VNF Manager Framework

Ve-VNFM Interface Specification

VNFM-ARCH-INFSPEC- Ve-VNFM-435200003

435200003 DRAFT

Revision 1.0

Copyright © 2006-2017 Aricent. All Rights Reserved. No part of this document may be reproduced, stored in a retrieval system, distributed or transmitted, in any form, or by any means, electronic or otherwise, including photocopying, reprinting, or recording, for any purpose, without the express prior written permission of The Aricent Group.

Printed in \_\_\_\_\_\_\_\_

TRADEMARKS:  Aricent and the Aricent Logo are trademarks and registered trademarks of  TheAricent Group or its subsidiaries or affiliates in the U.S. and certain other countries.  The use of these trademarks without Aricent’s  prior written consent is strictly prohibited. Other third party trademarks referenced are the property of their respective owners.  Please note that the above list is not all-inclusive, and the absence of any mark from this list does not mean that it is not an Aricent trademark.

DISCLAIMER The information in this book is provided “AS - IS”, without warranty of any kind, express or implied, including but not limited to a warranty of merchantability, fitness for any particular purpose, title or non-infringement of third party rights, and Aricent specifically disclaims all other warranties, conditions or representations. This document is provided for informational purposes only, and should not be construed as a representation or a commitment on the part of Aricent.  Information in this document is subject to change without notice.

REQUESTS For information requests or for obtaining permission for the use of this work, please submit a written request to: Corporate Marketing and Legal, The Aricent Group, 303 Twin Dolphin Drive, Suite 600 Redwood City, CA 94065

DOCUMENT No.: 435200003

**Contents**

|  |
| --- |
| *Ve-VNFM Interface Specification* |

[1. Overview 5](#_Toc453860307)

[1.1 Ve-VNFM Interface 5](#_Toc453860308)

[1.2 References 5](#_Toc453860309)

[2. Ve-VNFM Lifecycle Management 6](#_Toc453860310)

[2.1 Ve-VNFM Lifecycle Management Architecture 6](#_Toc453860311)

[2.2 Lifecycle Events 7](#_Toc453860312)

[2.3 End Point Broker (EPB) Library Requirements 7](#_Toc453860313)

[2.3.1 Prototype of update\_vdu\_management\_interface Function 8](#_Toc453860314)

[3. Ve-VNFM Lifecycle Management Interface 10](#_Toc453860315)

[3.1 Metadata Components 10](#_Toc453860316)

[3.2 Defining Metadata 10](#_Toc453860317)

[3.2.1 VDU LCM Metadata – vdu-lcm-data 10](#_Toc453860318)

[3.2.2 States – vdu-lcm-data::states 10](#_Toc453860319)

[3.2.3 Vendor Defined Functions – vd-functions 10](#_Toc453860320)

[3.2.4 Function – vd-functions::function-interface 10](#_Toc453860321)

[3.2.5 Events – vdu-lcm-data::events 11](#_Toc453860322)

[3.3 Broker Function 11](#_Toc453860323)

[3.3.1 VDU Management Interface 11](#_Toc453860324)

[3.3.2 vnfc\_connection\_points 12](#_Toc453860325)

[3.3.3 connection\_point 12](#_Toc453860326)

[3.4 Invoking Inter-EPB Function Calls 12](#_Toc453860327)

[3.4.1 VDU connection point interface 12](#_Toc453860328)

[3.4.2 self\_instances 12](#_Toc453860329)

[3.4.3 depends\_on\_vdu\_instances 13](#_Toc453860330)

[3.5 Default States 13](#_Toc453860331)

[3.6 Sample Metadata 13](#_Toc453860332)

[4. Ve-VNFM Monitoring Interface 15](#_Toc453860333)

[4.1 Architecture 15](#_Toc453860334)

[4.2 Measurement Requests over Ve-VNFM 16](#_Toc453860335)

[4.2.1 Monitoring Parameter in Descriptors (VNFd, NSd, VDU) 16](#_Toc453860336)

[4.2.2 Event 17](#_Toc453860337)

[4.2.3 Health Monitoring Parameter in Descriptors (VNFd, NSd, VDU) 17](#_Toc453860338)

[4.2.4 Defining Measurement Requests in LCM Metadata 18](#_Toc453860339)

[4.3 Measurement Reports from VNF to VNFM over Ve-VNFM 18](#_Toc453860340)

[4.3.1 Measurement Report – measurement\_report 18](#_Toc453860341)

[4.3.2 Monitoring Report - measurement\_report::monitoring\_report 19](#_Toc453860342)

[4.3.3 Health Monitoring Report 19](#_Toc453860343)

[4.3.4 RESTful API for VNFs to send Measurement Reports to VNFM 19](#_Toc453860344)

[4.3.4.1 Sending Measurement Reports - sendMeasReport 19](#_Toc453860345)

[4.4 Defining Actions on Monitoring Reports through Policies 20](#_Toc453860346)

[5. Revision History 22](#_Toc453860347)

# Overview

This document gives an overview of the Ve-VNFM interface and the definition of the Lifecycle management metadata through which the VNF vendor can define customized lifecycle including state machines, events and actions.

The terms of reference in this document can be referred from [ETSI GS NFV 03]

## Ve-VNFM Interface

The Ve-VNFM interface is the interface between VNF and the VNFM. Through this interface complex lifecycle management procedures can be carried out by the VNFM for the VNF. This is done as per the VNF state and the VNF or user events. This could be VNF events, Virtual Infrastructure events and user actions.

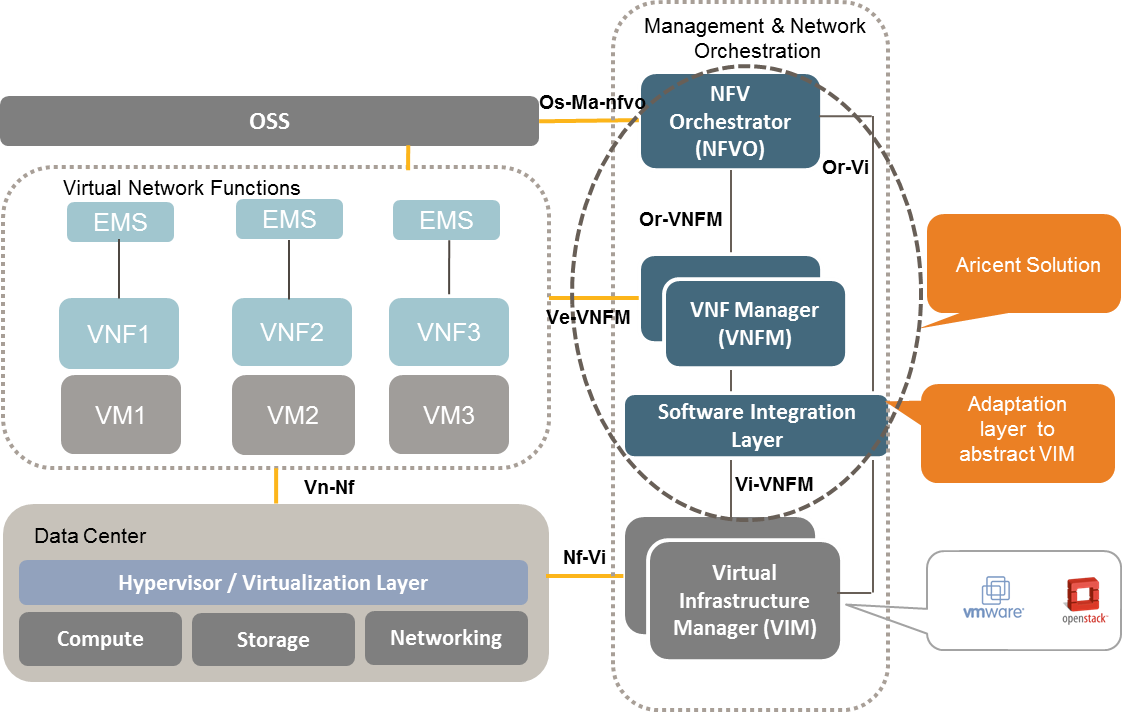


Figure 1 NFV Management and Network Orchestration Architecture and Interfaces

## References

The following external documents were used in preparing this document, and may be referenced at various points.

[ETSI MANO] ETSI GS NFV-MAN 001 *“Network Functions Virtualisation (NFV); Management and Orchestration” v*

[ETSI GS NFV 03] ETSI Group Specification ETSI GS NFV 003 *“Network Function Virtualization; Terminology for main concepts in NFV”*

[VNFM Arch] Aricent VNF Manager *“Architecture and Requirement Specification”*

# Ve-VNFM Lifecycle Management

The VNFM Manager Framework is an implementation of the VNF Manager in the NFV Management and Network Orchestration framework as defined by ETSI. Refer [ETSI MANO]

The overall architecture of the VNF Manager Framework is defined in the product architecture specification. Refer [VNFM Arch]

## Ve-VNFM Lifecycle Management Architecture

Through the Ve-VNFM interface, the VNF Manager performs lifecycle management (LCM) of the VNFs. The architecture of the Ve-VNFM interface is given in the below picture

<mxfile userAgent="Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.63 Safari/537.36" version="5.5.1.3" editor="www.draw.io" type="device"><diagram></diagram></mxfile>

Figure 2 VNF Lifecycle Management Architecture

VNFD Manager implements the VNF Management functionality (details in [VNFM Arch]). The VNF Lifecycle Manager (VNF LCM) manages the Ve-VNFM control interfaces for all VNFs. VNF LCM maintains the list of VNFs in the VNF table.

For each VNF, there can be multiple virtual deployment units (VDUs). Each VDU may contain multiple VNF components which are characterized by their unique connection points. Each VDU corresponds to one virtual machine.

The VNF LCM spawns a worker thread whenever there is a lifecycle event for a specific VDU. The Lifecycle management is defined by the VNF vendor through a metadata at the VDU level.

The protocol used by the end point broker library is decided by the VNF vendor and that implementation (client) needs to be part of the library. If REST is chosen, then REST client should be packaged as part of the library, if NETCONF is chosen then a similar client needs to be part of the library. If SSH is the chosen method, then the library should be able to execute scripts after opening SSH sessions with the VNF using a SSH client packaged as part of the library.

## Lifecycle Events

**VDU Level Event:** In case there is a VDU level event, the event will also reach VNFD Manager and it will send the same event to the VNF Lifecycle Manager (corresponding VNF id and VDU-id will be part of the REST message). The VNF lifecycle manager then spawns the worker threads for the specified VDU to perform the VDU level lifecycle event.

**VNF Level Event**: In case there is a VNF level event, the event will also reach VNFD Manager and it will send the same event to the VNF Lifecycle Manager (corresponding VNF id will be part of the REST message). The VNF lifecycle manager then performs the operations for each VDU which are part of the VNF as defined in the VDU level event.

Note: For a VNF level lifecycle event, the corresponding VDU level lifecycle event must be defined in the Metadata

**Network Service Level Event:** In case there is a NS level event, the event will reach the VNFD Manager. The VNFD Manager will send the same event individually to the VNF Lifecycle Manager (VNF LCM) for all VNFs defined in the network service (one REST message per VNF). The VNF Lifecycle Manager will handle individual VNF level REST requests as defined in the VNF level event

Note: For a NS level lifecycle event, the corresponding VDU level lifecycle event must be defined in the Metadata

## End Point Broker (EPB) Library Requirements

The following requirements are specified for the Broker Library Development

1. All classes / packages / functions defined in the metadata should be implemented with proper error handling
2. The worker thread will catch all the exceptions thrown by the library and handle the exceptions. The VNF Manager will send the exception cause to the event origin. It is therefore necessary for the EPB libraries to throw exceptions in case of failure. The stack trace will be dumped in a log file.
3. All the functions defined in the metadata should return bool
4. Functions called by VNFLCM should have following prototype
   1. public bool <function> (arg1, arg2) throws Exception
      1. Where arg1 (VDU management interfaces). Refer 3.3.1
      2. And arg2 is the argument mentioned in metadata (String)
5. The protocol used by the end point broker library is decided by the VNF vendor and that implementation (client) needs to be part of the library. If REST is chosen, then REST client should be packaged as part of the library, if NETCONF is chosen then a similar client needs to be part of the library. If SSH is the chosen method, then the library should be able to execute scripts after opening SSH sessions with the VNF using a SSH client packaged as part of the library.
6. In the EPB library one mandatory function needs to be implemented called update\_vdu\_management\_interface with only argument1, through which VNF Manager can update VDU management information

### Prototype of update\_vdu\_management\_interface Function

The prototype of the mandatory EPB function for updating VDU management information is

public bool update\_vdu\_management\_interface (arg1) throws Exception (Where arg1 (VDU management interfaces). Refer 3.3.1)

**Class Definition for argument 1**

public enum supported\_protocol {

REST, NETCONF

}

public class connection\_point

{

private String connectionPointId;

private InetAddress cpIPAddress;

private int cpPort;

public void setConnectionPointId (String cpId)

{connectionPointId = cpId;}

public void setCpIPAddress (InetAddress addr)

{cpIPAddress = addr;}

public void setCpPort (int port)

{cpPort = port;}

public String getConnectionPointId ()

{return connectionPointId ;}

public InetAddress getCpIPAddress ()

{return cpIPAddress ;}

public int getCpPort ()

{return cpPort ;}

}

public class vnfc\_connection\_points

{

private String vnfcId;

private connection\_point connectionPoint;

public void setVnfcId (String vnfc\_Id)

{vnfcId = vnfc\_Id;}

public void setConnectionPoint (connection\_point cp)

{connectionPoint = cp;}

public String getVnfcId ()

{return vnfcId ;}

public connection\_point getConnectionPoint ()

{return connectionPoint;}

}

public class VDUSpecificArgument

{

private InetAddress vnfLcmIPAddress ;

private int vnfLcmPort;

private InetAddress monIPAddress ;

private int monPort;

private supported\_protocol monMethod;

private vnfc\_connection\_points vcp;

public void setVnfLcmIPAddress (InetAddress addr)

{vnfLcmIPAddress = addr;}

public void setVnfLcmPort (int port)

{vnfLcmPort = port;}

public void setMonIPAddress (InetAddress addr)

{monIPAddress = addr;}

public void setMonPort (int port)

{monPort = port;}

public void setMonMethod(supported\_protocol method)

{monMethod = method;}

public void setVcp (vnfc\_connection\_points cp)

{vcp = cp;}

public InetAddress getVnfLcmIPAddress ( )

{return vnfLcmIPAddress ;}

public int getVnfLcmPort ( )

{return vnfLcmPort ;}

public InetAddress getMonIPAddress ( )

{return monIPAddress ;}

public int getMonPort ( )

{return monPort ;}

public supported\_protocol getMonMethod( )

{return monMethod ;}

public vnfc\_connection\_points getVcp ( )

{return vcp ;}

}

# Ve-VNFM Lifecycle Management Interface

## Metadata Components

VNF vendors can define the lifecycle states, events and actions through the LCM metadata. The keywords for the metadata are given in the next table:

## Defining Metadata

There is a need to define one metadata file per VDU.

### VDU LCM Metadata – vdu-lcm-data

|  |  |  |  |
| --- | --- | --- | --- |
| Keyword | Type | Cardinality | Description |
| id | String | 1 | Unique identifier for a virtual descriptor unit (VDU) as defined in the VNF Descriptor |
| communication | Bool (bi=0, uni=1) | 1 | Bidirectional / Unidirectional i.e. BI / UNI direction communication between VNF and VNFM |
| states | Structure | 1..n | Lifecycle Management Definition |
| delay | Value | 1 | In Milliseconds. Time to wait for the responses between VNF and VNFM to arrive |

### States – vdu-lcm-data::states

|  |  |  |  |
| --- | --- | --- | --- |
| Keyword | Type | Cardinality | Description |
| name | state-name | 1 | Name of the VNF state |
| init-state-function | vd-functions | 0..1 | Any function that needs to be executed when entering the state. This is optional  Refer for datatype vd-function 3.2.3 |
| events | Structure | 1..n | Events handled within given state and its corresponding state transition |

### Vendor Defined Functions – vd-functions

|  |  |  |  |
| --- | --- | --- | --- |
| Keyword | Type | Cardinality | Description |
| package-name | String | 1 | Name of the JAVA package where definition of function can be found |
| function-interface | function | 1..n | If init-state is defined, at-least 1 function needs to be defined. |

### Function – vd-functions::function-interface

|  |  |  |  |
| --- | --- | --- | --- |
| Keyword | Type | Cardinality | Description |
| function-name | String | 1 | Name of the function |
| function-argument | String | 1 | String that needs to be passed as argument to the init-state-function  It is recommended to use a JSON string as the argument. (but not mandatory)  Note: This is arg2 as defined in 4 |

### Events – vdu-lcm-data::events

|  |  |  |  |
| --- | --- | --- | --- |
| Keyword | Type | Cardinality | Description |
| event-name | String | 1 | String identifier of the event |
| next-success-state | state-name | 1 | Next state if the event is handled successfully |
| next-failure-state | state-name | 1 | Next state if there is failure to handle the event |
| actions | vd-functions | 1..n | Action to be taken when the event occurs. |

## Broker Function

The broker function (function-name) will have two arguments.

* Argument 1 VDU Management Interface: This will contain relevant information generated by the VNF Manager for the EPB to communicate with the VNF
* Argument 2: As defined in the metadata (function-argument)

### VDU Management Interface

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Cardinality | Description |
| vnf\_lcm\_ip\_address | InetAddress | 1 | VNF LCM Self IP Address |
| vnf\_lcm\_port | U32 | 1 | VNF LCM Self Port for receiving REST message in case of bi-directional communication with VNF |
| mon\_ip\_address | InetAddress | 1 | IP Address of the module where VDU (VNF components) should send their monitoring data. |
| mon\_port | U32 | 1 | Port of the Monitoring Module |
| mon\_method | enum | 1 | {REST, NETCONF} |
| vnfc\_connection\_points | Structure | 1..n | List of connection points for the VNF components |

### vnfc\_connection\_points

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Cardinality | Description |
| vnfc\_id | String | 1 | VNFC id |
| connection\_point | Structure | 1..n | List of connection point for specific VNFC |

### connection\_point

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Cardinality | Description |
| connection\_point\_Id | String | 1 | Connection point id |
| cp\_ip\_address | InetAddress | 1 | IP Address of the connection point |
| cp\_port | U32 | 1 | Virtual or Physical NIC’s port |

## Invoking Inter-EPB Function Calls

There could be cases where dependencies between the states of two different VDU / VNFs can exist. In such cases VDU1 Lifecycle transition may need to invoke a function towards VDU2 using the EPB library of VDU2. This case can be specified in the metadata of the VDU1 in the following way

In the metadata, there is a keyword “depends-on-vdu” which can be used to define the VDU which should be informed about the current VDU. This is an optional keyword, and if defined in any "actions", corresponding defined EPB Library (e.g. JAR file) will be loaded and the function will be called, with following two arguments

1. Argument 1: contains following information
   1. Connection point of all the instances of current VDU
   2. Connection point of all the instances of VDU defined in “depends-on-vdu” section.
2. Argument 2: As defined in the metadata (function-argument)

### VDU connection point interface

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Cardinality | Description |
| self\_instances | Structure | 1..n | List of instances of current VDU |
| depends\_on\_vdu\_instances | Structure | 1..n | List of instances of other VDU |

### self\_instances

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Cardinality | Description |
| vnfc\_connection\_points | Structure | 1..n | List of connection points for the VNF components |

### depends\_on\_vdu\_instances

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Cardinality | Description |
| vnfc\_connection\_points | Structure | 1..n | List of connection points for the VNF components |

## Default States

The factory settings of the metadata will contain the default states as stated by ETSI. The factory setting states are given below

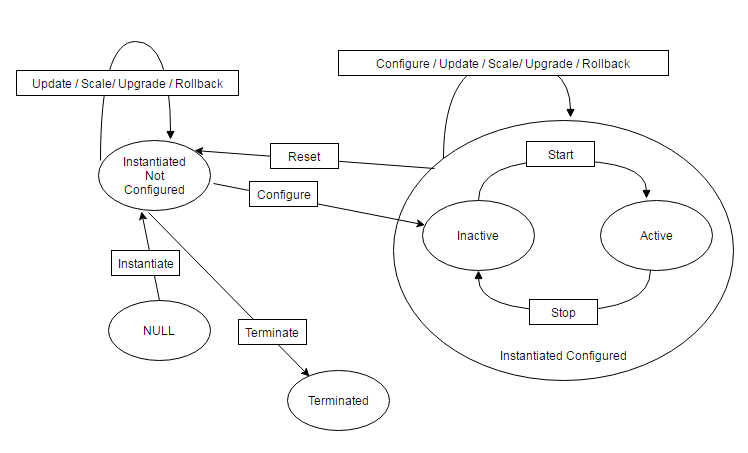


Figure 3 Default States in the Metadata

## Sample Metadata

{

id: “VNFD\_1\_VDU\_1”

communication: Bidirectional / Unidirectional

"states": [{

"name": "InstantiatedNotConfigured",

"init-state-function": [{

"package-name": "",

"Data": "",

"Name": ""

}],

"events": [{

"event-name": "Configure",

"next-success-state": "InstantiatedConfigureInactive",

"next-failure-state": "InstantiatedConfigureInactive",

"actions": [{

"package-name": "com.lcm.vnflcm.pluginjars.sshInterface",

“function-interface”: [{

“function-name”: "sshVnf",

“function-argument”: "{\"user\":\"user\",\"password\":\"abc@123\", \"command\":\"ls -ltr;dir\"}"

}]

}]

}, {

"event-name": "terminate",

"next-success-state": "Terminated

"next-failure-state": "InstantiatedConfigureInactive",

"actions": [{

"package-name": "com.lcm.vnflcm.pluginjars.sshInterface",

“function-interface”: [{

“function-name”: "sshVnf",

“function-argument”: "{\"user\":\"user\",\"password\":\"abc@123\", \"command\":\"ls -ltr;dir\"}"

}]

}]

}]

}

],

"delay": "In seconds to wait for this VNF to get completed metadata processing before going on to next VNF",

}

# Ve-VNFM Monitoring Interface

## Architecture

The monitoring architecture provides the following interface to monitor the health of the deployed VNFs

* An interface to the operator to define monitoring parameters through the VNF descriptors. This is the usual Or-VNFM or UI / CLI interface for defining VNF descriptors
* Additionally, the architecture provides a policy interface where policies can be defined. Through policies, the monitoring data can be co-related and actions like elastic scaling etc. can be performed

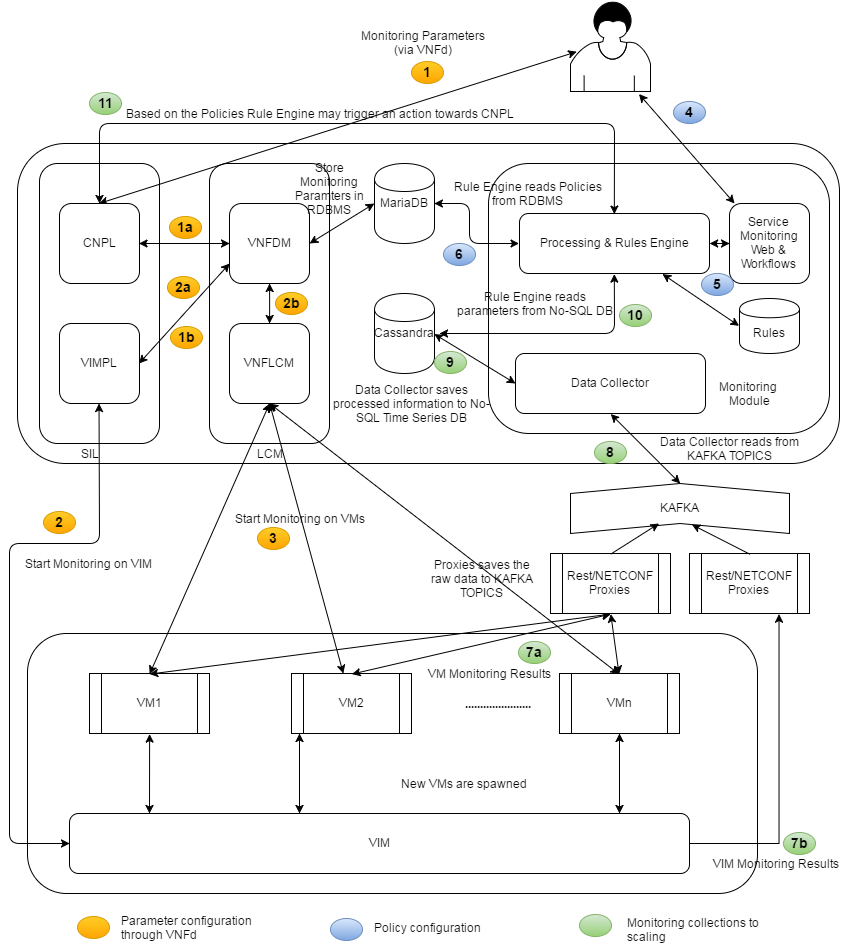


Figure 4 Monitoring Architecture and Context with Lifecycle Manager

1. The Monitoring parameters are configured for the VNFs at the VDU or VNF level through VNF descriptors during VNFd upload. The configuration Plugin (CNPL) or the Software Integration Layer (SIL) receives the VNFd.
   1. **1a**: After validation, the monitoring parameters are stored in the catalogs as well as in the resource database (MariaDB).
   2. **1b**: The virtual resource creation is initiated for the specific VIM using VIM Plugin (VIMPL)
2. VIM specific parameters are configured over the VIM telemetry service (e.g. Ceilometer) by the VIM Plugin.
   1. **2a**: VIM Plugin returns all relevant information on the resource creation to VNFD Manager (VNFDM)
   2. **2b**: The VNFD Manager (VNFDM) initiates lifecycle management through VNF Lifecycle Manager (VNF LCM)
3. VNF Lifecycle Manager initiates monitoring with VNFs (or VDUs) during the Lifecycle management operations (could be both at the instantiation or later phase) over Ve-VNFM interface using VDU (or VNF) specific End-point-broker plugins
4. Policies are used to define correlation of monitoring parameters and when scaling is required. It can also define the visualization data requirements, which the monitoring manager needs to display on the web-UI
5. Policies are processed and rules are created. Rules are stored in the rules database
6. Policies can have its own lifecycle management. The policies are stored in the MariaDB database
7. VNFs and VIM monitoring are sent by the respective producers. This is in REST (or NETCONF). The data is received by REST (or other) proxies which in turn posts the message to a message queue (e.g. KAFKA)
8. Data Collector plugins (who are aware of the payload messages), picks the KAFKA topics, processes the data
9. The data collector stores the data in the NoSQL Time Series database (Cassandra)
10. Rules engine will process the data from the database (using streaming libraries)
11. If policy thresholds are reached then it will trigger defined actions (e.g. Scaling).

## Measurement Requests over Ve-VNFM

Monitoring parameters are defined in the VNF, Network Service and Virtual Deployment Unit descriptors. These are received by VNFM over Or-VNFM or EMS or VNFM-Web-UI based on the deployment. In this chapter, the monitoring parameter definitions in descriptors and the corresponding communication messaging of measurement requests (with the monitoring parameters) over Ve-VNFM are specified.

### Monitoring Parameter in Descriptors (VNFd, NSd, VDU)

|  |  |  |  |
| --- | --- | --- | --- |
| Keyword | Type | Cardinality | Description |
| param\_name | String | 1 | Name of the parameter to be monitored |
| interface | Enum | 1 | Vi\_vnfm\_c = 0 (VIM specific) or ve\_vnfm\_c = 1 (VNF specific) |
| enabled | Bool | 1 | Indicates whether monitoring needs to be started (True) or stopped (False). |
| measurement\_type | Enum | 1 | periodic\_c = 0 (periodic reporting is required)  event\_c = 1 event based |
| periodicity | U32 | 0..1 | If periodic, define is the periodicity in seconds. (Currently default would be 10 seconds. Max 30 seconds, Min 5 seconds)  Not required if measurement\_type is event\_c |
| activation-event | Event | 0..1 | If measurement\_type is event\_c, then this element defines the event (how the event occurs). |
| deactivation-event | Event | 0..1 | If measurement\_type is event\_c, then this element defines how the event gets cancelled (how the event occurs). |

### Event

|  |  |  |  |
| --- | --- | --- | --- |
| Keyword | Type | Cardinality | Description |
| event\_name | String | 1 | Name of the Event |
| threshold\_type | Enum | 1 | U32 (0), float (1), String (2) |
| threshold | UNION | 1 | In case of activation-event: threshold above which the activation event should be triggered  In case of deactivation-event: threshold below which the deactivation event should be triggered  The type (either U32, float or String) depends on threshold\_type |

### Health Monitoring Parameter in Descriptors (VNFd, NSd, VDU)

|  |  |  |  |
| --- | --- | --- | --- |
| Keyword | Type | Cardinality | Description |
| param\_name | String | 1 | “health\_monitoring” |
| interface | Enum | 1 | ve\_vnfm\_c = 1 (VNF specific) |
| enabled | Bool | 1 | Indicates whether health monitoring needs to be started (True) or stopped (False). |
| measurement\_type | Enum | 1 | periodic\_c = 0 (periodic reporting) |
| periodicity | U32 | 1 | If periodic, define is the periodicity in seconds. (Currently default would be 10 seconds. Max 30 seconds, Min 5 seconds) |
| activation-event | Event | 0 | Not required |
| deactivation-event | Event | 0 | Not required |

### Defining Measurement Requests in LCM Metadata

A mandatory function in EPB library of VNF-LCM is required to initiate measurement requests towards VNF.

public bool sendMonitoringParam(String arg1, String arg2)

* arg1: VDU Management Interface (Refer 3.3.1): This will contain relevant information generated by the VNF Manager for the EPB to communicate with the VNF.
* arg2: should contain all the monitoring parameters defined at NSd/VNFd/VDU level for a particular VDU. it is an XML format of following type

<mon-params>

<mon-param type = "periodic">

<mon-param-name>param1</param-name>

<periodicity>100</periodicity>

….

</mon-param>

<mon-param type = "event">

<mon-param-name>param1</param-name>

<threshold>100</threshold>

…

</mon-param>

</mon-params>

If false is returned from sendMonitoringParam function, it means that monitoring is not enabled for this VDU.

It is expected that this function is called at-least once in the metadata (either from state-init of any state or during event handling of any state.

## Measurement Reports from VNF to VNFM over Ve-VNFM

VNFM receives measurement reports for the monitoring parameters over Ve-VNFM. The measurement reports terminate at the monitoring module.

### Measurement Report – measurement\_report

|  |  |  |  |
| --- | --- | --- | --- |
| Keyword | Type | Cardinality | Description |
| resource-id | String | 1 | Unique resource identifier (uniquely identifies a VIM or VNF resource) |
| num\_reports | U32 | 1 | Number of packed monitoring reports in a single measurement report. Value can be from 1 to 50 |
| monitoring\_reports | Structure | 1..50 | List of monitoring reports of individual parameters. Each report must be an for an unique parameter. If multiple reports received by the VNFM of a particular parameter, VNFM will only consider the last report in the sequence |
| Timestamp | Date | 1 | Timestamp of the report. |

### Monitoring Report - measurement\_report::monitoring\_report

|  |  |  |  |
| --- | --- | --- | --- |
| Keyword | Type | Cardinality | Description |
| resource-id | String | 1 | Unique resource identifier (uniquely identifies a VIM or VNF resource) |
| param-name | String | 1 | Name of the parameter to be monitored |
| interface | Enum | 1 | Vi\_vnfm\_c (VIM specific) or ve\_vnfm\_c (VNF specific) |
| type | Enum | 1 | gauge\_c (0), cumulative\_c (1), event\_c (2) |
| value\_datatype | Enum | 1 | U32 (0), float (1), String (2) |
| value | String | 1 | Parameter value or event |
| unit | String | 0..1 | Not applicable for event type |
| Timestamp | Date | 1 | Timestamp of the report. |

### Health Monitoring Report

Health monitoring report is a special monitoring report for reporting heartbeats. The REST acknowledgement from the VNFM can be considered as the heartbeat acknowledgement from VNFM.

|  |  |  |  |
| --- | --- | --- | --- |
| Keyword | Type | Cardinality | Description |
| resource-id | String | 1 | Unique resource identifier (uniquely identifies a VIM or VNF resource) |
| param-name | String | 1 | “health\_monitoring” |
| interface | Enum | 1 | ve\_vnfm\_c = 1 (VNF specific) |
| type | Enum | 1 | gauge\_c (0) |
| value | String | 1 | For future extensions. |
| unit | String | 0 | Not applicable for event type |
| Timestamp | Date | 1 | Timestamp of the report. |

### RESTful API for VNFs to send Measurement Reports to VNFM

The VNFs can use the following REST API to send the measurement reports to VNFM.

#### Sending Measurement Reports - sendMeasReport

**POST** datacollect/rest/api/measReport

**Send Measurement Reports to VNF-Manager**

VNFs sends measurement reports to VNF Manager’s monitoring module. The request body must include a payload that contains the Measurement Report (JSON, XML).

**Normal response codes** 200

**Error response codes**

Bad Request (400), Unauthorized (401), Forbidden (403), Not Found (404), Method Not Allowed (405), conflict (409), Request Entity Too Large (413), Unsupported Media Type (415), Service Unavailable (503)

**Request parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Style | Description | Type |
| measurementReport |  | Measurement Report as defined in 4.3.1 |  |
| templateType |  | Format of the object (json,xml etc.) |  |

**Response parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Style | Description | Type |
| Responses code | plain | Denotes success or failure of the action. | xsd:dict |
| Message | plain | Denotes success or failure of the action. | xsd:dict |

## Defining Actions on Monitoring Reports through Policies

VNF Manager is expected to co-relate VNF and VIM measurements and take appropriate actions based on user / operator defined policies. The policies can be defined via Policy interface (CLI or EMS or Web-UI).

One single policy which is not referring to any other policy is called an atom. Complex policies can be constructed by associating multiple atoms by defining policy type as “policy-id”. Here policy-id is the referred and already created policy (atom).

Below table defines the attributes of a typical policy (atom).

|  |  |  |  |
| --- | --- | --- | --- |
| **Keyword** | **Type** | **Cardinality** | **Description** |
| policy-id | String | 1 | Unique policy identifier |
| vdu\_id | String | 1 | VDU against which the policy has to be executed |
| type | Enum | 1 | monitoring\_parameter\_c = 0, policy\_id\_c=1,  event\_c=2 |
| referred\_name | String | 1 | If type = monitoring\_parameter\_c, then referred\_name should be the referred monitoring\_parameter [ATOM]  If type = policy\_id\_c, then it means that the new policy to be created is a complex policy and therefore the referred name should be the referred policy [COMPLEX]  If type = event\_c, then it means that new policy to be created is a complex polict, and therefore the referred name should be referred event [COMPLEX] |
| operator | Enum | 1 | If type = monitoring\_parameter\_c then the following values are valid: “>” = 0, “<” = 1, “<=” =2, “>=” = 3, “=” = 4 [ATOM]  If type = policy\_id\_c then the following values are valid: “AND” = 5, “OR” = 6 [COMPLEX POLICIES]  If type = event\_c then the following values are valid: “AND” = 5, “OR” = 6 [COMPLEX POLICIES] |
| compared\_parameter\_type | Enum | 1 | The parameter (threshold) that is compared with the Monitoring parameter [ATOM] U32 (0), float (1), String (2)  For complex policies, it would be the other operand and therefore the associated policy i.e. Policy-id. Policy\_id\_c (3) or an event (event\_c(4)) |
| compared\_parameter | UNION | 1 | If type = monitoring\_parameter\_c, then it is the threshold value and type will depend on the compared\_parameter\_type (either U32, float or String) [ATOM]  If type = policy\_id\_c, then it is the other policy which is an operand in the complex policy [COMPLEX POLICIES]  If type = event\_c, then it is the event which is an operand in the complex policy [COMPLEX POLICIES] |
| action | Enum | 1 | NULL = 0 (no action)  Scale-out = 1  scale-in = 2  notify = 3 (notify user interface or user)  Future: Can initiate specific alarms and increment counters. To be planned in later releases |

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Remarks** |
| **Ver0.1** | Aricent | 10.06.2016 | Draft version for review |
| **Ver1.0** | Aricent | 15.06.2016 | Baseline draft for usage and user feedback |