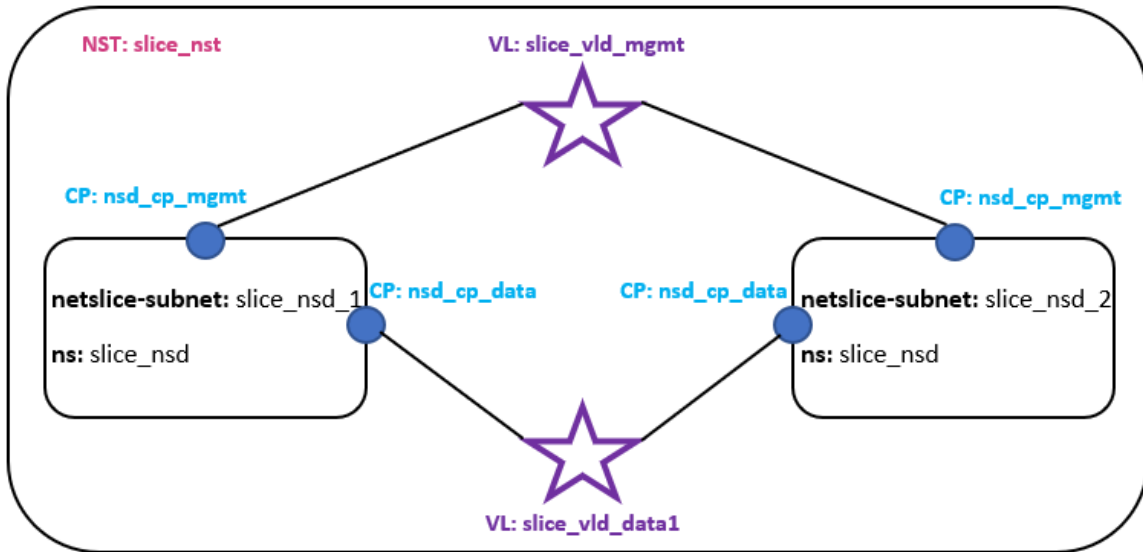


Figure 3 Network Slice descriptor.

4.2.2 Virtual Network Function

Virtual Network Function is a virtual network service running on top of the virtualization layer of the NFV and most of the time include virtualized firewalls, Network Address Translation (NAT), routers, Wide Area Network (WAN) optimization services. The user has the ability to create multiple connection points to connect the VDU interfaces with the outer of the VNF. Figure 4 shows a VNF with one VDU and 2 connection points. The VDU specifications are 2 vCPU, 4 GB, 10 GB Disk, and 'ubuntu1604' image.

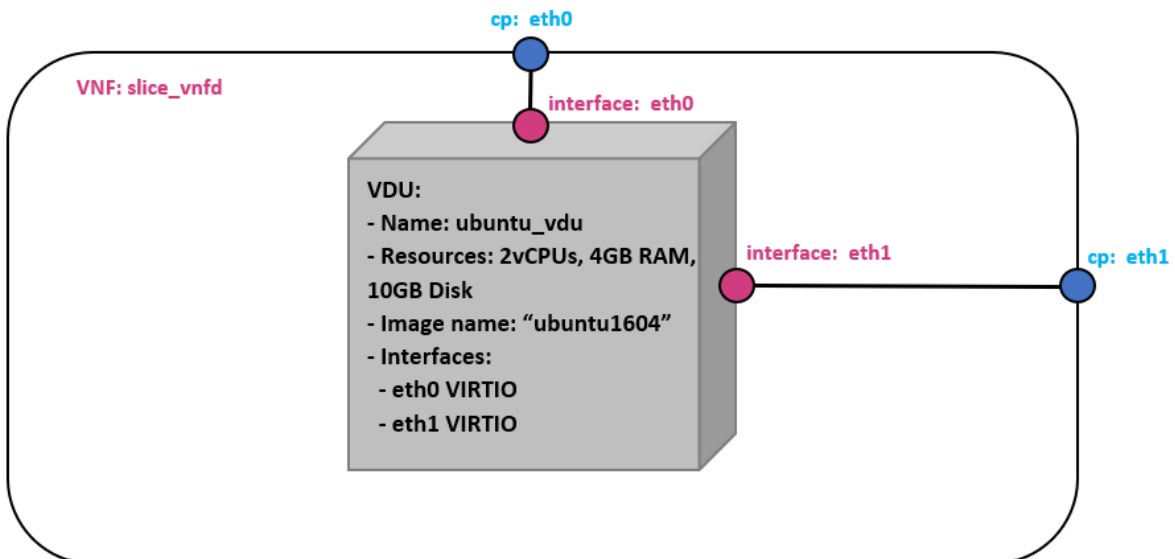


Figure 4 Virtual Network Function descriptor.

4.2.3 Network Services

Network Service is the combination of all VNFs and network slices running on the top of the VIM. Figure 5 shows a NS with 2 VNFs connected to a public network.

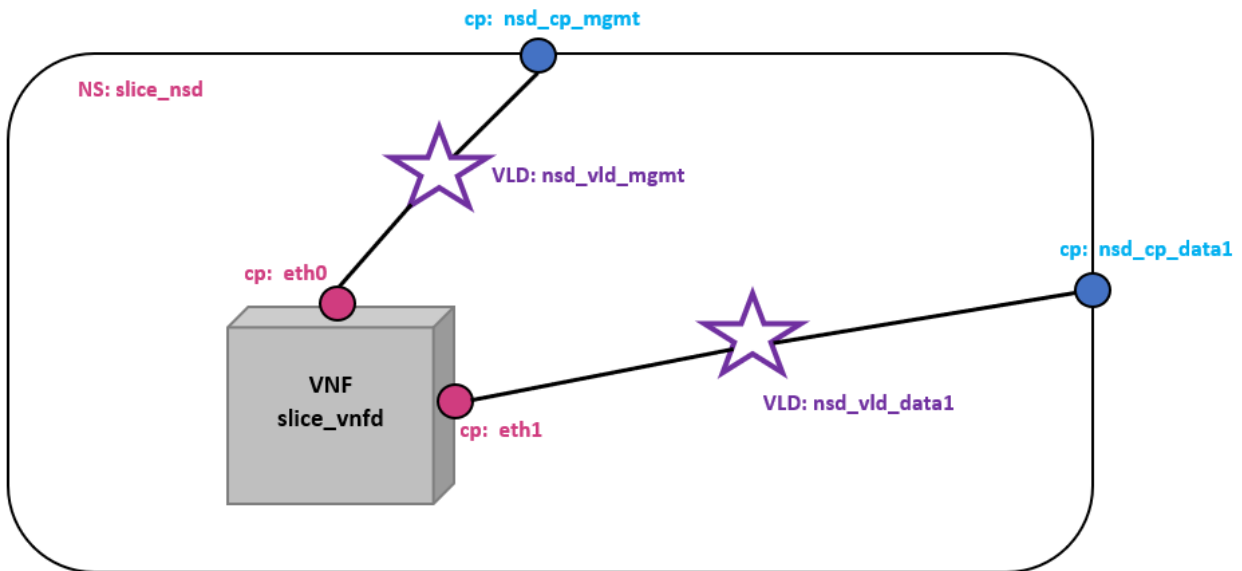


Figure 5 Network Service descriptor.

4.2.4 OSM architecture

As we can see in Figure 6, the OSM consists of 3 main blocks. (a) A Common Database based on NoSQL with an Object Storage, (b) a Kafka bus, (c) the NBI. Let's see a small description about them and understand how all these components work together:

- Kafka bus is used for communications between components, for real time streams of data. Is used to feed events to Complex Event Streaming Systems (CEP), Internet of Things (IoT) systems and to provide durability¹⁰.
- Common Database is what it is called, a database that stores the objects as the VNF packages, VNF Instances, Network Services etc.
- NBI as we said earlier is RESTful which admits both YAML/JSON and provides communications depending on the API calls between the Kafka bus and the Common Database.

¹⁰ <https://dzone.com/articles/what-is-kafka>

- Lightweight Life Cycle Manager (LCM) where the 2 containers, VNF Configuration and Abstracts (VCA), and Resource Orchestrator (RO) are running.
- VCA container is responsible for the configuration and the modeling of a VNF.
- RO container is responsible for the interconnection and the deployment of the VNF and VDUs in the instance with required resources, interacting with the Openstack controller.

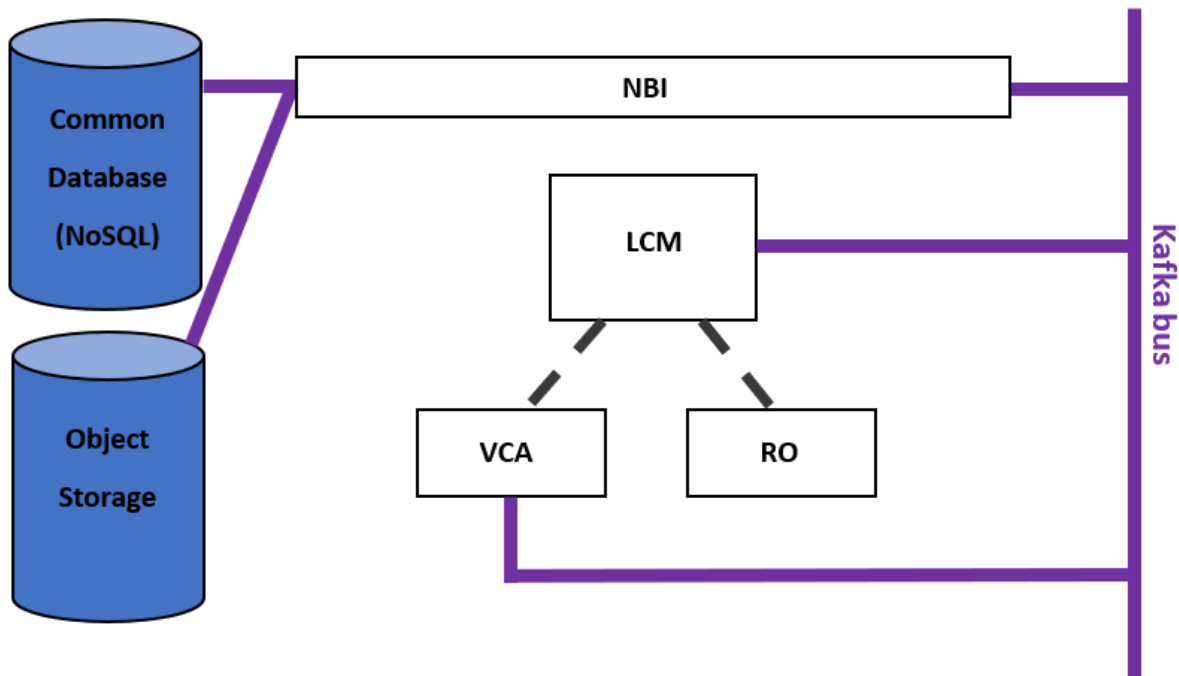


Figure 6 OSM architecture

Hence, a summary of the OSM architecture is that we have a unified Northbound Interface for endpoint calls between the Database and the Kafka bus, which in turn pass the real-time stream of data into the LCM container. VCA container will configure the VNF, and the RO container will deploy it¹¹.

¹¹ <https://www.youtube.com/watch?v=kCFxPV67Adw&t=405s>

5. Technology Enablers

This chapter presents all the technologies that were used in this thesis with a short description. Also noteworthy is the fact that these technologies are Open-Source and available for free, especially if someone wants to use them for academic purposes.

5.1. Open Source MANO

Open Source MANO (OSM) is a production quality MANO for NFV [10], hosted by ETSI open source community, available to everyone for academic and production purposes, VIM independent, capable for the management and orchestration of all VNFs. OSM is aligned to NFV Industry Specification Group (ISG) models, while the providing feedback is based on its implementation experience.

In OSM, Juju Charms can be used for interacting with the Network Functions like configure and monitor. This is achieved with the combination of YAML Ain't Markup Language (YAML) configuration files used to reduce the operations and ease the deployment process for cloud based services. Vendors such as ZTE¹² supports the OSM and other major European network operators have already adopted OSM for their production environment, like Telefonica¹³ and British Telecom¹⁴.

¹² <https://www.zte.com.cn/global/>

¹³ <https://www.telefonica.com/en/home>

¹⁴ <https://www.bt.com/>



Figure 7 Open Source Mano Dashboard.

5.2. Openstack

Openstack¹⁵ is an open source platform, an Infrastructure as a Service (IaaS) framework that uses virtual resources to manage and build public and private clouds. The tools called “Projects” include the Openstack platform, handling the computing for the cloud services of networking, compute, identity, storage, and image services. Openstack can also provide metrics, a RESTful Application programming Interface (API), and many CLI tools for debugging and operational procedures.



Figure 8 Openstack Logo.

¹⁵ <https://www.openstack.org/>

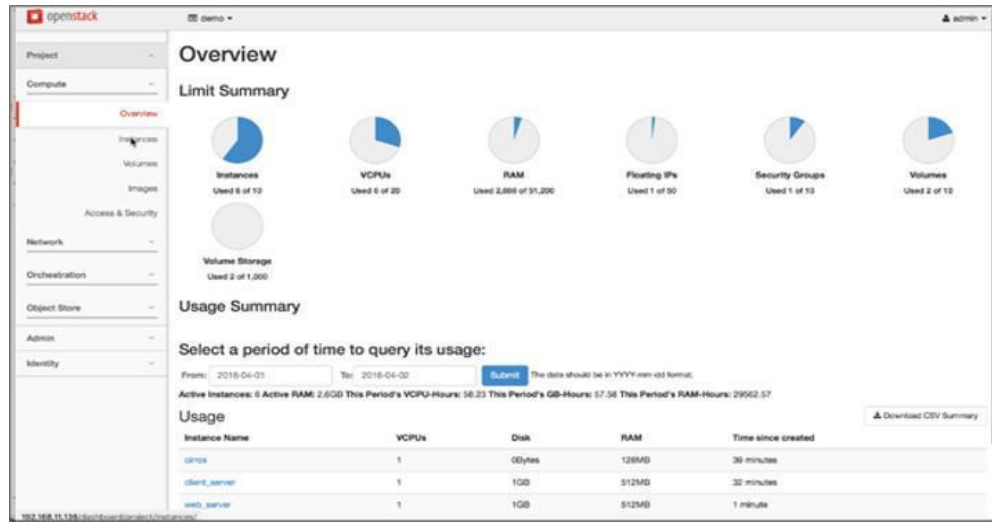


Figure 9 Openstack Dashboard.

5.3. VirtualBox

VirtualBox¹⁶ is a virtualization product for both home and professional use. The user can create virtual machines over almost any operating system. Some actions worth mentioning are the creation of different virtual cards, virtual disks, enabling virtual Dynamic Host Configuration Protocol (DHCP) for dynamic addressing and exporting custom images of the virtual machines.



Figure 10 VirtualBox Logo.

¹⁶ <https://www.virtualbox.org/>

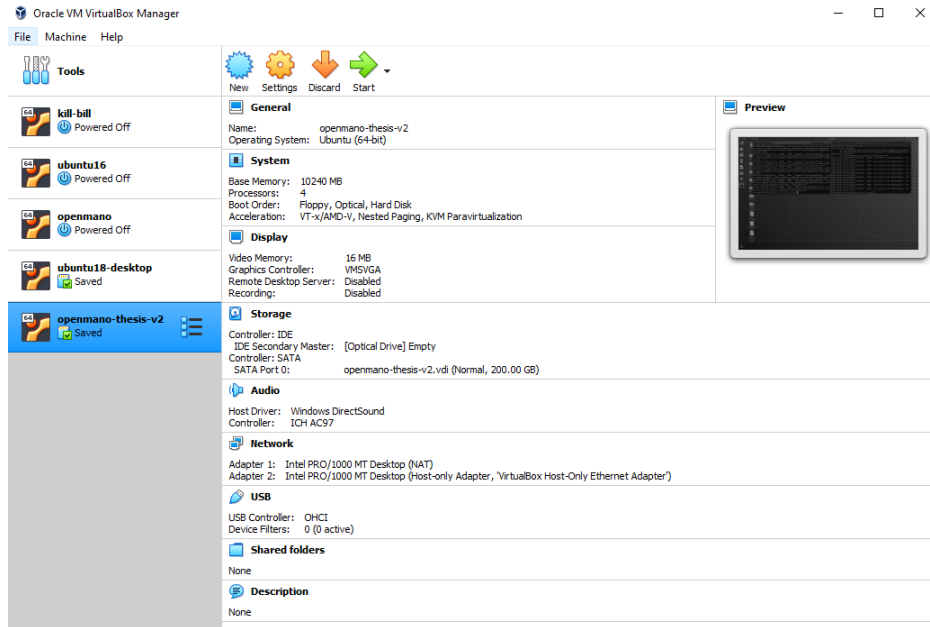


Figure 11 VirtualBox Menu.

5.4. Google Firebase Real-Time Database

Google Firebase¹⁷ is a Backend as a Service (BaaS). An open-source database system which uses real-time processing for many functionalities such as the handle of the continuously changing workloads state. Data is stored in JavaScript Object Notation (JSON) format and that makes it easy to use by developers and users without the problem of synchronization because when the data will change every client that is connected to the database will receive a real-time update of the newest data.



Figure 12 Firebase Logo.

¹⁷ <https://firebase.google.com/>

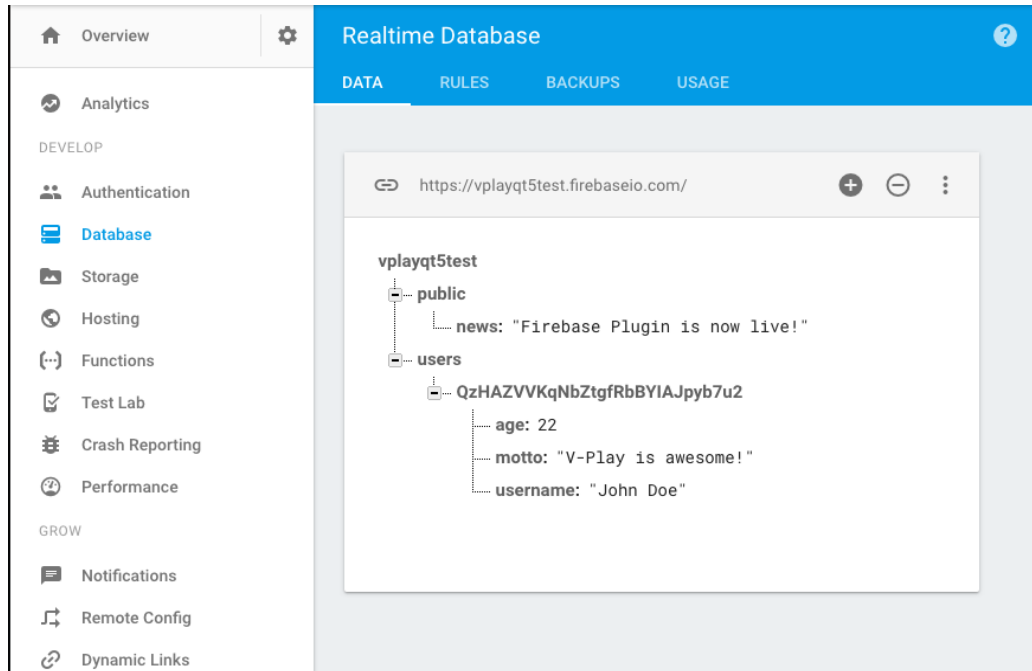


Figure 13 Firebase Dashboard.

5.5. NodeJS

NodeJS¹⁸ is an open-source, back-end, event-driven JavaScript runtime. Its peculiarity is to handle asynchronous data between the web-browser communication with the server, through server-side running scripts that help the user to build scalable network applications with dynamic content changes. Also, the developers can install and run a big variety of modules for their web applications through the Node Packaged Modules (NPM) manager which is an important and functional feature of NodeJS.



Figure 14 NodeJS Logo.

¹⁸ <https://nodejs.org/en/>

5.6. Angular

Angular¹⁹ is an open-source platform and web application framework using HyperText Markup Language (HTML) and Typescript for building single-page client applications, created, and led by the Angular Team at Google²⁰. Angular is written in Typescript and implements optional and core functionality as a part of TypeScript libraries that the users import into their apps.



Figure 15 Angular Logo.

5.7. NG ALAIN

NG-ALAIN²¹ is a front-end framework for creating administration platforms, based on the design principles developed by Ant Design²². There are a big variety of components, templates, and design kits to improve the development experience for both the user and the administrator.



Figure 16 NG-ALAIN Logo.

5.8. Apexcharts

Apexcharts²³ are modern and interactive open-source charts. Is one of the partners of FusionCharts²⁴ and together brings a wide range of data visualization components and the goodness of the open-sources charts.

¹⁹ <https://angular.io/>

²⁰ <https://about.google/>

²¹ <https://ng-alain.com/en>

²² <https://ant.design/>

²³ <https://apexcharts.com/>

²⁴ <https://www.fusioncharts.com/>



Figure 17 Apexcharts Logo.



Figure 18 Dashboard concept.

5.9. Falcon

Falcon²⁵ is a Web Server Gateway Interface (WSGI) library for building in very short time web REpresentational State Transfer (REST) APIs and back-ends for applications in python. The main difference with the building of other APIs is to avoid the unnecessary abstractions and the massive number of dependencies. The Falcon web framework embraces the developers to create a minimal and stable Hypertext Transfer Protocol (HTTP) and REST architectural style.



Figure 19 Falcon Logo.

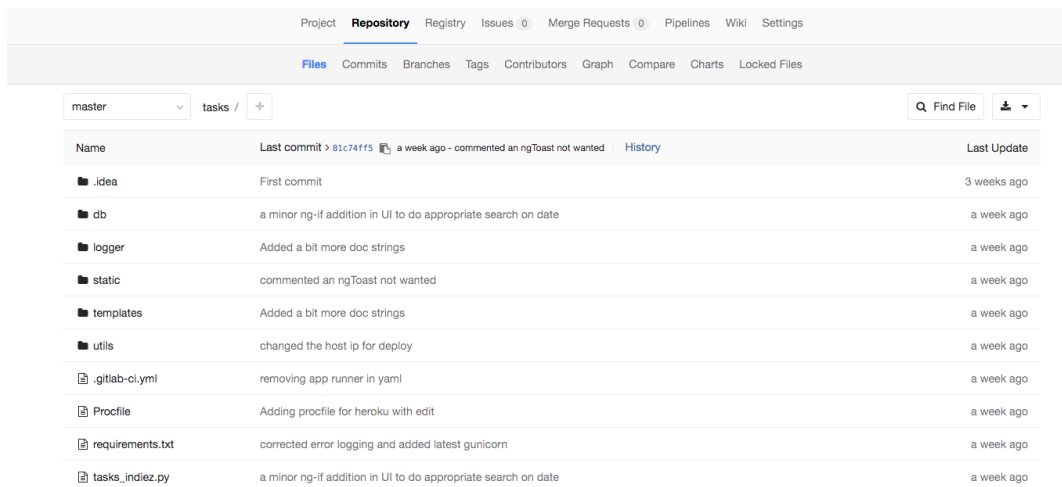
²⁵ <https://falcon.readthedocs.io/en/stable/>

5.10. GitLab

GitLab²⁶ is an open-source DevOps platform, that has: (a) powerful collaboration capabilities like agile planning, version control and code review, (b) Multi-cloud CI/CD that enables you to deploy anywhere, (c) Built-in security and monitoring right out of the box. GitLab is based on Git²⁷ protocol and you can self-host it in a container, in your own local server or on a cloud provider.



Figure 20 GitLab Logo.



Name	Last commit > 81c74ff5 a week ago - commented an ngToast not wanted	History	Last Update
■ .idea	First commit		3 weeks ago
■ db	a minor ng-if addition in UI to do appropriate search on date		a week ago
■ logger	Added a bit more doc strings		a week ago
■ static	commented an ngToast not wanted		a week ago
■ templates	Added a bit more doc strings		a week ago
■ utils	changed the host ip for deploy		a week ago
📄 .gitlab-ci.yml	removing app runner in yml		a week ago
📄 Procfile	Adding procfile for heroku with edit		a week ago
📄 requirements.txt	corrected error logging and added latest gunicorn		a week ago
📄 tasks_indie.py	a minor ng-if addition in UI to do appropriate search on date		a week ago

Figure 21 GitLab Dashboard.

5.11. Cloud-Init

Cloud-Init²⁸ is the industry standard multi-distribution method for the initialization of the instances, based on Python utilities and scripts. It is running during the boot of the VM and will provide and transfer information from VIMs (e.g., Openstack) to the cloud images or the virtual machines. It is formatted as ‘config’ and it uses predefined configurations.

²⁶ <https://about.gitlab.com/>

²⁷ <https://git-scm.com/>

²⁸ <https://cloudinit.readthedocs.io/en/latest/>

6. Architecture and Service Validation Framework

In this section, we will analyze the architecture and the implementation of the framework by providing a fully detailed description for each component.

6.1. Architecture

Figure 18 shows the high-level architecture of the thesis. It consists of many components that work together and use a common database in the cloud to share data between them.

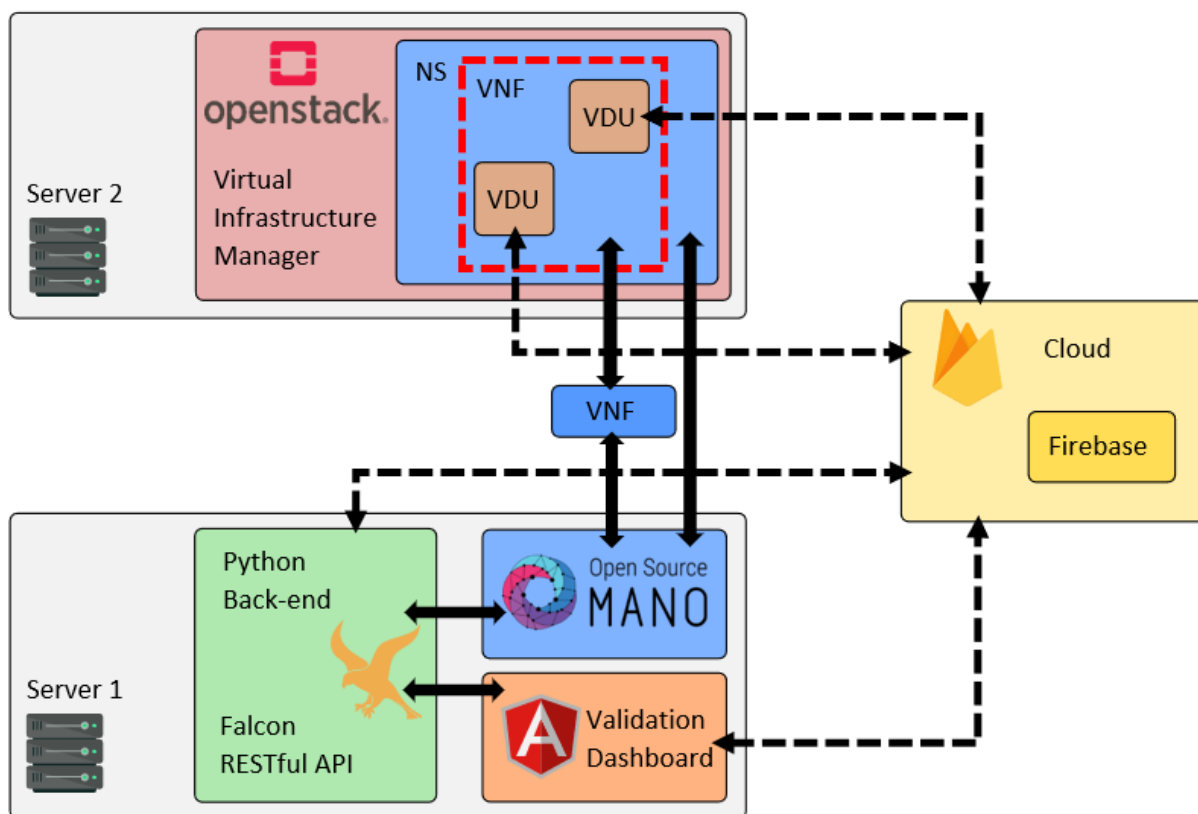


Figure 22 Architecture diagram.

6.1.1. Cloud

The cloud of this architecture is going to be the Google Firebase. Deployed and provided by Google's cloud infrastructure, Google Firebase can handle and store data in real time by multiple tenants connected to it. The data form is in JSON format so it can be easy for a user to

upload and download the data objects. Also, Google firebase can provide multiple subscriptions for the enterprises and the developers based on quota limits per month. All the communication between the validation dashboard, the VDUs and the Open Source Mano will be through Google Firebase.

6.1.2. Openstack as VIM

In a server of our network (server2 of Figure 21), we have installed an Openstack as a hypervisor and Virtual Infrastructure Manager. It will be responsible for the virtualization of the resources and the hosting of the network services from OSM. The Openstack must be visible to the server1 that has installed the OSM so the second one can reach the Openstack through the API endpoints. The 3 things that need to be done so the Openstack can host the Network Services are: (a) the management network (public) has to be connected to the internet so the Open Source MANO can reach it, and the VDUs has access to it, (b) the user has to create a Security Group, so the ingress and egress direction of the data packets be functional for the VDUs, (c) the user has to upload a custom image if it used by the VNF Descriptor package in OSM.

6.1.3. Back-End and Falcon as RESTful API

In server 2 there is the Back-End that is built by Falcon which is a web framework written in Python. Falcon supports some minimal front-end HTTP POST, GET etc. requests such as authentication, push data in firebase, uploads files that are used by VNF Descriptor in OSM. For the python interceptor we use the python 3.6²⁹ that gives us also the possibility to type the commands to the shell. The basic modules that we need to install are the firebase module and the yaml module. The first one will help us to connect with the Firebase and have access to the data and the second one will help us with the transformation of the yaml files to json. Every call that we make to retrieve data from the OSM client³⁰ and the basic format of them is yaml. To make them useful for us and store them in the database we must transform them into JSON objects. The

²⁹ <https://python.readthedocs.io/en/stable/tutorial/interpreter.html?highlight=re>

³⁰ <https://osm.etsi.org/docs/user-guide/10-osm-client-commands-reference.html?highlight=osm%20client>

Back-End is responsible for the smooth use of requests between the Validation Dashboard, the Open Source MANO, and the Firebase. Below we will see the sequence of commands that must be executed to better understand it.

6.1.4. Open Source MANO

As we said earlier Open Source MANO is the orchestrator of the NFV. We installed the stable and the latest version (EIGHT) of the OSM. The Back-End will communicate with the Open MANO for the upload of the VNF/NS packages and for the deploy of the instances. The OSM is responsible for the instantiation and the management of the VNFs and NSs lifecycle. The VCA container handles the VNF Modeling and Configuration through proxy charms, with attributes and primitives, such as to give to a VNF an external connection point etc. And on the other side the RO container is responsible for the deployment and the creation of virtual resources for the VDUs inside the instances, on the top of the VIMs.

6.1.5. Validation Dashboard

The Validation Dashboard is the main component of this thesis in which the users will validate the functionality of their VNFs. It is created by the NG-ALAIN an angular framework which uses the design principles of the Ant-Design. The Validation process of a VNF consists of 3 things:

- The VNFD generator which is a dynamic build form based on the JSON Schema³¹ standard. The user can create automatically a VNFD based on the ETSI standards for the OSM by avoiding making it manually means that it saves time and error avoidance.
- The Validator where the user can upload the VNFD package. The first validation is on the structure of the file as a yaml, and the second one is on the instantiation and the deployment of the VNF in VIM.

³¹ <http://json-schema.org/>

- The Dashboard where the VNF instance can be displayed if it is successfully deployed. The user can see every information of the instance, such as the connection points of the VNF and NS as well as the real time resource's usage of the VDUs.

6.2. Implementation

In this chapter, we will analyze all the components we used and installed for this framework and see the process of the implementation step by step. Furthermore, we will also show a use-case scenario for better understanding the functionality of the Validator.

The server that we have the Openstack VIM has the following specifications:

- 8 CPU Cores.
- 16 GB of RAM.
- 250 GB of Storage.
- 1 Gbps of Internet connection speed.

The Virtual Machine (VM) that we have the Validation Dashboard, the OSM and the Back-end has the following specifications:

- 4 vCPU Cores.
- 12 GB of RAM.
- 180 of Storage.
- 1 Gbps of Internet connection speed.

6.2.1. Openstack Installation

For the Openstack's installation (Devstack³²) we chose the All-in-One installation with a Single Interface³³ and the version of Train. The peculiarity of this installation is that we add the physical interface of the server to the Open vSwitch bridge and with that way the physical interface

³² <https://docs.openstack.org/devstack/latest/>

³³ <https://docs.openstack.org/devstack/rocky/guides/neutron.html>

is capable of transmitting self-service project traffic, management traffic and the Openstack API traffic. That means the management network (public) of Openstack has the same router and gateway as the physical network in which it has been deployed, and it can use the same subnet and DHCP. The only problem here is that if a VM is deployed in the management network there is a big chance to have an Internet Protocol (IP) address collision with the other physical servers in the network. As a solution we must create a configuration file before the installation of the Devstack, to configure the network allocation pool of IPs and choose a range that we will use only for our testing environment. Final step is to declare the Openstack VIM into the Open Source MANO Dashboard.

6.2.2. Openstack Custom Images

Openstack gives you the choice of uploading custom images for the virtual machines. For the need of this thesis, we create a custom image with the help of the Virtual Box that can extract custom images in the QCOW2 format. QCOW2³⁴ is a storage format for virtual disks supported by the QEMU processor emulator, ideal for our virtual machines. The operation system that we use in the custom images is the Ubuntu 16.04 Long Term Support (LTS) and the network interfaces uses DHCP configuration. The specific version of Ubuntu 16.04 is the Server Minimal. This version comes with no pre-installed packages from Ubuntu, except the basics for stable and better functionality. Using a cleaner distribution like this we save a respectable percentage of storage (2~3 gigabytes) so the custom image is as lite as it can be for the VMs. Before the extraction of the image, we also installed:

- **Cloud-init:** is the tool that we use for the configuration of the VM during the boot so we can install the npm packages that we want and run the Unix commands or the python we configure at the start. The configuration of the cloud-init is achieved through the **dpkg-reconfigure** which is an ubuntu tool.
- **NodeJS and NPM Package Manager:** for running scripts with JavaScript format.

³⁴ <https://people.gnome.org/~markmc/qcow-image-format.html>

- **Firestore package and python 3.6:** for running python scripts that will export the VMs characteristics and create a channel of communication between the VM and the Firestore Database.

6.2.3. Automation scripts

During the building of the Validation Dashboard the communication between the platform, the OSM and the Google Firestore was a challenge. As a solution for the creation of the VNFDs, the configuration, the Validation of them, the upload process etc., we create a pool of automated python scripts that are ready to run and serve a smooth communication between all these services and components. The scripts are largely written in Python and few of them in JavaScript and they are divided into x categories: (a) the validation scripts, (b) the initialization scripts, (c) the upload and deployment scripts, (d) the VDUs scripts, (e) the monitoring scripts and (f) the deletion scripts. We are going to give a small description about them:

- Validation scripts:** As we said earlier the users can create custom VNFs and NSs through the VNFD generator. After this procedure, the yaml descriptor file will be created and placed in the Desktop folder “VNFD-Generator” of the server. When the user selects the yaml file before starting the upload, the Validation scripts will be activated and will start running. With the help of the npm module **yaml-validator**³⁵ the script will first check if the file has the property yaml format and it’s not in other forms like JSON, XML etc. After that, based on the type of the file (e.g., VNFD, NSD, init) the script will check if the body of the file follows a specific structure that we have pre-defined based on the ETSI standards. The structure option gives us the choice of the most complex style of checking validity about a descriptor so that we can be sure that the user can run/deploy the network service/function smoothly and without any errors.
- Initialization scripts:** These specific scripts are activated when the user submit the already successfully validated descriptors. Let’s say that they “clean” in a way the Firestore Database and create free spaces to welcome the new entities and objects.
- Upload and Deployment scripts:** Upload scripts are different for each file (VNFD, NSD, init) and they are activated after the user submit the already successfully validated descriptors. It will start with the packages of the descriptors. Open Source MANO needs

³⁵ <https://www.npmjs.com/package/yaml-validator>

the packages to be uploaded in a .tar format so the script will place the yaml files (VNFD & NSD) in separate folders, with the difference that in the VNFD folder it will also put the init file inside. After that it will compress each directory to a tar file by running the command **tar -czvf <file-name>.tar.gz** . Finally, it will make a request through the OSM client and will upload the tar files to the OpenMANO Dashboard and to the Firebase Database as objects. The Deployment script is responsible to create the instance of the NS through the OSM client, to keep the logs of the instantiation and deployment, and upload them in the Firebase Database so the Validation Dashboard can retrieve them and display them to its users. It will also upload the timestamp of the upload event to be calculated later in the dashboard.

- d) **VDUs scripts:** These scripts located in a GitHub³⁶ repository and are activated by the init file that will run when a VDU/VM of the VNF will be deployed. The VDU will download this repository and will start running the script. The first thing that the script will do is to check if there is an internet connection and after that it will start installing the packages it needs(e.g., Firebase module). After that, the script will create an object to upload it later in the Firebase Database that includes some of its attributes like RAM, CPU, Operating System (OS) etc., and another object which will keep it updated every 30secs and concerns the percentage consumed by the RAM and the CPUS.
- e) **Monitoring scripts:** They are located at the server1 and are activated by the users if they choose to monitor the NS and VNF instances in the dashboard. The scripts run OSM client commands to retrieve the data about the instances in yaml format and convert them into json objects to upload them to the Firebase. Note that the scripts will run every 30 min to update the data. The Validation Dashboard using asynchronous data functions which is a characteristic of Typescript, can easily catch every change from the database that concerns these data.
- f) **Deletion scripts:** Finally, the deletion scripts will be activated after the user chooses to delete the Network Service that is running on that time and a sequence of actions will be executed in order of priority. The python script will use the OSM client and with an API call will delete the instance of the NS based on its Id (that we saved it in the server logs), then first it will delete the NS package and last one the VNF package. For the final task,

³⁶ <https://github.com/>

the script will delete all the objects in the Firebase Database associated with the NS and stop all the subscription to them so the Validation Dashboard could return to its primary form.

6.3. Framework Use Case

The Web Browser that we suggest and tested the Validation Dashboard is the Mozilla Firefox³⁷. The user enters the dashboard on port 4200 (e.g., 192.168.56.22:4200/#/login) and the first thing to do is to login(Figure 23) to continue.

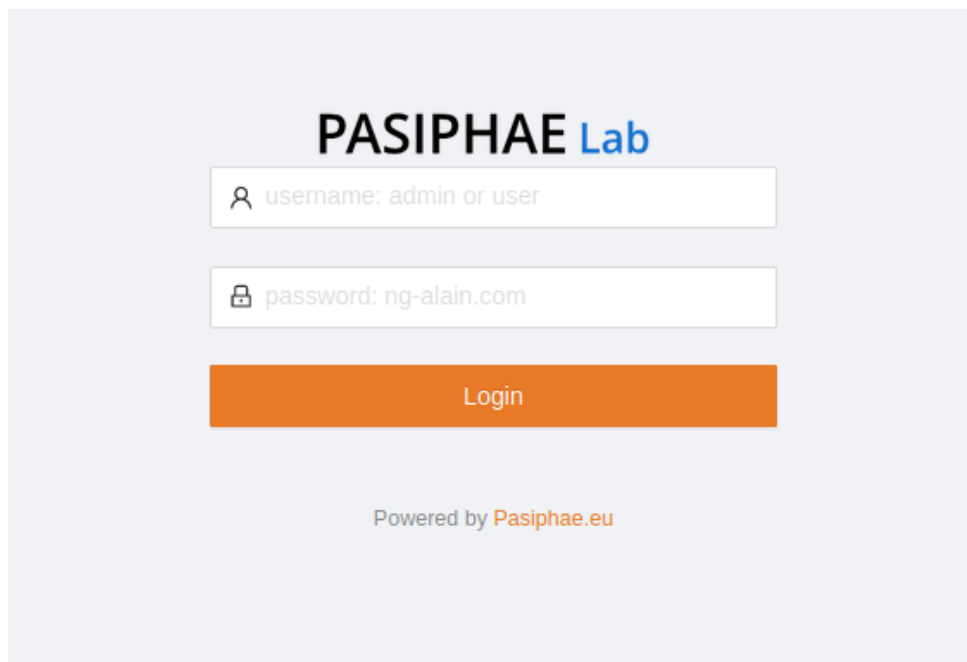


Figure 23 Dashboard Login page

In Figure 24 we can see the full view of Dashboard and observe the menu bar which consists of 4 tabs: (a) Dashboard, (b) Validator, (c) Instances, (d) VNFD Generator and (e) Info. As we can see, dashboard contains 2 charts which displays the percentage usage in RAM and vCPUs of the VDUs, 2 widgets of Openstack and OpenMANO which also displays the percentage usage of specific attributes in each framework, and 1 widget which displays the VDU, VNF, and NS instances.

³⁷ <https://www.mozilla.org/el/firefox/new/>

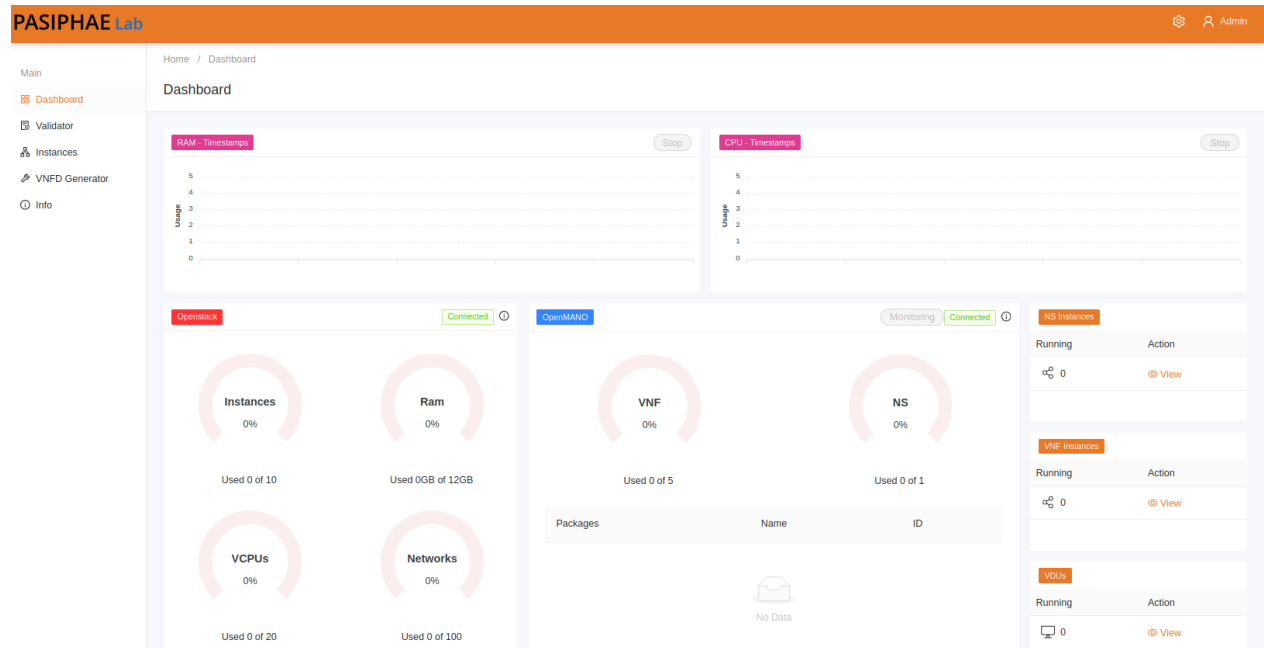


Figure 24 Dashboard

If the users want to avoid creating manually a VNFD because it can be an error-prone and time-consuming task, they must go to the Menu Bar and select the VNFD Generator tab. As we can see in Figure 25 the users can create internal Virtual Links Descriptors and external connection points so they can add below at the creation of the VNFD.

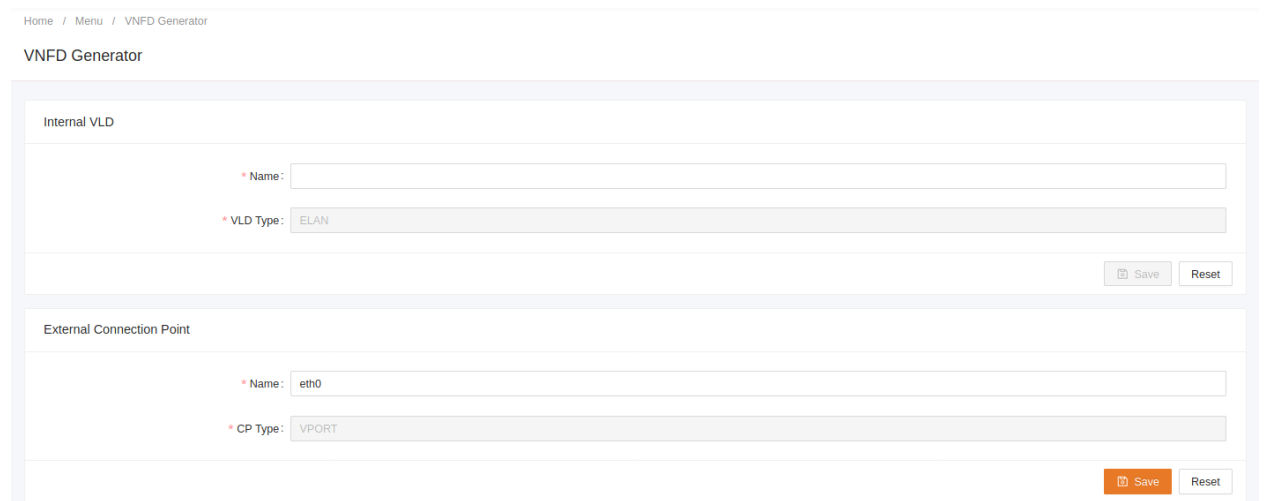


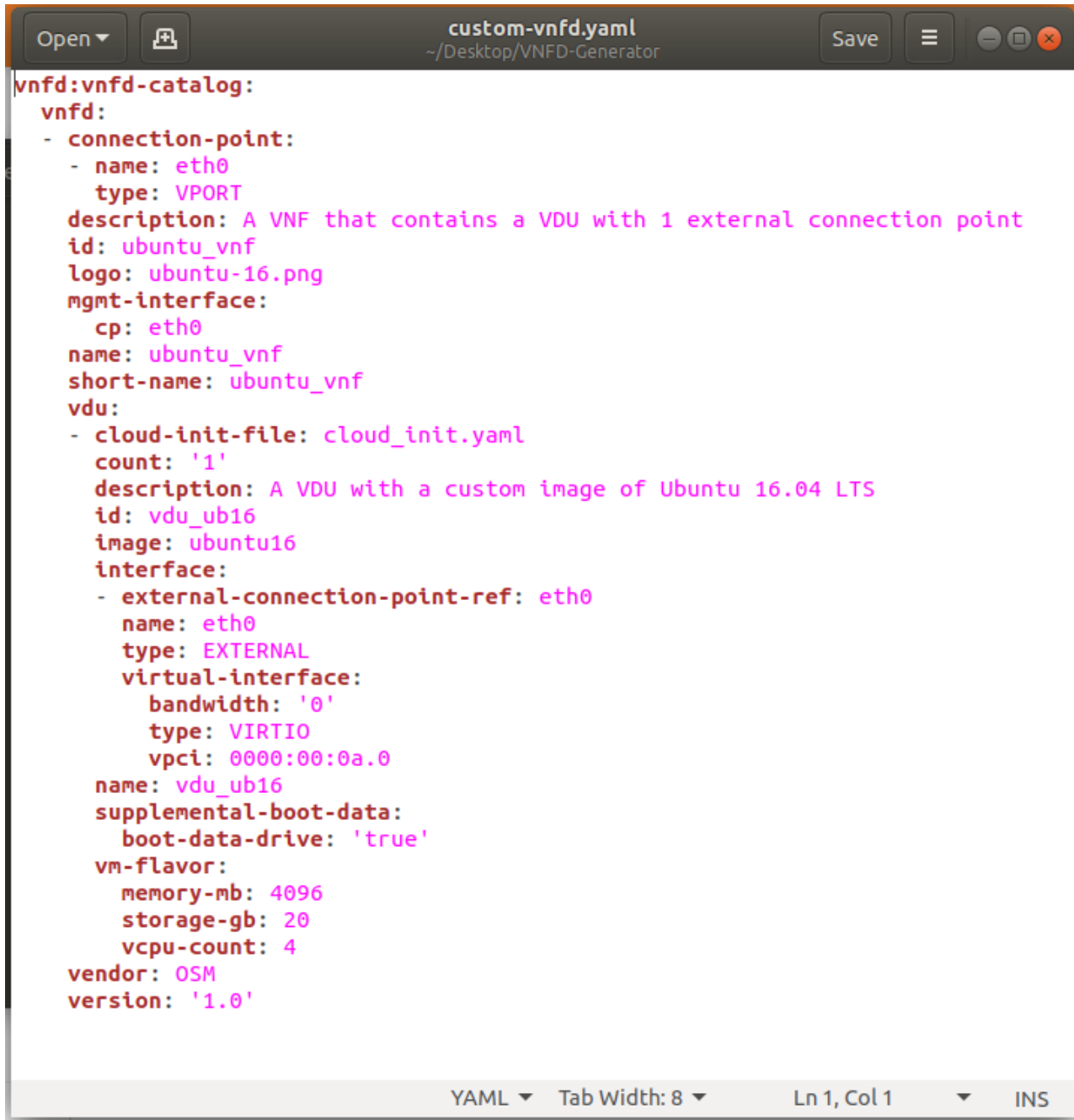
Figure 25 Internal VLD and External Connection Point form fields.

Figure 26 VNFD form field.

In the VNFD form field, a user can give the name and the description of the Descriptor as well as add up to 2 VDUs. For each VDU, the user except the filling of the form field must select at least 1 external point so the VNF can be visible and accessible to the outer network services.(Figure 26). Finally, a NSD must be created in exactly the same way as the VNFD logic. (Figure 27).

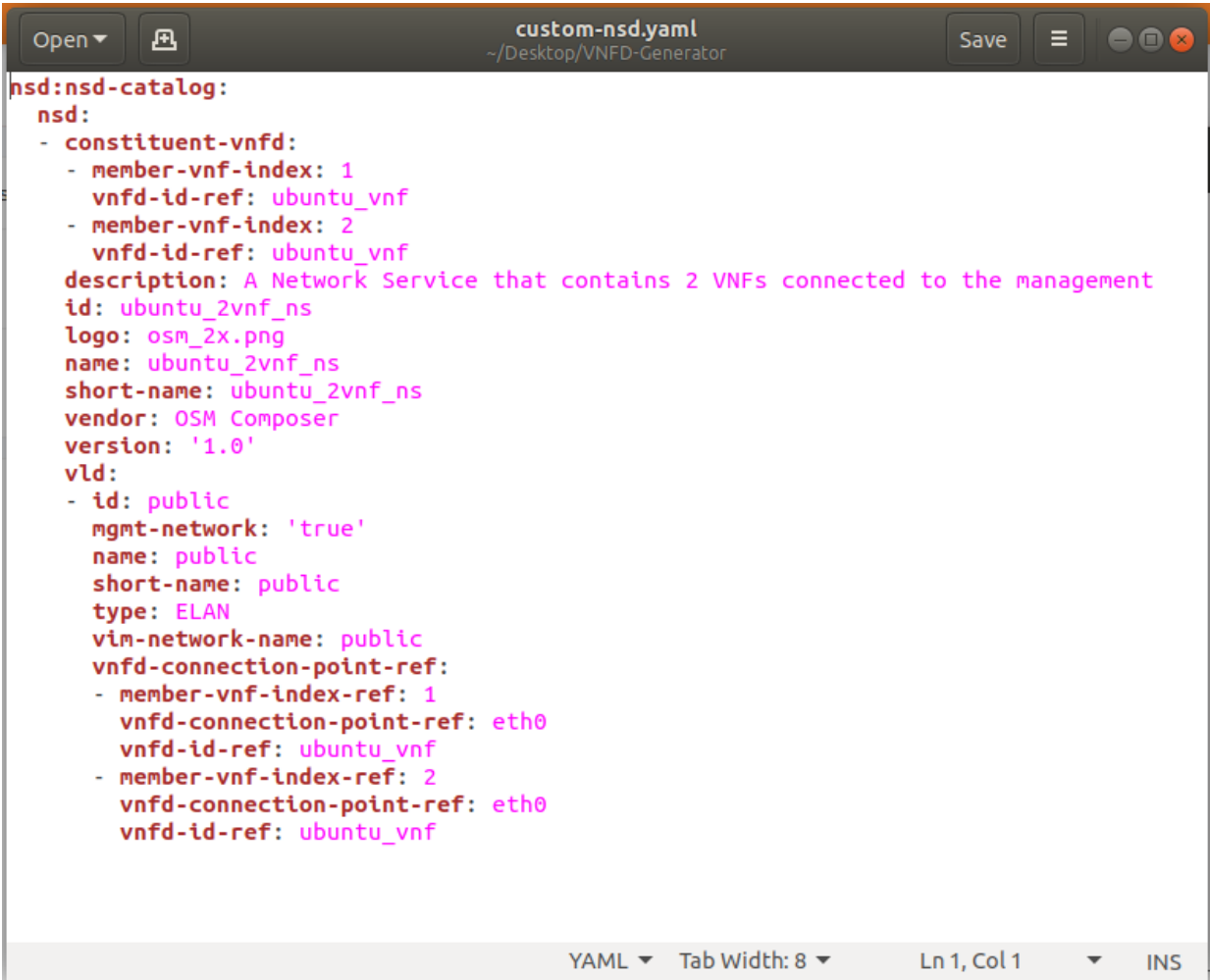
Figure 27 NSD field form.

After the submission of the VNFD and NSD forms, their data objects will be sent to the Back-end and a part of the Validation scripts will transform them from json into a yaml format files and place them in the VNFD-Generator dictionary.

The image shows a text editor window titled 'custom-vnfd.yaml' with the path '~/Desktop/VNFD-Generator'. The editor contains a YAML configuration for a VNF descriptor. The configuration is structured as follows: 'vnfd:vnfd-catalog:' followed by 'vnfd:' and a list of VNFs. The first VNF is 'ubuntu_vnf' with a description 'A VNF that contains a VDU with 1 external connection point'. It has a logo 'ubuntu-16.png', a management interface 'eth0', and a VDU 'vdu_ub16'. The VDU is described as 'A VDU with a custom image of Ubuntu 16.04 LTS' and has a count of '1'. It features an external connection point 'eth0' with a virtual interface 'vdu_ub16' that has a bandwidth of '0', type 'VIRTIO', and vpci '0000:00:0a.0'. The VDU also has supplemental boot data with 'boot-data-drive: true', a memory of 4096 MB, 20 GB storage, and 4 vCPUs. The VNF is from vendor 'OSM' and version '1.0'.

```
vnfd:vnfd-catalog:
  vnfd:
  - connection-point:
    - name: eth0
      type: VPORT
    description: A VNF that contains a VDU with 1 external connection point
    id: ubuntu_vnf
    logo: ubuntu-16.png
    mgmt-interface:
      cp: eth0
    name: ubuntu_vnf
    short-name: ubuntu_vnf
    vdu:
    - cloud-init-file: cloud_init.yaml
      count: '1'
      description: A VDU with a custom image of Ubuntu 16.04 LTS
      id: vdu_ub16
      image: ubuntu16
      interface:
      - external-connection-point-ref: eth0
        name: eth0
        type: EXTERNAL
        virtual-interface:
          bandwidth: '0'
          type: VIRTIO
          vpci: 0000:00:0a.0
        name: vdu_ub16
      supplemental-boot-data:
        boot-data-drive: 'true'
      vm-flavor:
        memory-mb: 4096
        storage-gb: 20
        vcpu-count: 4
    vendor: OSM
    version: '1.0'
```

Figure 28 VNF Descriptor file.



```
nsd:nsd-catalog:
  nsd:
  - constituent-vnfd:
    - member-vnf-index: 1
      vnfd-id-ref: ubuntu_vnf
    - member-vnf-index: 2
      vnfd-id-ref: ubuntu_vnf
    description: A Network Service that contains 2 VNFs connected to the management
    id: ubuntu_2vnf_ns
    logo: osm_2x.png
    name: ubuntu_2vnf_ns
    short-name: ubuntu_2vnf_ns
    vendor: OSM Composer
    version: '1.0'
  vld:
  - id: public
    mgmt-network: 'true'
    name: public
    short-name: public
    type: ELAN
    vim-network-name: public
    vnfd-connection-point-ref:
    - member-vnf-index-ref: 1
      vnfd-connection-point-ref: eth0
      vnfd-id-ref: ubuntu_vnf
    - member-vnf-index-ref: 2
      vnfd-connection-point-ref: eth0
      vnfd-id-ref: ubuntu_vnf
```

Figure 29 NS Descriptor file.

From the files an experienced user and accustomed to formatting VNF descriptor files can easily understand that the file describes a network service that includes 1 public network that has connected to it (from 2 external points), 2 VNFs with 1 VDU each. Then, the user should go to the Validator tab to check and upload the files that have been created. Each time the user uploads a file, a validation script runs separately for the specific file to check its validity in terms of its yaml type and ETSI standards. The results from the validation process can be seen in the YAML file Validation component. (Figure 30).

YAML files Validation

VNF	valid	Init	valid	NS	valid
Total of 0 structure validation error(s)		Total of 0 structure validation error(s)		Total of 0 structure validation error(s)	

VNFD:

You can Drag and Drop a file
Upload the VNF descriptor (yaml format)

ubuntu_vnf.yaml

Init File:

You can Drag and Drop a file
Upload the Init File descriptor (yaml format)

cloud_init.yaml

NSD:

You can Drag and Drop a file
Upload the NS descriptor (yaml format)

ubuntu_2vmf_ns.yaml

Figure 30 Upload and YAML validation process.

The upload and deployment status can be seen in the Deployment Stages component right after the submission of the form field (Figure 31). At that time, the Deployment script activated and started to communicate with the NBI channel of the OSM. It is responsible to deploy the network service to the OSM orchestrator as well as display the logs of this deployment such as the successful messages and the error warnings (Figure 32). In case of an error or a warning the user can press the “Abort Installation” button to cancel the process.

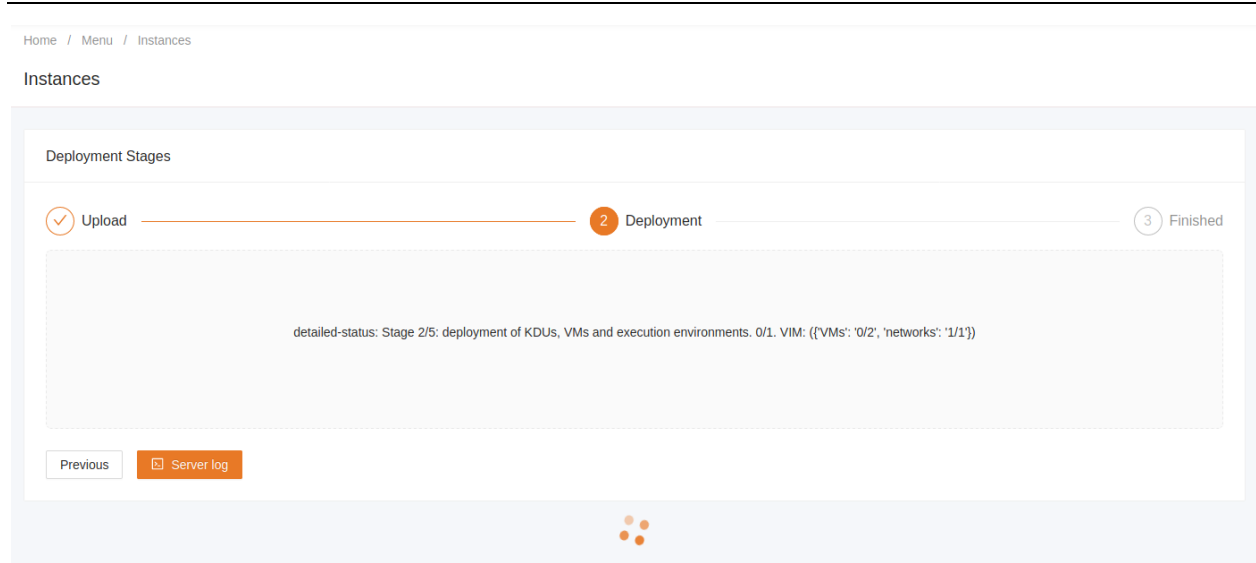


Figure 31 Deployment Stage component.

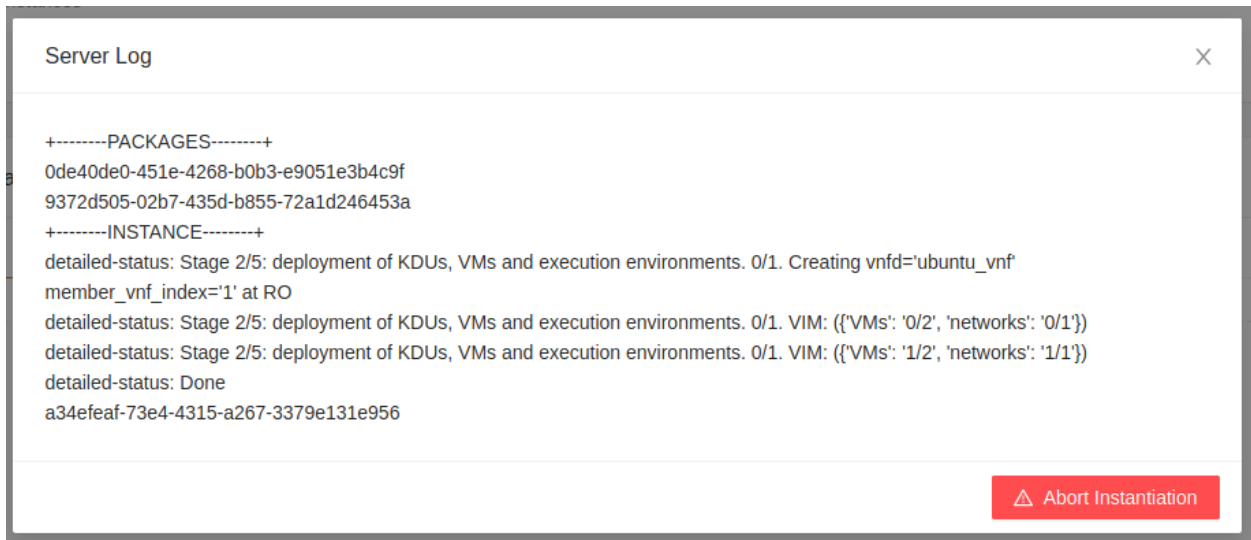


Figure 32 Server Log modal.

If the deployment is successful, the VDUs will appear both in Validation Dashboard (Figure 33 & Figure 35) and Openstack Network Topology (Figure 34).

VDUs				
IP address	DNS	Operating System	Status	Action
192.168.200.243	validator-ns-1-vdu-ubuntu16-1	Linux-4.4.0-142-generic-x86_64-with-Ubuntu-16.04-xenial	connected	View
192.168.200.247	validator-ns-2-vdu-ubuntu16-1	Linux-4.4.0-142-generic-x86_64-with-Ubuntu-16.04-xenial	connected	View

Figure 33 VDUs table.

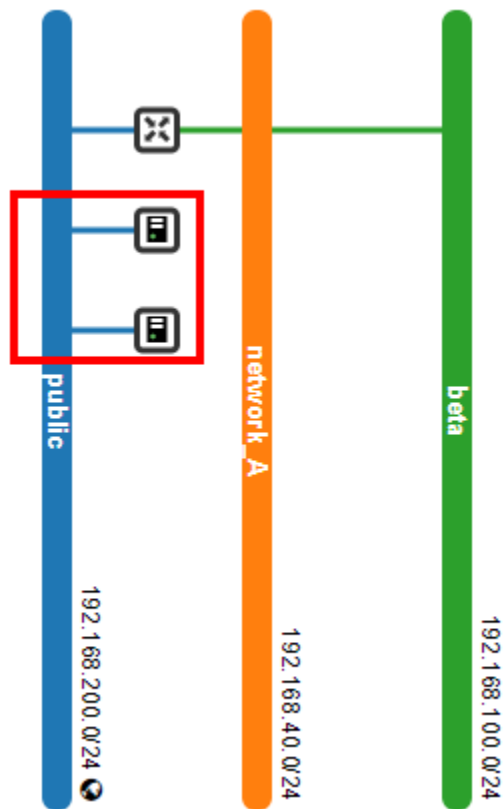


Figure 34 Openstack Network Topology.

The data streams in charts (Figure 35) which depict the usage percentage of RAM and vCPU of the VDUs will always be near to 0% because the VDUs are not running a script or an application at that moment, so there are no concerns about the functionality and confidential information of the chart. In Figure 36 we can observe the number of activated instances that are currently active on the OSM.

Dashboard

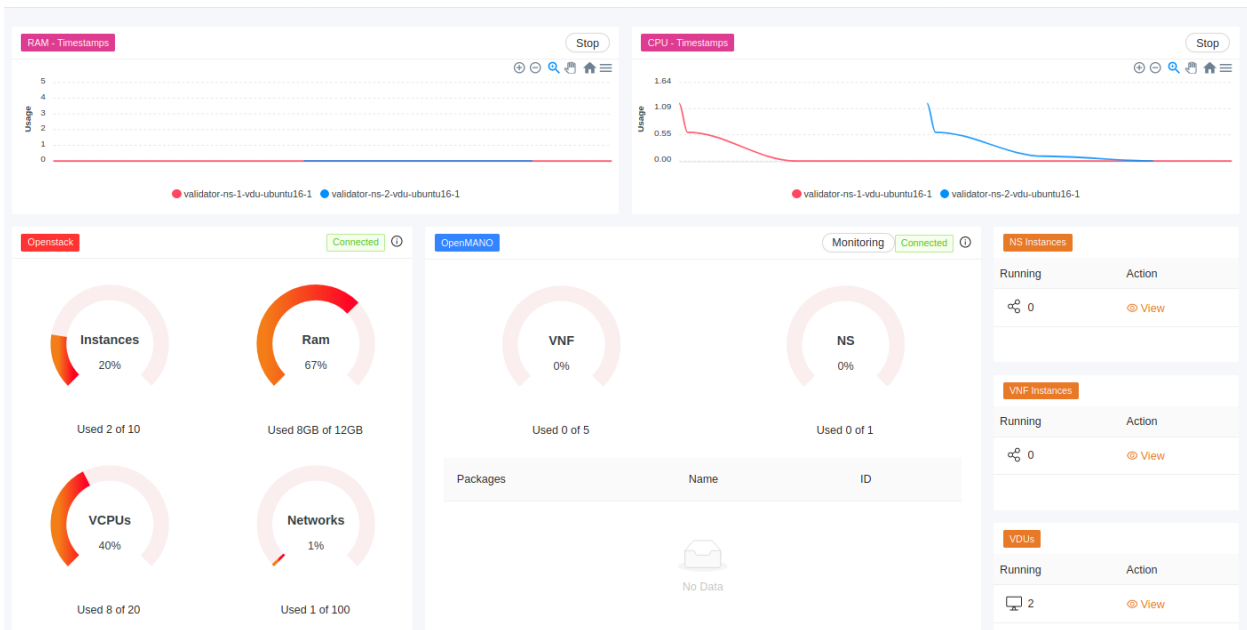


Figure 35 Dashboard after the NS deployment.

Home / Dashboard

Dashboard

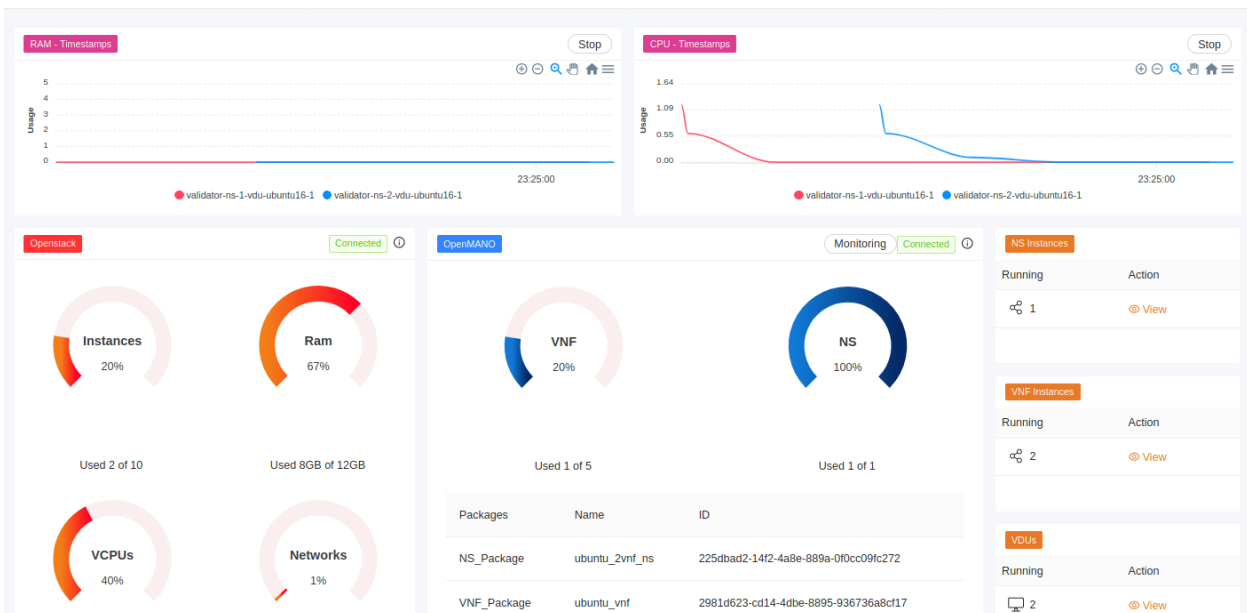


Figure 36 Dashboard during the Monitoring.

By pressing on the “View” button of any Instance, the dashboard will navigate the user to the Instances Tab (Figure 37). In this page there are all the NS (Figure 38 & Figure 39), VNF (Figure 40), and VDU (Figure 41) Instances that we previously defined in the description files and we can check their attributes of each one separately.

NS Instances				
Name	ID	NSD	Status	Action
validator-ns	e17b6be6-4d29-42c8-a3f8-84fe6eb451a0	ubuntu_2vnf_ns	READY	View

VNF Instances				
ID	VNFD	ID	NS Reference	Action
de677dfa-5c97-4242-81d6-06a5850bd3f8	ubuntu_vnf	192.168.200.243	e17b6be6-4d29-42c8-a3f8-84fe6eb451a0	View
85a30ac7-6cc3-464e-8ca3-8b3f07f4c554	ubuntu_vnf	192.168.200.247	e17b6be6-4d29-42c8-a3f8-84fe6eb451a0	View

VDUs				
IP address	DNS	Operating System	Status	Action
192.168.200.243	validator-ns-1-udu-ubuntu16-1	Linux-4.4.0-142-generic-x86_64-with-Ubuntu-16.04-xenial	connected	View
192.168.200.247	validator-ns-2-udu-ubuntu16-1	Linux-4.4.0-142-generic-x86_64-with-Ubuntu-16.04-xenial	connected	View

Figure 37 Instances Table.

NS Instance	
Attributes	Value
Admin Status	ENABLED
VNF reference	de677dfa-5c97-4242-81d6-06a5850bd3f8,85a30ac7-6cc3-464e-8ca3-8b3f07f4c554
Current Operation	IDLE
Datacenter	8769e3a0-76f4-448e-8d1a-b98a6ce53bd5
Description	default description
ID	e17b6be6-4d29-42c8-a3f8-84fe6eb451a0
Name	validator-ns
NS State	READY
NSD Name	ubuntu_2vnf_ns
NSD ID	225dbad2-14f2-4a8e-889a-0f0cc09fc272
Resource Orchestrator	osmopenmano

VLD	
Attributes	Value
ID	public
mgmt-network	true
Name	public
Type	ELAN

Figure 38 NS Instance (a)

VLD	
Attributes	Value
ID	public
mgmt-network	true
Name	public
Type	ELAN

VNF Connection points	
Attributes	Value
# VNF Index	1
Connection Point	eth0
VNFD ID	ubuntu_vnf

VNF Connection points	
Attributes	Value
# VNF Index	2
Connection Point	eth0
VNFD ID	ubuntu_vnf

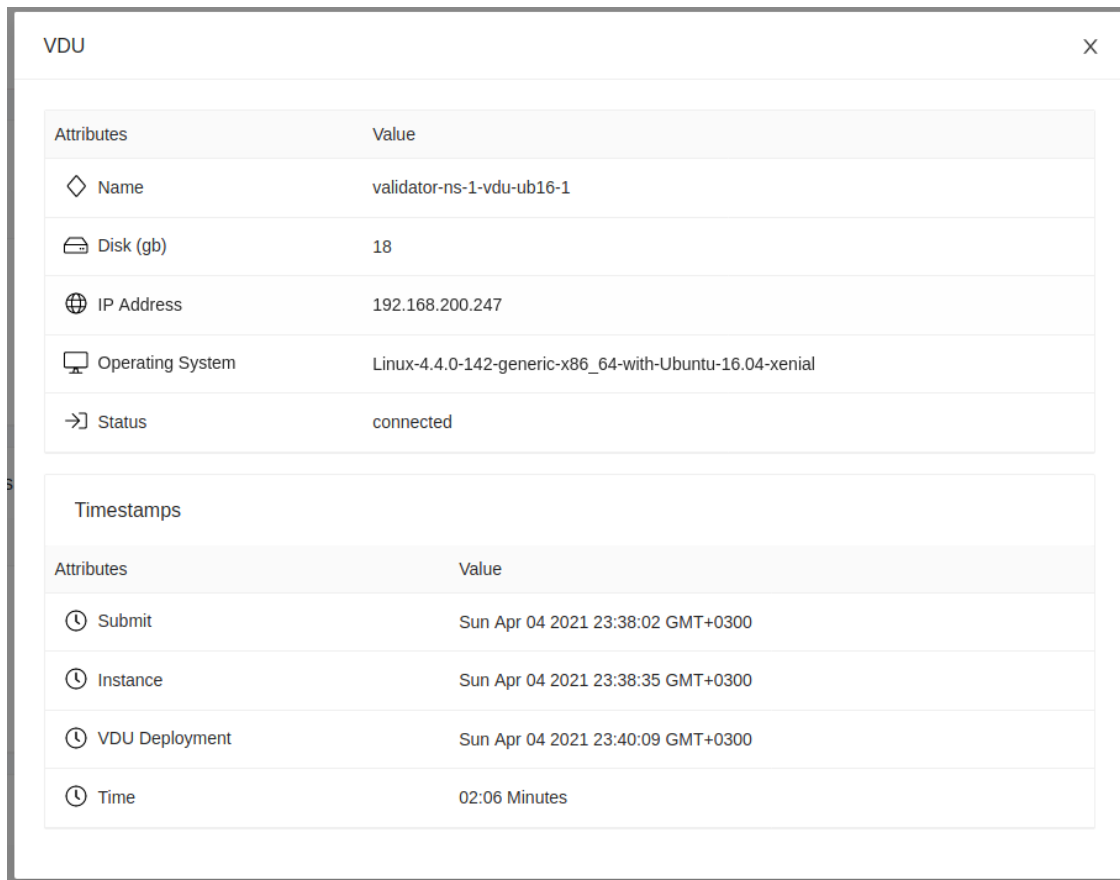
Figure 39 NS Instance (b)

Attributes		Value
ID		85a30ac7-6cc3-464e-8ca3-8b3f074c554
# VNF Index		2
NS Reference		e17b6be6-4d29-42c8-a3f8-84fe6eb451a0
VNFD ID		2981d623-cd14-4d8e-8895-936736a8cf17
VNFD Reference		ubuntu_vnf

VDU	
Attributes	Value
Name	validator-ns-2-vdu_ubuntu16-1
Status	ACTIVE

Interface	
Attributes	Value
VDU Name	validator-ns-2-vdu_ubuntu16-1
External connection point	eth0
IP Address	192.168.200.247
MAC Address	fa:16:3e:5d:55:dc
mgmt-vnf	true
NS VLD	public

Figure 40 VNF Instance



The screenshot shows a window titled "VDU" with a close button (X) in the top right corner. It contains two tables. The first table, titled "Attributes", lists various configuration parameters and their values. The second table, titled "Timestamps", lists key events in the instance's lifecycle along with their corresponding times.

Attributes	Value
Name	validator-ns-1-vdu-ub16-1
Disk (gb)	18
IP Address	192.168.200.247
Operating System	Linux-4.4.0-142-generic-x86_64-with-Ubuntu-16.04-xenial
Status	connected

Attributes	Value
Submit	Sun Apr 04 2021 23:38:02 GMT+0300
Instance	Sun Apr 04 2021 23:38:35 GMT+0300
VDU Deployment	Sun Apr 04 2021 23:40:09 GMT+0300
Time	02:06 Minutes

Figure 41 VDU Instance

6.3.1. Failure examples

In case the users will not use our VNFD Generator, we made some tests to check if our validation layers would recognize the errors, whether these are syntax errors or functional errors. The first test was created by putting spaces, words, or commas at random points in the yaml format of the VNFD to check the format validity and the second test was to remove an attribute from the NSD to check the structure validity. In Figure 42 and Figure 43 we can see that in both cases the Validation scripts/process worked perfectly, and they recognized the errors.

YAML files Validation

VNF	Init	NS
Invalid	Valid	Invalid
<pre>Failed to load the Yaml file "/home/tasos/uploads/vnfd/ubuntu_vnf/ubuntu_vnf.yaml:unknown" can not read a block mapping entry; a multiline key may not be an implicit key (14:8) 11 name: ubuntu_vnf 12 short-name: ubuntu_vnf 13 dsf 14 vdu:^ 15 - cloud-init-file: cloud_init.yaml 16 count: '1' Yaml format related errors in 1 files Total of 0 structure validation error(s)</pre>	<pre>Total of 0 structure validation error(s)</pre>	<pre>/home/tasos/uploads/nsd/ubuntu_2vnf_ns/ubuntu_2vnf_ns.yaml is not following the correct structure, missing: nsd:nsd-catalog.nsd[0].name Yaml format related errors in 1 files Total of 1 structure validation error(s)</pre>

Figure 42 Failure Example 1.

YAML files Validation

VNF **Invalid**

```
/home/tasos/uploads/vnfd/ubuntu_vnf/ubuntu_vnf.yaml is not following the correct structure, missing:
vnfd:vnfd-catalog.vnfd[0].name
Yaml format related errors in 1 files
Total of 1 structure validation error(s)
```

Figure 43 Failure Example 2.

To test the Deployment scripts as well, we created yaml valid files that can pass the first layer of validation, but they have wrong values in their descriptor attributes. Such as wrong implementation of init file that does not exist (Figure 44 & Figure 45) or a badly written VIM attribute (Figure 46 & Figure 47).

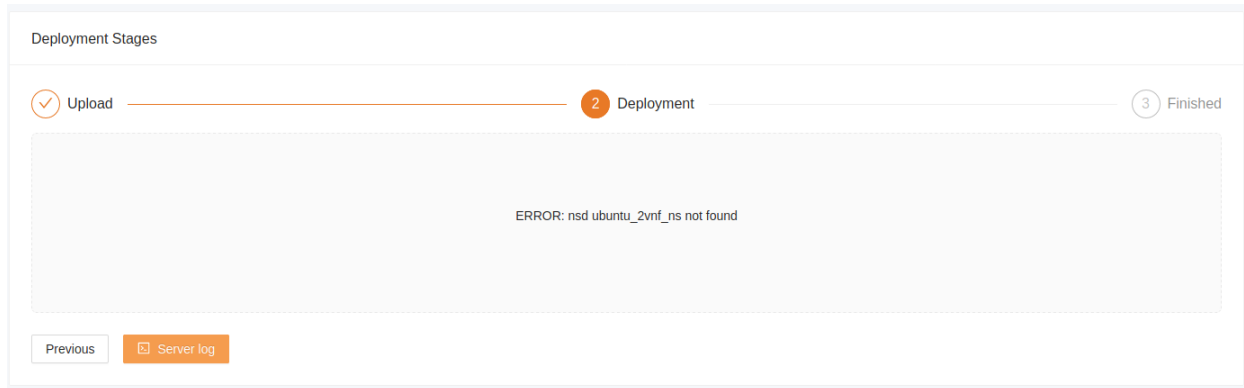


Figure 44 Deployment Stage error 1.

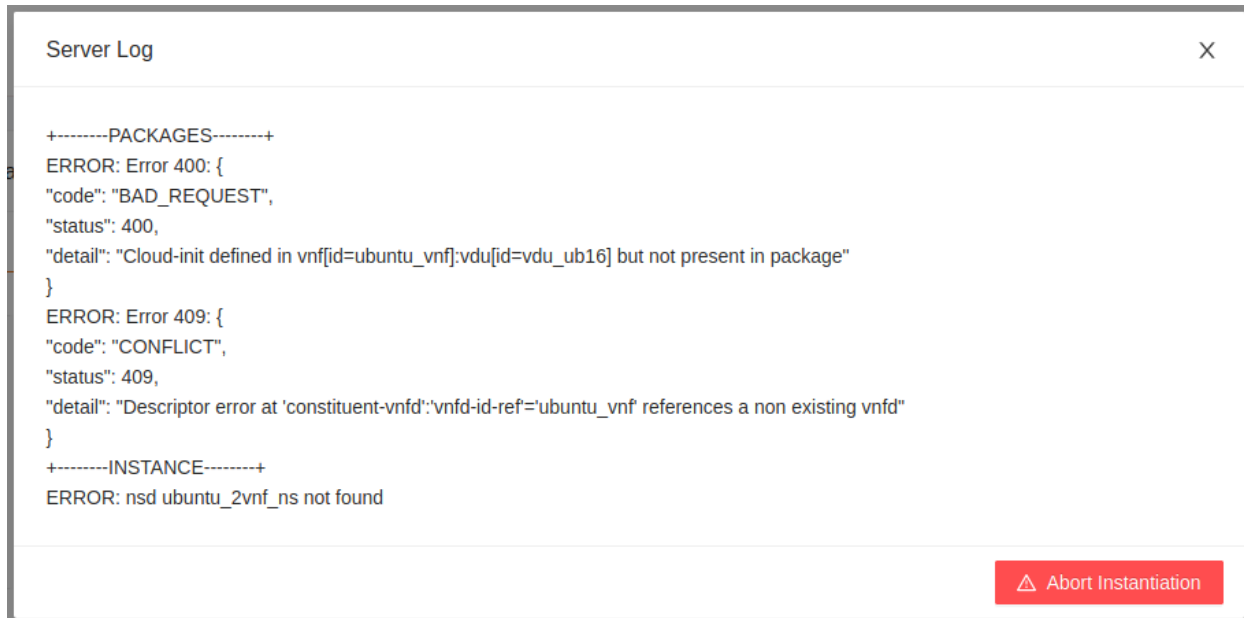


Figure 45 Server Log of error 1.

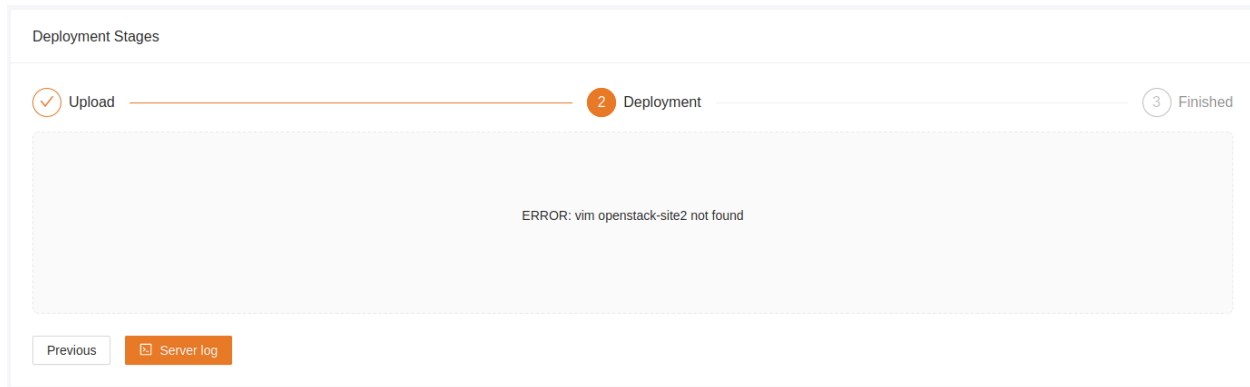


Figure 46 Deployment Stages error 2.

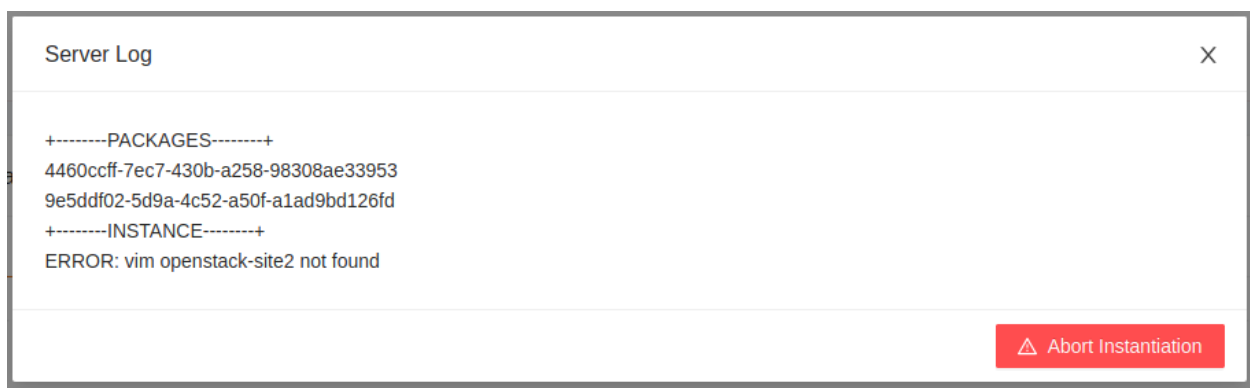


Figure 47 Server Log of error 2.

7. Evaluation

The assessment of the Validation Dashboard was done using 6 test cases with similar init-file but using different scenarios for VNF/NS descriptors each time. For each scenario we changed the number of: (a) VDUs, (b) the external connection points, (c) the internal connection points, (d) the internal VLDs, (e) and (f) the VLDs for the NS. Every NSD and VNFD that was created from our Generator came out with no syntax errors and passed the first validation layer with a high success rate. All the deployments were successful and valid, and we could confirm it from the display of instances in the Validation Dashboard. That makes our dashboard a reliable framework for someone who wants to create a VNF for the Open Source MANO and check its validity at the same time. Below we will give a little bit of information about our use cases and a small description for each one of them.

7.1. Test Case 1

In this case we created a VNF descriptor that contains 1 VDU with a custom image of Ubuntu 16.04 LTS and 1 external connection point (eth0). Then we added it twice in the NS descriptor and we connected each Connection Point (CP) with the management (public) network of the VIM. The VNF and NS descriptors passed the Validation layers successfully and the VDUs took 2:06 minutes to boot up (Figure 48).

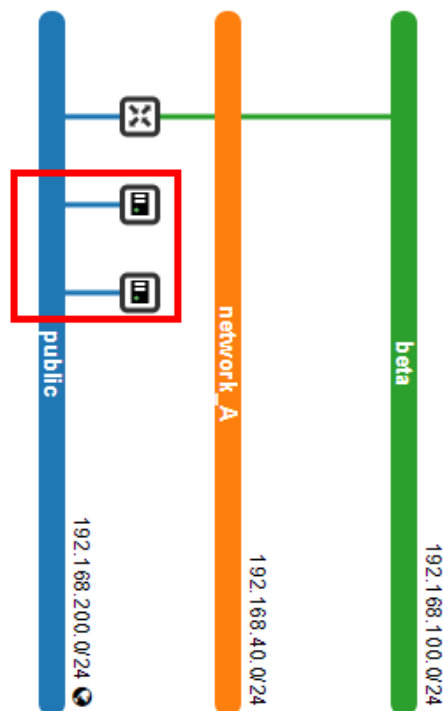


Figure 48 Test Case 1.

7.2. Test Case 2

In this case we created a VNF same as the one in Test Case 1 with the difference that we put into the VDU 2 external connection points (eth0, eth1). Then we added it in the NS descriptor, and we connected one of its CPs in the management network (public) and one with another network (network_A) which is not connected to the internet. The result was that the descriptors pass the Validation layers but the VDU script returns info (data) only for the port that is connected to the internet (that means the CP that is connected to the public network and not the network_A). We

manage to take these data (from the network_A) through the monitoring script which displays all the interface data from the VNF (Figure 50). The VDU boot time was 1:42 minutes (Figure 49).

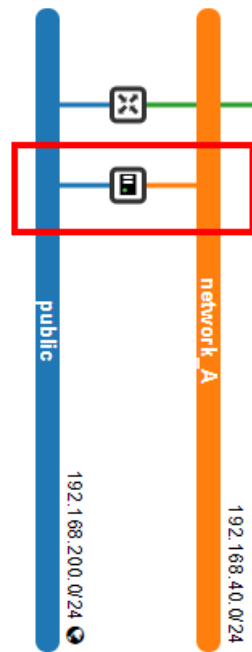


Figure 49 Test Case 2.

Interface	
Attributes	Value
VDU Name	validator-ns-1-udu_ubuntu16-1
External connection point	eth0
IP Address	192.168.200.245
MAC Address	fa:16:3e:02:ee:38
mgmt-vnf	true
NS VLD	public

Interface	
Attributes	Value
VDU Name	validator-ns-1-udu_ubuntu16-1
External connection point	eth1
IP Address	192.168.40.90
MAC Address	fa:16:3e:7e:b2:e0
NS VLD	network_A

Figure 50 Test Case 2 VNF Instance modal.

7.3. Test Case 3

In this case we created a VNF exactly the same as the one in the Test Case 1 with only one CP. We added it in a NS descriptor, and we connected its CP to an existing network (beta) that is already connected to the management network (public) with a router, and that means that has internet access. The Validation and the deployment process was a success, and the VDU took 1:34 minutes to boot up (Figure 51).

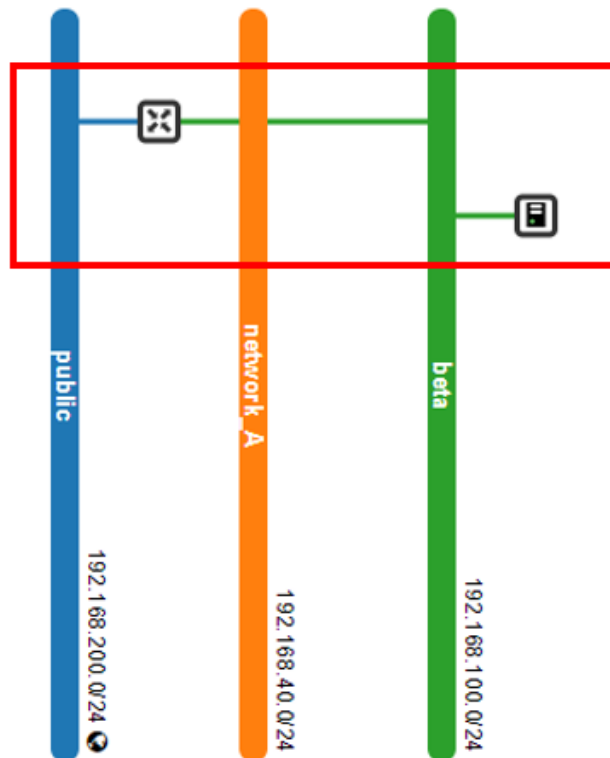


Figure 51 Test Case 3.

7.4. Test Case 4

In this case we created a VNF that contains 1 VDU with 1 CP and inside the cloud init file (Figure 53) we put a command to install an Apache³⁸ server. The CP was connected to the management network (public) based on the NSD. The deployment (Figure 54) and the Validation was a success again, as well as the installation of the Apache (Figure 55) by the cloud init file. The boot up time of the VDU comes to 1:50 minutes (Figure 52).

³⁸ <https://httpd.apache.org/>

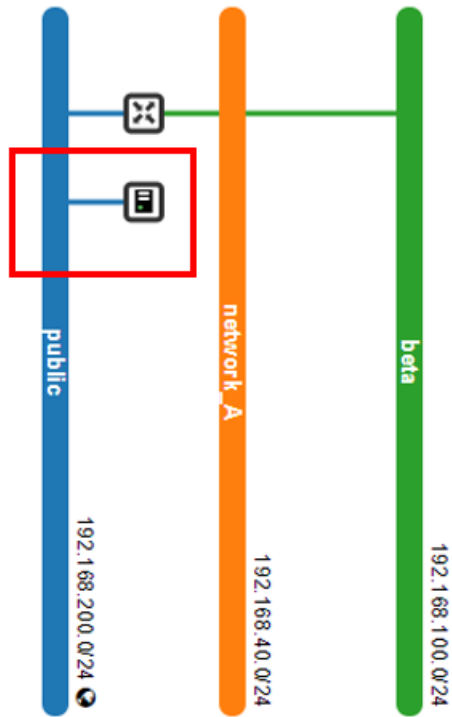


Figure 52 Test case 4.

```
#cloud-config
preserve_hostname: true
manage_etc_hosts: false

package_update: true

packages:
- python-pip
- git
- apache2
```

Figure 53 Test Case 4 cloud init file.

VDUs	
IP address	DNS
192.168.200.247	validator-ns-1-vdu-ubuntu16-1

Figure 54 Test Case 4 VDUs Table.

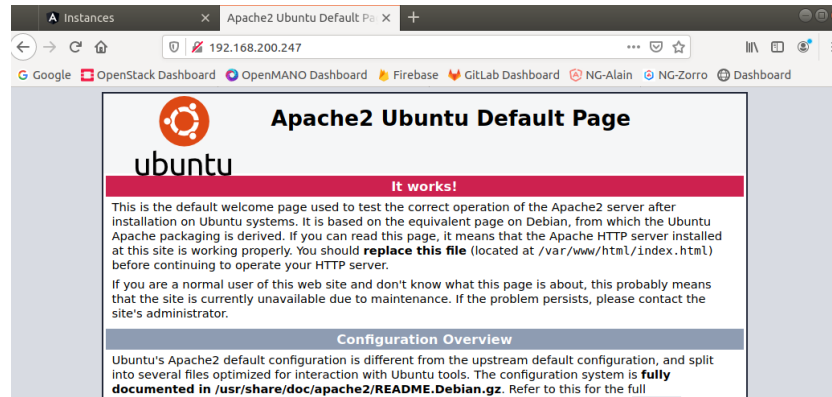


Figure 55 Test Case 4 Apache server running.

7.5. Test Case 5

In this case we created a VNF that contains 2 VDUs with the first VDU having 2 CPs, 1 external and 1 internal, and the second VDU with 1 internal CP only. We also created an internal VL (custom), and we connected the 2 internal CPs into it. Finally, we added the VNF in a NS descriptor and we connected the only external CP with the management network (public). The VNF successfully passed the Validation but only the VDU with the port that is connected to the public network sent data to the Firebase. The VDUs booted up in 2:40 minutes (Figure 56).

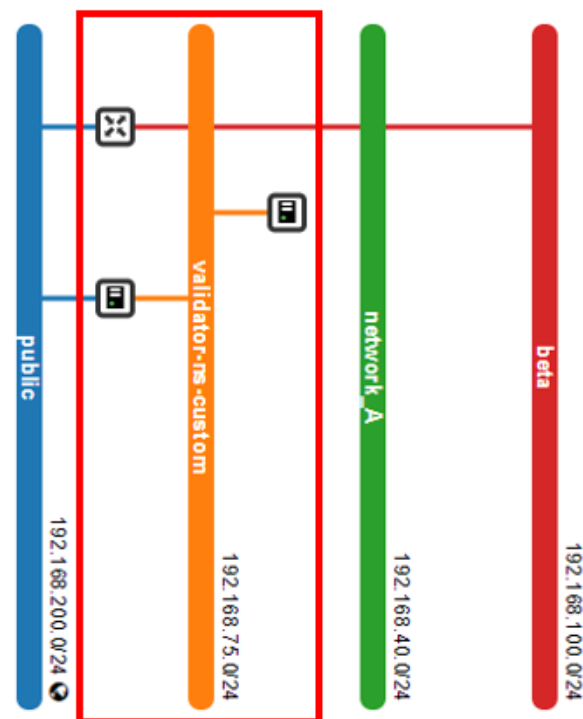


Figure 56 Test Case 5.

7.6. Test Case 6

In the final case we created a VNF similar with the one in the Test Case 1. We added it in a NS descriptor, and we linked its external CP with a network that is not connected to the management network (public) and it is isolated. The VDU never returned anything because of the internet connection loss, so the instance table, which was waiting data, kept loading indefinitely in an infinite loop (Figure 57). The solution is the creation of a Physical Network Unit (PDU). A PDU Descriptor (PDU) can contain a description of a router with the network interfaces that needs to link in it. But the PDU instance has to be deployed before the VNF and NS so it can be visible first to the VIM.

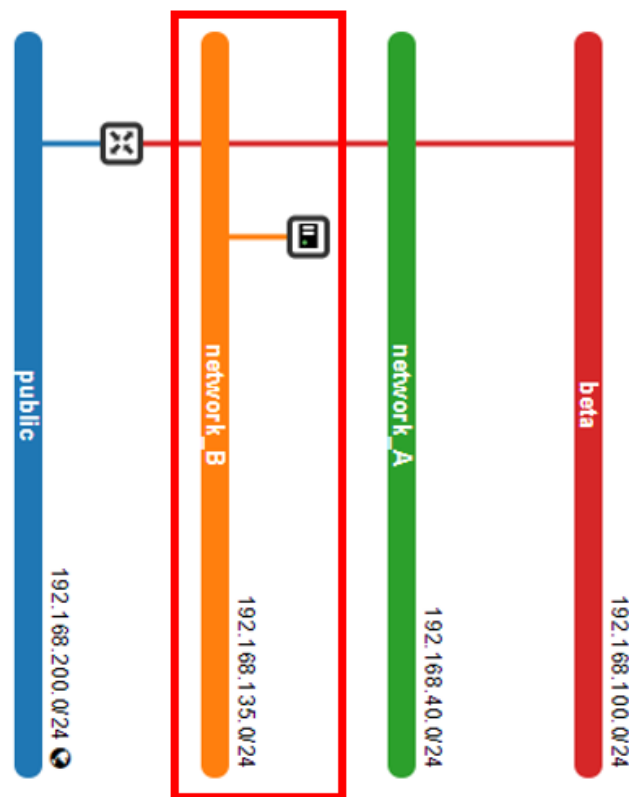


Figure 57 Test Case 6.

Summing up the evaluation of our Validation Dashboard, we saw that the Validation and the Deployment process worked effectively whenever requested. The users could check the syntax and functional validity of their files but also could easily create automated descriptors to avoid errors in each Test Case. Quick deployment was also observed, as well the start time of VDUs were between 1:30 and 2:30 minutes. Finally, one thing observed from some Test Cases was the

interface's data loss when the VDU/VM was linked in a network that has not internet access. We could retrieve the VDU's data only from the OSM client and not from the VDU's automated script that communicates with the Firebase Database in the cloud.

8. Conclusion

The evaluation of this project/thesis revealed the time saved, for the VNF vendors, to configure, build and validate a VNF, before production within the exact client environment. We presented an End-to-End and functional dashboard that is very easy to use. Additionally, prevailing the technologies that exist in the market, we managed to help the users to configure and validate their VNFs with a minimum effort. By proposing and implementing a VNF Generator and a VNF Validator that are user friendly, inside a good graphical environment. The sequence of actions that the users should follow on the dashboard is a 3-step process: (a) the users can build very quick and easy a VNF descriptor through the VNFD Generator by just filling the data form fields, (b) the users put the generated files into the Validator form and (c) the users waits and observe the validation process output as well the results of the NS deployments.

Our future goal is to work on improvements in the dashboard. Some of the improvements could be: (a) make even more generic the VNFD Generator so it can support more NFV architectures and more type of VIMs, (b) add a PDU Generator so the users can create virtual routers to connect their NS/NST, (c) create a tab in the menu bar, where the users can extend the utilities of the already deployed VNF instances / autoscaling³⁹, and (d) add the feature into the VDUs scripts to recognize and send data of the NSTs⁴⁰ that has no internet access.

Finally, we hope to inspire the OSM community to develop more VNF Generators and Validators or motivate them to use our Validation Dashboard to test and review it.

³⁹ <https://osm.etsi.org/docs/user-guide/05-osm-usage.html#autoscaling>

⁴⁰ <https://osm.etsi.org/docs/user-guide/05-osm-usage.html#sharing-a-network-slice-subnet>

9. References

- [1] L. Mamushiane, A. Lysko, T. Mukute, J. Mwangama, and Z. Toit, “Overview of 9 Open-Source Resource Orchestrating ETSI MANO Compliant Implementations: A Brief Survey”, *2019 IEEE 2nd Wireless Africa Conference (WAC)*., Aug 2019.
- [2] T. Nguyen and M. Yoo, “A VNF Descriptor for Tacker-based NFV Management and Orchestration”, *2018 International Conference on Information and Communication Technology Convergence (ICTC)*., Oct 2018.
- [3] M. Csoma, B. Koné, R. Botez, I. Ivanciu, A. Kora and V. Dobrota, “Management and Orchestration for Network Function Virtualization: An Open Source MANO Approach”, *2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet)*., Dec 2020.
- [4] T. Dreibholz, “A 4G/5G Packet Core as VNF with Open Source MANO and OpenAirInterface”, *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*., Sep 2020
- [5] M. Kourtis, M. McGrath, G. Gardikis, G.Xilouris, V. Riccobene, P. Papadimitriou, et al., “T-NOVA: An Open-source MANO Stack for NFV Infrastructures”, *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 586-602, Sept 2017.
- [6] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck and R.Boutaba, “Network Function Virtualization: State-of-the-art and Research Challenges”, *IEEE Communications Surveys & Tutorials*, vol.18, no. 1, pp. 236-262, Firstquarter 2016.
- [7] B. Han, V. Gopalakrishnan, L.ji and S.Lee, “Network function virtualization: Challenges and opportunities for innovations”, *IEEE Communications Magazine*, vol.53, no.2, pp. 90-97, Feb 2015.
- [8] P.Karamichailidis, K.Choumas and T.Korakis, “Demonstrating Multi-Domain Orchestration through Open Source MANO Openstack and OpenDaylight ”, *2019 IEEE Conference on Network Softwarization (NetSoft)*, pp. 263-265, Jun 2019.
- [9] Nguyen, T. H. Nguyen, T. Nguyen and M. Yoo, “Analysis of deployment approaches for virtual customer premises equipment”, *2018 International Conference on Information Networking (ICOIN)*., Jan 2018.
- [10] “Open Source MANO”, ETSI, 2020, [Online], Available: <https://osm.etsi.org/> , [Accessed: 02-Nov-2020].
- [11] “Openstack”, OpenStack LLC, 2020, [Online], Available: <https://openstack.org/> , [Accessed: 10-Nov-2020].
- [12] “Using Neutron with a Single Interface”, Openstack, 2020, [Online], Available: <https://docs.openstack.org/devstack/rocky/guides/neutron.html/>, [Accessed: 15-Nov-2020].

- [13] “YANG”, Wikipedia, 2020, [Online], Available: <https://en.wikipedia.org/wiki/YANG/> , [Accessed: 05-Dec-2020].
- [14] “Network function virtualization”, Wikipedia, 2020, [Online], Available: https://en.wikipedia.org/wiki/Network_function_virtualization, [Accessed: 04-Nov-2020].
- [15] “Structure Your Database”, Firebase Google, 2020, [Online], Available: <https://firebase.google.com/docs/database/rest/structure-data> , [Accessed: 16-Jan-2021].
- [16] “Check and validate descriptors”, Open Source MANO, 2020, [Online], Available: [https://osm.etsi.org/wikipub/index.php/Creating_your_own_VNF_package_\(Release_THREE\)](https://osm.etsi.org/wikipub/index.php/Creating_your_own_VNF_package_(Release_THREE)) , [Accessed: 02-Feb-2021].