ETSI GS NFV-EVE 004 V1.1.1 (2016-03)

Network Functions Virtualisation (NFV); Virtualisation Technologies; Report on the application of Different Virtualisation Technologies in the NFV Framework

Disclaimer

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

Reference
DGS/NFV-EVE004

Keywords
network, NFV, virtualisation

ETSI

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

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

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights

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

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

Foreword

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

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**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 reviews virtualisation technologies and studies their impact on the NFV architectural framework and specifications. It also provides an analysis of the pros and cons of these technologies.

2 References

2.1 Normative 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.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference/.

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 necessary for the application of the present document.

Not applicable.

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 003: "Network Functions Virtualisation (NFV); Terminology for main concepts in NFV".

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

[i.3] ETSI GS NFV-INF 001: "Network Functions Virtualisation (NFV); Infrastructure; Overview".

[i.4] ETSI GS NFV-INF 004: "Network Functions Virtualisation (NFV); Infrastructure; Hypervisor Domain".

[i.5] ETSI GS NFV-MAN 001: "Network Functions Virtualisation (NFV); Management and Orchestration".

[i.6] ETSI GS NFV-INF 007: "Network Functions Virtualisation (NFV); Infrastructure; Methodology to describe Interfaces and Abstractions".

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

[i.8] ETSI GS NFV-IFA 014: "Network Functions Virtualisation (NFV); Management & Orchestration; Network Service Descriptor template".

[i.9] ETSI GS NFV-IFA 011: "Network Functions Virtualisation (NFV); Management and Orchestration; VNF Packaging Specification".

[i.10] ETSI GS NFV-IFA 002: "Network Functions Virtualisation (NFV); Acceleration Technologies; VNF Interfaces Specification".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in ETSI GS NFV 003 [i.1] apply.

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in ETSI GS NFV 003 [i.1] and the following apply:

DevOps	Development and Operations
IPC	Interprocess Communication
LSM	Linux™ Security Module
OS	Operating System
OSS	Operations Support System
QoS	Quality of Service
TCP	Transmission Control Protocol
TPM	Trusted Platform Module
UTC	Coordinated Universal Time

4 Virtualisation technologies

4.1 Introduction

The ETSI NFV architectural framework as described in ETSI GS NFV 002 [i.2] identifies a virtualisation layer as a component of the NFV Infrastructure (NFVI). Typically, this type of functionality is provided for computing and storage resources in the form of hypervisors and VMs.

However, the NFV architectural framework does not restrict itself to using any specific virtualisation layer solution. Rather, it is expected that diverse virtualisation layers with standard features and open execution reference points towards Virtualised Network Functions (VNFs) and hardware can be used interchangeably.

4.2 Hypervisor-based solutions

4.2.1 Overview

A hypervisor is a software program that partitions the resources of a single hardware host and creates Virtual Machines (VM) isolated from each other. Each virtual machine appears to have the host's processor, memory and other resources, all to itself.

Each VM is assigned a virtualised CPU (vCPU), a virtualised NIC (vNIC) and a virtualised storage device (vStorage) created by the hypervisor. As pointed out in ETSI GS NFV-INF 004 [i.4], in practice, a vCPU may be a time sharing of a real CPU and/or in the case of multi-core CPUs, it may be an allocation of one or more cores to a VM. It is also possible that the hypervisor emulates a CPU instruction set that is different from the native CPU instruction set. However, emulation will significantly impact performance.

The hypervisor software runs either directly on top of the hardware (bare metal hypervisor, also known as Type I hypervisor) or on top of a hosting operating system (hosted hypervisor, also known as Type II hypervisor).

4.2.2 Application to NFV

The use of hypervisors is one of the present typical solutions for supporting the deployment of VNFs.

Although the NFV framework as defined in [i.2] is agnostic to the virtualisation technology, Management and Orchestration functions and interfaces as specified in ETSI GS NFV-MAN 001 [i.5] assume that virtualisation containers are virtual machines created by hypervisors. In particular, the operations and procedures at the Nf-Vi reference point were designed to act upon virtual machines rather than virtualisation containers in general. According to ETSI GS NFV-INF 004 [i.4], the VIM uploads the hypervisor on the compute nodes via the Nf-Vi/C reference point and requests the creation, modification and deletion of VMs via the Nf-Vi/H reference point. Requirements on hypervisors to make them suitable for use in an NFV environment are described in ETSI GS NFV-INF 004 [i.4].

The hypervisor-based approach does not place any constraint on the type of operating system that the VNF components (VNFCs) of a VNF are using. Both the operating system and the actual network application are part of the software image delivered by the VNF provider and loaded on the VM.

4.3 OS Containers

4.3.1 Overview

Container-based virtualisation, also called operating system (OS)-level virtualisation, is an approach to virtualisation which allows multiple isolated user space instances on top of a kernel space within the OS. The isolated guests are called containers.

Figure 1 provides a high-level comparison of the software architectures for hypervisor solutions where the VNFC software image loaded in the virtualisation container includes both a guest OS kernel and the actual application, and OS container solutions where the VNFC software image loaded in the virtualisation container only includes the actual network application.

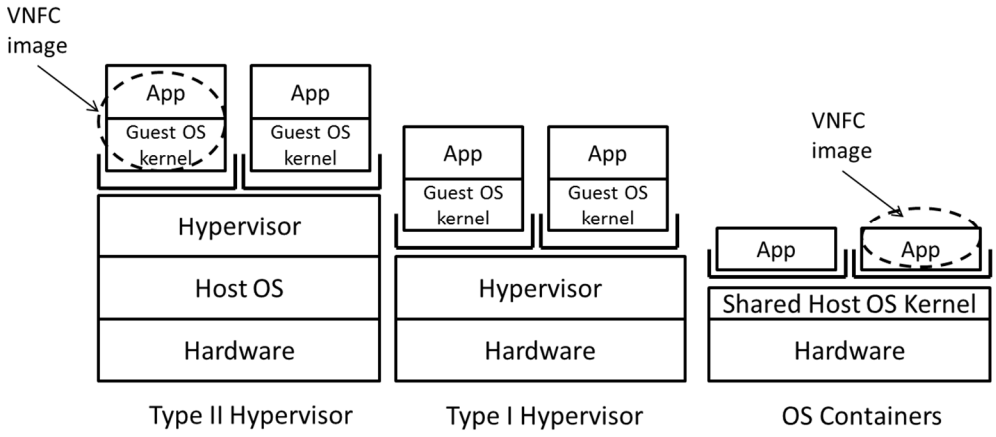

Figure 1: Hypervisor vs. OS Container solutions

The OS virtualisation technology allows partially shared execution context for different containers. Such a shared execution context is frequently referred to as a container pod. A pod might include shared file systems, shared network interfaces and other shared OS resources that are accessible from every container within that pod.

In addition to hypervisor-based execution environments that offer hardware abstraction and thread emulation services, the OS container execution environment provides kernel services as well. Kernel services include:

- Process control.

EXAMPLE 1: OS process creation; scheduling; wait and signal events; termination.

- Memory management.

EXAMPLE 2: Allocation and release of regular and large pages; handling memory-mapped objects and shared memory objects.

- File system management.
- File management.

EXAMPLE 3: Creation, removal, open, close, read and write file objects.

- Device management.

EXAMPLE 4: Request, release, configuration and access.

- Communication services.

EXAMPLE 5: Protocol stack services, channel establishment and release, PDU transmission and reception.

- System information maintenance.

EXAMPLE 6: Time and date, system and OS resource data, performance and fault indicators.

The OS container-to-VNFC logical interface is typically realized via:

- kernel system calls;
- signals to container processes;
- virtual file system mapped logical objects; and
- direct procedure calls into the container context.

OS virtualisation provides storage abstraction on file system level rather than on block device level. Each container has its separate file system view, where the guest file system is typically separated from the host file system. Containers within the same pod might share file systems where modifications made in one container are visible in the others.

Container file systems are realized either with standalone or with layered file systems. Standalone file systems are mapped into real file systems where all modifications made by the guest are stored in the backing real file system. Layered file systems take one or more base layers, and a writable overlay. A single layer is formed either from a real file system or from another layered file system structure. Layers are transparently overlaid and exposed as a single coherent file system. Typically, the lowermost layer contains an OS distribution with packages, libraries and run-times, while the overlay contains instance-specific customizations and modifications made by the container. While base layers are semi-permanently stored in image repositories, an overlay is disposable and its life time is coupled with the container life time.

4.3.2 Application to NFV

Because OS container solutions are based on a rather lightweight design where all VNFC instances share the same OS kernel, they are often considered as an alternative to hypervisor solutions when there is a need for deploying many instances of a network function (e.g. per-user instances for virtual residential gateway deployments).

4.4 Higher-level containers

4.4.1 Overview

Higher level containers are a level of virtualisation technologies more dealing with software code and its development, deployment, and runtime environment. So the level of abstraction is on the runtime environment, where source code written in a certain programming or scripting language is deployed onto the NFVI. A few characteristics of such systems are that:

1) source code is held and versioned in a code repository;
2) source code dependencies are explicitly defined and packaged into the deployed software;
3) code can be deployed into development, staging, or production environments without change;
4) configuration of the software is stored in the environment, typically through environment variables;

5) backing services such as data stores, message queues, and memory caches are accessed through a network and no distinction is made between local or third party services; and

6) processes are stateless and therefore enable easy scale-out.

Typically, those containers are used in continuous deployment models enabling fast DevOps models for telecommunication services.

4.4.2 Application to NFV

Higher-level containers are applied for VNFs that are delivered in source code and run on a particular execution engine on an NFVI. Compared to OS Containers and hypervisors, they are often considered as an alternative solution when there is a need for deploying a VNF in form of source code in so called DevOps environments.

4.5 Nesting of virtualisation technologies

4.5.1 Overview

Within the NFVI, the virtualisation layer may be composed of multiple nested sub-layers, each using a different virtualisation technology. In this case, only the top sub-layer and its technology are visible to the Virtualised Infrastructure Manager (VIM) and the partitions it creates provide the role of the virtualisation container as defined in ETSI GS NFV 003 [i.1]. Resource partitioning in the other sub-layers is typically provisioned by means outside the scope of NFV Management and Orchestration functions (e.g. by a dedicated non-NFV infrastructure OSS). An example shown in figure 2 is the case of a three levels virtualisation layer, where the top level uses a higher layer virtualisation technology, the layer below running OS container technology and the lowest layer uses the hypervisor technology. In this case, several higher-level containers, each hosting a VNFC instance, can run within each of the OS containers on one or several virtual machines created by the hypervisor.

Figure 2: Example sub-layering of nested virtualisation technologies

This is known as recursive virtualisation in ETSI GS NFV-INF 007 [i.6], which highlights that an operating virtual functional block can itself be a host functional block.

Besides the above considerations, virtualisation technologies can also be used inside a VNFC. However, this usage results from a decision of the VNF provider and is therefore outside the scope of the NFV framework and of the present document.

4.5.2 Application to NFV

4.5.2.1 General

The primary benefit of nesting is operational, since it enables more operational flexibility in integrating NFV as part of a larger deployment.

EXAMPLE: Since NFV's primary application is in the space of telecommunications and networking, a service provider might choose to run other non-telco services and applications on the same infrastructure, but managing their life cycle manually or using a different system than NFV Management and Orchestration functions.

Another example is when due to the internal organization of a service provider, the operations are divided between different organizational structures; with nesting, the organizational divide can also be implemented in the technical space.

In addition to these scenarios, there are couples of competing OS virtualisation solutions that might require slightly different OS execution environments, kernel versions, and hardware partitioning methods, e.g. VMs could support co-existence of different solutions within the same data centre.

Yet an additional use case is providing enforced security isolation between VNFCs running in OS containers via an additional virtualisation underlay using a hypervisor.

4.5.2.2 Virtualisation Container Interface Mapping

With nested virtualisation, each virtualisation sublayer provides container interfaces to the above layer while the NFVI container interface is composed from virtualisation container and virtual network container interfaces provided by the topmost virtualisation sublayer. Following multiplicity principles defined in ETSI GS NFV-INF 001 [i.3], lower layer container interfaces can be decomposed into multiple higher layer container interfaces, and a higher layer container interface can be composed from multiple lower layer containers.

Possible containerization interface mapping options include - but are not limited to - the three reference scenarios depicted in figure 3.

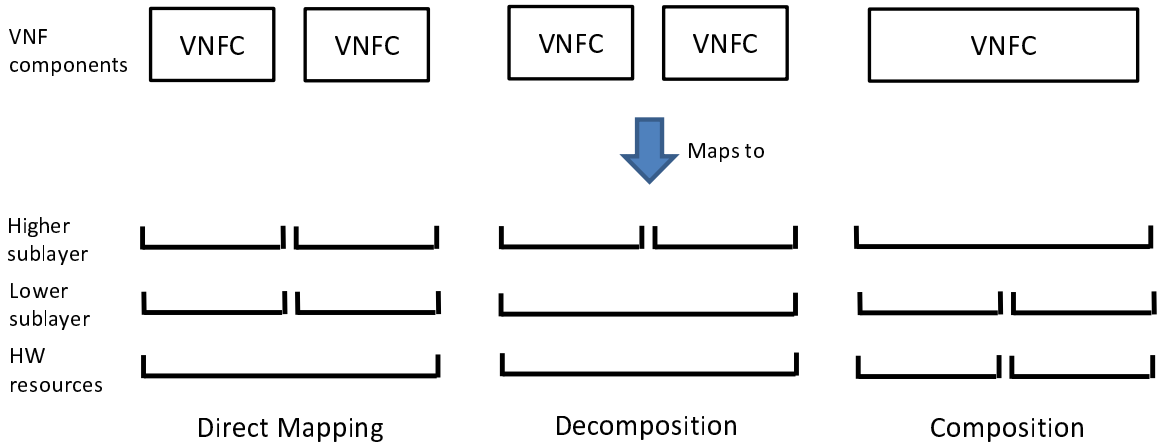

Figure 3: Virtualisation Container Interface Mapping

Direct container interface mapping provides one-to-one relationship between higher layer and lower layer virtualisation containers. As an example, OS virtualisation environment at the higher layer could be complemented with hardware virtualisation technologies at the lower layer for enforced VNFC isolation purposes. In that scenario, resource partitioning is performed by the lowermost layer while the higher layer offers adaptation and lightweight execution environment services. As an implication, multiple virtualisation sublayers are involved in VNFC execution environment preparation.

The container interface decomposition scenario splits a lower layer container interface into multiple virtualisation containers. For example, OS virtualisation can use resources from hardware virtualisation layer to create multiple OS containers hosting VNFC instances. In that scenario, hardware resource pool can be provisioned by the lower virtualisation sublayer while only the higher layers are involved in dynamic VNFC resource allocation.

The container interface composition scenario builds a higher layer container interface by aggregation of multiple lower layer container interfaces. In that way, the higher layer can expose aggregated resource capacity, or it can build a more fault-tolerant execution environment backed by resources from different domains.

4.5.2.3 Virtual Network Interface Mapping

With nested virtualisation, virtual network container interfaces are exposed by the topmost virtualisation sublayer towards VNFC instances, while all the virtualisation sub-layers provide networking functions for resource sharing and interconnection purposes. Networking functions provided by virtualisation sublayers include virtual network interface controllers, virtual switches and virtual routers as defined in ETSI GS NFV-INF 005 [i.7].

Figure 4 shows three examples of reference interconnection scenarios in nested virtualisation environment.

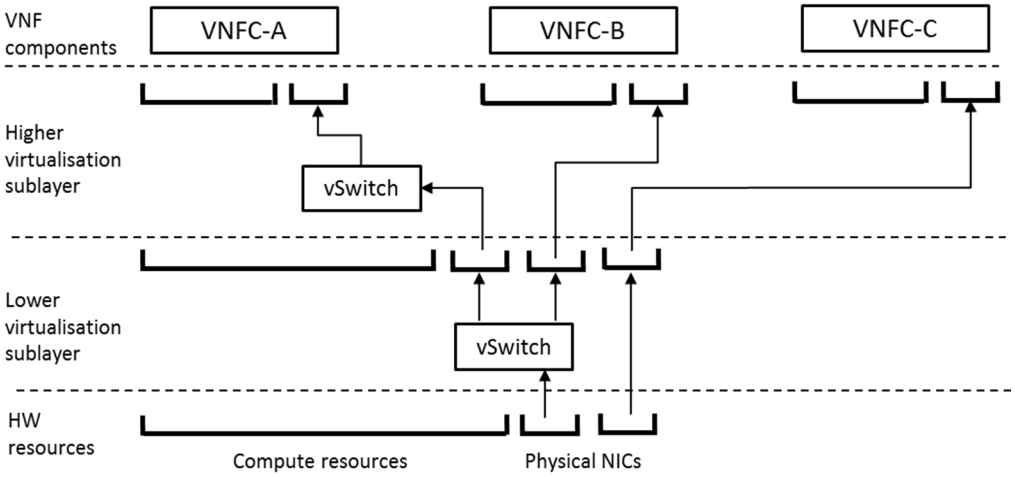

Figure 4: Virtual Network Interface Mapping

Virtual switching in the topmost virtualisation sublayer allows decomposition of a lower-layer virtual network interconnect interface into multiple network interfaces exposed towards VNFC instances, as demonstrated with case VNFC-A in figure 4. In that scenario, only the topmost sublayer is involved in dynamic VNFC interconnect configuration while the lower layers provide transparent access to the infrastructure network.

Virtual switching in the lowermost virtualisation sublayer can support lightweight higher virtualisation layers that do not include virtual switching, as demonstrated with case VNFC-B. As an implication, multiple layers are involved in VNFC interconnect configuration.

Direct virtual network interface mapping throughout nested layers enables memory mapped polled drivers to directly access physical network interfaces for acceleration purposes, as demonstrated with case VNFC-C. Similarly to the VNFC-B case, this scenario requires coordination between multiple layers on VNFC instantiation.

While virtual networks are defined as transparent L2- or L3-based networks in ETSI GS NFV-INF 004 [i.4], it is recognized that certain OS level virtualisation technologies apply application-layer interconnect fabrics between OS containers. For example, one frequently applied interconnect method is based on TCP port binding. Mapping of application-layer interconnect fabrics into the NFV framework is recommended for further studies.

4.6 Mixing of virtualisation technologies

4.6.1 Overview

In the case of mixing different virtualisation technologies, instances of the VNFCs of a certain VNF can run the different technologies, each benefiting of the particular characteristics of the virtualisation technology of choice. The different virtualisation technologies can be controlled by one or different VIMs. Figure 5 illustrates a deployment option where a single VIM controls three different virtualisation technologies.

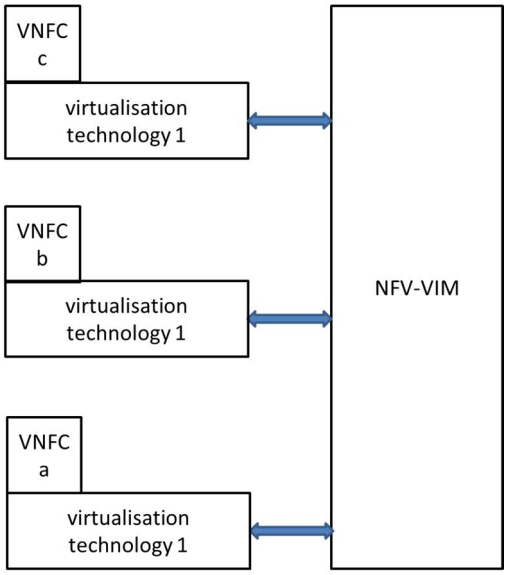

Figure 5: Example of Mixing of Virtualisation Technologies

4.6.2 Application to NFV

Mixing of virtualisation technologies is suitable to allow the provider of VNFs to have more flexibility in the use of the particular technology for a VNFC, and therefore a VNF can be composed of VNFCs running on different virtualisation technologies.

4.7 Mixing & Nesting of virtualisation technologies

As with nesting the NFVI virtualisation layer can be composed of multiple nested sub-layers, each using a different virtualisation technology. In addition, as illustrated in figure 6, the components of the VNF run on different nested layers, each benefiting of the particular characteristics of virtualisation technology of choice.

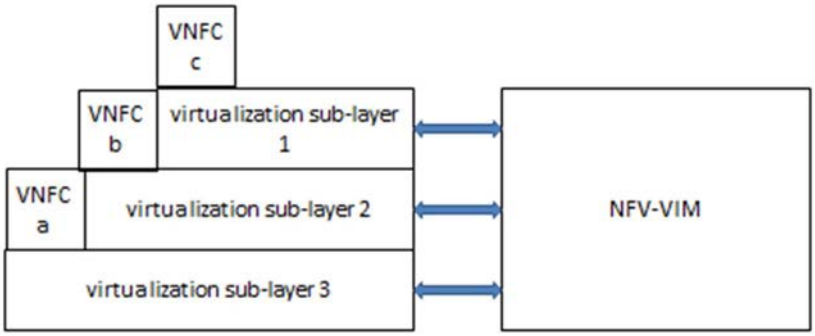

Figure 6: Example mixing of VNFCs on nested virtualisation technologies

5 Impact on the NFV architectural framework

5.1 Overview

For the virtualisation technologies other than those based on a hypervisor, the following clauses review their impact on the functional blocks of the NFV architectural framework.

5.2 OS Containers

5.2.1 Impact on the NFVI

The larger degree of sharing the underlying infrastructure allows an operator to run a high number of VNFC instances per compute/storage unit compared to the virtualisation technologies based on a hypervisor.

Since the kernel running the VNFC instances is shared, the configuration and functionality present in the kernel is the same for all the VNFC running on that kernel. This is in general not appropriate in cases where a VNFC has special needs in the kernel (for example a certain kernel module loaded or certain kernel settings done). However, the placement of VNFCs could take this into account, which makes that placement process even more complex.

5.2.2 Impact on Management and Orchestration

5.2.2.1 General

Which virtualisation technology the VNFCs of a VNF are designed for is not expected to have significant impact on Management and Orchestration functional blocks other than the VIM, on interfaces other than those used at the Nf-Vi reference point, and on descriptors other than the VNF Descriptor (VNFD).

5.2.2.2 Impact on ETSI NFV Phase 1 specifications

ETSI GS NFV-MAN 001 [i.5] assumes that the virtualisation technology in the NFVI relies on hypervisors and virtual machines. This is visible in the description of the Nf-Vi reference point (ETSI GS NFV-MAN 001 [i.5], clause 5.7.4), in the description of the operations applicable to this reference point (ETSI GS NFV-MAN 001 [i.5], clause 7.6), and in the description of some portions of the VNFD (ETSI GS NFV-MAN 001 [i.5], mainly clauses 6.3.1.2.1 and 6.3.1.2.5).

Supporting OS Containers alongside or in addition to hypervisors as a virtualisation technology would require generalizing the description of the Nf-Vi reference points and its operations to enable handling any type of virtualisation container.

Moreover, a leaf element equivalent to the `vm_image` element would have to be added to the list of `vnfd:vdu` base elements. This element would provide a reference to an application image and would have to be complemented by another element indicating the type (e.g. Linux™ or Windows™) of the OS and version of the OS or the OS kernel for which the image was generated. Additional information elements specific to the actual virtualisation technology (e.g. information related to a Docker™ Engine) could have to be added as well. For example, for the purpose of using Linux™-specific OS containers, the additional element might include required kernel features like namespaces, kernel capabilities or device files. An alternative solution would be to generalize the `vm_image` information element and the hypervisor related information elements.

NOTE 1: Docker™ is an open-source project that provides an additional layer of abstraction and automation of operating-system-level virtualisation on Linux™.

NOTE 2: Docker™ is a registered trademark of **Docker™, Inc. corporation** 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.

NOTE 3: Linux™ is a registered trademark of **Linus Torvalds**. 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.

NOTE 4: Windows™ is a registered trademark of **Microsoft™**. 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.

5.2.2.3 Impact on ETSI NFV IFA specifications

Interfaces addressed in the ETSI GS NFV IFA series are specified in a technology-agnostic manner. For example, information elements related to software images do not make any assumption about the type of image being managed. Therefore no impact on these specifications is foreseen.

Which virtualisation technology the VNFCs of a VNF are designed for is not expected to be visible at the level of the Network Service descriptors; therefore, no impact on ETSI GS NFV-IFA 014 [i.8] is foreseen.

It is expected that ETSI GS NFV-IFA 011 [i.9] will specify the VNFD template in way that truly accommodates both virtualisation technologies, which means that a VNFD complying with the template defined in this GS will be expected to contain sufficient information to enable the VNFM to provide the VIM with sufficient inputs to select NFVI nodes that support the execution environment (OS and other elements) assumed by the VNFCs, when creating virtualised compute resources.

5.2.3 Impact on VNFs

A fundamental difference with hypervisor-based solutions is that the VNFC images contained in a VNF package do not contain any kernel space code. The applications running in the containers have to use the same kernel as the host OS (i.e. the concept of guest OS does not exist).

5.2.4 Impact on Security

Application OS virtualisation into NFV context has several implications on the threat surface:

- OS virtualisation interface is a more sophisticated interface than hardware emulation interfaces with hypervisor-based virtualisation. Due to its wider attack surface, OS virtualisation is generally considered to be more security vulnerable than hardware virtualisation.
- Functional isolation of virtualised kernel services is challenged when co-located VNFC instances require different configurations, like different network stack or time service settings to avoid configuration interferences.
- Performance isolation and assurance could be challenged especially whenever virtualised kernel services are placed under resource pressure, either by an external attacker or by a co-located VNFC instance. Such attacks might lead to memory and/or CPU overconsumption and might exhaust logical OS resources negatively affecting the performance of other VNFC instances.
- Confidentiality protection could be challenged whenever multiple VNFC instances are collocated with shared kernel services, e.g. if memory pages are dynamically reassigned between VNFC instances.

There are numerous container-based technologies on various platforms. As a commonality, the secure separation between VNFC instances is limited by the security mechanisms implemented in the kernel.

This impact assessment examines the current state-of-the-art for Linux™ Containers at time of writing (December 2015). Linux™ containers are realized by a variety of different kernel technologies, including namespaces, control groups, system call capability restrictions and pluggable Linux™ Security Modules (LSM).

Linux™ namespace technology provides segregated namespaces for various system entities, including processes, networking, users, IPC, UTC and mount namespaces. Each namespace provide a virtualised view of certain OS constructs and configurations for a group of OS processes that are assigned to that particular namespace, therefore allow isolation of applications running in different containers. This technology is complemented by "control groups" for resource restriction purposes. As with namespaces, control groups divide processes into groups, and can enforce resource quotas by accounting resource usage on control group level. Additionally, device control groups can enforce fine-grained access control on a per-device level. The control groups available on a Linux™ system with a 3,18 kernel are cpuset, cpuacct, memory, devices, freezer, net_cls, net_prio, blkio, perf_event and hugetlb.

Linux™ capabilities divide privileges originally associated with superuser into distinct groups, which can be independently enabled or disabled on a per-process level. Within OS virtualisation context, capabilities can assign extra privileges to container for configuration of namespaced constructs while can prohibit containers from other system calls that could affect the overall system. For example, kernel module insertion, device object creation, Ethernet and time configurations and raw block device access capabilities are typically disabled in common OS container implementations. Linux™ capability restrictions can be enhanced with pluggable LSM modules, like SELinux to implement fine-grained security policies.

Recent security improvements are focusing on minimal host OS distributions for reducing the attack surface while executing host management tools in isolated management containers.

In general, Containers may be considered appropriate virtualisation options for functions which are not subject to the particular areas of concern outlined below. They offer various benefits - as outlined in the other clauses of the present document - and the security capabilities are suited to many other functions of VNFs.

There are two major areas of security concern when using Linux™ Containers: host kernel access and hardware pass-through, and these are related to NFV usages:

- Host kernel access: while most application classes can be executed exclusively in user-space, certain VNF types might require kernel-mode execution privileges for performance and for deterministic execution reasons, e.g. direct NIC access. Once a Container gains access to the host kernel, there is no way to stop it performing any operations on the host itself or other entities (VMs, Containers) on the same host, and compromise of a Container with access to such capabilities could lead to the compromise of the entire NFVI host.

- Hardware pass-through: when resources are passed directly to containers, mechanisms to ensure "friendly neighbour" isolation, stopping one application from starving other applications of resources, are currently immature.

In both cases, one approach would be to audit applications provisioning time in an attempt to avoid occurrences of such behaviour, but the code which would be performing such attacks, intentionally or unintentionally, is likely to be extremely difficult to spot, and every update would require such auditing. This approach is therefore considered unfeasible in most deployments.

There are also use cases where resources may be shared between Containers - shared memory or shared devices, for instance. There is currently no mature security model to cope with such requirements.

In case of hardware based security like Trusted Platform Module (TPM), the kernel needs to run the access policy to the hardware for those features.

5.2.5 Impact on NFV acceleration

ETSI GS NFV-IFA 002 [i.10] specifies a set of abstract interfaces enabling a VNF to leverage acceleration capabilities in the NFVI in an implementation-independent manner. This acceleration model assumes that a backend software module residing in the hypervisor implements an adaptation layer to the hardware or to hypervisor domain resources. Where this module would reside in the absence of a hypervisor requires further studies.

5.3 Higher-level containers

5.3.1 Impact on the NFVI

The NFVI needs to enable nested virtualisation as described in clause 4.6.

Platform services provided by the NFVI need to be exposed to the VNF as backing services and the binding of those backing services to a particular VNF and its environment needs to be supported.

Typically, there is a concept of port binding to a VNF. This port binding might have an impact to the networking of the NFVI.

5.3.2 Impact on Management and Orchestration

One impact is the source code repository, and whether this is held within or outside NFV Management & Orchestration functions. This has also an impact to the overall architecture. An alternative to a source code repository could be to generalize the virtualisation container image repository.

The VNFD will need to have information about the runtime environment and the required backing services for the implementation of the VNF. The software configurations would need to be under the control of a VNFM.

Management of dependencies on backing services would need to be performed by the VNFM.

5.3.3 Impact on VNFs

VNFs main impacts are the particular way they have to be implemented such that they run in the particular runtime environment.

Also typically the VNFC instances need to be able to start-up quickly for high-speed scale-out and error recovery. They also need to be terminated instantaneously, which enable quick configuration or service changes.

5.3.4 Impact on Security

Since security parameters are typically part of the particular instance of a deployment, they are stored in the virtualisation container, which needs the virtualisation container to be secure enough for the service requirements.

Otherwise there does not seem to be any other security impact compared to what has been identified in the hypervisor based security.

5.3.5 Impact on NFV acceleration

Clause 5.2.5 applies.

5.4 Nested Virtualisation

5.4.1 Impact on the NFVI

Within the NFVI, the virtualisation layer may be composed of multiple nested sub-layers, each using a different virtualisation technology.

5.4.2 Impact on Management and Orchestration

The NFVI is managed by one or more VIMs that have control on the lifecycle of the virtualisation containers created by the top-level sub-layer. However, the underlying virtualisation sub-layers are typically not visible from the VIM. So coordination might be more static or more difficult. For example, a reservation of underlying resources might need to be pre-provisioned. However, more advanced scenarios where the VIM would manage the lifecycle of the underlying partitions (e.g. VMs hosting OS containers) as well can be envisioned. This would require the VIM to embed appropriate logic to decide when to create such underlying partitions (e.g. create a new VM in case the existing ones are already loaded with too many OS containers, or in case a new OS container is to be strongly isolated from the other ones).

5.4.3 Impact on VNFs

None.

5.4.4 Impact on Security

Since the security impact depends on the particular virtualisation technology, the impact on security is the same as discussed in clauses 5.2.4, and 5.3.4.

5.4.5 Impact on NFV acceleration

This topic requires further studies.

5.5 Mixed Virtualisation

5.5.1 Impact on the NFVI

Virtualisation technologies may use different network interface types. Therefore, the NFVI network domain needs to be able to handle networking cross the different types of virtualisation technologies.

Similarly, different virtualisation technologies may use different types of storage technologies.

5.5.2 Impact on Management and Orchestration

The impact of mixing is that there might be several instances of a VIM for the different mixed virtualisation technologies, which might require some level of coordination.

Also due to interacting VNFC instances running on different virtualisation technologies, the resource allocation and the provisioning process might get more complex.

5.5.3 Impact on VNFs

The impact on the VNF is that the provider of a VNF and/or VNFC has more choice in implementing the piece of code on the virtualisation technology best suited for the task and logic it is going to perform.

5.5.4 Impact on Security

Since the security impact depends on the particular virtualisation technology, the impact on security is the same as discussed in clauses 5.2.4, and 5.3.4.

5.5.5 Impact on NFV acceleration

Clause 5.2.5 applies.

5.6 Nested and Mixed Virtualisation

5.6.1 Impact on the NFVI

Clauses 5.4.1 and 5.5.1 apply.

5.6.2 Impact on Management and Orchestration

Clauses 5.4.2 and 5.5.2 apply.

5.6.3 Impact on VNFs

Clauses 5.4.3 and 5.5.3 apply.

5.6.4 Impact on Security

Clauses 5.4.4 and 5.5.4 apply.

5.6.5 Impact on NFV acceleration

Clauses 5.2.5 applies.

6 Pros and Cons analysis

6.1 Features

While hypervisor-based virtualisation offers high flexibility in migration of VNFC instances from one compute host to another, OS virtualisation technologies exhibit a number of limitations in migration capabilities. This is inherently related to the distribution of VNFC execution states between user and kernel spaces, and difficulties in transparent migration of container-specific kernel-side structures, e.g. OS process descriptors, file descriptors or network sockets.

6.2 Performance considerations

One of the key objectives of virtualisation (container or hypervisor based) is to provide a level of abstraction. The abstracted resources can be hardware platform resources (CPU, memory, interfaces, etc.) or software resource like kernel, operating system services, etc. In the context of performance, it is critical to understand that the increased degree of virtualisation leads to increased abstraction. Abstraction can enable a lot of use cases like heterogeneous VNFs consolidation, sharing of platform resources, mobility of VNFs, etc. However, abstractions can introduce overheads for certain performance indicators.

Containers are a light weight operating system level virtualisation technology, which share the host kernel. Containers do not abstract hardware resources, unlike hypervisor-based virtualisation. Thus, for certain scenarios, e.g. platform micro and nano benchmarks, containers may perform better compared to hypervisor-based virtual machines. In other areas, containers may not improve performance or not support use cases, especially with regard to different kernel service adaptations being required by different VNFC instances.

EXAMPLE: Domain-specific communication protocols might require non-standard kernel modules for optimal performance.

To summarize, it is critical for NFV system designers to understand the abstraction and performance requirements of the VNF workloads before choosing the virtualisation technology fit for these workloads.

6.3 Security

OS containers are considered to provide lower security level than hypervisor-based solutions due to their wider attack surface, that is, the compromise of the OS may undermine the whole system.

While emerging technologies are providing fine-grained access control with OS containers for security enhancement purposes, the cost of security hardening comes in capability and feature reduction. Generally, hypervisor-based solutions provide more flexibility and higher backward compatibility to VNFC instances with application-defined kernels than security-hardened OS containers.

OS containers are not considered appropriate for deployments where VNFC instances require hardware pass-through capabilities or kernel-mode execution privileges, as compromise of such a container could lead to compromise of an NFVI host.

6.4 Resources management

6.4.1 Resources utilization

Compared with hypervisors-based solutions, OS Containers solutions build on a more lightweight design enabling overall resource optimization, as there is a single shared kernel. Additional memory and storage footprint reduction can be achieved by sharing the same OS distribution between container instances, e.g. using common base file system layers. For hypervisor solutions, every VM will have a dedicated OS. There is an inherent trade-off between footprint size and code base independence and code base sharing is an enabler for high container density.

Overall, compared to hypervisor-based solutions, this approach can be considered as means to run more VNFC instances on the same physical host, thereby reducing the infrastructure costs.

6.4.2 Resources isolation

For resource isolation, the challenge for any virtualisation solution is to account for and guarantee fair resource usage. Enforcement is intimately linked with scheduling mechanisms and policies. Policies may range from resource reservation to best effort, and their combination (fair resource sharing between different classes of QoS). Both the container-based and the hypervisor-based approaches feature scheduler architectural components in the different levels (VM/containers and hypervisor/OS) which allow enforcing a wide range of such policies. Both approaches are thus comparable at coarse-grained level.

6.4.3 Instantiation Time

Compared to hypervisor-based solutions, OS virtualisation offers a magnitude lower VNFC instantiation time due to avoidance of hardware and OS initialization procedures on start-up. Usage of a common base image layer, especially a common OS distribution, could provide additional savings by shortening image preparation time.

For short-living VNFC instances, e.g. with up to a few minutes life time, resource utilization efficiency is highly influenced by the instantiation time-to-productive time ratio. For such VNFC types, OS virtualisation provides significantly improved resource utilization efficiency compared to hypervisor-solutions.

6.5 Lifecycle Management

Most OS distributions have dependencies on particular kernel versions, that contain distribution-specific patches and configurations, and combinations of OS distribution with other kernel versions are generally not certified for production use. For VNF implementations having dependencies on particular OS distributions, this implies a dependency on the kernel version too.

When kernel services are provided by the infrastructure domain with OS virtualisation, VNF life cycle management could be tightly coupled with the life cycle management of the infrastructure domain as long as the VNF is dependent on a particular OS distribution. In such scenario, hypervisor-based solutions provide greater flexibility by decoupling the life cycle management of the tenant and the infrastructure domains.

In case of higher-layer containers, which are suitable when network services are developed and operated in a DevOps model, software is often deployed in the form of source code. So the life-cycle is very near to the current software development models and current practices with a software repository and the automated/semi-automated deployment of the software into a test and production environment. This is very beneficial for VNF implementation where the network service/network function changes quite often as a result of new or changing requirements. In general one can say they are beneficial everywhere where flexibility and the ability to perform frequent changes are more important than performance.

Many types of higher-level containers systems are prescribing an implementation model that makes them more suited for network functions that use a resource model that is based on automated scale-out and automated higher availability.

6.6 Open Source Support

Open Source implementation for hypervisors, OS containers and higher-level containers are available in the industry.

Many open source tools and systems exist for higher-level containers including management of higher-layer containers. They are typically linked with a programming language and execution environment for such a programming language.

Annex A (informative): Authors & contributors

The following people have contributed to this specification*:*

Rapporteur
Bruno Chatras, Orange

Other contributors
Marcus Brunner, Swisscom
Mike Bursell, Intel
Gergely Matefi, Ericsson
Valerie Young, Intel

Annex B (informative): Change History

Date	Version	Information about changes
November 2014	0.0.1	Scope and Table of Contents
January 2015	0.0.2	Output of NFV EVE#2 call, incorporates NFVEVE(14)000008R1
February 2015	0.0.3	Output of NFV EVE#3 call, incorporates NFVEVE(15)000021 & 22
February 2015	0.0.4	Output of NFV EVE#4 meeting held during NFV#9, incorporates NFVEVE(15)000038R1,58R1, 59R1 and 60R2
February 2015	0.0.4	Output of NFV EVE#4 meeting held during NFV#9, incorporates NFVEVE(15)000038R1,58R1, 59R1 and 60R2
May 2015	0.0.5	Output of NFV EVE meeting held during NFV#10, incorporates NFVEVE(15)000133r1, 135r1,136, 193r1
July 2015	0.1.0	Output of NFV EVE#9 virtual meeting, incorporates NFVEVE(15)000194R3, 219R2, 248R1, 249R1, 279
July 2015	0.2.0	Output of the NFV#11 meeting, incorporates NFVEVE(15)000132r2, 291r1
September 2015	0.3.0	Output of the NFVEVE#19 meeting, incorporates NFVEVE(15)000378r1, 379r1 & 382r1
October 2015	0.4.0	Output of the NFV#12 meeting, incorporates NFVEVE(15)000290r1, 401,402,426,436r2, 439, 443r1, 444r1, 445, 446, 450r1 and 451r2
December 2015	0.5.0	Output of the NFV EVE#23 call, incorporating NFVEVE(15)000455
January 2016	0.5.1	Output of the NFV EVE#25 call, incorporating NFVEVE(16)002R1 and NFVEVE(16)003R1
January 2016	0.6.0	Final draft incorporating NFVEVE(15)000325R2 and NFVEVE(16)000011R1

History

Document history		
V1.1.1	March 2016	Publication