ETSI GR NFV-EVE 022 V5.1.1 (2022-12)

Network Functions Virtualisation (NFV) Release 5; Architectural Framework; Report on VNF configuration

Disclaimer

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

Reference
DGR/NFV-EVE022

Keywords
configuration, NFV

ETSI

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

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

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

Important notice

The present document can be downloaded from:
http://www.etsi.org/standards-search

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the prevailing version of an ETSI deliverable is the one made publicly available in PDF format at www.etsi.org/deliver.

Users of the present document should be aware that the document may be subject to revision or change of status.
Information on the current status of this and other ETSI documents is available at
https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx

If you find errors in the present document, please send your comment to one of the following services:
https://portal.etsi.org/People/CommiteeSupportStaff.aspx

If you find a security vulnerability in the present document, please report it through our
Coordinated Vulnerability Disclosure Program:
https://www.etsi.org/standards/coordinated-vulnerability-disclosure

Notice of disclaimer & limitation of liability

The information provided in the present deliverable is directed solely to professionals who have the appropriate degree of experience to understand and interpret its content in accordance with generally accepted engineering or other professional standard and applicable regulations.
No recommendation as to products and services or vendors is made or should be implied.
No representation or warranty is made that this deliverable is technically accurate or sufficient or conforms to any law and/or governmental rule and/or regulation and further, no representation or warranty is made of merchantability or fitness for any particular purpose or against infringement of intellectual property rights.
In no event shall ETSI be held liable for loss of profits or any other incidental or consequential damages.

Any software contained in this deliverable is provided "AS IS" with no warranties, express or implied, including but not limited to, the warranties of merchantability, fitness for a particular purpose and non-infringement of intellectual property rights and ETSI shall not be held liable in any event for any damages whatsoever (including, without limitation, damages for loss of profits, business interruption, loss of information, or any other pecuniary loss) arising out of or related to the use of or inability to use the software.

Copyright Notification

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.
The content of the PDF version shall not be modified without the written authorization of ETSI.
The copyright and the foregoing restriction extend to reproduction in all media.

© ETSI 2022.
All rights reserved.

Contents

Intellectual Property Rights

Essential patents

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

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

Trademarks

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

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

Foreword

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

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 provides guidelines on the use of the VNF configuration options enabled by the NFV architectural framework. It identifies gaps in NFV specifications preventing interoperability between VNFs and independently-developed VNF configuration management functions, and/or preventing easy integration of VNF configuration management in VNF lifecycle management processes. The present document provides recommendations on normative work to be carried out to fill these gaps.

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

[i.2] Recommendation ITU-T M.3400 (02/2000):"Telecommunications management network: TMN management functions".

[i.3] IETF RFC 6241 (06/2011): "Network Configuration Protocol (NETCONF)".

NOTE: Available at https://www.rfc-editor.org/rfc/rfc6241.

[i.4] gNMI - gRPC Network Management Interface.

NOTE: Available at: https://github.com/openconfig/gnmi.

[i.5] ETSI GS NFV-SOL 001: "Network Functions Virtualisation (NFV) Release 4; Protocols and Data Models; NFV descriptors based on TOSCA specification".

[i.6] ETSI GS NFV-SOL 002: "Network Functions Virtualisation (NFV) Release 4; Protocols and Data Models; RESTful protocols specification for the Ve-Vnfm Reference Point".

[i.7] ETSI GS NFV-SOL 003: "Network Functions Virtualisation (NFV) Release 4; Protocols and Data Models; RESTful protocols specification for the Or-Vnfm Reference Point".

[i.8] ETSI GS NFV-SOL 004: "Network Functions Virtualisation (NFV) Release 4; Protocols and Data Models; VNF Package and PNFD Archive specification".

[i.9] ETSI GS NFV-SOL 005: "Network Functions Virtualisation (NFV) Release 4; Protocols and Data Models; RESTful protocols specification for the Os-Ma-nfvo Reference Point".

[i.10] ETSI GS NFV-SOL 006: "Network Functions Virtualisation (NFV) Release 4; Protocols and Data Models; NFV descriptors based on YANG specification".

[i.11] ETSI GS NFV-SOL 014: "Network Functions Virtualisation (NFV) Release 4; Protocols and Data Models; YAML data model specification for descriptor-based virtualised resource management".

[i.12] ETSI GS NFV-SOL 018: "Network Functions Virtualisation (NFV) Release 4; Protocols and Data Models; Profiling specification of protocol and data model solutions for OS Container management and orchestration".

[i.13] ETSI GS NFV-IFA 006: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Vi-Vnfm reference point - Interface and Information Model Specification".

[i.14] ETSI GS NFV-IFA 007: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Or-Vnfm reference point - Interface and Information Model Specification".

[i.15] ETSI GS NFV-IFA 008: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Ve-Vnfm reference point - Interface and Information Model Specification".

[i.16] ETSI GS NFV-IFA 009: "Network Functions Virtualisation (NFV); Management and Orchestration; Report on Architectural Options".

[i.17] ETSI GS NFV-IFA 011: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; VNF Descriptor and Packaging Specification".

[i.18] ETSI GS NFV-IFA 013: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Os-Ma-nfvo reference point - Interface and Information Model Specification".

[i.19] ETSI GR NFV-IFA 029: "Network Functions Virtualisation (NFV) Release 3; Architecture; Report on the Enhancements of the NFV architecture towards "Cloud-native" and "PaaS"".

[i.20] ETSI GR NFV-IFA 039: "Network Functions Virtualisation (NFV) Release 5; Architectural Framework; Report on Service Based Architecture (SBA) design".

[i.21] ETSI GS NFV-IFA 040: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Requirements for service interfaces and object model for OS container management and orchestration specification".

[i.22] ETSI GR NFV-EVE 019: "Network Functions Virtualisation (NFV) Release 4; Architectural Framework; Report on VNF generic OAM functions".

[i.23] ETSI Registry for non-MANO artifact sets. .

NOTE: Available at https://nfvwiki.etsi.org/index.php?title=Non_MANO_artifact_sets.

[i.24] IETF RFC 7396: JSON Merge Patch.

[i.25] IETF RFC 8040: "RESTCONF Protocol".

NOTE: Available at https://www.rfc-editor.org/rfc/rfc8040.

[i.26] IETF RFC 8526: "NETCONF Extensions to Support the Network Management Datastore Architecture".

NOTE: Available at https://www.rfc-editor.org/rfc/rfc8526.

[i.27] Kubernetes® Documentation: "API Reference".

NOTE: Available at https://kubernetes.io/docs/reference/.

3 Definition of terms, symbols and abbreviations

3.1 Terms

Void.

3.2 Symbols

Void.

3.3 Abbreviations

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

DSL	Domain Specific Language
gNMI	gRPC Network Management Interface
NETCONF	Network Configuration
OSI	Open Systems Interconnection
SSH	Secure SHell
TMN	Telecommunications Management Network

4 VNF configuration

4.1 General

Configuration management is one of the five functional areas of the Telecommunications Management Network (TMN) model. Recommendation ITU-T M.3400 [i.2] defines configuration management as a management functional area that provides functions to exercise control over, identify, collect data from and provide data to network elements. The present document focuses on configuration management for providing configuration data to VNFs and collecting such data from these VNFs.

4.2 VNF configuration items

A VNF instance is a set of software instances and virtualised resources needed for these software instances to execute. The scope of VNF configuration includes the configuration of both the software and the resources. Resource configuration is part of the VNF lifecycle management and relies on configuration data available in VNF descriptors and configuration data generated at runtime. The present document focuses on the configuration of the VNF software, also known as VNF application configuration.

NOTE 1: In the context of the present document the term "application configuration" should be understood in a wider sense than the configuration of the application layer of the OSI model.

EXAMPLE: The configuration of forwarding rules in a VNF that provides the functionality of a firewall falls in the "application configuration" category.

The data items configured as part of the VNF software configuration process can be classified in two broad categories, as illustrated in figure 4.2-1: virtualisation-dependent items and virtualisation-independent items:

- Virtualisation-dependent items are those items whose value is dependent on decisions made by the NFV infrastructure and/or the NFV management and orchestration functions. An example of such items is the VNF's VNFM IP address. Another example is the IP address assigned to a connection point of a VNFC instance to be configured on another VNFC of the same VNF that needs to communicate with the former (e.g. a load balancer VNFC).
- Virtualisation-independent items are all other items. Their values are typically determined in the OSS/BSS, as for a PNF. An example of such items is a forwarding rule for a VNF that provides the functionality of a firewall.

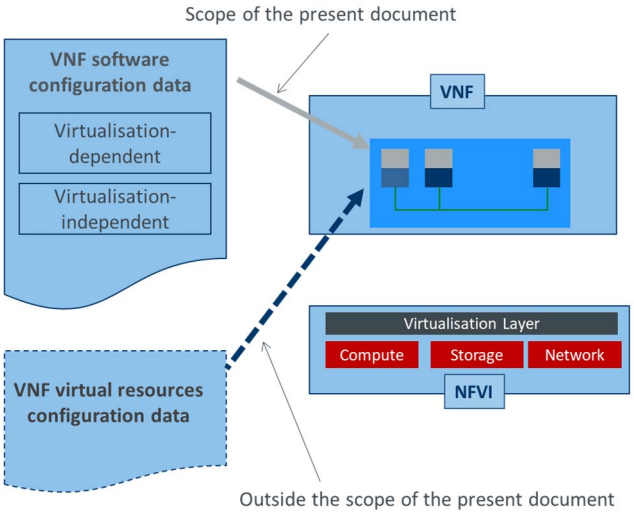

Figure 4.2-1: Overview of VNF configuration data

The data items configured as part of the VNF software configuration process can be further classified according to when to configure them. Some items are intended to be configured during the instantiation process, typically when the VNF instance cannot be fully up and running without the corresponding values being configured. Others can be configured at any time once the VNF instance is created and operational. The configuration of the first category of items is often known as "Day-0" configuration, while the configuration of items in the second category is often designated as "Day-1" (configured right after instantiation) or "Day-2" configuration (any time after "Day-1" configuration).

NOTE 2: This classification has to be understood from a VNFC instance viewpoint, e.g. scaling-out a VNF instance can require Day-0 configuration of the new VNFC instances.

4.3 VNF configuration methods

4.3.1 Overview

Clause 4.3 of the present document provides a description of the configuration methods supported by the NFV architectural framework at the time of publication. These methods are not mutually exclusive.

4.3.2 Method #A

4.3.2.1 General description

With this configuration method, the VNF receives configuration data directly from the OSS/BSS or from its EM. In the latter case the data can be originated in the EM or received by the EM from the OSS/BSS.

This method is only applicable to Day-1 and Day-2 configuration as it assumes that the VNF instance can communicate with the OSS/BSS or an EM. At least one VNFC instance is expected to act a configuration management agent in the VNF.

The configuration procedure can follow a push (configuration data pushed to the VNF by the OSS/BSS or EM) or pull model (configuration data retrieved by the VNF instance from the OSS/BSS or EM).

The specification of the interfaces between the VNF and its EM, as well as the specification of the interfaces between the OSS/BSS and EMs are outside the scope of NFV standardization.

NOTE 1: NETFCONF [i.3], gNMI [i.4] are examples of protocol solutions used in the industry on these interfaces.

VNF-specific data models for configuration (e.g. YANG data models) can be provided as non-MANO artefacts in the VNF package for use by the OSS and/or EM.

NOTE 2: onap_yang_modules [i.23] is an example of a set of non-MANO-artefacts registered for that purpose.

When used for configuring virtualisation-independent data, this method is like conventional methods applicable to a PNF as there is no involvement of NFV-MANO.

Use of this method for configuring virtualisation-dependent configuration data assumes that the OSS and/or EM retrieves the virtualisation-dependent values from the NFVO or the VNFM, prior to configuring the VNF. The OSS and/or EM will then map the retrieved information to the appropriate configuration items for the VNF.

Figure 4.3.2.1-1 illustrates this configuration method.

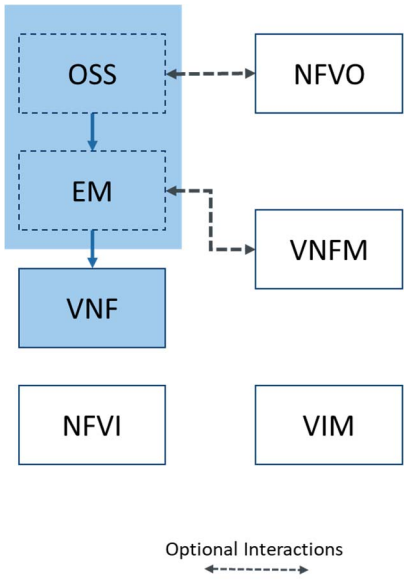

Figure 4.3.2.1-1: VNF configuration method #A

4.3.2.2 Detailed description

4.3.2.2.1 Push mode

4.3.2.2.1.1 OSS-initiated

In this mode the OSS can push configuration data to a VNF instance in direct mode or indirect mode. In direct mode the OSS connects to the VNF instance to push the configuration data. In indirect mode the OSS connects to the EM which in turn can push the configuration data to the VNF instance (see clause 4.3.2.2.1.2) or can wait for the VNF instance to pull these data from the EM.

The OSS can decide to push configuration data at any time once the VNF instance has been created or when specific lifecycle management events occur. To determine when to push configuration data, the OSS subscribes to receive notifications from the NFVO about the creation or modification of a VNF instance. ETSI GS NFV-IFA 013 [i.18] specifies the operations of the NS Lifecycle Management (LCM) interface for subscribing to notifications and for sending notifications.

If the configuration data to be pushed to the VNF instance includes virtualisation-dependent configuration items, or if a direct mode is used, a preamble procedure is used to retrieve VNF instance information from the NFVO, using the QueryNs operation of the NS LCM interface defined in ETSI GS NFV-IFA 013 [i.18]. If the configuration data to be pushed to the VNF instance includes virtualisation-dependent configuration items, the retrieved VNF instance information is used to obtain the values assigned to these items. If the direct mode is used, the retrieved VNF instance information is used to obtain the address(es) assigned to the VNF external connection point(s) where to push the configuration data.

For the case of virtualisation-independent configuration items an approach similar to the case of a PNF, with no NFV-MANO involvement, can be used.

Figure 4.3.2.2.1.1-1 provides an overview of the OSS-initiated push mode procedure in indirect mode with a preamble to retrieve VNF instance information from the NFVO. The following steps are identified:

1) The OSS sends a QueryNs request to the NFVO, as defined in clause 7.3.6 of ETSI GS NFV-IFA 013 [i.18], with a filter set on the VNF instance identifier and an attribute selector set to VnfInfo.

2) The NFVO provides the contents of the VnfInfo information element for the VNF instance, as specified in clause 8.3.3 of ETSI GS NFV-IFA 013 [i.18], in a QueryNs response.

3) The OSS pushes the VNF configuration data to the EM.

4) The EM forwards the received information to the VNF.

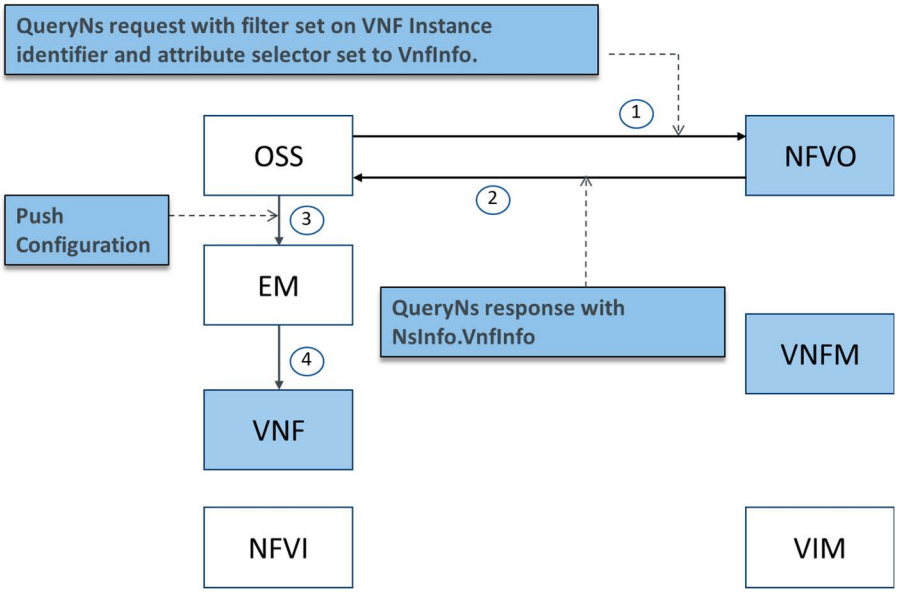

Figure 4.3.2.2.1.1-1: OSS-initiated push mode procedure

ETSI GS NFV-SOL 005 [i.9] specifies the Restful API for implementing the NS LCM interface including the QueryNs operation used in the preamble and the operations for managing notifications. The protocol used by the OSS for communicating with the VNF/EM and the protocol used between the EM and the VNF are outside the scope of the NFV specifications referenced in clause 2 of the present document.

4.3.2.2.1.2 EM-initiated

This mode can be used as part of the OSS-initiated procedure in push mode or independently (e.g. to push EM-generated configuration data or when the OSS is not responsible for VNF configuration management).

In the second case, the EM can push configuration data at any time once the VNF instance has been created or when specific lifecycle management events occur. To determine when to push configuration data, the EM subscribes to receive notifications from the VNFM about the creation or modification of a VNF instance. ETSI GS NFV-IFA 008 [i.15] specifies the operations of the VNF LCM interface for subscribing to notifications and for sending notifications.

If the configuration data to be pushed to the VNF instance includes virtualisation-dependent configuration items, a preamble procedure is used to retrieve VNF instance information from the VNFM, using the QueryVnf operation of the VNF LCM interface defined in ETSI GS NFV-IFA 008 [i.15], to obtain the values assigned to these items.

Figure 4.3.2.2.1.2-1 provides an overview of the EM-initiated push mode procedure with a preamble. The following steps are identified:

1) The EM sends a QueryVnf request to the VNFM, as defined in clause 7.2.9 of ETSI GS NFV-IFA 008 [i.15], with a filter set on the VNF instance identifier.

2) The VNFM provides the contents of the VnfInfo information element, as specified in clause 9.4.2 of ETSI GS NFV-IFA 008 [i.15], in a QueryVnf response.

3) The EM pushes the VNF configuration data to the VNF.

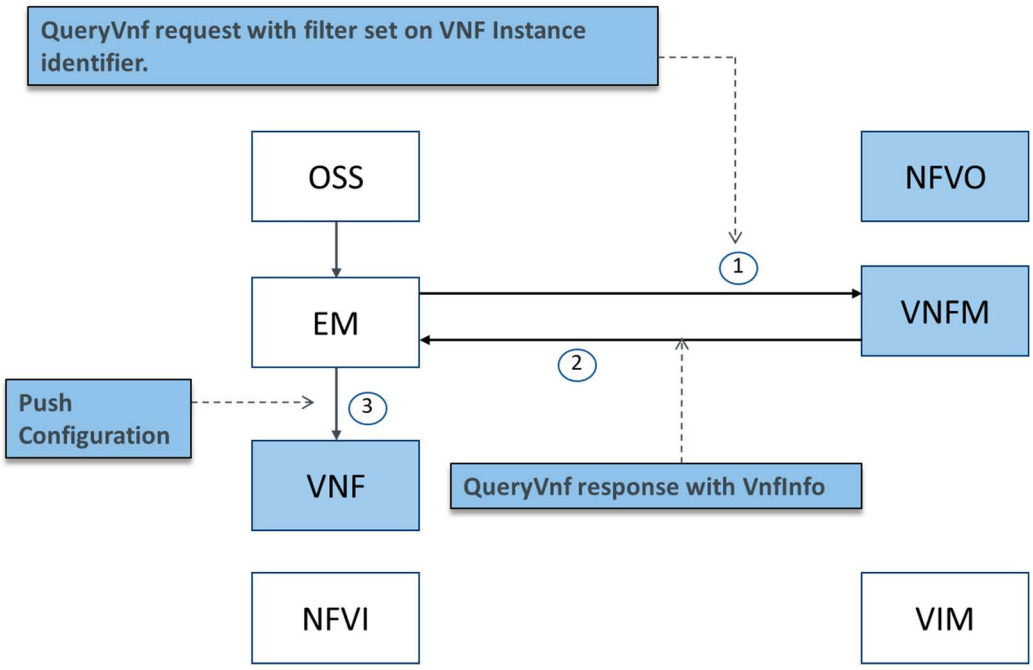

Figure 4.3.2.2.1.2-1: EM-initiated push mode procedure

ETSI GS NFV-SOL 002 [i.6] specifies the Restful API for implementing the VNF LCM interface including the QueryVnf operation used in the preamble and the operations for managing notifications. The protocol used by the EM for communicating with the VNF is outside the scope of the NFV specifications referenced in clause 2 of the present document.

4.3.2.2.2 Pull mode

In this mode the VNF pulls configuration data from the OSS or from the EM. The address where to send a pull request is either preconfigured in the VNF software (e.g. pre-agreed FQDN communicated at design time) or can provided to the VNF instance as a VNF configurable property set by the OSS or the EM, using configuration method #B or #C described in clauses 4.3.3 and 4.3.4.

4.3.2.2.3 Hybrid modes

The push and pull modes can be combined as follows:

- The EM can pull configuration data from the OSS upon receipt of a notification from the VNFM about the creation or modification of a VNF instance and push these configuration data to the VNF.
- The OSS can push configuration data from the EM and the EM can wait for the VNF instance to pull these data.

4.3.3 Method #B

4.3.3.1 General description

With this configuration method, a VNF instance receives configuration data from the VNFM in charge of managing its lifecycle.

This method is only applicable to Day-1 and Day-2 configuration as it assumes that the VNF instance can communicate with the VNFM. At least one VNFC instance is expected to act a configuration management agent in the VNF.

The configuration data values can be originated from the VNFM or received by the VNFM from the VIM, the EM or the NFVO. In the latter case they can be determined by the NFVO or received by the NFVO from the OSS/BSS.

The configuration procedure can follow a push (configuration data pushed to the VNF by the VNFM) or pull model (configuration data retrieved by the VNF instance from the VNFM).

The specifications of the interfaces between the VNF and the VNFM are contained in ETSI GS NFV-IFA 008 [i.15] (functional requirements) and ETSI GS NFV-SOL 002 [i.6] (API specification).

The items that can be configured using this method are VNF configurable properties declared in the VNFD and connection point configuration data.

NOTE: VNF configurable properties are not the same as modifiable attributes (i.e. extensions and metadata).

Figure 4.3.3.1-1 illustrates this configuration method.

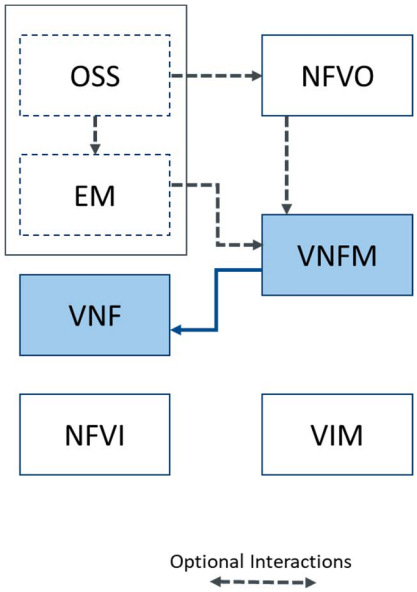

Figure 4.3.3.1-1: VNF configuration method #B

4.3.3.2 Detailed description

4.3.3.2.1 General

Within the VNFD, configurable properties can be declared at the VNF level (VnfConfigurableProperties) and/or at the VDU level (VnfcConfigurableProperties), as specified in clauses 7.1.12 and 7.1.6.7 of ETSI GS NFV-IFA 011 [i.17], respectively.

VNF-level configuration properties include:

- Standard properties to enable/disable auto scaling and auto healing for a VNF instance, to configure OAuth server identities and information to access the APIs produced by the VNFM.
- Additional VNF-specific configuration properties defined by VNF providers.

All VNFC-level configuration properties are VNF-specific properties defined by VNF providers.

While all configurable properties are declared in the VNFD, their values can be:

- Set in the VNFD (default values).
- Computed by an LCM script executed by the VNFM.
- Received by the VNFM from the NFVO or the EM using the following operations of the VNF lifecycle management interface defined in ETSI GS NFV-IFA 007 [i.14] and ETSI GS NFV-IFA 008 [i.15], respectively:
  - Instantiate VNF.

- Change VNF Flavour.
- Modify VNF Information.
- Change current VNF Package.

ETSI GS NFV-SOL 003 [i.7] and ETSI GS NFV-SOL 002 [i.6] specifies the Restful API implementing the VNF LCM interface.

The NFVO or the EM can either compute the values for the configurable properties or receive them from the OSS. The OSS can set configurable property values in the NFVO using the Instantiate NS or Update NS operations of the NS LCM interface specified in clause 7.3 of ETSI GS NFV-IFA 013 [i.18]. ETSI GS NFV-SOL 005 [i.9] specifies the Restful API for implementing this interface.

The TOSCA and YANG representations of configurable properties in a VNFD are specified in ETSI GS NFV-SOL 001 [i.5] and ETSI GS NFV-SOL 006 [i.10] respectively.

4.3.3.2.2 Push mode

In this mode, the VNFM sends a SetConfiguration operation of the VNF configuration interface defined in ETSI GS NFV-IFA 008 [i.15] to the VNF instance as soon as it is created and operational or when a modification occurs. The connection point where to push the configuration data is identified based on the contents of the VNFD. The VNFD for a VNF that supports the VNF configuration interface is expected to contain a VnfInterfaceDetails information element that associates this interface to a VNF external connection point descriptor and additional interface details, as specified in clause 7.1.8.16 of ETSI GS NFV-IFA 011 [i.17].

In addition to configurable properties, the SetConfiguration operation enables the VNFM to configure the addresses and ports that have been assigned to the VNF internal and external connection points.

Figure 4.3.3.2.2-1 illustrates the push mode procedure when configurable property values are provided by the OSS and the EM. The following steps are identified:

1) The configurable property values are sent by the OSS (1a) in an InstantiateNs or an UpdateNs with the update type parameter set to ModifyVnfInformation/InstantiateVnf/ChangeVnfDf/ChangeVnfPkg as specified in ETSI GS NFV-IFA 013 [i.18], or by the EM (1b) in an InstantiateVnf, ModifyVnfInformation, ChangeVnfFlavour or ChangeCurrentVnfPackage request as specified in ETSI GS NFV-IFA 008 [i.15].

2) The NFVO forwards the configurable properties to the VNFM (2a) in an InstantiateVnf, ModifyVnfInformation, ChangeVnfFlavour or ChangeCurrentVnfPackage request as specified in ETSI GS NFV-IFA 007 [i.14], depending on the operation received from the OSS.

3) The VNFM pushes the configurable properties to the VNF using the SetConfiguration operation of the VNF configuration interface defined in ETSI GS NFV-IFA 008 [i.15].

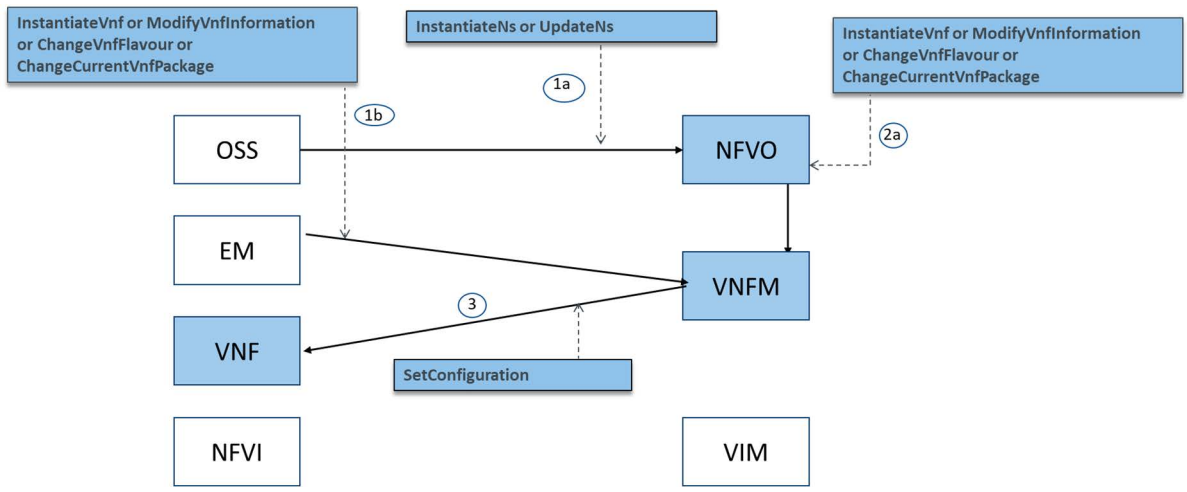

Figure 4.3.3.2.2-1: OSS- and EM-initiated push mode procedure

Clause 9 of ETSI GS NFV-SOL 002 [i.6] defines the Restful API implementing the VNF configuration interface.

The SetConfiguration is implemented using an HTTP PATCH method. Modifications are specified according to the rules of JSON Merge Patch (see IETF RFC 7396 [i.24]). Figure 4.3.3.2.2-2 illustrates the message flows for configuring a VNF instance using the VNF configuration API.

NOTE: As specified in ETSI GS NFV-SOL 002 [i.6], the support of the VNF configuration API is optional for a VNF. Whether a VNF supports this interface is specified in the VNFD.

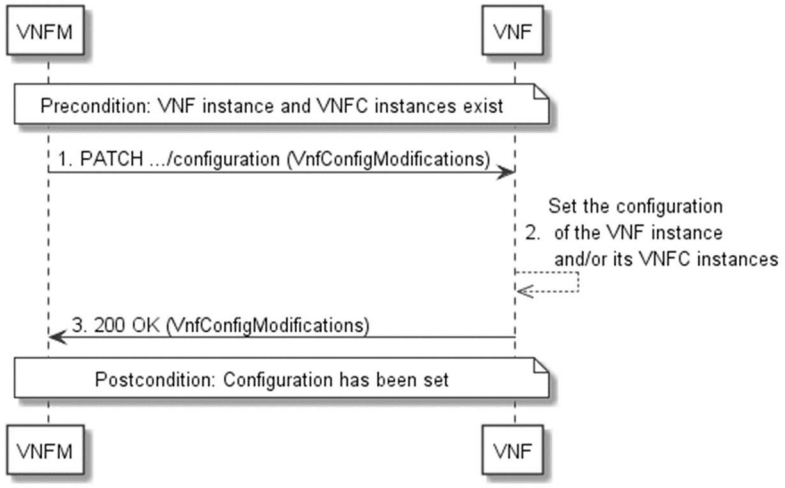

Figure 4.3.3.2.2-2: Restful implementation of the SetConfiguration operation

4.3.3.2.3 Pull mode

In this mode, the VNF instance retrieves configuration data from the VNFM, using the QueryVnf operation of the VNF LCM interface defined in ETSI GS NFV-IFA 008 [i.15]. This procedure can be triggered upon instantiation as soon as external connectivity is available to the VNF or upon receipt of a notification that some information has changed, which assumes that the VNF subscribes to receive these notifications.

In response to a QueryVnf request, depending on the attribute selectors set on the request, the VNF receives all or part of the attributes contained in the VnfInfo information element defined in clause 9.4.2 of ETSI GS NFV-IFA 008 [i.15], including, if requested, the VNF configurable properties.

An alternative approach is described in annex B of ETSI GS NFV-IFA 008 [i.15] for "self-managed" VNFs. In that case, upon receipt of a VNF LCM notification from the VNFM, the VNF invokes a GetOperationStatusRequest operation to obtain details about the operation occurrence. As the response sent by the VNFM contains all the parameters received in the LCM operation request, the VNF can determine which configuration changes are required. Once the configuration has been successfully updated the VNF sends a VNF Indicator notification to the VNFM. The format and values of the VNF indicator are declared by the VNF provider in the VNFD.

NOTE: This approach cannot be applied in the case when configurable properties values are provided during the VNF instantiation request, but it can be applied for subsequent procedures of configuration such as those performed via the Modify VNF information operation of the VNF LCM interface.

Clause 5 of ETSI GS NFV-SOL 002 [i.6] defines the Restful API implementing the VNF LCM interface.

4.3.4 Method #C

4.3.4.1 General description

With this configuration method, a VNF instance receives configuration data via the NFV infrastructure. This configuration method is intended to enable Day-0 configuration. The actual procedure depends on the selected VIM or CISM solution. For example, one solution is for the VIM to write configuration data to a special configuration drive that attaches to a VM instance when it boots.

Figure 4.3.4.1-1 illustrates this configuration method.

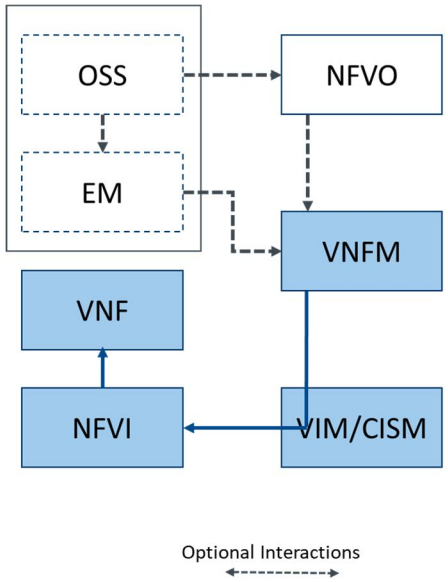

Figure 4.3.4.1-1: VNF configuration method #C

4.3.4.2 Detailed description

4.3.4.2.1 General

The configuration data to be pushed to the VNF are determined by the VNFM, based on information available in the VNFD and/or in configuration files included in MCIOPs (containerized VNFs). These configuration data can differ from one VNFC to another as the configuration method applies on a per VNFC and per instance basis. The actual configuration data values can be:

1) Specified in the VNFD or in the configuration files included in MCIOPs.

2) Received from the NFVO or the EM in the VnfConfigurableProperties information element (see clause 7.1.12 of ETSI GS NFV-IFA 011 [i.17]) of a VNF LCM operation request defined in ETSI GS NFV-IFA 007 [i.14] and ETSI GS NFV-IFA 008 [i.15], respectively.

In the latter case, the NFVO or the EM can either generate these values or receive them from the OSS. The OSS can provide these values to the NFVO using the Instantiate NS or Update NS operations of the NS LCM interface specified in clause 7.3 of ETSI GS NFV-IFA 013 [i.18].

Figure 4.3.4.2.1-1 provides an overview of the configuration procedure when configuration data are provided by the OSS or the EM at NS instantiation time. The following steps are identified:

1) The OSS sends an InstantiateNs request with VNF extensions and/or VNF configurable properties, and/or VNF LCM operation configuration parameters as specified in ETSI GS NFV-IFA 013 [i.18].

2) The NFVO sends an InstantiateVnf request as specified in ETSI GS NFV-IFA 007 [i.14] to the VNFM with the parameters received from the OSS.

3) The VNFM processes the received parameters and forwards them to the VIM or CISM depending on the type of VNF, according to the procedures described in clauses 4.3.4.2.2 and 4.3.4.2.3.

4) The VIM or the CISM provides the VNF configuration data to the VNF hosted in the NFVI, according to the procedures described in clauses 4.3.4.2.2 and 4.3.4.2.3, respectively.

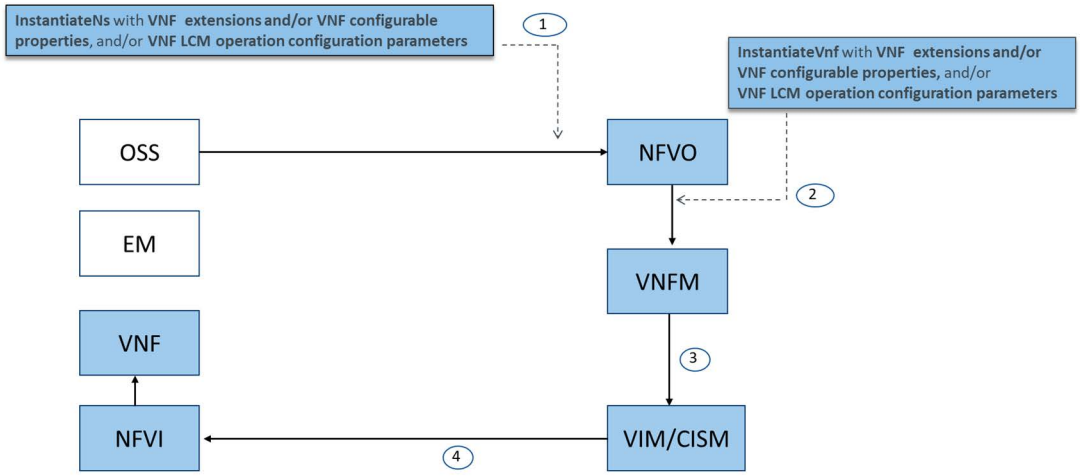

Figure 4.3.4.2.1-1: configuration data transfer procedure at NS instantiation time

ETSI GS NFV-SOL 005 [i.9] specifies the Restful API for implementing the NS LCM interface. ETSI GS NFV-SOL 003 [i.7] specifies the Restful API for implementing the VNF LCM interface exposed by the VNFM to the NFVO. ETSI GS NFV-SOL 002 [i.6] specifies the Restful API for implementing the VNF LCM interface exposed by the VNFM to the EM.

4.3.4.2.2 VM-based VNFs

The configuration data are specified in the boot data attribute of the VNFD, as specified in clause 7.1.6.2.2 of ETSI GS NFV-IFA 011 [i.17]. Boot data is an attribute of a VDU representing initialization data to be pushed to VNFC instances, via the VIM.

NOTE 1: This method cannot be used for Day-1 and Day-2 configuration.

The contents of this attribute can be a list of key-value pairs, a string or a URL pointing to an initialization file contained in the VNF package. This attribute can include volatile and/or persistent variable parts: declared in other information elements of the VNFD, that the VNFM fills with runtime information before pushing the attribute value to the VIM. These variable parts can be filled with the contents of one of more of the following information elements:

- VnfLcmOperationsConfiguration information element (see clause 7.1.5.2 of ETSI GS NFV-IFA 011 [i.17]).
- Extension attribute of the VnfInfoModifiableAttribute information element (see clause 7.1.14 of ETSI GS NFV-IFA 011 [i.17].
- VnfConfigurableProperties information element (see clause 7.1.12 of ETSI GS NFV-IFA 011 [i.17]).

The actual values of these variable parts can be set in the VNFD or received from the NFVO or the EM in a VNF LCM operation request defined in ETSI GS NFV-IFA 007 [i.14] and ETSI GS NFV-IFA 008 [i.15], respectively.

For each VNFC instance created from a VDU for which a boot data attribute exists, the VNFM pushes to the VIM the contents of the boot data attribute each time a new compute resource is created at VNF instantiation time or a later stage (e.g. VNF scaling out). The contents of the boot data attribute is conveyed in the UserData attribute of the AllocateVirtualisedComputeResouce operation specified in ETSI GS NFV-IFA 006 [i.13], in case of VNF-related resource management in direct mode.

The specifications of protocol solutions for the VNFM-VIM reference point as well as the mechanism to transfer the information to the NFVI and the Virtual Machines (VMs) are out of the scope of the NFV specifications referenced in clause 2 of the present document and are dependent on the cloud management solution implementing the VIM functionality. For OpenStack® the configuration information is provided by the VNFM in the "user data" parameter of the VM (server) creation request and relies upon the cloud-init mechanism. The VNFC instance running in the VM can access this data through the metadata service or config drive. This typically assumes that the cloud_init package is installed within the VM (as part of the Linux® distribution). ETSI GS NFV-SOL 014 [i.11] specifies a YAML data model of input/output parameters exchanged via the management of virtualised resource descriptors (being OpenStack HEAT Orchestration Templates (HOT) a concrete realization of them), which includes the modeling of the "userData" attribute defined in ETSI GS NFV-IFA 006 [i.13]. Similar solutions exist for most cloud management systems implementing the VIM functionality.

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

NOTE 3: Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Figure 4.3.4.2.2-1 illustrates the three options for specifying the contents of a boot data attribute in a TOSCA-based VNFD, as specified in ETSI GS NFV-SOL 001 [i.5]. In the first case, a set of key-value-pairs is used. In the second case a string to be sent to the VNF is specified. In the third case, a configuration file to be sent to the VNF is specified.

```
dbBackend:
  type: tosca.nodes.nfv.Vdu.Compute
  properties:
    ..
    boot_data:
      kvp_data:
        data: dns_server: { get_property: [vnf, modifiable_attributes, extensions, dns_server ] }
```

```
dbBackend:
   type: tosca.nodes.nfv.Vdu.Compute
     properties:
       ..
       boot_data:
         content_or_file_data
            contents: { concat: [ "#!/bin/bash\n",  "echo setting DNS sever to: ",
                          { get_property: [vnf, modifiable_attributes, extensions, dns_server ] }, "\n", "..."] }
```

```
dbBackend:
   type: tosca.nodes.nfv.Vdu.Compute
     properties:
       ..
       boot_data:
         content_or_file_data
           data: dns_server : { get_property: [vnf, modifiable_attributes, extensions, dns_server ] }
           source_path: { get_artifact : [ SELF, boot_data ] }
     artifacts:
       boot_data:
         type: tosca.artifacts.example
         file: implementation/templates/boot_data.file
```

Figure 4.3.4.2.2-1: Examples of boot_data properties in a TOSCA-based VNFD

4.3.4.2.3 Containerized VNFs

This method relies upon the mechanisms specified to enable MCIO configuration, see clauses 5.2.1 and 6.7 in ETSI GS NFV-IFA 040 [i.21]. The configuration data are declared in the VNFD (as configurable properties) and in configuration files within MCIOPs. The VNFM performs the mapping between the configurable properties and the corresponding items in the configuration files.

NOTE 1: In contrast to the VM-based solution, this method can be used for both Day-0, Day-1 and Day-2 configuration.

The technical realization depends on the container management solution implementing the CISM functionality. When Kubernetes® and Helm™ are used, the Helm™ charts corresponding to the MCIOPs associated to a VNF to be configured are expected to contain values.yaml file(s) with key-value pairs. The VNFM can create new values.yaml files to override the value in a key-value pair with a configurable property value. It then provides the new values.yaml files to the CISM using the OS container workload management service interface defined in ETSI GS NFV-SOL 018 [i.12] based on the Helm™ CLI, or it provides these new values.yaml files to its internal Helm™ client.

The VNFM gets the configurable property values from the VNFD or receives them from the NFVO or EM via the VNF LCM operation requests defined in ETSI GS NFV-IFA 007 [i.14] and ETSI GS NFV-IFA 008 [i.15], respectively. This method assumes that for each key a corresponding VNF-specific configurable property is defined in the VNFD, unless the key corresponds to a standard configurable property.

For example, these values.yaml files can be used to set configuration values in ConfigMap objects and/or Secret objects. In that case, for each VNFC to be configured, the manifest file of the Pod implementing the corresponding MCIO refers to one or more ConfigMap objects and/or Secret objects. For global VNF configuration, the Pod corresponds to a VNFC that act as an internal manager. For VNFC-specific configuration, the Pod corresponds to the MCIO associated to this VNFC.

The manifest file of a ConfigMap object or a Secret object contains a "data" field which contains a list of key-value pairs. The value in a key-value pair can be specified directly in the manifest file or can be a reference to a parameter in a values.yaml file associated to the Kubernetes® Pod. The method described above assumes that the second option is used. Once the new values.yaml files are received, the CISM or the internal Helm™ client applies the changes to the ConfigMap object or a Secret object, which is then mounted as volume attached to the Pod.

NOTE 2: Besides using ConfigMap and/or Secret objects, Kubernetes® provides alternative mechanisms to transfer configuration data to a Pod, e.g. using Init containers or a combination of commands and arguments declared in the Pod specification.

5 VNF configuration use cases

5.1 General

5.1.1 Introduction

This clause provides a set of VNF configuration use cases to help identify potential gaps in the specifications referenced in clause 2 of the present document. Clause 5.2 focuses on Day-0 configuration use cases. Clause 5.3 focuses on Day-1 configuration use cases. Unless otherwise stated, the use cases described in Clause 5.3 can also be regarded as Day-2 configuration use cases.

5.1.2 Actors

Table 5.1.2-1 describes the actors involved in the use cases.

Table 5.1.2-1: use case actors and roles

#	Actor	Description
1	Service Provider	The Service Provider responsible for the VNF instance to be configured.
2	NFV system	The NFV system managing and hosting the VNF instance to be configured. The term NFV system encompasses NFV-MANO functional blocks, the OSS, the EM and the NFVI.
3	VNF	The VNF instance to be configured.

5.2 Day-0 configuration use cases

5.2.1 UC#Day-0-1: Configuration of a VNF-specific parameter

5.2.1.1 Description

This use case aims at configuring a VNF-specific parameter with a value that the VNF instance needs to obtain before completing the instantiation process. The value is provided by the service provider. The VNF instance needs to obtain the value of a VNF-specific configuration parameter before completing the instantiation process. The VNF-specific parameter can be for example the URL of an HTTP proxy to be used by the VNF instance for communicating with any other entity.

5.2.1.2 Trigger

Table 5.2.1.2-1 describes the use case trigger.

Table 5.2.1.2-1: Day-0 configuration trigger

Trigger	Comment
The Service Provider wishes to instantiate a VNF.	

5.2.1.3 Pre-conditions

None.

5.2.1.4 Post-conditions

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

Table 5.2.1.4-1: Day-0 configuration post-conditions

#	Post-condition	Comment
1	The VNF is instantiated and has received sufficient configuration information to complete the instantiation process.	For example, if needed, the VNF is ready to request Day-1 or Day-2 configuration information via the HTTP proxy.

5.2.1.5 Operational Flows

Table 5.2.1.5-1 describes the base flow of this use case.

Table 5.2.1.5-1: Day-0 configuration, base flow

#	Actor/Role	Description
Begins When	Service Provider -> NFV system	The Service Provider requests the NFV system to instantiate the VNF and provides the value to be configured on the VNF instance.
1	NFV system -> VNF	The NFV system proceeds to the instantiation of the VNF and propagates to the VNF instance the value to be configured.
Ends When	VNF	The VNF instance has received and processed the VNF-specific parameter value.

5.2.2 UC#Day-0-2: Configuration of the VNFM address

5.2.2.1 Description

This use case aims at configuring the VNF with information enabling access to the NFV-MANO interfaces produced by the VNFM.

5.2.2.2 Trigger

Table 5.2.2.2-1 describes the use case trigger.

Table 5.2.2.2-1: Day-0 configuration trigger

Trigger	Comment
The Service Provider wishes to instantiate a VNF.	

5.2.2.3 Pre-conditions

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

Table 5.2.2.3-1: Day-0 configuration pre-conditions

#	Pre-condition	Comment
1	The VNFD contains the vnfmInterfaceInfo configurable property.	

5.2.2.4 Post-conditions

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

Table 5.2.2.4-1: Day-0 configuration post-conditions

#	Post-condition	Comment
1	The VNF is instantiated and has received sufficient configuration information to complete the instantiation process.	

5.2.2.5 Operational Flows

Table 5.2.2.5-1 describes the base flow of this use case.

Table 5.2.2.5-1: Day-0 configuration, base flow

#	Actor/Role	Description
Begins When	Service Provider -> NFV system	The Service Provider requests the NFV system to instantiate the VNF and provides the value to be configured on the VNF instance.
1	NFV system -> VNF	The NFV system proceeds to the instantiation of the VNF and propagates to the VNF instance the value to be configured.
Ends When	VNF	The VNF instance has received and processed the VNFM details.

5.2.3 UC#Day-0-3: Configuration of the EM address

5.2.3.1 Description

This use case aims at configuring the VNF with information needed to enable communication with the corresponding EM.

5.2.3.2 Trigger

Table 5.2.3.2-1 describes the use case trigger.

Table 5.2.3.2-1: Day-0 configuration trigger

Trigger	Comment
The Service Provider wishes to instantiate a VNF.	

5.2.3.3 Pre-conditions

Table 5.2.3.3-1 describes the pre-conditions of this use case.

Table 5.2.3.3-1: Day-0 configuration pre-conditions

#	Pre-condition	Comment
1	EM metadata information is included in VNFD.	According to ETSI GS NFV-IFA 011 [i.17], the VNFD supports a description of metadata to determine if an EM is used for the VNF and parameters describing how to connect to the EM. The metadata description in the VNFD is not changed during the whole VNF lifecycle. The IP address of the EM is not provided in the VNFD, instead it is assumed to be provided during the lifecycle of the VNF, since IP addressing can be deployment specific.

5.2.3.4 Post-conditions

Table 5.2.3.4-1 describes the post-conditions of this use case.

Table 5.2.3.4-1: Day-0 configuration post-conditions

#	Post-condition	Comment
1	The VNF is instantiated and has received sufficient configuration information to complete the instantiation process and connect to the EM.	

5.2.3.5 Operational Flows

Table 5.2.3.5-1 describes the base flow of this use case.

Table 5.2.3.5-1: Day-0 configuration, base flow

#	Actor/Role	Description
Begins When	Service Provider -> NFV system	The Service Provider requests the NFV system to instantiate the VNF and provides the value to be configured on the VNF instance.
1	NFV system -> VNF	The NFV system proceeds to the instantiation of the VNF and propagates to the VNF instance the value to be configured.
Ends When	VNF	The VNF instance has received and processed the EM address details.

5.3 Day-1 configuration use cases

5.3.1 UC#Day-1-1: Modification of a VNF-specific configuration value

5.3.1.1 Description

This use case aims at modifying VNF-specific parameter on an instantiated VNF with a value provided by the Service Provider.

5.3.1.2 Trigger

Table 5.3.1.2-1 describes the use case trigger.

Table 5.3.1.2-1: Day-1 configuration trigger

Trigger	Comment
The Service Provider wishes to modify a VNF-specific parameter value.	For example, the Service Provider identifies the need to change the HTTP proxy initially configured on the VNF instance for communicating with other entities.

5.3.1.3 Pre-conditions

Table 5.3.1.3-1 describes the pre-conditions of this use case.

Table 5.3.1.3-1: Day-1 configuration pre-conditions

#	Pre-condition	Comment
1	The VNF instance exists.	

5.3.1.4 Post-conditions

Table 5.3.1.4-1 describes the post-conditions of this use case.

Table 5.3.1.4-1: Day-1 configuration post-conditions

#	Post-condition	Comment
1	The VNF has received and processed the new value.	For example, if needed, the VNF is ready to request Day-1 or Day-2 configuration information via the new HTTP proxy.

5.3.1.5 Operational Flows

Table 5.3.1.5-1 describes the base flow of this use case.

Table 5.3.1.5-1: Day-1 configuration, base flow

#	Actor/Role	Description
Begins When	Service Provider -> NFV system	The Service Provider requests the modification of a VNF-specific parameter on a VNF instance and provides the new value to be configured.
1	NFV system -> VNF	The NFV system propagates to the VNF instance the new value.
Ends When	VNF	The VNF instance has received and processed the new value.

5.3.2 UC#Day-1-2: Configuration of an internal load balancer

5.3.2.1 Description

This use case aims at configuring an internal load balancer VNFC with the IP addresses of the connection points of other VNFC instances within the VNF instance. For the purpose of this use case it is assumed that the VNF is composed of two types of VNFCs: VNFC-A and VNFC-B. VNFC-A is a load balancer. The role of an instance of VNFC-A in the VNF is to receive a portion of the incoming traffic and distribute it across multiple instances of VNFC-B. The values that are configured depend on the actual deployment of the constituents of the VNF, and therefore, the present use case is about virtualisation-dependent data configuration.

5.3.2.2 Trigger

Table 5.3.2.2-1 describes the use case trigger.

Table 5.3.2.2-1: Day-1 configuration trigger

Trigger	Description
A decision to scale-out a VNFC within the VNF is made.	The decision can be made by the OSS, the NFVO or the VNFM.

5.3.2.3 Pre-conditions

Table 5.3.2.3-1 describes the pre-conditions of this use case.

Table 5.3.2.3-1: Day-1 configuration pre-conditions

#	Pre-condition	Comment
1	The VNF instance exists.	

5.3.2.4 Post-conditions

Table 5.3.2.4-1 describes the post-conditions of this use case.

Table 5.3.2.4-1: Day-1 configuration post-conditions

#	Post-condition	Comment
1	The VNFC-A instance has received the IP address of the connection point of the new VNFC-B instance.	VNFC-A can add the new VNFC-B instances to the list of VNFC-B instances across which incoming traffic can be distributed.

5.3.2.5 Operational Flows

Table 5.3.2.5-1 describes the base flow of this use case.

Table 5.3.2.5-1: Day-1 configuration, base flow

#	Actor/Role	Description
Begins When	Service Provider or NFV system	A decision to scale-out a VNF instance is taken. The Service Provider requests the NFV system to scale out the VNF instance or the NFV system decided to scale-out the VNF instance.
1	NFV system -> VNF	The NFV system scales-out the VNF instance and provides to the VNFC-A of the VNF instance the IP address(es) of the connection point(s) of the new VNFC-B instance(s)
Ends When	VNF	The VNFC-A instance has received the IP addresses and acknowledged the configuration update.

5.3.3 UC#Day-1-3: Configuration of proxy/firewall VNF

5.3.3.1 Description

This use case aims at configuring forwarding rules in a VNF implementing the function of an application layer proxy/firewall (e.g. an http proxy with filtering capabilities). Therefore, the present use case is about virtualisation-independent data configuration.

5.3.3.2 Trigger

Table 5.3.3.2-1 describes the use case trigger.

Table 5.3.3.2-1: Day-1 configuration trigger

Trigger	Comment
Forwarding rules need to be configured on the proxy/firewall VNF.	

5.3.3.3 Pre-conditions

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

Table 5.3.3.3-1: Day-1 configuration pre-conditions

#	Pre-condition	Comment
1	The proxy/firewall VNF has been instantiated.	

5.3.3.4 Post-conditions

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

Table 5.3.3.4-1: Day-1 configuration post-conditions

#	Post-condition	Comment
1	The proxy/firewall VNF has received the forwarding rules to enforce and is ready to process traffic.	

5.3.3.5 Operational Flows

Table 5.3.3.5-1 describes the base flow of this use case.

Table 5.3.3.5-1: Day-1 configuration, base flow

#	Actor/Role	Description
Begins When	Service Provider -> NFV system	The Service Provider provides to the NFV system the set of forwarding rules to be configured on the proxy/firewall VNF instance.
1	NFV system -> VNF	The NFV systems provides the set of forwarding rules to the proxy/firewall VNF instance.
Ends When	VNF	The proxy/firewall VNF instance has received and acknowledged the forwarding rules.

5.3.4 UC#Day-1-4: Distribution and storage of VNF configuration data

5.3.4.1 Description

This use case aims at showcasing the distribution of VNF configuration data to one or multiple VNF instances in a common manner and the storage of the VNF configuration data in the NFV system. Configuration data can be applied "synchronously" upon receipt of a specific request, or "asynchronously" when some event in the NFV system takes place. Configuration data can be pushed or pulled to/by the VNF instances.

5.3.4.2 Trigger

Table 5.3.4.2-1 describes the use case trigger.

Table 5.3.4.2-1: Distribution and storage of VNF configuration data

Trigger	Comment
The Service Provider wishes to provide configuration for one or more multiple VNF instances; VNF(s) can be instantiated, or not.	For example, the Service Provider identifies that several VNF instances should be provided a common set of virtualisation-independent or virtualisation-dependent configuration parameters among other sets of VNF-specific parameters.

5.3.4.3 Pre-conditions

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

Table 5.3.4.3-1: Distribution and storage of VNF configuration data

#	Pre-condition	Comment
1	The NFV system has dedicated or shared storage resources for configuration data.	The NFV system is expected to have some sort of "configuration data repository".

5.3.4.4 Post-conditions

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

Table 5.3.4.4-1: Distribution and storage of VNF configuration data

#	Post-condition	Comment
1	If there is an instantiated VNF that is affected by the new configuration values, the VNF has received and processed the new value.	
2	The VNF configuration data is available in the NFV system.	The VNF configuration data has been stored and can be applied to future VNF instances, if necessary.

5.3.4.5 Operational Flows

Table 5.3.4.5-1 describes the base flow of this use case.

Table 5.3.4.5-1: Distribution and storage of VNF configuration data

#	Actor/Role	Description
Begins When	Service Provider -> NFV system	The Service Provider provides VNF configuration data for one or multiple VNFs.
1	NFV system	The NFV system stores the VNF configuration data into the dedicated/shared storage resources.
2	NFV system -> VNF	If there is a VNF instance to which VNF configuration data applies, the values are propagated to the VNF instance.
Ends When	VNF NFV system	The VNF configuration data has been stored, and if applicable, the VNF instance has received and processed the new value.

6 Analysis and Key Issues

6.1 General

Clause 6.2 identifies key issues to be addressed when using the configuration methods described in clause 4 of the present document, regardless of the actual use case.

Clause 6.3 provides an analysis of the use cases described in clause 5 the present document with regards to the applicability of the configuration options described in clause 4 of the present document and identify associated key issues that are specific to the use case considered.

6.2 Common key issues

Table 6.2-1 describes the key issues to be addressed when using the configuration method #A.

Table 6.2-1: General key issues for configuration method #A

Issue #	Issue description
GKI-A-1	If the configuration procedure is driven by the OSS, the specifications referenced in clause 2 of the present document do not describe how does the OSS identify which type of connection points to use for pushing configuration data in direct mode to the VNF instance and in the non-direct mode what is the EM address where to push configuration data.
GKI-A-2	If the configuration procedure is driven by the OSS, the specifications referenced in clause 2 of the present document do not describe how does the OSS determine the protocol and data model to use for communicating with the VNF/EM. See note 1.
GKI-A-3	If the configuration procedure is driven by the EM, the specifications referenced in clause 2 of the present document do not describe how does the EM identify which type of connection points to use to push configuration data.
GKI-A-4	If the configuration procedure is driven by the EM, the specifications referenced in clause 2 of the present document do not describe how does the EM learn the VNFM address where to send subscription requests.
GKI-A-5	If the configuration procedure is driven by the EM, the specifications referenced in clause 2 of the present document do not describe how does the EM determine the protocol and data model to use for communicating with the VNF. See note 1.
GKI-A-6	If the configuration procedure is driven by the EM, either in push or pull mode, the specifications referenced in clause 2 of the present document do not describe how to configure the EM address or other related information into the VNF. See note 2.
NOTE 1: The issue is also applicable in the case of pulling configuration data from the OSS/EM by the VNF. NOTE 2: The issue about configuration of the EM address into the VNF for the case of pulling VNF configuration from the EM or for the VNF to establish necessary EM-VNF peering (to fulfil various use cases such as indicating the VNF where to send Alarm notifications and performance data, to indicate to the VNF by which entity it is being managed, etc.) is also analysed in clause 6.3.1.3.	

Table 6.2-2 describes the key issues to be addressed when using the configuration method #B.

Table 6.2-2: General key issues for configuration method #B

Issue #	Issue description
GKI-B-1	How to use this configuration method if the VNF configuration API specified in ETSI GS NFV-SOL 002 [i.6] is not supported.

Table 6.2-3 describes the key issues to be addressed when using the configuration method #C.

Table 6.2-3: General key issues for configuration method #C

Issue #	Issue description
GKI-C-1	When this configuration method is used for containerized VNFs, there is no standard solution for mapping VNF configurable property names to parameter names in values.yaml files.

6.3 Use case analysis

6.3.1 Day 0 use cases

6.3.1.1 UC#Day-0-1: Configuration of a VNF-specific parameter

6.3.1.1.1 Implementation options

All variants of configuration method #A and method B, as well as configuration method C can be used under certain conditions.

Configuration method #A can be used if the configuration of the VNF-specific parameter is not a pre-requisite for enabling communication between the VNF instance and the EM or the OSS. For example, if the VNF-specific parameter is an HTTP proxy, the communication between the VNF instance and the EM or the OSS is not mediated by this proxy.

Configuration method #B can be used if:

a) the configuration of the VNF-specific parameter is not a pre-requisite for enabling communication between the VNF instance and the VNFM. For example, if the VNF-specific parameter is an HTTP proxy, the communication between the VNF instance and the VNFM is not mediated by this proxy; and

b) a configuration property corresponding to the VNF-specific parameter is declared in the VNFD.

Configuration method #C can be used if:

a) a configuration property or modifiable attribute (extension) corresponding to the VNF-specific parameter is declared in the VNFD; and

b) in case of a VM-based VNF this configurable property or modifiable attribute is referenced as a variable part of a boot data attribute associated to a VDU.

6.3.1.1.2 Key Issues

There are no use case specific key issues in addition to those identified in clause 6.1.

6.3.1.2 UC#Day-0-2: Configuration of the VNFM address

6.3.1.2.1 Implementation options

All variants of configuration method #A and configuration method C can be used under certain conditions.

Configuration method #A can be used if the VNF instance can communicate with the OSS or the EM and the OSS knows the VNFM address or can retrieve it from the NFVO, or the EM knows the VNFM address or can retrieve it from the OSS.

Configuration method #C can be used if:

a) the vnfmInterfaceInfo configuration property is declared in the VNFD; and

b) in case of a VM-based VNF this configurable property is referenced as a variable part of a boot data attribute associated to a VDU; and

c) in case of a container-based VNF a parameter corresponding to this configurable property is declared in the values.yaml file included in an MCIOP associated to one of the VDUs.

6.3.1.2.2 Key Issues

There are no use case specific key issues in addition to those identified in clause 6.1.

6.3.1.3 UC#Day-0-3: Configuration of the EM address

6.3.1.3.1 Implementation options

Configuration method #B (both push and pull modes) and configuration method #C can be used under certain conditions.

In principle the IP address of the EM is not provided in the VNFD, instead assumed to be provided during the lifecycle of the VNF, since the EM's IP address can be deployment specific. Under configuration method #B the VNF instance can receive such information from the VNFM. The VNFM knows the EM's IP address from the OSS through NFVO interaction.

For configuration method #B, a modifiable configurable property is declared in the VNFD. The sequence for setting the configuration value with method #B includes:

- OSS to NFVO: ModifyVnfInfoData information element can be used.
- NFVO to VNFM: EM address information can be passed from the NFVO to VNFM as a VNF configurable property using the ModifyVnfInfo operation.

- VNFM to VNF: vnfSpecificData from VnfConfigurationData data type can be used according to ETSI GS NFV-IFA 008 [i.15] to configure EM address information.

Configuration method #C can be used if, in case of a VM-based VNF, a configurable property or modifiable attribute is referenced as a variable part of a boot data attribute associated to a VDU, as described in clause 4.3.4.2.2. In the case of a container-based VNF, a configurable property is declared in the VNFD and a corresponding parameter is declared in configuration files within MCIOPs as described in clause 4.3.4.2.3.

6.3.1.3.2 Key Issues

There are no use case specific key issues in addition to those identified in clause 6.1.

6.3.2 Day 1 use cases

6.3.2.1 UC#Day-1-1: Modification of a VNF-specific configuration value

6.3.2.1.1 Implementation options

All variants of configuration methods #A and #B can be used.

Configuration method #B can only be used if the VNF-specific configuration parameter are modelled as a configurable property declared in the VNFD and the OSS provides the configurable property value to the NFVO in an NS instantiate or NS update operation.

6.3.2.1.2 Key Issues

There are no use case specific key issues in addition to those identified in clause 6.1.

6.3.2.2 UC# Day-1-2: Configuration of an internal load balancer

6.3.2.2.1 Implementation options

All variants of configuration method #A and method B can be used.

Use of method #A assumes that prior to pushing configuration data to the VNF instance:

a) the OSS has subscribed to receive a notification from the NFVO when a VNF instance is scaled out and queries the NFVO to retrieve the description of the new connection points (i.e. in the NsInfo.VnfInfo data structure of the QueryNs response as defined in ETSI GS NFV-IFA 013 [i.18]; or

b) the EM has subscribed to receive a notification from the VNFM when a VNF instance is scaled out and queries the VNFM to retrieve the description of the new connection points (i.e. in the VnfInfo data structure of the QueryVnf response as defined in ETSI GS NFV-IFA 008 [i.15].

6.3.2.2.2 Key Issues

Table 6.3.2.2.2-1 describes key issues to be addressed when using method #A, in addition to those identified in clause 6.1 of the present document.

Table 6.3.2.2.2-1: Specific key issues for configuration method #A

Issue #	Issue description
SKI-A-1	If the configuration procedure is driven by the OSS, the presence of connection point information in VnfInfo is not mandatory for internal connection points unless they are re-exposed as external connection points.
SKI-A-2	If the configuration procedure is driven by the OSS, scaling procedures resulting from an auto-scale decision in the VNFM do not lead the NFVO to send a notification to the OSS.

6.3.2.3 UC#Day-1-3: Configuration of proxy/firewall VNF

6.3.2.3.1 Implementation options

All variants of configuration methods #A and #B can used.

However, use of configuration method #B implies that the set of firewall rules is modelled as a configurable property declared in the VNFD and the OSS provides the configurable property value to the NFVO in an NS instantiate or NS update operation.

6.3.2.3.2 Key Issues

Table 6.3.2.3.2-1 describes key issues to be addressed when using method #B, in addition to those identified in clause 6.1 of the present document.

Table 6.3.2.3.2-1: Specific key issues for configuration method #B

Issue #	Issue description
SKI-B-1	This method implies to convey a large amount of data via NFV-MANO without any added value. A key issue is how to mitigate the risk of overloading NFV-MANO functions and interfaces.
SKI-B-2	A key issue is to determine how to modify a single forwarding rule without having to re-send the entire set of rules.

6.3.2.4 UC#Day-1-4: Distribution and storage of VNF configuration data

6.3.2.4.1 Implementation options

This use case is about storage and distribution of VNF configuration to one or multiple VNF instances in a common manner. Information can be stored in the form of key-value pairs in a common storage. However, different implementations could consider a different approach on the format and the API used for storing and retrieving information to/from the common storage.

In the case of "VNF generic OAM functions" as defined by ETSI GR NFV-EVE 019 [i.22], centralized configuration operations can be associated with the "VNF configuration" and "Network configuration" VNF generic OAM functions, which can be smoothly integrated with the concept of common storage.

Configuration method #A and configuration method #B can also be used under certain conditions and specific interactions. Furthermore, VNF generic OAM functions can also be included in the process.

For the case of configuration method #A, when VNF generic OAM functions are not used, the configuration path is updated to "OSS/EM to (common storage) to VNF(s)". OSS/EM will write to the common storage the VNF configuration data to be provided to the registered VNFs.

For the case of configuration method #A, when VNF generic OAM functions are used, two configuration paths are possible:

- Common storage as frontend: the configuration path is "OSS/EM to (common storage) to VNF configuration/Network configuration managers to VNF".
- VNF generic OAM configuration managers as frontend: the configuration path is "OSS/EM to VNF configuration/Network configuration managers to (common storage) to VNF".

For the case of configuration method #B, the configuration path is "(OSS/EM or NFVO or EM) to VNFM to (common storage) to VNF". Instead of OSS/EM, in this case the VNFM writes to the common storage the VNF configuration to be provided to the registered VNFs. Similarly like before configuration is pushed to the appropriate VNFs (or pulled by the VNFs).

For the case of configuration method #B, when VNF generic OAM functions are used, two configuration paths are also possible:

- Common storage as frontend: the configuration path is "(OSS/EM or NFV or EM) to VNFM to (common storage) to VNF configuration/Network configuration managers to VNF".

- VNF generic OAM configuration managers as frontend: the configuration path is "(OSS/EM or NFVO or EM) to VNFM to VNF configuration/Network configuration managers to (common storage) to VNF".

For both methods #A and #B, for the interaction of the common storage with the different entities like OSS/EM, VNFM or VNFs, both push and pull modes are possible.

6.3.2.4.2 Key Issues

The concept of a centralized storage entity used to support VNF configuration operations (e.g. read, write) is not currently part of the overall ETSI NFV-MANO framework. To enable such functionality the following key issues have been identified:

- Setting up the "centralized storage for configuration" in the NFV framework: This key issue is about integrating the centralized storage entity into the NFV framework from an architectural perspective. How to manage the centralized storage itself and how this centralized storage is discovered by and connected to the Functional Blocks (FBs) and the VNF generic OAM functions are also part of this key issue.
- Protocols for centralized storage for configuration: This key issue is about investigating the appropriate protocols and APIs to allow reading and writing operations to the centralized storage by one or many FBs or VNF generic OAM functions towards the VNFs. How to enable notifications between the different entities when new configuration information is available is also part of this key issue analysis.

7 Potential Solutions

7.1 Introduction

This clause identifies solutions to address the key issues identified in clause 6 for all configuration methods.

7.2 Solutions for general key issues

7.2.1 Configuration method #A

7.2.1.1 General key issue#1 (GKI-A-1)

7.2.1.1.1 Solutions

7.2.1.1.1.1 Solution#1

This solution consists in adding a new attribute to the connection point descriptors to identify in a machine-processible format the connection points to be connected to the OSS.

NOTE: The Connection point descriptor (Cpd) information element defined in ETSI NFV-IFA 011 [i.17] contains a description attribute and a cpRole attribute. However, the description attribute is not intended to be machine processable and the cpRole attribute is meant to convey information on the role of the connection point with regards to the connectivity topology (e.g. it can specify whether a connection point plays a root or leaf role in an ethernet tree topology). These attributes are not suitable for conveying the above type of indication.

This solution can be used for the direct mode and for the non-direct mode when the EM is deployed and managed as a VNF, as described in ETSI GS NFV-IFA 009 [i.16]. In the first case the new attribute is set in the VNFD of the managed VNF. In the second case it is set in the VNFD of the EM. It assumes that the OSS is able to parse the contents of a VNFD. The actual addresses of the connection points created from the identified connection point descriptor are retrieved by the OSS from the NFVO using the QueryNs operation defined in ETSI GS NFV-IFA 013 [i.18].

7.2.1.1.1.2 Solution#2

This solution consists in identifying the connection point descriptor in a dedicated file included in the VNF package, as a non-MANO artefact. A public non-MANO artifact set identifier can be registered for that purpose according to the procedures described in ETSI GS NFV-SOL 004 [i.8].

This solution can be used for the direct mode and the non-direct mode when the EM is deployed and managed as a VNF. The non-MANO artifact identifies the relevant connection descriptor in the VNFD of the managed VNF or in the VNFD of the EM, respectively. The actual addresses of the connection points created from the identified connection point descriptor are retrieved by the OSS from the NFVO using the QueryNs operation defined in ETSI GS NFV-IFA 013 [i.18].

7.2.1.1.1.3 Solution#3

This solution consists in provisioning the EM addressing data in the OSS once the EM is deployed. This solution applies to the non-direct mode irrespective of how the EM is deployed.

7.2.1.1.1.4 Solution#4

The address of the configuration endpoint is discovered by the OSS using a discovery service provided by a service registry. This solution assumes that the VNF to be configured (direct mode) or its EM (non-direct mode) exposes a configuration service interface and register the interface details in a service registry similar to the NFV-MANO Service (NFV-S) registry identified in ETSI GR NFV-IFA 039 [i.20] as a potential enhancement to the NFV-MANO framework within the Release 5 scope.

7.2.1.1.1.5 Solution#5

The address of the configuration endpoint is discovered by the OSS using a discovery service provided by the VNF (direct mode) or the EM (non-direct mode). This solution assumes that all VNFs and EMs expose a discovery service interface at a well-known address.

7.2.1.1.2 Evaluation of solutions

Table 7.2.1.1.2-1 identifies the pros and cons of the different solutions.

Table 7.2.1.1.2-1: Pros and Cons analysis

Solution	Pros	Cons
Solution#1	Simplicity.	
Solution#2	Can be extended to address GKI-A-2.	Overhead, although the same solution can be used for addressing GKI-A-1 and GKI-A-2.
Solution#3		Lack of flexibility. Limited support for end-to-end automation of network service management.
Solution#4	Flexibility. Can be extended to address GKI-A-2.	Additional service registry function to be deployed.
Solution#5	Flexibility. Can be extended to address GKI-A-2.	Additional functionality in the VNF.

7.2.1.2 General key issue#2 (GKI-A-2)

7.2.1.2.1 Solutions

7.2.1.2.1.1 Solution#1

The protocol(s) and associated data model(s) to use are specified in dedicated files included in the VNF package, as non-MANO artifacts. A public non-MANO artifact set identifier can be registered for that purpose according to the procedures described in ETSI GS NFV-SOL 004 [i.8]. The artifact set would be expected to contain an interface description file specifying the protocol configuration information to access the VNF or the EM, as well as one or more files specifying the data model to be used for representing configuration data (e.g. YANG modules). The protocol configuration information typically includes a protocol identifier and various configuration parameters if applicable (e.g. URL root, port number, etc.). This solution is similar to solution#4 for GKI-A-1 with the difference that more information is contained in the non-MANO artifact set.

7.2.1.2.1.2 Solution#2

The protocol(s) and associated data model(s) to use are discovered by the OSS using a discovery service provided by the VNF (direct mode) or the EM (non-direct mode). This solution assumes that all VNFs and EMs expose a discovery service interface at a well-known address.

7.2.1.2.1.3 Solution#3

The protocol(s) and associated data model(s) to use are discovered by the OSS using a discovery service provided by a service registry. This solution assumes that all VNFs expose a configuration service interface and register the interface details in a service registry similar to the NFV-MANO Service (NFV-S) registry identified in ETSI GR NFV-IFA 039 [i.20] as a potential enhancement to the NFV-MANO framework within the Release 5 scope. This solution is similar to solution#4 for GKI-A-1 with the difference that the registry returns more information.

7.2.1.1.1.4 Solution#4

This solution consists in configuring in the OSS, by means outside the scope of standardization, the protocol(s) and associated data model(s) to use for communicating with the EM. This solution applies to the non-direct mode irrespective of how the EM is deployed.

7.2.1.1.1.5 Solution#5

This solution consists in using an agent-less configuration solution. An example of agent-less configuration solution is Ansible®. In that case, the VNF (direct mode) or the EM (non-direct mode) would be expected to provide the SSH server functionality and possibly a Python® interpreter depending on the complexity of the configuration procedure. The VNF package would typically contain, as non-MANO artefacts, Ansible playbooks (set of configuration instructions) to be executed by the OSS acting as an Ansible server.

NOTE 1: Ansible® is a trademark of Red Hat, Inc. in the United States and other countries.

NOTE 2: Python® is s a registered trademark of the Python Software Foundation ("PSF").

NOTE 3: An Ansible playbook can call another protocol (e.g. NETCONF) that is known to be supported by the VNF or EM.

7.2.1.2.2 Evaluation of solutions

Table 7.2.1.2.2-1 identifies the pros and cons of the different solutions.

Table 7.2.1.2.2-1: Pros and Cons analysis

Solution	Pros	Cons
Solution#1	A single solution can be built to address both GKI-A-1 and GKI-A-2.	
Solution#2	Flexibility. A single solution can be built to address both GKI-A-1 and GKI-A-2.	Additional functionality in the VNF.
Solution#3	Flexibility. A single solution can be built to address both GKI-A-1 and GKI-A-2.	Additional service registry function to be deployed.
Solution#4		Lack of flexibility. Limited support for end-to-end automation of network service management.
Solution#5	A single solution can be built to address both GKI-A-1 and GKI-A-2. No software to be installed on the EM or VNF beyond a Python® interpreter in some cases.	

7.2.1.3 General key issue#3 (GKI-A-3)

7.2.1.3.1 Solutions

7.2.1.3.1.1 Solution#1

In case of a VNF-independent EM, the solution relies upon the presence of a new attribute in the connection point descriptors to identify in a machine-processible format the connection points to be connected to the EM. In addition, the EM queries the VNFM to obtain the instance details of the corresponding connection points.

In case of a VNF-specific EM, the EM knows which connection point descriptors represent connection points to the connected to the EM and queries the VNFM to obtain their instance details.

7.2.1.3.2 Evaluation of solutions

Solution#1 relies on the same enhancement to the VNFD as solution#1 for GKI-A-1 and has the same pros and cons.

7.2.1.4 General key issue#4 (GKI-A-4)

7.2.1.4.1 Solutions

7.2.1.4.1.1 Solution#1

The VNFM uses the configuration method #B to provide the VNF with the interface details for subscribing to VNF lifecycle management notifications. The EM retrieves this information from the VNF. In case of a VNF-independent EM, it is assumed that a standard interface is exposed by the VNF for that purpose. In case of a VNF-specific EM, the means to retrieve this information are outside the scope of the present document.

7.2.1.4.1.2 Solution#2

The OSS provides the EM with the VNFM interface details for subscribing to VNF lifecycle management notifications. This solution assumes that the OSS can retrieve this information from the NFVO.

7.2.1.4.2 Evaluation of solutions

Table 7.2.1.4.2-1 identifies the pros and cons of the different solutions.

Table 7.2.1.4.2-1: Pros and Cons analysis

Solution	Pros	Cons
Solution#1		Assumes that the VNF be instantiated before the EM can get the VNFM interface details.
Solution#2		Assumes an evolution of the NS LCM interface exposed by the NFVO.

7.2.1.5 General key issue#5 (GKI-A-5)

7.2.1.5.1 Solutions

In case of a generic EM, the solutions applicable to the OSS in direct mode, as described in clause 7.2.1.2 apply. In case of a VNF-specific EM, the means to discover the protocols and data models used for the communication between the EM and the VNF are outside the scope of the present document.

7.2.1.5.2 Evaluation of solutions

See clause 7.2.1.2.

7.2.1.6 General key issue#6 (GKI-A-6)

7.2.1.6.1 Solutions

7.2.1.6.1.1 Solution#1

As indicated in the key issue "UC#Day-0-3: configuration of the EM address" described in clause 6.3.1.3, in the case of establishing VNF-EM peering, or for enabling the VNF to pull configuration data from the EM, the VNF is expected to have information about the connection points (or network interfaces) and corresponding IP addresses of the EM.

Similar to the case of the VNF-independent EM described in clause 7.2.1.3.1.1, the VNFD is enhanced, but in the present key issue case to support a standard description to determine if an EM is used and to specify how to connect to the EM. The solution proposes to provide such description as modifiable standard configurable properties, which define the attributes that can be configured into the VNF regarding the EM-related configuration such as EM's connection point IP address information.

7.2.1.6.1.2 Solution#2

This solution is similar to solution#1, but instead of defining a set of new standard configurable properties to determine the EM-related configuration into the VNF, the information is assumed the attributes can be defined by the VNF designer as part of "additionalConfigurableProperty" in the "VnfConfigurableProperties" of the VNFD.

7.2.1.6.2 Evaluation of solutions

Table 7.2.1.6.2-1 identifies the pros and cons of the different solutions.

Table 7.2.1.6.2-1: Pros and Cons analysis

Solution	Pros	Cons
Solution#1	• Configurable properties about EM-related configuration are clearly and unambiguously specified.	• Not all VNFs might need such type of EM-related configuration, so the new standard configurable properties are redundant (i.e. not applicable or used). See note 1. • Some complex modelling (e.g. specific parameter usage conditions, specific attributes, etc.) might be needed to cover as many deployment scenarios of management systems for the VNF as possible (e.g. with EM, VNF generic OAM functions, etc.).
Solution#2	• Configurable properties about EM-related configuration only defined when needed.	• Additional VNF-specific logic in the NFV-MANO (or in an LCM script) is needed to interpret the name(s) and values of the related configurable properties about EM-related configuration. See note 2.
NOTE 1: This might not be a major issue, because VNF configurable properties can be modelled with cardinality "0..1" or "0..N", enabling a VNF designer not to use them when not applicable. NOTE 2: NFV-MANO is expected to be able to interpret the semantics of the configurable property, e.g. to associate such information to other runtime information or status such as checking the availability of VNF generic OAM functions, consuming services from an EM such as VNF LCM coordination interfaces, etc.		

7.2.2 Configuration method #B

7.2.2.1 General key issue#1 (GKI-B-1)

7.2.2.1.1 Solutions

7.2.2.1.1.1 Solution#1

If the VNF configuration API specified in ETSI GS NFV-SOL 002 [i.6] is not supported, configuration method A can be used providing that the data model used for this configuration method enable transferring the same information as the configuration interface.

Configuration method A can be initiated by the OSS or the EM. A pre-requisite for the OSS or the EM to provide connection point configuration data to the VNF, is to retrieve these data from the NFVO or the VNFM (respectively). This can be achieved by using the QueryNS operation defined in ETSI GS NFV-IFA 013 [i.18] or the QueryVnf operation defined in ETSI GS NFV-IFA 008 [i.15].

7.2.2.1.1.2 Solution#2

This solution consists in using another API than the VNF configuration API specified in ETSI GS NFV-SOL 002 [i.6] (e.g. an API based on the Netconf [i.3] or gNMI [i.4] protocols).

The API could be an alternative standard for the Ve-Vnfm-vnf reference point that would be developed by ETSI NFV or a VNF-specific API. In the latter case, the VNFM would use this API via a Lifecycle Management (LCM) script included by the VNF provider in the VNF package and invoked when executing a lifecycle management operation. For example, the VNFD can specify that this lifecycle management script be invoked as a postamble to all LCM operations.

The data model for representing the configuration data on the Ve-Vnfm reference point would have to be embedded in the LCM script or provided as a separate artefact in the VNF package.

NOTE: This approach assumes that the Domain Specific Language used for LCM scripts enables requesting the VNFM to make use of a non-MANO interface. The DSL requirements specified in ETSI GS NFV-IFA 011 [i.17] do not include supporting this feature. Hence a VNFM complying with ETSI NFV specifications might not have the ability to execute such requests.

7.2.2.1.1.3 Solution#3

This solution consists in using a generic VNF configuration manager function as described in ETSI GR NFV-EVE 019 [i.22] that would expose the VNF configuration API specified in ETSI GS NFV-SOL 002 [i.6] and provide the necessary adaptation functionality for accessing the native configuration API exposed by the VNF.

7.2.2.1.1.4 Solution#4

This solution consists in using an agent-less configuration solution. For further details see Solution#5 for key issue GKI-A-2 in clause 7.2.1.1.1.5. The VNFM would invoke the agent-less solution either directly when executing a lifecycle management operation or via a Lifecycle Management (LCM) script included by the VNF provider in the VNF package and associated to lifecycle management operation postambles in the VNFD.

NOTE 1: The later approach assumes that the DSL used for LCM scripts enables requesting the VNFM to trigger an agent-less solution. The DSL requirements specified in ETSI GS NFV-IFA 011 [i.17] do not include supporting this feature. Hence a VNFM complying with ETSI NFV specifications might not have the ability to execute such requests.

Clause A.1 illustrates this solution with Ansible® as the agent-less configuration solution.

NOTE 2: ETSI GS NFV-SOL 001 [i.5] specifies that the DSL for an LCM script can be written in Python®, which is sufficient to support Ansible®.

7.2.2.1.2 Evaluation of solutions

Table 7.2.2.1.2-1 identifies the pros and cons of the different solutions.

Table 7.2.2.1.2-1: Pros and Cons analysis

Solution	Pros	Cons
Solution#1	This solution is enabled by the current specifications.	Implies additional message exchanges.
Solution#2	This solution does not restrict the type of configuration interface to be supported by a VNF.	Dependency on the LCM scripting language supported by the VNFM.
Solution#3	This solution does not restrict the type of configuration interface to be supported by a VNF and does not implies use of LCM scripts.	Assumes that a VNF configuration manager function is available.
Solution#4	No software to be installed on the VNF beyond a Python® interpreter in some cases.	Additional functionality in the VNFM (e.g. Ansible® server).

7.2.3 Configuration method #C

7.2.3.1 General key issue#1 (GKI-C-1)

7.2.3.1.1 Solutions

7.2.3.1.1.1 Solution#1

This solution consists in modelling the configuration procedure as a preamble of the operations of the vnflcm interface in the VNFD (e.g. instantiate_start). The implementation of these preamble operations is a script that creates a values.yaml file according to the values of the VNF configurable properties. The specification of the inputs for this operation provides the mapping between the names of the configurable properties in the VNFD and the names of the corresponding parameters in the values.yaml file.

Clause A.2 provides an example.

7.2.3.1.1.2 Solution#2

This solution is based on the introduction of a new information element in the ETSI NFV VNFD specification that allows the vendor to indicate the mapping between the configurable properties and parameters in the Helm™ charts.

Helm™ parameter names are chosen by the vendor, as well as the parameter hierarchy and the Helm™ chart names. Therefore, they are not known to NFV-MANO a priori.

In order to allow a generic VNFM to be able to provide values to Helm™ chart input parameters that are represented as configurable properties in the VNFD, the VNFD contains an information element that identifies the parameter in the Helm™ chart corresponding to every vendor defined configurable property. Values for those properties will typically be provided through VNF LCM interface at VNF instantiation or at a later stage and further injected into Helm™ by the VNFM.

Whereas this solution is described in the present document for configurable properties, it is also applicable for other vendor defined attributes, like modifiable attributes extensions or additional parameters of the VNF LCM operations, as well as NFV standard parameters in the VNF LCM operations that are also used to provide values to Helm™ charts input parameters, e.g. IP addresses, Network Attachment Definition (NAD) names, software image repositories, etc.

Clause A.3 provides an example.

7.2.3.1.2 Evaluation of solutions

Table 7.2.3.1.2-1 identifies the pros and cons of the different solutions.

Table 7.2.3.1.2-1: Pros and Cons analysis

Solution	Pros	Cons
Solution#1	Does not assume any change to the VNFD specifications.	• Assumes VNF-specific LCM scripts are attached to the preamble of LCM operations or standardizing an LCM script for that purpose. • NFV specifications do not specify that the output of a preamble is made available to the corresponding base operation. • Requires standardization of a new input parameter of the main LCM operation.
Solution#2	Does not assume the presence of VNF-specific LCM scripts in the VNF package.	Requires standardization of a new information element in the VNFD.

7.3 Solutions for specific key issues

7.3.1 Configuration method #A

7.3.1.1 Specific key issue#1 (SKI-A-1)

7.3.1.1.1 Solutions

7.3.1.1.1.1 Solution#1

This solution consists in mandating the presence for all internal connection points of connection point details in the VnfInfo information element returned in response to a QueryNs operation as defined in ETSI GS NFV-IFA 013 [i.18].

7.3.1.1.1.2 Solution#2

With this solution the NFVO triggers the configuration process towards the EM but requests the EM to retrieve the VnfInfo information element from the VNFM, using a QueryVnf operation as defined in ETSI GS NFV-IFA 008 [i.15].

7.3.1.1.1.3 Solution#3

This solution consists in using the EM-driven variant of configuration method#A rather than the OSS-driven procedure.

7.3.1.1.2 Evaluation of solutions

Table 7.3.1.1.2-1 identifies the pros and cons of the different solutions.

Table 7.3.1.1.2-1: Pros and Cons analysis

Solution	Pros	Cons
Solution#1	No dependency on the EM capabilities beyond being able to relay configuration data provided by the OSS.	Increases visibility of the NFVO on the VNF internals.
Solution#2	No changes required to the NS LCM interface.	Creates a dependency on the EM capabilities.
Solution#3	No changes required to the NS LCM interface.	Creates a dependency on the EM capabilities.

7.3.1.2 Specific key issue#1 (SKI-A-2)

7.3.1.2.1 Solutions

7.3.1.2.1.1 Solution#1

This solution consists in modifying the NFVO behaviour to enable sending a notification to the OSS if a VNF instance has been modified as a result of an auto-scale decision made by the VNFM.

7.3.1.2.1.2 Solution#2

This solution consists in using the EM-driven variant of configuration method#A rather than the OSS-driven procedure. The EM subscribes to receive the notifications.

7.3.1.2.2 Evaluation of solutions

Table 7.3.1.2.2-1 identifies the pros and cons of the different solutions.

Table 7.3.1.2.2-1: Pros and Cons analysis

Solution	Pros	Cons
Solution#1	No dependency on the EM capabilities beyond being able to relay configuration data provided by the OSS.	
Solution#2	No changes required to the NS LCM interface.	Creates a dependency on the EM capabilities.

7.3.2 Configuration method #B

7.3.2.1 Specific key issue#1 (SKI-B-1)

7.3.2.1.1 Solutions

7.3.2.1.1.1 Solution#1

This solution relies upon proper engineering and dimensioning of the NFV-MANO implementations and communication links, to minimize the risk of congestion and ensure that messages carrying critical management requests get prioritized if congestion cannot be avoided.

7.3.2.1.1.2 Solution#2

This solution consists in restricting the use of this configuration method to use cases where the amount of data to be configured remains under a given threshold.

7.3.2.1.2 Evaluation of solutions

Table 7.3.2.1.2-1 identifies the pros and cons of the different solutions.

Table 7.3.2.1.2-1: Pros and Cons analysis

Solution	Pros	Cons
Solution#1	Applicable to any VNF.	Increase the complexity of implementations and network dimensioning.
Solution#2	Reduce risk of overloading NFV-MANO.	Restrict the applicability of the configuration method.

7.3.2.2 Specific key issue#1 (SKI-B-2)

7.3.2.2.1 Solutions

7.3.2.2.1.1 Solution#1

This solution consists in modelling every individual piece of configuration data (e.g. every forwarding rule in case of a firewall VNF) as an individual configurable property.

7.3.2.2.1.2 Solution#2

This solution consists in leveraging a modification description format such as JSON Merge Patch (IETF RFC 7396 [i.24]).

7.3.2.2.2 Evaluation of solutions

Table 7.3.2.2.2-1 identifies the pros and cons of the different solutions.

Table 7.3.2.2.2-1: Pros and Cons analysis

Solution	Pros	Cons
Solution#1	No constraint on the serialization language used to represent the information to be configured.	Unnecessary overhead for every piece of information.
Solution#2	No overhead for individual information items.	Places a constraint on the data serialization language used to represent the information to be configured.

7.3.3 Configuration methods related to distribution and storage of VNF configuration data

7.3.3.1 Specific key issue: setting up the "centralized storage for configuration" in the NFV framework

7.3.3.1.1 Overview

Clause 7.3.3.1 describes potential solutions to address the following key issues about how the centralized storage for configuration introduced in the use case in clause 5.3.4 "Distribution and storage of VNF configuration data":

- can be integrated into the NFV framework from an architectural perspective;
- can be managed; and
- can be discovered and necessary connectivity to other functional blocks and/or VNF instances be established.

In the solutions analysis, and with the purpose to simplify the terminology usage and the documentation of potential solutions, the entity offering the "centralized storage for configuration" is referred as "Configuration Server", which can also include other components beyond the storage, such as data processing, data format translation and interface frontends. Furthermore, even though the Configuration Server is the central point for distributing configuration data, it does not imply that a single centralized storage resource is used. Instead, storage resources of the Configuration Server could be multiple and distributed.

Figure 7.3.3.1.1-1 illustrates a high-level concept of the Configuration Server. Figure 7.3.3.1.1-1 also illustrates that consumers of the centralized storage for configuration services interact with the Configuration Server.

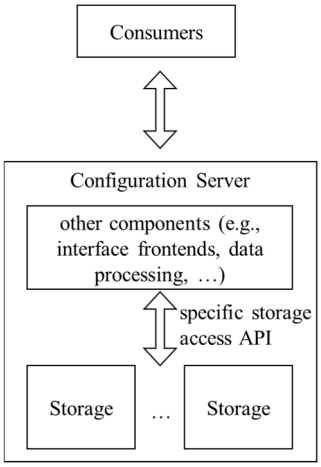

Figure 7.3.3.1.1-1: High-level concept of Configuration Server

7.3.3.1.2 Solutions

7.3.3.1.2.1 Solution #1: PaaS Service

The Configuration Server is realized as a PaaS Service, either as VNF Dedicated Service or VNF Common Service. If realized as a VNF Common Service, the Configuration Server can be used for multiple VNF instances. If realized as a VNF Dedicated Service, an instance of the Configuration Server can only be used by a single VNF instance. In both cases, it is expected that interactions with the Configuration Server are generic (e.g. common interface), even if a specific instance is dedicated to a specific VNF instance.

In this solution, the Configuration Server is managed as a PaaS Service, hence a framework for managing PaaS Services is leveraged. Nonetheless, for storing the configuration data, storage resources provided by the NFVI are still expected to be used, but not managed and visible directly by the consumers of the Configuration Server.

Furthermore, as a PaaS Service, the Configuration Server can provide, not only actual storage capabilities, but also other capabilities such as specific interface/API frontend for storing/retrieving data, adaptation of data formats and other data management and processing capabilities.

NOTE: As described in clause 7.1.1 of ETSI GR NFV-IFA 029 [i.19], PaaS Services can be deployed in different forms, such as VNFs, as NFVI resources, or as a new object class specific to the PaaS concept. Other solutions documented in clause 7.3.3.1 of the present document also refer to deploying and managing the Configuration Server in similar forms. In the present solution, the differential aspect is that the Configuration Server is managed as a PaaS Service and not the actual form of deployment.

Figure 7.3.3.1.2.1-1 illustrates an overview of the solution and the interactions between the Configuration Server and other components of the framework, including the VNF instances to which configuration data is applicable.

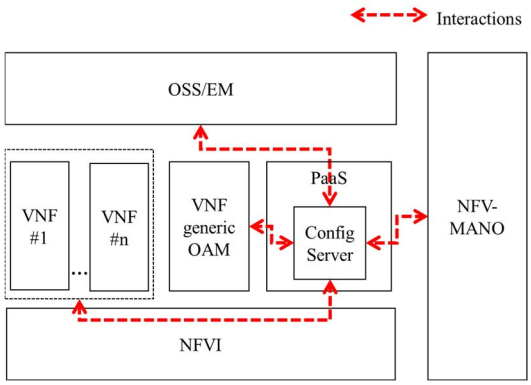

Figure 7.3.3.1.2.1-1: Overview of solution #1 based on PaaS Services

7.3.3.1.2.2 Solution #2: VNF generic OAM function integration

The Configuration Server is integrated into the VNF configuration and/or the Network configuration manager function of the VNF generic OAM framework described in ETSI GR NFV-EVE 019 [i.22].

In this solution, the Configuration Server is managed as a specific capability of the VNF configuration/Network configuration manager function. Management of the VNF configuration and Network configuration manager functions and thus of the configuration server is possible. Nonetheless, for storing the configuration data, storage resources provided by the NFVI are still expected to be used, but not managed and visible directly by the consumers of the Configuration Server. How VNF Generic OAM functions are managed is out of scope of the present document.

The interactions of other management functions with the Configuration Server do not happen "directly". Instead, access to store and read data from the Configuration Server happens through the VNF Configuration/Network configuration manager function.

NOTE 1: The configuration paths described in clause 6.3.2.4.1 for the cases of using VNF generic OAM functions framework are not applicable, because the present solution considers the integration with the actual VNF generic OAM function.

Figure 7.3.3.1.2.2-1 illustrates an overview of the solution and the interactions among the Configuration Server and other components of the framework, including the VNF instances for which configuration data is applicable.

NOTE 2: VNF generic OAM is represented as a logical building block in figure 7.3.3.1.2.2-1. This representation does not preclude any of the deployment documented in ETSI GR NFV-EVE 019 [i.22] for VNF generic OAM functions, such as: deployed as a VNF or deployed as PaaS Services (VNF Common/Dedicated Services). In the present solution, the differential aspect is that the Configuration Server is managed as a VNF generic OAM function and not the actual form of deployment.

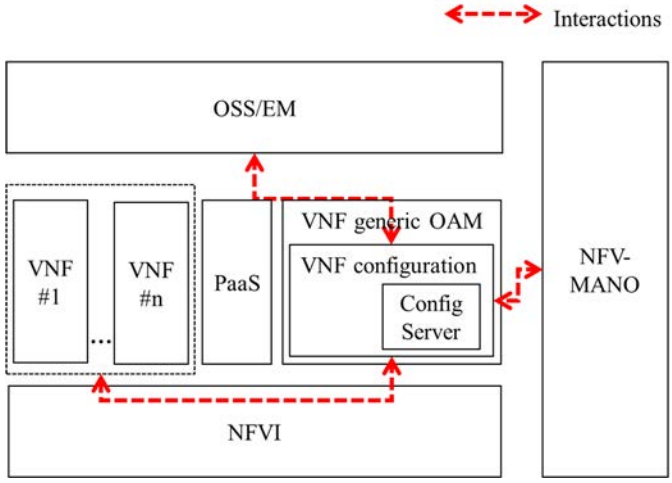

Figure 7.3.3.1.2.2-1: Overview of solution #2 based on VNF generic OAM function integration

7.3.3.1.2.3 Solution #3: NFVI resource

The Configuration Server is realized as a storage-type virtualised resource in the NFVI or as a Storage MCIO in the case of considering container-based VNF deployed, either fully or partially, on a CIS cluster. Storage could be of different types such as block, object or file. This solution envisions that the storage resource is accessible directly by the consumer and the API/interface exposed by the storage resource is used as is by the consumer. In this regard, the Configuration Server basically only includes storage components, like when exposing a storage hard drive as a volume to an Operating System (OS).

In this solution, the Configuration Server is managed as a virtualised resource in the NFVI or as a Storage MCIO in the CIS cluster. Aspects of management of virtualised resources and containerized workloads is hence applicable. The consumer of the Configuration Server has visibility of interacting with actual storage resources from the NFVI/CIS cluster.

The interactions of other management functions with the Configuration Server happen according to the storage format (block, object, or file) capabilities, which can determine in some cases the need of specific service and interface hooks for accessing the storage.

Figure 7.3.3.1.2.3-1 illustrates an overview of the solution and the interactions among the Configuration Server and other components of the framework, including the VNF instances for which configuration data is applicable.

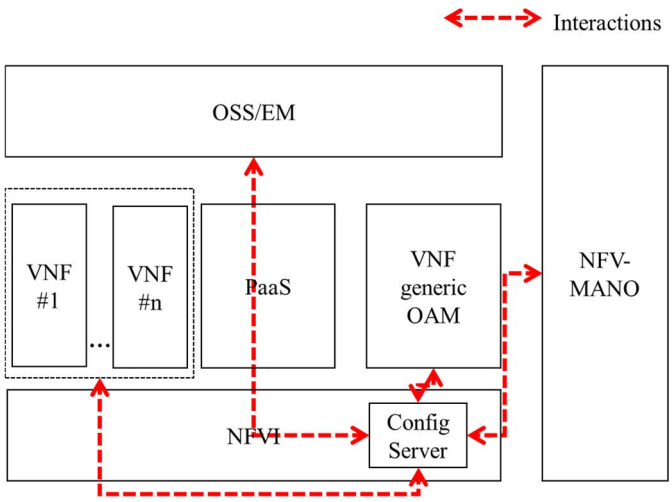

Figure 7.3.3.1.2.3-1: Overview of solution #3 based on NFVI resources

7.3.3.1.2.4 Solution #4: VNF

The Configuration Server is realized as a VNF that can be deployed or referenced by NSs with VNF instances to be configured.

In this solution, the Configuration Server is managed as a VNF. Aspects of management of VNF is hence applicable.

The interactions with the Configuration Server happen according to the interfaces/APIs exposed by the Configuration Server VNF. If VNF instances to be configured are expected to access the centralized storage to retrieve configuration, the connectivity requirements of the centralized storage towards the VNF can be defined via the NSD and the contained NS VLDs.

Figure 7.3.3.1.2.4-1 illustrates an overview of the solution and the interactions among the Configuration Server and other components of the framework, including the VNF instances for which configuration data is applicable.

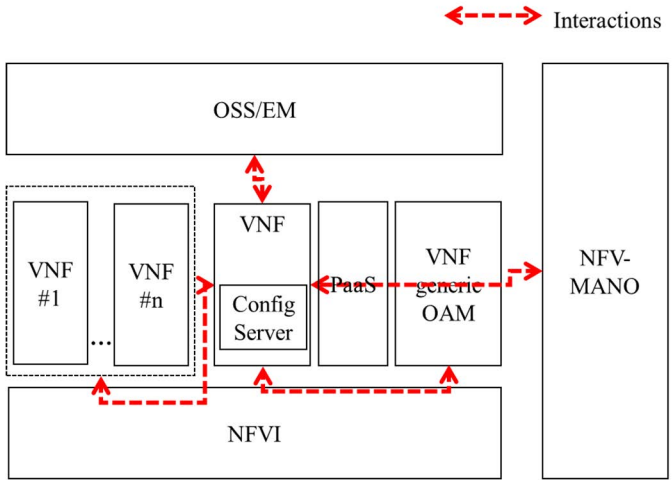

Figure 7.3.3.1.2.4-1: Overview of solution #4 based on VNF deployment

7.3.3.1.3 Evaluation of solutions

Table 7.3.3.1.3-1 identifies the pros and cons of the different solutions documented in the previous clauses.

Table 7.3.3.1.3-1: Pros and cons analysis

Solution	Pros	Cons
Solution #1	a) Simplified deployment and management by reusing the PaaS Services framework. b) If other management and orchestration components, such as VNF generic OAM functions, are also deployed using PaaS, a holistic complete solution can be delivered. c) Scope of the configuration data that can be stored and distributed via the Configuration Server in this solution can be broader and beyond VNF configuration.	a) No (known) standard mechanisms used to define the connectivity requirements and setup for interacting with PaaS Services, which can make more complex the integration of the solution.
Solution #2	a) Simplified deployment and management by reusing the VNF generic OAM functions framework. b) Interfaces/APIs produced by the VNF generic OAM function can be reused/extended for the purposes of storing and reading configuration data.	a) If the Configuration Server is integrated into both the VNF Configuration and Network configuration manager functions, what VNF configuration to distribute is not straightforward, and such VNF configuration data distribution split diminishes the purpose of centralizing the distribution of such a configuration data. b) Due to the scope of the "VNF generic OAM functions", implicitly, the scope of Configuration Server is narrowed to VNF configuration topics, and thus, it is unclear how the Configuration Server in this solution could be of use also for handling configuration of other managed objects than a VNF. c) No (known) standard mechanisms used to define the connectivity requirements and setup for interacting with VNF generic OAM functions when realized as PaaS Services, which can make more complex the integration of the solution.
Solution #3	a) Simplified deployment and management by reusing the virtualised resource and containerized workloads management features of VIM and CISM. b) Scope of configuration data that can be stored and distributed via the Configuration Server in this solution can be broader and beyond to VNF configuration.	a) Unclear support for additional data processing capabilities beyond simple storing/reading of data. b) Reusability and sharing of the NFVI resources to multiple VNF instances is not straightforward, as storage virtualised resources and storage MCIOs are generally coupled to their consuming VNF instance. In the case of storage MCIOs, these are also visible to a single CIS cluster (where these are enabled) and not to multiple CIS clusters.
Solution #4	a) Connectivity requirements to the Configuration Server can be well defined via NSD and VLD properties. b) Configuration Server can potentially provide additional data storage and data processing services as VNF's composition flexibility can be leveraged to integrate additional components and services. c) Scope of configuration data that can be stored and distributed via the Configuration Server in this solution can be broader and beyond to VNF configuration.	a) Potential need to integrate the VNF providing the Configuration Server into multiple (potentially many) NS. b) Increase of inter VNF dependencies to be considered when designing the NS, as VNF and other consumers (such as VNF generic OAM functions) are expected to be composed into the same NS for accessing the VNF used to deploy the Configuration Server.

7.3.3.2 Specific key issue: protocols for centralized storage for configuration

7.3.3.2.1 Overview

The key issue is about what protocols and APIs can be used to store and read configuration data into/from the centralized storage.

7.3.3.2.2 Solutions

7.3.3.2.2.1 Solution #1: REST-based HTTP protocol

This solution considers the use of a REST-based HTTP protocol for the interface produced by the Configuration Server to write and read configuration data. A specific new API is expected to be defined, which can be tailored to the purposes and capabilities offered by the Configuration Server.

HTTP protocol and its defined methods offers the capability to express basic Create, Read, Update and Delete (CRUD) operations to manage the configuration data, which can fulfil the basic capabilities of storing, modifying and reading such data.

Data configuration objects of generic types, e.g. Key-Value-Pairs (KVP), can be defined to convey any kind of flat configuration data structure, or specific data types can also be defined to form a hierarchical/composable data structure to be encoded in JSON payloads in the HTTP messages.

EXAMPLE: An example of this solution would be similar to the definition of the "ConfigurableProperties" attributes in payloads of "VnfInfoModificationRequest" and "VnfInfoModifications" used in the VNF LCM API specified in ETSI GS NFV-SOL 003 [i.7] and ETSI GS NFV-SOL 002 [i.6].

7.3.3.2.2.2 Solution #2: NETCONF (& RESTCONF)

This solution considers the use of NETCONF protocol, as specified in IETF RFC 6241 [i.3], for the interface produced by the Configuration Server. NETCONF provides a standardized way to update and modify configuration data on a managed network device through Remote Procedure Calls (RPCs). RESTCONF is specified in IETF RFC 8040 [i.25] and it is an HTTP-based REST API alternative to the NETCONF. NETCONF is used jointly with YANG: the former provides the communication protocol while the latter is the modelling language used to describe the configuration data. Data modelled in YANG can be encoded into XML or JSON.

NETCONF offers the capability to Create, Read, Update and Delete (CRUD) such resource objects, hence fulfilling the capabilities of storing, modifying and reading configuration data into the Configuration Server. In addition, it also offers other powerful operations for managing configurations such as: commits, copy configuration, locking configuration, etc.

Additional extensions to NETCONF specified in IETF RFC 8526 [i.26] enable the support of the network management datastore architecture with new operations for getting and editing data.

7.3.3.2.2.3 Solution #3: Protocol based on "API resource objects" defined by a specific API

This solution considers the use of specific API protocols and associated resource objects for the interface produced by the Configuration Server to write and read configuration data.

The resource objects are exposed through the API, sometimes referred to as "API resource objects". The resource object provides the capability to define data values, potentially in various formats such as string-based values, binary values, or event with certain specific encodings such as Base64. String-based values also enable the possibility to define complex data structures in the form of "configuration files", possibly coded in some other data modelling encoding format such as JSON. An instance of a resource object can be used to compile a set of configuration data at a desired granularity of choice by the consumer or the target configuration. In this case, the target configuration for a VNF instance could for instance be composed of the data included in one or more resource objects.

The API offers the capability to Create, Read, Update and Delete (CRUD) such resource objects, hence in effect fulfilling the capabilities of storing, modifying and reading configuration data into the Configuration Server.

EXAMPLE: An examples of this solution is the Kubernetes® API with "ConfigMap" and "Secret" API objects [i.27].

7.3.3.2.2.4 Solution #4: Ansible® based Configuration Server

This solution considers the use of Ansible® as an implementation of the Configuration Server, which could be further integrated as a PaaS Service (see clause 7.3.3.1.2.1 Solution #1: PaaS Service).

In terms API exposure, Ansible® Tower offers a web-based solution to the Ansible® server. A REST-based API as well as command line tools are provided to interact with the server.

Through the REST-based API, specific resources are defined according to the modelling and concepts of Ansible® in terms of organizing configuration information for respective inventoried hosts. An agent-less host (i.e. inventoried host) is a managed entity from the perspective of Ansible®, which in the NFV framework could be a VNF instance.

For the modelling of the configuration data, Ansible® supports the feature of templates that can be used to setup configuration files for the managed entities. The templates can also include variables that can be fed at runtime and thus make the templates versatile and reusable.

7.3.3.2.3 Evaluation of solutions

Table 7.3.3.2.3-1 identifies the pros and cons of the different solutions documented in the previous clauses.

Table 7.3.3.2.3-1: Pros and Cons analysis

Solution	Pros	Cons
Solution #1	a) Fully generic API can be defined using common and well-defined industry protocols/approach. b) Flexibility in defining the configuration data structure.	a) The creation of a new API could create additional solution space fragmentation where configuration protocols and data models have already been defined.
Solution #2	a) Fully generic and standard API and protocol can be leveraged. b) Flexibility in defining the configuration data structure with well-defined data model language (YANG).	a) Protocol can be somewhat complex beyond the simple purpose of storing/modifying/reading data, as certain capabilities of the solution are tailored to actual management of the configuration on a managed entity.
Solution #3	a) De-facto integration and support by specific infrastructure management and orchestration solutions, such as Kubernetes®. b) API objects typically provide powerful means to define flexible configuration data structures, either flat key-value-pair structures, or more complex data structure in "configuration files" using even other coding.	a) The solution might be constrained and reusable for only certain kind of deployments, either VM or container-based deployments, and to concrete infrastructure management and orchestration solutions such as Kubernetes®.
Solution #4	a) Well documented and used solution enabling to store and distribute configuration data. See note. b) Flexible solution in terms of defining common constructs that can be used for generic configuration tasks, while also providing the capability to define specific configuration tasks with scripting. c) Does not limit the data modeling and actual configuration files that can be stored and used for the configuration tasks.	a) Protocol can be somewhat complex beyond the simple purpose of storing/modifying/reading data, as most of the capabilities and mechanisms (e.g. Ansible modules) of the solution are designed to support other VNF management functions than just configuration.
NOTE: Ansible® also offers the capability to perform the actual configuration into the managed entities, as also documented in Solution #5 in clause 7.2.1.1.1.5.		

8 Recommendations for future work

8.1 General recommendations

REQ-GEN-1: It is recommended that mechanisms be specified to enable VNFs and EMs to register, in a service registry, information enabling other entities to discover the protocols and data models to use for providing configuration information (See GKI-A-2).

REQ-GEN-2: It is recommended that a requirement be specified on the DSL for LCM script to support invoking an agent-less configuration solution (e.g. Ansible®) (See GKI-B-4).

REQ-GEN-3: It is recommended that the pros and cons analysis made in clause 7 be turned into a set of guidelines.

REQ-GEN-4: It is recommended that mechanisms be specified in the NFV architectural framework to enable the distribution and storage of configuration data for one or multiple VNF instances in a common and centralized manner (also known as Configuration Server) (see UC#Day-1-4 key issues, see note 1).

NOTE 1: Even though the interactions to distribute and store configuration data is performed in a centralized manner, the storage of configuration data can be distributed.

REQ-GEN-5: It is recommended to select a solution for general key issue#1 for configuration method C (GKI-C-1) that supports the mapping of all other API parameters, in addition to configurable properties, into MCIOP input parameters.

NOTE 2: Other API parameters that will be mapped to MCIOP input parameters are extensions, LCM operation additional parameters and standard API parameters.

8.2 Recommendations on functional behaviour

REQ-FUNC-1: It is recommended that a requirement be specified for the NFVO to support the capability of sending a notification to the OSS if a VNF instance has been modified as a result of an auto-scale decision made by the VNFM. (See SKI-A-2).

REQ-FUNC-2: It is recommended that a requirement be specified for the NFVO to support the capability of including VNFM interface details for subscribing to VNF lifecycle management notifications in the VNF information made available to the OSS. (See GKI-A-4).

REQ-FUNC-3: It is recommended that requirements and recommendations be specified to determine the capabilities of the Configuration Server, including aspects of (not an exhaustive list): storing configuration data, transforming configuration data, enabling the provisioning and reading of configuration data, notifying about changes about stored configuration data.

REQ-FUNC-4: It is recommended that requirements be specified for the VNFM to support the capability of interacting with a Configuration Server for the purpose of providing/reading configuration data for/of managed VNF instances (see UC#Day-1-4 key issues).

REQ-FUNC-5: It is recommended that requirements be specified for the relevant VNF generic OAM functions (e.g. VNF configuration function, Network configuration function) to support the capability of interacting with a Configuration Server for the purpose of reading and applying configuration data for the managed VNF instances (see UC#Day-1-4 key issues).

8.3 Recommendations on descriptors

REQ-DESC-1: It is recommended that a new attribute be added to the connection point descriptors in the VNFD to identify in a machine-processible format the connection points to be connected to the OSS/EM. (See GKI-A-1, GKI-A-3).

REQ-DESC-2: It is recommended that new standard VNF configurable properties be specified in the VNFD to enable identifying and performing the configuration into the VNF of connection points and relevant protocol layer information (e.g. IP addresses) of the management system to which the VNF is to be connected/peered with (see GKI-A-3 and UC#Day-0-3, see note 1 and note 2).

NOTE 1: The recommendation uses the generic term "management system" to refer to any responsible function or system for the management of the VNF, such as EM, OSS, or even one or more VNF generic OAM functions as described in ETSI GR NFV-EVE 019 [i.22].

NOTE 2: By defining new VNF configurable properties, their setting into the VNF is already automatically supported by the VNF Configuration interface by means of the "vnfSpecificData" attribute. Therefore, a new recommendation on interfaces is not derived.

REQ-DESC-3: It is recommended that VNFD specifications be enhanced to mandate that the output of an LCM script triggered upon detection of a VNF lifecycle event corresponding to the start of a VNF LCM procedure as described in ETSI GS NFV-IFA 011 [i.17] be made available as input to the implementation of the corresponding LCM procedure.

NOTE 3: For a TOSCA-based VNFD, as specified in ETSI GS NFV-SOL 001 [i.5], this recommendation translates in mandating that the output of a preamble operation be made available as input to the corresponding base operation.

8.4 Recommendations on interfaces

REQ-IF-1: It is recommended that interfaces produced by the Configuration Server are specified for enabling the interoperability between consumers and the Configuration Server (see UC#Day-1-4 key issues, see note).

NOTE: The specification form can vary depending on the chosen solution at the normative phase, e.g. referencing a de-facto standard, profiling an existing standard, specifying new interfaces, etc.

8.5 Security related recommendations

8.6 Recommendations related to cross-organizations collaboration

8.7 Other recommendations

REQ-OTHER-1: It is recommended that VNFs be designed so as to be able to register their configuration interfaces details in a service registry, directly or indirectly (e.g. via a PaaS service). (See GKI-A-1, GKI-A-2.)

Annex A:
VNF Configuration Examples

A.1 VNF configuration with Ansible®

This clause provides an example where a VNFM configures a VNF instance via Ansible®. The VNFD complies with ETSI GS NFV-SOL 001 [i.5].

In this example the two configuration items are modelled as two configurable properties parameter_1 and parameter_2.

The implementation associated to the postamble of the instantiate VNF operation is a Python® script to call the Ansible® server. The Python® script, the Ansible® server code as well as the Ansible® playbooks are assumed to be available in the VNF package.

The values of the configurable properties are received from the NFVO by the VNFM at instantiation time and are used as input to the Python® script implementing the postamble of the instantiate VNF operation.

```
tosca_definitions_version: tosca_simple_yaml_1_3

description: VNF that can be configured using Ansible

imports:
  - etsi_nfv_sol001_vnfd_types.yaml  # all of TOSCA VNFD types as defined in ETSI GS NFV-SOL 001

data_types:

  Mycompany.datatypes.nfv.VnfConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfConfigurableProperties
    properties:
      additional_configurable_properties:
        type: MyCompany.datatypes.nfv.VnfAdditionalConfigurableProperties

  MyCompany.datatypes.nfv.VnfAdditionalConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfAdditionalConfigurableProperties
    properties:
      parameter_1:
        type: string
        required: true
        default: value_1
      parameter_2:
        type: string
        required: true
        default: value_2

node_types:

  MyCompany.ExampleVnf.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      descriptor_id: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      provider: MyCompany
      product_name: ExampleVnf
      software_version: '1.0'
      descriptor_version: '1.0'
      flavour_id:
        constraints: [ valid_values: [ simple, configurable ] ]
        default: simple
      flavour_description:
        default: ""
      vnfm_info: [ '0:MyCompany-1.0.0' ]
      configurable_properties:
        type: Mycompany.datatypes.nfv.VnfConfigurableProperties
        description: Describes the configurable properties of the VNF
        required: false
    interfaces:
      Vnflcm:
        type: tosca.interfaces.nfv.Vnflcm
        operations:
          instantiate:
            inputs:
              configurable_parameters:
```

```
                type: Mycompany.datatypes.nfv.VnfConfigurableProperties
                required: false


topology_template:

  inputs:
    configurable_properties:
      type: Mycompany.datatypes.nfv.VnfConfigurableProperties

  substitution_mappings:
    node_type: MyCompany.ExampleVnf.1_0.1_0
    requirements:
      virtual_link: [ singleVnfcExternalCp, virtual_link ]


  node_templates:

    ExampleVnf:
      type: MyCompany.ExampleVnf.1_0.1_0
      properties:
        flavour_id : configurable
        flavour_description: A configurable flavour
        configurable_properties: { get_input: configurable_properties }
      interfaces:
        Vnflcm:
          operations:
            instantiate:
              inputs:
                configurable_parameters:
                   additional_configurable_properties:
                      parameter_1: { get_property: [SELF, configurable_properties,
additional_configurable_properties, parameter_1 ] }
                      parameter_2: { get_property: [SELF, configurable_properties,
additional_configurable_properties, parameter_2 ] }
            instantiate_end:
              implementation: ansible_configuration
      artifacts:
        ansible_configuration:
          description: script to trigger VNF configuration. This script takes as input the
configurable_parameters input of the instantiate operation
          type: tosca.artifacts.Implementation.Python
          file: /other_files/ansible/ansible.py #available in the VNF package


    singleVnfc:
      type: tosca.nodes.nfv.Vdu.Compute
      properties:
        name: singleVnfc
        description: This is the single VNFC for this VNF
        vdu_profile:
          min_number_of_instances: 1
          max_number_of_instances: 2
      capabilities:
        virtual_compute:
          properties:
            virtual_memory:
              virtual_mem_size: 4 GB
            virtual_cpu:
              num_virtual_cpu: 2
      artifacts:
        sw_image:
          type: tosca.artifacts.nfv.SwImage
          file: singleVnfc.image.v1.0.qcow2
          properties:
            name: Software for the single VNFC
            version: '1.0'
            checksum:
              algorithm: sha-256
              hash: b9c3036539fd7a5f87a1bf38eb05fdde8b556a1a7e664dbeda90ed3cd74b4f9d
            container_format: bare
            size: 2 GB

    singleVnfcExternalCp:
      type: tosca.nodes.nfv.VduCp
      properties:
        layer_protocols: [ ipv4 ]
        description: connection point on an external VL
```

```
      requirements:
        - virtual_binding: singleVnfc
        #- virtual_link: # the target node is determined in the NSD
```

A.2 VNF configuration via Kubernetes® ConfigMaps

This clause provides an example where a VNFM configures a VNF instance via a Kubernetes® ConfigMap at instantiation time (Day-0 configuration). The VNFD complies with ETSI GS NFV-SOL 001 [i.5].

The MCIOPs associated to the VNF contains a ConfigMap manifest file where one or more configuration items are declared. The ConfigMap manifest file references a values.yaml file for setting the actual configuration values. These configuration items are also modelled as VNF configurable properties.

In the VNFD, the corresponding parameters used in the values.yaml files are structured as a map whose key is the name of the node template representing the MCIOP (Helm™ chart) which includes the ConfigMap manifest file where the values are to be injected.

The specification of the inputs for the instantiate operation provides the mapping between the names of these configurable properties in the VNFD and the names of the corresponding parameters in the values.yaml file. This solution assumes that a standard extensible input named "configurable_parameters" is defined for that purpose.

In this example, the implementation associated to the preamble of the instantiate VNF operation is a shell script that is expected to create a values.yaml file named "myvalues.yaml" and make it available to the Helm™ client using e.g. a Helm™ install command.

In this example, for simplification purposes the VNF has two VNFCs with a single container and the VNFD references a single MCIOP per VNFC.

The following yaml code illustrates the contents of the VNFD. The name of the values.yaml file is provided via an input parameter of the instantiate operation.

```
tosca_definitions_version: tosca_simple_yaml_1_3

description: VNF that can be configured using K8S ConfigMap at Day-0
imports:
  - etsi_nfv_sol001_vnfd_types.yaml  # all of TOSCA VNFD types as defined in ETSI GS NFV-SOL 001

data_types:

  MyCompany.datatypes.nfv.VnfConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfConfigurableProperties
    properties:
      additional_configurable_properties:
        type: MyCompany.datatypes.nfv.VnfAdditionalConfigurableProperties

  MyCompany.datatypes.nfv.VnfAdditionalConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfAdditionalConfigurableProperties
    properties:
      conf_prop_1: # configurable property related to mciop_A
        type: string
        required: true
        default: value_1
      conf_prop_2: # configurable property related to mciop_B
        type: string
        required: true
        default: value_2

  MyCompany.datatypes.nfv.VnfConfigurableParameters:
    derived_from: tosca.datatypes.nfv.VnfConfigurableParameters
    description: This data type models the configuration parameter values to be injected in the
values.yaml files of MCIOPs.
    properties:
      parameter_names_in_values_yaml:
        type: map #key is template name of the mciop to be used to convey the configurable parameter
values to the CISM
        required: true
        entry_schema:
          type: MyCompany.datatypes.nfv.PerMciopVnfConfigurationData
```

```
MyCompany.datatypes.nfv.PerMciopVnfConfigurationData:
    derived_from: tosca.datatypes.nfv.PerMciopVnfConfigurationData
    properties:
      helm_values_file_name:
        description: the name of the values.yaml file to be created to hold the configuration values
        type: string
        required: true
        default: "values.yaml"
      vnfConfigurableParameters:
        type: MyCompany.datatypes.nfv.PerMciopVnfConfigurableParameters
        required: true

  MyCompany.datatypes.nfv.PerMciopVnfConfigurableParameters:
    derived_from: tosca.datatypes.nfv.PerMciopVnfConfigurableParameters
    description: This data type models the configuration parameter values to be injected in the
values.yaml file of an MCIOP.
    properties
      parameter_1_name_in_values_yaml:
        type: string
        required: false
      parameter_2_name_in_values_yaml:
        type: string
        required: false

node_types:

  MyCompany.ExampleVnf.1_0.1_0:
    derived_from: tosca.nodes.nfv.VNF
    properties:
      descriptor_id: b1bb0ce7-ebca-4fa7-95ed-4840d70a1177
      provider: MyCompany
      product_name: ExampleVnf
      software_version: '1.0'
      descriptor_version: '1.0'
      flavour_id: Simple
      flavour_description:  ""
      vnfm_info: [ '0:MyCompany-1.0.0' ]
      configurable_properties:
        type: MyCompany.datatypes.nfv.VnfConfigurableProperties
        description: Describes the configurable properties of the VNF
        required: false
    interfaces:
      Vnflcm:
        type: tosca.interfaces.nfv.Vnflcm
        operations:
          instantiate:
            inputs:
              configurable_parameters:
                type: MyCompany.datatypes.nfv.VnfConfigurableParameters
                required: false


topology_template:

  inputs:
    configurable_properties:
      type: MyCompany.datatypes.nfv.VnfConfigurableProperties

  substitution_mappings:
    node_type: MyCompany.ExampleVnf.1_0.1_0
    requirements:
      virtual_link: [ singleExternalCp, virtual_link ]


  node_templates:

    ExampleVnf:
      type: MyCompany.ExampleVnf.1_0.1_0
      properties:
        flavour_id: Simple
        flavour_description:  A simple VNF
        vnfm_info: [ '0:MyCompany-1.0.0' ]
        configurable_properties: { get_input: configurable_properties }
      interfaces:
        Vnflcm:
          inputs:
            helm_chart_a: { get_artifact: [ mciop_a, vnfc_a_helm, LOCAL_FILE] }
            helm_chart_b: { get_artifact: [ mciop_b, vnfc_b_helm, LOCAL_FILE] }
```

```
          operations:
            instantiate:
              inputs:
                configurable_parameters:
                  parameter_names_in_values_yaml:
                    mciop_a:
                      helm_values_file_name: myvalues.yaml
                      vnfConfigurableParameters:
                        parameter_1_name_in_values_yaml: { get_property: [SELF,
configurable_properties, additional_configurable_properties, conf_prop_1 ] }
                        parameter_2_name_in_values_yaml: { get_property: [SELF,
configurable_properties, additional_configurable_properties, conf_prop_2 ] }
                    mciop_b:
                      helm_values_file_name: myvalues.yaml
                      vnfConfigurableParameters:
                        parameter_2_name_in_values_yaml: { get_property: [SELF,
configurable_properties, additional_configurable_properties, conf_prop_2 ] }
              instantiate_start:
                implementation: k8s_day0_configuration
      artifacts:
        k8s_day0_configuration:
          description: script to create the values.yaml files based on the  configurable_parameters
input to  the instantiate operation.
          type: tosca.artifacts.Implementation.Bash
          file: scripts/k8s_config.sh  #available in the VNF package

    mciop_a:
      type: tosca.nodes.nfv.Mciop
      requirements:
        - associatedVdu: VnfcA
      artifacts:
        vnfc_a_helm:
          description: Helm Chart for vnfc_a
          type: tosca.artifacts.nfv.HelmChart
          file: HelmCharts/vnfc_a.tgz

    mciop_b:
      type: tosca.nodes.nfv.Mciop
      requirements:
        - associatedVdu: VnfcB
      artifacts:
        vnfc_b_helm:
          description: Helm Chart for vnfc_b
          type: tosca.artifacts.nfv.HelmChart
          file: HelmCharts/vnfc_b.tgz

    VnfcA:
      type: tosca.nodes.nfv.Vdu.OsContainerDeployableUnit
      properties:
        name: singleVnfc
        description: This is VNFC-A
        vdu_profile:
          min_number_of_instances: 1
          max_number_of_instances: 2
        mcio_identification_data:
          name: mcio_a
          type: Deployment
      requirements:
        - container: VnfcA_SingleContainer


    VnfcA_SingleContainer:
      type: tosca.nodes.nfv.Vdu.OsContainer
      properties:
        name: container for VNFC-A
        description: main software
        requested_cpu_resources: 100 # In Milli-Cpus, ie 0.1 CPU
        cpu_resource_limit: 1000 # In Milli-Cpus, ie 1 CPU, single threaded
        requested_memory_resources: 10 MiB
        memory_resource_limit: 100 MiB
      artifacts:
        sw_image:
          type: tosca.artifacts.nfv.SwImage
          file: Artifacts/Images/SingleContainerImage
          properties:
            name: SCI
            version: "7.3"
            checksum:
```

```
            algorithm: sha-256
            hash: a411cafee2f0f702572369da0b765e2
          container_format: docker
          size: "1024MB"


  singleExternalCp:
    type: tosca.nodes.nfv.VduCp
    properties:
      layer_protocols: [ ipv4 ]
      description: connection point on an external VL
    requirements:
      - virtual_binding: VnfcA
      #- virtual_link: # the target node is determined in the NSD


  VnfcB:
    type: tosca.nodes.nfv.Vdu.OsContainerDeployableUnit
    properties:
      name: singleVnfc
      description: This is VNFC-B
      vdu_profile:
        min_number_of_instances: 1
        max_number_of_instances: 2
      mcio_identification_data:
        name: mcio_b
        type: Deployment
    requirements:
      - container: VnfcB_SingleContainer


  VnfcB_SingleContainer:
    type: tosca.nodes.nfv.Vdu.OsContainer
    properties:
      name: container for VNFC B
      description: main software
      requested_cpu_resources: 100 # In Milli-Cpus, ie 0.1 CPU
      cpu_resource_limit: 1000 # In Milli-Cpus, ie 1 CPU, single threaded
      requested_memory_resources: 10 MiB
      memory_resource_limit: 100 MiB
    artifacts:
      sw_image:
        type: tosca.artifacts.nfv.SwImage
        file: Artifacts/Images/SingleContainerImageB
        properties:
          name: SCI
          version: "7.3"
          checksum:
            algorithm: sha-256
            hash: b411cafee2f0f702572369da0b765e2
          container_format: docker
          size: "1024MB"
```

This solution assumes that the following data types are added to the etsi_nfv_sol001_vnfd_types.yaml file defined in ETSI GS NFV-SOL 001 [i.5].

```
  tosca.datatypes.nfv.VnfConfigurableParameters:
    derived_from: tosca.datatypes.Root
    description: abstract data type from which to derive VNF-specific types to model the
configuration parameter values to be injected in the values.yaml files of MCIOPs.
    properties:
      parameter_names_in_values_yaml:
        type: map #key is template name of the mciop to be used to convey the configurable parameter
values to the CISM
        required: true
        entry_schema:
          type: tosca.datatypes.nfv.PerMciopVnfConfigurationData

  tosca.datatypes.nfv.PerMciopVnfConfigurationData:
    derived_from: tosca.datatypes.Root
    description: abstract data type from which to derive VNF-specific types to describe
configuration data per MCIOP
    properties:
      helm_values_file_name:
        description: the name of the values.yaml file to be created to hold the configuration values
        type: string
```

```
        required: true
        default: "values.yaml"
      vnfConfigurableParameters:
        type: tosca.datatypes.nfv.PerMciopVnfConfigurableParameters
        required: true

  tosca.datatypes.nfv.PerMciopVnfConfigurableParameters:
    derived_from: tosca.datatypes.Root
    description: empty base type to derive VNF-specific data type representing configuration
parameters per MCIOP
```

A.3 VNF configuration via Helm™ chart parameterization

The present annex shows a stage 3 realization of the solution described in clause 7.2.3.1.1.2 valid for a TOSCA VNFD defined according to ETSI GS NFV-SOL 001 [i.5]. The information element describing the mapping between configurable properties declared in the VNFD and names of Helm™ charts input parameters is realized as a TOSCA policy.

This solution assumes that a new policy type is specified in the ETSI NFV VNFD stage 3 specification [i.5] that allows the VNF vendor to define instances of this policy in a particular VNFD. An instance of the policy will indicate the mapping between the configurable properties and parameters in the Helm™ charts.

The following example shows how a TOSCA VNFD uses a mapping policy to indicate how to map an HTTP proxy and an HTTPS proxy to input parameters in the Helm™ charts, where the values are communicated in VnfConfigurableProperties in ETSI GS NFV-SOL 003 [i.7].

In the example the HTTP proxy and HTTPS proxy parameters are defined in a values.yaml file corresponding to a Helm™ chart named "backEnd".

The following is an excerpt of the values.yaml file corresponding to the "backEnd" Helm™ chart showing the structure of the parameter containing the HTTP proxy and HTTPS proxy:

```
HttpProxies:
    HttpProxy: ""
    HttpsProxy: ""
```

The following is an excerpt of a TOSCA VNFD showing:

- the declaration of the http_proxy and https_proxy properties as part of a vendor defined datatype derived from VnfAdditionalConfigurableProperties;
- a policy instance indicating how to map the values of the http_proxy and https_proxy to Helm™ chart input parameters.

A generic VNFM will use the information from the mapping policy in the VNFD to inject the parameter values with the correct parameter names when requesting the installation of the Helm™ chart.

```
tosca_definitions_version: tosca_simple_yaml_1_3

node_types:
 MyCompany.SunshineDB.3_0.3_0:
  derived_from: tosca.nodes.nfv.VNF
  properties:
   flavour_id:
    constraints:
     - valid_values: [ simple ]
   configurable_properties:
    type: mycompany.datatypes.nfv.VnfConfigurableProperties
#   Other properties and interface omitted for brevity

data_types:
  mycompany.datatypes.nfv.VnfConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfConfigurableProperties
    properties:
      additional_configurable_property:
        type: mycompany.datatypes.nfv.VnfAdditionalConfigurableProperties
        required: true
```

```
  mycompany.datatypes.nfv.VnfAdditionalConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfAdditionalConfigurableProperties
    properties:
      http_proxy:
        type: string
        required: true
      https_proxy:
        type: string
        required: true



topology_template:
  substitution_mappings:
    node_type: MyCompany.SunshineDB.3_0.3_0
    requirements:
      virtual_link1: [ vduCp1, virtual_link ]


  node_templates:
    sunshine_db:
      type: MyCompany.SunshineDB.3_0.3_0
      properties:
        descriptor_id: 80d695e9-7c4e-46e8-9d2c-8e000376792f
        provider: MyCompany
        product_name: SunshineDB
        software_version: '3.0'
        descriptor_version: '3.0'
        vnfm_info:
          - '0:MyCompany-1.0.0'
        flavour_id: simple
        flavour_description: 'vnf simple flavour description'

    vdu1:
      type: tosca.nodes.nfv.Vdu.OsContainerDeployableUnit
      properties:
        name: "backend"
        description: "Backend node"
        vdu_profile:
          min_number_of_instances: 1
          max_number_of_instances: 4
        mcio_identification_data:
          name: be_mcio
          type: Deployment
      requirements:
        - container: Vdu1_Container1

    Vdu1_Container1:
      type: tosca.nodes.nfv.Vdu.OsContainer
      properties:
        name: "backend Container"
        description: "Backend node"
        requested_cpu_resources: 100 # In Milli-Cpus, ie 0.1 CPU
        cpu_resource_limit: 1000 # In Milli-Cpus, ie 1 CPU, single threaded
        requested_memory_resources: 10 MiB
        memory_resource_limit: 100 MiB
      artifacts:
        sw_image:
          type: tosca.artifacts.nfv.SwImage
          file: Artifacts/Images/backendimage
          properties:
            name: backend
            version: "7.3"
            checksum:
              algorithm: sha-256
              hash: a411cafee2f0f702572369da0b765e2
            container_format: docker
            size: "1024MB"

    vduCp1:
      type: tosca.nodes.nfv.VduCp
      description: "Cluster vNIC"
      properties:
        layer_protocols: [ ethernet ]
        vnic_type: normal
      requirements:
        - virtual_binding: Vdu1
```

```
backend_mciop:
    type: tosca.nodes.nfv.Mciop
    requirements:
      - associatedVdu: Vdu1
    artifacts:
      backend_helm:
        description: Helm Chart for lb Pod
        type: tosca.artifacts.nfv.HelmChart
        file: Artifacts/Scripts/backEnd.tgz

policies:
  - ShunsineDB_mapping:
      type: tosca.policies.nfv.HelmMappings
      properties:
        configurable_properties_mappings:
          http_proxy:
            helm_chart_mapping: "backEnd:HttpProxies.HttpProxy"
          https_proxy:
            helm_chart_mapping: "backEnd:HttpProxies.HttpsProxy"
      target: sunshine_db
```

Annex B:
Change History

Date	Version	Information about changes
14-10-2021	0.0.1	Adds Table of contents and scope - NFVEVE(21)000082 General descriptions of the 3 options in clause 4 - NFVEVE(21)000098r2
03-11-2021	0.0.2	Detailed descriptions of the 3 options in clause 4 - NFVEVE(21)000104r1 + fix references to SOL002 and IFA008.
08-12-2021	0.0.3	Basic use cases and initial analysis - NFVEVE(21)000124r1
13-01-2022	0.0.4	Potential Solutions for generic key issues - NFVEVE(21)000141r1
27-01-2022	0.0.5	Potential Solutions for generic key issues for method B - NFVEVE(22)000012r2 Removal of editor's note and restructure clause 7 as per NFVEVE(22)000013
04-03-2022	0.0.6	Clause 7 - Update to potential solutions for generic key issues - NFVEVE(22)00031r1 Clause 7 - Potential Solutions for specific key issues - NFVEVE(22)000024r2 EM address configuration use case - NFVEVE(22)00035 Use case configuration distribution and storage - NFVEVE(22)000034r1 Multiple clauses consistency updates - NFVEVE(22)000033r1
04-04-2022	0.0.7	EM address config key issues - NFVEVE(22)000038r1 Addressing various TBD statements - NFVEVE(22)000054r1 Key issues for config server UC - NFVEVE(22)000047r2 + removal of an editor's note as per NFVEVE(22)000013
25-04-2022	0.0.8	Use of Ansible - NVEVE(22)000060 EM address config additional issues - NVEVE(22)000064
02-05-2022	0.0.9	Update on Method C for containerized VNFs - NFVEVE(22)000061 + removal of the editor's note in clause 6.3.1.2.1
02-06-2022	0.0.10	DHCP server configuration - NFVEVE(22)000082 Further updates on Method C for containerized VNFs - NFVEVE(22)00083 Solution when Method C is applied to containerized VNFs - NFVEVE(22)000068r1 Solutions for distribution and storage of VNF configuration data - NFVEVE(22)000065r4
04-07-2022	0.0.11	Initial Set of Recommendations - NFVEVE(22)000094r1 Annex A.2 - EN on supporting multiple helm charts - NFVEVE(22)000112r1
08-07-2022	0.0.12	VNF configuration via Helm chart parameterization - NFVEVE(22)000107r2 Fixing references - NFVEVE(22)000118 Removing GKI-C-1 - NFVEVE(22)000119
24-08-2022	0.0.13	Fixing example in clause A.2 - NFVEVE(22)000134r1 Extending solution to cover EM address configuration - NFVEVE(22)000141 Adding recommendations EM address configuration - NFVEVE(22)000142r2
29-08-2022	0.1.0	Pros and Cons Analysis of Solutions for GKI-C2 - NFVEVE(22)000128r1 Addressing ENs Configuration Server - NFVEVE(22)000143r1 Adding recommendations Configuration Server - NFVEVE(22)000144 Addressing EN on K8S configuration methods - NFVEVE(22)000152r1
10-10-2022	0.1.1	Small Technical Improvements - NFVEVE(22)000168r3 Proposed Editorial Changes -- NFVEVE(22)000167 Further pros and cons analysis for GKI-C-1 - NFVEVE(22)000156r4 Updating Configuration Server descriptions - NFVEVE(22)000172 Review issues about EM address configuration - NFVEVE(22)000178r1 Fixing Examples in clauses A.1 and A.2 - NFVEVE(22)000171 + Remove a

History

Document history		
V5.1.1	December 2022	Publication