ETSI GR NFV-IFA 029 V3.3.1 (2019-11)

GROUP REPORT

Network Functions Virtualisation (NFV) Release 3; Architecture; Report on the Enhancements of the NFV architecture towards "Cloud-native" and "PaaS"

Reference

DGR/NFV-IFA029

Keywords

architecture, cloud, NFV, NFVI

Content

Intellectual Property Rights

Foreword

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

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 studies the potential impact on the NFV architecture of providing "PaaS"-type capabilities and supporting VNFs which follow "cloud-native" design principles, in particular the utilization of container technologies. It describes the related use cases and provides recommendations on the enhancements of the NFV specifications. Management and orchestration of VNFs deployed in containers are analysed and resulting recommendations on the enhancements of the NFV architecture are provided, including impacts on the NFV templates, considering dependencies on the hosting resources.

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-EVE 004 (V1.1.1): "Network Functions Virtualisation (NFV); Virtualisation Technologies; Report on the application of Different Virtualisation Technologies in the NFV Framework".

[i.2] The Twelve-Factor App.

NOTE: Available at https://12factor.net.

[i.3] ETSI GS NFV-IFA 013: "Network Functions Virtualisation (NFV) Release 2; Management and Orchestration; Os-Ma-Nfvo reference point - Interface and Information Model Specification".

[i.4] ETSI GS NFV-IFA 004: "Network Functions Virtualisation (NFV) Release 2; Acceleration Technologies; Management Aspects Specification".

[i.5] 3GPP TS 23.501 (V15.0.0) (2017-12): "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; System Architecture for the 5G System; Stage 2 (Release 15)".

[i.6] ETSI GS NFV-REL 003 (V1.1.2): "Network Functions Virtualisation (NFV); Reliability; Report on Models and Features for End-to-End Reliability".

[i.7] ETSI GS NFV-EVE 011: "Network Functions Virtualisation (NFV) Release 3; Virtualised Network Function; Specification of the Classification of Cloud Native VNF implementations".

[i.8] ETSI GS NFV 003: "Network Functions Virtualisation (NFV); Terminology for main concepts in NFV".

[i.9] Charter statement of CNCF®.

NOTE: Available at https://www.cncf.io/about/charter.

[i.10] Cloud Native Trail Map by CNCF®.

NOTE: Available at https://github.com/cncf/landscape/blob/master/README.md#trail-map.

[i.11] OCI™ Container Image Specification.

NOTE: Available at http://www.github.com/opencontainers/image-spec.

[i.12] OCI™ Container Runtime Specification.

NOTE: Available at http://www.github.com/opencontainers/runtime-spec.

[i.13] NIST Special Publication 800-146: "Cloud Computing Synopsis and Recommendations", 2012.

NOTE: Available at https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-146.pdf.

[i.14] Cloud Foundry documentation.

NOTE: Available at https://docs.cloudfoundry.org/services/overview.html.

[i.15] ETSI GS NFV-IFA 006 (V3.1.1): "Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Vi-Vnfm reference point - Interface and Information Model Specification".

[i.16] ETSI GR NFV-IFA 015 (V3.1.1): "Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Report on NFV Information Model".

[i.17] OpenStack®: "Heat Orchestration Template (HOT) specification".

NOTE: Available at https://docs.openstack.org/heat/latest/template_guide/hot_spec.html.

[i.18] ETSI GS NFV-IFA 005 (V3.1.1): "Network Functions Virtualisation (NFV) Release 3; Management and Orchestration; Or-Vi reference point - Interface and Information Model Specification".

[i.19] OpenStack® Zun API.

NOTE: Available at https://developer.openstack.org/api-ref/application-container/.

[i.20] OpenStack®: "Leveraging Containers and OpenStack".

NOTE: Available at https://www.openstack.org/containers/leveraging-containers-and-openstack.

[i.21] OpenStack® Magnum API.

NOTE: Available at https://developer.openstack.org/api-ref/container-infrastructure-management/.

[i.22] ETSI GS NFV-SEC 012 (V3.1.1): "Network Functions Virtualisation (NFV) Release 3; Security; System architecture specification for execution of sensitive NFV components".

[i.23] IETF RFC 6749: "The OAuth 2.0 Authorization Framework".

NOTE: Available at https://tools.ietf.org/html/rfc6749.

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.8] and the following apply:

cloud-native: software design principle with certain properties and a set of non-functional characteristics

NOTE: As defined in ETSI GS NFV-EVE 011 [i.7].

consumer VNF: VNF that consumes services

container image registry: function that stores container images and makes them available to other functions

NOTE: No assumption is made on the location of such a function.

Container Infrastructure Service (CIS): service that provides runtime environment for one or more container Virtualisation technologies

NOTE: Container Infrastructure Service can run on top of a bare metal or hypervisor-based Virtualisation.

Container Infrastructure Service instance: instance providing runtime execution environment for container

EXAMPLE: In Kubernetes® this is a node, which consists of three components: container runtime, kubelet and kube-proxy. In OpenStack® Zun this is a compute node, which consists of two components: container runtime, Zun-compute.

NOTE: Kubernetes® is a registered trademark of The Linux Foundation®. This information is given for the convenience of users of the present document and does not constitute an endorsement by ETSI of the product named.

Container Infrastructure Service Management (CISM): function that manages one or more Container Infrastructure Services

NOTE: It provides mechanisms for lifecycle management of the containers, which are hosting application components as services or functions.

infrastructure resource: resource provided by the infrastructure that can be used by Virtualisation containers

NOTE: Infrastructure resource can either be a virtualised compute, storage, or network resource.

NFV micro-service: atomic service module, delivered as an all-inclusive SW package, that covers a specific and coherent functional scope, is consumable over network interfaces, is managed independently from other micro-services, and runs as a computing process

PaaS service: modular service or a function provided by PaaS to Consumer VNFs

NOTE: A PaaS service can be a VNF Common Service or a VNF Dedicated Service.

Platform as a Service (PaaS): capability provided to the consumer to deploy onto the cloud infrastructure consumer-created or -acquired applications

NOTE: Cloud Computing Services are typically offered to consumers in one of three service models NIST SP 800-146 [i.13], page 2-1 - Infrastructure as a Service (IaaS), Platform as a Service (PaaS) or Software as a Service (SaaS). In particular for PaaS, the consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, platform services, but has control over the deployed applications and possibly over application hosting environment configurations.

Managed Container Infrastructure Object (MCIO): object managed and exposed by the Container Infrastructure Service Management, representing the desired and actual state of a containerized workload, including its requested and allocated infrastructure resources and applicable policies

EXAMPLE: In Kubernetes®, they are Pods, Services, Deployments, ReplicaSets, StatefulSets. In Apache Marathon™ with Mesos, they are Applications, Pods, Groups. In OpenStack® Zun, they are Capsules [i.19].

Managed Container Infrastructure Object Package (MCIOP): aggregate of declarative descriptor and configuration files for multiple Managed Container Infrastructure Objects

EXAMPLE: CNCF® [i.9] specifies the packaging framework Helm™, with the packaging format specified as Helm™ Charts. OpenStack® [i.17] specifies the orchestration framework Heat, with the descriptor format specified as HOT.

service resource: logical resource that can be used directly as constituent entity of a network service or a logical resource that can be used by one or more Managed Container Infrastructure Objects in the context of containerized workloads

NOTE: In the context of a network service, a service resource can be a NS, VNF, PNF, VNFFG or NFP.

EXAMPLE: In the context of containerized workloads, in Kubernetes®, they could be represented by quota assigned to namespaces.

VNF Common Service: modular service or a function with a lifecycle independent from its consumers and that is consumable by either one or multiple services

EXAMPLE: Messaging and storage services in IT PaaS systems, monitoring service, networked (or network-based) services like authentication and synchronization, and encryption service.

VNF Dedicated Service: modular service or a function with a lifecycle dependent on its consumers and that can only be consumed by a specific set of applications or services

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the abbreviations given in ETSI GS NFV 003 [i.8] and the following apply:

NOTE: An abbreviation defined in the present document takes precedence over the definition of the same abbreviation, if any, in ETSI GS NFV 003 [i.8].

5GCN	5G Core Network
CIS	Container Infrastructure Service
CISM	Container Infrastructure Service Management
CNCF®	Cloud Native Computing Foundation®
CNI™	Container Network Interface™
CNM	Container Network Management
COE	Container Orchestration Engine
CRI	Container Runtime Interface
CRM	Container Runtime Management
CRUD	Create, Read, Update and Delete
CSI	Container Storage Interface
CSM	Container Storage Management
NEF	Network Exposure Function
NRF	NF Repository Function
MCIO	Managed Container Infrastructure Object
MCIOP	Managed Container Infrastructure Object Package
OCI™	Open Container Initiative™
PaaS	Platform as a Service
UDSF	Unstructured Data Storage Function
VR	Virtualised Resource

4 Concepts

4.1 Introduction

This clause provides descriptions of the concepts which are used throughout the present document.

NOTE: The descriptions are only meant to illustrate the concepts and do not preclude any solution or implementation.

4.2 PaaS service types

4.2.1 VNF Common Service

4.2.1.1 Purpose

VNF Common Services provide services or functions which can be required by many consumers such as VNFs or other services. They provide a constituent service for composite applications with a functional scope being common to multiple consumers. They are built in a modular way and can be atomic or composed of multiple NFV Micro-Services.

NOTE: The present document uses "common service" and "VNF Common Service" as synonyms.

4.2.1.2 Functional constraints

VNF Common Services can be consumed by any type of VNF or other service.

4.2.1.3 Lifecycle constraints

VNF Common Services can be instantiated independently on the lifecycle of any consumer, i.e. an instance of a VNF Common Service can exist without being consumed. Only in case no instance of the VNF Common Service is available according to a certain consumer's requirement, a VNF Common Service is instantiated.

An instance of a VNF Common Service can be consumed by one of the two following ways:

- By multiple consumer instances at the same time, i.e. an instance of a VNF Common Service can be shared between multiple consumer instances.
- By only one consumer instance at a time, i.e. an instance of a VNF Common Service cannot be shared between consumer instances.

The instance sharing policy is determined by the individual VNF Common Service.

A VNF Common Service instance can only be terminated in case it is not consumed by any consumer.

The instantiation of VNF Common Service depends on the architectural choices being made. The different architectural options are described in clause 7.1.

4.2.1.4 Example

A generic monitoring service provided by the infrastructure, a platform, or framework is an example for a VNF Common Service. It is instantiated by its hosting infrastructure, platform, or framework independently on any consumer. Multiple consumers like VNF instances can consume the same instance by connecting to the monitoring service instance and invoking the provided operations.

Another example of a VNF Common Service would be a "Backing Service" according to the twelve factor application paradigms, see clause A.2.

4.2.2 VNF Dedicated Service

4.2.2.1 Purpose

VNF Dedicated Services provide services or functions which are required by a specific limited set of consumers such as VNFs or other services. They provide a constituent service for composite applications with a specific functional scope being dedicated to a limited set of consumers. They are built in a modular way and can be atomic or composed of multiple NFV Micro-Services.

NOTE: The present document uses "dedicated service" and "VNF Dedicated Service" as synonyms.

4.2.2.2 Functional constraints

VNF Dedicated Services can only be consumed by specific types of VNFs or other services.

4.2.2.3 Lifecycle constraints

VNF Dedicated Services are instantiated or terminated depending on the lifecycle of their consumer, i.e. an instance of a VNF Dedicated Service only exists as long as its consumer instance exists.

An instance of a VNF Dedicated Service can only be consumed by one consumer instance at a time and cannot be shared with other consumer instances.

The instantiation of VNF Dedicated Service depends on the architectural choices being made. The different architectural options are described in clause 7.1.

4.2.2.4 Example

The implementation of a communication protocol stack like the Session Initiation Protocol (SIP) is an example for a VNF Dedicated Service. Only VNFs terminating this protocol are allowed to consume an instance of its implementation, hence the functional constraint on the consumers. Each VNF instance does require its own instance of this protocol service where the exclusive instances are instantiated and terminated upon the instantiation and termination of the consuming VNF.

5 Use Cases

5.1 Examples of PaaS-type capabilities

5.1.1 Introduction to PaaS-type capabilities

5.1.1.0 Introduction

PaaS provides platform capabilities and/or service through common and open APIs.

The overheads of acquiring, setting up, integrating and managing of different NFVI platforms are eliminated when PaaS is used. This allows the developers to focus on building and managing the user experience, because the use of PaaS eliminates the need to set-up, manage and maintain the NFVI platforms.

PaaS capabilities can be invoked and utilized for a service from different NFVI providers and/or administrative domains. However, it may be easier to maintain security and performance if only one provider is used because of the elimination of potential management overhead due to multiple providers.

PaaS can be applied in the operator's NFV environment. The operator provides PaaS services to its customers (e.g. government, bank or enterprises in a vertical industry), to improve the efficiency of their application development workflow by reducing the effort on maintaining the PaaS capabilities built on NFV network. From the interoperability point of view, the support of PaaS-like functionality and its relation to NFV-MANO for virtualised resource management for supporting PaaS services, coordination of lifecycle management between PaaS services and Consumer VNFs which invokes those PaaS services needs to be analysed. This is the main objective for studying PaaS related use cases in the present document.

The most significant advantage of PaaS is the automatic management of the demand for different types and amount of resources for varying load conditions and services' needs without incurring costs for NFVI platform setup and maintenance.

5.1.1.1 Carrier-Grade PaaS

The use of common and open APIs in PaaS environment helps to achieve rapid service instantiation and automation of service management including tenant isolation. This improves the fault tolerance of the PaaS-based systems. PaaS-based applications and services can use these and other utilities for maintaining higher level of service continuity, especially in the case of state-less, container-based workloads.

It is therefore possible to configure and run PaaS systems with higher (five or more nine's, i.e. 99,999 % or higher) performance, availability, resiliency and stability. These are some of the required characteristics of Carrier-Grade PaaS.

5.1.2 PaaS with capability to support VNF Common Services

5.1.2.1 Introduction and goal

To enable the VNF provider to focus more on the business logic of the network function and reduce the complexity, some VNF Common Services can be provided for a Consumer VNF to be invoked via an open service interface, for example, services for messaging, databases, logging etc. The VNF Common Services can be provided by a PaaS. Figure 5.1.2.1-1 shows an example to illustrate Consumer VNFs invoking multiple VNF Common Services in a PaaS. A VNF Common Service instance can be shared by several Consumer VNF instances and a Consumer VNF instance can invoke one or more VNF Common Service instances. A VNF Common Service only provides one type of common function.

In general, the PaaS is independent of the Consumer VNF. Internal services or components of the Consumer VNF are not exposed outside the Consumer VNF and are not regarded as part of the PaaS.

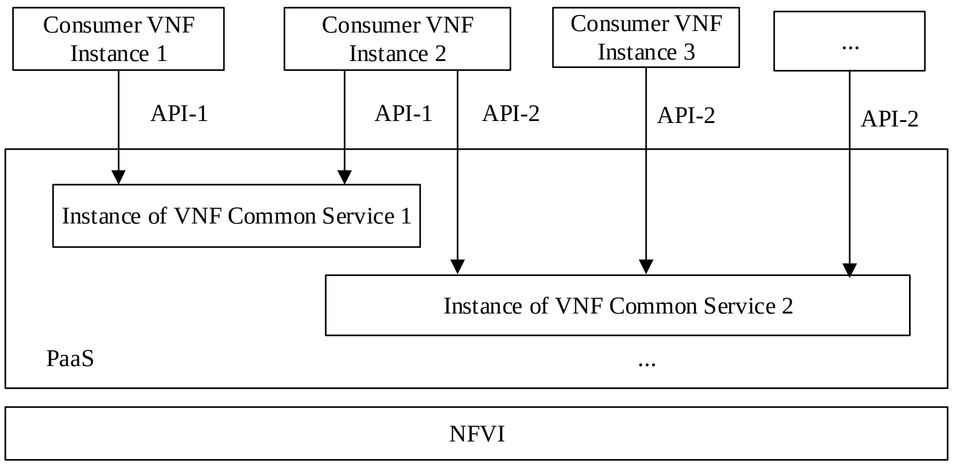


Figure 5.1.2.1-1: An example to illustrate Consumer VNF instances invoke multiple VNF Common Service instances in PaaS

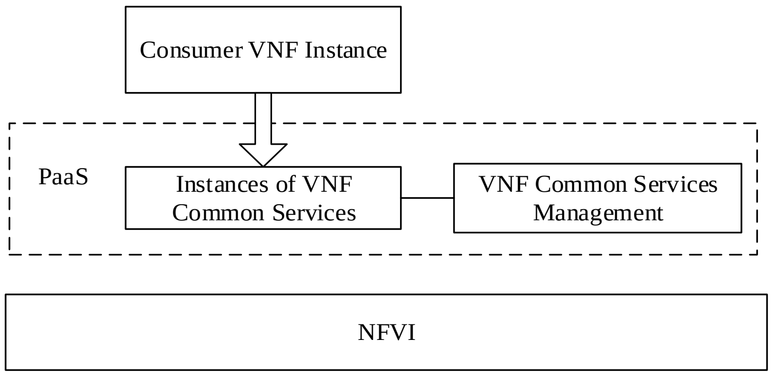
A VNF Common Service can be instantiated any time in a PaaS. The VNF Common Service instance can be pre-deployed by the PaaS before the VNF Common Service instance is requested by a Consumer VNF instance, which means when the PaaS deploys the VNF Common Service instance, it does not know which Consumer VNF instance will invoke it.

VNF Common Service instances are not visible in the NS level.

Figure 5.1.2.1-2 illustrates the PaaS capability to support VNF Common Services. The "Instances of VNF Common Services" and the "VNF Common Services Management" are part of the functions provided by PaaS. VNF Common Service instances are put in relationship with the Consumer VNF instances and these relationships can change dynamically during the life-cycle of VNF Common Service instances. For example, when the Consumer VNF instance is terminated, the relationship with the VNF Common Service instances it was using disappears but these VNF Common Service instances can still exist, because the life-cycle of a VNF Common Service instance is managed independently of the life-cycle of the Consumer VNF instance. When a new VNF Common Service instance is allocated to a Consumer VNF instance, a new relationship is established. "VNF Common Services Management" can use service registration and discovery mechanisms to manage and maintain the relationship.

Consumer VNF instances bind to one or several VNF Common Service instances, and during that binding process authorization for accessing the service needs to be performed. Eventually, the access will need to be authorized more often during the lifetime of the Consumer VNF instance.

PaaS can allocate a VNF Common Service instance or give authorization of accessing the API(s) to a Consumer VNF instance according to the requirements of the Consumer VNF provided when the Consumer VNF instance is deployed. PaaS should provide the Consumer VNF instance with information about the allocated VNF Common Service instance or APIs to guarantee the Consumer VNF instance can access the VNF Common Service instance and use it at runtime. These capabilities of the PaaS can also be provided by "VNF Common Services Management" function. How "VNF Common Services Management" function is implemented is out of scope of the present document.


Consumer VNF Instance
PaaS
Instances of VNF Common Services
VNF Common Services Management
NFVI

NOTE: This figure does not represent any particular architectural option.

Figure 5.1.2.1-2: Illustration of PaaS capability to support VNF Common Services

The goal of this use case is to describe how a VNF Common Service instance is allocated and used by a Consumer VNF instance.

5.1.2.2 Actors and roles

Table 5.1.2.2-1 describes the use case actors and roles.

Table 5.1.2.2-1: PaaS with capability to support VNF Common Services actors and roles

#	Role	Description
1	NFV-MANO system	The NFV Management and Orchestration functional blocks including the NFVO, the VNFM and the VIM.
2	Consumer VNF instance	The VNF instance which uses the VNF Common Service instance.
3	VNF Common Service instance	The service instance, which can be deployed independently and shared by different Consumer VNF instances.
4	VNF Common Services Management	The management function of the VNF Common Service instances, including the lifecycle management of the VNF Common Service instances, the management of the VNF Common Service packages and service registration and discovery, authentication and authorization, etc. How the VNF Common Services are packaged and installed is out of scope of the present document.

5.1.2.3 Pre-conditions

Table 5.1.2.3-1 describes the use case pre-conditions.

Table 5.1.2.3-1: PaaS with capability to support VNF Common Services pre-conditions

#	Pre-condition	Additional description
1	The Consumer VNF instance requires some VNF Common Service instances to provide the desired network functions.	The VNFD contains the information about the dependencies or requirements the Consumer VNF has on VNF Common Services.
2	The VNF Common Service instance, which the Consumer VNF instance needs to invoke, is instantiated in the PaaS.	N/A

5.1.2.4 Post-conditions

Table 5.1.2.4-1 describes the use case post-conditions.

Table 5.1.2.4-1: PaaS with capability to support VNF Common Services post-conditions

#	Post-condition	Additional description
1	The Consumer VNF instance can use the services exposed by the VNF Common Service instances in runtime and the relationship between the Consumer VNF instance and the VNF Common Service instances is established.	N/A

5.1.2.5 Flow description

Table 5.1.2.5-1 describes the use case flow for "PaaS with capability to support VNF Common Services".

Table 5.1.2.5-1: PaaS with capability to support VNF Common Services flow description

#	Actor/Role	Action/Description
Begins when	NFV-MANO	NFV-MANO is requested to instantiate a Consumer VNF which requires to invoke one or more VNF Common Service instances.
Step 1	NFV-MANO	NFV-MANO instantiates the Consumer VNF.
Step 2	NFV-MANO-> VNF Common Services Management	NFV-MANO requests the VNF Common Services Management to allocate a VNF Common Service instance. The step can be repeated in order to allocate several VNF Common Service instances.
Step 3	VNF Common Services Management	Common Services Management allocates a VNF Common Service instance according to the requirements from the NFV-MANO and returns the related information of the allocated VNF Common Service instance to NFV-MANO and to the Consumer VNF instance. The step can be repeated in order to allocate several VNF Common Service instances.
Ends when	Consumer VNF	The Consumer VNF instance can access the VNF Common Service instances and invoke them at runtime.

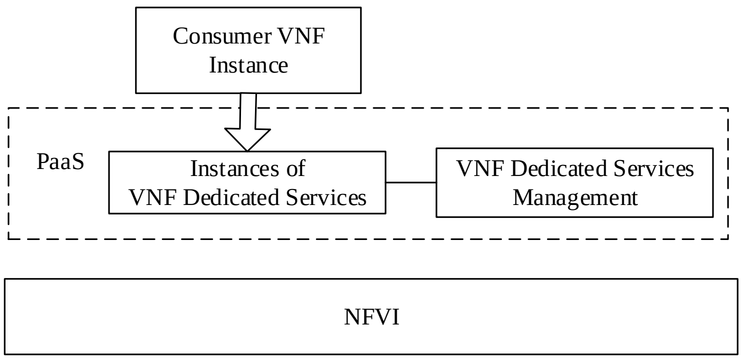
5.1.3 PaaS with capability to support VNF Dedicated Services

5.1.3.1 Introduction and goal

Besides the VNF Common Services, PaaS can also provide VNF Dedicated Services. A Consumer VNF instance can use one or more VNF Dedicated Service instances, but every VNF Dedicated Service instance can only be consumed by a limited set of Consumer VNF instances. A VNF Dedicated Service only provides one type of dedicated function.

Figure 5.1.3.1-1 illustrates the PaaS capability to support VNF Dedicated Services, the "Instances of VNF Dedicated Services" and the "VNF Dedicated Services Management" are part of the functions provided by the PaaS. The "VNF Dedicated Services Management" function is responsible for the lifecycle management of the VNF Dedicated Service instances.

A VNF Dedicated Service instance is created and assigned by the PaaS during the Consumer VNF instantiation when the VNF Dedicated Service is needed. The lifecycle of the VNF Dedicate Service instance is bound to the lifecycle of the Consumer VNF instance, which means when the Consumer VNF instance is terminated, all the VNF Dedicated Service instances, which the Consumer VNF instance invoked, should be terminated too. The relationship between the Consumer VNF instance and the VNF Dedicated Service instance can be monitored and maintained by the "VNF Dedicated Services Management" function.


Consumer VNF Instance
PaaS
Instances of VNF Dedicated Services
VNF Dedicated Services Management
NFVI

NOTE: This figure does not represent any particular architectural option.

Figure 5.1.3.1-1: Illustration of PaaS capability to support VNF Dedicated Services

The goal of this use case is to describe how VNF Dedicated Service instances are created and used by a Consumer VNF instance.

5.1.3.2 Actors and roles

Table 5.1.3.2-1 describes the use case actors and roles.

Table 5.1.3.2-1: PaaS with capability to support VNF Dedicated Services actors and roles

#	Role	Description
1	NFV-MANO system	The NFV Management and Orchestration functional blocks including the NFVO, the VNFM and the VIM.
2	Consumer VNF instance	The VNF which uses the VNF Dedicated Services.
3	Dedicated Service instance	The service instances which can be deployed independently and consumed by one Consumer VNF instance.
4	VNF Dedicated Services Management	The management function of the VNF Dedicated Service instances, including the lifecycle management of the VNF Dedicated Service instances, the management of the VNF Dedicated Service packages, etc.

5.1.3.3 Pre-conditions

Table 5.1.3.3-1 describes the use case pre-conditions.

Table 5.1.3.3-1: PaaS with capability to support VNF Dedicated Services pre-conditions

#	Pre-condition	Additional description
1	The Consumer VNF instance requires some VNF Dedicated Service instances to provide the desired network functions.	The VNFD contains the information about the dependencies or requirements the Consumer VNF has on VNF Dedicated Services.

5.1.3.4 Post-conditions

Table 5.1.3.4-1 describes the use case post-conditions.

Table 5.1.3.4-1: PaaS with capability to support VNF Dedicated Services post-conditions

#	Post-condition	Additional description
1	The Consumer VNF instance can use the services exposed by the VNF Dedicated Service instances at runtime and the relationship between the Consumer VNF instance and the VNF Dedicated Service instance is established.	N/A

5.1.3.5 Flow description

Table 5.1.3.5-1 describes the use case flow for "PaaS with capability to support VNF Dedicated Services".

Table 5.1.3.5-1: PaaS with capability to support VNF Dedicated Services flow description

#	Actor/Role	Action/Description
Begins when	NFV-MANO	NFV-MANO is requested to instantiate a Consumer VNF, which requires to invoke one or more VNF Dedicated Service instances.
Step 1	NFV-MANO	NFV-MANO instantiates the Consumer VNF.
Step 2	NFV-MANO -> VNF Dedicated Services Management	NFV-MANO requests the VNF Dedicated Services Management to create a VNF Dedicated Service instance. The step can be repeated in order to allocate several VNF Dedicated Service instances.
Step 3	VNF Dedicated Services Management	VNF Dedicated Services Management creates a VNF Dedicated Service instance according to the requirements from the NFV-MANO and returns the related information of the created VNF Dedicated Service instance to NFV-MANO and to the Consumer VNF instance. The step can be repeated in order to allocate several VNF Dedicated Service instances.
Ends when	Consumer VNF	The Consumer VNF instance can access the VNF Dedicated Service instances and invoke them at runtime.

5.1.4 PaaS with capability supporting container-based service

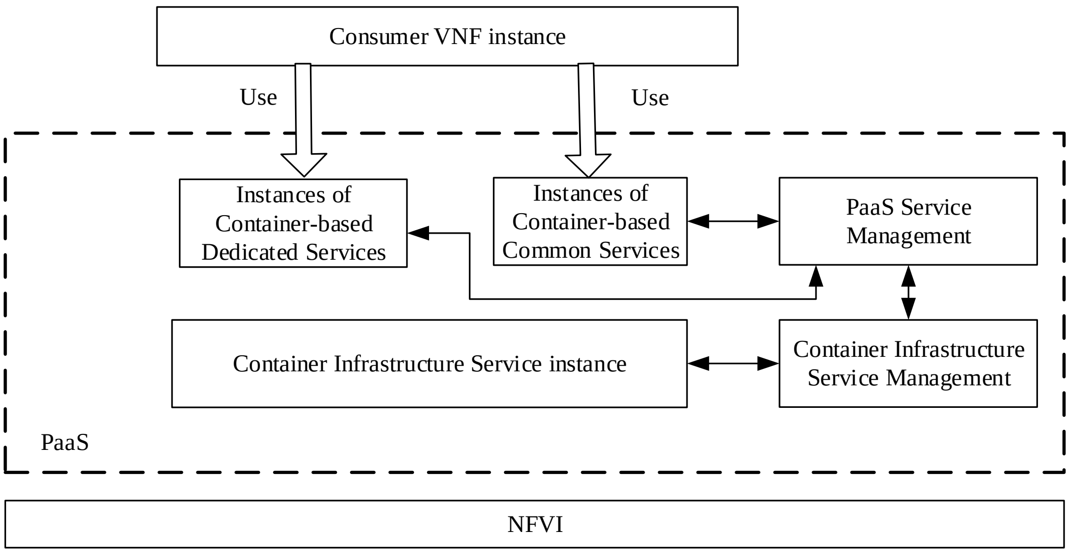
5.1.4.1 Introduction and goal

Container-based virtualisation (see ETSI GS NFV-EVE 004 [i.1]), also called operating system (OS)-level virtualisation, comparing with the hypervisor-based Virtualisation (e.g. VM), would be lighter for smaller image and memory consumption, faster for quick deployment and start-up and more agile for easy shipment, installation and migration.

NOTE: PaaS does not imply container-based virtualisation and containers-based virtualisation does not imply PaaS. The appearance of these two terms in clause 5.1.4 does not imply that any dependency exists. It simply represents an example.

In case the NFVI only supports the hypervisor-based virtualisation technologies, PaaS can offer the container-based service by nesting container virtualisation technologies in hypervisor-based virtualisation technologies. This will facilitate the PaaS to provide the container-based services, which may be VNF Common Services or VNF Dedicated Services. VNF Common Services deployed using containers are termed "container-based Common Services" in the present document. Dedicated services deployed using containers are termed "container-based Dedicated Service" in the present document.

Figure 5.1.4.1-1 gives an example to illustrate how PaaS supports the container-based services and how a VNF can use the container-based VNF Common Services and container-based VNF Dedicated Services provided by the PaaS. The Container Infrastructure Service is the execution environment for the container cluster where the container-based services run. The Container Infrastructure Service Management is responsible for the infrastructure resource management and lifecycle management of the execution environment for container cluster. PaaS Service Management includes VNF Common Services Management and VNF Dedicated Services Management.


Consumer VNF instance
Use
Use
Instances of Container-based Dedicated Services
Instances of Container-based Common Services
PaaS Service Management
Container Infrastructure Service instance
Container Infrastructure Service Management
PaaS
NFVI

NOTE: This figure does not represent any particular architectural option.

Figure 5.1.4.1-1: Example to illustrate PaaS support for the container-based services

The goal of the use case is to describe how to deploy a Consumer VNF instance, which uses container-based Common Service instances and container-based Dedicated Service instances provided by the PaaS.

5.1.4.2 Actors and roles

Table 5.1.4.2-1 describes the use case actors and roles.

Table 5.1.4.2-1: PaaS with capability supporting container-based services actors and roles

#	Role	Description
1	NFV-MANO	NFV Management and Orchestration functional blocks including the NFVO, the VNFM and the VIM.
2	PaaS Service Management	The management function of the PaaS service instances, including the lifecycle management of the container-based service instances, the lifecycle management of the service packages, service registration and discovery, authentication and authorization, etc.
3	Consumer VNF instance	VNF instance, which uses container-based Common Service instances and container-based Dedicated Service instances

5.1.4.3 Pre-conditions

Table 5.1.4.3-1 describes the use case pre-conditions.

Table 5.1.4.3-1: PaaS with capability supporting container-based services pre-conditions

#	Pre-condition	Additional description
1	The PaaS is running and the virtualised resources which can be used by PaaS are allocated.	N/A
2	The container-based Common Service instance(s) are instantiated.	N/A
3	The Consumer VNF instance consumes some container-based Common Service instances and container-based Dedicated Service instances to provide the desired network functions.	The VNFD contains the information about the dependencies or requirements, which the Consumer VNF has on container-based Common and Dedicated Services.

5.1.4.4 Post-conditions

Table 5.1.4.4-1 describes the use case post-conditions.

Table 5.1.4.4-1: PaaS with capability supporting container-based services post-conditions

#	Post-condition	Additional description
1	The required Consumer VNF instance is deployed and running.	N/A

5.1.4.5 Flow description

Table 5.1.4.5-1 describes the use case flow for "PaaS with capability supporting container-based service".

Table 5.1.4.5-1: PaaS with capability supporting container-based services flow description

#	Actor/Role	Action/Description
Begins when	NFV-MANO	NFV-MANO needs to instantiate a Consumer VNF, which has to invoke one or more container-based Common Service instances and container-based Dedicated Service instances.
Step 1	NFV-MANO	NFV-MANO instantiate the Consumer VNF.
Step 2	NFV-MANO-> PaaS Service Management	NFV-MANO requests the PaaS Service Management to allocate the container-based Common Service instances to the Consumer VNF instance.
Step 3	PaaS Service Management	The PaaS Service Management allocates the container-based Common Service instances to the Consumer VNF instance.
Step 4	NFV-MANO-> PaaS Service Management	NFV-MANO requests PaaS Service Management to create and allocate the container-based Dedicated Service instances for the Consumer VNF instance.
Step 5	PaaS Service Management	The PaaS Service Management creates the container-based Dedicated Service instances for the Consumer VNF instances.
Step 6	PaaS Service Management	The PaaS Service Management allocates the container-based Dedicated Service instances to the Consumer VNF instance.
Ends when	Consumer VNF	The Consumer VNF instance is deployed and runs normally.

5.1.5 PaaS with capability supporting dedicated container service

5.1.5.1 Introduction and goal

When the PaaS offers dedicated container service on top of the NFVI using nested virtualisation, as it is discussed in clause 5.1.4, the PaaS provides Container Infrastructure Service to the Consumer VNF directly without using the higher level dedicated or common services of the PaaS.

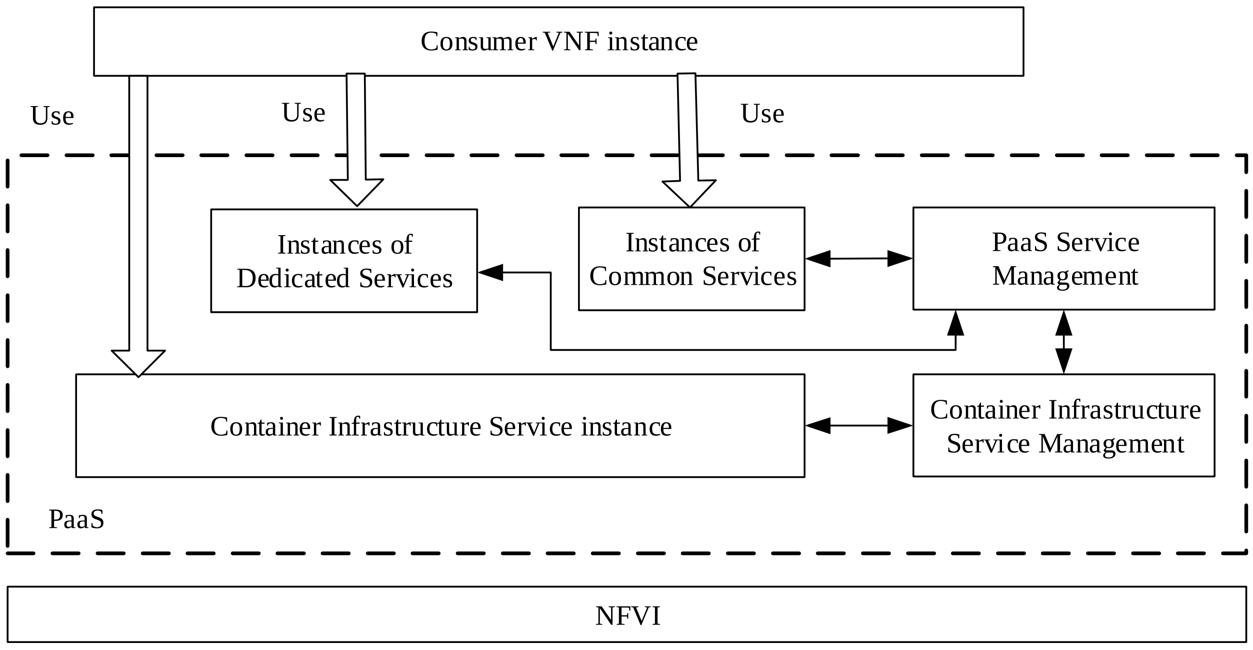
The Container Infrastructure Service provides the ability for Consumer VNFs to execute any capable software inside containers. The Container Infrastructure Service is running in parent virtualisation containers using the infrastructure resources of the NFVI and managed by the VIM. The Consumer VNF, when it uses the dedicated container service, may use containers which are started inside parent virtualisation containers provided by the Container Infrastructure Service. Usage of dedicated container services (e.g. the Container Infrastructure Service Instance in figure 5.1.5.1-1), VNF Dedicated and VNF Common Services are independent from each other.

The main functions that can be consumed from the Container Infrastructure Service are:

- Deployment of a container image on top of the shared OS kernel, in a parent virtualisation container or on bare metal.
- Resource management for the parent virtualisation container, that is allocating virtual machine or bare metal resources as required and deploy the basic software that is needed on that node for the deployment of OS-containers.
- Setup and make available the kernel services provided by the shared OS and described in ETSI GS NFV-EVE 004 [i.1], clause 4.3.
- Configure the network of the containers according to the needs of the Consumer VNF.

NOTE: The Container Infrastructure Service typically deploys an OS-container image in a virtualisation container (provided by the NFVI as a VM). But in some cases, the OS-containers will not be deployed using nested virtualisation, but on bare metal for performance reasons.

Figure 5.1.5.1-1 gives an example to illustrate how the Consumer VNF instance uses the dedicated container service instances, as well as VNF Common Service instances and VNF Dedicated Service instances provided by the PaaS. The Container Infrastructure Service instance is the execution environment for the container cluster where the Consumer VNF instance can run in. The Container Infrastructure Service Management is responsible for the infrastructure resource management of the execution environment and the lifecycle management of the containers running in the container cluster. PaaS Service Management includes VNF Common Service Management and VNF Dedicated Service Management functions.


Consumer VNF instance
Use
Use
Use
Instances of Dedicated Services
Instances of Common Services
PaaS Service Management
Container Infrastructure Service instance
Container Infrastructure Service Management
PaaS
NFVI

NOTE: This figure does not represent any particular architectural option.

Figure 5.1.5.1-1: Example to illustrate PaaS support for container services

The goal of the use case is to describe how to deploy a Consumer VNF instance, which uses dedicated container service instances provided by the PaaS.

5.1.5.2 Actors and roles

Table 5.1.5.2-1 describes the use case actors and roles.

Table 5.1.5.2-1: PaaS with capability supporting dedicated container services actors and role

#	Role	Description
1	NFV-MANO	NFV Management and Orchestration functional blocks including the NFVO, the VNFM and the VIM without the Container Infrastructure Service Management functionality.
2	Container Infrastructure Service Management	Management of the containers and their runtime environment. This can be a separate functional block or be integrated in an NFV-MANO component.
3	Consumer VNF instance	VNF instance, which uses Container Infrastructure Service instance.

5.1.5.3 Pre-conditions

Table 5.1.5.3-1 describes the use case pre-conditions.

Table 5.1.5.3-1: PaaS with capability supporting dedicated container services pre-conditions

#	Pre-condition	Additional description
1	The PaaS is running and the virtualised resources which can be used by PaaS have been allocated.	N/A
2	The Container Infrastructure Service instance has been instantiated.	See note 1.
3	Container images of the Consumer VNF instance are onboarded.	See note 2.
NOTE 1: It is also possible to initiate the Container Infrastructure Service instance on demand, as part of the initiation flow of the Consumer VNF instance, which needs the Container Infrastructure Service instance. NOTE 2: There are different options for the location for container image onboarding.		

5.1.5.4 Post-conditions

Table 5.1.5.4-1 describes the use case post-conditions.

Table 5.1.5.4-1: PaaS with capability supporting dedicated container services post-conditions

#	Post-condition	Additional description
1	The required Consumer VNF instances is deployed and running.	N/A

5.1.5.5 Flow description

Table 5.1.5.5-1 describes the use case flow for "PaaS with capability supporting dedicated container-based service".

Table 5.1.5.5-1: PaaS with capability supporting dedicated container-based services flow description

#	Actor/Role	Action/Description
Begins when	NFV-MANO	NFV-MANO identifies a need to instantiate a Consumer VNF, which consists of containers, needs to use dedicated container services.
Step 1	NFV-MANO	NFV-MANO starts to instantiate the Consumer VNF.
Step 2	NFV-MANO-> Container Infrastructure Service Management	NFV-MANO requests the Container Infrastructure Service Management to provide the Container Infrastructure Service instance to the Consumer VNF instance.
Step 3	Container Infrastructure Service Management	Container Infrastructure Service Management initiates the provision of the Container Infrastructure Service instance to the Consumer VNF instance by instantiating containers of the Consumer VNF in the dedicated container service.
Ends when	Consumer VNF instance	The Consumer VNF instance is deployed and runs normally.

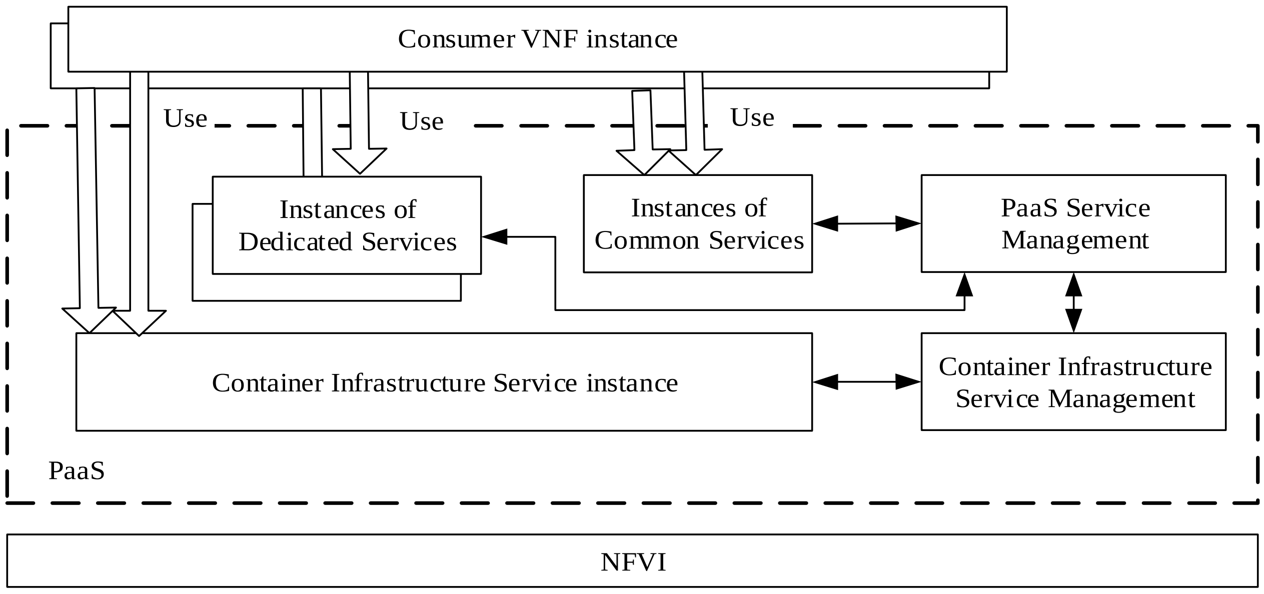
5.1.6 PaaS with capability supporting shared container service

5.1.6.1 Introduction and goal

When the PaaS offers shared container service on top of the NFVI using nested virtualisation, as it is discussed in clause 5.1.4, the PaaS provides Container Infrastructure Service to several Consumer VNFs directly without using the higher level dedicated or common services of the PaaS.

The Container Infrastructure Service provides the ability for Consumer VNFs to execute any capable software inside containers. The Container Infrastructure Service is running in parent virtualisation containers using the infrastructure resources of the NFVI and managed by the VIM, but at the same time several Consumer VNFs are capable to use the same Container Infrastructure Service. The Consumer VNF, when it uses the shared container service, may use containers which are started inside the parent virtualisation containers provided by the Container Infrastructure Service. Usage of shared container services (e.g. the Container Infrastructure Service Instance in figure 5.1.6.1-1), VNF Dedicated Services and VNF Common Services are independent from each other.

Figure 5.1.6.1-1 gives an example to illustrate how PaaS supports the shared container service and how the Consumer VNF instances can use the shared container service instance along with the common and dedicated services instances provided by the PaaS. The Container Infrastructure Service instance is the execution environment for the container cluster where the Consumer VNF instances can run in. The Container Infrastructure Service Management is responsible for the infrastructure resource management of the execution environment for and the lifecycle management of the containers running in the container cluster. PaaS Service Management includes VNF Common Service Management and VNF Dedicated Service Management functions.


Consumer VNF instance
Use
Use
Use
Instances of Dedicated Services
Instances of Common Services
PaaS Service Management
Container Infrastructure Service instance
Container Infrastructure Service Management
PaaS
NFVI

NOTE: This figure does not represent any particular architectural option.

Figure 5.1.6.1-1: Example to illustrate PaaS support for the shared container services

The goal of the use case is to describe how to deploy Consumer VNF instances, which use the shared container service instances provided by the PaaS.

5.1.6.2 Actors and roles

Table 5.1.6.2-1 describes the use case actors and roles.

Table 5.1.6.2-1: PaaS with capability supporting shared container services actors and role

#	Role	Description
1	NFV-MANO	NFV Management and Orchestration functional blocks including the NFVO, the VNFM and the VIM without the Container Infrastructure Service Management functionality.
2	Container Infrastructure Service Management	Management of the containers and their runtime environment. This can be a separate functional block or be integrated in an NFV-MANO component.
3	Consumer VNF instance	VNF instance which uses Container Infrastructure Service instance.

5.1.6.3 Pre-conditions

Table 5.1.6.3-1 describes the use case pre-conditions.

Table 5.1.6.3-1: PaaS with capability supporting shared container services pre-conditions

#	Pre-condition	Additional description
1	The PaaS is running and the virtualised resources which can be used by PaaS have been allocated.	N/A
2	The Container Infrastructure Service instance has been instantiated.	See note 1.
3	Container images of the Consumer VNF instance are onboarded.	See note 2.
NOTE 1: It is also possible to initiate the Container Infrastructure Service instance on demand, as part of the initiation flow of the Consumer VNF instance which needs the Container Infrastructure Service instance. NOTE 2: There are different options for the location for Container image onboarding.		

5.1.6.4 Post-conditions

Table 5.1.6.4-1 describes the use case post-conditions.

Table 5.1.6.4-1: PaaS with capability supporting shared container services post-conditions

#	Post-condition	Additional description
1	All required Consumer VNF instances are deployed and running.	N/A

5.1.6.5 Flow description

Table 5.1.6.5-1 describes the use case flow for "PaaS with capability supporting shared container services".

Table 5.1.6.5-1: PaaS with capability supporting shared container services flow description

#	Actor/Role	Action/Description
Begins when	NFV-MANO	NFV-MANO identifies a need to instantiate a Consumer VNF, which consists of containers, needs to use shared container services.
Step 1	NFV-MANO	NFV-MANO starts to instantiate the Consumer VNF.
Step 2	NFV-MANO-> Container Infrastructure Service Management	NFV-MANO requests the Container Infrastructure Service Management to provide the Container Infrastructure Service instance to the Consumer VNF instance.
Step 3	Container Infrastructure Service Management	Container Infrastructure Service Management initiates the provision of the Container Infrastructure Service instance to the Consumer VNF instance by instantiating containers of the Consumer VNF in the shared container service.
Ends when	Consumer VNF instance	The Consumer VNF instance is deployed and runs normally.

5.1.7 5G Core Network Use Cases

5.1.7.1 Overview and Background

The following description is based on the 5G system architecture as described in 3GPP TS 23.501 [i.5]. In general, the service-based architecture specified by 3GPP for the 5G system is addressing similar key principles than the present document for NFV. In more detail the following concepts from clause 4.1 in [i.5] are further elaborated on in the following, other aspects are out-of-scope for the present document or are for further study:

- Modularize the function design.
- Wherever applicable, define procedures (i.e. the set of interactions between network functions) as services, so that their re-use is possible.
- Enable each Network Function (NF) to interact with other NF directly if required. The architecture does not preclude the use of an intermediate function to help route Control Plane messages.

- Support a unified authentication framework.
- Support "stateless" NFs, where the "compute" resource is decoupled from the "storage" resource.
- Support capability exposure.

In clause 4.2 of 3GPP TS 23.501 [i.5] the architecture reference model is described and for reference the non-roaming case is show below. For a specific description please refer to 3GPP TS 23.501 [i.5].

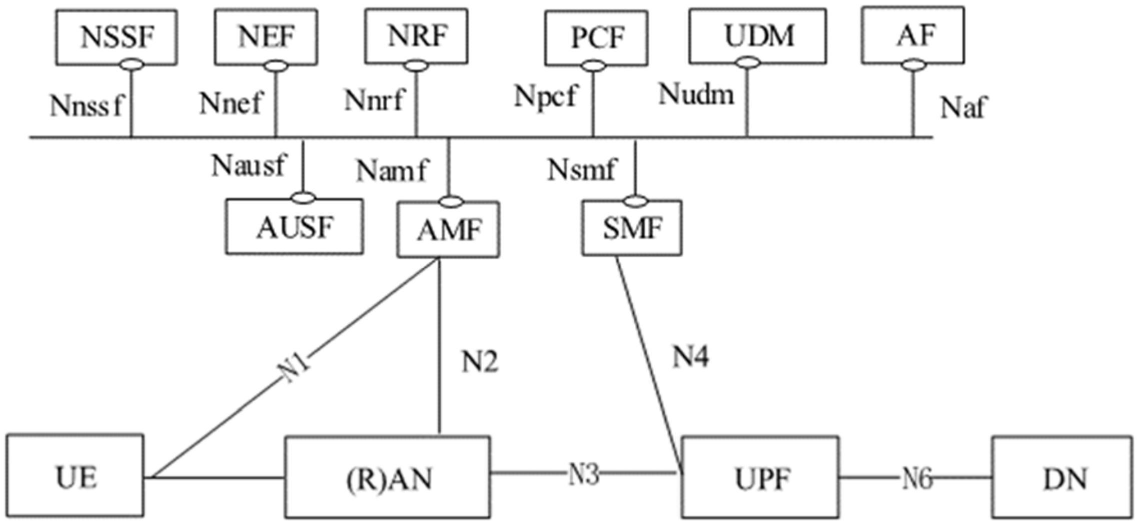


Figure 5.1.7.1-1: (Non-roaming) 5G System Service-based architecture (see figure 4.2.3-1 in 3GPP TS 23.501 [i.5])

Potential Common PaaS Service Features vs. 5GC network functions

The 5GCN is defining a set of NFs which are common to the 5G system and are consumed by many other 5G NFs. Those NFs can be implemented and deployed by VNFs exposing them as a set of common services. This has the benefit that these common services are exposed for general-purpose and can be consumed by other functions (including non 5G functions). The following functions are primary examples of exposing them as VNF common service functions. However, it needs to be noted that 3GPP specifies the 5GC NFs on a logical level which does not preclude any solution on how they are grouped, implemented and deployed as VNFs, including utilization of potential PaaS Services.

Communication bus/system: From figure 5.1.7.1-1, one feature is the communication between different NFs of the 5G core system, which can be implemented in different ways. A service framework enabling the inter-NF communication can be deployed by VNFs exposing common PaaS communication service features.

Storage: 3GPP TS 23.501 [i.5] in clause 4.2.5 is postulating the use of an unstructured data storage function (UDSF). Basically, this is a storage service with an interface defined by 3GPP, however the data structures to be stored are not standardized. Refer to clause 4.2.5 of 3GPP TS 23.501 [i.5] for a detailed description of this function. This storage service is meant to be consumed by various 5GC NFs. Since this storage service is separate from the NFs, it can be modelled and deployed as a VNF providing this common storage service, which can be consumed by other deployed VNFs.

NF exposed API/interfaces: From figure 5.1.7.1-1, the interfaces/APIs of the different 5G NF services need to be discovered and connected with each other. For example, at the time of invoking a NF service there needs to be a function to discover the API or interface of the required services of other NFs and the knowledge about how to connect to those interfaces is needed. Also prior to invoking a NF service, there might need to be a way of checking whether all dependencies are resolvable (whether the required NF services are installed and/or instantiated). (This function can be seen as part of the NF Repository Function (NRF) described in 3GPP TS 23.501 [i.5]). Again, a service framework enabling the NF service discovery can be implemented by VNFs exposing common PaaS communication service features.

Network Exposure Function (NEF): This function exposes certain NF capabilities of the 5G system to consumer applications internal or external to the 5G system. External exposure can be categorized as monitoring capability, provisioning capability, policy/charging capability and other capabilities to be defined during the further development of the 5G eco-system. Consumer applications can be any type of NFs using some of the exposed NF capabilities of the 5G network.

5.1.7.2 Use Case: 3rd Party Applications using Network Functions

5.1.7.2.1 Actors and roles

Table 5.1.7.2.1-1 describes the use case actors and roles.

Table 5.1.7.2.1-1: 3rd Party Applications using Network Functions actors and roles

#	Role	Description
1	3rd party application (Consumer VNF)	3rd party applications using VNF Common Services and specific 5G network functions exposed by the VNF(s).
2	PaaS	The platform to offer the capability required for many applications. Specifically, some storage and communication services for components of 3rd party application are needed since the developer does not want to re-implement them.
3	VNF	A VNF exposing particular functionality which is used by the 3rd party application (Consumer VNF).

5.1.7.2.2 Pre-conditions

Table 5.1.7.2.2-1 describes the use case pre-conditions.

Table 5.1.7.2.2-1: 3rd Party Applications using Network Functions pre-conditions

#	Pre-condition	Additional description
1	Contractual relationship between platform provider and developer agreed	N/A
2	3rd party application (Consumer VNF) invoking VNF Common Services and certain functions exposed by other VNFs is ready to be instantiated.	The VNF has well-defined interfaces (eventually standardized interfaces, like the NEF as defined in 3GPP). See also note 1.
3	The VNF to be used is instantiated and executing.	The interfaces exposed are discoverable and available for users to connect. See also note 2.
NOTE 1: The information about the required VNF Common Services and VNFs used are known at instantiation time of the 3rd party application in electronically readable format. Interfaces exposed by VNF Common Services might not be standardized but need to be known by the 3 rd party application. NOTE 2: There are several ways to call a service exposed by a VNF, therefore several options for connecting to the VNF are available and should be supported by the platform.		

5.1.7.2.3 Post-conditions

Table 5.1.7.2.3-1 describes the use case post-conditions.

Table 5.1.7.2.3-1: 3rd Party Applications using Network Functions post-conditions

#	Post-condition	Additional description
1	The 3rd party Application (Consumer VNF) is installed and executed.	N/A

5.1.7.2.4 Flow description

Table 5.1.7.2.4-1 describes the use case flow for "3rd Party Applications using Network Functions".

Table 5.1.7.2.4-1: 3rd Party Applications using Network Functions flow description

#	Actor/Role	Action/Description
Begins when	3rd party application	Delivery of 3rd party application package and request to instantiate the 3rd party application.
Step1	PaaS	3rd party application gets configured such that the VNF Common Services and the interface end-point of the VNF to be used is known and can be used.
Step2	3rd party application	Uses the VNF Common Services of the VNF over the exposed interfaces.
Step3	PaaS	In case of failures of VNF Common Services or the used VNF, initiate failure recovery actions.
Ends when	3rd party application	3rd party application has recovered from detected failures and continues to use the common services of the VNF over the exposed interfaces.

5.1.8 PaaS with capability supporting a service descriptor catalogue

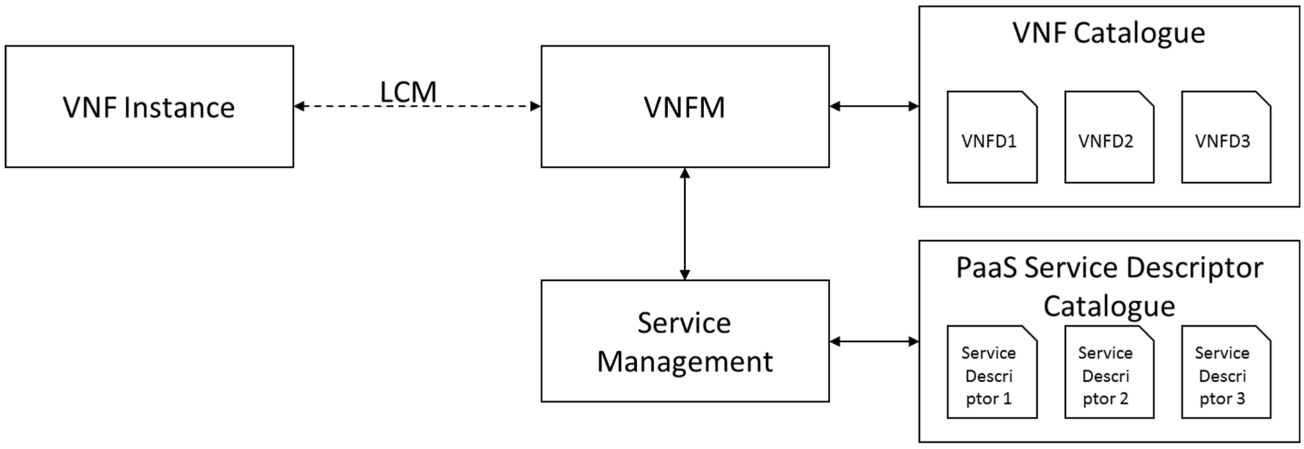
5.1.8.1 Introduction and goal

This use case introduces a PaaS service descriptor catalogue as repository for PaaS service descriptors. The PaaS service descriptors describe the services provided by the PaaS, so these services can be managed by a Service Management component.

The PaaS service descriptor catalogue contains descriptors for individual PaaS services. They are referenced by multiple VNF Descriptors to enable their re-use for instantiating multiple VNFs. During instantiation of a VNF, the service descriptors are used to substitute the references in the VNFD (e.g. TOSCA node template substitution). The PaaS service descriptors can be onboarded by extraction from a dedicated VNF package for PaaS services or can be provided by the PaaS environment hosting the PaaS service descriptor catalogue. The capabilities of all services with descriptors available from the catalogue are discoverable by service management functions.

This use case only describes the concept of the PaaS service descriptor catalogue and does not describe potential combinations with other uses cases described in the present document, such as types of the PaaS services. However, it is assumed that VNFs referencing service descriptors follow the concepts of the PaaS service types as described in clause 4.2.

The concept of template substitution only relates to VNF Descriptors and does not have any dependency or relationship to network service (NS) descriptors or packages.



NOTE: This figure does not represent any particular architectural option.

Figure 5.1.8.1-1: PaaS service descriptor catalogue relationship

Figure 5.1.8.1-1 illustrates the relationship of the PaaS service descriptor catalogue to other functional entities involved in the life-cycle management of VNFs using PaaS service descriptors.

The PaaS service descriptors from the catalogue are used during instantiation of VNFs which descriptors contain references or requirements on PaaS services. Before executing the instantiation of those VNFs, the VNFM requests the service management to discover the required PaaS service descriptors from the catalogue and replace the reference in the VNF Descriptor with the PaaS service descriptor by means of node template substitution. That is, the VNFM will use a temporary VNFD with substituted references for the instantiation of the VNFs.

NOTE 1: Certain scenarios might not require instantiating some of the VNFCs described by the substituted service references, for example in the case of representing a VNF Common Service. The service management analyses the fetched service descriptors and detects which VNFCs to be instantiated.

NOTE 2: The concept of using temporary VNFD's, which are not persistent requires a mechanism to retain traceability on which service descriptor was used during instantiation. A solution to this is not part of the present document.

Figure 5.1.8.1-2 shows an example for the stages of creating the temporary VNFD and instantiating the VNF using PaaS service descriptors from the catalogue.

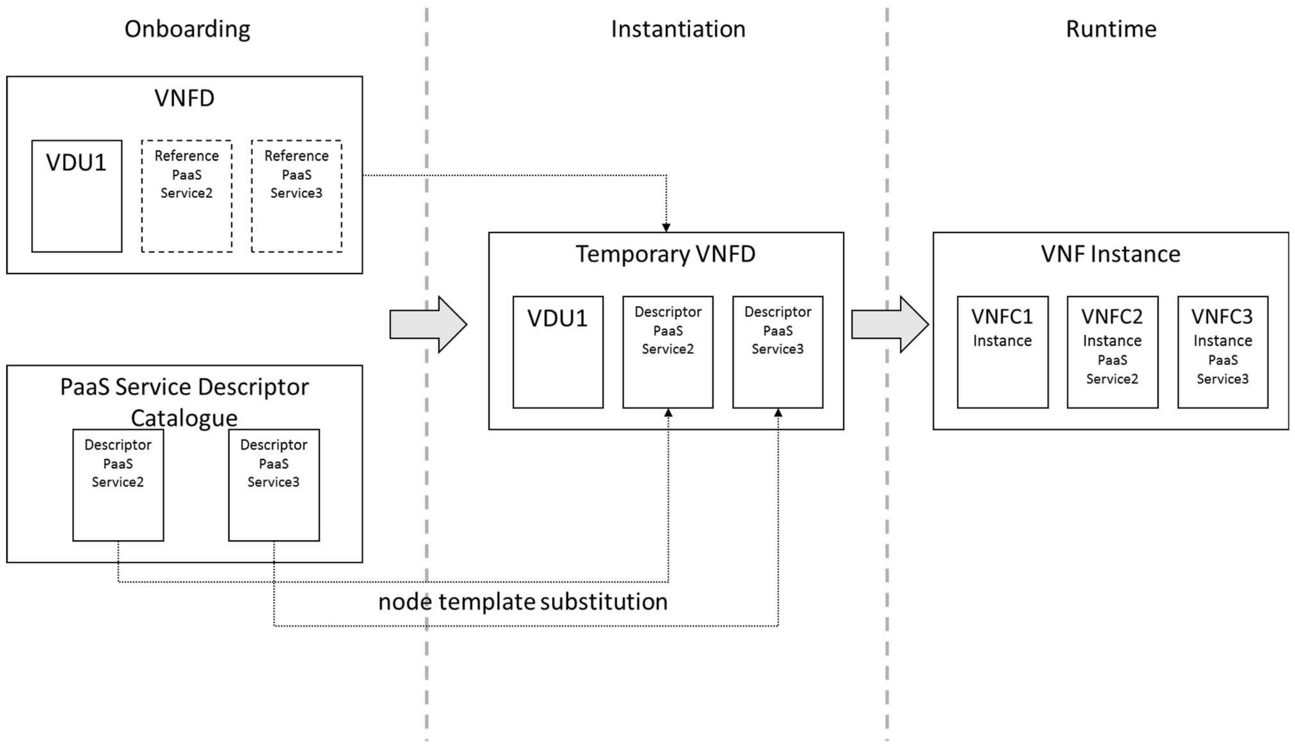


Figure 5.1.8.1-2: Stages for instantiating a VNF using PaaS service descriptors

The goal of this use case is to describe how a VNF is instantiated using PaaS service descriptors from the catalogue.

5.1.8.2 Actors and roles

Table 5.1.8.2-1 describes the use case actors and roles.

Table 5.1.8.2-1: PaaS service descriptor catalogue actors and roles

#	Role	Description
1	PaaS service descriptor catalogue	Repository containing PaaS service descriptors.
2	VNFM	The NFV functional block responsible for life-cycle management of the VNFs using the PaaS services.
3	Service Management	The function to discover PaaS service descriptors and perform node template substitution and to analyse which PaaS services need to be instantiated.
4	VNF	The VNF with a descriptor referencing PaaS service descriptors.

5.1.8.3 Pre-conditions

Table 5.1.8.3-1 describes the use case pre-conditions.

Table 5.1.8.3-1: PaaS service descriptor catalogue pre-conditions

#	Pre-condition	Additional description
1	The VNFD of the VNF to be instantiated is onboarded to the VNF catalogue.	The VNFD contains references to PaaS services.
2	The PaaS service descriptors referenced by the VNFD are included in and discoverable from the PaaS service descriptor catalogue.	The PaaS service descriptors are available.

5.1.8.4 Post-conditions

Table 5.1.8.4-1 describes the use case post-conditions.

Table 5.1.8.4-1: PaaS service descriptor catalogue post-conditions

#	Post-condition	Additional description
1	The VNF is instantiated, including VNFCs as instances of PaaS services.	N/A

5.1.8.5 Flow description

Table 5.1.8.5-1 describes the use case flow for "PaaS service descriptor catalogue".

Table 5.1.8.5-1: PaaS service descriptor catalogue flow description

#	Actor/Role	Action/Description
Begins when	VNFM	The VNFM is requested to instantiate the VNF.
Step 1	VNFM	The VNFM fetches the VNFD from the VNF catalogue.
Step 2	VNFM	The VNFM requests the Service Management to discover and replace the PaaS service references.
Step 3	Service Management	The Service Management discovers and fetches the PaaS service descriptors from the PaaS service descriptor catalogue.
Step 4	Service Management	The Service Management analyses the fetched PaaS service descriptors and decides if the described PaaS services need to be instantiated.
Step 5	Service Management	The Service Management creates a temporary VNFD that replaces the references for PaaS services in the VNFD with the PaaS service descriptors by means of node template substitution.
Step 6	Service Management	The Service Management returns the temporary VNFD to the VNFM together with information on which VNFCs need to be instantiated.
Step 7	VNFM	The VNFM instantiates the VNF.
Ends when	VNF	The VNF with the required VNFCs is instantiated.

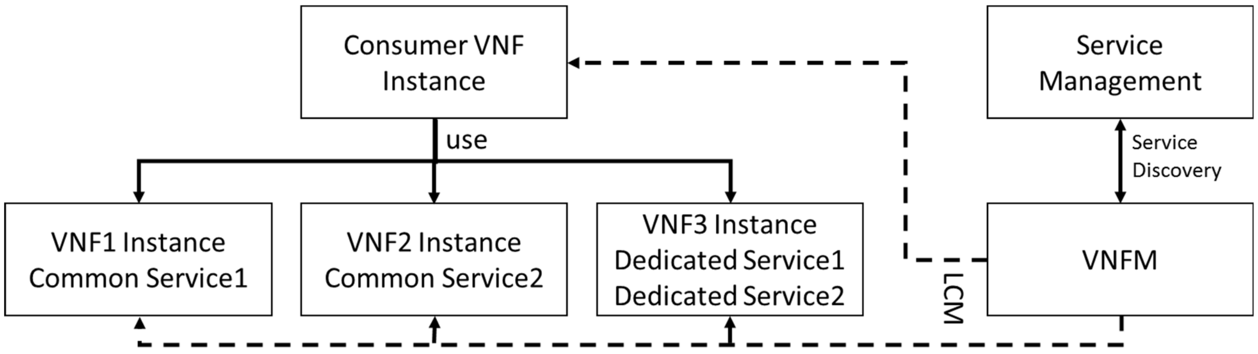
5.2 Examples of cloud-native design principles

5.2.1 Services provided by VNFs

5.2.1.1 Introduction and goal

This use case leverages on the cloud-native design principle of decomposition of network functions into independent components. It is based on the concept that a Consumer VNF consumes common and VNF Dedicated Services provided by VNFs. The Consumer VNF has requirements on common and VNF Dedicated Service capabilities exposed by the other VNFs. A VNF can provide one to many services, but all services provided by one VNF can only be of one type, either all be VNF Common Services or all be VNF Dedicated Services.

The VNFs providing VNF Common Services can be instantiated independent of the lifecycle of the Consumer VNF while the VNFs providing VNF Dedicated Services are instantiated when required by the Consumer VNF instance.


Consumer VNF Instance
use
Service Management
Service Discovery
VNF1 Instance Common Service1
VNF2 Instance Common Service2
VNF3 Instance Dedicated Service1 Dedicated Service2
VNFM
LCM

NOTE: This figure does not represent any particular architectural option.

Figure 5.2.1.1-3: Illustration of services provided by VNFs

The goal of this use case is to describe how common and VNF Dedicated Services provided by VNFs are instantiated and used by a Consumer VNF.

5.2.1.2 Actors and roles

Table 5.2.1.2-1 describes the use case actors and roles.

Table 5.2.1.2-1: Services provided by VNFs actors and roles

#	Role	Description
1	VNFM	The NFV functional block responsible for life-cycle management of the VNFs.
2	Service Management	The function to discover services provided by VNFs.
3	Consumer VNF	The VNF which uses the common and VNF Dedicated Services.
4	VNF1	The VNF providing VNF Common Service 1.
5	VNF2	The VNF providing VNF Common Service 2.
6	VNF3	The VNF providing VNF Dedicated Services 1 and 2.

5.2.1.3 Pre-conditions

Table 5.2.1.3-1 describes the use case pre-conditions.

Table 5.2.1.3-1: Services provided by VNFs pre-conditions

#	Pre-condition	Additional description
1	An instance of VNF1 providing VNF Common Service 1 is deployed and discoverable.	N/A

5.2.1.4 Post-conditions

Table 5.2.1.4-1 describes the use case post-conditions.

Table 5.2.1.4-1: Services provided by VNFs post-conditions

#	Post-condition	Additional description
1	The Consumer VNF is instantiated and can use the common and VNF Dedicated Services provided by VNF1, VNF2 and VNF3.	N/A

5.2.1.5 Flow description

Table 5.2.1.5-1 describes the use case flow for "Services provided by VNFs".

Table 5.2.1.5-1: Services provided by VNFs flow description

#	Actor/Role	Action/Description
Begins when	VNFM	The VNFM is requested to instantiate the Consumer VNF.
Step 1	VNFM	The VNFM requests the Service Management to discover the services required by the Consumer VNF.
Step 2	Service Management	The Service Management discovers the required services and returns the VNFs required by the Consumer VNF.
Step 3	VNFM	The VNFM discovers VNF1 already being instantiated.
Step 4	VNFM	The VNFM instantiates VNF2 and VNF3.
Step 5	VNFM	The VNFM instantiates the Consumer VNF.
Ends when	Consumer VNF	The Consumer VNF instance can access the common and VNF Dedicated Services provided by the VNFs 1/2/3 and use them at runtime.

5.2.2 Use case on n+k redundancy with load-balancing between distributed VNFC instances

5.2.2.1 Overview

The use case describes cloud-native VNFs consisting of a number of VNFC instances of the same type handling a certain traffic load jointly. Naturally, other VNFC instances can be part of the VNF as well. For the characterization of cloud-native VNFs refer to ETSI GS NFV-EVE 011 [i.7]. The assumption in this use case is that the VNFC instances run on different geographic locations. Therefore, in case of a failing VNFC instance, the fail-over will happen to another VNFC instance within the same or another location. Load balancing is done across several VNFC instances. The cases for any failure handling and load-balancing not known or detected by the load-balancer and failover agent are out of scope of this use case.

The assumption on the VNFC is that the state cannot be completely separated from the processing due to very low state access time requirements.

For sake of simplicity, there are only a few assumptions made on the network underlay. One assumption is that the network underlay is able to transport the traffic to different geographic locations. There are no assumptions made on what network technology is used in the network underlay nor on whether there is an interface required to the network underlay.

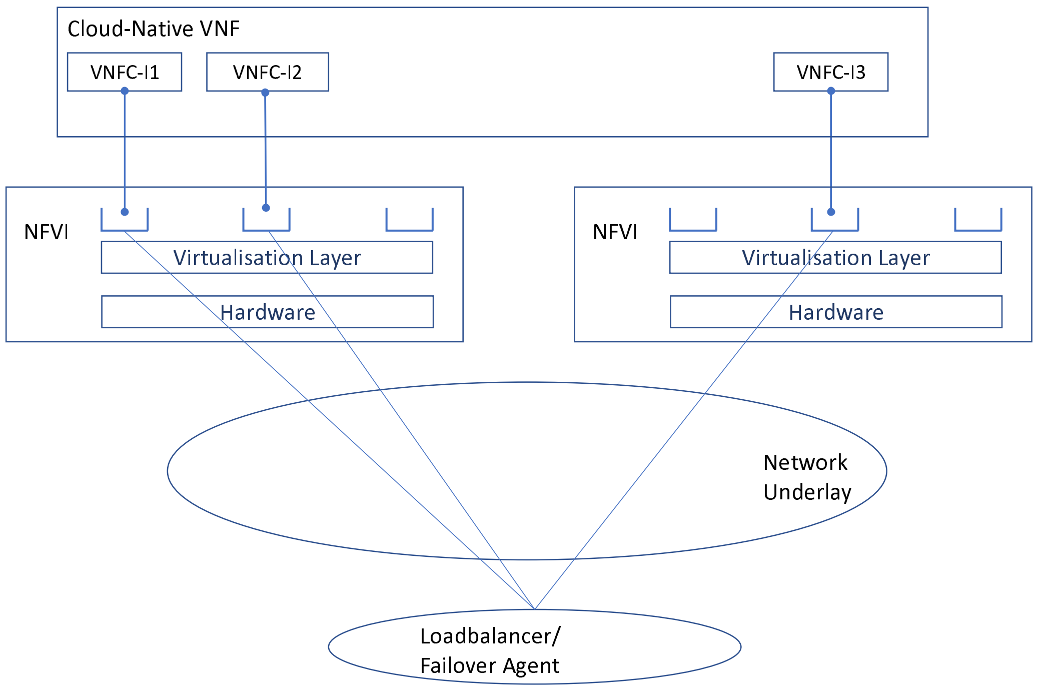


Figure 5.2.2.1-1: An example to illustrate n+k redundancy with load-balancing between distributed VNFC instances

There are several possible deployment options and specific implementation choices, which depend on the particular network function being implemented. For example, figure 5.2.2.1-1 shows the load-balancer/failover agent outside of the VNF, but it can be implemented and modelled as part of the VNF as well. Also, the load-balancer/failover agent can be a VNF or a PNF and they do not need to be a single VNF/PNF instance. Naturally, in such a setup this is a single point of failure and either the risk needs to be taken or measure have to be implemented to lower the risk of failing.

NOTE: In the present document, no further details of the single point of failure issue is described.

The roles of a load-balancer and a failover agent are in this description shown and described in a combined version, but those roles can naturally be separated and deployed differently. The impact of combining or separating those roles are out of the scope of the present document.

The reliability pattern of n+k redundancy is defined in ETSI GS NFV-REL 003 [i.6], clause 6.2.2 and is not further described here. However, the difference for this use case is that the VNFC instances are running at different locations and that there is a load-balancing and failover component placed at yet another location. No assumptions are made on where the standby and active components are located. No site separation between active and standby is needed.

Depending on the allowed switch-over time and state management approach, there might be different levels of instantiation of the stand-by VNFC instances. For example, the VNFC can be fully instantiated and running waiting for traffic, or the resources are allocated, but the VNFC is not started, or only state is available for quick start-up of the VNFC.

The benefit of n+k redundancy is, in common deployment with n>k, the lower amount of backup resources pre-allocated than a 1+1 setup. For resiliency purposes, the time to install, instantiate VNFC instances and eventually loading of state or configuration data of a VNFC instance is very relevant for how many of those instances need to be pre-allocated and running empty in hot stand-by.

Some special cases are listed in the following. When logic and state is fully separated and the instantiation of a VNFC is faster than required by the SLA, no hot stand-by VNFC instance is required (k==0) and the system can be run without pre-allocated redundancy units. For the well-known 1+1 reliability pattern used quite often in telecommunication systems, that means that n == k, such that for each VNFC instance a hot stand-by VNFC instance is pre-allocated.

In case the VNF is distributed over multiple NFVIs as shown in figure 5.2.2.1-1, hot-standby VNFC instances may be located on different NFVIs, so a failover may or may not cross NFVI borders.

5.2.2.2 Actors and roles

Table 5.2.2.2-1 describes the use case actors and roles.

Table 5.2.2.2-1: n+k redundancy with Load-balancing between distributed VNFC instances

#	Role	Description
1	Cloud-native VNF	A VNF spanning across more than one location implemented using several instances of VNFC of the same type.
2	VNFC Instance	A VNFC Instance is a unit for handling part of the traffic.
3	Load-balancer	A component in the network forwarding traffic load to several VNFC instances at different locations distributing the work load in a balanced way.
4	Failover agent	A component in the network that handles changes on traffic forwarding in case of failover cases. See note.
5	System users	All system users together generating traffic with different characteristics.
NOTE: For this use case, the failover agent is combined with the load-balancer. Alternatively, if applicable, they can run separately.		

5.2.2.3 Pre-conditions

Table 5.2.2.3-1 describes the use case pre-conditions.

Table 5.2.2.3-1: n+k redundancy with load-balancing between distributed VNFC instances

#	Pre-condition	Additional description
1	A set of VNFC instances instantiated.	N/A
2	The load balancer is instantiated and knows about the VNFC instances to be used for traffic.	The traffic is forwarded to the appropriate VNFC instances in a balanced manner.
3	The failover agent is instantiated and knows about the hot stand-by VNFC instances to forward traffic in failure cases.	N/A

5.2.2.4 Post-conditions

Table 5.2.2.4-1 describes the use case post-conditions.

Table 5.2.2.4-1: n+k redundancy with load-balancing between distributed VNFC instances

#	Post-condition	Additional description
1	The VNF keeps up and running with defined reliability and performance characteristics under load changes and under failures.	N/A

5.2.2.5 Flow description

Tables 5.2.2.5-1 and 5.2.2.5-2 describe the use case flows.

Table 5.2.2.5-1: n+k redundancy with load-balancing between distributed VNFC instances (scaling)

#	Actor/Role	Action/Description
Begins when	System users	The system users increase or decrease the traffic they generate, with this the traffic load running through the load balancer increases/decreases.
Step 1	Any actor	Any actor including the load balancer, the VNFC or the VNF detects overload or underload situations decides on and triggers instantiation or removal of VNFC instances. See note.
Step 2	Load-balancer	Changes its traffic forwarding pattern to include the new VNFC instance, remove the terminated VNFC instance from the set to forward to and rebalance accordingly. Interaction with the failover agent is required to handle the new situation.
Ends when		The system is in load-balanced state.
NOTE: The decision on which VNFC instance to terminate or on what NFVI to instantiate a new VNFC instance is out of the scope of the present document.		

Table 5.2.2.5-2: n+k redundancy with load-balancing between distributed VNFC instances (failure)

#	Actor/Role	Action/Description
Begins when	VNFC instance	A VNFC instance fails to work.
Step1	Any actor	Detects a failed active VNFC instance and triggers a failover. See note.
Step2	Failover agent	Changes its traffic forwarding pattern to include the newly active VNFC instance and removes the failed VNFC instance from the set to forward to and rebalance accordingly. Interaction with the load-balancer is required to handle the new situation.
Ends when		New instances are included, and the system is in load-balanced state.
NOTE: The decision on which hot stand-by VNFC instance to use in which NFVI to use is out of the scope of the present document.		

5.3 Examples specific to utilization of containers

5.3.1 Introduction

This clause provides use case description specific to the utilization of containers. Container as a Service (CaaS) is widely applied in the industry for improving the efficiency of enterprise DevOps activities, and it has the characteristics of both PaaS and Cloud-native studied in the present document.

The use cases in this clause are described in a logical view, in which the Container Infrastructure Service Management (CISM) does not mean any other functional block than the existing NFV-MANO functional blocks, but a logical entity with functionality to be integrated in NFV-MANO architecture. The detailed architectural impacts on NFV-MANO introducing by container related use cases refer to clause 7.2.

NOTE: In the context of the present document, the VNF or VNF component running in OS containers is called "containerized VNF" or "containerized VNFC". Either containerized VNF or containerized VNFC can be regarded as "containerized workload".

5.3.2 Container-based NFV Micro-Services within the VNF

5.3.2.1 Overview

In this use case, the VNF is implemented by NFV Micro-Services which internally use container technology.

Figure 5.3.2.1-1 shows containerized NFV Micro-Services within the VNF. Container Infrastructure Service Management and Container Infrastructure Services are part of the VNF. Container Infrastructure Service Management manages the lifecycle of Container Infrastructure Services it provides, and it also initiates container-specific virtual resource management based on the virtual machines allocated to the VNF. The functionality of both Container Infrastructure Service and Container Infrastructure Service Management is invisible to the external management entity like the VNFM. The details of internal communication between Container Infrastructure Service Management and Container Infrastructure Service are not known by NFV-MANO entity (e.g. the VNFM) as well. The VNFM manages the VNF/VNFCs and the VMs allocated to the VNF as usual, and the management of Container Infrastructure Service is left to the VNF itself. The internal structure of the VNF depicted in figure 5.3.2.1-1 is just an example and is not intended to preclude other ways to structure the contents of the VNF's VMs. Furthermore, the Container Infrastructure Service Management and the Container Infrastructure Service instances can be deployed in different VNFs.

NOTE: The case of bare metal container is not included by this use case.

As the functionality of both Container Infrastructure Service and the Container Infrastructure Service Management is invisible to NFV-MANO entities, this use case does not bring any requirements to NFV-MANO, but it is listed here for the sake of completeness, and therefore is not described in further detail except in clause 7.2.4 where a complete set of CISM to NFV MANO mapping options is discussed.

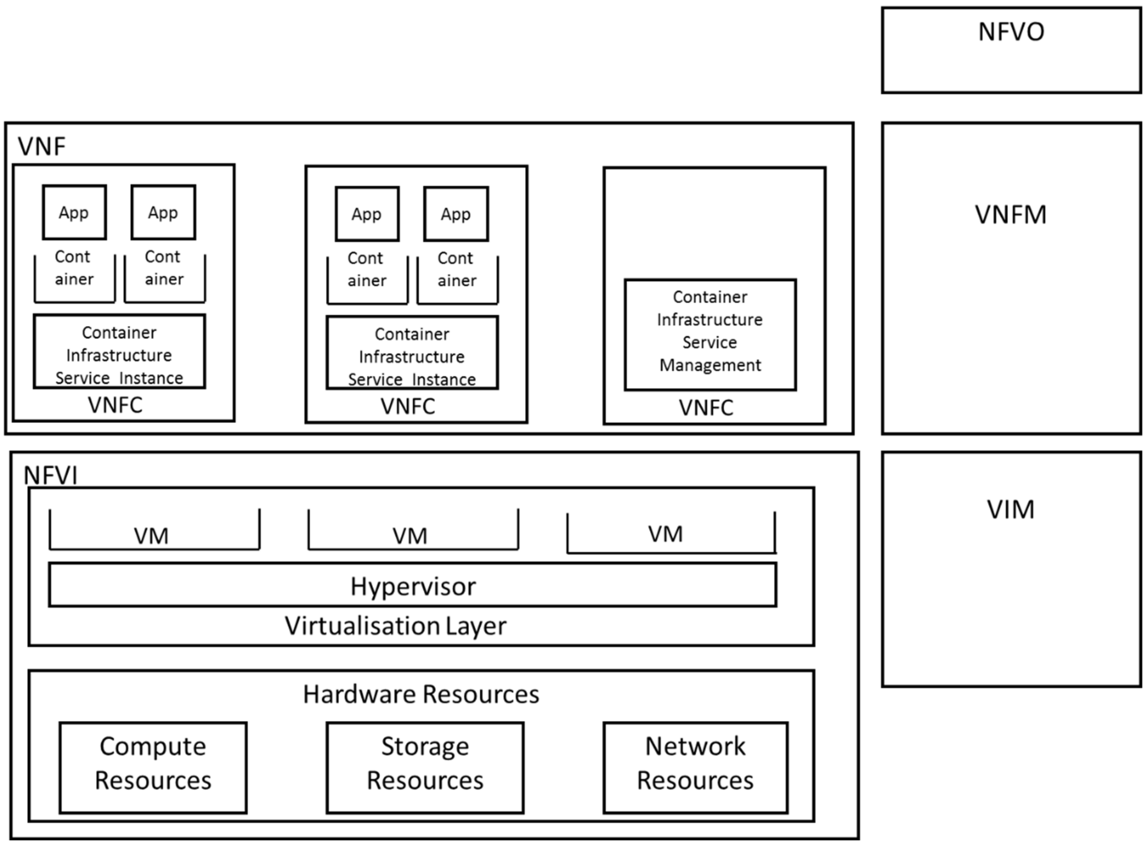


Figure 5.3.2.1-1: Containerized NFV Micro-Services within the VNF

5.3.2.2 Actors and roles

N/A.

5.3.2.3 Pre-conditions

N/A.

5.3.2.4 Post-conditions

N/A.

5.3.2.5 Flow description

N/A.

5.3.3 VNFC in container on bare metal

5.3.3.1 Overview

In this use case, a VNF is completely implemented using OS container technology. Each VNFC of the VNF is implemented as a single OS container (1:1 relation between VNFC and OS container) intended to run on bare metal, as shown in figure 5.3.3.1-1.

Additional isolation can be achieved by distributing the OS containers on different NFVI-Nodes.

NOTE 1: Other use cases will illustrate containerized VNFCs in virtual machines using nested virtualisation (see clause 5.3.4), VNFCs using a group of containers (see clause 5.3.5, e.g. Kubernetes® pod, OpenStack® Zun Capsule), or other relations between VNFC and OS container.

In the NFV environment this means that:

- NFVI provides Virtualisation containers that are OS containers:
  - The Virtualisation layer in NFVI is responsible to provide the container runtime environment for the containerized VNFC on all NFVI-Nodes.
  - In difference to an environment based on virtual machines, the NFVI is providing the OS.
  - The NFVI is also providing the virtual network so the containers can communicate within an NFVI-Node and between them.

NOTE 2: The containers are represented in figure 5.3.3.1-1 as pertaining to the NFVI domain showing resource related aspects. The VNFCs are represented here as separate boxes in the VNFs. This is done for illustration while unlike the case of VNFCs running as VMs, the container cannot be separated from the containerized VNFC it is executing.

- NFV-MANO provides the management (e.g. LCM) for the VNFCs:
  - NFV-MANO allocates the resources needed by the VNFCs.
  - A "Container Infrastructure Service Management" function will provide and provision the containers, when the VNFCs are deployed.
- NFV templates (e.g. VNFD, VDU) specify the use of container technology:
  - NFV-MANO uses the NFV templates to detect usage of containers.
  - The type of resources used includes the information about container usage.
  - NFV-MANO is provided with the necessary information for the LCM of the VNFCs, in this case as containers via the NFV templates.

VNFCs which are containers are shown as part of the VNF, while their runtime environment is provided by the NFVI.

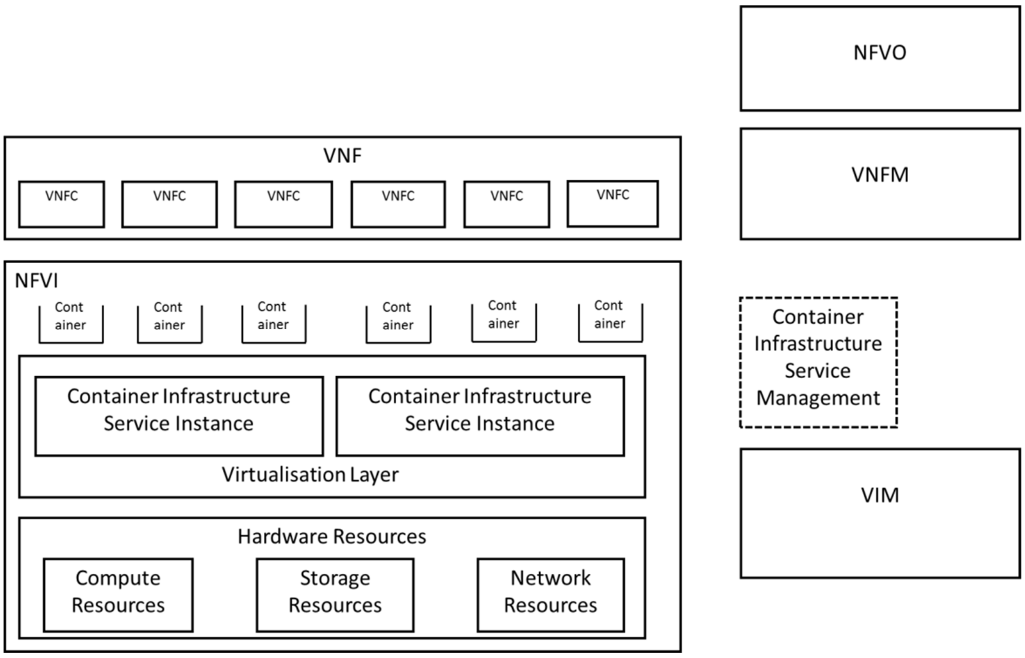


Figure 5.3.3.1-1: Illustration of VNFC in container on bare metal

5.3.3.2 Actors and roles

Table 5.3.3.2-1 describes the use case actors and roles.

Table 5.3.3.2-1: VNFC in container on bare metal actors and roles

#	Role	Description
1	NFV-MANO	NFV Management and Orchestration functional blocks including the NFVO, the VNFM and the VIM without the Container Infrastructure Service Management functionality.
2	Container Infrastructure Service Management	Management of the containers and their runtime environment. This can be a separate functional block or be integrated in an NFV-MANO component.
3	NFVI Virtualisation Layer	Contains the runtime environment for containers, which includes the OS.

5.3.3.3 Pre-conditions

Table 5.3.3.3-1 describes the use case pre-conditions.

Table 5.3.3.3-1: VNFC in container on bare metal pre-conditions

#	Pre-condition	Additional description
1	NFVI Virtualisation Layer provides the container runtime environment.	
2	NFV templates (e.g. VNFD, VDU) provide all information necessary for the deployment of container images.	This might include dependencies on OS types or specific drivers.

5.3.3.4 Post-conditions

Table 5.3.3.4-1 describes the use case post-conditions.

Table 5.3.3.4-1: VNFC in container on bare metal post-conditions

#	Post-condition	Additional description
1	The VNF can use the VNFCs deployed in the containers on bare metal.	N/A

5.3.3.5 Flow description

Table 5.3.3.5-1 describes the use case flow for "VNFC in container on bare metal".

Table 5.3.3.5-1: VNFC in container on bare metal flow description

#	Actor/Role	Action/Description
Begins when	NFV-MANO	NFV-MANO needs to instantiate a VNF which is implemented as VNFCs in containers on bare metal.
Step 1	NFV-MANO -> Container Infrastructure Service Management	NFV-MANO provides the container images and requests the Container Infrastructure Service Management to instantiate the containers using these images.
Step 2	Container Infrastructure Service Management -> Virtualisation Layer	The Container Infrastructure Service Management instantiates the containers using the container images.
Step 3	Container Infrastructure Service Management -> NFV-MANO	The Container Infrastructure Service Management responds to NFV-MANO the successful instantiation of the containers.
Step 3	NFV-MANO	NFV-MANO completes the instantiation of the VNF by using the instantiated containers as VNFCs.
Ends when	NFV-MANO	NFV-MANO has completed the instantiation of the VNF.

5.3.4 VNFC in container in virtual machine

5.3.4.1 Overview

In this use case, a VNF is completely implemented using OS container technology. Each VNFC is implemented as a single OS container intended to run in nested Virtualisation in a virtual machine (VM), as shown in figure 5.3.4.1-1. Like in the use case in clause 5.3.3, there is a 1:1 relation between VNFC and OS container.

NOTE 1: Other use cases illustrate containerized VNFCs on bare metal (see clause 5.3.3), VNFCs using a group of containers (see clause 5.3.5, e.g. Kubernetes® pod), or other relations between VNFC and container.

In the NFV environment this means that:

- NFVI provides VMs and inside the VMs there are Virtualisation containers that are OS containers:
    - In difference to the use case in clause 5.3.3, the Virtualisation layer provides a nested Virtualisation environment with containers inside VMs.
    - The Virtualisation layer in NFVI is responsible to provide the VM runtime environment as usual.
    - The Virtualisation layer in NFVI also needs to provide the OS in the VMs.
    - The Virtualisation layer in NFVI is responsible to provide the container runtime environment for the containerized VNFCs hosted by VMs running on a hypervisor layer of the NFVI-Nodes.
    - The NFVI is further providing the virtual network so the containers can communicate within a VM, within an NFVI-Node and across NFVI-Nodes.

NOTE 2: The containers are represented in figure 5.3.4.1-1 as pertaining to the NFVI domain showing resource related aspects. The VNFCs are represented here as separate boxes in the VNFs. This is done for illustration while unlike the case of VNFCs running as VMs, the container cannot be separated from the containerized VNFC it is executing.

- NFV-MANO provides the management (e.g. LCM) for the VNFCs:
    - NFV-MANO allocates the resources needed by the VNFCs.

- A "Container Infrastructure Service Management" function will provide and provision the containers, when the VNFCs are deployed.

- NFV templates (e.g. VNFD, VDU) specify the use of container technology:
  - NFV-MANO uses the NFV templates to detect usage of containers.
  - The type of resources used includes the information about container usage
  - Affinity and anti-affinity rules will specify whether containers (VNFCs) of the same VNF can be deployed on the same VM or have to run in different VMs, but can share the same NFVI-Node, while in other cases they need to be deployed on different NFVI-Nodes.
  - NFV-MANO is provided with the necessary information for the LCM of the VNFCs, in this case as containers via the NFV templates.

VNFCs which are containers are shown as part of the VNF, while their runtime environment is provided by the NFVI.

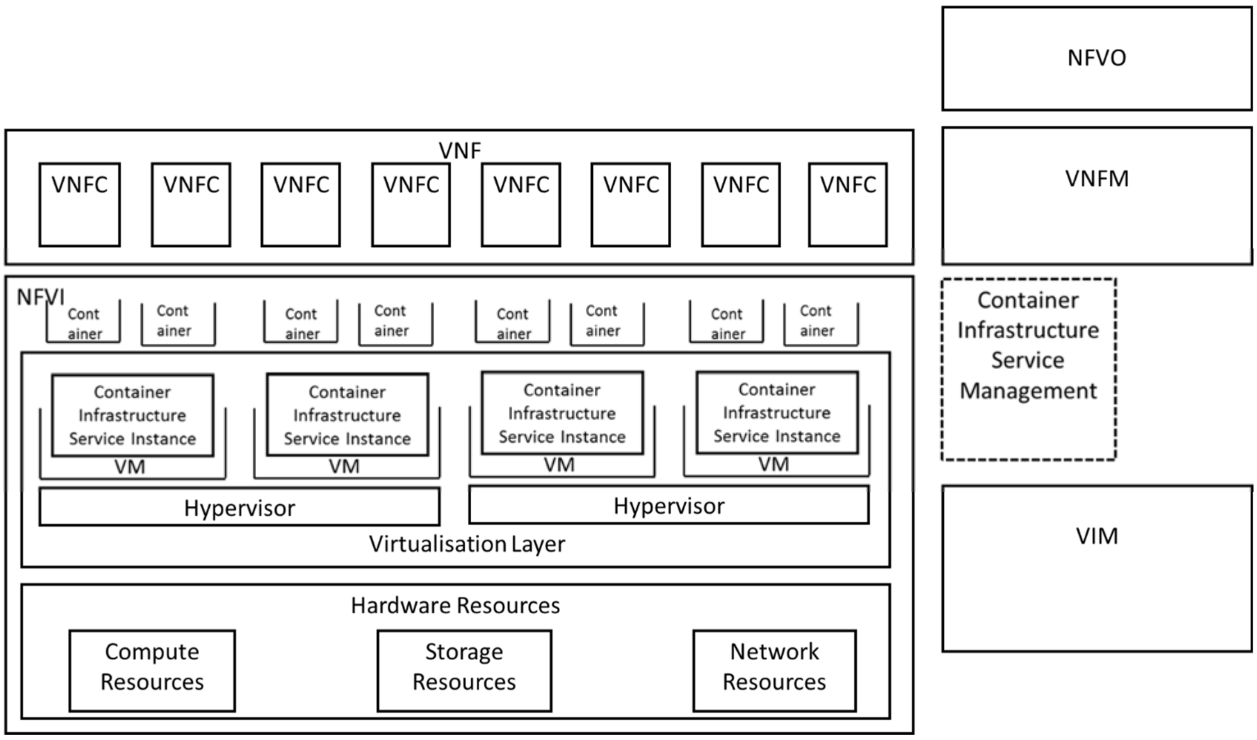


Figure 5.3.4.1-1: Illustration of VNFC in container in virtual machine

5.3.4.2 Actors and roles

Table 5.3.4.2-1 describes the use case actors and roles.

Table 5.3.4.2-1: VNFC in container in virtual machine actors and roles

#	Role	Description
1	NFV-MANO	NFV Management and Orchestration functional blocks including the NFVO, the VNFM and the VIM without the Container Infrastructure Service Management functionality.
2	Container Infrastructure Service Management	Management of the containers and their runtime environment. This can be a separate functional block or be integrated in an NFV-MANO component.
3	NFVI Virtualisation Layer	Contains the runtime environment for VMs and containers, which includes the OS.

5.3.4.3 Pre-conditions

Table 5.3.4.3-1 describes the use case pre-conditions.

Table 5.3.4.3-1: VNFC in container in virtual machine pre-conditions

#	Pre-condition	Additional description
1	NFVI Virtualisation Layer provides the container runtime environment.	
2	NFV templates (e.g. VNFD, VDU) provide all information necessary for the deployment of container images.	This might include information about dependencies on VM types, OS types or specific drivers. See note.
NOTE: Dependencies on VM types or OS types should be avoided.		

5.3.4.4 Post-conditions

Table 5.3.4.4-1 describes the use case post-conditions.

Table 5.3.4.4-1: VNFC in container in virtual machine post-conditions

#	Post-condition	Additional description
1	The VNF can use the VNFCs deployed in the containers in virtual machines.	N/A

5.3.4.5 Flow description

Table 5.3.4.5-1 describes the use case flow for "VNFC in container in virtual machine".

Table 5.3.4.5-1: VNFC in container in virtual machine flow description

#	Actor/Role	Action/Description
Begins when	NFV-MANO	NFV-MANO needs to instantiate a VNF, which is implemented as VNFCs, whereby each VNFC is deployed in a container in a virtual machine.
Step 1	NFV-MANO -> Container Infrastructure Service Management	NFV-MANO provides the container images and requests the Container Infrastructure Service Management to instantiate the containers using these images.
Step 2	Container Infrastructure Service Management -> Virtualisation Layer	The Container Infrastructure Service Management instantiates the containers using the container images. See note.
Step 3	Container Infrastructure Service Management -> NFV-MANO	The Container Infrastructure Service Management responds to NFV-MANO the successful instantiation of the containers.
Step 4	NFV-MANO	NFV-MANO completes the instantiation of the VNF by using the instantiated containers as VNFCs.
Ends when	NFV-MANO	NFV-MANO has completed the instantiation of the VNF.
NOTE: The LCM for the underlying VMs can be independent of the LCM of the VNF or be coupled with it. This is outside this use-case but is considered in the description of the implementation options.		

5.3.5 VNFC in a group of containers

5.3.5.1 Overview

In this use case, a VNF is completely implemented using OS container technology. Each VNFC of the VNF is implemented as a group of OS containers that form a Managed Container Infrastructure Object, as shown in figure 5.3.5.1-1. The OS containers of such group share the resources and run-time environment. An example for such a group of OS containers is a Kubernetes® pod or an OpenStack® Zun Capsule.

In the following description, the group of OS containers runs in nested Virtualisation like in the use case discussed in clause 5.3.4. A similar use case could use the group of OS containers on bare metal, similar to the use case in clause 5.3.3.

In the NFV environment this means that:

- NFVI provides virtual machines and inside the VMs there are Virtualisation containers that are OS containers as in the use case described in clause 5.3.4.

NOTE 1: The containers are represented in figure 5.3.5.1-1 as pertaining to the NFVI domain showing resource related aspects. The VNFCs are represented here as separate boxes in the VNFs. This is done for illustration while unlike the case of VNFCs running as VMs, the container cannot be separated from the containerized VNFC it is executing.

- NFV-MANO provides the management (e.g. LCM) for the VNFCs.
  - NFV-MANO allocates the resources needed by the VNFCs as in the use case described in clause 5.3.4.
  - A "Container Infrastructure Service Management" function will provide and provision the containers / groups of containers, when the VNFCs are deployed.
- NFV templates (e.g. VNFD, VDU) specify the use of container technology.
  - NFV-MANO uses the NFV templates to detect usage of containers and groups of containers.
  - When a VNFC is a group of containers the relation of VDU and (container) images should be 1 to n.
  - The type of resources used includes the information about container usage.
  - Affinity and anti-affinity rules will specify whether groups of containers (VNFCs) of the same VNF can be deployed on the same VM or have to run in different VMs, but can share the same NFVI-Node, while in other cases they need to be deployed on different NFVI-Nodes.
  - NFV-MANO is provided with the necessary information for the LCM of the VNFCs, in this case as groups of containers via the NFV templates.

VNFCs which are containers are shown as parts of the VNF, while their runtime environment is provided by the NFVI.

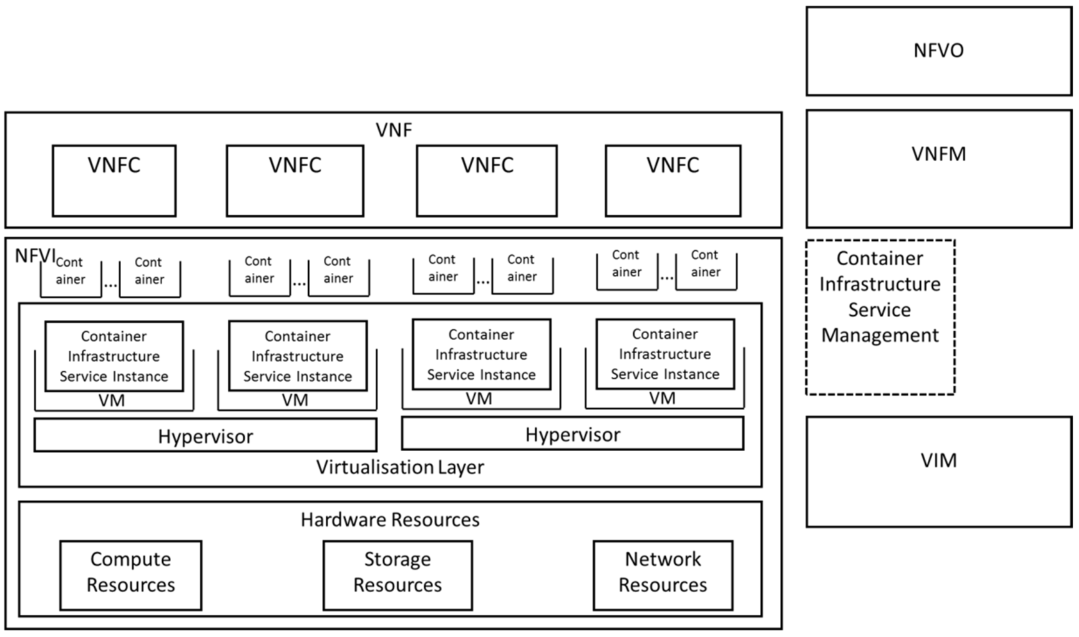


Figure 5.3.5.1-1: Illustration of VNFC in group of containers

NOTE 2: While figure 5.3.5-1-1 suggests a one-to-one mapping between a group of containers and a Container Infrastructure Service instance, more than one group of containers can run on the same Container Infrastructure Service instance.

5.3.5.2 Actors and roles

Table 5.3.5.2-1 describes the use case actors and roles.

Table 5.3.5.2-1: VNFC in group of containers actors and roles

#	Role	Description
1	NFV-MANO	NFV Management and Orchestration functional blocks including the NFVO, the VNFM and the VIM without the Container Infrastructure Service Management functionality.
2	Container Infrastructure Service Management	Management of the containers and their runtime environment. This can be a separate functional block or be integrated in an NFV-MANO component.
3	NFVI Virtualisation Layer	Contains the runtime environment for VMs and containers, which includes the OS.

5.3.5.3 Pre-conditions

Table 5.3.5.3-1 describes the use case pre-conditions.

Table 5.3.5.3-1: VNFC in group of containers pre-conditions

#	Pre-condition	Additional description
1	NFVI Virtualisation Layer provides the container runtime environment.	
2	NFV templates (e.g. VNFD, VDU) provide all information necessary for the deployment of container images.	This might include information about dependencies on VM types, OS types or specific drivers. See note.
NOTE: Dependencies on VM types or OS types should be avoided.		

5.3.5.4 Post-conditions

Table 5.3.5.4-1 describes the use case post-conditions.

Table 5.3.5.4-1: VNFC in group of containers post-conditions

#	Post-condition	Additional description
1	The VNF can use the VNFCs, and each VNFC is deployed in group of containers.	N/A

5.3.5.5 Flow description

Table 5.3.5.5-1 describes the use case flow for "VNFC in group of containers".

Table 5.3.5.5-1: VNFC in group of containers flow description

#	Actor/Role	Action/Description
Begins when	NFV-MANO	NFV-MANO needs to instantiate a VNF which is implemented as VNFCs, whereby each VNFC is deployed in a group of containers.
Step 1	NFV-MANO -> Container Infrastructure Service Management	NFV-MANO provides the container images and requests the Container Infrastructure Service Management to instantiate the containers using these images.
Step 2	Container Infrastructure Service Management -> Virtualisation Layer	The Container Infrastructure Service Management instantiates the containers using the container images. See note.
Step 3	Container Infrastructure Service Management -> NFV-MANO	The Container Infrastructure Service Management responds to NFV-MANO the successful instantiation of the containers.
Step 4	NFV-MANO	NFV-MANO completes the instantiation of the VNF by using the VNFCs deployed in the instantiated containers.
Ends when	NFV-MANO	NFV-MANO has completed the instantiation of the VNF.
NOTE: The LCM for the underlying VMs can be independent of the LCM of the VNF or be coupled with it. This is outside this use-case but is considered in the description of the implementation options.		

5.3.6 NFVI provides containers on bare metal and VM

5.3.6.1 Overview

In this use case, NFVI provides both OS containers on bare metal and on VMs at the same time. The resources for a container are the NFVI resources allocated by the VIM when the container instances are created.

The VNFs are deployed on Container Infrastructure Service Instances by instantiating the container images of the VNF components. It is transparent to the VNF how the OS Virtualisation and its runtime environment is provided.

The Container Infrastructure Service can be deployed on the hardware resource (bare metal) or deployed on VMs, as shown in figure 5.3.6.1-1.

NOTE 1: The containers are represented in this illustration of figure 5.3.6.1-1 as pertaining to the NFVI domain showing resource related aspects. The VNFCs are represented here as separate boxes in the VNFs. This is done for illustration while unlike the case of VNFCs running as VMs, the container cannot be separated from the containerized VNFC it is executing.

NOTE 2: There is no difference between Container Infrastructure Service Instance over bare metal and over VM.

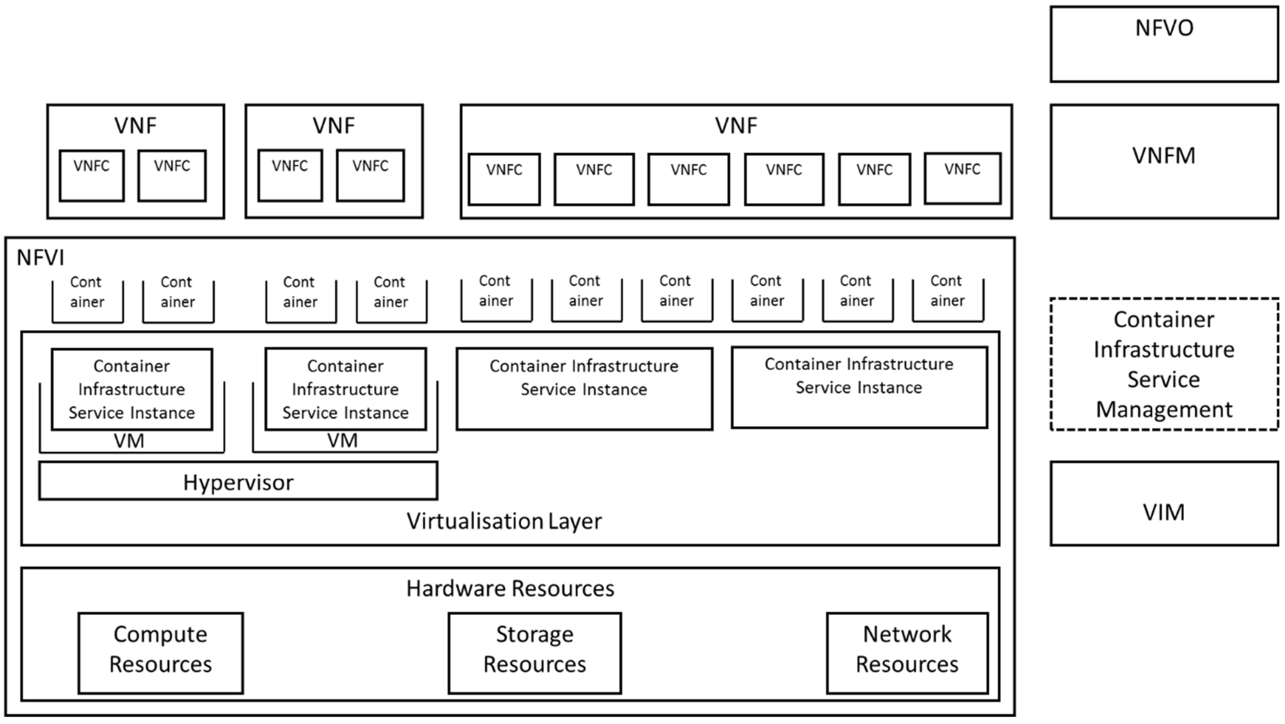

NFVO
VNF
VNFC
VNFC
VNF
VNFC
VNFC
VNF
VNFC
VNFC
VNFC
VNFC
VNFC
VNFC
VNFM
NFVI
Cont ainer
Container Infrastructure Service Instance
VM
Container Infrastructure Service Instance
VM
Container Infrastructure Service Instance
Container Infrastructure Service Instance
Hypervisor
Virtualisation Layer
Container Infrastructure Service Management
VIM
Hardware Resources
Compute Resources
Storage Resources
Network Resources

Figure 5.3.6.1-1: NFVI provides containers on bare metal and VM

The NFVI can provide one or more Container Infrastructure Service Instances using both bare metal and VMs. Some Container Infrastructure Service Instances are running on VMs and other running on bare metal. A variety of options are possible for container deployment. Every VNF can be hosted by one or more Container Infrastructure Service Instances.

5.3.6.2 Actors and roles

Table 5.3.6.2-1 describes the use case actors and roles.

Table 5.3.6.2-1: NFVI provides containers on bare metal and VM actors and roles

#	Role	Description
1	VNF	The VNF is deployed using resources for container.
2	NFVO	The NFV Orchestrator functional block.
3	VNFM	VNFM is responsible for the VNF LCM.
4	Container Infrastructure Service Management	The Container Infrastructure Service Management can send the commands to Container Infrastructure Service Instance for creating containers, releasing containers, etc.; Create, Read, Update, and Delete (CRUD) commands are used for the container resource management.
5	Container Infrastructure Service Instance	An instance providing runtime execution environment for container.

5.3.6.3 Pre-conditions

Table 5.3.6.3-1 describes the use case pre-conditions.

Table 5.3.6.3-1: NFVI provides containers on bare metal and VM pre-conditions

#	Pre-condition	Additional description
1	There are Container Infrastructure Service Instances available that can host the containers.	
2	NFV templates (e.g. VNFD, VDU) provide all information necessary for the deployment of containers.	
3	The container images are uploaded to and accessible via a container image registry.	The CISM has access to the container images available via the container image registry.

5.3.6.4 Post-conditions

Table 5.3.6.4-1 describes the use case post-conditions.

Table 5.3.6.4-1: NFVI provides containers on bare metal and VM post-conditions

#	Post-condition	Additional description
1	The VNFs are instantiated.	

5.3.6.5 Flow description

Table 5.3.6.5-1 describes the use case flow for "NFVI provides containers on bare metal and VM".

Table 5.3.6.5-1: Flow description that NFVI provides containers on bare metal and VM

#	Actor/Role	Action/Description
Begins when	NFVO	NFVO is triggered to instantiate containerized VNFs.
Step 1	NFVO -> VNFM	NFVO triggers VNFM to instantiate the VNFs.
Step 2	VNFM -> Container Infrastructure Service Management	VNFM requests the Container Infrastructure Service Management to instantiate the container images.
Step 3	Container Infrastructure Service Management <-> Container Infrastructure Service Instance	The Container Infrastructure Service Management instantiates the containers using the container images.
Step 4	Container Infrastructure Service Management -> VNFM	The CISM informs the VNFM that the container images are instantiated.
Step 5	VNFM	The VNFs are instantiated.
Ends when	VNFM	The VNFM has completed the instantiation of VNFs.

5.3.7 Containerized VNF Package Onboarding

5.3.7.1 Overview

This clause provides descriptions of use cases for the onboarding of VNF Packages for containerized workloads. It covers two different use cases, where the container images are either included or not included in the VNF Package to be onboarded.

It is assumed that the containerized VNF Packages contain declarative descriptor templates for the containerized workloads, included in managed container infrastructure object packages. For more information on managed container infrastructure objects and their packages, refer to clause 6.2.

It is further assumed that the NFVO has access to the managed container infrastructure object package registry, and that the NFVO has access to the container image registry.

5.3.7.2 Actors and roles

Table 5.3.7.2-1 describes the use case actors and roles.

Table 5.3.7.2-1: Containerized VNF Package Onboarding actors and roles

#	Role	Description
1	NFVO	The NFV Orchestrator functional block, responsible for onboarding VNF Packages.
2	Container image registry	The registry that maintains and provides access to container images.
3	Managed container infrastructure object package registry	The registry that maintains and provides access to managed container infrastructure object packages.

5.3.7.3 Onboarding of containerized VNF Package including container images

5.3.7.3.1 Pre-conditions

Table 5.3.7.3.1-1 describes the use case pre-conditions.

Table 5.3.7.3.1-1: Onboarding of containerized VNF Package including container images pre-conditions

#	Pre-condition	Additional description
1	The container-based VNF Package includes container images.	

5.3.7.3.2 Post-conditions

Table 5.3.7.3.2-1 describes the use case post-conditions.

Table 5.3.7.3.2-1: Onboarding of containerized VNF Package including container images post-conditions

#	Post-condition	Additional description
1	The container images are uploaded to and accessible from the container image registry.	
2	The managed container infrastructure object packages are uploaded to and accessible from the managed container infrastructure object package registry.	

5.3.7.3.3 Flow description

Table 5.3.7.3.3-1 describes the use case flow for "Onboarding of containerized VNF Package including container images".

Table 5.3.7.3.3-1: Flow description for onboarding of containerized VNF Package including container images

#	Actor/Role	Action/Description
Begins when	NFVO	NFVO receives a request to onboard a containerized VNF Package.
Step 1	NFVO	NFVO receives all artefacts included in the containerized VNF Package.
Step 2	NFVO -> container image registry	NFVO uploads all container images to the container image registry, using references for the images specified in the managed container infrastructure object packages.
Step 3	NFVO -> managed container infrastructure object package registry	NFVO uploads all managed container infrastructure object packages to the managed container infrastructure object package registry.
Ends when	NFVO	The NFVO has completed the onboarding of the containerized VNF Package.

5.3.7.4 Onboarding of containerized VNF Package not including container images

5.3.7.4.1 Pre-conditions

Table 5.3.7.4.1-1 describes the use case pre-conditions.

Table 5.3.7.4.1-1: Onboarding of container-based VNF Package not including container images pre-conditions

#	Pre-condition	Additional description
1	The configuration files included in the managed container infrastructure object package contain references for the access to the container images in a container image registry.	

5.3.7.4.2 Post-conditions

Table 5.3.7.4.2-1 describes the use case post-conditions.

Table 5.3.7.4.2-1: Onboarding of containerized VNF Package not including container images post-conditions

#	Post-condition	Additional description
1	The container images are accessible from the container image registry.	
2	The managed container infrastructure object packages are uploaded to and accessible from the managed container infrastructure object package registry.	

5.3.7.4.3 Flow description

Table 5.3.7.4.3-1 describes the use case flow for "Onboarding of containerized VNF Package not including container images".

Table 5.3.7.4.3-1: Flow description for onboarding of containerized VNF Package not including container images

#	Actor/Role	Action/Description
Begins when	NFVO	The NFVO receives a request to onboard a containerized VNF Package.
Step 1	NFVO	The NFVO receives all artefacts included in the container-based VNF Package.
Step 2	NFVO -> managed container infrastructure object package registry	The NFVO uploads all managed container infrastructure object packages to the managed container infrastructure object package registry.
Ends when	NFVO	The NFVO has completed the onboarding of the containerized VNF Package.

5.3.8 Scaling out Container Infrastructure Service Instances

5.3.8.1 Overview

This clause provides descriptions of a use case for scaling out new Container Infrastructure Service Instances when there are containers running on virtual machines where the containers have failed to be scheduled due to insufficient container infrastructure service instances.

5.3.8.2 Actors and roles

Table 5.3.8.2-1 describes the use case actors and roles.

Table 5.3.8.2-1: Scaling out Container Infrastructure Service Instances

#	Role	Description
1	Container Infrastructure Service Management	Management of the containers and their runtime environment.
2	VM-specific VR management	VM-specific VR management manages the NFVI including the virtualised resources.

5.3.8.3 Pre-conditions

Table 5.3.8.3-1 describes the use case pre-conditions.

Table 5.3.8.3-1: Scaling out Container Infrastructure Service Instances

#	Pre-condition	Additional description
1	There are containers running on virtual machines where the containers have failed to be scheduled due to insufficient container infrastructure service instances.	

5.3.8.4 Post-conditions

Table 5.3.8.4-1 describes the use case post-conditions.

Table 5.3.8.4-1: Scaling out Container Infrastructure Service Instances

#	Post-condition	Additional description
1	All containers are scheduled to the Container Infrastructure Service Instances.	The CISM has additional resources of Container Infrastructure Service Instances and is able to schedule all containers requested.

5.3.8.5 Flow description

Table 5.3.8.5-1 describes the use case flow for "Scaling out Container Infrastructure Service Instances".

Table 5.3.8.5-1: Scaling out Container Infrastructure Service Instances

#	Actor/Role	Action/Description
Begins when	Container Infrastructure Service Management	There are containers running on virtual machines where the containers have failed to be scheduled due to insufficient container infrastructure service instances.
Step 1	Container Infrastructure Service Management -> VM-specific VR management	Container Infrastructure Service Management calls VM-specific VR management to scale out Container Infrastructure Service Instances. See note.
Step 2	VM-specific VR management -> Container Infrastructure Service Management	VM-specific VR management provides more VM resources to Container Infrastructure Service Management and returns the result. See note
Ends when	Container Infrastructure Service Management	The containers that failed to be scheduled are allocated to the new Container Infrastructure Service Instance.
NOTE: If some containers are still failed to be scheduled due to insufficient container infrastructure service instances, the process of Step 1 and 2 is repeated, and more Container Infrastructure Service Instances are created.		

5.3.9 Scaling in Container Infrastructure Service Instances

5.3.9.1 Overview

This clause provides descriptions of a use case for Scaling in the Container Infrastructure Service Instances. In case that the Container Infrastructure Service Instances have been underutilized for an extended period of time, the Container Infrastructure Service Instances can be scaled in and their running containers can be migrated to other Container Infrastructure Service Instances.

5.3.9.2 Actors and roles

Table 5.3.9.2-1: Scaling in Container Infrastructure Service Instances

#	Role	Description
1	Container Infrastructure Service Management	Management of the containers and their runtime environment.
2	VM-specific VR management	VM-specific VR management manages the NFVI including the virtualised resources.

5.3.9.3 Pre-conditions

Table 5.3.9.3-1 describes the use case pre-conditions.

Table 5.3.9.3-1: Scaling in Container Infrastructure Service Instances

#	Pre-condition	Additional description
1	The Container Infrastructure Service Instances have been underutilized for an extended period of time.	The containers on these Container Infrastructure Service Instances can be migrated to other Container Infrastructure Service Instances.

5.3.9.4 Post-conditions

Table 5.3.9.4-1 describes the use case post-conditions.

Table 5.3.9.4-1: Scaling in Container Infrastructure Service Instances

#	Post-condition	Additional description
1	The underutilized Container Infrastructure Service Instances are successfully scaled in.	

5.3.9.5 Flow description

Table 5.3.9.5-1 describes the use case flow for "Scaling in Container Infrastructure Service Instances".

Table 5.3.9.5-1: Scaling in Container Infrastructure Service Instances

#	Actor/Role	Action/Description
Begins when	Container Infrastructure Service Management	Container Infrastructure Service Management decides to scale in managed Container Infrastructure Service instances when Container Infrastructure Service instances are underutilized for an extended period of time, and their running containers can be migrated. See note.
Step 1	Container Infrastructure Service Management	Container Infrastructure Service Management migrates the running Containers in the underutilized Container Infrastructure Service Instances to other Container Infrastructure Service Instances.
Step 2	Container Infrastructure Service Management -> VM-specific VR management	Container Infrastructure Service Management calls VM-specific VR management to scale in the underutilized Container Infrastructure Service Instances after their containers have been migrated to other Container Infrastructure Service Instances successfully.
Step 3	VM-specific VR management -> Container Infrastructure Service Management	VM-specific VR management releases all the VR resources of these underutilized Container Infrastructure Service Instances and returns the result to the Container Infrastructure Service Management.
Ends when	Container Infrastructure Service Management	Container Infrastructure Service Management removes these Container Infrastructure Service Instances from the list of managed Container Infrastructure Service Instances.
NOTE: Metrics used to detect that an instance is underutilized are outside the scope of the present document.		

5.3.10 Instantiating containerized workload with images accessible via local registry

5.3.10.1 Overview

In this use case, a local container image registry is connected with the CISM, facilitating the CISM to provide container image functionality. Here "local" means that the container image registry is located together with the CISM, e.g. in the same NFVI-PoP.

The CISM communicates with the registry to store container images into the registry, utilising the images for managing the lifecycle of containerized workload and deleting container images.

5.3.10.2 Actors and roles

Table 5.3.10.2-1 describes the use case actors and roles.

Table 5.3.10.2-1: Instantiating containerized workload with images accessible via local registry actors and roles

#	Role	Description
1	VNF	The VNF is deployed using resources for container.
2	NFVO	The NFV Orchestrator functional block.
3	VNFM	VNFM is responsible for the VNF LCM.
4	Container Infrastructure Service Management	The Container Infrastructure Service Management can send commands to the container image registry for storing, distributing and deleting container images, and querying information about container images.
5	container image registry	The local registry that provides access to the container images.

5.3.10.3 Pre-conditions

Table 5.3.10.3-1 describes the use case pre-conditions.

Table 5.3.10.3-1: Instantiating containerized workload with images accessible via local registry pre-conditions

#	Pre-condition	Additional description
1	VNF package is onboarded to the NFVO.	The VNF package contains information about the container images used by the VNF.
2	The local image registry is connected with the CISM.	The CISM needs to be configured to have the access right for the registry. The CISM can connect to multiple image registries.
3	The container images used by the VNF are uploaded and accessible via the local container image registry.	

5.3.10.4 Post-conditions

Table 5.3.10.4-1 describes the use case post-conditions.

Table 5.3.10.4-1: Instantiating containerized workload with images accessible via local registry post-conditions

#	Post-condition	Additional description
1	The VNFs are instantiated.	

5.3.10.5 Flow description

Table 5.3.10.5-1 describes the use case flow for "Instantiating containerized workload with images accessible via local registry".

Table 5.3.10.5-1: Flow description of instantiating containerized workload with images accessible via local registry

#	Actor/Role	Action/Description
Begins when	NFVO	NFVO is triggered to instantiate containerized workload.
Step 1	NFVO -> VNFM	NFVO triggers VNFM to instantiate the VNFs.
Step 2	VNFM -> CISM	VNFM requests the CISM to instantiate the containers and provide necessary information.
Step 3	CISM -> image registry	The CISM utilizes the information of the container images to instantiate the containerized workload.
Step 4	VNFM	The VNFs are instantiated.
Ends when	VNFM	The VNFM has completed the instantiation of VNFs.

5.3.11 Granting of service resources for containerized workloads

5.3.11.1 Overview

This clause provides descriptions of a use case that the NFVO has the capability to grant the service resources for the containerized workloads before the VNFM initiates service resource management for containerized workloads towards the CISM.

5.3.11.2 Actors and roles

Table 5.3.11.2-1 describes the use case actors and roles.

Table 5.3.11.2-1: Actors and roles for granting of service resources for containerized workloads

#	Role	Description
1	VNFM	Request the NFVO to grant the service resources for the containerized workloads.
2	NFVO	Grant the required service resources for the containerized workloads.

5.3.11.3 Pre-conditions

Table 5.3.11.3-1 describes the use case pre-conditions.

Table 5.3.11.3-1: Pre-conditions for granting of service resources for containerized workloads

#	Pre-condition	Additional description
1	The VNFM receives a request to perform VNF LCM operation.	

5.3.11.4 Post-conditions

Table 5.3.11.4-1 describes the use case post-conditions.

Table 5.3.11.4-1: Post-conditions for granting of service resources for containerized workloads

#	Post-condition	Additional description
1	The VNFM is available for initiating service management for containerized workloads.	

5.3.11.5 Flow description

Table 5.3.11.5-1 describes the use case flow for granting of service resources for containerized workloads.

Table 5.3.11.5-1: Flow for granting of service resources for containerized workloads

#	Actor/Role	Action/Description
Begins when	VNFM	The VNFM receives a request to perform the VNF (containerized workloads) LCM operation.
Step 1	VNFM -> NFVO	The VNFM requests the NFVO to grant the required service resources needed by the containerized workloads.
Step 2	NFVO -> VNFM	The NFVO grants the required service resources needed by the containerized workloads and returns the response to VNFM.
Ends when	VNFM	The granting of service resources for containerized workloads is completed.

5.3.12 Management of service resources for containerized workloads in indirect mode

5.3.12.1 Overview

This clause provides descriptions of a use case that the NFVO is used to manage the service resources for the containerized workloads in indirect mode when the CISM is triggered to perform service resource management for the containerized workloads by the VNFM.

5.3.12.2 Actors and roles

Table 5.3.12.2-1 describes the use case actors and roles.

Table 5.3.12.2-1: Actors and roles for management of service resources for containerized workloads in indirect mode

#	Role	Description
1	VNFM	Trigger service resources management for the containerized workloads.
2	Container Infrastructure Service Management	Perform service resource management for the containerized workloads.
3	NFVO	Request the management operation of the service resources for the containerized workloads in indirect mode.

5.3.12.3 Pre-conditions

Table 5.3.12.3-1 describes the use case pre-conditions.

Table 5.3.12.3-1: Pre-conditions for management of service resources for containerized workloads in indirect mode

#	Pre-condition	Additional description
1	The VNFM triggers the Container Infrastructure Service Management to perform service resource management for the containerized workloads.	

5.3.12.4 Post-conditions

Table 5.3.12.4-1 describes the use case post-conditions.

Table 5.3.12.4-1: Post-conditions for management of service resources for containerized workloads in indirect mode

#	Post-condition	Additional description
1	none	

5.3.12.5 Flow description

Table 5.3.12.5-1 describes the use case flow for the management of service resources of containerized workloads in indirect mode.

Table 5.3.12.5-1: Flow for management of service resources for containerized workloads in indirect mode

#	Actor/Role	Action/Description
Begins when	VNFM	The VNFM initiates service resource management for the containerized workloads.
Step 1	VNFM -> NFVO	In indirect mode, the VNFM invokes the service resource management request towards the NFVO to request the management of the service resources required by the containerized workloads.
Step2	NFVO -> Container Infrastructure Service Management	NFVO transfers the service resource management request towards the Container Infrastructure Service Management to request the management of the service resources required by the containerized workloads.
Step3	Container Infrastructure Service Management -> NFVO	The Container Infrastructure Service Management responds the successful allocation or release of the service resources for the containerized workloads.
Step 4	NFVO -> VNFM	The NFVO responds the successful allocation or release of the service resources for containerized workloads towards the VNFM.
Ends when	VNFM	Service resource management for the containerized workloads is completed.

6 Description of functionalities

6.1 Service Requirements and Capabilities

With the concepts of VNF Common Services and VNF Dedicated Services, which can be used by a Consumer VNF, mechanisms for service registration and discovery are needed.

The available VNF Common Services and VNF Dedicated Services need to be registered with the capabilities they provide to indicate which consumer requirements they are able to fulfil. The registered service capabilities include service instance access information. A Consumer VNF, which wants to use the service, needs to obtain a service access interface type and service access point to connect to the providing service instance (for example, it can be configured at installation time or looked-up at run-time or some other way).

A Consumer VNF specifies requirements on the service capabilities it wants to use. At a life-cycle event of a Consumer VNF instance, the required service capabilities need to be discovered. This service discovery resolves whether matching services are available and whether they fulfil the requirements posed on them from a functional and non-functional perspective.

6.2 Container Infrastructure Service Management

6.2.1 Introduction

This clause describes the functionality of Container Infrastructure Service Management, which is derived from use cases in clause 5. The consumable functionality of Container Infrastructure Service Management is exposed by one or several APIs. The study in clause 6.2 focuses on analysing the API(s) provided by de-facto container management standards (such as Kubernetes®) and deriving the functionality of the Container Infrastructure Service Management for supporting NFV-MANO management tasks.

6.2.2 Functional role

6.2.2.1 Management of containerized workloads

The Container Infrastructure Service Management (CISM) is responsible for the management of containerized workloads executed by the Container Infrastructure Service (CIS). The CISM is responsible for the deployment, monitoring, and lifecycle management of containerized workloads running in OS containers. It exposes corresponding APIs to its consumers and translates incoming requests into operations which are enforced towards the CIS. The CISM is further responsible to parse and interpret descriptors and configuration files of the containerized workloads.

6.2.2.2 Management of the resources for containers

The CISM is responsible for the management of the infrastructure resources for containers which contain computing, storage, and network resources.

The Container Runtime Management (CRM) is a functionality of CISM, and it manages container computing resources provided by the CIS via an interface, such as the Container Runtime Interface (CRI) of Kubernetes®, or the related interface used by OpenStack® Zun. The CRM enables the CISM to manage different container runtimes, such as Open Container Initiative™ (OCI™) conformant runtimes (CRI-O™), without the need to recompile. The CRM communicates with CIS Instances to manage the lifecycle of containers, as well as calls to interact with containers (such as exec/attach/detach).

The Container Storage Management (CSM) is a functionality of CISM, and it manages container storage resources provided by the CIS via an interface, such as the Container Storage Interface (CSI) of Kubernetes®, or the interface used by OpenStack® Zun to bind-mounting OpenStack® Cinder volumes to a container. The CSI provides a common management method for CISM to expose arbitrary storage systems to their containerized workloads.

The Container Network Management (CNM) is a functionality of CISM, and it manages container network resources provided by the CIS via an interface, such as the Container Network Interface (CNI™) of Kubernetes®, or the interface used by OpenStack® Zun, which integrates with OpenStack® Neutron by Kuryr-libnetwork to manage the network resources. The CNI™ provides a common management method between the network and container runtime environment.

NOTE: The interfaces, such as the above-mentioned examples of CRI, CSI and CNI™, are interfaces towards the CIS and are not exposed in the CISM northbound interface.

The infrastructure resources for containers are provided by the underlying system and are managed by the corresponding management entity of the Virtualisation technology.

The allocation and isolation of infrastructure resources for a certain tenant is dependent on the underlying Virtualisation technology and is provided by its respective management system(s).

In addition, the CISM is also responsible for the management of the service resources for containerized workloads, which can be exposed to the NFVO in the CISM northbound interface. It can provide constraints that limit aggregate service resource consumption per container workload. The limitations may include the quantity of MCIO(s), as well as the total amount of infrastructure compute resources and infrastructure storage resources that can be consumed by one or more MCIO(s).

6.2.3 Managed Container Infrastructure Objects

The CISM maintains the containerized workloads by managed container infrastructure objects as opposed to the VIM for hypervisor-based Virtualisation which maintains virtualised resources. The desired states of the managed container infrastructure objects are specified in their respective descriptors and configuration files. The descriptors of the managed container infrastructure objects may include references to container images which are exposed by a container image registry.

Examples of CISM managed container infrastructure objects in Kubernetes® are Pods, Services, Deployments, ReplicaSets, StatefulSets, see clause A.3 on Cloud-Native Computing Foundation for more information.

Higher level aggregates of CISM managed container infrastructure objects, representing aggregated containerized workload structures, are managed as packages. These managed container infrastructure object packages are used to instantiate and maintain containerized workloads and provide a higher abstraction level than individual CISM managed container infrastructure objects, declaring the relationships among them.

As an example, the CNCF® [i.9] specifies the packaging framework Helm™, extending a Kubernetes® based CISM. The packaging format is specified as Helm™ Charts, while their instantiated runtime objects are referenced as Helm™ Release.

6.2.4 Model Mapping

The managed container infrastructure objects of the CISM are not directly mappable to virtualised resources utilized by the CIS due to the dynamic nature of their states. Most of these objects do not have an equivalent in the current NFV information model as they do not support a direct resource mapping and they may relate to implementation specific solutions.

In the logical view, a VNFC can be mapped to the smallest manageable container infrastructure object supported by the CISM, such as a Pod, the smallest deployable object in the Kubernetes® object model, see figure 6.2.4-1 for illustration.

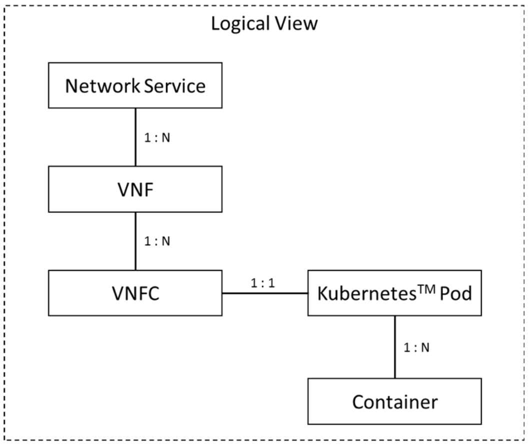


Figure 6.2.4-1: Example logical model mapping for Kubernetes®

A VNF Package is the smallest object able to be onboarded and a VNF is the object to be instantiated from a VNF Package in the current NFV information model. In the deployment view, a VNFD can be mapped to one or more managed container infrastructure object packages, such as Helm™ Charts, see figure 6.2.4-2. A simplification of the mapping can be achieved by the usage of hierarchical Helm™ Charts. In the runtime view, a VNF instance can be mapped to an instantiation of one or more managed container infrastructure object packages, such as a Helm™ Release, see figure 6.2.4-3.

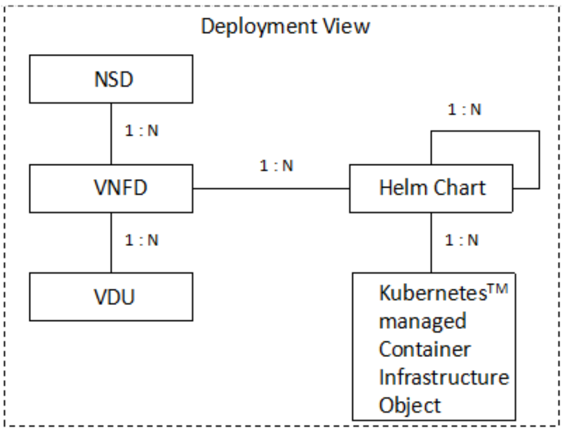


Figure 6.2.4-2: Example deployment object mapping for Kubernetes® and Helm™

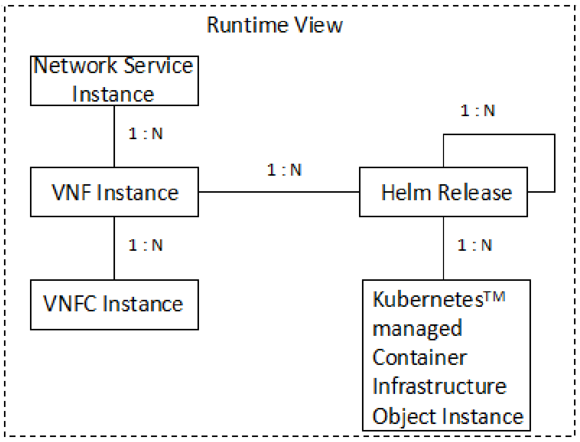


Figure 6.2.4-3: Example runtime object mapping for Kubernetes® and Helm™

6.2.5 Container Infrastructure Service Management exposure

The functionalities of the CISM are exposed to consumers of the NFV architecture via its APIs. The CISM APIs expose operations on the managed container infrastructure objects and managed container infrastructure object packages. However, because the managed container infrastructure objects lack equivalent objects in the current NFV information model, it is not recommended to manage the containerized workloads on an individual managed container infrastructure object granularity level. Instead it is recommended to manage the containerized workloads via the CISM APIs on a managed container infrastructure object package granularity level.

For the example of Kubernetes® and Helm™, containerized workloads (also known as containerized applications) are exposed and managed by the CISM via Helm™ Charts and instantiated as Helm™ Releases.

6.2.6 Container Image Management

According to the use cases described in clause 5, the CISM needs to provide functionality related to container image management, including retrieving information of a particular container image from NFV-MANO entities, maintaining that information and utilizing the container images to instantiate OS containers.

The container images are stored in one or more container image registries, which could be deployed locally or remotely (e.g. utilizing cloud services). The deployment of the registries is decided by the service provider, which could be part of the NFVI, within NFV-MANO or dedicated management entities. The container image registries are exposing APIs for managing the maintained container images, such as pushing and pulling container images, querying image information, and deleting container images. An example of such an API is specified by the OCI™ (https://www.opencontainers.org/) in their Distribution Specification.

The CISM needs to be configured to have access to one or more container image registries, and then pull necessary container images from the registries when instantiating OS containers.

The container images accessible from the container image registry are addressed and referenced by their names and the URLs of the respective container image registry.

7 Potential Architecture Enhancements

7.1 Architectural enhancement on PaaS related use cases

7.1.1 Overall NFV Architecture

7.1.1.1 General

Potential architectural enhancements depend on the design options selected to model PaaS services and thus PaaS services management. Three main design options can be envisioned:

- PaaS services are modelled as VNFs.
- PaaS services are modelled as a new type of NFVI resources.
- PaaS services are modelled as a new type of object specific to the PaaS layer.

7.1.1.2 PaaS services as VNFs

7.1.1.2.1 Overview

With this design option PaaS services are modelled as VNFs (see figure 7.1.1.2.1-1). It leverages the recursiveness of the NFV architectural framework. There are two types of VNFs involved in a network service: Consumer VNFs and the supporting VNFs, which are the VNFs hosting the VNF Common Services and the VNF Dedicated Services. Two variants can be considered depending on whether the VNFs hosting the PaaS services are visible at the NS design level or not.

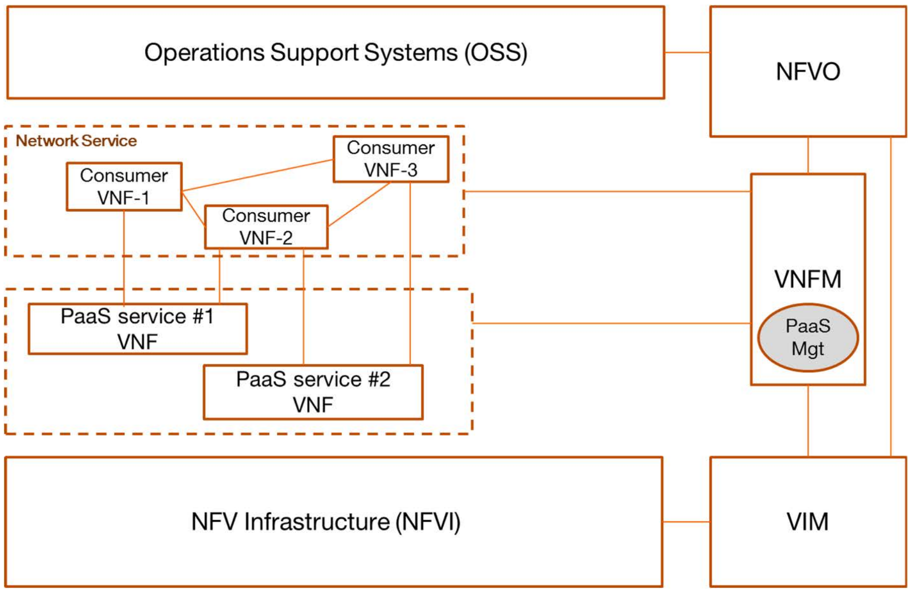


Figure 7.1.1.2.1-1: PaaS services as VNFs

7.1.1.2.2 PaaS services modelling

If the VNFs hosting the PaaS services are visible at the NS design level, the NSD references the VNFDs for instantiating both the VNFs hosting the PaaS services and the VNFs using these services (i.e. the Consumer VNFs). The dependencies between Consumer VNFs and PaaS VNFs can be specified in the "dependencies" attribute of NS deployment flavours within the NSD. Common services are modelled as VNF instances created within special purpose NS instances (e.g. ETSI GS NFV-IFA 013 [i.3], annex B). The NS instantiation request sent to the NFVO can reference an existing VNF instance, possibly shared with other NS instances in case of a PaaS common service.

Otherwise, the NSD only references the VNFDs used to instantiate Consumer VNFs and the dependencies of a Consumer VNF on PaaS services are specified as constraints within the VNFD of this Consumer VNF, for example by referencing the VNFD of the supporting VNFs from the VNFD of the Consumer VNF.

7.1.1.2.3 PaaS services management

When the VNFDs of the VNFs hosting PaaS services are visible in the NSD, the NFVO, the VNFM and the OSS (for VNF Common Services) play collectively the PaaS service management role. The OSS requesting the instantiation of the NS behaves as follows:

- When a Consumer VNF has a dependency on a PaaS common service, the OSS references an instance of a VNF hosting the PaaS common service in the NS instantiation request.

NOTE: It is assumed that the OSS has previously requested the instantiation of the VNFs hosting the PaaS common services within special purpose NS instances (e.g. ETSI GS NFV-IFA 013 [i.3], annex B).

- When a Consumer VNF has a dependency on a PaaS dedicated service, the OSS does not reference any VNF instance VNF hosting a PaaS service and assumes a VNF instance hosting the PaaS dedicated service will be created by the NFVO and VNFM according to standard NFV-MANO procedures.

When the VNFDs of the VNFs hosting PaaS services are not visible in the NSD, the VNFM plays the PaaS service management role. If the VNFD of a Consumer VNF specifies a dependency on a PaaS dedicated service, the VNFM triggers the instantiation of a VNF hosting the PaaS dedicated service.

- The connectivity between the Consumer VNF instance and the VNF instances hosting the PaaS services is managed using one of the following options.
- The connectivity between the Consumer VNF instance and the VNF instance hosting the PaaS service needs to be established. The mechanism to establish this connectivity is not addressed in the present document.

- A pre-established virtual link / virtual network is made available for connecting the appropriate connection points of the Consumer VNF. This virtual link provides access to all PaaS services. The identifier of this virtual link is provided by the NFVO in the granting response associated to the VNF instantiation procedure.

7.1.1.3 PaaS services as NFVI resources

7.1.1.3.1 Overview

With this design option PaaS services are modelled as a new type of NFVI resources (see figure 7.1.1.3.1-1).

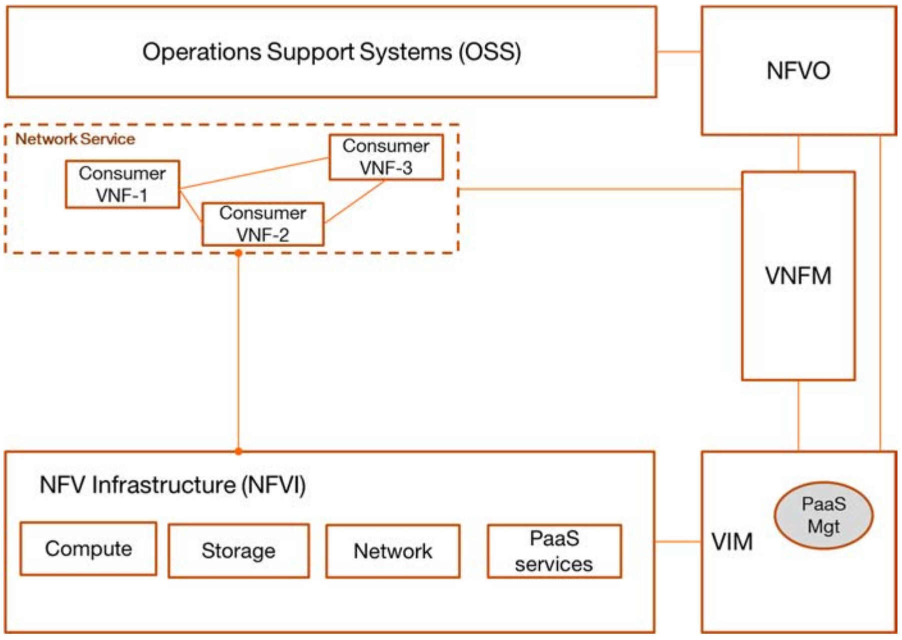


Figure 7.1.1.3.1-1: PaaS services as NFVI resources

7.1.1.3.2 PaaS services modelling

Both common and dedicated PaaS services are modelled as a new type of NFVI resources in a similar way as acceleration resources as described in ETSI GS NFV-IFA 004 [i.4] and illustrated in figure B.2.1-1. In such a case, the dependency of a Consumer VNF on a PaaS service is always specified as a constraint within the VNFD.

7.1.1.3.3 PaaS services management

The PaaS service management functionality is embedded in the VIM in a similar way as acceleration resources as described in ETSI GS NFV-IFA 004 [i.4] and illustrated in figure B.2.1-1.

When the VNFD of a Consumer VNF specifies a dependency on a PaaS dedicated service, the VNFM requests the VIM to create and allocate the corresponding resource.

When the VNFD of a Consumer VNF specifies a dependency on a PaaS common service, the VNFM requests the VIM to enable access to the corresponding resource from the virtualised compute resources hosting the Consumer VNF.

7.1.1.4 PaaS services as a new object class

7.1.1.4.1 Overview

With this design option PaaS services are modelled instances of a new object class that is specific to the PaaS concept (see figure 7.1.1.4.1-1).

NOTE: The term new object class is meant to represent an object that is not currently defined in the NFV information model, as described in ETSI GR NFV-IFA 015 [i.16].

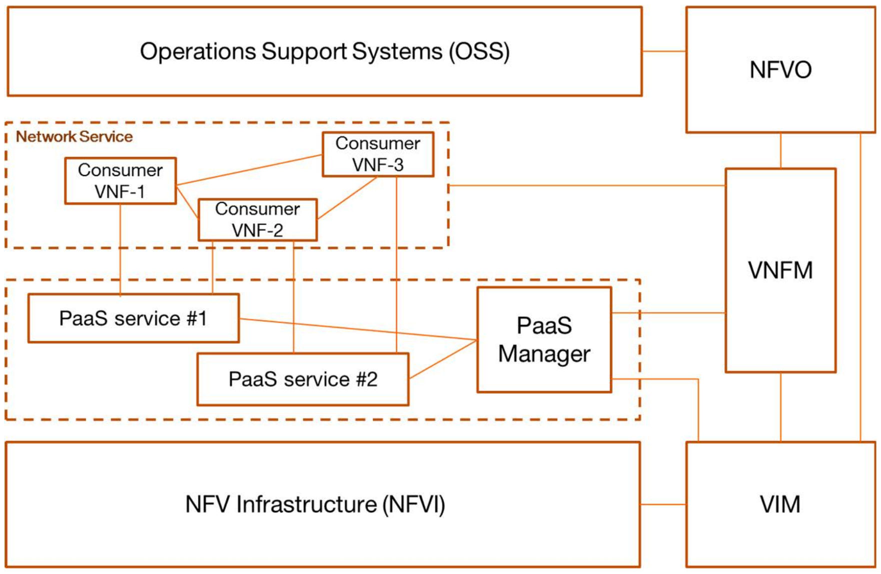


Figure 7.1.1.4.1-1: PaaS services as new architectural object class

7.1.1.4.2 PaaS services modelling

Both common and dedicated PaaS services are modelled instances of the new architectural object. In such a case, the dependency of a Consumer VNF on a PaaS service is always specified as a constraint within the VNFD.

7.1.1.4.3 PaaS services management

The PaaS service management functionality is provided by a new stand-alone functional block which consumes the services exposed by the VIM.

When the VNFD of a Consumer VNF specifies a dependency on a PaaS dedicated service, the VNFM requests the PaaS service manager to provide exclusive access to this PaaS service to the Consumer VNF being instantiated. Upon receipt of this request the PaaS service manager requests the VIM to create and allocate the corresponding resource.

When the VNFD of a Consumer VNF specifies a dependency on a PaaS common service, the VNFM requests the PaaS service manager to enable access to a shared instance of this PaaS service to the Consumer VNF being instantiated. Upon receipt of this request the PaaS service manager requests the VIM to enable access to the corresponding resource from the virtualised compute resources hosting the Consumer VNF.

NOTE: It is assumed that the PaaS service management has previously instantiated the PaaS common services.

7.1.2 MANO

7.1.2.1 PaaS services as VNFs

The following impact is identified:

- VNFD: The VNFD is extended with information about the dependencies of a Consumer VNF on PaaS services if the VNFs hosting these services are not visible in the NSD. This information includes a reference to the VNFDs to be used to instantiate VNFs hosting the PaaS services. Otherwise, this modelling approach has no impact on the VNFD.

- NSD: This modelling approach has no impact on the NSD.

- NFVO: This modelling approach has no impact on the NFVO.

- VNFM: The VNFM is extended with the capability to analyse the dependencies of a Consumer VNF on PaaS services if the VNFs hosting these services are not visible in the NSD and in case of VNF Dedicated Services with the ability to instantiate the corresponding supporting VNF instances (in which case the VNFM plays the PaaS service management role) and connect them to the Consumer VNF instance.

- VIM: This modelling approach has no impact on the VIM.

NOTE: The VIM functional block as currently specified is assumed to be able to manage multiple virtualisation technologies. Hence the virtualisation technology used to deploy the VNFs hosting the PaaS services might be different from the virtualisation technology used to deploy the Consumer VNFs. The VIM used to deploy the resources used by the Consumer VNFs and the VIM used to deploy the resources of the VNFs hosting PaaS services may but need not be the same.

7.1.2.2 PaaS services as NFVI resources

The following impact is identified:

- VNFD: The VNFD is extended with information about the dependencies of a Consumer VNF on PaaS services, in a similar way as the dependencies on other NFVI capabilities (e.g. acceleration capabilities).
- NSD: This modelling approach has no impact on the NSD.
- NFVO: This modelling approach has no impact on the NFVO.
- VNFM: This modelling approach has no impact on the VNFM beyond the ability to parse the dependency information and derive constraints to be submitted to the VIM when instantiating the required virtualisation resources.
- VIM: This approach needs a new module in the VIM for PaaS services management, similar to the acceleration management controller identified in ETSI GS NFV-IFA 004 [i.4], figure B.2.1-1.

7.1.2.3 PaaS services as a new object class

The following impact is identified:

- VNFD: The VNFD is extended with information about the dependencies of a Consumer VNF on PaaS services.
- NSD: This modelling approach has no impact on the NSD.
- NFVO: This modelling approach has no impact on the NFVO.
- VNFM: The VNFM is extended with the ability to parse the dependency information in the VNFD and to interact with the PaaS service manager (i.e. a new reference point is created between the VNFM and the PaaS service manager).
- VIM: This modelling approach has no functional impact on the VIM, but a new reference point is created between the VIM and the PaaS service manager.

NOTE: Furthermore, this solution has an impact on the overall NFV architectural framework as it identifies the need for a stand-alone PaaS service management functional block.

7.1.3 NFVI

7.1.3.1 PaaS services as VNFs

No impact on the NFVI is foreseen beyond the capability to support multiple virtualisation technologies and possibly nested virtualisation as well, as described in ETSI GS NFV-EVE 004 [i.1].

7.1.3.2 PaaS services as NFVI resources

This approach needs that PaaS services are to be modelled as a new type of virtualised resources, similar to acceleration resources.

7.1.3.3 PaaS services as a new object class

No impact on the NFVI is foreseen beyond the capability to support multiple virtualisation technologies and possibly nested virtualisation as well, as described in ETSI GS NFV-EVE 004 [i.1].

7.2 Architectural enhancement on container related use cases

7.2.1 Overall NFV Architecture

7.2.1.1 General

- Potential architectural enhancements on the following container related use cases focus on the aspect of CISM southbound/northbound interfaces, impact on NFV-MANO and impact on the NFVI.
- Nested Container Infrastructure Services in VMs hosting VNFCs in OS containers (see use case in clauses 5.3.4 and 5.3.5).
- Container Infrastructure Services on bare metal (see use case in clause 5.3.3).
- NFVI provides containers on bare metal and VM (see use case in clause 5.3.6).

7.2.1.2 Nested Container Infrastructure Services in VMs hosting VNFCs in OS containers

7.2.1.2.1 Overview

In the context of use cases in clauses 5.3.4 and 5.3.5, a design option of Container Infrastructure Services nested in virtual machines, hosting VNFCs in OS containers can be derived (see figure 5.3.4.1-1). The VNF is composed of containerized VNFCs and the function of a containerized VNFC is encapsulated in the container image and instantiated by the Container Infrastructure Service Management (CISM). An instance of a containerized VNFC runs on top of an instance of a Container Infrastructure Service which provides the container runtime environment. The function of container runtime environment is encapsulated in the virtual machine (VM) image and be instantiated in the NFVI layer.

7.2.1.2.2 CISM southbound interfaces

For the southbound interfaces of the CISM, the following interactions are analysed to identify potential architectural impacts in NFV-MANO:

- CISM-CIS interaction

  The CISM manages the CIS according to the request of the CISM northbound consumer. The CIS provides the container runtime environment (e.g. Docker^{TM}) which executes (e.g. start or stop) the containerized workload (e.g. VNF or VNFC) encapsulated in the OS container, and the containerized workload is managed by the CISM northbound consumer.

  The CIS provides multiple APIs to the CISM, including both infrastructure resource related ones and containerized workload related ones. For example, in Kubernetes$^{®}$ the infrastructure resource related APIs include Container Runtime Interface (CRI), Container Storage Interface (CSI) or Container Network Interface (CNI^{TM}), and containerized workload related APIs include the CRUD operations of Kubernetes$^{®}$ Pod or Service, which aligns with the concept "Managed Container Infrastructure Object" in clause 6.2.3. The APIs between the CISM and the CIS are out of the scope of ETSI NFV standards and generally follow de-facto container management standards like Kubernetes$^{®}$.

7.2.1.2.3 CISM northbound interfaces

For the northbound interfaces of the CISM, the following interactions are analysed to identify potential architectural impacts on NFV-MANO:

- CISM-VNFM interaction

  The VNFM consumes the lifecycle management of containerized workload running in OS containers which is provided by the CISM. As recommended in clause 6.2.5, the CISM APIs expose operations on managed container infrastructure object package to its northbound consumers. That means, the VNFM (as one of the CISM northbound consumers) translates the lifecycle management requirements for containerized VNF to the request on managing the container infrastructure object package and sends the request command to the CISM for the enforcement of container infrastructure object package management. It is an implementation issue of the CISM on how to further translate the incoming requests from the VNFM to operations which are enforced via communication towards the CIS.

- CISM-NFVO interaction

  The NFVO consumes the management of service resources for the containerized workload which is provided by the CISM in indirect mode.

7.2.1.3 Container Infrastructure Services on bare metal

7.2.1.3.1 Overview

In the context of use cases in clause 5.3.3, a design option of Container Infrastructure Services on bare metal can be derived (see figure 5.3.3.1-1). Compared to clause 5.3.4 use case on Nested Container Infrastructure Service in VMs hosting VNFCs in OS containers, the VNF is also composed of containerized VNFCs and the function of a containerized VNFC is encapsulated in the container image and instantiated by the Container Infrastructure Service Management (CISM). The main difference is that the image of container runtime environment is installed in the bare metal server in the NFVI layer.

7.2.1.3.2 CISM southbound interfaces

For the southbound interfaces of the CISM, the following interactions are analysed to identify potential architectural impacts in NFV-MANO:

- CISM-CIS interaction

  The analysis for CISM-CIS interaction in clause 7.2.1.2.2 applies for the use case of Container Infrastructure Services on bare metal.

7.2.1.3.3 CISM northbound interfaces

For the northbound interfaces of the CISM, the following interactions are analysed to identify potential architectural impacts on NFV-MANO:

- CISM-VNFM interaction

  The analysis for CISM-VNFM interaction in clause 7.2.1.2.3 applies for the use case of Container Infrastructure Services on bare metal.

- CISM-NFVO interaction

  The analysis for CISM-NFVO interaction in clause 7.2.1.2.3 applies for the use case of Container Infrastructure Services on bare metal.

7.2.1.4 NFVI provides containers on bare metal and VM

7.2.1.4.1 Overview

In the context of the use case discussed in clause 5.3.6, a design option for NFVI is to provide container infrastructure services on bare-metal and to provide container infrastructure services hosted by VMs as shown in figure 5.3.6.1-1. Maintaining consistency with the initial technology-agnostic definition of VIM, per ETSI GS NFV 003 [i.8], and combining this with the option to support multiple Virtualisation technologies of ETSI GS NFV-EVE 004 [i.1], clause 5.3.6 use case provides a hybrid scenario of both nested CIS in VMs hosting VNFCs in OS containers and CIS on bare metal.

7.2.1.4.2 CISM interfaces

The analysis in clauses 7.2.1.2.2, 7.2.1.2.3, 7.2.1.3.2 and 7.2.1.3.3 applies for the use case of NFVI providing containers on bare metal and VM.

7.2.2 MANO

7.2.2.1 Nested Container Infrastructure Service in VMs hosting VNFCs in OS containers

In the context of use cases in clauses 5.3.4, 5.3.5, 5.3.11 and 5.3.12, the following impact are identified:

- VNFD: The VNFD of a containerized workload is enhanced with the reference to one or more managed container infrastructure object packages which represent the containerized workload components. Infrastructure resources consumed by containerized workload are described in a declarative way by managed container infrastructure object packages. The information elements and their associated virtualised resource descriptors can be modified for the VNFD of a containerized workload, e.g. providing information necessary for managing container images, adding reference to one or more managed container infrastructure objects from a VDU. The VNFD of a containerized workload also includes the description for service resources which are used by managed container infrastructure objects in a managed container infrastructure object package.

- NSD: These use cases have no impact on the NSD.

- VIM: These use cases have no impact on the VIM. As in the existing NFV solutions, the VIM provides to its consumer the NFVI virtualised resources (compute, storage and network) within the operator's NFVI-PoP. The CISM is one of its northbound consumers.

- VNFM: The VNFM is enhanced with the capability to initiate the operations towards the CISM related to managed container infrastructure object package management. The VNFM is also enhanced with the capability to initiate the management of service resources for the containerized workload towards the NFVO in indirect mode, or initiate granting management of service resources for the containerized workload towards the NFVO in direct mode.

- NFVO: The NFVO is enhanced with the capability to access and consume interfaces exposed by a container image registry. The NFVO is also enhanced with the capability to initiate the management of service resources for the containerized workload towards the CISM in indirect mode, or grant service resources for the containerized workload requested by the VNFM in direct mode.

7.2.2.2 Container Infrastructure Service on bare metal

In the context of use cases in clauses 5.3.3, 5.3.11 and 5.3.12, the following impact is identified:

- VNFD: The analysis for VNFD impacts in clause 7.2.2.1 applies for the use case of Container Infrastructure Services on bare metal.

- NSD: These use cases have no impact on the NSD.

- VIM: These use cases have no impact on the VIM. The management of infrastructure resource in the bare metal server has no relation with the VIM for hypervisor-based virtualisation.

- VNFM: The analysis for VNFM impacts in clause 7.2.2.1 applies for the use case of Container Infrastructure Services on bare metal.
- NFVO: The analysis for NFVO impacts in clause 7.2.2.1 applies for the use case of Container Infrastructure Services on bare metal.

7.2.2.3 NFVI provides containers on bare metal and VM

In context of the use case discussed in clause 5.3.6, the following impacts are identified:

- VNFD: The analysis for VNFD impacts in clause 7.2.2.1 applies for the use case of NFVI provides containers on bare metal and VM.
- NSD: These use cases have no impact on the NSD.
- VIM: The capability of the VIM might be extended if container infrastructure object management is supported by the VIM.
- VNFM: The analysis for VNFM impacts in clause 7.2.2.1 applies for the use case of NFVI provides containers on bare metal and VM.
- NFVO: The analysis for NFVO impacts in clause 7.2.2.1 applies for the use case of NFVI provides containers on bare metal and VM.

7.2.3 NFVI

7.2.3.1 Nested Container Infrastructure Service in VMs hosting VNFCs in OS containers

The NFVI functionality needs to support the installation of virtual machines (VM) containing the container runtime environment image. Those VMs can be allocated by the VIM, which are be further used by the CISM for providing the Container Infrastructure Service to its consumer (e.g. the VNFM). The VMs containing container runtime environment image provided by the VIM may not be directly consumed by the VNFM for managing the lifecycle of containerized VNF.

7.2.3.2 Container Infrastructure Service on bare metal

This modelling approach has no impact on the NFVI. In the NFVI layer, the container runtime environment image is installed in the bare metal server and be further used by the CISM for providing the Container Infrastructure Service to its consumer (e.g. the VNFM).

7.2.3.3 NFVI provides containers on bare metal and VM

Container specific infrastructure support needs to be added in NFVI. However, this approach has no impact on the existing Hypervisor specific infrastructure support.

7.2.4 CISM to NFV-MANO mapping options

7.2.4.1 Introduction

Multiple options can be envisioned for mapping the CISM functionality to NFV-MANO. The following clauses describe several options and analyse their pros and cons.

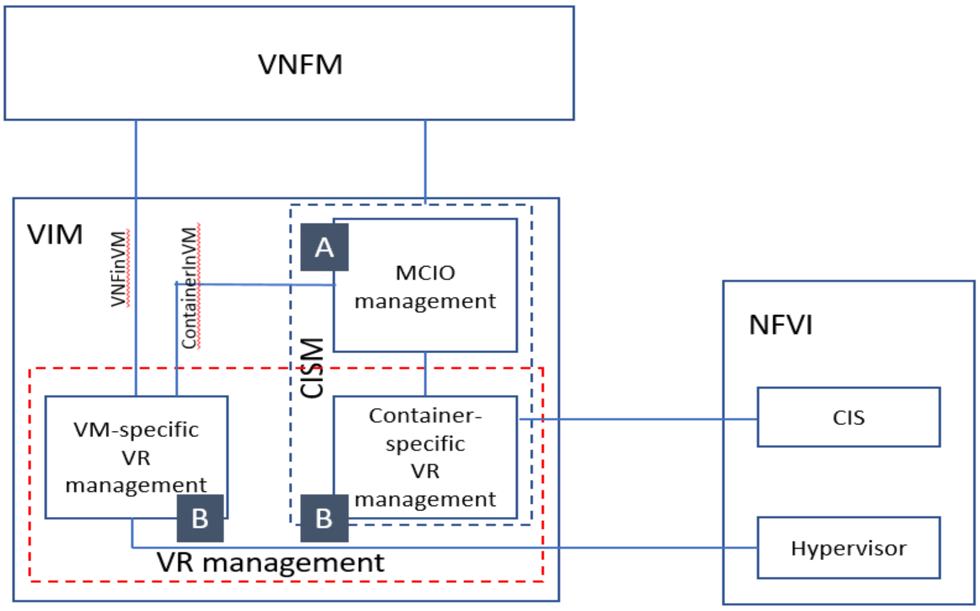
As described in clause 6.2.2, the CISM functionality has two main aspects:

- Function A: The management of managed container infrastructure objects (e.g. Kubernetes® Pods) with dynamic service resource allocation.
- Function B: The management of the virtualised resources exposed by the container runtime environment (i.e. the CIS). This functionality is not exposed to the CISM consumer.

In the following clauses, A and B are used to denote the two aspects of the CISM functionality.

7.2.4.2 Option#1: CISM embedded in the VIM

High-level description: The CISM functionality is embedded in the VIM, as illustrated in figure 7.2.4.2-1. Function A is an optional feature, that is only activated when containerized virtualisation is used.



NOTE: The figure focuses on the portion of NFV-MANO architecture relevant for the concepts of architectural impacts.

Figure 7.2.4.2-1: CISM embedded in the VIM

Relation to VIM:

- A VIM that only supports hypervisor-based Virtualisation only includes a VM-specific incarnation of Function B with an interface to hypervisors.
- A VIM that only supports containerized Virtualisation includes Function A and a container-specific incarnation of Function B with an interface to container infrastructure service instances in the NFVI.
- A VIM that supports both Virtualisation technologies includes Function A and both a container-specific incarnation of Function B and a VM-specific incarnation of Function B, each with different southbound interfaces to container infrastructure service instances or hypervisor (respectively).
- In case of the "Containers in VM" use case, Function A interacts with both incarnations of Function B.

Relation to other NFV-MANO functional blocks:

Depending on the type of VNF it manages, the VNFM is the client of the APIs exposed by Function A (in case of Containerized VNFs) or B (in case of VM-based VNFs).

Pros:

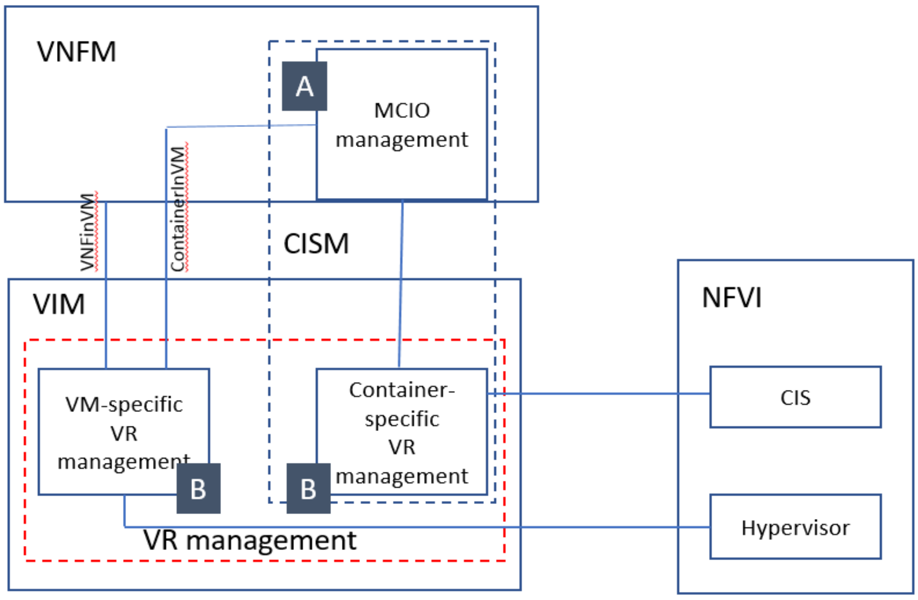
- Consistent with the initial technology-agnostic definition of VIM as per ETSI GS NFV 003 [i.8].

Cons:

- Adds an optional sub-function (Function A) to the VIM that is not applicable to all virtualisation technologies.
- Adds Function A to the VIM that exposes a higher abstraction level object model than currently assumed by ETSI GS NFV-IFA 005 [i.18] and ETSI GS NFV-IFA 006 [i.15].

7.2.4.3 Option#2: CISM distributed across VNFM and VIM

High-level description: The CISM functionality is spread across VNFM and VIM, supporting Function A and Function B, respectively, as illustrated in figure 7.2.4.3-1.



NOTE: The figure focuses on the portion of NFV-MANO architecture relevant for the concepts of architectural impacts.

Figure 7.2.4.3-1: CISM distributed across VNFM and VIM

Relation to VIM specifications:

- A VIM that only supports hypervisor-based Virtualisation only includes a VM-specific incarnation of Function B.
- A VIM that only supports containerized Virtualisation only includes a container-specific incarnation of Function B with an interface to container infrastructure service instances in the NFVI.
- A VIM that supports both Virtualisation technologies includes both a container-specific incarnation of Function B and a VM-specific incarnation of Function B, each with different southbound interfaces to container infrastructure service instances or hypervisor (respectively).
- In case of the Containers in VM use case, Function A in the VNFM interacts with both incarnations of Function B in the VIM.

Relation to other NFV-MANO functional blocks:

- A VNFM that supports containerized Virtualisation includes Function A.
- In case of the Containers in VM use case, Function A in the VNFM interacts with both incarnations of Function B in the VIM.

Pros:

- Consistent with the initial technology-agnostic definition of VIM.

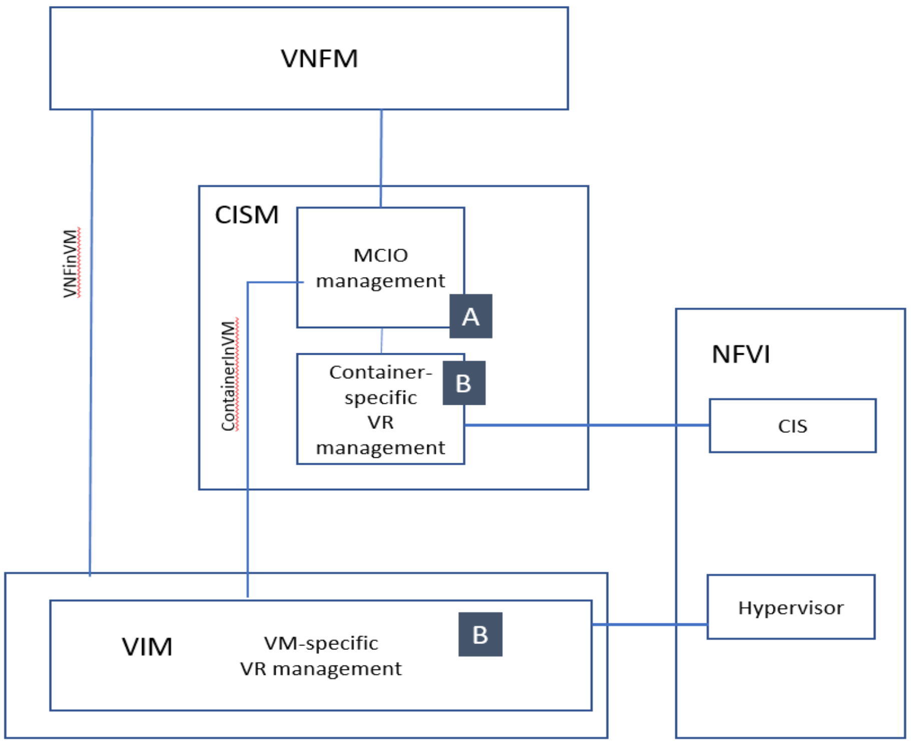
Cons:

- Interface between Function A and Function B becomes external and needs to be standardized.
- In the case of containerized incarnation of Function B, this interface might not match any real product design at the time of publication of the present document.

- Functional role of CISM spread over two functional blocks, requires synchronization between the two functional blocks.
- Exposure of complete managed container infrastructure object models to the VNFM, no abstraction. The VNFM does not stay technology-agnostic, it becomes aware of container Virtualisation.

7.2.4.4 Option#3: CISM as a stand-alone functional block

High-level description: The CISM functionality is assigned to new NFV-MANO functional block, named CISM as well, as illustrated in figure 7.2.4.4-1. The VIM is considered as a hypervisor-specific functional block.



NOTE: The figure focuses on the portion of NFV-MANO architecture relevant for the concepts of architectural impacts.

Figure 7.2.4.4-1: CISM as a stand-alone functional block

Relation to VIM specifications:

- In case of a Container on bare metal use case, the VIM is not involved.
- In case of a Container in VM use case, the CISM is the client of a VM-specific VIM.

Relation to other NFV-MANO functional blocks:

- A VNFM that supports containerized Virtualisation consumes Function A exposed by the CISM.

Pros:

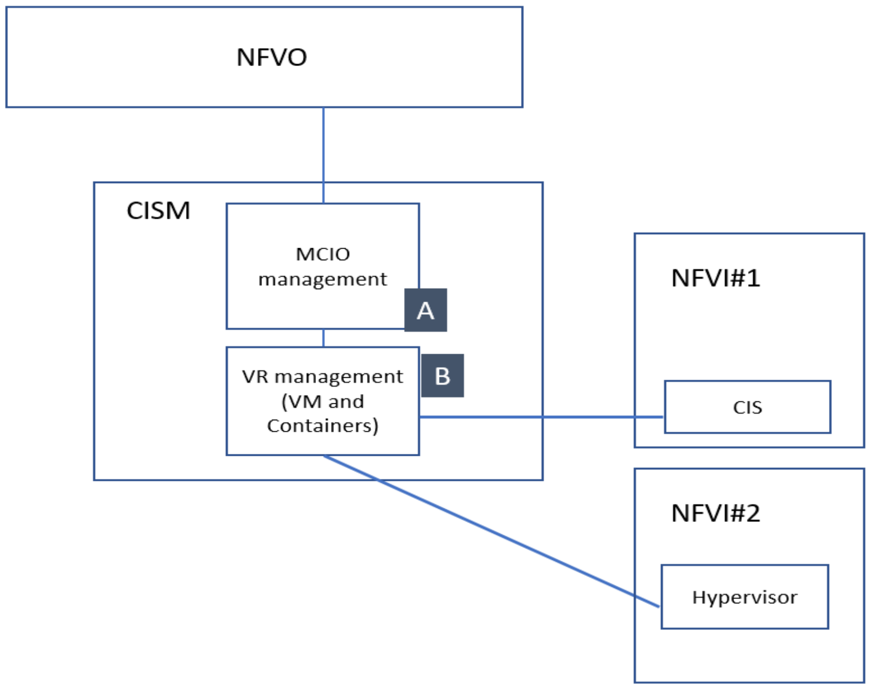
- Functional model is easy to map to established open source solutions, supporting de-facto standard interfaces.
- Clear separation of concerns for management of different Virtualisation technologies.
- Reflects the layered Virtualisation technologies of the NFVI in the NFV-MANO stack, especially for nested Virtualisation technologies such as container infrastructure services in hosted in virtual machines.

Cons:

- Implies redefining the concept of VIM to make it hypervisor-specific.
- The NFV functional architecture has two functional blocks with functionality Function B, requires synchronization between the two functional blocks.
- Potential functional overlap between the VIM and the CISM may go beyond Function B (e.g. Network Forwarding Path Management, Compute Host reservation Management, etc.).

7.2.4.5 Option#4: CISM-only replacing VIM and VNFM

High-level description: The CISM functionality is replacing the VIM and the VNFM, as illustrated in figure 7.2.4.5-1. The functionality of a VIM and many of the functionalities of the VNFM can be replaced by the CISM as well. The VNF can still be implemented using Virtual Machine(s) but managed by the CISM.



NOTE: The figure focuses on the portion of NFV-MANO architecture relevant for the concepts of architectural impacts.

Figure 7.2.4.5-1: CISM-only managing VMs and Containers

Relation to VIM specifications:

- Current VIM and many of the current VNFM functionalities are implemented in the CISM in a more automated way.

Relation to other NFV-MANO functional blocks:

- A NFVO that supports containerized Virtualisation consumes the service exposed by the CISM. This might require a new interface specification or a simplification of the existing interfaces. Some function of the VNFM might be better implemented in the NFVO than in the CISM.
- The VNFM functionality is in a simplified way integrated already in the CISM. The CISM is exposing a containerized interface, which is a higher-level of abstraction than VM-based approaches.

Pros:

- The functional model is easy to map to current open source solutions (e.g. Kubernetes®).

- Is more of a clear cut towards a new cloud-native architecture.
- Might simplify the architecture since the only objects exposed on the NFVO-CISM interface are managed container infrastructure objects (Kubernetes® calls them pods (set of containers)) or their packages, not single containers.
- The capability of also having VMs is a nice migration path towards a scenario where containers are the primary component and VMs are still able to run but managed by a CISM (eventually not with the full manageability as in the current specifications).

Cons:

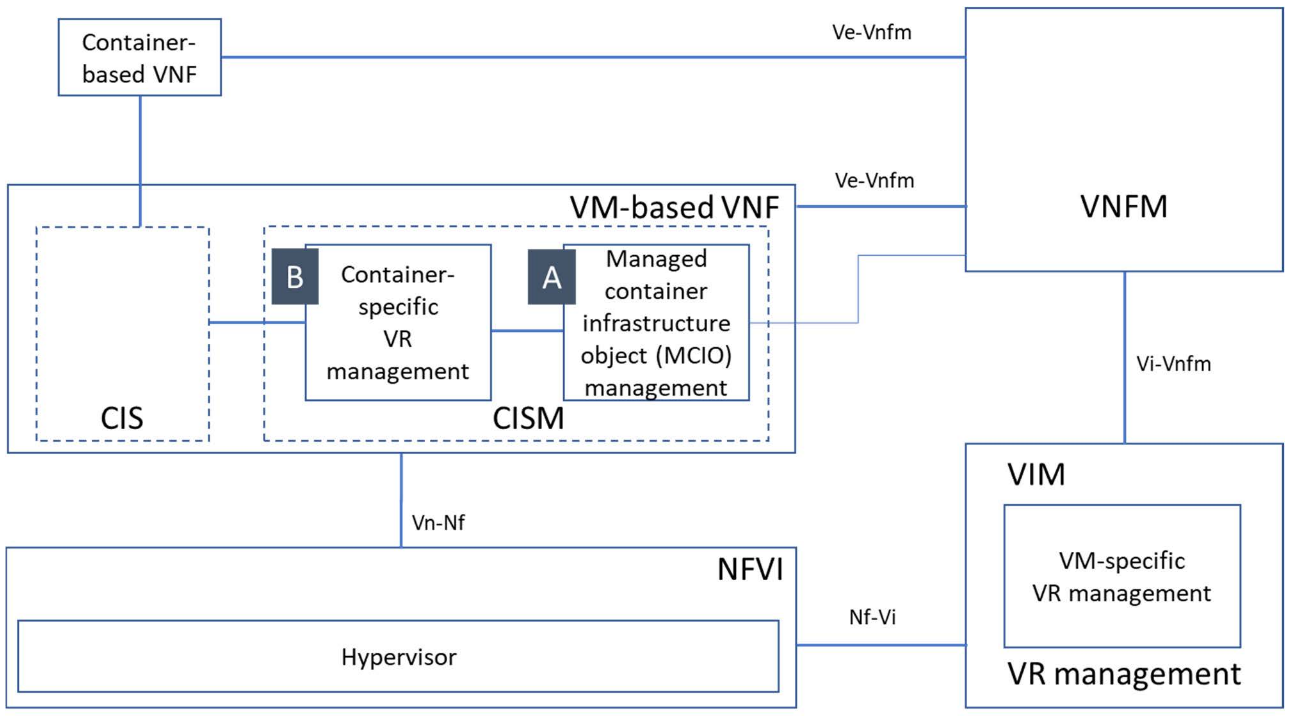
- Depending on the perspective, the merging of VNF management domain (VNFM) with the infrastructure management domain (VIM) can be regarded a cons, on the other hand it allows VNF management being more infrastructure management aware and therefore more automatable.
- Implies that a VIM and VNFM is not being used anymore in the way as specified in the NFV-MANO architecture. This will have backward compatibility issues with the NFV-MANO stack.
- VM-based VNFs might not get the full management capability as in the original NFV-MANO architecture.

7.2.4.6 Option#5: CISM embedded into VNF with support for shared container service

High-level description: The CISM functionality is embedded into a VNF and uses the NFVI resources allocated to the VNF as shown on figure 7.2.4.6-1. A VNF provides shared container services to one or more consumer, container-based, VNF instances. The VNFM instantiates the VM-based VNF and uses the CISM functionality of the created instance in a similar way as in Option#1 or Option#3.

NOTE 1: Whether the VM-based VNF providing the CISM functionality is integrated or not into a NS, and the implications in terms of connectivity and overall NS orchestration is not analysed in the present document.

NOTE 2: While figure 7.2.4.6-1 depicts the CISM functionality and the shared container services as pertaining to the same VNF, other scenarios where the CISM functionality is running in a separate VNF are also possible.


Container-based VNF
Ve-Vnfm
VNFM
Ve-Vnfm
VM-based VNF
B
Container-specific VR management
A
Managed container infrastructure object (MCIO) management
CIS
CISM
Vi-Vnfm
VIM
Vn-Nf
NFVI
VM-specific VR management
Nf-Vi
Hypervisor
VR management

NOTE: The figure focuses on the portion of NFV-MANO architecture relevant for the concepts of architectural impacts.

Figure 7.2.4.6-1: CISM embedded into VNF with support for shared container service

Relation to VIM specifications:

- No change to the current VIM specifications.

Relation to other NFV-MANO functional blocks:

- A VNFM that supports containerized Virtualisation consumes the service exposed by the CISM embedded into a VNF.

Pros:

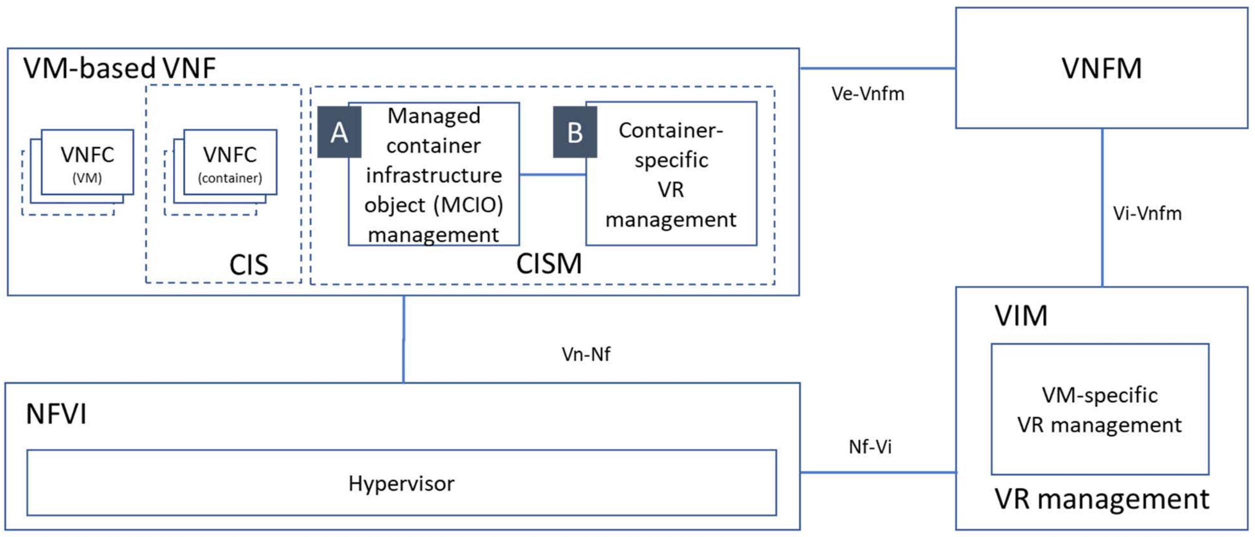
- Optimal usage of CISM compared to Option#6.

Cons:

- Nested resource usage is possible when one CISM serves several VNF instances. Mapping of these resources to a single resource and modelling the relations between the resources should be specified.
- No clear separation of functions (CISM and CIS is part of a VNF).
- Exposure of complete managed container infrastructure object models to the VNFM, no abstraction. The VNFM does not stay technology-agnostic, it becomes aware of container Virtualisation.

7.2.4.7 Option#6: CISM embedded into VNF without support for shared container service

High-level description: The CISM functionality is embedded into a VNF and uses the NFVI resources allocated to the VNF as shown on figure 7.2.4.7-1. In this option the CISM specific details are not visible for the VNFM unless they are communicated on Ve-Vnfm reference point. The lifecycle management of CISM and CIS are internal operations of the VNF.


VM-based VNF
VNFC (VM)
VNFC (container)
CIS
A
Managed container infrastructure object (MCIO) management
B
Container-specific VR management
CISM
Ve-Vnfm
VNFM
Vi-Vnfm
VIM
VM-specific VR management
VR management
Vn-Nf
NFVI
Hypervisor
Nf-Vi

NOTE: The figure focuses on the portion of NFV-MANO architecture relevant for the concepts of architectural impacts.

Figure 7.2.4.7-1: CISM embedded into VNF without support for shared container service

Relation to VIM specifications:

- No change to the current VIM specifications.

Relation to other NFV-MANO functional blocks:

- No change.

Pros:

- No change to the current architecture and interfaces.

Cons:

- No clear separation of functions (CISM and CIS is part of a VNF).
- Exposure of complete managed container infrastructure object models to the VNFM, no abstraction. The VNFM does not stay technology-agnostic, it becomes aware of container Virtualisation.

7.2.4.8 Comparison of the mapping options

This clause provides a comparison of the different options for mapping the CISM functionality to NFV-MANO, described in the previous clauses.

The comparison of the different options is provided in table 7.2.4.8-1.

Table 7.2.4.8-1: Comparison of the CISM to NFV-MANO functional mapping options

Evaluation Criteria	**Option#1: CISM embedded in the VIM**	**Option#2: CISM distributed across VNFM and VIM**	**Option#3: CISM as a stand-alone functional block**	**Option#4: CISM-only replacing VIM and VNFM**	**Option#5: CISM embedded into VNF with support for shared container service**
Maps to established open source solutions and de-facto standardized interfaces	Yes	No	Yes	Yes	Yes
Is consistent with the initial technology-agnostic definition of VIM	Yes	Yes	No	No	Yes
Provides a clear separation of concerns for the management of different Virtualisation technologies	No	No	Yes	No	Yes
Provides a single abstraction level exposed per functional block	No	No	Yes	No	No
Assigns the functional role of the CISM to one functional block	Yes	No	Yes	Yes	No
Provides a functional model abstracted from the virtualisation technology	Yes	No	No	Yes	Yes
Consolidates the management of individual virtualisation containers/resources in a single functional block	Yes	Yes	No	Yes	No
NOTE: This table does not include Option#6 (clause 7.2.4.7) as this option does not have any architectural impact.					

Every option in table 7.2.4.8-1 has its own pros and cons. To determine the target architecture for managing containers in the scope of NFV-MANO, the principle of backward compatibility and convenient mapping to the existing de-facto standards from open source community should be prioritized. Furthermore, the working border between ETSI standards and de-facto standards needs to be clear and have no duplicated or conflict implementation with each other.

Following these principles, Option 2 and 4 are determined to be excluded from the target architecture. Option 2 extends interface between MCIO management (Function A) and container-specific VR management (Function B) of the CISM to be external and eventually duplicates Vi-Vnfm reference point functionality as specified in ETSI GS NFV-IFA 006 [i.15] with internal APIs of de-facto standard (e.g. Kubernetes®) which are not exposed to northbound consumers. Option 4 is applying the container-oriented management approach to both containers and VMs, but has explicit backward compatibility issues, which cannot well be adapted to support the existing VM-based VNF management as specified in NFV-MANO.

Option 1 and 3 have great similarities in container management related APIs exposed to northbound consumers of the CISM. The main difference between these two options is the location to expose those APIs in NFV-MANO. Option 1 regards the CISM as a deployment entity which is embedded in the VIM, and Option 3 provides a logical view of the CISM which focuses on its interaction with other logical functions in NFV-MANO. On the other hand, both options have received concerns for potentially limiting their application. Option 1 needs to further resolve the gap between MCIO management (Function A) of the CISM and the scope of the existing VIM, which is initially designed for managing NFVI compute, storage and network resources. Option 3 needs to resolve potentially duplicated implementation of virtualised resource management between the CISM and the VIM, which will bring further synchronization between these two functions. It is left for normative work to provide a unified target architecture with the baseline of Option 1 and 3.

Option 5 has no impact on the existing NFM-MANO architecture functional blocks and interfaces, but it arranges these blocks in an unusual way. As a result of this, it is possible that one VNF is using the resources of another VNF. This aspect might have an effect in the modelling of the resources. The modelling of these nested resource usage cases was determined to be excluded from the target architecture.

Option 6 has no impact on the existing NFV-MANO architecture and can be regarded as a short-term solution for fulfilling container management in NFV-MANO where no extra effort for improving the existing NFV-MANO functionality is foreseen.

8 Recommendations and next steps

8.1 Recommendations derived from container related use cases

8.1.1 Managed Container Infrastructure Object Package

It is recommended that a Managed Container Infrastructure Object Package (MCIOP) is specified. The specification of the VNFD is recommended to be extended with the following aspects:

- It is recommended that the VNFD be extended with the capability to reference one or more MCIOP(s).

NOTE: The MCIOP contains dependencies, configurations and constraints to install and run the containerized workload.

- It is recommended that the VDU be extended with the capability to reference the descriptor templates of one or more MCIO(s) within the MCIOP(s) referenced by the VNFD.
- It is recommended that the VNFD be extended with the capability to provide the description of service resources used by MCIO(s) within a MCIOP.
- It is recommended that the VNFD be extended with the capability to provide all information necessary for the management of container images.

8.1.2 Interface to Container Infrastructure Service Management (CISM)

It is recommended that interfaces to Container Infrastructure Service Management (CISM) be specified taking into account container-specific features. As part of the CISM, functionality for MCIO management is specified for operations on managed container infrastructure objects, and functionality for MCIOP management is specified for operations on managed container infrastructure object packages.

The following aspects on interfaces to the CISM are recommended to be specified in the normative work:

- It is recommended that the CISM be enabled to provide the management of MCIOP(consumed by the VNFM).
- It is recommended that the CISM be enabled to provide the management of service resource related to containerized workload (consumed by the NFVO).

It is recommended that ETSI ISG NFV specifies requirements for the interfaces to the CISM (stage 1 and 2) and leave stage 3 to external bodies (e.g. de-facto standards).

8.1.3 Container Image Management

It is recommended that a container image registry function be specified. In addition, it is recommended to specify an interface (consumed by e.g. the NFVO) to access the container image registry for storing container images into the registry and deleting container images from the registry.

8.2 Recommendations derived from PaaS related use cases

8.2.1 PaaS Service Descriptor Catalogue

It is recommended that a function for a PaaS Service Descriptor Catalogue be specified, which is the basis for PaaS service management. The PaaS Service Descriptor Catalogue function maintains PaaS Service Descriptors and provides an interface for listing available PaaS services, which can be deployed as VNF Common Services or VNF Dedicated Services.

8.2.2 PaaS Service Registry

It is recommended that a function for a PaaS Service Registry be specified. The PaaS Service Registry function provides an interface for listing consumable PaaS service instances, and for getting information about the service access points and the mechanisms to consume the PaaS services. The PaaS Service Registry is the basis for service management of deployed and running PaaS services.

NOTE: It can be decided at the specification stage, that the functionality for PaaS Service Registry and PaaS Service Descriptor Catalogue is split or common, since it might make the function allocation more optimal.

8.3 Recommendations derived from cloud-native related use cases

No additional normative work is foreseen to be derived from cloud-native related use cases, according to the study in the present document.

8.4 Recommendations related to Security

ETSI GS NFV-SEC 012 [i.22] defines requirements for host system elements on which sensitive workloads are to be run. It also defines requirements to ensure isolation of sensitive workloads from non-sensitive workloads sharing a platform. Moreover, it discusses a wide range of different technologies which aim to increase the security of a host system for the workloads which will be executing on it.

The present clause provides recommendations focusing on security aspects. Table 8.4-1 provides the recommendations related to security. The present clause on recommendations related to security assumes that all entities are within one administrative domain.

Table 8.4-1: Recommendations related to Security

Identifier	Recommendation description	Comments and/or traceability
SEC.001	It is recommended that a requirement be defined for the system to provide means to ensure data integrity of any message (e.g. management of MCIOP).	
SEC.002	It is recommended that a requirement be defined for the system to provide means to verify the identity and authenticity of the source of any message (e.g. management of MCIOP).	

Identifier	Recommendation description	Comments and/or traceability
SEC.003	It is recommended that a requirement be defined for the system to provide means to verify the identity and authenticity of the consumer of any message (e.g. management of MCIOP).	
SEC.004	It is recommended that a requirement be defined for the system to provide means to ensure that requests sent by unauthorized entities are discarded and that only requests sent by authorized entities are processed. An example is to request to create a MCIOP for containerized VNF. See note.	
SEC.005	It is recommended that a requirement be defined for the system to raise an appropriate alarm in reaction to unauthorized operation attempts. See note.	
SEC.006	It is recommended that a requirement be defined to ensure the authenticity of the provider of an interface.	
SEC.007	It is recommended that a requirement be specified to verify the identity and authenticity of the MCIOP instances associated to a containerized VNF instance.	
SEC.008	It is recommended that a requirement be defined for the system to provide means to ensure the confidentiality of any system message to ensure it is not made available or disclosed to unauthorized entities. See note.	
Sec.009	It is recommended that a requirement be defined for the system to provide means to ensure that only authorized entities can take any action on the system (e.g. post queries, send requests, subscribe to receive notifications, provide and consume fault information report and resource information). See note.	
SEC.010	It is recommended that a requirement be defined for the system to provide means to ensure non-repudiation of the act of having sent a message by a sender.	
SEC.011	It is recommended that a requirement be defined for the system to provide means to prevent replay of any queries, requests, notification, fault information report or resource information.	
SEC.012	It is recommended that a requirement be defined for timestamps to be issued by a trusted source of time.	
SEC.013	It is recommended that a requirement be defined for the system to ensure against the unauthorized access to connectivity control, and control over any resources and the related information.	
SEC.014	It is recommended that a requirement be defined for the credentials used in the proof of identity (such as passwords, cryptographic keys, or key material) be properly secured by the sender and the receiver	
SEC.015	It is recommended that a requirement be defined for using a specific set of security best practices to be followed for all operations.	
SEC.016	It is recommended that a requirement be defined for the isolation among workloads running on OS-containers.	
SEC.017	It is recommended that a requirement be defined for the consistency of security policies among containerized workloads.	
SEC.018	It is recommended that a requirement be defined for the protection of container images.	
SEC.019	It is recommended that a requirement be defined for the protection of container image delivery from container image registries to container engines/host server.	
NOTE: With regards to the authorization relationship between the CISM and its northbound consumers (i.e. the NFVO or the VNFM in the context of the present document), for request/response related interface operations (e.g. MCIOP management, service resource management, subscription management), the CISM is the "resource server" and its northbound consumer is the "client" as defined by IETF RFC 6749 [i.23]. For notifications related operations, the CISM is the "client" and its northbound consumer is the "resource server" as defined by IETF RFC 6749 [i.23].		

Annex A: Related Industry Initiatives

A.1 Cloud Foundry

Cloud Foundry™ (https://www.cloudfoundry.org) is an open source project for cloud applications and PaaS. The Cloud Foundry platform is used to build, test, deploy and scale applications. It can be used on a selection of clouds, with different developer frameworks, and application services. Cloud Foundry has a container-based architecture that runs apps in any programming language. Several types of containers and container management systems can be used.

Of particular interest to the PaaS discussion in the present document is the mechanisms used for providing and using services (also known as composition of services). Typical applications depend on services such as databases, messages buses, logging and monitoring features or third-party APIs. Applications deployed on a Cloud Foundry platform access external services via a specific API for late binding of a service to the application and the service uses a particular API to register and being accessible in the platform.

For more details on the mechanism of invoking a service from an application and the late binding mechanism refer to [i.14].

A.2 Twelve Factor Applications

The twelve-factor application is an example of defining modern software architecture for applications using best practices. The target applications are primarily web-application rather than network-oriented applications. Therefore, this approach cannot be taken as the only way, rather it inspires the way of thinking for modern cloud-native and PaaS-type of application development. Details can be found at [i.2].

The twelve-factor application is a methodology for building software-as-a-service applications that:

- use declarative formats for setup automation, to minimize time and cost for new developers joining the project;
- have a clean contract with the underlying operating system or cloud platform, offering maximum portability between execution environments;
- are suitable for deployment on modern cloud platforms, obviating the need for servers and systems administration;
- minimize divergence between development and production, enabling continuous deployment for maximum agility; and
- can scale up without significant changes to tooling, architecture, or development practices.

The twelve-factor methodology can be applied to applications written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc.). The following 12 factors are listed:

Codebase: One codebase tracked in revision control; many deploys into various environments. There is only one codebase per application, but there will be many deploys of the application.

Dependencies: Explicitly declare and isolate dependencies. A twelve-factor application never relies on implicit existence of system-wide packages.

Configuration: Store configuration in the environment. A configuration is everything that is likely to vary between deploys (staging, production, developer environments, etc.). This includes:

- Resource handles to the database, memory cache and other backing services.
- Credentials to external services such as public cloud services.
- Per-deploy values such as the canonical hostname for the deploy.

Backing Services: Treat backing services as attached resources. A backing service is any service the application consumes over the network as part of its normal operation. Examples include datastores, messaging/queueing systems, SMTP services for outbound email and caching systems.

Backing services like the database are traditionally managed by the same systems administrators, who deploy the application's runtime. In addition to these locally-managed services, the application may also have services provided and managed by third parties.

Build, release, run: Strictly separate build and run stages. For example, it is impossible to make changes to the code at runtime, since there is no way to propagate those changes back to the build stage.

Processes: Execute the application as one or more stateless processes. Any data that needs to persist is required to be stored in a stateful backing service, typically a database. Sticky sessions are a violation of twelve-factor and should never be used or relied upon.

Port Binding: Export services via port binding. The twelve-factor application is completely self-contained and does not rely on runtime injection of a webserver into the execution environment to create a web-facing service. The web application exports HTTP as a service by binding to a port and listening to requests coming in on that port.

Concurrency: Scale out via the process model. In the twelve-factor application, processes are first-class citizens. Processes in the twelve-factor application take strong cues from the Unix process model for running service daemons. Using this model, the developer can architect their application to handle diverse workloads by assigning each type of work to a process type. The share-nothing, horizontally partitionable nature of twelve-factor application processes means that adding more concurrency is a simple and reliable operation. The array of process types and number of processes of each type is known as the process formation.

Disposability: Maximize robustness with fast start-up and graceful shutdown. The twelve-factor application's processes are disposable, meaning they can be started or stopped at a moment's notice. This facilitates fast elastic scaling, rapid deployment of code or configuration changes, and robustness of production deploys.

Development/Production parity: Keep development, staging, and production as similar as possible. The twelve-factor application is designed for continuous deployment by keeping the gap between development and production small. Looking at three gaps:

- Make the time gap small: a developer may write code, and have it deployed hours or even just minutes later.
- Make the personnel gap small: developers who wrote code are closely involved in deploying it and watching its behaviour in production.
- Make the tools gap small: keep development and production as similar as possible.

Logs: Treat logs as event streams. A twelve-factor application never concerns itself with routing or storage of its output stream this is an issue for the environment or in some cases for a backing service.

Administration processes: Run admin/management tasks as one-off processes. Developers will often wish to do one-off administrative or maintenance tasks for the application including for example database migration, running a shell with commands, or running some scripts.

NOTE: At the time of the creation of the present document, the twelve-factor app concept is most commonly used, but discussions on "Beyond 12 Factor App" for highly scalable and resilient cloud application have already started.

A.3 Cloud Native Computing Foundation® (CNCF®)

The Cloud Native Computing Foundation® is hosted by The Linux Foundation® and maintains Kubernetes®, one of the de-facto standard open source systems for managing containerized applications.

In their mission statement [i.9], they say:

1) Mission of the Cloud Native Computing Foundation®.

   The Foundation's mission is to create and drive the adoption of a new computing paradigm that is optimized for modern distributed systems environments capable of scaling to tens of thousands of self-healing multi-tenant nodes.

   Cloud native systems will have the following properties:

   a) Container packaged. Running applications and processes in software containers as an isolated unit of application deployment, and as a mechanism to achieve high levels of resource isolation. Improves overall developer experience, fosters code and component reuse and simplify operations for cloud native applications.

   b) Dynamically managed. Actively scheduled and actively managed by a central orchestrating process. Radically improve machine efficiency and resource utilization while reducing the cost associated with maintenance and operations.

   c) Micro-services oriented. Loosely coupled with dependencies explicitly described (e.g. through service endpoints). Significantly increase the overall agility and maintainability of applications. The foundation will shape the evolution of the technology to advance the state of the art for application management, and to make the technology ubiquitous and easily available through reliable interfaces.

A more business-focused definition is given in the Trail-Map [i.10]:

Cloud-native technologies, such as containers and microservices, empower organizations to develop and deploy scalable, agile applications and services in dynamic, distributed environments. By taking into account these characteristics, such systems are designed to be resilient, elastic, and loosely coupled, via manageable abstractions and declarative APIs, thereby enabling effective, reliable automation. This allows engineers to observe the applications and to safely make impactful changes, and results in processes and workflows that fully take advantage of these environments and minimize toil.

The trail map shows 10 steps for that in graphical way:

1) Containerization
2) CI/CD
3) Orchestration & Application Definition
4) Observability & Analysis
5) Service Mesh and Discovery
6) Networking
7) Distributed Database
8) Messaging
9) Container Runtime
10) Software Distribution

CNCF® currently hosts a number of projects (https://www.cncf.io/projects/), which cover the different technologies aspects mentioned in the trail map. Among them, some projects are widely adopted in the industry and worth consideration when implementing the cloud-native design.

Kubernetes®:

Kubernetes® is a portable, extensible open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. Kubernetes® has a number of features, it provides a container-centric management environment and can orchestrate computing, networking, and storage infrastructure on behalf of user workloads. This provides much of the simplicity of Platform as a Service (PaaS) with the flexibility of Infrastructure as a Service (IaaS), therefore Kubernetes® can be considered as a container platform, a microservice platform, as well as a portable cloud platform, etc.

Kubernetes® provides various objects to represent the desired state of the cluster and manages the cluster based on these objects to ensure the state of the cluster is consistent with what the objects define. Kubernetes® objects can describe what containerized applications are running (and on which nodes), the resources available to those applications and the policies around how those applications behave, such as restart policies, upgrades, and fault-tolerance. Typical objects including Pod, ReplicaSet, Deployment, Service, etc.

1) Pod

   The smallest and simplest unit of Kubernetes® object. A Pod encapsulates an application container (or, in some cases, multiple containers), storage resources, a unique network IP, and options that govern how the container(s) should run.

2) ReplicaSet

   ReplicaSet describes that a specified number of Pod replicas are desired to be running at the time.

3) Deployment

   Deployment provides declarative updates for Pods and ReplicaSets. Kubernetes® will change the current state to the desired state described by the Deployment object at a controlled rate.

4) Service

   Service is an abstraction which defines a logical set of Pods and a policy by which to access them - sometimes called a micro-service. It provides a permanent frontend which keeps track of the mortal backend Pods.

Helm™:

Helm™ is a tool for managing applications which are built on top of Kubernetes®. It provides Helm™ charts which are packages of pre-configured Kubernetes® resources. Helm™ manages charts to enable application packing, sharing, install and version control. There are several main concepts in Helm™:

- Chart: A Helm™ package. It contains all of the resource definitions necessary to run an application, tool, or service inside of a Kubernetes® cluster.
- Repository: collects and shares the charts.
- Release: an instance of a chart running in a Kubernetes® cluster. One chart can often be installed many times into the same cluster. And each time it is installed, a new release is created.
- Helm™ client: a command-line client for end users. It is responsible for managing charts.

Helm™ installs charts into Kubernetes®, which creates a new release for each installation. And new charts can be discovered in the Helm™ chart repositories. See figure A.3-1 for a better understanding of the architecture of Helm™ 3.

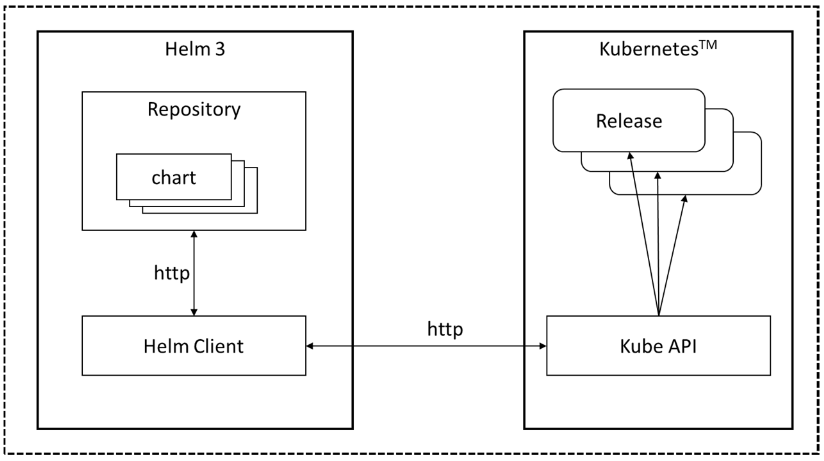


Figure A.3-1: Architecture for Helm^{TM} 3

Helm^{TM} chart is organized as a collection of files inside of a directory. The directory name is the name of the chart (without versioning information). A typical example of chart is as follow:

```
wordpress/                  # Name of the directory (and the chart)
        Chart.yaml          # A YAML file containing information about the chart
        LICENSE             # OPTIONAL: A plain text file containing the license for the chart
        README.md           # OPTIONAL: A human-readable README file
        requirements.yaml   # OPTIONAL: A YAML file listing dependencies for the chart
        values.yaml            # The default configuration values for this chart
        charts/             # A directory containing any charts upon which this chart depends
        templates/          # A directory of templates that, when combined with values
                            # will generate valid Kubernetes® manifest files
        templates/NOTES.txt # OPTIONAL: A plain text file containing short usage notes
```

CNI^{TM}:

CNI^{TM} (Container Network Interface), is a project which consists of a specification and libraries for writing plugins to configure network interfaces in Linux containers, along with a number of supported plugins. It is considered as a good framework to develop container network management solutions.

CNI^{TM} provides the capability to add a container to a network, delete a container from the network, check network status, report plugin versions, and support managing different network objects by various plugins, e.g. bridge, ipvlan, dhcp, etc.

A.4 Container Infrastructure Service Management / Container Orchestration Engine

Container Infrastructure Service Management (CISM) or Container Orchestration Engine (COE) is a distributed software what manages containers on a cluster of host computers. The CISM/COE provides an API for lifecycle management for the containers, schedules the containers based on their needs and the available resources, provides storage and connectivity for the containers. The host computers are either virtual machines or bare metal servers. The CISM/COE does not include the host operating system or the container runtime. Prominent CISM/COE implementations, at the time of writing the present document, are Kubernetes®, Apache Marathon with Mesos and Docker^{TM} (Swarm).

A.5 Open Container Initiative^{TM} (OCI^{TM})

The OCI^{TM} (https://www.opencontainers.org/) is an open-source The Linux Foundation® project based on activities originally initiated by Docker^{TM} and CoreOS among others. The objective is to define a standard container image format, and a set of standardized interfaces and behaviours on how to start a container. Both the OCI^{TM} image specification and the OCI^{TM} runtime specification have a reference implementation by OCI^{TM}.

As of September 2018, OCI^{TM} has published the following image and run-time specifications.

The OCI^{TM} Image specification [i.11] defines how to create an OCI^{TM} Image, including building a system, outputting an image manifest, a file system (layer) serialization, and an image configuration. The image manifest contains metadata about the contents and dependencies of the image that are required to build the final runnable system.

The OCI^{TM} Runtime Specification [i.12] defines how to download an OCI^{TM} Image, unpack that image into an OCI^{TM} Runtime file system bundle, and then run the "filesystem bundle" on disk.

A.6 OpenStack®

OpenStack® is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacentre, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface. There are a number of solutions for installing Kubernetes® and other application frameworks on top of OpenStack® [i.20].

Magnum:

Magnum is an OpenStack® project that provides a simple API [i.21] to deploy fully managed container clusters backed by a choice of several application platforms, including Kubernetes®. It is an example of a Kubernetes® deployment system that relies on OpenStack® APIs and cloud provider plugin.

Zun:

Zun is another OpenStack® project that offers a lighter-weight container service API [i.19] for managing individual containers without the need for managing servers or clusters.

OpenStack® Zun provider for Virtual Kubelet connects Kubernetes® cluster with OpenStack® in order to run Kubernetes® pods on OpenStack® Cloud. The pods on OpenStack® have access to OpenStack® tenant networks because they have Neutron ports in the subnets. Each pod will have private IP addresses to connect to other OpenStack® resources (i.e. VMs) within the tenant, optionally have floating IP addresses to connect to the internet, and bind-mount Cinder volumes into a path inside a pod's container.

Annex B: Example Description of Micro-service Architectural Pattern from Wikipedia

Micro-service architecture is a software architecture style in which complex applications are composed of small, independent services communicating with each other using language-agnostic APIs. These services are small, highly decoupled and focus on doing a small task, facilitating a modular approach to system-building. The 'small' mainly refers to the functional scope, and usually the small functional scope can result in the small code size and also the small size of the DevOps team, but the functional scope is the primer criteria.

According to the description in Wikipedia (https://en.wikipedia.org/wiki/Microservices), the defining characteristics for the services in the micro-service architecture that are frequently cited include:

- Services are often processes that communicate with each other over a network in order to fulfil a goal using technology-agnostic protocols.
- Services should be independently deployable.
- Services are easy to replace.
- Services are organized around capabilities, e.g. user interface front-end, logistics, billing, etc.
- Services can be implemented using different programming languages, databases, hardware and software environment, depending on what fits best.

History

Document history		
V3.3.1	November 2019	Publication