ETSI GR NFV-MAN 001 V1.2.1 (2021-12)

Network Functions Virtualisation (NFV); Management and Orchestration; Report on Management and Orchestration Framework

Disclaimer

The present document has been produced and approved by the Network Functions Virtualisation (NFV) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
RGR/NFV-MAN001ed121

Keywords
configuration, management, network, NFV, orchestration

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Group Report (GR) has been produced by ETSI Industry Specification Group (ISG) Network Functions Virtualisation (NFV).

ETSI GS NFV-MAN 001 V1.1.1 was published in December 2014. By then, it represented the view on the Network Functions Virtualisation (NFV) framework of the ETSI ISG NFV in the early days. Although part of the information it contained has become outdated, the ISG NFV decided to keep its content and produce the present revision for information and historical purposes, helping newcomers to get a global view of the initial NFV management and orchestration framework in one single document. In the present version, all normative provisions have been removed. Use of the present document for tutorial purposes is discouraged as the contents of some clauses are partially obsolete and can be misleading.

The contents of the present document are superseded by the contents of ETSI GS NFV 006 [i.11] and of the documents it references.

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document describes the management and orchestration framework for the provisioning of Virtualised Network Function (VNF), and the related operations, such as the configuration of the virtualised network functions and the infrastructure these functions run on. The objectives are to explain this framework, and the management and orchestration, identify topics that could serve in later gap analysis against current standards, identify best practices and provide guidance on how to address identified new topics. The focus of the present document is on aspects of management and orchestration that are specific to NFV.

The present document addresses the following topics of management and orchestration: architecture framework for management and orchestration of NFV, information elements, interfaces, provisioning, configuration, and operational management, including interworking with existing operations and management systems.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI GS NFV 001: "Network Functions Virtualisation (NFV); Use Cases".

[i.2] ETSI GS NFV 002: "Network Functions Virtualisation (NFV); Architectural Framework".

[i.3] ETSI GS NFV 003: "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV".

[i.4] ETSI GS NFV-INF 001: "Network Functions Virtualisation (NFV); Infrastructure Overview".

[i.5] ETSI GS NFV-INF 003: "Network Functions Virtualisation (NFV); Infrastructure; Compute Domain".

[i.6] ETSI GS NFV-INF 004: "Network Functions Virtualisation (NFV); Infrastructure; Hypervisor Domain".

[i.7] ETSI GS NFV-INF 005: "Network Functions Virtualisation (NFV); Infrastructure; Network Domain".

[i.8] ETSI GS NFV-SWA 001: "Network Functions Virtualisation (NFV); Virtual Network Functions Architecture".

[i.9] OpenStack®: Cloud Software. [Online].

NOTE 1: Available at: http://www.openstack.org.

NOTE 2: The OpenStack® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. ETSI is not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

[i.10] OpenStack® Havana. [Online].

NOTE: Available at: https://www.openstack.org/software/havana/.

[i.11] ETSI GS NFV 006: "Network Functions Virtualisation (NFV) Release 2; Management and Orchestration; Architectural Framework Specification".

[i.12] DTMF: DSP0243 version 2.0.1: "Open Virtualization Format Specification".

[i.13] DTMF: DSP8023 version 2.0.1: "OVF Envelope XSD".

[i.14] DTMF version 2.38.0: "Common Informational Model (CIM) Schema".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the terms given in ETSI GS NFV 003 [i.3] and the following apply:

administrative domain: collection of systems and networks operated by a single organization or administrative authority

NOTE: The components which make up the domain are assumed to interoperate with a significant degree of mutual trust among them based on a stable trust relationship, while a transient, specific trust relationship is expected to be established for interoperating with components in other domains.

connection point: information element representing the virtual and/or physical interface that offers the network connections between instances of NS, VNF, VNFC (based on the VDU information element), PNF and a VL

NOTE: Some examples of virtual and/or physical interfaces are a virtual port, a virtual NIC address, a physical port, a physical NIC address or the endpoint of an IP VPN.

consumer: role played by a functional block that consumes certain functions exposed by another functional block

deployment flavour: template that describes a specific deployment (of a Network Service or VNF) supporting specific KPIs (such as capacity and performance)

infrastructure domain: administrative domain that provides virtualised infrastructure resources such as compute, network, and storage or a composition of those resources via a service abstraction to another Administrative Domain, and is responsible for the management and orchestration of those resources

producer: role played by a functional block that produces certain functions, and exposes them externally through public interfaces to other functional blocks

resource orchestration: subset of NFV Orchestrator functions that are responsible for global resource management governance

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the abbreviations given in ETSI GS NFV 003 [i.3] and the following apply:

ACK	Acknowledge
ADC	Application Delivery Controller
API	Application Programming Interface
BRAS	Broadband Remote Access Server
BSS	Business Support System

CFS	Customer Facing Service
CIM	Common Information Model
CMS	Cloud Management System
CP	Connection Point
CPE	Customer Premise Equipment
CPU	Central Processing Unit
CSAR	Cloud Service Archive
DMTF	Distributed Management Task Force
DPDK	Data Plane Development Kit
DSL	Digital Subscriber Line
DSLAM	DSL Access Multiplexer
E2E	End to End
ECC	Error Correcting Code
EM	Element Management
EMS	Element Management System
FCAPS	Fault, Configuration, Accounting, Performance and Security
GB	GigaByte
GRE	Generic Routing Encapsulation
HA	High Availability
HW	Hardware
IaaS	Infrastructure as a Service
ID	IDentifier
IGMP	Internet Group Management Protocol
IMS	IP Multimedia System
I/O	Input/Output
IOMMU	Input/Output Memory Management Unit
IOTLB	Input/Output Translation Lookaside Buffer
IP	Internet Protocol
IPMI	Intelligent Platform Management Interface
ISG	Industry Specification Group
KPI	Key Performance Indicator
KQI	Key Quality Indicator
KVM	Kernel Virtual Machine
L2	Layer 2
L3	Layer 3
LAN	Local Area Network
LRO	Large Receive Offload
LSO	Large Segmentation Offload
MAC	Media Access Control
MEF	Metro Ethernet Forum
MLE	Measure Launch Environment
MPLS	Multi-Protocol Label Switching
MRB	Media Resource Broker
MRF	Media Resource Function
MTU	Maximum Transmission Unit
NCT	Network Connection Topology
NF	Network Function
NFP	Network Forwarding Path
NFV	Network Functions Virtualisation
NFVI	Network Functions Virtualisation Infrastructure
NFVI-PoP	NFVI Point of Presence
NFV-MANO	NFV Management and Orchestration
NFVO	Network Functions Virtualisation Orchestrator
NIC	Network Interface Card
NMS	Network Management System
N-PoP	Network Point of Presence
NS	Network Service
NSD	Network Service Descriptor
NSR	Network Service Record
NUMA	Non-Uniform Memory Architecture
NVGRE	Network Virtualisation using Generic Routing Encapsulation
OSS	Operations Support System

OVF	Open Virtualisation Format
PCI	Peripheral Component Interconnect
PCIe	PCI express
PGW	Packet Gateway
PNF	Physical Network Function
PNFD	Physical Network Function Descriptor
PNFR	Physical Network Function Record
PRS	Problem Resolution Standard
PXE	Preboot eXecution Environment
QoS	Quality of Service
RCA	Root Cause Analysis
RDMA	Remote Direct Memory Access
REST	REpresentational State Transfer
RFS	Resource Facing Service
RSS	Receive Side Scaling
RTT	Round Trip Time
SDN	Software-Defined Networking
SHA	Secure Hash Algorithm
SID	Information Framework
SLA	Service Level Agreement
SMT	Simultaneous Multithreading
SR-IOV	Single Root Input/Output Virtualisation
S-CSCF	Service Call Session Control Function
SWA	Software Architecture Work group
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TLB	Translation Lookaside Buffer
TOSCA	Topology and Orchestration Specification for Cloud Applications
URI	Uniform Resource Identifier
vCPU	virtual CPU
VDU	Virtualisation Deployment Unit
VEPA	Virtual Ethernet Port Aggregator
vEPC	virtual Evolved Packet Core
VIM	Virtualised Infrastructure Manager
VL	Virtual Link
VLAN	Virtual LAN
VLD	Virtual Link Descriptor
VLR	Virtual Link Record
VM	Virtual Machine
vMME	virtual Mobility Management Entity
VNF	Virtualised Network Function
VNFaaS	VNF as a Service
VNFC	Virtual Network Function Component
VNFD	Virtualised Network Function Descriptor
VNFFG	VNF Forwarding Graph
VNFFGD	VNFFG Descriptor
VNFFGR	VNFFG Record
VNFM	VNF Manager
VNFR	VNF Record
VNPaaS	Virtual Network Platform as a Service
vPGW	virtual Packet Gateway
VPN	Virtual Private Network
vRAM	virtual Random Access Memory
VXLAN	Virtual eXtensible LAN
WAN	Wide Area Network
WIM	WAN Infrastructure Manager
XML	Extensible Markup Language
XSD	XML Schema Definition

4 NFV Management and Orchestration: objectives and concepts

4.1 Overview

Clause 4 provides an overview of the NFV management and orchestration objectives.

Network Functions Virtualisation (NFV) adds new capabilities to communications networks and requires a new set of management and orchestration functions to be added to the current model of operations, administration, maintenance and provisioning. In legacy networks, Network Function (NF) implementations are often tightly coupled with the infrastructure they run on. NFV decouples software implementations of Network Functions from the computation, storage, and networking resources they use. The virtualisation insulates the Network Functions from those resources through a virtualisation layer.

The decoupling exposes a new set of entities, the Virtualised Network Functions (VNFs), and a new set of relationships between them and the NFV Infrastructure (NFVI). VNFs can be chained with other VNFs and/or Physical Network Functions (PNFs) to realize a Network Service (NS).

Since Network Services (including the associated VNF Forwarding Graphs (VNFFGs), Virtual Links (VLs), Physical Network Functions (PNFs), VNFs, NFVI and the relationships between them did not exist before the emergence of NFV, their handling requires a new and different set of management and orchestration functions. The Network Functions Virtualisation Management and Orchestration (NFV-MANO) architectural framework has the role to manage the NFVI and orchestrate the allocation of resources needed by the NSs and VNFs. Such coordination is necessary now because of the decoupling of the Network Functions software from the NFVI.

The virtualisation principle stimulates a multi-vendor ecosystem where the different components of NFVI, VNF software, and NFV-MANO architectural framework entities are likely to follow different lifecycles (e.g. on procurement, upgrading, etc.). This requires interoperable standardized interfaces and proper resource abstraction among them.

The present document focuses primarily on the differences introduced by the Network Functions Virtualisation process, and not on aspects of the Network Functions that remained constant (i.e. the latter continue to be the focus of pre-existing entities). Those differences can be described by grouping them in the following hierarchy:

- virtualised infrastructure;
- virtualised network function;
- Network Service.

The elaboration of NFV-MANO aspects below is structured according to this hierarchy. The management of end to end services is not in the scope of the present document.

NOTE: The concept of reusable VNF groups is not supported in the present document.

4.2 Management and Orchestration aspects of Network Functions Virtualisation Infrastructure

Network Functions Virtualisation Infrastructure (NFVI) resources under consideration are both virtualised and non-virtualised resources, supporting virtualised network functions and partially virtualised network functions.

Virtualised resources in-scope are those that can be associated with virtualisation containers, and have been catalogued and offered for consumption through appropriately abstracted services, for example:

- Compute including machines (e.g. hosts or bare metal), and virtual machines, as resources that comprise both CPU and memory.
- Storage, including: volumes of storage at either block or file-system level.

- Network, including: networks, subnets, ports, addresses, links and forwarding rules, for the purpose of ensuring intra- and inter-VNF connectivity.

The management and orchestration of virtualised resources handles NFVI resources (e.g. in NFVI Nodes), in NFVI Points of Presence (NFVI-PoPs). Management of non-virtualised resources is restricted to provisioning connectivity to PNFs, necessary when a NS instance includes a PNF connected to a VNF, or when the NS instance is distributed across multiple NFVI-PoPs or N-PoPs.

The virtualised resources are leveraged for providing VNFs with the resources they need. Resource allocation in the NFVI is a potentially complex task because a lot of requirements and constraints are applicable at the same time. Particularly requirements for network allocation add new complexity compared to known resource allocation strategies for computing resources in virtualised environments. For example, some VNFs require low latency or high bandwidth links to other communication endpoints.

Allocation and release of resources is a dynamic process, in response to consumption of those services by other functions. While the management and orchestrations function for virtualised infrastructure are VNF-unaware, resource allocations and releases can be needed throughout the VNF lifetime. An advantage of NFV is that with increasing load VNFs can dynamically consume services that allocate additional resource when scaling-out is triggered.

Services exposing virtualised resources include (non-exhaustive list):

- discovery of available services;
- management of virtualised resources availability/allocation/release;
- virtualised resource fault/performance management.

In the case of virtualised resources distributed across multiple NFVI-PoPs, those services could either be exposed directly by the management and orchestration functions for each individual NFVI-PoP, or via a higher-level service abstraction presenting the virtualised resources across multiple NFVI-PoPs. Both types of services could be exposed to the consuming functions. In the case of the higher level service abstraction previously mentioned, the management and orchestration of virtualised resources and non-virtualised networking resources across those NFVI-PoPs falls under the responsibility of the management and orchestration of the virtualised infrastructure that can in turn use the services exposed directly by the management and orchestration functions of a single or across multiple NFVI-PoPs. In order to provide those services, the management and orchestration of the virtualised infrastructure consumes services provided by the NFVI.

The NFV management and orchestration functions that coordinate virtualised resources in a single NFVI-PoP and/or across multiple NFVI-PoPs is expected to ensure exposure of services that support accessing these resources in an open, well known abstracted manner. These services can be consumed by other authenticated and properly authorized NFV management and orchestration functions (e.g. functions that manage and orchestrate virtualised network functions).

4.3 Management and Orchestration aspects of Virtualised Network Functions

Management and orchestration aspects of a VNF include traditional Fault Management, Configuration Management, Accounting Management, Performance Management, and Security Management (FCAPS), but the focus in the present document is on the newer aspect introduced by NFV. The decoupling of Network Functions from the physical infrastructure they use results in a new set of management functions focused on the creation and lifecycle management of the needed virtualised resources for the VNF, collectively referred to as VNF Management.

VNF Management functions are responsible for the VNF's lifecycle management including operations such as:

- Instantiate VNF (create a VNF using the VNF on-boarding artefacts).
- Scale VNF (increase or reduce the capacity of the VNF).
- Update and/or Upgrade VNF (support VNF software and/or configuration changes of various complexity).
- Terminate VNF (release VNF-associated NFVI resources and return it to NFVI resource pool).

The deployment and operational behaviour requirements of each VNF is captured in a deployment template, and stored during the VNF on-boarding process in a catalogue, for future use. The deployment template describes the attributes and requirements necessary to realize such a VNF and captures, in an abstracted manner, the requirements to manage its lifecycle. The VNF Management functions perform the lifecycle management of a VNF based on the requirements in the template. At instantiation NFVI resources are assigned to a VNF based on the requirements captured in the deployment template but also taking into consideration specific requirements, constraints, and policies that have been pre-provisioned or are accompanying the request for instantiation. The inclusion of lifecycle management requirements in the deployment template allows the VNF Management functions to handle similarly a very simple VNF with a single component or a highly complex VNF with multiple components and inter-dependencies - affording flexibility to the VNF provider.

During the lifecycle of a VNF, the VNF Management functions can monitor KPIs of a VNF, if such KPIs were captured in the deployment template. The management functions can use this information for scaling operations. Scaling can include changing the configuration of the virtualised resources (scale up, e.g. add CPU, or scale down, e.g. remove CPU), adding new virtualised resources (scale out, e.g. add a new VM), shutting down and removing VM instances (scale in), or releasing some virtualised resources (scale down).

The VNF Management performs its services by maintaining the virtualised resources that support the VNF functionality, without interfering with the logical functions performed by the VNFs, and its functions are exposed in an open, well known abstracted manner as services to other functions.

The services provided by VNF Management can be consumed by authenticated and properly authorized NFV management and orchestration functions (e.g. functions that manage Network Services).

4.4 Management and Orchestration aspects of Network Services

The Network Service Orchestration is responsible for the Network Service lifecycle management including operations such as:

- On-board Network Service, i.e. register a Network Service in the catalogue and ensure that all the templates describing the NS are on-boarded.
- Instantiate Network Service, i.e. create a Network Service using the NS on-boarding artefacts.
- Scale Network Service, i.e. grow or reduce the capacity of the Network Service.
- Update Network Service by supporting Network Service configuration changes of various complexity such as changing inter-VNF connectivity or the constituent VNF instances.
- Create, delete, query, and update of VNFFGs associated to a Network Service.
- Terminate Network Services, i.e. request the termination of constituent VNF instances, request the release of NFVI resources associated to NSs, and return them to NFVI resource pool if applicable.

The deployment and operational behaviour requirements of each Network Service is captured in a deployment template, and stored during the Network Service on-boarding process in a catalogue, for future selection for instantiation. The deployment template fully describes the attributes and requirements necessary to realize such a Network Service. Network Service Orchestration coordinates the lifecycle of VNFs that jointly realize a Network Service. This includes (not limited to) managing the associations between different VNFs, and when applicable between VNFs and PNFs, the topology of the Network Service, and the VNFFGs associated with the Network Service.

During the Network Service lifecycle, the Network Service Orchestration functions can monitor KPIs of a Network Service if such requirements were captured in the deployment template, and can report this information to support explicit request for such operations from other functions.

The Network Service Orchestration performs its services by using the VNF Management services and by orchestrating the NFV Infrastructure that supports the interconnection between VNFs functionality, and its functions are exposed in an open, well known abstracted manner as services to other functions. In order to fulfil its responsibilities, the Network Service Orchestration functions consume services exposed by other functions (e.g. Virtualised Infrastructure Management functions).

The services provided by Network Service Orchestration can be consumed by authenticated and properly authorized other functions (e.g. Operations Support System (OSS), Business Support System (BSS)).

4.5 Other management and orchestration aspects of NFV framework

4.5.0 Preamble

Clause 4.5 and sub-clauses provide a high level description of some additional management and orchestration aspects of NFV, that apply to management of virtualised resources, VNFs, and NSs.

4.5.1 Fault and performance management

Fault and performance management are functionalities that are expected to exist in any NFV framework. They support the assurance aspects of the lifecycle of any Network Service instance or VNF instance, in order to provide Service Level Agreement (SLA) management. Fault and performance management functionality is typically distributed over different functional blocks that are dedicated among others to fault and performance measurement, results calculation and aggregation, fault correlation, and fault resolution.

Fault notifications can be the result of several sources of faults: physical infrastructure (compute, storage, and networking related faults); virtualised infrastructure (e.g. VM-related faults), and application logic (i.e. VNF instance related faults).

Fault correlation and root-cause analysis are processes that determine the reason for faults conditions and the impact of fault conditions. Once correlated and analysed, the correlated fault information helps determining the necessary corrective actions, and it helps triggering such actions at one or more fault resolution points, within the NFV Framework or outside the NFV Framework (e.g. OSS).

Fault correlation can be centralized or distributed among multiple functional blocks, and no assumption is made here regarding either. Ideally, it is recommended that faults are analysed and resolved as soon as possible, hence at the functional block that has sufficient information to perform the root-cause analysis and correlation and to determine the necessary corrective action.

4.5.2 Policy Management

NFV policy management refers to the management of rules governing the behaviour of NFV-MANO functions (e.g. management of VNF or NS scaling operations, access control, resource management, fault management, etc.).

Policies are defined with conditions and corresponding actions. For example, a scaling policy states to execute the related actions if the required conditions (e.g. VNF low CPU usage) were to manifest during runtime. Different actions defined on the policy can be mutually exclusive resulting in process of selecting a particular action (or set of actions) to be executed. Once declared, a policy can be bound to one or more NS instances, VNF instances, and NFVI resources. NFV-MANO is expected to support the execution of policies both automatically and manually, (e.g. by requiring manual intervention from other operation systems, e.g. OSS/BSS).

4.5.3 Testing aspects of Network Services

As part of the NS instantiation process, as well as at any other times during the NS lifecycle, there could be a need to perform testing (performance, operational, functional, etc.) for various reasons. For example, performance testing is a key step of network engineering across all levels, and its main goal is to detect problems before a service goes live. Additionally, in NFV deployments, testing has a subtle yet critical role - tuning the particular NFVI configuration. Any sufficiently complex Network Service is expected to be tested across multiple aspects, and across different network paths (subsets or end-to-end); it can be tested as a whole, in parts (segments), or in parallel. It is recommended that the test plan for a Network Service is integrated and created as part of the design process of the service. The different ways to integrate a test plan in a NS design are deliberately not covered by the present document.

The testing of a single VNF is a subset case of testing of an entire Network Service. In particular it is a necessary step before placing a VNF "in service", e.g. on a new NFVI and in cases where it is determined that the VNF fails to perform at the expected levels. Healing a VNF is a process that in some cases can require functional testing, before determining the appropriate healing procedure.

NS testing is expected to have no impact on the services provided by the NS.

NS testing procedures can be described in appropriate templates, supporting automation via the Network Service Orchestration functions, or administered manually.

4.6 Relation of NFV management and orchestration with existing operations and management systems

4.6.1 Overview

NFV-MANO functions play an important role in delivering the business benefits envisioned by the NFV ISG. These benefits are achieved through interworking with other functional components in the NFV framework, as well as external entities, such as Operations and Business Support Systems (OSS/BSS).

OSS/BSS include the collection of systems and management applications that service providers use to operate their business. NFV is disruptive to many OSS/BSS functions, as the success of such a technology enabled business transformation is predicated on changes in mind-sets, skillsets and toolsets to support corresponding process re-engineering efforts.

NFV is a paradigm shift in how the networks that underpin today's service provider infrastructures are built and operated, and how the services they deliver are managed. New degrees of freedom are introduced to the network and its management as resources now can be added, changed, and removed dynamically. This change opens up a wave of new business opportunities. However, a new and highly agile operational approach is expected to take full advantage of these opportunities.

4.6.2 NFV-MANO Interworking with OSS/BSS to deliver NFV business benefits

NFV-MANO alone cannot deliver all the NFV business benefits; it is expected to integrate and interwork with other management entities for this purpose (e.g. OSS, BSS), using interfaces offered by those entities and offering its own interfaces to be used by external entities.

In order for service providers to achieve the full benefit of NFV, it is recommended to consider NFV-MANO solutions holistically alongside OSS/BSS integration and management requirements. Simply extending existing OSS/BSS models to account for virtualisation will not be sufficient, because this approach will not support the new value-added capabilities and services provided by NFV. Efforts are needed to ensure that NFV-MANO and OSS/BSS evolution is coordinated so as to jointly support the following:

- Open and consistent interfaces, to facilitate automation, self-service operations at service and product level that can respond with the speed and agility required by changing business needs.
- Adaptive automation, where service usage drives on-demand resource requirements, triggering feedback from the system that the management functions analyse and make changes to, enabling the infrastructure to provide the resources and services needed at that point in time.
- Orchestration, where policies (and other mechanisms) can guide the decisions required to change all or part of the system to perform a given function.
- Personalized services that are easily configured, by the operator and/or end-user at the service and/or network resource layers, to fit individual customer preferences and requirements.
- Technology-driven innovation, where rapid development, continuous integration, deployment, and experimentation, meet business and service operations agility and enable the migration to next generation operations.

4.6.3 Key Challenges and Considerations

Management of end-to-end services by OSS/BSS requires convergence on a common approach to presenting management services and management information from both legacy network systems and NFV based systems, so that accurate end-to-end management views can be derived.

Standards are essential to achieve this convergence process. However, experience has taught the industry that this is already a major issue for converged fixed and mobile network management including applications and services. It will be even more acute when managing end-to-end services across a mixture of NFV functions, infrastructure, and legacy interconnected network systems, in a highly dynamic NFV environment.

Some design patterns are used for converged fixed and mobile network operations. These patterns are shared between interfaces, and are exposed by diverse network management systems. This enables a common way to integrate an end-to-end view across multiple resources:

- To describe the semantics and relationships of information exchanged across management interfaces, a federated information model that provides common and consensually defined terms, concepts, and objects is required.
- To lower integration complexity, a set of common interface operations are required for covering request and response patterns, as well as the behaviour and sequencing of those requests and responses for each of fault, configuration, accounting, performance, inventory, and security management.
- To improve consistency in representing information in end-to-end applications, common data types and common vocabularies are required. For example: specification of fault codes and location codes used across multiple resources.

This pattern based approach, developed from prior converged network management studies, can help to inform and guide future NFV standards, open source and proof of concept project development and their integration approaches. More importantly, it can help the industry in managing converged networks (fixed and mobile) with a similar methodology and complementary functional realizations.

Another key challenge is automation. Operators want to speed up the ability to dynamically incorporate change. The agile management of network functions, applications, and services provides new opportunities for revenue while reducing costs. Automation can be supported by an aligned approach between the OSS/BSS and the NFV-MANO functionality based on, for example:

- Open and standardized interfaces.
- Common information models.
- Mappings to different data models.
- Ability to incorporate real-time data analytics.

The key challenge in this area is to provide a flexible and dynamic policy-based management approach.

Furthermore, the need for real-time processing (in conjunction with offline processing) is an additional key challenge that plays an increasingly important role. It is desirable to support real-time processing by automation.

The support of the processing of a huge amount of data, including data analytics, based on several data sources (e.g. structured, semi-structured, and unstructured data from the infrastructure as well as other sources), also in real time, is a further key challenge in the OSS/BSS and NFV context.

To address the need for dynamic operation of NFV management and orchestration, the management architecture supports run-time integration of management functions and processes. This is different from today's static integration, as it implies that interfaces are adaptive to support business agility.

In other words, the interfaces in NFV-MANO do not mandate a specific list of end points based on the information being exchanged; in contrast, they can instead allow for different entities to consume the information based on service necessity. A loosely-coupled set of interfaces will allow services and applications, as well as associated business and operational processes, to be data-driven and controlled by policies. This adaptive approach further enables a steady evolution of existing legacy systems to integrate with NFV.

4.7 Administrative Domains

In a typical scenario for NFV there will not be a single organization controlling and maintaining a whole NFV system. The organizations either can be departments of the same root organization e.g. a network department and an IT datacentre department or they can be different companies providing different functional blocks of the NFV Architectural Framework. The basic use cases for these scenarios are described in ETSI GS NFV 001 [i.1].

Administrative Domains can be mapped to different organizations and therefore can exist within a single service provider or distributed among several service providers. Administrative Domains can be collapsed, resulting in different deployment options.

For the purpose of the present document, the following administrative types of domains are identified:

- Infrastructure Domain.
- Tenant Domain.

NOTE: The present document only focuses on two types of Administrative Domains; additional domain types can be identified later to support different deployment scenarios.

An Infrastructure Domain can be defined by different criteria (e.g. by organization, by type of resource such as networking, compute and storage as in traditional datacentre environments, by geographical location, etc.), and multiple Infrastructure Domains can co-exist. An Infrastructure Domain can provide infrastructure to a single or multiple Tenant Domains. The Infrastructure Domain is application agnostic and thus has no insight of what is executed within a VNF.

A Tenant Domain can be defined by different criteria (e.g. organization, by type of Network Service, etc.), and multiple Tenant Domains can co-exist. A Tenant Domain can use infrastructure in a single or multiple Infrastructure Domains.

While the VNFs and Network Services reside in the Tenant Domain, in order to materialize they need the NFVI resources from the Infrastructure Domains. Therefore the Tenant Domain consumes resources from one or more Infrastructure Domains using the Infrastructure Domain orchestration functionality to orchestrate and operate virtual infrastructure resources required by VNFs and Network Services built using those VNFs. Typical management tasks within the Tenant Domain are VNF FCAPS. Typical management and orchestration tasks requiring the involvement of both the Tenant Domain and the Infrastructure Domain are on-boarding of VNFs, instantiation of VNFs and scaling of VNFs. The Tenant Domain is application aware, in the sense that functional blocks in this Administrative Domain understand and support the logical functionality of the VNFs.

Other typical relationships between a Tenant Domain and an Infrastructure Domain include (non-exhaustive):

- A Tenant Domain can use VNFs belonging to that Tenant Domain and/or to other Tenant Domains, in order to realize the Tenant Domain's Network Services.
- A Tenant Domain and an Infrastructure Domain could be mapped against the same or different organizations (in a particular in the case when the Infrastructure Domain is offering infrastructure to a Tenant Domain).
- While typically an Infrastructure Domain provides infrastructure services to a Tenant Domain, it can also provide infrastructure services to one or more other Infrastructure Domains.

The identification of different Administrative Domains leads to the requirement to use different management and orchestration functionality in those domains. The cross-relationships between the different Administrative Domains leads to the identification of the requirement that NFV-MANO functionality is not monolithic, but rather is expected to be layered: management and orchestration functionality in an Administrative Domain can provide services that are consumed by management and orchestration functionality in a different Administrative Domain.

However, the specific mapping of Administrative Domains to NFV-MANO functions in different layers is deliberately left out of scope for the present document, since many different mappings can be supported, depending on the specifics of the deployment environment.

5 Management and Orchestration architectural framework

5.1 Overview

This clause expands on the NFV Architectural Framework (ETSI GS NFV 002 [i.2]) by providing a functional architecture with more granular distribution and a more detailed description of the NFV Management and Orchestration (NFV-MANO) functionality, as well as a description of the reference points between NFV-MANO functional blocks and other functional blocks in the E2E NFV reference architecture. The NFV-MANO architectural framework focuses on the aspects brought into the operator's networks by NFV, hence mainly on the new functional blocks and reference points between those functional blocks. Other functional blocks and reference that could be necessary for a better understanding of the NFV-MANO architectural framework can be included and briefly described, but the detailed description for those are in-scope for other NFV documents, which the present document references.

5.2 Principles of NFV-MANO

The NFV-MANO architectural framework relies on a set of principles that support the combination of the concepts of distinct Administrative Domains and layered management and orchestration functionality in each of those domains:

- The architecture is expected to include the possibility of multiple NFV-MANO functional blocks in the Tenant Domain leveraging resources in the same Infrastructure Domain, and a general architectural principle of horizontal equality between those NFV-MANO functional blocks is also expected.
- Orchestration is provided in multiple functional blocks and it is important to emphasize that there is no one functional block that has primacy over others. A general architectural principle of equality used in distribution of orchestration functionality is expected.
- It is recommended that the NFV-MANO functional blocks in the Infrastructure Domain offers abstracted services to functional blocks in the Tenant Domain and that the aspects of resource management corresponding to these abstracted services are fully embedded as selectable resource policy within them.
- A possibility to leverage best cloud management practices such assignment of resources, according to the availability of the abstract services presented by the Infrastructure Domain to the Tenant Domain is recommended.
- The NFV-MANO functionality can be realized in different ways, for example, as a monolithic single instance, as a scalable system with multiple load-sharing instances, as a system of distributed cooperating instances, or as a functionally decomposed and distributed system. It can be realized as an extension of a cloud/network management system or as a separate system interacting with cloud/network management systems for realizing NFV.

The present document does not mandate any specific realization of the NFV-MANO architectural framework, and instead it recommends that the following best practices are leveraged and that the NFV-MANO architectural framework:

- lend itself to virtualisation and be implementable in software only;
- lend itself to possible distribution and scaling across the NFVI in order to improve service availability and support different locations;
- lend itself to full automation (ability to react to events in real-time without human intervention, and execute actions associated with those events, based on pre-provisioned templates and policies);
- lend itself to implementations that do not contain any single points of failures with the potential to endanger service continuity;
- lend itself to an implementation with an open architecture approach and expose standard or "de-facto" standard interfaces;
- support and help realize the feasible decoupling of VNF software from the hardware;

- support management and orchestration of VNFs and Network Services using NFVI resources from a single or across multiple NFVI-PoPs;
- support modelling of NFVI resource requirements of VNFs in an abstracted way;
- support modelling of NFVI resources in a way that an abstraction of them can be exposed by functionality in one layer, to functionality in a different layer.

5.3 NFV-MANO architectural framework overview

The following entities from the NFV architectural framework (ETSI GS NFV 002 [i.2]) are considered within the scope of the NFV-MANO architectural framework:

a) Functional blocks identified as belonging to NFV Management and Orchestration (NFV-MANO).

b) Other functional blocks that interact with NFV-MANO via reference points.

c) All reference points that enable communications to, from, and within NFV-MANO.

The NFV-MANO architectural framework represented in figure 5.1 is at a functional level and it does not imply any specific implementation. Multiple functional blocks can be merged internalizing the reference point between them. Finally, reference points drawn in bold and continuous lines and labelled are considered new and critical for NFV-MANO architectural framework, representing potential targets for development in open source projects and/or later standardization.

The NFV-MANO architectural framework follows the principles enumerated in clause 5.1. Each of the functional blocks has a well-defined set of responsibilities and operates on well-defined entities, using management and orchestration as applicable within the functional block, as well as leveraging services offered by other functional blocks.

The NFV-MANO architectural framework identifies the following NFV-MANO functional blocks:

- Virtualised Infrastructure Manager (VIM).
- NFV Orchestrator (NFVO).
- VNF Manager (VNFM).

NFV-MANO architectural framework identifies the following data repositories:

- NS Catalogue.
- VNF Catalogue.
- NFV Instances repository.
- NFVI Resources repository.

The NFV-MANO architectural framework identifies the following functional blocks that share reference points with NFV-MANO:

- Element Management (EM).
- Virtualised Network Function (VNF).
- Operation System Support (OSS) and Business System Support functions (BSS).
- NFV Infrastructure (NFVI).

The NFV-MANO architectural framework identifies the following main reference points:

- Os-Ma-nfvo, a reference point between OSS/BSS and NFVO.
- Ve-Vnfm-em, a reference point between EM and VNFM.
- Ve-Vnfm-vnf, a reference point between VNF and VNFM.

- Nf-Vi, a reference point between NFVI and VIM.
- Or-Vnfm, a reference point between NFVO and VNFM.
- Or-Vi, a reference point between NFVO and VIM.
- Vi-Vnfm, a reference point between VIM and VNFM.

Table 5.1 illustrates the mapping between the named reference points defined in ETSI GS NFV 002 [i.2] and the named reference points defined in the present document.

Table 5.1

Reference point in ETSI GS NFV 002 [i.2]	Reference point in the present document	Notes
Os-Ma	Os-Ma-nfvo	The Os-Ma reference point in ETSI GS NFV 002 [i.2] terminates at the NFV-MANO boundary. In the present document, the Os-Ma-nfvo reference point is derived from Os-Ma, and terminates at the NFVO boundary. In the present document, no other reference points have been identified between OSS/BSS and NFV-MANO functional blocks
Or-Vnfm	Or-Vnfm	Unchanged
Ve-Vnfm	Ve-Vnfm-em, Ve-Vnfm-vnf	The Ve-Vnfm reference point in ETSI GS NFV 002 [i.2] terminates at a boundary comprising multiple functional blocks. In the present document, the Ve-Vnfm-em and the Ve-Vnfm-vnf reference points are derived from Ve-Vnfm, and Ve-Vnfm-em terminates at the EM boundary, while Ve-Vnfm-vnf terminates at the VNF boundary
Vi-Vnfm	Vi-Vnfm	Unchanged
Or-Vi	Or-Vi	Unchanged
Nf-Vi	Nf-Vi	Unchanged
Vn-Nf	Vn-Nf	Unchanged

Other unnamed reference points have been added between NFV Orchestrator and the VNF Catalogue, VNFM and the VNF Catalogue, NFV Orchestrator and the NFV Service Catalogue, NFV Orchestrator and the NFV Instances repository, as well as between NFV Orchestrator and NFVI Resources repository.

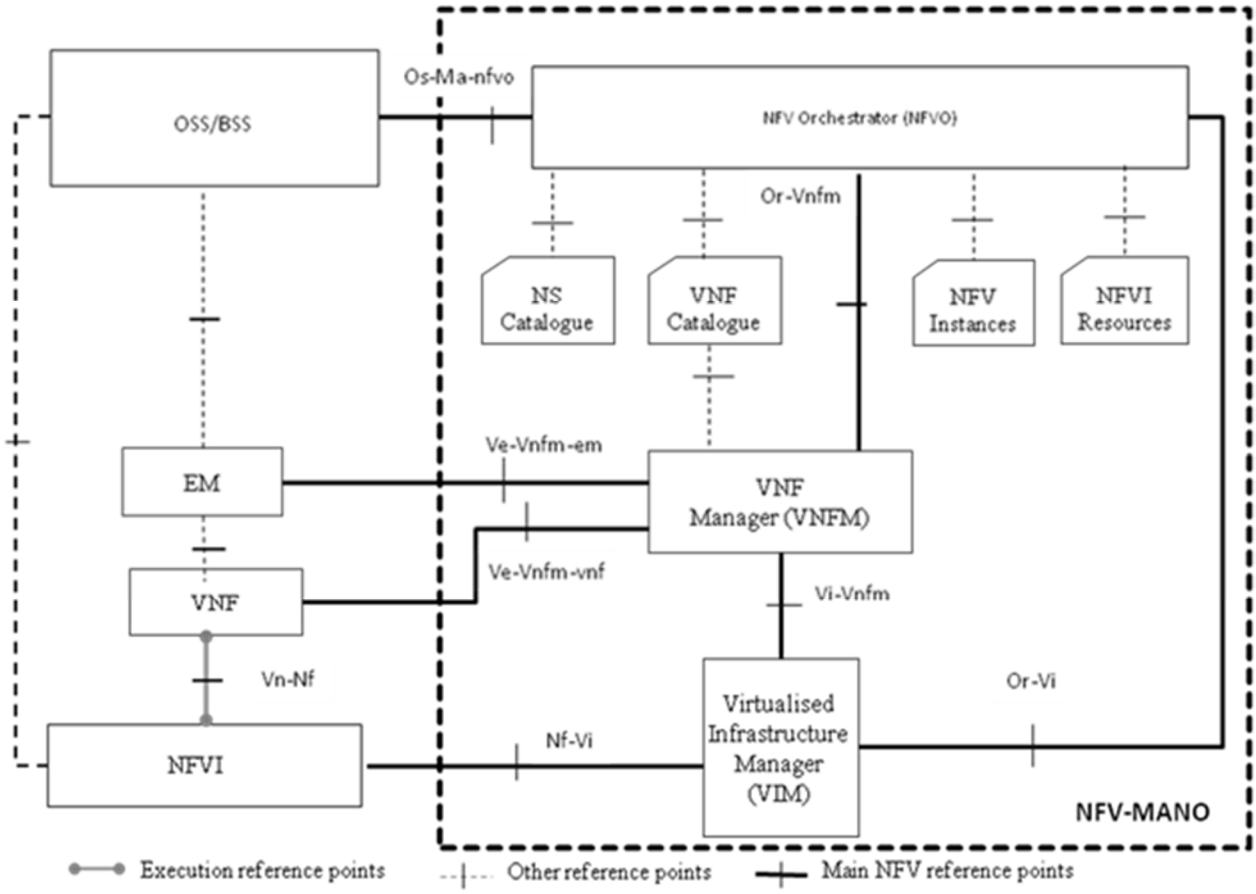


Figure 5.1: The NFV-MANO architectural framework with reference points

The following clauses define the functional blocks and the reference points in-scope for the NFV-MANO architectural framework.

5.4 NFV-MANO architectural framework functional blocks

5.4.1 NFV Orchestrator (NFVO)

Clause 5 describes the NFV-MANO architectural framework functional blocks and reference points.

The NFV Orchestrator has two main responsibilities:

- the orchestration of NFVI resources across multiple VIMs, fulfilling the Resource Orchestration functions described in clause 4.2;
- the lifecycle management of Network Services, fulfilling the Network Service Orchestration functions, as described in clause 4.4.

NOTE: In the present document, for simplicity reasons, the two responsibilities are kept within one functional block and sharing a common information base represented by the NFV Instances and NFV Resources repository. To support different multi-vendor deployments and/or different mappings of functionality to Administrative Domains, the two responsibilities can be separated in future. The NFVO uses the Network Service Orchestration functions to coordinate groups of VNF instances as Network Services that jointly realize a more complex function, including joint instantiation and configuration, configuring required connections between different VNFs, and managing dynamic changes of the configuration, e.g. for scaling the capacity of the Network Service. The Network Service Orchestration function uses the services exposed by the VNF Manager function and by the Resource Orchestration function.

The following list expresses the non-exhaustive set of capabilities provided by the NFVO, via its Network Service Orchestration functions. These capabilities can be exposed by means of interfaces consumed by NFV-MANO functional blocks or by authorized external entities:

- Management of Network Services deployment templates and VNF Packages (e.g. on-boarding new Network Services and VNF Packages). During on-boarding of NS and VNF, a validation step is required. To support subsequent instantiation of a NS, respectively a VNF, the validation procedure is expected to verify the integrity and authenticity of the provided deployment template, and that all mandatory information is present and consistent. In addition, during the on-boarding of VNFs, software images provided in the VNF Package for the different VNF components are catalogued in one or more NFVI-PoPs, using the support of VIM.
- Network Service instantiation and Network Service instance lifecycle management, e.g. update, query, scaling, collecting performance measurement results, event collection and correlation, termination.
- Management of the instantiation of VNF Managers where applicable.
- Management of the instantiation of VNFs, in coordination with VNF Managers.
- Validation and authorization of NFVI resource requests from VNF Managers, as those can impact Network Services (granting of the requested operation is expected to be governed by policies).
- Management of the integrity and visibility of the Network Service instances through their lifecycle, and the relationship between the Network Service instances and the VNF instances, using the NFV Instances repository.
- Management of the Network Service instances topology (e.g. create, update, query, delete VNF Forwarding Graphs).
- Network Service instances automation management (e.g. trigger automatic operational management of NS instances and VNF instances, according to triggers and actions captured in the on-boarded NS and VNF deployment templates and governed by policies applicable to those NS and VNF instances).
- Policy management and evaluation for the Network Service instances and VNF instances (e.g. policies related with affinity/anti-affinity, scaling, fault and performance, geography, regulatory rules, NS topology, etc.).

The NFVO uses the Resource Orchestration functionality to provide services that support accessing NFVI resources in an abstracted manner independently of any VIMs, as well as governance of VNF instances sharing resources of the NFVI infrastructure.

The following list expresses the non-exhaustive set of functionalities performed by the Resource Orchestration function. These functionalities can be exposed by means of interfaces and consumed by NFV-MANO functional blocks or by authorized external entities:

- Validation and authorization of NFVI resource requests from VNF Manager(s), as those can impact the way the requested resources are allocated within one NFVI-PoP or across multiple NFVI-PoPs (granting of the requested resources is governed by policies, and could require prior reservation).
- NFVI resource management across operator's Infrastructure Domains including the distribution, reservation and allocation of NFVI resources to Network Service instances and VNF instances by using an NFVI resources repository, as well as locating and/or accessing one or more VIMs as needed and providing the location of the appropriate VIM to the VNFM, when required.
- Supporting the management of the relationship between the VNF instances and the NFVI resources allocated to those VNF instances by using NFVI Resources repository and information received from the VIMs.
- Policy management and enforcement for the Network Service instances and VNF instances (e.g. NFVI resources access control, reservation and/or allocation policies, placement optimization based on affinity and/or anti-affinity rules as well as geography and/or regulatory rules, resource usage, etc.).
- Collect usage information of NFVI resources by VNF instances or groups of VNF instances, for example, by collecting information about the quantity of NFVI resources consumed via NFVI interfaces and then correlating NFVI usage records to VNF instances.

5.4.2 VNF Manager (VNFM)

The VNF Manager is responsible for the lifecycle management of VNF instances as described in clause 4.3. Each VNF instance is assumed to have an associated VNF Manager. A VNF manager can be assigned for the management of a single VNF instance, or the management of multiple VNF instances of the same type or of different types.

Most of the VNF Manager functions are assumed to be generic common functions applicable to any type of VNF. However, the NFV-MANO architectural framework is expected to also support cases where VNF instances need specific functionality for their lifecycle management, and such functionality can be specified in the VNF Package.

The following list expresses the non-exhaustive set of functions performed by the VNF Manager function. These functionalities can be exposed by means of interfaces and consumed by other NFV-MANO functional blocks or by authorized external entities:

- VNF instantiation, including VNF configuration if required by the VNF deployment template (e.g. VNF initial configuration with IP addresses before completion of the VNF instantiation operation).
- VNF instantiation feasibility checking, if required.
- VNF instance software update/upgrade.
- VNF instance modification.
- VNF instance scaling out/in and up/down.
- VNF instance-related collection of NFVI performance measurement results and faults/events information, and correlation to VNF instance-related events/faults.
- VNF instance assisted or automated healing.
- VNF instance termination.
- VNF lifecycle management change notification.
- Management of the integrity of the VNF instance through its lifecycle.
- Overall coordination and adaptation role for configuration and event reporting between the VIM and the EM.

The deployment and operational behaviour of each VNF is captured in a template called Virtualised Network Function Descriptor (VNFD) that is stored in the VNF catalogue. NFV-MANO uses a VNFD to create instances of the VNF it represents, and to manage the lifecycle of those instances. A VNFD has a one-to-one correspondence with a VNF Package, and it fully describes the attributes and requirements necessary to realize such a VNF. NFVI resources are assigned to a VNF based on the requirements captured in the VNFD (containing resource allocation criteria, among others), but also taking into consideration specific requirements, constraints, and policies that have been pre-provisioned or are accompanying the request for instantiation and can override certain requirements in the VNFD (e.g. operator policies, geo-location placement, affinity/anti-affinity rules, local regulations).

The information elements to be handled by the NFV-MANO, including the VNFD among others, will be designed to guarantee the flexible deployment and portability of VNF instances on multi-vendor and diverse NFVI environments, e.g. with diverse computing resource generations, diverse virtual network technologies, etc. To achieve this, hardware resources will be properly abstracted and the VNF requirements will be described in terms of such abstraction.

The VNFM has access to a repository of available VNF Packages and different versions of them, all represented via their associated VNFDs. Different versions of a VNF Package can correspond to different implementations of the same function, different versions to run in different execution environments (e.g. on different hypervisors, dependent on NFVI resources availability information, etc.), or different release versions of the same software. The repository can be maintained by the NFVO or another external entity.

5.4.3 Virtualised Infrastructure Manager (VIM)

The Virtualised Infrastructure Manager (VIM) is responsible for controlling and managing the NFVI compute, storage and network resources, usually within one operator's Infrastructure Domain (e.g. all resources within an NFVI-PoP, resources across multiple NFVI-PoPs, or a subset of resources within an NFVI-PoP) as described in clause 4.2. A VIM can be specialized in handling a certain type of NFVI resource (e.g. compute-only, storage-only, networking-only), or can be capable of managing multiple types of NFVI resources (e.g. in NFVI-Nodes).

EXAMPLE: A VIM exposes northbound open interfaces that support management of NFVI virtualised compute, storage, and networking resources.

The southbound interfaces interface with a variety of hypervisors and Network Controllers in order to perform the functionality exposed through its northbound interfaces. Other VIM implementations can directly expose the interfaces exposed by such compute, storage, Network Controllers as specialized VIMs. A particular example of a specialized VIM is a WAN Infrastructure Manager (WIM), typically used to establish connectivity between PNF endpoints in different NFVI-PoPs. The WIM and the overall topic of network connectivity management are described in clause 5.6. The VIM implementation is out of scope for NFV-MANO; however the interfaces exposed by VIMs are in-scope.

The following list expresses the set functions performed by the VIM. These functionalities can be exposed by means of interfaces consumed by other NFV-MANO functional blocks or by authorized external entities:

- Orchestrating the allocation/upgrade/release/reclamation of NFVI resources(including the optimization of such resources usage), and managing the association of the virtualised resources to the physical compute, storage, networking resources. Therefore, the VIM keeps an inventory of the allocation of virtual resources to physical resources, e.g. to a server pool (ETSI GS NFV-INF 004 [i.6]).

- Supporting the management of VNF Forwarding Graphs (create, query, update, delete), e.g. by creating and maintaining Virtual Links, virtual networks, sub-nets, and ports, as well as the management of security group policies to ensure network/traffic access control (ETSI GS NFV-INF 005 [i.7]).

- Managing in a repository inventory related information of NFVI hardware resources (compute, storage, networking) and software resources (e.g. hypervisors), and discovery of the capabilities and features (e.g. related to usage optimization) of such resources.

NOTE 1: Inventory repositories of NFVI hardware and software resources managed by the VIM are not explicitly shown in the NFV-MANO architectural framework.

- Management of the virtualised resource capacity (e.g. density of virtualised resources to physical resources), and forwarding of information related to NFVI resources capacity and usage reporting.

- Management of software images (add, delete, update, query, copy) as requested by other NFV-MANO functional blocks (e.g. NFVO). While not explicitly shown in the NFV-MANO architectural framework, the VIM maintains repositories for software images, in order to streamline the allocation of virtualised computing resources. A validation step, performed by VIM, is required for software images before storing the image (e.g. VNF package on-boarding and update). Image validation operations during run-time, e.g. during instantiation or scaling, are outside the scope of the current version of the present document.

- Collection of performance and fault information (e.g. via notifications) of hardware resources (compute, storage, and networking) software resources (e.g. hypervisors), and virtualised resources (e.g. VMs); and forwarding of performance measurement results and faults/events information relative to virtualised resources.

- Management of catalogues of virtualised resources that can be consumed from the NFVI. The elements in the catalogue can be in the form of virtualised resource configurations (virtual CPU configurations, types of network connectivity (e.g. L2, L3), etc.), and/or templates (e.g. a virtual machine with 2 virtual CPUs and 2 GB of virtual memory).

NOTE 2: The virtualised resources catalogues are not explicitly shown in the NFV-MANO architectural framework.

5.4.4 NS Catalogue

The NS Catalogue represents the repository of all of the on-boarded Network Services, supporting the creation and management of the NS deployment templates (Network Service Descriptor (NSD), Virtual Link Descriptor (VLD), and VNF Forwarding Graph Descriptor (VNFFGD) via interface operations exposed by the NFVO. Clause 6 describes in detail the type of information elements captured in the mentioned descriptors.

5.4.5 VNF Catalogue

The VNF Catalogue represents the repository of all of the on-boarded VNF Packages, supporting the creation and management of the VNF Package (VNF Descriptor (VNFD), software images, manifest files, etc.) via interface operations exposed by the NFVO. Both NFVO and VNFM can query the VNF Catalogue for finding and retrieving a VNFD, to support different operations (e.g. validation, checking instantiation feasibility). Clause 6 describes in detail the type of information elements captured in the VNFD.

5.4.6 NFV Instances repository

The NFV Instances repository holds information of all VNF instances and Network Service instances. Each VNF instance is represented by a VNF record, and each NS instance is represented by an NS record. Those records are updated during the lifecycle of the respective instances, reflecting changes resulting from execution of NS lifecycle management operations and/or VNF lifecycle management operations. This supports NFVO's and VNFM's responsibilities in maintaining the integrity and visibility of the NS instances, respectively VNF instances, and the relationship between them. Clause 6 describes in detail the type of information elements captured in the NS record and VNF record.

5.4.7 NFVI Resources repository

The NFVI Resources repository holds information about available/reserved/allocated NFVI resources as abstracted by the VIM across operator's Infrastructure Domains, thus supporting information useful for resources reservation, allocation and monitoring purposes. As such, the NFVI Resources repository plays an important role in supporting NFVO's Resource Orchestration and governance role, by allowing NFVI reserved/allocated resources to be tracked against the NS and VNF instances associated with those resources (e.g. number of VMs used by a certain VNF instance at any time during its lifecycle).

5.5 Other functional blocks

5.5.0 Preamble

The functional blocks described in clause 5.5 are not considered part of NFV-MANO, but are described because they exchange information with NFV-MANO functional blocks.

5.5.1 Element Management (EM)

The Element Management is responsible for FCAPS management functionality for a VNF. This includes:

- Configuration for the network functions provided by the VNF.
- Fault management for the network functions provided by the VNF.
- Accounting for the usage of VNF functions.
- Collecting performance measurement results for the functions provided by the VNF.
- Security management for the VNF functions.

The EM can be aware of virtualisation and collaborate with the VNF Manager to perform those functions that require exchanges of information regarding the NFVI Resources associated with the VNF.

5.5.2 Operations Support System/Business Support System (OSS/BSS)

The OSS/BSS are the combination of the operator's other operations and business support functions that are not otherwise explicitly captured in the present architectural framework, but are expected to have information exchanges with functional blocks in the NFV-MANO architectural framework. OSS/BSS functions can provide management and orchestration of legacy systems and can have full end to end visibility of services provided by legacy network functions in an operator's network.

5.5.3 Network Functions Virtualisation Infrastructure (NFVI)

Within the scope of the present document, the NFVI encompasses all the hardware (e.g. compute, storage, and networking) and software (e.g. hypervisors) components that together provide the infrastructure resources where VNFs are deployed.

The NFVI can also include partially virtualised NFs. Examples of such partially virtualised network functions are related to "white box" switches, hardware load balancers, DSL Access Multiplexers (DSLAMs), Broadband Remote Access Server (BRAS), Wi-Fi® access points, CPEs, etc., for which a certain part of the functionality is virtualised and is in scope of NFV-MANO while other parts are built in silicon (PNF) either due to physical constraints (e.g. digital interfaces to analogue physical channels) or vendor design choices. The present document does not cover the management of PNFs and it is assumed here that it is being taken care of by some other entity, for example the OSS/BSS or an Network Controller. The present document covers the configuration of the connectivity between adjacent VNF(s) and PNF(s) that are comprised in the same Network Service.

5.6 Network Controllers

5.6.1 Overview

This clause describes the roles and responsibilities of Network Controllers, the VIM and the NFVO specifically regarding the orchestration of the intra-NFVI-PoP and inter-NFVI-PoP connectivity between components of a given VNF or various VNFs in a VNF Forwarding Graph as well as connectivity to the PNFs.

The Network Controllers, when present, provide intuitive programmatic interfaces along the lines of the network interfaces called out in clause 7 and thus can be best categorized as an abstraction layer below the VIM for a given NFVI-PoP. Each NFVI-PoP or Administrative Domain can include a Network Controller (e.g. SDN controller) responsible for providing the programmable network interfaces that enable the establishment of connectivity within the domain.

There are various types of Network Controllers categorized by:

- the type of network which it is virtualising (NFVI-PoP Network Controller vs. WAN Network Controller, etc.);
- how the virtualisation is performed (overlays vs. partitioning);
- the underlying technology and its southbound interfaces.

A given NFVO Resource Orchestration function can consume "Network as a Service" interfaces being exposed by multiple VIMs to provision end-to-end virtual networks. The virtual network is a service provided by the NFVI. Virtual networks establish connectivity between components of a VNF or are used to implement a VNF Forwarding Graph (VNFFG). They have well defined service characteristics.

In order to facilitate VIMs support of multiple Network Controllers from multiple vendors in their Administrative Domain (via southbound interfaces), it is necessary to develop well-defined interfaces between VIMs and Network Controllers.

5.6.2 NFVO, VIM and Network Controllers

Network Controllers can form a hierarchy in a client/server relationship where each "server" Network Controller exposes an interface to request connectivity services, i.e. virtual networks, to its clients. At the lowest layer, Network Controllers have visibility into the devices they control directly. At the highest layer, Network Controllers provide connectivity services to an application and provide abstraction of the underlying resources. Each layer in the hierarchy provides a different level of abstraction and can establish connectivity services by configuring the forwarding tables of the Network Functions within its domain directly or by requesting connectivity from "server" Network Controllers or a combination of both.

Figure 5.2 shows a hybrid network environment example illustrating the goal of NFV to have fully programmatic open interfaces for service and Resource Orchestration within and across NFVI-PoPs. The figure shows:

- An example NFVI network that provides the necessary connectivity services to establish an end-to-end service between Endpoint 1 and Endpoint 2 through VNF 1 and VNF 2 as shown by the VNFFG representation.
- A model where the WAN between NFVI-PoPs provides Virtual Links based on network connectivity services between NFVI-PoPs, and where the NFVO requests Virtual Links based on network connectivity services through the Or-Vi reference point. This implies Resource Orchestration, an NFVO function, to have an abstracted view of the network topology and interfaces to underlying VIMs and WAN Infrastructure Manager (WIM) to request connectivity services.
- One VIM per NFVI-PoP but this is not mandated as there could be multiple VIMs in a single NFVI-PoP, each responsible for an Administrative Domain. Similarly, a single VIM could be responsible for multiple NFVI-PoPs. There could be multiple WIMs. Each VIM/WIM provides connectivity services within its domain and provides an abstraction of the underlying resources through a well-defined interface.

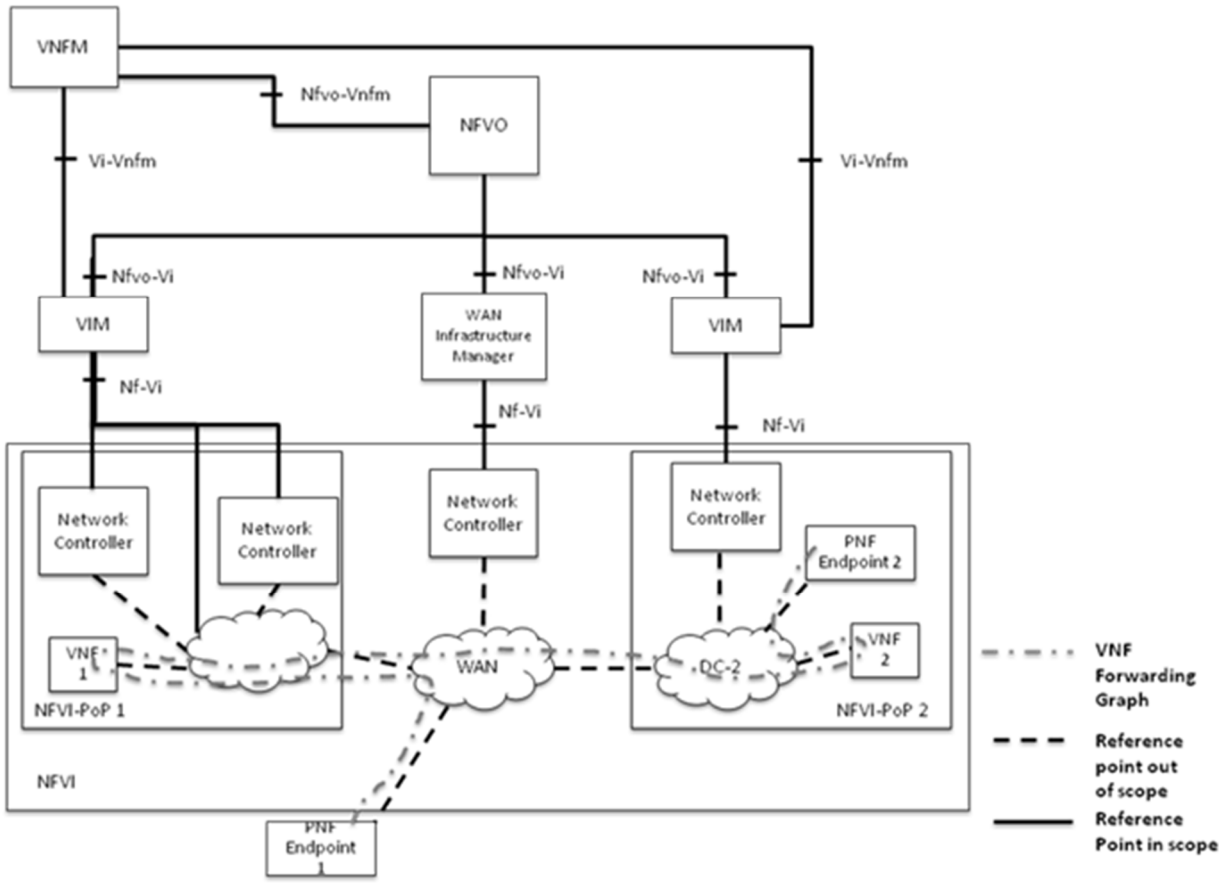


Figure 5.2: Network Controller example

For this example, multiple connectivity services are requested by the Resource Orchestration component of the NFVO:

- Virtual network between gateway in NFVI-PoP 1 and VNF1.

- Virtual network between gateway in NFVI-PoP 2 and VNF2.
- Virtual network between gateway in NFVI-PoP 1 and gateway in NFVI-PoP 2.
- Physical connectivity service between the physical Endpoint 1 and a gateway in NFVI-PoP 1.
- Physical connectivity service between the physical Endpoint 2 and a gateway in NFVI-PoP 2.

This requires each NFVI-PoP VIM to establish a virtual network within its NFVI-PoP, the WIM to establish a virtual network between the NFVI-PoP 1 gateway and the NFVI-PoP 2 gateway, the WIM to establish connectivity between the PNF endpoint1 and NFVI-PoP 1 gateway, and the NFVI-PoP 2 VIM to establish connectivity between PNF endpoint 2 and the NFVI-PoP 2 gateway.

A VIM and/or a WIM can either:

- Interface with the underlying Network Controllers to request virtual connectivity services. In such a case, the VIMs/WIMs interface with the Network Controllers over the Nf-Vi reference point. There could be multiple Network Controllers under a VIM (for example, if different virtual network partitioning techniques are used within the domain); in this case, the VIM is responsible to request virtual networks from each underlying Network Controller and setup the interworking function between them. This acknowledges the fact that there might be existing Network Controllers in the NFVI already deployed and in use for connectivity prior to the instantiation of the VIM/NFVO.
- Establish the connectivity services directly by configuring the forwarding tables of the underlying Network Functions, hence becoming the Network Controller part of the VIM (although it is not explicitly shown in the figure). In such a case, the VIM directly controls some elements, for example, a software switch (a.k.a. vSwitch). This could also be the case if there are Network Functions that do not fall within the Administrative Domain of a Network Controller.

The result is a hierarchy of control and orchestration of network resources to provide the connectivity services, from the bottom up: Network Function, Network Controllers, VIM/WIM and NFVO Resource Orchestration. Each level in the hierarchy provides abstraction of resources, a northbound interface to request connectivity services and the connectivity services themselves. The higher level Network Controllers have higher level abstractions while the lower level Network Controllers have lower levels of abstraction.

The Resource Orchestration functions of the NFVO interfaces the VIM/WIMs through well-defined interfaces comprised in the Or-Vi reference point for requesting creation and management of virtual networks.

5.6.3 Roles and responsibilities

In the example presented in figure 5.2, the Resource Orchestration function of the NFVO is responsible for the following aspects related to NFVI connectivity services:

- Global view of the network characteristics of the various logical links.
- Orchestrate the VNFs required for a NS in a manner best suited to match the network constraints specified by the Network Service Descriptor (NSD), see clause 6.
- Orchestrate connectivity over a combination of PNFs and VNFs.
- Monitor the abstracted end-to-end Network Service utilization aspects such as bandwidth, latency, etc. between the various NFVI-PoPs.
- Support and invoke the underlying VIM/WIM northbound interfaces to request individual connectivity services to construct the end-to-end service. Example connectivity services include E-LINE, E-LAN and E-TREE services as defined in ETSI GS NFV-INF 005 [i.7]. Connectivity services include internal connectivity between components of a VNF making up each VNF, external connectivity between the VNFs in a VNF Forwarding Graph and connectivity between the VNFs and PNFs.

The VIMs are responsible for the following aspects related to NFVI connectivity services over NFVI-PoPs:

- Provide a "Network as a Service" northbound interface to the higher layers, e.g. NFVO and VNF Manager.

- Abstract the various southbound interfaces and overlay/network partitioning mechanisms exposed by the underlying NFVI network.
- Invoke the underlying NFVI network southbound interfaces, whether they are Network Controllers or Network Functions, to construct the service within the domain.

The WIMs provide connectivity services between the NFVI-PoPs and connectivity to Physical Network Functions. This functional block can be added as a specialized VIM, in particular for new greenfield virtualised deployments. Alternatively, a WIM can also be an existing component of the OSS/BSS functional block. The WIMs are responsible for the following aspects related to the NFVI connectivity services:

- Path computation based on quality assurance factors such as jitter, RTT, delay & bandwidth calendaring.
- Establish connectivity over the physical network (e.g. set of MPLS tunnels).
- Provide a northbound interface to the higher layers, e.g. NFVO, to provide connectivity services between NFVI-PoPs or to physical network functions.
- Invoke the underlying NFVI network southbound interfaces, whether they are Network Controllers or Network Functions, to construct the service within the domain.

The Network Controller function is responsible for the following aspects related to NFVI connectivity services:

- Resource management and tracking of network resources and attributes such as bandwidth, jitter, delay; etc.
- Implement southbound interfaces to the compute and network resources providing the connectivity services to create overlay tunnels (e.g. VXLAN, NVGRE, MPLS over GRE) or network partitions (for example using OpenFlow).
- Abstract the information exposed by the underlying NFVI network via various southbound interfaces.
- Invoke the underlying NFVI network southbound interfaces, whether they are Network Controllers or Network Functions, to construct the service within the domain.

5.6.4 NFVI network EMS/NMS vs. Network Controllers

While figure 5.2 illustrates an overall NFV goal, it is not expected that the first NFV deployments will use Network Controllers with programmatic/open interfaces for all network segments. It is quite possible that the NFVI network will have Network Controllers for some domains but not for others. Figure 5.3 shows a similar example to the one described in clause 5.6.2 but illustrates an example starting point where parts of the NFVI require an EMS/NMS to configure the resources. This could be the case when the resources are not reconfigured regularly, i.e. for static provisioning, or when a Network Controller with programmatic interfaces is not available for these resources. In that case, the configuration is either done manually or through a proprietary interface.

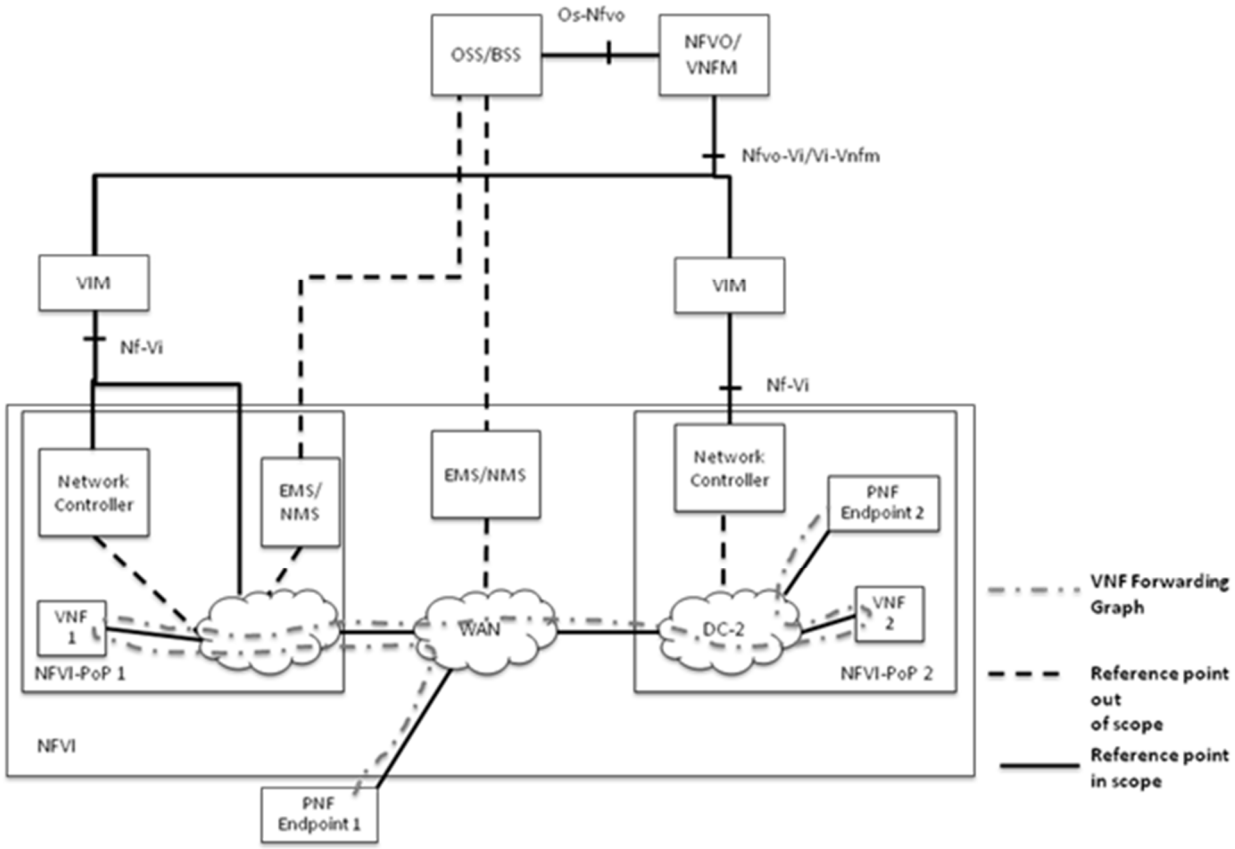


Figure 5.3: EMS/NMS for the NFVI network

5.7 NFV-MANO reference points

5.7.1 Os-Ma-nfvo

Clause 5 describes reference points between NFV-MANO functional blocks and between those and other functional blocks.

This reference point is used for exchanges between OSS/BSS and NFV Orchestrator, and supports the following:

- Network Service Descriptor and VNF package management.
- Network Service instance lifecycle management:
  - Network Service instantiation.
  - Network Service instance update (e.g. update a VNF instance that is comprised in the Network Service instance).
  - Network Service instance query (e.g. retrieving summarized information about NFVI resources associated to the Network Service instance, or to a VNF instance within the Network Service instance).
  - Network Service instance scaling.
  - Network Service instance termination.
- VNF lifecycle management:
  - For VNF lifecycle management, the NFV Orchestrator identifies the VNF Manager and forwards such requests (see Or-Vnfm description).

- Policy management and/or enforcement for Network Service instances, VNF instances and NFVI resources (for authorization/access control, resource reservation/placement/allocation, etc.).
- Querying relevant Network Service instance and VNF instance information from the OSS/BSS.
- Forwarding of events, accounting and usage records and performance measurement results regarding Network Service instances, VNF instances, and NFVI resources to OSS/BSS, as well as and information about the associations between those instances and NFVI resources, e.g. number of VMs used by a certain VNF instance.

5.7.2 Ve-Vnfm-em

This reference point is used for exchanges between EM and VNF Manager, and supports the following:

- VNF instantiation.
- VNF instance query (e.g. retrieve any run-time information).
- VNF instance update (e.g. update configuration).
- VNF instance scaling out/in, and up/down.
- VNF instance termination.
- Forwarding of configuration and events from the EM to the VNFM.
- Forwarding of configuration and events regarding the VNF from the VNFM to the EM.

NOTE: This reference point is only used if the EM is aware of virtualisation.

5.7.3 Ve-Vnfm-vnf

This reference point is used for exchanges between VNF and VNF Manager, and supports the following:

- VNF instantiation.
- VNF instance query (e.g. retrieve any run-time information).
- VNF instance update (e.g. update configuration).
- VNF instance scaling out/in, and up/down.
- VNF instance termination.
- Forwarding of configuration and events from the VNF to the VNFM.
- Forwarding of configuration, events, etc. regarding VNF, from the VNFM to the VNF.
- Verification that the VNF is still alive/functional.

5.7.4 Nf-Vi

This reference point is used for exchanges between Virtualisation Infrastructure Manager and NFV Infrastructure, and supports the following:

- Allocate VM with indication of compute/storage resource.
- Update VM resources allocation.
- Migrate VM.
- Terminate VM.
- Create connection between VMs.

- Configure connection between VMs.
- Remove connection between VMs.
- Forwarding of configuration information, failure events, measurement results, and usage records regarding NFVI (physical, software, and virtualised resources) to the VIM.

5.7.5 Or-Vnfm

This reference point is used for exchanges between NFV Orchestrator and VNF Manager, and supports the following:

- NFVI resources authorization/validation/reservation/release for a VNF.
- NFVI resources allocation/release request for a VNF.
- VNF instantiation.
- VNF instance query (e.g. retrieve any run-time information).
- VNF instance update (e.g. update configuration).
- VNF instance scaling out/in, and up/down.
- VNF instance termination.
- VNF package query.
- Forwarding of events, other state information about the VNF that could impact also the Network Service instance.

5.7.6 Or-Vi

This reference point is used for exchanges between the NFV Orchestrator and the VIM, and supports the following:

- NFVI resource reservation/release.
- NFVI resource allocation/release/update.
- VNF software image addition/deletion/update.
- Forwarding of configuration information, events, measurement results, usage records regarding NFVI resources to the NFV Orchestrator.

5.7.7 Vi-Vnfm

This reference point is used for exchanges between the VNF Manager and the Virtualised Infrastructure Manager, and supports the following:

- NFVI resources reservation information retrieval.
- NFVI resources allocation/release.
- Exchanges of configuration information between reference point peers, and forwarding to the VNF Manager such information for which the VNFM has subscribed to (e.g. events, measurement results, and usage records regarding NFVI resources used by a VNF).

5.8 Interfaces description approach

5.8.1 Overview

This clause describes the interface design approach in NFV-MANO architectural framework. The detailed description of the interfaces is the subject of clause 7.

While reference points are a good way to identify peer-to-peer relationships between functional blocks, descriptions of the interfaces provide a deeper understanding of how capabilities provided by "producer" functional blocks are exposed to "consumer" functional blocks. This approach allows to:

- Facilitate analysis of requirements against existing standardized interfaces, leading to identification of gaps and subsequent recommendations on how to address them.
- Lead to exposure of the appropriate level of abstraction via the interfaces, in such a way that it:
  - Provides guidance to bring consistency among existing and future interfaces to be standardized.
  - Provides the functional definitions, without limiting implementation choices of the functional blocks.
  - Emphasizes contract-based and producer-consumer interface paradigm to support the dynamic nature of the interfaces among functional blocks, without limiting the design and implementation options.
  - Illustrates that the same function can be provided by a functional block to several other functional blocks without restricting the communication paths between functional blocks, within the boundaries of the NFV architectural framework.
  - Illustrates that different functional blocks can expose the same functionality in an abstracted manner to support different implementation options.
- Help define functional blocks' capabilities in such a way that orchestration of different entities is provided in multiple functional blocks, if appropriate; it is recommended that those functional blocks offer abstracted services to other functional blocks.
- Emphasize the value of designing granular functions that can be performed via atomic transactions using the exposed interfaces.
- Recommend the use of configuration and authorization policies to play a role in implementing the path that an operational flow can take, when multiple alternatives are possible.
- Enable configuration and re-configuration of the levels of granularity, frequency and types of information being collected for fault, performance management and event data at runtime.

5.8.2 Producer-consumer paradigm

A functional block can play the role of a producer, or the role of a consumer or both. Interfaces represent a published contract offered by a producer functional block, exposing externally the functions produced. They can be consumed by any other consumer functional block that is authenticated and authorized to access and consume the functions offered; authentication mechanisms and authorization policies are defined to control access to and consumption of functions produced by a functional block, to the level of granularity desired by the owner of the producer functional block. A producer functional block can offer different functions, and can offer different levels of granularity within each function; authorization policies are expected to reflect such granularity because different access and consumption rights can be granted to different consumer functional blocks.

A consumer functional block can also have a choice of sending requests to several producer functional blocks offering the same function; in some cases the function offered is completely identical, in other cases one producer functional block offers the function with optional extensions. Depending on implementation choices, configuration policies can be used to determine which producer functional block will be invoked by a consumer functional block to provide the desired function.

For notification interfaces, the general paradigm used is a publish/subscribe where the publishers issue notifications through the interface without knowledge of what if any subscribers there are. Subscribers using the notification interface express interest in some notifications or groups of notifications and only receive those of interest. For a notification interface, the publisher plays the producer role and the subscriber the consumer role.

Between the choices dictated by configuration policies available at the consuming functional block, the implementation choices of manufacturers offering the functions and the choices dictated by authorization policies at the producing functional block, the ability to support some preferred operational flows, and disallow other operational flows in the service provider's environment becomes straightforward.

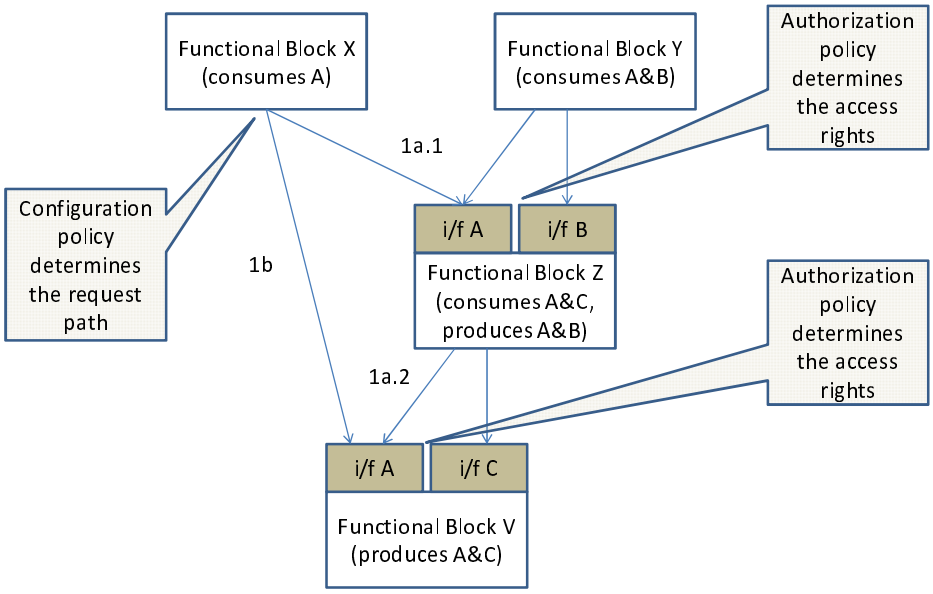


Figure 5.4: Interface concept in a Producer-Consumer paradigm

Figure 5.4 illustrates the notion of producer and consumer functional blocks, and the interfaces that the producer functional blocks expose, through which the consumer functional blocks consumes the functions provided. The following are noticeable:

- The same interface can be exposed by different producer functional blocks (e.g. interface A). It is also possible (not shown) that functional block Z can expose interface A with additional extensions or additional behaviours (e.g. which can be provided by producer functional block Z, but perhaps not by producer functional block V).
- A producer functional block can serve multiple consumer functional blocks via the same interface (e.g. producer functional blocks Z and V can serve multiple consumers via interface A).
- A functional block can be at the same time a producer and a consumer of the same interface (e.g. functional block Z both produces and consumes interface A, or of different interfaces (e.g. functional block Z produces interface A and consumes interface C).
- The design supports multiple flow alternatives that can be directed by the use of policies. In the figure, consumer functional block X can be configured to request a certain function from either producer functional block Z (via flow 1a) or from producer functional block V (via flow 1b). Access to either of the functions is controlled by authorization policies associated with interface A on each of producer functional blocks Z and V. In operational flow 1.a, producer functional block Z processes the request and adds value (it could offer functional extensions or additional behaviour (e.g. implementing some global policies) via interface A, or it could provide additional services based on other information it has access to, e.g. policies, which could not be available to producer functional block V) before sub-contracting the function to producer functional block V for completion.
- Each producer functional block in NFV-MANO can maintain a set of policy enforcement rules. These rules can enumerate the list of other entities that are authorized to consume and perform specific operations over an interface; additionally, these policies could be defined on a per VNF basis. The actual development and implementation of policy rules are highly implementation specific and out of scope for the present document.

6 NFV management and orchestration information elements

6.1 Introduction

Clause 6 describes relevant information elements for the on-boarding and lifecycle management of VNF and NS.

The Network Service describes the relationship between VNFs and possibly PNFs that it contains and the links needed to connect VNFs that are implemented in the NFVI network. Links are also used to interconnect the VNFs to PNFs and endpoints. Endpoints provide an interface to the existing network, including the possibility of incorporating Physical Network Functions to facilitate evolution of the network.

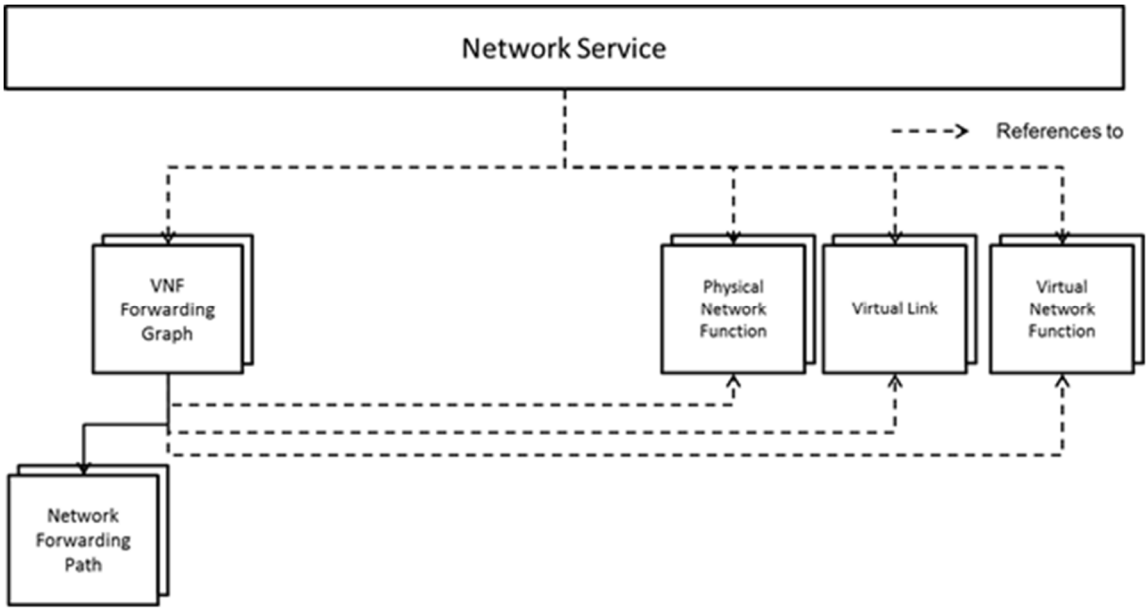


Figure 6.1: Top-level information elements

This clause describes several categories of information:

- Information that resides in descriptors. These are deployment templates that contain relatively static information used in the process of on-boarding VNFs and NSs.
- Information that resides in records. These contain relatively dynamic run-time data representing e.g. VNF or NS instances; this data is maintained throughout the lifetime of the instance.
- Information that can be passed between functional blocks at these interfaces.

The information is structured into information elements. An information element contains specific information. It can contain a single value only, but can also contain further information elements. Hence, information elements can build a tree structure specifying a containment tree of information elements. The information element can be classified into three different types: leaf, reference and element which are basically an element itself. Each of these information elements has a unique name along the whole path in the tree that lead to that element from the root of the tree.

A leaf is a single information element that specifies a value within the scope of the present document. The data type of the value is dependent on the information it carries. The number of occurrences of the same leaf element inside its parent element is specified by the cardinality of the element.

A reference element is an information element that carries a reference to another information element. This can be represented by an URI but depends on the concrete implementation of the information model. The number of occurrences of the same reference element inside its parent element is specified by the cardinality of the element.

A sub element is an information element itself that specifies another level in the tree. The number of occurrences of the same sub element inside its parent element is specified by the cardinality of the element.

As explained above each of these elements has a name that is unique among the branches of a tree. The name of the element is represented as part of the clause headers of the clauses below. The clauses itself contain tables with four columns. The first column contains the name of the information element, the second one gives its type. The third column gives the cardinality of the information element. If the cardinality is a positive integer x then the element occurs exactly x times. If the cardinality is given as a range then this number of occurrences is within that range. A range starting with "0" implies that the element can be omitted. The last column describes the content of the information element. If the information element is a sub element the elements contained can be found by a subsequent section whose heading is the name of this information element. For some elements and additional example column is provided explaining the meaning of the element by an example.

To describe a Network Service and the components comprising the Network Service, information elements representing these components are introduced. There are four information elements defined apart from the top level Network Service (NS) information element:

- Virtualised Network Function (VNF) information element.
- Physical Network Function (PNF) information element.
- Virtual Link (VL) information element.
- VNF Forwarding Graph (VNFFG) information element.

A set of VLs in a Network Service form a Network Connection Topology (NCT) as described in ETSI GS NFV-SWA 001 [i.8]. A VNFFG can reference other information elements in the Network Service such as PNFs, VLs and VNFs. A VNFFG also contains a Network Forwarding Path (NFP) element. Figure 6.1 depicts the high level structure of the information elements. NS, VNF and PNF information elements include Connection Point descriptions.

The information elements can be used in two different contexts: as descriptors in a catalogue or template context or as instance records in a runtime context. Figure 6.2 depicts the usage of the information elements in different contexts.

A Network Service Descriptor (NSD) is a deployment template for a Network Service referencing all other descriptors which describe components that are part of that Network Service.

A VNF Descriptor (VNFD) is a deployment template which describes a VNF in terms of its deployment and operational behaviour requirements. It is primarily used by the VNFM in the process of VNF instantiation and lifecycle management of a VNF instance. The information provided in the VNFD is also used by the NFVO to manage and orchestrate Network Services and virtualised resources on the NFVI. The VNFD also contains connectivity, interface and KPIs requirements that can be used by NFV-MANO functional blocks to establish appropriate Virtual Links within the NFVI between its VNFC instances, or between a VNF instance and the endpoint interface to the other Network Functions.

A VNF Forwarding Graph Descriptor (VNFFGD) is a deployment template which describes a topology of the Network Service or a portion of the Network Service, by referencing VNFs and PNFs and Virtual Links that connect them.

A Virtual Link Descriptor (VLD) is a deployment template which describes the resource requirements that are needed for a link between VNFs, PNFs and endpoints of the Network Service, which could be met by various link options that are available in the NFVI. The NFVO can select an option following consultation of the VNFFG to determine the appropriate NFVI to be used based on functional (e.g. dual separate paths for resilience) and other needs (e.g. geography and regulatory requirements).

A Physical Network Function Descriptor (PNFD) describes the connectivity, Interface and KPIs requirements of Virtual Links to an attached Physical Network Function. This is needed if a physical device is incorporated in a Network Service to facilitate network evolution.

The NFVO on-boards all descriptors. NSD, VNFFGD, and VLD are "on-boarded" into a NS Catalogue; VNFD is on-boarded in a VNF Catalogue, as part of a VNF Package. The present document does not specify how or where PNFD is on-boarded; it is assumed that this is out of scope since that information is not specific to NFV, and can be readily made available in current deployments. The integrity and authenticity of NSD, VNFFGD, VLD, VNFD and PNFD are validated during on-boarding procedure. Multiple versions of a descriptor can exist in a catalogue, and the NFV Orchestrator could be expected to update the deployed network configuration to match an updated version of the descriptor. The triggering of updates would be subject to business policies outside the scope of the present document. On instantiation, the NFV Orchestrator or VNFM receive instantiation parameters from the entity initiating the instantiation operation. Instantiation input parameters are used to customize a specific instantiation of a NS or VNF.

Instantiation input parameters contain information that identifies a deployment flavour to be used and could refer to existing instances of VNF/PNF, which is expected to be incorporated in the instantiation process.

As a result of the instantiation operation records are created to represent the newly created instances. The Network Service Record (NSR), the VNF Record (VNFR), the Virtual Link Record (VLR) and the VNFFG Record (VNFFGR) are created based on the information given in the descriptors together with additional runtime information related to the component instances. A PNF Record (PNFR) represents an instance related to a pre-existing PNF which is part of a Network Service and contains a set of runtime attributes regarding PNF information, including connectivity relevant to the NFVO. Figure 6.2 gives an overview how descriptors and records relate to each other.

The NSR, VNFR, VNFFGR and VLR information elements provide a collection of data items that could be expected to model the state of instances of NS, VNF, VNFFG or VL, respectively; they are a result of combining NSD, VNFD, VNFFGD and VLD information elements with input/output parameters exchanged via different interfaces produced and/or consumed by NFV-MANO functional blocks, and do not imply any particular implementation.

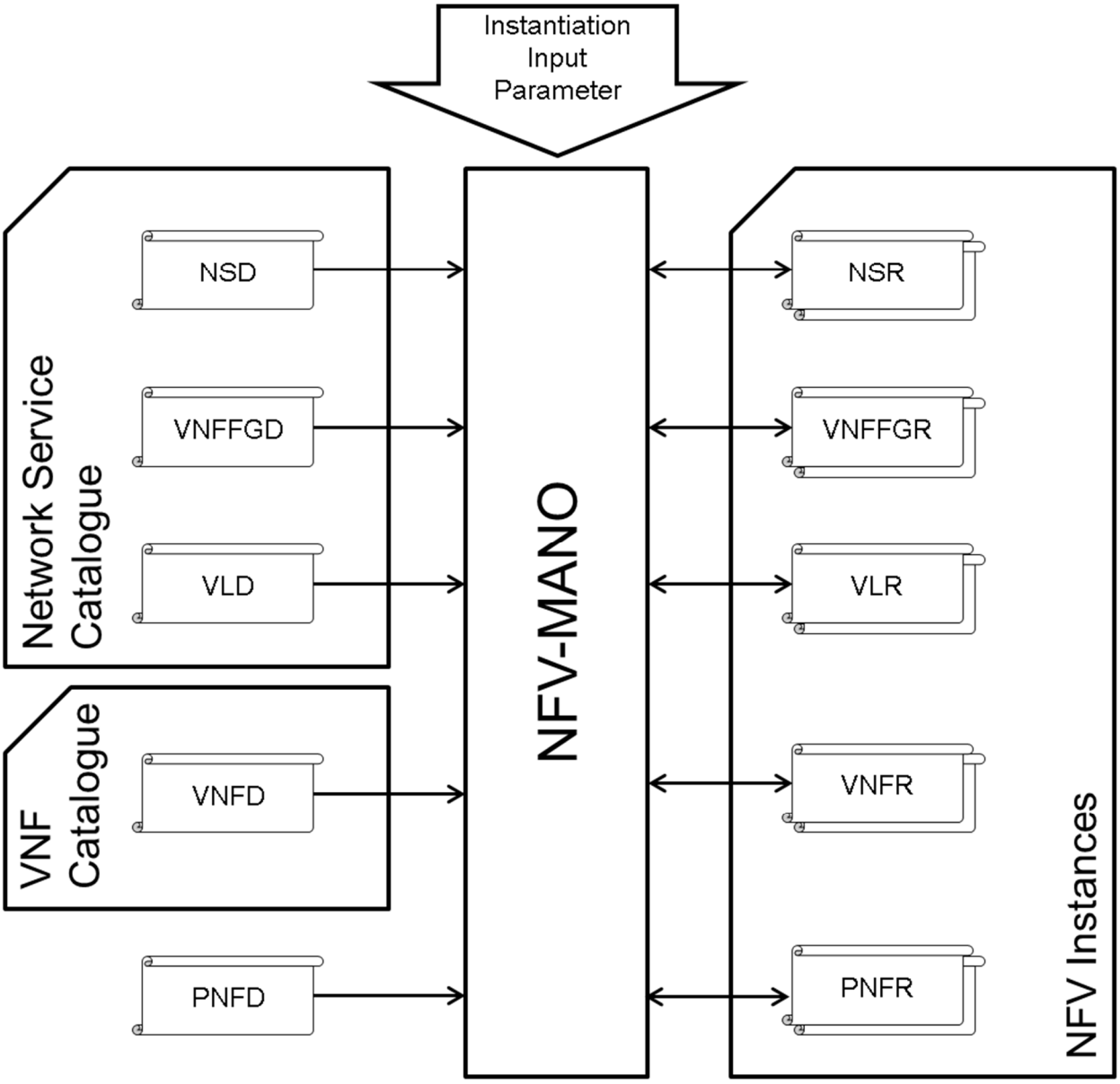


Figure 6.2: Information elements in different context

6.2 Network Service information elements

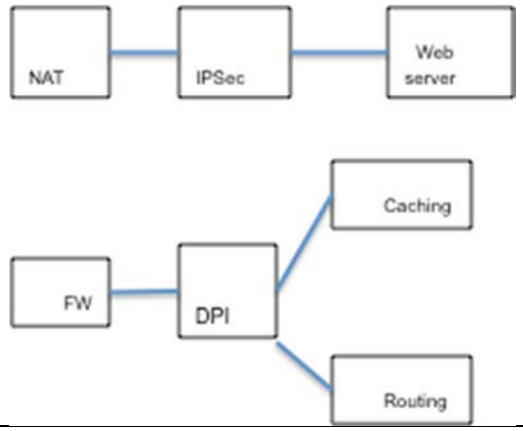
6.2.0 Preamble

The Network Service Descriptor (NSD) consists of static information elements as defined below. It is used by the NFV Orchestrator to instantiate a Network Service, which would be formed by one or more VNF Forwarding Graphs, VNFs, PNFs and VLs. The NSD also describes deployment flavours of Network Service.

The following is not a complete list of information elements constituting the NSD but a minimum subset needed to on-board the Network Service (NS).

6.2.1 Network Service Descriptor (nsd)

6.2.1.1 nsd base element

Identifier	Type	Cardinality	Description
Id	Leaf	1	ID of this Network Service Descriptor.
vendor	Leaf	1	Provider or vendor of the Network Service.
version	Leaf	1	Version of the Network Service Descriptor.
vnfd	Reference	1...N	VNF which is part of the Network Service, see clause 6.3.1. This element is required, for example, when the Network Service is being built top-down or instantiating the member VNFs as well.
vnffgd	Reference	0...N	VNFFG which is part of the Network Service, see clause 6.5.1. A Network Service might have multiple graphs, for example, for: 1. Control plane traffic. 2. Management-plane traffic. 3. User plane traffic itself could have multiple NFPs based on the QoS, etc. The traffic is steered amongst 1 of these NFPs based on the policy decisions.
vld	Reference	0…N	Virtual Link which is part of the Network Service, see clause 6.4.1.
lifecycle_event	Leaf	0…N	Defines NS functional scripts/workflows for specific lifecycle events (e.g. initialization, termination, scaling).
vnf_dependency	Leaf	0…N	Describe dependencies between VNF. Defined in terms of source and target VNF i.e. target VNF "depends on" source VNF. In other words a source VNF exists and connects to the service before target VNF can be initiated/deployed and connected. This element would be used, for example, to define the sequence in which various numbered network nodes and links within a VNFFG will be instantiated by the NFV Orchestrator. 

Identifier	Type	Cardinality	Description
monitoring_ parameter	Leaf	0...N	Represents a monitoring parameter which can be tracked for this NS. These can be network service metrics that are tracked for the purpose of meeting the network service availability contributing to SLAs (e.g. NS downtime). These can also be used for specifying different deployment flavours for the Network Service in Network Service Descriptor, and/or to indicate different levels of network service availability. Examples include specific parameters such as calls-per-second (cps), number-of-subscribers, no-of-rules, flows-per-second, etc. 1 or more of these parameters could be influential in determining the need to scale-out.
service_ deployment_ flavour	Element	1...N	Represents the service KPI parameters and its requirement for each deployment flavour of the NS being described, see clause 6.2.1.3. For example, there could be a flavour describing the requirements to support a vEPC with 300k calls per second. There could be another flavour describing the requirements to support a vEPC with 500k calls per second.
auto_scale_policy	Leaf	0...N	Represents the policy meta data, which can include the criteria parameter & action-type. The criteria parameter is a supported assurance parameter. An example of such a descriptor could be: • Criteria parameter: calls-per-second. • Action-type: scale-out to a different flavour ID.
connection_point	Element	1…N	This element describes a Connection Point which acts as an endpoint of the Network Service, see clause 6.2.1.2. This can, for example, be referenced by other elements as an endpoint.
pnfd	Reference	0...N	PNFs which are part of the Network Service, see clause 6.6.1.
nsd_security	Leaf	0...1	This is a signature of nsd to prevent tampering. The insertion of the particular hash algorithm used to compute the signature, together with the corresponding cryptographic certificate to validate the signature is recommended.

6.2.1.2 Connection Point (nsd:connection_point)

Identifier	Type	Cardinality	Description
id	Leaf	1	ID of the Connection Point.
type	Leaf	1	This can be for example a virtual port, a virtual NIC address, a physical port, a physical NIC address or the endpoint of an IP VPN enabling network connectivity.

6.2.1.3 Service deployment flavour (nsd:service_deployment_flavour)

6.2.1.3.1 Base element

Identifier	Type	Cardinality	Description
id	Leaf	1	ID of the deployment flavour.
flavour_key	Leaf	1	Assurance parameter against which this flavour is being described. The key could be a combination of multiple assurance parameters with a logical relationship between them. The parameters present are monitoring_parameter supported in clause 6.2.1.1. For example, a flavour of a virtual EPC could be described in terms of the assurance parameter "calls-per-second" (cps).
constituent_vnf	Element	1…N	Represents the characteristics of a constituent flavour element, see clause 6.2.1.3.2.

6.2.1.3.2 Constituent VNF (nsd:service_deployment_flavour:constituent_vnf)

Identifier	Type	Cardinality	Description
vnf_reference	Reference	1	Reference to a VNFD declared as vnfd in the Network Service via vnf:id.
vnf_flavour_id_ reference	Reference	1	References a VNF flavour (vnfd:deployment_flavour:id) to be used for this service flavour, see clause 6.2.1.3.1.
redundancy_model	Leaf	0...1	Represents the redundancy of instances, for example, "active" or "standby".
affinity	Leaf	0...1	Specifies the placement policy between this instance and other instances, if any.
capability	Leaf	0...1	Represents the capabilities of the VNF instances. An example of capability is instance capacity (e.g. capability = 50 % × NS capacity).
number_of_instances	Leaf	1	Number of VNF instances satisfying this service assurance. For a Gold flavour of the vEPC Network Service that satisfies an assurance of 96k cps, 2 instances of the vMME VNFs will be required.

6.2.2 Network Service Record (nsr)

6.2.2.1 nsr base element

Identifier	Type	Cardinality	Description
id	Leaf	1	ID of the Network Service instance.
auto_scale_policy	Leaf	0...N	See clause 6.2.1.1.
connection_point	Element	1...N	See clause 6.2.1.2.
monitoring_parameter	Leaf	1...N	Monitoring parameter used in this instance.
service_deployment_flavour	Reference	1	References the nsd:service_deployment_flavour used to instantiate this Network Service instance.
vendor	Leaf	1	See clause 6.2.1.1.
version	Leaf	1	See clause 6.2.1.1.
vlr	Reference	0...N	See clause 6.4.2.
vnfr	Reference	1...N	See clause 6.3.2.
lifecycle_event	Leaf	0...N	See clause 6.2.1.1.
vnf_dependency	Leaf	0...N	See clause 6.2.1.1.
vnffgr	Reference	0...N	See clause 6.5.2.
pnfr	Reference	1...N	See clause 6.6.2.
descriptor_reference	Reference	1	The reference to the Network Service Descriptor used to instantiate this Network Service.
resource_reservation	Leaf	0...N	Resource reservation information identification (potentially per individual VIM) for NFVI resources reserved for this NS instance.
runtime_policy_info	Leaf	0...N	Generic placeholder for input information related to NS orchestration and management policies to be applied during runtime of a specific NS instance (e.g. for NS prioritization, etc.).
status	Leaf	1	Flag to report status of the Network Service.
notification	Leaf	1...N	System that has registered to received notifications of status changes.
lifecycle_event_history	Leaf	0...N	Record of significant Network Service lifecycle events.
audit_log	Leaf	0...N	Record of detailed operational events.

6.2.2.2 Connection Point (nsr:connection_point)

Identifier	Type	Cardinality	Description
id	Leaf	1	ID of the Connection Point instance.
type	Leaf	1	See clause 6.2.1.2.

6.3 Virtualised Network Function information elements

6.3.0 Preamble

The information elements listed in clause 6.3 is not a complete list of information elements constituting the VNF information elements, but rather a minimum representative subset considered necessary to on-board the VNF.

6.3.1 VNF Descriptor (vnfd)

6.3.1.1 vnfd base information elements

A VNF Descriptor (VNFD) is a deployment template which describes a VNF in terms of deployment and operational behaviour requirements. The VNFD also contains connectivity, interface and KPIs requirements that can be used by NFV-MANO functional blocks to establish appropriate Virtual Links within the NFVI between VNFC instances, or between a VNF instance and the endpoint interface to other Network Functions.

Connection Points of a VNF can be mapped to interfaces as described in ETSI GS NFV-SWA 001 [i.8].

Identifier	Type	Cardinality	Description
Id	Leaf	1	ID (e.g. name) of this VNFD.
vendor	Leaf	1	The vendor generating this VNFD.
descriptor_version	Leaf	1	Version of the VNF Descriptor.
version	Leaf	1	Version of VNF software, described by the descriptor under consideration.
vdu	Element	1…N	This describes a set of elements related to a particular VDU, see clause 6.3.1.2.
virtual_link	Element	0...N	Represents the type of network connectivity mandated by the VNF vendor between two or more Connection Points, see clause 6.3.1.3.
connection_point	Element	1...N	This element describes an external interface exposed by this VNF enabling connection with a VL, see clause 6.3.1.4 (see note).
lifecycle_event	Leaf	0...N	Defines VNF functional scripts/workflows for specific lifecycle events (e.g. initialization, termination, graceful shutdown, scaling out/in, update/upgrade, VNF state management related actions to support service continuity).
dependency	Leaf	0…N	Describe dependencies between VDUs. Defined in terms of source and target VDU, i.e. target VDU "depends on" source VDU. In other words sources VDU exists before target VDU can be initiated/deployed.
monitoring_parameter	Leaf	0...N	Monitoring parameters, which can be tracked for this VNF. Can be used for specifying different deployment flavours for the VNF in a VNFD, and/or to indicate different levels of VNF service availability. These parameters can be an aggregation of the parameters at VDU level e.g. memory-consumption, CPU-utilization, bandwidth-consumption, etc. They can be VNF specific as well such as calls-per-second (cps), number-of-subscribers, no-of-rules, flows-per-second, VNF downtime, etc. One or more of these parameters could be influential in determining the need to scale.
deployment_flavour	Element	1...N	Represents the assurance parameter(s) and its requirement for each deployment flavour of the VNF being described, see clause 6.3.1.5.

Identifier	Type	Cardinality	Description
auto_scale_policy	Leaf	0...N	Represents the policy meta data, which can include the criteria parameter and action-type. The criteria parameter is a supported assurance parameter (vnf:monitoring_parameter). Example of such a descriptor could be: • Criteria parameter → calls-per-second. • Action-type → scale-out to a different flavour ID, if exists.
manifest_file	Leaf	0…1	The VNF package can contain a file that lists all files in the package. This can be useful for auditing purposes or for enabling some security features on the package.
manifest_file_security	Leaf	0…N	The manifest file can be created to contain a digest of each file that it lists as part of the package. This digest information can form the basis of a security mechanism to ensure the contents of the package meet certain security related properties. If the manifest file contains digests of the files in the package, then it is recommended that the manifest file also notes the particular hash algorithm used to enable suitable verification mechanisms. Examples of suitable hash algorithms include, but are not limited to SHA-256, SHA-384, SHA-512, and SHA-3. In conjunction with an appropriate security signing mechanism, which can include having a security certificate as part of the VNF package, the digest information can be used to help ensure the contents of the VNF package have not been tampered with.
NOTE: The connection between the VNF and the VL is expressed by the VLD referencing this Connection Point. The Connection Point can also be attached to internal Virtual Links (vnfd:virtual_link:id).			

6.3.1.2 Virtual Deployment Unit (vnfd:vdu)

6.3.1.2.1 vnfd:vdu base elements

Information elements concerning the VDU are defined in the table below. For structuring the table, it is split into several parts with each clause below covering one of it.

Identifier	Type	Cardinality	Description
id	Leaf	1	A unique identifier of this VDU within the scope of the VNFD, including version functional description and other identification information. This will be used to refer to VDU when defining relationships between them.
vm_image	Leaf	0...1	This provides a reference to a VM image (see note).
computation_requirement	Leaf	1	Describe the required computation resources characteristics (e.g. processing power, number of virtual CPUs, etc.), including Key Quality Indicators (KQIs) for performance and reliability/availability.
virtual_memory_resource_element	Leaf	1	This represents the virtual memory needed for the VDU.
virtual_network_bandwidth_resource	Leaf	1	This represents the requirements in terms of the virtual network bandwidth needed for the VDU.
lifecycle_event	Leaf	0...N	Defines VNF component functional scripts/workflows for specific lifecycle events (e.g. initialization, termination, graceful shutdown, scaling out/in).
constraint	Leaf	0...1	Placeholder for other constraints.

Identifier	Type	Cardinality	Description
high_availability	Leaf	0...1	Defines redundancy model to ensure high availability examples include: • ActiveActive: Implies that two instance of the same VDU will co-exists with continuous data synchronization. • ActivePassive: Implies that two instance of the same VDU will co-exists without any data synchronization.
scale_in_out	Leaf	0...1	Defines minimum and maximum number of instances which can be created to support scale out/in.
vnfc	Element	1...N	Contains information that is distinct for each VNFC created based on this VDU.
monitoring_parameter	Leaf	0...N	Monitoring parameter, which can be tracked for a VNFC based on this VDU. Examples include: memory-consumption, CPU-utilization, bandwidth-consumption, VNFC downtime, etc.
NOTE: A cardinality of zero allows for creating empty virtualisation containers as per ETSI GS NFV-SWA 001 [i.8].			

6.3.1.2.1.1 VNFC (vnfd:vdu:vnfc)

Identifier	Type	Cardinality	Description
id	Leaf	1	Unique VNFC identification within the namespace of a specific VNF.
connection_point	Element	1...N	Describes network connectivity between a VNFC instance (based on this VDU) and an internal Virtual Link.

6.3.1.2.1.2 Connection Point (vnfd:vdu:vnfc:connection_point)

Identifier	Type	Cardinality	Description
id	Leaf	1	ID of the Connection Point.
virtual_link_reference	Reference	1	References an internal Virtual Link (vnfd:virtual_link:id, see clause 6.3.1.3) to which other VNFCs can connect.
type	Leaf	1	This can be, for example, a virtual port, a virtual NIC address, a physical port, a physical NIC address or the endpoint of an IP VPN enabling network connectivity.

6.3.1.2.2 vnfd:vdu information elements related to CPUs

An implementation can have particular processor requirements. This could be for many reasons such as code related dependencies such as the use of processor instructions, via tool suite generated dependencies such as compiler based optimizations targeting a specific processor, or validation related dependencies as it was tested on a particular processor. By specifying the processor characteristics in the VDU descriptor, VNFs could be able to be deployed on suitably equipped platforms thereby facilitating a more efficient, higher performing and/or more reliable deployment. However, note that careful consideration will be given to the use of these elements as they could limit portability of a VNF. It is expected that a VNF is able to be deployed and instantiated by NFV-MANO even if these characteristics are not specified in the VDU descriptor.

Identifier	Type	Cardinality	Description
cpu_instruction_set_extension	Leaf	0...N	Instruction sets are often enhanced with instruction set extensions. This element represents instruction set extensions that the VDU has been developed, optimized or tested with.
cpu_model	Leaf	0...N	The CPU model for which the VDU has been developed, compiled with CPU model specific optimizations, optimized or validated on.
cpu_model_specification_binding	Leaf	0...1	VDUs can be developed, compiled, optimized or validated on particular CPU models. Some deployments could wish to permit the VDU to be deployed on a platform with the specified CPU only, or with an alternative CPU with the same architecture, instruction set, and if specified, instruction set extensions, or with a CPU of equivalent or greater capability.
cpu_min_clock_speed	Leaf	0...1	The minimum CPU clock speed can be one of the elements that the development and validation of the VDU has been considered with. This can be in conjunction with some of the other CPU elements such as CPU Model. Requiring a minimum clock speed can be part of a deployment requirement necessary to help ensure particular performance or timing related characteristics are met in the deployment.
cpu_core_reservation	Leaf	0...1	The number of CPU cores allocated to the VDU. This could be necessary to help ensure particular performance or timing related characteristics are met in the deployment.
cpu_simultaneous_multi_threading_ hw_thread_specification	Leaf	0...1	The use of Simultaneous Multi-Threading HW is an efficient way to increase the compute capacity of a platform. SMT HW threads share some CPU core resources. In some VDU implementations, it could be necessary to very explicitly control the HW thread allocation on a platform. This could be to help ensure locality in data caches or as a mechanism to enhance determinism.
cpu_core_oversubscription_policy	Leaf	0...1	The VDU can co-exist on a platform with multiple VDUs or VMs and as such will be sharing CPU core resources available in the platform. It could be necessary to specify the CPU core oversubscription policy in terms of virtual cores to physical cores/threads on the platform. This policy could be based on required VDU deployment characteristics such as high performance, low latency, and/or deterministic behaviour.
cpu_core_and_hw_thread_allocation_ topology_policy	Leaf	0...1	The VDU can be designed to use a specific mapping of virtual CPUs to HW threads or cores with a specific allocation topology in order to ensure locality in data caches and maximize performance. The VDU will not specify which physical resources to use, but specify if virtual CPUs are coupled together as HW threads belonging to the same core, or as belonging to the same processor.
cpu_last_level_cache_size	Leaf	0...1	The size of the last level cache could impact the performance of the VDU, particularly for cache intensive workloads.
cpu_direct_io_access_to_cache	Leaf	0...1	The ability of an I/O device to have direct access to the CPU cache enables considerable memory access savings and for I/O intensive workloads can offer significant performance benefits.

Identifier	Type	Cardinality	Description
cpu_translation_look_aside_buffer_parameter	Leaf	0...N	The Translation Look-aside Buffer (TLB) is a cache for address translation typically used by a hardware based memory management units. The Input/Output TLB (IOTLB) is a cache for address translation related to remapping hardware. The availability of a TLB and an IOTLB can significantly improve the performance of a virtual machine. A number of parameters of the TLBs impact the performance potential. These include: • TLB Size. • TLB Large Page Support. • IOTLB Size. • IOTLB Large Page Support.
cpu_hot_add	Leaf	0...1	Hot add CPU is the ability to dynamically add CPUs to a running system. The new CPU can immediately replace a failing CPU via migration or be brought on-line later.
cpu_support_accelerator	Leaf	0...N	This capability refers to support by the CPU and associated chipsets of a data processing accelerator framework, together with its libraries and drivers.

6.3.1.2.3 vnfd:vdu information elements related to memory

An implementation could have particular memory requirements. This could be for many reasons such as intensive memory needs and/or a design requirement and/or a validation requirement as the implementation could have been validated on platforms with particular memory characteristics. By specifying the memory characteristics in the VDU descriptor, the VNF could be able to be deployed on suitably equipped platforms thereby facilitating a more efficient, higher performing and/or more reliable deployment. However, note that careful consideration will be given to the use of these elements as they could limit portability of a VNF. It is expected that a VNF is able to be deployed and instantiated by NFV-MANO even if these characteristics are not specified in the VDU descriptor.

Identifier	Type	Cardinality	Description
memory_parameter	Leaf	0...N	There are a number of memory related parameters that can have a significant impact on the performance and/or reliability of the VDU. These include: • Memory Type. • Memory Speed. • Number of memory channels. • Size of available memory. • Reliability characteristics such as Memory Error Correction codes. • Memory oversubscription policy. • Memory bandwidth required per VDU. • Number of large pages required per VDU. • Non-Uniform Memory Architecture (NUMA) Allocation Policy, i.e. in NUMA architecture how memory allocation that is cognisant of the relevant process/core allocation is specified. This applies also to allocation of huge pages.
memory_hot_add	Leaf	0...1	Hot add memory is the ability to add physical memory while the system is running. Added memory can immediately replace failing memory via migration or be brought on-line later.

6.3.1.2.4 vnfd:vdu information elements related to security

An implementation could have particular platform security requirements. This could be for many reasons such as a sensitive nature of the implementation or due to requirements such as needed enhanced security. By specifying the platform security characteristics in the VDU descriptor, the VNF could be able to be deployed on suitably equipped platforms thereby facilitating a more efficient, higher performing and/or more secure, reliable deployment. However, note that careful consideration will be given to the use of these elements as they could limit portability of a VNF. It is expected that a VNF is able to be deployed and instantiated by NFV-MANO even if these characteristics are not specified in the VDU descriptor.

Identifier	Type	Cardinality	
platform_security_parameter	Leaf	0...N	The VDU could require some element of platform security in order to be deployed. The security parameters could include the availability of features such as: • The ability to generate true random numbers. The availability of true random numbers can be fundamental to the security of some protocols. • The availability of a Measure Launch Environment (MLE). An MLE can provide the basis for a trusted platform solution and help protect the platform against attacks on vectors such as integrity, confidentiality, reliability, and availability.

6.3.1.2.5 vnfd:vdu information elements related to hypervisors

An implementation could have particular hypervisor related requirements. This could be for many reasons such as being developed on a particular hypervisor, or requiring particular hypervisor capabilities such as support for large TLB pages. By specifying the hypervisor characteristics in the VDU descriptor, the VNF could be able to be deployed on suitably equipped platforms thereby facilitating a more efficient, higher performing and/or more reliable deployment. However, note that careful consideration will be given to the use of these elements as they could limit portability of a VNF. It is expected that a VNF is able to be deployed and instantiated by NFV-MANO even if these characteristics are not specified in the VDU descriptor.

Identifier	Type	Cardinality	Description
hypervisor_parameter	Leaf	1...N	There are a number of hypervisor related parameters that can have a significant impact on the deployment and performance of the VDU. These include: • Hypervisor type (see note 1). • Hypervisor version as a VDU could be validated with a particular version. • Hypervisor Address Translation support parameters including: – Second Level Address Translation (see note 2). – Second Level Address Translation with Large page support. – Second Level Address Translation for I/O. – Second Level Address Translation for I/O with Large page. support. Where "Large" is considered to be 1 GB or greater. – Support for interrupt remapping, i.e. supporting the IOMMU in the hypervisor. – Support of data processing acceleration libraries in the hypervisor, i.e. for acceleration libraries which require hypervisor support for high performance.
NOTE 1: "BARE" could be included in this list if the network function component was to be deployed on a non-virtualised system, but still managed via the Virtualised Infrastructure Manager. NOTE 2: This is expected to be included as the platform could support this feature.			

6.3.1.2.6 vnfd:vdu information elements related to PCIe

An implementation could have particular PCIe related requirements. This could be for many reasons such as requiring a particular PCIe device, or a particular performance capability related to the PCIe bus. By specifying the PCIe characteristics in the VDU descriptor, the VNF could be able to be deployed on suitably equipped platforms thereby facilitating a more efficient, higher performing and/or more reliable deployment. However, note that careful consideration will be given to the use of these elements as they could limit portability of a VNF. It is expected that a VNF is able to be deployed and instantiated by NFV-MANO even if these characteristics are not specified in the VDU descriptor.

Identifier	Type	Cardinality	Description
platform_pcie_parameter	Leaf	0...N	There are a number of PCIe related parameters that can have a significant impact on the deployment and performance of the VDU. These include: • PCIe generational capabilities. • PCIe bandwidth. • PCIe Device Pass-through. • PCIe SR-IOV as the VDU could require that an SR-IOV virtual function from the specified PCIe device can be allocated to the VM. • PCIe Device Assignment Affinity. The VDU could require for performance reasons the ability to allocate a partitionable PCIe Device capability such as a NIC port, an entire NIC or a NIC virtual function to the VDU while also ensuring that the selected device is locally connected to the same processor. (see note).
pcie_advanced_error_reporting	Leaf	0...1	Detecting and reporting correctable and un-correctable (fatal and non-fatal) PCIe errors to software for error handling and remediation.
platform_acceleration_device	Leaf	0...N	The VDU could have been developed, optimized or tested with an acceleration device such as a crypto accelerator that can typically be accessed over a PCIe bus.
NOTE: This could be relevant for use cases that are prescriptive about the locality of CPU HW Thread (vCPU) allocations, and would perform better with locally attached devices for efficiency/performance reasons such as avoiding unnecessary inter-socket communications.			

6.3.1.2.7 vnfd:vdu information elements related to network interfaces

An implementation could have particular network interface related requirements. This could be for many reasons such as requiring particular NIC offload features, or a particular driver model for performance reasons. By specifying the network interface related characteristics in the VDU descriptor, the VNF could be able to be deployed on suitably equipped platforms thereby facilitating a more efficient, higher performing and/or more reliable deployment. However, note that careful consideration will be given to the use of these elements as they could limit portability of a VNF. It is expected that a VNF is able to be deployed and instantiated by NFV-MANO even if these characteristics are not specified in the VDU descriptor.

Identifier	Type	Cardinality	Description
network_interface_card_capability	Leaf	0...N	The VDU could have been developed, optimized or tested with certain NIC capabilities to benefit items such as performance or scalability. These include: • TCP Large Segmentation Offload (LSO) for offload the segmentation of large TCP messages into MTU sized packets from the CPU to the NIC. • Large Receive Offload (LRO), i.e. the inverse of LSO by coalescing incoming TCP/IP packets into larger segments for processing in the CPU. • Checksum Offload. • Receive Side Scaling (RSS), for packet distribution between cores. • Flow Director, for more fine grained (than RSS) packet distribution between cores. • Mirroring of packets between interfaces. • Availability of Independent Rx/Tx queues for VM so that queue pairs in the NIC can be allocated to the VMs. • Jumbo Frame support. • VLAN tag stripping. • RDMA support. • SR-IOV support. • Data processing acceleration software library support, e.g. DPDK® - see note.
network_interface_bandwidth	Leaf	0...N	The network speed/bandwidth to be guaranteed per requested NIC.
data_processing_acceleration_library	Leaf	0...N	Name and version of the data processing acceleration library used. Orchestration can match any NIC that is known to be compatible with the specified library.
NOTE: DPDK® is an example of a suitable product available commercially. This information is given for the convenience of users of the present document and does not constitute an endorsement by ETSI of this product.			

6.3.1.2.8 vnfd:vdu information elements related to virtual switches

An implementation could have particular virtual switch related requirements. This could be for many reasons such as virtual switch feature set or configuration interface, or for performance characteristic reasons. By specifying the virtual switch related characteristics in the VDU descriptor, the VNF could be able to be deployed on suitably equipped platforms thereby facilitating a more efficient, higher performing and/or more reliable deployment. However, note that careful consideration will be given to the use of these elements as they could limit portability of a VNF. It is expected that a VNF is able to be deployed and instantiated by NFV-MANO even if these characteristics are not specified in the VDU descriptor.

Identifier	Type	Cardinality	Description
vswitch_capability	Leaf	0...N	The VDU could have been developed, optimized or tested with a particular vSwitch and could require specifying the vSwitch type, version and key features such as overlay tunnel termination support.

6.3.1.2.9 vnfd:vdu information elements related to general reliability and availability

An implementation could have particular reliability and availability related requirements. This could be for many reasons such as having an implementation that wishes to be aware of corrected or uncorrected errors in the platform for reliability reasons. By specifying the reliability and availability related characteristics in the VDU descriptor, the VNF could be able to be deployed on suitably equipped platforms thereby facilitating a more reliable deployment. However, note that careful consideration will be given to the use of these elements as they could limit portability of a VNF. It is expected that a VNF is able to be deployed and instantiated by NFV-MANO even if these characteristics are not specified in the VDU descriptor.

Identifier	Type	Cardinality	Description
corrected_error_notification	Leaf	0...N	For tracking system CORRECTABLE errors.
uncorrected_error_notification	Leaf	0...N	Detects UNCORRECTED system bus errors, ECC errors, parity errors, cache errors, and translation lookaside buffer errors - raising exceptions.

6.3.1.2.10 vnfd:vdu information elements related to storage

An implementation could have particular storage requirements. This could be for many reasons such as requiring storage size, type or performance characteristics. By specifying the storage characteristics in the VDU descriptor, the VNF could be able to be deployed on suitably equipped platforms thereby facilitating a more efficient, higher performing and/or more reliable deployment. However, note that careful consideration will be given to the use of these elements as they could limit portability of a VNF. It is expected that a VNF is able to be deployed and instantiated by NFV-MANO even if these characteristics are not specified in the VDU descriptor.

Identifier	Type	Cardinality	Description
storage_requirement	Leaf	0...1	Required storage characteristics (e.g. size), including Key Quality Indicators (KQIs) for performance and reliability/availability.
rdma_support_bandwitdh	Leaf	0...1	The VDU could have been developed, optimized or tested with a storage supporting RDMA over a given bandwidth.

6.3.1.3 VNF internal Virtual Link (vnfd:virtual_link)

Identifier	Type	Cardinality	Description
id	Leaf	1	Unique identifier of this internal Virtual Link.
connectivity_type	Leaf	1	Connectivity type (e.g. E-Line, E-LAN or E-Tree).
connection_points_references	Reference	2...N	References to Connection Points (vnfd:vdu:vnfc:connection_point:id, vnfd:connection_point:id), e.g. of type E-Line, E-Tree, or E-LAN.
root_requirement	Leaf	1	Describes required throughput of the link (e.g. bandwidth of E-Line, root bandwidth of E-Tree, and aggregate capacity of E-LAN).
leaf_requirement	Leaf	0...1	Describes required throughput of leaf connections to the link (for E-Tree and E-LAN branches).
qos	Leaf	0...N	Describes the QoS options to be supported on the VL e.g. latency, jitter, etc.
test_access	Leaf	0...1	Describes the test access facilities to be supported on the VL (e.g. none, passive monitoring, or active (intrusive) loopbacks at endpoints.

6.3.1.4 Connection Point (vnfd:connection_point)

Identifier	Type	Cardinality	Description
id	Leaf	1	ID of the Connection Point.
virtual_link_reference	Reference	0...1	References an internal Virtual Link (vnfd:virtual_link:id, see clause 6.3.1.3) to which other VDUs, NFs, and other types of endpoints can connect.
type	Leaf	1	This could be for example a virtual port, a virtual NIC address, a physical port, a physical NIC address or the endpoint of an IP VPN enabling network connectivity.

6.3.1.5 Deployment flavour element (vnfd:deployment_flavour)

Identifier	Type	Cardinality	Description
id	Leaf	1	ID of the VNF flavour
flavour_key	Leaf	1	Monitoring parameter & its value against which this flavour is being described. When present the parameters is as a vnfd:monitoring_parameter, see clause 6.3.1.1. An example is a flavour of a virtual PGW could be described in terms of the parameter "calls-per-second". There could be a flavour describing what it takes to support a vPGW with 10k calls per second.
constraint	Leaf	0...N	Constraint that this deployment flavour can only meet the requirements on certain hardware.
constituent_vdu	Element	1...N	Represents the characteristics of a constituent flavour element, see table below. Examples include Control-plane VDU & Data-plane VDU & Load Balancer VDU Each needs a VDU element to support the deployment flavour of 10k calls-per-second of vPGW, Control-plane VDU could specify 3 VMs each with 4 GB vRAM, 2 vCPU, 32 GB virtual storage, etc. Data-plane VDU could specify 2 VMs each with 8 GB vRAM, 4 vCPU, 64 GB virtual storage, etc.

Constituent VDU (vnfd:deployment_flavour:constituent_vdu)

Identifier	Type	Cardinality	Description
vdu_reference	Reference	1	References a VDU which is used for this deployment flavour by vnfd:vdu:id, see clause 6.3.1.2.1.
number_of_instances	Leaf	1	Number of VDU instances required
constituent_vnfc	Reference	1...N	References VNFCs which is used for this deployment flavour by vnfd:vdu:vnfc:id

6.3.2 VNF Record (vnfr)

6.3.2.0 Preamble

Following instantiation, a Virtualised Network Function Record will be created to index the virtualised resources allocated to each VNF instance, and allow traceability of associated links across multiple systems forming the NFV-MANO architectural framework. This record will include sufficient information to allow future changes to the deployed VNF instance in the light of a scalability update. VNFC instances related to a VDU are part of the VNF Record.

6.3.2.1 vnfr base elements

Identifier	Type	Cardinality	Description
auto_scale_policy	Leaf	0...N	See clause 6.3.1.1
connection_point	Element	1...N	See clause 6.3.1.1
dependency	Leaf	0...N	See clause 6.3.1.1
deployment_flavour	Reference	1	Reference to selected deployment flavour (vnfd:deployment_flavour:id)
id	Leaf	1	ID of the VNF instance
lifecycle_event	Leaf	0...N	See clause 6.3.1.1
localization	Leaf	0...1	A language attribute can be specified to identify default localization/language
monitoring_parameter	Leaf	1...N	Active monitoring parameters
vdu	Element	1...N	VDU elements describing the VNFC-related relevant information, see clause 6.3.1.1
vendor	Leaf	1	See clause 6.3.1.1
version	Leaf	1	See clause 6.3.1.1
virtual_link	Element	0...N	Internal Virtual Links instances used in this VNF, see clause 6.3.1.1
parent_ns	Reference	1...N	Reference to records of Network Service instances (nsr:id) that this VNF instance is part of
descriptor_reference	Reference	1	The reference to the VNFD used to instantiate this VNF
vnfm_id	Leaf	1	The identification of the VNFM entity managing this VNF
connected_external_virtual_link	Reference	1...N	Reference to a VLR (vlr:id) used for the management access path or other internal and external connection interface configured for use by this VNF instance
vnf_address	Leaf	1...N	A network address (e.g. VLAN, IP) configured for the management access or other internal and external connection interface on this VNF
status	Leaf	1	Flag to report status of the VNF (e.g. 0=Failed, 1= normal operation, 2= degraded operation, 3= offline through management action)
notification	Leaf	1...N	Listing of systems that have registered to received notifications of status changes
lifecycle_event_history	Leaf	0...N	Record of significant VNF lifecycle event (e.g. creation, scale up/down, configuration changes)
audit_log	Leaf	0...N	Record of detailed operational event, (e.g. VNF boot, operator logins, alarms sent)
runtime_policy_info	Leaf	0...N	Generic placeholder for input information related to VNF orchestration and management policies to be applied during runtime of a specific VNF instance (e.g. for VNF prioritization, etc.)

6.3.2.2 VNF internal Virtual Link (`vnfr:virtual_link`)

Identifier	Type	Cardinality	Description
`id`	Leaf	1	ID of the internal Virtual Link instance
`connectivity_type`	Leaf	1	See clause 6.3.1.3
`connection_point_reference`	Reference	2...N	Connection Points attached to this Virtual Link (`vnfr:connection_point`, `vnfr:vdu:vnfc_instance:connection_point`) e.g. of type E-Line, E-Tree, or E-LAN.
`root_requirement`	Leaf	1	Available throughput of the link (e.g. bandwidth of E-Line, root bandwidth of E-Tree, and aggregate capacity of E-LAN)
`leaf_requirement`	Leaf	0…1	Available throughput of leaf connections to the link (for E-Tree and E-LAN branches)
`qos`	Leaf	0…N	QoS options available on the VL, e.g. latency, jitter, etc.
`test_access`	Leaf	0…1	Test access facilities available on the VL (e.g. none, passive monitoring, or active (intrusive) loopbacks at endpoints

6.3.2.3 Connection Point (`vnfr:connection_point`)

Identifier	Type	Cardinality	Description
`id`	Leaf	1	ID of the Connection Point instance
`virtual_link_reference`	Reference	1	References an internal Virtual Link (`vnfr:virtual_link:id`, see clause 6.3.2.2) to which other NFs can connect
`type`	Leaf	1	See clause 6.3.1.4

6.3.2.4 Virtual Deployment Unit (vnfr:vdu)

6.3.2.4.1 vnfr:vdu base element

Identifier	Type	Cardinality	Description
computation_requirement	Leaf	0...1	See clause 6.3.1.2.1
constraint	Leaf	0...N	See clause 6.3.1.2.1
lifecycle_event	Leaf	0...N	See clause 6.3.1.2.1
high_availability	Leaf	0...1	See clause 6.3.1.2.1
number_of_instances	Leaf	1	See clause 6.3.1.2.1
scale_in_out	Leaf	0...1	See clause 6.3.1.2.1
virtual_network_bandwidth_ resource	Leaf	1	See clause 6.3.1.2.1
vm_image	Leaf	1	See clause 6.3.1.2.1
vnfc_instance	Element	1...N	Contains information that is distinct for each VNFC instance created based on this VDU
parent_vdu	Reference	1	Reference to the VDU (vnfd:vdu:id) used to instantiate this element.
cpu_instruction_set_extension	Leaf	0...N	See clause 6.3.1.2.2
cpu_model	Leaf	0...N	See clause 6.3.1.2.2
cpu_model_specification_binding	Leaf	0...1	See clause 6.3.1.2.2
cpu_min_clock_speed	Leaf	0...1	See clause 6.3.1.2.2
cpu_core_reservation	Leaf	0...1	See clause 6.3.1.2.2
cpu_simultaneous_multi_ threading_hw_thread_ specification	Leaf	0...1	See clause 6.3.1.2.2
cpu_core_oversubscription_ policy	Leaf	0...1	See clause 6.3.1.2.2
cpu_core_and_hw_thread_ allocation_topology_policy	Leaf	0...1	See clause 6.3.1.2.2
cpu_last_level_cache_size	Leaf	0...1	See clause 6.3.1.2.2
cpu_direct_io_access_to_cache	Leaf	0...1	See clause 6.3.1.2.2
cpu_translation_look_aside_ buffer_parameter	Leaf	0...N	See clause 6.3.1.2.2
cpu_hot_add	Leaf	0...1	See clause 6.3.1.2.2
memory_parameter	Leaf	0...N	See clause 6.3.1.2.3
memory_hot_add	Leaf	0...1	See clause 6.3.1.2.3
platform_security_parameter	Leaf	0...N	See clause 6.3.1.2.4
hypervisor_parameter	Leaf	1...N	See clause 6.3.1.2.5
platform_pcie_parameter	Leaf	0...N	See clause 6.3.1.2.6
pcie_advanced_error_reporting	Leaf	0...1	See clause 6.3.1.2.6
platform_acceleration_device	Leaf	0...N	See clause 6.3.1.2.6
network_interface_card_ capability	Leaf	0...N	See clause 6.3.1.2.7
polled_mode_driver	Leaf	0...N	See clause 6.3.1.2.7
vswitch_capability	Leaf	0...N	See clause 6.3.1.2.8
corrected_error_notification	Leaf	0...N	See clause 6.3.1.2.9
uncorrected_error_notification	Leaf	0...N	See clause 6.3.1.2.9
storage_requirement	Leaf	0...1	See clause 6.3.1.2.10

6.3.2.4.2 VNFC instance (vnfr:vdu:vnfc_instance)

Identifier	Type	Cardinality	Description
id	Leaf	1	Unique VNFC identification within the namespace of a specific VNF instance.
vim_id	Leaf	1	VIM instance identification that manages the virtualisation container associated with this VNFC instance.
vc_id	Leaf	1	Unique virtualisation container identification (e.g. VM id) within the namespace managed by the corresponding VIM instance.
connection_point	Element	1...N	See clause 6.3.2.4.3.

6.3.2.4.3 Connection Point (`vnfr:vdu: vnfc_instance:connection_point`)

Identifier	Type	Cardinality	Description
`id`	Leaf	1	ID of the Connection Point.
`virtual_link_reference`	Reference	1	References an internal Virtual Link (`vnfd:virtual_link:id`, see clause 6.3.1.3) to which other VNFCs can connect.
`type`	Leaf	1	This could be for example a virtual port, a virtual NIC address, a physical port, a physical NIC address or the endpoint of an IP VPN enabling network connectivity.

6.4 Virtual Link information elements

6.4.0 Preamble

Exposure of connectivity options to the NFVO will require specified Virtual Link Descriptors.

The NFVO can require information from a VNFFG to determine what connectivity facilities are needed to interconnect VNFs, and this is likely to be with a "technology neutral" abstraction. Data will be required to be passed to a lower level system to enable logical configuration of pre-existing hardware and software networking components to fulfil a connectivity request.

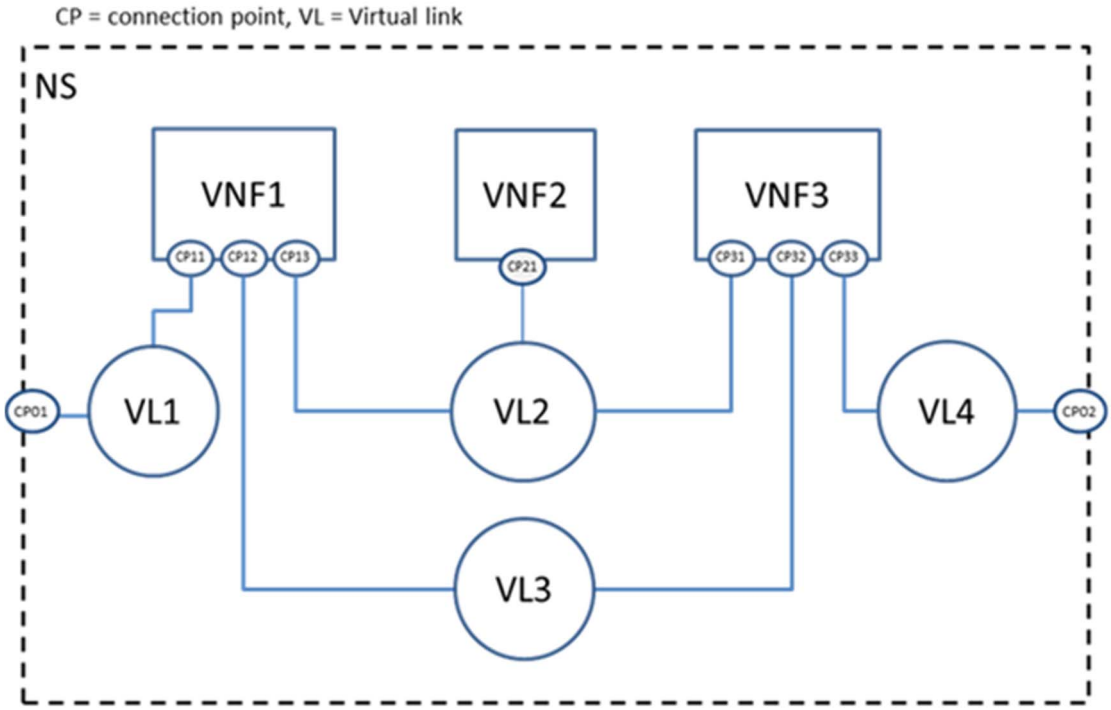


Figure 6.3: Network connection topology of a Network Service using VNFs, VLs, and Connection Points

In ETSI GS NFV-INF 001 [i.4] three example service types have been identified for VNF connections, aligned with the MEF E-Line, E-LAN, and E-Tree services. Use cases for these could be as follows:

- E-Line - For a simple point to point connection between a VNF and the existing network.
- E-LAN - where a VNF instance exchanges information with all other VNF instances within a Network Service to ensure synchronization.
- E-Tree - where traffic from a trunk interface is routed to a specific branch, such as in a load balancing application or management system connections.

Implementation of these services within the Infrastructure network can be dependent on the physical locations of the link end points. For example, an E-Line link between two VNFs sharing a hypervisor on one server could be connected using a Virtual Switch (vSwitch), a switch function in the Network Interface Controller (NIC), or a Virtual Ethernet Port Aggregator (VEPA) on an external Ethernet switch.

Likewise, ETSI GS NFV-INF 005 [i.7] also describes Layer 3 Services, such as Layer 3 Provider Provisioned VPN (L3VPN) services and others.

Infrastructure vendors could be obliged to make all options available to accommodate Service Provider requirements for performance and Security. Similarly, options involving L2 in L3 tunnelling could be required where the VNFs are instantiated on physically separated servers.

6.4.1 Virtual Link Descriptor (vld)

The VLD provides a description of each Virtual Link. This type of information can be used by the NFVO to determine the appropriate placement of a VNF instance, and by the VIM responsible for managing the virtualised resources of the selected placement to determine the allocation of required virtualised resources on a host with adequate network infrastructure. The VIM, or another Network Controller, can also use this information to establish the appropriate paths and VLANs.

The VLD describes the basic topology of the connectivity (e.g. E-LAN, E-Line, E-Tree) between one or more VNFs connected to this VL and other required parameters (e.g. bandwidth and QoS class). The VLD connection parameters are expected to have similar attributes to those used on the ports on VNFs in ETSI GS NFV-SWA 001 [i.8]. Therefore a set of VLs in a Network Service can be mapped to a Network Connectivity Topology (NCT) as defined in ETSI GS NFV-SWA 001 [i.8]. Figure 6.3 shows an example of a Network Connection Topology (NCT) described by the use of VLs referencing Connection Points of VNFs and the NS.

Identifier	Type	Cardinality	Description
id	Leaf	1	ID of the VLD
vendor	Leaf	1	Vendor generating this VLD
descriptor_version	Leaf	1	Version of this VLD
number_of_endpoints	Leaf	1	Number of endpoints available on this VL (e.g. E-Line=2)
root_requirement	Leaf	1	Throughput of the link (e.g. bandwidth of E-Line, root bandwidth of E-Tree, and aggregate capacity of E-LAN)
leaf_requirement	Leaf	0...1	Throughput of leaf connections to the link (for E-Tree and E-LAN branches)
qos	Leaf	0...N	QoS options available on the VL, e.g. latency, jitter, etc.
test_access	Leaf	1	Test access facilities available on the VL (e.g. none, passive monitoring, or active (intrusive) loopbacks at endpoints
connection	Reference	2...N	A reference to an attached Connection Point (nsd/vnfd/pnfd:connection_point:id)
connectivity_type	Leaf	1	Connectivity types, e.g. E-Line, E-LAN, or E-Tree.
vld_security	Leaf	0...1	This is a signature of vld to prevent tampering. The insertion of the particular hash algorithm used to compute the signature, together with the corresponding cryptographic certificate to validate the signature is recommended.

6.4.2 Virtual Link Record (vlr)

Records are expected to be kept of the resources used to implement a virtual network link, including bandwidth.

It is assumed that the VLR information elements contain the same elements as the VLD. If elements are added or removed this is highlighted in the respective clause.

NOTE: In the E-Line case where there are two end points, the record could be attached to the VNFR Connection Point, but this would also expect to include an identifier of the "other" end of the link for service assurance and fault management purposes. In the E-Tree case there are trunk and branch end points. The VNFR could still be used at the branch augmented with an identifier of the "trunk" end. At the Trunk end, multiple "other" (e.g. branch) end point identifiers would be needed, which gets extended every time a new branch is added. In the E-LAN case there are multiple end points and mesh connectivity. This could mean that each VNFR would have a list of all the other end points in the mesh, and that every VNFR would be updated each time a new end point is added to the LAN. In this scenario it gets hard to identify what is the "master" version of the configuration to recover in the event of a database corruption. Again an alternative is that all the end points could have a 1:1 relationship with the VLR holding the definition of the entire mesh in one place. Noting that every VNF needs at least one port connection, it is proposed that the simplest solution is to have a consistent approach using a VLR including the bandwidth and QoS class. The endpoint VNFR would then only need a pointer to the VLR associated with each SWA port interface. Consequently it is suggested that a VLR record will represent each deployed Virtual Link instance.

Identifier	Type	Cardinality	Description
`id`	Leaf	1	ID if the Virtual Link instance
`vendor`	Leaf	1	See clause 6.4.1
`version`	Leaf	1	See clause 6.4.1
`number_of_enpoints`	Leaf	1	See clause 6.4.1
`root_requirement`	Leaf	1	See clause 6.4.1
`leaf_requirement`	Leaf	0...1	See clause 6.4.1
`qos`	Leaf	0...N	See clause 6.4.1
`test_access`	Leaf	1	See clause 6.4.1
`connection`	Reference	1...N	References to Connection Point instances (`nsr/vnfr/pnfr:connection_point:id`), see clause 6.4.1
`parent_ns`	Reference	1	The reference for the Network Service instance (`nsr:id`) that this VL instance is part of
`vnffgr_reference`	Reference	0...N	References to the records of the VNFFG instances in which this VL instance participates
`descriptor_reference`	Reference	1	Reference to the id of VLD used to instantiate this VL
`vim_id`	Leaf	1	The reference to the system managing this VL instance
`allocated_capacity`	Leaf	1...N	Bandwidth allocated for each of the QoS options on this link
`status`	Leaf	1	Flag to report status of the VL (e.g. 0=Link down, 1= normal operation, 2= degraded operation, 3= Offline through management action)
`notification`	Leaf	1...N	System that has registered to received notifications of status changes
`lifecycle_event_history`	Leaf	0...N	Record of significant VL lifecycle event (e.g. Creation, Configuration changes)
`audit_log`	Leaf	0...N	Record of detailed operational events (e.g. link up/down, Operator logins, Alarms sent)
`connectivity_type`	Leaf	1	Connectivity types, e.g. E-Line, E-LAN, or E-Tree

6.4.3 Relation between internal Virtual Links and Virtual Links

Figure 6.4 illustrates the combined use of internal Virtual Links as described in clause 6.3 and (external) Virtual Links as described in clause 6.4.1. In this example VNF2 comprises 3 VNFCs connected to an internal Virtual Link. The first VNFC has two Connection Points: one to the external Virtual Link, another one to the internal Virtual Link. The Virtual Links are those depicted in figure 6.3.

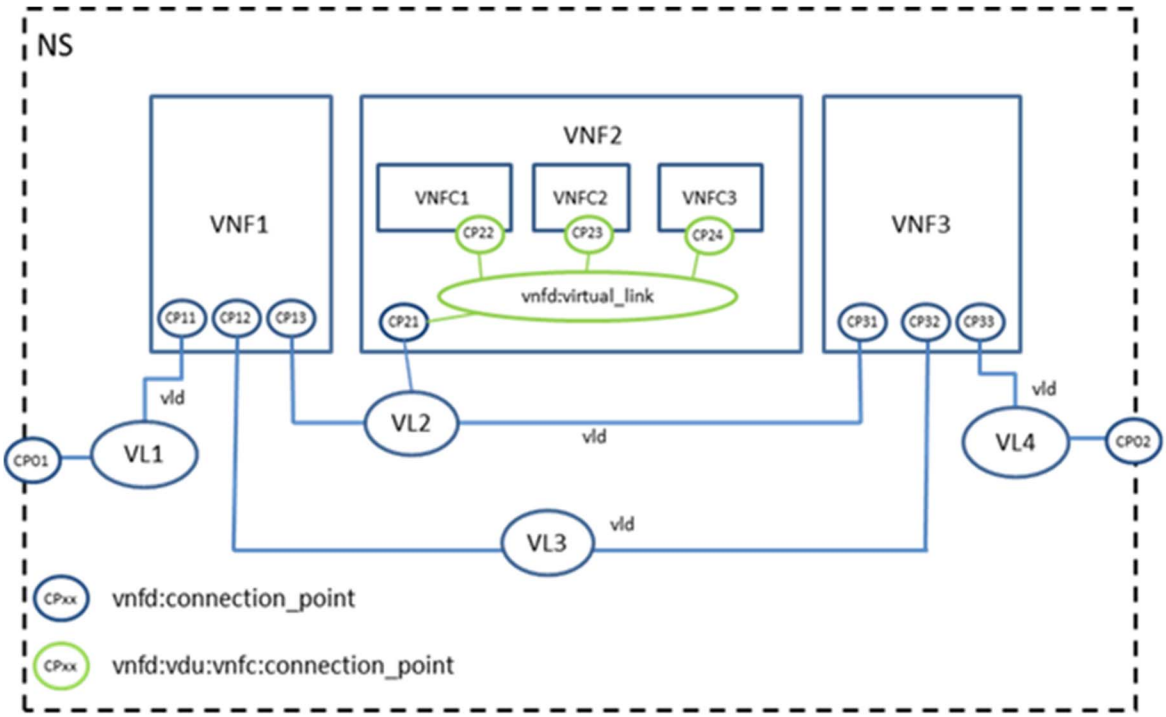


Figure 6.4: Relationships between internal and external Virtual Links

6.5 Virtualised Network Function Forwarding Graph information elements

6.5.0 Preamble

A VNFFG information element contains metadata about the VNF Forwarding Graph itself, references to VLs, VNFs and PNFs (as shown for instance in use cases described in clause F.1), and Network Forwarding Path elements, which in turn include policies (e.g. MAC forwarding rules, routing entries, etc.) and references to Connection Points (e.g. virtual ports, virtual NIC addresses, etc.).

NOTE: When no Network Forwarding Paths element is included in a VNF Forwarding Graph descriptor, the forwarding path applicable to a traffic flow is decided at runtime (e.g. routeing of signalling messages across call servers).

Figure 6.5 shows an example of two VNFFGs established on top of a network topology. VNFFG1 has two NFPs (VNFFG1:NFP1 and VNFFG1:NFP2) whereas VNFFG2 only has a single NFP (VNFFG2:NFP1).

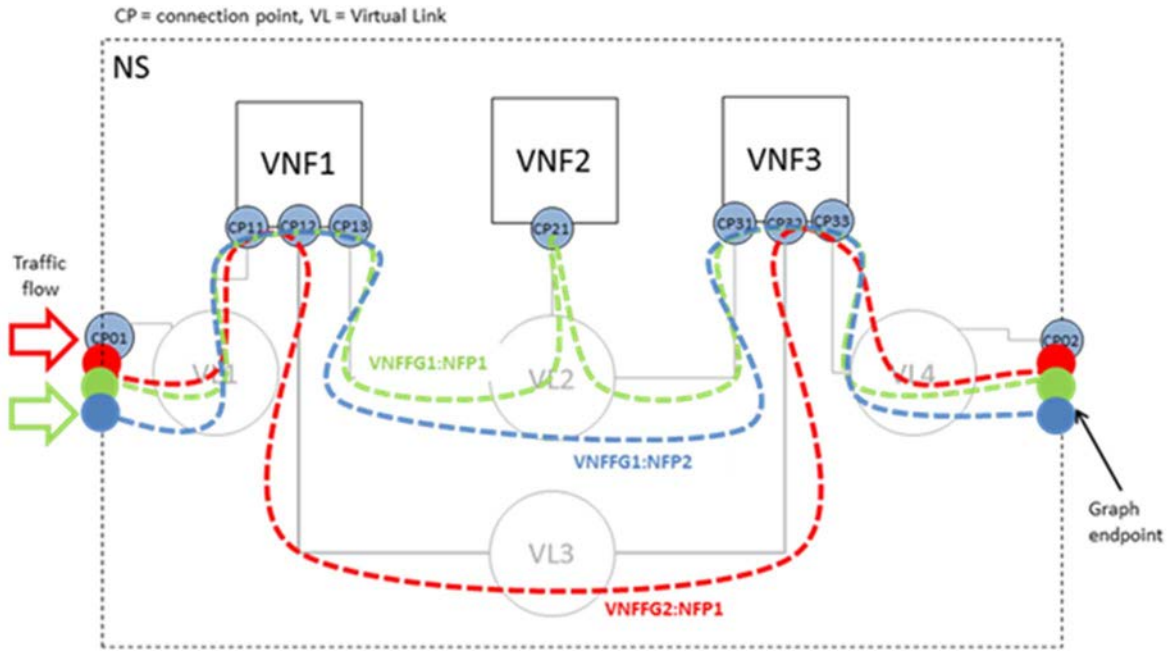


Figure 6.5: Network Service with two VNFFGs with different NFPs

The NFV Architectural Framework (ETSI GS NFV 002 [i.2]) defines a Network Service as the subset of the end to end service formed by Virtualised Network Functions and associated Virtual Links instantiated on the NFVI, as shown in figure 6.6.

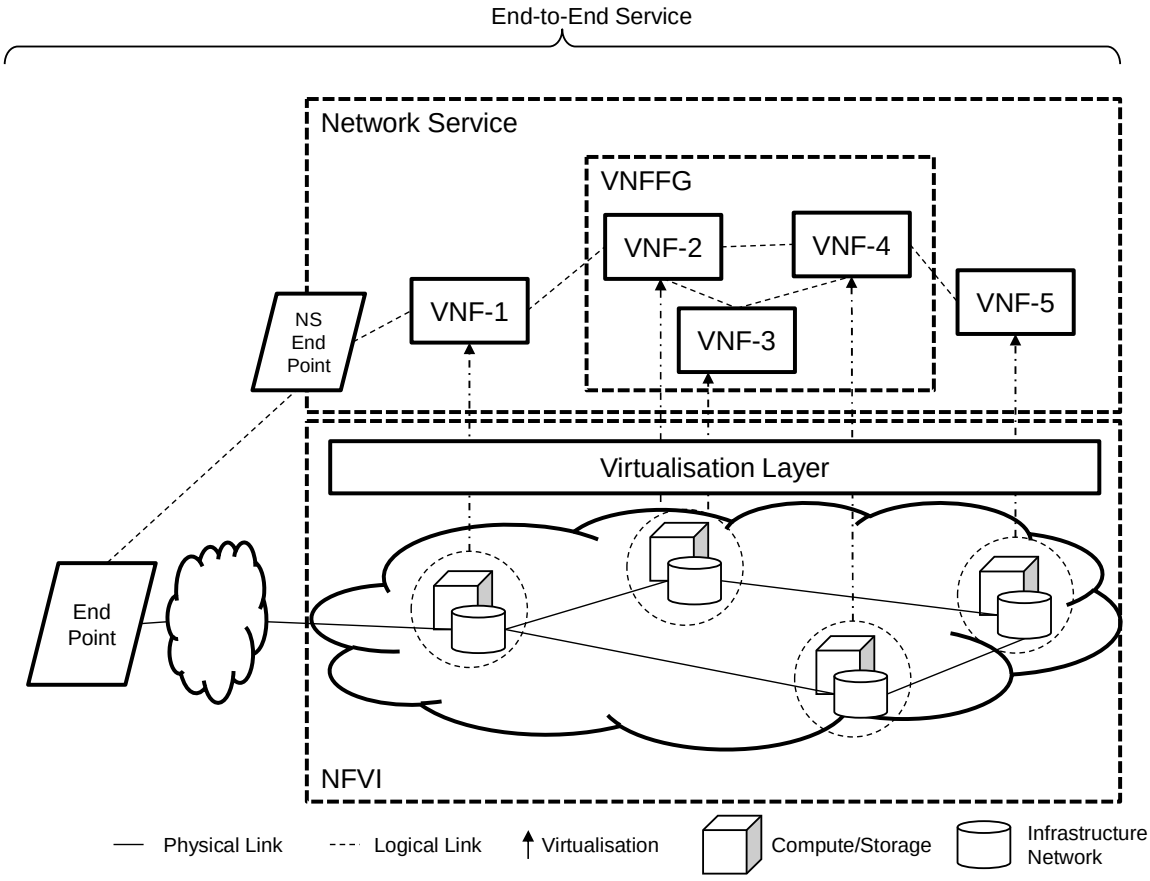


Figure 6.6: Example of a Network Service as part of an end-to-end service with VNFs and a forwarding graph

The Network Service Descriptor, used as a deployment template by NFVO, will include references to three subcomponents:

- Zero or more VNF Forwarding Graphs identifying the required VNFDs, and their required VLDs.
- A definition of the interface to the physical devices, enabling the NFVO to select the appropriate VLDs as defined in the PNFD.

VNF Forwarding Graphs (VNFFGs) are expected to be developed by service providers or their systems integration partners, and could be customized from industry specified templates that encapsulate the common telecommunication service patterns.

After instantiation of the VNFFG, the VNF Forwarding Graph Record (VNFFGR) has references to the VNF, PNF, and Virtual Link instances used to instantiate the VNF Forwarding Graph. The VNFFG can have a requirement for adding a service ID/tag to the VNFFGR and all VNF and PNF records used to fulfil the described VNFFG. This can help fingerprint/track everything related to that service in a NFVO world (that could span multiple CMSs), and will be needed to ensure that effective test access can be achieved for fault localization.

Further study is needed to determine how the VNFFG will track changes to the deployed instances, such as additional VNF instances being brought into service to handle scalability required for a traffic peak, and migration of VNF workloads to an alternative infrastructure to enable operational maintenance or provide business continuity in the event of major link or site failure. VNF instance migration is not addressed in the present document.

6.5.1 VNF Forwarding Graph Descriptor (`vnffgd`)

6.5.1.1 `vnffgd` base element

A top level descriptor referencing the information elements used by this VNFFG.

Identifier	Type	Cardinality	Description
`id`	Leaf	1	ID of the VNFFG Descriptor
`vendor`	Leaf	1	Specify the vendor generating this VNFFG
`version`	Leaf	1	Specify the identifier (e.g. name), version, and description of service this VNFFG is describing
`number_of_endpoints`	Leaf	1	Count of the external endpoints (`connection_point` elements) included in this VNFFG, to form an index
`number_of_virtual_links`	Leaf	1	Count of the VLs (`dependent_virtual_link` elements) used by this VNFFG, to form an index
`dependent_virtual_link`	Reference	1...N	Reference to a VLD (`vld:id`) used to instantiate this Forwarding Graph
`network_forwarding_path`	Element	0...N	This element describes a Network Forwarding Path within the VNFFG, see clause 6.5.1.2
`connection_point`	Reference	2...N	Reference to Connection Points (`nsd/vnfd/pnfd:connection_point:id`) forming the VNFFG including Connection Points attached to PNFs, see clause 6.6.1.2
`descriptor_version`	Leaf	1	Version of this VNFFGD
`constituent_vnfs`	Reference	1...N	Reference to a VNFD (`nsd:deployment_flavours:constituent_vnf:id`) used to instantiate this VNF Forwarding Graph
`vnffgd_security`	Leaf	0...1	This is a signature of vnffgd to prevent tampering. The insertion of the particular hash algorithm used to compute the signature, together with the corresponding cryptographic certificate to validate the signature is recommended

6.5.1.2 Network Forwarding Path (vnffgd:network_forwarding_path)

Name	Type	Cardinality	Description
id	Leaf	1	Specify the identifier (e.g. name) of the Network Forwarding Path
policy	Leaf	0...1	A policy or rule to apply to the NFP
connection	Leaf	1...N	A tuple containing a reference to a Connection Point in the NFP and the position in the path

6.5.2 VNF Forwarding Graph Record (vnffgr)

6.5.2.0 Preamble

A top level record of the specific component resource instances used to form this VNFFG instance, and its operational status.

6.5.2.1 vnffgr base element

Identifier	Type	Cardinality	Description
id	Leaf	1	A unique identifier for the instance of the VNFFG
descriptor_reference	Reference	1	Record of the VNFFGD (vnffgd:id) used to instantiate this VNFFG
parent_ns	Reference	1	Reference to the record (nsr:id) for Network Service instance that this VNFFG instance is part of
dependent_virtual_link	Reference	1...N	Reference to record for Virtual Link instance (vlr:id) used to instantiate this VNFFG
status	Leaf	1	Flag to report status of the VNFFG (e.g. 0=Failed, 1= normal operation, 2= degraded operation, 3= Offline through management action)
notification	Leaf	1...N	Listing of systems that have registered to received notifications of status changes
lifecycle_event_history	Leaf	0...N	Record of significant VNFFG lifecycle events (e.g. creation, configuration changes)
audit_log	Leaf	0...N	Record of detailed operational events, (e.g. graph up/down, alarms sent)
network_forwarding_path	Element	0...N	List of Connection Points which form a Network Forwarding Path and description of policies to establish and rules to choose the path
connection_point	Reference	2...N	Reference to Connection Points (nsr/vnfr/pnfr:connection_point:id) forming the VNFFG
member_vnfs	Reference	1...N	VNF instance used to instantiate this VNFFG
vendor	Leaf	1	See clause 6.5.1.1
version	Leaf	1	See clause 6.5.1.1
number_of_endpoints	Leaf	1	See clause 6.5.1.1
number_of_vnfs	Leaf	1	See clause 6.5.1.1
number_of_pnfs	Leaf	1	See clause 6.5.1.1
number_of_virtual_links	Leaf	1	See clause 6.5.1.1

6.5.2.2 Network Forwarding Path (vnffgr:network_forwarding_path)

Name	Type	Cardinality	Description
id	leaf	1	A unique identifier for the instance of the NFP
policy	leaf	0...1	A policy or rule to apply to the NFP
connection	leaf	1...N	A tuple containing a reference to a Connection Point instance in the NFP instance and the position in the path

6.6 Physical Network Function information elements

6.6.1 PNF Descriptor (pnfd)

6.6.1.0 Preamble

The PNF Descriptor (PNFD) will be used by the NFVO to create links between VNFs and the Physical Network Functions, probably by communication with an appropriate specialized Virtual Infrastructure Manager (e.g. WAN Infrastructure Manager) using a pre-NFV existing interface comprised in the Or-Vi reference point.

The Information contained within the PNFD is limited to the description of the connectivity requirements to integrate PNFs. It contains:

- The Connection Points exposed by the PNF.
- The Virtual Link/links the Connection Point attaches to.

6.6.1.1 pnfd base element

Identifier	Type	Cardinality	Description
id	Leaf	1	The ID (e.g. name) of this PNFD.
vendor	Leaf	1	The vendor generating this PNFD.
version	Leaf	1	The version of PNF this PNFD is describing.
description	Leaf	1	Description of physical device functionality.
connection_point	Element	1...N	This element describes an external interface exposed by this PNF enabling connection with a VL.
descriptor_version	Leaf	1	Version of the PNF descriptor.
pnfd_security	Leaf	0...1	This is a signature of pnfd to prevent tampering. The insertion of the particular hash algorithm used to compute the signature, together with the corresponding cryptographic certificate to validate the signature is recommended.

6.6.1.2 Connection Point (pnfd:connection_point)

Identifier	Type	Cardinality	Description
id	Leaf	1	ID of the Connection Point.
type	Leaf	1	This could be for example a virtual port, a virtual NIC address, a physical port, a physical NIC address or the endpoint of an IP VPN enabling network connectivity.

6.6.2 PNF Record (pnfr)

6.6.2.1 pnfr base element

Following instantiation of a Network Service, PNF records would be created in an inventory indexed to the Network Service Record to facilitate service management activities.

Name	Type	Cardinality	Description
id	Leaf	1	A unique identifier for the instance of the PNF
vendor	Leaf	1	See clause 6.6.1.1
version	Leaf	1	See clause 6.6.1.1
description	Leaf	1	See clause 6.6.1.1
connection_point	Element	1...N	See clause 6.6.1.1
parent_ns	Reference	1	The reference for the record of the NS instance (nsr:id) that this PNF instance is part of
descriptor_reference	Reference	1	The reference to the version of PNFD (pnfd:version) used to instantiate this PNF
vnffgr	Reference	0...N	References to the records of VNFFG instances to which this PNF is participating
oam_reference	Leaf	1	The reference to the system managing this PNF
connected_virtual_link	Reference	1.,.N	References to the VLRs (vlr:id) used to for the management access path and all other external connection interfaces configured for use by this PNF instance
pnf_address	Leaf	1...N	The network addresses (e.g. VLAN, IP) configured for the management access and all other external connection interfaces on this PNF

6.6.2.2 Connection Point (pnfr:connection_point)

Identifier	Type	Cardinality	Description
id	Leaf	1	ID of the Connection Point instance
type	Leaf	1	See clause 6.6.1.2

6.7 VNF Instantiation input parameter

VNF Instantiation Input Parameters are exchanged in the messages between functional blocks in the NFV-MANO architectural framework. In the present clause, only input parameters considered for the VNF instantiation operation are covered. Other input parameters potentially needed for other VNF lifecycle operations can be covered later.

The following is not a complete list of parameters but a minimum subset considered necessary to instantiate a VNF and identify the required deployment flavour for VNF. Parameters for operations on VNF component level (per VDU) have not been explored and needed to be detailed; in the absence of those, the default in the VNFD/VDU would apply.

Information element	Cardinality	Description
VNFD identification	1	Information that uniquely identifies the VNFD based upon which this VNF is being instantiated.
Scaling methodology	0...1	On-demand, auto-scaling, or scaling based on a management request.
Flavour ID	1	The ID of the deployment flavour of the VNFD based on which this VNF needs instantiation. Note that these parameters are from amongst the list specified in the VNFD, see clause 6.3.1.
Threshold descriptor	0...N	A threshold descriptor has the info including but not restricted to monitoring parameter, threshold values, action, etc. Note that these monitoring parameters are from amongst the list specified in the VNFD, see clause 6.3.1.2.1.
Auto-scale policy value	0...N	Represents the values for the policy meta data, which can include the values for criteria parameter & action-type. The available criteria parameter and action types are defined in vnfd:auto_scale_policy. Example of values are: • Calls-per-second: > 100k. • Action-type: scale-out to flavour ID 3.
Constraints	0...N	Represents a placeholder for any specific constraints. EXAMPLE: Geographical location for each of the VNF components, based on respective VDUs.

Information element	Cardinality	Description
NS instance identification	0...N	Represents information that uniquely identifies NS instance(s) to which this VNF instance participates.
VNF address	0...N	Represents an address assigned to the VNF instance, bound to one of its connection points (as described in clause 6.3.2.1 vnfr:base element). See note.
NOTE: A cardinality of 0 corresponds to the case where all addresses used by a VNF instance are dynamically assigned by Management & Orchestration functions.		

6.8 Network Service Instantiation Input Parameters

Network Service (NS) Instantiation Input Parameters are exchanged in the messages functional blocks in the NFV-MANO architectural framework. In the present document, only input parameters considered for the NS instantiation operation are covered. Other input parameters potentially needed for other NS lifecycle operations can be covered later. The following is not a complete list of parameters but a minimum subset considered necessary to instantiate a Network Service.

Information element	Cardinality	Description
NSD identification	1	Information that uniquely identifies the NSD based upon which this Network Service is being instantiated.
Reference to an existing VNF instance	1...N	List of references to existing VNF instances. This is required, for example, when the Network Service is being instantiated bottom-up wherein the members are already existing and are chained together.
Scaling methodology	0...1	On-demand, auto-scaling, or scaling based on a management request.
Flavour ID	1	The ID of the deployment flavour of the NSD based on which this NS needs instantiation. Note that these parameters are from amongst the list specified in the NSD, see clause 6.2.1.3).
Threshold descriptor	0...N	A threshold descriptor has the info including but not restricted to monitoring parameter, threshold values, action, etc. Note that these monitoring parameters are from amongst the list specified in the NSD (refer clause 6.2.1.1).
Auto-scale policy value	0...N	Represents the values for the policy meta data, which can include the values for criteria parameter & action-type. The available criteria parameter and action types are defined in `nsd:auto_scale_policy`. The NS is scaled out/in according to the increment/decrement corresponding to the difference in the resources corresponding to the current flavour ID and scaling flavour ID. Example values are: • Calls-per-second: > 100k. • Action-type: scale-out to flavour ID 3.
Constraints	0...N	Represents a placeholder for any specific constraints. EXAMPLE: Geographical location, requirement to first create an NFVI resource reservation, etc.

7 NFV-MANO interfaces

7.0 Preamble

Interfaces are defined focusing on the function they expose. Recommendation are made with respect to which functional blocks can expose a particular interface and which functional blocks can consume such an interface; this supports mapping of an interface to a reference point, while identifying the producer and consumer of such interface in that reference point.

7.1 Interfaces concerning Network Services

7.1.1 Network Service Descriptor management

7.1.1.1 Description

Interface Name	Network Service Descriptor Management			
Description	This interface allows an authorized consumer functional block to manage the Network Service Descriptor (NSD), including any related VNFFGD and VLD.			
Notes	While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties.			
Produced By	NFVO.			
Consumed By	OSS.			
Applicable Reference Point(s)	Os-Ma-nfvo.			

7.1.1.2 Operations

Operations	Description	Notes
On-board Network Service Descriptor	This operation allows submitting and validating a Network Service Descriptor (NSD), including any related VNFFGD and VLD.	Upon successful completion the Network Service Descriptor is stored in the NS catalogue, and can be used for Network Service lifecycle management.
Disable Network Service Descriptor	This operation allows disabling a Network Service Descriptor, so that it is not possible to instantiate it any further.	Upon successful completion, that Network Service Descriptor is disabled in the NS catalogue. This operation has no effect on the NS instances previously created, using the NSD.
Enable Network Service Descriptor	This operation allows enabling a Network Service Descriptor.	Upon successful completion, that Network Service Descriptor is enabled in the NS catalogue. This operation has no effect on the NS instances previously created, using the NSD.
Update Network Service Descriptor	This operation allows updating a Network Service Descriptor (NSD), including any related VNFFGD and VLD. This update might include creating/deleting new VNFFGDs and/or new VLDs.	Upon successful completion, the Network Service Descriptor is updated in the NS catalogue (it can be a new Network Service Descriptor, depending on a case-by-case). This operation has no effect on NS instances previously created, using the original NSD.
Query Network Service Descriptor	This operation is used to query the information of the Network Service Descriptor (NSD), including any related VNFFGD and VLD.	The operation allows retrieving information from the NSD, VNFFGDs, and VLDs. Examples include: NSD version, list of participating VNFs, service_deployment flavour, auto_scale_policy, etc.
Delete Network Service Descriptor	This operation is used to remove a disabled Network Service Descriptor.	

7.1.2 Network Service lifecycle management

7.1.2.1 Description

Interface Name	Network Service Lifecycle Management.			
Description	This interface allows an authorized consumer functional block to perform lifecycle operations on Network Service instances, such as instantiate, terminate, query, etc. This includes (not limited to) managing the associations between different VNFs, and of VNFs when connected to PNFs, the topology of the Network Service, and the VNF Forwarding Graphs associated with the service.			
Notes	While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties.			
Produced By	NFVO.			
Consumed By	OSS.			
Applicable Reference Point(s)	Os-Ma-nfvo.			

7.1.2.2 Operations

Operations	Description	Notes
Instantiate Network Service	This operation allows instantiating a Network Service.	As part of the instantiation of NS, some kind of resource reservation information can be exchanged.
Terminate Network Service	This operation allows terminating a Network Service instance. Graceful or forceful termination might be possible based on input parameter.	
Query Network Service	This operation allows retrieving Network Service instance attributes.	
Scale Network Service	This operation allows scaling a Network Service instance.	
Update Network Service	This operation allows updating a Network Service instance.	Different options can be covered under this operation. For example, an update can perform instantiation of a new VNF instance and updating an existing VNF Forwarding Graph.
Create VNFFG	This operation allows creating a new VNF Forwarding Graph instance for a given Network Service instance.	This can be considered as a sub-operation of the Update Network Service operation.
Delete VNFFG	This operation allows deleting an existing VNF Forwarding Graph instance within a Network Service instance.	This can be considered as a sub-operation of the Update Network Service operation.
Query VNFFG	This operation allows retrieving VNFFG instance attributes.	This can be considered as a sub-operation of the Update Network Service operation.
Update VNFFG	This operation allows updating an existing VNF Forwarding Graph instance for a given Network Service instance.	This can be considered as a sub-operation of the Update Network Service operation.
Create VL	This operation allows creating a new VL for a given Network Service instance.	
Delete VL	This operation allows deleting an existing VL within a Network Service instance.	
Update VL	This operation allows updating an existing VL for a given Network Service instance.	
Query VL	This operation allows retrieving VL instance attributes.	

7.1.3 Network Service lifecycle change notification

7.1.3.1 Description

Interface Name	Network Service Lifecycle Change Notification			
Description	This interface is used to provide runtime notifications related with the changes made to Network Service instances including (not limited to) instantiating/terminating/modifying Network Service, adding/deleting VNF to a NS, adding/deleting/changing VNF Forwarding Graphs and VLs in a NS. These notifications are triggered after completion of the corresponding lifecycle operation.			
Notes	(1) These notifications facilitate updating consuming functional blocks regarding completion of operations that could have been triggered earlier (e.g. for keeping the OSS service management updated). (2) This interface is complementary to the Network Service Lifecycle Management interface. (3) Guaranteeing delivery of notifications is considered as an implementation issue and deliberately left out of scope. (4) As the information carried in notifications is the most interesting for notification interfaces, the Notify operation is described, while details of the mechanism for registering for notifications (e.g. subscribe) and all other possible operations related to the actual notification delivery mechanism are deliberately left out of scope. (5) The event notification type and format is not in-scope for the present document. (6) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties.			
Produced By	NFVO.			
Consumed By	OSS.			
Applicable Reference Point(s)	Os-Ma-nfvo.			

7.1.3.2 Operations

Operations	Description	Notes
Notify	This operation allows providing lifecycle change notifications on Network Services.	

7.1.4 Network Service performance management

7.1.4.1 Description

Interface Name	Network Service Performance Management			
Description	This interface allows performance management (measurement results collection and notifications) on Network Services. Network Service metrics might be calculated from measurement results coming from the underlying layers. Examples of Network Service metrics might be related to network performance, e.g. latency and usage of the Virtual Links, e.g. average/maximum bandwidth used for a certain time interval, or resource consumption, e.g. number of VMs or vCPUs used by this Network Service.			
Notes	(1) Guaranteeing delivery of notifications is considered as an implementation issue and deliberately left out of scope. (2) As the information carried in notifications is the most interesting for notification interfaces, the Notify operation is described, while details of the mechanism for registering for notifications (e.g. subscribe) and all other possible operations related to the actual notification delivery mechanism are deliberately left out of scope. (3) NFVO can forward performance information to the consumer functional block received from other functional blocks, mapped to VNF, NS or some combination of those. (4) The event notification type and format is not in-scope for the present document. (5) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties.			
Produced By	NFVO.			
Consumed By	OSS.			
Applicable Reference Point(s)	Os-Ma-nfvo.			

7.1.4.2 Operations

Operations	Description	Notes
Get performance measurement results	This operation allows collecting performance measurement results generated on Network Services.	
Notify	This operation allows providing performance notifications on Network Services.	

7.1.5 Network Service fault management

7.1.5.1 Description

Interface Name	Network Service Fault Management.			
Description	This interface is used to provide fault information on Network Services. These include (not limited to) fault information resulting from the processing of information received from other functional blocks, as well as forwarding of fault information received from other functional blocks after correlation to the Network Service instance affected. These notifications facilitate fault management operation on Network Services performed by OSS.			
Notes	(1) Guaranteeing delivery of notifications is considered as an implementation issue and deliberately left out of scope. (2) As the information carried in notifications is the most interesting for notification interfaces, the Notify operation is described, while details of the mechanism for registering for notifications (e.g. subscribe) and all other possible operations related to the actual notification delivery mechanism are deliberately left out of scope. (3) NFVO can forward fault information received from other functional blocks, mapped to VNF, NS or some combination of those. (4) The event notification type and format is not in-scope for the present document. (5) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties.			
Produced By	NFVO.			
Consumed By	OSS.			
Applicable Reference Point(s)	Os-Ma-nfvo.			

7.1.5.2 Operations

Operations	Description	Notes
Notify	This operation allows providing fault notifications on Network Services.	
Get NS fault information	This operation allows collecting Network Service fault information.	

7.2 Interfaces concerning Virtualised Network Functions

7.2.1 VNF Package management

7.2.1.1 Description

Interface Name	VNF Package Management.			
Description	This interface allows an authorized consumer functional block to manage the VNF Package, including the VNFD and the software image(s).			
Notes	While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties.			
Produced By	NFVO.	NFVO.		
Consumed By	OSS.	VNFM.		
Applicable Reference Point(s)	Os-Ma-nfvo.	Or-Vnfm.		

7.2.1.2 Operations

Operations	Description	Notes
On-board VNF Package	This operation allows submitting and validating the VNF Package.	Upon successful completion, the VNF Package is stored in the VNF catalogue, and can be used for VNF lifecycle management.
Disable VNF Package	This operation allows disabling the VNF Package, so that it is not possible to instantiate any further.	Upon successful completion, that VNF Package is disabled in the VNF catalogue. This operation has no effect on the VNF instances previously created, using the VNF Package.
Enable VNF Package	This operation allows enabling the VNF Package.	Upon successful completion, that VNF Package is enabled in the VNF catalogue. This operation has no effect on the VNF instances previously created, using the VNF Package.
Update VNF Package	This operation allows updating the VNF Package.	Upon successful completion the VNF Package is updated in the VNF catalogue (it can be a new VNF Package, depending on a case-by-case). This operation has no effect on VNF instances previously created, using the original VNF Package.
Query VNF Packages	This operation is used to query information on VNF Packages.	An example of information retrieved regarding VNF Package is VNFD.
Delete VNF Package	This operation is used to remove a disabled VNF Package.	

7.2.2 VNF software image management

7.2.2.1 Description

Interface Name	VNF Software Image Management.			
Description	This interface allows an authorized consumer functional block to manage the VNF software images in VIM.			
Notes	(1) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties. (2) The interface exposure assumes (but does not restrict) VNF software images are stored in repositories managed by the VIM(s) in order to minimize delays incurred on transferring and installing such software images as part of the VNF instance lifecycle. (3) The consumption of the interface over Os-Ma-nfvo reference point can only apply to a subset of the operations described in clause 7.2.2.2, in particular to "Copy Image" and "Query Image". The rest of operations can be performed by using the proper "VNF Package management" interface as described in clause 7.2.1.			
Produced By	VIM.	VIM.	NFVO.	
Consumed By	NFVO.	VNFM.	OSS.	
Applicable Reference Point(s)	Or-Vi.	Vnfm-Vi.	Os-Ma-nfvo.	

7.2.2.2 Operations

Operations	Description	Notes
Add Image	This operation allows adding new VNF software images to the image repository.	The image repository to be used for VNF instances is assumed to be in the VIM.
Delete Image	This operation allows deleting in the VNF software images from the image repository.	
Update Image	This operation allows updating the VNF software images in the image repository.	
Query Image	This operation allows querying the information of the VNF software images in the image repository.	For example, this would allow retrieving a selection of images previously provisioned, based on filtering criteria (e.g. all images available to a particular VIM).
Copy Image	This operation allows copying images from a VIM to another.	

7.2.3 VNF lifecycle operation granting

7.2.3.1 Description

Interface Name	VNF Lifecycle Operation Granting.			
Description	This interface allows an authorized consumer functional block to request a producer functional block permission and relevant information for performing certain VNF lifecycle operations (e.g. instantiation, scaling, etc.), so that it is ensured that such operations are consistent with operator's policies as to avoid raising conflicts in terms of resources usage or Network Service planning.			
Notes	(1) It is assumed that policies used in the granting of the permission are in effect at the producer of the interface (or a third party functional block from which the producer of the interface can check), in order to avoid potential conflicts with regards to managing Network Service lifecycle, other VNF lifecycle operation permission requests, and resources. (2) The permission procedure will result in granting or denying the execution of the lifecycle operation. In case of granting the permission, the permission can include as well specific information needed to execute the operation (e.g. Infrastructure Domain where instantiation is performed). Therefore, the procedure can have an impact on the sub-sequent interaction of the consumer functional block with other functional blocks, and in particular, with entities responsible for resource management (e.g. the permission obtained via this interface could be expected to be presented in sub-sequent interactions to other producer functional bocks). (3) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties.			
Produced By	NFVO.			
Consumed By	VNFM.			
Applicable Reference Point(s)	Or-Vnfm.			

7.2.3.2 Operations

Operations	Description	Notes
Grant Lifecycle Operation	This operation allows requesting the permission to perform a certain VNF lifecycle operation on a new or existing VNF. The sub-type of lifecycle operation is parameterized in the operation.	The grant lifecycle operation request contains information representative of necessary components to execute the requested VNF lifecycle operation. In particular, the request contains sufficient information for the producer functional block to make an impact assessment and grant or not grant permission in accordance with available policies. For instance, in some cases the granting of a VNF scale-out operation could require knowing how many more virtualised resources are needed, as well as some form of reservation (e.g. internal to NFVO or to VIM) could be required. The response to the grant request can return information about such reservation to the consumer functional block. In other cases it could only require knowing the type of lifecycle operation. In addition, the granted operations provides enough information back to the consumer functional block in order to further execute the lifecycle operation, e.g. Infrastructure Domain endpoint.

7.2.4 VNF lifecycle management

7.2.4.1 Description

Interface Name	VNF Lifecycle Management.			
Description	This interface allows an authorized consumer functional block to perform lifecycle operations on VNF(s), i.e. all operations needed to request and manage associations of NFVI Resources to a VNF, and maintain such associations in conformance with the VNF Descriptor and authorized run-time changes, throughout the lifecycle of the VNF.			
Notes	(1) The specific operations authorized by this interfaces need to be continued by the operations facilitated by the "Virtualised Resources Management" interface. (2) The NFVO produced interface can be extended or derived from the one produced by VNFM. (3) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties.			
Produced By	VNFM.	NFVO.		
Consumed By	NFVO.	OSS.		
Applicable Reference Point(s)	Or-Vnfm.	Os-Ma-nfvo.		

7.2.4.2 Operations

Operations	**Description**	**Notes**
Instantiate VNF	This operation allows creating a VNF instance.	
Query VNF	This operation allows retrieving VNF instance state and attributes.	Attributes returned can include for example number and location of VMs allocated to the VNF instance.
Scale VNF	This operation allows scaling (out/in, up/down) a VNF instance.	
Check VNF instantiation feasibility	This operation allows verifying if the VNF instantiation is possible.	No VNF instance is created as a result of the operation.
Heal VNF	This operation is used to request appropriate correction actions in reaction to a failure.	This assumes operational behaviour for healing actions by VNFM has been described in the VNFD. An example could be switching between active and standby mode.
Update VNF software	This operation allows applying a minor/limited software update (e.g. patch) to a VNF instance.	The software update implies no structural changes (e.g. in configuration, topology, behaviour, redundancy model). The goal is to not require termination/re-instantiation, or at least not for the entire VNF. This operation is only required if the software update procedure requires a change in the VNF infrastructure e.g. changing an image. Software updates that can be performed using solely pre-existing functional blocks (e.g. EM) could not invoke this operation.
Modify VNF	This operation allows making structural changes (e.g. configuration, topology, behaviour, redundancy model) to a VNF instance.	This could be expected to be designed as a transaction, rather than an atomic operation. Depending on the "modification plan", it could involve multiple catalogue Management and VNF lifecycle management atomic operations, and could involve multiple VNFM managers. It is possible that this transaction cannot be exposed as an operation by the VNFM, and only offered as an extension when exposed by the NFVO.

Operations	Description	Notes
Upgrade VNF software	This operation allows deploying a new software release to a VNF instance.	This could be expected to be designed as a transaction, rather than an atomic operation. Depending on the "software upgrade plan", it could involve multiple catalogue Management and VNF lifecycle management atomic operations, and could involve multiple VNFM managers. It is possible that this transaction cannot be exposed as an operation by the VNFM, and only offered as an extension when exposed by the NFVO. This operation is only required if the software upgrade procedure requires a change in the VNF infrastructure e.g. changing an image. Software upgrades that can be performed using solely pre-existing functional blocks (e.g. EM) could not invoke this operation.
Terminate VNF	This operation allows terminating gracefully or forcefully a previously created VNF instance.	

7.2.5 VNF lifecycle change notification

7.2.5.1 Description

Interface Name	VNF Lifecycle Change Notification			
Description	This interface is used to provide runtime notifications related to the state of the VNF instance, as a result of changes made to VNF instance including (not limited to) changes in the number of VDUs, changing VNF configuration and/or topology due to auto-scaling/update/upgrade/termination, switching to/from standby etc.			
Notes	(1) These notifications facilitate updating consuming functional blocks regarding completion of operations that could have been triggered earlier (e.g. for keeping the NFV Instances catalogue updated and/or the EM updated). (2) This interface is complementary to the VNF Lifecycle Management interface; when the VNF Lifecycle Management interface is produced by the VNFM, then VNFM also produces the VNF Lifecycle Changes Notification interface; when the VNF Lifecycle Management interface is produced by the NFVO, then the NFVO also produces the VNF Lifecycle Changes Notification interface. (3) When VNFM is the producer, and NFVO the consumer, there is no expectation that the NFVO automatically forwards such notifications to the OSS. However, NFVO can process such notifications to assess the impact on a Network Service instance, and issue at its turn notifications to the OSS via the Network Service Lifecycle Changes Notification interface. (4) Guaranteeing delivery of notifications is considered as an implementation issue and deliberately left out of scope. (5) As the information carried in notifications is the most interesting for notification interfaces, the Notify operation is described, while details of the mechanism for registering for notifications (e.g. subscribe) and all other possible operations related to the actual notification delivery mechanism are deliberately left out of scope. (6) The event notification type and format is not in-scope for the present document. (7) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties.			
Produced By	VNFM.	NFVO.	VNFM.	
Consumed By	NFVO.	OSS.	EM.	
Applicable Reference Point(s)	Or-Vnfm.	Os-Ma-nfvo.	Ve-Vnfm-em.	

7.2.5.2 Operations

Operations	Description	Notes
Notify	This operation allows providing notifications on state changes of a VNF instance, related to the VNF Lifecycle.	

7.2.6 VNF configuration

7.2.6.1 Description

Interface Name	VNF Configuration.		
Description	This interface allows configuring a VNF after successful instantiation.		
Notes	(1) This interface is exposed by every VNF. (2) The operations of this interface work on "configuration objects", which are structured data entities that represent a collection of configuration parameters. (3) If consumed by the VNFM, this interface can be used to pass configuration parameters to the VNF without being aware of the semantics of these configuration parameters. (4) Guaranteeing delivery of notifications is considered as an implementation issue and deliberately left out of scope. (5) As the information carried in notifications is the most interesting for notification interfaces, the Notify operation is described, while details of the mechanism for registering for notifications (e.g. subscribe) and all other possible operations related to the actual notification delivery mechanism are deliberately left out of scope. (6) The event notification type and format is not in-scope for the present document. (7) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties.		
Produced By	VNF	VNF	
Consumed By	VNFM	EM	
Applicable Reference Point(s)	Ve-Vnfm-vnf	Out of scope	

7.2.6.2 Operations

Operations	Description	Notes
Get Config Object	Retrieve configuration data for a specific configuration object	
Create Config Object	Create a configuration object	
Delete Config Object	Delete a configuration object	
Set Config Object Attributes	Modify a configuration object	
Notify Config Change	This operation allows providing notifications about configuration changes for a configuration object	

7.2.7 VNF performance management

7.2.7.1 Description

Interface Name	VNF Performance Management.		
Description	This interface is used to provide VNF performance management (measurement results collection and notifications) related to the behaviour of the VNF application-layer.		
Notes	(1) VNF application-layer refers to "VNF as an application". (2) VNF application-layer performance-related information can enable triggering run-time operations like auto-scaling based on (not limited to) exceeding set thresholds for calls-per-second, number-of-subscribers. (3) The VNFM produced performance-related information can just be the replication of VNF produced performance-related information for forwarding the same information further (e.g. to understand implications on Network Services). No reference point exists between VNF and NFVO; hence any communication between VNF and NFVO flows through the VNFM. (4) Guaranteeing delivery of notifications is considered as an implementation issue and deliberately left out of scope. (5) As the information carried in notifications is the most interesting for notification interfaces, the Notify operation is described, while details of the mechanism for registering for notifications (e.g. subscribe) and all other possible operations related to the actual notification delivery mechanism are deliberately left out of scope. (6) The event notification type and format is not in-scope for the present document (e.g. events indicating a need for scaling). (7) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties.		
Produced By	VNF.	VNFM.	VNF.
Consumed By	VNFM.	NFVO.	EM.
Applicable Reference Point(s)	Ve-Vnfm-vnf.	Or-Vnfm.	Out of scope.

7.2.7.2 Operations

Operations	Description	Notes
Get performance measurement results	This operation allows collecting performance measurement results generated on resources.	
Notify	This operation allows providing notifications with application-layer performance measurement results.	

7.2.8 VNF fault management

7.2.8.1 Description

Interface Name	VNF Fault Management.		
Description	This interface allows providing VNF application-layer fault information (e.g. network function configuration failures, communication failures between software modules).		
Notes	(1) VNF application-layer refers to "VNF as an application". (2) VNF application-layer fault information can facilitate fault management operations performed by other functional blocks (e.g. fault correlation, root-cause analysis) and/or can be used to trigger VNF instance healing operations. (3) The VNFM produced fault information can just be the replication of VNF produced fault information for forwarding the same information further. No reference point exists between VNF and NFVO; hence any communication between VNF and NFVO flows through the VNFM. (4) This interface can be used for VNF health-check by an external functional block (e.g. VNFM); the mechanisms to enable, trigger and/or use this interface for VNF health-check are not covered in the present document. (5) Guaranteeing delivery of notifications is considered as an implementation issue and deliberately left out of scope. (6) As the information carried in notifications is the most interesting for notification interfaces, the Notify operation is described, while details of the mechanism for registering for notifications (e.g. subscribe) and all other possible operations related to the actual notification delivery mechanism are deliberately left out of scope. (7) The event notification type and format is not in-scope for the present document. (8) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties.		
Produced By	VNF.	VNFM.	VNF.
Consumed By	VNFM.	NFVO.	EM.
Applicable Reference Point(s)	Ve-Vnfm-vnf.	Or-Vnfm.	Out of scope.

7.2.8.2 Operations

Operations	Description	Notes
Get VNF fault information	This operation allows collecting VNF application-layer fault information.	
Notify	This operation allows providing application-layer fault notifications.	

7.3 Interfaces concerning virtualised resources

7.3.1 Virtualised resources catalogue management

7.3.1.1 Description

Interface Name	Virtualised Resources Catalogue Management.			
Description	This interface allows an authorized consumer functional block to query the catalogues of virtualised resources and get notifications about their changes.			
Notes	(1) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties. (2) Guaranteeing delivery of notifications is considered as an implementation issue and deliberately left out of scope. (3) As the information carried in notifications is the most interesting for notification interfaces, the Notify operation is described, while details of the mechanism for registering for notifications (e.g. subscribe) and all other possible operations related to the actual notification delivery mechanism are deliberately left out of scope. (4) The event notification type and format is not in-scope for the present document.			
Produced By	VIM.	VIM.		
Consumed By	NFVO.	VNFM.		
Applicable Reference Point(s)	Or-Vi.	Vi-Vnfm.		

7.3.1.2 Operations

Operations	Description	Notes
Query Resource Catalogue	This operation allows retrieving the list of catalogued virtualised resources, and/or a specific catalogued resource on which the consumer is allowed to perform subsequent operations.	
Notify Resources Catalogue Changes	This operation provides change notifications on virtualised resources catalogues managed by the producer functional block.	

7.3.2 Virtualised resources capacity management

7.3.2.1 Description

Interface Name	Virtualised Resources Capacity Management.			
Description	This interface allows an authorized consumer functional block to perform operations related to NFVI-PoP capacity and usage reporting (see note (1)). The interface allows retrieving information about: • NFVI-PoP total resources capacity over which virtualised resources are provisioned. • Virtualised resources capacity and density, e.g. how many virtualised resources can be created from existing NFVI-PoP resources. • Statistics and mapping of NFVI-PoP resources usage to virtualised resources usage (see note (2)), global at NFVI-PoP level, and per deployed virtualised partition (see note (3)). The interface enables the capture of information for reporting NFVI-PoP total resources usage and executing analytics for capacity planning, capacity changes, and consequently for Network Service planning, etc.			
Notes	(1) The management of the NFVI-PoP capacity (e.g. adding, removing physical hosts, adding network switches, etc.), the topology, and its configuration, as well as the management of specific physical equipment are out of the scope of this interface. (2) Virtualised resources usage is facilitated by the "Virtualised Resources Management" interface. (3) A virtualised partition refers to a group of allocated virtualised resources that the producer of the interface can correlate (e.g. as managed under the same deployment project). (4) Guaranteeing delivery of notifications is considered as an implementation issue and deliberately left out of scope. (5) As the information carried in notifications is the most interesting for notification interfaces, the Notify operation is described, while details of the mechanism for registering for notifications (e.g. subscribe) and all other possible operations related to the actual notification delivery mechanism are deliberately left out of scope. (6) The event notification type and format is not in-scope for the present document. (7) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties.			
Produced By	VIM.			
Consumed By	NFVO.			
Applicable Reference Point(s)	Or-Vi.			

7.3.2.2 Operations

Operations	Description	Notes
Query Capacity	This operation allows querying the capacity usage of an NFVI-PoP. The operation can be used to gather information at different levels, from specific virtualised partition capacity usage, to total capacity availability in the NFVI-PoP.	
Notify Capacity Changes	This operation allows notifying about capacity changes in the NFVI-PoP.	

7.3.3 Virtualised resources management

7.3.3.1 Description

In the following, the types of virtualised resources that can be consumed from an NFVI-PoP comprise computing, storage and networking; they have been described in clause 4.2.

Interface Name	Virtualised Resources Management.			
Description	This interface allows an authorized consumer functional block to perform operations on virtualised resources available to the consumer functional block. The interface includes common operations for creating, querying, updating and terminating compute, storage and network isolated virtualised resources, or a composition of different types in a resource grouping (see note (1)), as well as managing virtualised resource reservations.			
Notes	(1) A virtualised resource grouping consists of one or more computing, storage and network virtualised resources associated with each other and that can be handled as an atomic object. (2) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties.			
Produced By	VIM.	VIM.	NFVO.	
Consumed By	NFVO.	VNFM.	VNFM.	
Applicable Reference Point(s)	Or-Vi.	Vi-Vnfm.	Or-Vnfm.	

7.3.3.2 Operations

Operations	Description	Notes
Allocate Resource	This operation allows requesting the instantiation and assignment of a virtualised resource to the VNF, as indicated by the consumer functional block.	
Query Resource	This operation allows querying a virtualised resource, i.e. retrieve information about an instantiated virtualised resource.	
Update Resource	This operation allows updating the configuration and/or parameterization of an instantiated virtualised resource.	Virtualised resources can have associated elements like templates expressing the set of configurations that can be used when creating, reading, updating, and deleting resources. At the discretion of the interface producer and Service Provider's policy, a given authorized consumer functional block could be allowed the creation of new templates and/or configurations.
Scale Resource	This operation allows scaling a virtualised resource by adding or removing capacity, e.g. adding vCPUs to a virtual machine.	
Migrate Resource	This operation allows moving virtualised resources between locations. For instance, the operation performs the migration of a computing resource from one host to another host; while for a storage resource, it migrates the resource from one storage location to another.	The migration operation relies and executes its commands based on the abstraction of hardware resources performed by the producer of the interface (or the implementer of such abstractions).
Operate Resource	This operation allows executing specific commands on certain allocated virtualised resources. Examples on compute resources can include (but not limited to): start, stop, pause, suspend, capture snapshot, etc.	
Release Resource	This operation allows de-allocating and terminating an instantiated virtualised resource. This operation frees resources and returns them to the NFVI resource pool.	
Create Resource Reservation	This operation allows requesting the reservation of a set of virtualised resources to a consumer functional block without performing the steps of "Allocate Resource".	
Query Resource Reservation	This operation allows querying an issued resources reservation, e.g. to discover the virtualised resources included in a specific reserved resources pool, or the amount of free resources in such a pool.	

Operations	Description	Notes
Update Resource Reservation	This operation allows updating an issued resources reservation to increase or decrease the amount of virtualised resources in the reserved resources pool.	Depending on Service Provider policy, deployments could be expected to prevent the update of virtualised resource reservations when resources in the reserved set are still allocated.
Release Resource Reservation	This operation allows releasing an issued resources reservation, hence freeing the reserved virtualised resources.	Depending on Service Provider policy, deployments could be expected to prevent the release of virtualised resource reservations when resources in the reserved set are still allocated.

7.3.4 Virtualised resources performance management

7.3.4.1 Description

Interface Name	Virtualised Resources Performance Management			
Description	This interface allows providing performance management (measurement results collection and notifications) related to virtualised resources (see note (1)) including (not limited to) resource consumption level, e.g. vCPU power consumption, VM memory usage oversubscription, VM disk latency, etc.			
Notes	(1) Only types of resources that have been catalogued and offered through right abstractions to consumer functional blocks are in scope (refer to clause 7.3.3 "Virtualised Resources Management"). (2) The VNFM produced notifications can be a replication of VIM produced notifications for forwarding the same information further. (3) Guaranteeing delivery of notifications is considered as an implementation issue and deliberately left out of scope. (4) As the information carried in notifications is the most interesting for notification interfaces, the Notify operation is described, while details of the mechanism for registering for notifications (e.g. subscribe) and all other possible operations related to the actual notification delivery mechanism are deliberately left out of scope. (5) The event notification type and format is not in-scope for the present document. (6) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties.			
Produced By	VIM.	VIM.	VNFM.	
Consumed By	NFVO.	VNFM.	EM.	
Applicable Reference Point(s)	Or-Vi.	Vnfm-Vi.	Ve-Vnfm-em.	

7.3.4.2 Operations

Operations	Description	Notes
Get performance measurement results	This operation allows collecting performance measurement results generated on virtualised resources.	
Notify	This operation allows providing notifications with performance measurement results on virtualised resources.	

7.3.5 Virtualised resources fault management

7.3.5.1 Description

Interface Name	Virtualised Resources Fault Management		
Description	This interface allows providing fault information related to the resources (see note (1)) visible to the consumer functional block, including virtual containers (VMs) crashes, virtual network ports errors, virtual container's to storage disconnection, etc. The interface also provides information about faults related to the pools of resources, for instance, reserved resources unavailable, resource exhaustion, etc.).		
Notes	(1) Only types of resources that have been catalogued and offered through right abstractions to consumer functional blocks are in scope (refer to clause 7.3.3 "Virtualised Resources Management"). (2) The VNFM produced notifications can be a replication of VIM produced notifications for forwarding the same information further. (3) Guaranteeing delivery of notifications is considered as an implementation issue and deliberately left out of scope. (4) As the information carried in notifications is the most interesting for notification interfaces, the Notify operation is described, while details of the mechanism for registering for notifications (e.g. subscribe) and all other possible operations related to the actual notification delivery mechanism are deliberately left out of scope. (5) The event notification type and format is not in-scope for the present document. (6) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties.		
Produced By	VIM.	VIM.	VNFM.
Consumed By	NFVO.	VNFM.	EM.
Applicable Reference Point(s)	Or-Vi.	Vnfm-Vi.	Ve-Vnfm-em.

7.3.5.2 Operations

Operations	Description	Notes
Get resource fault information	This operation allows collecting virtualised resource fault information.	
Notify	This operation allows providing fault notifications on virtualised resources.	

7.4 Policy administration interface

7.4.1 Description

Interface Name	Policy Management			
Description	This interface allows an authorized consumer functional block to manage NFV policies.			
Notes	(1) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties. (2) This interface is exposed by VNFM and VIM to provide a way to manage policies related with VNF and NFVI Resources respectively on request from NFVO.			
Produced By	NFVO.	VNFM.	VIM.	
Consumed By	OSS.	NFVO.	NFVO.	
Applicable Reference Point(s)	Os-Ma-nfvo.	Or-Vnfm.	Or-Vi.	

7.4.2 Operations

Operations	Description	Notes
Create policy	This operation allows defining policy rules include conditions and actions	
Update policy	This operation allows updating an existing policy	This involves modifying policy metadata including conditions, actions
Delete policy	This operation allows delete policy after being created	
Query policy	This operation allows querying about a particular policy or a querying the list of available policies	
Activate policy	This operation enables activating an available policy	
De-activate policy	This operation enables de- activating an active policy	

7.5 Network Forwarding Path management interface

7.5.1 Description

Interface Name	Network Forwarding Path Management.			
Description	This interface allows an authorized consumer functional block to perform Network Forwarding Path management and notification operations. This interface provides the facility to have policy based linkages on a VNF Forwarding Graph as expressed by a Network Forwarding Path.			
Notes	(1) The Network Forwarding Path containing an ordered list of Connection Points along with rules/policies associated to the list. (2) Network Forwarding Path rule related information can enable Network Controllers to configure accordingly forwarding tables in NFVI network resources. (3) As the information carried in notifications is the most interesting for notification interfaces, the Notify operation is described, while details of the mechanism for registering for notifications (e.g. subscribe) and all other possible operations related to the actual notification delivery mechanism are deliberately left out of scope. (4) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties. (5) The event notification type and format is not in-scope for the present document. (6) The Network Forwarding Path is limited to the resource controlled by VIM exposing the interface.			
Produced By	VIM.			
Consumed By	NFVO.			
Applicable Reference Point(s)	Or-Vi.			

7.5.2 Operations

Operations	Description	Notes
Create Network Forwarding Path	This operation allows creating a Network Forwarding Path.	
Update Network Forwarding Path	This operation allows updating the information associated with a Network Forwarding Path.	
Delete Network Forwarding Path	This operation allows deleting a Network Forwarding Path.	
Query Network Forwarding Path	This operation allows querying information about a specified Network Forwarding Path instance.	
Notify	This operation allows providing information about a Network Forwarding Path rule.	

7.6 NFVI hypervisor management interface

7.6.1 Description

The hypervisor exposes a wide array of functionalities, grouped by the following (non-exhaustive) categories of information:

- Host and VM Configuration/Life cycle.
- Resources and VM inventory management.
- Networking/connectivity.
- CPU, Pools, Clusters management and metrics.
- Memory and storage management and metrics.
- Utilities for task and distributed scheduling.
- Alarms and Events management.
- Logging for SLA, debugging.

Interface Name	NFVI Hypervisor Management Interface.			
Description	This interface allows an authorized consumer functional block to request a producer functional block to perform operations on hypervisor-accessed resources (e.g. compute, storage and networking) in the NFVI.			
Notes	(1) This interface is exposed by NFV participating Hypervisors. (2) This interface maps to the Nf-Vi/H interface described in ETSI GS NFV-INF 001 [i.4] and ETSI GS NFV-INF 004 [i.6]. (3) Guaranteeing delivery of notifications is considered as an implementation issue and deliberately left out of scope. (4) As the information carried in notifications is the most interesting for notification interfaces, the Notify operation is described, while details of the mechanism for registering for notifications (e.g. subscribe) and all other possible operations related to the actual notification delivery mechanism are deliberately left out of scope. (5) The event notification type and format is not in-scope for the present document. (6) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties.			
Produced By	NFVI.			
Consumed By	VIM.			
Applicable Reference Point(s)	Nf-Vi.			

7.6.2 Operations

Operations	Description	Notes
Create virtual machine	This operation is needed to create and start a virtual machine	This and all subsequent virtual machine related operations fall under the "Host & VM configuration//Lifecycle" and "Resources and VM inventory management" hypervisor information categories. An example is the "create" command in virsh (KVM); rules apply for different VM goals such as affinity requirements.
Shutdown virtual machine	This operation is needed to shut down a virtual machine	An example is the "shutdown" command in virsh (KVM).
Destroy virtual machine	This operation is needed to destroy a virtual machine	This does an immediate ungraceful shutdown of the virtual machine. An example is the "destroy" command in virsh (KVM).

Operations	Description	Notes
Update virtual machine	This operation is needed to update a created virtual machine	This can be used to update memory or vCPU of the virtual machine or attach, update or detach a disk or network interface. Examples include the following commands in virsh (KVM): • update-device, setmem, setvcpus; • attach-device, attach-disk, attach-interface; • detach-device, detach -disk, detach -interface.
List the virtual machines	This operation is needed to list the virtual machines	An example is the "list" command in virsh (KVM).
Query a virtual machine by name id	This operation is needed to show the details of a given virtual machine	Examples are the "dumpxml, dominfo" commands in virsh (KVM).
Reboot a virtual machine	This operation reboots a virtual machine	An example is the "reboot" command in virsh (KVM).
Suspend a virtual machine	This operation suspends a virtual machine	The suspended VM will not consume the processor resources but the memory will be maintained. An example is the "suspend" command in virsh (KVM).
Resume a virtual machine	This operation resumes a virtual machine, which has been previously suspended	An example is the "resume" command in virsh (KVM).
Save a virtual machine	This operation stops the virtual machine and save the data to a file.	An example is the "save" command in virsh (KVM).
Restore a virtual machine	Recreate the virtual machine from a file created by the save operation.	An example is the "restore" command in virsh (KVM).
Create Storage Pool	This operation creates and starts a storage pool	This, and all subsequent storage pool and virtual machine storage related operations fall under the "Memory and storage management and metrics" hypervisor information category. Examples include pool-build, pool-create command in virsh (KVM).
Modify Storage Pool	This operation modifies a specified storage pool	An example is pool-edit command in virsh (KVM).
List Storage pools	This operations lists the storage pools known to the hypervisor	An example is pool-list command in virsh (KVM).
Delete Storage Pool	This operation deletes a specified storage pool	Examples include pool-delete or pool-destroy command in virsh (KVM).
Query Storage Pool	This operation provides the information about a storage pool	An example is pool-info command in virsh (KVM).
Create Virtual Machine storage	This operation creates a storage for the virtual machine on a given pool	Examples include vol-create, vol-create-as, vol-create-from commands in virsh (KVM).
Delete Virtual Machine Storage	This operation deletes the specified virtual machine storage	An example is vol-delete command in virsh (KVM).
List Virtual machine storage	This operations lists the virtual machine storage on a storage pool	An example is vol-list command in virsh (KVM).
Query Virtual Machine Storage	This operation provides the information about the specified virtual machine storage	Examples include vol-info, vol-dumpxml, vol-pool command in virsh (KVM).
Create Snapshot	This operation creates a snapshot for a virtual machine	Snapshots take the disk, memory, and device state of a virtual machine at a specified point-in-time, and save it for future use. An example is the "snapshot create" command in virsh (KVM).
Create hypervisor policies	This operation creates a hypervisor policy (a policy regarding hypervisor-accessed resources, e.g. compute, storage, networking) for a NFV tenant	This and all subsequent hypervisor policy operations fall under the "Resources and VM inventory management" hypervisor information category.
Update hypervisor policies	This operation updates a hypervisor policy for a NFV tenant	
List hypervisor policies	This operation lists all the hypervisor policies created for a given NFV tenant	
Query hypervisor policy	This operation shows the details of a given specific hypervisor policy relative to different resources and VMs in the database	

Operations	Description	Notes
Delete hypervisor policy	This operation deletes the hypervisor policy relative to specific resources	
Change VM configuration	These operations adds/removes resources to/from the VM	This falls under the "Utilities" hypervisor information category, covering tasks relative to the VM scaling, porting, or consolidating workloads; rules for placement can apply, such as affinity; power management.
Migrate virtual machine	This operation migrates the specified VM to another hypervisor	This falls under the "Utilities" hypervisor information category. Examples include migrate command in virsh (KVM).
Get hypervisor measurement results	This operation allows retrieving performance and reliability measurement results. These are multiple operations	This falls under the "Logging for SLA and debugging", "CPUs, clusters, pools" and "Memory and Storage" hypervisor information categories. Examples include metrics on: CPU processors, Memory, Fans, Temperature, Voltage, Power, Network, Battery, Storage, Cable/Interconnect, Software components, affinity Watchdog, PCI devices; VM stalls.
Create virtual network device	This operation creates a specified virtual network device	This and all subsequent operations referring to "virtual network device" fall under the "Network/connectivity" hypervisor information category. The intent here is to represent this as a broad category, rather than listing in detail each virtual network device type and the operations allowed on it.
Delete virtual network device	This operation deletes a specified virtual network device	
Update the virtual network device	This operation updates a virtual network device	
List the virtual network devices	This operation lists virtual network devices created by the requesting entity	
Query a virtual network device	This operation retrieves details of a specified virtual network device	
Notify	This operation allows providing notifications about hypervisor-accessed resources changes	This falls under the "Events and alarms management" hypervisor information category.

7.7 NFVI compute management interface

7.7.1 Description

Interface Name	NFVI Compute Management Interface.			
Description	This interface allows an authorized consumer functional block to request a producer functional block to perform management operations on physical compute and storage resources in the NFVI.			
Notes	(1) The description and operations of the interface are not complete in the present document. In the present document, it is assumed that NFV-MANO supports the use of this interface in order to provide notifications from NFVI to the VIM regarding changes in physical resources (e.g. fault information, inventory information), without defining the specific interface operations. (2) This interface maps to the Nf-Vi/C interface described in ETSI GS NFV-INF 001 [i.4] and ETSI GS NFV-INF 003 [i.5].			
Produced By	NFVI.			
Consumed By	VIM.			
Applicable Reference Point(s)	Nf-Vi.			

7.8 NFVI networking management interface

7.8.1 Description

Interface Name	NFVI Networking Management Interface.			
Description	This interface allows an authorized consumer functional block to request a producer functional block to perform management operations on networking resources in the NFVI.			
Notes	(1) While some operations in this interface can also be exposed via the NFVI Hypervisor Management Interface (handling the "Network/Connectivity" hypervisor information category), this interface focuses on functionality exposed by network (SDN) controllers comprised in NFVI, via appropriate abstractions. (2) This interface maps to the Nf-Vi/N interface described in ETSI GS NFV-INF 001 [i.4] and ETSI GS NFV-INF 005 [i.7]. (3) The operations represent an example of a high level abstraction of functionality that could be exposed, but with the understanding that in alternative implementations, several finer level abstractions could be exposed to collectively achieve each of the outlined operations. (4) Guaranteeing delivery of notifications is considered as an implementation issue and deliberately left out of scope. (5) As the information carried in notifications is the most interesting for notification interfaces, the Notify operation is described, while details of the mechanism for registering for notifications (e.g. subscribe) and all other possible operations related to the actual notification delivery mechanism are deliberately left out of scope. (6) The event notification type and format is not in-scope for the present document. (7) While not shown explicitly, interfaces can be consumed by authenticated and authorized other parties.			
Produced By	NFVI.			
Consumed By	VIM.			
Applicable Reference Point(s)	Nf-Vi.			

7.8.2 Operations

Operations	Description	Notes
Create virtual network	This operation is needed to create virtual networks(L2/L3 overlay or infrastructure) for inter VNF or inter VNFC interconnectivity	
Delete virtual network	This operation is needed to delete the virtual networks which have been created for inter VNF/inter VNFC connectivity	
Update the virtual network	This operation is needed to update the attributes of a created virtual network belonging to a tenant, e.g. updating the admin status of a virtual network	
List the virtual networks	This operation is needed to list the virtual networks belonging to a NFV tenant	
Query a virtual network	This operation is needed to show the details of a given virtual network	
Create a subnet	This operation creates a subnet for a given virtual network	
Update a subnet	This operation updates the information associated to a subnet	
List the subnets	This operation is needed to list the subnets belonging to a virtual network	
Query subnet	This operation is needed to show the details of a given subnet	
Delete a subnet	This operation deletes a subnet associated with a virtual network	Deletion fails if any of the IP addresses in the subnet are still allocated to any of the VNF VMs
Create port	This operation creates a port on a given virtual network	
Update port	This operation updates the information associated with a port associated with a virtual network	
List ports	This operation lists the ports associated with a virtual network	
Query port	This operation shows the details of a given port	
Delete port	This operation deletes a port from the virtual network	
Notify	This operation is needed to provide notifications regarding virtual networks, subnets, ports	

7.9 Interfaces exposed between different service providers

In order to support federation between different service providers' NFV Infrastructures and some of the NFV use cases (e.g. NFVIaaS, VNFaaS, VNPaaS) exchanges between service providers are supported via interfaces exposed by a service provider A and consumed by another service provider B.

There are multiple scenarios to consider, all technically possible (including various combinations). In all cases, authentication is necessary and authorization policies (e.g. role-based) controls the access of service provider B to functions exposed by service provider A:

1) Functional blocks deployed by service A allow access to any functional block belonging to service provider B:

   a) In this case, no new interface is needed to be specified; policies determine who has access to what and in what conditions.

2) A designated NFV-MANO functional block deployed by service provider A is identified as a single point of contact that exposes interfaces for interactions with functional blocks deployed by service provider B. This has 2 sub-cases:

   a) Interfaces that are already exposed by the designated functional block deployed by service provider A are sufficient; the subset of the interfaces to be exposed for consumption by functional blocks of service provider B, and the appropriate authorization policies are expected to be defined.

   b) New interfaces are needed to be defined for some reason; new authentication mechanism and authorization policies are expected to be defined.

3) Some service provider A's OSS exposes interfaces to other service provider B functional, in which case blocks; new interfaces could be needed to be exposed to satisfy the requirements; authentication mechanisms and authorization policies are expected to be defined.

Among the options presented, the recommendation is for service provider A to expose interfaces to external parties (e.g. service provider B) at a single point of contact in the NFV-MANO architecture, in order to minimize security risks and integration and operational complexity.

The recommendation is that the functional block designated as the single point of contact for exposing service provider A's interfaces is the NFV Orchestrator, for the following reasons:

- It is the designated NFV-MANO entity that handles lifecycle management of Network Services, and most interactions between different service providers are likely to be at the level of services produced by one service provider, and consumed by another service provider.
- It is already targeted to, and can re-expose, as needed, interfaces (extended or derived) that other NFV-MANO functional blocks (VIM, VNFM) produce.
- It requires less additional development/integration in order to fulfil this role, in comparison to other functional blocks in the NFV-MANO Architectural Framework, because it is the only functional block in the NFV-MANO Architectural Framework that either produces and/or consumes interfaces to/from all other functional blocks in the NFV-MANO Architectural Framework: OSS/BSS, VNFM, and VIM.

NOTE: Additional interfaces mentioned in 2) b) above can be assessed and described in a later stage.

Annex A:
VNF Instance management and orchestration case study

A.1 IMS MRF management and orchestration case study

A.1.0 Case study description

This case study assumes NFVO and NFV Infrastructure (NFVI) as main actors.

This clause is aimed at describing an IMS MRF orchestration and management use case.

A brief summary of the identified VNFs is provided in figure A.1.

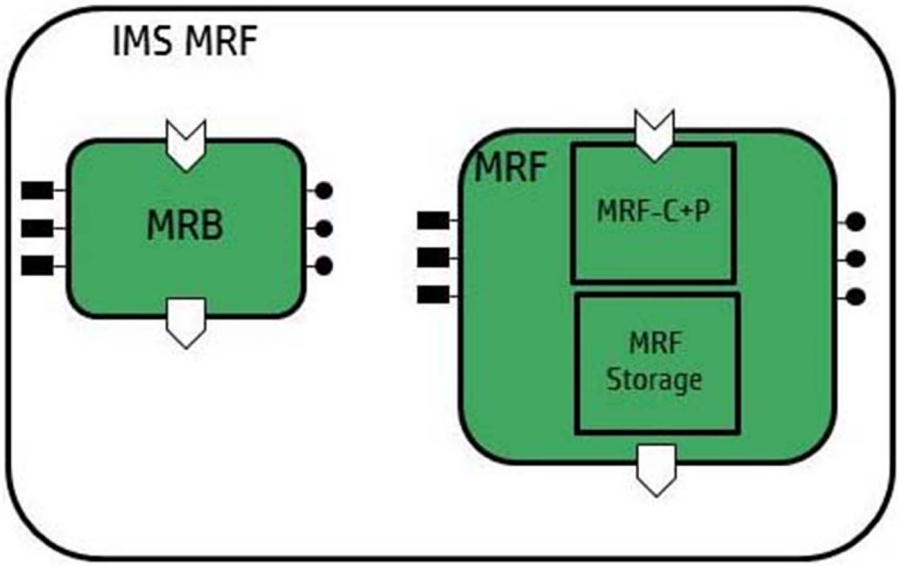


Figure A.1: IMS MRF VNF Forwarding Graph and VNFs

Two VNFs and one VNF Forwarding Graph have been identified for this use case:

- MRB VNF.
- MRF VNF with 2 VNF components :MRF-C+P and MRF storage.
- IMS MRF VNF Forwarding Graph made of 2 VNFs: MRB and MRF.

The details of the IMS MRF VNF Forwarding Graph and the VNFs that compose it are described in ETSI GS NFV-SWA 001 [i.8].

Note that as for the VNF mapping, depending on vendor choice, the deployment use case might be slightly different. The VNF deployment use case proposed here can be considered as a typical use case. It is not the goal of the present document to cover all possible IMS MRF deployment scenarios.

A possible target deployment of an IMS MRF VNF Forwarding Graph is shown in figure A.2.

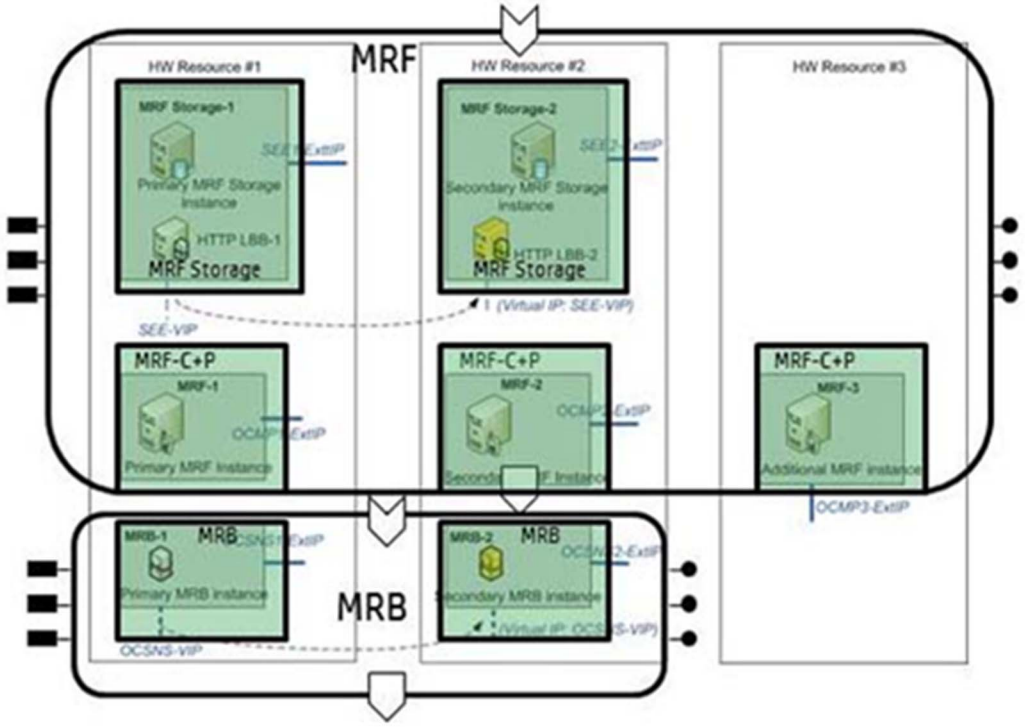


Figure A.2: IMS MRF target deployment

This deployment example is showing the IMS MRF VNF Forwarding Graph deployed over 3 HW resources and 7 VMs (grey boxes). It illustrates also the affinity and anti-affinity rules described in the previous clause.

HW resource #1 contains 3 VMs:

- VM-1 for an instantiation of MRB VNF (MRB-1), used as active.
- VM-2 for an instantiation of MRF-C+P (MRF-1).
- VM-3 for an instantiation of MRF Storage (MRF Storage-1), used as master.

HW resource #2 contains also 3 VMs:

- VM-4 for an instantiation of MRB (MRB-2), used as standby.
- VM-5 for an instantiation of MRF-C+P (MRF-2).
- VM-6 for an instantiation of MRF Storage (MRF Storage-2), used as slave.

HW resource #3 contains a single VM (VM-7) with an instantiation of MRF-C+P (MRF-3).

MRB, MRF-C+P and MRF-Storage have 3 different redundancy models that have direct implication for NFVO:

- Active/Standby for MRB.
- N+1 Active for MRF-C+P.
- Master/Slaves for MRF-Storage.

As per affinity rule, MRB, MRF-C+P and MRF Storage are co-located on the same hardware resources.

As per anti-affinity rules, MRB primary and secondary are located on different hardware resources. Same is true for MRF Storage primary and secondary.

Prerequisites are:

- The code for both VNFs (MRB, MRF) and the corresponding VNFDs have been developed and tested.

- The IMS MRF VNF Forwarding Graph and the corresponding VNF Forwarding Graph Descriptor (VNFFGD) has been developed and tested.
- VM images for the various components (MRB, MRF-C+P, MRF Storage) are available.
- This scenario assumes that any needed network setup required as prerequisite (e.g. VLAN with the appropriate QoS, holes in firewall opened, multicast/IGMP enabled) has been done. This is not a requirement and might be considered done dynamically as part of the deployment.

A.1.1 IMS MRF on-boarding

The overall process of on-boarding the IMS MRF VNF Forwarding Graph is shown on the figure A.3.

The overall sequencing is the following:

1) NFVO receives a request to on-board the MRB VNF with the MRB VNFD attached.
2) NFVO validates the content of the MRB VNFD and if valid, stores it in its VNF catalogue.
3) NFVO receives a request to on-board the MRF VNF with the MRF VNFD attached.
4) NFVO validates the content of the MRF VNFD and if valid, stores it in its VNF catalogue.
5) NFVO receives a request to on-board the IMS MRF VNF Forwarding Graph with the IMS MRF VNFFGD attached.
6) NFVO validates the content of the IMS MRF VNFFGD. This would include checking that the MRB and MRF VNFD are present in the VNF catalogue, as they are referenced by the IMS MRF VNFFGD. If the IMS MRF VNFFGD is valid, NFVO stores it in its NS catalogue.
7) If any error, return error to caller.

The 2 VNFDs (MRB, MRF) and the IMS MRF VNF Forwarding Graph are now on-boarded and available to the NFVO.

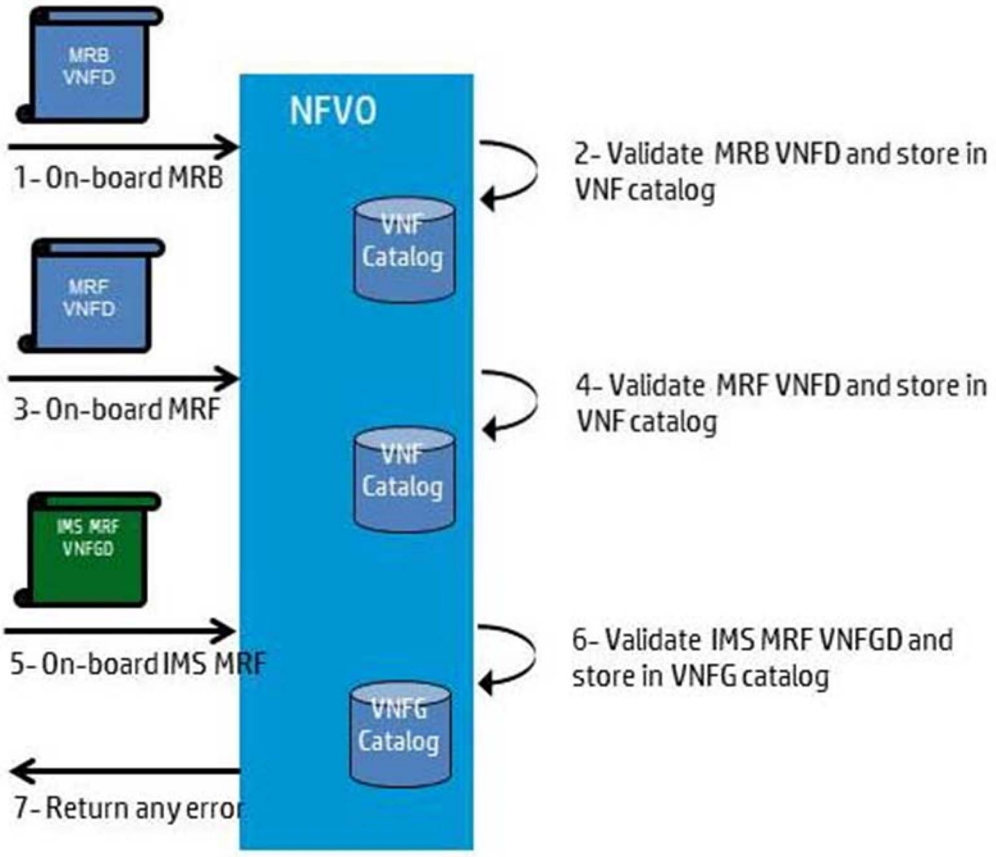


Figure A.3: IMS MRF VNF Forwarding Graph on-boarding

A.1.2 IMS MRF instance provisioning and configuration

Figure A.4 summarizes the overall deployment use case.

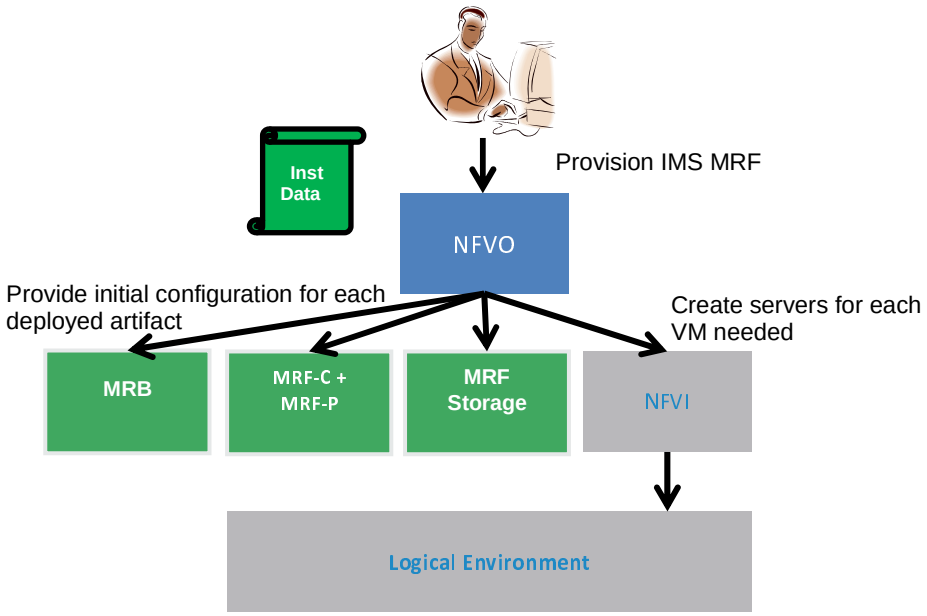


Figure A.4: IMS MRF VNF Forwarding Graph deployment use case

The assumption is that NFVO, when receiving the command to provision an IMS MRF VNF Forwarding Graph, receives at the same time the instantiation definition data, containing all the instantiation information needed:

- The number of MRB to provision (2 in this use case).
- The number of MRF-C+P pairs to deploy (3 in this use case).
- The number of MRF storage to deploy (2 in this case).
- IP address for S-CSCF external component.
- IP address for app server external component.

NOTE 1: IP addresses for the various instances can be provided in the instantiation definition information or can be constructed dynamically. The latter is assumed in this use case.

NOTE 2: Instantiation definition data can be provided in various possible forms: file, parameters of the request; the exact format would depend on the selected request format.

It is also assumed that the 2 VNFs for MRB, MRF and the IMS MRF VNF Forwarding Graph have been on-boarded as shown in clause A.1.1 IMS MRF instance on-boarding and are present in the VNF and NS catalogues of the NFVO.

The overall sequencing of this typical deployment is as follow:

1) NFVO receives the 'provision IMS MRF' request with the instantiation definition data defined above.
2) NFVO checks that the IMS MRF VNF Forwarding Graph is on-boarded and that the IMS MRF VNFFGD is present in the VNFFG catalogue as well as the MRB and MRF VNFD in the VNF catalogue.
3) NFVO checks the validity of the provided instantiation data against the catalogued VNFDs for MRB and MRF and the IMS MRF VNFFGD.
4) NFVO applies the affinity and anti-affinity rules for each VNF and VDU to determine location of instances:
    a) The 2 MRB instances, MRB-1 and MRB-2 are expected to be on separate HW resources to guarantee availability.
    b) The 3 instances of MRF-C+P are expected to be on separate HW resources to guarantee availability.
    c) The 2 MRF storage instances are expected to be on separate HW resources to guarantee availability.
    d) Each MRF storage instance are preferably co-located with an MRF instance as per affinity rule to get better performance.
    e) Result of the validation is the need for 7 VMs on 3 different HW resources as described earlier. The characteristics needed from each VM are provided by the information in the VNFD.
5) NFVO validates the appropriate resources identified in the previous step are available for fulfilling this request.
6) If any error, return error to caller.

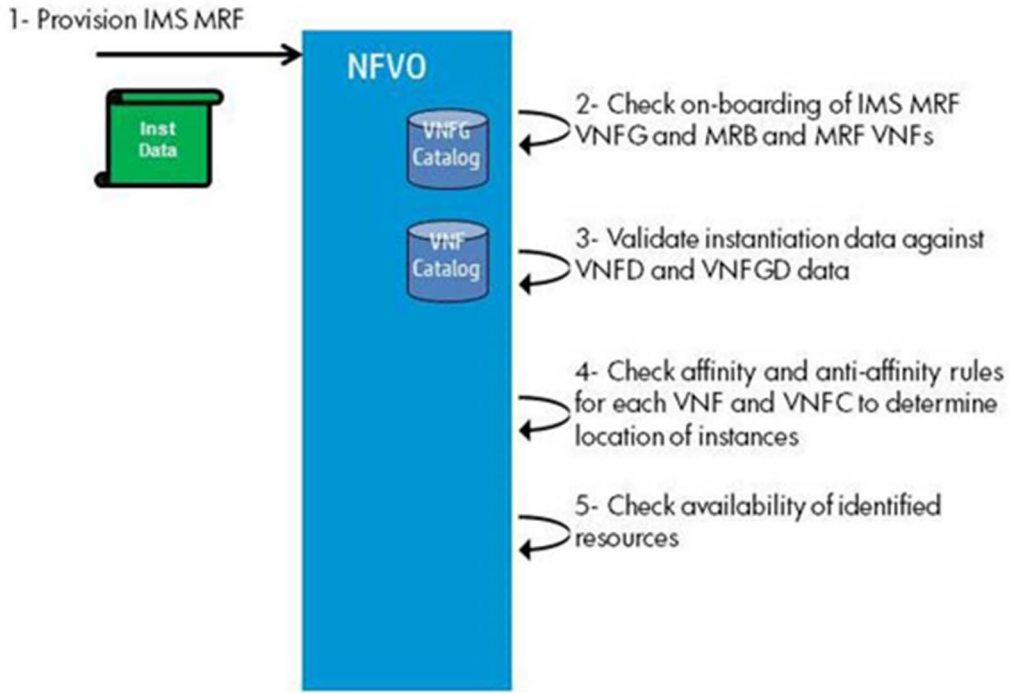


Figure A.5: IMS MRF deployment - initial steps

7) NFVO sends a create server command to the NFV infrastructure (for instance a cloud management system (CMS) to create VM-1, hosting instance MRB-1 with information coming from MRB VNF.

NOTE 3: The Cloud Management System might not always be present and if not, NFVO would directly create the VM as shown in the next step.

8) The NFVI in turn sends a request to the hypervisor to create the VM-1 using the provided image. NFVI returns the information on the VM created to NFVO.

9) NFVO sends a create server command to the NFVI to create VM-2, hosting instance MRF-1 with information coming from MRB VNF.

10) NFVI in turn sends a request to the hypervisor to create the VM-2 using the provided image. NFVI returns the information on the VM created to NFVO.

11) NFVO sends a create server command to the NFVI to create VM-3, hosting instance MRF Storage-1 with information coming from MRB VNF.

12) NFVI in turn sends a request to the hypervisor to create the VM-3 using the provided image. NFVI returns the information on the VM created to NFVO.

13) If any error, return error to caller.

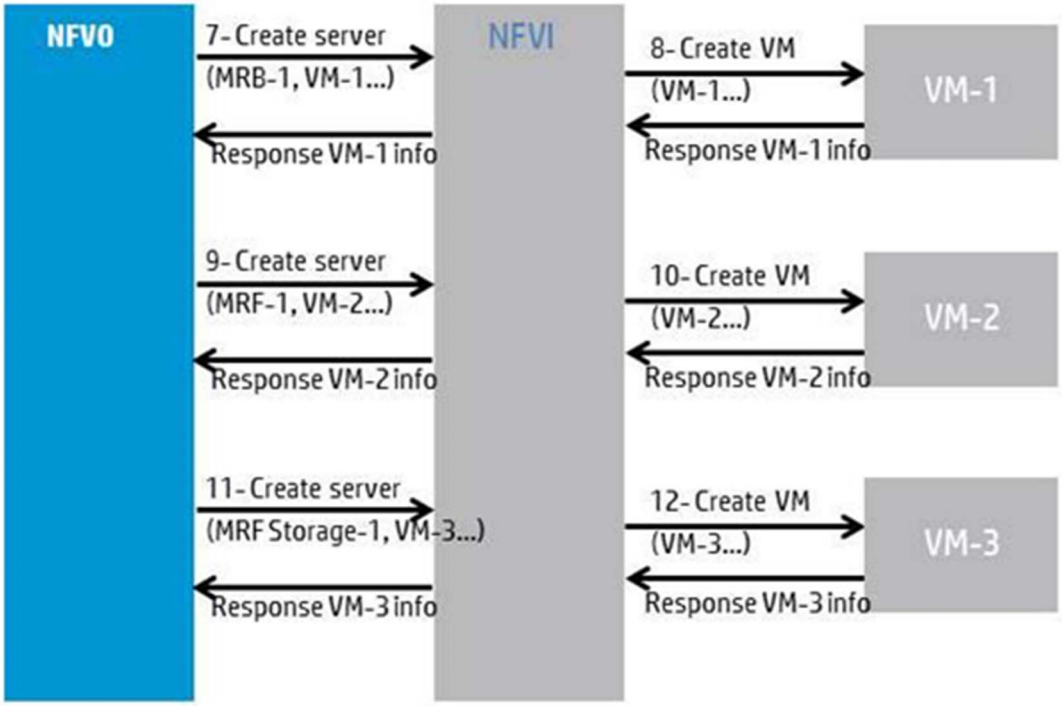


Figure A.6: IMS MRF deployment steps for HW resource 1

14) Steps 7 to 13 are repeated for the second HW resources to create:

   a) VM-4 hosting MRB-2 (role: standby);

   b) VM-5 hosting MRF-2, (role: active);

   c) VM-6 hosting MRF Storage 2 (role: slave).

15) Steps 8 & 9 are repeated for the third HW resource to create VM-7 hosting MRF-3 (role: active).

NOTE 4: Steps 7 to 15 are presented as ordered for ease of reading, but creation of servers might be done in any order and might be parallelized.

16) Once the VM is created, its starts and waits for its instantiation definition file.

17) NFVO augments the Instantiation Definition data with data from the VNFD and data obtained as result of the VM creation (e.g. IP addresses, subnet mask, etc.). This global instantiation definition file contains information on the type of VDU running on each VM (MRB, MRF, MRF Storage), HA role for each instance (active/standby, master/slave), IP addresses of the other instances of same type for redundancy, IP addresses of external components (S-CSCF, App Server).

NOTE 5: There might be multiple ways of providing configuration information to the application and this use case is just using a simple one.

18) NFVO then pushes the augmented instantiation definition file to a pre-defined storage location that will be mounted by the corresponding VM. This file will be used to provide the initial configuration of the application.

19) Once all instantiation definition files have been pushed, NFVO provides a successful response to the provision IMS MRF.

20) Each VM reads the instantiation definition file, discovers its VDU type and start the application.

NOTE 6: This case study is using an Instantiation File read by the VM for configuring the VDU. There might be a lot of alternative ways to do the initial configuration of the IMS MRF components, but from a scenario point of view, with the same end result.

21) Based on its VDU type and the data from the Instantiation Definition file, the application instance (MRB-1, MRB-2, MRF-1, MRF-2, MRF-3, MRB Storage-1, and MRB Storage-2) synchronizes with each other. For instance, secondary registers with primary.

NOTE 7: Start command might be explicitly sent from NFVO or EM to MRF or MRB or both, or can be implicit, i.e. as soon as it is configured, each component can check with its peer and start in the right order. In this use case, application start is automatic and the application is started when the VM is configured and it automatically configures itself once it got its Instantiation Definition file.

This is illustrated by the following sequence diagram where VM #X is one of the VM instantiated and VNF #X is one of the application artefact instances.

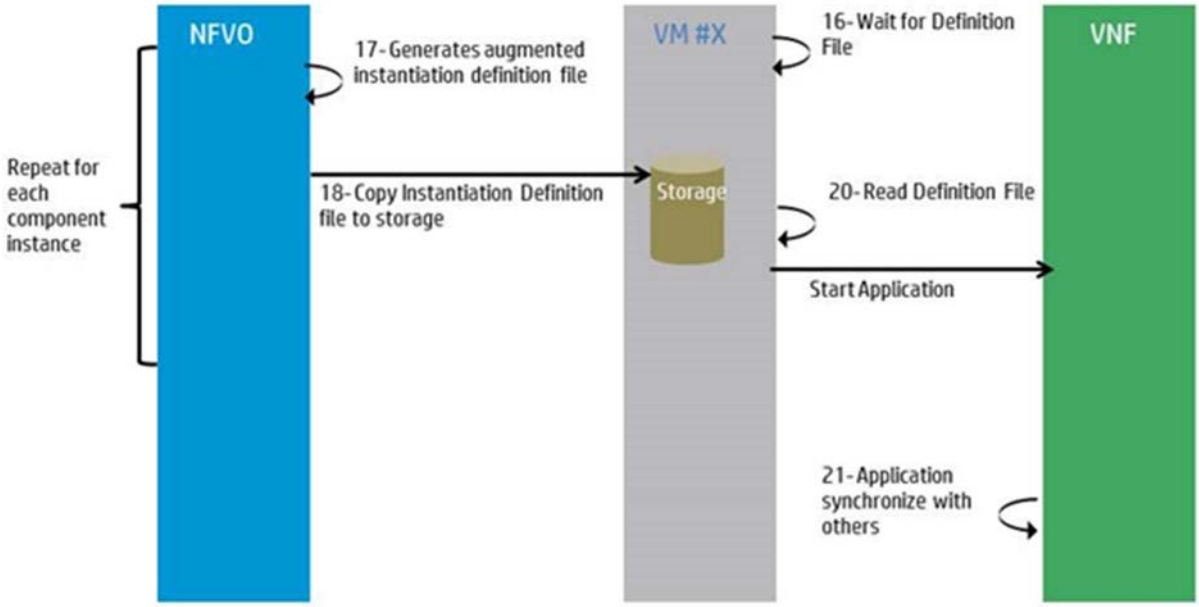


Figure A.7: IMS MRF Deployment Sequence Diagram - Configuration phase

A.2 Network Service fault Management case study

Network Service is achieved through a set of network functions which are VNFs and/or existing network elements. When a network element fails, it can affect performance anomaly of the others and even cause a series of alarms which are almost fake alarms.

Network Service fault refers to Network Service performance anomaly, including service interruption, performance degradation, etc. It can happen that the Network Service performance runs far from the expected capacity as configured at the beginning of Network Service instantiation. At this time, the VNFs could still work, but probably at low performance.

For example, Network Service faults in NFV are shown in figure A.8. This example assumes that the Network Service is composed of the following network functions which are NE1, NE2, NE3, VNF-4, VNF-5 and VNF-6. Moreover, the example assumes the NMS/OSS has the topology of the Network Service. NE2 and NE3 are the same type of devices, e.g. routers in an administration domain. NE1 has the routing knowledge of the Network Service. When NE2 fails to offer any service, the traffic of Network Service which is used to pass through NE2 will change to pass through NE3. At the same time NE2 will send an alarm (e.g. outage alarm) to NMS through EM-2. VNF-4/5 will also send alarm (e.g. VNF-4 out of service alarm, VNF-5 overload alarm) to NMS/OSS via EM-4/5 respectively. The Network Service fault management also requires NMS/OSS and NFVO to support subscription to Network Service fault information.

For Network Service fault management, it is recommended that NMS/OSS supports correlate Network Service fault alarms and NFVI fault events, for Root Cause Analysis (RCA) purpose, eventually, can enhance network availability in NFV environment.

Regarding the enhancements to existing NMS/OSS, for Network Service fault management and relevant NFVI information of VM, hardware and networking service, it Is recommended that NMS/OSS supports to subscribe and receive relevant NFVI fault event from NFVO. It is recommended that NFVO supports to subscribe and receive Network Service fault information from network management system.

For Networks Service management and RCA reasons, NMS can trigger EM to query and receive events about virtualised resources used by the VNF, e.g. failures on a VM.

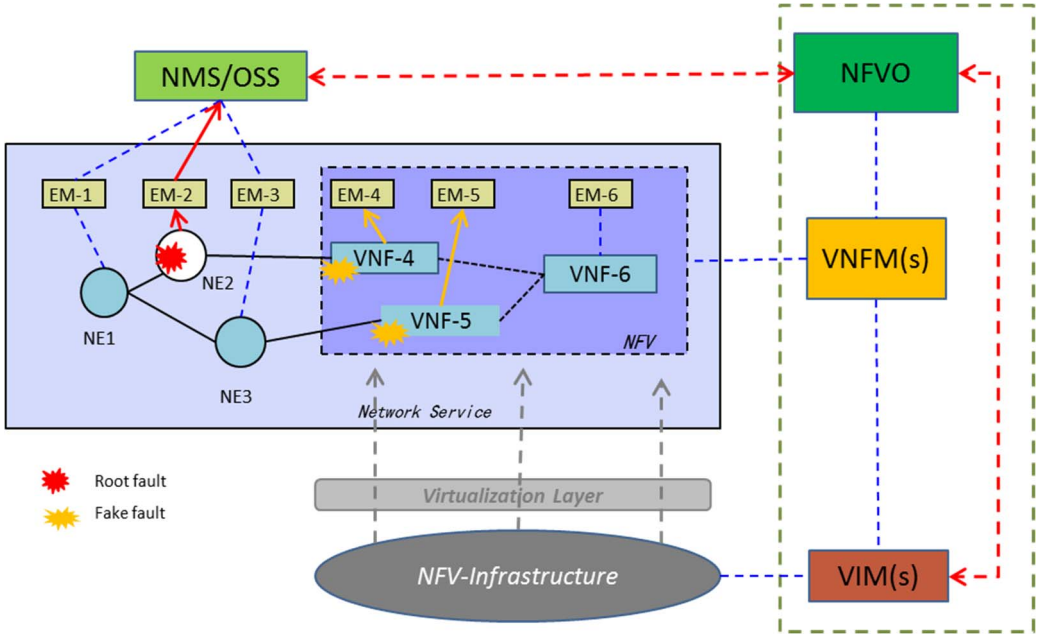


Figure A.8: Network Service fault

Annex B:
VNF lifecycle management

B.1 Introduction

This annex represents a collection (non-exhaustive) of management flows related to VNF Lifecycle that are presented for information and illustration only and not for implementation purpose.

The following principles are used for all the flows in this clause:

- The NFVO is the single point of access for all requests from the OSS to simplify the interfacing.
- The NFVO handles lifecycles of Network Service and VNF Forwarding Graph.
- The VNF Manager handles VNF lifecycle from an application point of view.
- The NFVO has the end-to-end view of the resources being allocated across Network Services and VNFs by VNF Managers: all requests for resource allocation transit through, or are verified and granted by the NFVO.

All the flows in this clause are informative, representing best practices for each of the lifecycle operation and have the main goal of identifying needed interfaces as well as information needed on those interfaces.

At each step, in case of failure, an immediate return might happen. All the failure cases have not been shown on the sequence diagrams or on the text for sake of simplicity.

The dotted-line arrows represent the return path of the request or in some specific cases, a notification. The return path of a request can be a true reply in case of a request/reply exchange or a notification otherwise.

For VNF lifecycle management flows, allocation of resource can be done by the NFVO or by the VNF Manager. For each lifecycle operation, the 2 options, when applicable, are presented in separate clauses.

B.2 VNF Package on-boarding flows

B.2.0 Use Case diagram

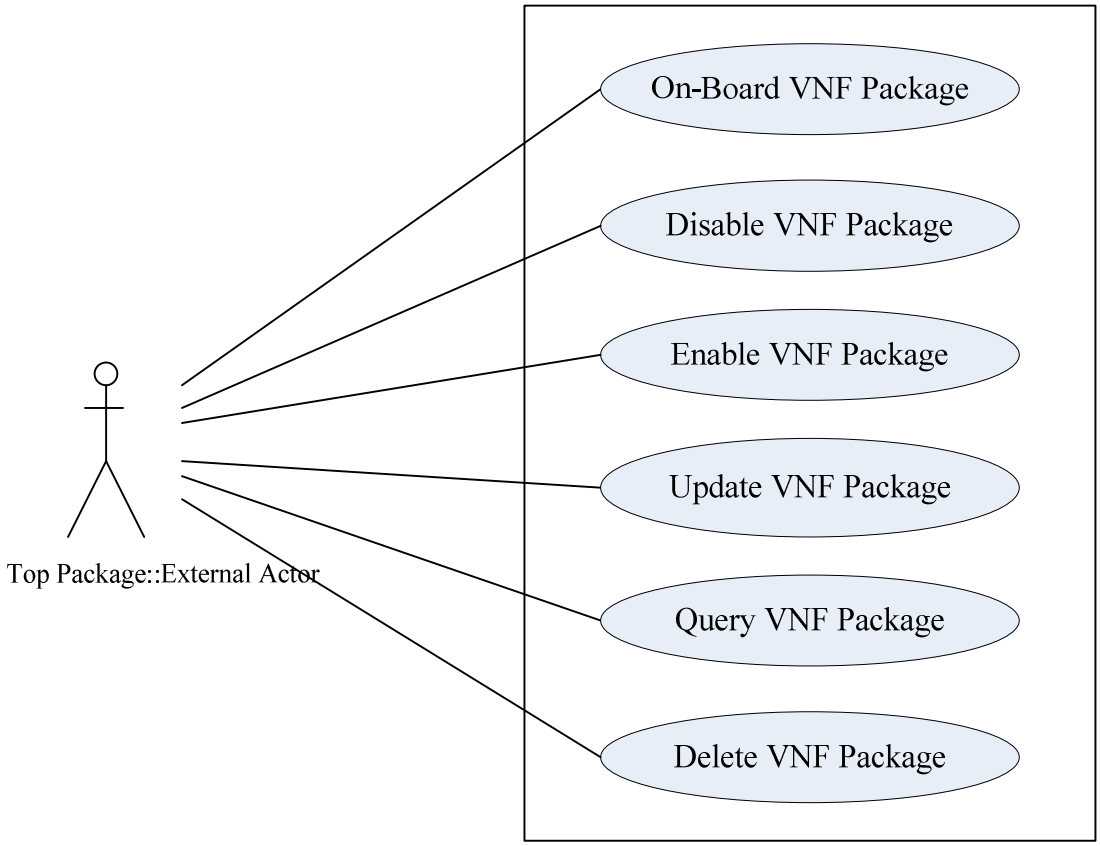


Figure B.1: Use case diagram for VNF Package on-boarding

The use case diagram above provides the following use cases related to VNF Package on-boarding:

- On-board VNF Package.
- Disable VNF Package.
- Enable VNF Package.
- Update VNF Package.
- Query VNF Package.
- Delete VNF Package.

B.2.1 On-board VNF Package flow

VNF Package on-boarding refers to the process of submitting VNF Package to the NFVO to be included in the catalogue.

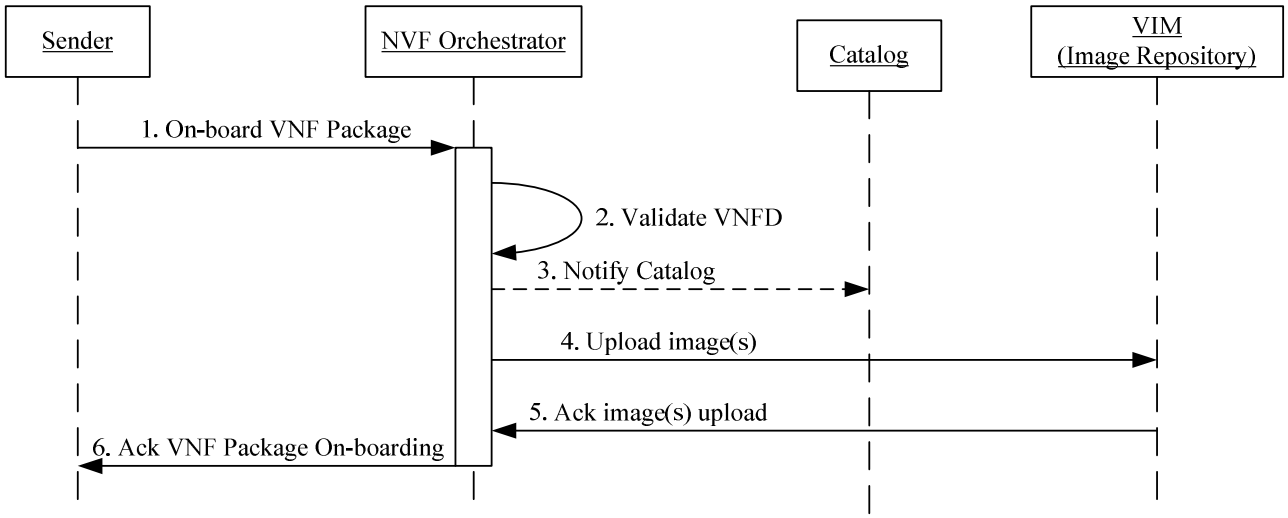


Figure B.2: VNF Package on-boarding message flow

The main steps for VNF Package on-boarding are:

1. VNF Package is submitted to NFVO for on-boarding VNFD using the operation On-board VNF Package of the VNF Package Management interface.
2. NFVO processes the VNFD including (not limited to):
   a) Checking for the existence of mandatory elements.
   b) Validating integrity and authenticity of the VNFD using manifest file and manifest file security received in VNFD.
3. NFVO notifies the catalogue.
4. NFVO makes VM images available to each applicable VIM using the operation Add Image of the VNF Software Image Management interface. It is expected that the VIM validates the integrity of VNF software images(s) as part of this operation.
5. VIMs acknowledge the successful uploading of the image.
6. NFVO acknowledges the VNF Package on-boarding to the sender.

B.2.2 Disable VNF Package flow

Disabling a VNF package refers to the process of marking a VNF Package as disabled in the catalogue, so that it is not possible to instantiate VNFs with it any further.

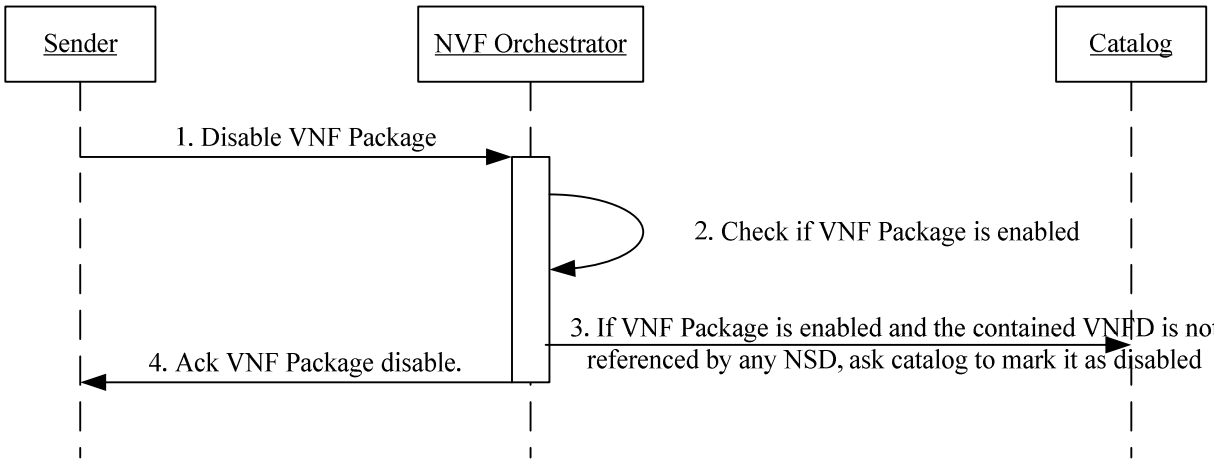


Figure B.3: Disable VNF Package message flow

The main steps for disabling a VNF Package are:

1. A request to disable a VNF Package is submitted to the NFVO using the operation Disable VNF package of the VNF package management interface.
2. The NFVO processes the request and checks if the VNF package exists, is enabled and the contained VNFD is referenced by any NSD.
3. If the VNF Package is enabled and the contained VNFD is not referenced by any NSD, the NFVO notifies the catalogue to disable the VNF Package in the catalogue.
4. The NFVO acknowledges the VNF Package disable request.

B.2.3 Enable VNF Package flow

Enabling a VNF Package refers to the process of marking a VNF Package as enabled in the catalogue, so that it can be used to instantiate VNFs again.

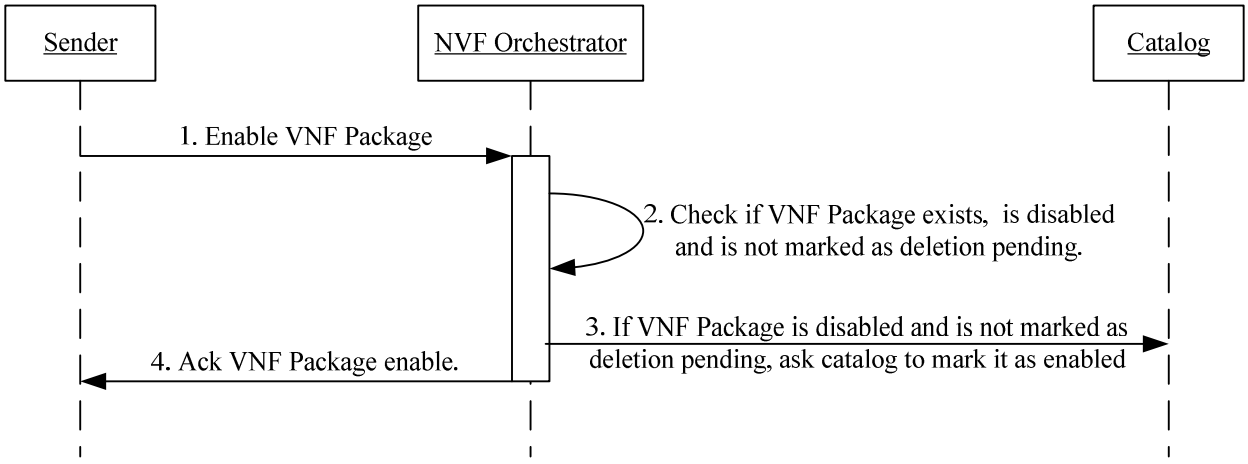


Figure B.4: Enable VNF Package message flow

The main steps for enabling a VNF Package are:

1. A request to enable a VNF Package is submitted to the NFVO using the operation Enable VNF Package of the VNF Package management interface.
2. The NFVO processes the request and checks if the VNF Package exists, is disabled and is not marked as deletion pending. Optionally, the NFVO can validate the stored VNFD before enabling it.
3. If the VNF Package is disabled and is not marked as deletion pending, the NFVO notifies the catalogue to enable the VNF Package in the catalogue.
4. The NFVO acknowledges the VNF Package enable request.

B.2.4 Update VNF Package flow

Updating VNF Package refers to the process of submitting a modified VNF Package to the NFVO to be included in the catalogue.

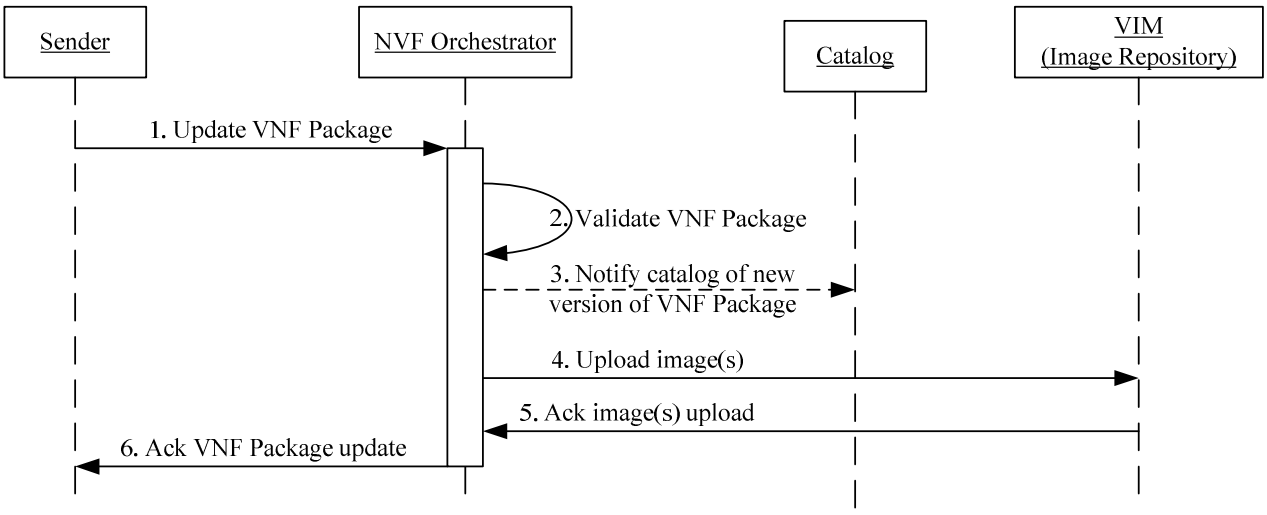


Figure B.5: Update VNF Package message flow

The main steps for VNF Package update are:

1. Modified VNF Package is submitted to the NFVO using the operation Update VNF Package of the VNF Package management interface.
2. The NFVO checks if a VNF Package exists already for this VNF and is not marked as deletion pending and if so, processes the updated VNF Package including (not limited to):
   a) Checking for the existence of mandatory elements.
   b) Validating integrity and authenticity of the VNFD using manifest file and manifest file security received in VNFD.
3. The NFVO notifies the catalogue for insertion of a new version of the VNF Package in the catalogue.

NOTE 1: Some existing VNFs might still use the previous version of the VNF Package, so modifying it will create a new version in the catalogue.

4. Optionally, The NFVO makes new VM images available to each applicable VIM using the operation Update Image of the VNF Image Management interface. It is expected that the VIM validates the integrity of VNF software images(s) as part of this operation.
5. VIMs acknowledge the successful uploading of the image if step 4 has been done.

NOTE 2: Steps 4 and 5 are expected to be performed if VNF images are to be updated. As an alternative, the images could come from EM or VNF and be uploaded to VIM by VNFM.

6. The NFVO acknowledges the VNF Package update.

B.2.5 Query VNF Packages flow

Querying VNF Packages allows returning from the catalogue the information of the VNF Packages.

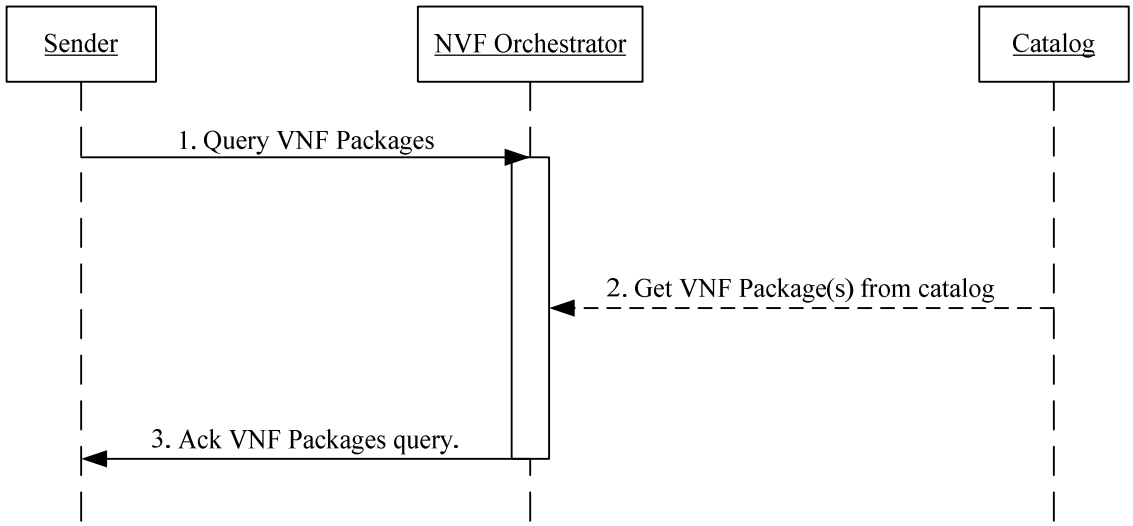


Figure B.6: VNF Package query message flow

The main steps for VNF Package query are:

1. The NFVO receives a query request for the VNF Package(s) using the operation Query VNF Packages of the VNF Package management interface. One or more filter parameters can be included in the Query operation to filter the VNF Packages.
2. The NFVO gets from the catalogue the VNF Package(s) that satisfies the specified filter conditions.
3. The NFVO acknowledges the VNF Packages query.

B.2.6 Delete VNF Package flow

Note that there might be multiple versions of the VNF Package, and the assumption is that the delete request removes all versions.

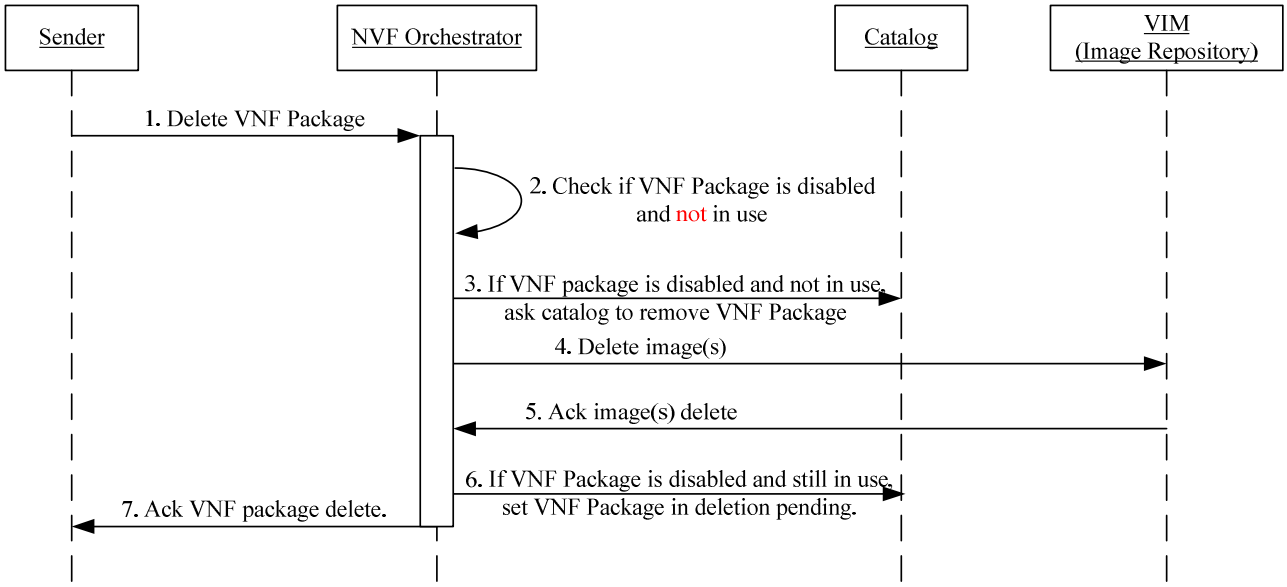


Figure B.7: Delete VNF Package message flow

The main steps for VNF Package deletion are:

1. Request to delete VNF Package is submitted to the NFVO using the operation Delete VNF Package of the VNF Package Management interface.
2. The NFVO checks if the VNF Package is disabled and not in use. If VNF Package is not disabled, the request is rejected.
3. If VNFD is disabled and not in use, then ask the catalogue to remove all the versions of the VNF Package. The catalogue will then remove it/them.
4. The NFVO deletes image(s) from VIM(s) stored during VNF Package on-boarding.
5. VIM acknowledges the successful deleting of the image(s).
6. If VNF Package is disabled and still in use, the NFVO set the VNF Package in deletion pending.
7. The NFVO acknowledges VNF Package deletion request.

NOTE: If the VNF Package is in deletion pending, the NFVO will later check if there is any VNF instance using the VNF Package during the VNF instance termination process. If there is no VNF instance using the VNF Package any more, the NFVO asks the catalogue to remove the corresponding version(s) of the VNF Package.

B.3 VNF instantiation flows

B.3.1 VNF instantiation flows with resource allocation done by NFVO

B.3.1.1 VNF Check Feasibility

The Check Feasibility runs a feasibility check of the VNF instantiation or scaling request to reserve resources before doing the actual instantiation/scaling.

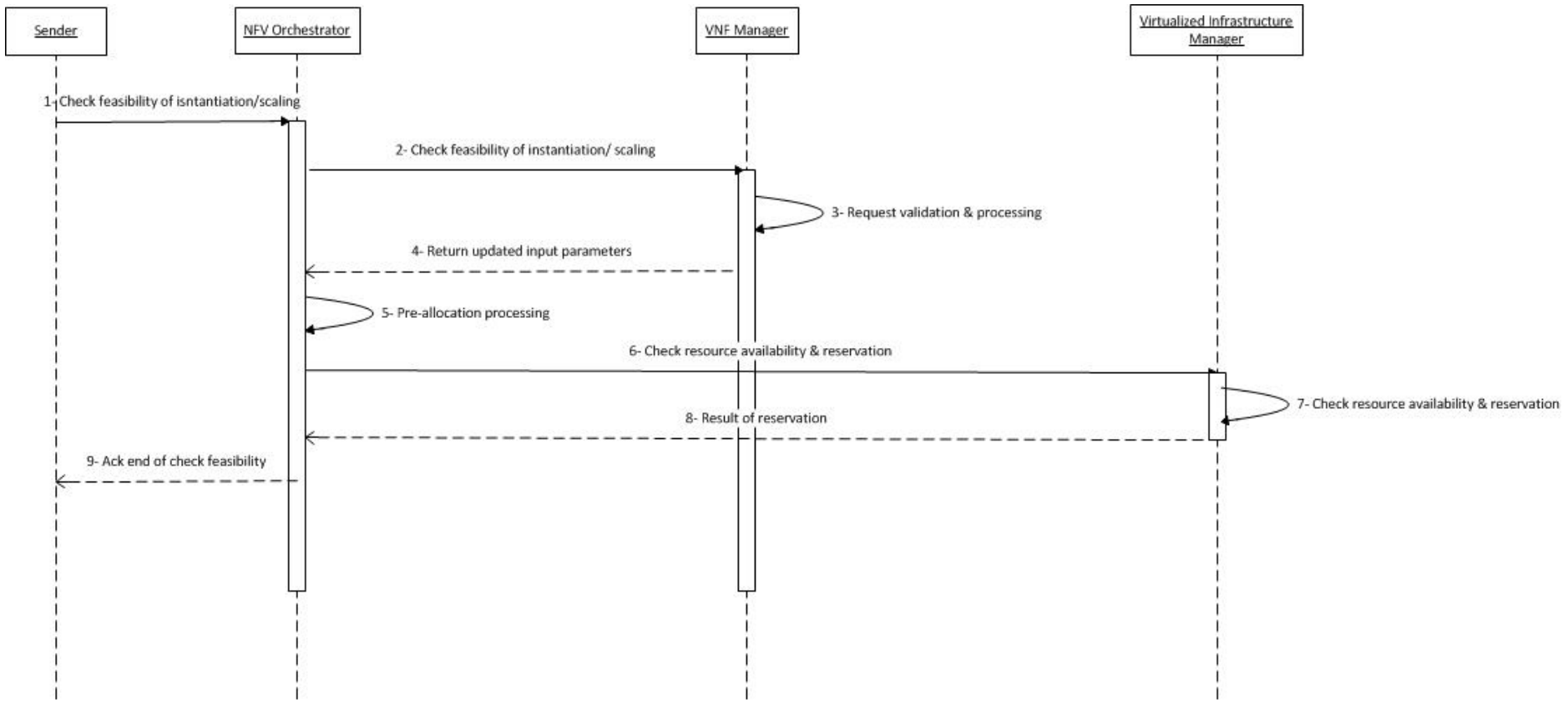


Figure B.8: VNF check feasibility message flow

NFVO receives a request to check feasibility of VNF instantiation/scaling. This request might come from an OSS, commissioning of a new VNF or VNF scaling, or part of an order for a Network Service instantiation/scaling. In most of the case, it will be an option of a VNF instantiation or scaling request.

The main steps for VNF check feasibility are:

1. NFVO receives a request to check feasibility of VNF instantiation/scaling.
2. NFVO calls VNF Manager to check feasibility of instantiation/scaling request with the input parameters provided using the operation Instantiate/Scale VNF of the VNF Lifecycle Management interface.
3. VNF Manager validates the request and processes the VNFD and the input parameters. This might include modifying/complementing the input parameters with VNFD data and VNF lifecycle specific constraints. See details in clause B.3.1.5 VNF Manager: Request validation and processing.
4. VNF Manager returns in response to step 2 the (possibly) updated list of input parameters to NFVO as response of the Check VNF instantiation/scaling feasibility.
5. NFVO executes any needed pre-allocation processing work. See details in clause B.3.1.6 NFVO: Pre-allocation processing.
6. Optionally, NFVO requests to VIM for availability of additional resources (compute, storage and network) needed for the various VDUs of the VNF instance and reservation of those resources using the operation Create Resource Reservation of the Virtualised Resources Management interface.
7. VIM checks the availability of additional resources (compute, storage and network) needed for the various VDUs of the VNF instance and reserves them.
8. VIM returns result of reservation back to NFVO as response to step 6.
9. The NFVO acknowledges the completion of the check feasibility.

NOTE: The feasibility check can be considered from 2 perspectives:

1) Feasibility from the perspective of the overall availability of resources. This is the role of the NFVO initiating the feasibility check/reservation.

2) Feasibility from the perspective whether the VNF instantiation/scaling with its input parameters and the VNFD make sense. This is the VNF Manager role initiating the feasibility check/reservation. This particular operation has been specified on the VNF Lifecycle management interface where the interface is produced by the VNFM and consumed by the NFVO.

This flow considers the first perspective. Another variant, if this flow is not executed, would be to consider the feasibility check as an option done by the VNF Manager.

B.3.1.2 VNF instantiation flow

VNF instantiation refers to the process of identifying and reserving the virtualised resources required for a VNF, instantiating the VNF and starting the VDU associated with each VNF.

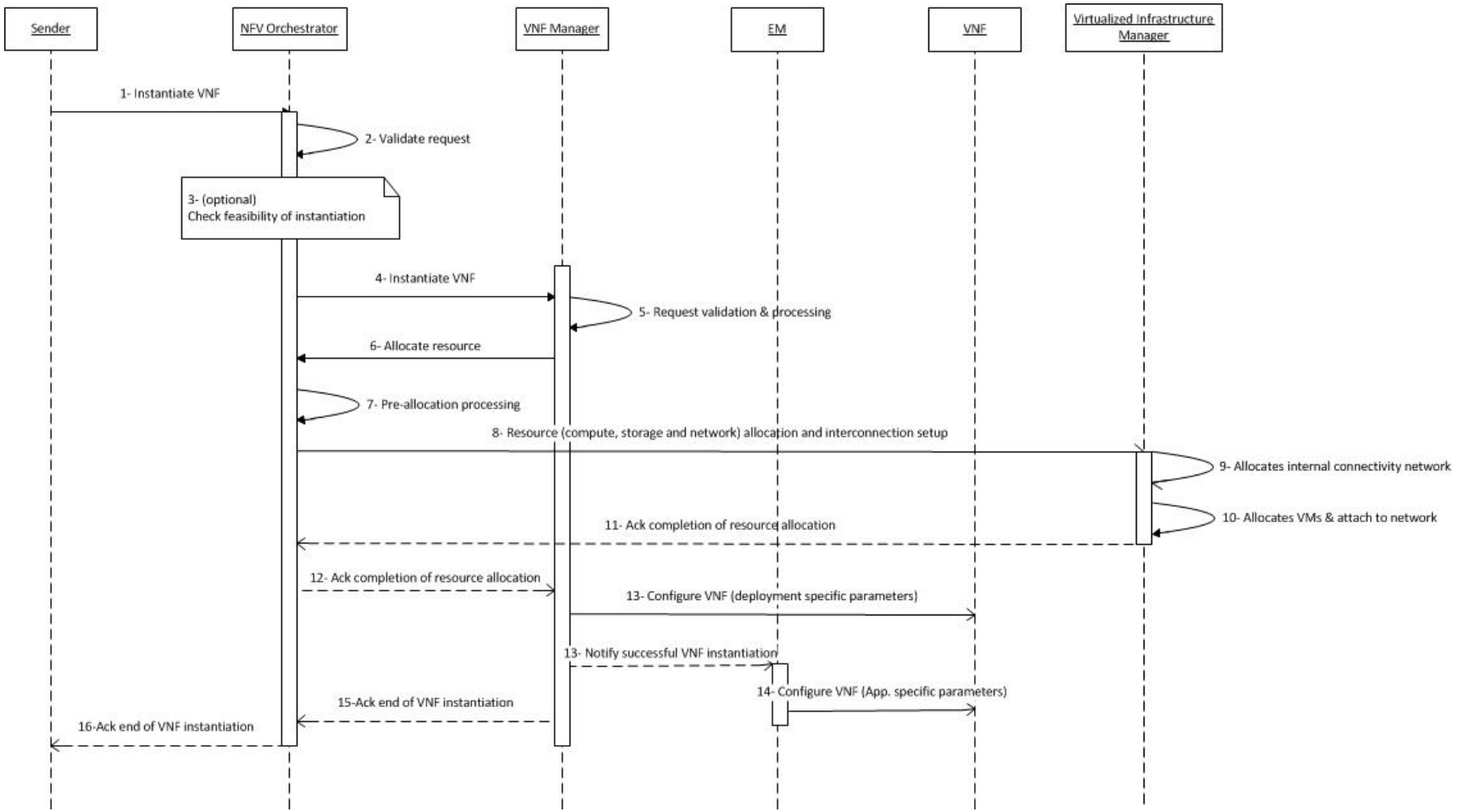


Figure B.9: VNF instantiation message flow

NFVO receives a request to instantiate a new VNF. This request might come from an OSS, commissioning of a new VNF or part of an order for a Network Service instantiation, or might come from the VNF Manager when the need to instantiate a new VNF is detected by the VNF Manager itself or by the EM. Thus the Sender in the above diagram can be the OSS or a VNF Manager.

The main steps for VNF instantiation are:

1. NFVO receives a request to instantiate a new VNF using the operation Instantiate VNF of the VNF Lifecycle Management interface along with instantiation data required to deploy the VNF.

2. NFVO validates the request. See details in clause B.3.1.3 NFVO: Validation.

3. Optionally, NFVO runs a feasibility check of the VNF instantiation request to reserve resources before doing the actual instantiation as described in clause B.3.1.1.

4. NFVO calls VNF Manager to instantiate the VNF, with the instantiation data and, if step 3 has been done, the reservation information using the operation Instantiate VNF of the VNF Lifecycle Management interface. See details in clause B.3.1.4 NFVO: Request to VNF Manager to instantiate the VNF.

5. VNF Manager validates the request and processes it. This might include modifying/complementing the input instantiation data with VNFD data and VNF lifecycle specific constraints.

6. VNF Manager then calls the NFVO for resource allocation using the operation Allocate Resource of the Virtualised Resources Management interface. See details in clause B.3.1.5 VNF Manager: Request validation and processing.

7. NFVO executes any needed pre-allocation processing work. See details in clause B.3.1.6 NFVO: Pre-allocation processing.

8. NFVO requests allocation of resources to the VIM (compute, storage and network) needed for the various VDUs of the VNF instance using the operation Allocate Resource of the Virtualised Resources Management interface. See details in clause B.3.1.7 Orchestration: Resource allocation (compute, storage and network) and interconnection setup.

9. VIM allocates the internal connectivity network.

10. VIM allocates the needed compute (VMs) and storage resources and attaches instantiated VMs to internal connectivity network.

NOTE: The VIM utilizes southbound interfaces towards the NFVI to allocate resources. Allocation includes starting up services such as a VM.

11. Acknowledgement of completion of resource allocation back to NFVO.

12. NFVO acknowledges the completion of the resource allocation back to VNF Manager, returning appropriate configuration information.

13. VNF Manager configures the VNF with any VNF specific lifecycle parameters and using the get/create/set config object operations over the VNF configuration interface. Additionally, the VNFM notifies the EM (if present) of the new VNF.

14. The EM configures the VNF using the get/create/set config object operations over the VNF configuration interface.

15. VNF Manager acknowledges the completion of the VNF instantiation back to the NFVO.

16. The NFVO acknowledges the completion of the VNF instantiation.

B.3.1.3 NFVO: validation

When receiving a request to instantiate a VNF, the NFVO executes the following steps:

1) Verify validity of request to instantiate VNF. It includes validating that the sender is authorized to issue this request.

2) If needed, validate parameters passed for VNF instantiation for technical correctness and policy conformance, e.g. mandatory parameters present, instantiation parameters within policies.

As an option, the NFVO can run a feasibility check of whether the VNF instantiation request is feasible or not up to the reservation of resources before doing the actual instantiation. This complete step is optional, but when executed, all sub-steps are done. It would involve additional exchange with the VNF Manager to get VDU details and with VIM to get resource availability.

In case of validation error, the NFVO would return the call after this step.

This assumes that the NFVO knows how to verify that the VNF Manager is up and running.

B.3.1.4 NFVO: request to VNF Manager to instantiate the VNF

The NFVO requests the VNF Manager to instantiate the VNF.

If the resource reservation (step 3) has been done beforehand, then the NFVO will use as input parameters the instantiation data and the reservation information provided by VIM.

If the resource reservation has not been done, then the NFVO will use as input parameter the instantiation data provided.

B.3.1.5 VNF Manager: request validation and processing

The VNF Manager executes the following steps:

1) Verify validity of request to instantiate VNF.

2) If needed, validate parameters passed for VNF instantiation for technical correctness.

3) Any other VNF lifecycle specific validation, e.g. license check.

4) Based on VNF lifecycle constraints, modify/complement the input parameters coming from the NFVO with information from VNFD data and application specific constraints.

NOTE: Those changes are compatible with both the VNFD and the policies defined at NFVO level.

As an option, VNF Manager can ask for resource reservation either globally as a single request, concatenating the information related to all VDUs in the same instantiation data structure or can issue to the NFVO multiple requests (one per VDU). In all cases, VNF Manager will send the reservation request(s) to the NFVO that will process it (them) with the VIM.

The VNF Manager then calls the NFVO for allocating the resources needed for this VNF instance. For sake of simplicity, a single call is assumed, but one call per VDU might be possible.

B.3.1.6 NFVO: pre-allocation processing

When receiving a request to allocate resources for a VNF, the NFVO executes the following steps:

1) If needed, validate parameters passed for VNF resource allocation for technical correctness (e.g. against VNFD) and policy conformance.

2) Location selection: The selection of where to locate a VNF instance could be based on the request, available resources, the nature of the VNF, the Network Service(s) in which the VNF instance is participating in as well as defined policies. Note that not all VDU instances will be placed in the selected location-it only serves to provide the primary affinity/non-affinity point for the VNF instance.

3) Dependency checking: Availability of all required external dependencies from the required location is expected to be checked. If the VNF instance has any QoS requirements, it is also expected to be verified that they can be met in the selected location. Note that the QoS requirements could be on compute or network resources or on external services on which the VNF instance is dependent.

B.3.1.7 Orchestration: resource allocation (compute, storage and network) and interconnection setup

The NFVO executes the following steps:

1) Resource pool selection: The resource pool to be used is expected to be selected. Note that this is not the same as the VNF location. Multiple resource pools could exist in the same location or some VDU instances that are part of a VNF instance could expect to be located remotely from the rest.

2) Request instantiation of the internal connectivity network: For a VNF that requires dedicated virtual networks to interconnect VDU instances (networks that are only used as internal to the VNF instance) these virtual networks are expected to be created.

3) Requesting instantiation of the needed compute and storage resources from the infrastructure (Virtualised Infrastructure Manager). Note that there might be multiple distributed VIMs.

4) Attach instantiated VMs to internal connectivity network.

Each virtual NIC on the VMs hosting the VDUs that constitute the VNF instance is expected to be appropriately connected. If there are particular QoS requirements on a network connection, the network can expect to be properly configured to support these.

At the end of this step, the NFVO acknowledges the completion of the resource and network allocation back to VNF Manager and returns back information on the allocated resources and network connections.

Note that resource allocation request is likely to be asynchronous and depending on the implementation of the VIM, the NFVO might have to poll for completion of resource allocation or wait for a notification from VIM.

B.3.2 VNF instantiation flows with resource allocation done by VNF Manager

B.3.2.1 VNF instantiation from EM

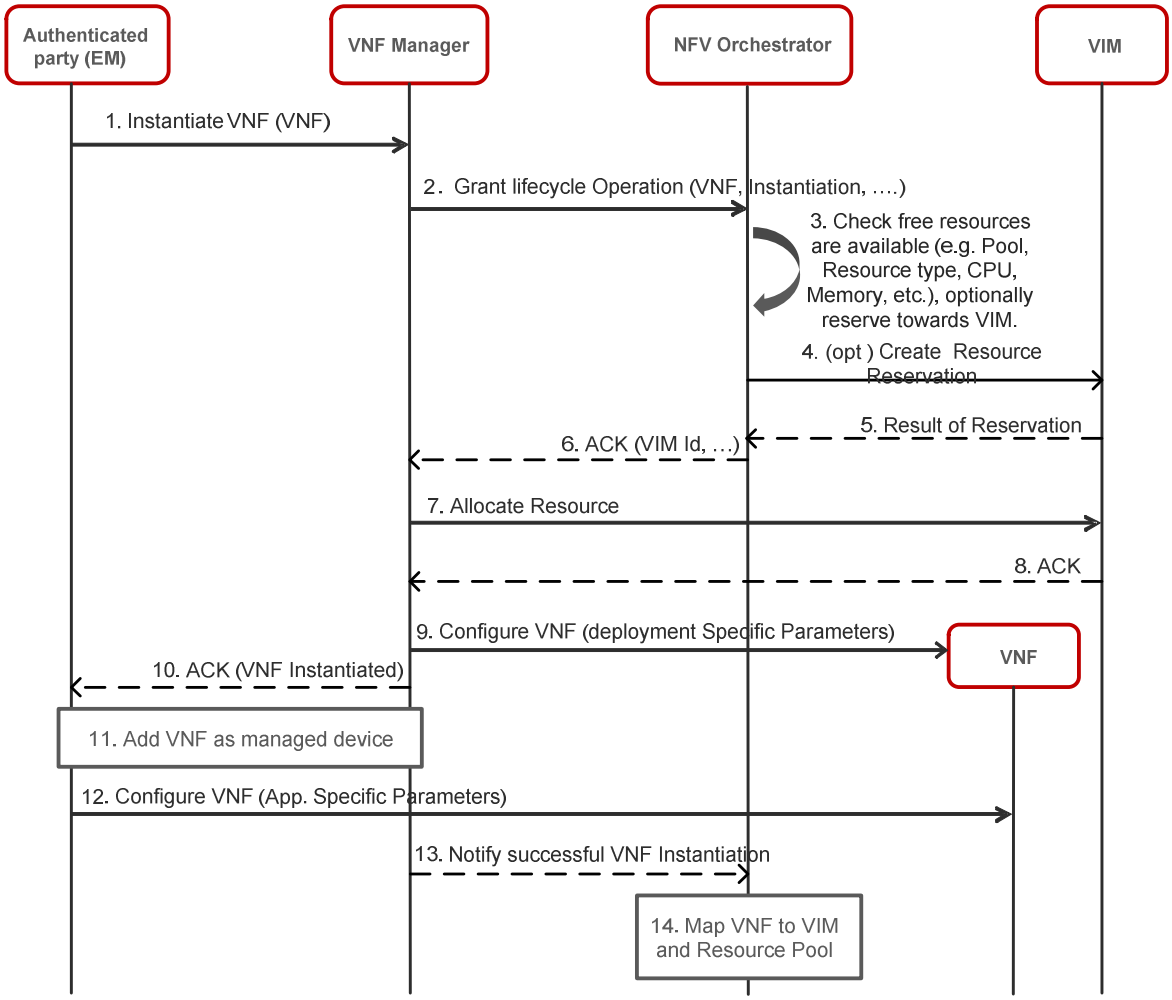


Figure B.10: VNF Instantiation from EM flow

NOTE: This flow is valid, when the EM consumes the VNF Lifecycle Management interface exposed by the VNFM over the Ve-Vnfm reference point.

The main steps for VNF instantiation from EM are:

1. EM requests to the VNF Manager instantiation of a new VNF in the infrastructure using the operation Instantiate VNF of the VNF Lifecycle Management interface. It sends information on which VNF type the VNF is to be instantiated.

2. The VNF Manager requests granting to the NFVO to instantiate the VNF according to the information in the VNFD (CPU, Memory, IP, etc.) using the operation Grant Lifecycle Operation of the VNF Lifecycle Operation Granting interface.

3. The NFVO checks resource request from VNFM against its capacity database for free resource availability.

4-5. The NFVO can otherwise optionally do resource reservation for the requested resources by using the Create Resource Reservation operation over the Virtualised Resources Management interface.

6. The NFVO responds to the VNF Manager request by sending the VIM Identifier, as well as the needed reservation information if performed (step 4), to inform the VNF Manager where to instantiate the VNF as a response to step 2.

7. The VNF Manager sends the request to create and start the VMs as appropriate and as instructed by the NFVO, sending VIM Identifier and VM parameters using the operation Allocate Resource of the Virtualised Resources Management interface.
8. The VIM creates and starts the VMs and relevant networking resources, then acknowledges successful operation to the VNF Manager.
9. VNF Manager configures VNF data being specific for VNF instantiation using the add/create/set config object operations of the VNF Configuration interface.
10. VNF Manager notifies successful VNF instantiation back to the EM as response to request made in step 1.
11. EM and VNF Manager add the new VNF as managed device.
12. EM configures the VNF with application specific parameters.
13. VNF Manager reports successful VNF instantiation to the NFVO using the VNF Lifecycle Change Notification interface. The NFVO now is aware that the new VNF is instantiated in the infrastructure.
14. VNF NFVO maps the VNF to the proper NFVI-PoP and Resource Pool.

B.3.2.2 VNF instantiation from NFVO

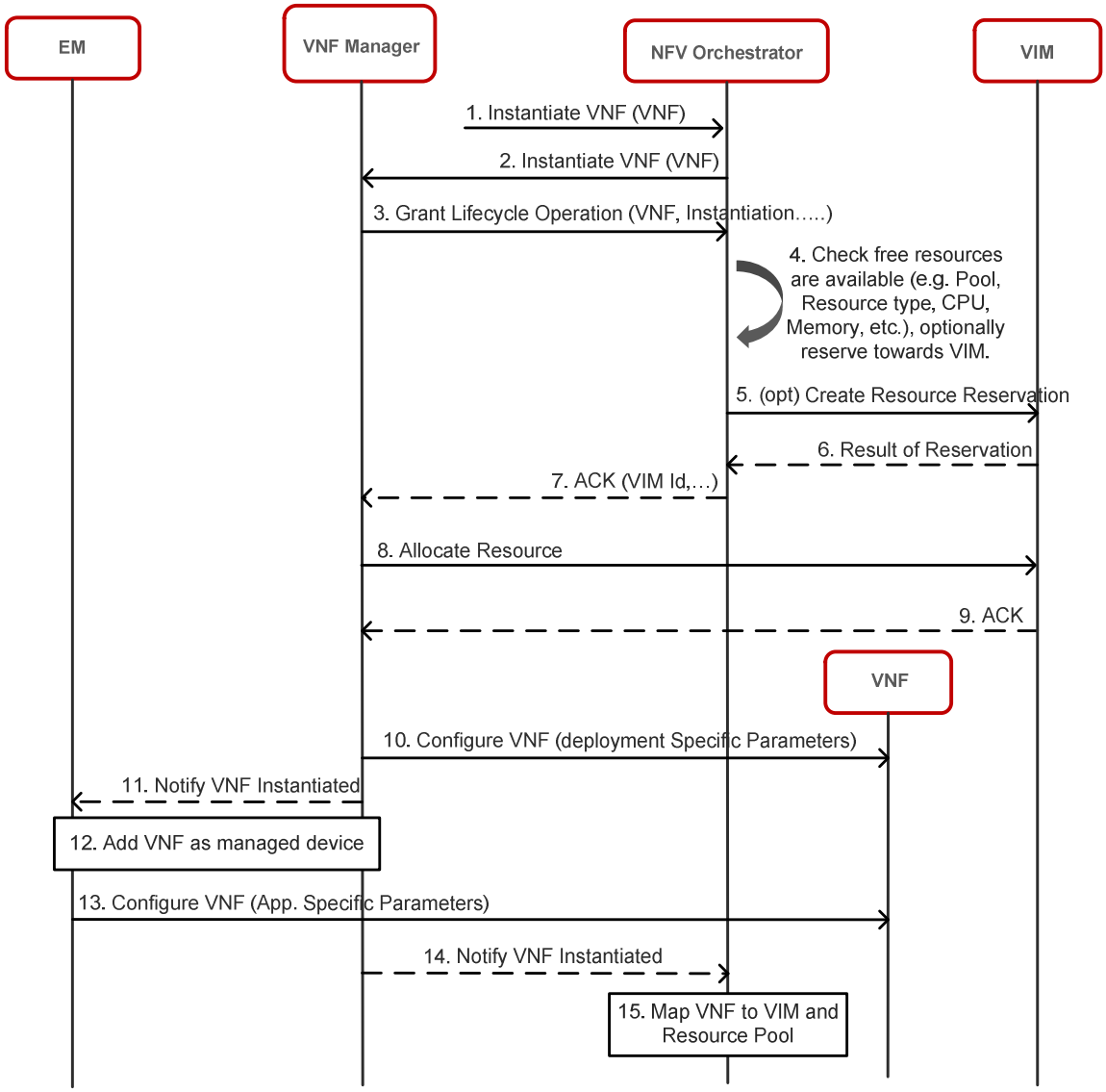


Figure B.11: VNF instantiation from NFVO flow

The main steps for VNF instantiation from NFVO are:

1. The NFVO receives a trigger to instantiate a VNF in the infrastructure (this can be a manual trigger or an automatic service creation trigger request e.g. from the OSS/BSS) using the operation Instantiate VNF of the VNF Lifecycle Management interface.

2. NFVO requests to the VNF Manager instantiation of a new VNF in the infrastructure using the operation Instantiate VNF of the VNF Lifecycle Management interface. VNF instantiation information is included.

3. The VNF Manager requests granting to the NFVO to instantiate the VNF according to the information in the VNFD (CPU, Memory, IP, etc.) using the operation Grant Lifecycle Operation of the VNF Lifecycle Operation Granting interface.

4. The NFVO checks resource request according to the information received in the granting request against its capacity database for free resource availability.

5-6. The NFVO can otherwise optionally do resource reservation for the requested resources by using the Create Resource Reservation operation over the Virtualised Resources Management interface.

7. The NFVO responds to the VNF Manager request by sending the VIM Identifier, as well as other needed reservation information if performed (step 5), to inform the VNF Manager where to instantiate the VNF as a response to step 4.

8. The VNF Manager sends the request to create and start the VMs as appropriate and as instructed by the NFVO, sending VIM Identifier and VMs parameters using the operation Allocate Resource of the Virtualised Resources Management interface.

9. The VIM creates and starts the VMs and the networking resources then acknowledges successful operation to the VNF Manager.

10. VNF Manager configures VNF data being specific for VNF instantiation using the add/create/set config object operations of the VNF Configuration interface.

11. VNF Manager notifies the EM that a new VNF is created using the VNF Lifecycle Change Notification interface.

12. EM and VNF Manager add the new VNF as managed device.

13. EM configures the VNF with application specific parameters.

14. VNF Manager reports successful VNF instantiation to the NFVO as response to request made in step 3. The NFVO now is aware that the new VNF is instantiated in the infrastructure.

15. The NFVO maps the VNF to the proper VIM and resource pool.

B.4 VNF instance scaling flows

B.4.1 Detecting need to scale

VNF instance scaling is often the result of a service quality threshold being crossed - whether because service quality is no longer acceptable, requiring expanding capacity or because service quality and utilization is such that capacity can be contracted without affecting quality delivered.

About the source that will initiate the decision process, it can come from:

1) VNF, when it embeds a monitoring function/threshold crossing detection and event notification. VNF can send that event to EM and decision about actions can be implemented in EM and forwarded to VNF Manager. VNF can send that event directly to the VNF Manager as well.

2) VNF Manager, when reception of a single VNF or infrastructure event might be sufficient to detect the need to scale, then information on the event to monitor and the corresponding scaling action might be provided in the VNF Descriptor.

3) VIM, when VIM implements a monitoring function/threshold crossing detection and event notification. Event would be about network congestion, number of sessions, etc. VNF Manager will listen to those events and implement the decision about actions.

4) EM when the monitoring function/threshold crossing detection and event notification is not in the VNF. Decision about actions can be implemented in EM and forwarded to VNF Manager.

5) OSS/BSS when the monitoring function/threshold crossing detection & event notification is not in the EM or crosses several EM. OSS/BSS would be both the detector and decision point.

6) OSS/BSS when this is a change management process/capacity planning process based on traffic projections for example.

7) Manual change triggered by an operator.

From the list above, the scaling use cases can be grouped in 3 categories:

1) Auto-scaling, in which the VNF Manager monitors the state of a VNF instance and triggers the scaling operation when certain conditions are met. For monitoring a VNF instance's state, it can for instance track infrastructure-level and/or VNF-level events. Infrastructure-level events are generated by the VIM. VNF-Level events can be generated by the VNF instance or its EM.

2) On-demand scaling, in which a VNF instance or its EM monitor the state of a VNF instance and trigger a scaling operation through explicit request to the VNF Manager.

3) Scaling based on management request, where the scaling request is triggered by some sender (OSS/BSS or operator) towards VNFM via the NFVO.

For sake of simplicity, the same flow will be used for both cases even if some simplification might be possible.

B.4.2 Determining scaling action

The nature of the VNF and the measurement results that cross their thresholds will generally indicate the type of change required. These can be:

- configuration changes to the VM (scale up, e.g. add CPU or memory);
- add a new VDU instance (scale out);
- shut down and remove instances (scale in);
- release resources from existing instances (scale down);
- increase available network capacity;
- provide increased bandwidth (or other network changes).

Determining the scaling action could require looking beyond the VNF instance being examined. For example, solving a quality issue with VNF instance-1 could require changes to VNF instance-2, which might be of a different VNF type. For example:

- The reason VNF instance-1 is performing poorly is because of congestion caused by VNF instance-2. If it is costly to move VNF instance-1 but not costly to move VNF instance-2, the right solution is to move VNF instance-2, even if the service quality it is delivering acceptable.
- VNF instance-1 could have an external dependence on VNF instance-2. If VNF instance-2 can only adequately support a given number of components, VNF instance-1 creation or scaling could require that VNF instance-2 is proactively scaled.

Note that such use cases would most likely be detected by an OSS or an EM if both VNF types are managed by the same EM.

B.4.3 Scaling flow with resource allocation done by NFVO

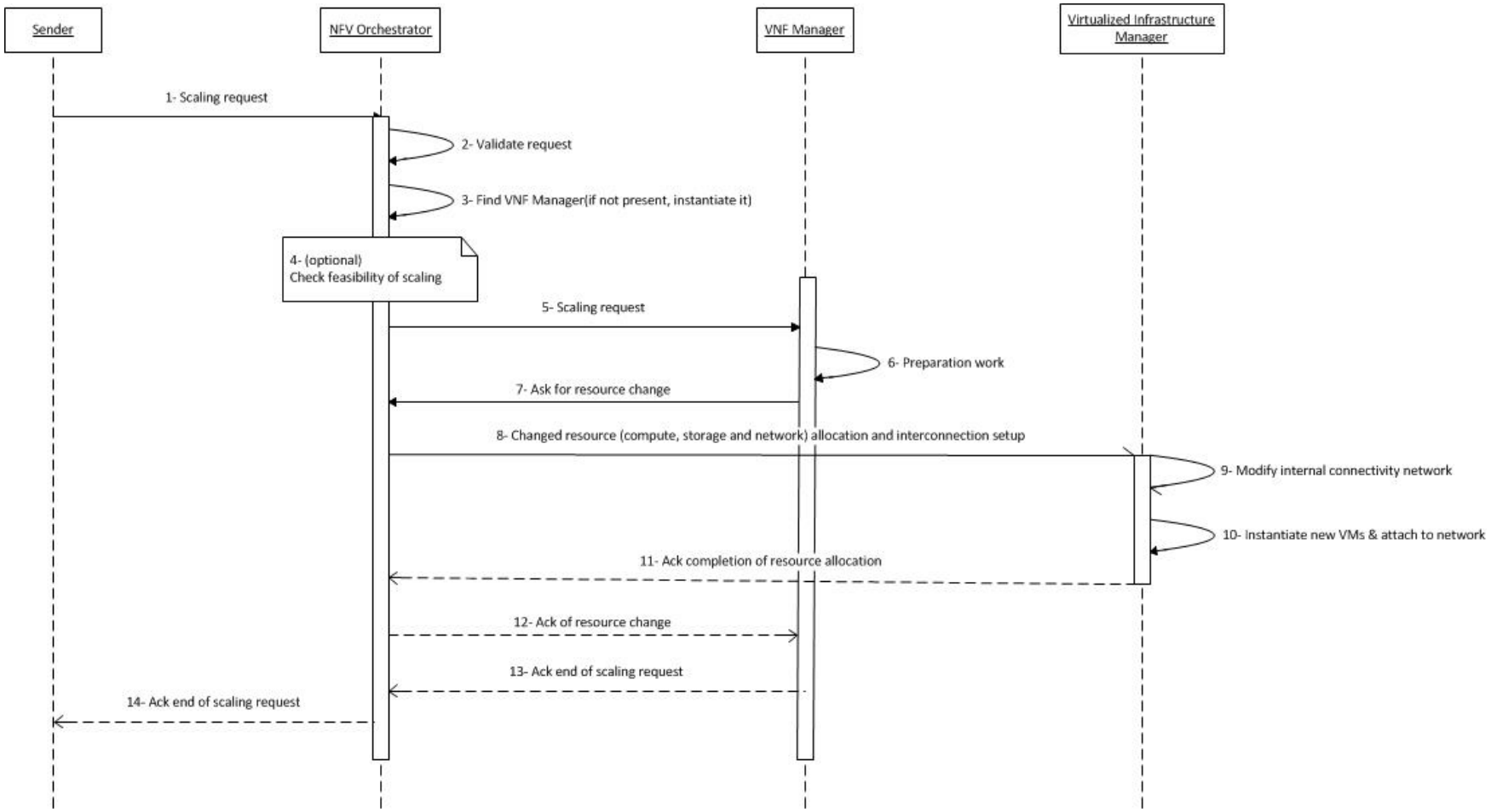


Figure B.12: VNF instance scaling message flow

In accordance with clause B.4.1 the sender can be the VNF manager, or OSS/BSS, or else be manually triggered by an operator. The possible originators of the scaling request are listed in clause B.4.1.

In case the VNF Manager is the one issuing the scaling request, some of these flows might be simplified.

The main steps for the VNF instance scaling are:

1) The NFVO receives the scaling request from the sender, e.g. OSS using the operation Scale VNF of the VNF Lifecycle Management interface.

2) The NFVO validates the request for policy conformance.

3) NFVO finds the VNF Manager relevant for this VNF type.

4) Optionally, NFVO runs a feasibility check of the VNF scaling request to reserve resources before doing the actual scaling as described in clause B.3.1.1.

5) The NFVO sends the scaling request to the VNF Manager, with the scaling data and, if step 4 has been done, the reservation information using the operation Scale VNF of the VNF Lifecycle Management interface.

6) The VNF Manager executes any needed preparation work: request validation, parameter validation. This might include modifying/complementing the input scaling data with VNF lifecycle specific constraints. If step 4 was done then the VNFM will skip step 6.

7) The VNF Manager calls the NFVO for resource change using the operation Allocate Resource or Update Resource or Scale Resource of the Virtualised Resources Management interface.

8) NFVO requests from VIM allocation of changed resources (compute, storage and network) needed for the scaling request using the operations Allocate Resource or Update Resource or Scale Resource of the Virtualised Resources Management interface.

9) VIM modifies as needed the internal connectivity network.

10) VIM creates and starts the needed new compute (VMs) and storage resources and attaches new instantiated VMs to internal connectivity network.

11) Acknowledgement of completion of resource change back to NFVO.

12) NFVO acknowledges the completion of the resource change back to VNF Manager.

13) The VNF Manager configures the scaled VNF as necessary using the add/create/set config object operations of the VNF configuration interface.

14) VNF Manager acknowledges the end of the scaling request back to the NFVO.

15) The NFVO acknowledges the end of the scaling request back to the requester.

In case the VNF Manager is issuing the scaling request, steps 1 to 3 of this flow and steps 1 to 3 of the check feasibility flow will be skipped and step 4 of the feasibility flow will be replaced by an (optional) request from the VNF Manager to the NFVO to grant the scaling request and optionally return the reserved resources information, using the VNF lifecycle operation granting interface. Steps 5 and 6 of this flow will also be skipped in this case.

B.4.4 Scaling flows with resource allocation done by VNF Manager

B.4.4.1 Automatic VNF expansion triggered by VNF performance measurement results

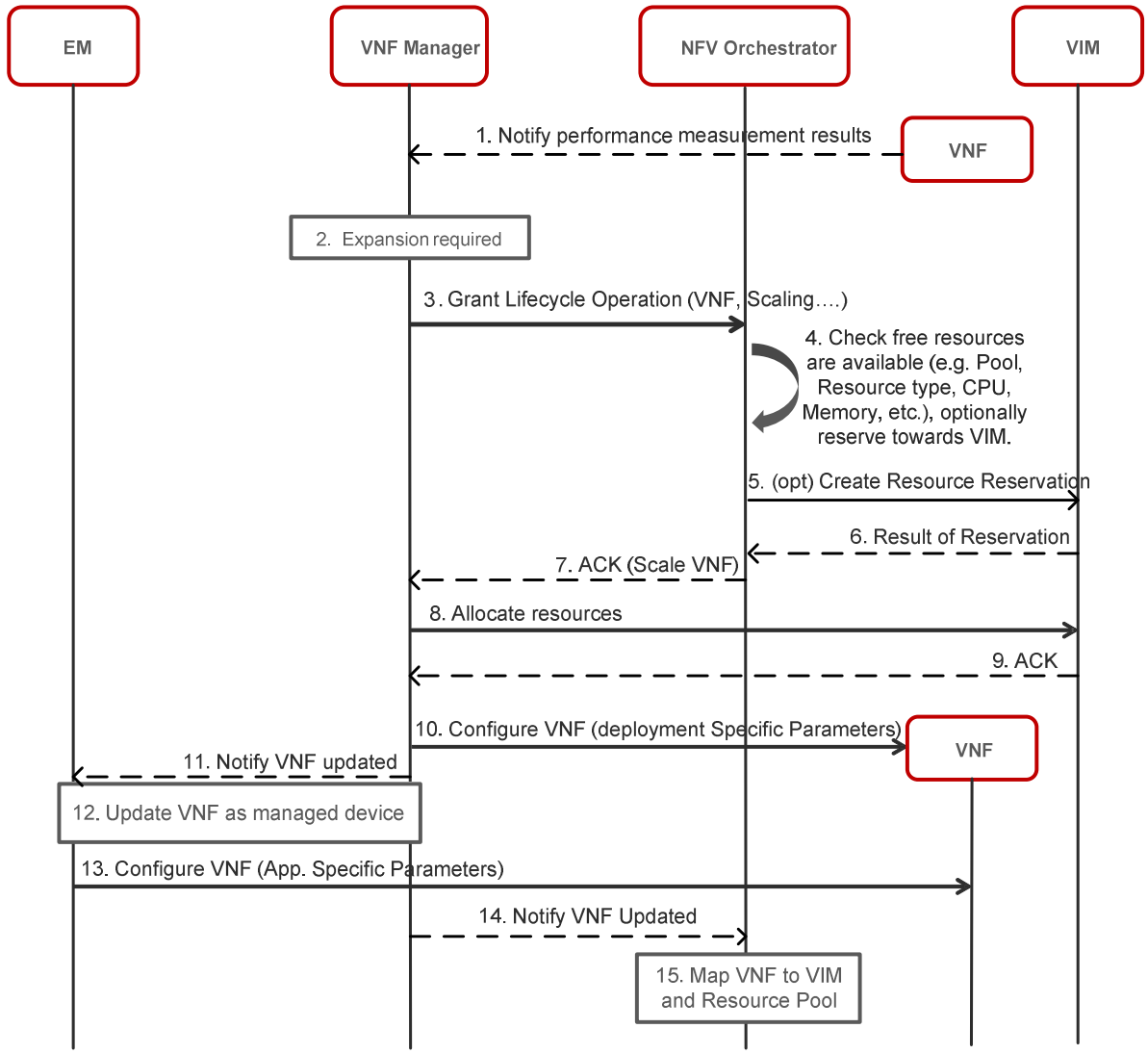


Figure B.13: Automatic VNF expansion flow triggered by VNF performance measurement results

VNF expansion refers to the addition of capacity that is deployed for a VNF. Expansion can result in a scale out of a VNF by adding VNFCs to support more capacity or can result in a scale-up of virtualised resources in existing VNF/VNFCs. VNF expansion can be controlled by an automatic process or can be a manually triggered operation. The following flow achieves automatic VNF expansion with the scale out example.

The main steps for the automatic VNF expansion are:

1. The VNF Manager collects measurement results from the VNF (application specific) using the operation Notify or Get performance measurement results of the VNF Performance Management interface.
2. VNF Manager detects a capacity shortage that requires expansion (more resources).
3. The VNF Manager requests granting to the NFVO for the VNF expansion based on the specifications listed in the VNFD (CPU, Memory, IP, etc.) using the operation Grant Lifecycle Operation of the VNF Lifecycle Operation Granting interface.

4. The NFVO takes scaling decision and checks resource request (CPU, Memory, IP, etc.) against its capacity database for free resource availability.

5-6. The NFVO can otherwise optionally do resource reservation for the requested resources by using the Create Resource Reservation operation over the Virtualised Resources Management interface.

7. The NFVO grants the scale-out operation of the VNF to the VNF Manager and sends back sufficient information to further execute the scaling operation.

8. The VNF Manager sends the request to create and start the VMs as appropriate and as instructed by the NFVO, sending VIM Identifier and VMs parameters using the operations Allocate Resource or Update Resource or Scale Resource of the Virtualised Resources Management interface.

9. The VIM creates and starts the VMs and the relevant networking resources, then acknowledges successful operation to the VNF Manager.

10. VNF Manager configures VNF data specific for VNF instantiation using the add/create/set config object operations of the VNF Configuration interface.

11. VNF Manager notifies the EM that an existing VNF is updated with additional capacity using the VNF Lifecycle Change Notification interface.

12. EM and VNF Manager update the VNF as managed device.

13. EM configures the VNF with application specific parameters.

14. VNF Manager reports successful VNF expansion to the NFVO using the VNF Lifecycle Change Notification interface. The NFVO now is aware that the new VNF configuration is instantiated in the infrastructure.

15. The NFVO maps the VNF to the proper VIM and resource pool.

B.4.4.2 EM initiated VNF expansion

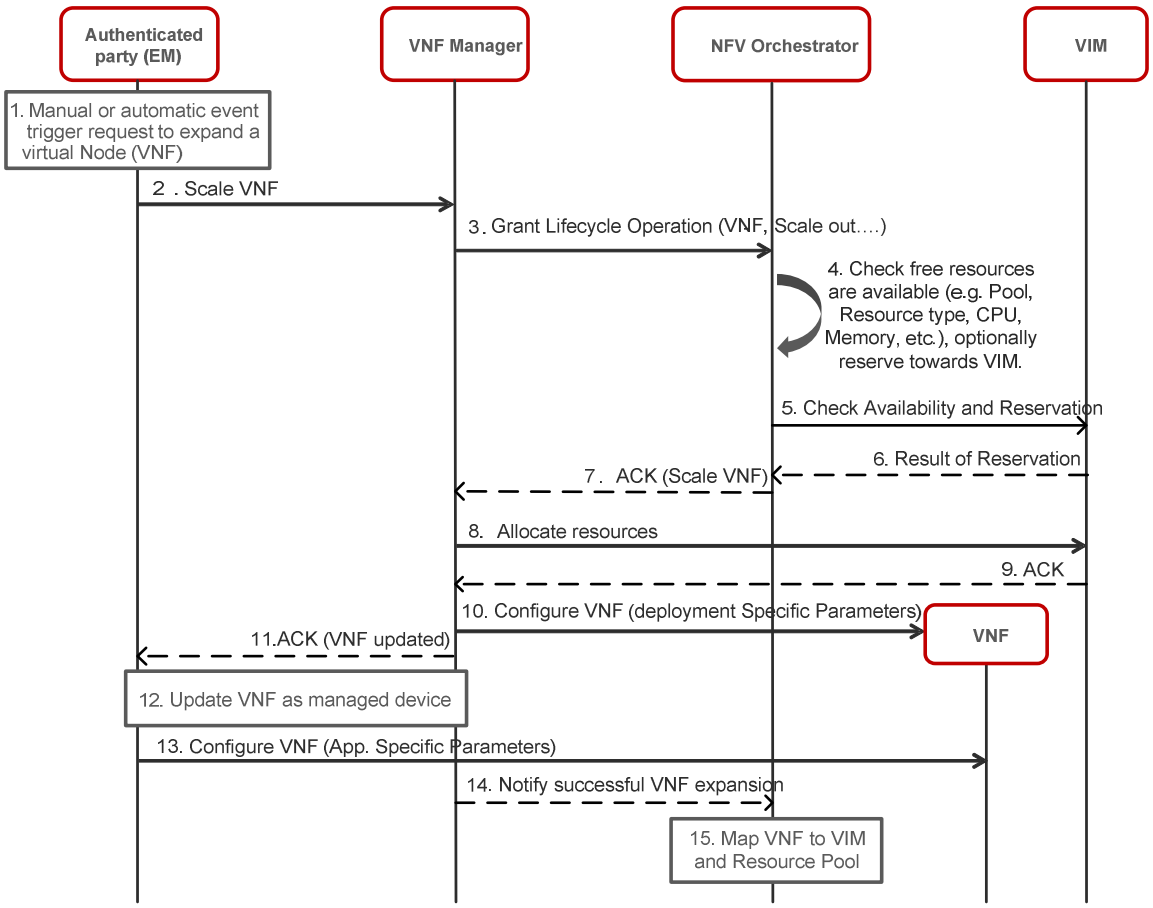


Figure B.14: EM initiated VNF expansion flow

NOTE: This flow is valid, when the EM consumes the VNF Lifecycle Management interface exposed by the VNFM over the Ve-Vnfm reference point.

The main steps for the EM initiated VNF expansion are:

1. Manual Operator's request or automatic event to expand the capacity of a virtual node (VNF).

2. EM requests capacity expansion to the VNF Manager using the operation Scale VNF of the VNF Lifecycle Management interface.

3. The VNF Manager requests granting to the NFVO for the VNF expansion based on the specifications listed in the VNFD (CPU, Memory, IP, etc.) using the operation Grant Lifecycle Operation of the VNF Lifecycle Operation Granting interface.

4. The NFVO takes scaling decision and checks resource request according to the VNF template (CPU, Memory, IP, etc.) against its capacity database for free resource availability.

5-6. The NFVO can otherwise optionally do resource reservation for the requested resources by using the Create Resource Reservation operation over the Virtualised Resources Management interface.

7. The NFVO grants the scale-out operation of the VNF to the VNF Manager and sends back sufficient information to further execute the scaling operations.

8. The VNF Manager sends the request to create and start the VMs as appropriate and as instructed by the NFVO, sending VIM Identifier and VMs parameters using the operations Allocate Resource or Update Resource or Scale Resource of the Virtualised Resources Management interface.

9. The VIM creates and starts the VMs and the relevant networking resources, then acknowledges successful operation to the VNF Manager.

10. VNF Manager configures VNF data being specific for VNF instantiation using the add/create/set config object operations of the VNF Configuration interface.
11. VNF Manager notifies to the EM that an existing VNF has been updated as requested with additional capacity.
12. EM and VNF Manager update the VNF as a managed device.
13. EM configures the VNF with application specific parameters.
14. VNF Manager reports successful VNF expansion to the NFVO using the VNF Lifecycle Change Notification interface. The NFVO now is aware that the new VNF configuration is instantiated in the infrastructure.
15. The NFVO maps the VNF to the proper VIM and resource pool.

B.4.4.3 Automatic VNF contraction triggered by VNF performance measurement results

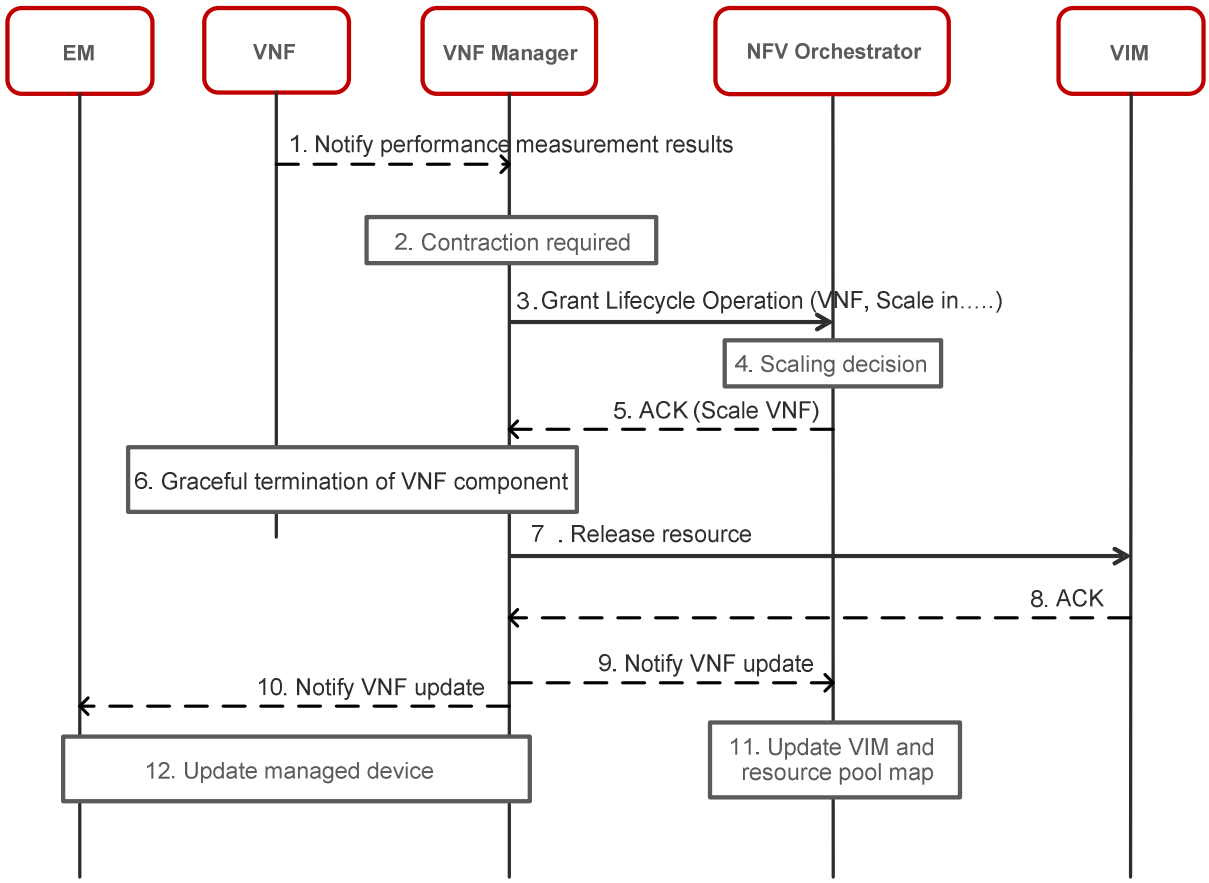


Figure B.15: Automatic VNF contraction flow triggered by VNF performance measurement results

VNF contraction refers to the removal of capacity from a deployed VNF. Contraction can result in scale-in of a VNF by removing VNFCs to free resources in the NFVI, or can result in scale-down of computing and memory resource from existing VNF/VNFCs. VNF contraction can be controlled by an automatic process or can be manually triggered by operator intervention, it typically occurs upon load conditions relaxes below a configured threshold. The following flow achieves automatic VNF contraction with the scale in example.

The main steps for the automatic VNF contraction are:

1. The VNF Manager collects measurement results from the VNF (Application Specific) using the operation Notify or Get performance measurement results of the VNF Performance Management interface.
2. VNF Manager detects a capacity release opportunity and triggers contraction (release of resources).

3. The VNF Manager requests validation to the NFVO for the VNF contraction based on proper template (CPU, Memory, IP, etc.) using the operation Grant Lifecycle Operation of the VNF Lifecycle Operation Granting interface.
4. The NFVO takes a scaling decision (e.g. based on configured policies).
5. The NFVO grants the scale-in operation of the VNF to the VNF Manager.
6. VNF gracefully terminates a VNF component (i.e. without affecting the ongoing service).
7. Once the application is shut down (no more traffic is handled), the VNF Manager requests deletion of the VM(s) to the VIM using the operation Release Resource of the Virtualised Resources Management interface.
8. The VIM releases resources and acknowledges to the VNF Manager.
9. VNF Manager reports successful VNF contraction to the NFVO using the VNF Lifecycle Change notification interface. The NFVO now is aware that resources have been released by the VIM.
10. VNF Manager notifies the EM that an existing VNF is updated with capacity release using the VNF Lifecycle Change Notification interface.
11. The NFVO updates the proper VIM and resource pool map.
12. EM and VNF Manager update the VNF as a managed device.

B.4.4.4 EM initiated VNF contraction

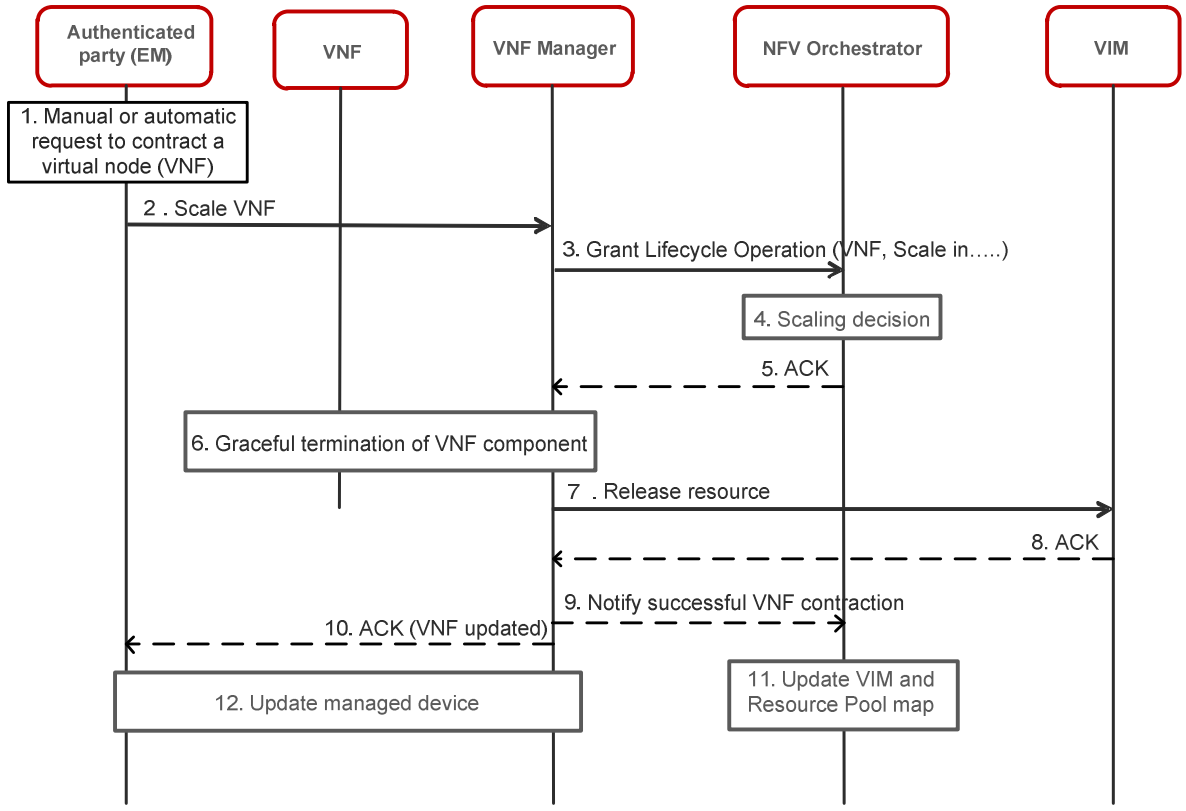


Figure B.16: EM initiated VNF contraction flow

NOTE: This flow is valid, when the EM consumes the VNF Lifecycle Management interface exposed by the VNFM over the Ve-Vnfm reference point.

The main steps for the EM initiated VNF contraction are:

1. Manual Operator's request or automatic event to contract the capacity of a virtual node (VNF).
2. EM requests capacity release for the VNF to the VNF Manager using the operation Scale VNF of the VNF Lifecycle Management interface.
3. The VNF Manager requests validation to the NFVO for the VNF Contraction based on the specifications listed in the VNFD (CPU, Memory, IP, etc.) using the operation Grant Lifecycle Operation of the VNF Lifecycle Operation Granting interface.
4. The NFVO takes a scaling decision (e.g. based on configured policies).
5. The NFVO grants the scale-in operation of the VNF to the VNF Manager.
6. VNF gracefully terminates a VNF component (i.e. without affecting the ongoing service).
7. Once the application is shut down (no more traffic is handled), the VNF Manager requests deletion of the VM(s) to the VIM using the operation Release Resource of the Virtualised Resources Management interface.
8. The VIM releases resources and acknowledges to the VNF Manager.
9. VNF Manager reports successful VNF contraction to the NFVO using the VNF Lifecycle Change Notification interface. The NFVO now is aware that resources have been released by the VIM.
10. VNF Manager acknowledges to the EM that an existing VNF has been updated as requested with capacity release.
11. The NFVO updates the proper VIM and resource pool map.
12. EM and VNF Manager update the VNF managed device.

B.5 VNF instance termination flows

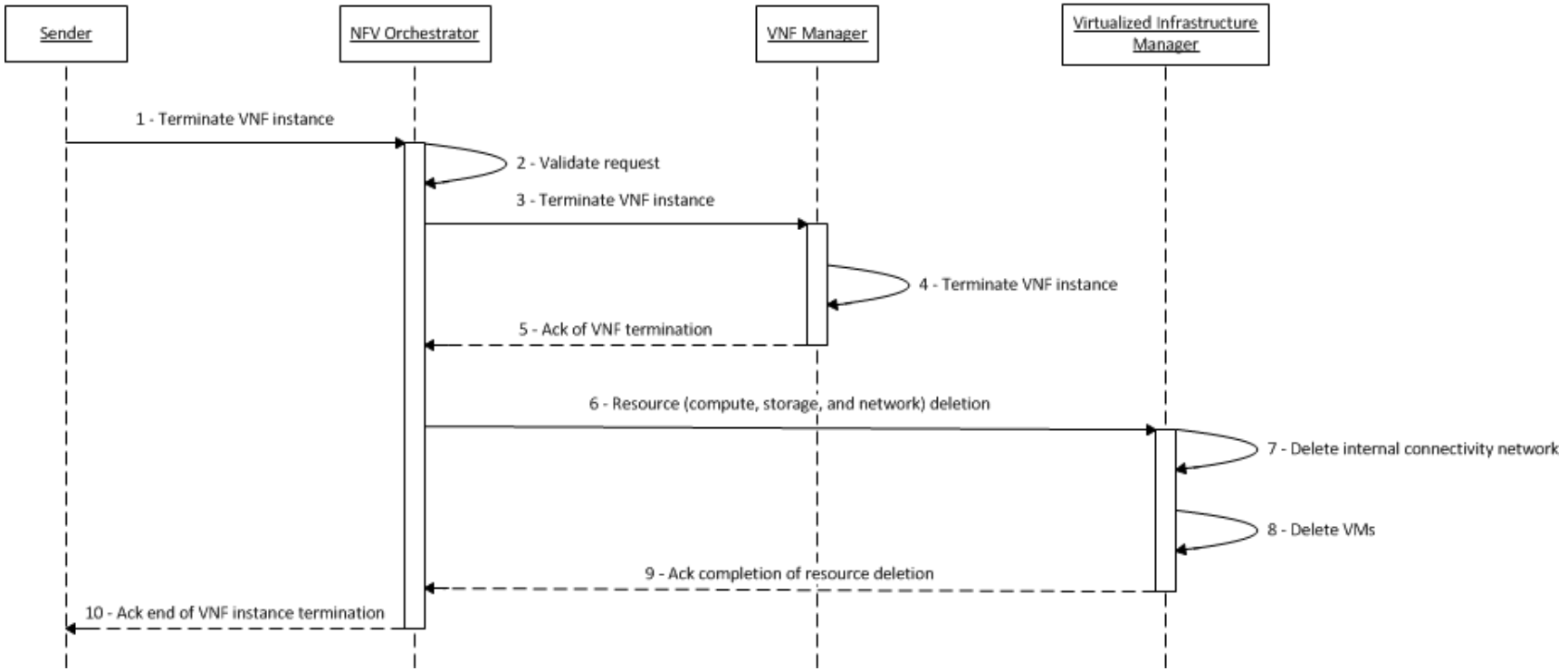


Figure B.17: VNF instance termination message flow

NFVO receives a request to terminate an existing VNF instance. This request might be sent from an OSS, decommissioning of a VNF or part of an order, or might come from the VNF Manager when the need to terminate a VNF is detected by VNF Manager itself or by the EM.

The main steps for VNF instance termination are:

1. NFVO receives a request to terminate an existing VNF instance using the operation Terminate VNF of the VNF Lifecycle Management interface.
2. NFVO validates the request. It verifies the validity of the request (including sender's authorization) and verifies that the VNF instance exists.
3. NFVO calls VNF Manager to terminate of the VNF using the operation Terminate VNF of the VNF Lifecycle Management interface.
4. VNF Manager terminates the VNF. This step can include a graceful shutdown of the VNF possibly in coordination with other management entities or the VNF itself.
5. Once the VNF is terminated, VNF Manager acknowledges the completion of the VNF termination back to the NFVO.
6. Using resource information kept for this VNF instance, NFVO requests deletion of resources (compute, storage and network) used by the various VDUs of the VNF instance using the operation Release Resource of the Virtualised Resources Management interface.
7. VIM deletes the internal connectivity network.
8. VIM deletes the compute (VMs) and storage resources of the various VDUs of the VNF instance.
9. Acknowledgement of completion of resource release back to NFVO.
10. NFVO acknowledges the completion of the VNF instance termination using the VNF Lifecycle Change Notification interface.

If the termination request is issued by the VNF Manager, then steps 1 to 3 are optional. If the termination request is issued by the EM, Step 1 Terminate VNF Instance will be sent to VNF Manager, then, forwarded to NFV Orchestrator. Steps 1 to 3 are optional.

B.6 NFV fault management

This clause provides examples of operational flows for fault management. While such flows apply to each VNF lifecycle, their scope is broader than the lifecycle of a single VNF instance, because certain faults can impact multiple VNFs and multiple NSs, and fault information can be further processed for purposes different than the lifecycle of the VNF (e.g. data analytics, capacity/inventory management, SLA management, etc.).

The concept of fault and performance management is described in clause 4.5.

Fault information can be the result of several different sources of faults: physical infrastructure (i.e. physical NFVI compute, storage, and networking related faults); virtualised infrastructure (e.g. VM-related faults), and application logic (i.e. VNF instance related faults).

When fault information related to the same primary cause is issued by some or all of those multiple sources, it is expected to be correlated. In the NFV-MANO architecture, such correlation could happen in multiple places: the NFVO, the VNF Manager, the EM and/or some OSS; the NFV architectural framework has the flexibility to support any of these alternatives, as well as combinations of these alternatives. Once correlation point(s) are chosen, other functional blocks will forward the fault information to the targeted correlation point.

While multiple alternatives may be possible, it is not the intent of this clause to document all possible fault management operational flows. The definition of the interfaces and policies controlling such interfaces allows for a large variety of alternative flows to be implemented, as per Service Provider and vendor implementation agreements.

In this clause, a single fault management flow alternative is presented. The flow illustrates collection of fault information from multiple sources, different fault correlation points, and different fault resolution/correction points. The flow is to be understood as a snapshot in the continuous process of fault management, which is not initiated by an on-demand trigger, but rather is a continuous cycle of monitoring functional blocks for fault information and reacting by processing that information.

The sequence diagram below shows an example of NFV fault management with multiple options, depending on the fault correlation point, and the fault resolution point. For simplification, VIM represents collectively the source of fault information for all virtualised infrastructure resources.

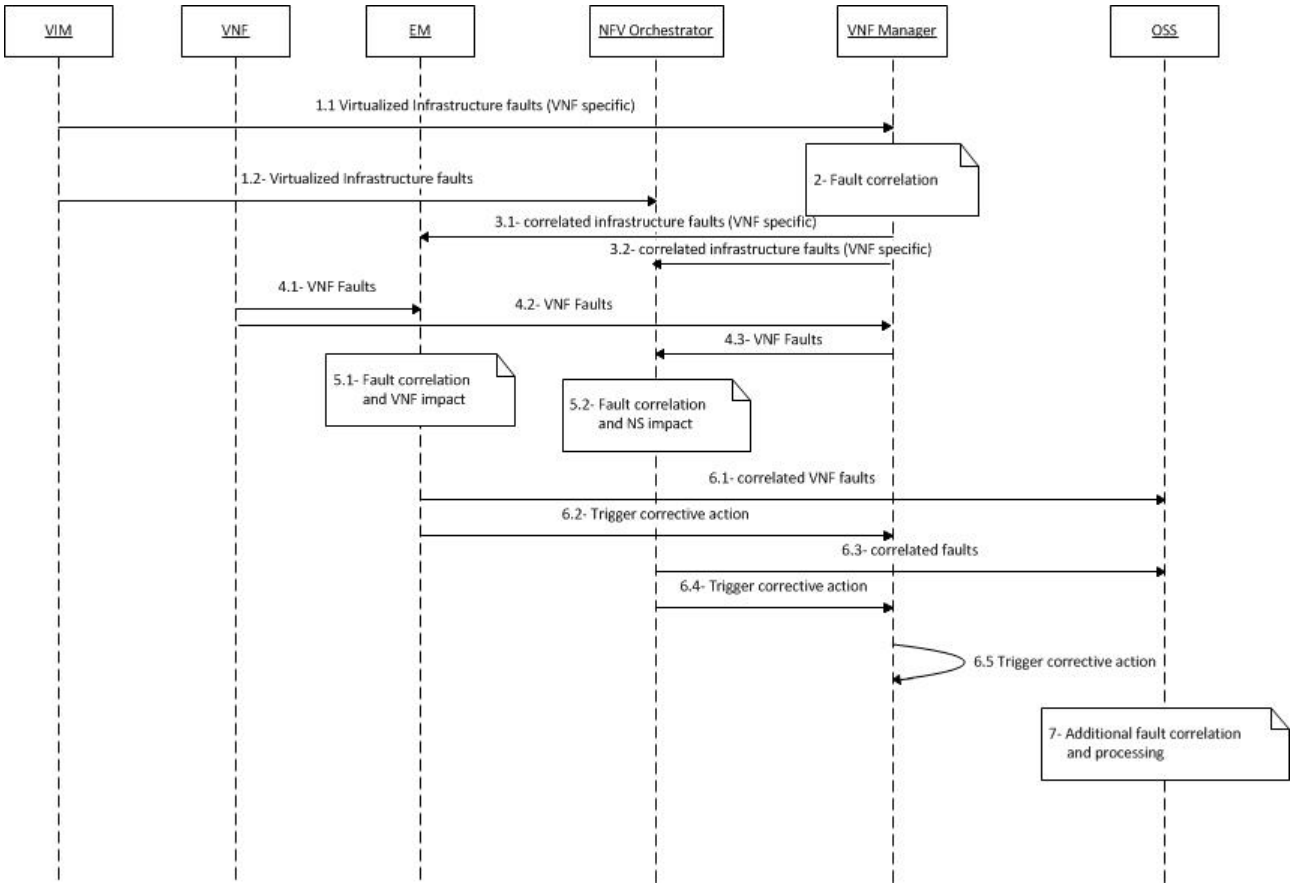


Figure B.18: NFV fault management flow

The main steps of the NFV fault management flow are:

1. Virtualised infrastructure faults (1.1) related to a specific VNF instance are received by the VNF Manager for that VNF instance. Infrastructure faults not related to a specific VNF can also be received directly by the NFVO (1.2 for virtualised infrastructure faults).

2. VNF Manager can perform its own fault correlation for certain selected events.

3. VNF Manager can forward correlated fault information regarding the VNF instance to different other fault correlation points, primarily EM (3.1) and NFVO (3.2).

4. VNF instance can send application layer faults to different correlation points, primarily the EM (4.1) and VNFM (4.2). VNFM can further forward the fault information to NFVO (4.3).

5. Fault correlation can happen at any of the fault correlation points identified. EM can perform fault correlation to determine the root cause and the impact on the VNF (5.1). NFVO can perform fault correlation to determine the root cause and the impact on the Network Service (5.2). To support some scenarios, the NFVO can just map infrastructure faults to VNF and/or NS, without correlating with VNF application layer faults, and forward the results to an OSS (see 6.3).

6. At the different fault correlation points, a fault resolution action can ensue. EM can forward correlated fault information regarding the VNF instance to an OSS (6.1). EM can trigger a corrective action request towards the VNF Manager (6.2). NFVO can forward correlated fault information to an OSS (6.3). NFVO can trigger a corrective action towards the VNF Manager (6.4). VNF Manager can itself trigger a corrective action if responsible for correlating certain events (6.5).

7. Additional fault correlation and processing can happen in the OSS.

NOTE 1: As described in this clause, there is not necessarily a single event correlation and/or resolution point in the NFV Framework. Event Correlation/Resolution can, and it is recommended that it is executed as soon as the event is fully understood, and what action to be taken is known. The danger is that an action can be taken prematurely and/or that conflicting actions can be taken in different places in the event management architecture.

EXAMPLE: In NFV the expectation is that certain event correlation and resolution will be performed in the VIM, other in the NFVO, other in the VNF Manager, other in the EMS, other in an NMS/OSS - depending on the nature of the event.

The corrective actions will be different, depending on event. Some corrective actions can in fact involve executing some of the VNF instance lifecycle management flows (e.g. VNF scaling, VNF termination) or some of the Network Service instance lifecycle management flows (e.g. Network Service termination).

NOTE 2: Some of the flows require the presence of a mechanism to set thresholds on particular metrics, and a mechanism to allow a functional block to selectively subscribe to event notifications sent by the VIM to the subscribed functional block, when the threshold regarding a specific metric related a virtualised resource is reached. It is also required by some flows that a VIM can also forward events triggered by physical infrastructure faults (e.g. NFVI-PoP down).

Annex C:
Network Service lifecycle management flows

C.1 Introduction

This annex represents a collection (non-exhaustive) of management flows related to the Network Service Lifecycle that are presented for information and illustration only and not for implementation purpose.

The following principles are used for all the flows in this clause:

- The NFVO is the single point of access for all requests from the OSS to simplify the interfacing.
- The NFVO handles lifecycles of Network Service and VNF Forwarding Graph.
- The VNF Manager handles VNF lifecycle from an application point of view.
- The NFVO has the end-to-end view of the resources being allocated across Network Services and VNFs by VNF Managers, so all requests for resource allocation transit through the NFVO.

All the flows in this clause are informative, representing best practices for each of the lifecycle operation and have the main goal of identifying needed interfaces as well as information needed on those interfaces.

At each step, in case of failure, an immediate return might happen. All the failure cases have not been shown on the sequence diagrams or on the text for sake of simplicity.

The dotted-line arrows represent the return path of the request or in some specific cases, a notification. The return path of a request can be a true reply in case of a request/reply exchange or a notification otherwise.

C.2 Network Service on-boarding flows

C.2.0 Use Case diagram

The use case diagram shown by figure C.1 provides the following use cases related to Network Services on-boarding:

- On-board Network Service Descriptor.
- Disable Network Service Descriptor.
- Enable Network Service Descriptor.
- Update Network Service Descriptor.
- Query Network Service Descriptor.
- Delete Network Service Descriptor.

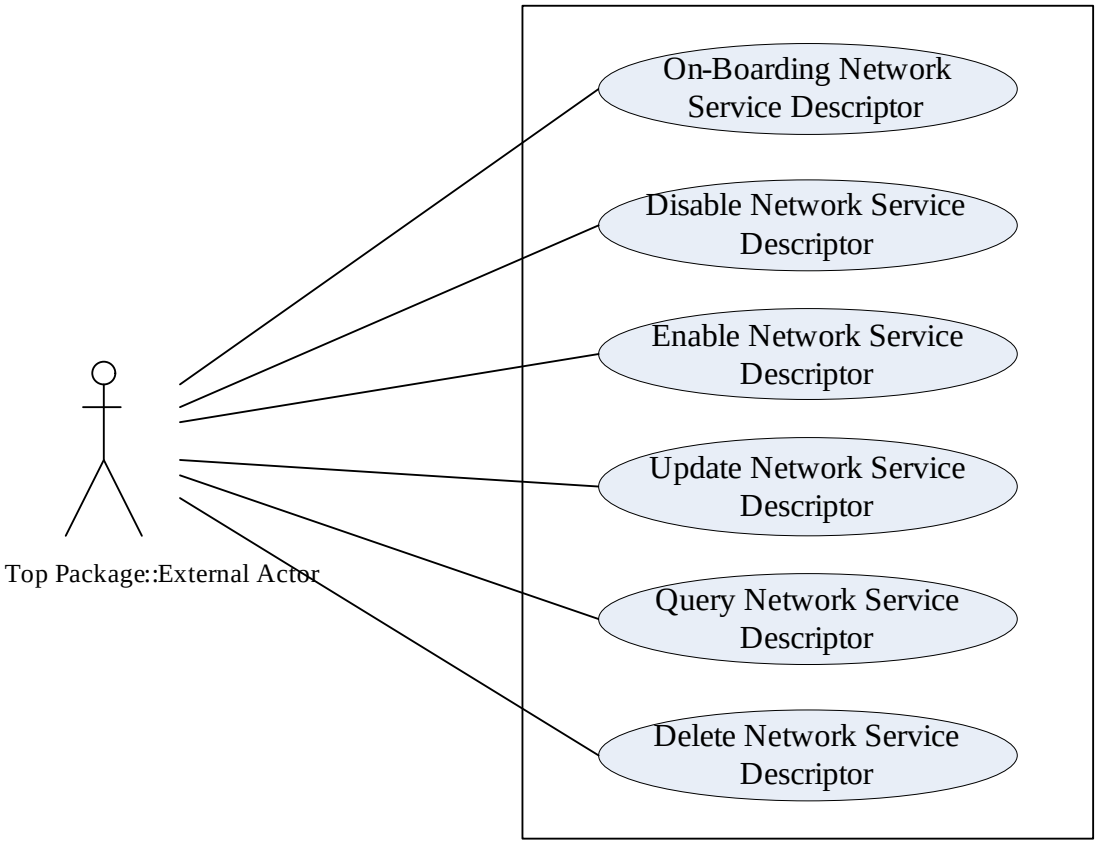


Figure C.1: Use case diagram for Network Service on-boarding

C.2.1 On-board Network Service Descriptor flow

Network Service on-boarding refers to the process of submitting a Network Service Descriptor (NSD) to the NFVO to be included in the catalogue.

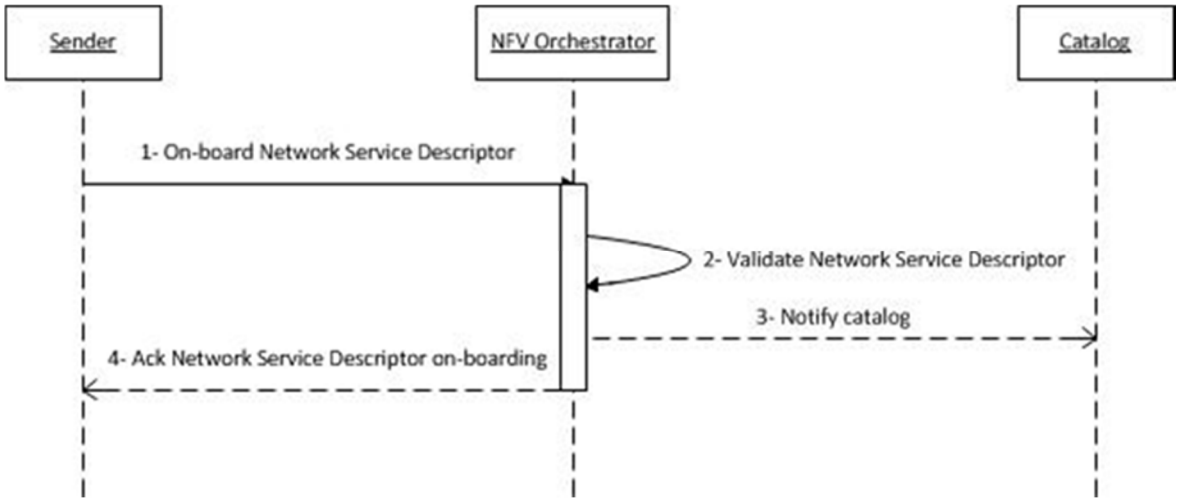


Figure C.2: Network Service Descriptor on-boarding message flow

Note that the entity Sender above could be any entity that sends the Network Service Descriptor to the NFVO on behalf of the operator; it can be the operator itself or a service design application or a vendor or an entity in the SP domain.

The main steps for Network Service on-boarding are:

1. Network Service Descriptor is submitted to the NFVO for on-boarding the Network Service using the operation On-board Network Service Descriptor of the Network Service Descriptor interface.

2. The NFVO processes the Network Service Descriptor including (not limited to):

   a. Validate the integrity and authenticity of NSD, The security information needed for validation can be provided as part of NSD.

   b. Checking presence of VNF Package for the VNFs that are part of the Network Service.

   c. Checking for the existence of mandatory elements.

   d. Checking the presence of needed external interfaces required to provide the Network Service, in the VNF Descriptors of the VNFs that are part of the Network Service.

3. The NFVO notifies the catalogue for insertion of the Network Service Descriptor in the catalogue. The catalogue will insert the NSD with its version number.

4. The NFVO acknowledges the Network Service on-boarding.

C.2.2 Disable Network Service Descriptor flow

Disabling a Network Service Descriptor refers to the process of marking a Network Service Descriptor (NSD) as disabled in the catalogue, so that it is not possible to instantiate Network Services with it any further.

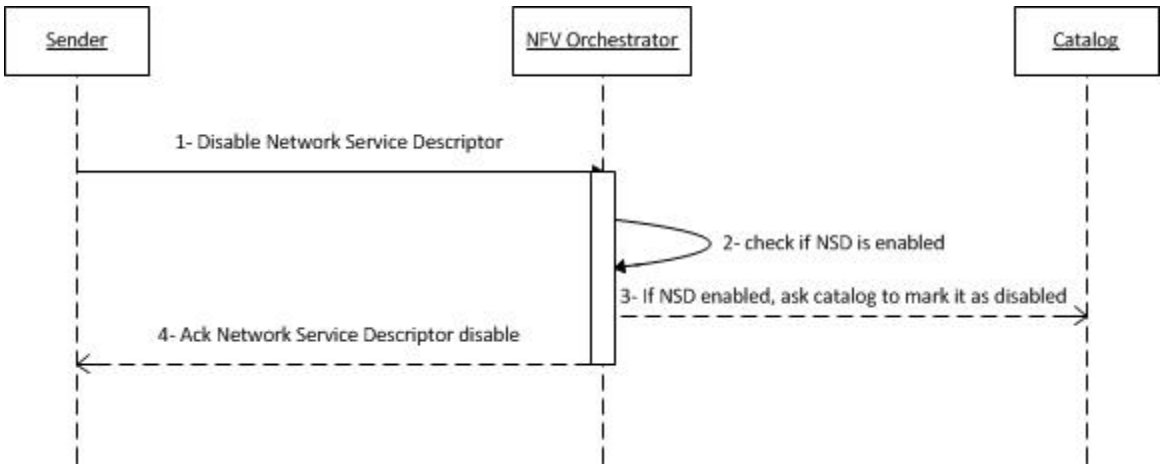


Figure C.3: Network Service Descriptor disable message flow

The main steps for disabling a Network Service Descriptor are:

1. A request to disable a Network Service Descriptor is submitted to the NFVO using the operation Disable Network Service Descriptor of the Network Service Descriptor interface.

2. The NFVO processes the request and checks if the NSD exists and is enabled.

3. If the NSD is enabled, The NFVO notifies the catalogue to disable the Network Service Descriptor in the catalogue.

4. The NFVO acknowledges the Network Service Descriptor disable request.

C.2.3 Enable Network Service Descriptor flow

Enabling a Network Service Descriptor refers to the process of marking a Network Service Descriptor (NSD) as enabled in the catalogue, so that it can be used to instantiate Network Services again.

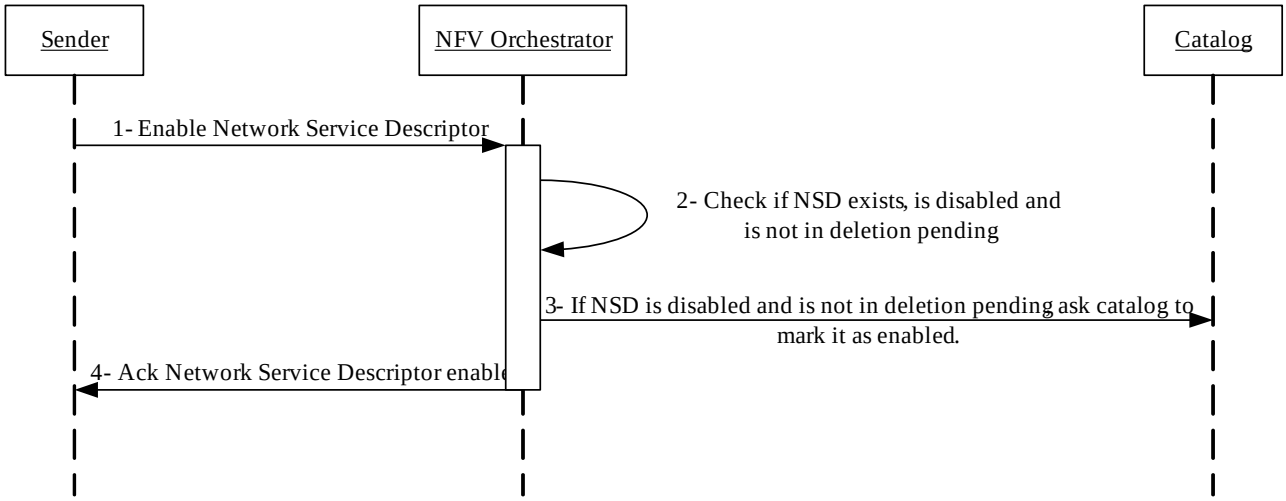


Figure C.4: Network Service Descriptor enable message flow

The main steps for enabling a Network Service Descriptor are:

1. A request to enable a disabled Network Service Descriptor is submitted to the NFVO using the operation Enable Network Service Descriptor of the Network Service Descriptor interface.
2. The NFVO processes the request and checks if the NSD exists, is disabled, and is not marked as deletion pending. Optionally, the NFVO can validate the integrity of the NSD.
3. If the NSD is disabled and is not marked as deletion pending, The NFVO notifies the catalogue to enable the Network Service Descriptor in the catalogue.
4. The NFVO acknowledges the Network Service Descriptor enable request.

C.2.4 Update Network Service Descriptor flow

Network Service Descriptor update refers to the process of submitting a modified Network Service Descriptor (NSD) to the NFVO to be included in the catalogue. This update might include creating/deleting new VNFFGs and/or new VLDs.

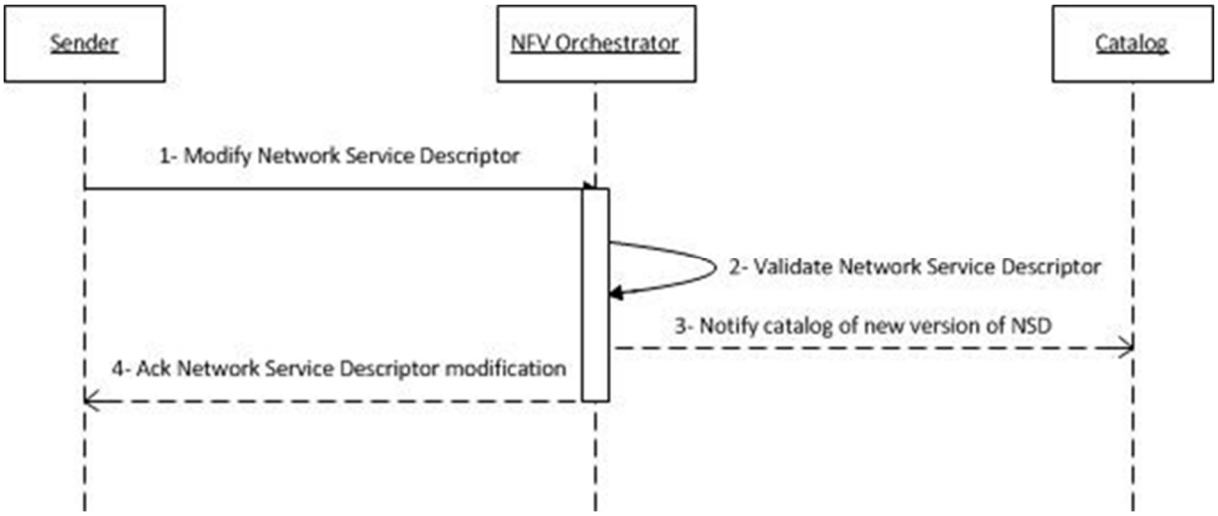


Figure C.5: Network Service Descriptor update message flow

The main steps for Network Service Descriptor update are:

1. Modified Network Service Descriptor is submitted to the NFVO for the Network Service using the operation Update Network Service Descriptor of the Network Service Descriptor interface.

2. The NFVO checks if a NSD exists already for this NS and is not marked as deletion pending and if so, processes the Network Service Descriptor including (not limited to):

   a. Validate the integrity and authenticity of NSD, The security information needed for validation can be provided as part of NSD.

   b. Checking presence of VNF Package for the VNFs that are part of the Network Service.

   c. Checking for the existence of mandatory elements.

   d. Checking the presence of needed external interfaces required to provide the Network Service, in the VNF Descriptor of the VNFs that are part of the Network Service.

3. The NFVO notifies the catalogue for insertion of a new version of the Network Service Descriptor in the catalogue.

NOTE: Some existing Network Services might still use the previous version of the NSD, so updating it will create a new version in the catalogue.

4. The NFVO acknowledges the Network Service Descriptor update.

C.2.5 Query Network Service Descriptor flow

Querying Network Service Descriptor allows returning from the catalogue the information of the Network Service Descriptors, including any related VNFFGD and VLD.

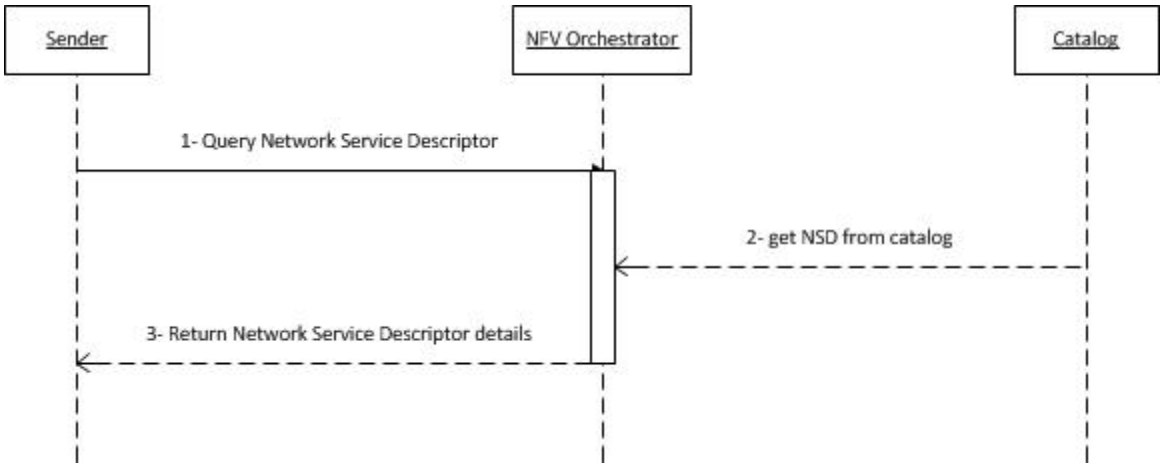


Figure C.6: Network Service Descriptor query message flow

The main steps for Network Service Descriptor query are:

1. The NFVO receives a query request for the Network Service Descriptor using the operation Query Network Service Descriptor of the Network Service Descriptor interface. One or more filter parameters can be included in the Query operation to filter the Network Service Descriptors.

2. The NFVO gets from the catalogue the Network Service Descriptors satisfying the filter conditions in detail, including any related VNFFGD and VLD.

3. The NFVO returns the Network Service Descriptor details.

C.2.6 Delete Network Service Descriptor flow

Network Service Descriptor deletion refers to the process of asking the NFVO to delete a Network Service Descriptor (NSD) from the catalogue.

Note that there might be multiple versions of the NSD, but the assumption is that the delete request remove all versions, otherwise, it fails.

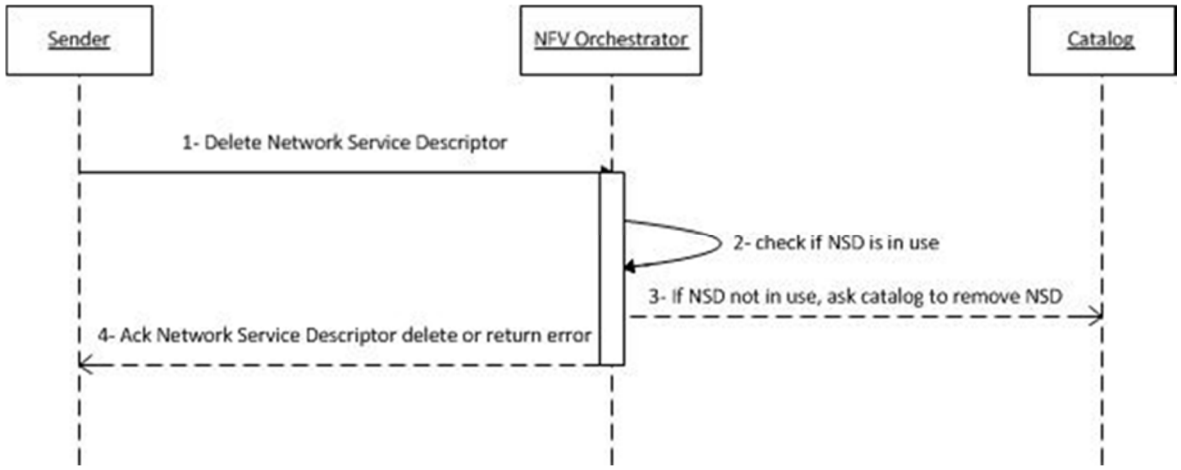


Figure C.7: Network Service Descriptor deletion message flow

The main steps for Network Service deletion are:

1. Request to delete Network Service Descriptor is submitted to the NFVO using the operation Delete Network Service Descriptor of the Network Service Descriptor interface.

2. The NFVO checks if the Network Service Descriptor is disabled and in use. If NSD is not disabled, the request is rejected.

3. If NSD is disables and not in use, then ask the catalogue to remove all versions of the NSD. The catalogue will then remove them.

NOTE 1: If NSD is disabled and still in use, the NFVO set the NSD in deletion pending.

4. The NFVO acknowledges Network Service Descriptor deletion request.

NOTE 2: If the NSD is in deletion pending, the NFVO will later check if there is any NS instance using the NSD during the NS instance termination process. If there is no NS instance using the NSD any more, the NFVO asks the catalogue to remove the corresponding version(s) of the NSD.

C.3 Network Service instantiation flows

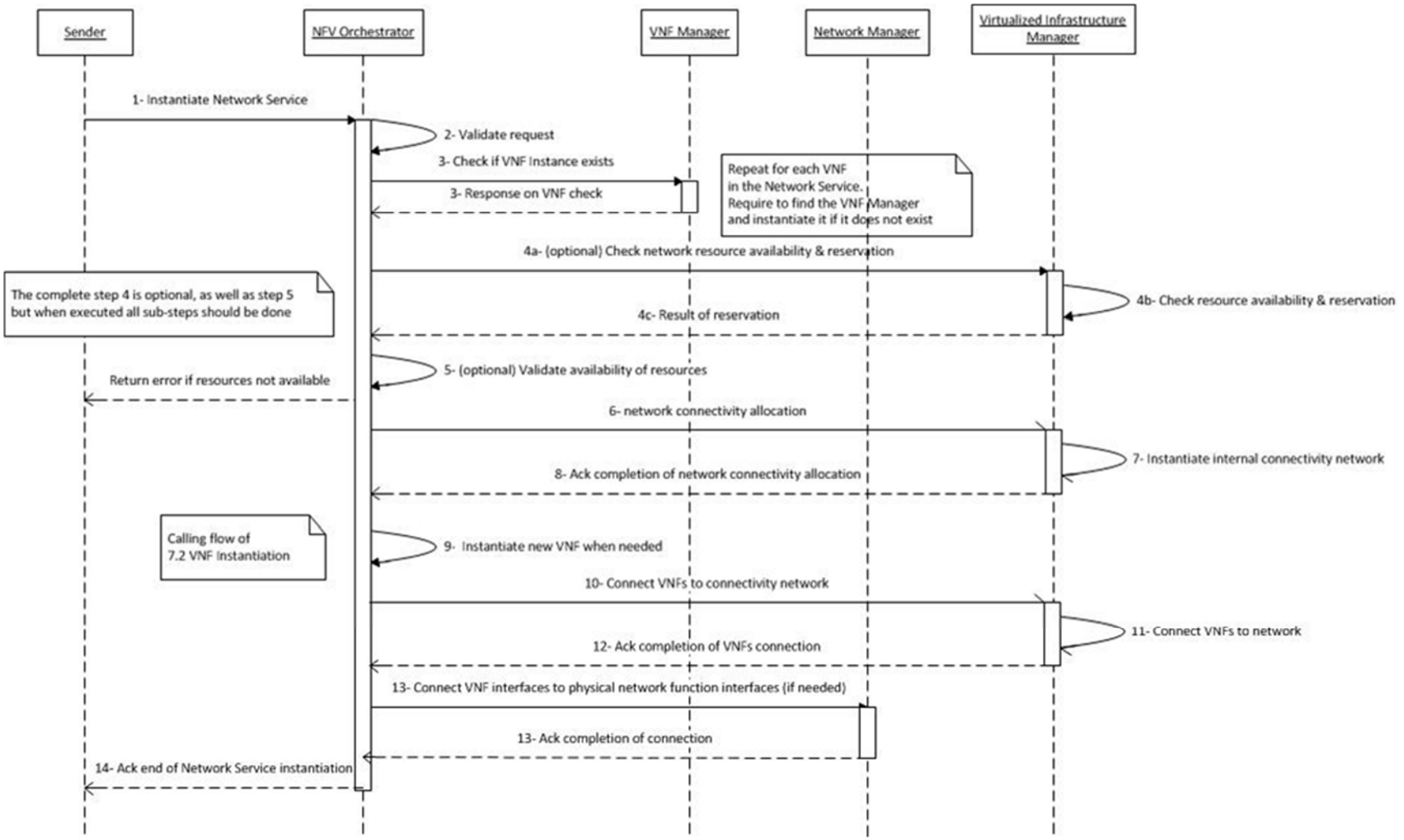


Figure C.8: Network Service instantiation message flow

NFVO receives a request to instantiate a new Network Service. This request might come from an OSS, receiving an order for Network Service instantiation.

The NFVO is the single point of access for Network Service instantiation simplifying interfacing with OSS.

When creating a Network Service, several variants might be possible:

a) No VNF instance needed for this Network Service exists, so the Network Service instantiation will include the instantiation of the needed VNF instances.

b) All needed VNF instances might already be instantiated: in this case, Network Service instantiation would only deal with the interconnection of the VNF instances.

c) A combination of the above where some VNF instances might exist and some might need to be created and some of the network connectivity between the VNFs might already exist, and it only requests to be extended.

The instantiation flows below covers variants (c), thus checking if the needed VNF instances exist.

No assumption is made either on whether VNFs can be shared or not between Network Services.

The flow below assumes a single VNF Forwarding Graph for the Network Service. If multiple VNF Forwarding Graphs are defined for a Network Service, then steps 3 to 13 would be repeated for each VNF Forwarding Graph.

The main steps for Network Service instantiation are:

1. NFVO receives a request to instantiate a new Network Service using the operation Instantiate Network Service of the Network Service Lifecycle Management interface.

2. NFVO validates the request, both validity of request (including validating that the sender is authorized to issue this request) and validation of the parameters passed for technical correctness and policy conformance. In case the Network Service contains multiple VNF Forwarding Graphs, policy rules might result in only a subset being valid for a given Network Service instance.

3. For each VNF instance needed in the Network Service, the NFVO checks with the VNF Manager if an instance matching the requirements exists already using the operation Query VNF of the VNF Lifecycle Management interface. If such a VNF instance exists, it will be used as part of the Network Service.

NOTE 1: It requires NFVO, if needed, to find the corresponding VNF Manager and to instantiate it if it does not exist. For sake of simplicity, those steps have been collapsed.

4. Optionally, NFVO runs a feasibility check of the VNF interconnection setup. Steps 3 to 5 constitute the feasibility check of the request. Step 3 consists of the following sub-steps:

    a. NFVO requests to VIM availability of network resources needed for the VNF Interconnection and reservation of those resources using the operation Create Resource Reservation of the Virtualised Resources Management interface. Note that some of the network connectivity between the VNFs might already exist.

    b. VIM checks the availability of network resources needed for the VNF Interconnection and reserves them.

    c. VIM returns result of reservation back to NFVO.

5. Optionally, once the list of VNF instances to be provisioned is known and assuming it is not empty, the NFVO validates if resources are available to honour the VNF instantiation requests and if so, reserves them using the operation Create Resource Reservation of the Virtualised Resources Management interface.

6. NFVO requests from VIM instantiation of the network connectivity using the operations Allocate Resource or Update Resource from the Virtualised Resources Management interface. Note that some of the network connectivity between the VNFs might already exist and might only need to be extended.

7. VIM instantiates the connectivity network needed for the Network Service.

8. VIM acknowledges completion.

9. Assuming the list of VNF instances to be provisioned is not empty, the NFVO instantiate the new VNF instances needed. This is done by calling the "Instantiate VNF" request using the operation Instantiate VNF of the VNF Lifecycle Management interface as illustrated in clause 7.2, VNF Instantiation.

10. Once all VNF instances are available and for the VNFs not already connected, NFVO requests VIM to connect them together using the operations Allocate Resource or Update Resource from the Virtualised Resources Management interface. It includes:

    a. Requesting VIM to connect external interfaces of each VNFs.

    b. Requesting VIM to attach needed VDUs (VMs) to the Network Service's connectivity network.

11. VIM connects needed VDUs (VMs) to the connectivity network.

12. VIM acknowledges completion.

13. If needed, NFVO requests Network Manager to connect VNF external interfaces to physical network function interfaces.

NOTE 2: The Network Manager can be an OSS, an NMS, an EM, or a WIM.

14. NFVO acknowledges the completion of the Network Service instantiation.

NOTE 3: Connection to physical network functions is managed by the Resource Orchestration function of the NFVO.

C.4 Network Service instance scaling

C.4.1 Network Service instance scale-out

The NSD can be on boarded with 1 or more deployment flavours each having their resource requirements in terms of number of VNF instances, interconnectivity, links, etc.

The following example illustrates how Network Service instance scale-out works. The assumption is that 2. The Network Service has been instantiated based on a flavour-A(1 × VNF-A + 2 × VNF-B). At that point:

a) The NS can be scaled based on auto-scaling policy (in the NSD) to a different NS deployment flavour-B (2 × VNF-A + 2 × VNF-B, which is described in the same on boarded NSD).

b) The Sender can manually initiate the NS scaling from the instantiated flavour -A to the deployment flavour-B; For VNF-A it instantiates a new instance, and for VNF-B it scales those existing instances.

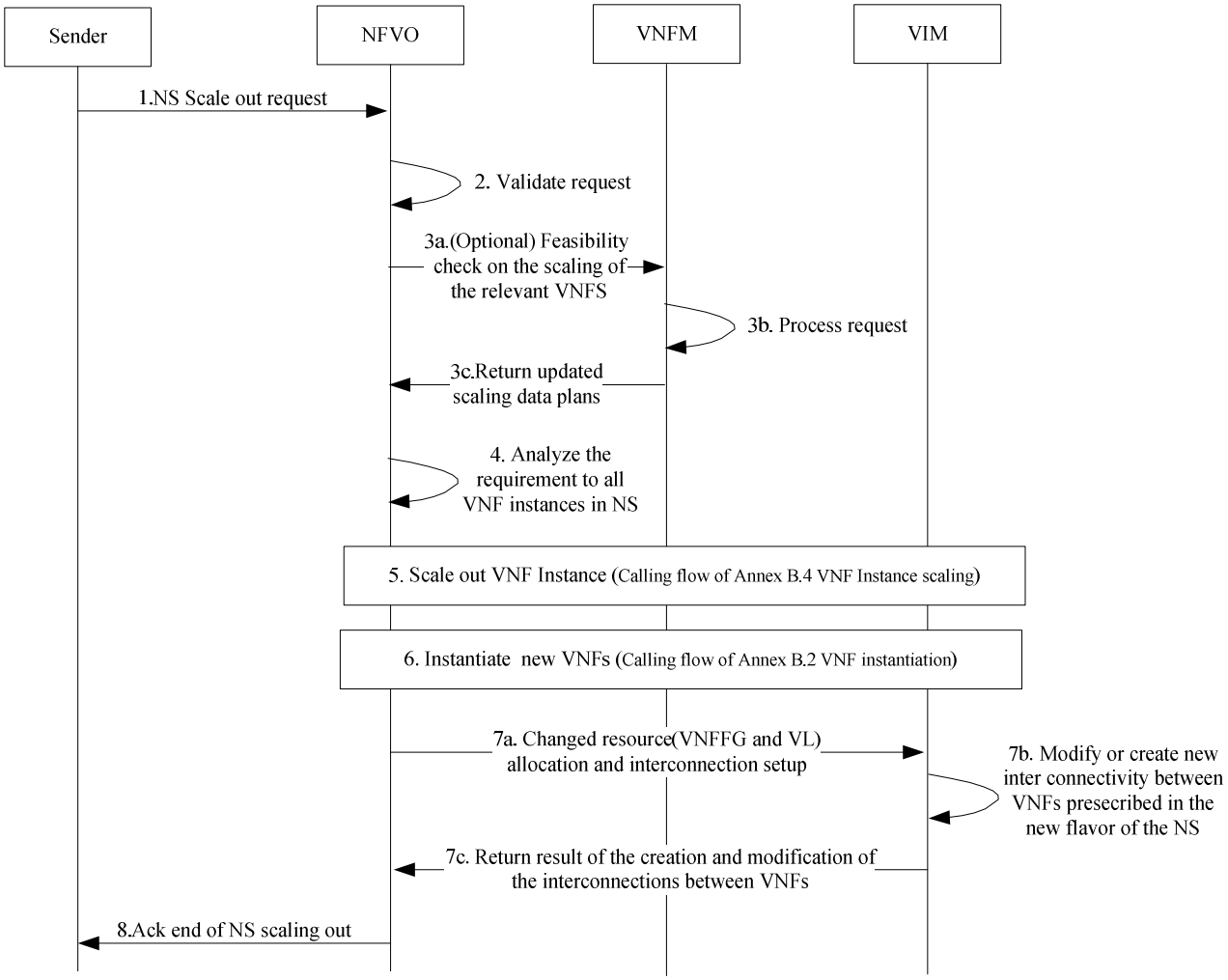


Figure C.9: Network Service scaling out message flow

NFVO receives a request to scale a Network Service instance. This request might come from an OSS, receiving an order for Network Service instance scaling.

The NFVO is the single point of access for Network Service lifecycle simplifying interfacing with OSS.

1. The Sender requests the NS to be scaled out to a new deployment flavour which is already present in the preloaded NSD.

2. NFVO validates the request, both validity of request (including validating that the sender is authorized to issue this request) and validation of the parameters passed for technical correctness and policy conformance. NFVO correlates the request with the on boarded NSD in the NS catalogue.

3. (Optional) Check feasibility:

   a) NFVO would check feasibility of scaling the relevant member VNFs to reflect the new flavour.

   b) Each VNF Manager would process this request and determine if the VNF can be scaled out to the new flavour.

   c) VNF Manager(s) returns the result of the feasibility check on the VNF scaling out.

4. Scaling-out a NS involves scaling-out its constituent VNFs. It is foreseen that a VNF can be scale-out in two different ways either by allocating more resources to VNF instance or by instantiating a new VNF instance. It is foreseen to have these preferences documented as scaling mechanism in related VNFDs. On receiving the scale-out request NFVO will identify, based on related NSD, the VNF instance(s) needed to be scale-out and the related VNF scaling mechanism. Depending on which scaling mechanism to perform, NFVO will either execute step 5 or step 6 or even both. The new VNF instance will be created (Step 6) in case the existing VNF Instance has already scaled-out to its maximum capabilities, e.g. due to their its ability of automatic scaling.

5. VNF instance scaling flow as provided in clause B.4.

6. VNF instantiation flow as provided in clause B.3.

7. The NFVO would request VIM to allocate the changed resources:

   a) NFVO would request VIM to allocate the changed resources (such as interconnectivity between VNFs required by the new deployment flavour as mandated by the VNFFGDs and VLDs).

   b) The VIM would allocate the interconnectivity accordingly.

   c) VIM would return the result of the operation to the NFVO.

8. The NFVO acknowledges the end of the scaling request back to the requester.

C.4.2 Network Service instance scale-in

The following example illustrates how Network Service instance scale-in works. The assumption is that the Network Service has been instantiated based on a flavour-B (2 × VNF-A + 2 × VNF-B). At that point:

a) The NS can be scaled based on auto-scaling policy (in the NSD) to a different NS deployment flavour-A (1 × VNF-A + 2 × VNF-B, which is described in the same on boarded NSD).

b) The Sender can manually initiate the NS scaling from the instantiated flavour -B to the deployment flavour-A; For VNF-A it terminates an existing instance, and for VNF-B it scales in those existing instances.

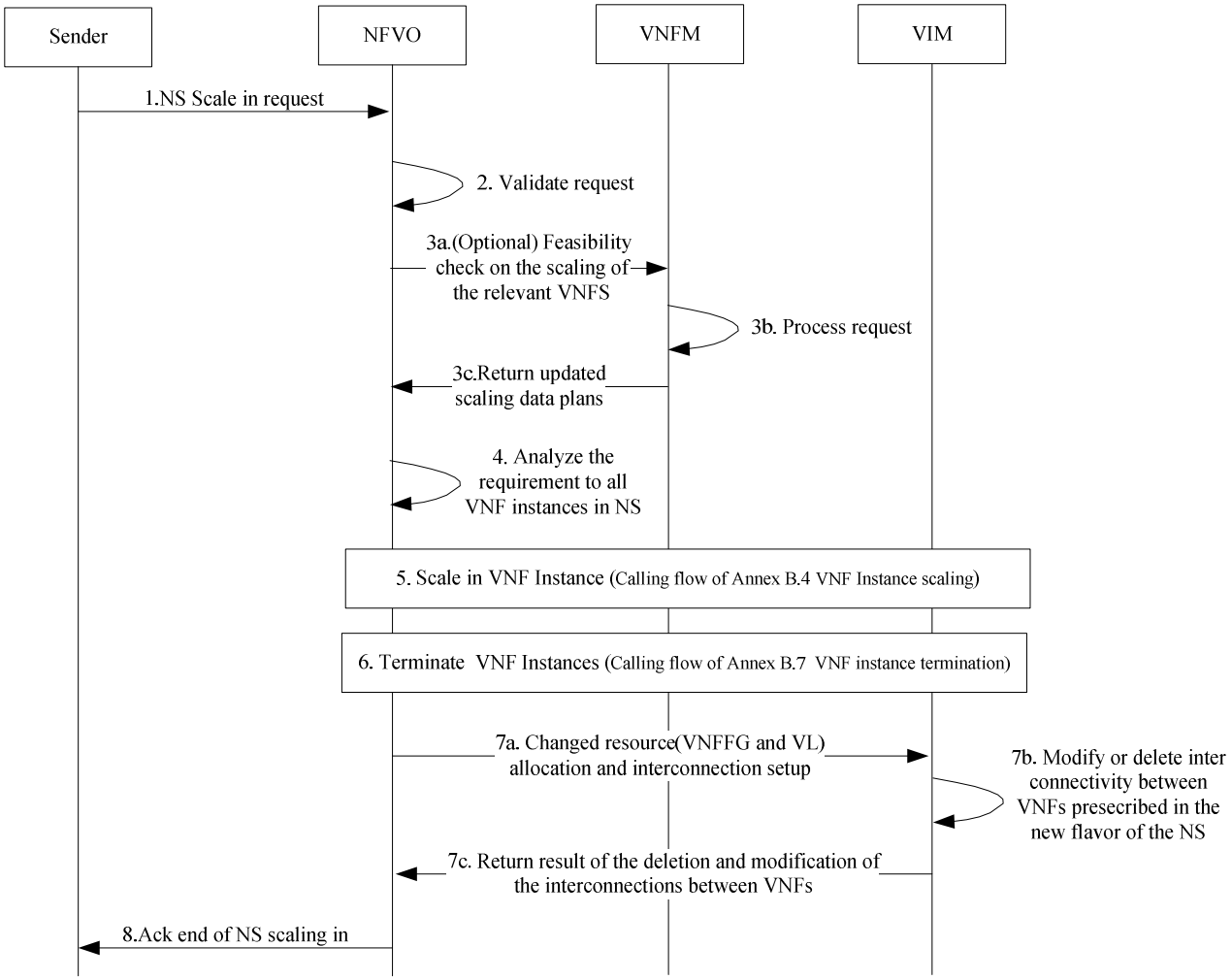


Figure C.10: Network Service scaling in message flow

NFVO receives a request to scale in a Network Service instance. This request might come from an OSS, receiving an order for Network Service instance scaling in.

The main steps for Network Service scaling in are:

1. The Sender requests the NS to be scaled in to a new deployment flavour which is already present in the preloaded NSD.

2. NFVO validates the request, both validity of request (including validating that the sender is authorized to issue this request) and validation of the parameters passed for technical correctness and policy conformance.

3. (Optional) Check feasibility:

   a) NFVO would check feasibility of scaling the relevant member VNFs to reflect the new flavour.

   b) Each VNF Manager would process this request and determine if the VNF can be scaled in to the new flavour.

   c) VNF Manager(s) returns the result of the feasibility check on the VNF scaling in.

4. Scaling-in a NS involves scaling-in its constituent VNFs. It is foreseen that a VNF can be scale-in in two different ways either by revoking allocated resources to VNF instance or by terminating an entire VNF instance. It is foreseen to have these preferences documented as scaling mechanism in related VNFDs. On receiving the scale-in request NFVO will identify, based on related NSD, the VNF instances needed to be scale-in and the related VNF scaling mechanism. Depending on which scaling mechanism to perform, NFVO will either execute step 5 or step 6 or even both. The existing VNF instance will be terminated (Step 6) in case it has already scaled-in to its minimum capabilities, e.g. due to its ability of automatic scaling.

5. VNF instance scaling flow as provided in clause B.4.
6. VNF instance termination flow as provided in clause B.5.
7. The NFVO would request VIM to modify or delete the changed resources:
   a) NFVO would request VIM to modify or delete the changed resources (such as interconnectivity between VNFs required by the new deployment flavour as mandated by the VNFFGDs and VLDs).
   b) The VIM would modify or delete the interconnectivity accordingly.
   c) VIM would return the result of the operation to the NFVO.
8. The NFVO acknowledges the end of the scaling request back to the requester.

C.5 Network Service instance update flows due to VNF instance modification

Network Service instance update due to VNF Instance modification is about replacing existing VNF Instances with new Instances by on boarding a new VNFD instantiating a VNF Instance out of it and then updating an existing VNF Forwarding Graph with the instantiated VNF for the Network Service.. As a new VNFD is required, the new Instance could differ in terms of deployment (e.g. NFVI Resource requirements, external interfaces) and operational (e.g. lifecycle management) characteristics of the VNF, basically all what can be present in VNFD can change. Since a VNF Instance modification can change the requirements/dependencies on other VNF, the process of modifications could require changing dependent VNFs prior to the target VNF.

Figure C.11 shows the before and after picture of this kind of Network Service instance update. It shows that VNF X, in NS A, is identified to be modified in terms of its storage requirements and the external interfaces. In order to honour the changed storage requirement the resources assigned to VNF Z are upgraded.

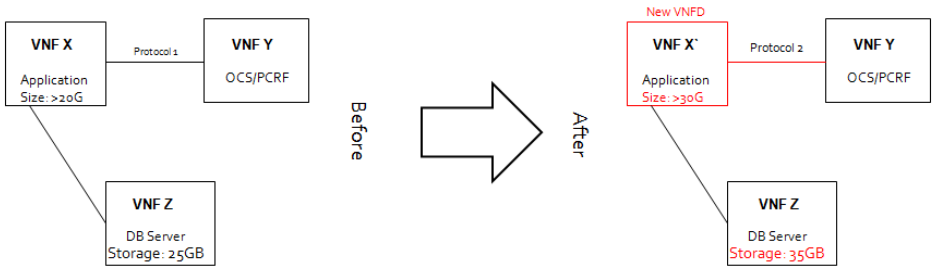


Figure C.11: Before and after a VNF instance update

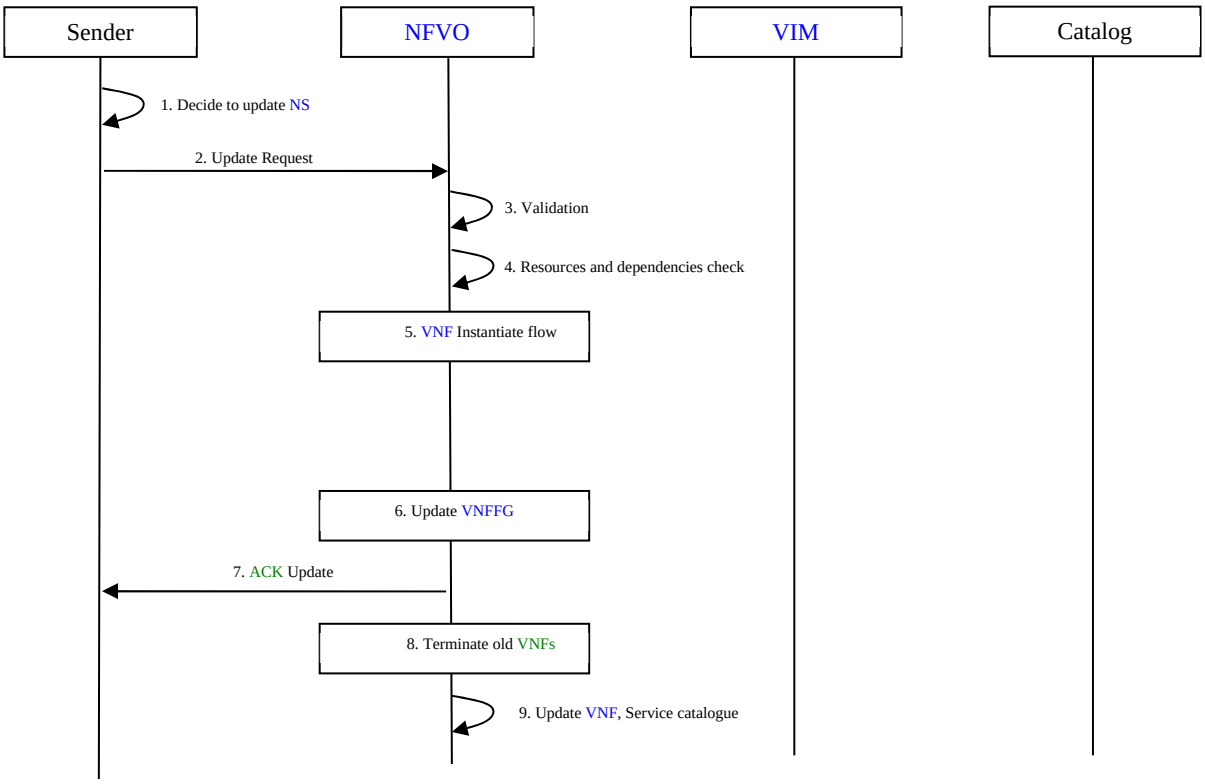


Figure C.12: NS instance update due to VNF instance modification flow

The main steps for the NS Update due to VNF instance modification are:

1. Sender makes a decision to update a NS by modifying one more constituent VNF Instances.
2. Sender sends an update request to the NFVO to update a particular Network Service instance using Update Network Service operations of Network Service lifecycle management interface. The request will include:
   a. Identification of the existing Network Service instance that needs an update.
   b. Identification of the existing VNFD whose Instances (comprising in the Network Service) are expected to be updated.
   c. A reference to the on-boarded new VNFD for the VNF to be updated.

NOTE 1: In case a new VNFD is provided it is on-boarded before executing modify.

   d. Optionally, dependent VNFDs identification and requirements (e.g. version, resources, etc.).

3. The NFVO validates the request including validation that the sender is authorized to issue this request and validation of the parameters passed for technical correctness and policy conformance.
4. NFVO identifies the instances of the dependent VNFs according the dependent VNFDs identification provided and analyses or validates the impact of the Network Service update including checking dependencies (e.g. based on version, existence, resources, etc.) with other VNF Instances (as stated in related NSD) and querying the resources assigned to them (as specified in the old VNFD), to check if the capabilities (e.g. version, resources)assigned to the dependent VNF Instances are enough to honour the dependencies. NFVO can find that the capabilities of dependent VNF Instances cannot support the requirements of the target VNF Instance, e.g. the version of the dependent VNF is lower then what is required in the new VNFD. The NFVO will then create a list of VNFs to be modified considering the dependencies with dependent VNFs. This can include modifying dependent VNFs Instances prior to modifying the target VNF Instance (identified according to the new VNFD identification provided in the request), e.g. in terms of assigned resources, versions, etc.

NOTE 2: VNF Package will be considered on-boarded beforehand. That could require related NSD to be updated accordingly.

5. To modify the VNFs identified above, the NFVO initiates VNF instantiation flow for each VNF to be modified providing instantiation data using the operation "Instantiate VNF" of the VNF Lifecycle Management interface. This includes identifying the VNFD for the dependent VNF instances to be used to update the instance of dependent VNF. In case several VNF instances are expected to be modified and errors occur during single VNF instance modification, roll-back mechanisms are required in order to revert back the VNF instances which have been already modified, i.e. revert to the old VNF instance. These procedures require interaction with corresponding VNF Managers while instantiating and terminating the VNF instances.

NOTE 3: This flow example only shows the steps corresponding to the instantiation and termination of modified VNF instances. The additional steps for modifications on other dependent VNF instances are not shown.

6. Once all new VNF instances are available, NFVO updates the VNF forwarding graphs which includes applying new VNF instances to the forwarding graph in place of the old VNF instances. The flows of the VNF Forwarding Graph update process are described in clause C.7.

7. NFVO acknowledges the update.

8. NFVO monitors old VNF and terminates them when appropriate by calling VNF Termination flow.

9. NFVO updates the NFV Instances repository to reflect the existence new VNF Instances just created.

C.6 Network Service instance termination flows

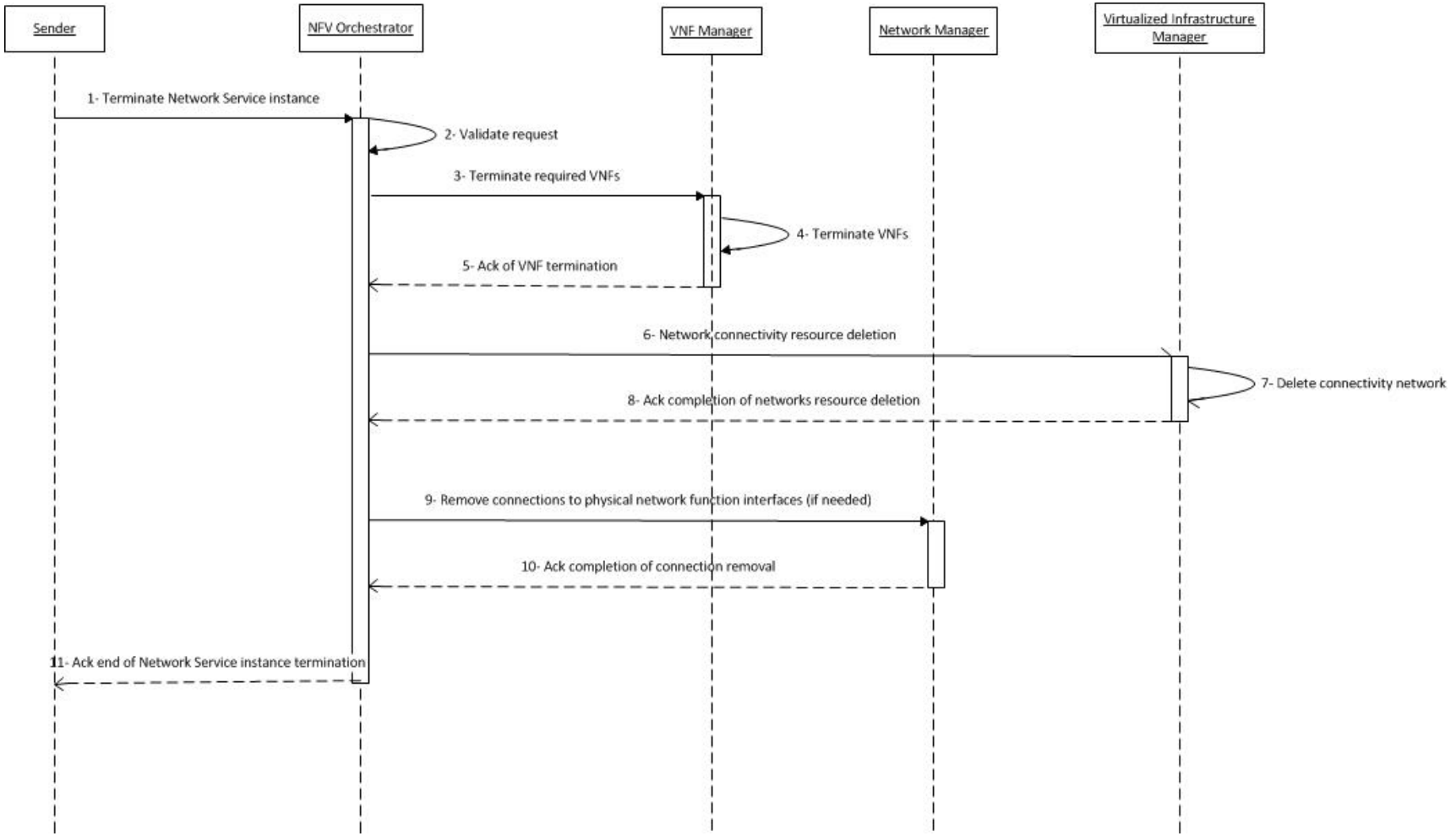


Figure C.13: Network Service termination message flow

NFVO receives a request to terminate a Network Service instance. This request might come from an OSS, receiving an order for Network Service instance termination.

The NFVO is the single point of access for Network Service termination simplifying interfacing with OSS.

When terminating a Network Service instance, several variants might be possible:

a) Some VNF instances were already instantiated before the instantiation of the Network Service. In this case, those VNFs will not be removed.

b) All affected VNF instances contribute to the Network Service that will be terminated and were created when instantiating the Network Service. In this case, all those VNF instances are terminated, and the interconnectivity between these VNF instances is removed.

c) Some VNF instances are contributing to other Network Service instances. In this case, only those VNF instances that do not contribute to other Network Services instances are terminated, and the interconnectivity between them is removed, leaving the other VNF instances in place and the interconnectivity between them intact.

The flow below assumes a single VNF Forwarding Graph for the Network Service instance. If multiple VNF Forwarding Graphs are defined for a Network Service instance, then steps 6 to 8 would be repeated for each VNF Forwarding Graph.

Note that it is assumed that VNF Forwarding Graphs are not shared between Network Service instances.

The main steps for termination of a Network Service instance are:

1. NFVO receives a request to terminate a Network Service instance using the operation Terminate Network Service of the Network Service Lifecycle Management interface.

2. NFVO validates the request. It verifies the validity of the request (including sender's authorization) and verifies that the Network Service instance exists.

3. NFVO requests VNF Manager to terminate any VNF instances that were instantiated along with the Network Service instantiation provided they are not used by another Network Service. This is done by calling the "Terminate VNF instance" request using the operation Terminate VNF of the VNF Lifecycle Management interface as illustrated in clause 7.2.4, VNF instance termination.

4. VNF Manager terminates the required VNF. This step might include some graceful termination of the VNFs involved possibly in coordination with other management entities or the VNFs themselves.

5. Once the VNFs are terminated, the VNF Manager acknowledges completion of the termination request back to the NFV Orchestrator.

6. Using information kept for this Network Service instance, NFVO requests deletion of network connectivity for this Network Service instance using the operations Release Resource or Update Resource of the Virtualised Resources Management interface. Note that some network connectivity might have been present before the instantiation of the Network Service. This connectivity will not be deleted.

7. VIM deletes the connectivity network for this Network Service instance.

8. VIM acknowledges the completion of resource deletion back to NFVO.

9. If needed, NFVO requests Network Manager the removal of connections to physical network function interfaces.

NOTE: The Network Manager can be an OSS, an NMS or an EM.

10. Network Manager acknowledges completion of the removal of connections.

11. NFVO acknowledges the completion of the Network Service instance termination. If the NSD is in deletion pending and there is no Network Service instance using it any more, the NFVO asks the catalogue to remove the corresponding version(s) of the NSD. The catalogue will then remove it.

C.7 VNF Forwarding Graph lifecycle management flows

C.7.0 Preamble

The interface "Network Service lifecycle management" defined in clause 7.1.2 describes the operations for the VNFFG lifecycle management.

This clause provides the flows of these operations for the VNFFG lifecycle management:

- Create VNF Forwarding Graph;
- Update VNF Forwarding Graph;
- Query VNF Forwarding Graph; and
- Delete VNF Forwarding Graph.

The Network Service instance is expected to be updated when any members (e.g. VNF or VL) composing this Network Service is changed. Therefore, the instantiation of a new VNF and modification of the information on VLR are needed for updating the Network Service. Each VNFFG is formed by multiple Connection Points and possibly a Network Forwarding Path (NFP) which indicates a traffic flow of the Network Service. The VNFFG is expected to be updated when any VNFFG members (e.g. Connection Points or NFP) is changed. Therefore, the modification of VNFFG Record and update of NFP are needed for updating the VNFFG.

C.7.1 Create VNF Forwarding Graph

This flow refers to the process of creating a new VNF Forwarding Graph for an existing Network Service. Note that the corresponding Network Service has been instantiated already with the needed VNFs and VLs.

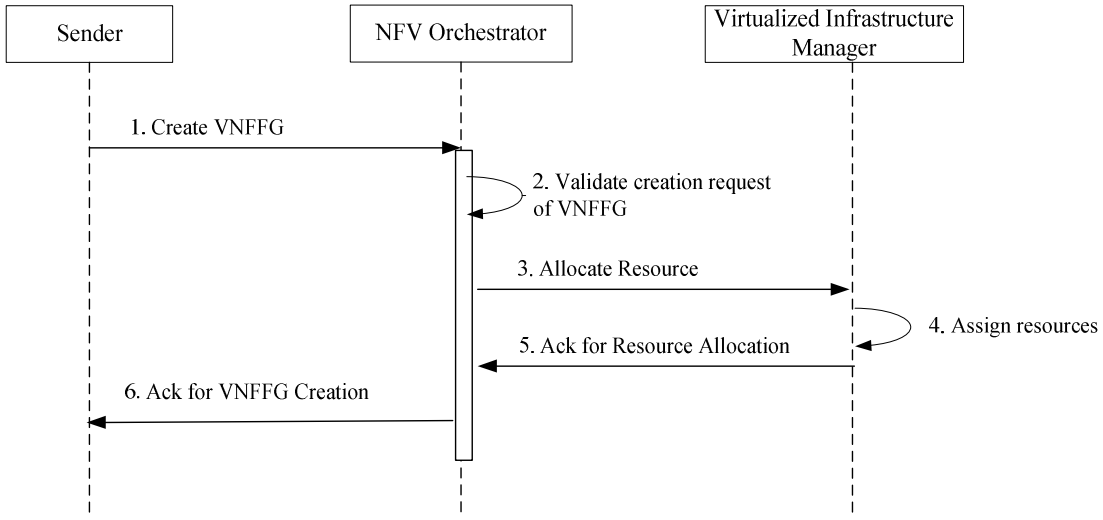


Figure C.14: VNFFG creation message flow

The main steps for creating a VNF Forwarding Graph are:

1. The NFVO receives a request to create a new VNF Forwarding Graph for an existing Network Service using the operation Create VNFFG of the Network Service Lifecycle Management interface.
2. The NFVO validates the request including validation that the sender is authorized to issue this request and validation of the parameters passed for technical correctness and policy conformance.
3. The NFVO requests the allocation of the necessary network resources to set up the VNF Forwarding Graph using the operation Allocation Resource of the Virtualised Resources Management interface.
4. The VIM assigns the needed network resources for the VNF Forwarding Graph.

5. The VIM returns result of resource allocation back to the NFVO.
6. The NFVO returns result of VNF Forwarding Graph creation back to the Sender.

C.7.2 Update VNF Forwarding Graph

This flow refers to the process of updating an existing VNF Forwarding Graph for a Network Service. For this, the information of VNF Forwarding Graph Record and NFP associated to the VNF Forwarding Graph are updated. Note that this update process assumes that a new VNF or new VNFs might already be instantiated to be incorporated into an existing VNF Forwarding Graph.

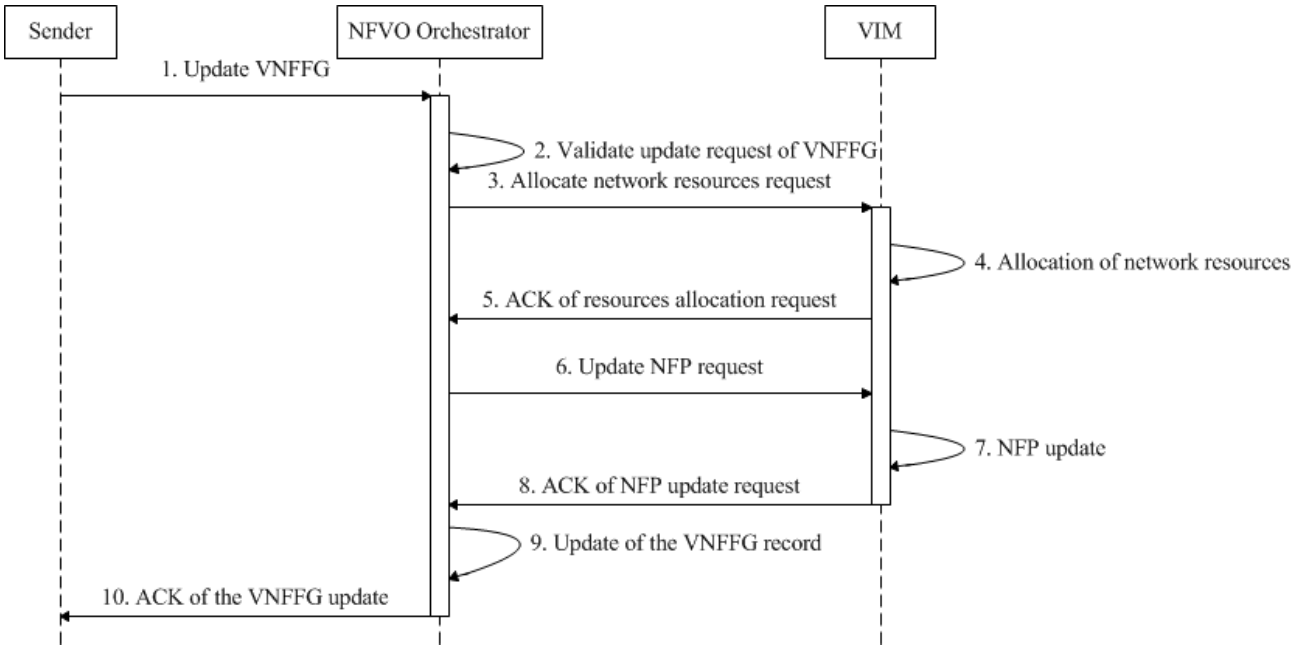


Figure C.15: VNFFG update message flow

The main steps for updating a VNF Forwarding Graph are:

1. The NFVO receives a request to update a VNF Forwarding Graph for an existing Network Service using the operation Update VNFFG of the Network Service lifecycle management interface.
2. The NFVO validates the request including validation that the sender is authorized to issue this request and validation of the parameters passed for technical correctness and policy conformance.
3. The NFVO requests to allocate necessary resources and network connectivity to update the VNF Forwarding Graph using the operation Allocation Resource of the Virtualised Resources Management interface.
4. The VIM assigns the needed network resources for the VNF Forwarding Graph.
5. The VIM returns result of resource allocation back to the NFVO.
6. The NFVO requests to the VIM to update NFP using the operation Update Network Forwarding Path of the Network Forwarding Path rule management interface.
7. The VIM processes the NFP update and performs the required configuration.
8. The VIM acknowledges the NFVO the NFP update.
9. The NFVO updates the VNF Forwarding Graph Record to reflect the information of new VNFs.
10. The NFVO returns result of VNF Forwarding Graph update back to the Sender.

C.7.3 Query VNF Forwarding Graph

This flow refers to the process of retrieving the information of an existing VNF Forwarding Graph for a Network Service.

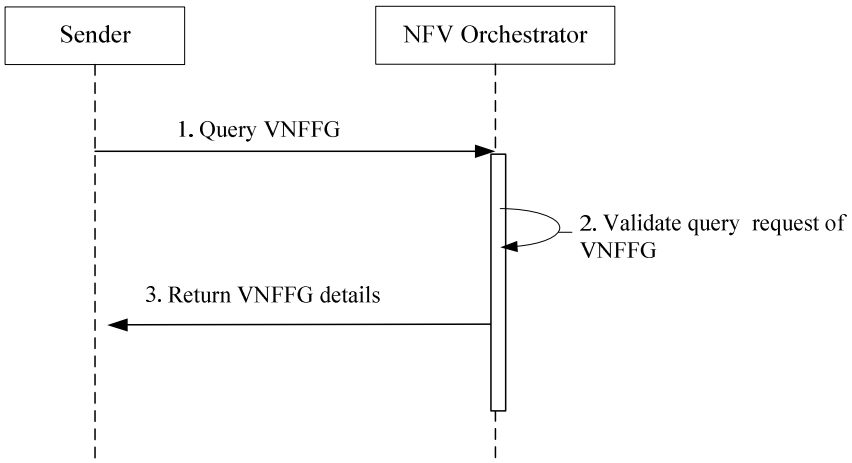


Figure C.16: VNFFG query message flow

The main steps for querying a VNF Forwarding Graph are:

1. The NFVO receives a request to query a VNF Forwarding Graph for an existing Network Service using the operation Query VNFFG of the Network Service Lifecycle Management interface.
2. The NFVO validates the request including validation that the sender is authorized to issue this request and validation of the parameters passed for technical correctness and policy conformance.
3. The NFVO returns the detailed information of VNF Forwarding Graph from the Network Service/VNF Forwarding Graph repository to the Sender.

C.7.4 Delete VNF Forwarding Graph

This flow refers to the process of deleting an existing VNF Forwarding Graph which belongs to an existing Network Service.

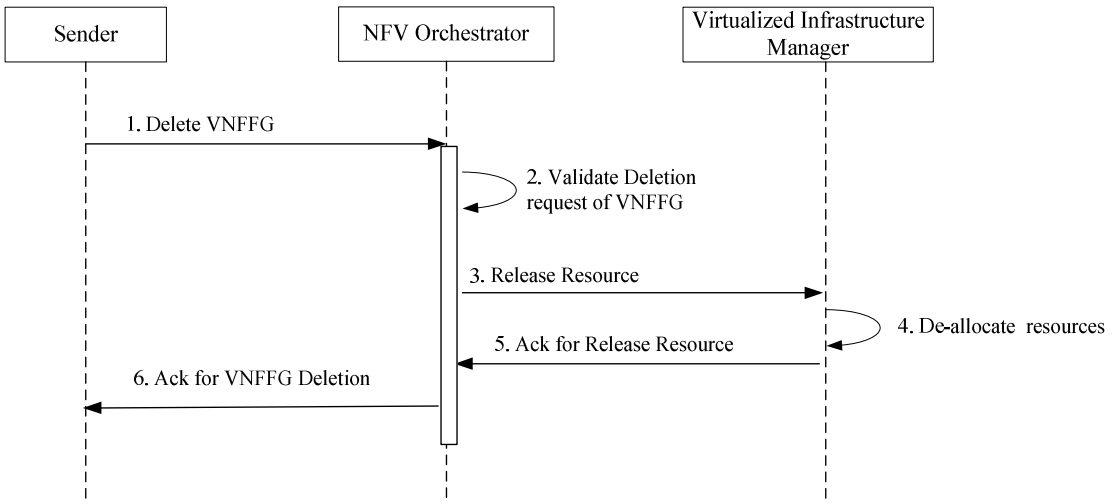


Figure C.17: VNFFG delete message flow

The main steps for deleting a VNF Forwarding Graph are:

1. The NFVO receives a request to delete a VNF Forwarding Graph for an existing Network Service using the operation Delete VNFFG of the Network Service Lifecycle Management interface.
2. The NFVO validates the request including validation that the sender is authorized to issue this request and validation of the parameters passed for technical correctness and policy conformance.
3. The NFVO requests the release of the network resources assigned to this VNF Forwarding Graph using the operation Release Resource of the Virtualised Resources Management interface.
4. The VIM de-allocates the network resources used by the VNF Forwarding Graph.
5. The VIM returns result of resource de- allocation back to the NFVO.
6. The NFVO returns result of VNF Forwarding Graph deletion back to the Sender.

Annex D:
Orchestration flows

D.1 Introduction

This annex represents a collection (non-exhaustive) of Orchestration flows that are presented for information and illustration only and not for implementation purpose.

D.2 NFVI-PoP setup and configuration

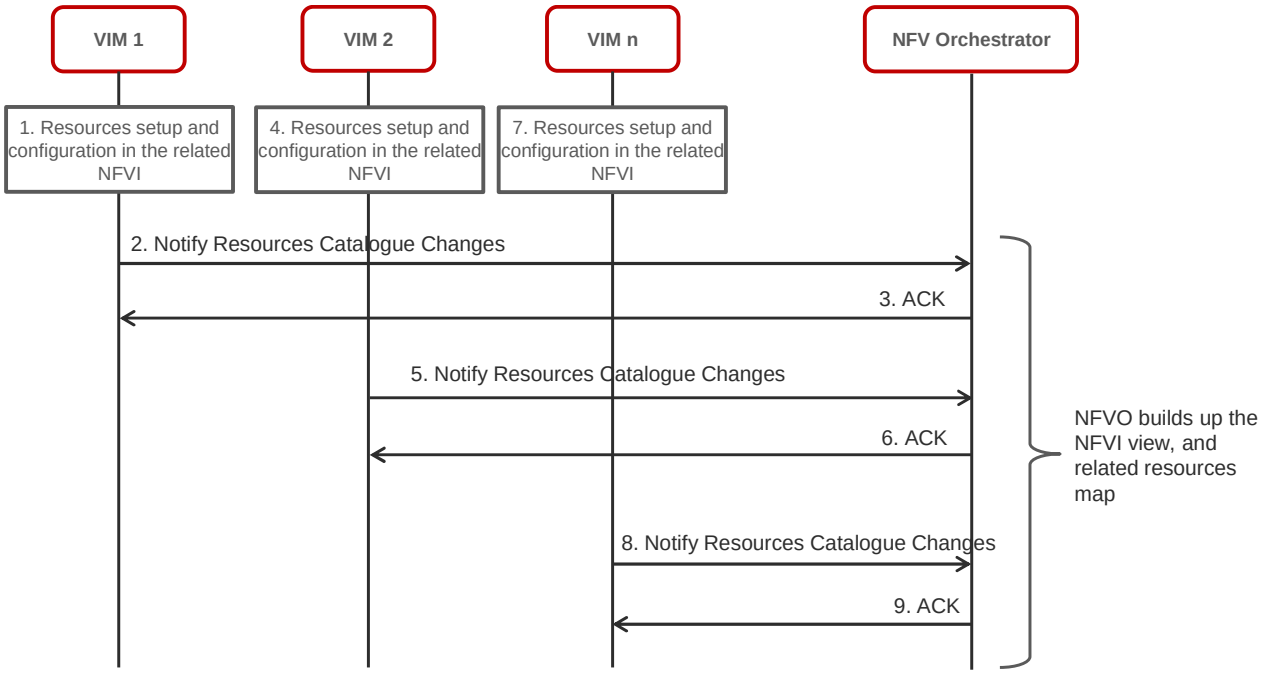


Figure D.1: NFVI-PoP setup and configuration flow

1. Operator setup the NFVI-PoP configuration in the VIM including processing and storage resources.
2. The VIM notifies the NFVO that one or more NFVI-PoPs have been created including all the relevant information (VIM Identifier, NFVI-PoP, location, resource list, capacity and specifications) using the operation Notify Resources Catalogue Changes of the Resources Catalogue Management interface.
3. The NFVO acknowledges the NFVI-PoP creation.

4 to 9. Step 1 to 3 are repeated for each of the VIM (if more than one are deployed) and any time a NFVI-PoP is setup.

The NFVO has all the required information to build the NFVI view including NFVI-PoPs and resources mapping.

D.3 Resources provisioning/de-provisioning

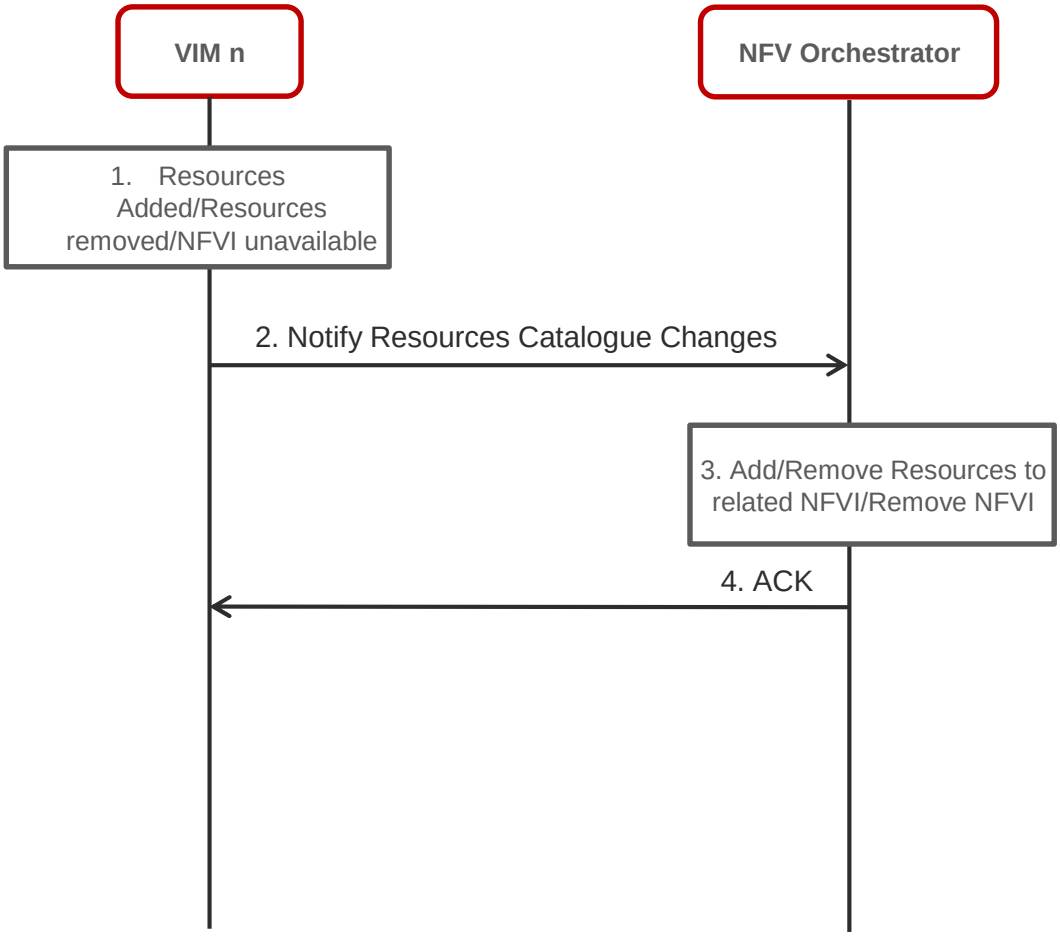


Figure D.2: Resources provisioning/de-provisioning flow

1. HW resources are provisioned/de-provisioned to a NFVI-PoP in the VIM n, or a NFVI-PoP is deleted from the VIM n.
2. The VIM notifies the NFVO that more/less resources are now available including all the relevant information (VIM Identifier, NFVI-PoP, Resource list, capacity and specifications) using the operation Notify Resources Catalogue Changes of the Resources Catalogue Management interface.
3. The NFVO adds resources to the proper NFVI-PoP and updates the mapping.
4. The NFVO acknowledges resource provisioning and update.

This procedure is executed every time resources are provisioned/de-provisioned in any of the NFVI-PoP configured in the infrastructure, or the NFVI-PoP is deleted. The NFVO has all the required information to update the NFVI view including NFVI-PoPs and resources mapping.

Annex E:
VNFD/NSD representations using TOSCA

E.1 Describing IMS MRF with TOSCA

E.1.1 TOSCA meta-model

Clause E.1 provides an illustrative description of IMS MRF using the Topology and Orchestration Specification for Cloud Applications (TOSCA) Service Template.

TOSCA specification provides a language to describe service components and their relationships using a service topology, and it provides for describing the management procedures that create or modify services using orchestration processes. The combination of topology and orchestration in a Service Template describes what is needed to be preserved across deployments in different environments to enable interoperable deployment of cloud services and their management throughout the complete lifecycle (e.g. scaling, patching, monitoring, etc.) when the applications are ported over alternative cloud environments.

Figure E.1 shows the relevance of TOSCA meta-model with NFV descriptors modelling requirements. It provides the mapping between the IMS MRF VNFD/NSD requirements (in red) and the corresponding TOSCA element (in black) to be used to fulfil that requirement. It also shows what all TOSCA CSAR Package can include which imply its relevance to VNF Package.

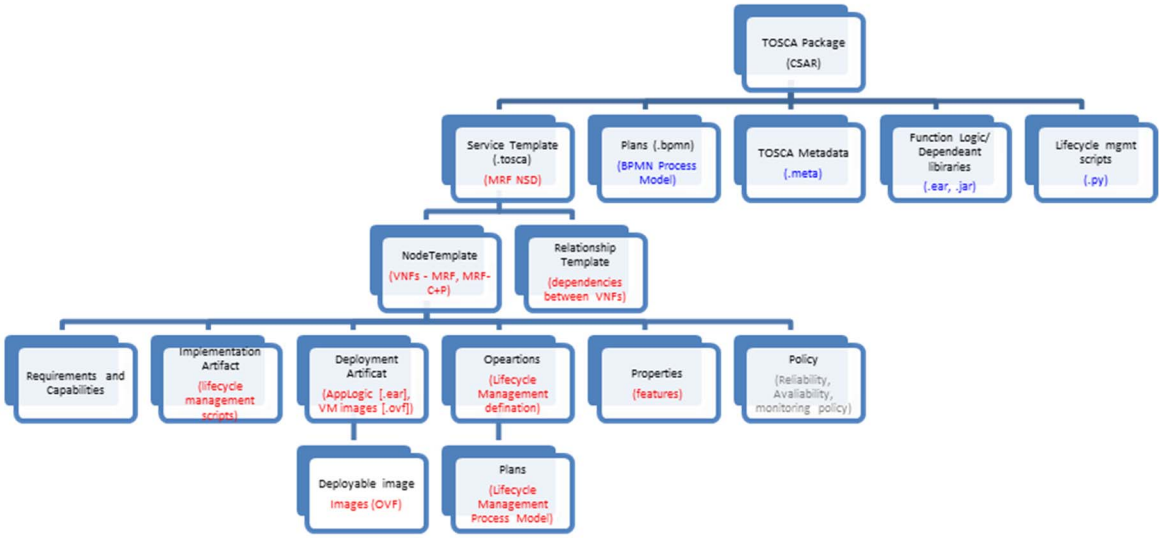


Figure E.1: Mapping of IMS MRF VNFD/NSD to TOSCA elements

E.1.2 IMS MRF representation in TOSCA

E.1.2.0 Preamble

This clause provides information about how TOSCA Service Template can be used to describe VNFD/NSD in a standardized format.

E.1.2.1 IMS MRF NSD

```
<Definitions name="IMSMRFServiceTemplateDefinition"
             targetNamespace="http://www.etsi.org/sample">
  <Tags>
    <Tag name="author" value="someone@example.com"/>
  </Tags>
  <Types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
               elementFormDefault="qualified"
               attributeFormDefault="unqualified">
<!-- This element may be used to model general information elements related with VNF e.g
"vendor", "VNF_ID", etc.-->
      <xs:element name="ApplicationProperties">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Vendor" type="xs:string"/>
            <xs:element name="VNFD" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>

<!-- Level-1 element describing IMS MRF NFV Service.-->
  <ServiceTemplate id="IMSMRFServiceTemplate">
<!-- This element describe the Topology of MRF NFV Service including MRB and MRF C+P and the
relationship (dependencies) between them.-->
    <TopologyTemplate id="IMSMRFService">
<!-- Element describing MRB VNF.-->
      <NodeTemplate id="MyMRB"
                    name="My MRB"
                    nodeType="abc:MRBNode">
<!-- VNF related properties e.g VNFC_no,-->
        <Properties>
          <ApplicationProperties>
            <Vendor>VendorX</Vendor>
            <VNFD>IMS_MRB_x2</VNFD>
          </ApplicationProperties>
        </Properties>
      <NodeTemplate/>
<!-- Element describing MRF VNF.-->
      <NodeTemplate id="MyMRF"
                    name="My MRF"
                    nodeType="abc:MRFNode"/>
<!-- Dependencies between MRB and MRF i.e for a MRF to work an MRB is needed.-->
      <RelationshipTemplate id="MyDeploymentRelationship"
                            relationshipType="dependOn">
        <SourceElement id="MyMRF"/>
        <TargetElement id="MyMRB"/>
      </RelationshipTemplate>

    </TopologyTemplate>

    <Plans>

      <Plan id="InstantiateMRB"
            planType="http://docs.oasis-
             open.org/tosca/ns/2011/12/PlanTypes/InstantiatePlan"
            planLanguage="http://docs.oasis-
             open.org/wsbpel/2.0/process/executable">
        <PlanModelReference reference="prj:InitiateMRB"/>
      </Plan>

      <Plan id="InstantiateMRF "
            planType="http://docs.oasis-
             open.org/tosca/ns/2011/12/PlanTypes/TerminationPlan"
            planLanguage="http://docs.oasis-
             open.org/wsbpel/2.0/process/executable">
        <PlanModelReference reference="prj:InitiateMRF"/>
      </Plan>
    </Plans>

  </ServiceTemplate>

  <NodeType name="MRBNode">
    <documentation xml:lang="EN">
      A reusable definition of a MRB node type representing an
      MRB VNF  that can be deployed.
    </documentation>
```

```
75      <NodeTypeProperties element="ApplicationProperties"/>
76      <Interfaces>
77        <Interface name="InstantiationInterface">
78          <Operation name="InstantiateMRB">
79  <!-- These can be the runtime information provided at MRB Instantiation -->
80            <InputParameters>
81              <InputParamter name="InstanceName"
82                             type="xs:string"/>
83              <InputParamter name="NumberOfInstance"
84                             type="xs:integer"/>
85            </InputParameters>
86          </Operation>
87        </Interface>
88      </Interfaces
89    </NodeType>
90
91    <NodeType name="MRFNode"
92              targetNamespace="http://www.example.com/sample">
93      <NodeTypeProperties element="ApplicationProperties"/>
94      <Interfaces>
95        <Interface name="InstantiationInterface">
96          <Operation name="InstantiateMRB"/>
97          <InputParameters>
98              <InputParamter name="InstanceName"
99                             type="xs:string"/>
100             <InputParamter name="NumberOfInstance"
101                            type="xs:integer"/>
102         </InputParameters>
103       </Interface>
104     </Interfaces>
105   </NodeType>
106
107   <RelationshipType name="dependOn">
108     <documentation xml:lang="EN">
109       A reusable definition of relation that expresses dependencies between two VNF in NFV
Service..
110     </documentation>
111   </RelationshipType>
112
113 </Definitions>
```

E.1.2.2 MRB VNFD

```
01  <Definitions name="MRBServiceTemplateDefinition"
02               targetNamespace="http://www.etsi.org/sample">
03    <Tags>
04      <Tag name="author" value="someone@example.com"/>
05    </Tags>
06    <Types>
07      <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
08                 elementFormDefault="qualified"
09                 attributeFormDefault="unqualified">
10        <xs:element name="ApplicationProperties">
11          <xs:complexType>
12            <xs:sequence>
13              <xs:element name="Vendor" type="xs:string"/>
14              <xs:element name="ID" type="xs:string"/>
15            </xs:sequence>
16          </xs:complexType>
17        </xs:element>
18
19    <ServiceTemplate id="MRBServiceTemplate">
20      <TopologyTemplate id="MRBService">
21
22        <NodeTemplate id="MyMRB"
23                      name="My MRB"
24                      nodeType="abc:MRBNode">
25          <Properties>
26            <ApplicationProperties>
27              <Vendor>VendorX</Vendor>
28              <ID>MRB_x2</ID>
29            </ApplicationProperties>
30          </Properties>
31          <DeploymentArtifacts>
32            <DeploymentArtifact name="MRBVMImage"
33                                type="http://www.example.com/
34                                      ns/tosca/2011/12/
```

```
                                    DeploymentArtifactTypes/VMref">
                VM-edf2cf99 <!-- This can be a reference to a VM Image (e.g AMI) included in
the CSAR.-->
          </DeploymentArtifact>
        </DeploymentArtifacts>

      <NodeTemplate/>

    </TopologyTemplate>

     <Plans>

       <Plan id="InstantiateMRB"
             planType="http://docs.oasis-
              open.org/tosca/ns/2011/12/PlanTypes/InstantiatePlan"
             planLanguage="http://docs.oasis-
              open.org/wsbpel/2.0/process/executable">
         <PlanModelReference reference="prj:InitiateMRB"/>
     </Plan>

   </ServiceTemplate>

   <NodeType name="MRBNode">
     <documentation xml:lang="EN">
       A reusable definition of a MRB node type representing an
       MRB VNF  that can be deployed.
     </documentation>
     <NodeTypeProperties element="ApplicationProperties"/>
     <Interfaces>
       <Interface name="InstantiationInterface">
         <Operation name="InstantiateMRB">
           <InputParameters>
             <InputParamter name="InstanceName"
                            type="xs:string"/>
             <InputParamter name="NumberOfInstance"
                            type="xs:integer"/>
           </InputParameters>
         </Operation>
       </Interface>
     </Interfaces
   </NodeType>
 </Definitions>
```

E.1.2.3 MRF VNFD

```
<Definitions name="MRFServiceTemplateDefinition"
             targetNamespace="http://www.etsi.org/sample">
  <Tags>
  <Tag name="author" value="someone@example.com"/>
  </Tags>
  <Types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
               elementFormDefault="qualified"
               attributeFormDefault="unqualified">
      <xs:element name="ApplicationProperties">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Vendor" type="xs:string"/>
            <xs:element name="ID" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>

  <ServiceTemplate id="MRFServiceTemplate">
    <TopologyTemplate id="MRFService">

      <NodeTemplate id="MyMRFC+P"
                    name="My MRFC+P"
                    nodeType="abc:MRFC+PNode">
        <Properties>
          <ApplicationProperties>
            <Vendor>VendorX</Vendor>
            <ID>IMS_MRFC+P_x2</ID>
          </ApplicationProperties>
        </Properties>
          <DeploymentArtifacts>
          <DeploymentArtifact name="MRFC+PVMImage"
```

```
                                type="http://www.example.com/
                                      ns/tosca/2011/12/
                                      DeploymentArtifactTypes/VMref">
                    VM-edf2cf89
            </DeploymentArtifact>

        <NodeTemplate/>

        <NodeTemplate id="MyMRFStorage"
                      name="My MRF Storage"
                      nodeType="abc:MRFStorageNode"/>

 </TopologyTemplate>

      <Plans>

        <Plan id="InstantiateMRFC+P"
              planType="http://docs.oasis-
               open.org/tosca/ns/2011/12/PlanTypes/InstantiatePlan"
              planLanguage="http://docs.oasis-
               open.org/wsbpel/2.0/process/executable">
          <PlanModelReference reference="prj:InitiateMRFC+P"/>
        </Plan>

        <Plan id="InstantiateMRFStorage "
              planType="http://docs.oasis-
               open.org/tosca/ns/2011/12/PlanTypes/TerminationPlan"
              planLanguage="http://docs.oasis-
               open.org/wsbpel/2.0/process/executable">
          <PlanModelReference reference="prj:InitiateMRFStorage"/>
        </Plan>
      </Plans>

    </ServiceTemplate>

    <NodeType name="MRFC+PNode">
      <documentation xml:lang="EN">
        A reusable definition of a MRFC+P node type representing an
        MRB VNF  that can be deployed.
      </documentation>
      <NodeTypeProperties element="ApplicationProperties"/>
      <Interfaces>
        <Interface name="InstantiationInterface">
          <Operation name="InstantiateMRFC+P">
            <InputParameters>
              <InputParamter name="InstanceName"
                             type="xs:string"/>
              <InputParamter name="NumberOfInstance"
                             type="xs:integer"/>
            </InputParameters>
           </Operation>
        </Interface>
      </Interfaces
    </NodeType>

    <NodeType name="MRFStorageNode"
              targetNamespace="http://www.example.com/sample">
      <NodeTypeProperties element="ApplicationProperties"/>
      <Interfaces>
        <Interface name="InstantiationInterface">
          <Operation name="InstantiateMRFStorage"/>
          <InputParameters>
              <InputParamter name="InstanceName"
                             type="xs:string"/>
              <InputParamter name="NumberOfInstance"
                             type="xs:integer"/>
          </InputParameters>
        </Interface>
      </Interfaces>
    </NodeType>
 </Definitions>
```

Annex F:
YANG/XML VNFD & NSD model

F.1 Use Cases of Network Service with Physical Network Functions

There are significant benefits for service providers if physical devices attached externally to the Infrastructure network could be included in a Network Service VNF Forwarding Graph "to facilitate network evolution".

There are a number of use cases for this:

a) Notably in some mobile Gi services LAN and Internet data centre front end architectures, application delivery controller stages are used to provide traffic routing and resilience switching between "middlebox" processing stages. This will also be useful in the NFV case where load balancing is needed for "stateful" appliances where the go and return flows go through the same logical device. While software based Application Delivery Controller (ADC) functions are available at the server/hypervisor level, providing ADC functionality across multiple server instances is currently dominated by use of hardware appliances. This requirement is expected to be removed in the long term as the mobile and fixed core evolves to a series of web services, but that will take some time.

b) Service providers could wish to have racks of appliances connected to the infrastructure network either because the function cannot be easily virtualised, or they have a number of legacy devices that still have service life and can be "cloudified". The end user could not care if the "firewall service" is delivered from software or hardware appliances, as "it is all in the cloud".

c) Putting a modern orchestration wrap round some hardware devices could get round challenges of updating legacy OSS to introduce new services quickly.

F.2 Examples

F.2.1 NSD for Gi LAN Network Service

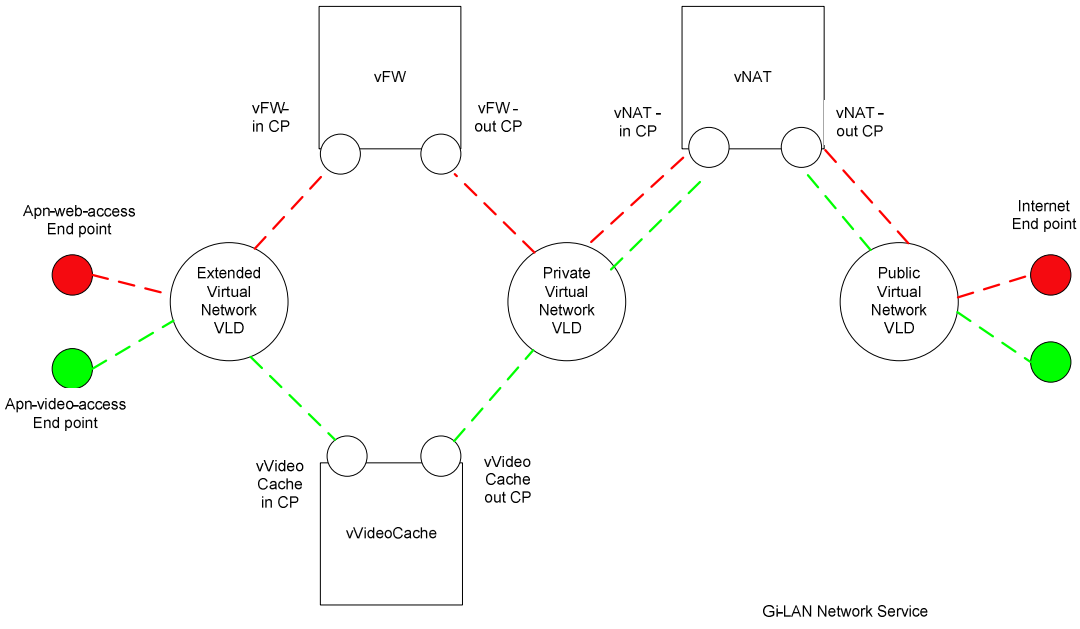


Figure F.1: Gi-LAN Network Service example used for modelling NSD and VNFD

```
<?xml version="1.0" encoding="UTF-8"?>
<nsd xmlns="urn:ietf:params:xml:ns:yang:nfvo"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="urn:ietf:params:xml:ns:yang:nfvo nsd.xsd ">
  <name>Gi-Lan-Service</name>
  <nsd-id/>
  <provider>ETSI</provider>
  <description>Gi-LAN services</description>
  <version>1.0</version>

  <monitoring-params>
         <param-id>num-sessions</param-id>
          <description>
            Total number of sessions the Network Service can handle
          </description>
  </monitoring-params>

  <end-points>
        <apn-webaccess>
            <end-point-id>apn-webaccess</end-point-id>
            <description>APN for web access</description>
        </apn-webaccess>

        <apn-videoaccess>
        <end-point-id>apn-videoaccess</end-point-id>
        <description>APN for video access</description>
        </apn-videoaccess>

        <internet>
        <end-point-id>internet</end-point-id>
        <description>Internet Connection Point</description>
         </internet>
  </end-points>

  <flavours>
      <Gold>
      <flavour-id>Gold</flaur-id>
      <description>Gold Service</description>
       <assurance-params>
            <param-id>num-sessions</param-id>
            <value>1000000</value>
       </assurance-params>

       <member-vnfs>
            <vNAT>
                  <member-vnf-id>vNAT</member-vnf-id>
                  <vnf-flavour>Gold</vnf-flavour>
             </vNAT>
             <vFW>
                 <member-vnf-id>vFW</member-vnf-id>
                 <vnf-flavour>Gold</vnf-flavour>
             </vFW>
             <vVideoCache>
                 <member-vnf-id>vVideoCache</member-vnf-id>
                 <vnf-flavour>Gold</vnf-flavour>
             </vVideoCache>
        </member-vnfs>

        <member-vlds>
             <member-vld-id>PrivateVirtualNetwork</member-vld-id>
             <member-vld-id>PublicVirtualNetwork</member-vld-id>
             <member-vld-id>ExtendedVirtualNetwork</member-vld-id>
        </member-vlds>

        <forwarding-graphs>

           <WebAccess>
                <forwarding-graph-id>WebAccess</forwarding-graph-id>
                <direction>bi-directional</direction>

                 <network-forwarding-path>
                      <end-point-src>apn-webaccess</end-point-src>
                      <forwarding-policy/>
                      <vld>ExtendedVirtualNetwork</vld>
                      <dst-connection-point>
                             <vnf>vFW</vnf>
                             <vnf-connection-point>pkt-in</vnf-connection-point>
```

```
                    </dst-connection-point>

                     <src-connection-point>
                         <vnf>vFW</vnf>
                           <vnf-connection-point>pkt-out</vnf-connection-point>
                     </src-connection-point>
                    <forwarding-policy/>
                    <vld> PrivateVirtualNetwork </vld>
                    <dst-connection-point>
                            <vnf>vNAT</vnf>
                            <vnf-connection-point>pkt-in</vnf-connection-point>
                    </dst-connection-point>

                     <src-connection-point>
                         <vnf>vNAT</vnf>
                           <vnf-connection-point>pkt-out</vnf-connection-point>
                     </src-connection-point>
                   <forwarding-policy/>
                   <vld> PublicVirtualNetwork </vld>
                   <end-point-dst>internet</end-point-dst>
                </network-forwarding-path>
            </WebAccess>

             <VideoAccess>
                   <forwarding-graph-id>VideoAccess</forwarding-graph-id>
                   <direction>bi-directional</direction>

                   <network-forwarding-path>
                        <end-point-src>apn-videoaccess</end-point-src>
                        <forwarding-policy/>
                        <vld>ExtendedVirtualNetwork</vld>
                        <dst-connection-point>
                              <vnf>vVideoCache</vnf>
                                 <vnf-connection-point>pkt-in</vnf-connection-point>
                         </dst-connection-point>

                          <src-connection-point>
                              <vnf>vVideoCache</vnf>
                                <vnf-connection-point>pkt-out</vnf-connection-point>
                          </src-connection-point>
                          <forwarding-policy/>
                          <vld> PrivateVirtualNetwork </vld>
                          <dst-connection-point>
                                <vnf>vNAT</vnf>
                                   <vnf-connection-point>pkt-in</vnf-connection-point>
                           </dst-connection-point>

                            <src-connection-point>
                                <vnf>vNAT</vnf>
                                  <vnf-connection-point>pkt-out</vnf-connection-point>
                            </src-connection-point>
                            <forwarding-policy/>
                           <vld> PublicVirtualNetwork </vld>
                           <end-point-dst>internet</end-point-dst>
                        </network-forwarding-path>
                </VideoAccess>
            </forwarding-graphs>
        </Gold>
    </flavours>
</nsd>
```

F.2.2 VNFD for Virtual Firewall

```
<?xml version="1.0" encoding="UTF-8"?>
<vnfd xmlns="urn:ietf:params:xml:ns:yang:nfvo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:ietf:params:xml:ns:yang:nfvo vnfd-1.xsd ">
  <name>Firewall</name>
  <vnfd-id/>
  <provider>ETSI</provider>
  <description>Firewall</description>
  <version>1.0</version>
  <workflows>
    <init/>
    <terminate/>
    <graceful-shutdown/>
```

```
</workflows>
<connection-points>
      <management-port>
      <name>mgmt-interface</name>
      <description>Management interface</description>
      </management-port>
     <pkt-in>
               <name>pkt-in</name>
       <description>Interface for packet in</description>
     </pkt-in>
     <pkt-out>
      <name>packet-out</name>
      <description>Packet out interface</description>
     </pkt-out>
</connection-points>
<internal-connectivity/>
<monitoring-params>
        <num-rules>
        <para-id>num-rules</para-id>
        <description>Total number of flow the VNF can handle</description>
        </num-rules>
</monitoring-params>
<flavours>

<Gold>
     <flavour-id>Gold</flavour-id>
     <description>Gold Service flavour</description>
     <assurance-params>
         <param-id>num-rules</param-id>
         <value>1000000</value>
     </assurance-params>
     <vdus>
          <vFW>
                <vdu-id>vFW</vdu-id>
                <num-instances>10</num-instances>
                <workflows/>
                <vm-spec>
                      <uri>http://www.example.com/support/vFW1.qcow</uri>
                </vm-spec>
                <storage/>
                <cpu>
                    <num-vpu>2</num-vpu>
                </cpu>
               <memory>
                    <total-memory-gb>8</total-memory-gb>
                  </memory>
                  <other-constraints/>
                 <network-interfaces>
                      <management-interface>
                            <name>management-interface</name>
                            <description>
                              Interface Used for management interface
                            </description>
                            <connection-point-ref>
                              connection-points/management-interface
                            </connection-point-ref>
                            <properties>
                               <driver>
                                  <id>driver</id>
                                 <value>e1000</value>
                               </driver>
                           </properties>
                        </management-interface>
                        <pkt-in>
                            <name>pkt-in</name>
                            <description>Packet in interface</description>
                            <connection-point-ref>
                              connection-points/pkt-in
                            </connection-point-ref>
                            <properties>
                                 <driver>
                                     <id>driver</id>
                                     <value>DPDK-PMD</value>
                                </driver>
                            </properties>
                        </pkt-in>
                        <pkt-out>
                             <name>pkt-out</name>
```

```
                        <description>Packet out interface</description>
                        <connection-point-ref>
                          connection-points/pkt-out
                        </connection-point-ref>
                        <properties>
                           <driver>
                               <id>driver</id>
                               <value>DPDK-PMD</value>
                           </driver>
                        </properties>
                    </pkt-out>
                </network-interfaces>
        </vFW>
    </vdus>
</Gold>

<Silver>
     <flavour-id>Silver</flavour-id>
     <description>Silver Service flavour</description>
     <assurance-params>
         <param-id>num-rules</param-id>
         <value>500000</value>
     </assurance-params>
     <vdus>
         <vFW>
             <vdu-id>vFW</vdu-id>
             <num-instances>5</num-instances>
             <workflows/>
             <vm-spec>
                   <uri>http://www.example.com/support/vFW1.qcow</uri>
             </vm-spec>
             <storage/>
             <cpu>
                    <num-vpu>2</num-vpu>
               </cpu>
               <memory>
                    <total-memory-gb>4</total-memory-gb>
               </memory>
               <other-constraints/>
               <network-interfaces>
                    <management-interface>
                         <name>management-interface</name>
                         <description>
                           Interface Used for management interface
                         </description>
                         <connection-point-ref>
                           connection-points/management-interface
                         </connection-point-ref>
                         <properties>
                            <driver>
                                <id>driver</id>
                               <value>e1000</value>
                            </driver>
                        </properties>
                     </management-interface>
                    <pkt-in>
                        <name>pkt-in</name>
                        <description>Packet in interface</description>
                        <connection-point-ref>
                          connection-points/pkt-in
                        </connection-point-ref>
                        <properties>
                            <driver>
                                <id>driver</id>
                                <value>DPDK-PMD</value>
                             </driver>
                     </properties>
                     </pkt-in>
                     <pkt-out>
                         <name>pkt-out</name>
                         <description>Packet out interface</description>
                         <connection-point-ref>
                           connection-points/pkt-out
                         </connection-point-ref>
                         <properties>
                             <driver>
                                 <id>driver</id>
                                 <value>DPDK-PMD</value>
```

```
                                    </driver>
                                </properties>
                                </pkt-out>
                            </network-interfaces>
                        </vFW>
                    </vdus>
                </Silver>

            </flavours>
            <dependencies/>
            <autoscaling-policies/>
        </vnfd>
```

F.2.3 VNFD for Virtual Carrier Grade NAT

```
<?xml version="1.0" encoding="UTF-8"?>
<vnfd xmlns="urn:ietf:params:xml:ns:yang:nfvo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:ietf:params:xml:ns:yang:nfvo vnfd-1.xsd ">
  <name>CGNAT</name>
  <vnfd-id/>
  <provider>ETSI</provider>
  <description>CGNAT</description>
  <version>1.0</version>
  <workflows>
    <init/>
    <terminate/>
    <graceful-shutdown/>
  </workflows>
  <connection-points>
       <management-port>
             <name>mgmt-interface</name>
             <description>Management interface</description>
       </management-port>
       <pkt-in>
           <name>pkt-in</name>
           <description>Interface for packet in</description>
       </pkt-in>
       <pkt-out>
          <name>packet-out</name>
         <description>Packet out interface</description>
       </pkt-out>
     </connection-points>
     <internal-connectivity/>
      <monitoring-params>
            <para-id>num-sessions</para-id>
            <description>
              Total number of sessions the VNF can handle
            </description>
      </monitoring-params>
     <flavours>

         <Gold>
              <flavour-id>Gold</flavour-id>
              <description>Gold Service flavour</description>
              <assurance-params>
                   <param-id>num-sessions</param-id>
                   <value>100000</value>
              </assurance-params>
              <vdus>
                  <vCGNAT>
                        <vdu-id>vCGNAT</vdu-id>
                        <num-instances>10</num-instances>
                        <workflows/>
                        <vm-spec>
                             <uri>http://www.example.com/support/vFW1.qcow</uri>
                        </vm-spec>
                        <storage/>
                        <cpu>
                             <num-vpu>2</num-vpu>
                        </cpu>
                        <memory>
                              <total-memory-gb>8</total-memory-gb>
                          </memory>
                          <other-constraints/>
                           <network-interfaces>
```

```
                        <management-interface>
                                <name>management-interface</name>
                                <description>
                                  Interface Used for management interface
                                </description>
                                <connection-point-ref>
                                  connection-points/management-interface
                                </connection-point-ref>
                                <properties>
                                    <driver>
                                        <id>driver</id>
                                       <value>e1000</value>
                                    </driver>
                                </properties>
                            </management-interface>
                            <pkt-in>
                                  <name>pkt-in</name>
                                  <description>Packet in interface</description>
                                  <connection-point-ref>
                                    connection-points/pkt-in
                                  </connection-point-ref>
                                  <properties>
                                      <driver>
                                              <id>driver</id>
                                              <value>DPDK-PMD</value>
                                      </driver>
                                  </properties>
                             </pkt-in>
                             <pkt-out>
                                   <name>pkt-out</name>
                                   <description>Packet out interface</description>
                                   <connection-point-ref>
                                     connection-points/pkt-out
                                   </connection-point-ref>
                                    <properties>
                                        <driver>
                                            <id>driver</id>
                                            <value>DPDK-PMD</value>
                                        </driver>
                                   </properties>
                               </pkt-out>
                        </network-interfaces>
                </vCGNAT>
        </vdus>
</Gold>

 <Silver>
     <flavour-id>Silver</flavour-id>
     <description>Silver Service flavour</description>
     <assurance-params>
          <param-id>num-sessions</param-id>
          <value>50000</value>
      </assurance-params>
     <vdus>
          <vCGNAT>
               <vdu-id>vCGNAT</vdu-id>
               <num-instances>5</num-instances>
               <workflows/>
               <vm-spec>
                     <uri>http://www.example.com/support/vFW1.qcow</uri>
               </vm-spec>
               <storage/>
               <cpu>
                     <num-vpu>2</num-vpu>
                </cpu>
                <memory>
                       <total-memory-gb>4</total-memory-gb>
                </memory>
                <other-constraints/>
                <network-interfaces>
                      <management-interface>
                              <name>management-interface</name>
                              <description>
                                Interface Used for management interface
                              </description>
                              <connection-point-ref>
                                connection-points/management-interface
                              </connection-point-ref>
```

```
                                        <properties>
                                           <driver>
                                               <id>driver</id>
                                               <value>e1000</value>
                                           </driver>
                                        </properties>
                                    </management-interface>
                                    <pkt-in>
                                          <name>pkt-in</name>
                                          <description>
                                            Packet in interface
                                          </description>
                                          <connection-point-ref>
                                            connection-points/pkt-in
                                          </connection-point-ref>
                                           <properties>
                                               <driver>
                                                    <id>driver</id>
                                                    <value>DPDK-PMD</value>
                                               </driver>
                                           </properties>
                                    </pkt-in>
                                    <pkt-out>
                                           <name>pkt-out</name>
                                           <description>
                                             Packet out interface
                                           </description>
                                           <connection-point-ref>
                                             connection-points/pkt-out
                                           </connection-point-ref>
                                            <properties>
                                                <driver>
                                                   <id>driver</id>
                                                   <value>DPDK-PMD</value>
                                                </driver>
                                           </properties>
                                     </pkt-out>
                              </network-interfaces>
                        </vCGNAT>
                </vdus>
            </Silver>
        </flavours>
        <dependencies/>
        <autoscaling-policies/>
    </vnfd>
```

F.2.4 VNFD for Virtual Video Cache

```
<?xml version="1.0" encoding="UTF-8"?>
<vnfd xmlns="urn:ietf:params:xml:ns:yang:nfvo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:ietf:params:xml:ns:yang:nfvo vnfd-1.xsd ">
  <name>VideoCaching</name>
  <vnfd-id/>
  <provider>ETSI</provider>
  <description>Video Caching</description>
  <version>1.0</version>
  <workflows>
    <init/>
    <terminate/>
    <graceful-shutdown/>
  </workflows>
  <connection-points>
         <management-port>
              <name>mgmt-interface</name>
              <description>Management interface</description>
          </management-port>
          <pkt-in>
        <name>pkt-in</name>
        <description>Interface for packet in</description>
          </pkt-in>
          <pkt-out>
        <name>packet-out</name>
        <description>Packet out interface</description>
          </pkt-out>
    </connection-points>
```

```
<internal-connectivity/>
 <monitoring-params>
        <para-id>num-sessions</para-id>
        <description>Total number of sessions the VNF can handle</description>
 </monitoring-params>
 <flavours>

     <Gold>
          <flavour-id>Gold</flavour-id>
          <description>Gold Service flavour</description>
          <assurance-params>
               <param-id>num-sessions</param-id>
               <value>100000</value>
          </assurance-params>
          <vdus>
              <vVideoCache>
              <vdu-id>vVideoCache</vdu-id>
              <num-instances>10</num-instances>
              <workflows/>
              <vm-spec>
                   <uri>http://www.example.com/support/vFW1.qcow</uri>
              </vm-spec>
              <storage>
              <cache>
                  <id>cache</id>
                  <size>100g</size>
              </cache>
              </storage>
              <cpu>
                 <num-vpu>2</num-vpu>
             </cpu>
            <memory>
                <total-memory-gb>16</total-memory-gb>
            </memory>
           <other-constraints/>
           <network-interfaces>
                 <management-interface>
                        <name>management-interface</name>
                        <description>
                          Interface Used for management interface
                        </description>
                         <connection-point-ref>
                           connection-points/management-interface
                         </connection-point-ref>
                         <properties>
                         <driver>
                             <id>driver</id>
                             <value>e1000</value>
                         </driver>
                         </properties>
                  </management-interface>
                 <pkt-in>
                        <name>pkt-in</name>
                        <description>Packet in interface</description>
                        <connection-point-ref>
                          connection-points/pkt-in
                        </connection-point-ref>
                        <properties>
                        <driver>
                            <id>driver</id>
                            <value>DPDK-PMD</value>
                        </driver>
                        </properties>
                   </pkt-in>
                   <pkt-out>
                         <name>pkt-out</name>
                         <description>Packet out interface</description>
                         <connection-point-ref>
                           connection-points/pkt-out
                         </connection-point-ref>
                         <properties>
                         <driver>
                             <id>driver</id>
                             <value>DPDK-PMD</value>
                         </driver>
                         </properties>
                    </pkt-out>
              </network-interfaces>
```

```
       </vVideoCache>
    </vdus>
</Gold>

<Silver>
    <flavour-id>Silver</flavour-id>
    <description>Silver Service flavour</description>
    <assurance-params>
          <param-id>num-sessions</param-id>
          <value>50000</value>
    </assurance-params>
    <vdus>
         <vVideoCache>
         <vdu-id>vVideoCache</vdu-id>
         <num-instances>5</num-instances>
         <workflows/>
         <vm-spec>
                 <uri>http://www.example.com/support/vFW1.qcow</uri>
           </vm-spec>
           <storage>
               <cache>
                     <id>cache</id>
                     <size>100g</size>
              </cache>
          </storage>
          <cpu>
               <num-vpu>2</num-vpu>
          </cpu>
          <memory>
               <total-memory-gb>4</total-memory-gb>
           </memory>
          <other-constraints/>
          <network-interfaces>
                 <management-interface>
                       <name>management-interface</name>
                       <description>
                         Interface Used for management interface
                       </description>
                       <connection-point-ref>
                         connection-points/management-interface
                       </connection-point-ref>
                        <properties>
                            <driver>
                                <id>driver</id>
                                <value>e1000</value>
                            </driver>
                        </properties>
                 </management-interface>
                 <pkt-in>
                      <name>pkt-in</name>
                      <description>Packet in interface</description>
                      <connection-point-ref>
                        connection-points/pkt-in
                      </connection-point-ref>
                      <properties>
                      <driver>
                          <id>driver</id>
                          <value>DPDK-PMD</value>
                          </driver>
                      </properties>
                  </pkt-in>
                  <pkt-out>
                        <name>pkt-out</name>
                        <description>Packet out interface</description>
                        <connection-point-ref>
                          connection-points/pkt-out
                        </connection-point-ref>
                        <properties>
                             <driver>
                                 <id>driver</id>
                                 <value>DPDK-PMD</value>
                             </driver>
                         </properties>
                      </pkt-out>
                  </network-interfaces>
             </vVideoCache>
       </vdus>
   </Silver>
```

```
      </flavours>
      <dependencies/>
      <autoscaling-policies/>
</vnfd>
```

F.2.5 VLDs

```
  <?xml version="1.0" encoding="UTF-8"?>
<vld xmlns="urn:ietf:params:xml:ns:yang:nfvo" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:ietf:params:xml:ns:yang:nfvo vld.xsd ">
  <name>ExtendedVirtualNetwork</name>
  <vld-id/>
  <provider>ETSI</provider>
  <description>Extended Virtual Network - Interworking with a Gateway Router </description>
  <version>1.0</version>
  <latency-assurance>10ms</latency-assurance>
  <max-end-points>100</max-end-points>
</vld>


  <?xml version="1.0" encoding="UTF-8"?>
<vld xmlns="urn:ietf:params:xml:ns:yang:nfvo"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="urn:ietf:params:xml:ns:yang:nfvo vld.xsd ">
  <name>PrivateVirtualNetwork</name>
  <vld-id/>
  <provider>Generic</provider>
  <description>Pure Virtual Network</description>
  <version>1.0</version>
  <latency-assurance>20ms</latency-assurance>
  <max-end-points>100</max-end-points>
  <properties/>
</vld>



  <?xml version="1.0" encoding="UTF-8"?>
<vld xmlns="urn:ietf:params:xml:ns:yang:nfvo" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:ietf:params:xml:ns:yang:nfvo vld.xsd ">
  <name>PublicVirtualNetwork</name>
  <vld-id/>
  <provider>Generic</provider>
  <description>Pure Virtual Network</description>
  <version>1.0</version>
  <latency-assurance>10ms</latency-assurance>
  <max-end-points>50</max-end-points>
  <properties/>
</vld>
```

F.3 YANG schema

F.3.1 YANG schema for NSD

This annex targeted as a preliminary example model towards using YANG as a candidate for modelling the ETSI NFV information elements.

```
module nsd {

    namespace "urn:ietf:params:xml:ns:yang:nfvo";
    prefix "nsd";

    import ietf-yang-types {
        prefix yang;
    }
    import ietf-inet-types {
        prefix inet;
    }

    organization
    "example.org";
```

```
contact
"Firstname Lastname - lastname@example.com";

description
"Network Service Descriptor";

revision 2013-12-23 {
    description
     "Initial revision";
}

container nsd {

    leaf name {
        description "Network Service Name";
        type string;
    }

    leaf id {
        description "Network Service Descriptor ID";
        type yang:uuid;
    }

    leaf vendor {
        description "Provider";
        type string;
    }

    leaf description {
        description "Description of the Network Service Descriptor";
        type string;
    }

    leaf version {
        description "Version";
        type string;
    }


    list monitoring-params {
        key param-id;
        leaf param-id {
            type string;
        }

        leaf description {
            type string;
            description "Description";
        }
    }

    list connection-points {
        description "End Points for the Network Service";
        key connection-point-id;
        leaf connection-point-id {
            type string;
        }

        leaf description {
            type string;
            description "Description of the End point";
        }
    }


    list flavours {
        key flavour-id;

        leaf flavour-id {
            description "Flavour id";
            type string;
        }

        leaf description {
            description "Description of the flavour";
            type string;
        }
```

```
            list flavour-key {
                description "Flavour key";

                key param-id;
                leaf param-id {
                    type leafref {
                        path "../../../monitoring-params/param-id";
                    }
                }
                leaf value {
                    type uint32;
                }
            }

            list constituent-vnfs {
                key member-vnf-id;
                leaf member-vnf-id {
                    type string;
                }

                leaf vnfd {
                    type yang:uuid;
                }
            }

            container vnf-dependencies {
                list vvnf-dependency-list {
                    key vnf-id;
                    leaf vnf-id {
                        type leafref {
                            path "../../../member-vnfs/member-vnf-id";
                        }
                    }
                    leaf depends-on {
                        type leafref {
                            path "../../../member-vnfs/member-vnf-id";
                        }
                    }
                }
            }
            list member-vlds {
                key member-vld-id;
                leaf member-vld-id {
                    type string;
                }

                leaf vld-id {
                    type yang:uuid;
                }
            }
        }
    }
}
```

F.3.2 YANG schema for VNFD

```
module vnfd {

    namespace "urn:ietf:params:xml:ns:yang:nfvo";
    prefix "vnfd";

    import ietf-yang-types {
        prefix yang;
    }

    import ietf-inet-types {
        prefix inet;
    }

    organization
    "example.org";

    contact
    "Firstname Lastname - lastname@example.com";

    description
```

```
"VNF Descriptor";

revision 2013-12-23 {
    description
     "Initial revision";
}

container vnfd {

    leaf name {
        description "VNFD Name";
        type string;
    }

    leaf vnfd-id {
        description "UUID of the VNF Descriptor";
        type yang:uuid;
    }

    leaf provider {
        description "Vendor";
        type string;
    }

    leaf description {
        description "Description of the VNF";
        type string;
    }

    leaf version {
        description "Version";
        type string;
    }

    container workflows {
        container init {
            /* Needs expansion */
        }
        container terminate {
            /* Needs expansion */
        }
        container graceful-shutdown {
            /* Needs expansion */
        }
    }

    list connection-points {
        description "External interfaces exposed by this VNF";
        key connection-point-id;
        leaf connection-point-id {
            type string;
        }

        leaf description {
            type string;
            description "Description of the Connection Point";
        }
    }
    list internal-virtual-link {
        key connectivity-id;
        leaf connectivity-id {
            type string;
        }
        leaf connectivity-description {
            type string;
            description "Description";
        }
        leaf-list connection-end-points {
            type leafref {
                path "../../flavours/vdus/network-interfaces/name";
            }
        }
    }
    list monitoring-params {
        key param-id;
        leaf param-id {
            type string;
        }
```

```
    leaf description {
        type string;
        description "Description";
    }
}

list flavours {
    key flavour-id;

    leaf flavour-id {
        description "Flavour id";
        type string;
    }

    leaf description {
        description "Description of the flavour";
        type string;
    }

    list assurance-params {
        description "Flavour key";

        key param-id;
        leaf param-id {
            type leafref {
                path "../../../monitoring-params/param-id";
            }
        }
        leaf value {
            type uint32;
        }
    }

    /* TODO: Add licensing and pricing info */

    list vdus {
        description "Virtual Deployable Units";

        key vdu-id;
        leaf vdu-id {
            description "";
            type string;
        }

        leaf num-instances {
            description "";
            type uint16;
        }

        container workflows {
            container init {
                /* Needs expansion */
            }
            container terminate {
                /* Needs expansion */
            }
            container graceful-shutdown {
                /* Needs expansion */
            }
        }

        container vm-spec {
            choice image {
                case image-uri {
                    leaf pkg-uri {
                        type inet:uri;
                    }
                }
                case image-uuid {
                    leaf pkg-uuid {
                        type yang:uuid;
                    }
                }
            }
        }
        list storage {
            key storage-id;
```

```
    leaf storage-id {
        type string;
    }
    leaf size-gb {
        type uint16;
    }
    list properties {
        description "Properties for the storage";

        key property-id;
        leaf property-id {
            type string;
        }
        leaf property-description {
            type string;
            description
                "Description of the property";
        }
        leaf property-value {
            type string;
        }
    }
}
container cpu {
    leaf num-vpu {
        type int16;
    }
    list properties {
        description "Properties for the cpu";

        key property-id;
        leaf property-id {
            type string;
        }
        leaf property-description {
            type string;
            description
                "Description of the property";
        }
        leaf property-value {
            type string;
        }
    }
}
container memory {
    leaf total-memory-gb {
        type int16;
    }
    list properties {
        description "Properties for the memory";

        key property-id;
        leaf property-id {
            type string;
        }
        leaf property-description {
            type string;
            description
                "Description of the property";
        }
        leaf property-value {
            type string;
        }
    }
}

list other-constraints {
    description "Other Constraints for the VDU";

    key constraint-id;
    leaf constraint-id {
        type string;
    }
    leaf constraint-description {
        type string;
        description
            "Description of the constraint";
    }
```

```
                    leaf constraint-value {
                        type string;
                    }
                }
                list network-interfaces {
                    key name;
                    leaf name {
                        type string;
                    }
                    leaf description {
                        type string;
                    }
                    choice connection-point {
                        case connection-point {
                            leaf connection-point-ref {
                                type leafref {
                                    path "../../../../connection-points/connection-point-id";
                                }
                            }
                        }
                        case internal-connectivity {
                            leaf internal-connectivity-ref {
                                type leafref {
                                    path "../../../../internal-connectivity/connectivity-id";
                                }
                            }
                        }
                    }
                    list properties {
                        description "Properties for the interface";

                        key id;
                        leaf id {
                            type string;
                        }
                        leaf value {
                            type string;
                        }
                    }
                }
            }
        }

        container dependencies {
            list vdu-dependency-list {
                key vdu-id;
                leaf vdu-id {
                    type leafref {
                        path "../../../flavours/vdus/vdu-id";
                    }
                }
                leaf depends-on {
                    type leafref {
                        path "../../../flavours/vdus/vdu-id";
                    }
                }
            }
        }

        list autoscaling-policies {
            key scaling-policy-id;
            leaf scaling-policy-id {
                type string;
            }
        }
    }
}
```

F.3.3 YANG schema for VLD

```
module vld {

    namespace "urn:ietf:params:xml:ns:yang:nfvo";
    prefix "vld";

    import ietf-yang-types {
```

```
        prefix yang;
    }

    import ietf-inet-types {
        prefix inet;
    }

    organization
    "example.org";

    contact
    "Firstname Lastname - lastname@example.com";

    description
    "Virtual Link Descriptor";

    revision 2013-12-23 {
        description
         "Initial revision";
    }

    container vld {

        leaf name {
            description "Description of the Virtual Link Descriptor";
            type string;
        }

        leaf vld-id {
            description "UUID of the VLD";
            type yang:uuid;
        }

        leaf provider {
            description "Provider";
            type string;
        }

        leaf description {
            description "Description of the VNF";
            type string;
        }

        leaf version {
            description "Version";
            type string;
        }

        leaf latency-assurance {
            type uint16;
        }

        leaf max-end-points {
            type uint16;
        }

        list properties {
            description "Vendor specific properties";

            key property-id;
            leaf property-id {
                type string;
            }
            leaf property-description {
                type string;
                description
                    "Description of the property";
            }
            leaf property-value {
                type string;
            }
        }

    }
}
```

Annex G:
TM Forum SID service model

NFV-based network functions (VNFs) co-exist with traditional non NFV-based NFs (this includes PNFs as well as logical NFs that are not NFV-based) to enable simple and scalable gradual deployment of VNFs and other NFV concepts.

NFV-based portions of the network also integrate with OSS and BSS functions. Integration of operations and processes is on a network wide basis. Hence, the integration crosses the boundaries between NFV-based and non-NFV-based functions.

When replacing traditional NFs by VNFs, it is recommended that the impact on any major E2E processes that are affected is minimized; examples include:

- Service Fulfilment.
- Service Provisioning.
- Billing.
- Etc.

E2E processes driven by the OSS/BSS functions cross NFV and non-NFV boundaries.

As described in the NFV End-to-End Architecture document, an end-to-end Network Service (e.g. mobile voice/data, Internet access, or a virtual private network) can be defined as a forwarding graph of Network Functions (NFs) and end points/terminals, which is "what" an operator provides to customers.

This is illustrated by figure G.1.

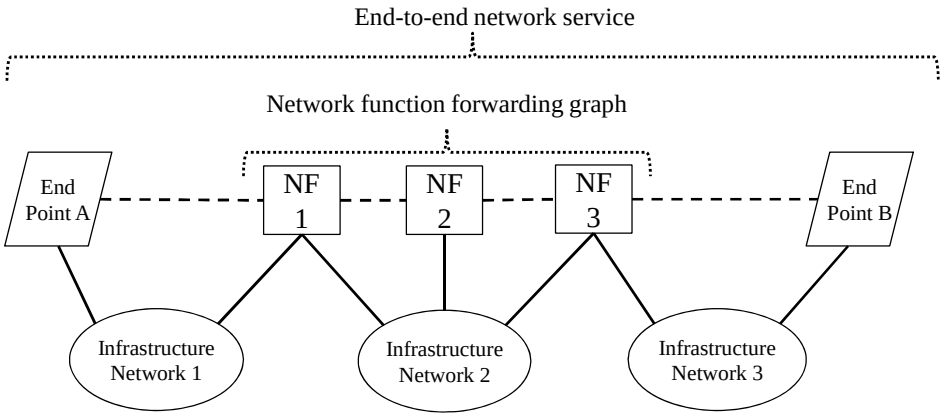


Figure G.1: Graph representation of an end-to-end Network Service

The SID Service Model from the TM Forum is used by most OSS/BSS systems.

To provide easy integration with existing OSS/BSS systems, a mapping between end-to-end Network Services that include VNFs or VNF Forwarding Graphs and the SID Service Model is expected.

This would help the Operators to simplify the NFV roll-out, reducing the impact on OSS/BSS. This Service Model is used on the interface between OSS/BSS and Management & Orchestration (Os-Ma) and also on the interface between OSS/BSS and EM.

Figure G.2 is showing a simplified view of the SID Service Model.

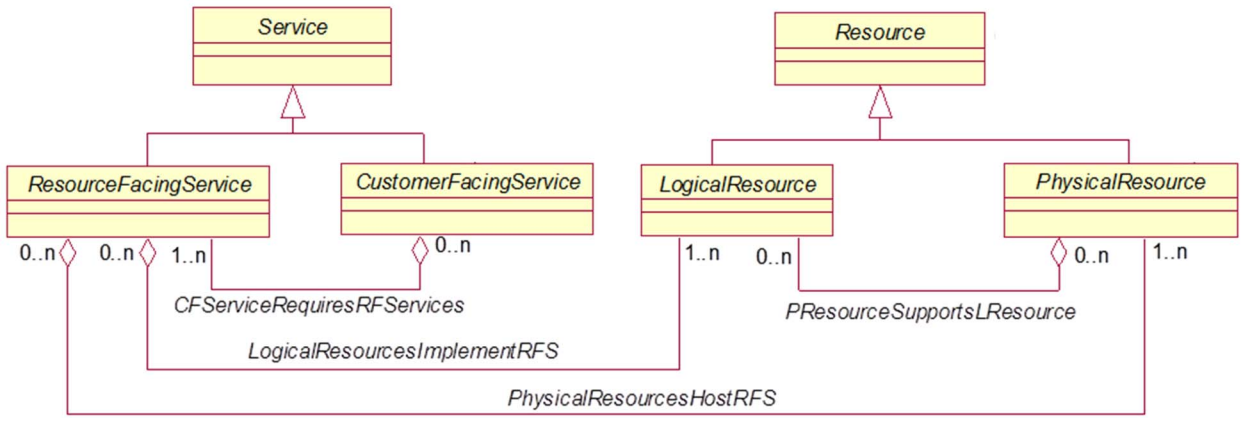


Figure G.2: Simplified SID Service Model

NFV defines a number of key NFV concepts. At a minimum, the concepts of a VNF, Network Service, Network Service Descriptor (NSD) and VNF Graph are expected to be introduced in the Service Model above to provide the needed integration.

A VNF is a type of SID LogicalResource. As a SID LogicalResource, VNF implements ResourceFacingServices (RFSs) that are required to provide CustomerFacingServices (CFSs).

A Network Service has two important characteristics:

1) it defines a collection of LogicalResources (e.g. VNFs) that have a defined topology; and

2) the Network Service as a whole can offer a set of interfaces and services that result from arranging the VNFs in a particular topology and grouping them together as a component.

A Network Service can provide CFSs and/or RFSs (both are optional!). A Network Service contains at least 1 VNF. A Network Service contains at least 1 VNF Forwarding Graph. A VNF Forwarding Graph has at least one node, and at least one node is a VNF.

Figure G.3 is showing those changes.

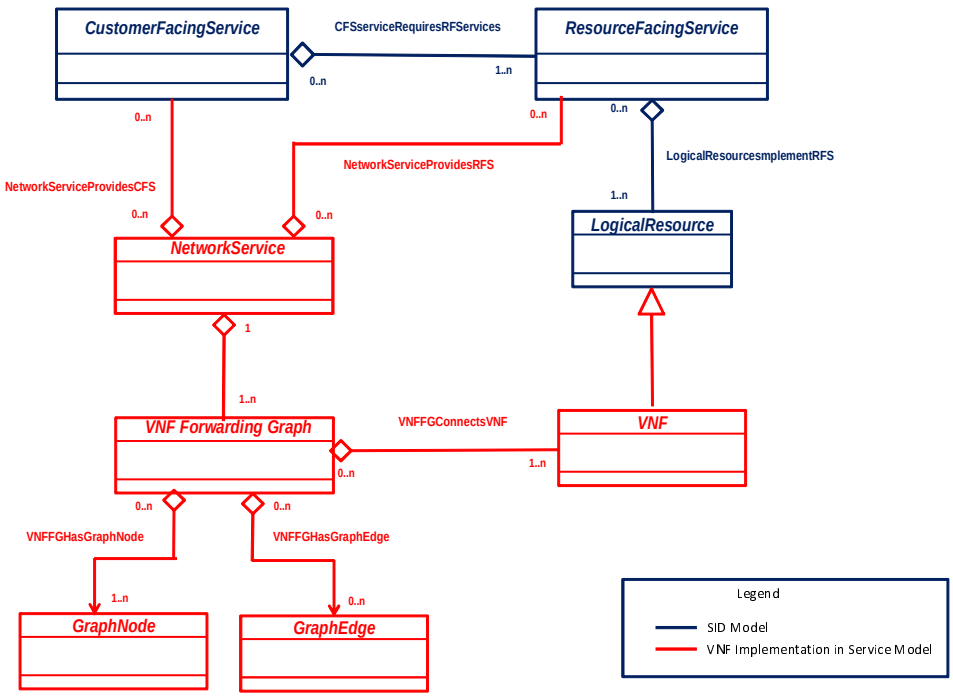


Figure G.3: VNF Implementation in SID Service Model

Note that figure G.3 is mainly illustrative and intended as an input to the TM Forum SID.

Annex H:
Open Virtualisation Format

The Distributed Management Task Force (DMTF) Open Virtualisation Format (OVF) specification version 2.0.1, document number DSP0243, dated 2013-08-30 [i.12] and the related OVF Envelope XML Schema Document (XSD) version 2.0.1, dated 2013-08-25, DSP8023 [i.13] were developed to describe the packaging and distribution of software to be run in a virtual machine. The document status is identified as a DMTF standard. The authors of the OVF specification identified several characteristics that the specification complies with which include:

- Distribution considerations including verification and integrity checking.
- Single and multiple VM configurations.
- Virtualisation platform neutrality.
- Vendor and platform neutrality.
- Design extensibility.
- Localizable.
- Openness.

OVF relies on a number of DMTF Common Information Model (CIM) standards. CIM has defined ~1 378 classes/subclasses, with another ~61 Problem Resolution Standard (PRS) classes that can be treated as CIM extensions. This contribution references the CIM V2.38.0 schema [i.14].

The OVF specification relies on a number of Common Information Model (CIM) classes in the VirtualHardwareSection such as the `CIM_VirtualSystemSettingData`, `CIM_ResourceAllocationSettingData`, `CIM_EthernetPortAllocationSettingData`, and `CIM_StorageAllocationSettingData`. These classes have many subclasses.

OVF and the CIM classes can also be used to describe multiple VM based solutions via the `VirtualSystemCollection` elements. This can or not align with some topology considerations for the VNFD.

Although the specification was not developed with explicit considerations for some of the more exacting requirements expected by NFV deployments, the broad adoption of support for OVF in cloud related products and its extensible design means that it could be considered a reasonable starting point for VDU elements and could be considered as one of the standards that the NFV-interested community can refer to in order to accelerate NFV deployments, drive interoperability between NFV deployments and in particular, focus on how VDU elements could be described in a standards compliant way.

The annex in the OVF standard document combined with a number of schemas and the CIM profiles listed on the DMTF website contain examples of OVF descriptors.

Annex I:
Other information

I.1 OpenStack

I.1.1 Introduction

OpenStack provides an Infrastructure as a Service (IaaS) solution through a set of interrelated services. Each service offers an Application Programming Interface (API) that facilitates their integration.

OpenStack Programs represent core functionalities that are essential to the completion of OpenStack's vision and are provided by services. Projects represent the implemented services and belong to a Program.

Figure I.1 shows the relationships among the OpenStack core services. For more information about OpenStack services refer to [i.9]. Note that in the present clause, OpenStack information refer to Havana release (October 2013) [i.10].

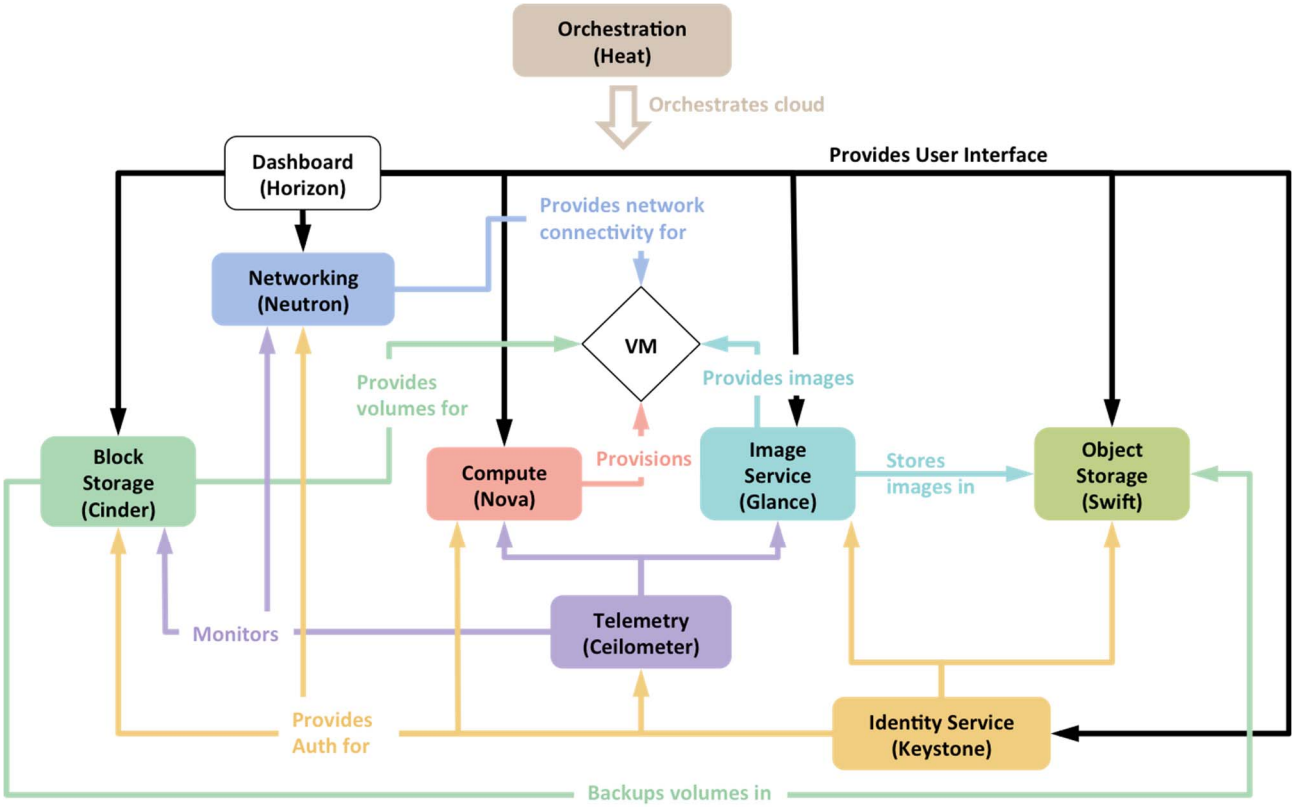


Figure I.1: OpenStack conceptual architecture (source [i.9])

I.1.2 NFV-related features in OpenStack community

Following is a list of projects in the OpenStack community which are related to ETSI NFV. Note that the information contained herein refers to Havana release (October 2013) [i.10] and projects/programs status as of February 2014.

Project	Description	References
Snabb NFV	Open source NFV system for OpenStack Neutron. The design goals of the project are: • Robustness: no bottleneck, no single point of failure. • High capacity (40+ Gbps Ethernet per host). • Transparency: connect to operator's own network. Snabb NFV consists of two pieces of software: • Snabb mechanism driver: an extension of the OpenStack Neutron modular layer 2 (ML2) plugin. ML2 plugin is a framework that allows simultaneous utilization of several L2 networking technologies. • Snabb Switch: software switch on compute hosts. Configuration and management is performed using a subset of the standard Neutron API.	For more information: https://github.com/SnabbCo/snabbswitch/tree/snabbnfv-readme/src/designs/nfv
Climate	Climate is an OpenStack service for virtual and physical resource reservation. The service aims at handling a calendar of reservations for various resources based on various reservation criteria. The project is split into three main development tracks: • Core part development, including leases/reservation management, plugins, etc. • Physical resource reservations. • Virtual resource reservations.	For more information: https://launchpad.net/climate

Following is a list of programs in the OpenStack community which are of particular interest due to specific requirements from ETSI NFV.

Program	Description	References
Telemetry (code-name Ceilometer)	Ceilometer focuses its activity on the following primary purposes: • Collection of metering data, e.g. CPU and network costs. • Collection of data by monitoring notifications sent by services, or by polling the infrastructure. • Configuration of the type of collected data based on various operating requirements. • Access of the metering data through the REST APIs. Ceilometer is an integrated project and it is available in the OpenStack Havana release.	For more information: https://wiki.openstack.org/wiki/Ceilometer
Baremetal (code-name Ironic)	The program aims at supporting baremetal host provisioning in OpenStack. The main outcome is the "baremetal driver", a hypervisor driver for OpenStack Nova Compute that allows provisioning and management of physical hardware using common cloud APIs and tools. Current implementation of the driver provides functionality for managing the baremetal database, power control of managed hardware via IPMI, PXE boot of baremetal nodes, etc. As of February 2014, Ironic is incubation stage, i.e. not yet integrated in the OpenStack release.	For more information: https://wiki.openstack.org/wiki/Baremetal and https://wiki.openstack.org/wiki/Ironic

History

Document history		
V1.1.1	December 2014	Publication as ETSI GS NFV-MAN 001
V1.2.1	December 2021	Publication