Contents

1 About this document..........................................................................................................................4

2 What is CBAM.......................................................................................................................................5

3 VNF integration process overview.......................................................................................................7
   3.1 Project prerequisites...........................................................................................................................8
   3.2 Visualizing your VNF.......................................................................................................................10
   3.3 Writing templates.............................................................................................................................11
   3.4 Testing your VNF..............................................................................................................................13
   3.5 Productizing your VNF.....................................................................................................................14
   3.6 VNF developers' checklist................................................................................................................15

4 Template system overview....................................................................................................................17
   4.1 Lifecycle management high level architecture................................................................................17
   4.2 Lifecycle management workflows at a glance................................................................................18
   4.3 Template artifacts.............................................................................................................................19

5 ETSI NFV concepts...............................................................................................................................22
   5.1 ETSI architecture.............................................................................................................................22
   5.2 VNF Descriptor (IFA011) basic terminology....................................................................................23
1 About this document

This document is an excerpt from Template Developer Guide, DN09245073, available from Nokia. For further details, refer to that document.

Document history

Changes between document issues are cumulative. Therefore, the latest document issue contains all changes made to previous issues.
2 What is CBAM

CloudBand Application Manager (CBAM) is a VNF Manager built according to the architectural framework and specifications defined by the ETSI NFV Industry Specification Group. Therefore, a key functionality of CBAM is to perform the lifecycle management of the VNFs.

With regards to the VNF lifecycle management implementation, CBAM takes a pragmatic approach. Based on the current market and industry trends, it adopts the de facto standard and open technologies. These evolved technologies contribute to the powerful management capabilities and flexibility of Nokia’s VNFM, whereas openness is also guaranteed. Beyond standards compliance and openness, CBAM also pays attention to extensibility. For the most important lifecycle management use cases, several built-in workflow facilities are included to minimize the VNF integration / on-boarding efforts. For those VNFs that have such requirements, customized lifecycle workflows can be defined or custom commissioning protocols can be added.

Looking at CBAM from the "south" end, it creates and manages Heat stacks from Heat templates (HOTS) written by the VNF providers. Heat has more modeling power than the ETSI NFV descriptors (VNFD), it has a massive community acceptance and support, therefore, the VNF resource topology is primarily defined in the Heat templates. This lets the VNF providers use the best of Heat to meet all the special requirements they might have for organizing the resources. When the Heat stack is created, Ansible playbooks can be run for the commissioning (initial configuration, graceful shutdown) of the VNF. The resource management and commissioning operations are orchestrated in Mistral workflows.

Looking at CBAM from the "north" end, it performs VNF lifecycle management operations initiated by the NFV Orchestrator (NFVO). Such operations are, for example, "Instantiate VNF" or "Terminate VNF", which result in Heat stack operations (create stack or delete stack, respectively) in turn and involve the necessary granting handshake and lifecycle change notifications as well. The resource
requirements of the VNF and the supported lifecycle management operations are described in the VNF descriptor (VNFD) written by, again, the VNF providers. It enables the NFVO to perform the necessary capacity calculations before granting the VNFM to carry out the operations.

With regards to the northbound interfaces, CBAM selected the ETSI NFV GS-specified APIs to ensure the interoperability with the standards-compliant NFVO vendors. CBAM follows the related specification work closely and also contributes to the ETSI NFV GS standardization process.
3 VNF integration process overview

This chapter provides information about the process of setting up a development environment and creating a Virtualized Network Function (VNF) package. This process is a proposal - feel free to adopt any other process according to your needs and circumstances.

General process

![General Process Diagram]

This process is a general one, and is, by no means, the only way to develop VNF packages. Our practice shows that the incremental process is an effective one. It gives good results with short milestones and allows for an easy adoption of new data and concepts.

Related Links
- Project prerequisites on page 8
- Visualizing your VNF on page 10
- Writing templates on page 11
- Testing your VNF on page 13
- Productizing your VNF on page 14
- VNF developers’ checklist on page 15
3.1 Project prerequisites

Project management

Development of VNF templates is a development like any other, and proper project management can keep things under control, in order and on time. A good plan, with periodic sync meetings and milestone checks have proved to be very good ways of tracking the progress and making sure things are moving.

The text below gives an idea of potential milestones and goals, but it is very important to note that every VNF, VNF team and development methodology is different, so each will have its own variations in the project plan. Make sure you create a plan that suits your needs, and follow it through.

Knowledge

As VNF integrator, you should be familiar with:

- HOT (Heat Orchestration Template) syntax and concepts
- ETSI NFV concepts, especially the ones related to how VNFs are structured
- Ansible
- (optional) Mistral workflow engine

Once those are understood, you need to have a deeper understanding of this document and how all these concepts converge together.
**Lab**

To be able to develop your VNF package, you should have an OpenStack (Liberty or later) environment to work with.

CBAM is hardware agnostic, it even works on simulated setups (running on development laptops). Note, however, that such setups may be very slow.

Once the lab is ready, install CBAM. Make sure you use the latest CBAM version and the matching documentation. For more information on CBAM installation, see the *Installing CloudBand Application Manager* document.

**Development environment**

As a VNF integrator, you need an editor with indent tracking and capability of highlighting syntax of YAML files, such as VIM, Sublime, PyCharm, UltraEdit, Eclipse or XCode.

You should also have a tool for working with RESTful APIs. We recommend *Postman*. It comes as a Chrome extension, but can be installed as a standalone application, which is recommended. It allows you to prepare and save your queries and run them later several times. It also has scripting capabilities which might be used for semi-automated testing, but this is not needed for CBAM basic development.

**Testing and verifying prerequisites**

Once you have your lab ready and CBAM installed, you need to verify that it is properly installed and is ready to accept your development.

First check the state of the CBAM stack, using the instructions in the *Release Notes* and *Installing CloudBand Application Manager*.

Next try to establish an ssh connection to the CBAM instance using the private key provided for the stack creation and the user named *cbam*. This is also covered in *Installing CloudBand Application Manager*.

To iron out issues with the setup, we recommend trying to deploy MariaDB sample VNF that we're including as a sample. Follow the instructions in the *Release Notes* of each drop on how to obtain and run the MariaDB example.

If you encounter issues with any of the above items, your setup is not ready and you should fix those.

Customers can obtain from Nokia a MariaDB VNF package to look for examples of how VNFD, HOT and Ansible artifacts are written, organized and commented. That might give you ideas on how you can organize your work, though you should note that MariaDB is an "all-inclusive" example and can be a bit hard to understand at first, as it includes every possible functionality implemented in CBAM.

**Related Links**

*VNF integration process overview* on page 7
3.2 Visualizing your VNF

We recommend that you start modeling by visualizing your VNF in a tool (such as Visio or Dia) that allows you to draw a topology of the VNF and visualize the relationship of VMs, networks, ports and other entities. The sample topology diagram in the document Template Developer Guide - Appendix: MariaDB Example gives you an idea of what such a diagram may look like. Note, that this is just an example, you might choose a different approach.

If your VNF is capable of scaling, highlight the resources that are instantiated regardless of the scale (rigid part of VNF) and what are the resources that get multiplied during scaling (scaling steps). This gives you an idea of how to organize your resources in HOT files and how to describe them all in VNFD. The above example also covers that part of the visualization.

Once you have your diagrams, it is useful to highlight ETSI concepts using different colors - that would help you define VNFD and make sure you do not forget an important piece of information later on.

Finally, every VNF needs commissioning and having all MOPs and going over the commissioning steps as part of preparation is a good practice that would help you understand and plan for Ansible playbook creation work later on. Development of non-parametrized playbooks can start at this stage, in case you have separate team members that take care of Ansible only, although we recommend an iterative approach in development, to make sure the number of "moving parts" is kept in control.

Related Links

VNF integration process overview on page 7
3.3 Writing templates

Most of the work related to developing VNF templates is about understanding how a VNF is modeled and planning your work. To properly model a VNF, awareness of ETSI concepts is essential.

Key to successful VNF package is incremental work - when writing templates, you should start small and build from there. In every single step, introduce one new concept and then perform some testing. This way, new "features" can be absorbed one by one, and progress can be tracked in a predictable manner.
Implementing model in HOT

For an easy start, we suggest that you begin by creating a static HOT that will get your VNF instantiated in a default (minimal) size. To do this, you do not need to worry about parameters and everything in the HOT file(s) can be made explicit and hardcoded, including flavor names, image names, network IDs and other information. The goal at the end of this step is to have HOT file(s) that can be verified using a heat-stack tool, and without CBAM.
After this step is done and after HOT file(s) are made and verified, you can switch to parameterizing the HOTs. CBAM provides a series of parameters that determine how your VNF gets instantiated, which include exact flavor name to be used, image name, dynamically created networks and so on. A list of CBAM parameters is described in CBAM Template guides. Parametrization is important because it allows your VNF to be customized to the environment where it would get deployed, and those parameters are calculated and provided by CBAM.

As a next step you need to consider how you want to implement scaling on Heat level: which resources need to be replicated together based on scaling attributes and place them into a resource group.

**Writing VNFD**

When you create your first VNFD, start with grantless mode (the scenario where NFVO is not used). Without grant, the most important part of your VNFD is the heat_mapping, and the first goal to achieve is to manage to instantiate the stack via CBAM.

The second goal is being able to successfully run a sample ansible playbook against all the VDUs that you are targeting. Only after these should you start implementing other parts of your VNFD in line with your now (close to) complete HOT files and in line with the ETSI requirements. CBAM only allows the usage of valid VNFD templates. For this reason, your template must always be valid and consistent even if it is incomplete.

**Creating a playbook**

Playbooks are just loosely coupled with the other templates, so you can develop them even before starting your work on the HOT files, but it is recommended to have at least the final topology of your VNFD done. Moving connection points around can have effect on your playbooks.

Do consider the CBAM generated ansible inventory though when you are writing your playbook. For this reason it might be a good practice to finish your playbooks as the last step.

**Related Links**

*VNF integration process overview* on page 7
3.4 Testing your VNF

When you are testing your topology description (whether the HOT files directly using Heat or your CBAM template package via CBAM), your focus should be VNF topology (and not VNF functionality), so you should use dummy (i.e. empty) playbooks, dummy images (such as plain Ubuntu or Cirros images) and smaller volumes. Instantiation, termination and scaling is much faster while you are still able to check whether the appropriate resources have been deployed and the interconnections are established as expected (like network setup). This applies to initial instantiation and scaling as well. Once your topology behaves as you expect, you can switch to the real images and playbooks to see if configuration works as well.

**Related Links**

*VNF integration process overview* on page 7

3.5 Productizing your VNF

To declare your product fully integrated with CBAM, you should have gone through all the steps in this chapter. You should have tested your VNF package with all the configurations, all the environments and all the scenarios (like blade failure, upgrade) that your VNF supports. If you used timeouts anywhere in your templates you should consider how the LCM operations will behave on a heavily-used infrastructure.

You should also test your template package with all the VIMs you plan to support (such as CBIS, RDO Liberty, NCIO and so on). For each, you should measure KPIs related to your VNFs in order to provide performance statement for the combination you will officially support (VNF + CBAM + VIM + Hardware). Expect functional and performance differences due to limitations and variations of different VIM and Hardware platforms.

**Related Links**
## 3.6 VNF developers' checklist

<table>
<thead>
<tr>
<th>Prerequisites - knowledge</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HOT</strong></td>
<td>Are you familiar with overall HOT syntax, Nova, Neutron and Cinder ResourceTypes, intrinsic functions?</td>
</tr>
<tr>
<td><strong>ETSI VNF</strong></td>
<td>Do you know how VNF structure is mapped, what is VNFC, VDU and how ScalingAspects are defined?</td>
</tr>
<tr>
<td><strong>TOSCA profile</strong></td>
<td>Have you seen sample TOSCA VNFD and know how to map your VNF data into it?</td>
</tr>
<tr>
<td><strong>Ansible</strong></td>
<td>Do you know how to write a playbook to contact your VNF and provide initial commissioning?</td>
</tr>
<tr>
<td><strong>Mistral</strong></td>
<td>(optional) Do you know how to describe VNF specific actions in Mistral language?</td>
</tr>
<tr>
<td><strong>CBAM template guide</strong></td>
<td>Have you read and understood CBAM template guide to understand how everything comes together?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prerequisite - lab and development tools</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OpenStack environment</strong></td>
<td>Do you have OpenStack Liberty or later environment that you can use? Does it have separate tenant credentials allocated for your development?</td>
</tr>
<tr>
<td><strong>CBAM</strong></td>
<td>Did you download and install latest CBAM drop?</td>
</tr>
<tr>
<td><strong>Editor</strong></td>
<td>Do you have editor of your choice, with YAML highlighting capabilities?</td>
</tr>
<tr>
<td><strong>Postman</strong></td>
<td>Did you install tool for interfacing APIs?</td>
</tr>
<tr>
<td><strong>Test of MariaDB</strong></td>
<td>Can you operate MariaDB in your environment, using your APIs?</td>
</tr>
<tr>
<td>Writing templates</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>VNF model</strong></td>
<td>Drawing of a VNF topology, networks, ports, with highlighted rigid and scalable groups, external connection points</td>
</tr>
<tr>
<td><strong>Static HOT</strong></td>
<td>HOT files implementing topology in a static way, no parametrization</td>
</tr>
<tr>
<td><strong>VNFD for instantiate</strong></td>
<td>VNFD for static HOT file for instantiation only.</td>
</tr>
<tr>
<td><strong>Test: Basic instantiation</strong></td>
<td>Can you instantiate your VNF with default size, without parameters (both with and without CBAM)</td>
</tr>
<tr>
<td><strong>HOT with parameters</strong></td>
<td>Parametrized HOT file, implementing CBAM data structure</td>
</tr>
<tr>
<td><strong>Ansible for commissioning</strong></td>
<td>Self contained playbook, its only inputs are the Ansible inventory and extra vars generated by CBAM</td>
</tr>
<tr>
<td><strong>Test: Parametrized instantiation</strong></td>
<td>Can you instantiate your VNF using parameters, using CBAM?</td>
</tr>
<tr>
<td><strong>VNFD/HOT for scaling</strong></td>
<td>The heat stack should be able to scale without any information form the VNFD</td>
</tr>
<tr>
<td><strong>Ansible for scaling</strong></td>
<td>Same as for commissioning but directed to those VMs that are affected by the scaling (new nodes, management nodes...)</td>
</tr>
<tr>
<td><strong>Test: Scaling VNF</strong></td>
<td>Can you scale your VNF?</td>
</tr>
<tr>
<td><strong>Test: Complete VNF testing</strong></td>
<td>Can you perform all functions together?</td>
</tr>
<tr>
<td><strong>Healing</strong></td>
<td>Custom workflow capable of restoring one or more VMs (VDUs) of the VNF</td>
</tr>
</tbody>
</table>

**Related Links**

*VNF integration process overview* on page 7
4 Template system overview

4.1 Lifecycle management high level architecture

VNF lifecycle management comprises a set of different but still interrelated functional areas:

- Management of the structure / topology of the VNF and coordination of the VNF level lifecycle operations execution.
- Management of the VIM resources as a part of the executed lifecycle operations.
- Performing initial configuration / commissioning of the VNF software as well as performing other software configuration tasks as a part of the lifecycle operations, such as the graceful shutdown of VNFCs.

![Figure 6: CBAM Life-cycle Execution Architecture](image-url)
These functional areas are handled as separate functional blocks within the CBAM architecture:

<table>
<thead>
<tr>
<th>Functional area</th>
<th>CBAM component</th>
</tr>
</thead>
<tbody>
<tr>
<td>VNF commissioning and general software configuration</td>
<td>Ansible</td>
</tr>
<tr>
<td>VIM resource management</td>
<td>OpenStack Heat</td>
</tr>
</tbody>
</table>
| VNF topology management and the execution of the lifecycle operations            | **VNF Topology and Life-cycle Manager (TLM)** controls the overall VNF topology and orchestrates the VNF lifecycle management workflows across the participating components.  
**Mistral** used as a workflow execution engine it is responsible for performing the actual steps of the workflows, providing visibility and control over the execution process.  
**VNF Inventory** stores the persistent VNF models and makes it accessible for the lifecycle workflow orchestration performed by TLM and for the actual workflow execution steps carried out in Mistral. |

There are certain overlaps between the responsibilities of CBAM components:

- OpenStack Heat also provides various means for managing the structure of the VNFs (for example, through the Heat ResourceGroup mechanism), which overlaps with the CBAM TLM functionality. In this respect, TLM is considered to provide the internal orchestration responsibility, whereas Heat stores the low-level model and performs the actual execution. Beyond that, TLM handles the topology management according to the ETSI NFV-defined VNF structural model, while Heat brings its own model.

- Initial configuration of the VNFs may also be performed via Heat, using the simple userdata mechanism or the more structured SoftwareConfig, SoftwareDeployment and related mechanisms. CBAM includes Ansible for this purpose, and you should use it whenever possible.
4.2 Lifecycle management workflows at a glance

Lifecycle management (LCM) workflows typically:

- process input parameters
- send 'start lifecycle change' notification to the NFVO and other interested parties
- request grant from the NFVO with additional VIM specific parameters
- receive grant from the NFVO with additional VIM specific parameter values
- manipulate the Heat stack
- perform initial configuration/commissioning of the VNF
- register or update the VNF-related information in the EMS
- send 'result lifecycle change' notifications to the NFVO and other interested parties.
4.3 Template artifacts

VNF Descriptor (VNFD) is the root template in the VNF template package. It describes the topology of the VNF by means of ETSI NFV (phase 2) terms such as VDUs, Connection Points, Virtual Links, External Connection Points, Scaling Aspects, Instantiation Levels and Deployment Flavours.

It is TOSCA / YAML based and it is faithfully modeling the VNFD information model defined by the respective ETSI NFV GS phase 2 IFA 011 specification. For more information, see [https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=44577](https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=44577)

VNFD extensions

The VNFD extensions define the lifecycle management (LCM) operations of the VNF by referring to the necessary template artifacts, such as Heat templates, Mistral workflow templates, Ansible playbooks and customization scripts (JavaScript). The mapping between ETSI NFV modeling elements and the corresponding Heat resources is also defined here.

Just like the VNFD, it is TOSCA / YAML based, however the utilized TOSCA type system is specific to CBAM.

Heat template (HOT)

The Heat template (HOT) is the primary source of VNF topology definition in OpenStack. The cloud resources and their relations are all described in the Heat template, authored by the VNF Provider. The corresponding information elements in the VNFD must be consistent with the definitions in the Heat
Introducing the Template Developer Guide

Template system overview

HOT files are described in YAML according to the official OpenStack provided schema. For more information on Heat templates, see [http://docs.openstack.org/developer/heat/template_guide/hot_spec.html](http://docs.openstack.org/developer/heat/template_guide/hot_spec.html).

Mistral workflow template

Lifecycle Management (LCM) operations are run as Mistral workflows and are defined in workflow templates.

The typical LCM workflows (instantiate, scale, terminate) are provided as part of CBAM (reside in the Mistral workflow library) and can be used "as is" by the VNF Providers. Hence in many cases, there is no Mistral workflow template in the VNF template package. These prewritten workflows are extendable with custom configuration actions (Ansible playbooks) and data processing scripts (JavaScript for advanced parameter processing). For providing additional advanced life-cycle operations or for providing alternative VNF specific implementations, custom workflows can be added, which may use CBAM's extensive Mistral action library.

Mistral workflows are described in YAML files according to the official OpenStack Mistral project provided schema. For more information on Mistral workflows, see [http://docs.openstack.org/developer/mistral/dsl/dsl_v2.html](http://docs.openstack.org/developer/mistral/dsl/dsl_v2.html).

Ansible playbook

Ansible playbooks can be used in the LCM workflows for VNF configuration / commissioning tasks.

Ansible playbooks are YAML files, according to the official Ansible schemas. For more information on Ansible, see [https://www.ansible.com/](https://www.ansible.com/)

Customization script (JS)

Customization scripts written in JavaScript can be used in the LCM workflows for customization and advanced parameter processing.
5 ETSI NFV concepts

5.1 ETSI architecture

The NFV architecture consists of three layers:

1. VIM deals with the lowest level of abstraction by managing the virtualized resources, such as Virtualized Compute (VM), virtualized Networking and virtualized Storage.
2. VNFM deals with VNF instance management, that is, lifecycle management of VNF instances.
3. NFVO deals with Network Service management and overall management of resource use.

The complete NFV functionality is designed around the above described layering principle.

The following figure illustrates the ETSI NFV specifications relevant to each reference point:
5.2 VNF Descriptor (IFA011) basic terminology

ETSI NFV provides two views of the Information Model:

- Template view, which describes the model of the descriptors
- Logical view, which describes the runtime view of the VNF/NS instances

In the figure above gray class represents the logical view, also known as, the run-time model of the VNF, while the blue colored classes represent the template view, the descriptors.

VNF is existing only in the logical view, which represents the VNF instance, once instantiated by the VNFM. The corresponding class in the template view is the VNFD as VNF is instantiated (deployed) using the VNFD.
A VNF may be created by multiple components. These are called VNF Components (VNFC). VNFC exists also only in the logical view. The corresponding class of the VNFC in the template view is the VDU (Virtualized Deployment Unit). Therefore VNFC is instantiated (deployed) by using the VDU.

**Generic-VNFM**

ETSI NFV has defined two types of VNF Managers. One of them is called a Generic-VNFM (G-VNFM). G-VNFM provides a possibility to execute LCM scripts defined in Domain Specific Language (DSL). What DSL can be is not specified in rel-2. The G-VNFM, besides providing the LCM script execution environment, has a standardized interface towards the VNF and EM, that is, the Ve-Vnfm-em and Ve-Vnfm-vnf interfaces are fully standardized and those interfaces are used for communication towards the VNF/EM.

A G-VNF can rely only on those information from the VNF/EM that is defined in IFA008 and IFA011 specifications. The indicator interface can carry information from the VNF application state. What information and when is delivered by the VNF or EM is defined in the VNFD.

**Specific-VNFM**

Specific-VNFM (S-VNFM) is the other type of VNFM that is specified in ETSI NFV. The difference to the G-VNFM is that the interface to the VNF/EM (that is, the Ve-Vnfm-em and Ve-Vnfm-vnf reference points) can be proprietary as an S-VNFM and the VNF it manages are both produced by the same vendor. The lifecycle management of the VNFs are not necessarily based on the DSL defined LCM scripts, but can be defined in a vendor-specific way. The Or-Vnfm and the Vnfm-Vi reference points are still the standard defined ones.

In a specific VNFM it is possible to get whatever information the VNFM needs from the VNF application (such as: PM counters or alarms that may be used for scaling purposes).

**VDU (Virtualized Deployment Unit)**

VDU is the template that describes the resource needs of a VNFC when a VNFC is deployed.

**Deployment Flavour**

It specifies a given deployment configuration of a VNF in terms of its internal topology and resource needs. The external topology of a Deployment Flavour (DF) is fixed. This means that the amount of external connections of one DF is fixed and cannot be changed. In case of a need for scaling this means that the connections need to be reserved and when there is no need for it the connection remains there, but is not used by the VNF.

There can be multiple DFs defined in one VNFD. VNF Scaling operates within the boundaries of a DF and therefore this shall be non service-impacting.

Changes between DFs are also possible, the operation is called Deployment Flavour Switching. This operation may be service-impacting (there is no requirement for it to be non-service impacting) due to the possibility of having different topologies defined in different DFs.
VDU profile

VDU Profile describes the boundaries of the DF in terms of the minimum and maximum number of VNFC instances that can be deployed. The minimum and maximum number of VNFC instances given in the VDU Profile gives the boundaries also for the VNF scaling.

Instantiation level

Instantiation level describes a particular VNF configuration (number of VNFC instances created per VDU) for instantiation. The VNFD may contain multiple instantiation level definitions. One of the instantiation levels is defined as default.

During VNF instantiation it is possible to provide one of the instantiation levels as input for the operation. This enables different VNF sizes at instantiation; that is, it is possible to instantiate a VNF in a middle size, instead of instantiating it for the minimum and then scaling up. This enables to achieve the requested VNF capacity in one step.

To better understand the meaning of instantiation levels the figure above describes the possible VNF configurations where there are two types of VNFCs (2 VDUs).

It is up to each vendor to decide on the number of instantiation level definitions. It is possible to define all possible configurations or only a subset of them. That is, it is possible to define all the gray points as instantiation levels, but it is also possible that only the red, yellow and blue points are defined as instantiation levels. However at least one instantiation level needs to be defined. This one instantiation level can act as a default instantiation level during VNF instantiation.

Scaling aspect

Scaling aspect is the same as the scaling dimension term used in CAM. It describes to what scale in horizontal scaling in an abstracted way.

In VNFD, it may be optionally defined by referencing a group of VNFD elements (VDU, VL) for example, Scaling of call processing aspect may require scaling of two different VDUs.

VnfElementGroup class is not needed for VNFs to be managed by S-VNFM but is required for VNFs to be managed by G-VNFM.

Scale level

Each scale level defines a valid size of the VNF with regard to the associated scaling aspect. Scaling takes place in discrete steps, changing the size from one level to another one.
Introducing the Template Developer Guide

ETSIs NFV concepts

There is a relationship between the scaling levels and instantiation levels. In scaling space, an instantiation level is represented by a tuple of scale levels, one per scaling aspect. This allows to know in which aspects scale-in/out is possible from a given instantiation level.

**External Connectivity**

ETSI NFV has defined two ways for connecting a VNF to an external network.

Either an internal CPD (Inter Connection Point Descriptor) of a VDU is exposed as external CPD or an internal VLD is exposed as external connection.

In the logical view this means that an external connection point is either a port of a VNFC or a port of an internal Virtual Link.

**VNF Indicator**

VNF indicators are declared by the VNF vendor in the VNFD. Indicators can tell information about the VNF application state. A VNF or its EM can send notification to the VNFM when a specific criterion is met. For example, if the VNF is in overload situation or if some specific counter reaches a threshold. The condition on when to send the notification and what content to send is up to the VNF vendor and is declared in the VNFD.

The notification and the way how the Indicator value changes are sent from VNF/EM to the VNFM is described in the VNF Indicator interface in the IFA008 specification. The VNF indicators then can be re-exposed to the NFVO by the VNFM.

**Performance management**

There is a clear separation between virtualization layer parameter configuration and application layer parameter configuration. Application layer parameter configuration is the task of the traditional management system, like the EM. In Nokia case it is the NetAct/SAM.

The virtualization layer parameter configuration is the task of the VNFM. There are several locations in VNFD for different configurable properties. The VNFD can contain information for example, what
Introducing the Template Developer Guide

ETSI NFV concepts

different features the VNF is capable of, what LCM operations the VNF supports, parameters for each VNF LCM operation (VNFD:DeploymentFlavour:VnfLcmOperationsConfiguration).

There are parameters defined in the VNFD that reach the VNF and can be modified by the VNFM by the ModifyVnfConfiguration operation (VNFD:configurableProperties) and parameters that do not reach the VNF, but changes information in the VNFM about the specific VNF (VnfInfoModifiableAttributes). This latter one can be modified via the ModifyVnfInfo operation.

**VNFD model**

*Figure 10: VNFD Model*
Introducing the Template Developer Guide

ETSNI NFV concepts

Figure 11: VNF Deployment Flavour view

Figure 12: VDU view
Introducing the Template Developer Guide

ETSI NFV concepts

Figure 13: VNF logical model

Related Links

ETSI architecture on page 22

http://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/
- published ETSI NFV IFA specification versions
https://docbox.etsi.org/ISG/NFV/Open/Drafts
- draft ETSI NFV IFA specification versions
https://docbox.etsi.org/ISG/NFV/Open/Drafts/IFA011_VNF_Packaging_Spec
- NFV IFA011 draft specification
https://docbox.etsi.org/ISG/NFV/Open/Drafts/IFA015_NFV_Information_Model
- the ETSI NFV Information model is available in electronic format in Papyrus in IFA015 specification.

Note: IFA015, containing the overall consolidated NFV Information Model, is an informative specification.

- IFA016 Papyrus guidelines on installing and opening the NFV Information Model