UNIVERSITY OF PIRAEUS

DEPARTMENT OF DIGITAL

SYSTEMS

POSTGRADUATE PROGRAM "DIGITAL COMMUNICATIONS AND SYSTEMS"

Introduction, Overview & Experimentation Analysis of ETSI Open Source Management & Orchestration (OSM MANO) Architecture

Master Thesis

Candidate

Kapridakis Iosif

Advisor

Prof. Angelos Rouskas

Piraeus, September 2018

Contents

Acronyms

API	Application Programming Interface
BSS	Business Support System
CAPEX	Capital Expenditures
CLI	Command Line Interface
COTS	Commercial Off-the-Shelf
CPE	Customer Premises Equipment
IM	Information Model
ISP	Internet Service Provider
DC	Data Centre
DCI	Data Centre Interconnect
DHCP	Dynamic Host Configuration Protocol
EMS	Element Management System
EPC	Evolved Packet Core
EPS	Evolved Packet System
ETSI	European Telecommunications Standards Institute
FW	Firewall
GRE	Generic Routing Encapsulation
HA	High Availability
IOT	Internet of Things
IMS	IP Multimedia Subsystem
IPMI	Intelligent Platform Management Interface
L3VPN	Layer 3 Virtual Private Network
LAN	Local Area Network
LISP	Locator/ID Separation Protocol
LSO	Lifecycle Service Orchestration
LTE	Long-Term Evolution
MANO	Management and Orchestration
MPLS	Multi-Protocol Label Switching
NAT	Network Address Translation
NF	Network Function
NFFG	NF Forwarding Graph (ETSI Terminology for chaining)
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NFVO	Network Function Virtualization Orchestrator
NS	Network Service
NSD	Network Service Descriptor
NSH	Network Services Header (Service chaining encapsulation)
NVGRE	Network Virtualization Using Generic Routing Encapsulation
NVO	Network Virtualization Overlay
ODL	OpenDaylight SDN Controller
OLT	Optical Line Terminal
OPEX	Operating Expense

OSS	Open Source Software
OSS	Operations Support System
POC	Proof of Concept
POD	Point of Delivery
PNF	Physical Network Function
PNFD	Physical Network Function Descriptor
PGW	Packet Data Network Gateway
REST	Representational State Transfer
SDN	Software-Defined Networking
SF	Service Function
SFC	Service Function Chaining (IETF Terminology)
SNMP	Simple Network Management Protocol
SFF	Service Function Forwarder
SFP	Service Function Path
SMBs	Small and Medium Business
UI	User Interface
VLAN	Virtual Local Area Network
VLD	Virtual Link Descriptor
VNF	Virtual Network Function
VNFC	VNF Component
VNFD	Virtual Network Function Descriptor
VNFM	Virtual Network Function Manager
VNFFG	Virtual Network Function Forwarding Graph
VNFFGD	Virtual Network Function Forwarding Graph Descriptor
VIM	Virtual Infrastructure Manager
VRF	Virtual Routing and Forwarding
VXLAN	Virtual Extensible Local Area Network
VXLAN-GPE	Virtual Extensible LAN Generic Protocol Extension
WAN	Wide Area Network
YAML	Yet Another Markup Language
YANG	Yet Another Next Generation
XML	Extensible Markup Language

Abstract

In the first part of this thesis we are going to cover an introduction of ETSI NFV Framework and the architecture around it. Mention will be made in the current network architecture and the limitation of existing mode, the advantages of new technologies and a history behind NFV Framework. Afterwards an in depth theoretical study of high and low level of ETSI MANO, in order for the reader to get familiarize with all the layers surround it such as VIM, VNFM, Operation and Orchestration layer and NFV reference points.

Second part of this thesis is dedicated in the OSM MANO, an ETSI hosted open source MANO, closely aligned with ETSI NFV. Reader will educate upon OSM history, the scope of NFV, the mapping of OSM to ETSI NFV MANO, MANOs software components such as Interfaces, EPA characteristics, NSDs, VNFDs and VLD. We will cover how OSM MANO is architected, the key components of it, how every key component is configured and the interconnections between them. As NFV are using also SDN technologies, we will also mention how NFV and SDN are collaborated in order that this new network model, of detached software from hardware will be a common established ground, in the future. We will also briefly mention some of the projects that help OSM MANO come to life (LXD, Cloud Platforms, Openstack, OpenDayLight).

In the third and last section of this thesis we will see how OSM is installed, what are the specifications around it, a day 0 to day 2 configuration, OSM three new features compared with one and two, and our experiences and problems with the installation.

Scope and approach

Working as a network engineer in a modest small domestic company, the author of this thesis has to tackle various challenges. Taking into consideration the reduced budget policy of the industry nowadays, network engineers come across a network model controlled, to a large extent, by vendors software and hardware, controlled resources with proprietary tools end exceedingly expensive licenses to unlock features of a router or firewall p.ex. surprisingly already purchased. In a fast-growing environment like this, network engineers need to be fast in thinking onto provisioning and troubleshooting of services. Moreover, vendors hardware, overtime, is being declared Out of Service, Out of Support, Out of Sales thus making it an unusable box of working RAM, working CPU, and working functionalities. At this point we have to mention that in some cases the provisioning and the troubleshooting of vendor boxes is not implemented by company's network engineers but by vendors, because difficult tasks knowledge is not publicly known, creating time gap, operation cost and manhours wasted. With all the above the motivation for the elaboration of this thesis is to examine technologies and projects that tackle these challenges. Projects that use existing hardware, with fast deployment and economic solutions, with no need of special hardware and proprietary tools, scalable, flexible, vendor independent and opened sourced, manageable by company's personnel. Scope of this thesis is to examine ETSI NFV framework, Operation, usability, and also to study OSM MANO as a tool that will question the above challenges, resulting a more vendor free, fast easily operational monitoring network environment.

1) Introduction

In the recent years of Networking Architecture there is a new approach towards the infrastructure - Physical and the principals around them.

Vendors and Telco's Research and Developing are focusing more on the exploitation of existing resources rather than investing in new hardware.

The need of new services and the constant switching of networks, and the need of 'on the fly change' of a network, accentuate the necessity of a new model approach. Until now we had a physical hardware operating the resources - services for which they were entrusted, thus any change in the networking table or service was service affecting. The new model and approach now is to decouple network table and services from the physical appliance, virtualize functions, introducing the NFV (Network Functions Virtualization).

NFV is a revolutionary way handling operation configuration and management on the Infrastructure. The above decoupling liberates and abilities a fast configuration dynamic ('on the fly') change towards services application requirement and network change. Datacenter Infrastructure (Servers, switches, storage etc.) from the static used type will be transformed into a dynamic more agile environment. VNF virtualizes those functions, eliminating the need for specific hardware dedicated on a specific service or function.

The functionality of a physical hardware is transformed into a virtual instance usually termed as virtual machines (VM) which has the ability to function like a conventional hardware. VNFs act as building blocks that can then be chained together to create comprehensive communication services. Chains can be modified dynamically to leverage complex service topologies via VNF orchestration.

1.2) Limitation of the existing model use.

With the upcoming rise of technology in new and traditional continent users -IoT-, increase in capacity bandwidth, hardware resources, infrastructure overall is mandatory, causing a respectively increase of costs. Infrastructure models, as until now, have limitation to reciprocate in those demands compared with operational, scalability-flexibility, and management costs.

Scalability: Resources of hardware and software vendors devices have limitations over the fast change of network tables and usage. More network usage requires more resources needed by the machine its self, and more resources needed, require more power on the hardware box to couple these necessities.

Flexibility: Vendors equipment are using a proprietary model structure. Each hardware has a specific -and only- use, handling its software and hardware functions. Expansion in services and resources demand new licenses from its vendor appliance thus importing limitations at flexibility combining hardware and software to new usages.

Monitoring Management: Existing network devices are using SNMP, syslog, Netflow or comparable systems to gather information about Monitoring. But some devices are using a non-standarized MIB library or their interpretation syslog messages. This makes difficult to Operators to have one Operation Management Tool. Also, some Vendors in depth analytics Management Tools are not available from the beginning but need of silence, increasing costs.

Time gap: From the perspective of Service Providers, this new dynamically shifted environment causes new needs in services that require new hardware and Software demands, to keep up with the fast-growing usage of network. SPs need to constantly upgrade their network equipment, depending on the new releases of Vendors and also constant redesign of network to meet the new demands. The complexity of these decisions results a time gap between implementation and offering to customers.

Operation costs: Most of the time SPs have a support contract with a specific Vendor and a highly trained team to Operate on Vendors machines. Group training on a specific Vendors equipment results a difficulty in changing Vendors as new training is needed and also time to get familiar with new hardware.

Network upgrade cost: Up until now network maintenance on devices includes a team designing the new upgrade and personnel to perform physical and software changes to apply them. Because of the Service affection due to maintenances these

kinds of changes are performed during of business hour resulting a increasing of costs and personnel usage.

Over/under capacity: Existing network implementation results services and capacity cannot be scaled up or down on-demand nor can be scaled based on geographic demand. Reduced operational efficiency due to the diversity of the network infrastructure and management platforms. Lack of orchestration leads to manual installation and inability to modify the capacity on-demand. [1]

For reasons above a new model approach was needed to compensate with new needs. NFV architecture environment visualizes network functions making these functions to coexist in several network devices with dissimilar vendors machines. De-attaching software from hardware machine provides a variety of solutions depending of the needs. Scalability and flexibility to network usage, sharing the existing resources, reducing the cost of ownership providing new services on demand and capacity solutions.

1.3) Overview and Advantages

In generally, better use of network resources as well as increased network scalability and agility is greatly aided by the use of VNFs. Additionally, as VNFs take the place of physical hardware, further advantages ensue, such as increase of available physical space, reduction of power consumption and operational and capital expenditure.

Hardware flexibility: NFV architecture combines memory, network capability, storage of the traditional physical hardware of many vendors into a dynamic block usage. Service providers can now choose from several vendors appliances to provide sources to their own need of a Network. Also, service providers can reuse existing machines from their data-center.

Cost: Having the ability to deploy new services onto the existing Infrastructure thus reducing operator CAPEX (Capital Expenses) and OPEX (Operational Expenses) through reduced equipment costs and reduced power consumption.

Monitoring management: Operation architect over a wide range of Vendors devices and controlling the Network Functions through a single spot gives the ability of one universal monitoring tool analyzing-reporting the network function availability and needs to provision.

Reduced time gap: NFV network services can be deployed on demand 'on the fly' depending on the usage and the needs. Adding - removing, provision functions and capacity, automatically from software tools, without the need of onsite personnel or installation, thus reducing costs. Functions and operations that reduce the time to apply the new services need to consumers.

Operation cost: Unhooking Software from hardware gives the freedom to deploy to a common entity different NFVs. Service providers utilize this freedom to existing machines so new services can be deployed without the need of a maintenance window or purchasing new hardware and software specific for that necessity, therefore Operational cost drops.

Network upgrade: Maintenance of a network to meet modern needs, upgrading and re-provision of it, could be decided on the fly, without complex design and Service affecting downtime. New services creation on demand absorbing resources from the existing machines combining them to new releases.

Vendor independence: With the NFV architecture, comping resources from several machines, shifting weight to Software than hardware its self, Service Providers are not

restricted to usually one Vendor. Multiple machine can handle the same Network functions to support network and services. p.ex A bug on a hardware machine won't be Service affecting as Network Function will be automatically moved to another machine, giving the opportunity to personnel to fix the bug or use the machine for another function that doesn't conflict the above.

Security: Security is a constant challenge in network environments. A secure path is required that will allow operators to manage and provision the network, whilst enabling their customers to use their private virtual services-environment.

Automation: With NFV Management and Orchestration Framework (NFV-MANO) all above mentioned performances can be handle automatically, with no personnel on site to move physically links to hardware entities, scaling up or down the capacities or services involved to Network Functions with a centralized perspective. [2]

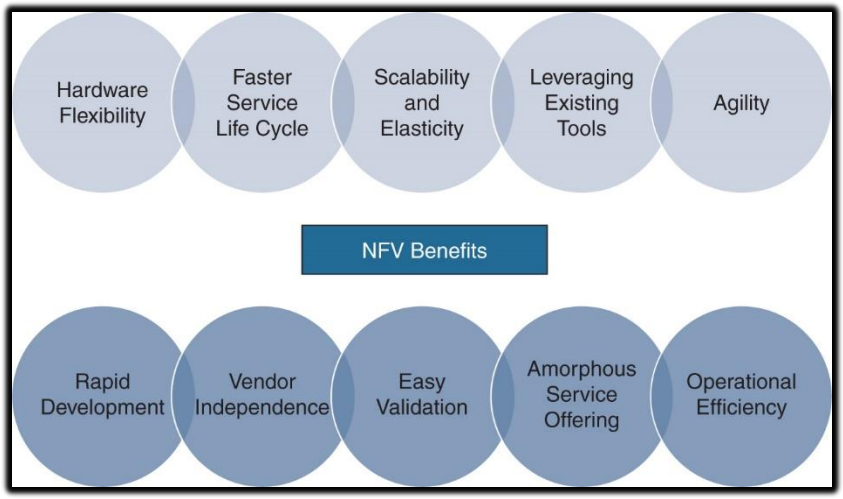


Image 1. NFV Benefits
[http://ptgmedia.pearsoncmg.com/images/chap1_9780134463056/elementLinks/01fig18_alt.jpg]

2) NFV Framework

In this chapter we will reference out a short history analysis of the Framework introduced and also the High and Low-level analysis of it. High level analysis introduces, in an abstract way, the key structures of ETSIs NFV blocks and their functionalities, where Low level analysis consists of a more detailed reference of the individual components, such as blocks and reference points and their, occasionally, overlapping functions.

2.1) History

Current network architecture management is vendor defined and has limited room for customization. Only with vendors support can enhancements introduced in management capabilities or new requirements be implemented. On the other hand, with the introduction of virtualized network's architecture, a multitude of resources are available, managed and operated in a variety of levels, more detailed and individual. Automation, coordination and interconnection of functional blocks and layers, introduced by the NFV Architectural Framework, are more agile and scalable. [1]

This results to the need for another functional block to the framework that communicates with and manages both the VNF and NFVI blocks. This block manages the deployment and interconnections of the VNFs on the hardware and allocates the hardware resources to these VNFs.

The inherent ability of the MANO block to fully supervise and manage the entities, enables it to be fully aware of their operational state, utilization and usage statistics, thus making it an excellent interface for gathering and utilizing data by the billing and operational systems. [3]

To unsure the standardization of this framework in In October 2012, a specification group, "Network Functions Virtualization", published a white paper at a conference in Darmstadt, Germany, on software-defined networking (SDN) and OpenFlow. The group, part of the European Telecommunications Standards Institute (ETSI), was made up of representatives from the telecommunication industry from Europe and beyond (AT&T, BT, Deutsche Telekom, Orange, Telecom Italia, Telefonica and Verizon). Now, years later, a large community of experts are working intensely to develop the required standards for NFV as well as sharing their experiences of its development and early implementation. ETSI counts over 245 individual companies as its members, among which 37 of the world’s biggest service providers as well as

telecoms and IT vendors. ETSI has successfully completed Phase 3 of its work with the publication of ETSI Group Specifications. These specifications, build on the third release of ETSI documents published in October 2017 include an infrastructure overview, updated architectural framework, and descriptions of the compute, hypervisor and network domains of the infrastructure. These specifications also include service quality metrics, security and trust, resilience and Management and Orchestration (MANO). Following the white paper publication, numerous more in-depth material has been released, namely a standard terminology definition and use cases scenarios for NFV which prospective Network Virtualization vendors and operators refer to. [4]

2.2) High level Analysis

As mentioned earlier NFV allows developing Software to run on different identity devices sharing the hardware, the exact antithesis of the traditional network architecture. The key components of the ETSI - NFV Architecture is composed of three major structure units.

1) Network functions virtualization infrastructure (NFVI)

2) Virtual Network Functions (VNF)

3) Network functions virtualization management and orchestration architectural framework (NFV-MANO Architectural Framework)

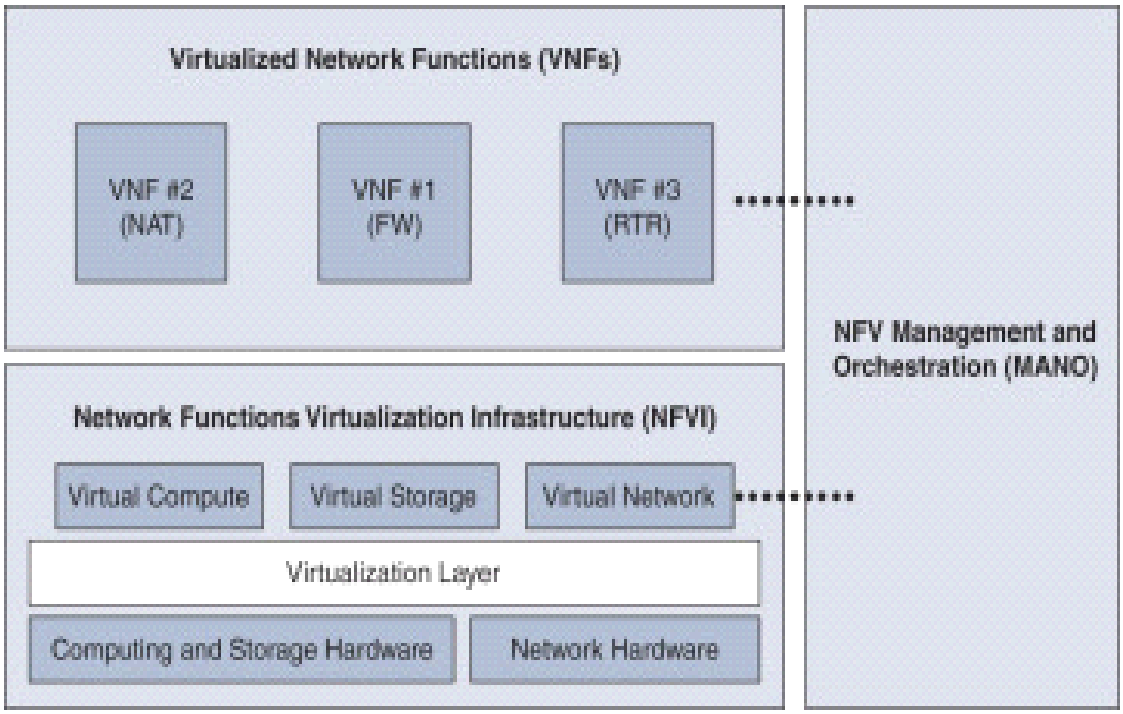


Image 2. High level ETSI NFV Framework
[http://ptgmedia.pearsoncmg.com/images/chap1_9780134463056/elementLinks/01fig03_alt.jpg]

1) NFVI: Refers to the area of all hardware and software components who build the area to host the VNF. Hardware for the Virtual Machines, Software for the Virtualization and the Virtualized resources.

2) VNF: Software executions of Network Functions that are stationed over the NFVI.

3) NFV-MANO: Gathers of the information for the functions, interfaces, repositories and reference points of Network Functions for managing and orchestrate over them. An independent block that communicates with VNF and NFVI blocks and creates/deletes resources.

VNFs can be allocated as a multifunction of VNF working together or as standalone. The protocols used to a Network Function that are being virtualized into a VNF do need to be aware of the virtualized execution. p.ex. we could have 2 stand-alone VNFs (a) DNS, (b) Mail Server that run separately from each other and are not physically connected handling by one 'higher' VNF that is responsible for the interconnection.

As for mentioned earlier with the specific architecture we have detached Software for hardware machine, so all these VNFs can be implemented into several hardware machines that have Processors, Memory, Storage and minimum of one network interface. Moreover, with this architecture we can have a combination of several machines working together to provide p.ex. more memory to a specific Network Function when need. Resulting a cluster of resources independent from each other to provide to the necessitate uses. [5]

2.3) Lower Level Analysis

As part of the Low-level analysis MANO Frame can be analyzed in three block functions as below:

- Virtualized Infrastructure Manager (VIM)

- Virtualized Network Function Manager (VNFM)

- NFV Orchestrator (NFVO)

Also, we will refer to NFV Framework layers,

2.3.1) VIM:

As part of MANO block, VIM is responsible for managing the resources, storage computing networking, the virtualized hardware and the correspondence with the Virtualization layer. So, VIM has all the information needed to manage hardware resources and performance (availability, status, power and utilization). VIM can control the resources for the NFVi parallel working with other blocks (VIM or other) to determine the amount of resources needed for Network Functions both on the same Point of present or across the domain assigned to, but also into multiple NFVI.

The role of a VIM is to configure the compute, hypervisor and infrastructure network domains. As mentioned earlier there is a correlation between NFVI and VIM. NFVIs through the Virtualization Layer join the hardware physical resources to Virtual Hardware. Such as Computing, Storage and Network resources are perceived as Virtual Computing, Storage and Network, as per ETSI framework. Therefore Computing (CPU plus Memory) can be converted to pool and be used by different host through a cluster formatted resource. Virtual Storage can be formatted into NAS topology and be assigned or de-assigned to devices when needed. Respectively Virtual Networking can combine NICs available ports from routers, switches Optical transponders and Wi-Fi to provide capacity when needed. It must be mentioned that these functional blocks are independent from each other and not mandatory to run on a single device. The whole process and management of this function is managed from VIM.

2.3.2) Virtualized Network Functions Manager-Layer

The VNFM is responsible for the lifecycle management of VNF instances. Each Network Function is associated with one VNFM. A VNFM could be assigned into one or multiple VNF instances same or different type. As a Network Function is designed to work in any hardware device that has the generic functions required but isn't aware of which device is running on. Thus, when a VNF requires more resources a forwarded message will occur from the VNFM informing the VIM that more resources are needed and that message will be transferred to the VM through Virtualization layer locating the required resources.

As mentioned earlier a network service could be associated with one or more NFVs. In the multiple form of NFVs some functions may have correlated holdings in other VNFs or would require a specific flow of execution. Responsible for that is VNF-Forwarding-Graph (VNF-FG) or service chaining (create, query, update, delete), e.g by creating, maintaining the virtual networks, Virtual Links, subnets, and ports.

The Element Management is responsible for the FCAPS (Fault, Configuration, Performance, Accounting, Security) management of VNF's network application.

- Fault management for the network functions which are provided by the VNF.
- Configuration for the network functions which are provided by the VNF.
- Accounting for the usage of VNF functions.
- Performance Measurement results collection for the functions provided by the VNF.
- Security management for the VNF functions.

To perform those functions that require exchanges of information regarding the NFVI Resources associated with the VNF, the EM is aware of virtualization and cooperates with the VNF Manager. It also intercommunicates with the VNFs to reach the VNFM, providing a proxy to the VNFM for management as well as operations of the VNFs. The FCAPS are still managed by VNFM, but it can be supported by the EM to interact with the VNF for this form of management. [5]

2.3.3) Operational and Orchestration Layer

From, the reference point of Service Providers, transition to NFV architecture model won't cause the reorganization of management tools. OSS/BSS systems could easily transact to NFV through NFVO. NFVO supervises end-to-end services deployed in communication with VIM and NFVManager.

This layer handles the information's about the resources from VIM's managing and also across multiple VIMs, through Resource Orchestration. Moreover, through Service Orchestration has the management of Network Services, plus has the management of Network Service and Network services instantiation such as query, scaling, updating queries, management performance collective information plus termination of above. It also manages the initiation of VNFM and NFVs, validates the resources needed NFVI for a service instance through VNF manager. Furthermore, NFVO manages the creation, update and delete of Forwarding Graphs.

Independently of any VIMs and while utilizing the Resource Orchestration functionality (RO), the NFVO supplies services that assist accessing the NFVI resources, as well as handling of VNF instances pooling resources with the NFVI.

As a summarization of the above we can split this layer into two functionalities between the Resource Orchestration that allocates and manages NFVI resources to the VIMs referred to and the Service Orchestration, that defines the interconnection of NFVIs used by VNFs providing a Service.

2.3.4) NFV Reference Points

In the NFV framework key role in the interconnection between logical functional blocks have the Reference Points. They play a key role to implement a service instance as they are responsible to ensure that flow of information between block is consistent and exchange information needed between those logical blocks. Below is a detailed report of those Reference Points in correlation to the boundaries happening and each responsibility.

Os-Ma-nfvo. This reference point is responsible for information exchange between OSS/BSS and NFVO, pertaining VNF package management, VNF and Network Service lifecycle management (instantiating, updating, querying, scaling, and terminating) and Network Service policy management such as authorization and access. Additionally, it forwards usage, events, and network performance Service Instances to OSS/BSS and querying VNF instances and network services to NFVO.

Ve-Vnfm-vnf. This reference point is responsible for information exchange between VNF and VNFM, responsible for instantiating, querying, updating, scaling (up or down) and terminating the VMs. It is also responsible for VNF function, events and configuration from VNFM to VNF as well as events and configuration from VNF to VNFM.

Ve-Vnmf-em. This reference point is responsible for information exchange between VNFM and EM. Ve-Vnmf-em, handles the same information exchange (instantiating, querying, updating, scaling up or down and terminating the VMs, configuration exchange), between VNFM and EM.

Nf-Vi. It is responsible for information exchange between NFVI and VIM. This reference point creates, configures and removes inter VM connections; allocates, updates, migrates and terminates VMs. Moreover, it transfers information to the VIM regarding usage records, failure events and configuration, for NFVI resources (virtual, software, physical).

Or-Vnfm. This reference point is responsible for information exchange between NFVO and VNFM. Forwards state and events information to the VNF. It also handles instantiation, updates, scales, state and termination package query of the VNF.

Or-Vi. This reference point is responsible for information exchange between NFVO and VIM. It supports configuration, events, results and usage of NFVI to NFVO, handles VNF image update, allocation/deallocation. Finally, it communicates the updates, releases and reservations of NFVI resources.

Vi-Vnfm. This reference point is responsible for information exchange between VIM and VNFM. It handles NFVI resource information (reservation, allocation and release) and passes information of measurements results, usage and events used by VNF for a specific NFVI resource.

Vn-Nf. This reference point is responsible for information exchange between NFVI and VNF. It communicates information regarding performance, lifecycle, and portability requirements of VNF. [1]

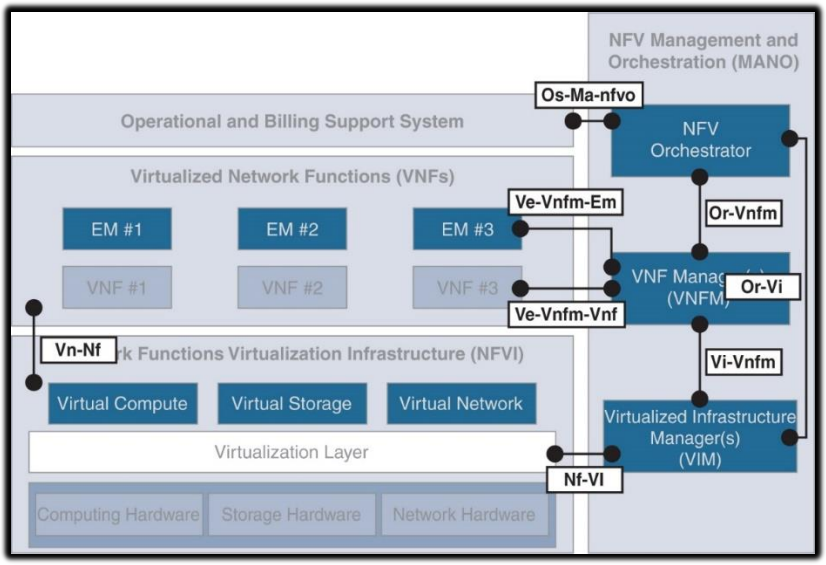


Image 3. Reference Points
[http://ptgmedia.pearsoncmg.com/images/chap1_9780134463056/elementLinks/01fig07_alt.jpg]

2.3.5) Summarization

A summarization and end-to-end view of Orchestration.

-The full view of a Service Instance is perceptible by NFVO.

-Required NFVs are triggered by NFVO and communicated to VNFManager.

-VNFM then calculated the needed VMs and resources to that particular instance and passes this information to NFVO in order to satisfy VNF needs.

-Validation of resources needed to the particular instance and requesting these resources, if available.

-That request is passed through to VIM to locate and create and apply those needs to VMs.

-VIM passes information through Virtualization Layer for that creation.

-VMs are created.

-VIM is responding back to NFVO for the competition of resources creation.

-VNFM is informed by NFVO that the requested VM are available.

-VNFM instantiates the available VNFs.

-VNF configures accordingly the VNFs.

-VNFs created successfully so VNFManager interacts with NFVO reporting that VNFs are successfully created, configured and available to use.

3) Motivation, Interdependence, Considerations.

In this chapter we will analyze the motive behind the development of NFV environment in relation with new network design architecture inserted, alongside with other new technologies such as the overlapping function of SDN and NFV. Furthermore, we analyze how these two new models operate together and the differences between them. Another aspect is the insertion of new Operating Systems - Platform – Project and the use of them in the OSM MANO. We will summarize some key characteristics of the new technologies used such as Cloud Computing and Projects such as LXD, Openstack and OpenDayLight. Lastly reference will be made into the NFV Design Considerations.

3.1) Motivation

The management of NFV environments is a challenging task in itself because of the huge amount of data available, with several VNFs migrations in the total network environment, dynamic resource allocation, requirements of tenants, and the information of VNFs. Moreover, information about the physical infrastructure, such as CPU, memory resources, bandwidth etc. All the above describe a highly dynamic environment and network that network engineers must address in a production network. From engineer's perspective we have to understand and integrate:

- Number of packets processed and the available resources to them.
- Geographically necessity of resources, and the conflicts around VNFs.

- Service infrastructure requirements, both physical and virtual.
- Production of future network expansion.

All the above reasons force the community and the industry to produce a highly adaptive, standardized model, convenient and easy to use, ready to reduce operating and complexity costs.

3.2) Integration of SDN and NFV

Due to the dynamic network requirements of cloud data-centers, Software Defined Networks were born by Open Network Foundation that introduced the term. Key aspects of SDN are the following:

- Automation of the network lifecycle management reducing operational costs and improving users experience.
- Centralization of the control management functions, as they are decoupled from physical hardware imported to the cloud.
- Abstraction of management through APIs with direct network interaction. [6]

SDN has been equally exciting as NFV, and both of them are highly complementary but not depended from each other. SDN and NFV decouple Hardware from Software, but with different trajectories. While NFV is trying to consolidate many network functions running on switches, servers, storage, with high level automated management, and quick on-the-fly implementations, SDN tries to enhance network performance, simplify network installations, ease maintenance procedures.

Traditional router network function through routers, with the FIB (Forwarding Information Database) network packets are processed on the data plane passing information to the right path with the appropriate routing information.

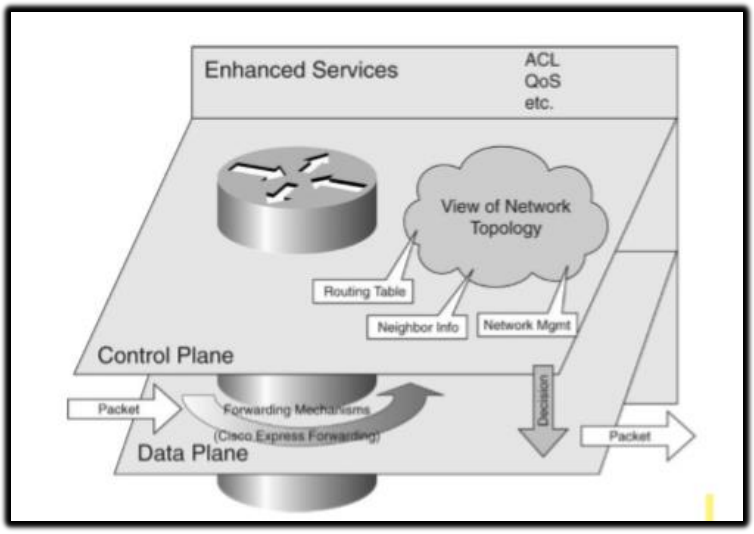


Image 4. Traditional router Data, Control plane
[https://i2.wp.com/www.networkset.net/wp-content/uploads/2010/07/cef.png?resize=500%2C300]

Packets entering router are processed to the data plane. Control plane with Routing Information (data)Base (RIB) and Label Information Base (LIB) process them in software and used to populate FIB and the LFIB. Routing tables, neighbors, and network management is held in control plane. Packet without a known FIB entry, router passed it to control plane, to make the appropriate decision and pass the information ruled to FIB again. [7]

With the centralization of the SDN devices, a controller has information about the network neighbors, making the appropriate traffic routing decisions.

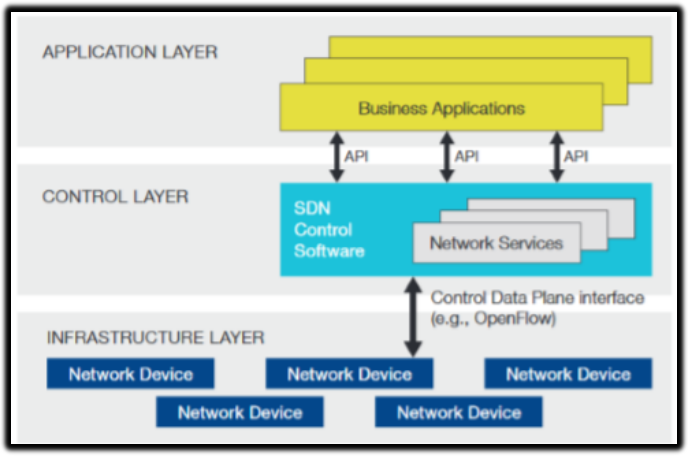


Image 5. SDN high level architecture. [
https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf]

In the image above, we see the SDNs high-level architecture

- Infrastructure layer communicates with network devices forwarding information, with predefined interfaces. Formed by the network elements (NE) that provide flow switching and other data-plane functions. Software defined controller knows the network structure and services processed and it can make the appropriate optimal traffic decisions.
- Application layer is held by Operating/Business Applications, associated with the network. This association is performed through APIs, consuming SDN services.
- Control layer contains the network intelligence and manages the network forwarding behavior through an open interface such as OpenFlow-based on the applications requirements. Also, control layer makes abstraction views of the network and thus simplifies maintenance or security.

Summing up NFV and SDN have common goals towards networking and complete each other. SDN can support NFV infrastructures providing scalable and on-demand networking according to the changing VNF connectivity requirements for both virtual and physical networking infrastructures. Classic networking is shifting into an improved vendor independent automation and management network with finer granularity of control. The acceleration of new service innovation is enabled with programmability by operators, enterprises, independent vendors and users.

3.3) Cloud Platforms

Cloud computing is a significant advancement in the delivery of information technology and services. By providing on demand access to a shared pool of computing resources in a self-service, dynamically scaled and metered manner, cloud computing offers compelling advantages in cost, speed, and efficiency.

Contrary to traditionally deployments that require applications to be bound to a particular infrastructure, cloud brings in capabilities to allow applications to be dynamically deployed onto the appropriate infrastructure at runtime. This elastic aspect of Cloud computing allows applications to scale and grow on demand without needing traditional patches of upgrades.

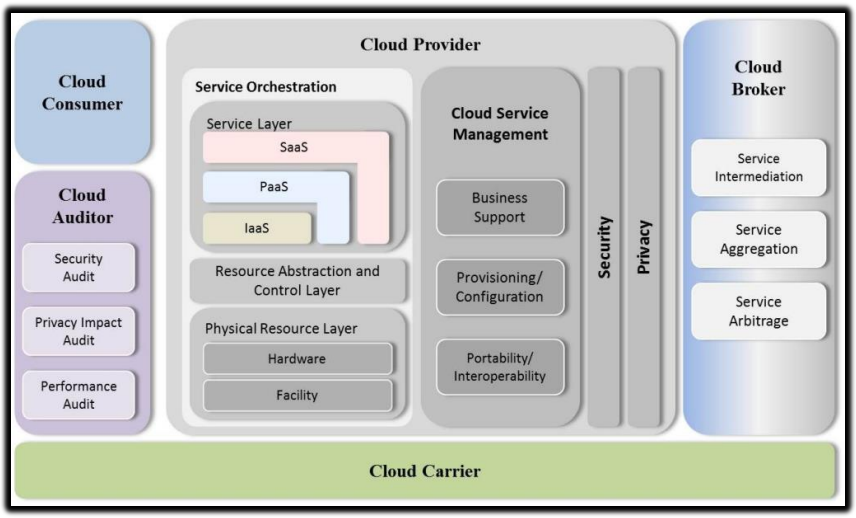


Image 6. Conceptual Architectural Model
[https://www.nist.gov/sites/default/files/documents/itl/cloud/SP_500_293_volumeII.pdf]

Cloud Computing inserts the concepts of resource pooling, virtualization, dynamic provisioning, utility and commodity computing within the public Cloud or create a private Cloud that meets these needs. What Cloud computing is trying to achieve is self-service provisioning across tenants. Those tenants could be a project, division or even a different company. Major categories of models associated with cloud computing are:

- Infrastructure as a Service (IaaS): Managed by the service providers, VMs are available to customers with the physical hardware (servers, storage and networking).
- Platform as a Service (PaaS): Managed by the service provider and all the details about the physical and virtual equipment abstracted from the developer, the platform for programmers is provided as a service.
- Software as a Service (SaaS): Managed by the service provider, an application is made available to customers, while all the details of the underlying platform are abstracted.

The key components of the Cloud infrastructure are summarized below.

- The logical abstraction layer that pools the physical resources which support the Cloud Infrastructure.
- Cloud Builder that configures and operates the Cloud platforms and infrastructure as a Service.
- Cloud Application Builder that develops applications for the Cloud and deploys them on the PaaS (Platform as a Service) platform and offer as SaaS (Software as a Service) services. Cloud Builders support multiple application builders and applications.

- SaaS consumers that consume the software services and Cloud Application Builder support multiple SaaS consumers. Also, a hybrid model is available to SaaS consumers with functionalities partly provided by classic IT or internal private Cloud on one hand and public Cloud on the other.
- Cloud Application Builder and Cloud Builder supported by the Cloud Management infrastructure.
- Application management, provided by the Cloud Builder to manage and provision applications deployed on the Cloud, includes the self-service capabilities. [8]

Centralized command and control becomes a must while the need for automated provisioning of virtual networking equipment almost becomes a requirement, with the parallel usage of existing physical devices and new virtual ones. This is the optimum place of cloud computing, SDN and NFV. It is important to link the development environment with the production environment, allowing patches and upgrades to be deployed quickly and SDN and NFV can help in this, building fenced networks for development, quality assurance, production thus resulting in fewer errors. Also, complementary they bring down the cost of procuring and deploying hardware, and consequently the CAPEX for the telecom operator comes down significantly.

Issue	NFV	Cloud Computing
Approach	Service/Function Abstraction	Computing Abstraction
Formalization	ETSI NFV Industry Standard Group	DMTF Cloud Management Working Group
Latency	Expectations for low latency	Some latency is acceptable
Infrastructure	Heterogeneous transport (Optical, Ethernet, Wireless)	Homogeneous transport (Ethernet)
Protocol	Multiple Control Protocols (e.g OpenFlow, SNMP)	OpenFlow

Table 1. Comparison of NFV in Telecommunication network and Cloud Computing. [http://www.maps.upc.edu/rashid/files/NFVSURVEY.pdf]

3.4) LXD

Linux containers daemon (LXD) are self-contained execution environments—with their own, isolated CPU, memory, block I/O, and network resources—that share the kernel of the host operating system, seemingly like a VM, thus increasing speed and density. LXD was created by Stéphane Graber, who works for Canonical.

Compared to traditional server virtualization, containers are an appealing proposition in an application environment that has web-scale requirements. Containers decouple applications from operating systems, which means that users can have a clean and minimal Linux operating system and run everything else in one or more isolated container.

Canonical points out that:

- LXD achieve 14.5 times greater density than KVM (Kernel-based Virtual Machine).
- LCD launches instances 94% faster than KVM.
- LXD provides 57% less latency than KVM.

LXD behaves exactly as hypervisor but eliminates the overhead of virtualization or machine emulation. LXD's density relies on the fact that the same kernel is managing all the workload processes, thus improving quality of service and latency issues. LXD services higher density than KVM, as the underlying hypervisor can handle common processes more effectively. [9]

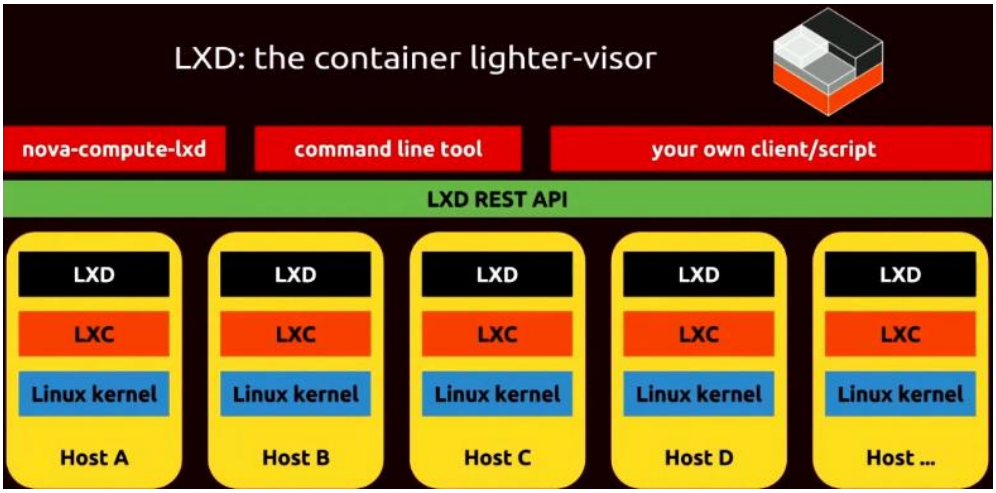


Image 7. Graphical depiction of LXD usage [LXD Containers - Ubuntu Submit 2015]

- LXD run on any architecture (Intel, AMD, IBM) and in any cloud.
- They are compatible with existing Linux container technologies.
- Bring storage, network, and remote API interfaces to containers.
- Compatible with any Linux distro (Ubuntu, RedHat, CentOS)

Below is an example of the display of installed LXD in OSM MANO host environment, showing the Host environment that incorporates the host containers as the image above.

```
osm@osm:~$ lxc list
+-------+---------+-------------------------------+------+-----------+-----------+
| NAME  |  STATE  |             IPV4              | IPV6 |    TYPE   | SNAPSHOTS |
+-------+---------+-------------------------------+------+-----------+-----------+
| RO    | RUNNING | 10.122.240.90 (eth0)          |      | PERSISTENT | 0         |
+-------+---------+-------------------------------+------+-----------+-----------+
| SO-ub | RUNNING | 10.122.240.73 (eth0)          |      | PERSISTENT | 0         |
+-------+---------+-------------------------------+------+-----------+-----------+
| VCA   | RUNNING | 10.44.127.1 (lxdbr0)          |      | PERSISTENT | 0         |
|       |         | 10.122.240.140 (eth0)         |      |           |           |
+-------+---------+-------------------------------+------+-----------+-----------+
osm@osm:~$
```

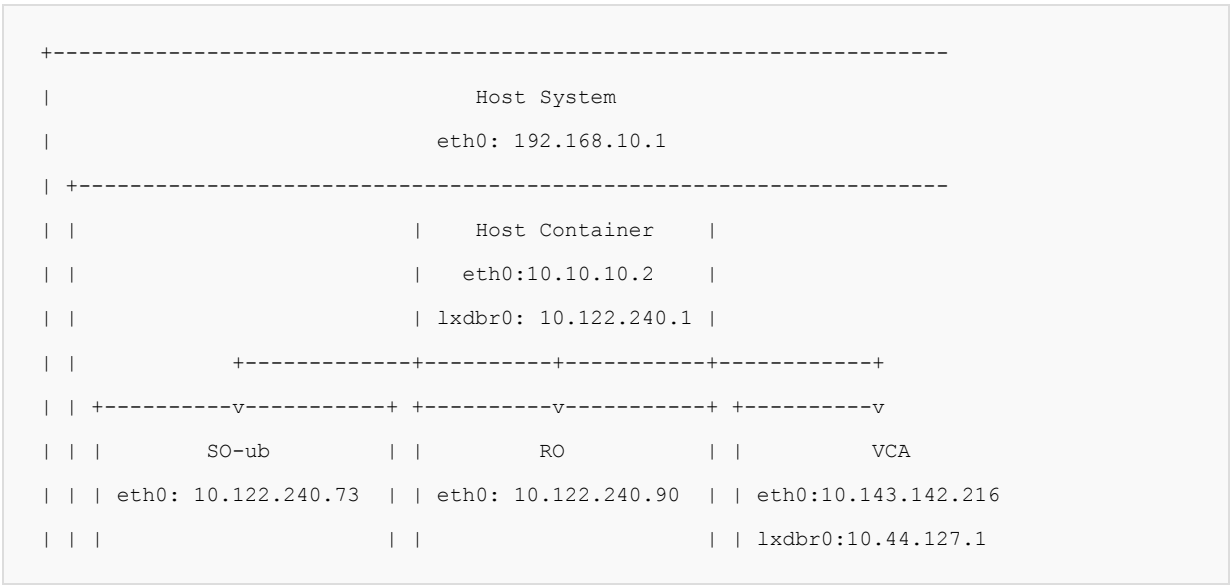


Image 8. LXC list of installed OSM release three.

3.5) OPENSTACK

On May 2018 Openstack foundation released the media a presentation expounds choosing MANO OSM for the right Orchestrator into End-to-End NFV with Openstack [10]

Openstack is an open-source platform framework enabling cloud deployment and management suitable for IaaS (Infrastructure as a Service) service models. OpenStack began in 2010, with its initial release, as a joint project of Rackspace Hosting and NASA. As of 2016, it is managed by the OpenStack Foundation, a non-profit corporate entity established in September 2012, with more than 500 companies joined. [11]

Openstack choose OSM MANO from Openstacks tacker, OpenBaton, ONAP (Open Network Automation Platform) as it is a proven working solution, with large community activity and not technical deformations.

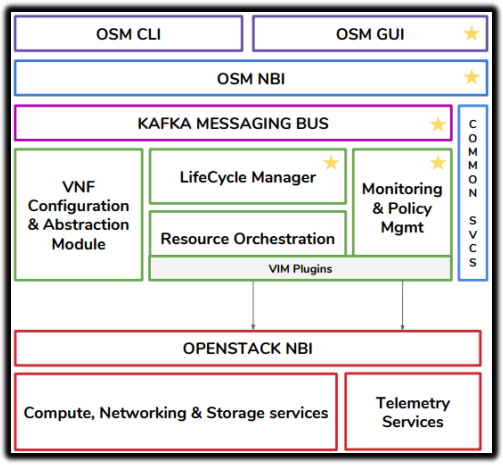


Image 9. OpenStack and OSM MANO
[https://qnpic1.fangketong.net/201809/14/20180914_18133_1197475_3.png%21web]

The OpenStack architecture is built upon RESTful modular services. End users can interact either through the APIs or the provided CLIs and dashboards. Openstack supports telemetry, orchestration and advanced network services.

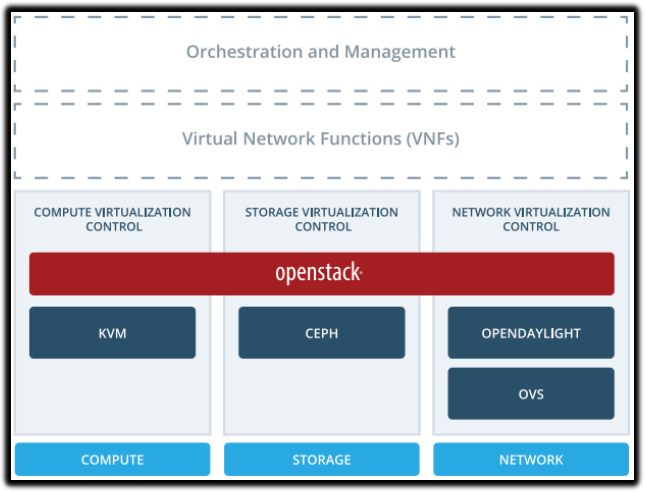


Image 10. Openstack and NFV
[https://wiki.opnfv.org/display/INF/OpenStack]

OpenStack is fundamental to the Virtualized Infrastructure Manager (VIM) as part of the Management and Orchestration (MANO) function that controls the assignment of virtualized compute, storage and network resources from the NFVI to support the VNFs. The primary OpenStack projects involved are:

- Nova: OpenStack Compute for managing bare metal or virtual servers.
- Cinder: OpenStack Block Storage for virtual storage
- OpenStack Networking (code named Neutron) providing virtual networking [12]

OpenStack provides all the functionalities required to control and manage an NFVI by handling the Virtual Machines that host and run the VNFs, including the creation and management of IP addresses and the networks between them.

3.6) OpenDaylight

OpenDayLight is a collaborative open source project hosted by The Linux Foundation. (2013). The software is written in Java programming language and the goal of the project is to promote network functions virtualization (NFV) as also software-defined networking (SDN).

It is characterized by highly availability, scalability, and modularity. Modern heterogeneous multi-vendor networks can be 'fed' by its multi-protocol controller infrastructure built for SDN deployments. OpenDaylight provides a model-driven service abstraction platform that allows users to write apps that easily work across a wide variety of hardware and south-bound protocols. [13]

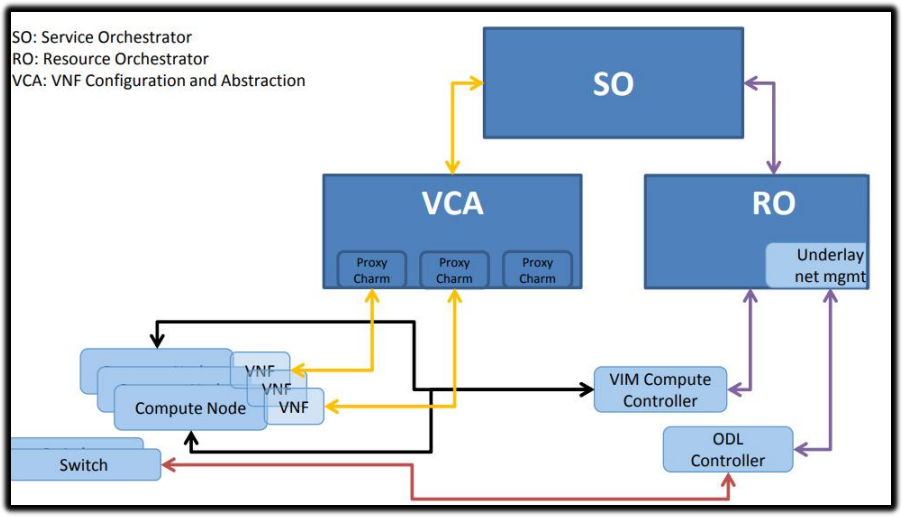


Image 11. Open Day Light and OSM.
[http://vialimachicago.com/review-nfv-architecture-ideas/what-is-the-best-nfv-orchestration-platform-a-review-of-14]

The OpenDaylight architecture includes:

- The controller platform, which contains services and applications which implement controller logic independently of the underlying network technologies below.
- Southbound interfaces and protocol plugins. OpenDaylight offers a ModelDriven Service Abstraction Layer (MD-SAL) which allows applications to control a broad range of underlying networking technologies.

- On top of the components above, there are the several north-bound APIs, the AAA (Authentication, Authorization and Accounting) layer and the DLUX Interface framework. [14]

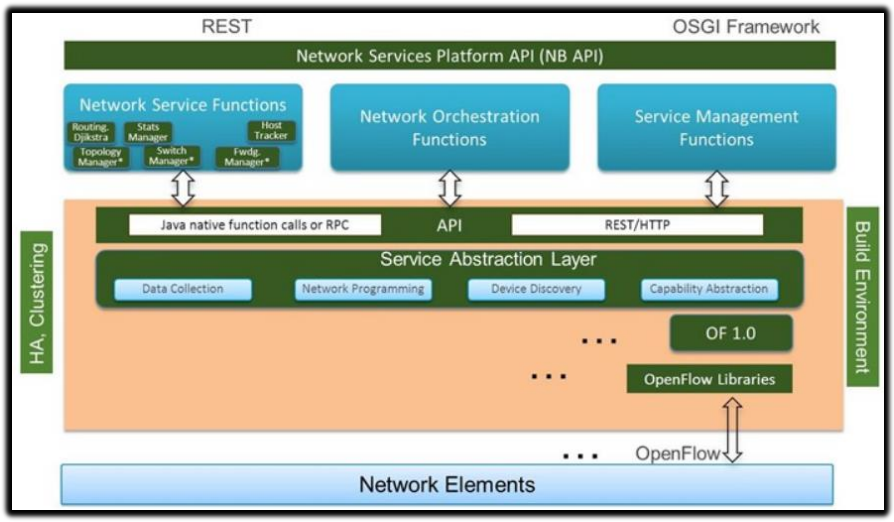


Image 12. OpenDayLight Architectural Framework.
[https://wiki.opendaylight.org/view/OpenDaylight_Controller:Architectural_Framework]

3.7) NFV Design Considerations

With the steady growth of NFV, it is vital to point out that Virtualization of Network Services into virtual instances, is not only sufficient. Along with the QoS and SLA assurance for NFV to be acceptable, they have to meet the following requirements:

- Vertical Integration: NFV enlarges carriers' network architecture with its nature to decouple hardware and software generalization. NFV is a complex project with multiple technologies, interfaces and vendors. Thus, it is important that vertical cloud platform integration is performed smoothly and quickly.
- Network Service Deployment: Since multiple components, in current networks, are provided by different vendors and involved during processes, interoperability can cause problems in Network Service deployment. From OSS/BSS up to NFVO, VNFM, VIM, Cloud OS and COTS. Descriptors, such as NSD, VNFD should also follow the standardized format for a successful service instantiation.
- Service Assurance: Functionality is equivalent to Service performance and high availability. When there is a failure in the network, the information should be propagated quickly to the upper layers, so that each component can use this information effectively. Automation is a key factor in NFV lifecycle management and different components or blocks, have self-healing mechanisms build in. These mechanisms are triggered under specific conditions, such as notifications

that have been received from the network. This information is vital for a successful recovery, without unexpected outages.

- Software Upgrade: Changes in software of a component, or block, in any layer, could cause problems to the other layers. This could impact existing services and their users. Upgrades must be checked before applying and there should be a roleback procedure in case of a failure upgrade.
- Security: Both the NFVI physical and virtual resources should be protected. Plus, processes, policies and practices.
- Heterogeneity: NFV platform must be open to run applications from different vendors, also ISPs should have the flexibility to choose from different hardware and software.
- Automation: Automation of processes is important to the success of NFV. In order NFV to succeed that automation should remain and expand. [15]

4) OSM MANO Project

4.1) OSM History

OSM was originally design by operators to coordinate the production of NFV networks. The project, meeting the standards of ETSI NFV, primarily distributes an open source Management and Orchestrator (MANO) stack. It is worth mentioning that it was released in Github (Git-repository hosting service) under Apache 2 license.

Following the NFV Framework-History section, it's worth mentioning that its publications have moved from pre-standardization studies to detailed specifications (Release 2 and Release 3) and the early Proof of Concepts (PoCs) efforts have evolved and led to interoperability events (Plugtests). Community is still working intensely to develop the required standards for NFV as well as sharing their experiences of NFV implementation and testing. [16]

Telefónica, BT, Canonical, Intel, Mirantis, RIFT.io, Telekom Austria Group, and Telenor are some of OSM's founding members while its initial participants include Benu Networks, Brocade, Comptel, Dell, Indra, Korea Telecom, Metaswitch, RADWare, Red Hat, Sandvine, SK Telecom, Sprint, Telmex, xFlow and 6WIND.

4.2) Why OSM?

The author of this thesis found the operating principals fascinating. The open source principal and the community model that all parts are helping with testing and bug fixes making the project open for individuals as us to get familiar with, and companies to adapt their current network in a same horizontal procedure as a standard.

As mentioned above the OSM Project is a wide-open community driven by Service Providers requirements and supported by key players in the industry of virtualization space. OSM MANO embraces the complexity required for deployments in field. At real time and real scenarios. With EPA support, Multi-VIM operation, Multi-site activity, and the detachment of RO and SO. It also provides a CLI and GUI configuration tool, plus is multi-vendor making it friendly for network engineers. Moreover, OSM MANO has a comprehensive set of L2 network connections. This combination hides the low-level complexity to network engineers while assures consistent deployments.

All the above make the specific project simple, easy to use and adapt to current network challenges.

4.3) Scope of OSM

Current MANO approaches and Static/ Traditional Operators are focused partially either on the L3 and part of L2 process or either on Layer 2 and Layer 1., but never through the above Layers, and cloud operations are involved with Layer 0. OSMs scope is to cover all the Layers from 3 into 1 with the ability to talk as an entity with Cloud Operation. Also delineates a clear movement between the layers and modules, broadly aligned with ETSI-NFV.

The scope of the OSM model can be summarized into four Architectural principals alongside with 4 Layers show and analyzed below.

	NETWORK CREATION	FULFILLMENT	ASSURANCE
(L3) SERVICE OPERATION (Chaining of VAS & self-care portal)		• Add users to VPN • Add and chain VAS to VPN (self-care)	QoE monitoring
(L2) SERVICE MANAGER (VPN service)	VPN service design	VPN deployment	VPN service monitoring
(L1) NETWORK DEPARTMENT (Network core)	Network scenario creation & deployment (PE per site)	Network scenario provisioning	Network scenario monitoring
(L0) NFVI OPERATOR	Installation of switches & servers	• Tenant creation. • Allocation of tenant quotes.	• Monitoring of usage of resources by tenants. • Monitoring of NFVI infrastructure

Table 2. Layers of the NFV approach

Layering: real operation is multi-layered by nature, but current mano approaches are partial.

Abstraction: The new challenges in the Network Industry require a flexible and a scalable network. There is a need for abstraction to provide the originally intended independence that will allow networks to scale as required. OSM is trying to provide that required abstraction through the network, moving up/down the layers offering clear differentiation in the levels of abstraction/details presented.

Modularity: Two of the key components of the ETSI NFV architectural framework are the NFV Orchestrator and VNF Manager, known as NFV MANO. Service orchestration is also required for operators to enable true NFV services. Open Source software can make the application of an NFV Architecture (following ETSI standards) easier, supply the ETSI ISG NFV with crucial feedback, while making NFV implementations' interoperability all the more likely. Even within layers, clear modularity enabled with plugin model preferred to facilitate module replacements as OSM community develops.

Simplicity: OSM says that it has a one click installation and also that the big community supporting the OSM with seeding codes can hide the complexity around the project helping network engineers focus on their activity. Plus, GUI/CLI tools helping the user configuration.

4.4) OSM Mapping to ETSI NFV MANO

In this section we are going to see how the ETSI NFV Model is being incorporated into the OSM Model.

In the below image is illustrated the approximate mapping of scope between the OSM components and the ETSI NFV MANO logical view.

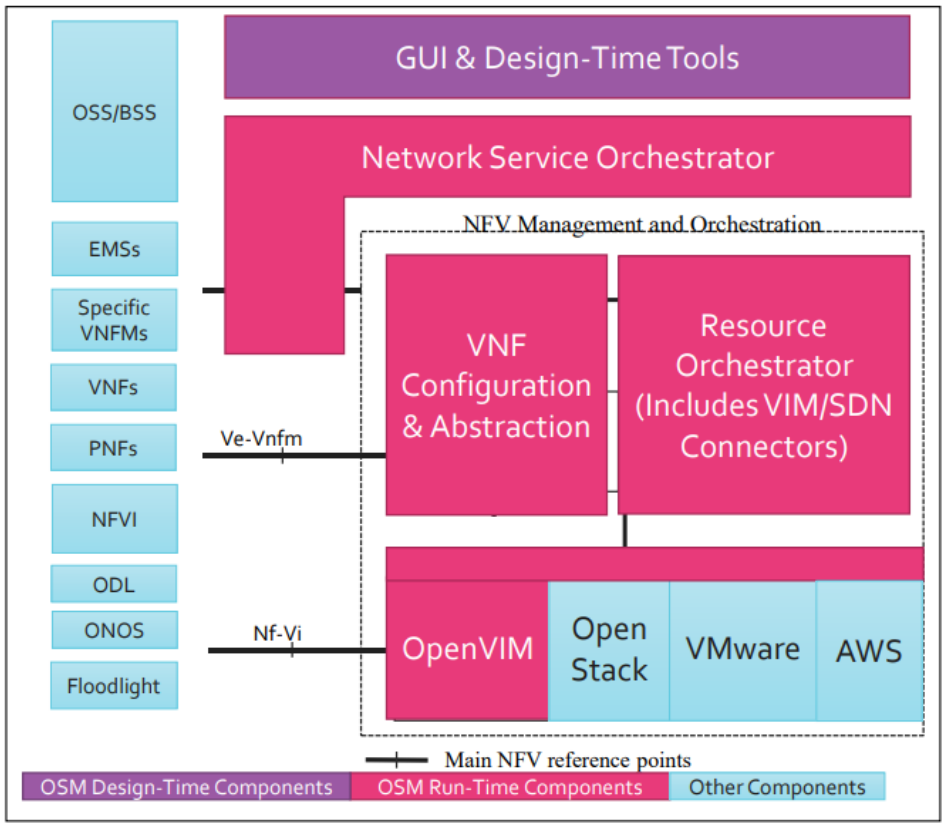


Image 13. OSM Mapping to ETSI NFV MANO
[OSM release three – white paper]

Run-Time Scope

The run-time scope of OSM includes:

- An Automated Service Orchestration environment that permits and makes the functional concerns of the multitude of lifecycle phases of a NFV based complex service, easier.
- A superset of ETSI NFV MANO where the significant supplementary area of the scope includes Service Orchestration and provision for SDN control.
- Provision of a plugin model for the incorporation of various SDN controllers.
- Provision of a plugin model for the incorporation of various VIMs.
- A reference VIM, developed for Enhanced Platform Awareness (EPA) so as to allow high performance VNF deployment.
- An incorporated “Generic” VNFM, also supporting a “Specific” one.
- Provision for Physical Network Functions incorporation into an automated Network Service deployment.
- Supporting both Greenfield and Brownfield scenarios.

- A variety of tools and interfaces (GUI, CLI, Client and REST) allowing access to all features.

Design-Time Scope

The design-time scope of OSM includes:

- Network Service Definition, CRUD operations (Create/Read/Update/Delete).
- Model-Driven environment with Data Models aligned with ETSI NFV
- VNF Package Generation simplification.
- A Graphical User Interface (GUI) to reduce the network service design time. [17]

4.5) MANO Software Components

We previously covered the functions of VIM, NFVO and VNFM. In order to analyze the structure of MANO we have to refer some other chains and functions. The sections of Interfaces use and characteristics, EPA attributes, YANG models, NSD and VNFD Descriptors will follow in order to get familiar with the functionality of OSM.

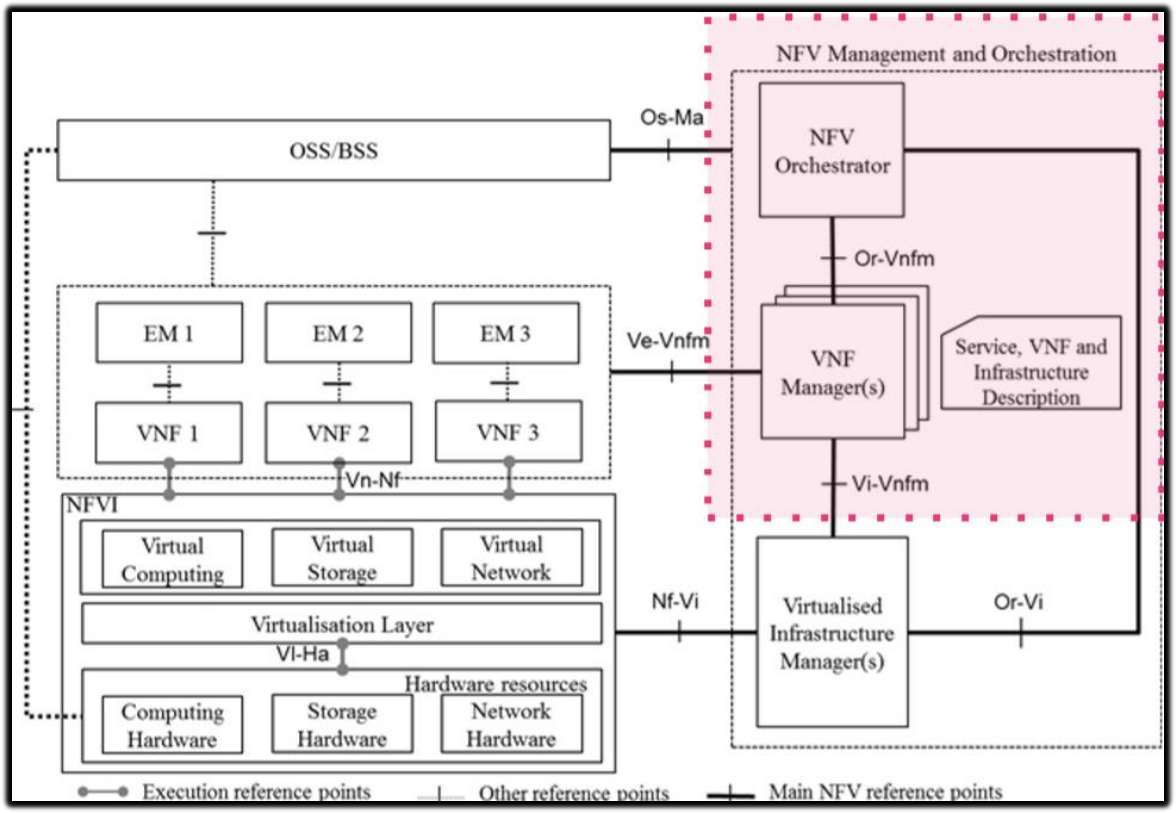


Image 14. The NFV MANO Architecture
[https://wiki.sdn.ieee.org/pages/viewpage.action?pageId=65780]

4.5.1) Interface

The Os-ma-nfvo (Os-Ma in image 14) interface is designed with open, standards-based APIs, such as NETCONF and REST, and common information models, such as YANG. This specific layout allows upper-level orchestrators, such as Business Process Orchestrators or Service Orchestrators, to automate the entire service bring up process.

4.5.2) EPA

In an architecture such as a legacy, chassis-based deployment, network function suppliers have chosen a specific CPU for the network function guarantying the bandwidth and latency across the chassis. On the contrary, Datacenter Architecture Virtual machines may be allocated in several physical hosts anywhere within the same datacenter, and both the host and the physical links between these hosts can be oversubscribed. As on the physical host the CPU cores of a specific virtual machine may belong to different sockets, this could result to memory access and cache issues. The aforementioned divergence could cause VNFs with performance characteristics differences to emerge. Enhanced Platform Awareness supported by OSM can solve these issues. The Virtualized Infrastructure Manager during the initial allocation of virtual machines can discover EPA attributes. Among the VNF instantiation process, VNF request characteristics are compared to the abilities of the virtual machine, in order to allocate workload placement across the corresponding VMs. This design also supports placements such as:

- High data rate workloads, such as load balancing, hugepage setup, CPU pinning, and PCI pass through.
- Best-effort workloads, such as statistics gathering or log output.
- Workloads that form part of the same network service (same service chain) in the same switching domain.
- Distributing workloads, such as firewalling, DHCP, or other premise-related tasks, to a remote customer premise device.
- Advanced security capabilities, such as Quick Assist Technology (QAT) crypto assist and Trusted Platform Module. [18]

4.5.3) Network Service Descriptor (nsd:nsd)

Used for designing the service chains, the network service descriptor (NSD) is the top-level constructor. The NSD is used by the NFV orchestrator to instantiate a network service, consisting of static information elements. Also, it has four elements:

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

The NSD references one or more VNFDs. These VNFs are connected VLDs, and the traffic flow in the service chain is determined by VNFFGD. The NSD also opens a number of connection points, permitting connections to other network services.

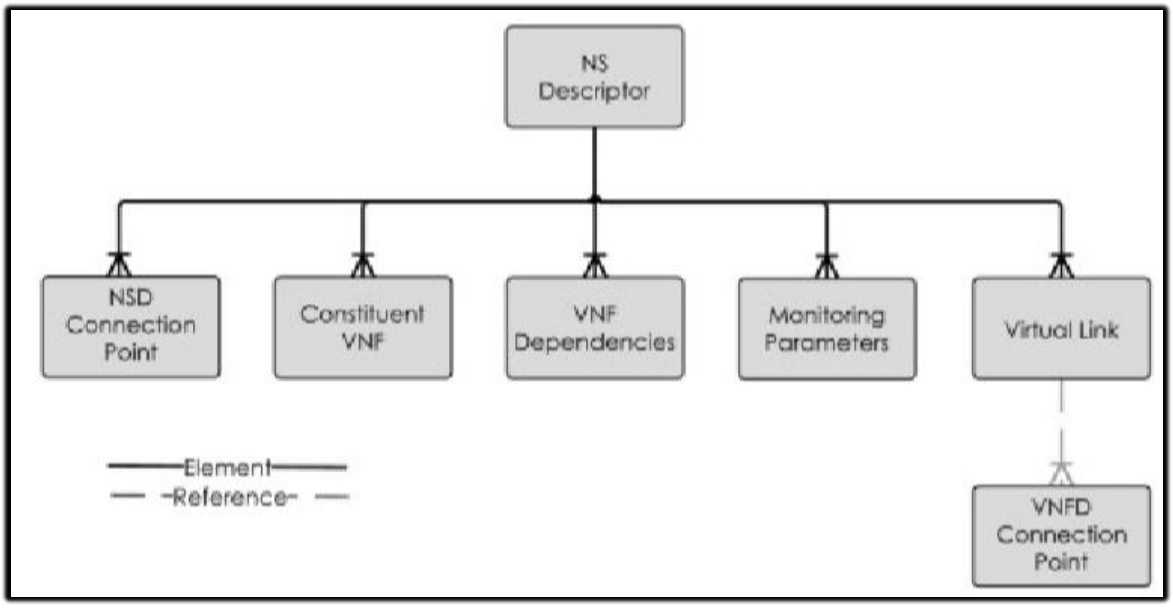


Image 15. High Level Object Model for NSD [OSM information model-https://osm.etsi.org/wikipub/images]

NSD Data Model**:** Catalog for the network service descriptor.

nsd:connection-point : list of references to network service connection points.

nsd:constituent-vnfd: list of Virtual Network Function Descriptors (VNFDs) that are part of this network service.

nsd:scaling-group-descriptor: Scaling group descriptor within this network service, defines a group of VNFs

nsd:placement-groups: list of placement groups at the network service level.

nsd:ip-profiles: List of IP profiles that characterize the IP parameters for the virtual link.

nsd:vnf-dependency: List of VNF dependencies of the Network Service Descriptor (NSD).

nsd:monitoring-param: List of monitoring parameters from VNFs to communicate with the Network Service Record (NSR).

nsd:input-parameter-xpath: List of customizable (during instantiation) XPaths to parameters within the Network Service Descriptor.

nsd:parameter-pool: Pool of parameter values whence to draw during network service configuration.

nsd:service-primitive: Network service-level service primitives for the Network Service Descriptor.

nsd:initial-config-primitive: NSD initial set of configuration primitives which are implemented at network service.

nsd:terminate-config-primitive: List of configuration primitives, implemented before shutting down the network service.

nsd:cloud-config: NSD cloud configuration parameters to include a list of public keys user wants to inject into each VM as part of network service instantiation. [18]

4.5.4) Virtual Network Function Descriptor

The virtual network function descriptor (VNFD) is a deployment template that describes the attributes of a single VNF, used primarily by the VNF manager (VNFM) in the process of VNF instantiation and lifecycle management of a VNF instance. Provided information in the VNFD is also used by the NFV orchestrator (NFVO) to manage and orchestrate network services and virtualized resources on the NFV infrastructure (NFVI).

VNFD also contains of the followings:

- VNF images, containing the application and the Launchpad.
- Connection points and virtual links, KPI requirements and interfaces, utilized by MANO functional blocks to provide the appropriate virtual links.
- Virtual deployment unit (VDU) that determine the VM/VNFC requirements (compute, storage, and network).
- Platform resource requirements, (CPU, interfaces, memory, and network).
- Special characteristics associated with performance as well as EPA attributes.
- Scaling properties.

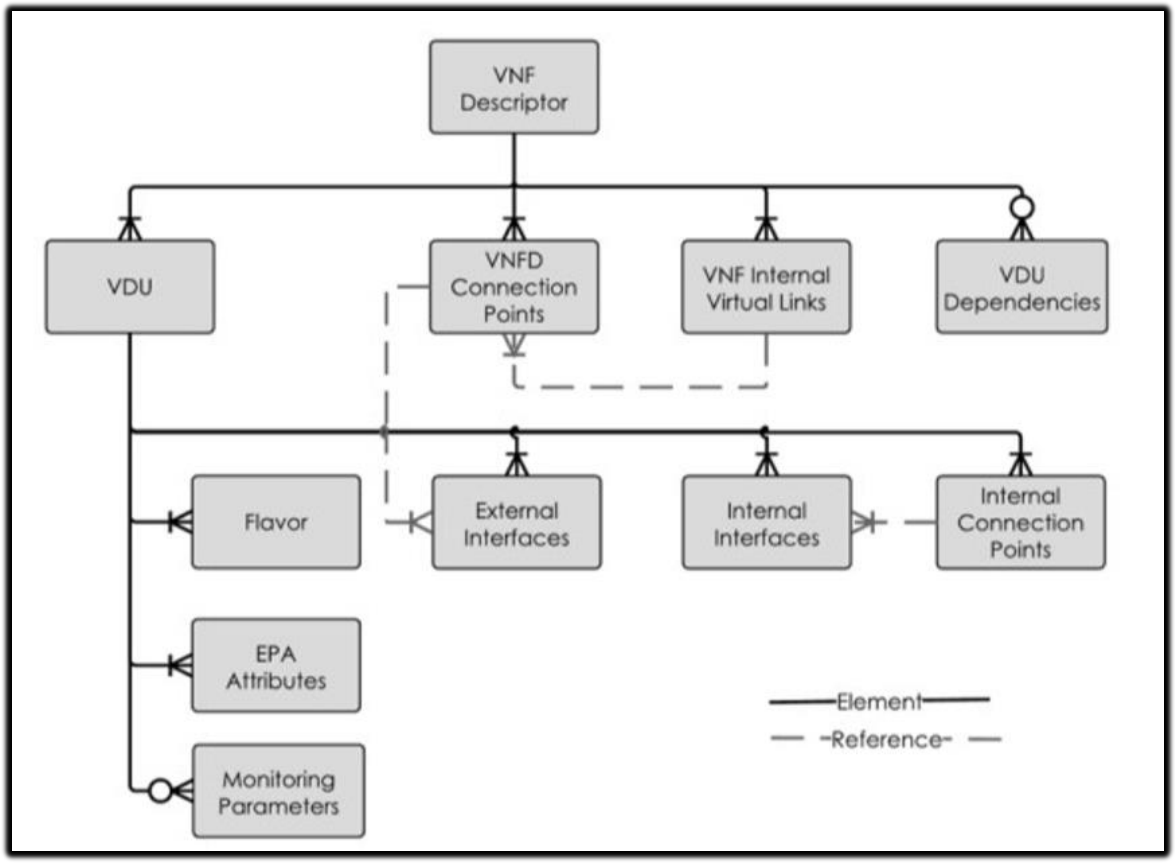


Image 16. High Level Object Model for VFND [OSM information model- https://osm.etsi.org/wikipub/images]

The above image figures the High-level object model for VFND. The VNFD contains sets of VDUs, internal and external connection points as well as internal virtual links. The internal aspects (connection points and virtual links) characterize the connections of the VMs inside the VNF. On the other hand, NSD uses the external connection points in order to chain VNFs. Apart from getting information about EPA attributes, VM image and flavor, the VDUs also define the individual VNF parts.

During the process of VNF instantiation and during lifecycle management of a VNF instance, VNFM uses a VNFD. Analytically, the information that is provided in the VNFD is been used also by the NFVO to orchestrate and manage virtualized resources and network services on the NFVI. Elements in the VDU (Virtual Deployment Unit) determine software components and the compute resources. The VDU handles information about software components, networking resources, CPU, memory and storage in the VM. So as, any individual VNF has external and internal connection points, abstracting the virtual interface used by the container/ VM, each virtual interface inside the VM has assigned a connection point. In order to connect VNFC internals to VNF, internal connections points are being used. These connection points, (internal or

external), are connected using virtual link and each virtual link has references, two or more connection points as seen in the above image. [18]

VNFD has also EPA capabilities captured in the VNFD Virual Deployment Unit. Those elements in summarization are:

- Hugepages: The CPU marks the RAM used, by a process, consuming memory. Many platforms default allocate value as frames of 4K bytes RAM. Those frames of bytes are being called pages and can be swapped to disk, etc. Since the Operating Systems and the CPU has to remember everything, from which page to the stored place, as process address space is virtual, it is understandable that the more pages to process, the more time is needed to locate the mapped memory. A process of 2GB of memory consumption, is translated to 524288 entries to look up (2GB / 4K), and if one Page Table Entry consume 8bytes, that's approximately 4MB (524288 * 8) to look-up. [19] Without standard 4k pages and the use of hugepages we can improve network performance. Fewer pages are needed and there are fewer Translation Look-aside Buffers (TLBs, high speed translation caches).

- CPU Pinning: Processor affinity, or CPU pinning, is called the process that enables the binding or unbinding a process to a specific central processing unit (CPU) or ranges of CPUs. These processes will be executed on the pinned CPU or CPUs and not in any CPU available. At the time of resource allocation, each item in the queue, has a tag indicating its designated processor. [20] Often in OpenStack deployments, hosts are configured to permit over-commit of CPUs.
  Default scheduler in Openstack attempts to launch a maximal number of VMs per host rather than optimizing the individual VM's performance, leading to conflicts occurring between two guests VMs. There could be extended periods when the guest vCPU is not scheduled by the host, and thus leading to latency. To avoid a latency issue, the guest should be pinned to a dedicated physical CPU.

- Guest NUMA Awareness: The NUMA (Non-uniform memory access) topology and CPU pinning features in OpenStack provide high-level control over how instances run on hypervisor CPUs and the topology of virtual CPUs available to instances. These features help minimize latency and maximize performance. In the traditional symmetric multiprocessing (SMP) architectures, all of the available CPUs access the same memory. With the growing of CPU, and as modern CPUs are faster than the memory they use, the memory bus starts to have a bottleneck effect, as data remain in the queue to be processed, not feeding the CPU. In contrast with the above, in a NUMA system, memory is

separated into multiple memory nodes (or cells), associated with particular CPUs. Ensuring that all memory accesses are local to the NUMA node and can be accessed faster than the other memory nodes, thus avoiding latency. [5] When running workloads on NUMA hosts, the CPUs executing the processes should be on the same node as the memory used. [21]

- PCI Pass-Through: The PCI device passthrough capability allows a physical PCI device, from the host machine, to be assigned directly to a guest machine. The guest OS drivers are using the device hardware directly without relying on any driver capabilities from the host OS. [22] Guest virtual machines might need direct access to the PCI devices to avoid contention with other VMs. In addition, PCI pass-through significantly improves performance since the hypervisor layer is bypassed.

- Data Direct I/O: Direct I/O is a feature that supports direct reads/writes from/to storage device to/from user memory space bypassing system page cache. By most operating systems, buffered I/O is, in most cases, the default I/O mode enabled. [23] In the specific case, Intels' DDIO makes the processor cache the primary source and destination of the I/O data rather than main memory, helping to deliver lower latency, lower power consumption and increased bandwidth. [24]

- Cache Monitoring Technology: Intel's Cache Monitoring Technology (CMT) provides visibility into shared platform resource utilization (via L3 cache occupancy), which enables improve application profiling, better scheduling, improved determinism, and improved platform visibility to track down applications which may be over-utilizing shared resources and thus reducing the performance of other co-running applications. [25]

- Cache Allocation Technology: Intel Cache Allocation Technology (CAT) allows an operating system, hypervisor, or similar system management agent to specify the amount of L3 cache space an application can fill. [26]

VNFD Data Model **(vnfd:vnfd):** Descriptor details for the Virtual Network Function (VNF).

vnfd:vnf-configuration: Holds information about the VNF configuration for the management interface.

config-method:script: Script container for configuring the VNF. This script will be executed in the Launchpad, and all required dependencies for the script should be available in the Launchpad system.

config-method:juju: Juju container for configuring the VNF.

service-primitive:parameter: List of parameters to the primitive.

vnfd:mgmt-interface: Interface over which the VNF is managed.

vnfd:internal-vld: List of internal Virtual Link Descriptors (VLDs). The internal VLDs detail, basic topology connectivity (E-LAN) between VNFC internal components, within the system.

vnfd:ip-profiles: List of IP profiles that describe the IP characteristics for the virtual link.

ip-profile-params:dns-server: List of DNS servers associated with this IP profile.

ip-profile-params:dhcp-params: Container for DHCP parameters.

vnfd:connection-point: List of external connection points, in which each VNF,has one or more points that are used to connect a VNF to other VNFs or to external networks and exposes these connection points to the orchestrator (NFVO).

vnfd:vm-flavor: Describes another term for a VM instance.

vnfd:guest-epa: Describes geust OS EPA attributes.

vnfd:vswitch-epa: Describes Open vSwitch EPA attributes.

vnfd:hypervisor-epa: Describes Hypervisors EPA attributes.

vnfd:host-epa: Determines host-level EPA attributes.

vnfd:alarm: Describes alarm information.

vnfd:image-properties: Supplies checksum file and image name of VM during the launch.

vnfd:internal-connection-point: Describing a list of internal connection points that are used to connect the VNF components internal to the VNF. Each VNFC has zero to more internal connection points.

vnfd:internal-interface: Describing a list of internal interfaces for VNF enabling intra-VNF traffic.

vnfd:external-interface: List of external interfaces for VNF enabling intra-VNF traffic.

vnfd:volumes: Defines disk volumes to be attached to the VDU, such as when a VNF requires multiple disks to boot the virtual machine.

vnfd:vdu-dependency: List of virtual deployment unit (VDU) dependencies, from which the orchestrator determines the order of startup among the VDUs.

vnfd:monitoring-param: List of monitoring parameters at the VNF level.

vnfd:placement-groups: List of placement groups at VNF level. The placement group construct defines the compute resource placement strategy in a cloud environment.

VDU Data Model **(vnfd:vdu)**

Another key part of VNFD is VDU. VDUs are virtual machines that host the network function. Some of the functionalities involved in the processes of a VDU are the above.

- Virtual machine specification
- Computation properties such as number of CPUs, number of cores per CPU, number of threads per core, RAM size, disk size and memory page size.
- Storage requirements
- Initiation and termination scripts
- High availability redundancy model
- Scale out/scale in limits

4.5.5) Virtual Link Descriptor **(nsd:vld)**

A virtual link descriptor (VLD) is a deployment template that describes the resource requirements needed for a link between VNFs, PNFs and endpoints of the network service, that are available in the NFVI. The NFVO can select an option after evaluating the VNFFG to determine the appropriate NFVI to be used based on functional and other

needs such as regulatory requirements and geographical needs. As always in ETSIs MANO, network connections are defined by virtual links plus its connection points, described as below.

- Connection of a network service to the outside world, such as the network service endpoint, described in the NSD.
- Connections between VNFs within a network service, such as the external interface of the VNF, described in the VNFD.
- Connections between VMs, described in the VNFC.

Also, we have to mention that there are two types of virtual links in the VLD, containing Quality of Service and bandwidth information requirements of the interconnection.

- The internal virtual links, which can be connected to external VNF interfaces and VNFCs.
- The external virtual links, which can be connected to network service endpoints and external VNF interfaces

4.5.6) VNF Forwarding Graph Descriptor **(nsd:vnffgd)**

Specified by a network service provider, a virtual network function forwarding graph (VNFFG) is a graph of bi-directional logical links that connect network function nodes, where at least one node is a VNF through which network traffic is directed. The VNFFG model is defined at the network service level. One or more VNFFG descriptors can be defined in the network service descriptor (NSD).

A VNFFG model consists of a list of rendered service path (RSP) and list of classifier components.

Rsp: List of the Rendered Service Paths (RSP) for the VNFFGD.

rsp:vnfd-connection-point-ref: List of references to connection points.

classifier:match-attributes: A list of packet filters that identifies the packet stream to be fed to the RSP.

4.5.7) MANO YANG Models

YANG is a data modeling language used to design configuration and state data manipulated by the Network Configuration (NETCONF) Protocol [RFC 6241], NETCONF remote procedure calls, and NETCONF notifications.

nsd.yang Model

The nsd.yang file defines the Network Service Descriptor (NSD), the top-level deployment of a network service. The nsd module contains attributes for a group of network functions, which together constitute a service definition. These attributes contain the relationship requirements of the VNFs which are chained together as a service. The NSD references one or more VNFDs, as well as other descriptors that are used for designing the service chains. [27], [28].

4.6) OSM Low Level Analysis

In this section we will analyze the Low-Level design of OSM MANO Architecture and key software components of its function. Introducing the UI, RO, SO, VCA and their operation.

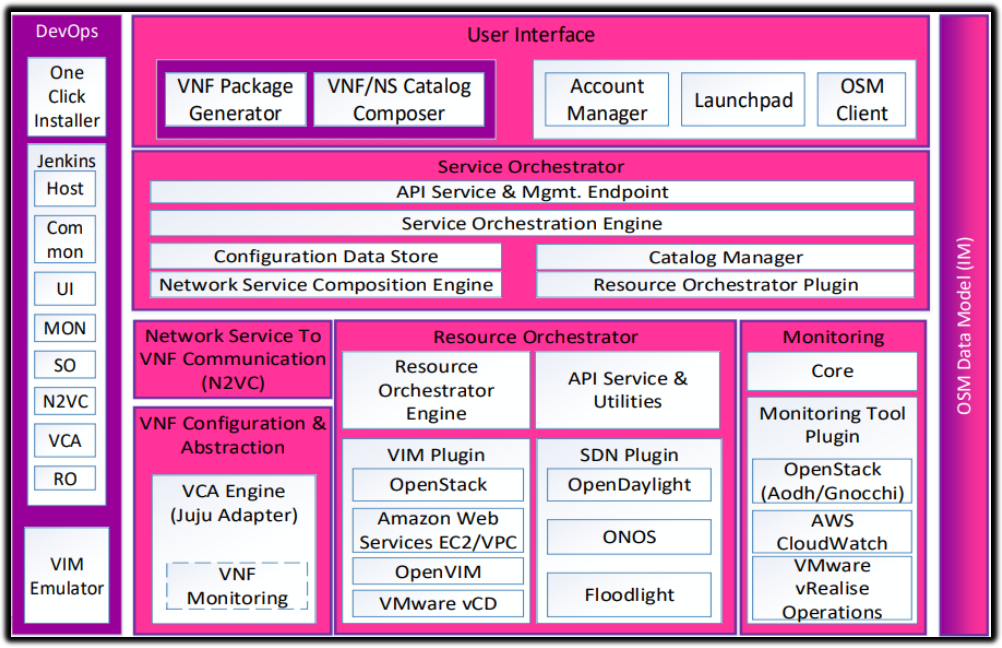


Image 17. OSM Architecture

[OSM release three – white paper]

In the above image we can see the analytic OSM Architecture. With the User Interface, Service Orchestrator, Resource Orchestrator, DevOps, Monitoring Tools deployed, and the OSM data model (IM).

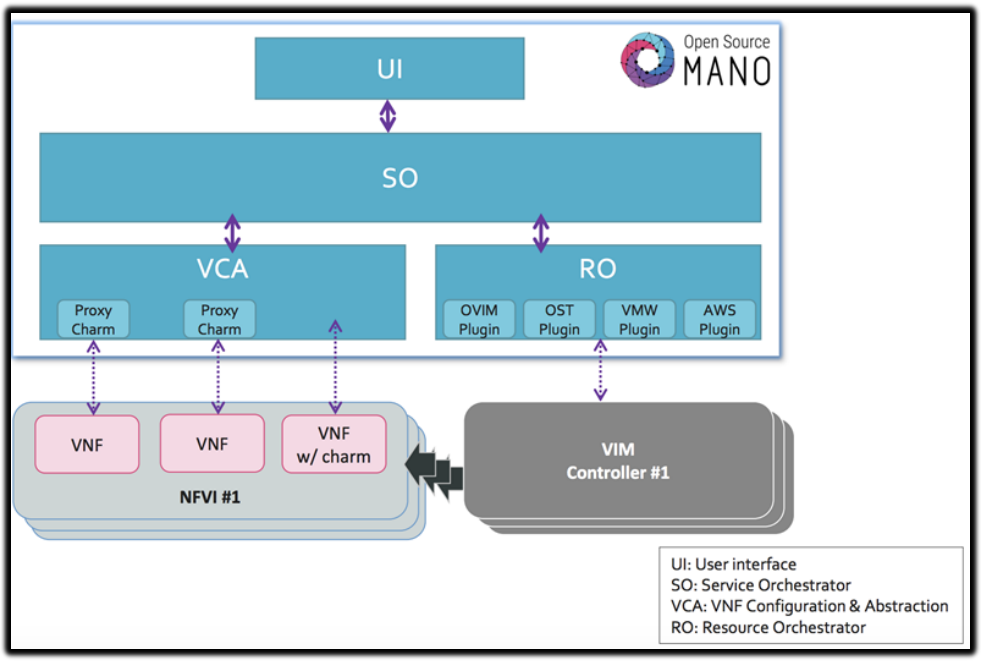


Image 18. Today's OSM simplified Architecture
[OSM release three – white paper]

In the image above, we can see the simplified OSM Architecture, containing the OSM scope (UI, SO, VCA, RO) and the VIM and NFVI (outside of OSM scope). User from CLI or GUI can interact with the Service Orchestrator, and vice versa. Service Orchestrator interacts with VCA and RO.

Resource Orchestrator is responsible for providing the interface into RO and for deployment and the interconnection with required resources, interacting with Openvim Controller and OpenStack controller. Also, it provides accurate assignment of resources at VM level and the proper assignment of interfaces I/O to the VM.

VNF Configuration and Abstraction is handling the VNF Modeling and Configuration through proxy charms, with primitives and attributes. It also responsible for actions and notification from and to the VNFs and Element Managers (EM). When assisted by Juju, it gives the facility to create generic or specific indirect-mode VNFMs, via charms that can support the interface the VNF/EM chooses to export.

Lastly Service orchestrator is handling the end to end services. Providing the primary API endpoint into OSM. Handling lifecycle management and service execution. It is also responsible of supporting the concept of multi-tenancy, project, users, and role-based access control. [29]

4.7) OSM MANO: openMANO

Part of OSM MANO project, was an open source project released in GitHub under Apache 2 license. It is a python-based code, with 45k code lines and 38 forks.

Is a practical implementation of the reference architecture for Management & Orchestration (MANO) under standardization at ETSI's NFV ISG. OpenMANO follows an NFVO-centric approach, granting the deterministic allocation of resources and with a simplified VNF instance lifecycle management at the NFVO (VNF instantiation and termination). OpenMANO is friendly for Network engineers and has embeded network scenarios. It provides NFVO and VIM with CLI and GUI, supporting EPA-aware High performance VNFs. Is has REST-BASED APIs and is OpenStack friendly. It also has multi-vendor capabilities by design. OpenMANO works with Network Scenarios via descriptors, providing enhanced platform awareness (EPA) natively.

4.8) OSM MANO key Components

The following section covers a more in depth of the characteristics of OSM. Analyzing the configuration available through Python CLI and GUI interface. Covering how every section of component is communicating with each other.

Openmano was composed by 3 software modules.

- Openvim (Server and client)
- Openmano (server and client)
- Openmano-gui (web interface)

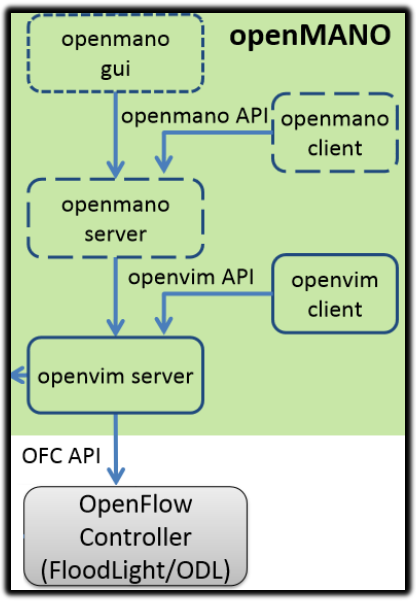


Image 19. OpenMANO components and logical interconnections
https://osm.etsi.org/wikipub/images/2/OSM(16)000020_MWC_demo_components__OpenMANO.pdf]

Openmano is responsible for the tenant and the datacenter management, VNF catalogue management, Network scenarios catalogue management, the Network Services deployment and the VNF, and also is responsible simplified VNF life cycle management. While Openvim computes node management, the management of NFVI tenant, images, flavors, Network and ports. Virual Machines are deployed with EPA management, natively and bridged layer 2 networks.

4.8.1) OPENVIM: the VIM module

Openvim was originally part of OpenMANO. It was an open source project implementing ETSI MANO stack, such as VIM (openvim), NFVO+VNFM (openmano) along with GUI (openmano-gui). With the creation of ETSI Open Source MANO (OSM), the NFVO+VNFM (openmano) was contributed to OSM as seed code. [30] Openvim with the interaction of OpenFlow Controller (floodlight/OpenDay Light) synthesize the NFV VIM. Openvim is responsible for the interaction with compute nodes through libvirt. It is tested on compute nodes based on Intel Xeon E5 processors, LINUX as host Operating System, KVM as hypervisor. Also, we have to mention here that Openflow switch is controlled by proactive rules and that image storage is based on NAS.

Openvim has 5 modes to be runed as illustrated below.

MODE	Purpose	Required Infrastracture
Normal	Regular operation	Compute nodes OpenFlowswitch
Host only	Deploy without OpenFlowswitch and controller	Compute nodes
Development	VNF development (deploys withoutEPA)	Low Performance compute node
Test	Test openMANO installation and API	-
OF only	Test openflowintegration	Openflow Switch

Table 3. Modes of Openvim Configuration. This thesis testing installation was run into test mode.

Openvim Main Characteristics

- Host Management: Host addition is done manually through a host descriptor file and hosts can be administratively set up or down.
- Tenant Management: Tenants delimit the property and scope of flavors, images, vms, nets.

- Network Management: Networks are pure Layer 2 networks. With PtP used to create E-line service between two data plane interfaces, data used to create E-LAN service with data plane interfaces and bridge-data, used to create an E-LAN service based on pre-provisioned linux bridges.
- Port management: The ports are attached to networks similar to OpenStack. There are two types of ports. Instance-related ports, with VM interfaces created and deleted as part of the VM life cycle and External Ports, set explicitly by the network administrator in order to define connections to PNF or external networks physically attached to the Openflow switch.
- Image Management: Disk images to be use for a virtual machine. supports of incremental images.
- Flavor management: Are a description of virtual machines requirement regarding number of CPUs, memory and NICs.
- VM instance management: Besides traditional actions (create, delete, list), allows actions over VMs (shutdown, start, pause, resume, rebuild, reboot)

Below you can find an output of the openvim usage through the CLI, and the configure choices available.

$ openvim -h usage: openvim [-h] [--version]

{config,image-list,image-create,image-delete,image-edit,vm-list,vmcreate,vm-delete,vm-edit,vm-shutdown,vm-start,vm-rebuild,vm-reboot,vm-createImage,portlist,port-create,port-delete,port-edit,port-attach,port-detach,host-list,host-add,hostremove,host-edit,host-up,host-down,net-list,net-create,net-delete,net-edit,net-up,netdown,flavor-list,flavor-create,flavor-delete,flavor-edit,tenant-list,tenantcreate,tenant-delete,tenant-edit,openflow-port-list,openflow-clear-all,openflow-netreinstall,openflow-net-list}

4.8.2) Openmano / OSM client.

As we analyze previously, openMANO has the characteristics of hiding the complexity to the network engineers, as it does not handle or computes nodes, VMs and manages only nodes and links. Computes VNF and NS definitions via descriptors. Additionally, operates NS instances creation and termination and the associated VNF creation.

In the matter of tenant management, which intended to create groups of resources and delimit the property and scope of VNF and NS, and the actions over them (instantiation, termination), are separate from the openvims tenant space. They are handled by different programs with different databases.

Concerning the Datacenter management, a new datacenter must be added in order to interact with a specific pool of resources. Datacenters are not directly available to tenants, so an openmano tenant must be attached to a datacenter and a VIM tenant, therefore datacenter nets can be inherited as external networks to be used. A datacenter is characterized by the type, openvim (by default) or openstack, the URL of the VIM that manages the specific datacenter, and the VIM configuration attributes.

4.8.3) Structure / Concept of the VNF.

As we previously stated a VNF is a software-based network function that can be deployed on an NFV datacenter, with is structure VNFCs/VMs, plus internal and external connections. In the below image we can see an example of a single VM VNF and a multi VM VNF.

4.8.4) OPENMANO VNF Descriptor

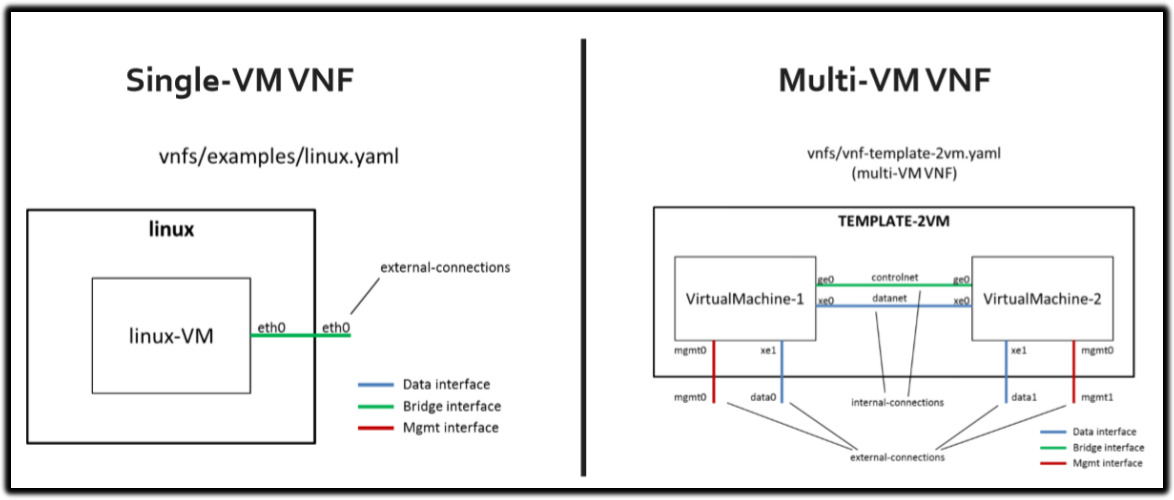


Image 20. Example of a single VM VNF and a multi VM VNF
[https://image.slidesharecdn.com/2-150616082339-lva1-app6892/95/introduction-to-open-mano-27-638.jpg?cb=1434444766]

The VNF descriptor is categorized by:

- Name: Unique name of the VNF

- External connections: External interfaces of that VNF that can be connected in an NS to other VNFs or networks, properties: name, type, such as management/bridge/data and mapping to a VNF structure)
- Internal connections: Defining how VNFC/VMs are interconnected. This attribute is only required when a VNF consists several VMs. Properties: name, type, such as management/bridge/data and a list of interconnected VNFC/VMs (and their interfaces).
- VNFC: List of components virtual machines this VNF is composed of.
- VNFC properties: name and the image path. At the time of a new VNF is added to the catalogue, openvim creates new VM images for each given VNFC, based on that particular part.

VNFCs properties are categorized in the traditional requirements such as

- vcpus: number of virtual CPUs
- ram: number of virtual CPUs
- bridge-ifaces: virtio interfaces with no high I/O performance requirements. They will be attached to Linux bridges in the host.

And the EPA properties requirements:

- numas: CPU, memory and interface requirements for high I/O performance.

Another key component of mano is the Network Service Descriptor.

NS are topologies of VNFs and their interconnections external and internal.

Here is an example of a simple network scenario and a complex with 3 nodes, and their interconnections.

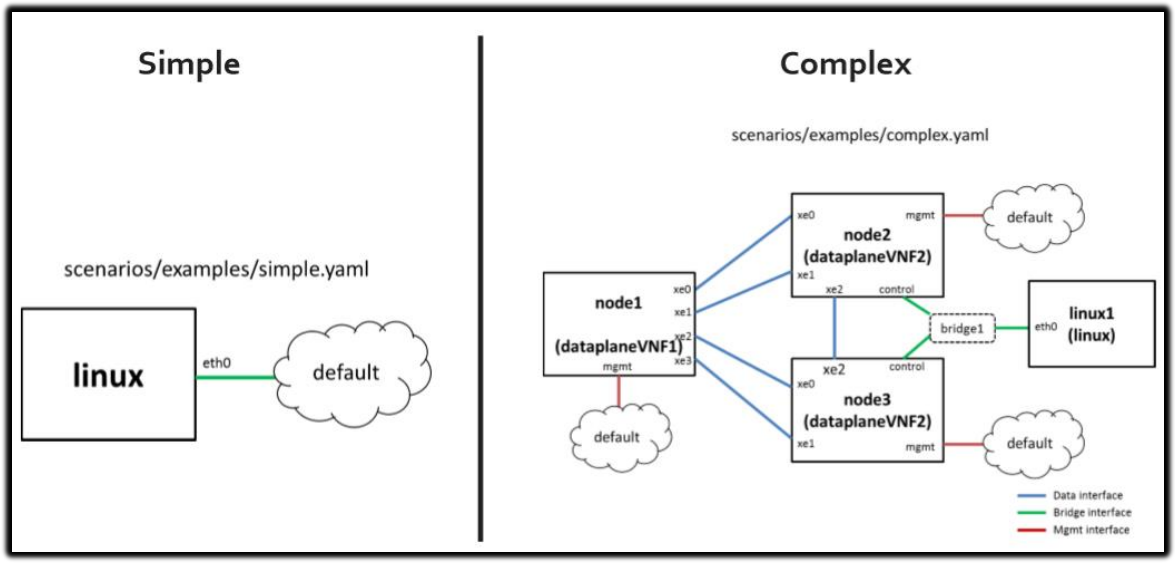


Image 21. Example of a simple and complex NS scenario
[https://image.slidesharecdn.com/2-150616082339-lva1-app6892/95/introduction-to-open-mano-27-638.jpg?cb=1434444767]

Openmanos' Network Service Descriptor is categorized by:

- name: unique name of the network scenario.
- Topology: defines the VNFs and the networks interconnecting them.
- VNF: name and VNF model (id or name) and match a previously created VNF.
- Netwoks: name
- Type of network: name of the network (in case of external/public datacenter network), bridge (for control plane internal/private networks), dataplane (for data plane internal/private networks).
- List: list of VNFs and interfaces connected to that network.

Below you can find an output of the openmano usage through the CLI, and the configure choices available.

$ openmano -h usage: openmano [-h] [--version] {config,vnf-create,vnf-list,vnf-delete,scenario-create,scenariolist,scenario-delete,scenario-deploy,scenario-verify,instance-scenario-list,instancescenario-delete,tenant-create,tenant-delete,tenant-list,tenant-edit,datacenteredit,datacenter-create,datacenter-delete,datacenter-list,datacenter-attach,datacenterdetach,datacenter-net-edit,datacenter-net-update,datacenter-net-delete,datacenter-netlist}

4.8.5) OSM GUI

OSM mano comes with a graphical user interface to accelerate the network service design time phase and deploy the VNF. It can be accessed via Firefox or Chrome (default). GUI characteristics are:

- Access to network scenario definitions and instances.
- Drag and drop scenario builder with access to the VNF catalogue
- Actions between instances over a network scenario VNF, into a NS (stop, shutdown, delete, deploy) and over specific VNF instances.

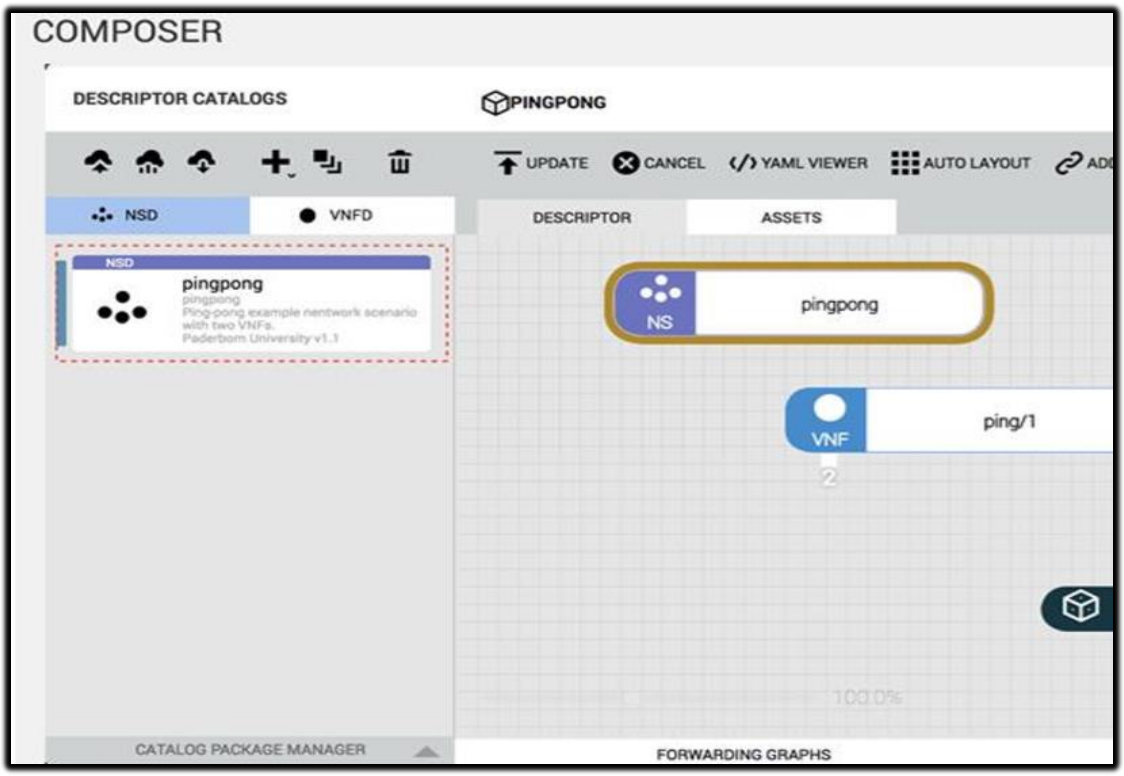


Image 22. OSM GUIs composer, containing NSD and VNF.

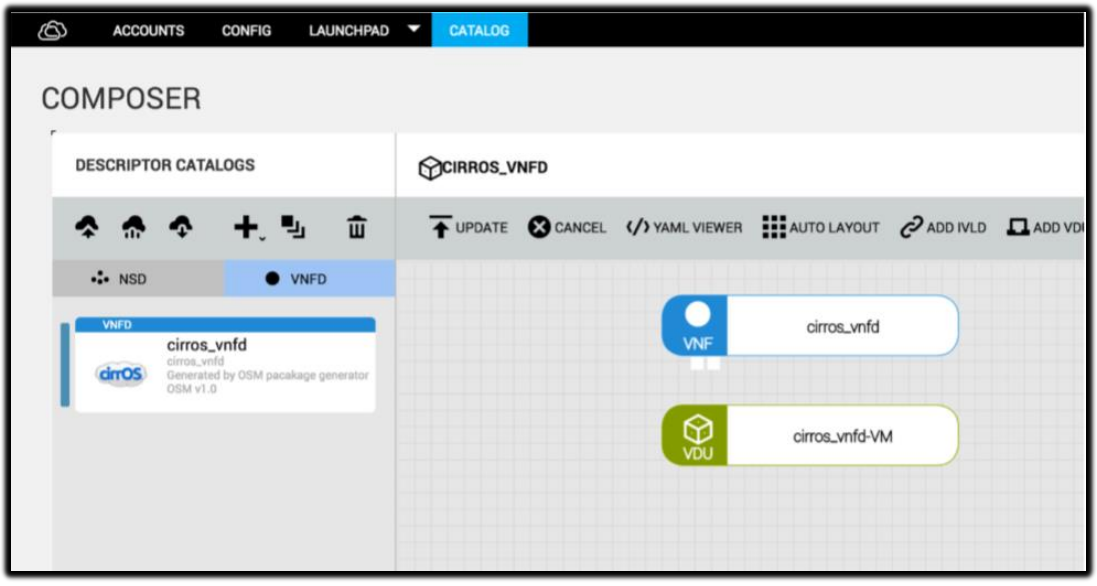


Image 23. OSM GUI composer, containing vnfd.

4.8.6) CHARMS

Another key aspect of the OSM MANO project are charms. VNF's operational procedures are embedded in the VNF Package, and are encapsulated in charms, which are controlled by VCA.

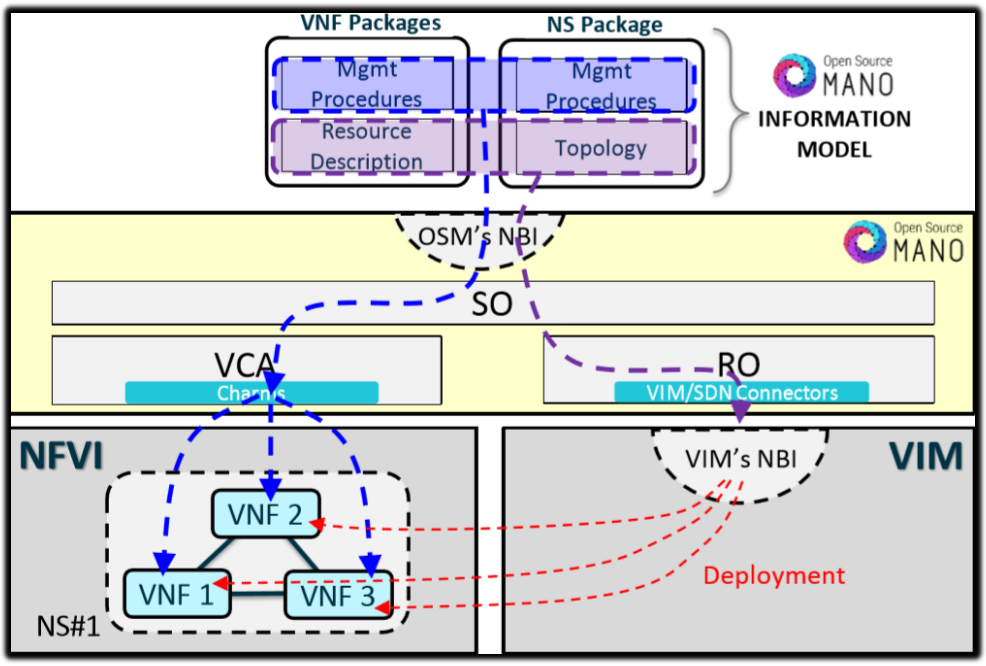


Image 24. OSM MANO workflow

[http://www.visualpcs.com/open-source-mano-osm-addressing-interoperability-challenges-in-nfv/]

A charm is a collection of scripts and metadata that encapsulate the distilled DevOps knowledge of experts in a particular product. These charms make it easy to reliably and repeatedly deploy applications, then scale them as required with minimal effort.

Driven by Juju, these charms manage the complete lifecycle of the application, including installation, configuration, clustering, and scaling.

A charm is a set of actions and hooks. With actions to be programs, and hooks are events/signals. For commodity and reusability, those actions and hooks are grouped in layers. A charm will always have one layer, and that layer has incorporated some actions and hooks, also that specific layer can import other layers. The resulting charm has all the actions and hooks from all the layers joined together, plus additional default actions and hooks.

The SO directs the RO to create a virtual machine using the selected VNF image. When that has successfully completed, the SO will instantiate a LXD container, managed by

Juju, with the proxy charm. The proxy charm will then communicate with the VNF virtual machine to do Day 1 configuration. [31]

4.9) Day 1 and Day 2 Configuration.

In this chapter we are going to see the Day 1, 2 configuration flows of OSM MANO project with charms. With VNF descriptors map charm actions to VNF primitives, thus providing a full set of enablers for NFV management.

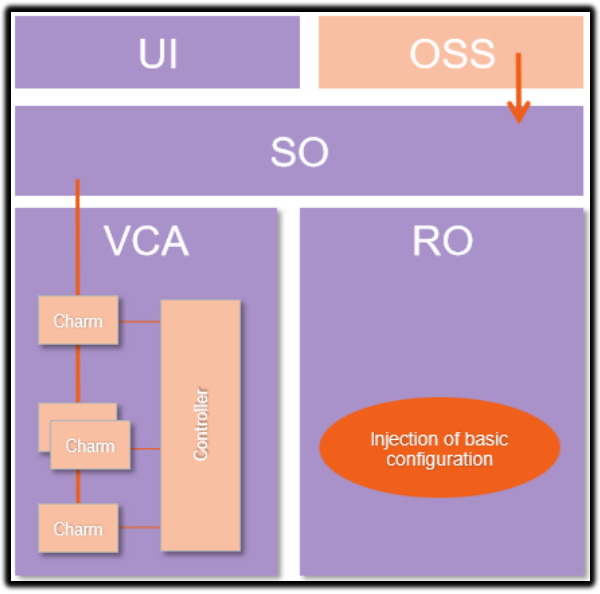


Image 25. OSM block at Instantiation time

Day-1 Configuration

At instantiation time, Day-0, the OSM Northbound API creates a new Network Service instance, while VNF Descriptor injects the basic configuration settings (SSH keys, hostname, user-data, scripting) via charms actions. Also, Network Service Descriptor has the initial config and scripting.

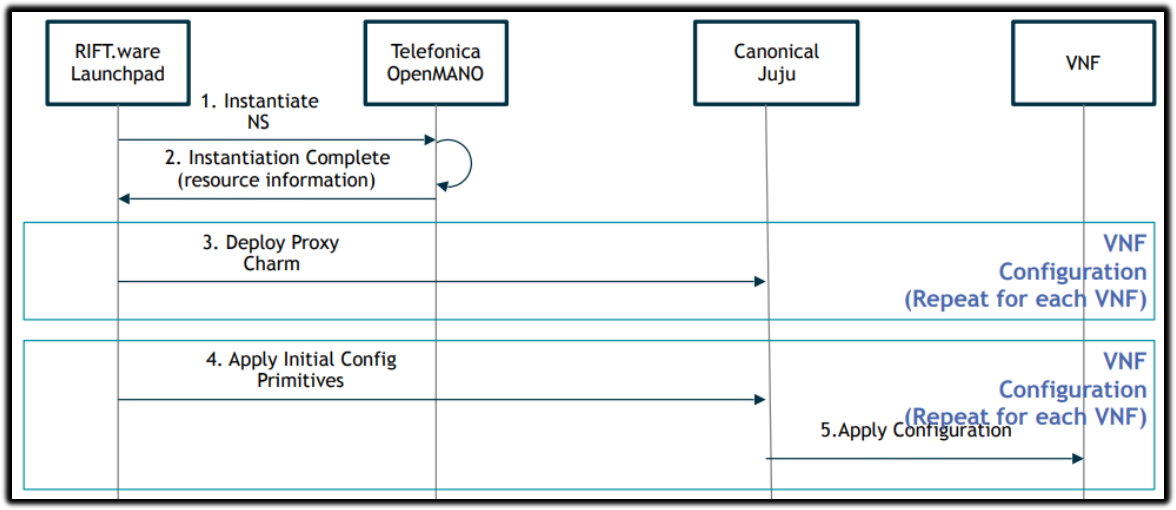
Day-2 Configuration

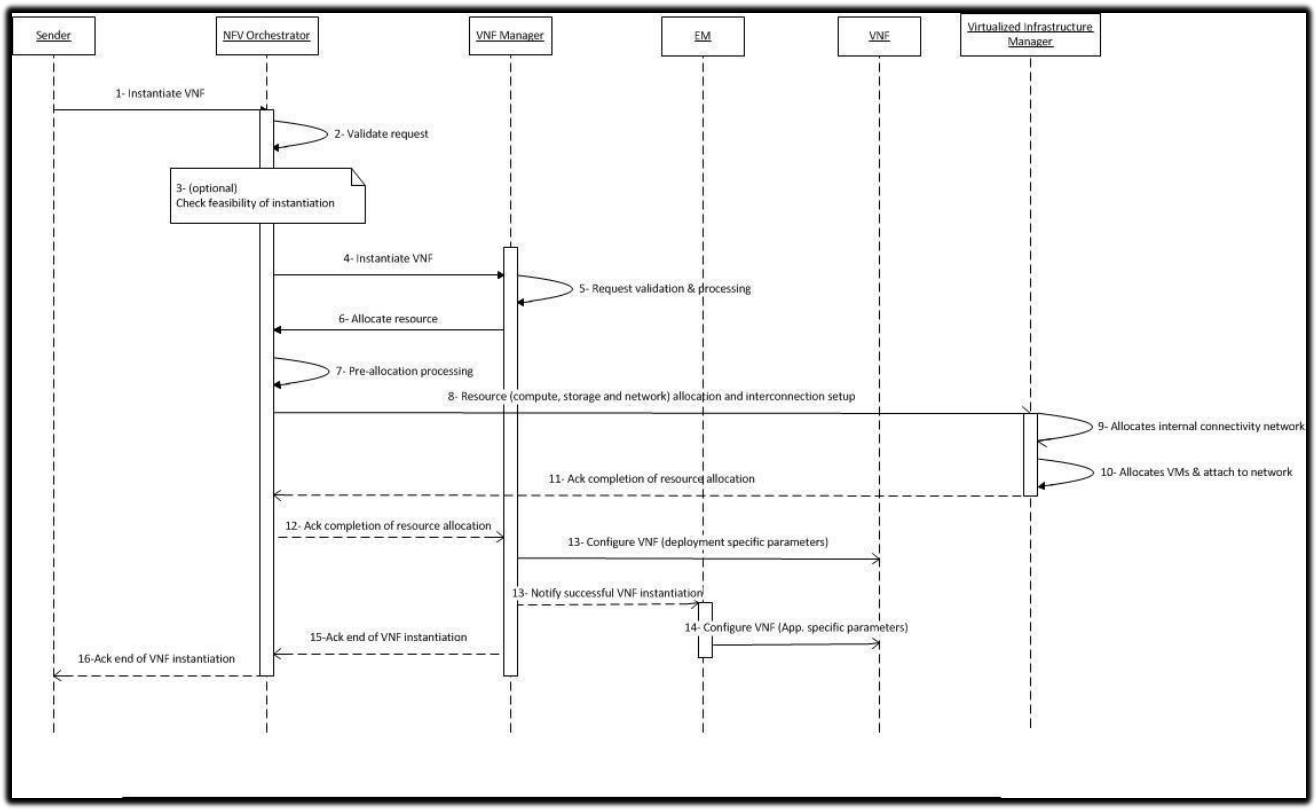
During day-2 configuration, OSMs NB API gets the NS instance records and calls NS or VNF primitive. At the same time VNFD has the config primitive with charms actions, also NSD has the service primitive and scripts.

Day-2 Scaling

At day-2 scaling time, OSM NB API gets the NS instance record and scales in/out the instance. VNFD via charms actions is injected with the basic configuration of cloud initiation, while NSD initiates scaling groups and pre/post scaling primitives.

In the below images we can see how the instantiation is sequenced and actions taking place, from the Launchpad to the VNF use.





Images 26. Examples of instantiation sequence diagram.
[https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf]

4.10) What's new with OSM release THREE

Release THREE also includes the support of projects. A "project" is used to group things such as VNFDs, NSDs, NS instances, and datacenters (VIMs). These projects are shared spaces where users can access and operate a given set of Network Services (NS) and Virtual Network Functions (VNF), enabling collaborative work with orchestration. Also, for release three security has been updated, providing extensions related to access control security. Furthermore OSM 3 allows the definition of different roles, defined by admin user, with different sets of privileges. All users are mapped to at least one of these roles. It has been engineered and tested to be in support of Operators RFx processes, quick installation in VNF vendor, system integrator and operator environments. OSM 3 enhances interoperability with other components (VNFs, VIMs, SDN controllers, monitoring tools) and provides a plug-in framework to make platform maintenance and extensions significantly easier to provide and support.

- Multi-tenancy & RBAC
- Monitoring Module
- Enhanced VIM support & emulation, support including different generations and versions of Openstack, VMWare, and AWS
- NB API Consolidation
- Affinity/Anti-Affinity Rules
- CI/CD Workflow
- Information Model Consolidation
- Use of Generic VNFM
- Restructure of NFVO and VNFM into SO/RO/VCA

4.11) Installing OSM release Three

In this section an analysis will be made on how the OSM MANO is installed and configured through the wiki of OSM release three. Minimum requirements of the installation along with a step by step configuration.

Release Three was presented at November 2017.
https://osm.etsi.org/wikipub/index.php/OSM_Release_THREE

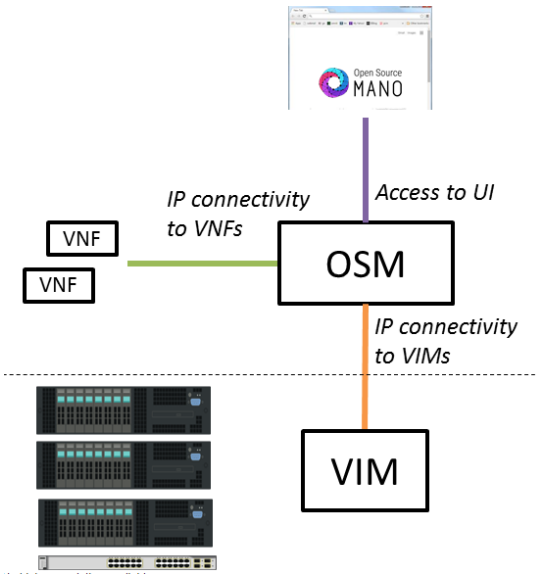


Image 27. OSM interaction with VIM and VNF.
[https://osm.etsi.org/wikipub/index.php/OSM_Release_THREE]

OSM talks to the VIM for the deployment of VNFs and VLs connecting them and OSM talks to the VNFs deployed in a VIM to run day-0, day-1 and day-2 configuration primitives.

In order for OSM to work, it is assumed that, each VIM has an API endpoint reachable from OSM, each VIM has a so-called management network which provides IP address to VNFs and that management network is reachable from OSM.

The recommended requirement to run OSM are:

- 8 CPUs, 16 GB RAM, 80GB disk and a single interface with Internet access
- Ubuntu16.04 (64-bit variant required) as base image (http://releases.ubuntu.com/16.04/), configured to run LXD containers.

There are multiple options for installing OSM, including:

- Installing into LXC containers
- From binaries
- From source code
- From prepared LXC images
- Installing into Docker Containers

```
- wget https://osm-download.etsi.org/ftp/osm-3.0-three/install_osm.sh
- chmod +x install_osm.sh
- ./install_osm.sh
```

Installation of OSM Client. From version v3.0.2 the OSM installer installs OSM client by default in your system, so there is no need of installation. If we type osm into CLI we will see the output. For the installation of OSM client we have:

```
curl http://osm-download.etsi.org/repository/osm/debian/ReleaseTHREE/OSM%20ETSI%20Release%20Key.gpg | sudo apt-key add -

sudo add-apt-repository -y "deb [arch=amd64] http://osm-download.etsi.org/repository/osm/debian/ReleaseTHREE stable osmclient"

sudo apt-get update

sudo apt-get install -y python-osmclient
```

We need to specify the OSM host, either via an environment variable or via the osm command line.

```
export OSM_HOSTNAME=`lxc list | awk '($2=="SO-ub"){print $6}'`
export OSM_RO_HOSTNAME=`lxc list | awk '($2=="RO"){print $6}'`
```

From the OSM client we have the below options:

```
config-agent-add
 config-agent-delete
 config-agent-list
 ns-create
 ns-delete
 ns-list
 ns-monitoring-show
 ns-scale
 ns-scaling-show
 ns-show
 nsd-delete
 nsd-list
 ro-dump
 upload-package
 vcs-list
 vim-create
 vim-delete
 vim-list
 vim-show
 vnf-list
 vnf-monitoring-show
 vnf-show
 vnfd-delete
 vnfd-list
```

If the installation is complete without any problem or need of troubleshooting the GUI should be accessible. Login with admin/admin credentials.

The IP of the user interface is the IP created by lxd, and the port is 8443.

At this time, with the below commands, VIM List and Network Service Descriptors list will show empty.

```
osm vim-list
osm nsd-list
```

Next, we should ensure that components have been adequately integrated during the installation process

SO - RO - SO - VCA Config Agent.

Associate a VIM (or VIMs) to OSM

OpenStack (real, or VIM Emulator)

VMWare VCD

Amazon Web Services

4.12) Installing OpenVIm

In order to run openvim in normal mode and deploy dataplane VNFs, an appropriate infrastructure is required. Below a reference architecture for an openvim-based DC deployment.

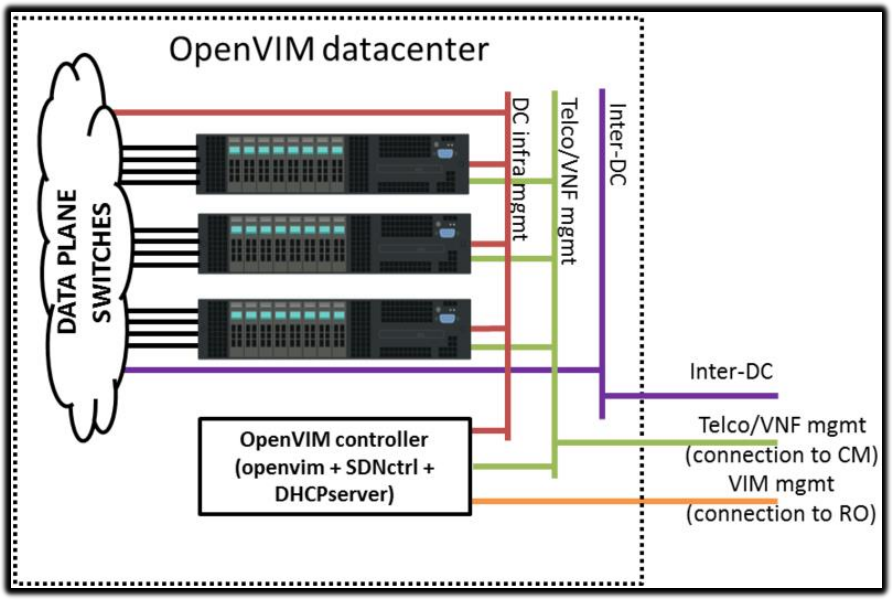


Image 28. Openvim Infrastructure
[https://osm.etsi.org/wikipub/index.php/OSM_Release_THREE]

- Openvim needs to be accessible from Resource Orchestrator.
- To make its API accessible from Resource Orchestrator (openmano). (Vim mgmt.)
- To be connected to all compute servers through a network. (DC infrastructure)
- To offer management IP addresses to VNFs for VNF configuration from CM. Telco/VNF mgmt.

The compute nodes besides been connected to the DC infrastructure Network must be connected also to two additional networks.

- Telco/VNF management network (juju server), used by CM (Configuration Management) to configure the VNFs, and
- Inter-DC-network, in order to connect two sites.

Openvim is installed using:

```
wget -O install-openvim.sh "https://osm.etsi.org/gitweb/?p=osm/openvim.git;a=blob_plain;f=scripts/install-openvim.sh;hb=1ff6c02ecff38378a4d7366e223cefd30670602e"

chmod +x install-openvim.sh

sudo ./install-openvim.sh -q    # --help  for help on options
```

OSM also supports different kind of VIM like Oenstack, VMware vCloud Director, Amazon Web Services. Also, OSM can manage external SDN controllers to perform the dataplane underlay network connectivity on behalf of the VIM.

Announcing the OpenVim site:

```
osm vim-create --name openvim-site --auth_url http://x.x.x.x:9080/openvim --account_type openvim --description "Openvim site" --tenant osm --user dummy --password dummy
```

or openstack

```
osm vim-create --name openstack-site --user admin --password userpwd --auth_url http://x.x.x.x:5000/v2.0 --tenant admin --account_type openstack
```

Uploading VNF image to VIM.

In the osm ftp server there are examples of VNF and NS images.

https://osm-download.etsi.org/ftp/osm-3.0-three/examples

To onboard an image into openvim we must copy our image to the NSF shared folder and execute

```
openvim image-create --name cirros034 --path /mnt/openvim-nfs/cirros-0.3.4-x86_64-disk.img
```

Onboarding a VNF.

There are two ways to onboard a VNF. Either from GUI or via the OSM client.

From the OSM client we execute

```
osm upload-package cirros_vnf.tar.gz

osm vnfd-list
```

And from the UI, we go to Gatalog→Import→VNFD, and drag and drop the selected .tar

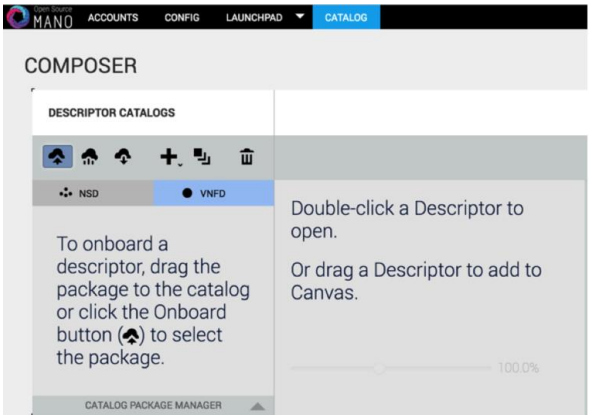


Image 29. OSM Composer
[https://osm.etsi.org/wikipub/index.php/OSM_Release_THREE]

Onboarding a NS.

Again, there are two ways to onboard a NS.

From OSM client we execute

```
osm upload-package cirros_2vnf_ns.tar.gz
osm nsd-list
```

or via UI we follow Catalog → Import→ and drag and drop the selected .tar

Insatiate the NS

From the osm client we execute

```
osm ns-create --nsd_name cirros_2vnf_ns --ns_name <ns-instance-name> --
vim_account <data-center-name>
osm ns-list
```

and from the user interface we follow Launchpad → Insatiate → Select the NS to be instantiated → next → add name to the NS instance → Launch.

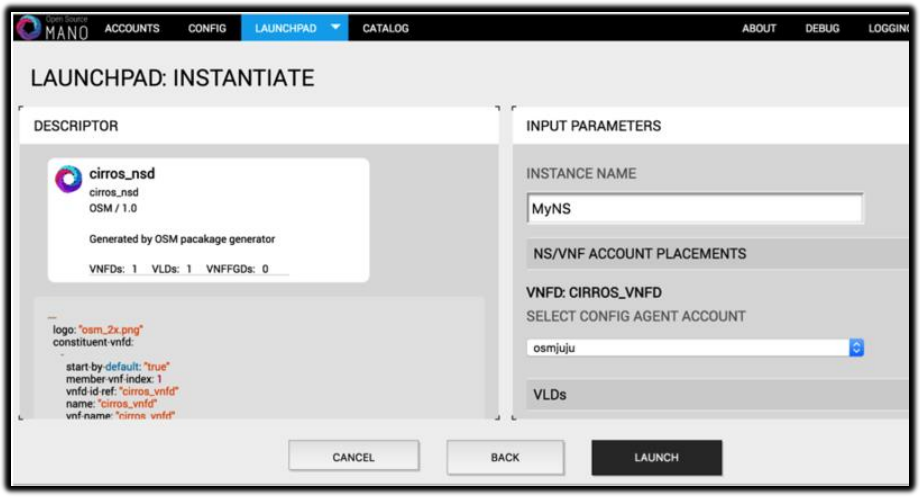


Image 30. NSD Insatiate
[https://osm.etsi.org/wikipub/index.php/OSM_Release_THREE]

5) NFV in Market.

In order to illustrate the response of the NFV we had to cover how the network industry incorporates the new NFV Architecture. Thus, in this section there is reference about NFV implementations and products from industry. Tables below show the available products in the market, organized by category, open source, standardization and the subcategory including their applications. Major companies have products using the NFV Architecture both open source and not.

5.1) MANO Category.

MANO Category			
Company	**OPNFV**	**ETSI NFV ISG**	**Sub-Category**
Brocade SDN Controller	**Y**	**Y**	Orchestration, VNF Manager, OpenFlow management for underlying PNF. Netconf configuration management for VNF and PNF.
Cisco Virtual Topology	**N**	**N**	SDN SW overlay provisioning and management system
Cisco Network Services Orchestrator	**N**	**N**	Orchestration
Cisco Elastic Services Controller	**N**	**N**	Orchestration, VNF Manager, VIM
Juniper Networks Contrail	**Y**	**Y**	Orchestration, VNF Manager, Virtualized Infrastructure Manager
Ensemble Orchestrator	**Y**	**Y**	Orchestration, VNF Manager
Affirmed Mobile Content Cloud	**N**	**N**	Orchestration
Chameleon SDS	**N**	**Y**	Orchestration
Anuta Network NCX	**N**	**Y**	Orchestration, VNF Manager
Athena	**N**	**N**	Orchestration, VNF Manager, Virtualized Infrastructure Manager
Avaya SDN Fx Architecture	**N**	**N**	Orchestration, Virtualized Infrastructure Manager
Brocade VNF Manager	**Y**	**Y**	VNF Manager
Ubuntu OpenStack	**Y**	**Y**	Virtualized Infrastructure Manager
CA Virtual Network Assurance	**N**	**N**	Orchestration, VNF Manager, Virtualized Infrastructure Manager
Exanova Service Intelligence	**N**	**Y**	Service Assurance; related to TRAM
Blue Planet	**Y**	**Y**	Orchestration
Ericsson Network & Cloud Manager	**Y**	**Y**	Orchestration, VNF Manager
Network Virtuora OM & RV	**N**	**N**	Orchestration, VNF Manager
Huawei FusionSphere	**Y**	**Y**	Orchestration, VNF Manager, Virtualized Infrastructure Manager
MRV Pro-Vision	**N**	**Y**	Orchestration
NI - Controller	**N**	**N**	Orchestration
NI – Framework	**N**	**N**	Orchestration
NEC Orchestrator	**Y**	**Y**	Orchestration, VNF Manager, Virtualized Infrastructure Manager
CloudBand (Nokia)	**Y**	**Y**	Orchestration, VNF Manager, Virtualized Infrastructure Manager, Network service orchestrator
Network Service Orchestrator Solution (Oracle)	**Y**	**Y**	Orchestration
PLUMgrid OpenStack Networking Suite	**N**	**N**	Orchestration, VNF Manager, Virtualized Infrastructure Manager
CloudShell (Qualisystems)	**N**	**N**	Orchestration, Virtualized Infrastructure Manager
ETX-2I vCPE Platform (RAD)	**N**	**Y**	Orchestration, Virtualized Infrastructure Manager
CloudMetroo 100 (TELCO Systems)	**N**	**N**	Virtualized Infrastructure Manager
vCloud NFV Platform	**Y**	**Y**	Virtualized Infrastructure Manager, Day 2 Operations Management

5.2) Infrastructure Category.

Infrastructure Category			
Company	**OPNFV**	**ETSI NFV ISG**	**Sub-Category**
Cisco NFV Infrastructure	N	N	NFV HW platform, NFV SW platform, VIM, HW acceleration, SW acceleration, Element Management System, value-added SW tools
Titanium Server	Y	Y	NFV Software Platform
6WINDGate Packet Processing Software	N	Y	6WindGate is a software networking stacks that can be used to remove Linux Bottlenecks in the creation of NFV
6WIND Virtual Accelerator	Y	Y	SW Acceleration
FSP 150 ProVM (adva)	Y	Y	EMS, HW acceleration, SW acceleration, NFV hardware
Fast Path Accelerator (Aricent)	N	Y	SW acceleration, SDK to build NFV solutions
Calvium ThunderX	Y	Y	NFV Hardware Platform
Corsa 10G/100G SDN Switches	N	Y	NFV Hardware Platform
Dell Open Networking Switches	Y	N	NFV Hardware Platform
EMC Provider Cloud Systems	Y	Y	NFV Hardware Platform, NFV Software Platform
Ericsson Cloud Execution Environment	Y	Y	HW acceleration, SW acceleration, NFV hardware platform, NFV software platform
NPS-400 Network Processor	N	N	HW acceleration
HPE NFV System Family (HP)	Y	Y	NFV Software Platform
Intel Open Network Platform Server	Y	Y	NFV Hardware Platform, NFV Software Platform, SDK to build NFV solutions
Mellanox MSX1410-OCP	N	N	HW Acceleration, SW Acceleration, NFV Hardware Platform
Midokura Enterprise MidoNet	Y	N	NFV Software Platform
Netronome Agile CX	N	N	HW Acceleration, SW Acceleration, NFV Hardware Platform
NoviFlow NoviSwitch	N	Y	NFV Hardware Platform, NFV Software Platform
NoviWare	N	Y	NFV Software Platform
Nuage Networks VSP	N	N	NFV software platform. The VSP consists of software-based policy management and analytics, SDN controller, virtualized routing, and switching
Red Hat NFV Platform	Y	Y	NFV Software Platform

Table 4 . NFV in Market [SDx Central, Market Report, Mega NFV Report Pt. 1: MANO and NFVI]

ANNEX A

In this chapter we will analyze the experience installing OSM MANO from our perspective. The aim was to install OSM MANO and thus get familiar with open source MANO for NFV. We used a laptop with 4 physical CPUs and 4 GB RAM, with Ubuntu 16.04. Minimum requirements for OSM MANO is 4CPUs and 8 GB RAM, with 40 GB disk and a single interface with internet access.

Installing from binaries (as it is recommended) was more than 90 minutes. The preparation of LXC images was up to 10 minutes and adding the VIM emulator is more than 10 minutes.

From the beginning we had problems installing the OSM as it was constantly failing to install. As you can see below:

```
###  so_is_up: FATAL error: OSM Failed to startup. SO failed to startup

BACKTRACE:

### FATAL /tmp/installosm.qvR4J3/jenkins/common/logging 39
```

After troubleshooting this, we had problems connecting to the UI, as we could ping the UI, but we couldn't connect from Chrome Browser.

```
ERROR: Command failed: "/usr/rift/container_tools/mkcontainer --modes U
I-base --rw-version ${PLATFORM_VERSION}"

##  Ε£Ε±Ε² 06 Ε™Ε±Ε½ 2018 08:23:37 ΕΌΕΌ EET main: UI install complete.
Return code was 1


### Ε£Ε±Ε² 06 Ε™Ε±Ε½ 2018 08:23:37 ΕΌΕΌ EET main: FATAL error: UI insta
ll failed

BACKTRACE:

### FATAL /tmp/installosm.fYaiE4/jenkins/common/logging 39
```

Also, we had problems connecting to the OSM client, as connection was refused. We addressed these problems reconfiguring Iptables manually, getting access to the UI.

During the troubleshooting and the configuration, it was obvious that the laptop couldn't handle the Processes for mentioned, as it was constantly lagging and freeze up.

Several installations were done, with different options tested but either we had to troubleshoot in order to move forward, and either Laptop was under-functioning.

We took the decision to upload the installation to the cloud. We choose Microsoft Azure Cloud Services, and we have committed 8 CPUs and 32 GB RAM with Ubuntu 16.04.

Below you would find some output charts of the installation process CPU usage and the Disk Write bytes.

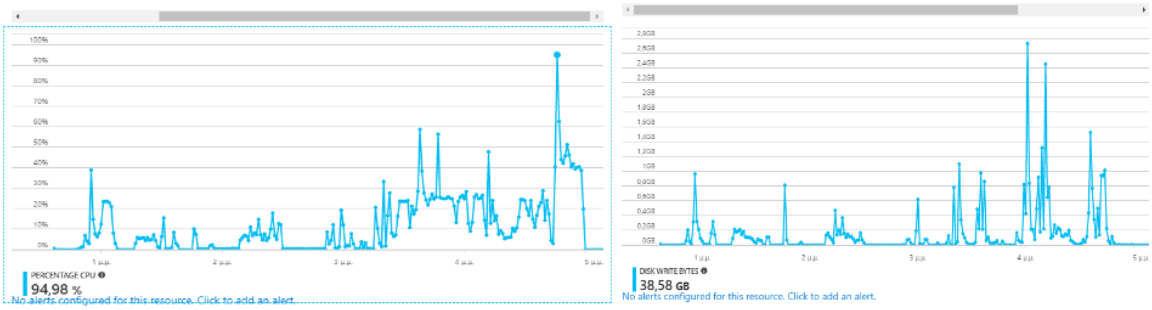


Image 31. CPU usage during installation.

As you can see this is a demanding installation. At the pick of the installation we had 94,5 % CPU usage of the 8 committed CPUs.

Installation was successful, and we had connection with the UI without troubleshooting.

We ensured that components have been adequately integrated during the installation process.

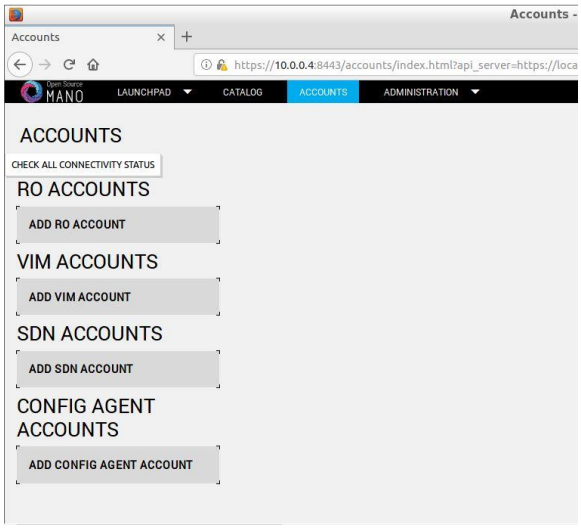


Image 32. Accounts.

And we have associated a VIM emulator to OSM.

```
osm@osm:~$ osm vim-list
+----------+--------------------------------------+
| vim name | uuid                                 |
+----------+--------------------------------------+
| emu-vim1 | e8bb0190-a715-11e8-b9cc-00163ef87882 |
+----------+--------------------------------------+
osm@osm:~$
```

Image 33. Output osm vim-list.

Then we checked the accounts of the OSM.

Below is an output of the RO account installation.

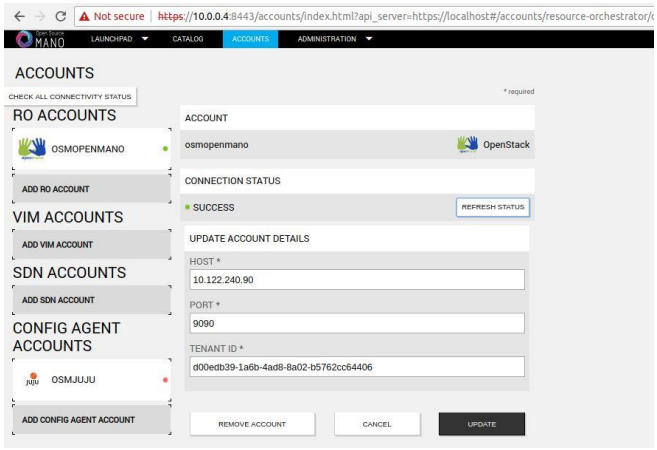


Image 34. RO accounts.

We can see the RO account name, ROs container IP, the corresponding API port, the Default tenant id and Juju installation.

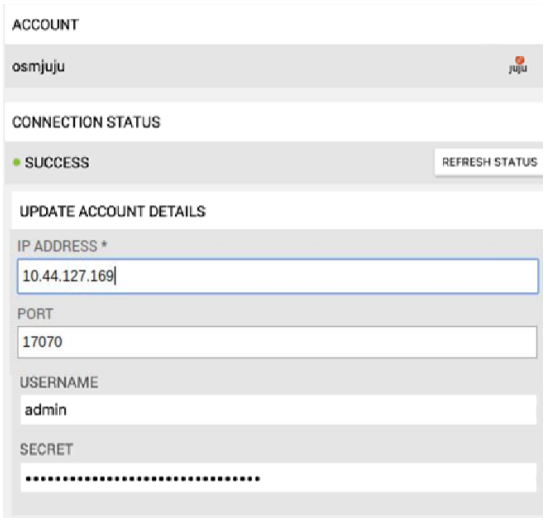


Image 35. Juju account.

Consequently, we tried to upload NS / VNF packages from here:

https://osm-download.etsi.org/ftp/osm-3.0-three/examples/cirros_2vnf_ns/

We have uploaded them with both ways. Through CLI and through the UI. OSM suggested that we should upload VNFs firstly and then the NS, as NS usually contain references to existing VNFs.

Below you are able to see the successfully uploaded VNF and NS.

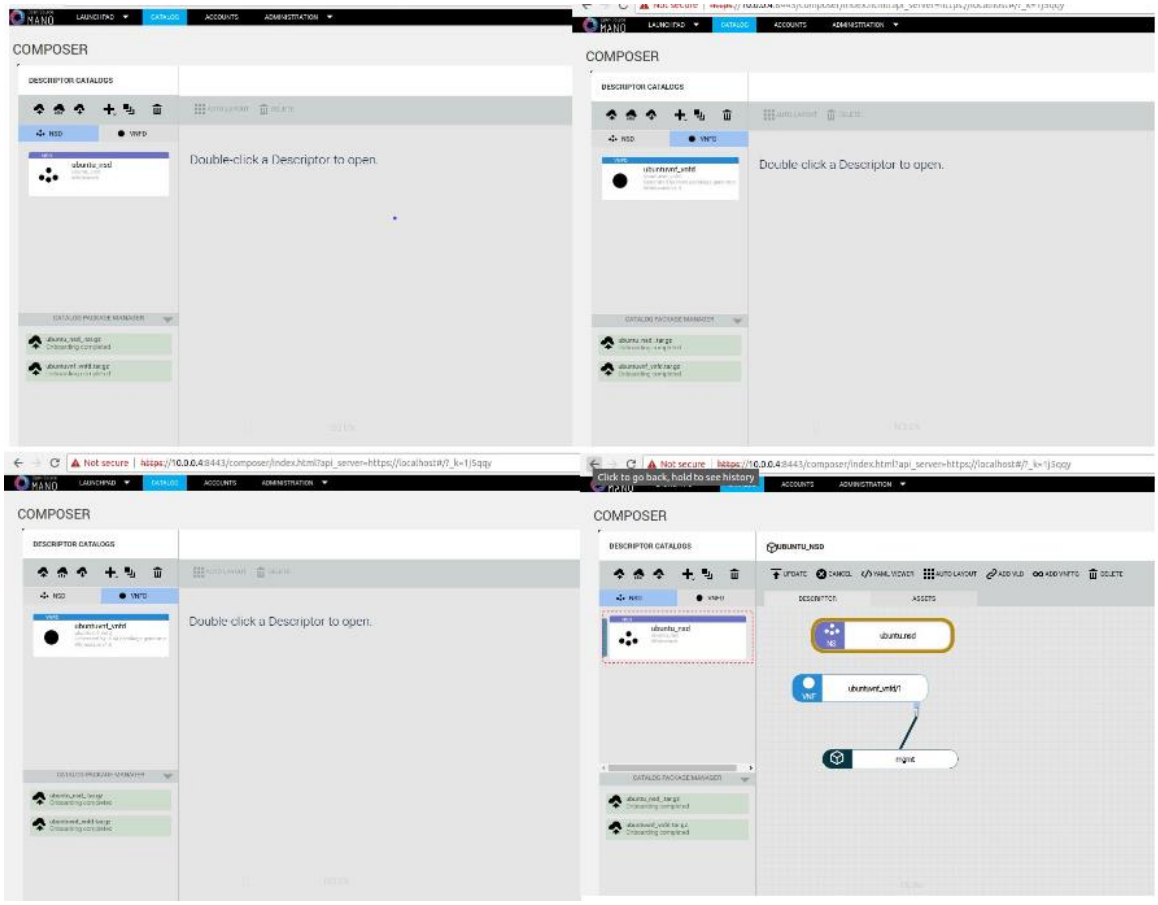


Image 36. Installed VNF and NS.

Also, through OSM client we can the above NS /VNF by executing osm vnfd-list and osm nsd-list

```
osm@osm:~$
osm@osm:~$ osm vnfd-list
+----------------+----------------+
| vnfd name      | id             |
+----------------+----------------+
| ubuntuvnf_vnfd | ubuntuvnf_vnfd |
+----------------+----------------+
osm@osm:~$ osm nsd-list
+------------+--------------------------------------+
| nsd name   | id                                   |
+------------+--------------------------------------+
| ubuntu_nsd | a8fa4bf9-12e3-4b68-9169-1ae51aab404f |
+------------+--------------------------------------+
osm@osm:~$
```

Image 37. Output osm vnfd-list, osm nsd-list

Proving that NSD and VNF were successfully onboarded.

We then tried to upload an example network package with our installed VIM emulator.

Below you can see the topology of the example uploaded. It is called Pingpong. Powered by Paderborn University.

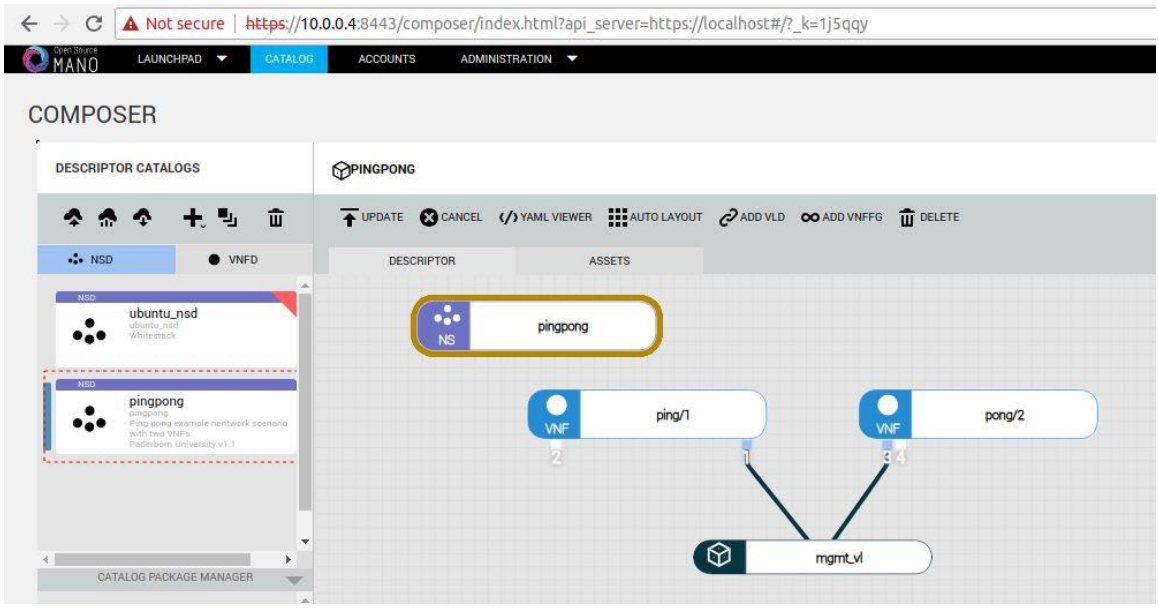


Image 38. Pingpong Topology.

The example states the two VNFs (ping, pong) are managed through mngmt_vl and we will try to ping from one to another, and vice versa.

In the below image you can see the Pingpong NSD preview.

Image 39. Pingpong NSD preview

Also, in the OSM client CLI we can see the new VNFs and NS uploaded.

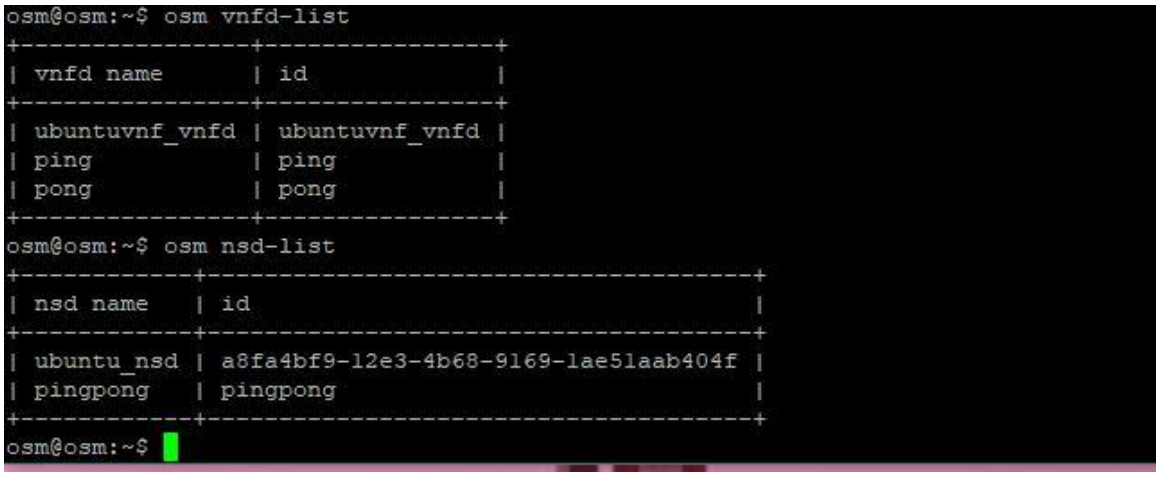


Image 40. Output nsd-list

After that we have tried to initiate the VNFs.

```
osm@osm:~$ osm vnf-list
+------------------------+--------------------------------------+---------------
-----+-------------------+
| vnf name               | id                                   | operational st
atus | config status     |
+------------------------+--------------------------------------+---------------
-----+-------------------+
| default__test__ping__1 | 592e6ffc-bd9a-44e3-af90-b43e7a7ffec2 | init
     | config-not-needed |
| default__test__pong__2 | fa418f58-7e3a-41a2-bb0a-e523ac0ab4e6 | init
     | config-not-needed |
+------------------------+--------------------------------------+---------------
```

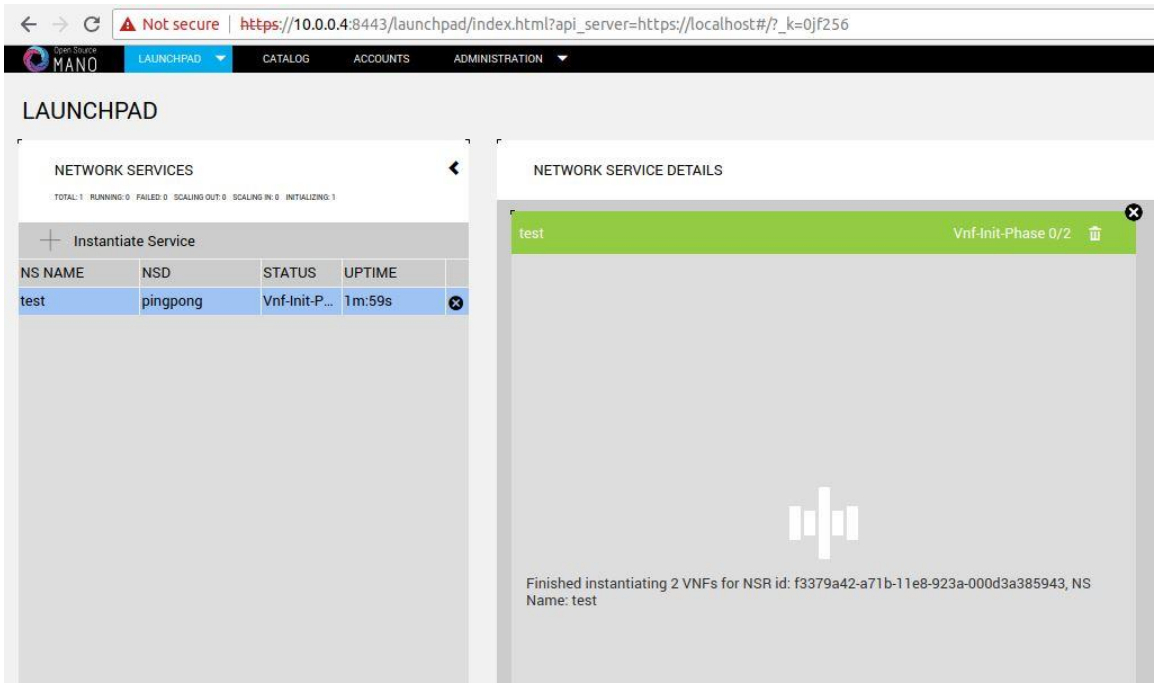


Image 41. Initiation

And we got a fail message in the initiation of the Network Services.

```
osm@osm:~$ osm ns-list
+------------------+--------------------------------------+--------------------
---------------+
| ns instance name | id                                   | operational status |
 config status |
+------------------+--------------------------------------+--------------------
---------------+
| test             | f3379a42-a71b-11e8-923a-000d3a385943 | failed             |
 configuring   |
+------------------+--------------------------------------+--------------------
---------------+
osm@osm:~$
```

Image 42. Fail message on initiate.

The problem was as we have found out that the juju failed communicating with the OSM.

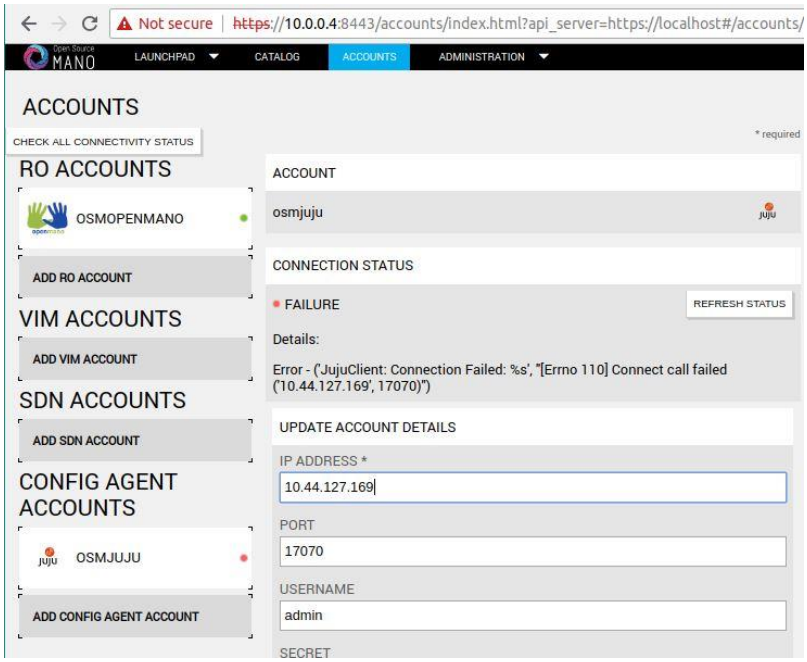


Image 43. Juju connection failure.

We have tried to troubleshoot this problem but with no success. Also, we have studied the Troubleshooting guide of OSM Release three, but there was no success. We also, tried to restart the VCA but it didn't work.

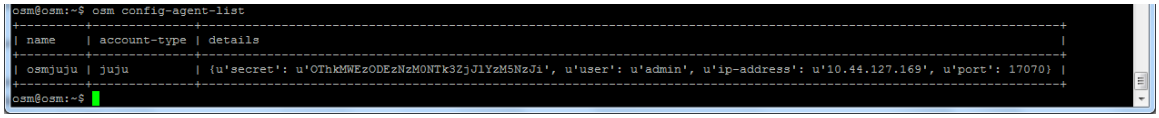


Image 44. Output osm config-agent-list

Services RO and SO, in the OSM MANO, were working properly as you can see below.

RO Service

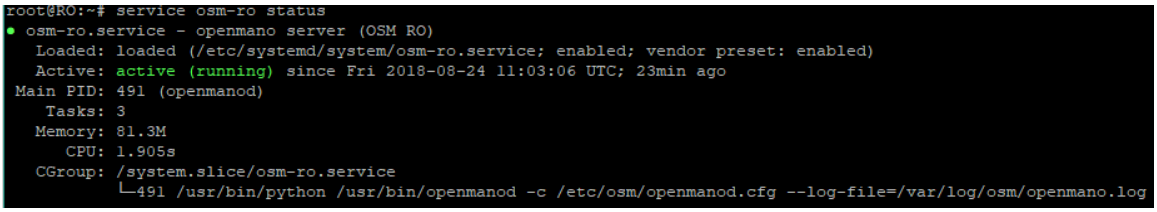


Image 45. RO status

SO status

```
● launchpad.service - RIFT.ware Launchpad
   Loaded: loaded (/etc/systemd/system/launchpad.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2018-08-24 11:03:17 UTC; 24min ago
 Main PID: 383 (rwmain)
    Tasks: 324 (limit: 65536)
   Memory: 5.6G
      CPU: 43min 26.374s
```

Image 46. SO status

Conclusions

In this last section we are going to share our comments and thoughts with our experience with OSM MANO.

Firstly, it was very difficult to install OSM MANO three into a common laptop/PC, as the minimum requirements were unapproachable. First step installation fails were very common and even thought we managed to overcome there were problems configuring iptables, firewall in order to communicate with OSM GUI at the proper suggested port. (Several times we had connection refused reply although we had followed installation guide step by step). Moreover, after installation finished, laptop/PC was always crashing and lagging, due to the CPU overload. As installation to a common laptop was not reliable, author tried to take advantage company's resources but with no success. Due to GDPR and the fact that a public IP was needed in order to control this project, permission was denied for security reasons. In order to overcome this problem we uploaded project to a well-known cloud computing platform purchasing minimum requirements. As a result, we managed to overcome all the above problems, communicating with GUI. Thereafter we had several other problems getting all the services to communicate and configure properly the test example. We had a stalemate with juju configuration thus we were unable to initiate the test example.

Judging from a network engineers point of view OSM MANO has to overcome some problems in order to be introduced to company's environment shifting to this new network model. We strongly believe that a more detailed installation guide would help network engineers and people testing OSM MANO, as existing ones cover the basics, lack in illustration, and also some previous knowledge of python is needed to understand the interconnections between blocks and their functionality.

Also, we believe that troubleshooting guides should be more developed explaining all the details in installation by step-by-step presentations. Network Engineers are not always familiar with python and charms, so it is difficult for us to troubleshoot problems concerning the OSM MANO.

As this is an open environment we believe that an option of an installed VM with all the installed components of OSM MANO should be available to be downloaded. In

that case network engineers could focus on the OSM MANO functions and not on the installation. They could get more familiar with the product thus introducing this exciting new environment in a job-related environment.

Lastly, we think that in order to OSM MANO to be a common ground in industry, more POC (Proof of Concept) presentations must be given, than the two available [OSM three – white paper], for more people to be convinced to bring this project into real industry environment applications.

Nevertheless, OSM MANO is constantly changing, with new releases every six months, with tested new environments. The community is seeding code in order to make OSM MANO more agile and simple. An exciting new environment that would determine the future network.

ANNEX B

In this annex we have some examples diagrams of a how a Multivim environment and OSM DC Infrastructure are connected using OpenVim, Openstack and OSM controller. Also the last images shows a logical diagram of OSMs VNFD.

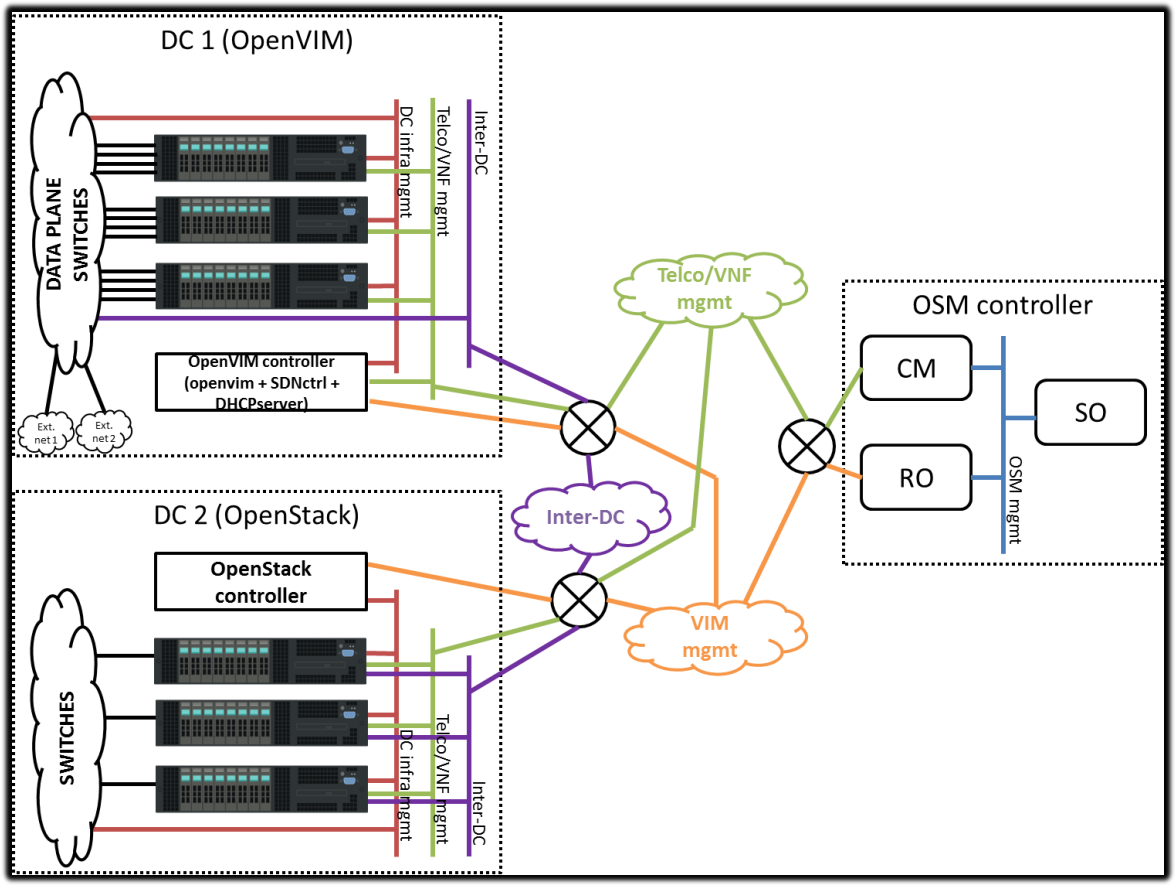


Image 47. MultiVIM infrastructure
[https://osm.etsi.org/wikipub/index.php/OSM_Release_THREE]

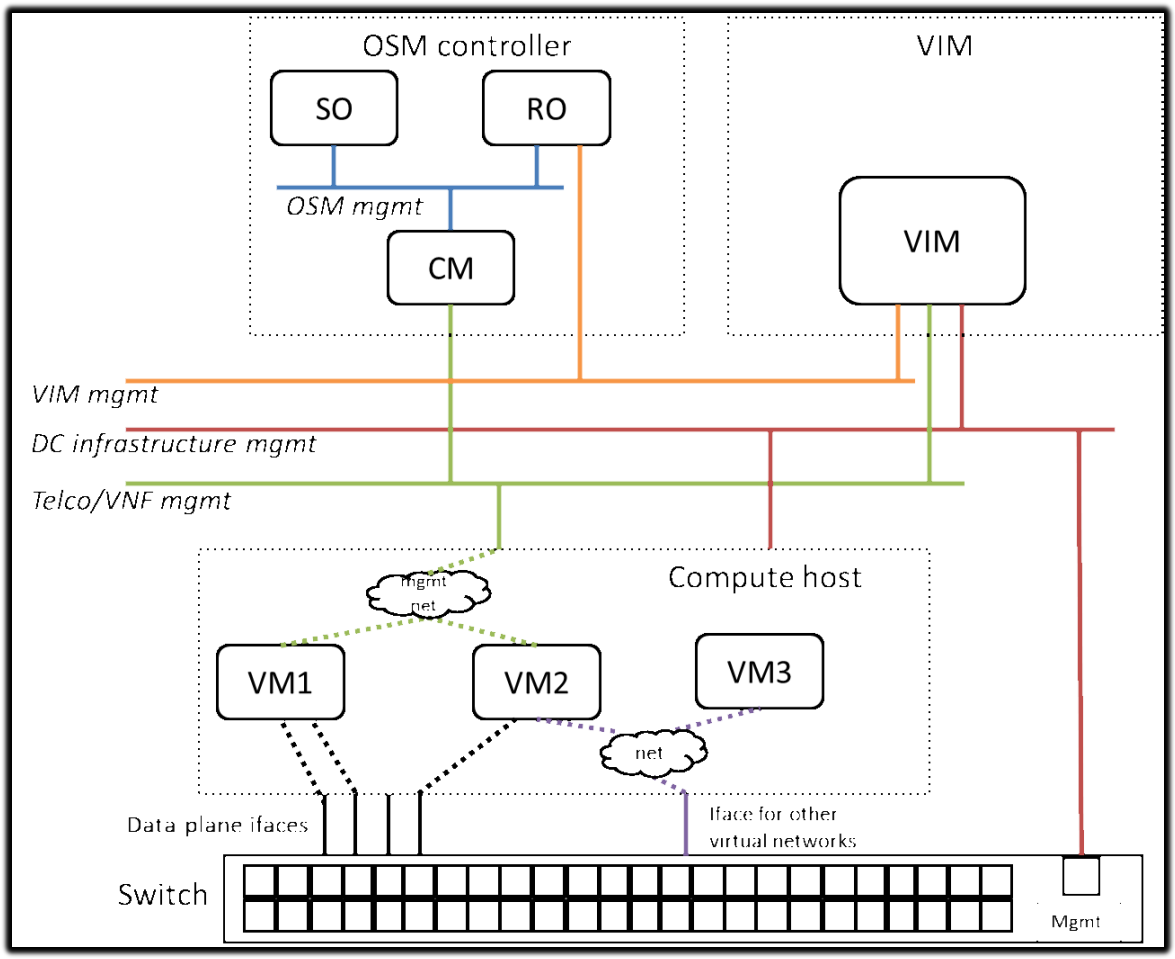


Image 48. OSM DC Infrastructure

[https://osm.etsi.org/wikipub/index.php/OSM_Release_THREE]

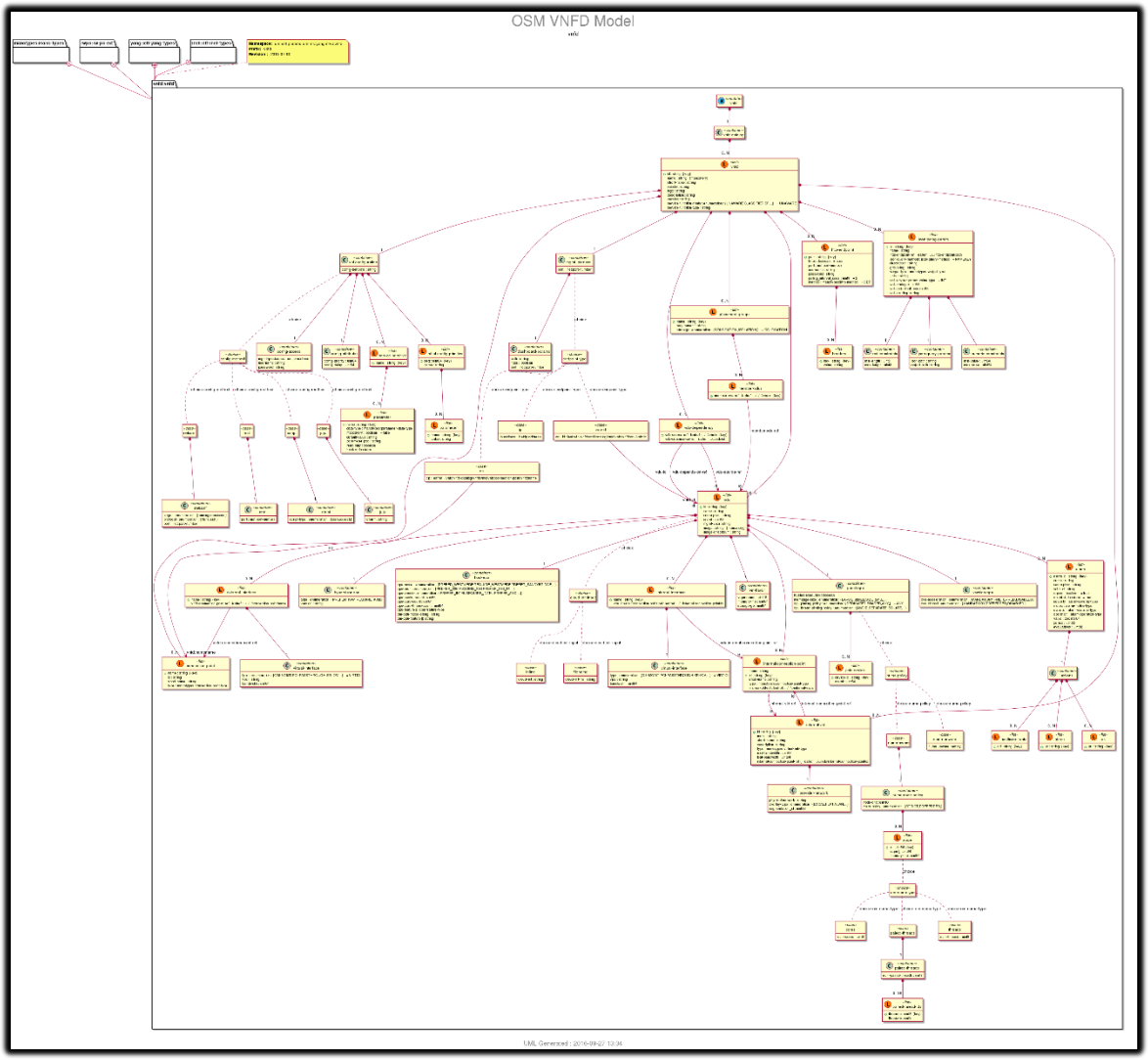


Image 49. OSM VNFD Model
[https://osm.etsi.org/wikipub/images]

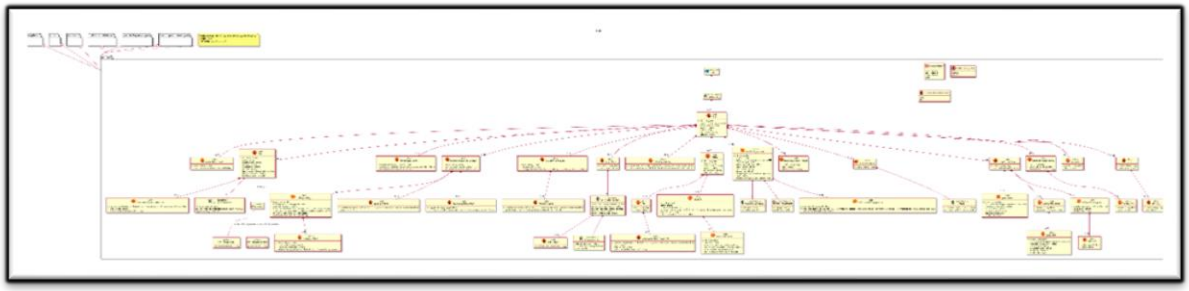


Image 50. OSM NSD Model
[https://osm.etsi.org/wikipub/images]

References

[1] Network Functions Virtualization (NFV) with a Touch of SDN , By Rajendra Chayapathi, Syed F. Hassan, Paresh Shah Jan 25, 2017

[2] https://www.networkcomputing.com/networking/5-nfv-benefits-trends-driving-them/1187036662

[3] https://www.networkworld.com/article/3253118/virtualization/what-is-nfv-and-what-are-its-cost-performance-and-scaling-benefits.html

[4] https://en.wikipedia.org/wiki/Network_function_virtualization

[5]https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf

[6] https://www.opennetworking.org/, Open Networking Foundation

[7] The Control Plane, Data Plane and Forwarding Plane in Networks, by BRENT SALISBURY

[8] Meridian: An SDN Platform for Cloud Network Services, Mohammad Banikazemi, David Olshefski, Anees Shaikh, John Tracey, and Guohui Wang, IBM T. J. Watson Research Center

[9] https://linuxcontainers.org/lxd/introduction/

[10] [https://www.openstack.org/assets/presentation-media/Achieving-end-to-end-NFV-with-OpenStack-and-Open-Source-MANO.pdf

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

[12] https://www.openstack.org/assets/telecoms-and-nfv/OpenStack-Foundation-NFV-Report.pdf

[13] https://wiki.opendaylight.org/view/Main_Page

[14] https://wiki.opendaylight.org/view/OpenDaylight_Controller:Architectural_Framework

[15] https://www.gsma.com/futurenetworks/wp-content/uploads/2017/05/Virtualisation.pdf

[16] https://www.etsi.org/technologies-clusters/technologies/nfv

[17] https://osm.etsi.org/images/OSM-Whitepaper-TechContent-ReleaseTHREE-FINAL.pdf

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

[19] https://wiki.debian.org/Hugepages

[20] http://www.tmurgent.com/WhitePapers/ProcessorAffinity.pdf

[21] https://docs.openstack.org/nova/pike/admin/cpu-topologies.html

[22] https://libvirt.org/guide/html/Application_Development_Guide-Device_Config-PCI_Pass.html

[23] https://github.com/facebook/rocksdb/wiki/Direct-IO

[24] https://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html

[25] https://software.intel.com/en-us/blogs/2014/12/11/intels-cache-monitoring-technology-use-models-and-data

[26] https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology

[27] OSM_R2_Information_Model.pdf, July 19th, 2017

[28] Osm-r1-information-model-descriptors.pdf, October 3, 2016

[29] OSM release three – white paper, ETSI OSM Community, October 2017

[30] https://github.com/nfvlabs/openvim

[31] https://osm.etsi.org/wikipub/index.php/Creating_your_own_VNF_charm_(Release_THREE)