HPE NFV Director

On-Boarding Guide

Release 4.2
Second Edition

Hewlett Packard
Enterprise

Notices

Legal notice

Printed in the US

Trademarks

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.
Oracle and Java are registered trademarks of Oracle and/or its affiliates.

Adobe®, Acrobat® and PostScript® are trademarks of Adobe Systems Incorporated.

Microsoft®, Internet Explorer, Windows®, Windows Server 2007®, Windows XP®, and Windows 7® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Firefox® is a registered trademark of the Mozilla Foundation.

Google Chrome® is a trademark of Google Inc.

EnterpriseDB® is a registered trademark of EnterpriseDB.

Postgres Plus® Advanced Server is a registered U.S. trademark of EnterpriseDB.

UNIX® is a registered trademark of The Open Group.

X/Open® is a registered trademark, and the X device is a trademark of X/Open Company Ltd. in the UK and other countries.

Red Hat® is a registered trademark of the Red Hat Company.

Apache CouchDB, CouchDB, and the project logo are trademarks of The Apache Software Foundation.

Node.js project. Joyent® and Joyent's logo are registered trademarks of Joyent, Inc.

Neo4j is a trademark of Neo Technology.

VMware ESX, VMWare ESXi, VMWare vCenter and VMWare vSphere are either registered trademarks or trademarks of VMware, Inc. in the United States and other jurisdictions

Contents

Notices **1**
Preface **15**
About this guide 15
Audience 15
Document history 15
Chapter 1 NFV Director: On Board to a Resource Pool **16**
1.1 Introduction 16
1.2 Understanding the elements of a Resource Pool 16
1.3 How to model a simple Resource Pool 18
1.3.1 Introduction 18
1.4 The elements of the Resource Pool 22
1.4.1 Introduction 22
1.4.2 DATA_CENTER element 22
1.4.3 VIM element 23
1.4.4 AUTHENTICATION:OPENSTACK Element 24
1.4.5 AUTHENTICATION:DCN Element 24
1.4.6 REGION:OPENSTACK Element 25
1.4.7 COMPUTE:OPENSTACK Element 25
1.4.8 STORAGE:OPENSTACK Element 25
1.4.9 NETWORKING:OPENSTACK Element 26
1.4.10 IMAGE_STORAGE:OPENSTACK Element 26
1.4.11 AVAILABILITY_ZONE:OPENSTACK Element 27
1.4.12 HOSTAGGREGATES:OPENSTACK Element 27
1.4.13 IMAGE:OPENSTACK Element 27
1.4.14 HYPERVISOR:VCENTER Element 28
1.4.15 SERVER:GENERIC Element 28
1.5 The main ways to model a Resource Pool 29
1.5.1 The example 01. One AVAILABILITY_ZONE 29
1.5.2 The example 02. More than One AVAILABILITY_ZONE 31
1.5.3 The example 03. More than one REGION. Two SERVERs. Two Availability Zones 32
1.5.4 The example 04. More than one REGION. One SERVER. One Availability Zone 33
1.5.5 The example 05. More than one REGION. More than two SERVERs. More than two Availability Zones 34
1.5.6 The example 06. A Resource Pool for SDN 35
1.5.7 The example 07. A Resource Pool for vCenter 36
1.6 The behavior of a Resource Pool depending on the attributes 38
1.6.1 First case: no Availability_Zone specified, no Server specified 39
1.6.2 Second case: No Server specified 40
1.6.3 Third case: All attributes specified 41
1.7 How the DCN allows us to communicate between Data Centers 41
1.8 How to export our DC to the GUI web 43
1.8.1 Upload the new Data Center Template 43
1.8.2 Create the New Data Center from Template 45
1.8.3 Create the New Data Center from Instance 47
1.9 NFV Director: VIM executions 50
1.9.1 Introduction 50
1.9.2 The “Template” category 50

1.9.2.1 Neutron Project - Networking 51
1.9.2.2 Nova Project - Compute 52
1.9.2.3 Glance Project - Images 53
1.9.2.4 Cinder Project –Storage 53
1.9.3 Difference between versions of VIMs. 54
1.9.4 Check the attributes of the VIMs. 55
Chapter 2 NFV Director: On Board to a VNF 58
2.1 Introduction 58
2.2 Understanding the elements of a VNF 58
2.3 How to model a simple VNF 59
2.3.1 Introduction 59
2.4 The elements of the VNF 63
2.4.1 Introduction 63
2.4.2 VNF Artifact 63
2.4.3 VNF_Component Artifact 64
2.4.4 End_Point Artifact. 64
2.4.5 Virtual_Port Artifact 65
2.4.6 Virtual_Machine Artifact 65
2.4.7 Virtual_Memory Artifact 66
2.4.8 Virtual_Core Artifact. 67
2.4.9 Virtual_Disk Artifact 67
2.5 Examples of VNFs 67
2.5.1 Introduction 67
2.5.2 Example 01: The simplest VNF 68
2.5.3 Example 02: VNF, Networks & Subnetworks 69
2.5.4 Example 03: A not so simple VNF 71
2.6 Upload a VNF to the solution 73
2.6.1 Through the Resource Modeler 74
2.6.2 Through a set of REST requests 76
Chapter 3 NFV Director: On Board to the Modes 80
3.1 Introduction 80
3.2 What is a Mode 80
3.3 Why different modes are needed 80
3.4 Characteristics of Modes 80
3.4.1 Default Mode 80
3.4.2 Bottom-Up 81
3.4.3 What can be created in each mode? 81
Chapter 4 NFV Director: On Board to the TLD 82
4.1 Introduction 82
4.2 Artifacts related with the Modes 82
4.3 Rest of the Artifacts of the TLD 83
4.4 Attributes and categories of a TLD 84
4.4.1 About the TLD (Task List Definition) artifact 84
4.4.2 About the TD (Task Definition) artifact 84
4.5 The trees and the UUIDs in the TLDs 86
4.6 Operations and TLDs 87
Chapter 5 NFV Director: On Board to the Workflows 89
5.1 Introduction 89

5.2 Upload a Workflow to the Solution 89
5.2.1 Using the workflow designer tool 89
5.2.2 Using specific commands 90
Chapter 6 NFV Director: State & Lifecycle of the components 91
6.1 Introduction 91
6.2 State & Lifecycle Virtual Network 91
6.3 State and Lifecycle Virtual Machine 92
6.4 State and Lifecycle VNF 93
Chapter 7 NFV Director: TOSCA Support 95
7.1 Introduction 95
7.2 TOSCA at a glance 95
7.2.1 TOSCA service template 96
7.3 NFVD TOSCA support 97
7.3.1 VDU supported information 97
7.3.2 CP supported information 98
7.3.3 Supported TOSCA service template example 98
Chapter 8 Localizing the NFV Director UI 101
8.1 Introduction 101
8.2 How to do 101
8.2.1 Language support 101
8.2.2 Localization files 101
8.2.3 Data 103
Chapter 9 On-Boarding Guide examples and References 105
9.1 Introduction 105
9.2 List of Examples per Chapter 105
9.2.1 Subchapter 1.5 Examples 105
9.2.1.1 “RP_Example_01” 105
9.2.1.2 “RP_Example_02” 105
9.2.1.3 “RP_Example_03” 105
9.2.1.4 “RP_Example_04” 105
9.2.1.5 “RP_Example_05” 106
9.2.1.6 “RP_Example_06” 106
9.2.2 Subchapter 1.6 Examples 106
9.2.2.1 “RP_Example_DC” 106
9.2.3 Subchapter 1.7 Examples 106
9.2.3.1 “RP_Example_VSD_1” 106
9.2.3.2 “RP_Example_VSD_2” 107
9.2.3.3 “RP_Example_VSD_3” 107
9.2.4 Chapter 2 Examples 107
9.2.4.1 “Example_VNF_00” 107
9.2.4.2 “Example_VNF_01” 107
9.2.4.3 “Example_VNF_02” 107
9.2.5 References 107
Chapter 10 NFV Director Assurance 108
10.1 Introduction 108
10.1.1 Monitoring Plane 108
10.1.2 Correlation plane 109
10.1.3 Autonomous Action plane 110

10.1.4 Threshold Crossing Alarms collection 110
10.2 Monitor Modeling 111
10.2.1 Monitor artifact 111
10.2.2 Condition artifact 112
10.2.3 MonitorArgument Artifact 113
10.2.4 Virtual Machine Monitoring credentials 114
10.2.5 Understanding other elements of Monitoring 115
10.3 Action Modeling 124
10.3.1 Action artifact 124
10.3.2 Action parameters artifact attributes 125
10.3.3 Associated artifact ID for action 126
10.4 Out of box monitors provided by NFV Director 126
10.4.1 Process monitors for OpenStack services 128
10.4.2 Physical server monitors 128
10.4.3 Virtual machine monitors 129
10.4.4 Self monitors 130
10.4.5 Virtual network for Nuage 130
10.5 NFVD Self-Monitoring 132
10.5.1.1 How NFVD Self-Monitoring works? 132
10.5.1.2 Default NFVD model for simplex deployment setup 132
10.5.1.3 Monitoring in SiteScope 135
10.5.1.4 Self-Monitoring of NFVD High Availablity (HA) setup 137
10.5.1.5 Self-Monitoring of NFVD Geo Redundancy (GR) setup 139
10.5.1.6 Customizations for Self-Monitors 140
10.6 Monitoring Operations 147
10.7 Multi SiteScope support 148
10.7.1 SiteScopes associated at different levels 149
10.7.2 No SiteScope associated 150
10.8 Tagging monitors in SiteScope 150
10.9 NEO4J Index Rebuilding Tool 151
10.8 State Propagation 151
10.8.3 State propagation functionality 152
10.8.4 Flow 152
10.8.5 State propagation process 153
10.8.6 SiteScope alarms 153
10.8.7 Operational status mapping 154
10.8.7.6 Sample operational state mapping (Out of the box) 154
10.8.7.7 Alarm severity level 154
10.8.7.8 Operational status calculation 154
10.8.8 State propagation model changes 154
10.8.8.6 Artifact definitions 154
10.8.8.7 PROPAGATION_RULE artifact 155
10.8.8.8 Key Attributes 155
10.8.8.9 Relationship definitions 155
10.8.8.10 Configurations 155
10.8 Integrating with external VNFM alarms 158
10.8.3 Key attributes in VNFM alarms 158
10.10 NFV Director Notification Interface 158

10.10.1 Life Cycle Management (LCM) Notifications 161
10.10.2 State Change Notifications 178
10.10.3 Operation success/failure Notifications 179
10.10.4 Self Monitor Alarms 181
10.9 Extending Action capabilities using UCA-Automation 182
10.9.3 Correlation and autonomous process 183
10.9.3.6 Alarm enrichment 183
10.9.3.7 Autonomous action 188
10.9.3.8 Action status and reporting 189
Customizations 190
10.10.5 Custom monitoring 190
10.10.5.1 Creating a custom SiteScope template 191
10.10.5.2 Creating a custom monitor palette for GUI 198
10.10.6 Custom event collection 221
10.10.6.1 Essential CA configuration 222
10.10.6.2 CA configuration file for OM alarm processing 222
10.10.6.3 Update Channel Adapter 223
10.10.7 Custom automated action 223
10.10.8 Custom closed-loop action 224
10.10.9 Custom correlation ValuePack 233
10.10 SiteScope customizations 234
10.10.3 Accessing SiteScope 234
10.10.4 Overview of the SiteScope dashboard 234
10.10.4.6 Analyzing data in SiteScope dashboard 236
10.10.4.7 SiteScope templates and monitoring 238
10.10.5 Alerts section 240
10.10.5.6 Configuring an alert 241
10.10.5.7 Prerequisites 241
10.10.5.8 Creating/copying an alert 241
10.10.5.9 Creating a new alert 241
10.10.5.10 Copying an Alert Definition 241
10.10.5.11 Testing the alert 242
10.10.5.12 Disabling an alert - optional 242
10.10.6 SNMP preferences 242
10.10.7 Sending SiteScope Alerts 245
10.10.8 User management preferences 247

List of figures

Figure 1: RP_Example_01.xml 19

Figure 2: Credentials category of an Authentication artifact 20

Figure 3: General category of a Hypervisor artifact 20

Figure 4: Credentials category of a Hypervisor artifact 20

Figure 5: Details of “RP_Example_01”, one Availability Zone 29

Figure 6: Detailed view of the “RP_Example_02”, two Availability Zones, and one Region 31

Figure 7: Detail of the “Resource Pool_Example_03”, two Availability Zones, and two Region 32

Figure 8: Detail of the “RP_Example_04”, One Availability Zones, and two Region 33

Figure 9: Detail of the “RP_Example_05”, Three Availability Zones, and two Regions 34

Figure 10: Detail of the “RP_Example_06”, VSD structure, Management network 36

Figure 11: Detailed view of the “RP_Example_07”, Datacenter, Hypervisor vCenter 37

Figure 12: Detail of new artifacts and relationships generated in the deployment of vCenter 38

Figure 13: First case, No Zone or Server 39

Figure 14: Second case, a valid Zone, no Server 40

Figure 15: Third case, a valid Zone and a valid server 41

Figure 16: Diagram of the communication between two DC through the DCN Network 42

Figure 17: Diagram of the communication between three DC through the DCN Network 43

Figure 18: Uploading a new Data Center 44

Figure 19: Creating a new Data Center from a Template file 45

Figure 20: Locating the ID in the DataCenter 46

Figure 21: Example of a “create instance” operation 46

Figure 22: Starting and stopping discovery 47

Figure 23: Creating a new Data Center from an Instance 48

Figure 24: RP_Example_VNF_00.xml 60

Figure 25: VNF Group tree 68

Figure 26: VNF Group tree plus VNF Component Group tree 68

Figure 27: Group trees of a VNF element 69

Figure 28: VNF, Networks & Subnetworks 70

Figure 29: Relationship between VPort and Subnetwork 71

Figure 30: A complex example of VNF 72

Figure 31: Upload a VNF through the Resource Modeler 74

Figure 32: VNF Template upload complete 75

Figure 33: VNF Template Catalog 75

Figure 34: Assign Templates to Organization 76

Figure 35: Execute template upload with correct visibility 77

Figure 36: VNF Template Catalog 77

Figure 37: Assign Templates to Organization 78

Figure 38 Available elements per mode 81

Figure 39: Tree Group configuration window 87

Figure 40: Log in to the Workflow Designer 89

Figure 41 : State and lifecycle of a Virtual Network 91

Figure 42: State and lifecycle of a Virtual Machine 92

Figure 43: State and lifecycle of a VNF 94

Figure 44: Obtained VNF from TOSCA template example 100

Figure 45: Localized and not localized attributes 103

Figure 46: Integration of monitoring threshold crossing alerts for correlation 111

Figure 47: Monitor with two error conditions and different actions 113

Figure 48: High-level resource model 115

Figure 49: SiteScope – Self monitor template 118

Figure 50: Action Modeling artifacts 124

Figure 51: Monitors supported out of box 127

Figure 52: Process monitors for OpenStack services 128

Figure 53: Physical server monitors 129

Figure 54: Virtual machine monitors 130

Figure 55: Virtual network monitors for Nuage 131

Figure 56: Graphical view of NFVD Self-Management VNF instance 132

Figure 57: NFVD Component 133

Figure 58: NFVD components, select one of the NFVD components (Assurance Gateway : VNF Component) 133

Figure 59: NFVD Components, select Actions and Click Edit to see the default details of Assurance Gateway: VNF Component) 133

Figure 60: Monitor Process Example in GUI with ENABLED attribute, Assurance Gateway: VNF Component 134

Figure 61: LOG_MONITOR Example in the GUI shows the LogPath, LogFile and LogPattern editable fields, for Assurance Gateway: VNF Component example 134

Figure 62: Modeling the Self Monitor component 135

Figure 63: Self monitors 136

Figure 64: SiteScopes associated at different levels 149

Figure 65: No SiteScope associated 150

Figure 66: Monitor associated with DataCenter 151

Figure 67: Propagation modes 152

Figure 68: State propagation flow 153

Figure 69: Correlation and autonomous action process 183

Figure 70: Action status snapshot 189

Figure 71: Action reporting snapshot 190

Figure 72: SiteScope - Create custom templates 191

Figure 73: SiteScope - New template 192

Figure 74: SiteScope - Create a new template variable 193

Figure 75: SiteScope - Enter groupDescription as new template variable 193

Figure 76: SiteScope - New group 194

Figure 77: SiteScope - Create new group 195

Figure 78: SiteScope - New Monitor 195

Figure 79: SiteScope - Enter variable to associate with template 196

Figure 80: SiteScope – Custom template hierarchy 196

Figure 81: SiteScope - Create custom monitor variables 197

Figure 82: SiteScope - Configure condition expression 198

Figure 83: Create a new custom monitor palette in the GUI 198

Figure 84: Standard VM palette connectors 199

Figure 85: Create new connector for standard VM palette 199

Figure 86: Rename the group 200

Figure 87: Regenerate UUID 200

Figure 88: Export VM palette in offline mode 201

Figure 89: Export custom VM palette 201

Figure 90: Search for custom VM in the GUI 202

Figure 91: Launch NFV Visual Designer wizard 203

Figure 92: Name the new NFV Diagram 204

Figure 93: Drag MONITOR:CUSTOM from definitions 205

Figure 94: SiteScope - Configure template path 206

Figure 95: Add variable restrictions 207

Figure 96: Add restrictions in General category 208

Figure 97: MONITOR_ARGUMENTS 209

Figure 98: Create new monitor palettes 209

Figure 99: Drag MONITOR_ARGUMENTS:CYPHER and configure restrictions 209

Figure 100: Create relationships between MONITOR_CUSTOM and MONITOR_ARGUMENTS:ARGUMENTS 210

Figure 101: Adding restrictions for conditions 210

Figure 102: Adding restrictions for actions 211

Figure 103: Create action parameters 212

Figure 104: Create action parameter categories 212

Figure 105: Create a new group for MONITOR:CUSTOM 213

Figure 106: Add components to the group 214

Figure 107: Create connectors for the component palettes 215

Figure 108: Create connectors from VNF to action 216

Figure 109: Create connectors from VNFC to action 217

Figure 110: Create connectors from VM to action 218

Figure 111: Final component diagram after creating all the components 219

Figure 112: Export palette in offline mode 219

Figure 113: Export the custom monitor palette 220

Figure 114: Search for custom monitor in the GUI 221

Figure 115: New custom monitor in the GUI 221

Figure 116: List of UCA Value Packs for NFV Director 223

Figure 117: UCA/Service Inventory view in HPSA 224

Figure 118: Create a new custom action and problem in the HPSA UCA/ActionFramework Inventory 225

Figure 119: Create a new action 225

Figure 120: Provide new action parameter details with Open Loop 226

Figure 121: Create a new action with Closed Loop 226

Figure 122: Create a new problem 227

Figure 123: Enter properties for the new Open Loop problem 228

Figure 124: Enter properties for the new Closed Loop problem 228

Figure 125: List the new problems 228

Figure 126: UCA/Parameters view in HPSA Inventory 229

Figure 127: Create a new workflow template to map an action to an HPSA workflow 229

Figure 128: Enter properties for the new Open Loop action workflow template 230

Figure 129: Enter properties for the new Closed Loop action workflow template 230

Figure 130: NFVD/Parameters view in HPSA Inventory 231

Figure 131: Select Custom workflow 231

Figure 132: Creating a new custom workflow 231

Figure 133: Set attributes to action parameters 232

Figure 134: Map new action to the action parameter in the GUI 232

Figure 135: List of UCA value packs in NFV Director 233

Figure 136: Add a filter to the OrchestraFilter.xml file 233

Figure 137: Configure the OrchestraConfiguration.xml file 234

Figure 138: SiteScope dashboard 235

Figure 139: View monitor status in SiteScope dashboard 236

Figure 140: View configured and triggered alerts in SiteScope dashboard 237

Figure 141: Acknowledging monitors in the SiteScope dashboard 237

Figure 142: Viewing monitor history in the SiteScope dashboard 238

Figure 143: SiteScope sample template 240

Figure 144: SiteScope monitor context 240

Figure 145: SiteScope - send alerts always 245

Figure 146: SiteScope - send alert once 246

List of tables

Table 1: Document history 15

Table 2: NETWORKING:OS Template Attributes 52

Table 3: NETWORKING:OS:DCN Template Attributes. 52

Table 4: COMPUTE:OS Template Attributes 53

Table 5: IMAGE_STORAGE:OS Template Attributes 53

Table 6: STORAGE:OS Template Attributes. 54

Table 7: Localization example 102

Table 8: Monitor artifact attributes 111

Table 9: Condition artifact attributes 112

Table 10: Out of the box monitors for components 116

Table 11: VNFC's with MONITOR:SELF 116

Table 12: Escape characters 119

Table 13: Regular expressions for out of box monitors 120

Table 14: Escape characters 123

Table 15: Action artifact attributes 124

Table 16: Action parameters artifact attributes 125

Table 17: Physical server counters 128

Table 18: Virtual machine counters 129

Table 19: Linux and Windows Process monitor attributes 130

Table 20: Virtual network counters for Nuage 130

Table 21: Key attributes in VNFM alarms 158

Table 22: NFVD alarm fields 159

Table 23: Autonomous action alarm attributes 188

Table 24: NFVD Alarm attributes 222

Table 25: SiteScope dashboard context buttons 235

Table 26: Status and availability levels in the SiteScope dashboard 238

Table 27: SiteScope template objects 239

Table 28: SiteScope alerts 241

Table 29: SNMP user interface elements 242

Table 30: SNMP Preferences - SNMP Connection Settings 243

Table 31: SNMP Preferences - Advanced Settings 244

Table 32: User Management preferences 247

Preface

About this guide

This document is intended to guide NFV Director users into the designing of VNF, scenarios, and entities to use with the system.

Audience

This document is written for any level of user of NFV Director: Domain users, Organization Users, VDC Users, Group Users and Datacenter users.

For installation and configuration information, please refer to the *NFV Director Installation and Configuration Guide*.

For user guidance, please refer to the *NFV Director User Guide*.

Document history

Table 1: Document history

Edition	Date	Description
1.0	28 February 2017	First Edition
2.0	14 March 2017	Second Edition. Formatting

Chapter 1
NFV Director: On Board to a Resource Pool

Note: Normally, NFV Director will automatically discover resources from VIM. Refer to VIM integration guide for more details on how to perform discovery. Manual resource modelling is required only when discovery is not used.

1.1 Introduction

This chapter describes the creation of a Resource Pool. We are going to treat this Resource Pool as scenarios with different levels of customization.

The scope of this guide is to show the user how to design scenarios that meet specific expectations and objectives. This guide describes the step by step process of designing such scenarios and discusses the elements the scenarios are composed of.

To achieve the above goal, this guide explains which artifacts take part in the creation of a Resource Pool and what values must be set for certain attributes to get the expected results and behavior. An artifact is an element of the scenario. When the guide refers to an artifact, it is referring a specific representation of a network topology element. This would be a Network, a Virtual Port, a Server, or more complex elements as the ones related with the OpenStack platform, Storage, Compute, etc. These artifacts have different categories with different attributes depending on the element that they represent. In the following chapters of this guide, the user is provided with a set of examples explaining the different categories.

When opening a Resource Pool, the user sees a structure of elements identified by their artifact Definition, with a specific and unique id. It contains a number of categories with a set of attributes. These artifacts are connected with different types of relationships to achieve a specific target, respecting the hierarchy of execution.

The Resource Pool has a tree structure. Also, the different artifacts can be grouped in trees or subtrees, each tree must have a unique Root Artifact. In order to make each scenario easier to understand, we are going to explain their conformation step by step through the relationships and function of the artifacts in each scenario.

1.2 Understanding the elements of a Resource Pool

The first element to take into consideration when modeling a Resource Pool is the `DATA_CENTER` artifact, the main element of the Resource Pool.

A Datacenter can be connected to a VIM or a Server. It can be connected to other artifacts, too, but these two are the most important for understanding the behavior of a scenario. A VIM is a Virtual Infrastructure Manager responsible for the orchestration of the artifacts below it. A VIM must always be connected to a Data Center as a child. Also, this artifact it is going to be the unique parent of the artifact `HYPERVISOR`. Besides, this is the one that handles the artifact `AUTHENTICATION`.

The `HYPERVISOR` artifact is responsible for monitoring the virtual machines hosted by the scenario and the servers present in the scenario. There is a wide range of HYPERVISORs, as there is a wide range of operating systems that can be used on virtualization. The HYPERVISORs that we use most are of types KVM, VCENTER and VMWARE. These are not the unique HYPERVISORs that we use, but the user can master these three easily and find them very useful.

The AUTHENTICATION artifact is responsible for carrying on the authentication in the OpenStack platform. The artifact must have at least user, login and pass present to grant proper access to the platform. The AUTHENTICATION artifact must have all the attributes of the category CREDENTIALS correctly filled.

The AUTHENTICATION artifact must always be the parent of the REGION:OPENSTACK artifact, which defines which REGIONs are to be used during the different executions that can take place over the scenario. It defines a name and a type, and every artifact that takes the defined "Type" into consideration should be of the same Type to achieve the goal of the processes launched over them.

After the REGION artifact, we can find three main elements, STORAGE, COMPUTE and IMAGE_STORAGE, each of them having a specific objective.

The STORAGE:OPENSTACK artifact is the responsible for the Block Storage service (CINDER), this means that the artifact STORAGE should present storage resources to end users that can be consumed by the OpenStack Compute Project. It virtualizes pools of block storage devices and provides end users with a self-service API to request and consume those resources without requiring any knowledge of where or on what type of device their storage is deployed. This almost sums up what the artifact COMPUTE does.

The REGION artifact also acts as the first level of Zoning in our ResourcePool, the second level of Zoning is represented by the artifact AVAILABILITY_ZONE:OPENSTACK, allowing us to divide our zones at scenario level and to reflect in the ResourcePool the limitations of a real world asset. As a geographically isolated zone, the AVAILABILITY_ZONE artifact is related to the artifact REGION:OPENSTACK.

The artifact COMPUTE allows us to access the cloud computing fabric controller known as NOVA. It is designed to manage and automate pools of computer resources, and can work with widely available virtualization technologies, as well as bare metal. KVM and VMware are available choices to be used as HYPERVISOR.

The artifact IMAGE_STORAGE:OPENSTACK allows us to provide discovery, registration, and delivery services for disk and server images through the OpenStack image Service GLANCE. Stored images can be used as templates. The artifact IMAGE_STORAGE:OPENSTACK can also be used to store and catalog an unlimited number of backups.

The Image Service can store disk and server images on a variety of back-ends, including OpenStack Object Storage. The Image Service API provides a standard REST interface for querying information about disk images. It also lets clients stream the images to new servers.

An IMAGE:OPENSTACK can often be thought of as "virtual machine templates." Images can also mean standard installation media, such as ISO images. Essentially, they contain bootable file systems that are used to launch instances. In order to properly identify the images that the user is going to use in their scenario, the user must fill at least the attributes present in the category CONTENT. The fields INFO.ID and GENERAL.Name would be useful as well.

The SERVER:GENERIC artifact represents a server in the real world. It has a CPU, DISK, MEMORY and a CORE associated in order to virtualize all the parts needed for the proper running of the server. The attributes present in the category GENERAL should be filled properly, paying special attention to the attributes Name, hostname and usage_mode.

The NETWORKING:OPENSTACK artifact is responsible for the correct deployment of the networks related to the scenario, in this matter, if we only have one NETWORKING in the scenario, only one Network will be deployed, regardless of the number of VIMs present in the scenario. This artifact is directly related to the OpenStack Networking Project NEUTRON. This is an SDN networking project focusing on delivering networking-as-a-service (NaaS) in virtual computer environments. Neutron is designed to address deficiencies in "baked-in" networking technology found in cloud environments, as well as the lack of VDC control in multi-VDC environments over the network topology and addressing.

After this short review of each artifact that plays a role in the different processes in our scenario, we can explain the main ways to model scenarios and define different behaviors during the execution of diverse processes.

1.3 How to model a simple Resource Pool

1.3.1 Introduction

TIP: To properly follow this paragraph, please open the file "RP_Example_DC01.nfvd" with the resource modeler tool. The file is in the same folder as the document. Examples: `RP_Example_01.nfvd` , `RP_Example_01.xml`.

To create a functional model of a Resource Pool, we will go through each step of the development discussing the different possibilities and combinations of the artifacts.

If you already have opened the file mentioned above, you can see the first element of the model, this is a `DATA_CENTER`, in the categories of the artifact you can see that the `GENERAL.Name` is equal to the identifier given to the `DATA_CENTER`. These are the two crucial attributes of this element. The `DATA_CENTER` is connected to a `VIM:HELION`, with a relationship of type `CONTAINS`. The `DATACENTER` has the possibility to contain more than one `VIM` of more than one type. There is no specific need to configure the `VIM:HELION`, only to have it present in the scenario with status `INSTANTIATED`, ready to be activated.

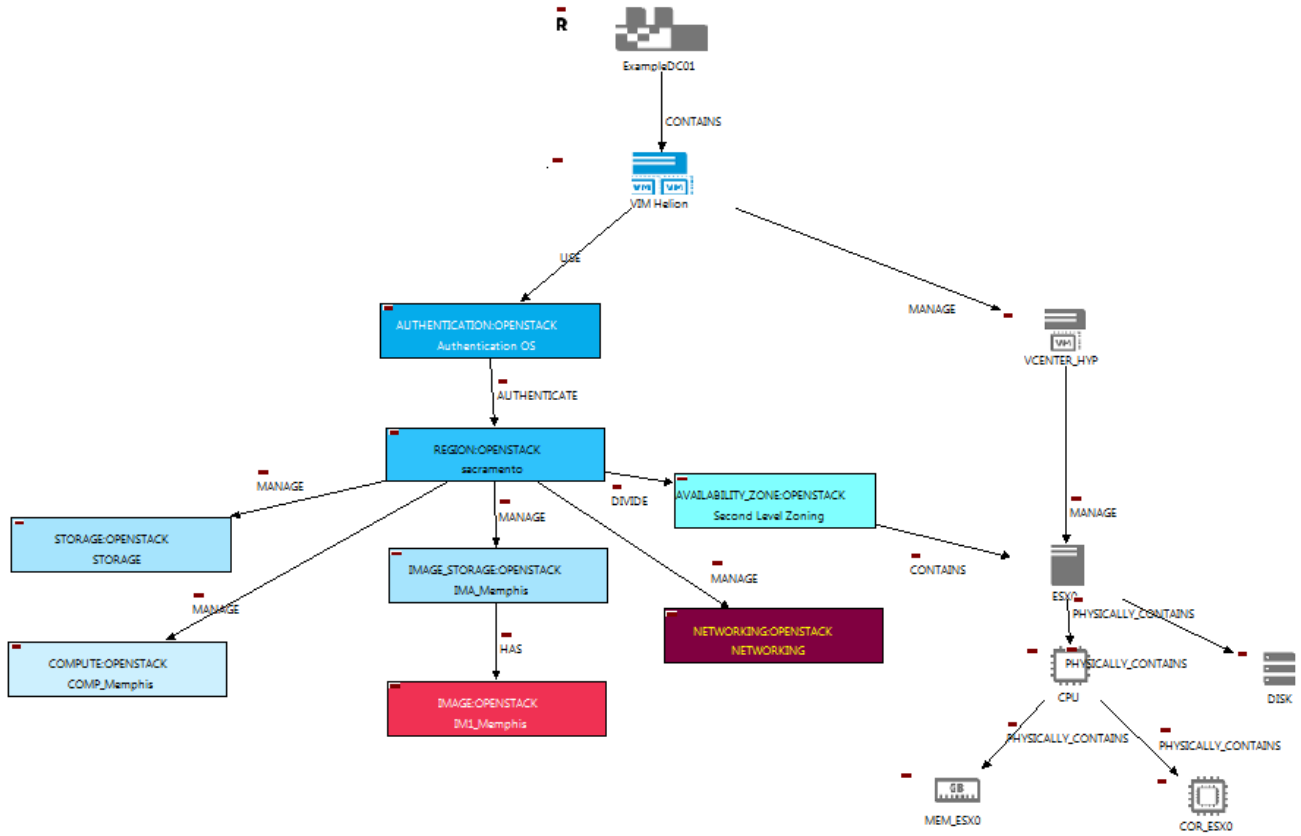


Figure 1: RP_Example_01.xml

The main artifacts that act as children of a VIM are `AUTHENTICATION:OPENSTACK` and `HYPERVISOR`. As already stated, we can have different types of `HYPERVISOR` in our scenario. In this case, the type is `VCENTER`. This means that the `HYPERVISOR` is going to monitor a Virtual_Machine and it is going to use a "vmdk" image during the activation process.

As mentioned above, the `AUTHENTICATION:OPENSTACK` artifact is responsible for connecting to and logging in to the OpenStack platform. Obviously, the most crucial attributes to be set are `Url`, `Login` and `Password` in the category `CREDENTIALS`, the rest of the attributes of the category `IdentityVersion`, `TenantName`, and `UserId` are also mandatory and should be filled before getting back to modelling. Pay special attention to the `Url`, make sure that it is correct by checking and re-checking that the OpenStack data at our disposal matches the data in our scenario. The URL address present in the following figure is the one that was used during the creation of this guide. To know more about the OpenStack Platform and its processes of authentication, please refer to the *OpenStack administration guide* [(5)].

CREDENTIALS

label	M	value
Url	*	http://10.222.0.102:5000/v2.0/tokens
Login	*	admin
Password	*	i8ksweEmn6
IdentityVersion	*	V2
TenantName	*	admin
UserId	*	6265d1505965498096699a48d39e6baa

Figure 2: Credentials category of an Authentication artifact

The `HYPERVISOR`, in this case, `VCENTER`, is responsible for monitoring the virtual machines hosted by the scenario. It is very important to set all of the attributes in the categories `GENERAL` and `CREDENTIAL` correctly. In our case, this is:

GENERAL

label	M	value
Host	*	HCG-CCP-CPX-N0001-NETCLM
Name	*	domain-c44(Compute B2B)
Type	*	VCENTER

Figure 3: General category of a Hypervisor artifact

CREDENTIALS

label	M	value
Host	*	10.212.0.52
Port	*	22
Login	*	root
Password	*	vmware

Figure 4: Credentials category of a Hypervisor artifact

In this case, all the values present in the attributes of the categories are present in the OpenStack Platform, therefore the values are correct. However, depending on your OpenStack installation, the values of the attributes may vary.

Once we are sure that the `AUTHENTICATION:OPENSTACK` artifact is set correctly with our OpenStack Data, we can continue with the artifact `REGION:OPENSTACK`. This artifact is easy to set for activation, we only need to set the `GENERAL.Name` attribute with the name set in our OpenStack platform for the `REGION` in which we are going to launch the activation. This is tremendously important because Keystone needs to send us back the token and the correct URL address for the process that we are about to launch.

When checking the `REGION` artifact, you can see five artifacts related to it. Four of them respond to an OpenStack module, and are used with it. Without them, it will be impossible to use the features of the platform, the `AVAILABILITY_ZONE` artifact is a zoning element.

`STORAGE:OPENSTACK`: As described above, this is the artifact responsible for the storage control and the consumption of the resources associated. The OpenStack Module associated is `CINDER`. The most important category of the artifact is `TEMPLATES`. All the attributes in this category must be set properly. In our case, the attributes have the following values:

```
Manufacturer ->          HPE
Element_type->           CINDER
Version      ->          HELION CG 2.0
```

`COMPUTE:OPENSTACK`: As described above, it is the artifact responsible for managing and automating pools of computer resources. The OpenStack Module associated is `NOVA`. The most important category of the artifact is `TEMPLATES`. All the attributes in this category must be set properly. In our case, the attributes have the following values:

```
Manufacturer ->          HPE
Element_type->           NOVA
Version     ->           HELION CG 2.0
```

In this case, the command that will be launched to activate the server will look like this:

```
$ nova boot --image <uuid> --flavor m1.tiny --key_name test –availability-zone
nova:server2
```

NETWORKING:OPENSTACK: As described above, this artifact is responsible for the correct deployment of the networks related to the scenario. The OpenStack Module associated is NEUTRON. if this artifact is not present, there will be no network deployed during any process that involves the scenario.

The next artifact is IMAGE_STORAGE:OPENSTACK. As mentioned above, this allows us to provide discovery, registration, and delivery services for disk and server images. The OpenStack Module associated is NOVA. The most important category of the artifact is TEMPLATES. All the attributes in this category must be set properly. In our case, the attributes have the following values:

```
Manufacturer ->          HPE
Element_type->           GLANCE
Version     ->           HELION CG 2.0
```

Notice the attribute Version above. The value of the attribute is HELIONGC_v2.0 this means that the OpenStack to which we are aiming at is a platform HPE Helion. In other words, it is a platform OpenStack of type HPE Helion. If you need to have another type of OpenStack platform such as Kilo, it is enough to change the attribute version to the one that meets your needs. This should be done in all the artifacts that contain the attribute Version.

AVAILABILITY_ZONE:OPENSTACK: It is an artifact that should be present in the scenario to manage the zoning of the ResourcePool. It is not mandatory to fill some of the attributes, but this artifact must be present in the scenario for the correct behavior. The SERVERs will be connected to this artifact instead of the first option, the DATA_CENTER. This allows us to divide and manage a real zone through the AVAILABILITY_ZONE artifact.

There are only two main artifacts that need to be discussed, IMAGE_OPENSTACK and SERVER:GENERIC. Of course, in the scenario we have more artifacts than these two, but the artifacts that are children of the SERVER are inherent to it. These artifacts are CPU, DISK, MEMORY and CORE. All of them are GENERIC, as their parent is SERVER.

IMAGE_OPENSTACK represents the image that it is going to be used in the VIRTUAL_MACHINES. To identify which images are to be used, the attribute File must be set with the correct value in the category CONTENT in the artifact. This is the specific name of the activation file that the IMAGE has in the platform OpenStack. If the file is not found, the process will fail. The other important attribute of this artifact is GENERAL.Name. It must be equal to the name of the IMAGE in the OpenStack Platform.

In the artifact SERVER:GENERIC all the attributes present in the category GENERAL must be set, especially Type , usage_mode and hostname. Type represents what type of project module we are going to use in OpenStack, and usage_mode is going to take different values depending on the entity involved in the process involving our scenario. In this case, the value is shared. Each of the server's children is a virtualization of a physical device, so we should set the attributes of the artifacts based on the specs of real hardware.

```
CPU:GENERIC        No special settlement of the attributes needed.
CORE:GENERIC       We should set the INFO.Amount, in our case, the value is
"64"(GHz)
```

```
MEMORY:GENERIC    We should set the INFO.Amount, in our case, the value is
“247910”(Mb)
DISK:GENERIC      We should set the INFO.Amount, in our case, the value is
“4096” (Gb)
```

1.4 The elements of the Resource Pool

1.4.1 Introduction

In this section, we are going to discuss the most common elements used in our Resource Pool.

1.4.2 DATA_CENTER element

Data Center

`DATA_CENTER:GENERIC`: Main element of the Scenario. This element must always be the root of the main tree of the Scenario. The identifier of the artifact and the attribute `GENERAL.Name` must be configured with the same value.

Attributes that should be filled:

```
GENERAL.Name            The name of the Datacenter that is going to be used
                        in the rest of the artifacts implied in the Resource
                  Pool, for example, SHARED_RESOURCE artifacts.
```

Artifacts that can be related to the artifact:

```
VIM:HELION                    CONTAINS
VIM:OPENSTACK                 CONTAINS
VIM:GENERIC                   CONTAINS
VNF:GENERIC                   CONTAINS
ZONE:EXTERNAL                 CONTAINS
ZONE:INTERNAL                 CONTAINS
ZONE:FRONTEND                 CONTAINS
ZONE:BACKEND                  CONTAINS
ZONE:FIREWALL                 CONTAINS
ZONE:CE                       CONTAINS
RACK:GENERIC                  PHYSICALLY_CONTAINS
SERVER:GENERIC                PHYSICALLY_CONTAINS
POLICY:ENTITY_RANGE           APPLY
SHARED_NETRESOURCE            USE
```

1.4.3 VIM element

VIM: Main element of the Scenario. This element must always be present in the main tree, and always after a **DATA_CENTER:GENERIC** artifact. The Virtualized Infrastructure Manager handles the virtualization of physical hardware in the data center by integrating the hardware with the virtual machine manager. Using a hypervisor, the virtual machine manager provides the ability to create multiple virtual compute, network, and storage elements. The virtual machines provide lifecycle management functions (create, edit, delete, start, and stop) for the virtual data center elements related to compute, network, and storage functions.

Attributes that should be filled:

```
GENERAL.Name:           Name of the VIM, highly recommended to be filled.
GENERAL.Host:           Host of the VIM, highly recommended to be filled.
CREDENTIALS.Login       Depends on whether the future uses of the Resource
                  Pool should be filled or not. Fill with the user
            identifier.
CREDENTIALS.Password    Depends on whether the future uses of the Resource
                  Pool should be filled or not. Fill with the user
            password.
```

Artifacts that could be related to the artifact:

```
HYPERVISOR:GENERIC            MANAGE
HYPERVISOR:KVM                MANAGE
HYPERVISOR:VMWARE             MANAGE
HYPERVISOR:VCENTER            MANAGE
HYPERVISOR:IRONIC             MANAGE
TENANT:OPENSTACK              MANAGE
DOMAIN:OPENSTACK              MANAGE
AUTHENTICATION:OPENSTACK      USE
SAN:GENERIC                   CONTAINS
```

1.4.4 AUTHENTICATION:OPENSTACK Element

`AUTHENTICATION:OPENSTACK`: Critical artifact in the Resource Pool. It is responsible for carrying on the registration and logging on the platform. All the attributes present in the category CREDENTIALS must be filled properly.

Artifacts that could be related to the artifact:

```
REGION:OPENSTACK          AUTHENTICATE
```

Before fill the attributes of the CREDENTIALS category the user should know that depending on the version of Keystone (Keystone is the OpenStack module that is responsible for the identification of the user in the platform) the user should have an artifact DOMAIN related to the VIM, in order to have a correct behavior of the scenario, this artifact is needed with the version 3.0, for the version 2.0 this artifact is not needed.

The artifact DOMAIN

Attributes that should be filled:

```
CREDENTIALS.Url                Address of the OpenStack Platform that we are
                          trying to connect to.
CREDENTIALS.Login              User Identifier to gain access to the platform.
CREDENTIALS.Password           User password to gain access to the platform.
CREDENTIALS.IdentityVersion    Version of the Identity module of the targeted
                               platform.
CREDENTIALS.TenantName         Name of the VDC under whom our Resource Pool it
                               is going to operate.
CREDENTIALS.UserId             Id of the User mentioned above.
```

1.4.5 AUTHENTICATION:DCN Element

AUTHENTICATION:DCN: Critical artifact on the Resource Pool, It is the responsible to carry on the registration and logging on the platform, all the attributes present in the category CREDENTIALS must be filled totally.

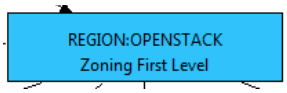
Artifacts that could be related with the artifact:

```
NETWORKING:DCN            MANAGE
```

Attributes that must be filled:

```
CREDENTIALS.Url                Address of the OpenStack Platform that we are
                               trying to connect to.
CREDENTIALS.Login              User Identifier to gain access to the platform.
CREDENTIALS.Password           User password to gain access to the platform.
CREDENTIALS.Admin_enterprise   Enterprise administrator.
```

1.4.6 REGION:OPENSTACK Element

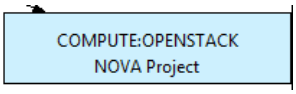


REGION:OPENSTACK: It is the artifact that represents the first level of the zoning in OpenStack. Each Region has its own full OpenStack deployment, including its own API endpoints, networks and compute resources. Different Regions share one set of Keystone and Horizon to provide access control and Web portal.

Attributes that should be filled:

```
GENERAL.Name            Name of the REGION OpenStack.
GENERAL.Description     Description of the REGION OpenStack.
GENERAL.Type            Type of the REGION OpenStack.
```

1.4.7 COMPUTE:OPENSTACK Element



COMPUTE:OPENSTACK: This artifact is directly related to the OpenStack's module **NOVA**. The module is responsible for the management of the OpenStack's pools of resources.

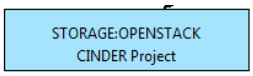
Artifacts that could be related with the artifact:

```
FLAVOR:HELION_CG        HAS
```

Attributes that should be filled:

```
TEMPLATE.Manufacturer   Name of the creator of the version of the platform OS
                        that it is going to be used, in our case "HPE".
TEMPLATE.Element_type   Name of the project OpenStack that we are planning to
                        use, in this case, it is NOVA.
TEMPLATE.Version        The most important of the three attributes of the
                        category, this attribute must be set with the correct
                        name and version of the project OpenStack to use. It is
                        crucial that the names match completely.
```

1.4.8 STORAGE:OPENSTACK Element



STORAGE:OPENSTACK: This artifact is directly related to the OpenStack's module **CINDER**. The module is responsible for the management of the Block Storage service in OpenStack.

Artifacts that could be related with the artifact:

```
LUN:GENERIC        MANAGE
```

Attributes that should be filled:

```
TEMPLATE.Manufacturer   Name of the creator of the version of the platform OS
                        that is going to be used, in our case, it is "HP".
TEMPLATE.Element_type   Name of the project OpenStack that we are going to use,
                        in this case, it is NOVA.
```

```
TEMPLATE.Version          The most important of the three attributes of the
                          category, this attribute must be set with the correct
                          name and version of the project OpenStack to use. It is
                          crucial that the names match completely.
```

1.4.9 NETWORKING:OPENSTACK Element

NETWORKING:OPENSTACK: This artifact is directly related to OpenStack's **NEUTRON** module, which is responsible for the management and the creation of network processes in the OpenStack platform.

The presence of the "NETWORKING:OPENSTACK" artifact in the scenario is crucial, it defines which version of the NEUTRON module is going to be used to create and manage networks in the Openstack Platform. Neutron is the name of the project that covers all the actions over networks and subnets in Openstack. When you create networks in the Openstack platform, you need to specify a name, version and manufacturer for the NETURON module that you wish to use

Artifacts that could be related as **NETWORKING:OS**'s children:

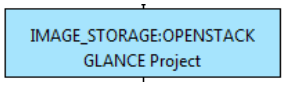
```
NETWORK:OPENSTACK         MANAGE
```

Attributes that should be filled:

```
TEMPLATE.Manufacturer     Name of the creator of the platform OS version that is
                          going to be used. In our case, it is "HP".

TEMPLATE.Element_type     Name of the project OpenStack that we are going to use,
                          in this case NOVA.
TEMPLATE.Version          The most important attribute of the three in the
                          category. This attribute must be set with the correct
                          name and version of the project OpenStack to be used.
                          It is crucial that the names match completely.
```

1.4.10 IMAGE_STORAGE:OPENSTACK Element



IMAGE_STORAGE:OPENSTACK: This artifact is directly related to the OpenStack module **GLANCE**, which provides discovery, registration, and delivery services for disk and server images.

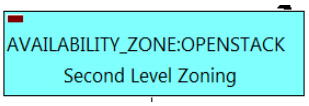
Artifacts that could be related as IMAGE_STORAGE's children:

```
LUN:GENERIC        MANAGE
```

Attributes that should be filled:

```
TEMPLATE.Manufacturer     Name of the creator of the version of the platform OS
                          that is going to be used, in our case, it is "HP".
TEMPLATE.Element_type     Name of the project OpenStack that we are going to use,
                          in this case NOVA.
TEMPLATE.Version          The most important of the three attributes of the
                          category, this attribute must be set with the correct
                          name and version of the project OpenStack to be used.
                          It is crucial that the names match completely.
```

1.4.11 AVAILABILITY_ZONE:OPENSTACK Element



`AVAILABILITY_ZONE:OPENSTACK`: Inside a Region, compute nodes can be logically grouped into Availability Zones (AZ). When launching new VM instance, we can specify an AZ or even a specific node in an AZ to run the VM instance.

Artifacts that could be related as AZ's children:

`HOSTAGGREGATES:OPENSTACK`	DIVIDE
`SERVER:GENERIC`	CONTAINS

1.4.12 HOSTAGGREGATES:OPENSTACK Element

`HOSTAGGREGATES:OPENSTACK`: Besides AZs, compute nodes can also be logically grouped into Host Aggregates. Host Aggregates have metadata to tag groups of compute nodes.

Artifacts that could be related as `HOSTAGGREGATES`'s children:

`SERVER:GENERIC`	`CONTAINS`

1.4.13 IMAGE:OPENSTACK Element

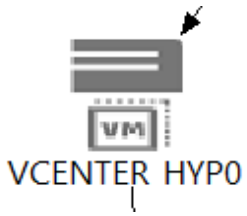
`IMAGE:OPENSTACK`: Besides AZs, compute nodes can also be logically grouped into Host Aggregates. Host Aggregates have metadata to tag groups of compute nodes.

IMAGE:OPENSTACK only acts as child of other artifacts.

Attributes that should be filled:

`GENERAL.Name`	`Name of the IMAGE:OPENSTACK that is going to be used in the Resource Pool.`
`GENERAL.Description`	`Description of the IMAGE:OPENSTACK that is going to be used in the Resource Pool.`
`INFO.ID`	`UUID identifier of the image in our OpenStack platform`
`CONTENT.File`	`Exact name and extension of the image file in our OpenStack platform.`

1.4.14 HYPERVISOR:VCENTER Element



`HYPERVISOR:VCENTER`: Hosted hypervisor of type VCENTER.

Regardless of the type of Hypervisor, the artifact's related child is always `SERVER:GENERIC`.

Attributes that should be filled:

```
GENERAL.Name                    Name of the HYPERVISOR.
GENERAL.Host                    Host present in OpenStack, should be UP, in the
                                same region and zone.
GENERAL.Type                    Type of the Hypervisor present in OpenStack
                                platform.
CREDENTIALS.Host                Host address for the Hypervisor.
CREDENTIALS.Port                Port number for the Hypervisor.
CREDENTIALS.Login               User Login to gain access to the platform.
CREDENTIALS.Password            User password to gain access to the platform.
```

1.4.15 SERVER:GENERIC Element

`SERVER:GENERIC`: A virtualized server with the specifications defined by its child artifacts and their attributes.

Artifacts that could be related as `Server`'s children:

```
LUN:GENERIC                     CONNECTED
POLICY:OVER_SUBCRIPTION         APPLY
DISK:GENERIC                    PHYSICALLY_CONTAINS
CPU:GENERIC                     PHYSICALLY_CONTAINS
CARD:GENERIC                    PHYSICALLY_CONTAINS
```

Attributes that should be filled:

```
GENERAL.Name                    Name of the SERVER.
GENERAL.Type                    Type of the module that it is going to use the
                                element.
GENERAL.hostname                Host present in OpenStack. If it matches the
                                Host attribute of the HYPERVISOR, they are
                           paired.
GENERAL.usage_mode              The mode in which the server is going to be
                                initiated, such as shared, and so on.
GENERAL.Class                   Class of the server.
STATUS.Operationa_Status        The status changes depending on the use of the
                                Resource Pool, normally the value will be UP.
STATUS.Admin_Status             The status changes depending on the use of the
                                Resource Pool, normally the value will be UP.
```

1.5 The main ways to model a Resource Pool

Example 1 shows what elements are part of a Resource Pool and what type of relationships exist between these elements. In this part of the document, we are going to explain what the consequences of our choices are during the modelling of our Resource Pool.

We will go through the range of configurations that our artifacts can manifest once created.

In Example 1, our `SERVER:GENERIC` is connected as child with the artifact `AVAILABILITY_ZONE`, and to the `HYPERVISOR`. When this configuration is activated, it is going to activate a `VIRTUAL_MACHINE` in the only `REGION` present in the scenario through the `AVAILABILITY_ZONE` provided. The VIM will not have to decide between more than one region, and the server is going to be activated from the `DATA_CENTER`, through the AZ. Therefore, the `SERVER` will be reachable by everyone with access to it, in the following examples you will see how the `AVAILABILITY_ZONE`s act as a zoning element, along with the `REGION`s.

1.5.1 The example 01. One AVAILABILITY_ZONE

Examples:"RP_Example_01.nfvd",

The main different between the first example and this second is that the last one has an `AVAILABILITY_ZONE:OPENSTACK` artifact added. As already mentioned, this artifact is a second level zoning element, and it is always related to a `REGION:OPENSTACK` artifact as child. It represents an electrically isolated zone, and the limitations that are associated to the presence of the artifact in the scenario come directly from the real world.

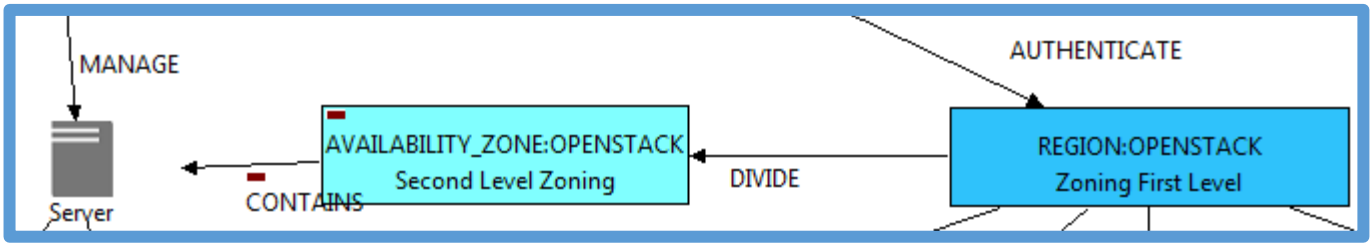


Figure 5: Details of "RP_Example_01", one Availability Zone

During the activation process, a `VIRTUAL_MACHINE` is going to be activated in the only `REGION` present in the scenario, so the VIM will not have to decide between more than one region, and the server is going to be activated from inside the `REGION` present, and inside the `AVAILABILITY_ZONE` present in the scenario. This way, the `SERVER` will only be reachable through that region, in that AZ. This is most common way to do it, in the following examples you will see how the system behaves with more than one `REGION` and AZs, also with more than one `SERVER`.

In this case, the command to activate the server will be as follows:

```
$ nova boot --image <uuid> --flavor m1.tiny --key_name test –availability-zone
nova:server2
```

```
$ nova hypervisor-list
+----+---------------------+
| ID | Hypervisor hostname |
+----+---------------------+
| 1  | server2             |
| 2  | server3             |
| 3  | server4             |
+----+---------------------+
```

1.5.2 The example 02. More than One AVAILABILITY_ZONE

"RP_Example_02.nfvd"
"RP_Example_02.xml"

In this example Resource Pool, we can see that the main difference is that we have two artifacts of type `AVAILABILITY_ZONE`. This means that we had to reflect the need for two zones to be managed during the processes of the system.

When we have two AZs in the scenario the user should keep a couple of things in mind: One `SERVER:GENERIC` only can be part of one AZ. Remember, these artifacts are electrically isolated zones in the real world, so it makes no sense trying to connect them to our scenario, they are not going to work. Also, a Region cannot share an Availability Zone with another Region.

As you can see, we only have one `HYPERVISOR` connected to our two servers. The number and type of `HYPERVISOR`s connected to `SERVER`s is not limited, because `HYPERVISOR` artifacts allow multiple operative systems to share a single hardware host.

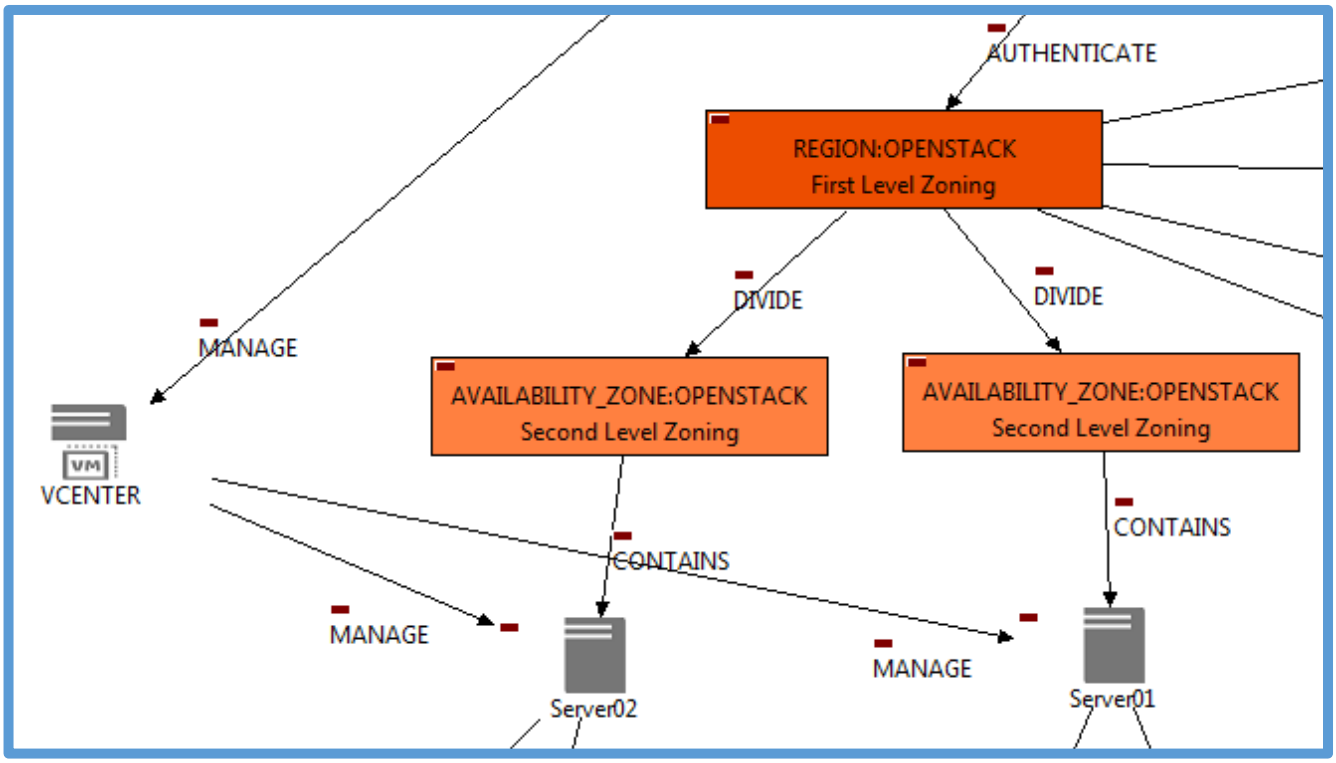


Figure 6: Detailed view of the "RP_Example_02", two Availability Zones, and one Region

During the activation process, a `VIRTUAL_MACHINE` is going to be activated in the only `REGION` present in the scenario, so the VIM will not have to decide between more than one regions. The servers are going to be activated through the `REGION` present in the Resource Pool, and through the `AVAILABILITY_ZONE` that is the owner of each server, so the `SERVER`s will be reachable only through the AZs that act as the server's parent. Of course, it is possible to add more servers to each AZ, and they can be managed by the same `HYPERVISOR` or with a new one of any type. In the following examples, you will see how the system behaves with more than one `REGION`.

In this case, the commands that activate the server will look like this:

```
$ nova boot --image <uuid> --flavor m1.tiny --key_name test –availability-zone
nova_server
```

In case we need to choose between servers, we just need to type `"nova:"+ "Name of the server"`, and choose the server's name.

1.5.3 The example 03. More than one REGION. Two SERVERs. Two Availability Zones.

"RP_Example_03.nfvd"
"RP_Example_03.xml"

In this example Resource Pool, we can see that the main difference is that we have two artifacts of type `REGION`, each one of them with the same artifact children, but with different configurations in their attributes, one `REGION` aims for the activation of a `.qcow2 IMAGE`, and the other aims for the activation of a `.vmdk IMAGE`. These differences become important when the `SERVER` is needed. For the moment, the scenario offers two different possibilities:

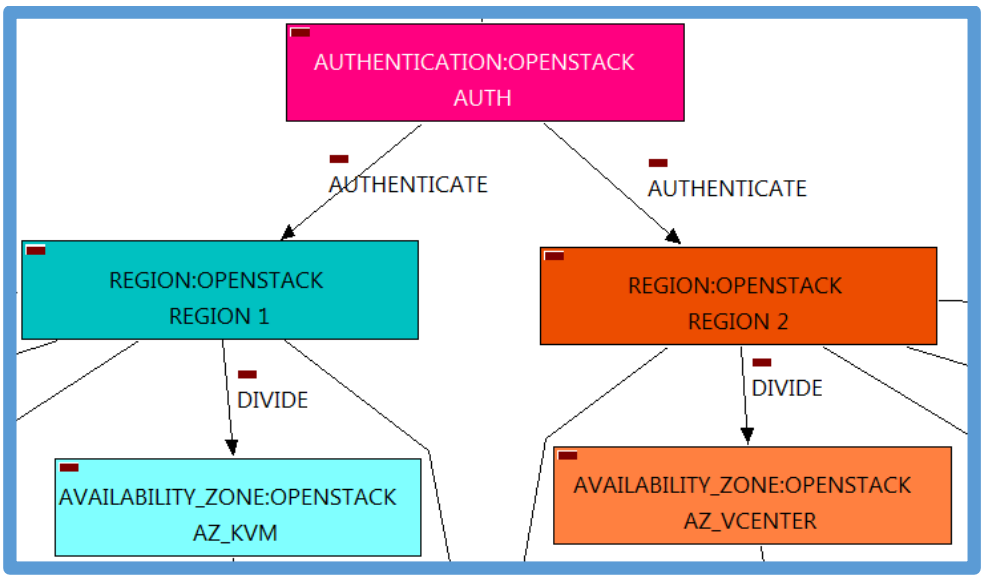


Figure 7: Detail of the "Resource Pool_Example_03", two Availability Zones, and two Region

The activation command includes the AZ and the server to be activated, no problem will be encountered, and this means that also the REGION is specified. The command that will be used is:

```
$ nova boot --image <uuid> --flavor "f.Name" --key_name "keyName" --
availability-zone "zoneHost1"
```

1.5.4 The example 04. More than one REGION. One SERVER. One Availability Zone

"RP_Example_04.nfvd"
"RP_Example_04.xml"

In this example Resource Pool, we can see that the main difference is that we have two artifacts of type `REGION`, each one of them with the same artifact children, but with different configurations in their attributes, one `REGION` aims for the activation of a `.qcow2 IMAGE`, and the other aims for the activation of a `.vmdk IMAGE`. These differences become important when the `SERVER` is needed. For the moment, the scenario offers two different possibilities:

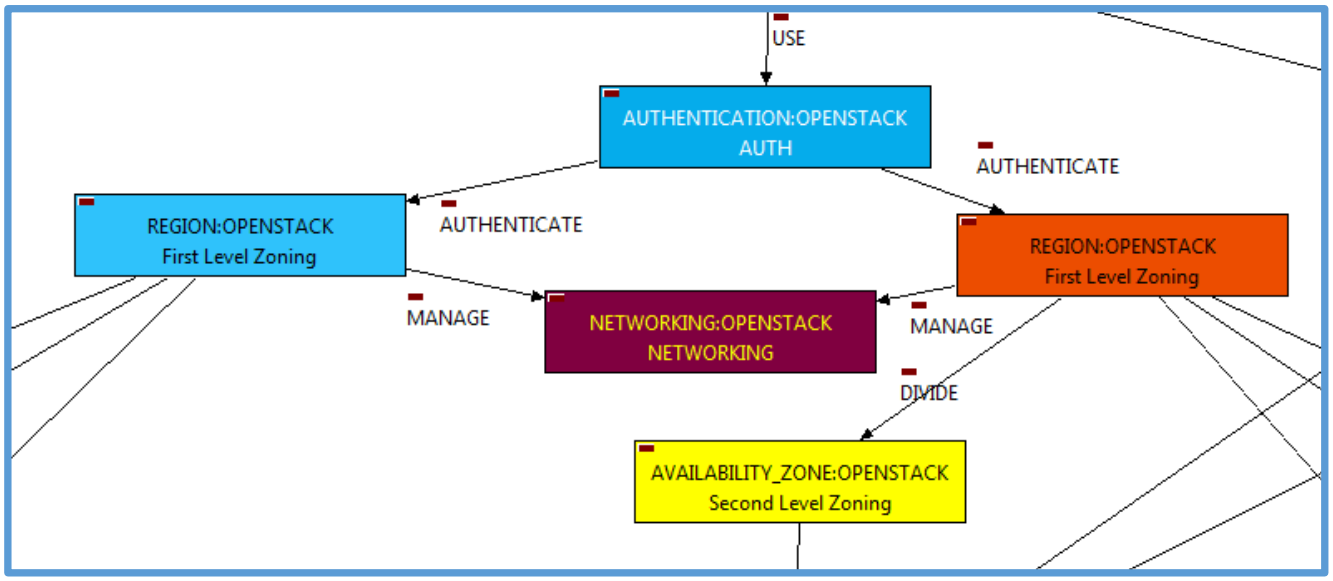


Figure 8: Detail of the "RP_Example_04", One Availability Zones, and two Region.

The activation command includes present the AZ and the server to be activated, so no problem will be encountered. This means that also the `REGION` is specified. The command that will be used is similar to the following:

```
$ nova boot --image <uuid> --flavor m1.tiny --key_name test –availability-zone
nova:server2
```

In another case, the activation command does not include the AZ. That will not be a problem, because we only have one AZ. If the REGION has more than one AZs the `REGION` will choose between them arbitrarily. If the activation command does not contain the `REGION`, the VIM will decide arbitrary in which REGION the activation it is going to take place.

During an activation operation, will try to reach the server through the `REGION` without specifying an AZ. The process will end with an error, if the user tries to activate a Server from the Datacenter level. The system will determine the most suitable place for the activation to take place. In this case, it is going to be the only server present.

1.5.5 The example 05. More than one REGION. More than two SERVERs. More than two Availability Zones.

"RP_Example_05.nfvd"
"RP_Example_05.xml"

In this example Resource Pool, we can see that the main difference is that we have two artifacts of type `REGION`, each one of them with the same artifact children, but different configurations on their attributes. One `REGION` aims to activate of a `.qcow2 IMAGE`, and the other aims to activate a `.vmdk IMAGE`. Also, each `REGION` has its own AZ, and in this particular case, we have another AZ that is part of the first REGION artifact.

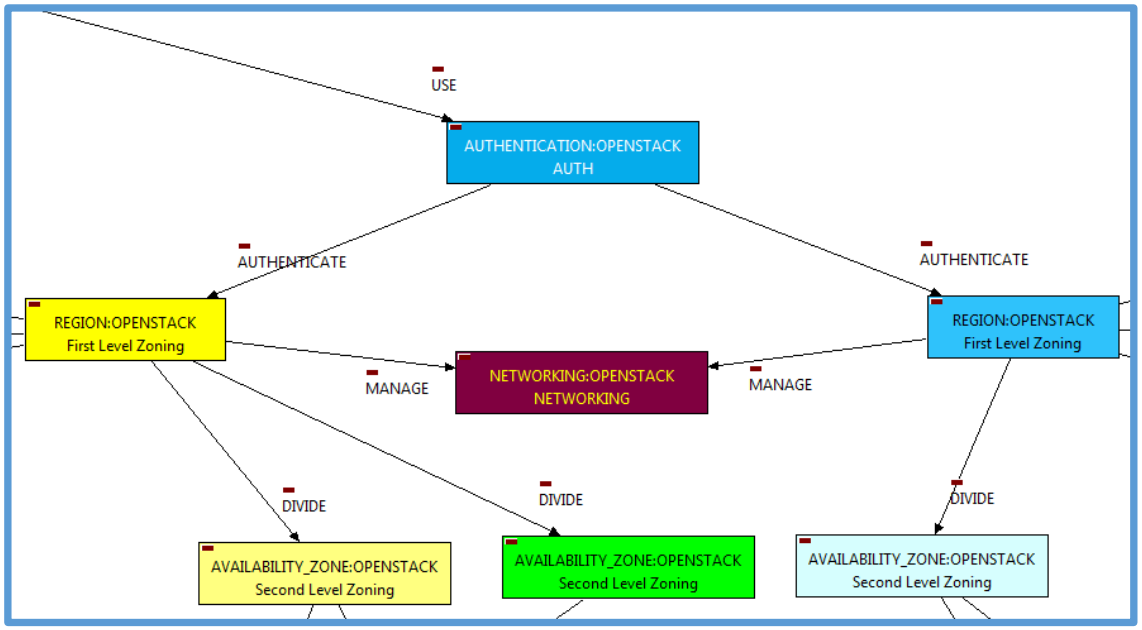


Figure 9: Detail of the "RP_Example_05", Three Availability Zones, and two Regions

The Resource Pool has five `SERVER`s, two of them are connected to one of the dedicated AZ of one REGION, another two to the other `REGION` in the same way as the first `REGION`, and one of the servers it is connected to the AZ that is related to the first `REGION`.

We must keep in mind that a `SERVER`, can only be connected to one AZ. It will not work properly if connected to more than one AZ. Also, our Resource Pool has three `HYPERVISOR`s, one of each type, `KVM`, `VCENTER` and `GENERIC`, connected adequately to the server that we choose. A `HYPERVISOR` can be assigned to more than one server, we consider it highly recommended to assign a `HYPERVISOR` of the type needed to various Servers.

The moment of activation in this Resource Pool can result in two possible scenarios:

If the activation command has been entered without a specific `SERVER`, the VIM chooses the `REGION` arbitrarily, and the AZ chooses the `SERVER` arbitrarily (if has more than one).

```
$ nova boot --image <uuid> --flavor "f.Name" --key_name "keyName"
```

If a REGION has been specified in the activation command, and we also specified AZ as reachable, it will choose one to use from among the ones connected to the REGION (in case there are more than one). After this decision was made, the processes follow as described in the previous paragraph.

```
$ nova boot --image <uuid> --flavor “f.Name” --key_name “keyName”
```

If an Availability_Zone was specified in the activation command, but nothing more:

In this case, the AZ will choose between the reachable SERVERs and use it. After this decision was made, the processes follow as described in the previous paragraph.

```
$ nova boot --image <uuid> --flavor “f.Name” --key_name “keyName” --
availability-zone nova:zoneServer1
```

The last possible case is that the command has all the proper information set. This means that the command was executed with a REGION, an AZ and a specific SERVER as targets.

```
$ nova boot --image <uuid> --flavor m1.tiny --key_name test –availability-zone
nova:server2
```

This case is extended in RP_Example_DC. In this example, the user will find a Resource Pool with two REGIONs, two Availability_Zones for each REGION, and two Servers per Availability_Zone.

1.5.6 The example 06. A Resource Pool for SDN

“RP_Example_06 .nfvd”
“RP_Example_06 .xml”

First of all, the user should know what the VSD is. The VSD is a policy engine for managing users, compute resources and network resources located on a server outside of the HP Helion OpenStack Carrier Grade cloud. The VSD Architect is a browser-based interface for management tasks on the VSD.

In order to have the elements needed for using our Resource Pool with the VSD, there are two main changes to do in our scenario. First, we must be sure that our NETWORKING artifact is of the type OPENSTACK:DCN in this example, because the Resource Pool is going to be used on both platforms.

If our Resource Pool is going to be used only in one of the platforms, it is enough to have a NETWORKING artifact of the type required. Also, we can have more than one NETWORKING in the scenario. This responds to the possibility of having NETWORKS that are going to be used only in one platform but in the same Resource Pool.

Also, to use the NETWORKING artifacts correctly, we must be sure that we only have connected or we are going to have connected to each of our NETWORKING artifacts one NETWORK, and not more than one NETWORK.

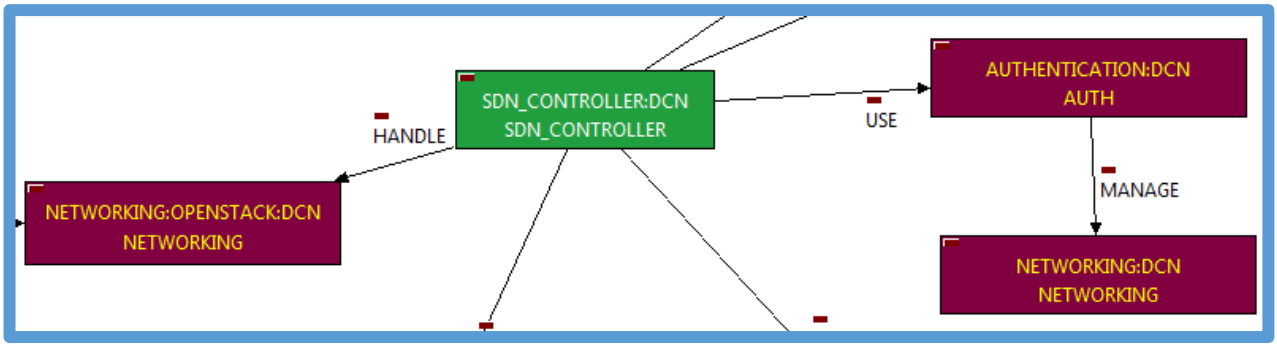


Figure 10: Detail of the "RP_Example_06", VSD structure, Management network

Once we have changed our `NETWORKING` artifact to the one of the type needed, we include a new artifact in our Resource Pool, the `SDN_CONTROLLER` artifact. This one is a bit special, because it is not the child of any other artifact, but it is the parent of all the `MANAGEMENT` Structure (example Management file). There is no special setting of the attributes of the artifact, but its presence on the scenario is crucial.

This `SDN_CONTROLLER` artifact must be connected to our `NETWORKING:OPENSTACK:DCN` with a relationship of type `HANDLE`.

All the artifacts related to the platform `OPENSTACK` as `TENANT`, `NETWORK` or `SUBNETWORK` should be set with the values of the attributes present in the platform. This data is accessible in OpenStack depending on your OpenStack build.

Another thing to keep in mind when modelling for a VSD platform is the presence of the artifacts of type `SHAREDNET_RESOURCE:DCN`, normally two. One per FLOATING IPs, and another for the shared networks of the management structure. In their attribute `DCName` in the category `INFO`, these artifacts must have the same exact value that the `DATACENTER:GENERIC` has in the attribute `GENERAL.Name`. This is a crucial point in order to get the expected behavior from the rest of the components.

Also, in this Resource Pool, the user will find another `REGION` with a `NETWORKING:OPENSTACK` artifact, this means that the `REGION`s present in this example are going to operate as follows:

```
REGION with Networking:OS:          Only operates with Networks in the
                                    OpenStack Platform.
REGION with Networking:OS:DCN:      Operates with Networks in both platforms.
SDN_CONTROLLER:                     Only Operates in VSD platform.
```

Once we check the previous requirements the Resource Pool will behave in the same way that the examples mentioned.

1.5.7 The example 07. A Resource Pool for vCenter

"RP_Example_07 .nfvd"
"RP_Example_07 .xml"

If the platform used for activation is a vCenter server, we have to consider that the Resource Pool used is a little bit different.

We are not using Openstack, so `VIM:GENERIC`, `AUTHENTICATION:OPENSTACK`, `REGION:OPENSTACK` and `AVAILABILITY_ZONE:OPENSTACK` artifacts will not appear in this case. That is because vCenter is not a VIM, it is a

centralized platform that provides management of several ESX servers: we do not need to manage networks, subnetworks, and so on.

So, the `DATACENTER:GENERIC` artifact will be directly connected to the `HYPERVISOR:VCENTER`, as a `VIM:GENERIC` on a traditional Resource Pool, with the relationship type `MANAGE`.

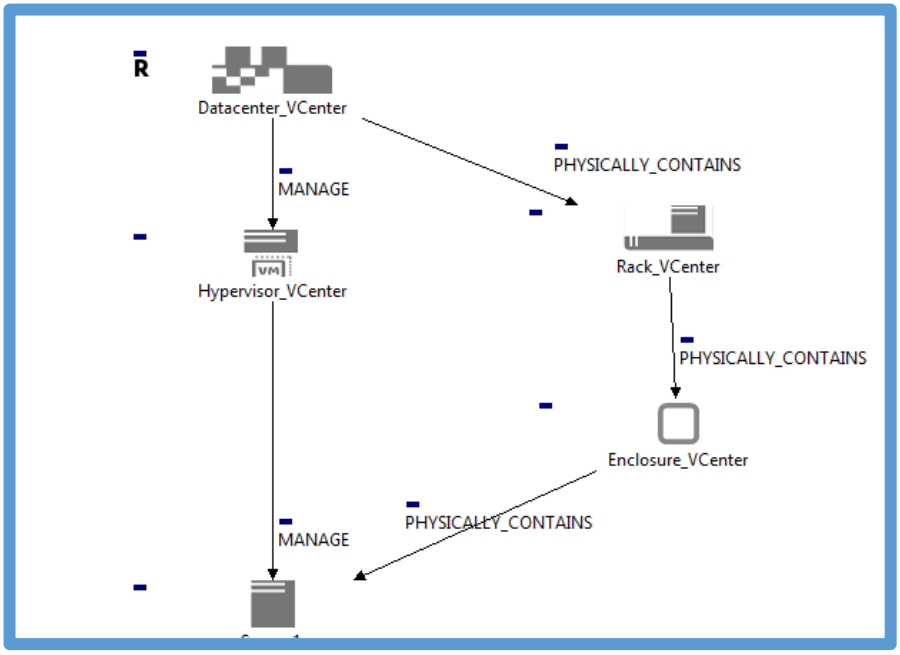


Figure 11: Detailed view of the "RP_Example_07", Datacenter, Hypervisor vCenter

The `HYPERVISOR:VCENTER`, as we already know, can have one or several `SERVER:GENERIC` as children, and these two artifacts contain all the information that is used to establish the connection to the vCenter server and the specified ESX cluster in which we are going to do the activation. The rest of the Resource Pool is basically the same.

`HYPERVISOR:VCENTER` must have the following attributes filled:

```
GENERAL.Host:                       Host of the vCenter server

CREDENTIALS.Host:                   Connection URL of the vCenter
                                    (https://{HOSTNAME}/sdk).

CREDENTIALS.Login:                  Login user of the vCenter server.

CREDENTIALS.Password:               Password of the vCenter server.
```

And `SERVER:GENERIC` artifact must specify the ESX cluster in which we are going to operate:

```
GENERAL.hostname:                   Hostname of the ESX cluster.
```

`PORT:GENERIC` artifact will now specify whether the port is accessible for the vCenter (dedicated to `VIRTUALIZATION`) or not :

```
INFO.Dedicated_To:                          VIRTUALIZATION/OTHERS
```

Another thing to take into consideration is that for vCenter Resource Pool based, we need to specify the VLAN, therefore the `NETWORK:GENERIC` attribute:

```
PROVIDER.segmentation_id:            VLAN.
```

The field PROVIDER.segmentation_id of the NETWORK:GENERIC artifact, only mandatory for vCenter Scenarios, for classic Resource Pool it could be empty.

During the execution, two more new vCenter specific artifacts will be created (so, they do not need to be included in the initial Resource Pool, just to be familiar with them):

`VSWITCH:VCENTER:` NFVDirector will manage only one vSwitch (with `NFVD` name) per ESX. An ESX, as we have said, is a server in our vCenter structure. So, in the end, only one `VSWITCH:VCENTER` artifact per Server will be created.

`VSWITCH:VCENTER` will be children of `SERVER:GENERIC`. Artifacts that could be related as vSwitch's children are:

```
PORT:GENERIC                    ACCESS (if it's Dedicated_To=VIRTUALIZATION)
PORT_GROUP:VCENTER              HAS
```

`PORT_GROUP:VCENTER` will be the child of `VSWITCH:VCENTER` and `NETWORK:GENERIC`. Artifacts that could be related as Port_Group children are:

```
VIRTUAL_PORT:GENERIC                 ALLOCATED
```

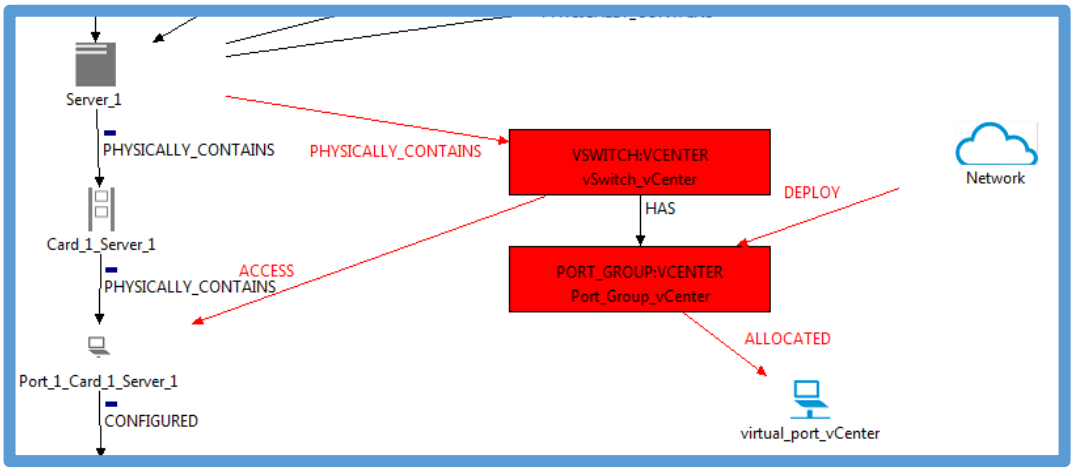


Figure 12: Detail of new artifacts and relationships generated in the deployment of vCenter.

1.6 The behavior of a Resource Pool depending on the attributes

"RP_Example_DC .nfvd"
"RP_Example_DC .xml"

To clarify what happens when, in a Resource Pool, the activation does not specify all the possible parameters in a boot call, we are going to explain three different possibilities over the same scenario, the resource pool that we are going to use as example is referenced in the top right side of this page.

To understand the scenario that we are describing, this section should be followed along with the example referenced above.

1.6.1 First case: no Availability_Zone specified, no Server specified

If you have already opened the example file, you can see that it is made up of two `REGIONS` artifact to separate zones at first level. Each `REGION` has two `AVAILABILITY_ZONE`s connected, each `AVAILABILITY_ZONE` has two `SERVER`s connected. To summarize it, we should have two `REGION`s, four `AVAILABILYTY_ZONE`s and eight `SERVER`s in our scenario.

If during a boot launching from nova, the command does not specify the `Availability_Zone`, neither is it going to specify the `SERVER`. So the command is only going to assure that the boot will take place in the `REGION` specified, this means that we will not know which `AVAILABILITY_ZONE` – reachable from the `REGION` where the boot is taking place – is the chosen to harbor the boot. In other words, the command is going to choose between two AZ and four SERVERs to carry out the booting.

In order to have the possibility to specify one or another AZ, we should fill the `GENERAL.Name` attribute of the artifact. This way we can enter our command with a clear target.

To avoid this behavior, we recommend to always provide the Resource Pool with all the mandatory attributes. The case is shown in the following figure.

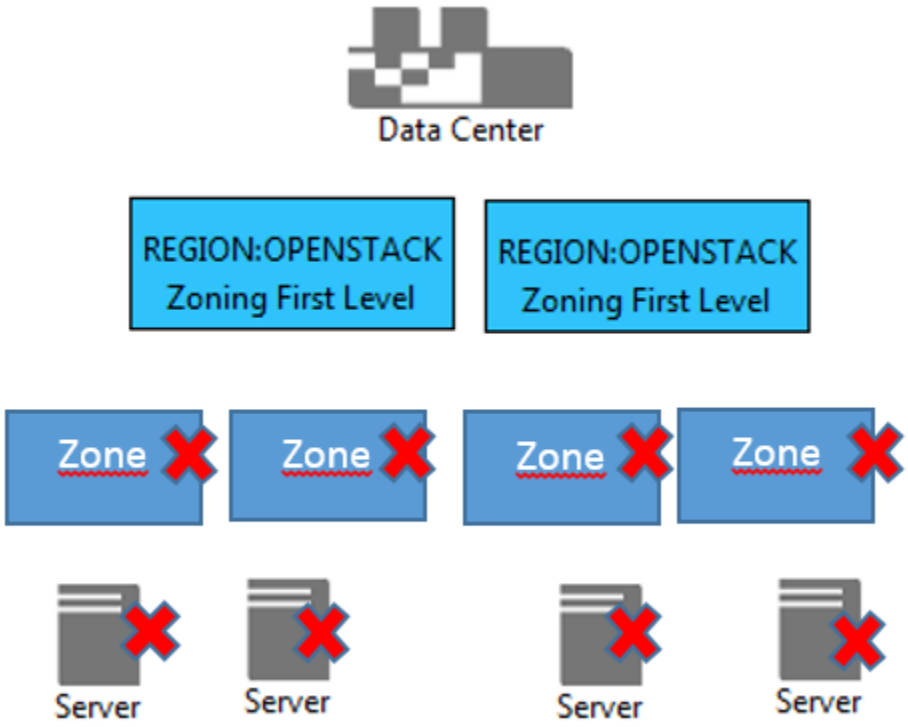


Figure 13: First case, No Zone or Server.

1.6.2 Second case: No Server specified

During a boot launching from nova, if the command does not specify the `SERVER`, the command is only going to assure that the boot it will take place in the `AVAILABILITY_ZONE` specified. This means that we are going to know in which AZ the booting is going to take place, we will also know the `REGION`. The only thing that is going to remain unknown is which server (from the two reachable from the AZ given) is the as the target of the booting.

In order to have the possibility to specify one or another `SERVER`, we should fill the `GENERAL.Name` attribute of the artifact. This way we can enter our command with a clear target.

The case is shown in the following figure.

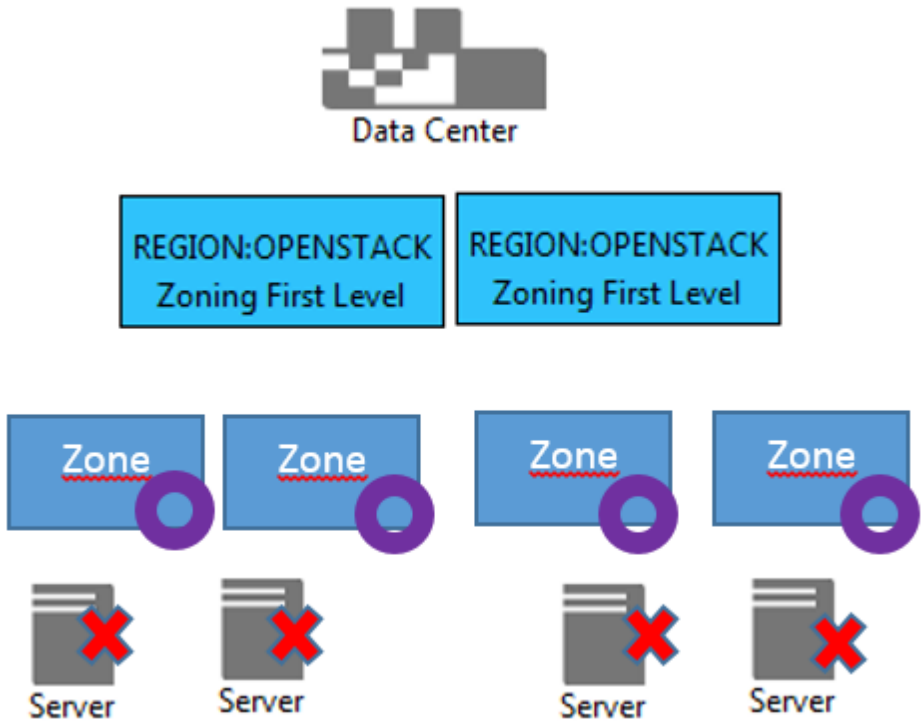


Figure 14: Second case, a valid Zone, no Server.

1.6.3 Third case: All attributes specified

In this case, all the elements necessary for the booting are present in the command. This means that REGION, AZ and SERVER are all present in the command, so no element of the booting will remain unknown.

The case is shown in the following figure.

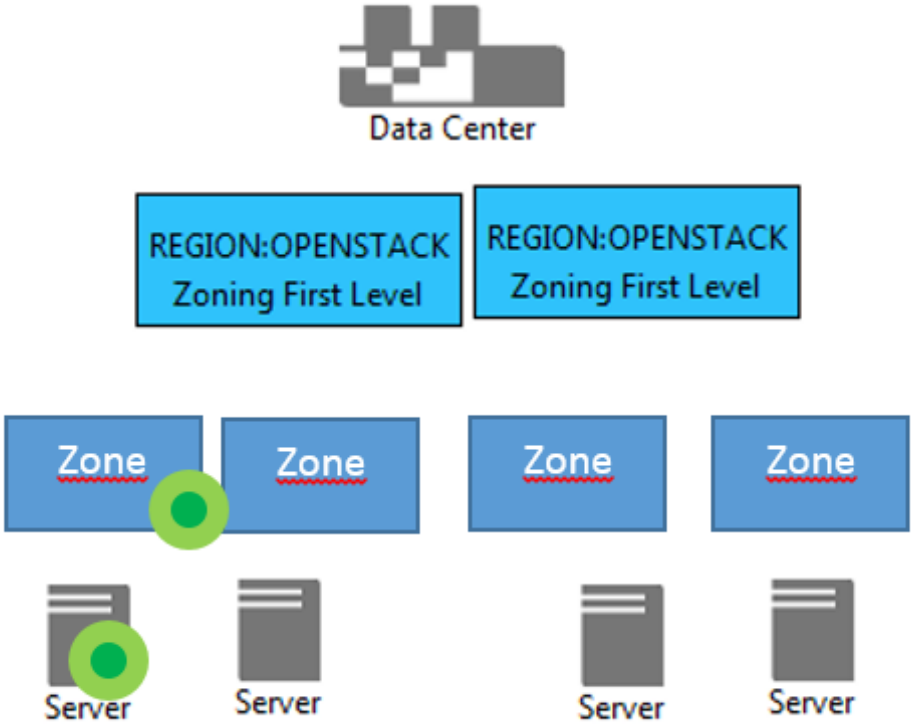


Figure 15: Third case, a valid Zone and a valid server

1.7 How the DCN allows us to communicate between Data Centers

"RP_Example_VSD_1/2/3.nfvd"
"RP_Example_VSD_1/2/3.xml"

In this section, we are going to explain how the DCN artifacts allow us to communicate between Data Centers apparently without any connection between them, this is only possible through the VSD platform, and with the implementation in the scenario of a structure that was shown before in the document, but we will reproduce it again to clarify.

In the Resource Pool examples (reference in the top right side of this page), we can see a RESOURCE_POOL with three different DATA_CENTERS without any connection between them, the first artifact to take into consideration is the NETWORKING artifact, this one is responsible for connecting our VSD structure to the rest of our Resource Pool. In this case, we have created three different NETWORKING, one per each DATACENTER. This is not necessary, and was made in this fashion to clarify, the other way to do it is sharing the NETWORKING artifact between REGIONs.

We are going to have three possible situations with our Resource Pool. The first one is that our VSD structure does not exist, so there is no solution or possibility to communicate between our `DATA_CENTERS`, the second case could be that our Resource Pool has a VSD structure and this structure brings the possibility to connect two of our `DATA_CENTERS` (represented in the "`RP_Example_VSD_2.nfvd`" file). This is achievable if our artifact `SDN_CONTROLLER` is connected to two different `NETWORKING` artifacts connected to different `REGION`s OpenStack. These artifacts must be of the adequate type to create the right type connection between elements, so the artifact `NETWORKING` could be of type `DCN`, `OPENSTACK` or a hybrid between these two.

Once this Resource Pool is in use, we should have at least one `Network` per each VIM in status `ACTIVE`. Also, we will have a DCN Network associated to the VSD structure, and this is what allows us to communicate between `DATA_CENTERS` and between each `VIRTUAL_MACHINE` connected to the `SDN_CONTROLLER` artifact in our Resource Pool. The following figure shows us how the communication between VMs it is produced.

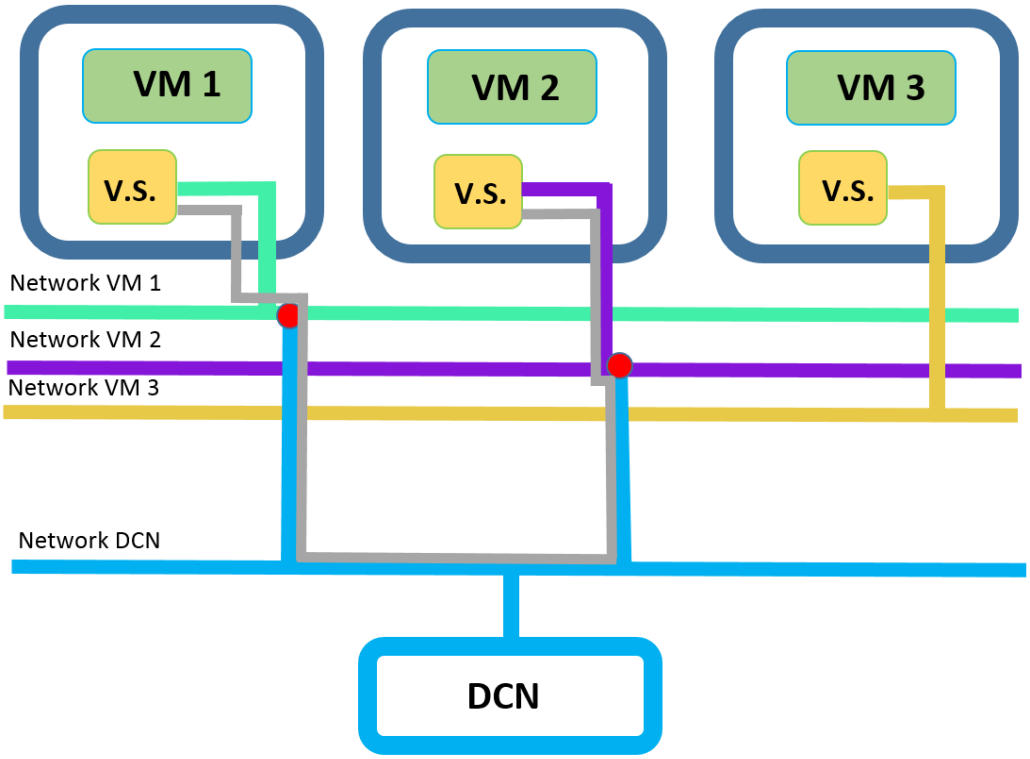


Figure 16: Diagram of the communication between two DC through the DCN Network

The third possible situation that we could encounter with our Resource Pool is that all the `DATA_CENTERS` are connected to three different `NETWORKING` artifacts connected to different `REGION`s OpenStack (represented in the "`RP_Example_VSD_3.nfvd`" file). These artifacts must be of the adequate type to create the right type of connection between elements, so the artifact `NETWORKING` could be of type `DCN`, `OPENSTACK` or a hybrid between these two. This configuration permits us to communicate the three `DATA_CENTERS` between them, the following figure shows us how the communication is produced.

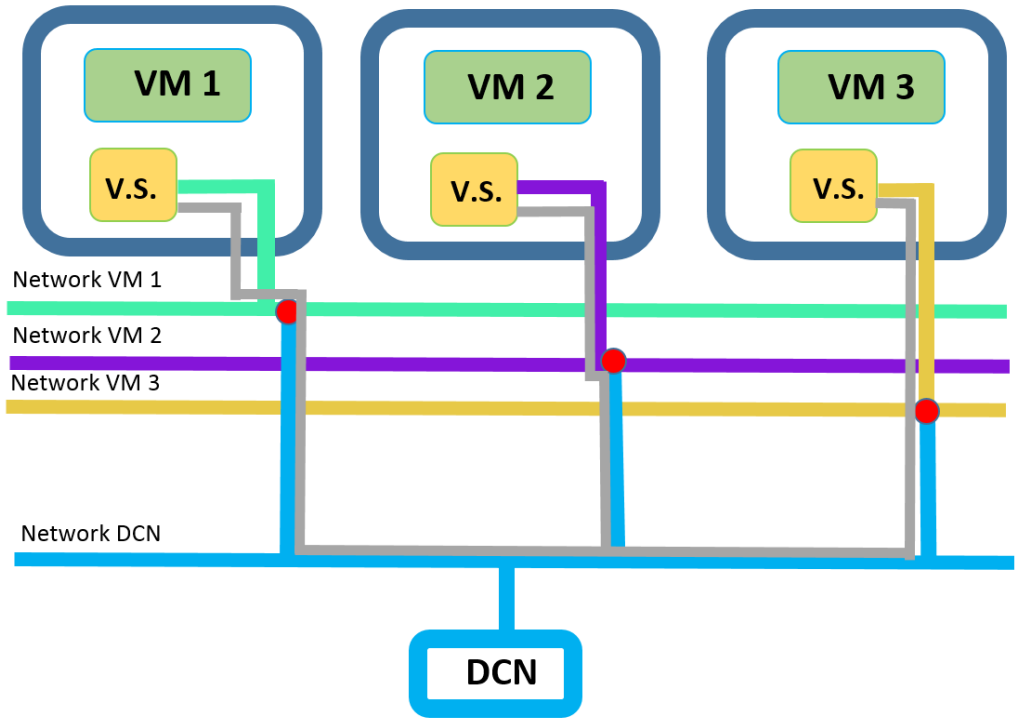


Figure 17: Diagram of the communication between three DC through the DCN Network

1.8 How to export our DC to the GUI web

The only way to have the designed Data Center visible and prepared to be used in the web is by uploading the component with the SoapUI tool through a REST Post. After the upload, the user will create the Data Center with the same tool but a different call.

1.8.1 Upload the new Data Center Template

To make our Data Center available for use in the GUI web, we should upload the component first. For that, the user must use a tool that permits the sending of REST calls. In our case, the tool used is `SoapUI`.

Our API defines the specifications of our call. In this case, the resource that we are going to use is defined by `/nfvd/product/catalog/upload`. The type of the call that is going to be executed is `POST`, because we are going to add an element to the catalog. The following image shows the window allowing us to execute the upload.

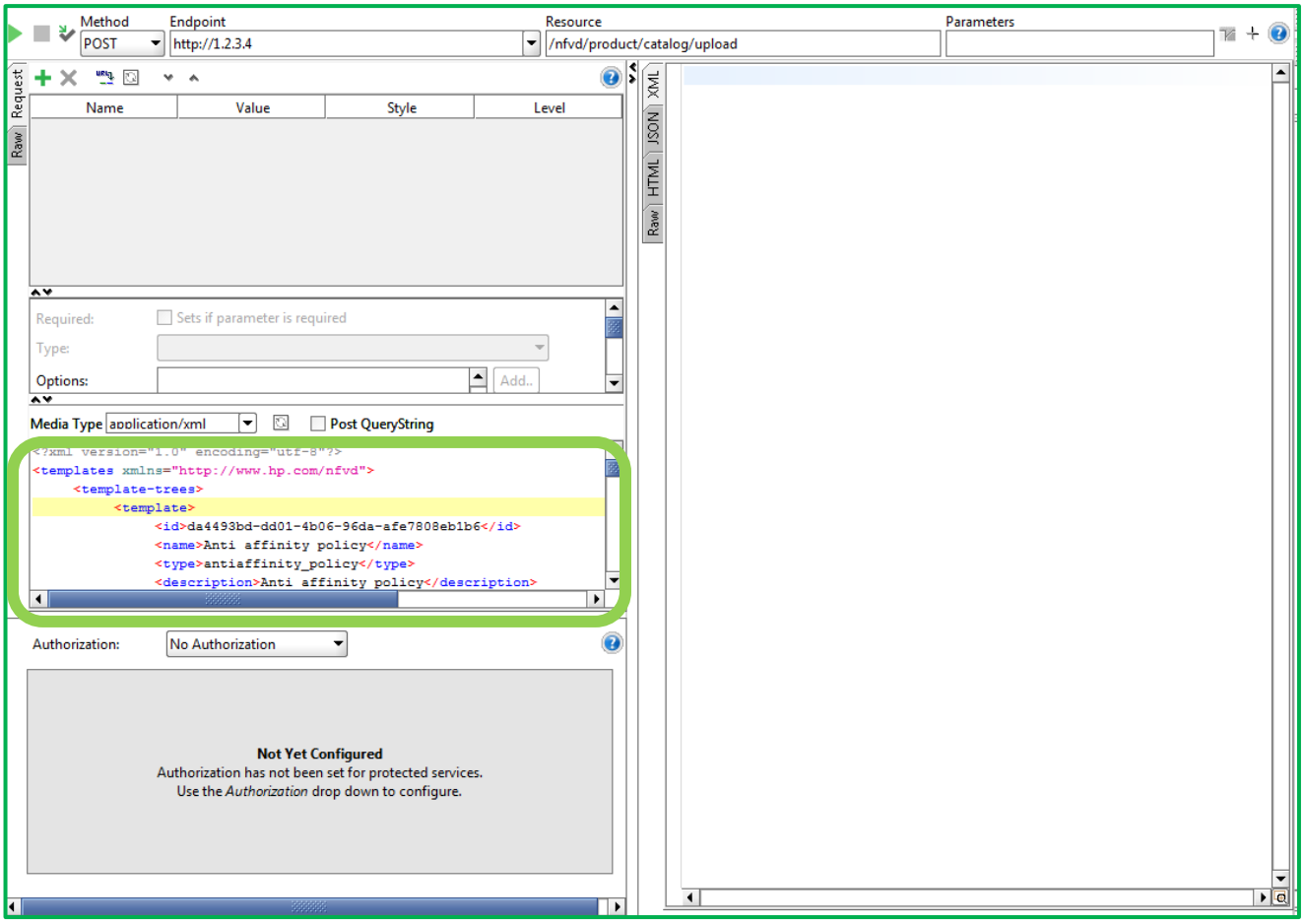


Figure 18: Uploading a new Data Center

As you can see in the image above, the Method chosen is `POST`, `Endpoint` refers to the IP address of the target. We are going to upload a component to a machine, so we must know some information about this machine. The field `Resource` has the value of the call previously mentioned.

The most important GUI area for the update is the window highlighted by a light green shape above. In this window, the user should paste the XML code that is the Data Center itself. This means that when the user designs the VNF in the offline designer called NFV Designer the user export the file to an XML. The process of exporting the Data Center is described in the *NFVDesigner User Manual*. The user should export such elements as a template, not as an instance.

Once the user has properly exported the Data Center, the user should open the `.xml` file related to the data center with an editor, and copy all the XML code to the space highlighted with the light green shape. Once all the code is pasted in the region, the user should check that there is no modification or alteration in the code, and also confirm that the code follows the code structure to be uploaded.

If the user has checked the `Method`, `Resource`, `EndPoint` and `Content`, it must be confirmed that all the headers needed for the call are present. In this case, the user is going to need an `X-token` header. The value of this token is not proportionate here for obvious security reasons.

Once all is checked, the user should click the  button, this action will start the upload, If the upload is successful, the big space in the right side of the call window will show the response code, that is, the result of the call. It also going to provide the `templateID` assigned to the `DataCenter` element once updated.

If the update was successful, the user can continue with the next subchapter. If not, the user should review the previous chapter, or the chapter on designing the datacenter.

1.8.2 Create the New Data Center from Template

To make our Data Center available for use in the web GUI, we should create an instance of the component in the system. In order to do that, the user will use a tool that permits the sending of REST calls. In our case, the tool used is SoapUI.

Our API defines the specifications of our call. In this case, the resource that we are going to use is defined by `/nfvd-ext/domains/{domainId}/operations`. The type of the call that is going to be executed is `POST`, because we are going to create an instance of an element present in the catalog. The following image shows the window that allows us to execute the creation:

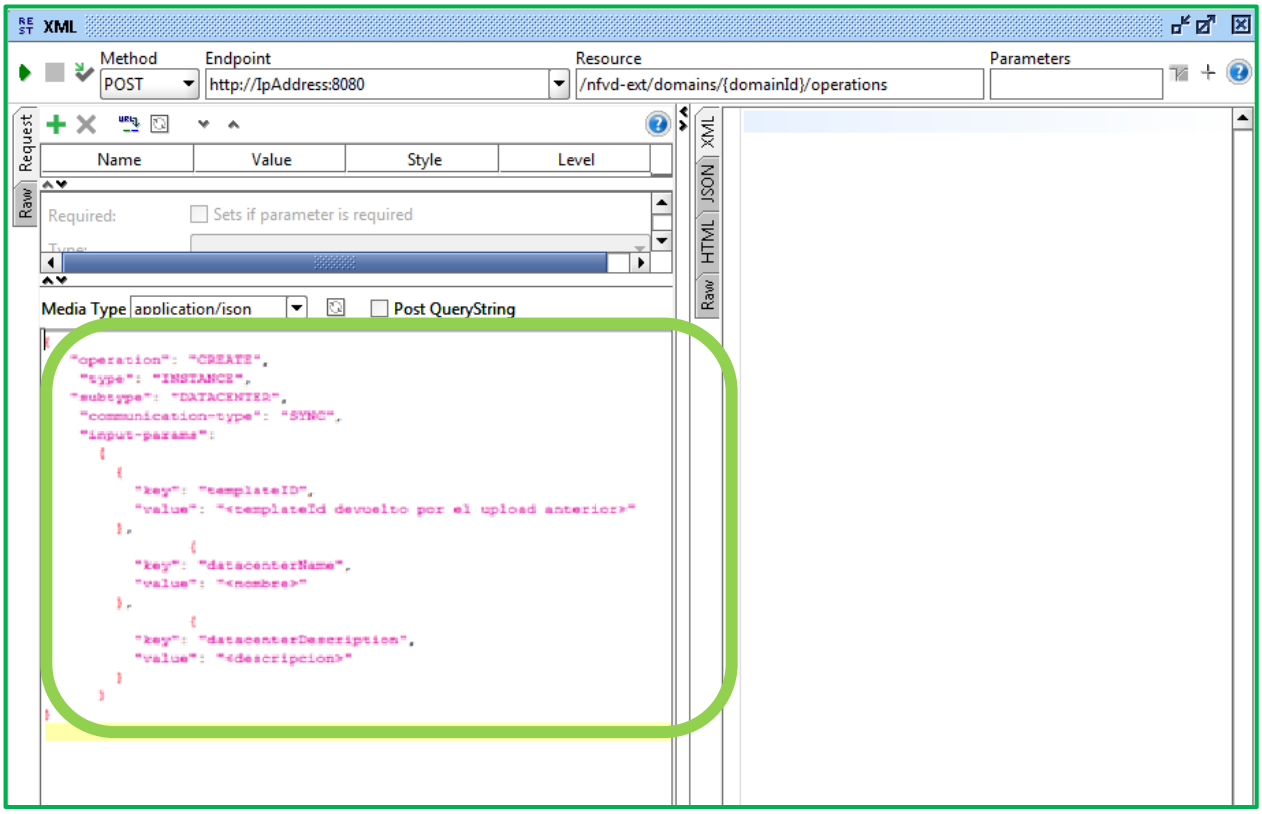


Figure 19: Creating a new Data Center from a Template file

As you can see in the image above, the Method chosen is `POST`. `Endpoint` refers to the IP address of the target. We are going to create an instance in the system, so we must know some information about this machine. The field `Resource` has the value of the call previously mentioned.

We also need to know how to get the `domainId` parameter. To obtain it, we should execute a new request of type `GET`, with the following `Resource`: `/nfvd-ext/domains`. We will configure the request as described previously. The id we need to use to create our Data Center is circled in green:

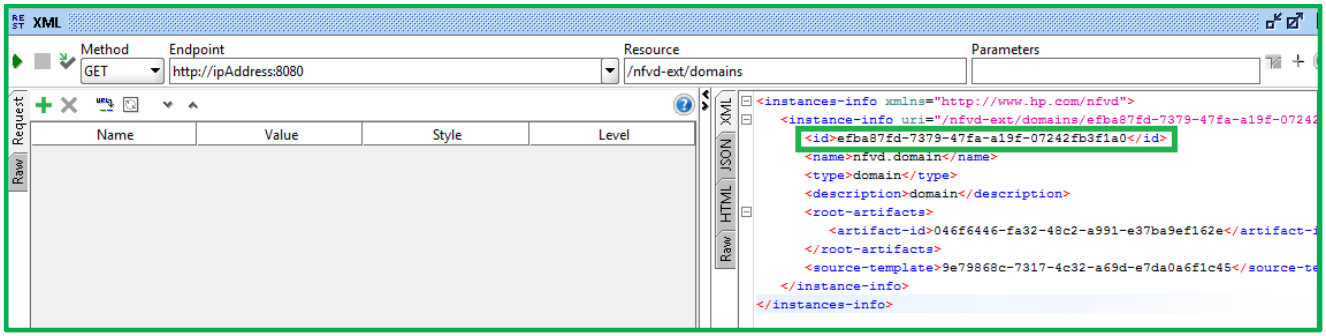


Figure 20: Locating the ID in the DataCenter

We need to copy that `domainId` and paste the value in the `domainId` field of the request `/nfvd-ext/domains/{domainId}/operations`. Once set, there is only one more another step to go to finish the creation of our datacenter.

The most important GUI area for the creation is the window highlighted by a light green shape in the following figure. In this window, the user can find the code of the call that is going to create the instance of the DataCenter in question:

```
{
   "operation": "CREATE",
    "type": "INSTANCE",
   "subtype": "DATACENTER",
    "communication-type": "SYNC",
    "input-params":
      [
        {
          "key": "templateID",
          "value": "<templateId devuelto por el upload anterior>"
        },
                {
          "key": "datacenterName",
          "value": "<nombre>"
        },
                {
          "key": "datacenterDescription",
          "value": "<descripcion>"
        }
      ]
}
```

Figure 21: Example of a “create instance” operation

There are three fields related to this call:

`"key": "templateID":`	In this field, the user should write the “templateId” that the upload call throws once executed. It is an ID in UUID format.
`"key": "datacenterName":`	The name of the Datacenter, present in the XML file associated to the component.

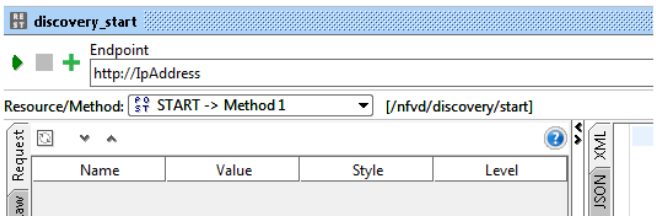
`"key": "datacenterDescription":` The description of the Datacenter, present in the XML file associated with the component.

If the user has checked `Method`, `Resource`, `EndPoint` and `Content`, it must be confirmed that all the headers needed for the call are present. In this case, the user is going to need an `X-token` header. The value of this token is not proportionate here for obvious security reasons.

Once everything is checked, the user should click the button, this action will start the creation. If the creation is successful, the big space in the right side of the call window will show the response code, that is, the result of the call.

If the creation was successful, the user will see the Data Center available in the web GUI, fully available to the users with the right permissions and scope.

In some cases, the upload requires a "Manual data load", which will fix the problem of visibility. In this case, to perform a "Manual Data load", we must send a new REST request with the SoapUI. The method should be `POST`, the endpoint will be the IP address of the machine where our solution is installed. The Resource will be `/nfvd/discovery/start`. We do not need to add body to the request. Once executed, wait for a few seconds for the discovery of elements, and then launch another REST request to stop the discovery. The configuration will be the same for both requests, the only difference will be the resource/method used. To stop, use `nfvd/discovery/stop`.



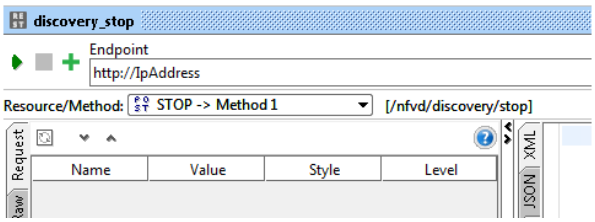


Figure 22: Starting and stopping discovery

1.8.3 Create the New Data Center from Instance

To have our Data Center available for use in the web GUI, first we must upload the component. To do that, the user will use a tool that permits the sending of REST calls. In our case, the tool used is SoapUI.

Our API defines the specifications of our call. In this case, the resource that we are going to use is defined by `/nfvd/discovery`. The type of the call that is going to be executed is `POST`, because we are going to add an element to the catalog. The following image shows the window that allows us to execute the upload:

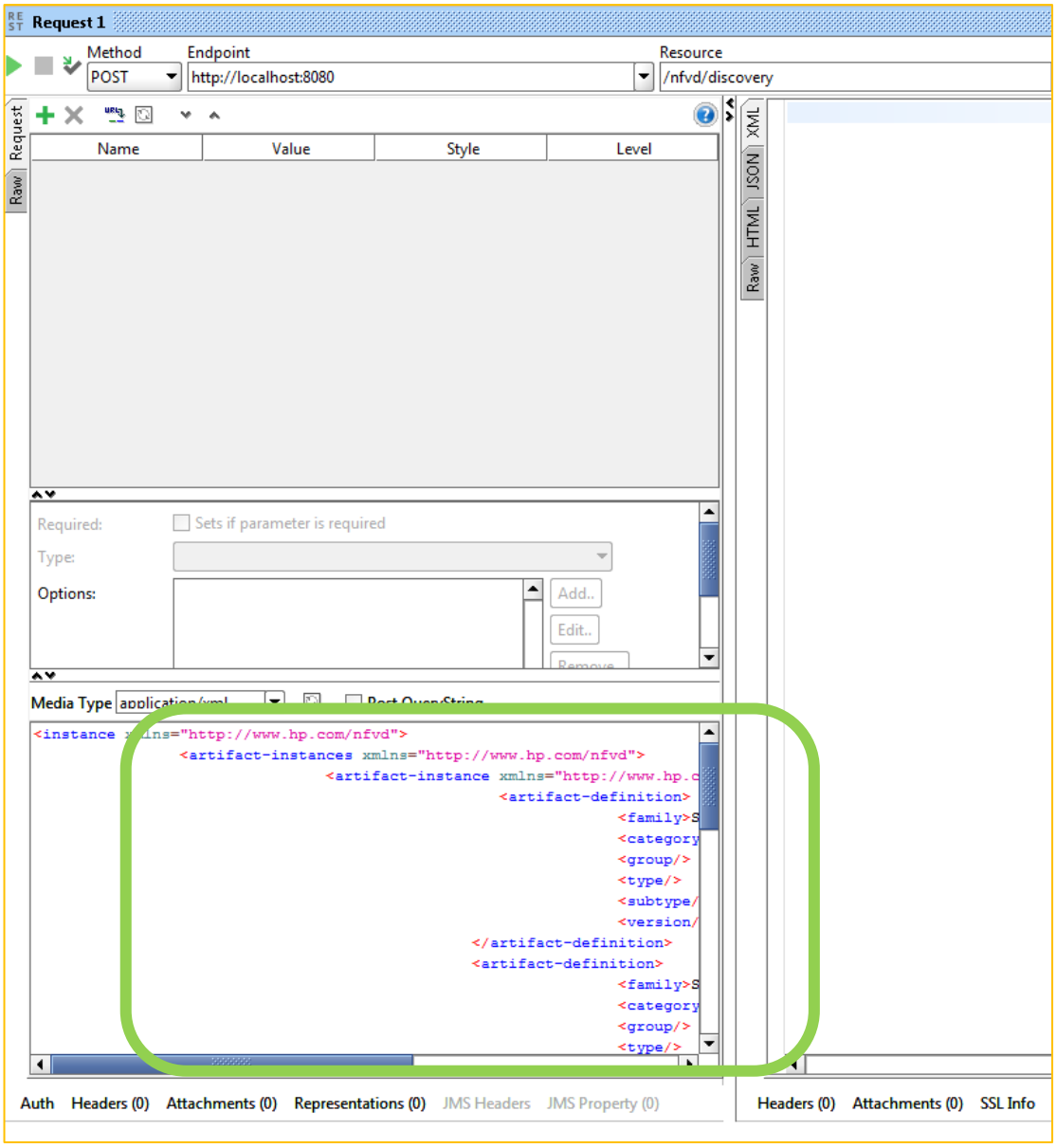


Figure 23: Creating a new Data Center from an Instance

As you can see in the image above, the Method chosen is `POST`, `Endpoint` refers to the IP address of the target. We are going to upload a component to a machine, so we must know some information about this machine. The field `Resource` has the value of the call previously mentioned.

The most important GUI area for the update is the window highlighted by a light green shape above. In this window, the user should paste the XML code that is the Data Center itself. This means that when the user designs the VNF in the offline designer called "NFV Designer" the user export the file to an XML. The process of exporting the Data Center is described in the *NFVDesigner User Manual*. The user should export such elements as a template, not as an instance.

Once the Data Center has been properly exported, the user should open the .xml file related to the data center with an editor. The user needs to change the first tag or enclose the XML code of the element by the lines:

```
<instance xmlns="http://www.hp.com/nfvd">
… (XML code of the element)
</instance>
```

This should enclose in the new tags all the XML code that is defined inside the tags `<artifact-instances>` and `<relationship-instances>`, the result should look like to the following code:

```
<instance xmlns="http://www.hp.com/nfvd">
<artifact-instances xmlns="http://www.hp.com/nfvd">
<artifact-instance xmlns="http://www.hp.com/nfvd">
<artifact-definition>
<family>SERVER</family>
<category>GENERIC</category>
<group/>
<type/>
<subtype/>
<version/>
</artifact-definition>
<categories>
…
</categories>
<status>
<label>ENABLED</label>
<visible-label>ENABLED</visible-label>
<enabled>true</enabled>
</status>
<id>4da20e8b-e9e0-31b8-b032-4ea9c6012ae5</id>
<identifier>SERVER:9e484330465db5c93c7ca2bc79664032</identifier>
<physical>true</physical>
<creation-timestamp>2016-01-20T11:44:54.532+0100</creation-timestamp>
<update-timestamp>2016-01-20T11:44:54.532+0100</update-timestamp>
</artifact-instance>
…
</artifact-instances>
<relationship-instances xmlns="http://www.hp.com/nfvd">
<relationship-instance xmlns="http://www.hp.com/nfvd">
<relationship-type>PHYSICALLY_CONTAINS</relationship-type>
<parent-artifact-id>2886f716-15c3-3f15-8e51-5f0b50b9a19b</parent-artifact-id>
<child-artifact-id>89459be7-7a35-3882-b526-e0f8b5972442</child-artifact-id>
<status>
<label>ENABLED</label>
<visible-label>ENABLED</visible-label>
<enabled>true</enabled>
</status>
<creation-timestamp>2016-01-20T11:44:54.532+0100</creation-timestamp>
<update-timestamp>2016-01-20T11:44:54.532+0100</update-timestamp>
</relationship-instance>
…
</relationship-instances>
</instance>
```

After editing, the user will copy all the XML code to the space delimited by the light green shape. Once all the code is pasted in the region, the user should check that there is no modification or alteration in the code, and also that the code respects the code structure to be uploaded.

If the user has checked the `Method`, `Resource`, `EndPoint` and `Content`, it must be confirmed that all the headers needed for the call are present. In this case, the user is going to need an `X-token` header. The value of this token is not proportionate here for obvious security reasons.

Once all is checked, the user should click the  button, this action will start the upload, If the upload is successful, the big space in the right side of the call window will show the response code, that is, the result of the call.

Once this operations is finished, the user will need to start the discovery service. To do that, the user should launch a `POST`, the `Endpoint` refers to the IP address of the target. In this case, to start the discovery service, the user will use the operation `/nfvd/discovery/start`. After a few seconds, the user should execute the operation `/nfvd/discovery/stop`. These operations will make the discovery service discover our new element.

If the discovery was successful, the user can continue with the next subchapter. If not, the user should review the previous chapter, or the chapter on designing the datacenter.

1.9 NFV Director: VIM executions

1.9.1 Introduction

In our scenarios, our off-line designer differentiates between six types of different VIMs, these types are:

```
VIM:CS8           Based on OpenStack Havana, obsolete.
VIM:GENERIC       A VIM that can be used as one of any kind depending on the
                  attributes given.
VIM:HELION        HPE OpenStack version, variant: Helion HCG (Carrier Grade).
VIM:ICEHOUSE      VIM for scales in OpenStack ICEHOUSE.
VIM:KILO          VIM for deployment in OpenStack Kilo.
VIM:OPENSTACK     VIM OpenStack.
```

Each VIM operates in a specific version, and almost all of them have the majority of their attributes common. For example, the VIMs `ICEHOUSE`, `KILO`, `HELION` and `OPENSTACK` are all VIMs OpenStack. This means that all of them work in a very similar way on the OpenStack platform, but have slight differences depending on the operation that they perform.

1.9.2 The “Template” category

In all the artifacts available in our offline designer, the ones that use OpenStack modules always have the category “Template”. This category has three attributes: `Manufacturer`, `Element_type` and `Version`. There is a good reason for naming this category as “Template”. When our system is trying to activate, a VIM will check these attributes and will try to find a OpenStack template that matches the three values on the category “Template”. With the three attributes, our system generates a file name that is going to represent the template that the user is trying to find. In the system, there is a list of possible templates over different modules of OpenStack. The composing of this file follows a set of strict rules. The goal of composing the file name this way is to always find a specific template file.

If the platform can match the template of the user’s component module (this is the OpenStack module the user is pretending to use), our VIM will start to make use of that module in the platform with all the features the module can deliver. Each version has modifications that make the module more suited for specific functions, or a feature is changed to enhanced.

Ideally, each of the templates always match an available template. If a search for a correct template for the artifact present in our DC uses the specific template identifying the use of a specific OpenStack Project was not successful, the system will use a default template for the module that is going to be used by the specific artifact.

If the search for a valid template file was not successful, the system will use a default VIM. This, in the case of the system, is a `VIM:OPENSTACK`. One mismatch in the template's search is enough to change the user's VIM to the default VIM.

In the current version, the system allows the use of `VIM:HELION` , this VIM type is specific for Helion Carrier Grade machines (HCG).

The three Template category attributes are `Manufacturer`, `Element_type` and `Version`. They are used to generate a specific and unique filename that allows the VIM to use specific module projects in the OpenStack platform.

In our offline designer, there are a wide range of artifacts that use OpenStack modules, or have the access to using of some OpenStack modules. The artifacts that use these modules and the modules used are:

```
NETWORKING::      Neutron. (Whatever the version, or manufacturer).
COMPUTE::         Nova.
IMAGE_STORAGE::   Glance.
STORAGE::         Cinder.
```

The list above reflects the most used artifacts using OpenStack projects. The first name on the list is the name the artifact has in the offline designer. The column of the right defines the name of the OpenStack project used by the artifact. To have a proper use of the module in the platform OpenStack, the system will generate a file name that should match one of the template files present in the system.

These template files in the system are named to match the specifications of the OpenStack platform on each module. The VIM's modules template files have a name that is created from the attributes of the category "Template" of each artifact.

The template files have similar names:

CREATE_SERVER_HP_NOVA_HELIONCG_v1.1.template

As you can see, the name of the file is a concatenation of the command that we execute and the name of the manufacturer followed by the element type, ending with the version.

Now we are going to explain the how each "Template" category must be filled on each of the artifacts that use the previously listed OpenStack projects.

1.9.2.1 Neutron Project - Networking

Neutron is an OpenStack project to provide "networking as a service" between interface devices managed by other OpenStack services (Nova). The Neutron project has an elevated number of plugins that allows us to use it along with a wide range of Network devices.

In our case, and as we explained before, we usually use a `VIM:HELION` in this version, so, by default, our `NETWORKING` artifacts are going to have `"Manufacturer" : "HP"` as a value for the attribute, but for the other two attributes of the category, the system offers two options depending on the type of the artifact that can be used.

In our system, the user has two types of Neutron modules at their disposition. One Neutron, represented by the artifact `NETWORKING:OPENSTACK`, and another Neutron represented by the artifact `NETWORKING:OPENSTACK:DCN`. If the user chooses the first one, all the networking task are going to be developed by an OpenStack Neutron Generic module. Remember that we are talking about networking modules of a HPE Helion OpenStack, so this module is customized from an OpenStack Neutron module. In case the user chooses the second type, all the Networking tasks are left to the DCN, through its artifact `SDN_CONTROLLER`.

The attributes for a `NETWORKING:OPENSTACK` would be:

Table 2: NETWORKING:OS Template Attributes.

Category.attribute	OpenStack	Helion/Helion CG
`TEMPLATE.Manufacturer:`	`OPENSTACK`	`HP`, or the name of the manufacturer of the module.
`TEMPLATE. Element_Type:`	`Neutron`, we are using a plain OpenStack module, not usual in this version.	`Neutron`, we are using a customize neutron module by HPE.
`TEMPLATE. Version:`	This will be the value of the Neutron project, in the OpenStack platform that the user is aiming to use.	`HELIONCG_v1.1`, it will be something similar, the name of the OpenStack used, and the version available.

The attributes for a `NETWORKING:OPENSTACK:DCN` would be:

Table 3: NETWORKING:OS:DCN Template Attributes.

Category.attribute	OpenStack	Helion/Helion CG
`TEMPLATE.Manufacturer:`	`OPENSTACK`	`HP`
`TEMPLATE. Element_Type:`	`Neutron`, we are using a plain OpenStack module, not usual in this version.	`DCN` we are using a module DCN, developed by HPE, that delegate all the networking task in the element `SDN_CONTROLLER` and its related artifacts.
`TEMPLATE. Version:`	`KILO`, the new one is `LIBERTY`. This will be the actual version of the OpenStack platform.	`v3.2`, `v4.0` it will be something similar, the version number of the Element type.

1.9.2.2 Nova Project - Compute

Nova, OpenStack Compute service is used for hosting and managing cloud computing systems. It is a component-based architecture enabling quicker additions of new features. It is fault tolerant, recoverable and provides API-compatibility. In our case, and as we explained before, we usually use a `VIM:HELION` in this version, so, by default, our `COMPUTE` artifacts are going to have `"Manufacturer": "HP"` as the value of the attribute. The `Element_type` will usually be `NOVA`, and the attribute `Version` depends on the version of OpenStack platform used.

The attributes for a `COMPUTE:OPENSTACK` would be:

Table 4: COMPUTE:OS Template Attributes.

Category.attribute	OpenStack	Helion/Helion CG
`TEMPLATE.Manufacturer:`	`OPENSTACK`	`HP`
`TEMPLATE. Element_Type:`	`Nova`, we are using a plain OpenStack module, usual in this version and artifact.	`Nova`, we are using a plain OpenStack module, usual in this version and artifact.
`TEMPLATE. Version:`	`KILO`, the new one is `LIBERTY`, This will be the actual version of the OpenStack platform.	`HELIONCG_v1.1`, if we are using a HCG OpenStack. If we were using a different OpenStack we should introduce the name and version of the platform used.

1.9.2.3 Glance Project - Images

The OpenStack's module **GLANCE**, allows us to provide discovery, registration, and delivery services for disk and server images. The artifact related to this module in our offline designer is the `IMAGE_STORAGE:OPENSTACK`.

In our case, and as we explained before, we usually use a `VIM:HELION` in this version, so, by default, our `IMAGE_STORAGE` artifacts are going to have `"Manufacturer": "HP"` as the value of the attribute, the `Element_type` will usually be `GLANCE`, and the attribute `Version` depends on the version of the OpenStack platform used. It could be generic from the version of the platform OpenStack, or could be the name and version of the HPE OpenStack version.

The attributes for an `IMAGE_STORAGE:OPENSTACK` would be:

Table 5: IMAGE_STORAGE:OS Template Attributes.

Category.attribute	OpenStack	Helion/Helion CG
`TEMPLATE.Manufacturer:`	`OPENSTACK`	`HP`
`TEMPLATE. Element_Type:`	`Glance`, we are using a plain OpenStack module, usual in this version and artifact.	`Glance`, we are using a plain OpenStack module, usual in this version and artifact.
`TEMPLATE. Version:`	`KILO`, the new one is `LIBERTY`, This will be the actual version of the OpenStack platform.	`HELIONCG_v1.1`, if we are using a HCG OpenStack. If we were using a different OpenStack we should introduce the name and version of the platform used.

1.9.2.4 Cinder Project –Storage.

Cinder is a Block Storage service for OpenStack. It is designed to present storage resources to end users that can be consumed by the OpenStack Compute Project (Nova). This is done by using either a reference implementation (LVM) or plugin drivers for other storage. The short description of Cinder is that it virtualizes pools of block storage devices and provides end users with a self-service API to request and consume those resources without requiring any knowledge of where and on what type of device their storage is physically deployed. The artifact related to this module in our offline designer is `STORAGE:OPENSTACK`.

In our case, and as we explained before, we usually use a `VIM:HELION` in this version, so, by default, our `STORAGE` artifacts are going to have "Manufacturer": "HP" as the value of the attribute. The `Element_type` will usually be `CINDER`, and the attribute `Version` depends on the version of OpenStack platform used. It could be generic from the version of the platform OpenStack, or could be the name and version of the HPE OpenStack version.

The attributes for a `STORAGE:OPENSTACK` would be:

Table 6: STORAGE:OS Template Attributes.

Category.attribute	OpenStack	Helion/Helion CG
`TEMPLATE.Manufacturer:`	`OPENSTACK`	`HP`
`TEMPLATE. Element_Type:`	`Cinder`, we are using a plain OpenStack module, usual in this version and artifact.	`Cinder`, we are using a plain OpenStack module, usual in this version and artifact.
`TEMPLATE. Version:`	The last one `KILO`, the new one is `LIBERTY`, This will be the actual version of the OpenStack platform.	`HELIONCG_v1.1`, if we are using a HCG OpenStack. If we were using a different OpenStack we should introduce the name and version of the platform used.

1.9.3 Difference between versions of VIMs.

The VIMs are constantly changing, so we are always going to have new versions and new features available. This guide is about the features and functionalities of version 4.0 of the NFV Director product, but the VIMs listed above work with previous versions, too.

`VIM:CS8` is obsolete. It used to work with version 3.0. However, it is still in the system as some of its processes are still needed.

`VIM:ICEHOUSE` belongs to version 3.0, and is also near obsolete.

`VIM:KILO` belongs to version 4.0, the only remarkable difference is that in a `VIM:KILO`, the Virtual Ports associated should be deleted by the user.

`VIM:GENERIC` and `VIM:OPENSTACK` respond to the same necessity: `VIM:GENERIC` allows to configure a VIM to be used as any other VIM, and `VIM:OPENSTACK` is used to have a `VIM:GENERIC` for the OpenStack Platform.

Finally, `VIM:HELION` and its variant `Helion Carrier Grade`: The Helion VIM is the success achieved by HPE to get a fully functional OpenStack VIM. The differences between a simple Helion and a Helion CG are:

Helion: Only standard Flavors, no Extra Specs allowed.

Helion CG: Flavor with Extra Specs allowed.

Creation of ports happens immediately, there are three types of ports to work with.

1.9.4 Check the attributes of the VIMs.

This section will help the user to check if the VIM is perfectly configured for use.

Normally, a VIM will only need the right credentials to be configured. At the first connection to the Openstack platform, all the fields will have to be filled. Not only in the VIM element, but also in the `AUTHENTICATION` artifact that carries out the logging on in the platform. We should remark that this is not a common action, the manual configuration of a VIM or its `AUTHENTICATION` artifact is not recommended.

The artifact `AUTHENTICATION` will require a set of correct credentials:

- `Url:` The Url of the Openstack Platform where the user wants to be logged in.
- `Login:` The user login.
- `Password:` The password for the login.
- `TenantName:` The name of the Tenant "admin" of the platform, this is the element that act as admin of the Openstack platform.
- `UserId:` This attribute will be filled during the creation of the VDC.
- `IdentityVersion:` The version of the identity module that is used by the platform.

After the first connection to the platform, some fields of the artifact `AUTHENTICATION` will be filled with the information harvested form the Openstack platform. These attributes are:

- `GENERAL.UserId:` The ID of the administrator user.
- `ROLES.AdminRoleId:` The ID of the role of the admin.
- `ROLES.MemberRoleId:` The ID of the role of a member.

In case we need to manually update these attributes, we will need to follow the steps listed below.

We will need to send a REST request from our SoapUI (or any tool that we can use for this purpose). As Endpoint, the request should have the IP address of the machine that hosts the solution. As Resource/Method, we will use a `POST`, and as Url for the request we will aim for the identity module of the platform, `https://IDENTITY:5000/v2.0/tokens`, where "IDENTITY" will be the specified IP address for the keystone admin url. Also, we should make sure that the request contains the right headers:

- Accept: application/json
- Content-Type: application/json

Once we have configured our request, we will copy in the body the following code:

```
{
    "auth": {
        "tenantName": "Client's tenant name",
        "token": {
            "id": "Identifier of the token."
        }
    }
}
```

To execute the request correctly, we need to know the `tenantName` (admin) and the token ID. The token ID is used constantly in the solution in most of the requests executed.

Once executed, we should look for the following at the bottom of the response body:

```
……
     ],
        "user": {
            "username": "admin",
            "roles_links": [],
            "id": "1f568815cb8148688e6ee9b2f7527dcc",
            "roles": [
                {
                    "name": "service"
                },
                {
                    "name": "admin"
                }
            ],
            "name": "admin"
        },
        "metadata": {
            "is_admin": 0,
            "roles": [
                "8341d3603a1d4d5985bff09f10704d4d",
                "2e66d57df76946fdbe034bc4da6fdec0"
            ]
        },
        "trust": {
            "id": "394998fa61f14736b1f0c1f322882949",
            "trustee_user_id": "269348fdd9374b8885da1418e0730af1",
            "trustor_user_id": "3ec3164f750146be97f21559ee4d9c51",
            "impersonation": false
        }
……
```

The `GENERAL.UserId` will correspond with the text "id": "1f568815cb8148688e6ee9b2f7527dcc"

The `ROLES.AdminRoleId` would be "8341d3603a1d4d5985bff09f10704d4d"

The `ROLES.MemberRoleID` would be "2e66d57df76946fdbe034bc4da6fdec0"

The ID will not be the same, obviously, but this code indicates the position of the attribute's value in any response for this type of request.

Once we have identified the values of the attributes, we can set them in the `AUTHENTICATION` element.

We will continue checking the VIM. For the VIM we need to configure four attributes of two categories:

`GENERAL.Name:` The name of the VIM.

`GENERAL.Host:` The IP address of the machine that hosts our Openstack Platform.

`CREDENTIALS.Login:` The login name for our VIM.

`CREDENTIALS.Password:` The password for our VIM.

If these attributes exist, and the `AUTHENTICATION` artifact is correctly configured, the VIM will be modified upon its first connection to the Openstack Platform (if we have configured correctly both artifacts inside our Datacenter).

Remember that this is NOT the usual way to configure our artifacts and elements, this actions should be carried out with extreme caution, a bad configuration of these artifacts will lead to a disastrous behavior of the solution.

Chapter 2
NFV Director: On Board to a VNF

2.1 Introduction

This chapter of the document provides guidance through the creation of more complex components. We are going to treat a component as a set of artifacts that compose the component.

The modelling of a component can be a tough task. This document is intended intention to offer an introduction and a guide to the creation of different types of VNF with different specifications.

To achieve our goal, we cover what artifacts take part in the creation of a VNF (Virtual Network Function) and what values must be set in their attributes to get the expected behavior.

- You are going to see a structure of elements identified by their Definition artifact, with a specific and unique ID. That has a number of categories with a set of attributes.
- These artifacts are connected to each other with relationships of different types to achieve a specific target respecting the hierarchy of execution.
- The Component has a tree structure. Also, the different artifacts can be grouped in trees or subtrees. Each tree must have a unique Root Artifact.

2.2 Understanding the elements of a VNF

The first element to take in consideration when you are about to model a VNF is the VNF artifact. This is the main element of the VNF. It also contains the name of the VNF, the one that it is going to be used during all the processes where the artifact it is going to be included.

A VNF is a highly customizable element. If we take for example the number of possible VNFs in function of the type, we find that the designer allows us to create four different types of VNF elements, Generic, NFVD , LB, and FW. The first one is what we consider a default VNF, the second type is a VNF preset with some attributes useful for its future use, the two last types are special VNFs, with a specific goal to achieve. In those cases, the FW develops a Firewall role, and the Load Balancer role balancing the traffic load between networks.

A VNF is connected at least to two artifacts, commonly, a `VNF_Endpoint` and a `VNF_Component` artifact. These artifacts are crucial for the behavior of the VNF. In the case of the endpoint, the component needs an access point for the rest of the elements that will need to communicate with the VNF during the coming process. `VNF_Component` acts as a list that connect the VNF and the VM, this means that our VNF (main element) could have one or more `VNF_Component`s that list one or more `Virtual_Machine`s. If the VNF grows following this pattern, the functionality of the element will not be harmed. The `VNF_Component` works as a Virtual Machine classifier.

To be of use, every `End_Point` must be connected to a Virtual Port. This Virtual Port is the element that is going to be used to configure the connection between the VNF and whatever the component/element/artifact/entity that is in the other end of the `End_Point`. This means that, when the Virtual Port is being configured, a specific port number and name will be assigned to the artifact, making the Virtual Port suitable for use.

Virtual Machine is the artifact that gives the VNF component its functionality. The Virtual Machine artifact is the target of the majority of the processes available in the system. It is also the final component. That is, normally, when some action take place in the component or VNF, the final objective is some modification over a Virtual machine.

The minimum components that a Virtual Machine must have are `Virtual_Disk`, `Virtual_Core`, and `Virtual_memory`. It is plain to see that the components respond to the needs of a computer.
Intuitively, the user can guess that the attributes to configure in this artifacts are going to be amounts of Gbs or Mbs, or the amounts of cores, and so on.

The component `Virtual_Core` has a set of attributes that should be controlled to develop the expected behavior. During the activation of the Virtual Machine in some controlled process, the attributes of the category `INFO`, `CORE_ARCHITECTURE`, `NUMA_ID`, `SHARED_VCPU` are very important. they will be checked to guarantee that the values meet a set of specific requirements, it usually is a combination of the values of the three attributes. The way these attributes should be filled depends on the Virtual Machine activation, what type of image is going to be used in that specific Machine, which file is going to be used, what conditions dictated by the environment must be met, and so on.

After this brief description of an elemental VNF, we are in position to continue with the creation of more complex components.

All the components present in this chapter are `GENERIC`.

2.3 How to model a simple VNF

2.3.1 Introduction

TIP: To properly follow this paragraph, please open the file `RP_Example_VNF_00.nfvd` with the resource modeler tool. The file is in the same folder as the document. Examples: `RP_Example_VNF_00.nfvd`, `RP_Example_VNF_00.xml`.

To achieve a functional model of a VNF, we will go over each step of the development process discussing the different possibilities and combinations of the artifacts.

If you already have opened the file mentioned above, you can see the first element of the model. This is a VNF. To make the artifact more legible the `identifier` was set to `Example_VNF_00`. In the categories of the artifact, you can see that the `GENERAL.Name` is equal to the identifier given to the VNF. These two are the only crucial attributes of this element. The VNF is connected to a `VNF_Component`, with a relationship of type `INCLUDE`. The VNF has the possibility to contain more than one `VNF_Component`s. To configure the `VNF_Component` there is no specific requirement, only that it must be present in the scenario with status `INSTANTIATED`, ready to be used.

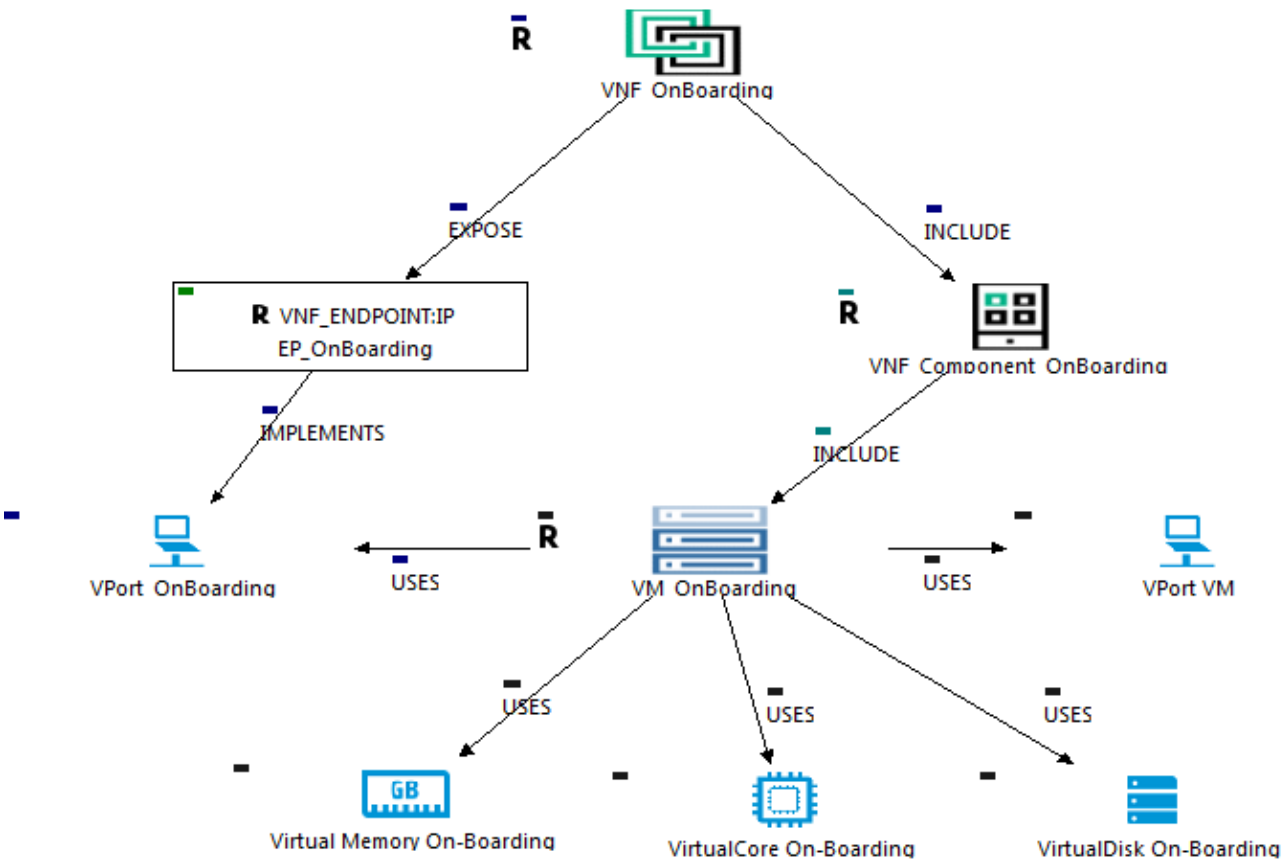


Figure 24: RP_Example_VNF_00.xml

The artifact `VNF_Endpoint` must be present in the design, functioning over the connection of the VNF with the rest of the elements implied in the different future processes. The EP will act as the point of connection between the VNF and the outside artifacts, making VNF an isolated component.

`Virtual_port` is an artifact that is usually configured by the element that takes possession of it. This means that the configuration of the attributes is going to take place when needed, or during the activation of the `Virtual_port` parent element. Although the configuration is normally unattended (the customization is available in the web GUI), we will explain the main attributes of a Virtual Port.

The two main categories of the `Virtual_port` artifact are `INFO` and `SDN`, the category `INFO` is references attributes that are going to be part of the OpenStack platform. Attributes such as `ID`, `Name`, `Type`, `MAC`, `Ipadress`, `Network Type`, `Physical_PORT_ID` or `PCI_SLOT` are mostly filled by the system. In case the user – for some arbitrary reason – wants to fill some of the attributes mentioned, the user must make sure that the values that are about to be set are the right ones. For example, the attribute `ID` should be filled with the ID of the element in the platform OpenStack, any other attribute will result in faulty operation. Other attributes, such as `Physical_Port_ID` or `PCI_SLOT` will be filled or not, depending on the use of the Virtual Port.

The category `SDN` has the same function as the category `INFO`, but it is used by the VSD platform.

The Virtual Machine element has a wide list of categories:

```
GENERAL:          The general attributes of the component.
STATUS:           Attributes about the state or possible states of the
                  component.
VIM:              Attributes to specify with more accuracy, information about
                  the
                  VIM using the VM.
HYPERVISOR:       Attributes to specify with more accuracy, information about
                  the HYPERVISOR using the VM.
LAST_OPERATION:   Info about the last operation executed.
CREATION_MODE:    Info about some conditions during the creation of the
                  component.
CARRIER_GRADE:    Attributes used when the VM is of type carrier grade.
SCALE:            Attributes for the escalation process of the Virtual Machine.
CONFIG:           Attributes to configure user data treatments.
CREDENTIALS:      Attributes related to the Admin credentials.
KEYPAIR:          Attributes related to the KEYPAIR validation.
INTEGRATION:      Attributes that carry punctual information for specific
                  processes.
```

The category to needs an explanation for a beginner approach is `GENERAL`. In this category, the user is going to find seven attributes:

```
Name:                   Name of the Virtual Machine.
Type:                   Type of the Virtual Machine, quite important.
Description:            Description of the Virtual Machine.
Hostname:               Info about the hostname.
Management_access:      In case of having a Management Network that use this
                        VM.
Image                   Type of image to be used in the VM, quite important.
ImageOSName:            In case the image is associated to an image present in
                        OpenStack.
```

The attributes `Type` and `Image` are crucial for the correct behavior of the machine. `Type` sets the type of the image that is going to be used. These types could be mainly `KVM` and `VCENTER`, but the number of possible images suitable to be used over a VM is far bigger than two, depending on the usage of the machine and the characteristics of the platform that we are using.

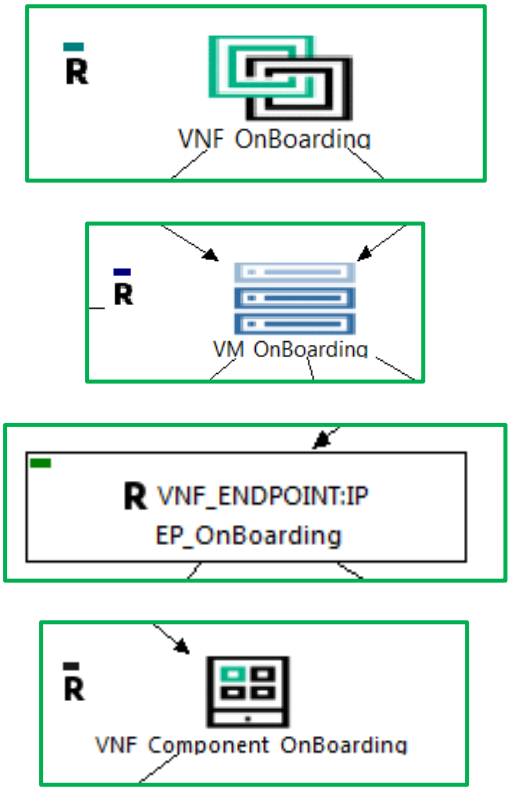
The attribute `Image` will be used to find the image in the OS platform, so the user should make sure that the image is present in the platform and the value of the attribute matches with the name in the platform.

The artifacts needed to make a Virtual Machine fully functional are Virtual Memory, Virtual Disk, and Virtual Core at minimum. The needs of these artifacts are trivial, a `VMemory` needs an `Amount` and a `Page Size` value. In this case, the amount is the size of the memory in Mb, and the `PAGE_SIZE` is the size of each page.

For the Virtual Core, only the `Amount` is needed, which refers to the number of cores in the element.

For the Virtual Disk, is only necessary to set the `Amount` of Gb that the disk is going to have. Each Virtual Disk artifact only represents one disk, despite of the size set.

As you can see in the general image of the component, this VNF has four `RootArtifacts`, so we are going to find at least four trees.



These are the `RootArtifact`s present in the VNF that is being designed. As you can see, the VNF artifact is `RootArtifact` of the VNF's tree. The Virtual Machine is the `RootArtifact` of the Virtual Machine's tree, the VNF_EndPoint is a special case, because is `RootArtifact` and unique member of the VNF_Ep's tree, and the `VNF_Component` artifact is `RootArtifact` of the `VNF_Component`'s tree.

All the future designs done with this version of the system should follow this pattern, each main element should be `RootArtifact`, each element and relationship should be part of a tree or subtree, a tree cannot have more than one `RootArtifact`, and an element could be part of more than one tree. Such this cases should be reviewed thoroughly.

2.4 The elements of the VNF

2.4.1 Introduction

In this section, we are going to show the most common elements used in our VNFs.

2.4.2 VNF Artifact

The VNF element is the main element of the VNF component. This element must always be the root of the main tree of the component.

A VNF is a Virtual Network Function. The possibilities and different configurations that this definition allows should help the user to understand the vast number of possible combinations.

A VNF only has sense if the user is focused in its components, the number, type and configuration of this components define which function is going to be developed by the VNF.

Attributes that should be filled:

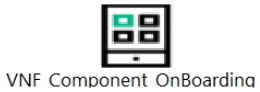
```
GENERAL.Name      Name that is going to be used for the VNF, editable.
GENERAL.Type      Type of the VNF that is going to be displayed, editable.
```

Artifacts that could be related with the artifact:

```
VNF:GENERIC                    INCLUDE
VNF_COMPONENT:GENERIC          INCLUDE
VNF_COMPONENT:SITESCOPE        INCLUDE
CONDITION:GENERIC              MEASURE
LINK:GENERIC                   MEASURE
NETWORK:GENERIC                USES
NETWORK:OPENSTACK              USES
NETWORK:EXTERNAL               USES
NETWORK:INTERNAL               USES
NETWORK:FRONTEND               USES
NETWORK:BACKEND                USES
POLICY:ENTITY_RANGE            APPLY
POLICY:VALUE_GENERATION        APPLY
POLICY:VALUE_VALIDATION        APPLY
POLICY:AFFINITY                APPLY
POLICY:ANTI_AFFINITY           APPLY
POLICY:OVER_SUBCRIPTION        APPLY
POLICY:OVER_SUBCRIPTION        APPLY
POLICY:POSTPRE_PROCESSING      APPLY
VNF_ENDPOINT:BIDIRECTIONAL     EXPOSE
VNF_ENDPOINT:IP                EXPOSE
```

2.4.3 VNF_Component Artifact

The VNF Component element is a classifier of Virtual Machines. There is no important configuration of the element needed, only its presence in the VNF is the component that is used as point of access to a list of VMs related to it.



A VNF component is a Virtual Network Function component. This artifact is considered as an extension of the VNF. It guarantees the access to a specific set of Virtual Machines. It also allows the design to be leveled, and to create another layer below the VNF.

Attributes that should be filled:

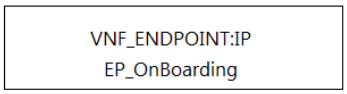
```
GENERAL.Name      Name that is going to be used for the VNF component,
                  editable.
GENERAL.Type      Type of the VNF component that is going to be displayed,
                  editable.
```

Artifacts that could be related with the artifact:

```
VNF_COMPONENT:GENERIC              INCLUDE
VIRTUAL_MACHINE:GENERIC            INCLUDE
VIRTUAL_MACHINE:KVM                INCLUDE
VIRTUAL_MACHINE:VMWARE             INCLUDE
PHYSICAL_MACHINE:GENERIC           INCLUDE
CONDITION:GENERIC                  MEASURE
NETWORK:OPENSTACK                  USES
NETWORK:EXTERNAL                   USES
NETWORK:INTERNAL                   USES
NETWORK:FRONTEND                   USES
NETWORK:BACKEND                    USES
POLICY:ENTITY_RANGE                APPLY
POLICY:VALUE_GENERATION            APPLY
POLICY:VALUE_VALIDATION            APPLY
POLICY:AFFINITY                    APPLY
POLICY:ANTI_AFFINITY               APPLY
POLICY:OVER_SUBCRIPTION            APPLY
POLICY:OVER_SUBCRIPTION            APPLY
POLICY:POSTPRE_PROCESSING          APPLY
```

2.4.4 End_Point Artifact.

The `End_Point` artifact is the point of access of the VNF element for the rest of the entities or elements. Normally, an `End_Point` has one Virtual Port connected through it, the `VNF_EndPoint` must be present on the element. Although no configuration is needed, the presence of the artifact is crucial. Otherwise, the element will be created but it will be isolated from the rest of the system.



An `End_Point` requires no configuration, but the user can edit the `Name` and `Description` as usual. For rest of the attributes, the decision is left to the user. However, we recommend not to edit what you do not know about.

Artifacts that could be related with the artifact:

```
ENDPOINT:GENERIC              CONNECTED
ZONE:INTERNAL                 CONNECTED
ZONE:EXTERNAL                 CONNECTED
ZONE:FRONTEND                 CONNECTED
ZONE:BACKEND                  CONNECTED
LINK:GENERIC                  IMPLEMENTS
```

2.4.5 Virtual_Port Artifact

The Virtual Port artifact works along with the `End_Point` or the `Virtual_Machine`, or any element allowed to have a Virtual Port for its use. `End_Point` and `Virtual_Machine` are ones of the most used. An `End_Point` requires no configuration to be used, but the user should choose the proper type when it is created and included in the VNF. Depending on what it will be used for, our `End_Point` could be of seven kinds:

VPort OnBoarding

```
VIRTUAL_PORT:BACKEND
VIRTUAL_PORT:FRONTEND
VIRTUAL_PORT:EXTERNAL
VIRTUAL_PORT:INTERNAL
VIRTUAL_PORT:MANAGEMENT
VIRTUAL_PORT:PRIVATE
VIRTUAL_PORT:PROVIDER
VIRTUAL_PORT:BACKEND
```

An Virtual Port requires of configuration, but the user can edit the `Name` and `Description` as usual. For the rest of the attributes, the decision left to the user, but remember that the Virtual Port could be used by more than one element. The element takes possession of the artifact and it is configured as the system dictates depending on the needs of the process required.

Artifacts that could be related with the artifact:

```
INTERFACE:GENERIC        CONFIGURED
VLAN:DCN                 ASSIGNED
```

2.4.6 Virtual_Machine Artifact

The Virtual Machine artifact represents the Machine that is going to be activated. It is a main artifact in the VNF element. To have it properly configured the Virtual Machine, the user should fill the fields related to the image that the machine is going to use during the activation. These are the type of image and the file that is going to be used.

VM OnBoarding

A Virtual Machine must be configured properly. To be activated, a Virtual Machine needs to have two things configured. A specific type of the image that is going to be activated (that also defines what type of Virtual Machine is once it has been activated), and a specific type of file for that image. Also, a Virtual Machine must be related to all the artifacts required to be used with all the guarantees. So, the Virtual Machine should have at least a Virtual Core, Virtual Memory and Virtual Disk artifact, all of them properly set.

Attributes that should be filled:

```
GENERAL.Name       Name that is going to be used for the Virtual Machine,
                   editable.
GENERAL.Type       Type of the Image that is going to be used in the VM,
                   editable.
GENERAL.Image      Type of the Image file that is going to be used in the VM,
                   editable.
```

Artifacts that could be related with the artifact:

```
VIRTUAL_CORE:GENERIC          USES
VIRTUAL_MEMORY:GENERIC        USES
VIRTUAL_DISK:GENERIC          USES
VIRTUAL_PORT:MANAGEMENT       USES
VIRTUAL_PORT:GENERIC          USES
VIRTUAL_PORT:EXTERNAL         USES
VIRTUAL_PORT:INTERNAL         USES
VIRTUAL_PORT:FRONTEND         USES
VIRTUAL_PORT:BACKEND          USES
VIRTUAL_LUN:GENERIC           USES
POLICY:ENTITY_RANGE           APPLY
POLICY:ENTITY_ASSIGN          APPLY
POLICY:ENTITY_SCALE           APPLY
POLICY:VALUE_GENERATION       APPLY
POLICY:VALUE_VALIDATION       APPLY
POLICY:AFFINITY               APPLY
POLICY:ANTI_AFFINITY          APPLY
POLICY:OVER_SUBCRIPTION       APPLY
POLICY:POSTPRE_PROCESSING     APPLY
```

2.4.7 Virtual_Memory Artifact

The Virtual Memory artifact is the provider of memory to be used by the Virtual Machine. It is easy to understand and plain to see that the important attributes are the ones that customize the memory, basically, the size and amount of the memory.

Virtual Memory On-Boarding

The two attributes that must be configured in the Virtual Memory artifact are both of category `INFO`. In this category, the user should fill the attributes `Amount` and `PAGE_SIZE`. The first one refers to the size of the memory in Mb, the second one refers to the size of each page in the memory, and is set in Kb. Of course, the user can edit the `Name` and `Description` as usual.

Attributes that should be filled:

```
INFO.Amount        Size in Mb that the memory will have at the end of the
                   process.
INFO.PAGE_SIZE     Size in Kb that each page of the memory is going to have.
```

The memory only can be parent of a policy of type `Entity Scale`. A policy that dictates the rules for an escalation operation over the Virtual Memory.

2.4.8 Virtual_Core Artifact.

The Virtual Core artifact is the provider of the computing for the Virtual Machine. It is easy to understand and plain to see that the attributes important are the ones that customize the disk, basically amount of the disk. But the artifact also has attributes that should be configured depending on the use of the Virtual Machine. These attributes are `CORE_ARCHITECTURE`, `NUMA_ID` and `SHARED_VCPU`.

The attribute that must be configured in the Virtual Core artifact refers to the physic specifications, the attribute is `Amount` and it refers to the number of Cores that the Virtual Core is going to have. Of course, the user can edit the `Name` and `Description` as usual.

The attributes that should be configured based on the use of the Virtual Machine can take different values:

```
CORE_ARCHITECTURE:      It is plain to see that depending of the value of this
                        Attribute, the Core of the Virtual Core will behave in
                        one way or another. The two possible values that this
                        attribute can take are shared and dedicated. The first
                        one means that the core could be shared in some
                        situations, the second means that it designates the
                        Virtual Core to be used only by the VM.
NUMA_ID:                Can only take two values, which are necessary when
                        speaking of NUMA machines. The values are 0 and 1.
SHARED_VCPU:            This attribute sets the number of Virtual CPUs that are
                        going to be shared in case the CORE_ARCHITECTURE
                        attribute has the indicated value.
```

2.4.9 Virtual_Disk Artifact

The Virtual Disk artifact is the provider of storage to the Virtual Machine. It is easy to understand and plain to see that the important attributes are the ones that customize the disk, basically, the amount of the disk.

The attribute that must be configured in the Virtual Disk artifact refers to the physical specifications. The attribute is `Amount`, and it refers to the size of the Virtual Disk, in Gb.

Of course, the user can edit the `Name` and `Description` as usual.

2.5 Examples of VNFs

2.5.1 Introduction

Through this section, our intention is to explain the possibilities to create a wide range of VNFs with the offline designer. This part of the Onboarding Guide is designed to be followed by examples.

Example_VNF_00.xml
Example_VNF_00.nfvd

2.5.2 Example 01: The simplest VNF

The shortest set of elements that could be considered a VNF is the one that was showed in chapter 2.3.1, in Figure 1. In the following figure, you can see the reason why this is the simplest VNF. Form the artifact `VNF OnBoarding`, only two relationship are visible, one of type `EXPOSE` and one of type `INCLUDE`. Also, you can see that all the elements in the image belong to the same group, except for `EP_OnBoarding`, which should have its own tree, and which must be the `RootArtifact` of its tree. This VNF is simple, because we only have one `EndPoint`. Also, our VNF only includes one element.

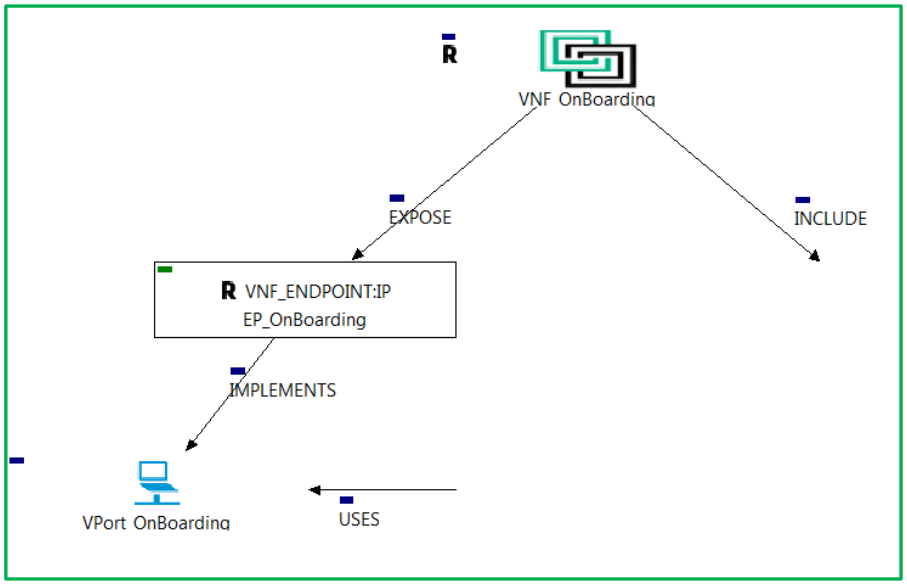


Figure 25: VNF Group tree

In the following image, we extend the image of the elements by adding the tree of the `VNF_Component`.

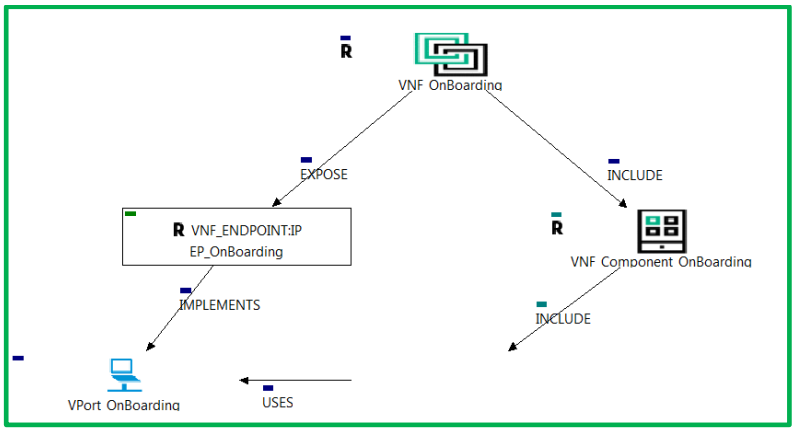


Figure 26: VNF Group tree plus VNF Component Group tree

As you can see, `VNF_Component OnBoarding` is the only element `INCLUDE`d in the VNF. These is also a characteristic of a simple VNF. Notice that `VNF_Component OnBoarding` must be the `RootArtifact` of its own tree. This tree should include the relationship between the `VNF_Component` and its elements. In this case, this `VNF_Component` only includes one element. In this case, and normally in all the cases, it is going to be a Virtual Machine. In the following image, we add the Virtual Machine tree.

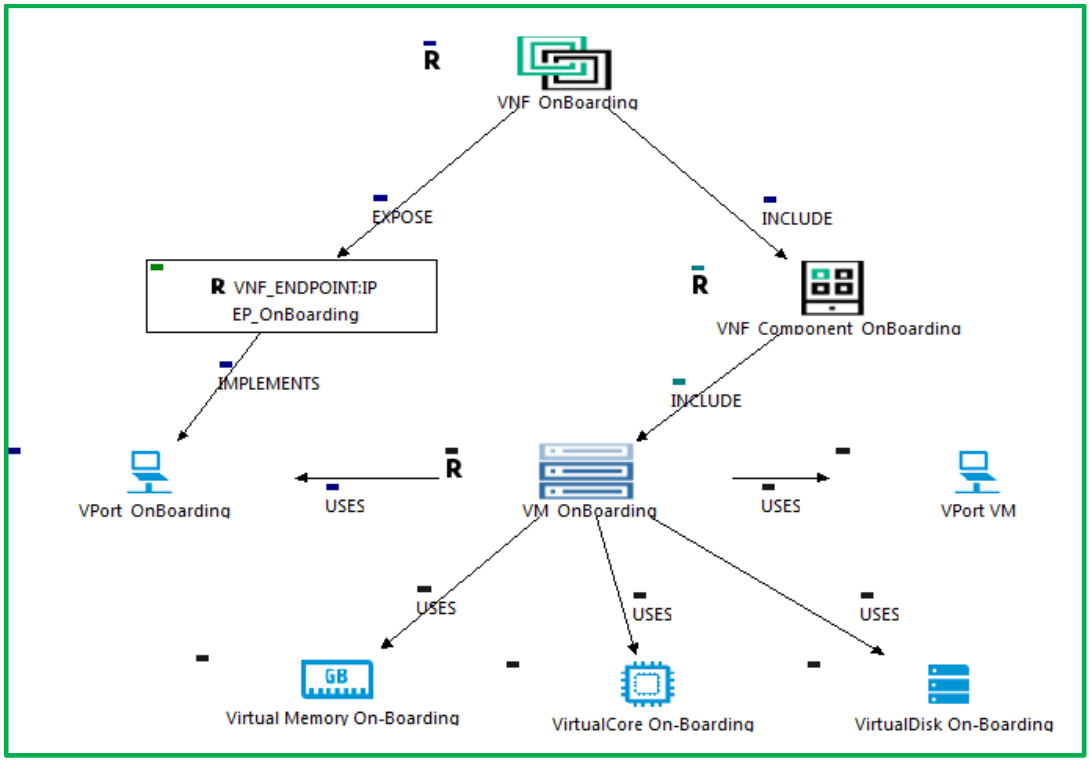


Figure 27: Group trees of a VNF element

As you can see, the image shows the group of the Virtual Machine along the rest of the elements of the component. The group of the Virtual Machine should have all the elements associated to the VM, such as `Virtual Disk`, `Virtual Core`, `Virtual Memory`, and so on. Also, the VM may need a Virtual Port to be connected to networks and subnetworks. This is the case that the image above shows. As you see, the VM must be the `RootArtifact` of its own tree.

When you finish editing your VNFs, you should be totally sure that each element is member of at least one tree. Otherwise it is very likely that some operations will end unsuccessfully.

In the following sections, we introduce the user to more complex VNFs.

2.5.3 Example 02: VNF, Networks & Subnetworks.

"Example_VNF_01.xml"
"Example_VNF_01.nfvd"

A second approach to the creation of VNFs is the VNF with Network and Subnetworks. In this case, we also create another VNF component with another Virtual Machine and another `End_Point`, as you can see in the image below:

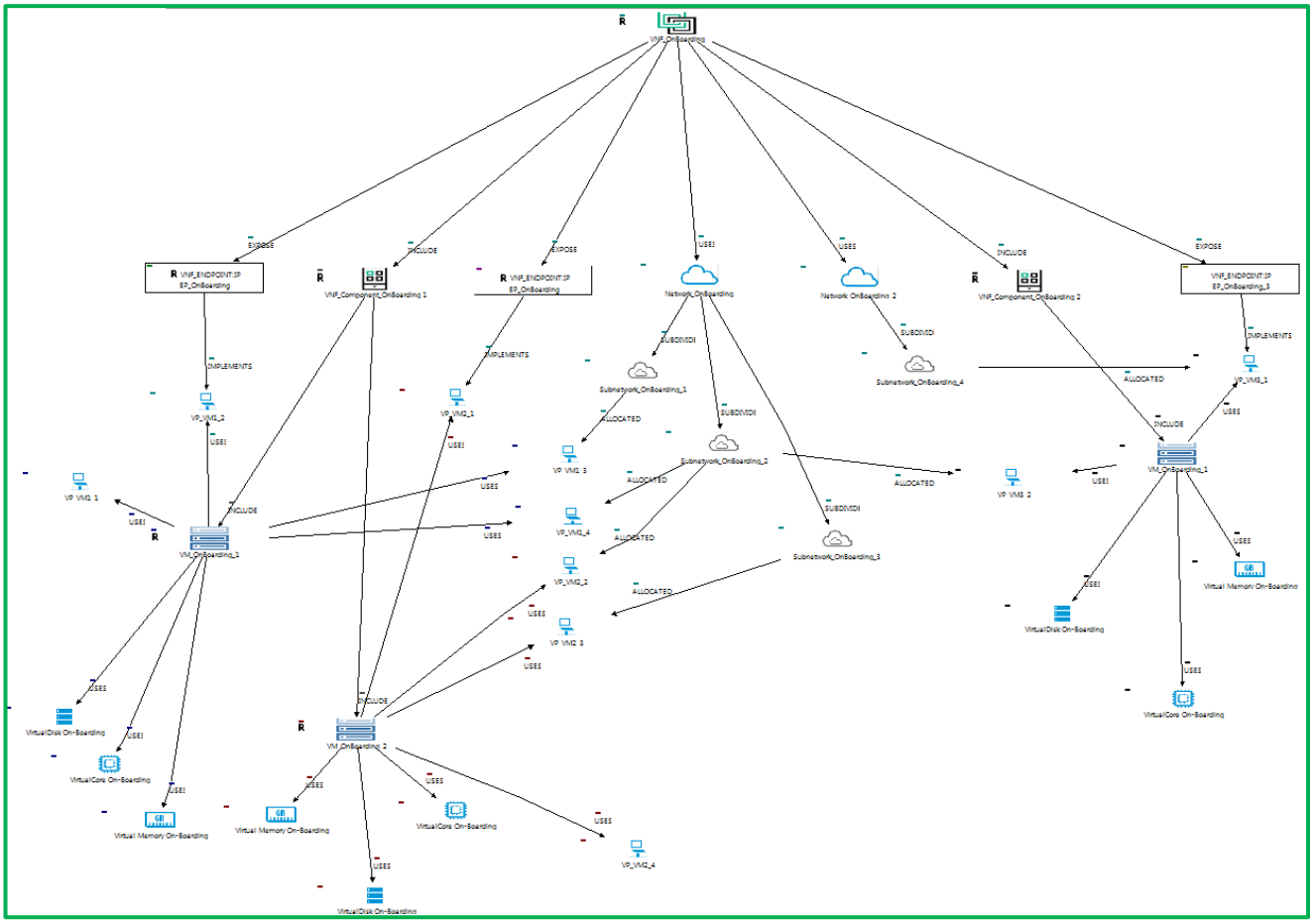


Figure 28: VNF, Networks & Subnetworks

To include a Network or Subnetwork in our VNF, we need to create the artifact, and it will be connected to the VNF directly. This way, the VNF will use the Network. If the Network must have Subnetworks (which is highly recommended) they will be directly connected to a Network that will be subdivided by these Subnetworks. For this VNF, we only have a Subnetwork per Network and only one Network in all the VNF.

As you can see, our two Virtual Machines are connected to the same Subnetwork, through two different Virtual Ports. It is possible to connect the two VM to the same Virtual Port, but it is preferable to allow one Virtual Port per connection between entities. It is the same approach as the one used with the EndPoints and the VM. They are connected by a Virtual Port, and each entity acts on one side of the Virtual Port.

Another possibility to connect a Network to a VM is to connect the Subnetwork directly using the Virtual Port between the EndPoint and the Virtual Machine, as the following image shows.

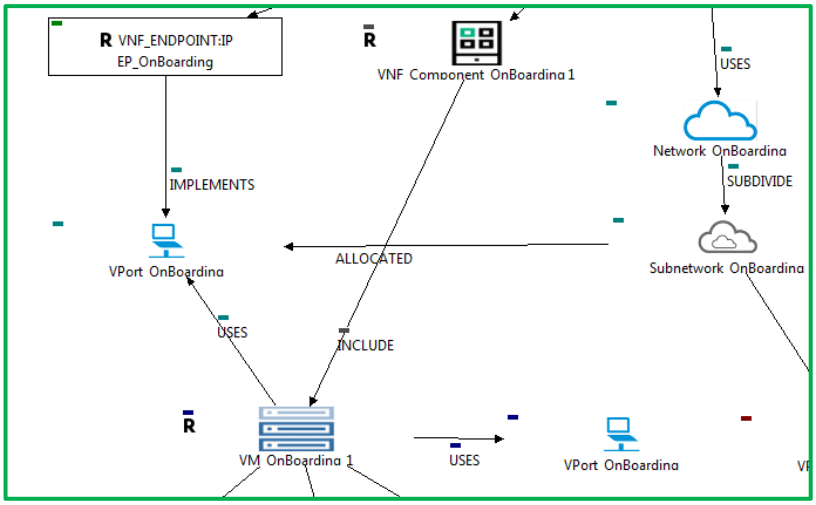


Figure 29: Relationship between VPort and Subnetwork

The user should keep some limitations in mind regarding the connections between EndPoints and Virtual Machines and other entities.

An EndPoint can have more than one Virtual Ports that are going to be used to connect with different elements. So an EndPoint could give service to more than one Virtual Machines through these Virtual Ports.

A Virtual Machine is going to be connected to one (and only one) EndPoint only through a Virtual Port.

A subnetwork will be connected to a Virtual Port. That only happens if we want to connect to the entity that is connected to that Virtual Port.

In the following subsection, a more complex VNF is shown.

2.5.4 Example 03: A not so simple VNF

"Example_VNF_02.xml"
"Example_VNF_02.nfvd"

In the VNF described by the image below, we can see a VNF that has two VNF Components. One of them with two Virtual Machines and the other one only with one Virtual machine. Also, the VNF has three EndPoints, one per each Virtual Machine present in VNF, as well we can see two Networks, one of them has three subnetworks associated, and the other one has only one Subnetwork related to it.

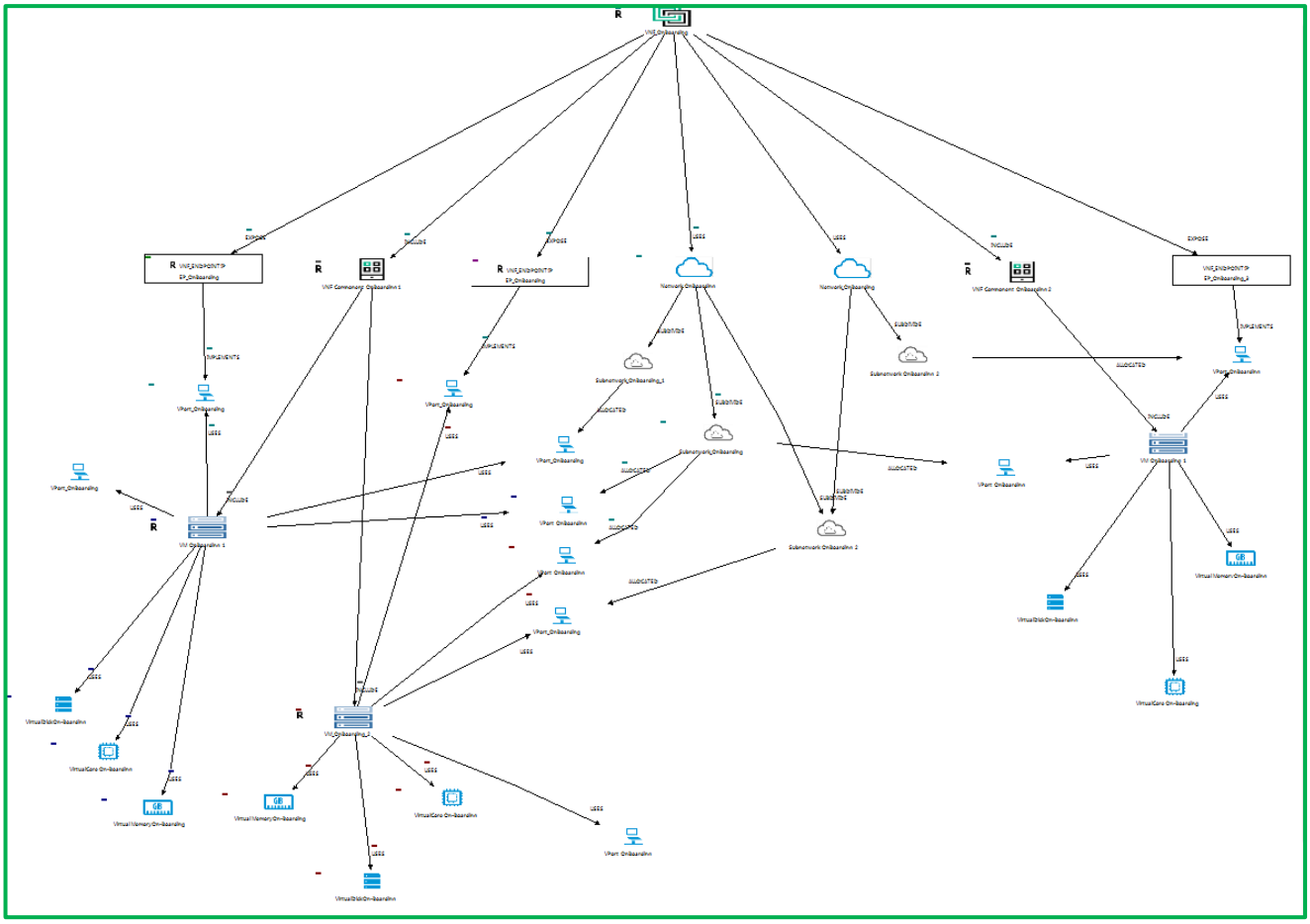


Figure 30: A complex example of VNF

After describing the main elements of the VNF, we are going to explain what the function of each Virtual Port is. They will be listed from the Virtual Machine that owns the Virtual Port.

Virtual Port of the VM 1:

```
VP_VM1_1:   Free Virtual Port, suitable to be connected to a new EP.
VP_VM1_2:   Virtual Port that connects the VM1 and the EP1 of the VNF.
VP_VM1_3:   Virtual Port that connects the VM1 to the Subnetwork
            “Subnetwork_OnBoarding_1” of the Network 1.
VP_VM1_4:   Virtual Port that connects the VM1 to the Subnetwork
            “Subnetwork_OnBoarding_2” of the Network 1.
```

The first and second Virtual Port associated to VM1 do not need explanation, as it is plain to see what they are dedicated to. The third Virtual Port connects VM1 with Network 1 through Subnetwork 1. And the fourth Virtual Port connects VM1 with Network 1 through Subnetwork 2. So, at first sight, VM1 only is connected to Network1. But after reviewing the rest of the Virtual Machines connections we will look at it from another perspective.

Virtual Port of VM2:

```
VP_VM2_1:   Virtual Port that connects the VM2 and the EP2 of the VNF.
VP_VM2_2:   Virtual Port that connects the VM2 to the Subnetwork
            Subnetwork_OnBoarding_2 of the Network 1.
VP_VM2_3:   Virtual Port that connects the VM2 to the Subnetwork
            Subnetwork_OnBoarding_3 of the Network 1.
VP_VM2_4:   Free Virtual Port, suitable to be connected to a new EP.
```

The first and last Virtual Port associated to VM2 do not need explanation, as itis plain to see what they are dedicated to. The second Virtual Port connects VM2 with Network 1 through Subnetwork 2 and the third Virtual Port connects VM2 with Network 1 through Subnetwork 3. So, at first sight, VM2 only is connected to Network1.

Virtual Port of VM3:

```
VP_VM3_1:   Virtual Port that connects VM3 and the EP3 of the VNF, and also
            connects VM3 with the Subnetwork_OnBoarding_4 of Network 2.
VP_VM3_2:   Virtual Port that connects VM3 to the
            Subnetwork_OnBoarding_2 of Network 1.
```

The first Virtual Port allows access to the EndPoint, and the unique Subnetwork of `Network_OnBoarding_2`, so VM3 has access to the `Network_OnBoarding_2`, and Virtual Port 2 allows VM3 connection to the `Network_OnBoarding_1` through the `Subnetwork_OnBoarding_2`.

Once we reviewed all the connections of the VMs, we can conclude that VM1 can access VM2 through the route:

VM_OnBoarding_1→VP_VM1_3-→ Subnetwork_OnBoarding_1→ Network_OnBoarding→ Subnetwork_OnBoarding_3→ VP_VM2_3→ VM_OnBoarding_2.

Also, VM1 will be able to access the VM3 with the following route:

VM_OnBoarding_1→VP_VM1_3-→ Subnetwork_OnBoarding_1→ Network_OnBoarding→ Subnetwork_OnBoarding_2→ VP_VM3_2→ VM_OnBoarding_3.

VM2 will be able to reach VM1 by definition because they are below the same `VNF_Component`, but for VM3, VM1 will able to reach the machine using the route:

VM_OnBoarding_2→VP_VM1_2-→Subnetwork_OnBoarding_2→VP_VM3_2→ VM_OnBoarding_3.

Also, VM3 will be able to access VM1 and VM2 through the `Subnetowrk_OnBoarding_2`. After reaching the subnetwork, VM3 has access to the Network. Therefore, the other two VM are accessible, too.

2.6 Upload a VNF to the solution

There are two ways to upload our VNFs to the NFV-Director Solution. One is using the offline designer, and the other is using REST requests to upload the template.

2.6.1 Through the Resource Modeler

To upload our VNF directly from the Resource Modeler to the NFV-Director, we need to be connected to the specific machine that hosts the solution. Once connected, click in the `File` tag of the top menu of the modeler, and choose `Export`. A new window will appear allowing us to configure the exporting of the template.

IMPORTANT:

If we choose this method, we must assure that the VNF has all the connectors properly configured before attempting its upload.

If we choose this method, we must have a Server and the credentials configured to connect to and access the machine where our NFV-Director is hosted.

As `Diagram Type`, select the value **Template**, and we will export **All Items**. After selecting these two options, click on **Finish**. If some problem occurs, an error window is shown to inform the user about the error cause. If there are no problems, a window confirming the successful export appears.

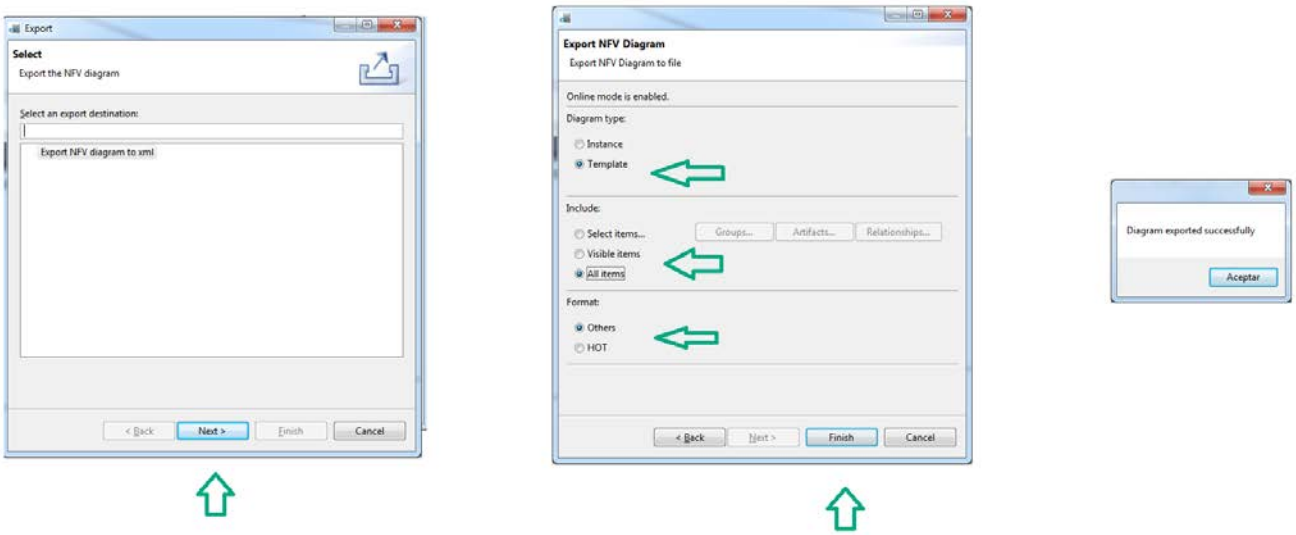


Figure 31: Upload a VNF through the Resource Modeler

Once the template is uploaded to the solution, we should attach it to the entity of our choice. In this case, we attach the template to the domain. We use a REST request through the SoapUI. As indicated on the following illustration, the Endpoint of the request should be the machine that hosts the solution, and the Method should be **POST**, and the Resource will be:

```
/nfvd-ext/domains/{domainId}/catalog/{templateId}/attach
```

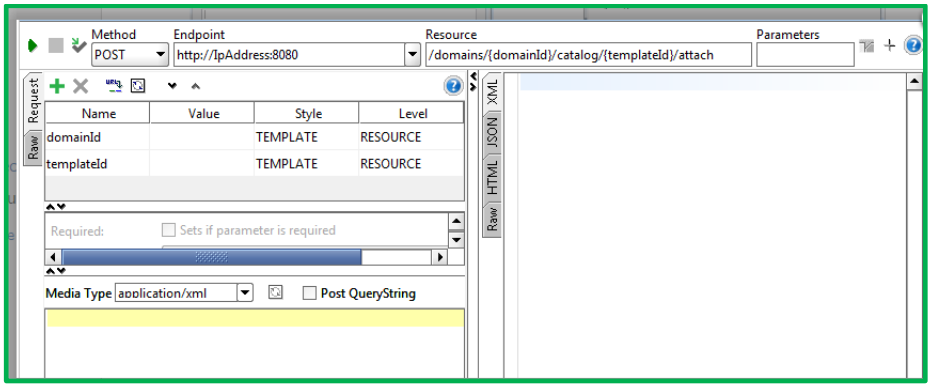


Figure 32: VNF Template upload complete

As `domainId` parameter value, we should introduce the ID of the tree that has the domain as `RootArtifact`, that is, the domain's tree ID, and the template tree ID. With these parameters correctly set, we can launch the request. With this request, we will set the correct visibility for the VNF template that we just have uploaded.

To check if our VNF was correctly uploaded, log into the solution with the preferred user level. It is recommended to use a domain level user to have access to all the VNF templates. Go to the section **Administration** in the top menu of the GUI, and select **VNF Template Catalog**.

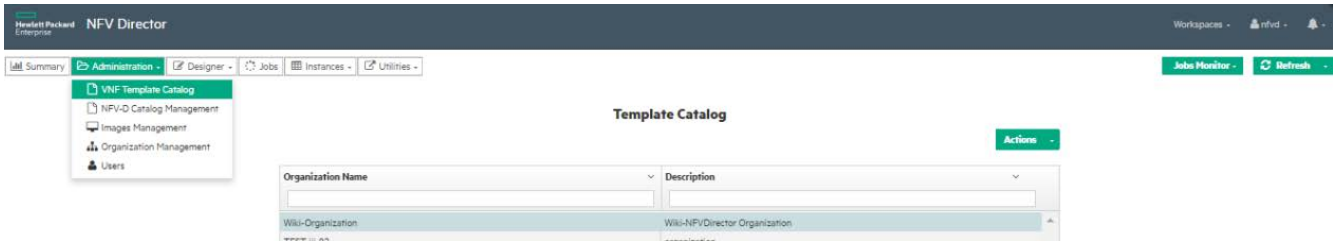


Figure 33: VNF Template Catalog

After the selection of our organization clicking it, click **Actions**. From the displayed list, select **Assign VNF Templates**, and a new window will display the VNF templates that can be assigned to our organization.

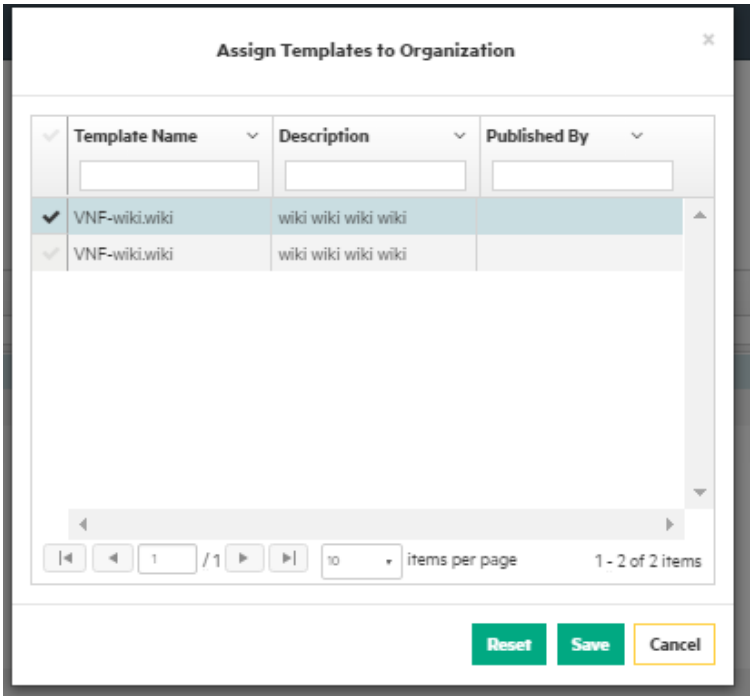


Figure 34: Assign Templates to Organization

As we can see, "VNF-wiki-wiki" is the one that we have just uploaded by the method described. It will be available for the designers of the solution as well.

We can upload the VNF template to be attached at different levels to different entities. The only change in the previous steps is the REST request launched and listed by entity. The requests would be:

To upload the VNF Template to an Organization level:

```
/nfvd-
ext/domains/{domainId}/organizations/{organizationId}/catalog/{templateId}/attach
```

To upload the VNF Template to a VDC level:

```
/nfvd-
ext/domains/{domainId}/organizations/{organizationId}/tenants/{tenantId}/catalog/{tem
plateId}/attach
```

To upload the VNF Template to a VNF Group level:

```
/nfvd-
ext/domains/{domainId}/organizations/{organizationId}/tenants/{tenantId}/vnfs/{vnfGro
upId}/catalog/{templateId}/attach
```

2.6.2 Through a set of REST requests

Once we have our VNF template designed and after exporting the design to an xml file, we can upload the template by a REST request. We will describe the request at domain level. At the end of this section, we will list the request required to upload the VNF template with a different level.

We will configure a new REST request in SoapUI (or any other tool available). The method will be **POST**, the Endpoint should be the IP address of the machine that hosts the solution, and the body should content our designed VNF exported to an xml file as a template. The resource will be:

```
/nfvd-ext/domains/{domainId}/catalog/upload
```

This request will upload the template and set the correct visibility for the component at the level specified by the request. To execute this request, we should fill the parameter `domainId` with the ID of the domain's tree.

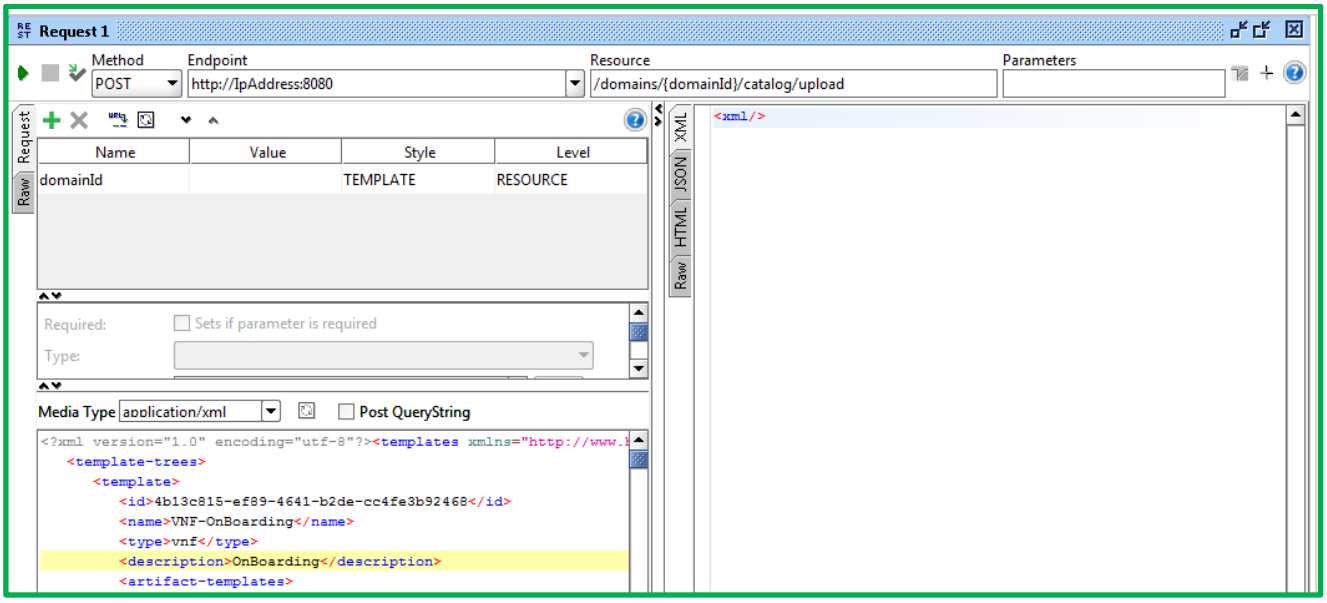


Figure 35: Execute template upload with correct visibility

To check if our VNF was correctly uploaded, log into the solution with the preferred user level. It is better to use a domain level user (to have access to all the VNF templates). Go to the section **Administration** in the top menu of the GUI, and select **VNF Template Catalog**.

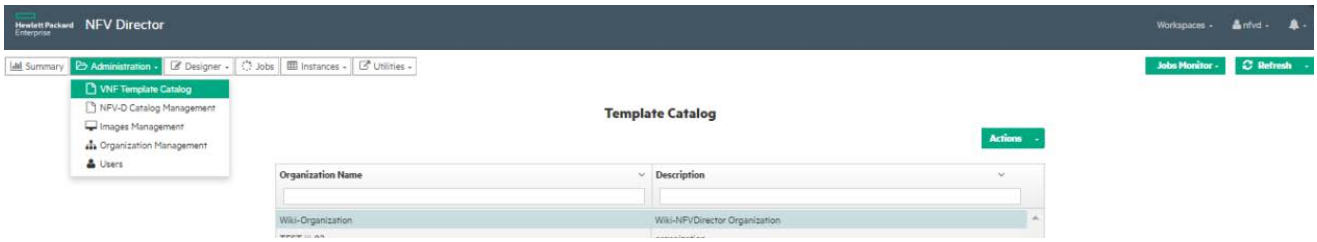


Figure 36: VNF Template Catalog

After the selection of our organization by clicking it, click **Actions**. From the displayed list, select **Assign VNF Templates**. A new window will display the VNF templates that can be assigned to our organization.

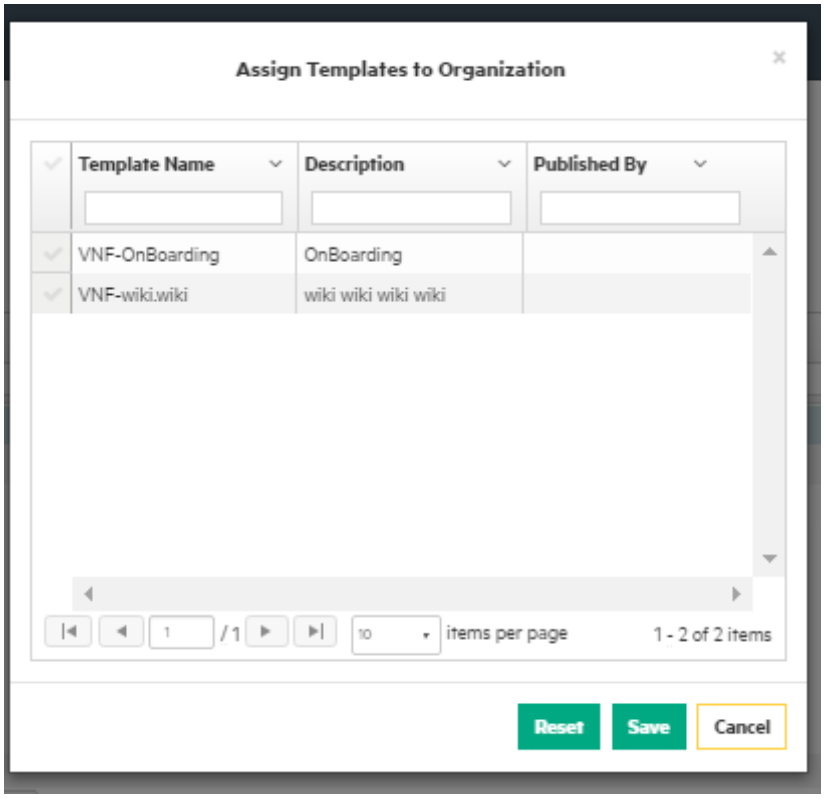


Figure 37: Assign Templates to Organization

As we can see, "VNF-wiki-wiki" is the one that we have just uploaded by the method described. It will be available for the designers of the solution as well.

To upload the VNF Template to an Organization level:

```
/nfvd-ext/domains/{domainId}/organizations/{organizationId}/catalog/upload
```

To execute this request, we should fill the `domainId` parameter with the ID of the domain's tree, and the `organizationId` parameter with the organization's tree ID.

To upload the VNF Template to a VDC level:

```
/nfvd-
ext/domains/{domainId}/organizations/{organizationId}/tenants/{tenantId}/catalog/uplo
ad
```

To execute this request, we should fill the `domainId` parameter with the ID of the domain's tree, the `organizationId` parameter with the organization's tree ID, and the `tenantId` parameter with the ID of the tenant's tree.

To upload the VNF Template to a VNF Group level:

```
/nfvd-
ext/domains/{domainId}/organizations/{organizationId}/tenants/{tenantId}/vnfs/{vnfGro
upId}/catalog/upload
```

To execute this request, we should fill the `domainId` parameter with the ID of the domain's tree, the `organizationId` parameter with the organization's tree ID, and the `tenantId` parameter with the ID of the tenant's tree, and the `vnfGroupId` with the ID of the vnfGroup's tree ID.

Using this method, there is no need to attach the template to some entity. It will be done by the request.

Chapter 3 NFV Director: On Board to the Modes

3.1 Introduction

This chapter of the On-Boarding guide provides guidance to understand the several modes available.

In this chapter, we review the following:

- What is a mode, how to make one and why a user needs additional modes.
- Detailed information and explanation of the supported modes: Default or NFVO (NFV Orchestrator), and Bottom-Up or VIM Managed.

3.2 What is a Mode

A mode is set of operations that a user can execute. Each mode can have a different set of operations. Each operation executes a series of tasks to accomplish its goal. This design is explained in chapter 4.

3.3 Why different modes are needed

NFV Director provides two modes: Default, where everything is orchestrated by the product; and Bottom-Up, where we delegate some of the functionalities to the specific VIM. A customer might want either to extend one or both of the current available modes, or even to create a new one due to needs that are not covered by any of the existing ones.

New modes are created when a user wants to perform certain things with their VIMs, and those needs are not covered by any VIMs available.

3.4 Characteristics of Modes

Currently, NFV Director has two available modes installed:

3.4.1 Default Mode

Default mode (also called NFV Orchestrator) is the one used when the user wants to delegate all operations and VIM management to the NFV Director. This mode has the following characteristics:

- Manage all quotas from all entities (Organization, VDC, VNF Group)
- Manage resources available in the Datacenter
- Always create Tenants, Networks, Flavors, and so on in the deployments of each entity (for VDC, NFV Director will create a Tenant; for Virtual Link, NFVD will create a Network and Subnetwork; and so on
- Requires a user that has admin permissions over the admin Tenant

3.4.2 Bottom-Up

Bottom-Up mode (also called VIM Managed) is used when the user knows or wants to manage their VIM personally. This mode has the following characteristics:

- Will not manage any kind of quota of any entity
- Will not manage any resources in the Datacenter
- As the VIM user provided may not have admin permissions, Bottom-Up operations will not create some of the entities needed (such as Tenant, Flavors, and so on). Instead, it will use the ones already created in the VIM.

3.4.3 What can be created in each mode?

The following table will help to the user understand the available elements of each mode:

		Is able to Create the item in NFVO manages (NFVO mode)				Is it visible in NFVD? (NFVO mode)				Is able to Create the item in VIM manages (VIM mode)				Is it visible in NFVD? (VIM mode)			
		Images	Flavors	Security groups	Networks	Images	Flavors	Security groups	Networks	Images	Flavors	Security groups	Networks	Images	Flavors	Security groups	Networks
NFVD System Creates the ITEM	ITEM is PUBLIC in OpenStack	NO	YES	NA	YES	NA	YES. After full discovery	NA	YES. After full discovery	NO	NO	NA	NO	NA	NA	NA	NA
	ITEM is PRIVATE in OpenStack	YES	NO	YES	YES	YES. After full discovery	NA	YES. After full discovery	YES. After full discovery	YES	NO	YES	YES	YES	NA	YES	YES
User MANUALLY Creates the ITEM directly in OpenStack (after full or refresh discovery)	ITEM is PUBLIC in OpenStack	YES	YES	NA	YES	YES	YES	NA	YES	YES	YES	NA	YES	YES	YES	NA	YES
	ITEM is PRIVATE in OpenStack	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES

Figure 38 Available elements per mode

Chapter 4
NFV Director: On Board to the TLD

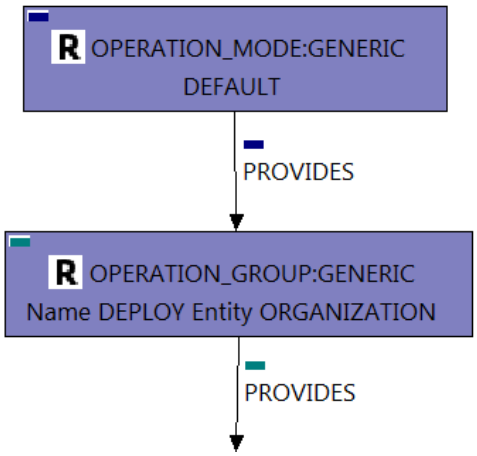
4.1 Introduction

This chapter of the On-Boarding guide provides guidance to understanding the Task List Definition.

What are you going to see when you open it?

- You are going to see a structure of elements that are identified by their artifact Definition, with a specific and unique ID. That has many categories that classify a set of attributes.
- The main categories that you are going to use in the `TaskListDefinition` are `GENERAL`, FIND, `SET`, `EXECUTE` and `ROLLBACK`.
- These artifacts are connected to each other with relationships of different types (mainly `PROVIDES` and `EXECUTE`) to reach a specific target respecting the hierarchy of execution.
- This is accomplished by launching one or more workflows in each level of the structure until we reach the final element of the set.
- The TLD have a tree structure, also the different artifacts can be grouped in trees or subtrees. Each tree must have a unique Root Artifact.
- To make each TLD easier to understand, we are going to explain the conformation and normal execution of the TLD step by step.

4.2 Artifacts related with the Modes.



`OPERATION_MODE:GENERIC:` Main element of the TLD. In this case, the `GENERAL.Name` is `Default`. This value is related to the execution with Modes. When we run this TLD, it is going to run in `Default` mode. There are other possible modes to be executed. To execute with a different mode, the TLD that the user will need is the one created specifically for that mode. It is usually identified by the name of the mode.

This element must always be the root of the Operation Mode tree of the TLD. Normally, in a TLD without policies, this tree only has two elements: the `OPERATION_MODE` artifact, and the relationship of type `PROVIDES` that links the artifact with its unique possible child, the `OPERATION_GROUP` artifact.

`OPERATION_GROUP:GENERIC:` It is always child of the `OPERATION_MODE:GENERIC`. It contains the name of the action and the entity that is going to take place when the TLD is executed, in this case `DEPLOY` and `ORGANIZATION`. Also, the `OPERATION_GROUP:GENERIC` is the root artifact of the TLD's subtree that hosts all the Task List Definitions and Task Definitions present.

The type of relationship between `OPERATION_MODE` and `OPERATION_GROUP` must be always of type `PROVIDES`, and should be set as `ENABLED`. As you can see in the image above, both artifacts are `RootArtifacts` of their respective trees.

Also, notice that the two of them only have one child, for the `OPERATION_MODE`, this is the usual situation. In the case of `OPERATION_GROUP`, it is possible that the artifact has as child policy. Such policies are represented by policy artifacts of different types, mainly, assignation policies.

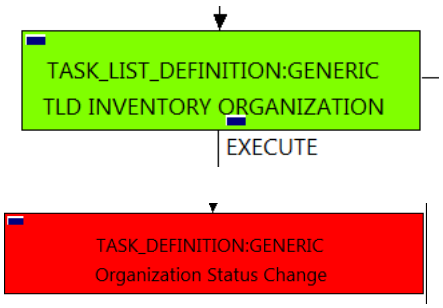
`OPERATION_GROUP:GENERIC`, and `OPERATION_MODE:GENERIC` do not execute any Workflow.

The only mode directly available now is `Default`. This mode allows the users to execute the operations available. The modes can be extended by using operations like `Deploy_FW`. Another possibility is the use of a new mode named `Simulated`. This mode is not pre-loaded, but using it requires it being configured and loaded. The `Simulated` mode was developed to reproduce the effects after launching the operation that is simulated.

The Subtree that has the `OPERATION_GROUP` artifact as Root element is the one that contains all the Task List Definitions, Task Lists and the relationships that relate them to each other. Once the TLD has started the execution, this subtree of the main tree will remain in the memory to be executed as the TLD.

4.3 Rest of the Artifacts of the TLD

The rest of the TLD is formed by elements of type `TASK_LIST_DEFINITION:GENERIC` and `TASK_DEFINITION:GENERIC`. Those are the ones that execute tasks (WorkFlows) to reach our goal. In this case, our goal is the deployment of an `ORGANIZATION` with mode “Default”.



`TASK_LIST_DEFINITION:GENERIC:` Could be child of a `OPERATION_GROUP:GENERIC`, or child of another `TASK_LIST_DEFINITION:GENERIC`. So the children (if any) may only be `TASK_LIST_DEFINITION:GENERIC` or `TASK_DEFINITION:GENERIC`

The type of relationship between `OPERATION_MODE:GENERIC` and `TASK_LIST_DEFINITION:GENERIC` must always be of type `PROVIDES`, and should be set as `ENABLED`.

The type of relationship between `TASK_LIST_DEFINITION:GENERIC` and `TASK_LIST_DEFINITION:GENERIC`, must always be of type `EXECUTE`, and should be set as `ENABLED`.

The type of relationship between `TASK_LIST_DEFINITION:GENERIC` and `TASK_DEFINITION:GENERIC` must always be of type `EXECUTE`, and should be set as `ENABLED`.

The elements `TASK_LIST_DEFINITION:GENERIC` do not execute workflows. They guarantee the correct order and hierarchy in the execution of the `TASK_DEFINITON:GENERIC`, which, in fact, are responsible for executing workflows.

From now on, and to make easier the understanding of the TLDs, we are going to explain the functionality of each set of `TASK_LIST_DEFINITION:GENERIC`, and the numerous `TASK_DEFINITION:GENERIC` children of the previously mentioned `TASK_LIST_DEFINITION:GENERIC.`

Basically, `TASK_LIST_DEFINITION:GENERIC` connects what we can consider “units of execution”. Each `TASK_DEFINITION:GENERIC` has a WORKFLOW assigned to be executed when the execution of the TLD reaches them.

4.4 Attributes and categories of a TLD

In a TLD the user can find four different types of artifacts. Mainly, these artifacts are `OPERATION_GROUP`, `OPERATION_MODE`, `TASK_LIST_DEFINITION` and `TASK_LIST`. Other possible artifacts present in the TLD could be `POLICY:ASSIGMENT` and others.

The Operation Mode and the Operation Group Artifacts have been defined and explained in the previous chapter, so we are focusing on explaining the different attributes and categories of the TLD and TD artifacts.

4.4.1 About the TLD (Task List Definition) artifact.

The artifact Task List Definition is responsible for the division and modularity of the TLD. This artifact is always the parent of a number of Task Definitions, but the parent of at least one. Also, it can be the parent of another Task List Definition.

The TLDs are divided in blocks of TLDs and TDs. Such blocks usually represent one or various steps in the execution of the operation. Usually, when the TLD is in a provision process, the TLD will have more than one child TDs. If it is an activation, normally, the TLD will have one TD as child.

4.4.2 About the TD (Task Definition) artifact

The artifact Task Definition, like the rest of our elements, has a set attributes assigned to a set of categories. These categories define and limit the possible configurations of the TLD. In order to understand all the possibilities these elements offer, each category is explained in proper depth.

Category: `GENERAL`

Attributes: `Name`, `Description`. **Not mandatory**.

Responsible for the name and description of the TD.

Category that allows the user to locate and aim for the right element. To achieve this, the user has can set three configurable attributes:

Category: `FIND`

Attributes: `Main Artifact`, `Condition`, `and Path.` **Not mandatory**.

`Main Artifact`: Represents what kind of artifact is sought.

`Condition`: This attribute permits the user try to find the target artifact by validating or consulting the `STATUS` of an artifact, a variable that represents specific artifacts, and so on.

`Path`: Allows the user to introduce a specific path which will used to locate the specific artifact designated by the path. These attributes also work with conditions in the path.

***

Category that allows the user to locate and aim for the right element. To achieve this, the user has can set three configurable attributes:

Category: `SET`

Attributes: `Running_Status`, Status: **Not mandatory**.

`Running_Status`: Represents the current and expected status of the artifact that we are trying to update, modify or locate.

`Status`: New Status to be set in the artifact that we are working with.

In this category, the user can set the workflow that will be executed, the initial status of the TD (Active or Inactive), and the order of execution of the different Task Definitions of the Task List.

Category: `EXECUTE`

Attributes: `Running_Status`, Status: **Not mandatory**.

: Represents the workflow that will be executed when the execution of the TD reach this point of the TLD.

`Inactive`: New Status to be set in the artifact that we are working with.

`OrderBy`: Attribute that allows the user to set the order of execution of the TLD's TDs, in case that the TLD has more than one TDs.

In this category, the user can set the values for the attributes related to the Rollback process. This means that in case of error during the execution of the TLD, each TD will have set a workflow that will be executed when an error takes place, along with other attributes that configure the way and conditions to execute the rollback process.

Category: `ROLLBACK`

Attributes: `Behaviour_on_error`, `Rollback_Status`, `Number_of_retries`, `Wait_between_retries`, `Codes_to_do_Rollback`, and `Workflow`. **Not mandatory**.

`Behavior_on_error`: Represents the workflow that will be executed when the execution of the TD reaches this point of the TLD.

`Rollback_Status`: New Status to be set in the artifact that we are working with.

`Number_of_retries`: Attribute that set the number of attempts for the specific process that is taking place in the TD.

`Wait_between_retries`: Represents the time that the execution will wait between operation's attempts.

`Codes_to_do_Rollback`: /***

`Workflow`: Represents the workflow that will be executed when the execution of the TD has had some problem during some of the process, the workflow represented by this field is the one that will be launched in case of error.

In the category DATA, the user can set the lock status for the artifact managed by the task in execution. This means that if the value of the attribute `Lock` is `true`, then when the task is finished, the artifact will be locked. If the value is false, the artifact will not be locked at the end of the execution.

Category: `DATA`

Attributes: `Lock`. **Not mandatory**.

`Lock`: Boolean attribute that allows the user to set the status of an artifact as locked at the end of the Task definition execution.

The `INPUT_MAPPING` category includes a list with all the parameters that are going to be used in the execution of the workflow present in the category `EXECUTE`, in the attribute `Workflow`. Each attribute is separated by the character "`;`".

Category: `INPUT_MAPPING`

Attributes: `MAPPING_LIST`. **Not mandatory**.

`MAPPING_LIST`: List of parameters that are going to be used by the `EXECUTE.Workflow` attribute to execute the workflow present in the adequate conditions with the correct parameters.

4.5 The trees and the UUIDs in the TLDs

In our TLDs, all our artifacts are grouped by trees. Inside a tree the user can find more trees acting as subtrees. Also, each tree has a unique `RootArtifact`. In the case of the TLDs, such trees follow certain rules that every user must know to properly operate with them.

To see the tree's specifications the user should click **Groups** in the right-side menu of the NFVDesigner. These will show the groups present and the number of elements in the group. To see the configuration of the tree, the user must select the group by clicking it and select **Edit Group**. If the user follows these steps correctly, a window is displayed.

In the image below, you can see the different attributes of one of the trees present in the resource pool. Apart from the fields `Group name`, `Type` and `Description`, the main field is `Id`. This ID number is common for all the trees in the TLDs. This means this tree `Id` is present and unique in all the TLDs. Remember this when you try to modify these arguments.

The user should be aware that if the button `Regenerate UUID` is used, all the UUID will be changed in the TLD, including the tree's IDs. The user has to control these changes to guarantee the integrity of the tree's IDs.

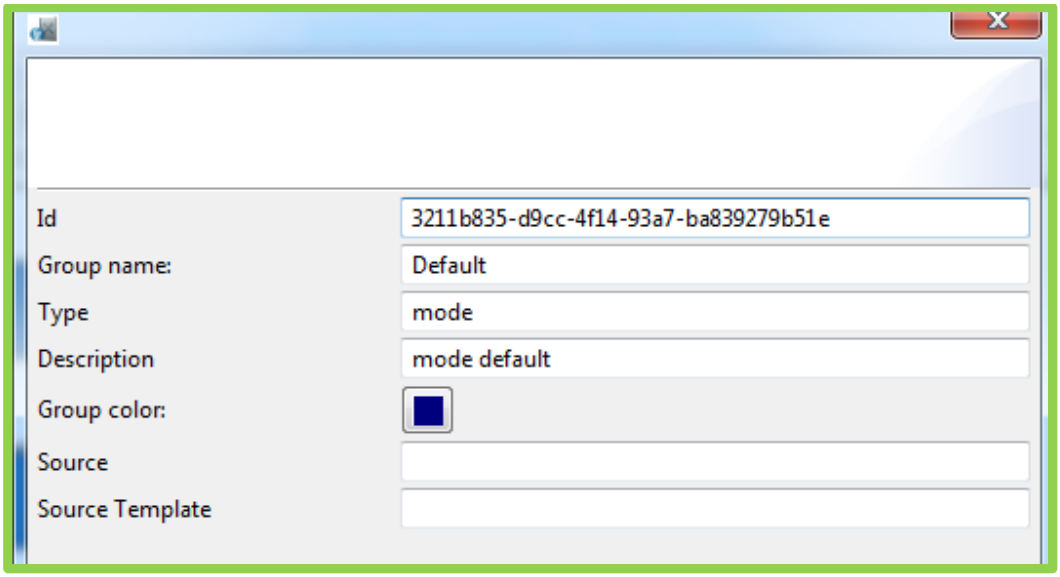


Figure 39: Tree Group configuration window

4.6 Operations and TLDs

The Task List Definitions are responsible for the correct behavior of a set of Tasks that are brought together to perform a specific action- These actions are called operations. Our operations are Deploy, Undeploy, Escalation processes, etc.

Describing all the Operations that can be executed by the system takes a huge amount of time, so we created a set of documents that are guide you in each of the Deploy and Undeploy Operations of the main entities in our system.

These operations are located in the following documents:

- *NFV Director V4.1 On-Boarding Guide Operations – Deploy Organization*
- *NFV Director V4.1 On-Boarding Guide Operations – Deploy Tenant*
- *NFV Director V4.1 On-Boarding Guide Operations – Deploy Virtual_Link*
- *NFV Director V4.1 On-Boarding Guide Operations – Deploy VNF*
- *NFV Director V4.1 On-Boarding Guide Operations – Deploy FW*
- *NFV Director V4.1 On-Boarding Guide Operations – Undeploy FW*
- *NFV Director V4.1 On-Boarding Guide Operations – Undeploy VNF*
- *NFV Director V4.1 On-Boarding Guide Operations – Undeploy Virtual_Link*
- *NFV Director V4.1 On-Boarding Guide Operations – Undeploy Tenant*
- *NFV Director V4.1 On-Boarding Guide Operations – Undeploy Organization*

Notice that each of these documents have a file associated. These files develop the operation describe in each document.

Also, the user should notice that to Deploy a VNF, we should follow the hierarchy expressed in the list above. Deploy should be attempted by the user in a new scenario. First, deploy an Organization, then deploy a VDC, followed by the deploy of a Virtual Link. At this moment, the user only has the possibility of deploy a VNF. The Firewall is a special case of VNF, so the user only can deploy a VNF of any kind.

If an Undeploy is needed, the user should start the adequate Undeploy. In an ideal situation, the deployments were clean and everything ran smoothly, but, in reality, one or two Undeploys are going be required. Depending on which entity has failed

during the deployment, the user must use the right Undeploy. It makes no sense to apply the Undeploy of a VNF to an Organization, it will result in an error.

Chapter 5
NFV Director: On Board to the Workflows

5.1 Introduction

This chapter will describe how upload and refresh a new workflow in the solution. There are two ways to include a workflow in NFV Director, using the Workflow Designer Tool or using specific commands in the CLI.

5.2 Upload a Workflow to the Solution

5.2.1 Using the workflow designer tool

Once we have Workflow Designer open, we can select the **Deployment** option in the top menu. This section of the menu will allow us to upload one or several workflows to our database and HPSA solution. The different options for the deployment of the workflows to the database are:

- Save and Deploy Current Workflow.
  - Allows to deploy the workflow which the user is currently working on.
- Deploy All Open Workflows.
  - Allows to deploy all the workflows that are open or exist in the Workflow Designer UI.
- Deploy Workflows.
  - Allows to select and deploy workflows that exist in the file system.
- Undeploy Workflows.

When deployment is done for the first time, the user will be prompt for database user and password, and optionally, for database instance, port number and host name. The values should match the values provided when running `ActivatorConfig`, which are the values for the system database. The Workflow Designer will remember the values as long as it is running.

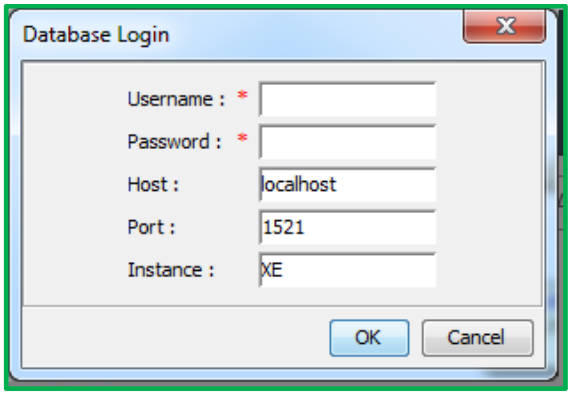


Figure 40: Log in to the Workflow Designer

To upload of the workflows selected correctly, we should fill the fields `Username`, `Password` and `Host`. `Username` and `Password` should contain a correct user and password for the database that will store the workflows. The `Host` should be the correct IP address that provides access to the database.

If all the workflows are uploaded successfully, a confirmation window will pop up.

5.2.2 Using specific commands

To upload and deploy a workflow, it will be necessary to execute the following command:

- Execute: `cd /opt/OV/ServiceActivator/bin/`
- Deploy the workflow using a command in the CLI in the HPSA designer tool:

```
sh designer -dbName [database name] -dbHost [host] -dbPort [port] -dbUser [database
user] -dbPassword [database password] -deployWorkflows /(folder where the workflow is
stored)
```

You could see the command parameters executing the command: `sh designer -h`

Usage: `designer [OPTIONS] [workflow files]`

[OPTIONS]

`-version`		Display version information and exit
`-native`		Set native look and feel
`-config`	`cfg`	Alternate configuration file
`-dbHost`	`<DBHOST>`	Name of the database host. Defaults to configured db host
`-dbName`	`<DBSID>`	Name of the database instance.
`-dbPort`	`<DBPORT>`	Oracle database port. Default is 1521
`-dbUser`	`<DBUSER>`	User name of the database instance
`-dbPassword`	`<DBPASSWD>`	Password of database user
`-listWorkflows`		List deployed workflows
`-downloadWorkflow wf`		Download the specified workflows
`-deleteWorkflow wf`		Mark the specified workflows as deleted
`-deployWorkflows wf`		Deploy the specified workflows

To finish, reload the workflows in the activator dashboard.

Chapter 6
NFV Director: State & Lifecycle of the components

6.1 Introduction

This chapter contains a set of diagrams to explain the different states that some entities take during the different operations that the system can develop.

6.2 State & Lifecycle Virtual Network

In case of a Virtual Network, the states that the entity can take are reduced to the `ACTIVE` and `INSTANTIATED` states. Usually, an artifact is created with an `INSTANTIATED` status. After a `Deploy` operation, the status of the artifact will change to `ACTIVE`. Other entities have more operations associated to their lifecycle, but in the case of the Virtual Networks, the only possible state after the `Undeploy` operation is the deletion of the artifact.

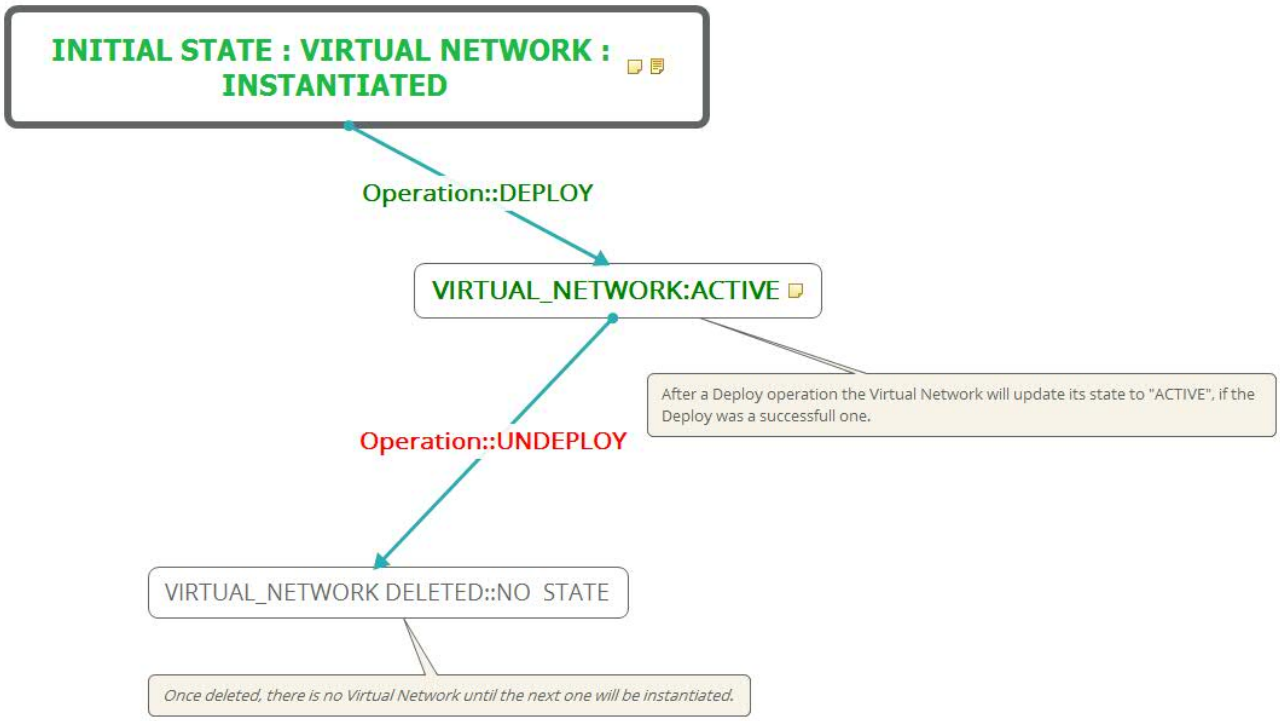


Figure 41 : State and lifecycle of a Virtual Network

6.3 State and Lifecycle Virtual Machine

In the case of the Virtual Machine, the different states that the entity can take are determined by the different operations available over the entity itself. These operations for a Virtual Machine are `Deploy`, some Scale Operations, `Start`, `Stop` and the `Undeploy` operation.

The first state reachable by a Virtual machine (as for the rest of entities) is `INSTANTIATED`. Once the component has executed a successful deploy, its status changes to `ACTIVE`.

Once activated, the Virtual Machine can launch a Scale UP or a Scale DOWN. These two operations will not alter the status of the Virtual Machine. However, if during some of the scale operations some of the machines that are being created or decimated suffer any issue, these machines will change their status to `ERROR`, so the status that the user is going to see is `ON_ERROR/ERROR`.

Other possible operation over a Virtual Machine is the `Stop` operation. Obviously, it is not suitable to stop a machine that is not activated. Once stopped, the Virtual Machine has two possible ways to take. If the Virtual Machine executes a `Start` Operation, it will return to an `ACTIVE` state and it will have all the operations previously mentioned available, as the image below shows. The other option is to execute an `Undeploy` operation, which will delete the Virtual Machine if it ends successfully.

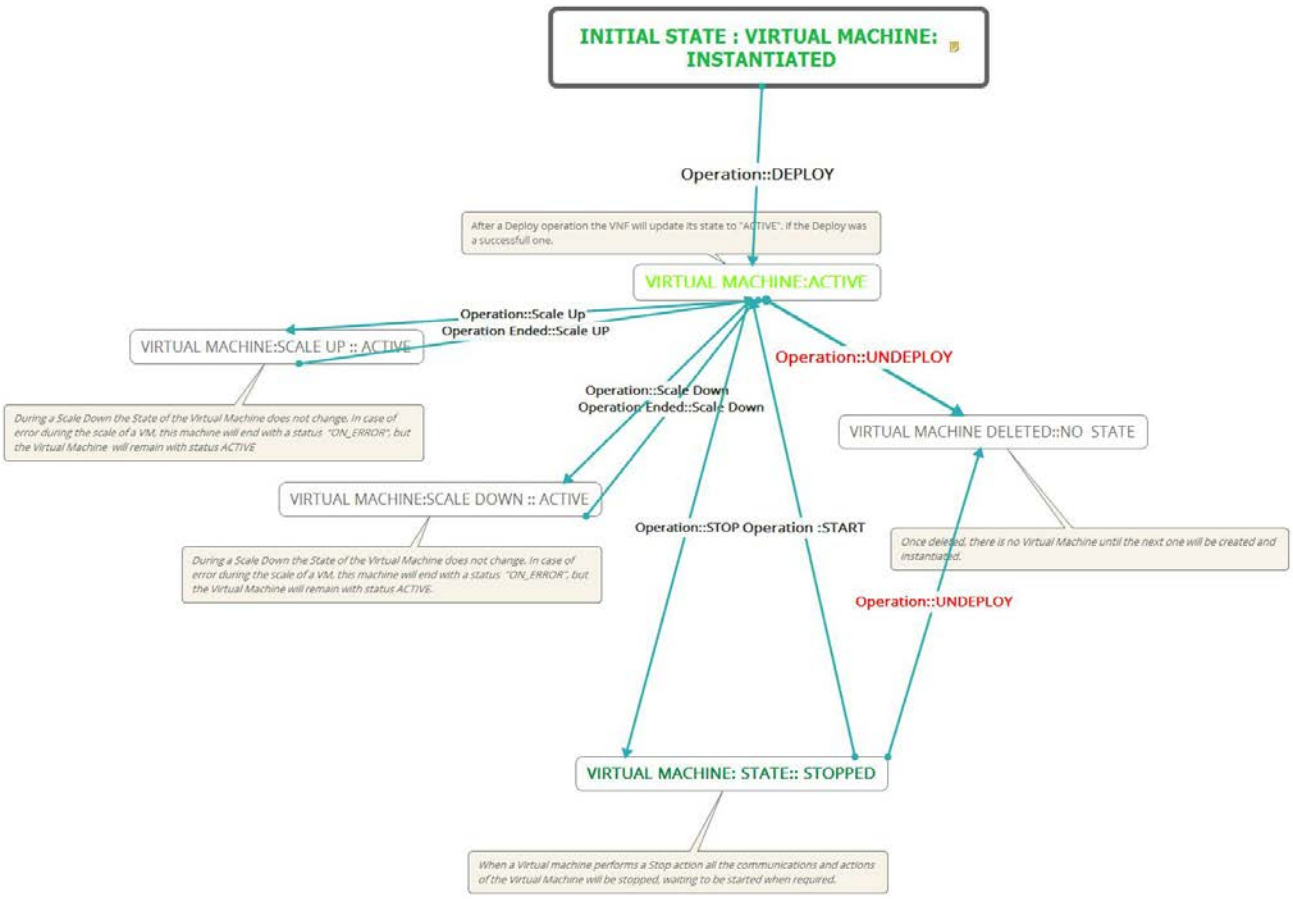


Figure 42: State and lifecycle of a Virtual Machine

6.4 State and Lifecycle VNF

In the case of the VNF, the different states that the entity can take are determined by the different operations available over the entity itself. These operations for a VNF are `Deploy`, some Scale Operations, `Start`, `Stop` and the `Undeploy` operation.

The first state reachable by a VNF (as for the rest of entities) is `INSTANTIATED`. Once the component has executed a successful deploy, its status changes to `ACTIVE`.

Once activated, the VNF can launch a Scale UP or a Scale DOWN. These two operations will not alter the status of the Virtual Machine. However, if during some of the scale operations some of the machines that are being created or decimated suffer any issue, these machines will change their status to `ERROR`, so the status that the user is going to see is `ON_ERROR`/`ERROR`.

The only operation that can delete the element is the `Undeploy` operation. If the user needs to delete the VNF, it is enough to launch the previously mentioned operation. If the operation ends properly, the element has been deleted.

Notice that the VNF Components that belong to the VNF will not change their status, they will remain in `INSTANTIATED` status while the VNF changes its status to `ACTIVE`. We should consider the VNF Components as elements that group our Virtual Machine and the different elements that conform a VNF without interfering with the operation of the VNFs.

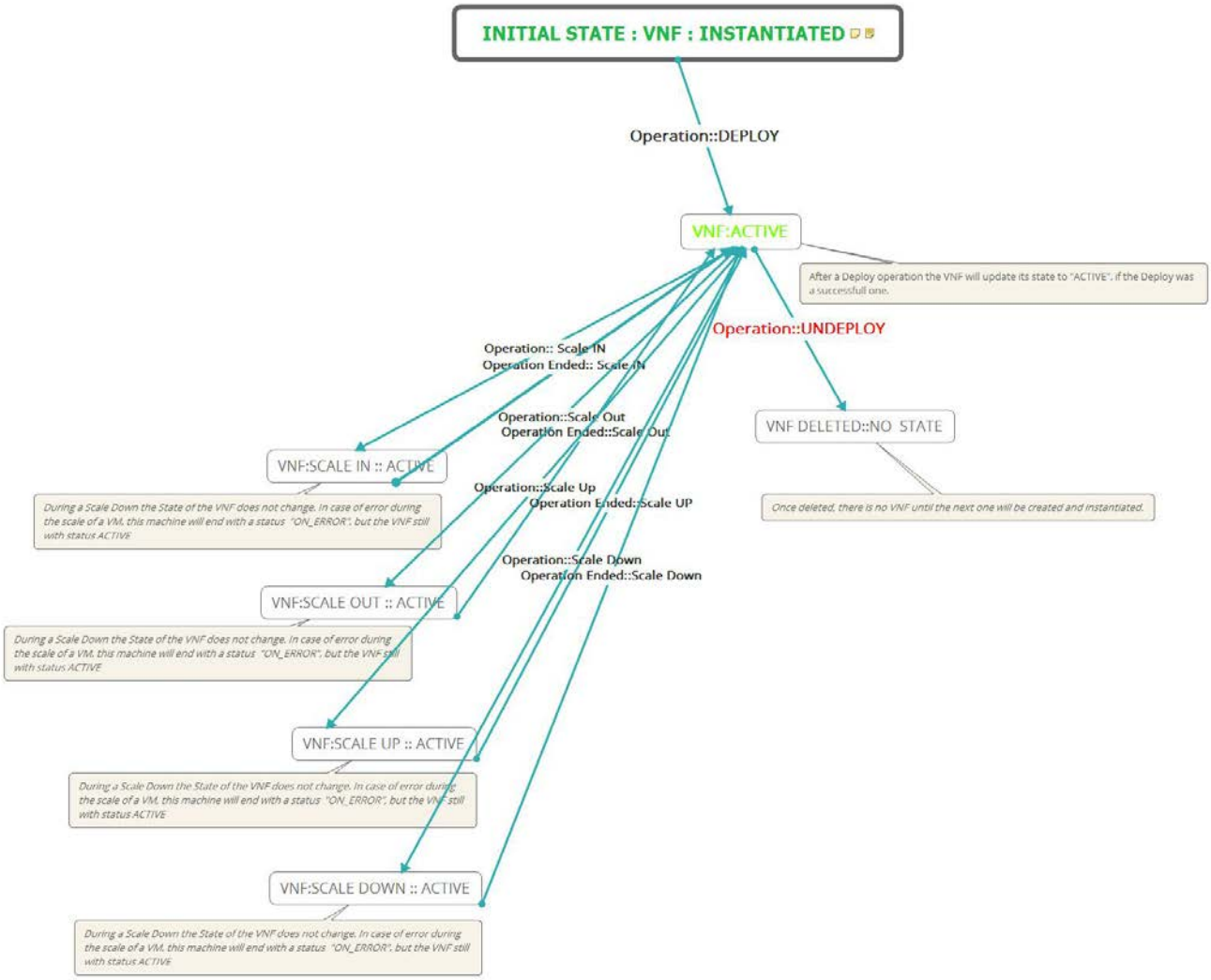


Figure 43: State and lifecycle of a VNF

Chapter 7 NFV Director: TOSCA Support

7.1 Introduction

In this section, we explain how TOSCA specification is supported in the current version of NFVD.

TOSCA is a specification used to describe the topology of cloud applications, and is a standard provided by OASIS.

A template describing the topology of a VNF can be detailed using TOSCA. In the current NFVD version, a TOSCA VNF template can be imported to NFVD, so that a similar NFVD template will be available. On the other hand, a VNF template existing in NFVD can be exported to an equivalent TOSCA template.

In this version, TOSCA support is limited to VNFs, and there are some restrictions. Not all fields in a TOSCA template can be mapped to an NFVD template, and not all data in an NFVD template can be exported to TOSCA. These restrictions are explained deeply in the next sections.

7.2 TOSCA at a glance

As stated in a previous section, TOSCA (Topology and Orchestration Specification for Cloud Applications), is an OASIS standard language to describe a topology of cloud-based services.

TOSCA uses “service templates” to describe cloud workloads as a topology template. The topology template is a graph of node templates and relationships templates.

- “Node templates” model the components a workload is made up of
- “Relationship templates” model the relations between those components

Every node template and relationship template has a type or definition (node types and relationship types). These definitions detail the information the node or relationship is composed of. This information can be:

- Attributes
- Properties
- Capabilities: features provided by a component
- Requirements: to express that a component "depends on" a feature provided by another component

These definitions can be extended by inheritance and are grouped in profiles. The following are two profiles provided by OASIS:

- TOSCA Simple Profile in YAML: a simple profile to describe services in cloud
- TOSCA Simple Profile in YAML for NFV: this profile describes new types that extend the “Simple Profile” ones, for describing specific entities of NFV (VNF, VDU, CP, VLINK)

Services templates that have been written using those profiles are in YAML format. YAML is a serialization language similar to JSON where indentation is used instead of brackets and braces. It is a human-readable language.

An example of a node definition in Simple Profile:

```
tosca.nodes.Compute:
```

```
  derived_from: tosca.nodes.Root
  attributes:
    private_address:
      type: string
    public_address:
      type: string
    networks:
      type: map
      entry_schema:
        type: tosca.datatypes.netwotk.NetworkInfo
    ports:
      type: map
      entry_schema:
        type: tosca.datatypes.netwotk.PortInfo
  requirements:
    - local_storage:
        capability: tosca.capabilities.Attachment
        node: tosca.nodes.BlockStorage
        relationship: tosca.relationships.AttachesTo
        occurrences: [0, UNBOUNDED]
  capabilities:
    host:
      type: tosca.capabilities.Container
      valid_source_types: [tosca.nodes.SoftwareComponent]
    endpoint:
      type: tosca.capabilities.EndpointAdmin
    os:
      type: tosca.capabilities.OperatingSystem
    scalable:
      type: tosca.capabilities.Scalable
    binding:
      type: tosca.capabilities.network.Bindable
```

The definition above details how a compute is expressed in Simple Profile. We can see that definition inherits from Root type, so it contains its properties, attributes and so on.

Valid requirements, capabilities and its types are also indicated.

The definitions can be extended by any provider. OpenStack describes a set of definitions that can be interpreted by its orchestrator, Tacker. The definitions are better adapted to OpenStack than the ones provided by OASIS. That is why they have been used in NFVD for this first TOSCA support.

7.2.1 TOSCA service template

The service templates documents detail the topology of a cloud-based service, and how it has to be managed and orchestrated. A template document can detail a VNF and can be imported in NFVD. And a TOSCA service template document in YAML can be obtained from an existing NFVD template.

A service template document has the following sections:0

- Tosca definitions version: indicates the profile used for this template
- Description: a string briefly describing the template
- Metadata: pairs of key-value strings used to add meta-information
- Topology template: this section contains the nodes that represent the components in the topology, and the relationships between those nodes

7.3 NFVD TOSCA support

As said in previous sections, service templates describing VNFs can be imported in NFVD and VNFs can be exported to TOSCA service templates in YAML. However, there are some restrictions.

Currently “Virtual Machines” and “Connection Points” are supported, but “Virtual Links” are not.

In this version, several node types extended by OpenStack for Tacker are supported. These are the following:

- tosca.nodes.nfv.VDU.Tacker
  It represents an ETSII VDU. This VDU is mapped in NFVD, as a VNF Component with just a Virtual Machine.

- tosca.nodes.nfv.CP.Tacker
  A Connection Point can represent both a port and an endpoint.

The supported TOSCA template has the following structure:

```
tosca_definitions_version:
   This defines the TOSCA definition version on which the template is based.
   Currently tosca_simple_profile_for_nfv_1_0_0.

description:
   A brief description about the template.

metadata:
   template_name: A name to be given to the template.

topology_template:
   Describes the topology of the VNF under node_template field.
   node_template:
       Describes node types of a VNF.
       VDU:
           Describes properties and capabilities of Virtual Deployment
           Unit.
       CP:
           Describes properties and capabilities of Connection Point.
```

7.3.1 VDU supported information

Data that can be mapped from a “VDU.Tacker” to a NFVD Template:

Properties

- “image”: in this version, it is important to indicate a valid image name in this property. In other case, the template would be created, but it will not be possible to deploy it.

Capabilities

- "nfv_compute": to indicate the compute capabilities exposed by this VDU. The capabilities properties that can be indicated are:
    - "num_cpus": number of CPUs. It must be indicated by an integer value
    - "mem_size": size of the memory. It must be indicated by an integer followed by the unit (always in MB). For example:1024 MB
    - "disk_size": size of the compute local disk. It must be indicated by an integer followed by the unit (always in GB). For example:1 GB
    - "mem_page_size": size of the memory page. It must be indicated by an integer

7.3.2 CP supported information

Data that can be mapped from a "CP.Tacker" to an NFVD Template:

Properties

- "ip_address": a valid IP address (but no checking is done)
- "mac_address": a valid MAC address (but no checking is done). Currently it is not taken into account when deploying the template
- "order": port order
- "type": port type. Currently only virtio value is supported.

Requirements

- "virtualBinding": indicates the name of the VDU this CP belongs to
- "virtualLink": in this version virtual links are not supported in TOSCA templates. But this field is used to indicate the name of the endpoint the port belongs to. If this requirement is not indicated, an endpoint with the CP name will be added.

7.3.3 Supported TOSCA service template example

This is an example of a TOSCA service template in which all fields are supported and can be mapped to a NFVD VNF template. A valid VNF template called "vnf_tosca_template_test" would be created in NFVD from this TOSCA template if importing.

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0

description: Simple Vnf for testing

metadata:
  template_name: vnf_tosca_template_test

topology_template:

  node_templates:

    VDU1:
      type: tosca.nodes.nfv.VDU.Tacker
      properties:
        image: test2
```

```
      capabilities:
        nfv_compute:
          properties:
            num_cpus: 1
            mem_size: 1024 MB
            disk_size: 1 GB
            mem_page_size: 2
    VDU2:
      type: tosca.nodes.nfv.VDU.Tacker
      properties:
       image: test2
      capabilities:
        nfv_compute:
          properties:
            num_cpus: 1
            mem_size: 1024 MB
            disk_size: 1 GB
            mem_page_size: 2
    CP11:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
         ip_address: 200.0.0.4
         mac_address: 00-50-56-C0-00-01
         order: 0
         type: virtio
      requirements:
        - virtualBinding: VDU1
        - virtualLink: net0
    CP21:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
         ip_address: 200.0.0.5
         mac_address: 00-50-56-C0-00-02
         order: 1
         type: virtio
      requirements:
        - virtualBinding: VDU2
        - virtualLink: net0
```

A NFVD VNF Template similar to the TOSCA template above would have two VNF components with a virtual machine each (called “VDU1” and “VDU2”), and an endpoint called “net0” made up of two ports (one in “VDU1” virtual machine and other one in “VDU2”).
This VNF would have the following structure:

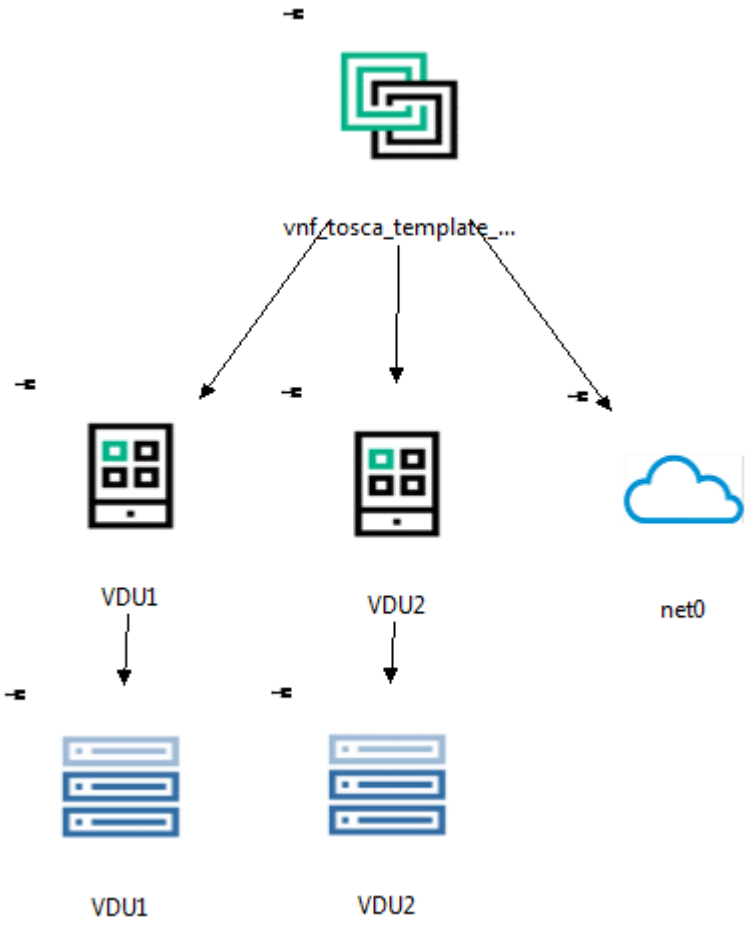


Figure 44: Obtained VNF from TOSCA template example

Chapter 8
Localizing the NFV Director UI

8.1 Introduction

This chapter describes how to localize the NFV Director UI.

8.2 How to do

NFV Director UI is an add-on to the Unified OSS Console framework. As such, it relies on UOC for localization support.

8.2.1 Language support

The list of languages supported by the application is defined at run-time in the configuration file: `/var/opt/uoc2/server/public/conf/config.json`

```
"languages": [{
        "name": "English",
        "languageCode": "en-us"
    }, {
        "name": "Français",
        "languageCode": "fr-fr"
    }, {
        "name": "Español",
        "languageCode": "es-es"
    }],
```

When a user logs in, his locale will be determined based on his preferences (user profile), or his browser settings. The application will use the closest available locale (for example a user with locale `es-mx` would fall back to `es-es`). If the user's locale is not configured for the application, the locale falls back to `en-us`.

8.2.2 Localization files

Strings and labels to be localized are located in `json` files with the same name as the locale (for example, `en-us.json`, `fr-fr.json`).

The NFVD-UI add-on contains two localization files for the English default:

```
client\addons\nfvd\modules\nfvd-module\en-us.json
client\addons\nfvd_portal\modules\nfvd-portal-module\langs\nfvd\en-us.json
```

Each file contains a set of KEYS, and their associated localized value.

```
{
    "WORDS": {
        "CREATE": "Create",
        "DELETE": "Delete",
…
}
```

The keys are organized hierarchically (for example, to the key `WORDS.CREATE` the localized English value associated is `“Create”`).

To localize the application in a new language (for instance, French), create a file named as the locale (fr-fr.json), duplicate the keys hierarchy and assign a localized value to each key.

```
{
    "WORDS": {
        "CREATE": "Créer",
        "DELETE": "Supprimer",
…
}
```

IMPORTANT: Keys are not localized, only values are.

Table 7: Localization example

en-us.json	fr-fr.json	
"NAME": "Name"	"NAME": "Nom"	**GOOD**
"NAME": "Name"	"NOM": "Nom"	**BAD!**

IMPORTANT: The syntax must be JSON-compliant. Values are enclosed in quotes. If the localized value must contain a quote, it must be escaped.

```
“KEY”: “A string with \”quotes\””
```

IMPORTANT: The localized values may contain variable placeholders that will be filled when the string is displayed. The placeholder must be kept with the same syntax (and the variable name must not be localized.

```
English: "FAIL_TO_DELETE_INSTANCE": "Could not delete instance: {{name}}. Error
message: {{message}}.",
French: "FAIL_TO_DELETE_INSTANCE": "Impossible de supprimer l’instance:
{{name}}. Message d’erreur: {{message}}.",
```

If a message is not localized in a language, the application will use the default English value.

8.2.3 Data

The data displayed in the UI is generally not localized.

Components like the VDC Manager Inspector are generic and can display any entity and attribute, as the data is returned by the API.

However, the application is pre-configured with a subset of known entities, known categories, known attributes, which are localizable.

When displaying a type of artifact or attribute that is not known, it is displayed as is.

On the following illustration, the attribute `NAME` is localized and shown (in French) as `Nom`.

However, the attribute `DECREASEAMOUNT` is not localized.

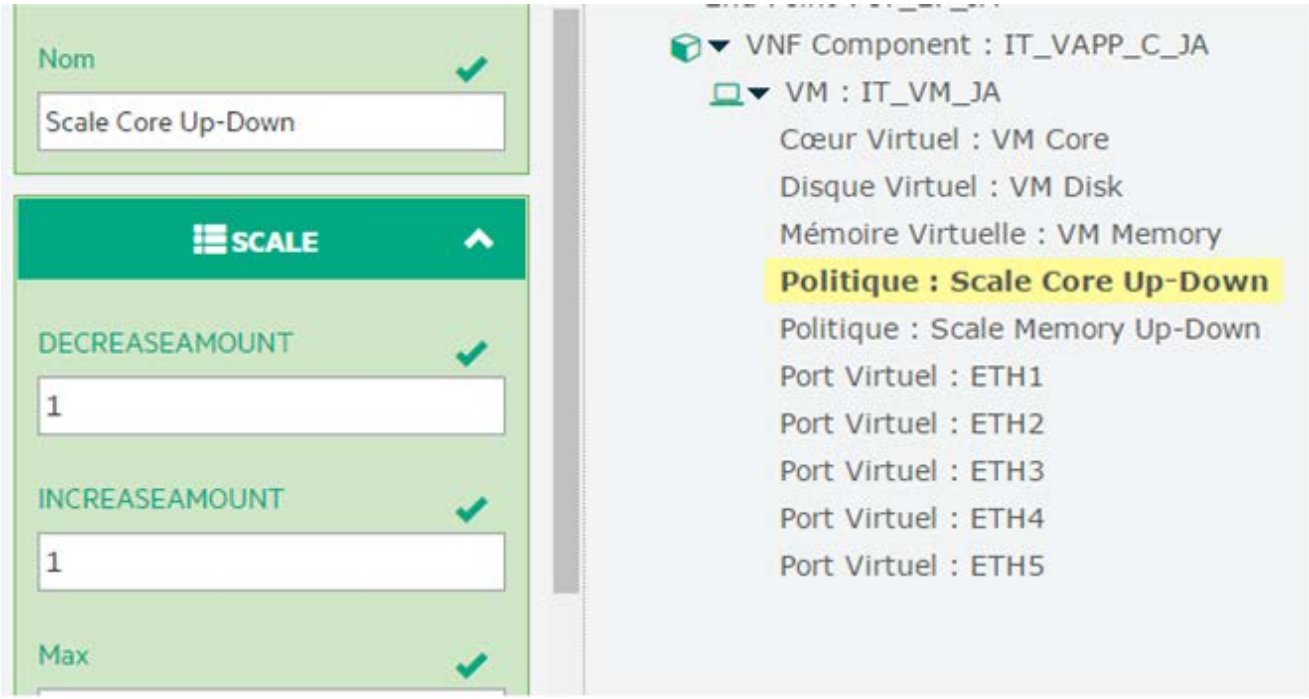


Figure 45: Localized and not localized attributes

NOTE:

UOC provides the framework on which the NFV Director UI add-on runs.

The current UOC version used by NFV Director V4.0 only supports English and French.

So even if the NFV Director UI add-on is localized in Chinese, all the UOC widgets or components used by NFV Director UI will still be displayed in English.

This will change in the next version of UOC.

Chapter 9
On-Boarding Guide examples and References

9.1 Introduction

This chapter provides a short description of the examples used inside the Guide. The location of the examples is also discussed in this section.

9.2 List of Examples per Chapter

9.2.1 Subchapter 1.5 Examples

This is the chapter responsible for the introduction to the modelling of a Resource Pool. These examples help the users understand the different steps and the RP's complexity. The examples will contain different number of REGIONs, `AVAILABILITY ZONE`s and Servers to explain what types of configuration are available.

9.2.1.1 "RP_Example_01"

The essential Resource Pool, the minimum exponent of what a RP should be. One Region, One `Availability_Zone` and one Server.

Related files: `RP_Example_01.nfvd`, `RP_Example_01.xml`.

9.2.1.2 "RP_Example_02"

The second step in complexity, one `Region`, two `Availability_Zones` and two servers, In this case, the scenario will have more possibilities during the different operations that are going to take place in the application.

Related files: `RP_Example_02.nfvd`, `RP_Example_02.xml`.

9.2.1.3 "RP_Example_03"

The third step in complexity, two Region, two `Availability_Zones` and two Servers. In this case, the scenario will behave differently, depending on the operations that are going to be executed.

Related files: `RP_Example_03.nfvd`, `RP_Example_03.xml`.

9.2.1.4 "RP_Example_04"

The fourth step in complexity, two Regions, one `Availability_Zone` and one Server. In this case, when the activation processes take place over the Region with the `Availability_Zone`, the scenario will behave as expected. In the case of the other `REGION`, there will be slight differences.

Related files: `RP_Example_04.nfvd`, `RP_Example_04.xml`.

9.2.1.5 “RP_Example_05”

The fifth step in complexity, two Regions, three `Availability_Zone`s and five Servers. In this case, when the activation processes take place over the Region with only one `Availability_Zone`, the scenario will behave as expected. Although this `REGION` has two Servers, the execution will decide which one is the better option to be used. In the case of the other `REGION`, there will be slight differences. A `REGION` with two `Availability_Zone`s will behave depending on the original level of the operation.

Related files: `RP_Example_05.nfvd`, `RP_Example_05.xml`.

9.2.1.6 “RP_Example_06”

The sixth step in complexity, two Regions, two `Availability_Zone`s and three Servers, with an `SDN_CONTROLLER` artifact. This means that this Resource Pool is going to operate over a VSD platform. Also, in this example, the user will find the three types of `NETWORKING` artifacts that are used in the system.

Related files: `RP_Example_06.nfvd`, `RP_Example_06.xml`.

9.2.2 Subchapter 1.6 Examples

This is the Chapter that explains how a Resource Pool will behave depending on the attributes given for the operation, in order to achieve this various options are offered over the same Resource Pool, to make easier to understand the different situations that the user will find.

9.2.2.1 “RP_Example_DC”

This example shows a Resource Pool with two REGIONs, each REGION with two Availability_Zones, and each AZ with two Servers, over this scenario the chapter describes three different possibilities.

Compose by the files: “RP_Example_DC.nfvd”, “RP_Example_DC.xml”.

9.2.3 Subchapter 1.7 Examples

This is the Chapter that explains how a Resource Pool will communicate between Data Centers in a multi Data Center scenario, this communication will be allowed by the SDN_CONTROLLER artifact.

9.2.3.1 “RP_Example_VSD_1”

This example shows three DataCenters without any possibility to communicate between them.

Compose by the files: “RP_Example_VSD_1.nfvd”, “RP_Example_VSD_1.xml”.

9.2.3.2 “RP_Example_VSD_2”

This example shows three DataCenters with the possibility to communicate between two of them, leaving the third one isolated of the rest.

Compose by the files: “RP_Example_VSD_2.nfvd”, “RP_Example_VSD_2.xml”.

9.2.3.3 “RP_Example_VSD_3”

This example shows three DataCenters with the possibility to communicate between them.

Compose by the files: “RP_Example_VSD_3.nfvd”, “RP_Example_VSD_3.xml”.

9.2.4 Chapter 2 Examples

This is the Chapter responsible to explain how the user should create a component VNF.

9.2.4.1 “Example_VNF_00”

This example shows the simplest VNF the user could design.

Compose by the files: “Example_VNF_00.nfvd”, “Example_VNF_00.xml”.

9.2.4.2 “Example_VNF_01”

This example shows a second level VNF, more than one VNF Component, more than one End_Point, more than one Virtual machine, and a Network and Subnetwork to operate.

Comprise of the files: “Example_VNF_01.nfvd”, “Example_VNF_01.xml”.

9.2.4.3 “Example_VNF_02”

This example shows a complex VNF, more than two EndPoints, two Virtual Machines and Components, more than one Network, with more than one Subnetwork, and a combination of the different possibilities of connection between them.

Compose by the files: `Example_VNF_02.nfvd`, `Example_VNF_02.xml`.

9.2.5 References

(1)(2)(3) http://docs.openstack.org/user-guide-admin/cli_nova_specify_host.html

(4) https://kimizhang.wordpress.com/2013/08/26/openstack-zoning-regionavailability-zonehost-aggregate/

(5) http://docs.openstack.org/admin-guide/

Chapter 10
NFV Director Assurance

10.1 Introduction

The NFV Director Assurance component consists of three planes: Monitoring plane, Correlation plane, and Autonomous Action plane.

10.1.1 Monitoring Plane

The monitoring plane consists of three parts:

- An integration component called Assurance Gateway
- A Monitoring Engine that is agentless: Site Scope
- Mediation to collect events from external sources: Open Mediation

The Assurance Gateway is responsible for the following:

- Receive and process the VNF topology information from Fulfillment, and updates in a graph database used for monitoring and topology-based correlation later.
- Receive and processe VNF monitoring related notifications from Fulfillment and provides corresponding action to the agentless monitoring component.
- Provide lifecycle management (LCM) notifications for artifacts.
- Export topology and KPI data for integrations with OSS, analytics, and other applications.

The agentless monitoring component, built using HPE SiteScope, can collect a wide variety of KPIs, issuing events or executing commands when pre-defined thresholds are crossed for those KPIs. As different VNFs have varying monitoring needs, the monitoring points and thresholds are automatically configured by the NFV Director when the VNF is provisioned or modified. As an agentless solution, the NFV Director does not require installation of monitoring agents on the target systems.

The individual VNF managers are responsible for monitoring their own internal faults and performance, possibly augmented with traps and key performance indicators (KPIs) provided by the NFV Director. When VNF management functionality is provided by the NFV Director through an embedded VNF manager, it might be necessary to collect application-specific monitoring information. HPE SiteScope provides the capability to develop custom monitors for VNF resources and VNF applications.

HPE SiteScope templates standardize a set of monitor types and configurations into single re-usable tempalates. These can then be repeatedly deployed as a group of monitors targeting multiple VNFs. Templates accelerate the deployment of monitors across the enterprise through a single-operation deployment of groups, monitors, alerts, remote servers, and configuration settings.

Finally, HPE NFV Director includes HPE Open Mediation, which provides the following capabilities:

- Allows the integration of multiple products.
- Defines common communication patterns:
    - Alarm flow

    - Resynchronization
    - Action invocation
    - Topology notification
- Provides numerous connectivity features:
    - Web Services
        - SOAP/HTML
        - SOAP/JMS
        - HTML/REST
    - Files
        - Local file access
        - FTP/FTPS/SFTP
    - Database
        - JDBC
    - Enterprise Java
        - JMS
        - JMX
        - RMI
    - Other
        - TCP/UDP
        - HTTP/HTTPS
        - IRC
        - LDAP
        - SMTP/POP3/IMAP
        - RSS
        - SMPP
        - SNMP
        - XMPP

In the context of the NFV Director, OpenMediation is used to integrate with HPE SiteScope, EMS, VNF Manager, or any other source of events into NFV Director.

10.1.2 Correlation plane

The Unified Correlation Analyzer for Event Based Correlation product (also known as UCA Expert by analogy with the legacy TeMIP Expert software) offers a new and generalized event based correlation solution.

Based on the JBoss Drools 5.5.0.Final rule engine, UCA for EBC offers the capability to create comprehensive functional correlation sets called Value Packs that implement the correlation logic. This correlation is performed by running certain rules (the rules are written in a Java-based language). Any Value Pack can support or use predefined functionalities, such as Alarm collection, filtering, lifecycle, as well as Generic Actions.

The UCA for EBC can perform the following functions:

- Collect alarms and map them into the Operator Alarm model (Alarm) based on a combination of X733 and OSS/J Fault Management Model.
- Run several scenarios (rule engines) in parallel and in sequence to implement complex correlation algorithms. Each set of scenarios implementing a single correlation solution is grouped inside a UCA for EBC Value Pack.
- Dispatch Alarm objects for different scenarios.
- Execute rules based on the scenario input stream and generate suitable output, for example, actions to external systems.
- Control the scenario input stream using an alarm-based filtering layer.

- Execute actions such as storing in a database, creating a Trouble Ticket, creating a new Alarm, grouping Alarms, forwarding an Alarm to another scenario, or executing a Generic Action through the OSS Open Mediation v6.2 (NOM v6.2) layer.

Rule files are JBoss Rule files, so both JBoss Expert and Fusion rules are supported. JBoss Drools Expert and JBoss Drools Fusion are JBoss Drools basic modules.

Over this basic functionality, UCA for EBC also provides a software development kit (SDK) that allows solution developers to easily build UCA for EBC Value Packs (Functional Correlation block). Administration tools (both command line and a GUI) are also available to manage, monitor, and troubleshoot the product.

UCA for EBC can be connected with a mediation bus (OSS Open Mediation v7.0), providing the capability to collect alarms from any source (NMS) and perform actions in return.

All collected alarms are persisted into an alarm database.

10.1.3 Autonomous Action plane

The UCA Automation software, a combination of both business rule engine and workflow engine, enables a clear separation of what and how to automate. All complexities of automation, such as how to access a network resource (can be a network element, an element component, an EMS, or NMS), identifying its credentials and the specific transport mechanism used to connect to the resource, determining the supported OS versions of the device and specific commands that need to be sent are abstracted from the business rules.

This software enables administrators to create, update, and read the business rules with utmost clarity and to maintain them efficiently. It also allows administrators to store the acquired automation knowledge in the form of business rules, focusing on what and not how.

Another advantage of the UCA Automation software is that for most resolution automations, the operator only needs to know the business rules and not the technologies in the background to implement day-to-day operational changes to the decisions.

Thus, the UCA Automation System is a platform for building value added resolution automations based on a judicious combination of business rules and workflows.

10.1.4 Threshold Crossing Alarms collection

Each monitor deployed in SiteScope is capable of generating an alerts for good, error, and warning conditions based on information received from the Monitor.

SiteScope sends SNMPv2 alerts to the OM, which transforms these SNMP alerts to a format recognizable and usable by the UCA.

NFV Director provides a default SNMP template file, `NFVD_SNMP_TEMPLATE_XML.xml`, to integrate SiteScope alerts with the UCA correlation platform. This file enables SiteScope to generate and send SNMPv2 traps to the OM.

The default path of this template file: `<SiteScopeProduct>/templates.snmp/NFVD_SNMP_TEMPLATE_XML.xml`

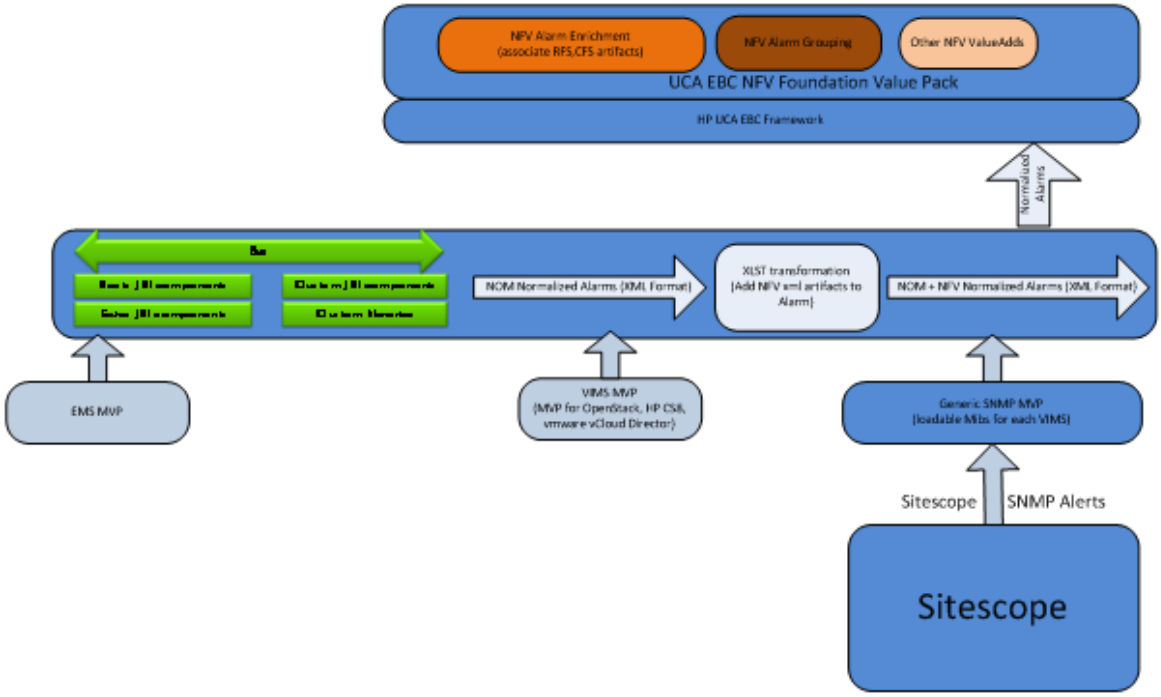


Figure 46: Integration of monitoring threshold crossing alerts for correlation

10.2 Monitor Modeling

10.2.1 Monitor artifact

A monitor artifact takes the following attributes.

Table 8: Monitor artifact attributes

Attribute	Description	Constraints
`Name`	Name of the monitor.	The name should be same as that of SiteScope monitor template. Mandatory: Yes
`Description`	Human readable description of the monitor.	Mandatory : No
`Frequency`	Time value in seconds at which the monitor runs periodically.	Mandatory : Yes Should not be less than 20 seconds.
`Deployment Path`	The path where the monitor should be deployed on SiteScope.	Mandatory: No. If this field is empty, a default path will be automatically built.
`Template Path`	The path of the SiteScope monitor template from where it is to be deployed.	Generically used with custom monitors. Mandatory: Yes, only when custom monitors are used.

10.2.2 Condition artifact

Table 9: Condition artifact attributes

Attribute	Description	Constraints
`Name`	The name of the condition.	Mandatory: Yes
`Type`	Type of condition.	Available values: • Good • Warning • Error
`Expression`	An expression that describes a condition. See Out of Box monitors provided with NFV Director to provide a valid and supported expression.	Mandatory: Yes, only when Out of Box NFV Director monitors are used. Optional when custom monitors are used.

In V4 the user can configure multiple conditions of each type –Error, Good, or Warning.

For example: In the Virtual Machine CPU template there are two counters:

- `availability` – indicates if template is able to collect metrics
- `cpu_usage_average` – indicates CPU usage

In this case the user can create two separate ERROR conditions the following thresholds:

- `availability contains` n/a
- `cpu_usage_average` > 90

The following figure indicates a monitor with two error conditions and different actions.

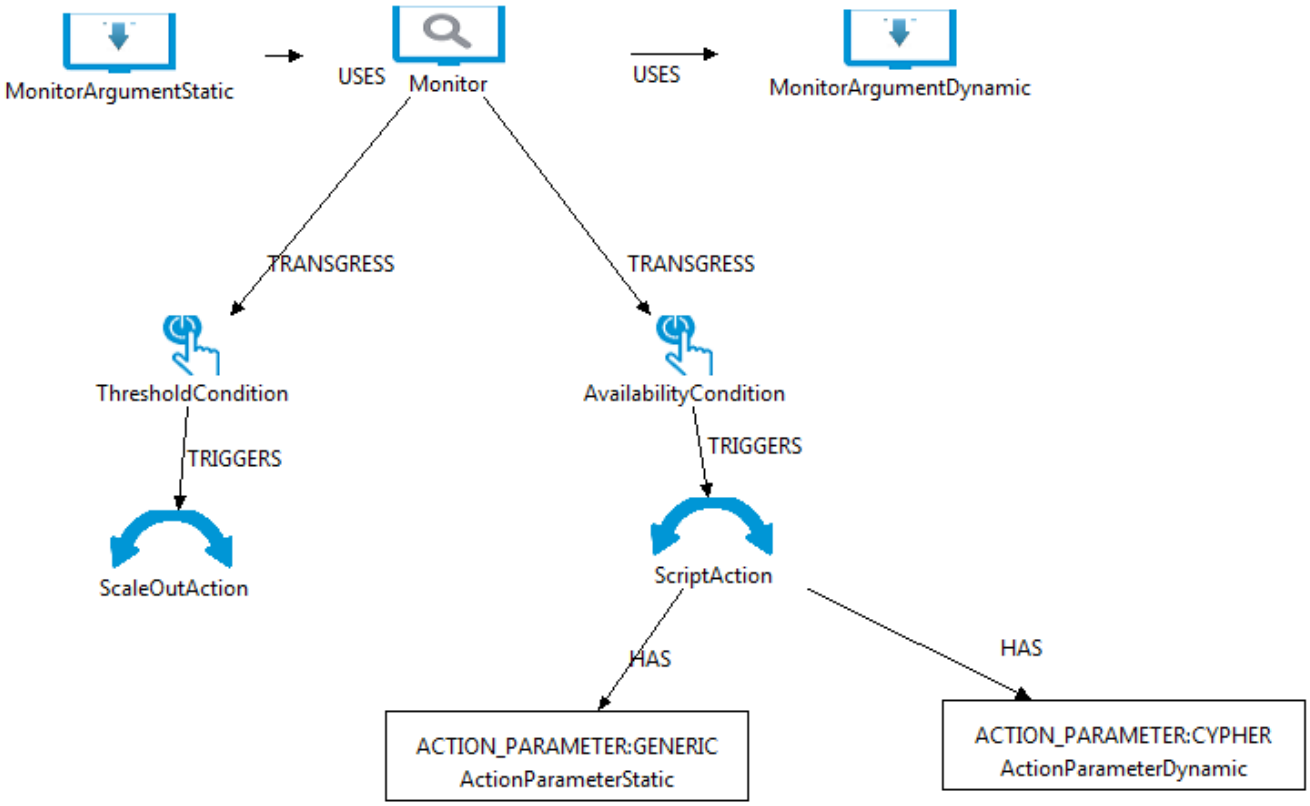


Figure 47: Monitor with two error conditions and different actions

10.2.3 MonitorArgument Artifact

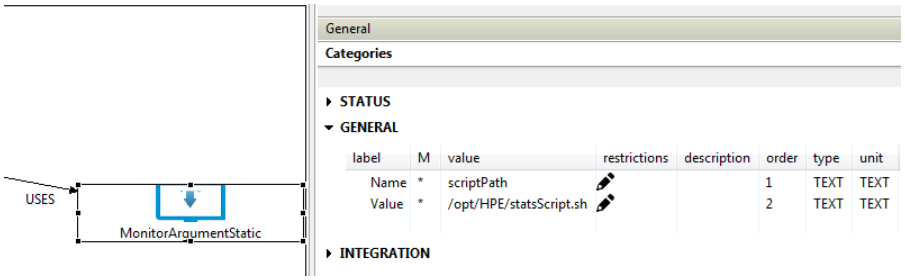
Monitor Argument artifacts can be associated to monitors.

It's a way of passing static/dynamic arguments to a monitoring service like Sitescope. For example, a custom monitor might need the following arguments:

a. scriptPath: that provides a path to the script on the system to be executed for monitoring. scriptPath variable can be passed as hardcoded value by using MONITOR_ARGUMENTS:ARGUMENTS.
b. host: hypervisor hostname of the VM. Since the hypervisor hostname would not be known during template design time, it would have to be picked up dynamically from NFVD topology during orchestration. ipAddress variable can be passed by using MONITOR_ARGUMENTS:CYPHER

MonitorArgument are of two types:

- Arguments – It contains name-value pair. Attribute "name" has to map to mandatory variables required by template in SiteScope. Attribute "value" will have the hardcoded value that the user wants to configure.

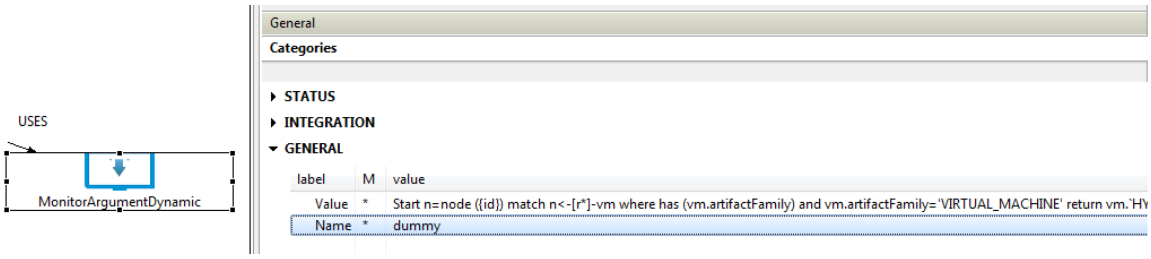


- CYPHER – In the value field, cypher query can be provided to pick up the mandatory variables from nfvd topology. Kindly refer NEO4J 1.9.6 documentation for more information regarding cyphers. Only `GET` queries are supported here, `CREATE` and `UPDATE` queries will not work and will be ignored.
  For example:

Ex: "Start n=node ({id}) match n<-[r*]-vm where has (vm.artifactFamily) and vm.artifactFamily='VIRTUAL_MACHINE' return vm.`HYPERVISOR.HOSTNAME` as host, 'constant' as parameter" .

- {id} – refers to the MONITOR to which the MONITOR_ARGUMENT:CYPHER is related with relation USES. It would be filled by NFVD automatically.

The user must provide aliases like `host` and `parameter` that match the mandatory variables in SiteScope templates.



10.2.4 Virtual Machine Monitoring credentials

Monitoring credentials for a Virtual Machine are dynamically picked up from one of the following components:

- Authentication
- VIM
- Hypervisor

`DEPLOYMENT.Type` in Monitor:Generic artifact can be set as:

- `AUTO` – Credentials are received from Authentication if ceilometer is supported by VIM. If Authentication is not present, then VIM is used. If VIM is not present, then Hypervisor is used to receive credentials.
- `VIM` – Credentials are received from VIM.
- `HYPERVISOR` – Credentials are received from HYPERVISOR.

10.2.5 Understanding other elements of Monitoring

The following illustration shows a high-level resource model:

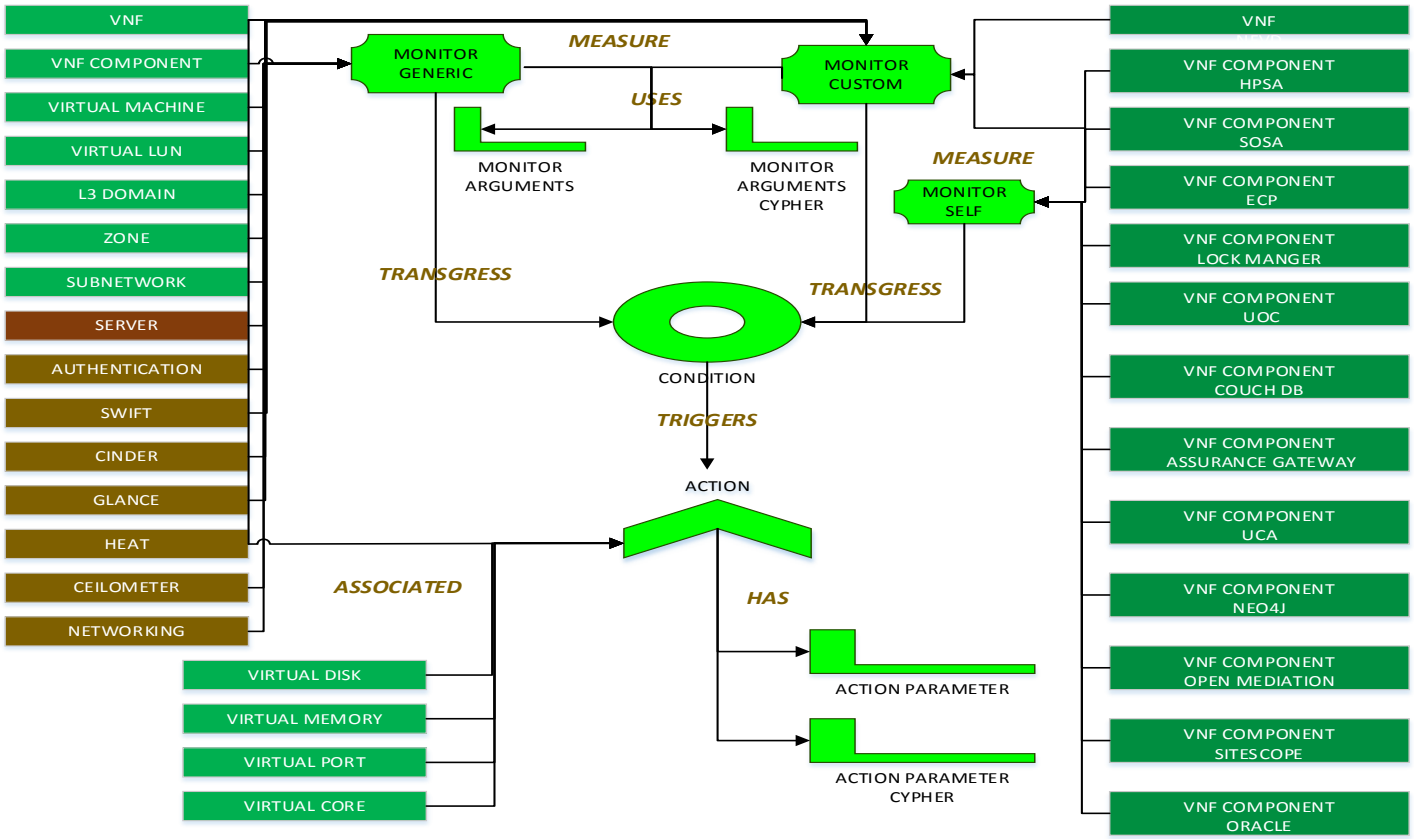


Figure 48: High-level resource model

NFVD supports out of the box monitors for the following components:

Table 10: Out of the box monitors for components

Component	Monitors
`VIRTUAL_MACHINE`	• CPU • Memory • DiskRead • DiskWrite • DiskUsage • NetworkRx • NetworkTx
`SERVER`	• CPU • Memory • DiskRead • DiskWrite • NetworkRx • NetworkTx
`AUTHENTICATION`	Process
`SWIFT`	Process
`CINDER`	Process
`GLANCE`	Process
`HEAT`	Process
`CEILOMETER`	Process
`NETWORKING`	Process
`L3Domain`	Network
`Zone`	Network
`Subnetwork`	Network
`Virtual LUN`	Storage

All of the components will have a `MONITOR:GENERIC` related to them with relationship type `MEASURE`.

Users can create custom monitors `MONITOR:CUSTOM` for any of the components and relate it to the component with relationship type `MEASURE`.

`MONITOR:SELF` can only be associated to the VNFC's listed in the following table.

Table 11: VNFC's with MONITOR:SELF

Component	Monitors
`VNFC:SOSA`	• Process • Logs
`VNFC:LOCK_MGR`	• Process • Logs
`VNFC:ECP`	• Process

	• Logs
`VNFC:HPSA`	• Process • Logs
`VNFC:ASSURANCE_GATEWAY`	• Process • Logs
`VNFC:NEO4J`	• Process • Logs
`VNFC:OPEN_MEDIATION`	• Process • Logs
`VNFC:SITESCOPE`	• Process • Logs For Windows and Linux.
`VNFC:UCA`	• Process • Logs
`VNFC:ORACLE`	• Process • Logs
`VNFC:POSTGRES`	• Process • Logs
`VNFC:UOC`	• Process • Logs
`VNFC:COUCHDB`	• Process • Logs

`MONITOR:SELF` needs to be associated to the VNFC only in cases where the user requires activity based on threshold breach.

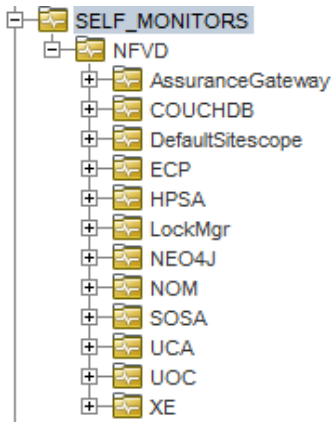


Figure 49: SiteScope – Self monitor template

NOTE: The GENERAL. Name attribute of VNF:NFVD should be unique across sites in Geo Redundant Setup

OpenStack services like Compute, Authentication (such as Keystone), Swift, Glance, Heat, Cinder, Networking, and Ceilometer can be monitored by associating them with the `MONITOR:GENERIC` artifact with `MEASURE` relationship.

`MONITOR` can be associated to `MONITOR_ARGUMENTS`, which have two versions:

- `MONITOR_ARGUMENT:ARGUMENT` - it contains name-value pairs, where name has to map to mandatory variables required by template in SiteScope.
- `MONITOR_ARGUMENT:CYPHER` - in the value field, cypher query can be provided to receive the mandatory variables from neo4j.

Example:

```
Start n= node ({id}) match n<-[r*]-vm where has(n.artifactId) and
vm.artifactFamily='VIRTUAL_MACHINE' return vm.`HYPERVISOR.HOSTNAME` as host,
'constant' as parameter
```

Where

- `{id}` refers to the `MONITOR's NodeId`, which Assurance completes automatically. The user does not have to pass this value manually and it can be used as is.
- Alias host and parameter refer to variables in the SiteScope template. `host` is a dynamic value in topology and `parameter` is a constant value. If alias does not match the mandatory variables in the SiteScope template exactly, then those values will be ignored.

Make sure XML characters in the cypher provided as part of the value field are escaped according to the following table.

CAUTION: Resource Modeler performs automatic character escaping. If the XML code is manually edited, then the user has to perform character escaping as indicated in the following table.

Table 12: Escape characters

Symbol	Escape character
"	"
'	'
<	<
>	>
&	&

Only `GET` queries are supported here. `CREATE` and `UPDATE` queries will not work and will be ignored.

`MONITOR` will be associated to `CONDITION` with `TRANSGRESS` relationship type.

`CONDITION` can be one of the following three types:

- ERROR
- WARNING
- GOOD

`CONDITION` has an `EXPRESSION` attribute under the `GENERAL` category. This parameter is used to define the expression, which contains the counter and threshold value.

Example:

```
cpu_usage_average > 90
```

Where `cpu_usage_average` is the counter variable and `90` is the threshold value. In the SiteScope template, the counter value will map to the variable such as `cpu_usage_average_error, cpu_usage_average_good`, `cpu_usage_average_warning` (for example, `counter_ConditionType`).

The **TYPE** attribute under the **GENERAL** category can have the following values:

- ERROR
- WARNING
- GOOD

The following table indicates the expressions available for various out of the box monitors.

NOTE: A new availability threshold is available only for ERROR condition. This enables users to configure an action when SiteScope is not able to get KPIs from an external system, due to issues such as connection problems, system not responding, and so on. This is introduced to clearly distinguish between scenarios where KPI metrics threshold was breached and scenarios where SiteScope is unable to collect the threshold.

Table 13: Regular expressions for out of box monitors

Monitor	ERROR	WARNING	GOOD
CPU	`cpu_usage_average` > THRESHOLD	`cpu_usage_average` > THRESHOLD	`cpu_usage_average` < THRESHOLD
	`cpu_usage_average` >= THRESHOLD	`cpu_usage_average` >= THRESHOLD	`cpu_usage_average` >= THRESHOLD
	`cpu_usage_average` < THRESHOLD	`cpu_usage_average` < THRESHOLD	`cpu_usage_average` < THRESHOLD
	`cpu_usage_average` <= THRESHOLD	`cpu_usage_average` <= THRESHOLD	`cpu_usage_average` < THRESHOLD
	`cpu_usage_average` == THRESHOLD	`cpu_usage_average` == THRESHOLD	`cpu_usage_average` == THRESHOLD
	availability contains n/a		
DiskRead	`disk_read_requests` > THRESHOLD	`disk_read_requests` > THRESHOLD	`disk_read_requests` < THRESHOLD
	`disk_read_requests` >= THRESHOLD	`disk_read_requests` >= THRESHOLD	`disk_read_requests` >= THRESHOLD
	`disk_read_requests` < THRESHOLD	`disk_read_requests` < THRESHOLD	`disk_read_requests` < THRESHOLD
	`disk_read_requests` <= THRESHOLD	`disk_read_requests` <= THRESHOLD	`disk_read_requests` <= THRESHOLD
	`disk_read_requests` == THRESHOLD	`disk_read_requests` == THRESHOLD	`disk_read_requests` == THRESHOLD
	availability contains n/a		
DiskWrite	`disk_write_requests` > THRESHOLD	`disk_write_requests` > THRESHOLD	`disk_write_requests` < THRESHOLD
	`disk_write_requests` >= THRESHOLD	`disk_write_requests` >= THRESHOLD	`disk_write_requests` >= THRESHOLD
	`disk_write_requests` < THRESHOLD	`disk_write_requests` < THRESHOLD	`disk_write_requests` < THRESHOLD
	`disk_write_requests` <= THRESHOLD	`disk_write_requests` <= THRESHOLD	`disk_write_requests` <= THRESHOLD

	disk_write_requests == THRESHOLD	disk_write_requests == THRESHOLD	disk_write_requests == THRESHOLD
	availability contains n/a		
DiskUsage	disk_usage > THRESHOLD	disk_usage > THRESHOLD	disk_usage < THRESHOLD
	disk_usage >= THRESHOLD	disk_usage >= THRESHOLD	disk_usage >= THRESHOLD
	disk_usage < THRESHOLD	disk_usage < THRESHOLD	disk_usage < THRESHOLD
	disk_usage <= THRESHOLD	disk_usage <= THRESHOLD	disk_usage <= THRESHOLD
	disk_usage == THRESHOLD	disk_usage == THRESHOLD	disk_usage == THRESHOLD
	availability contains n/a		
Memory	memory_usage_average > THRESHOLD	memory_usage_average > THRESHOLD	memory_usage_average < THRESHOLD
	memory_usage_average >= THRESHOLD	memory_usage_average >= THRESHOLD	memory_usage_average >= THRESHOLD
	memory_usage_average < THRESHOLD	memory_usage_average < THRESHOLD	memory_usage_average < THRESHOLD
	memory_usage_average <= THRESHOLD	memory_usage_average <= THRESHOLD	memory_usage_average <= THRESHOLD
	memory_usage_average == THRESHOLD	memory_usage_average == THRESHOLD	memory_usage_average == THRESHOLD
	availability contains n/a		
NetworkRx	network_bytes_received > THRESHOLD	network_bytes_received > THRESHOLD	network_bytes_received < THRESHOLD
	network_bytes_received >= THRESHOLD	network_bytes_received >= THRESHOLD	network_bytes_received >= THRESHOLD
	network_bytes_received < THRESHOLD	network_bytes_received < THRESHOLD	network_bytes_received < THRESHOLD
	network_bytes_received <= THRESHOLD	network_bytes_received <= THRESHOLD	network_bytes_received <= THRESHOLD
	network_bytes_received == THRESHOLD	network_bytes_received == THRESHOLD	network_bytes_received == THRESHOLD
	availability contains n/a		
NetworkTx	network_bytes_transmitted > THRESHOLD	network_bytes_transmitted > THRESHOLD	network_bytes_transmitted < THRESHOLD
	network_bytes_transmitted >= THRESHOLD	network_bytes_transmitted >= THRESHOLD	network_bytes_transmitted >= THRESHOLD
	network_bytes_transmitted < THRESHOLD	network_bytes_transmitted < THRESHOLD	network_bytes_transmitted < THRESHOLD
	network_bytes_transmitted <= THRESHOLD	network_bytes_transmitted <= THRESHOLD	network_bytes_transmitted <= THRESHOLD
	network_bytes_transmitted == THRESHOLD	network_bytes_transmitted == THRESHOLD	network_bytes_transmitted == THRESHOLD
	availability contains n/a		

Process	status == 1	Not Applicable	status == 0
Logs	matches > THRESHOLD	Not Applicable	matches < THRESHOLD
Network	network_bytes_in_received > THRESHOLD	network_bytes_in_received > THRESHOLD	network_bytes_in_received < THRESHOLD
	network_bytes_out_transmitted > THRESHOLD	network_bytes_out_transmitted > THRESHOLD	network_bytes_out_transmitted < THRESHOLD
	network_packets_dropped_by_rate_limit > THRESHOLD	network_packets_dropped_by_rate_limit > THRESHOLD	network_packets_dropped_by_rate_limit < THRESHOLD
	network_packets_in_dropped > THRESHOLD	network_packets_in_dropped > THRESHOLD	network_packets_in_dropped < THRESHOLD
	network_packets_in_fault > THRESHOLD	network_packets_in_fault > THRESHOLD	network_packets_in_fault < THRESHOLD
	network_packets_in_received > THRESHOLD	network_packets_in_received > THRESHOLD	network_packets_in_received < THRESHOLD
	network_packets_out_dropped > THREHOLD	network_packets_out_dropped > THRESHOLD	network_packets_out_dropped < THRESHOLD
	network_packets_out_fault > THRESHOLD	network_packets_out_fault > THRESHOLD	network_packets_out_fault < THRESHOLD
	network_packets_out_transmitted > THRESHOLD	network_packets_out_transmitted > THRESHOLD	network_packets_out_transmitted < THRESHOLD
	availability contains n/a		

CONDITION will be associated to ACTION with TRIGGERS relationship type.

- GENERAL.Type: Possible values are SCALE_UP, SCALE_DOWN, SCALE_IN, SCALE_OUT, and SCRIPT.
- GENERAL.Path: Specify the location of the executable script if action type is SCRIPT.
- GENERAL.Operation_Mode: Possible values are CLOSED_LOOP or OPEN_LOOP. The default value is CLOSED_LOOP. Setting the value to OPEN_LOOP requires authorization by the user in the UCA_AUTOMATION console.

ACTION can be associated to ACTION_PARAMETER, which has two versions:

- ACTION_PARAMETER:GENERIC - it contains name value pairs required to execute an action (for example, it can be script variables for a script or input parameter for HPSA workflow).
- ACTION_PARAMETER:CYPHER - in value field, cypher query can be provided to receive the mandatory variables from NEO4J.

Example:

```
Start n=node ({id}) match n<-[r*]-vm where has(n.artifactId) and
vm.artifactFamily='VIRTUAL_MACHINE' return vm.`HYPERVISOR.HOSTNAME` as host,
'constant' as parameter
```

Where

- `{id}` refers to `ACTION's NodeId`, which is automatically completed by Assurance. The user does not have to pass this value, it can be used as is.
- `Alias host`, `parameter` refer to variables required to execute the action. `host` is a dynamic value in topology, and `parameter` is a constant value.

CAUTION: Resource Modeler performs automatic character escaping. If the XML code is manually edited, the user has to perform character escaping as indicated in the following table.

Table 14: Escape characters

Symbol	Escape character
"	"
'	'
<	<
>	>
&	&

Only GET queries are supported here. CREATE and UPDATE queries will not work and will be ignored.

10.3 Action Modeling

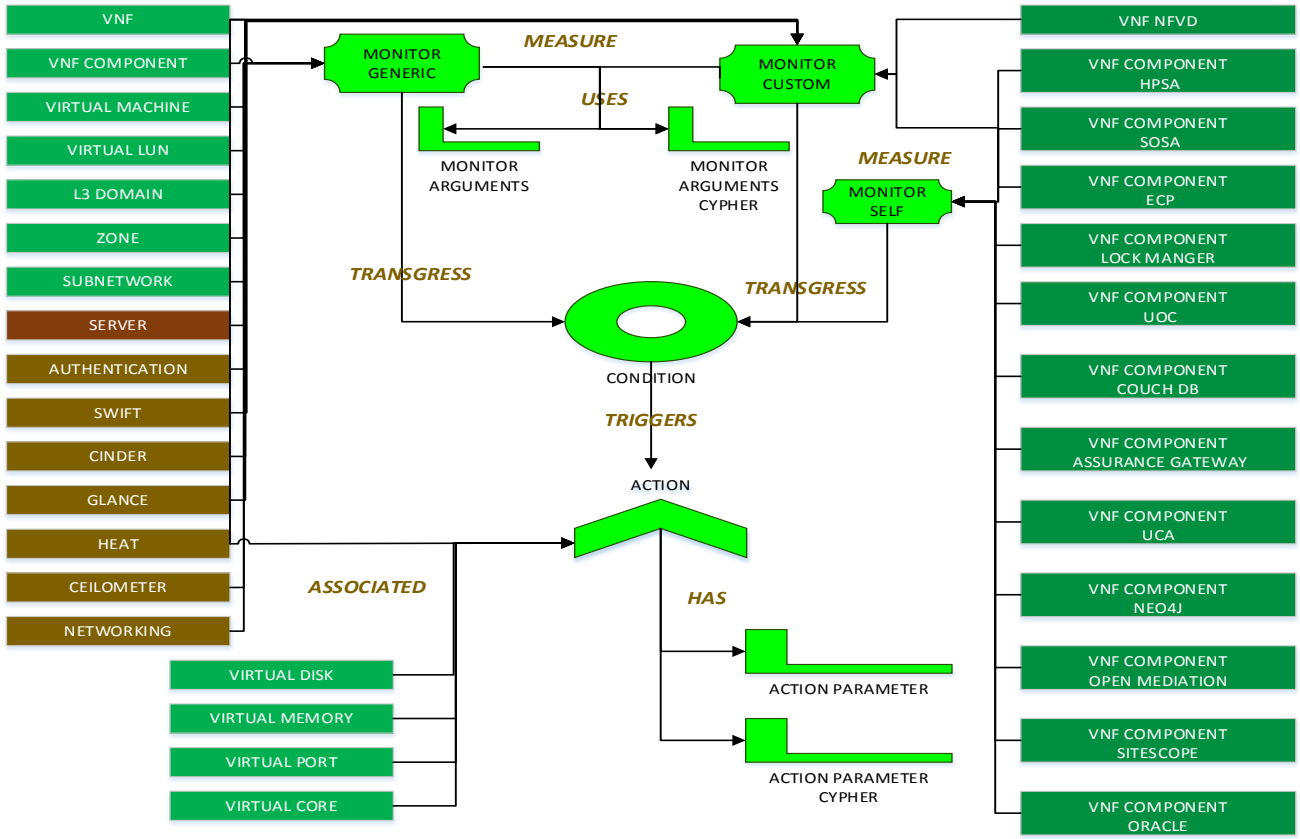


Figure 50: Action Modeling artifacts

10.3.1 Action artifact

Table 15: Action artifact attributes

Attribute	Description	Values
Name	A human readable name.	Mandatory: Yes
Type	Type of action.	Five available actions: • Scale-In • Scale-Out • Scale-Up • Scale-Down • Script Mandatory: Yes
Description	A human readable text.	Mandatory: No
Scale Value	The scale value (integer).	Valid only when type is Scale.
Script Path	The executable script.	Valid only when type is Script.

10.3.2 Action parameters artifact attributes

Table 16: Action parameters artifact attributes

Attribute	Description	Values
`Name`	A name.	Mandatory: Yes, if Action Type is Script and the script takes arguments as input.
`Type`	Future use.	
`Value`	A value associated with the name.	

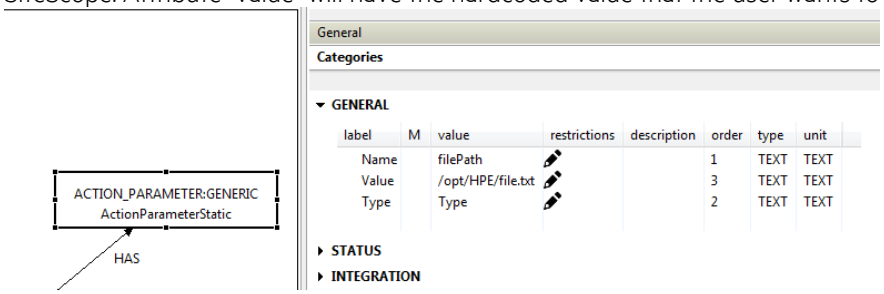
Action parameters artifacts can be associated to Action.

It is a way of passing static/dynamic arguments to an action like script. For example, a script might need the following arguments:

c. filePath: that provides a path to the file on the system. filePath variable can be passed as a hardcoded value by using ACTION_PARAMETER:GENERIC.
d. host: hypervisor hostname of the VM. Since the hypervisor hostname would not be known during template design time, it would have to be picked up dynamically from NFVD topology during orchestration. ipAddress variable can be passed by using ACTION_PARAMETER:CYPHER

Action Parameters are of two types:

- GENERIC – It contains name value pair. Attribute "name" has to map to mandatory variables required by template in SiteScope. Attribute "value" will have the hardcoded value that the user wants to configure.

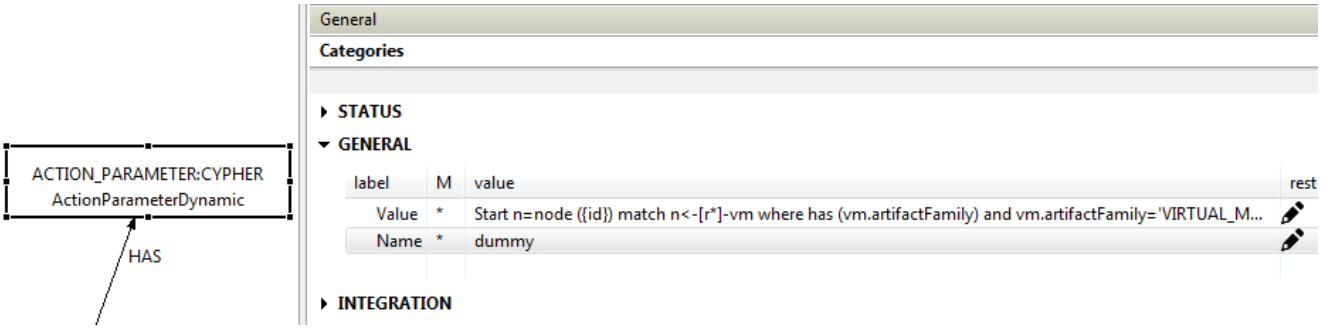


- CYPHER – In value field, cypher query can be provided to pick up the mandatory variables from nfvd topology. Kindly refer NEO4J 1.9.6 documentation for more information regarding cyphers. Only **GET** queries are supported here, **CREATE** and **UPDATE** queries will not work and will be ignored.
  For example:

Ex: "Start n=node ({id}) match n<-[r*]-vm where has (vm.artifactFamily) and vm.artifactFamily='VIRTUAL_MACHINE' return vm.`HYPERVISOR.HOSTNAME` as host, 'constant' as parameter" .

- {id} – refers to ACTION to which the ACTION_PARAMETER:CYPHER is related with relation USES. It would be filled by NFVD automatically. .

The user must provide aliases like `host` and `parameter` that match the mandatory variables in SiteScope templates.



10.3.3 Associated artifact ID for action

An action can be associated to any artifact in the NFV Model. Typically it is associated to:

- a Virtual Machine
- resources of a Virtual Machine (CPU, DISK, memory, and so on.)
- VNFC component
- VNF
- NS
- When a KPI breach is detected by the monitor, the action type and the associated artifact ID determine the action to be taken on the required artifact.

Example-1: Scale-Out

When there is KPI breach (for example, `ERROR`: high CPU usage) by the `CPU` monitor, and the action type is `Scale-Out`, and it is associated to a `VIRTUAL_MACHINE` artifact, then the NFV Director automatically triggers a scale-out action for the associated `VIRTUAL_MACHINE`.

Example-2: Script Action

When there is KPI breach (for example, `WARNING`: high DISK usage) by the `DISK` monitor, and the action type is `Script`, then the NFV Director automatically executes the script specified in the action attribute `ScriptPath` by taking action parameters as input values to the script.

10.4 Out of box monitors provided by NFV Director

The following figure indicates the list of Monitors supported out of the box.

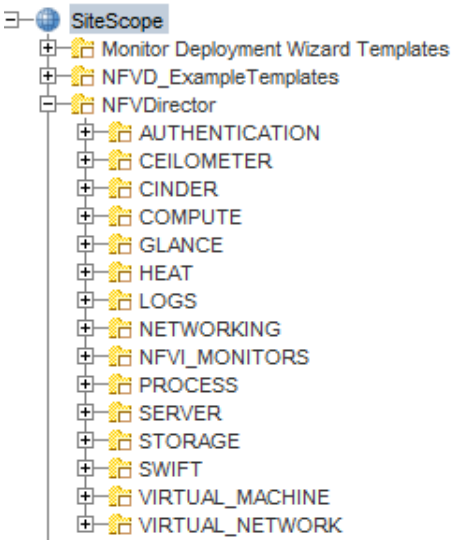


Figure 51: Monitors supported out of box

10.4.1 Process monitors for OpenStack services

Status is the only counter supported here in conditional expressions. Only `ERROR` (non-zero status value) and `GOOD` (zero status value) conditions are supported. The `version` field indicates Keystone version (V2 and V3). `urlType` indicates either the use of public or private IP Address to access the service.

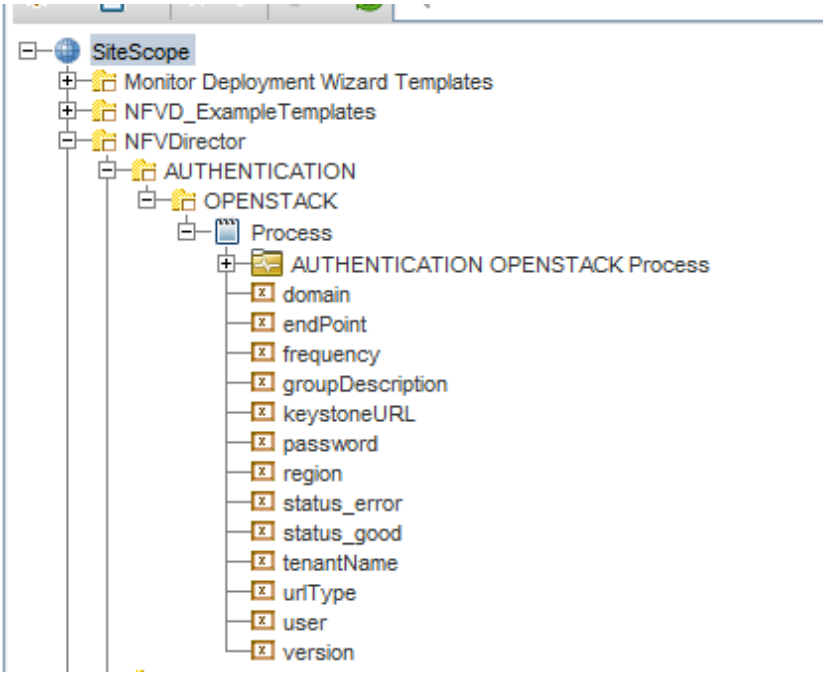


Figure 52: Process monitors for OpenStack services

10.4.2 Physical server monitors

The following table and figure indicate the counters available for physical servers:

Table 17: Physical server counters

Counter	Description
CPU	`cpu_usage_average`
Memory	`memory_usage_average`
DiskRead	`disk_read_requests`
DiskWrite	`disk_write_requests`
NetworkRx	`network_bytes_received`
NetworkTx	`network_bytes_transmitted`

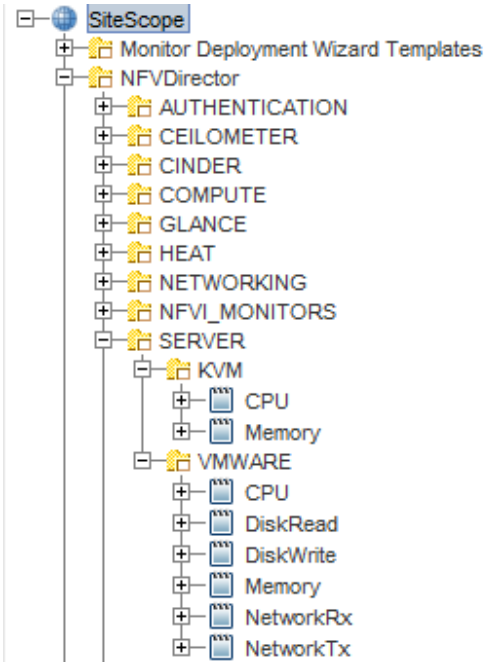


Figure 53: Physical server monitors

10.4.3 Virtual machine monitors

VMWARE monitor also supports VCENTER. The following table and figure indicate the counters available for virtual machines:

Table 18: Virtual machine counters

Counter	Type	Description
CPU	Numeric (%value)	%cpu_usage_average
Memory	Numeric	memory_usage_average
DiskRead	Numeric	Number of disk_read_requests
DiskWrite	Numeric	Number of disk_write_requests
DiskUsage	Numeric	disk_usage
NetworkRx	Numeric	network_bytes_received
NetworkTx	Numeric	network_bytes_transmitted

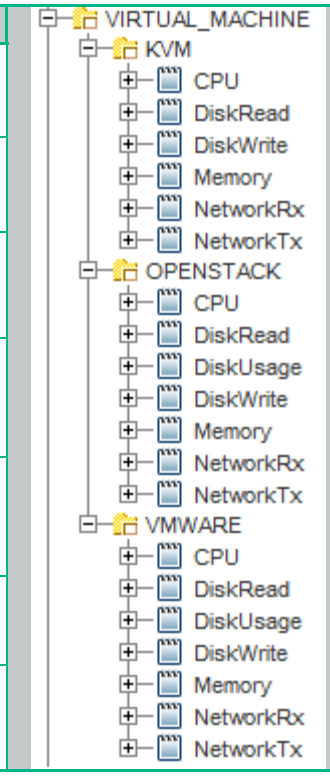


Figure 54: Virtual machine monitors

10.4.4 Self monitors

Self-monitoring, is the capability of NFVD to monitor its own health. NFVD monitors, by default, all processes that are part of its sub-components. It updates the current status of these processes periodically and generates a notification if it detects any failure.

This capability can be further customized to trigger an action when a specific condition is breached and to add more monitors. like monitor log files for errors, monitoring of CPU, memory and disk resources of the host system/VM that is running NFVD. Additionally, custom-developed monitors can be used to monitor any other useful parameters.

NFVD monitors the health of its components using SiteScope which is designated as the default one and the default SiteScope itself is monitored by assurance gateway process.

By default, NFVD supports monitoring the following:

- All sub-component processes
- Optionally, log file monitoring for these processes (disabled by default)

The Process monitor has Windows and Linux templates.

Table 19: Linux and Windows Process monitor attributes

	Linux	Windows
Error	`pid contains n/a`	`status contains not found`
		`status contains unavailable`
Good	`pid does not contain n/a`	`status contains running`

10.4.5 Virtual network for Nuage

The virtual network for Nuage supports multiple counters.

Table 20: Virtual network counters for Nuage

Counter	Description
Bytes received	`network_bytes_in_received`
Bytes transmitted	`network_bytes_out_transmitted`
Packet drop rate limit	`network_packets_dropped_by_rate_limit`
Received packets dropped	`network_packets_in_dropped`
Received faulty packets	`network_packets_in_fault`
Packets received	`netwowk_packets_in_received`
Transmitted packets dropped	`network_packets_out_dropped`

Transmitted faulty packets	`network_packets_out_fault`
Packets transmitted	`network_packets_out_transmitted`

Figure 55: Virtual network monitors for Nuage

10.5 NFVD Self-Monitoring

10.5.1.1 How NFVD Self-Monitoring works?

When NFVD is installed, by default, NFVD is modelled as a VNF, and its sub-components as VNF_COMPONENTs. This is called a self-management VNF instance. Based on the parameters configured as part of this self-management VNF instance, the assurance gateway of NFVD would deploy the process monitors into default SiteScope and assurance gateway starts the monitoring of default SiteScope,

All updates on parameter settings to self-management instance are automatically reflected in self-monitors. Normally, this check is made every 15 mins, after the assurance gateway starts up.

10.5.1.2 Default NFVD model for simplex deployment setup

Below figure shows a graphical view of the NFVD self-management VNF instance.

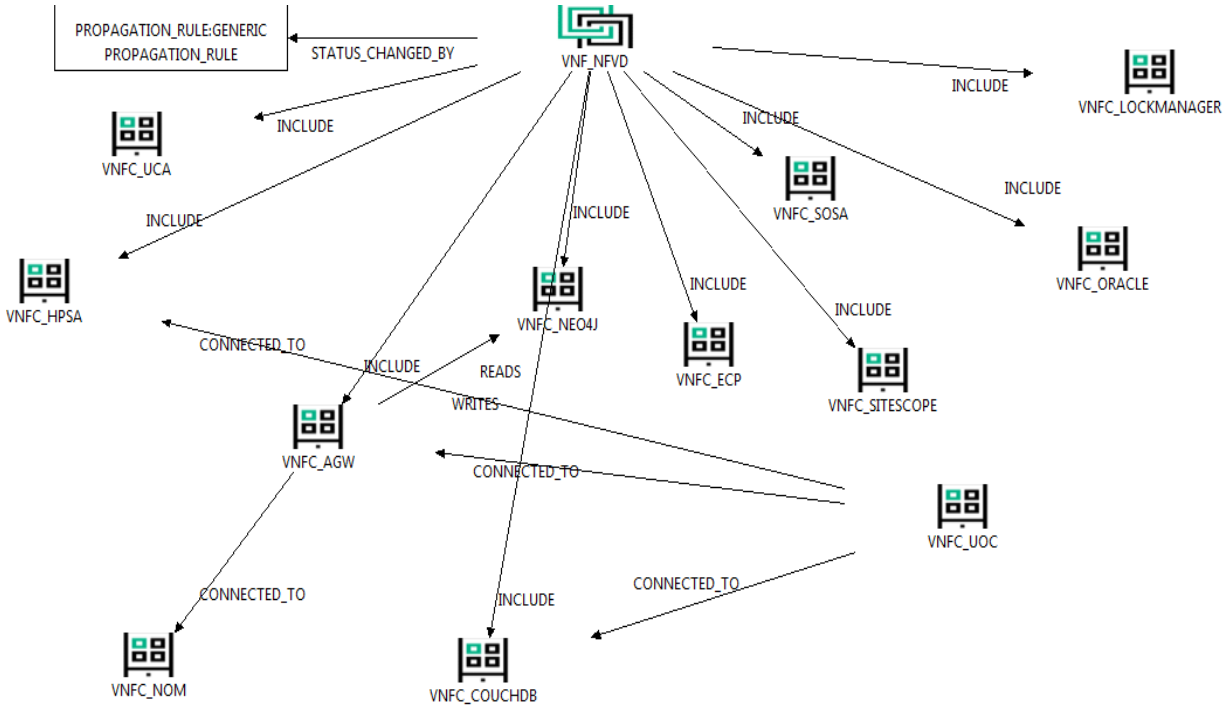


Figure 56: Graphical view of NFVD Self-Management VNF instance

The parameter settings can be edited/updated using NFVD GUI (and/or by VNF resource modeler tool) by performing the following steps:

1. Login to NFVD GUI as domain user (nfvd).
2. Navigate to Instances Menu.
3. Select VNFs.
4. Select VNF of category NFVD.

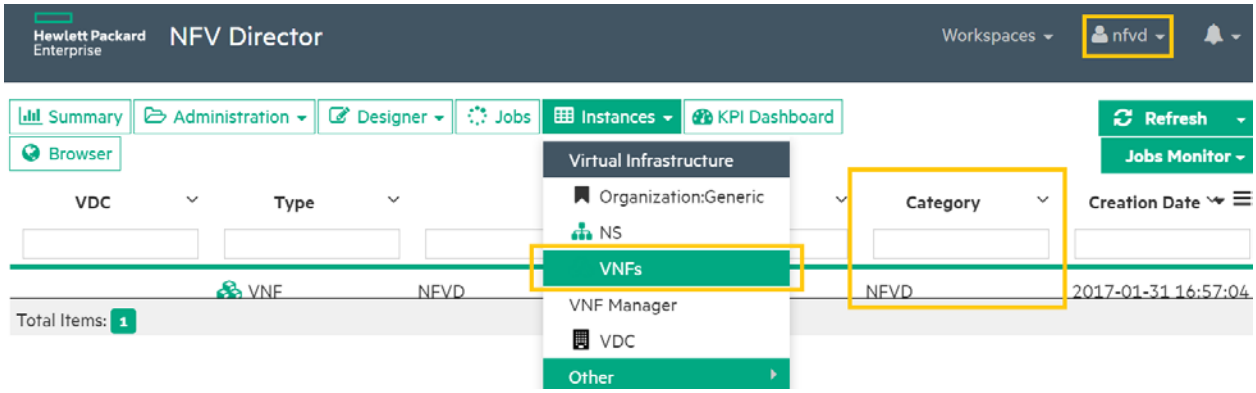


Figure 57: NFVD Component

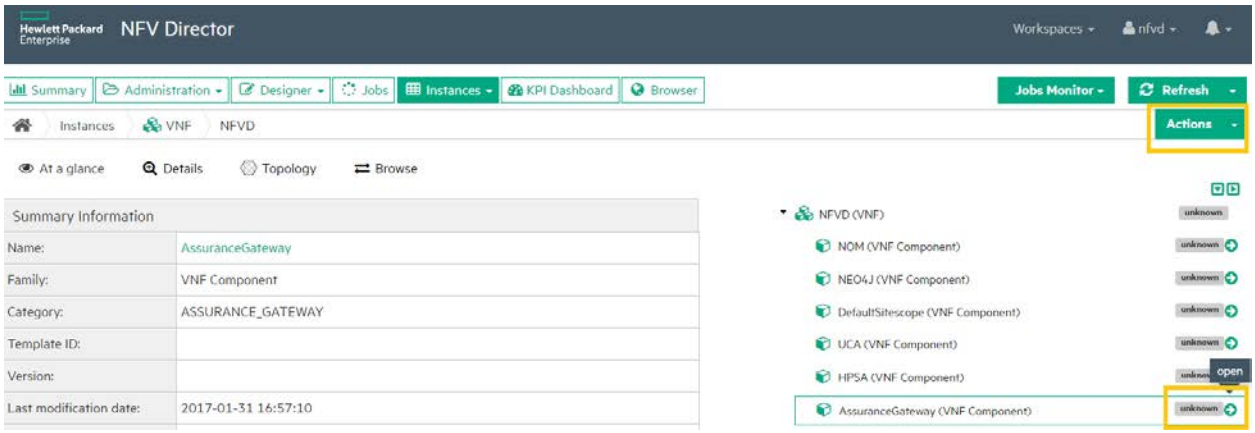


Figure 58: NFVD components, select one of the NFVD components (Assurance Gateway : VNF Component)

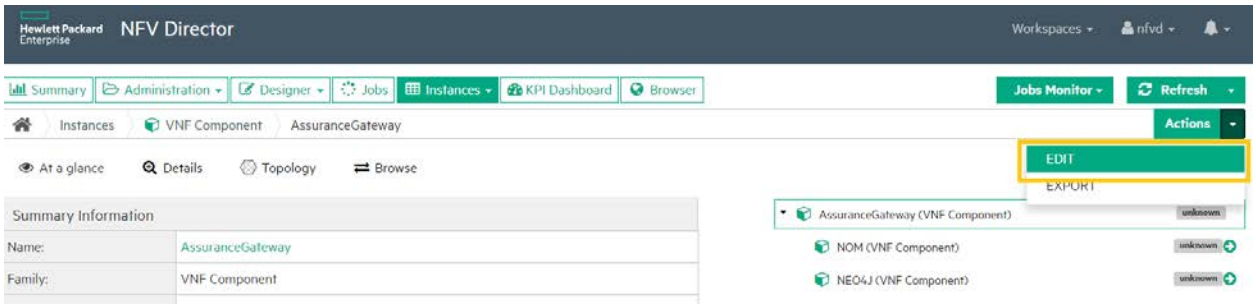


Figure 59: NFVD Components, select Actions and Click Edit to see the default details of Assurance Gateway: VNF Component)

Process and Log monitors are controlled by ENABLED attribute.

ENABLED Attribute	MONITOR	LOG_MONITOR
True	Both process and Log Monitor will be enabled	Note: Log monitor will be deployed only on provision of log path and log file name and log pattern
False	No monitoring. Process Monitor is disabled	No monitoring. Log monitor is disabled

Note: ENABLED: false means MONITOR process and LOG_MONITOR are disabled.

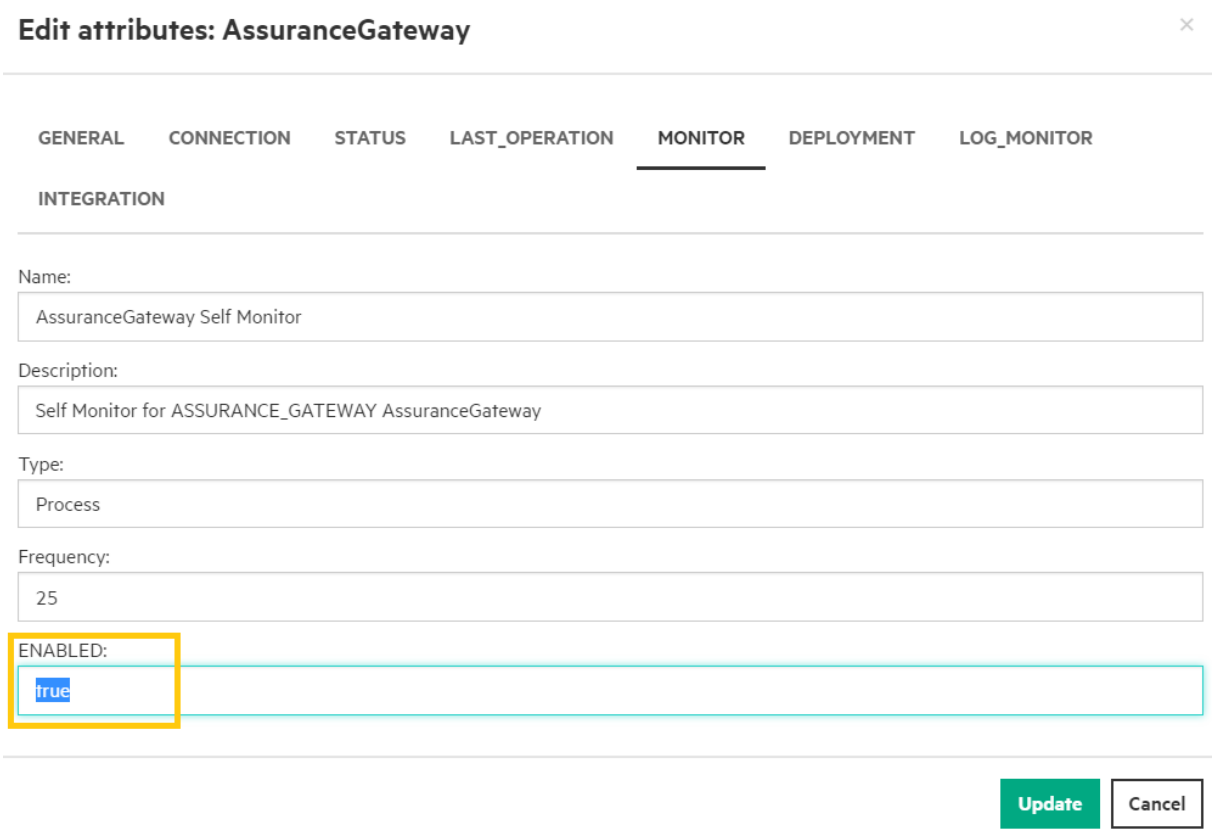


Figure 60: Monitor Process Example in GUI with ENABLED attribute, Assurance Gateway: VNF Component

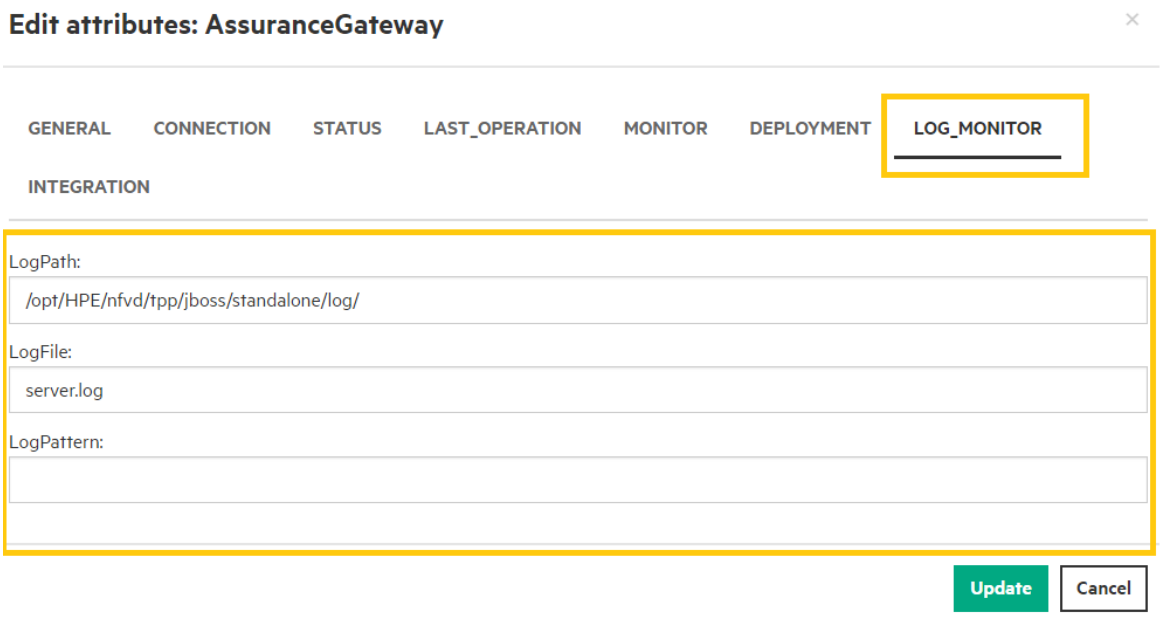


Figure 61: LOG_MONITOR Example in the GUI shows the LogPath, LogFile and LogPattern editable fields, for Assurance Gateway: VNF Component example

In order to simplify this task, a sample definition of self-management VNF instance is provided, and is located at in the Assurance machine at: /opt/HPE/nfvd/example

The following figure shows one of the components Assurance Gateway, similarly, all components are modelled.

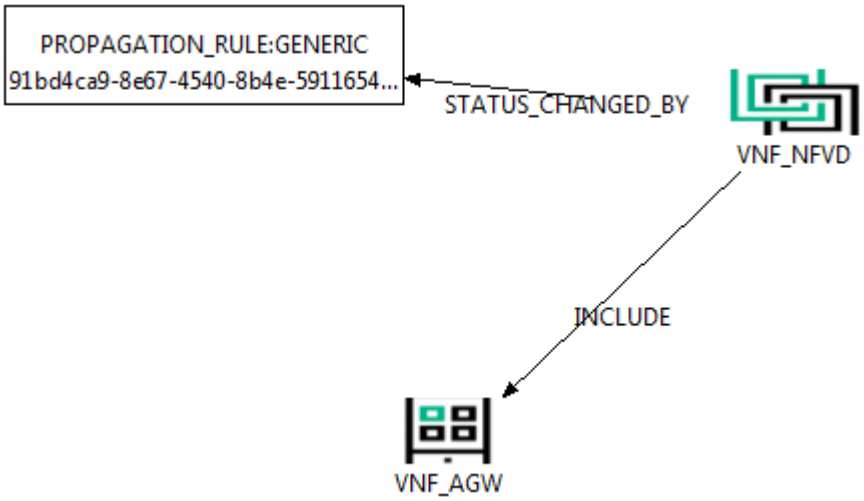


Figure 62: Modeling the Self Monitor component

Note: PROPAGATION_RULE controls how the status of children affects their parent objects, and is used to reflect and consolidate the status of individual components into overall status of NFVD.

10.5.1.3 Monitoring in SiteScope

This section describes the monitoring templates defined in SiteScope for NFVD self-monitoring. They can be accessed by performing the following steps:

1. Login to SiteScope , http://<SiteScope host IP>:18888, as admin user
2. Navigate to Templates Tab
3. Under NFVDirector section, select PROCESS.

Following NFVD components are monitored using default SiteScope:

- HPSA
- UCA-EBC
- NEO4J
- Assurance Gateway
- SOSA
- LOCK MANAGER
- ECP
- SiteScope (non-default ones)
- UOC
- CouchDB
- Oracle
- OpenMediation

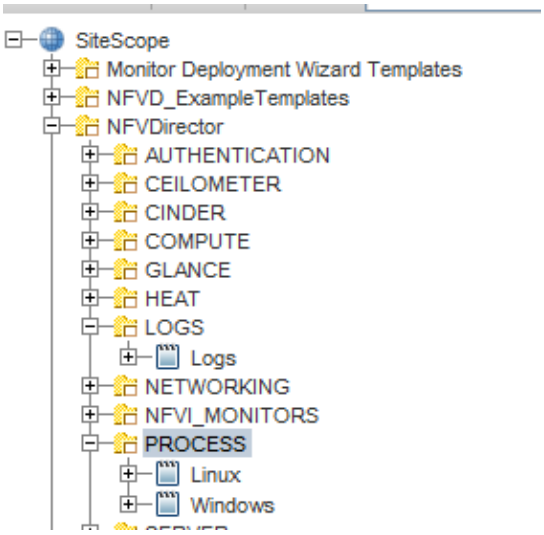


Figure 63: Self monitors

10.5.1.3.1 SiteScope view for deployed monitors

Deployed monitors can be viewed in SiteScope by performing the following steps:

1. Login to SiteScope , http://<SiteScope host IP>:18888, as admin user
2. Navigate to Monitors Tab

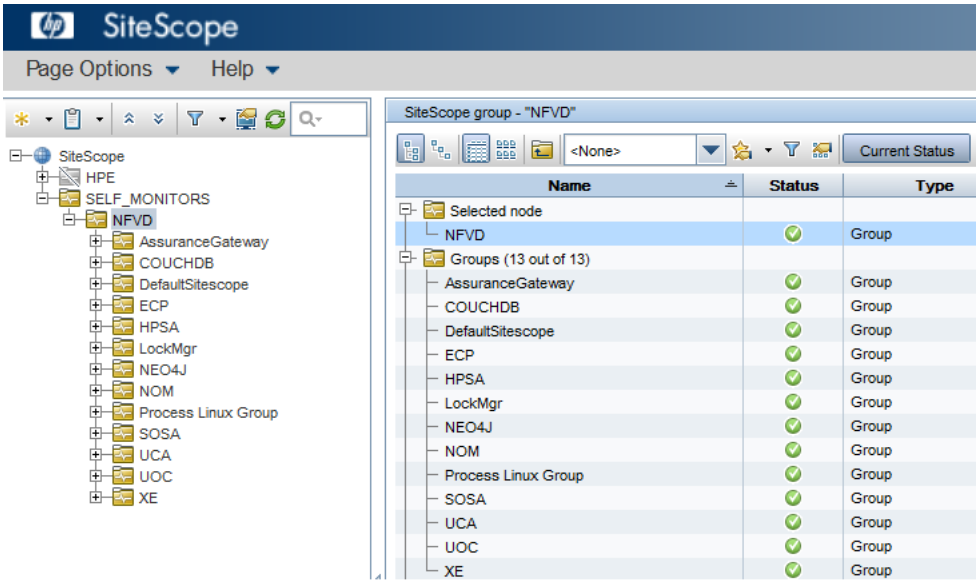


Figure 55: SiteScope monitors tab, showing NFVD self-monitor as deployed

10.5.1.3.2 SiteScope view for Self-Monitor status

Self monitors status can be viewed in SiteScope by performing the following steps:

1. Login to SiteScope , http://<SiteScope host IP>:18888, as admin user
2. Select Unified Console option

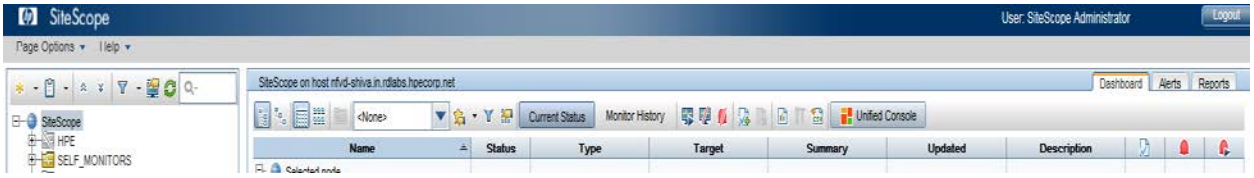


Figure 55: SiteScope Unified Console option

The following screen appears in SiteScope, after clicking the Unified Console option.

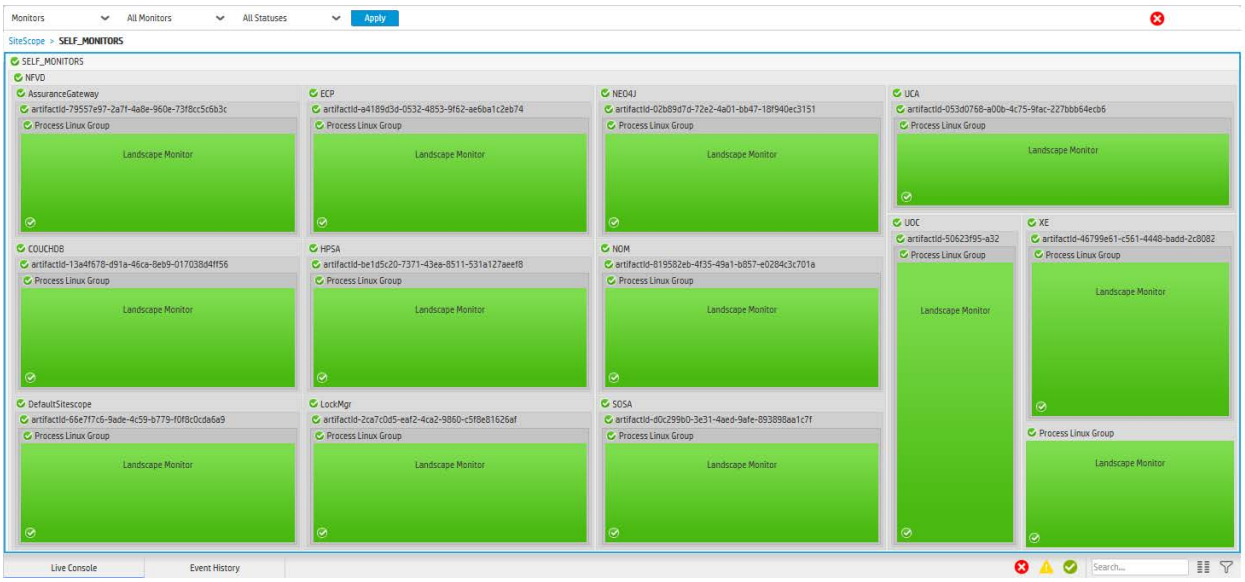


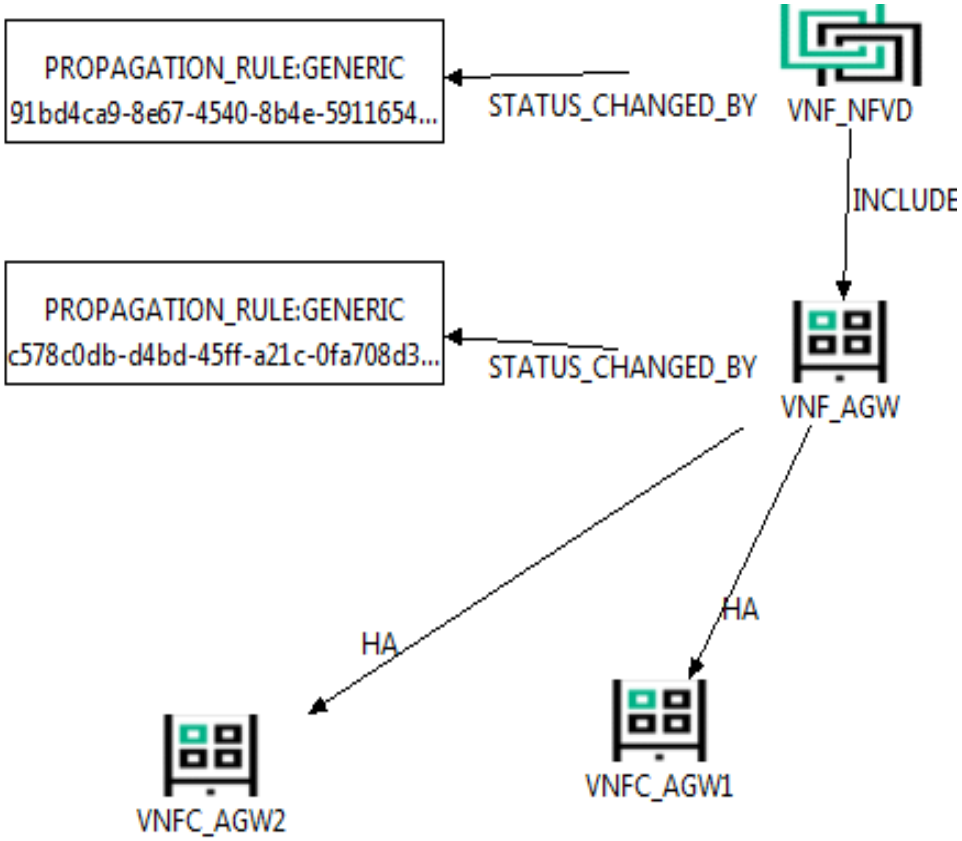
Figure 56: SiteScope Unified Console showing NFVD self-monitors up and running

10.5.1.4 Self-Monitoring of NFVD High Availablity (HA) setup

In a HA setup, there would be two instances of each and every component in NFVD, primary and secondary. Components can be in active/active or active/passive or active/hot-standby modes.

10.5.1.4.1 Self-Management VNF instance for HA setup

In a HA setup, there would be two instances of each component. For example:



Above Figure shows one of the components Assurance Gateways.

10.5.1.4.2 SiteScope view for NFVD HA Self-Monitor status

Self-Monitor status can be viewed in SiteScope by performing the following steps:

1. Login to SiteScope , http://<SiteScope host IP>:18888, as admin user.
2. Select Unified Console option.

The following will be displayed in SiteScope:

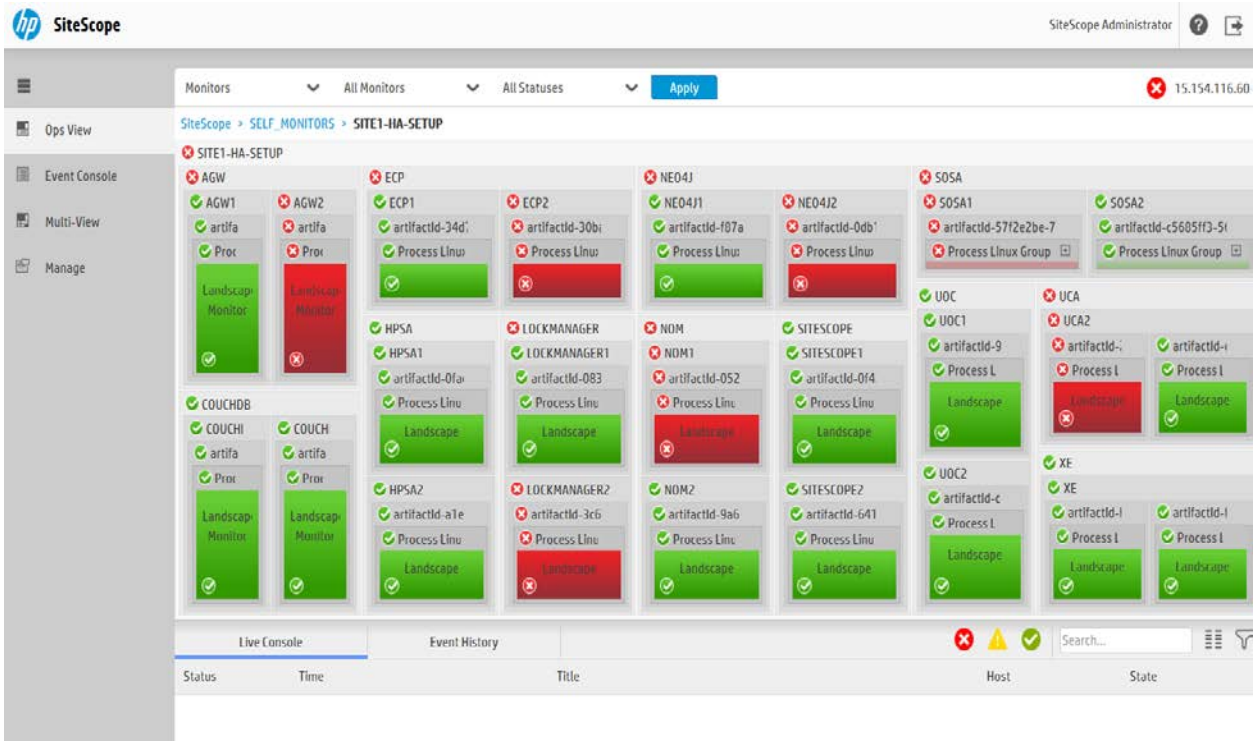


Figure 57: SiteScope Unified Console showing NFVD self-monitors in HA setup

NOTE: In the snapshot shown above, the modes of the components in HA setup used are the following:

NFVD COMPONENT	MODE OF OPERATION
HPSA/UOC/COUCHDB	ACTIVE:ACTIVE
Assurance Gateway/SOSA/LOCK MANAGER/UCA-EBC/NEO4J	ACTIVE:PASSIVE
SITESCOPE	ACTIVE/HOT STANDBY

10.5.1.5 Self-Monitoring of NFVD Geo Redundancy (GR) setup

NFVD GR setup is composed of two sites having one HA setup each, and working in tandem.

10.5.1.5.1 Self-Management VNF instance for GR setup

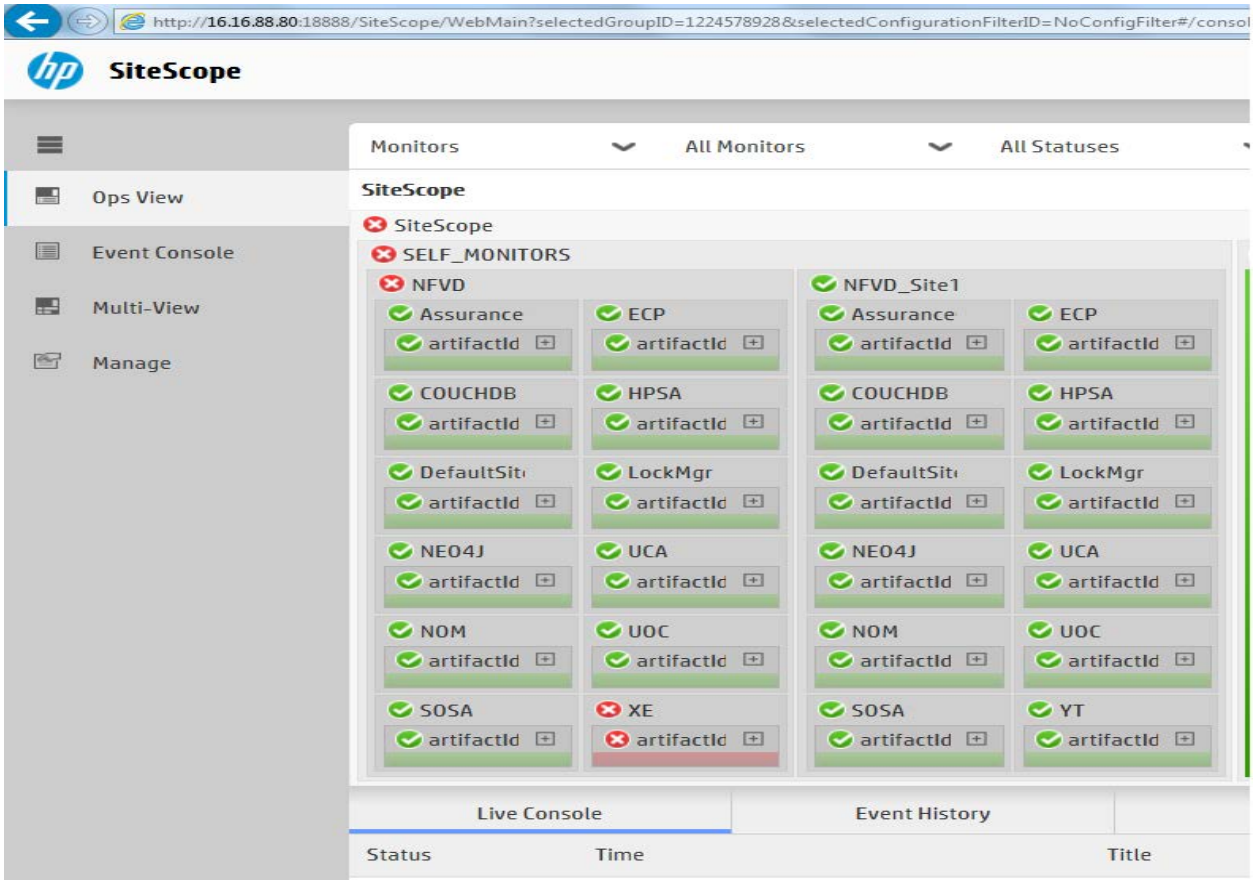
It is similar to “Self-Management VNF instance for HA setup”, but there would be two sites with this model.

10.5.1.5.2 SiteScope view for NFVD GR Self-Monitor status

Self-monitor status can be viewed in SiteScope by performing the following steps:

1. Login to SiteScope , http://<SiteScope host IP>:18888, as admin user.
2. Select Unified Console option, as shown below.

The follwing screen will be displayed in SiteScope:



The snapshot above is for two NFVD sites having Simple setup. Here, NFVD: represents SITE1, NFVD_SITE1: represents SITE2

10.5.1.6 Customizations for Self-Monitors

Prerequisite - NFVD resource modeler

This section explains how actions can be triggered for NFVD components when a condition breaches some threshold.

Monitor: Self gives the capability to monitor a VNFC, VM with CONDITION and ACTION, (just like VNF). The Action can be script execution.

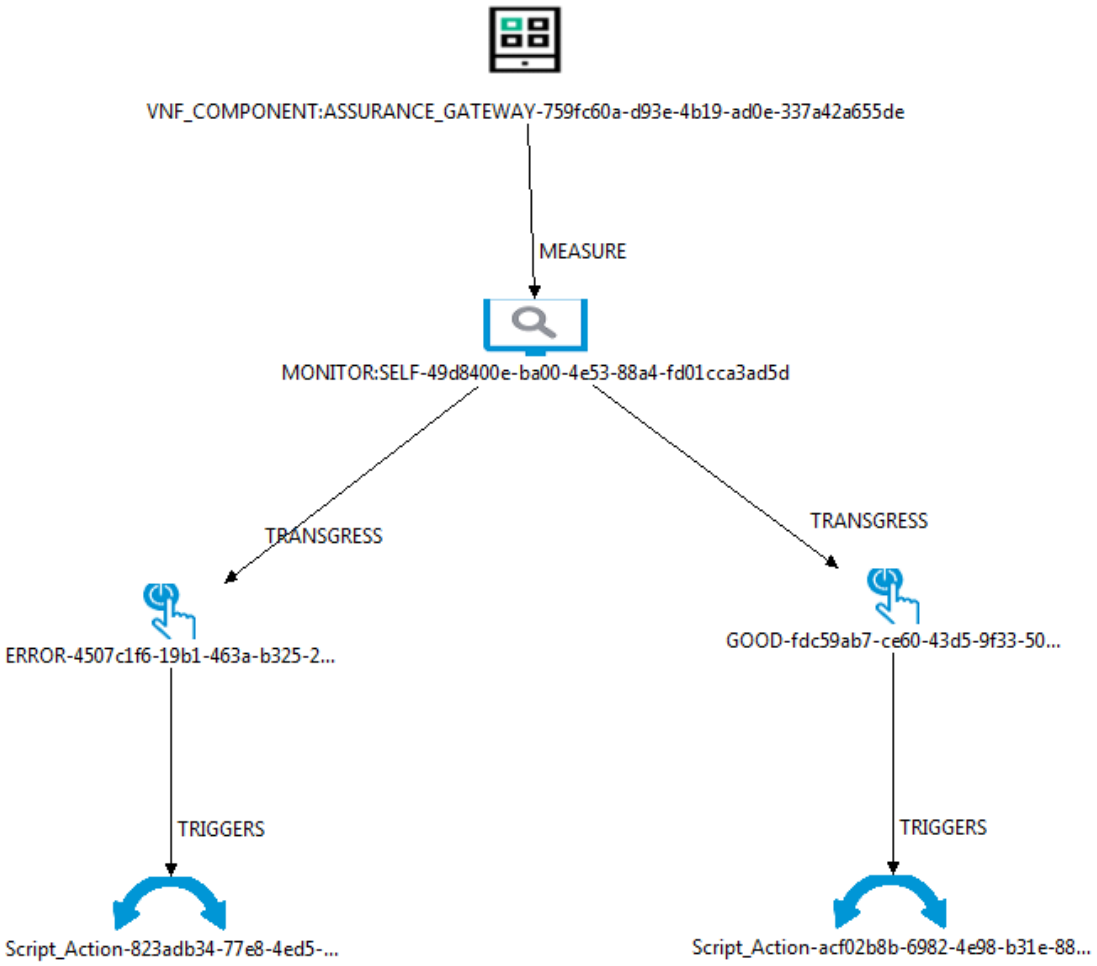


Figure 102: Monitor: Self Example in resource modeler.

In a simplex NFVD setup as shown above, the VNF Component, such as an Assurance Gateway component (example component) with a new UUID is designed in the resource modeler. It can be attached to a MONITOR:SELF with MEASURE relationship and in turn to CONDITION and ACTION artifacts.

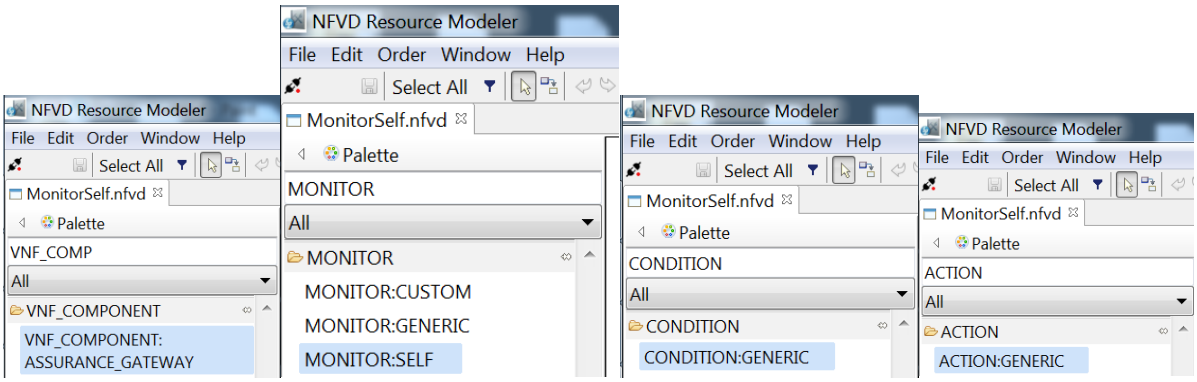


Figure 103: How to select VNFC,Monitor: Self,Condition,Action Example in resource modeler

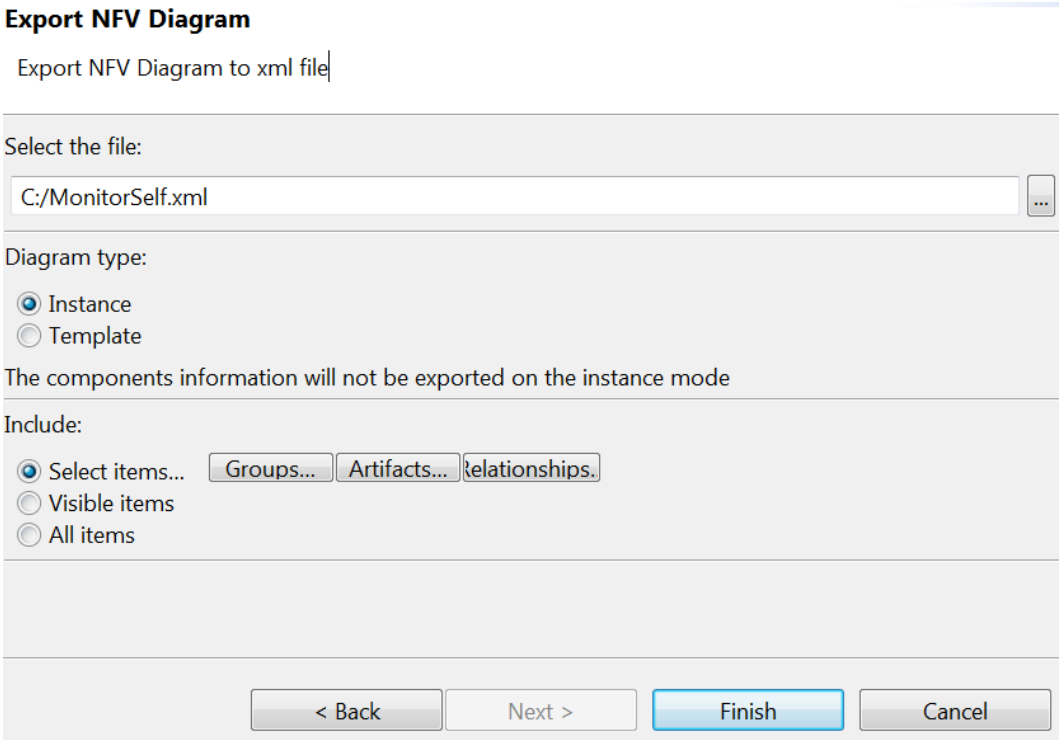


Figure 104: Export as instance, Example in resource modeler

Go to file > export > make sure instance is selected. (Fill all the mandatory variables of each node, otherwise it will not allow the export)

Important:

1. In the above design, VNF Component: Assurance Gateway artifact should not be uploaded.
2. Note down the artifact Id from GUI for VNF_Component instance.

   Example: artifactid of VNF_Component : Assurance Gateway

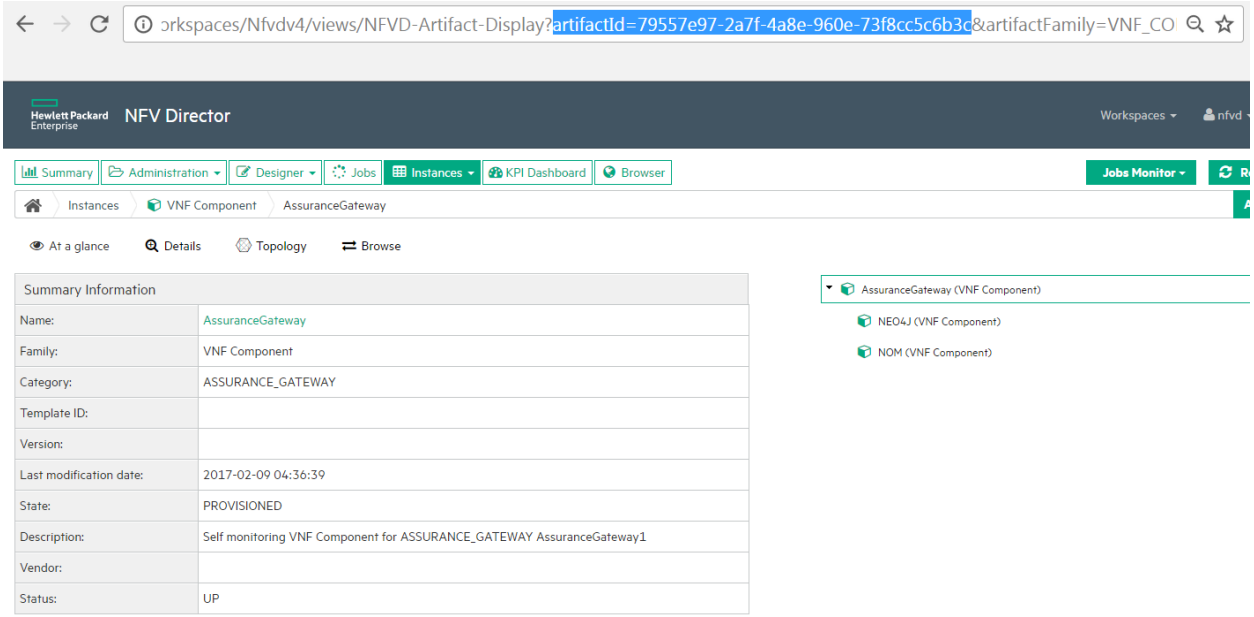


Fig 105 Note Down VNFC artifact Id from GUI

1. Replace the parent-artifact-id with noted VNFC artifactId.

Ex: POST http://{ FF-IP -ADDRESS >:8080/nfvd/instance/relationship

```
<relationship-instances xmlns="http://www.hp.com/nfvd">
      <relationship-instance xmlns="http://www.hp.com/nfvd">
            <relationship-type>MEASURE</relationship-type>
            <parent-artifact-id>{replace noted VNFC artifactid from gui}</parent-artifact-id>
            //VNFComponent ArtifactId
            <child-artifact-id>cf73b9fa-7ff4-4092-bfeb-6e015f4fa1be</child-artifact-id>
            <categories/>
            <status>
                  <label>ENABLED</label>
                  <visible-label>ENABLED</visible-label>
                  <enabled>true</enabled>
            </status>
            <creation-timestamp>2017-01-24T17:42:52.508+0530</creation-timestamp>
            <update-timestamp>2017-01-24T17:42:52.508+0530</update-timestamp>
      </relationship-instance>
</relationship-instances>
```

Monitor: CUSTOM

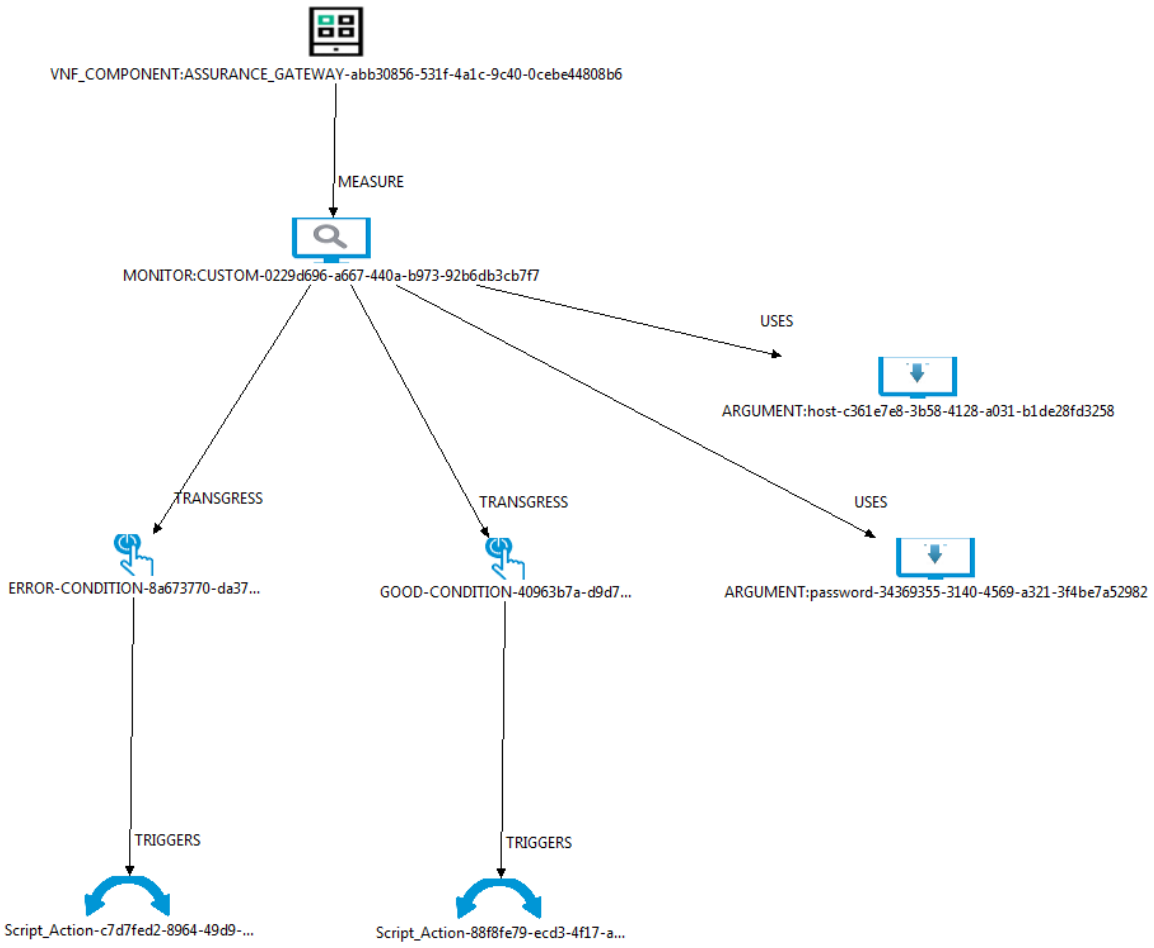


Fig 106 Monitor Custom in resource modeler

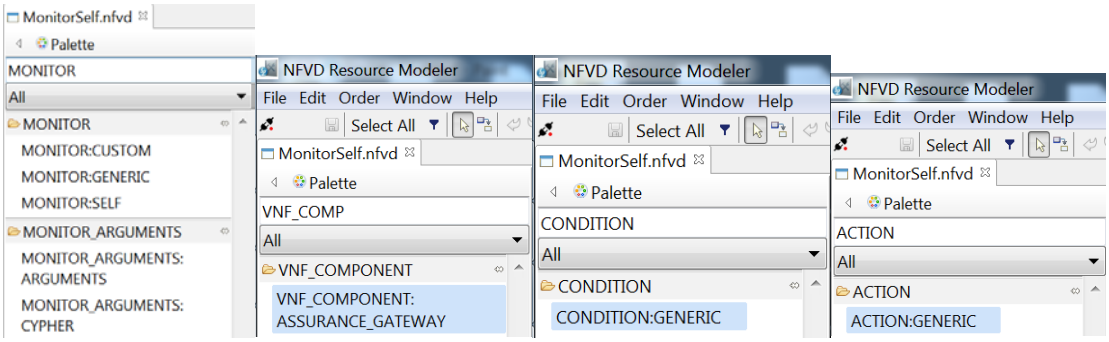


Figure 107: How to select VNFC,Monitor: Custom, Condition, Action, Monitor arguments Example in resource modeler

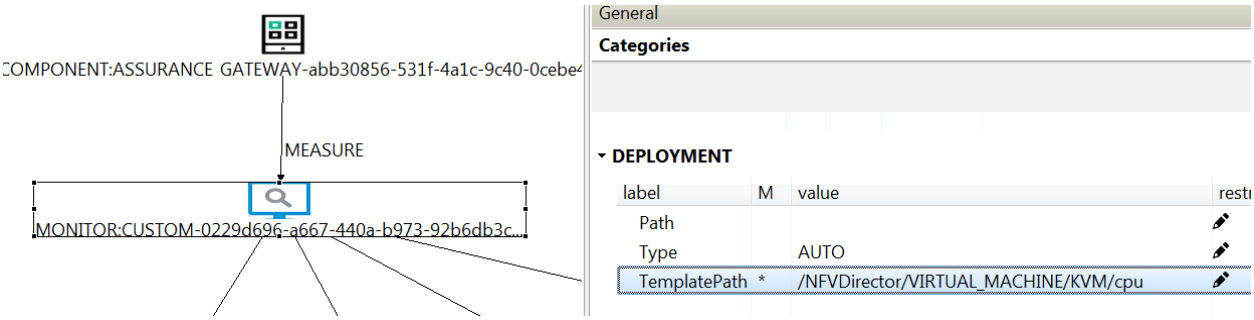


Fig 108 Template path is mandatory for Monitor: Custom

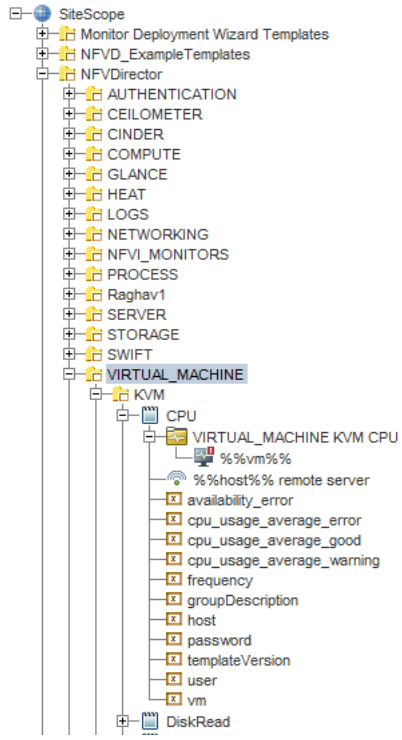


Fig 109 For example, /NFVDirector/VIRTUAL_MACHINE/KVM/cpu is a existing template path from SiteScope.

Under a selected template path , there are variables like host, password, user, vm and so on. We can attach these as Monitor Arguments, (cpu_usage_average_error, groupDescription, templateVersion and all do not need to pass, also, any default value variables do not need to pass.)

IMPORTANT: variable names present in the template should match – monitor argument name and value can be provided (as shown below):

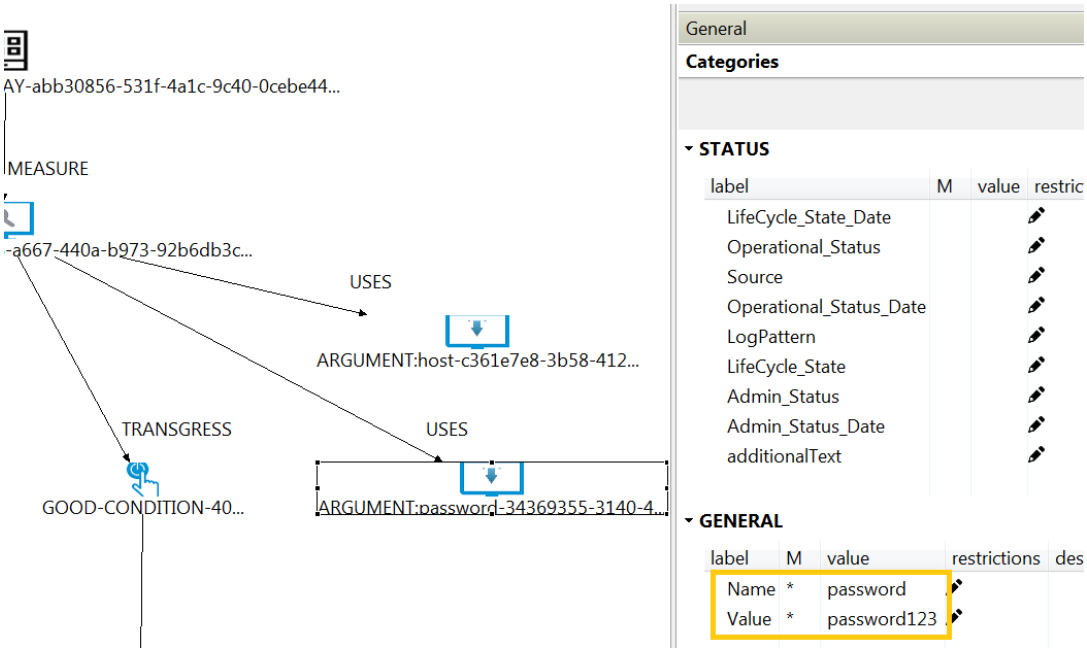


Fig 110 Monitor Argument Name should match the variable names in template path

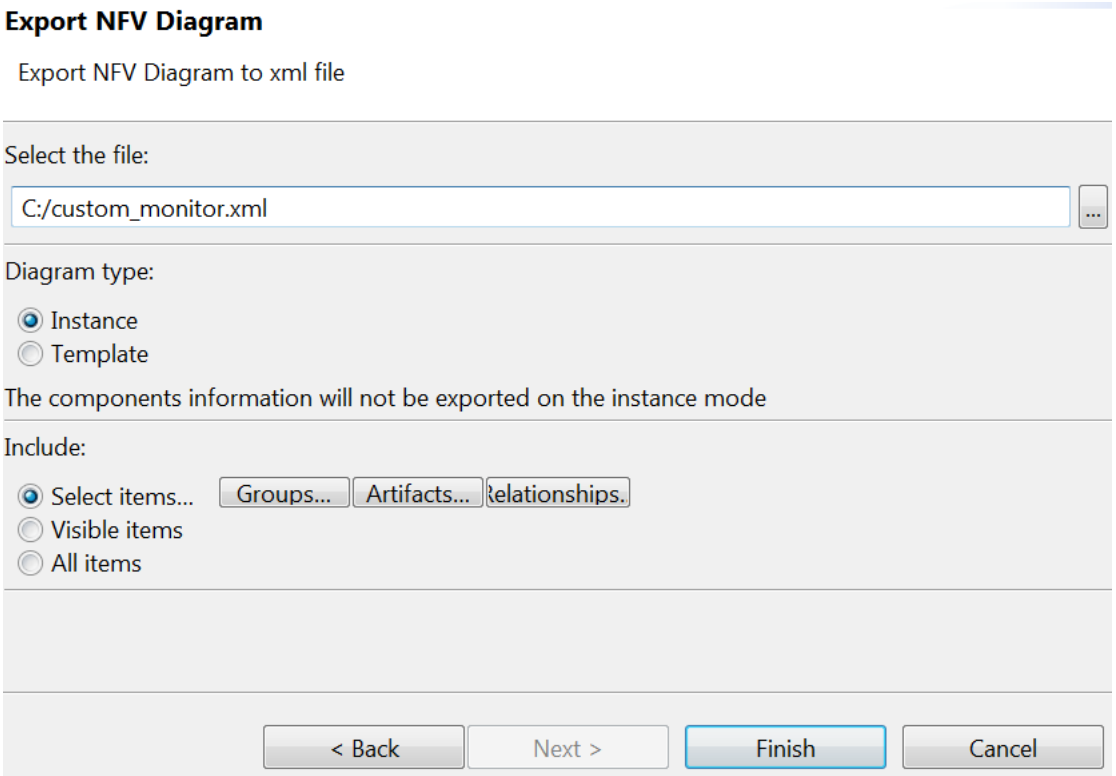


Fig 111 Go to file > export > make sure instance is selected. (Fill all the mandatory variables of each node, otherwise it will not allow the export)

Important:

1. In the above design, VNF Component: Assurance Gateway artifact should not be uploaded.
2. Note down the artifact Id from GUI for VNF_Component instance.

   Example: artifactId of VNF_Component: Assurance Gateway

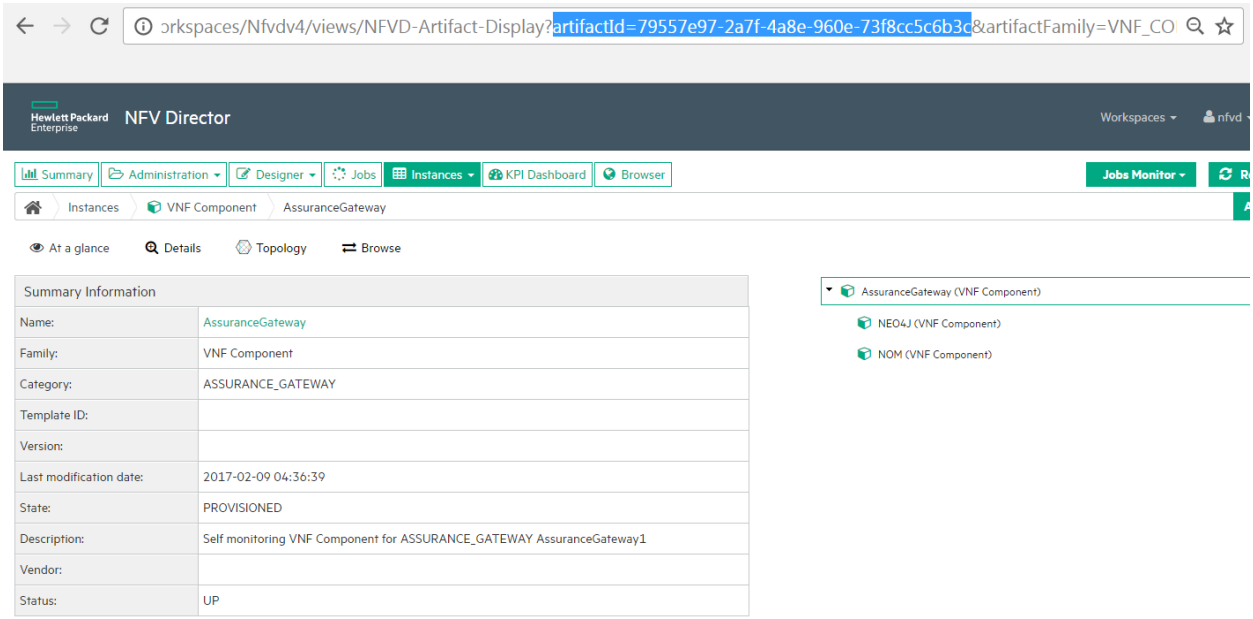


Fig 112 Note Down VNFC artifact Id from the GUI

3. Replace the parent-artifact-id with noted VNFC artifactId.

Example: http://{FF-IP -ADDRESS>:8080/nfvd/instance/relationship

```
<relationship-instances xmlns="http://www.hp.com/nfvd">
        <relationship-instance xmlns="http://www.hp.com/nfvd">
                <relationship-type>MEASURE</relationship-type>
                <parent-artifact-id>{replace noted VNFC artifactid from gui}</parent-artifact-id>
                <child-artifact-id>cf73b9fa-7ff4-4092-bfeb-6e015f4fa1be</child-artifact-id>
                <categories/>
                <status>
                        <label>ENABLED</label>
                        <visible-label>ENABLED</visible-label>
                        <enabled>true</enabled>
                </status>
                <creation-timestamp>2017-01-24T17:42:52.508+0530</creation-timestamp>
                <update-timestamp>2017-01-24T17:42:52.508+0530</update-timestamp>
        </relationship-instance>
</relationship-instances>
```

4. 4. Here monitor: custom needs to be deployed using the following Assurance APIs:

Example:

```
POST
Error! Hyperlink reference not valid.
http://<Assurance-ip-address>:18080/nfvd/instance/artifact/monitor/start/{custom-monitor-artifact-id}
```

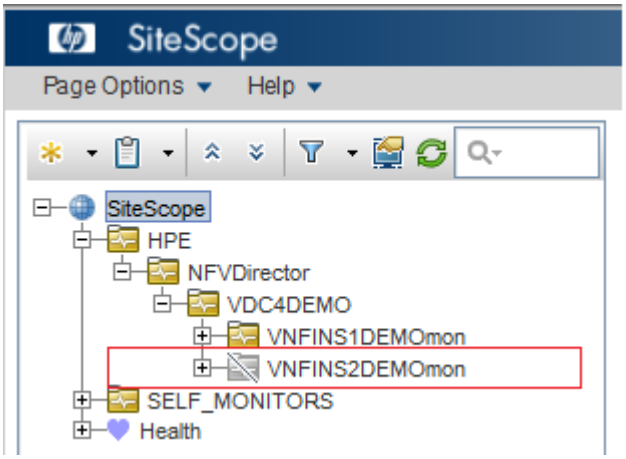
10.6 Monitoring Operations

The following operations are associated with a Monitor:

- Deploy a monitor
- Start a Monitor
- Stop a Monitor
- UnDeploy a Monitor

NOTE: Directory structure associated with the VNF monitors is built using the name of the VNF and Tenant. If the name contains a space or characters other than “A-Za-z0-9_!@#$%^*(){}\\/-“ it will be replaced with “_”.

NOTE: When a VNF is undeployed, the directory structure associated to the VNF monitors are not deleted in SiteScope. These dangling directory structures are harmless, and can be manually deleted from the SiteScope GUI.



10.7 Multi SiteScope support

Monitors can be deployed on more than one SiteScope depending on the resource tree – monitor association.

Multiple `VNFC:SITESCOPE` can be created but only one can have the `IS_DEFAULT` attribute set to `TRUE`. The flag marks the `VNFC:SITESCOPE` as the default SiteScope, which can be used for monitor deployment in case when no `VNFC:SITESCOPE` is attached to the resource tree. For example, `DATACENTER`, etc. If the user has set `IS_DEFAULT` to `TRUE` in multiple `VNFC:SITESCOPE`, then an error will be returned by the assurance gateway because it cannot select a unique `VNFC:SITESCOPE`.

If there are multiple `VNFC:SITESCOPE` created and associated to DATACENTER, then the `VNFC:SITESCOPE` with the highest WEIGHTAGE (attribute in GENERAL category) is used as the VNFC:SITESCOPE for monitor deployment. If there are multiple `VNFC:SITESCOPE` artifacts having same weightage value, then one is picked randomly.

All self monitors will always be deployed on the default SiteScope only. For example, all monitors configured under `VNF:NFVD` will be deployed in the default SiteScope only.

If you need to delete or update a `VNFC:SITESCOPE` instance, then make sure all the monitors on that SiteScope are undeployed before the delete or update operation.

NOTE: A delete/update operation on `VNFC:SITESCOPE` will only perform the DB related operations and will not affect any changes on the monitor.

NOTE: All VNFC:SITESCOPE should be associated to VNF:NFVD, with VNF:NFVD as parent and VNF:SITESCOPE as child, with INLUCDE relationship

10.7.1 SiteScopes associated at different levels

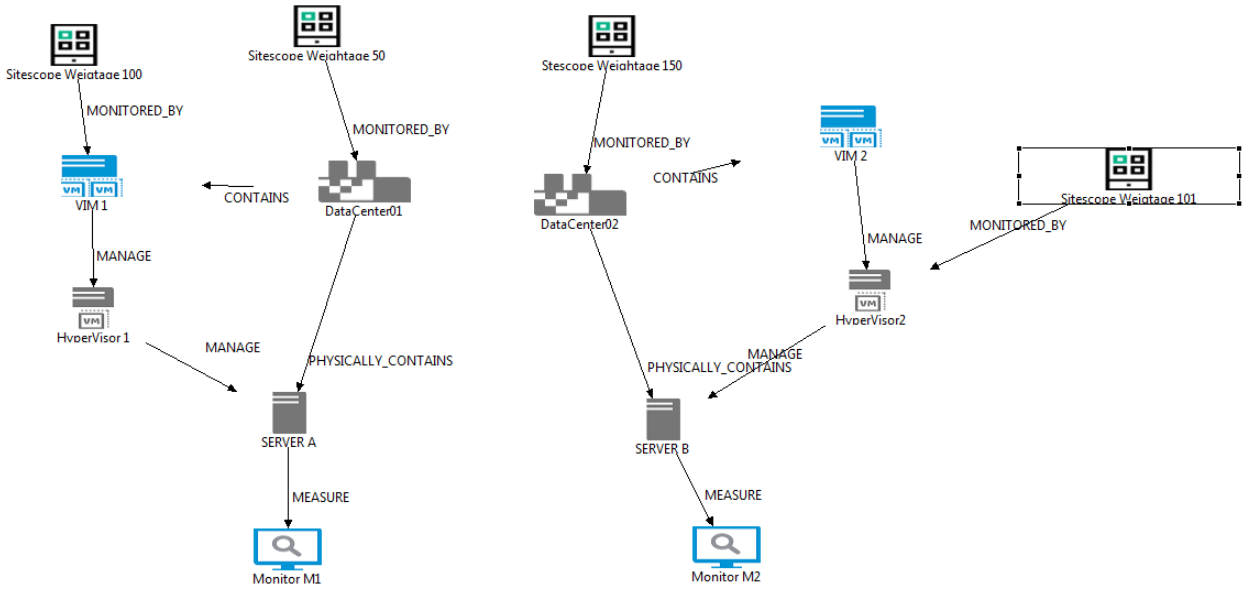


Figure 64: SiteScopes associated at different levels

The deployment in the previous figure indicates 4 SiteScopes associated at different levels:

- DataCenter01 with weightage 50
- DataCenter02 with weightage 150
- VIM1 with weightage 100
- HyperVisor2 weightage 101

In case of Monitor M1, the Assurance Gateway finds two SiteScopes: one connected to DataCenter01 and another to VIM1. The SiteScope connected to VIM1 is chosen because it has the higher weightage, 100.

In case of Monitor M2, the Assurance Gateway finds two SiteScopes: one connected to DataCenter02 and another to HyperVisor2. The SiteScope connected to DataCenter02 is chosen because as it has the higher weightage, 150.

10.7.2 No SiteScope associated

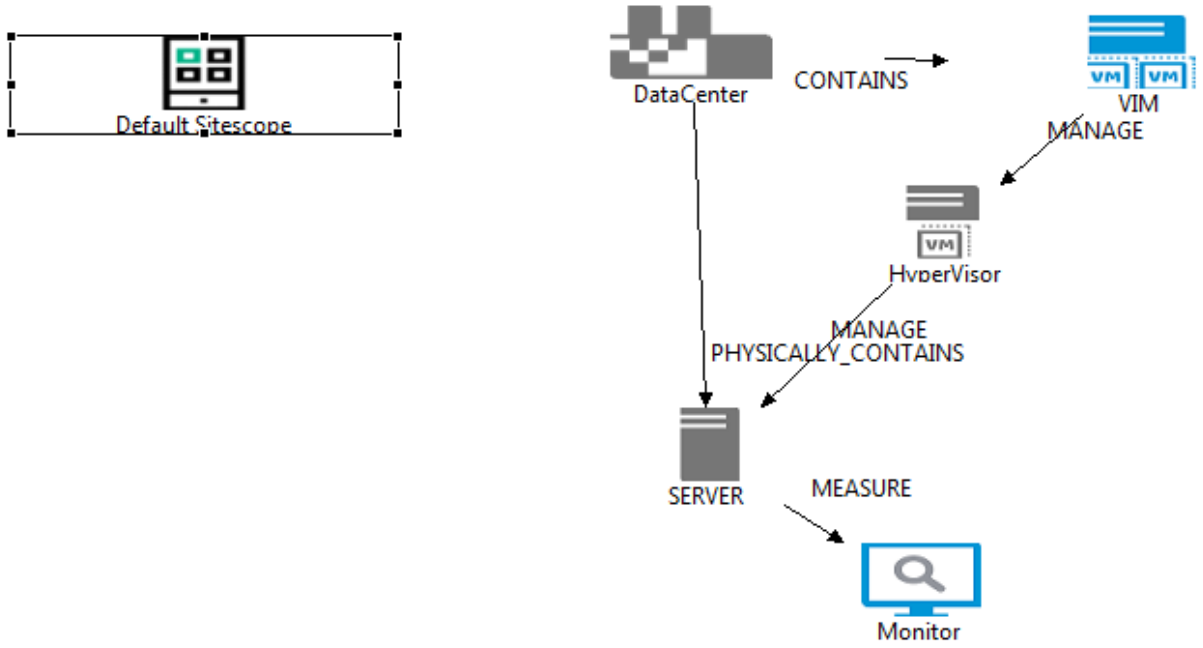


Figure 65: No SiteScope associated

If, as indicated in the figure, there is no SiteScope connected, then AGW will try to deploy the monitor in the default SiteScope. At any time, the NFVD system should not have more than one SiteScope designated as DEFAULT.

10.8 Tagging monitors in SiteScope

`VNFC:SITESCOPE` has a `SiteScopeTag` attribute in the GENERAL category. The default values are `DATACENTER` and `NETWORK_SERVICE`. This feature enables the grouping of monitors associated with a particular DATACENTER/NETWORK_SERVICE in SiteScope.

Example:

There are two datacenters, D1 and D2. Monitor M1 is created under D1 and monitor M2 is created under D2. In SiteScope in MultiView D1 appears in the header D1 and the associated M1 monitor under it.

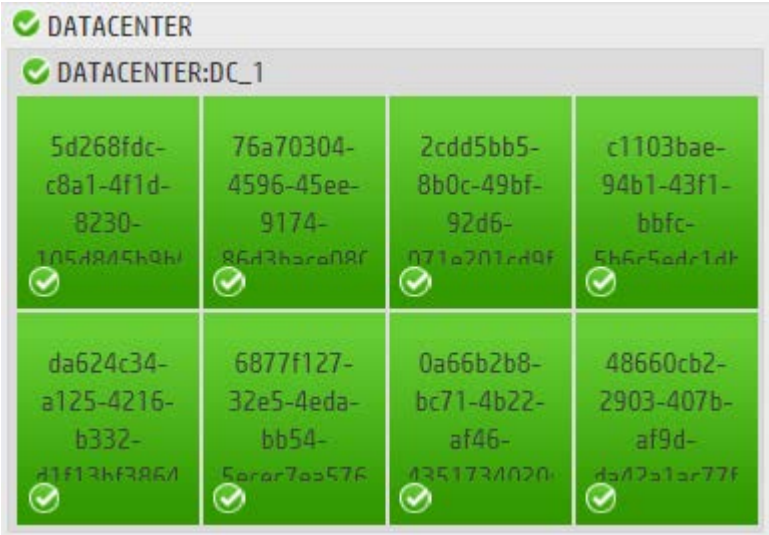


Figure 66: Monitor associated with DataCenter

10.9 NEO4J Index Rebuilding Tool

In cases where indexes in neo4j get corrupted due to issues like disk full or fd limit reached, the user may need to rebuild the indexes. Execute the following tool to rebuild the indexes:

```
/opt/HPE/nfvd/agw/tools/topologyIndexBuilder.sh
```

10.8 State Propagation

If a Virtual machine's status is affected due to the single CPU failure, then its operation status may get affected. Similarly, the operational status of its parent components may also get affected.

Therefore, this automated change of operational status for a calculated hierarchy of components here is considered as state propagation.

In a chain like NS > VNF > VNFC > VM, if there is no propagation rule at any intermediate level, the state does not get propagated further up the chain.

Propagation of status happens based on propagation modes: **LATEST**, **HIGHEST**, or **NONE**.

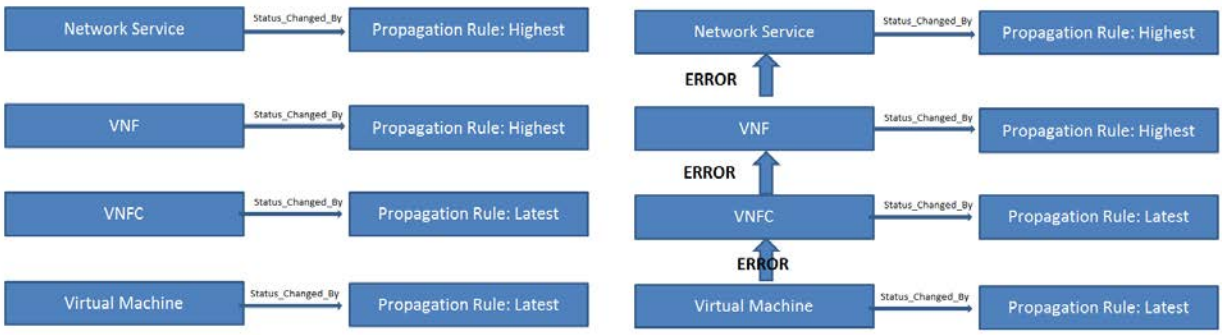


Figure 67: Propagation modes

10.8.3 State propagation functionality

State propagation is a concept of changing and propagating the operational status of a component and its affected parent components, which is triggered by an alarm on a source component.

Example: If a Virtual machine's status is affected due to the single CPU failure, then its operation status may get affected. Similarly, the operational status of its parent components may also get affected. Therefore, this automated change of operational status for a calculated hierarchy of components here is considered as state propagation.

10.8.4 Flow

Status change propagation can be triggered by SiteScope or a VNFM alarm.

Typically, after the completion of state propagation, the user can see the status change in the **Fulfillment UI**, and the same is notified back to the **Assurance gateway** through update notification calls.

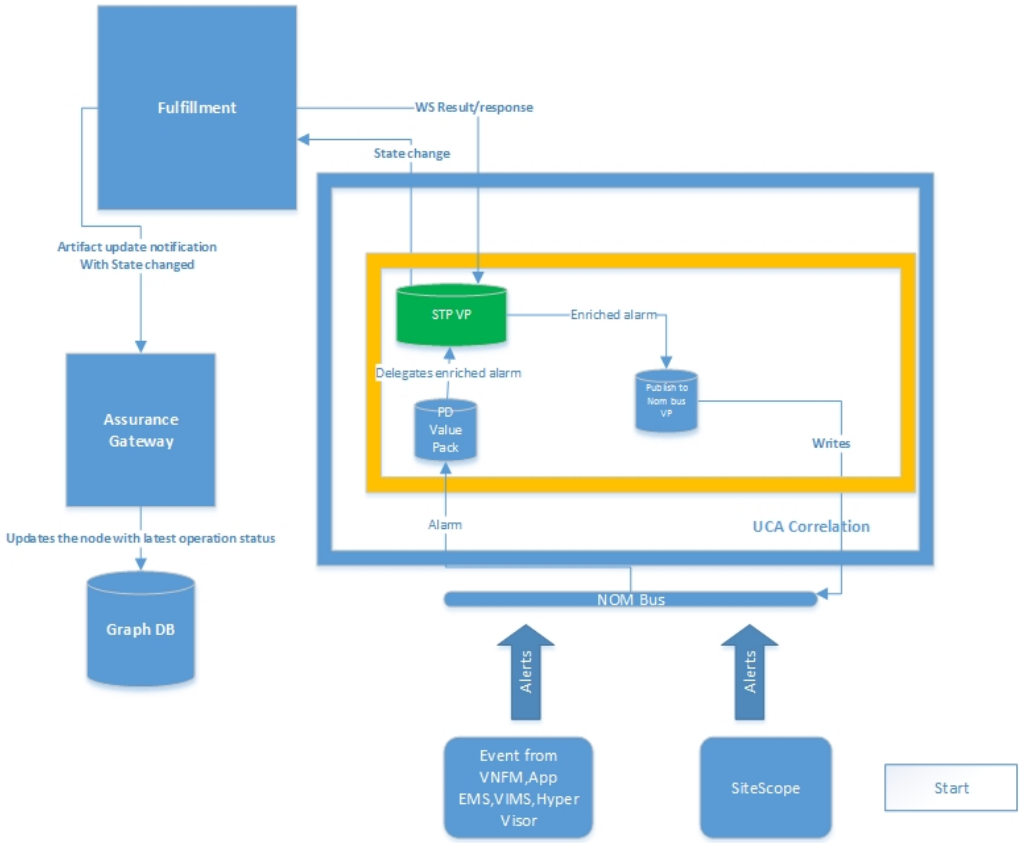


Figure 68: State propagation flow

10.8.5 State propagation process

In NFV Director, there are two alarm sources, but a new alarm source can be dynamically handled if they send the alarm in given X.733 format with few NFVD-related data.

10.8.6 SiteScope alarms

SiteScope is a component of NFVD and it generates alarms when any KPI breach occurs.

It is configured to send `alertseverity` in the X.733 `AdditionalText` field of an alarm.

```
<additionalText>_customPropertiesValues=|_httpPort=8888|_webserverAddress=15.15
4.112.75|alertHelpURL=http://127.0.0.1:18088/SiteScope/sisdocs/doc_lib/index.ht
m?single=false&amp;context=system_avail&amp;topic=config_sis_alert|diagnosticTr
aceRoute=|errorOnly=|goodOnly=
VirtualMachine/37add4bb-80f1-49d6-93eb-
b1203d5eafba/Realtime/cpu/usage.average[]: 0|FullGroupId=
|SiteScopeURL=http://127.0.0.1:18088/SiteScope|SiteScopeuserurl=http://127.0.0.
1:18088/SiteScope/userhtml/SiteScope.html|state=VirtualMachine/37add4bb-80f1-
49d6-93eb-b1203d5eafba/Realtime/cpu/usage.average[]=0%|tag=|targetHost=cms-
vcentre.ind.hp.com|targetIP=15.213.49.32|targetIPVersion=IPV4|templateDeployPat
h=Script
Path/nfvddemo/TEST_2_Server/TEST_2_Frontend/TEST_2_VM_Linux_xsmall/artifactId-
14170989360221/VMWARE VM CPU Monitor|time=4:39 AM
11/28/14|warningOnly=|customerId=&amp;lt;customerId&amp;gt;|alertSeverity=good
```

```
</additionalText>
```

The `alertSeverity` field contains the severity of an alarm, and it can have the following values:

- `ERROR`
- `WARNING`
- `GOOD`

NOTE: A `Good` value in `alertSeverity` means the clearance of an alarm, and it indicates normal operating condition.

10.8.7 Operational status mapping

The user needs to map the `alertSeverity` value of an alarm to a meaningful operational status of their choice, in the `$UCA_EBC_DATA/instances/default/deploy/UCA_NFVD_StatePropagation3.0/conf/alarmmapping.property` file.

10.8.7.6 Sample operational state mapping (Out of the box)

```
#Maintain the order or severity for alarms, lowest first.
INTERMDEDIATE=
good=normal operation
warning=degraded operation:Warning
MINOR=degraded operation:Minor
MAJOR=degraded operation:Major
CRITICAL=degraded operation:Critical
error=degraded operation
CLEAR=normal operation
WARNING=degraded operation:WARNING
```

10.8.7.7 Alarm severity level

The order of severity is read from the property file. Alarm severity is listed from lowest severity first.

This will be used when there is a need to compare the severity of operational status from its previous status.

10.8.7.8 Operational status calculation

The concept of state propagation is simple: for a given component's `alertSeverity`, the corresponding mapped `Operational status` will be read from the property file and the same will be set and propagated up the hierarchy.

10.8.8 State propagation model changes

The changes in artifact definitions, instance, and relationships are the following:

10.8.8.6 Artifact definitions

All the components have a category called STATUS that is displayed as follows. The STATUS category has three key attributes related to State propagation.

- `Operational_Status` indicates the current operational status of that component. There is no default value for this field. Its values will be derived from the `alarmmapping.property` file.
- `Operational_Status_Date` indicates the date and time (in UTC format) when the operational status was changed, driven by alarm raised time.
- `Source` denotes if the state propagation was triggered by NFVD (internally) or by some other external entity, for example, VNFM.

10.8.8.7 PROPAGATION_RULE artifact

This new artifact is introduced to further decide if operational status of components needs to be changed, and if yes, how to change that.

10.8.8.8 Key Attributes

- `Name`: Any user text to distinguish the rule.
- `Propagation_mode` decides how the `operational_status` for a given component must be calculated. This can have the following values:
  - `LATEST`: In this mode, as soon as the alarm is received by the STP value pack, it will fetch the corresponding operational status from the `alarmmapping.property` file for a given `alertSeverity,` and will set and propagate the same status without any comparison or calculation.
  - `HIGHEST`: In this mode, there will be a comparison of the previous operational status with the current operational status for a specific component and the highest severity will be set and propagated.
  - `RULE_BASED`: In this mode, the operational status will be calculated based on user defined rules. The alarm will be delegated to the rule-based value pack.
  - `NONE`: In this mode, state propagation will not take place.
- `Propagation_rule`: An optional attribute, if the propagation mode is `RULE_BASED`, then the user needs to specify the rule value pack name.

10.8.8.9 Relationship definitions

When you wish to enable state propagation for any component, starting from any level in the hierarchy, you must define a relationship between that specific component and the `PROPAGATION_RULE` artifact with relationship type set as `STATUS_CHANGED_BY`.

10.8.8.10 Configurations

The state propagation value pack requires the details of the assurance database and fulfillment web service endpoint. You can configure the same, as explained below, and redeploy the value pack.

NFVD database and Fulfillment configurations

The following parameters must be set in the `/var/opt/UCA-EBC/instances/default/deploy/UCA_NFVD_StatePropagation-3.0/conf/statepropagation.property` file.

```
#The URL for fulfilment for state propagation
FULFILLMENT_URL=http://localhost:8071/ngws/service?wsdl
#The URL for NFVD database
NFVD_DB_URL=http://localhost:17373/db/data
```

```
#Set if alarm after STP needs to be published to NOM Bus. value true/false
PUBLISH_TO_NOM=true
FULFILLMENT_REST_URL=http://localhost:8080
USE_SOSA_API=false
```

FULFILLMENT_URL

This is the URL of fulfillment where the status of all the affected components will be changed and propagated.

NFVD_DB_URL

This is the URL of the NFVD database from where all related parents will be fetched.

PUBLISH_TO_NOM

If this flag is set to true, once the states are propagated, the enriched alarm will be published to the OM bus. Any other value will not publish the alarm.

Parent hierarchy configuration

The following XML configurations in the `$UCA_EBC_DATA/instances/default/deploy/UCA_NFVD_StatePropagation-4.0/conf/CypherQueryData.xml` file define the available parent-child relationship as per the current artifact definition. If a new parent or child type is introduced, then it can be configured here to propagate the states to the new parent or child type.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This xml describes what are the possible parent family type for any
        child family type -->
<CypherQueryData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="CypherQueryData.xsd">
        <ChildParentRel>
                <childNode>
                        <childFamilyType>VIRTUAL_MACHINE</childFamilyType>
                        <parentNode>

<parentFamilyType>VNF_COMPONENT</parentFamilyType>
                        </parentNode>
                </childNode>

                <childNode>
                        <childFamilyType>VNF_COMPONENT</childFamilyType>
                        <parentNode>

<parentFamilyType>VNF_COMPONENT</parentFamilyType>
                        </parentNode>
                        <parentNode>
                                <parentFamilyType>VNF</parentFamilyType>
                        </parentNode>
                </childNode>

                <childNode>
                        <childFamilyType>VNF</childFamilyType>
                        <parentNode>
                                <parentFamilyType>VNF</parentFamilyType>
                        </parentNode>
                        <parentNode>
```

```
<parentFamilyType>NETWORK_SERVICE</parentFamilyType>
                        </parentNode>
                </childNode>

                <childNode>
                        <childFamilyType>NETWORK_SERVICE</childFamilyType>
                        <parentNode>

<parentFamilyType>NETWORK_SERVICE</parentFamilyType>
                        </parentNode>
                </childNode>
        </ChildParentRel>

        <whereArtifactIdClause> where has(n.artifactId) and
n.artifactId=</whereArtifactIdClause>
        <whereArtifactIdInClause> where has(n.artifactId) and n.artifactId IN
</whereArtifactIdInClause>
        <whereFamilyClause>where has(m.artifactFamily) and
m.artifactFamily=</whereFamilyClause>
        <whereNameClasue>where has(n.`GENERAL.Name`) and
n.`GENERAL.Name`=</whereNameClasue>
        <startCypher>START n = node(</startCypher>
        <startCypherAllNode>START n = node(*) </startCypherAllNode>
        <matchCypher>) match </matchCypher>
        <nTomIncomingRel> (n) &lt;-- (m) </nTomIncomingRel>
        <statusChangeRel> (n)-[:STATUS_CHANGED_BY]->(m) </statusChangeRel>
        <returnM> RETURN m</returnM>
        <returnN> RETURN n</returnN>
        <singleQuote>'</singleQuote>
</CypherQueryData>
```

Example: If you have a use case where the status needs to be propagated to the parent of Network service, you just need to add one more `parentNode` tags with a given parent family type in the `NETWORK_SERVICE childNode` tag.

10.8 Integrating with external VNFM alarms

The source of alarm for state propagation can be VNFM. The alarms coming from VMFM have different attributes and those alarms already contain old and new states.

Therefore, the mapping from severity of alarm to actual operational state will not be required, but the severity level of operational state must be maintained in the `/var/opt/UCA-EBC/instances/default /deploy/UCA_NFVD_StatePropagation-4.0/conf/operationastatuslist.property` file.

VNFM alarm will be received by the **Assurance gateway** as an update artifact notification and then will be sent to the OM bus.

```
#Maintain the order or severity for operational status, in ascending order with
lowest first
power-on
power-down
degraded_operation
normal_operation
```

10.8.3 Key attributes in VNFM alarms

Table 21: Key attributes in VNFM alarms

Attribute	Description
`SourceArtifactId`	This will contain the artifact ID of the alarm originating node.
`alarmName`	This will have a value of `OperationalStatusChange`, indicating that the alarm is an operational state change alarm.
`newState`	This will contain the actual operational state of the component that needs to be propagated.
`oldState`	This will contain an old operational state of the component.

10.10 NFV Director Notification Interface

NFVD provides a VNF Lifecycle Changes Notification interface, Os-Nfvo, which provides runtime notifications related to the state of the VNF instance, as a result of changes made to the VNF instance including (but not limited to) changes in the number of VDUs, changing VNF configuration and/or topology due to auto-scaling/update/upgrade/termination, switching to/from standby, and so on.

Topology changes are published as notifications to the JMS topic named, `com.hp.openmediation.topologyNotifications` hosted on OpenMediation. Any external system like OSS can consume the alarms from the topic.

In such cases, NFVD publishes a life cycle change notification to the JMS topic named

```
com.hp.openmediation.topologyNotifications
```

The body of the notification is a string containing an XML document according to the `http://hp.com/openmediation/topologyNotifications/2011/06` schema.

Failure alarms are published on the following JMS topic:

`com.hp.openmediation.alarms`,

The body of the notification is a string that contains a well-formed and valid XML document, according to the `http://hp.com/openmediation/alarms/2011/08` schema.

All of these can be consumed by any integrating systems.

The consumer must use JMS connection and connect to the broker running on the assurance host and default port 10000. If the consumer is interested only in specific messages, then the consumer may use the JMS Message Selector to specify the messages of interest. All the custom field values are available as JMS properties.

Example:

Set `alarmName= LifeCycleStateChangeNotifiction` to select only lifecycle notifications.

Assurance-generated alarms are configured to publish to a specific URL and topic. These are available in the `VNFC:OPENMEDIATION` self-management artifact Instance.

For Operational Status notification, `oldState` and `newState` represent the old and new operational statuses.

The mandatory fields are `alarmName` set to `LifeCycleStateChangeNotification` and `artifactId` set to the value of the artifact ID of the originated managed object. The alarms will include topology information, which can be used by OSS. A channel adapter may be required to transform the VNFM traps/alarms to the required X.733 format.

For more information about the alarm creation notification schema, refer to the *OSS Open Mediation Functional Specification Guide* from Open Mediation product.

For more information about VNF Lifecycle Changes Notification, refer to *NFV-MAN001v062*, Sections 7.2.4 and 7.2.5.

> Note: Any external alarms can also be fed into NFV Director, for example, from any VNFM that can send traps or alarms. These external alarms must be collected by open mediation and converted into X.733 format using a custom-developed channel adapter.

Table 22: NFVD alarm fields

Alarm field	Description
`Identifier`	Unique alarm ID.
`sourceIdentifier`	Source filter recognized by the UCA EBC.
`alarmRaisedTime`	Time when the alarm was triggered.
`originatingManagedEntity`	Originating entity for create/delete/scale operations. Format: `<ARTIFACT_FAMILY> <ARTIFACT_ID>` Example: `VIRTUAL_MACHINE 123456798`

`originatingManagedEntityStructure`	Class: `ARTIFACT_FAMILY` Instance: `ARTIFACT_ID`
`alarmType`	Mandatory field according to OM schema requirements. Set to `UNKNOWN` for lifecycle events.
`probableCause`	Mandatory field according to OM schema requirements. Set to `UNKNOWN`.
`perceivedSeverity`	Mandatory field. Added as indeterminate because alarm will be a lifecycle alarm.
`networkState`	Mandatory field according to OM schema requirements. Status of the alarm with respect to the problem raised by the alarm.
`operatorState`	Mandatory field according to OM schema requirements. Status of the alarm with respect to the operator.
`problemState`	Mandatory field according to OM schema requirements. Status of the alarm with respect to the associated ticket.
`customField:alarmName`	Name of the alarm. Operational Status Alarm: `OperationalStatusChangeNotification` Lifecycle State Alarm: `LifeCycleStateChangeNotification`
`customField:oldState`	Old lifecycle state in case of lifecycle alarms. Old status in case of operational status alarms.
`customField:newState`	New lifecycle state in case of lifecycle alarms. New operational status in case of operational status change alarm.
`customField:serverHostname`	Server hostname.
`customField:vmHostName`	Virtual machine hostname.
`customField:hypervisorID`	Hypervisor identifier. Either UUID or any identifier to identify the Hypervisor for the specific virtual machine.
`customField:SourceArtifactId`	Source artifact ID of the originating alarm.
`customField:vmName`	Virtual machine name.
`customField:source`	Alarm generation source. • `Internal`: In case of state propagation alarm. • `AGW`: In case of lifecycle and operational status alarm generated within the assurance gateway. • `External`: In case of notification from VNFM.
`customField:NOMType`	Variable, which can be used as a selector for fetching alarms. Currently UCA-EBC CA requires NOMType.
`customField:alarmSubtype`	Indicates the process that generated the alarm. Alarms are generated either during artifact create/update/delete process, or during relationship create/delete process. • `ArtifactAlarm` in case of artifact alarm.

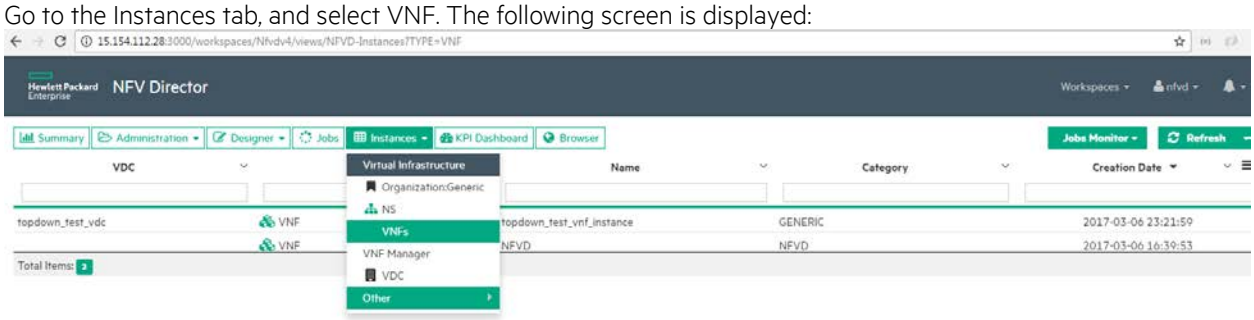
	• `RelationAlarm` in case of relationship changes alarm.
`customField:NFVTopology`	Topology information.
`additionalText`	Notification event details received from VNFM.

10.10.1 Life Cycle Management (LCM) Notifications

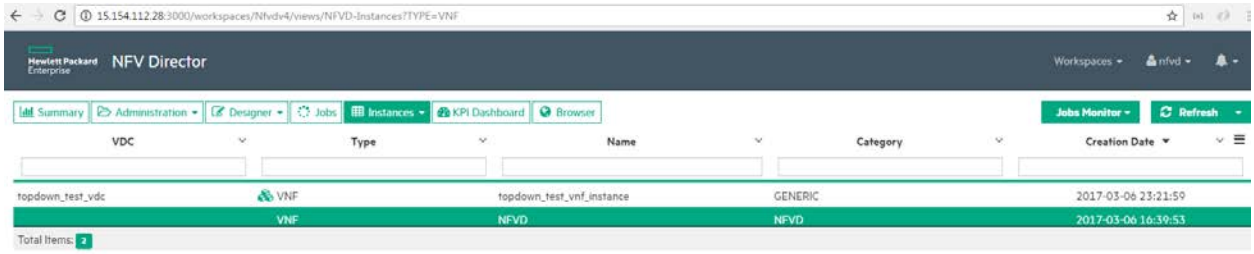
The topic and the JMS connection details are present in VNF COMPONENT:OPEN_MEDIATION, as shown below in the NFVD GUI.

Steps:

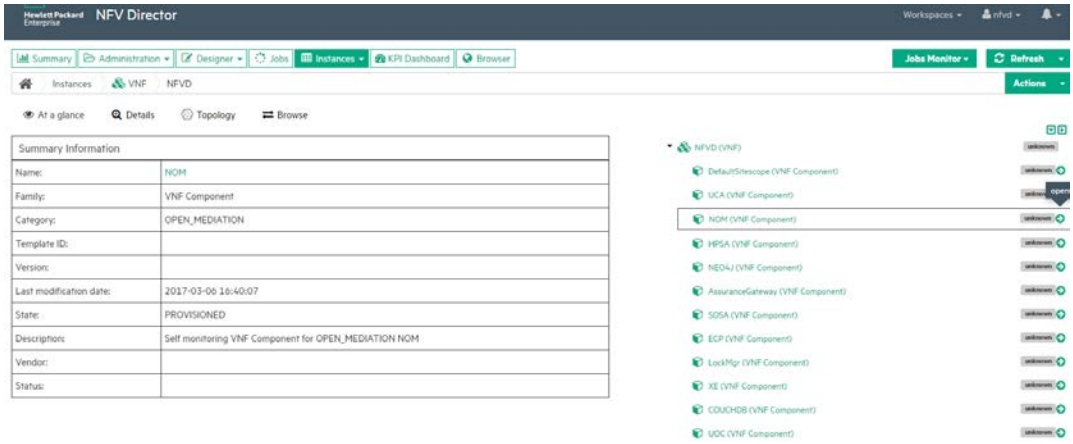
1. Login to NFVD GUI, as domain user.
2. Go to the Instances tab, and select VNF. The following screen is displayed:



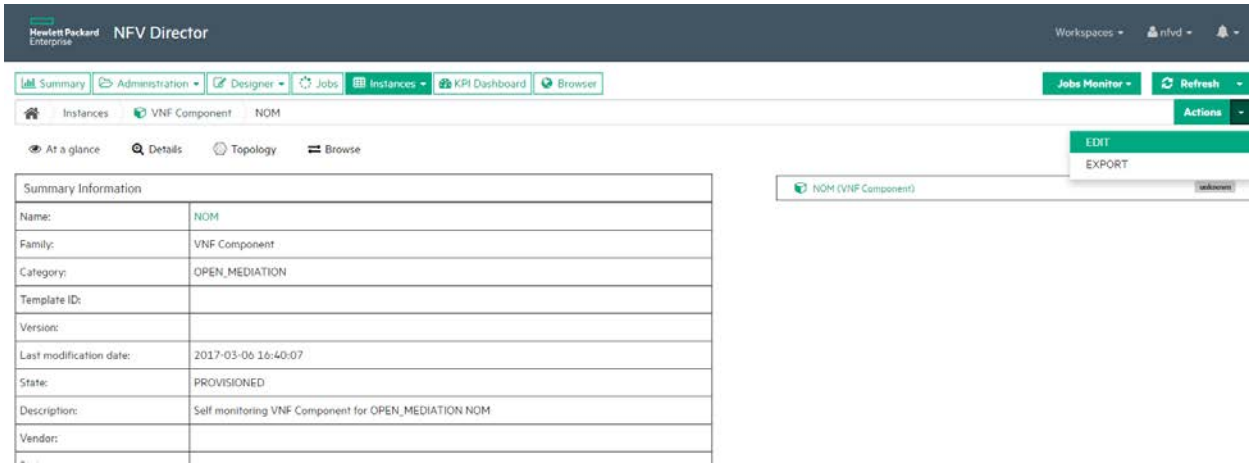
3. Select the VNF with the category “NFVD”, as shown below:



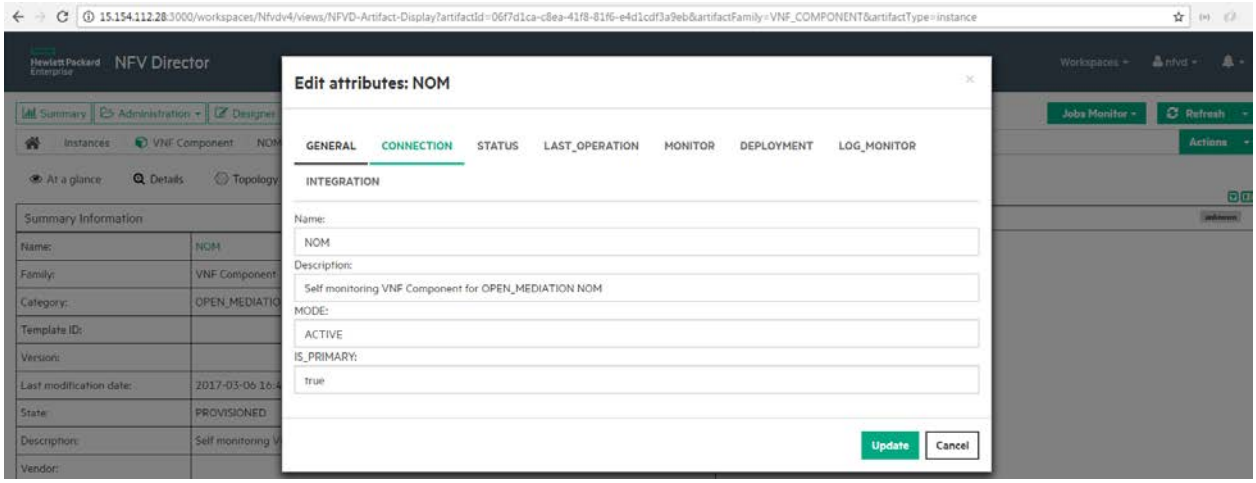
4. Click the VNF with category NFVD, and the following screen is displayed. Select NOM(VNF_COMPONENT) on the right side, as shown below. Click open.



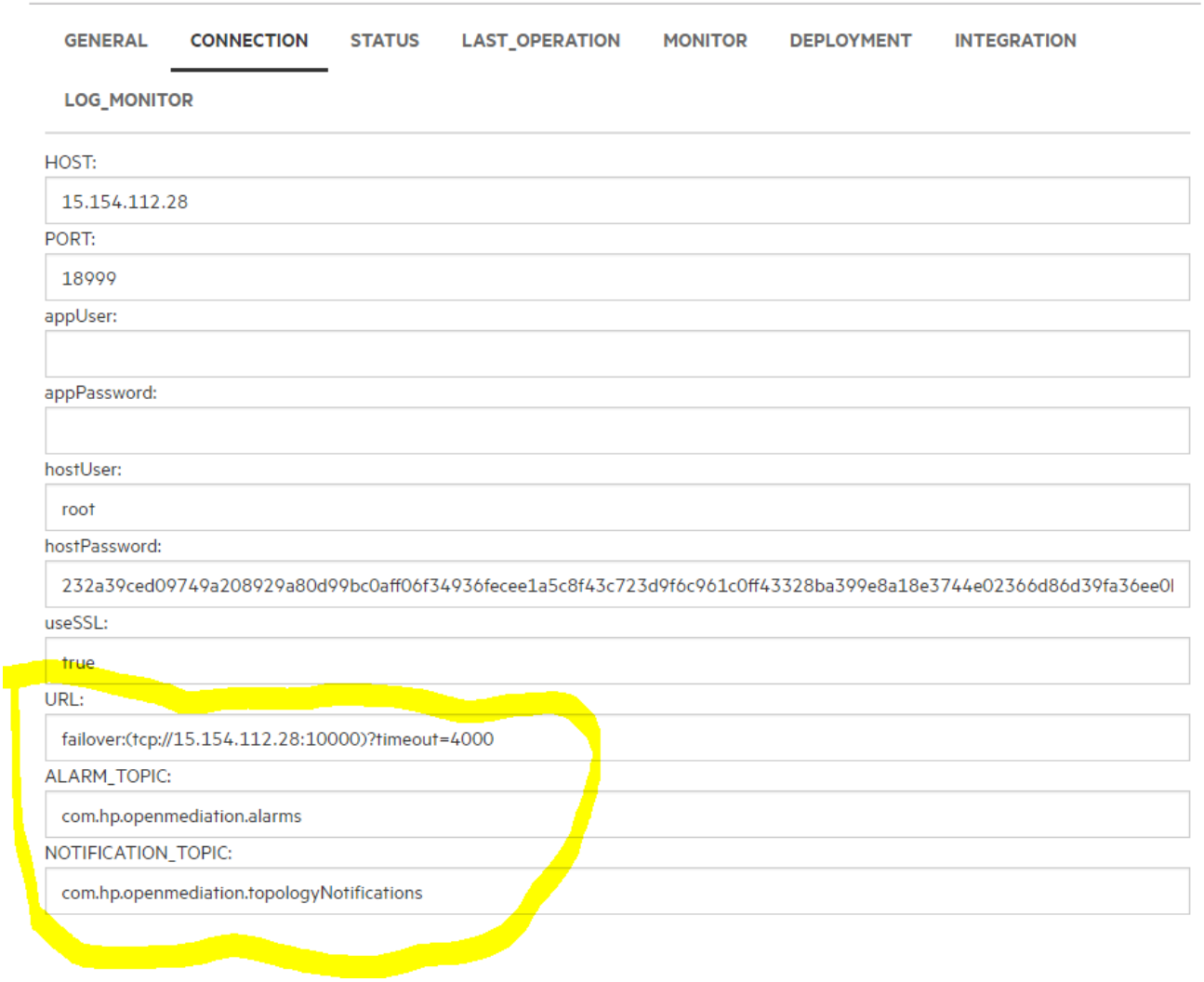
5. The following screen is displayed. Click EDIT option in the ACTION tab.



6. The following screen is displayed. Select the CONNECTION category, as shown in the image.



7. The following screen is displayed:



Notifications can be filtered using the suitable JMS selector. These notifications can be filtered based on the following selectors:

- artifactCategory
- artifactType
- NOMOriginalProvider
- OPERATION_TYPE
- artifactId
- NOMType
- NOMOriginalProviderHost
- sourceIdentifier
- artifactFamily

Here, OPERATION_TYPE can take the following values:

0 – For Create Notification

1- For Update Notification
2- For Delete Notification

3- For Operation State Notification

For more information about the topology notification schema, refer to the *OSS Open Mediation Functional Specification Guide* from Open Mediation product.

Sample Notifications are listed below:

LCM Creation Notification:

a) LCM Create Artifact:

Selectors are as below

{artifactCategory=OPENSTACK, artifactType=artifact, NOMOriginalProvider=NFVD-AGW, OPERATION_TYPE=0, artifactId=**035ea2b4-a9b1-383c-8f62-84f0cc815d3c**, NOMType=http://hp.com/openmediation/alarms/2011/08, NOMOriginalProviderHost=nfvdvm49.in.rdlabs.hpecorp.net, externalId= **network-eb447c2c-22bc-4444-a288-28ec2a817ba6**, sourceIdentifier=NFVD-AGW, artifactFamily=NETWORK}


```
<Notifications xmlns="http://hp.com/openmediation/topologyNotifications/2011/06">
	<notification>
		<objectCreation>
			<managedObject>
				<classInstance className="NETWORK:OPENSTACK"
instanceName="VIRATNetwork"/>
			</managedObject>
			<eventTime>2017-03-02T11:51:36.213+05:30</eventTime>
			<objectInfo>
				<artifactId>035ea2b4-a9b1-383c-8f62-84f0cc815d3c</artifactId>
				<externalId>network-eb447c2c-22bc-4444-a288-28ec2a817ba6</externalId>
				<sourceIndicator>NFVD-AGW</sourceIndicator>
				<attributeList>
					<attributeIdentifierValue>
						<identifier>STATUS.LifeCycle_State_Date</identifier>
						<value>NA</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>artifactCategory</identifier>
						<value>OPENSTACK</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>LAST_OPERATION.Result_code</identifier>
						<value/>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>artifactStatus</identifier>
						<value>ACTIVE</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>STATUS.Operational_Status.TYPE</identifier>
						<value>TEXT</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>STATUS.Admin_Status_Date.ORDER</identifier>
						<value>8</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
```

```
                <identifier>artifactSubtype</identifier>
                <value/>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>artifactStatusEnabled</identifier>
                <value>true</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>PROVIDER.network_type</identifier>
                <value>vxlan</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>STATUS.Operational_Status_Date.UNIT</identifier>
                <value>Date</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>STATUS.Operational_Status</identifier>
                <value>NA</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>STATUS.additionalText.UNIT</identifier>
                <value>TEXT</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>LAST_OPERATION.Result_description</identifier>
                <value/>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>artifactVersion</identifier>
                <value/>
            </attributeIdentifierValue>
            <attributeIdentifierValue>

<identifier>STATUS.Admin_Status_Date.MANDATORY</identifier>
                <value>false</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>STATUS.Operational_Status_Date.ORDER</identifier>
                <value>2</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>CREATION_MODE.Connect</identifier>
                <value>false</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>STATUS.Admin_Status_Date.TYPE</identifier>
                <value>Date</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>STATUS.Admin_Status_Date.UNIT</identifier>
                <value>Date</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>GENERAL.Description</identifier>
                <value/>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>INSTANTIATE.Shared</identifier>
                <value>false</value>
```

```
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>STATUS.Source.MANDATORY</identifier>
                        <value>false</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>

<identifier>STATUS.Operational_Status_Date.MANDATORY</identifier>
                        <value>false</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>artifactPhysical</identifier>
                        <value>false</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>GENERAL.Name</identifier>
                        <value>VIRATNetwork</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>STATUS.Admin_Status.UNIT</identifier>
                        <value>TEXT</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>GENERAL.net_id</identifier>
                        <value/>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>STATUS.Source.TYPE</identifier>
                        <value>TEXT</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>STATUS.Source.UNIT</identifier>
                        <value>TEXT</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>INSTANTIATE.Admin_state</identifier>
                        <value>true</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>

<identifier>STATUS.LifeCycle_State_Date.MANDATORY</identifier>
                        <value>false</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>STATUS.Operational_Status_Date</identifier>
                        <value>NA</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>GENERAL.Type</identifier>
                        <value/>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>STATUS.LifeCycle_State_Date.UNIT</identifier>
                        <value>Date</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>lastUpdateTimestamp</identifier>
                        <value>2017-03-02T11:51:31.041+0530</value>
                    </attributeIdentifierValue>
```

```
<attributeIdentifierValue>
        <identifier>STATUS.Operational_Status_Date.TYPE</identifier>
        <value>Date</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>LAST_OPERATION.Operation</identifier>
        <value/>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>creationTimestamp</identifier>
        <value>2017-03-02T11:51:31.035+0530</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>INSTANTIATE.ID</identifier>
        <value>eb447c2c-22bc-4444-a288-28ec2a817ba6</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.Admin_Status.ORDER</identifier>
        <value>7</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.Operational_Status.ORDER</identifier>
        <value>1</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>artifactFamily</identifier>
        <value>NETWORK</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>artifactGroup</identifier>
        <value/>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>artifactVisibleStatus</identifier>
        <value>ACTIVE</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>INTEGRATION.ExternalId</identifier>
        <value>network-eb447c2c-22bc-4444-a288-
28ec2a817ba6</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>artifactType</identifier>
        <value/>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>ASSURANCE_KPI.Severity</identifier>
        <value>NA</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.Source.ORDER</identifier>
        <value>3</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.Operational_Status.UNIT</identifier>
        <value>TEXT</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.LifeCycle_State.UNIT</identifier>
```

```
        <value>TEXT</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.LifeCycle_State</identifier>
        <value>NA</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.additionalText.MANDATORY</identifier>
        <value>false</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.Admin_Status</identifier>
        <value>NA</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.Operational_Status.MANDATORY</identifier>
        <value>false</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>ASSURANCE_KPI.VALUE</identifier>
        <value>NA</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.Source</identifier>
        <value>NA</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.LifeCycle_State.ORDER</identifier>
        <value>4</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.LifeCycle_State.TYPE</identifier>
        <value>TEXT</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>ASSURANCE_KPI.UNIT</identifier>
        <value>NA</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.Admin_Status_Date</identifier>
        <value>NA</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.additionalText</identifier>
        <value>NA</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>ASSURANCE_KPI.LABEL</identifier>
        <value>NA</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>ASSURANCE_STATUS.Severity</identifier>
        <value>NA</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.LifeCycle_State_Date.ORDER</identifier>
        <value>5</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
```

```
                    <identifier>STATUS.LifeCycle_State_Date.TYPE</identifier>
                    <value>Date</value>
                </attributeIdentifierValue>
                <attributeIdentifierValue>
                    <identifier>INSTANTIATE.External</identifier>
                    <value>false</value>
                </attributeIdentifierValue>
                <attributeIdentifierValue>
                    <identifier>PROVIDER.physical_network</identifier>
                    <value/>
                </attributeIdentifierValue>
                <attributeIdentifierValue>
                    <identifier>ASSURANCE_KPI.DESCRIPTION</identifier>
                    <value>NA</value>
                </attributeIdentifierValue>
                <attributeIdentifierValue>
                    <identifier>ASSURANCE_KPI.COUNTERNAME</identifier>
                    <value>NA</value>
                </attributeIdentifierValue>
                <attributeIdentifierValue>
                    <identifier>ASSURANCE_OPERATION</identifier>
                    <value>create</value>
                </attributeIdentifierValue>
                <attributeIdentifierValue>
                    <identifier>STATUS.additionalText.TYPE</identifier>
                    <value>TEXT</value>
                </attributeIdentifierValue>
                <attributeIdentifierValue>
                    <identifier>artifactId</identifier>
                    <value>035ea2b4-a9b1-383c-8f62-84f0cc815d3c</value>
                </attributeIdentifierValue>
                <attributeIdentifierValue>
                    <identifier>PROVIDER.segmentation_id</identifier>
                    <value>1074</value>
                </attributeIdentifierValue>
                <attributeIdentifierValue>
                    <identifier>templateId</identifier>
                    <value/>
                </attributeIdentifierValue>
                <attributeIdentifierValue>
                    <identifier>STATUS.LifeCycle_State.MANDATORY</identifier>
                    <value>false</value>
                </attributeIdentifierValue>
                <attributeIdentifierValue>
                    <identifier>STATUS.additionalText.ORDER</identifier>
                    <value>6</value>
                </attributeIdentifierValue>
                <attributeIdentifierValue>
                    <identifier>STATUS.Admin_Status.MANDATORY</identifier>
                    <value>false</value>
                </attributeIdentifierValue>
                <attributeIdentifierValue>
                    <identifier>STATUS.Admin_Status.TYPE</identifier>
                    <value>TEXT</value>
                </attributeIdentifierValue>
            </attributeList>
        </objectInfo>
    </objectCreation>
</notification>
```

```
</Notifications>
```


b) LCM Create Relationship:

Selectors are as below:

{parentArtifactCategory= OPENSTACK, artifactType=Relationship, OPERATION_TYPE=0, childArtifactFamily= FLAVOR, NOMOriginalProviderHost=nfvdvm49.in.rdlabs.hpecorp.net, childArtifactId=05a61e03-bfa0-3be1-b2b4-4449cbbf985f, sourceIdentifier=NFVD-AGW, NOMOriginalProvider=NFVD-AGW, parentArtifactFamily=TENANT, childArtifactCategory=OPENSTACK, NOMType=http://hp.com/openmediation/alarms/2011/08, childExternalId= flavor-41c59e49-8d8a-46ec-87e5-3f9165fb5b90-86f7ec06-4262-49d9-bbd9-50fcd368ad2e, parentArtifactId=996bb7e1-0a95-390d-9272-f9465e45a42c, parentExternalId= tenant-04828f9f9a8049bdab4330ebb63fe4e8}


```
<Notifications xmlns="http://hp.com/openmediation/topologyNotifications/2011/06">
	<notification>
		<objectCreation>
			<managedObject>
				<classInstance className="RELATIONSHIP:HAS"/>
				<classInstance className="TENANT:OPENSTACK" instanceName="admin"/>
				<classInstance className="FLAVOR:OPENSTACK"
instanceName="SANDEEP40flavor"/>
			</managedObject>
			<eventTime>2017-03-02T11:51:36.082+05:30</eventTime>
			<objectInfo>
				<sourceIndicator>NFVD-AGW</sourceIndicator>
				<attributeList>
					<attributeIdentifierValue>
						<identifier>childArtifactId</identifier>
						<value>05a61e03-bfa0-3be1-b2b4-4449cbbf985f</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>parentArtifactId</identifier>
						<value>996bb7e1-0a95-390d-9272-f9465e45a42c</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>childExternalId</identifier>
						<value>flavor-41c59e49-8d8a-46ec-87e5-3f9165fb5b90-86f7ec06-
4262-49d9-bbd9-50fcd368ad2e</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>parentExternalId</identifier>
						<value>tenant-04828f9f9a8049bdab4330ebb63fe4e8</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>relationShipType</identifier>
						<value>HAS</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>objectType</identifier>
						<value>HAS</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>sourceIdentifier</identifier>
						<value>NFVD-AGW</value>
					</attributeIdentifierValue>
```

```
                    </attributeList>
                </objectInfo>
            </objectCreation>
        </notification>
</Notifications>
```


c. LCM Update Notification:

a) Artifact:

Selectors are as below:

{artifactCategory=GENERIC, artifactType=artifact, NOMOriginalProvider=NFVD-AGW, OPERATION_TYPE=1, artifactId=c4111b54-57dc-39c4-ba6d-a2110027e757, NOMType=http://hp.com/openmediation/alarms/2011/08, NOMOriginalProviderHost=15.154.112.29, externalId=extra-specs-a908fef6-2979-4072-a912-d84246655704, sourceIdentifier=NFVD-AGW, artifactFamily=EXTRA_SPECS}}


```
<Notifications xmlns="http://hp.com/openmediation/topologyNotifications/2011/06">
        <notification>
            <attributeValueChange>
                <managedObject>
                    <classInstance className="VNF_COMPONENT:NEO4J"
instanceName="VNFC_NEO4J_Raghav"/>
                </managedObject>
                <eventTime>2017-03-03T12:24:50.209+05:30</eventTime>
                <attributeValueChangeInfo>
                    <sourceIndicator>NFVD-AGW</sourceIndicator>
                    <artifactId>10c599d0-bf6f-4be0-b101-d071320d156a</artifactId>
                    <externalId/>
                    <attributeValueChangeDefinition>
                        <attributeValueChangeEntry>
                            <identifier>lastUpdateTimestamp</identifier>
                            <newValue>2017-03-03T12:24:47.609+0530</newValue>
                            <oldValue>2017-03-02T11:39:56.729+0530</oldValue>
                        </attributeValueChangeEntry>
                        <attributeValueChangeEntry>
                            <identifier>GENERAL.Name</identifier>
                            <newValue>VNFC_NEO4J_Raghav</newValue>
                            <oldValue>VNFC_NEO4J</oldValue>
                        </attributeValueChangeEntry>
                    </attributeValueChangeDefinition>
                </attributeValueChangeInfo>
            </attributeValueChange>
        </notification>
</Notifications>
```


d. LCM Delete Notification:

a) Artifact:

Selectors are as below:

{artifactCategory=OPENSTACK, artifactType=artifact, NOMOriginalProvider=NFVD-AGW, OPERATION_TYPE=2, artifactId=**132eae02-6bf9-3e83-8ec6-be9762937501**, NOMType=http://hp.com/openmediation/alarms/2011/08, NOMOriginalProviderHost=, externalId= **security-group-41c59e49-8d8a-46ec-87e5-3f9165fb5b90-a5932f2d-9b96-4af0-8e41-726cd0ffaf05**, sourceIdentifier=NFVD-AGW, artifactFamily= SECURITY_GROUP }


```
<Notifications xmlns="http://hp.com/openmediation/topologyNotifications/2011/06">
	<notification>
		<objectDeletion>
			<managedObject>
				<classInstance className="SECURITY_GROUP:OPENSTACK"
instanceName="default"/>
			</managedObject>
			<eventTime>2017-03-03T12:29:27.060+05:30</eventTime>
			<objectInfo>
				<artifactId>132eae02-6bf9-3e83-8ec6-be9762937501</artifactId>
				<externalId>security-group-41c59e49-8d8a-46ec-87e5-3f9165fb5b90-a5932f2d-
9b96-4af0-8e41-726cd0ffaf05</externalId>
				<sourceIndicator>NFVD-AGW</sourceIndicator>
				<attributeList>
					<attributeIdentifierValue>
						<identifier>STATUS.LifeCycle_State_Date</identifier>
						<value>NA</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>artifactCategory</identifier>
						<value>OPENSTACK</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>STATUS.Operational_Status.TYPE</identifier>
						<value>TEXT</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>artifactStatus</identifier>
						<value>ACTIVE</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>STATUS.Admin_Status_Date.ORDER</identifier>
						<value>8</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>artifactSubtype</identifier>
						<value/>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>artifactStatusEnabled</identifier>
						<value>true</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>STATUS.Operational_Status_Date.UNIT</identifier>
						<value>Date</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>STATUS.Operational_Status</identifier>
						<value>NA</value>
					</attributeIdentifierValue>
					<attributeIdentifierValue>
						<identifier>STATUS.additionalText.UNIT</identifier>
```

```
                <value>TEXT</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>artifactVersion</identifier>
                <value/>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>STATUS.Operational_Status_Date.ORDER</identifier>
                <value>2</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>

<identifier>STATUS.Admin_Status_Date.MANDATORY</identifier>
                <value>false</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>STATUS.Admin_Status_Date.TYPE</identifier>
                <value>Date</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>STATUS.Admin_Status_Date.UNIT</identifier>
                <value>Date</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>SECURITY_GROUP.Id</identifier>
                <value>a5932f2d-9b96-4af0-8e41-726cd0ffaf05</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>GENERAL.Description</identifier>
                <value>Default security group</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>STATUS.Source.MANDATORY</identifier>
                <value>false</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>

<identifier>STATUS.Operational_Status_Date.MANDATORY</identifier>
                <value>false</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>artifactPhysical</identifier>
                <value>false</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>GENERAL.Name</identifier>
                <value>default</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>STATUS.Admin_Status.UNIT</identifier>
                <value>TEXT</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>STATUS.Source.UNIT</identifier>
                <value>TEXT</value>
            </attributeIdentifierValue>
            <attributeIdentifierValue>
                <identifier>STATUS.Source.TYPE</identifier>
                <value>TEXT</value>
```

```
</attributeIdentifierValue>
<attributeIdentifierValue>

<identifier>STATUS.LifeCycle_State_Date.MANDATORY</identifier>
<value>false</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
<identifier>GENERAL.Type</identifier>
<value/>
</attributeIdentifierValue>
<attributeIdentifierValue>
<identifier>STATUS.Operational_Status_Date</identifier>
<value>NA</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
<identifier>STATUS.LifeCycle_State_Date.UNIT</identifier>
<value>Date</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
<identifier>lastUpdateTimestamp</identifier>
<value>2017-03-03T08:53:38.919+0530</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
<identifier>STATUS.Operational_Status_Date.TYPE</identifier>
<value>Date</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
<identifier>creationTimestamp</identifier>
<value>2017-03-02T12:00:37.696+0530</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
<identifier>STATUS.Operational_Status.ORDER</identifier>
<value>1</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
<identifier>STATUS.Admin_Status.ORDER</identifier>
<value>7</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
<identifier>artifactGroup</identifier>
<value/>
</attributeIdentifierValue>
<attributeIdentifierValue>
<identifier>artifactFamily</identifier>
<value>SECURITY_GROUP</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
<identifier>artifactVisibleStatus</identifier>
<value>ACTIVE</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
<identifier>INTEGRATION.ExternalId</identifier>
<value>security-group-41c59e49-8d8a-46ec-87e5-3f9165fb5b90-
a5932f2d-9b96-4af0-8e41-726cd0ffaf05</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
<identifier>artifactType</identifier>
<value/>
</attributeIdentifierValue>
```

```
<attributeIdentifierValue>
        <identifier>ASSURANCE_KPI.Severity</identifier>
        <value>NA</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.Source.ORDER</identifier>
        <value>3</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.Operational_Status.UNIT</identifier>
        <value>TEXT</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.LifeCycle_State</identifier>
        <value>NA</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.LifeCycle_State.UNIT</identifier>
        <value>TEXT</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.additionalText.MANDATORY</identifier>
        <value>false</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.Admin_Status</identifier>
        <value>NA</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.Operational_Status.MANDATORY</identifier>
        <value>false</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>ASSURANCE_KPI.VALUE</identifier>
        <value>NA</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.Source</identifier>
        <value>NA</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.LifeCycle_State.ORDER</identifier>
        <value>4</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.LifeCycle_State.TYPE</identifier>
        <value>TEXT</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>ASSURANCE_KPI.UNIT</identifier>
        <value>NA</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.Admin_Status_Date</identifier>
        <value>NA</value>
</attributeIdentifierValue>
<attributeIdentifierValue>
        <identifier>STATUS.additionalText</identifier>
        <value>NA</value>
```

```
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>ASSURANCE_STATUS.Severity</identifier>
                        <value>NA</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>ASSURANCE_KPI.LABEL</identifier>
                        <value>NA</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>STATUS.LifeCycle_State_Date.ORDER</identifier>
                        <value>5</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>STATUS.LifeCycle_State_Date.TYPE</identifier>
                        <value>Date</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>ASSURANCE_KPI.DESCRIPTION</identifier>
                        <value>NA</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>ASSURANCE_KPI.COUNTERNAME</identifier>
                        <value>NA</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>ASSURANCE_OPERATION</identifier>
                        <value>create</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>STATUS.additionalText.TYPE</identifier>
                        <value>TEXT</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>artifactId</identifier>
                        <value>132eae02-6bf9-3e83-8ec6-be9762937501</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>templateId</identifier>
                        <value/>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>STATUS.LifeCycle_State.MANDATORY</identifier>
                        <value>false</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>STATUS.additionalText.ORDER</identifier>
                        <value>6</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>STATUS.Admin_Status.MANDATORY</identifier>
                        <value>false</value>
                    </attributeIdentifierValue>
                    <attributeIdentifierValue>
                        <identifier>STATUS.Admin_Status.TYPE</identifier>
                        <value>TEXT</value>
                    </attributeIdentifierValue>
                </attributeList>
            </objectInfo>
```

```
                </objectDeletion>
        </notification>
</Notifications>
```


e. LCM Delete Relationship:

Selectors/JMS properties are as below:

{parentArtifactCategory= **OPENSTACK**, artifactType=Relationship, OPERATION_TYPE=2, childArtifactFamily=**SECURITY_GROUP_RULE**,NOMOriginalProviderHost=nfvdvm49.in.rdlabs.hpecorp.net, childArtifactId= **ae805dd1-470b-3185-921d-7c7c3a6b4a6a**, sourceIdentifier=NFVD-AGW, NOMOriginalProvider=NFVD-AGW, parentArtifactFamily= **SECURITY_GROUP**, childArtifactCategory=**OPENSTACK**, NOMType=http://hp.com/openmediation/alarms/2011/08, childExternalId= **security-group-rule-41c59e49-8d8a-46ec-87e5-3f9165fb5b90-b275fd3d-d189-4b24-834b-aad6dadad221**, parentArtifactId=**132eae02-6bf9-3e83-8ec6-be9762937501**, parentExternalId= **security-group-41c59e49-8d8a-46ec-87e5-3f9165fb5b90-a5932f2d-9b96-4af0-8e41-726cd0ffaf05**}}


```
<Notifications xmlns="http://hp.com/openmediation/topologyNotifications/2011/06">
        <notification>
                <objectDeletion>
                        <managedObject>
                                <classInstance className="RELATIONSHIP:ATTACH"/>
                                <classInstance className="SECURITY_GROUP:OPENSTACK"
instanceName="default"/>
                                <classInstance className="SECURITY_GROUP_RULE:OPENSTACK"
instanceName="default"/>
                        </managedObject>
                        <eventTime>2017-03-03T12:29:27.040+05:30</eventTime>
                        <objectInfo>
                                <sourceIndicator>NFVD-AGW</sourceIndicator>
                                <attributeList>
                                        <attributeIdentifierValue>
                                                <identifier>childArtifactId</identifier>
                                                <value>ae805dd1-470b-3185-921d-7c7c3a6b4a6a</value>
                                        </attributeIdentifierValue>
                                        <attributeIdentifierValue>
                                                <identifier>parentArtifactId</identifier>
                                                <value>132eae02-6bf9-3e83-8ec6-be9762937501</value>
                                        </attributeIdentifierValue>
                                        <attributeIdentifierValue>
                                                <identifier>childExternalId</identifier>
                                                <value>security-group-rule-41c59e49-8d8a-46ec-87e5-
3f9165fb5b90-b275fd3d-d189-4b24-834b-aad6dadad221</value>
                                        </attributeIdentifierValue>
                                        <attributeIdentifierValue>
                                                <identifier>parentExternalId</identifier>
                                                <value>security-group-41c59e49-8d8a-46ec-87e5-3f9165fb5b90-
a5932f2d-9b96-4af0-8e41-726cd0ffaf05</value>
                                        </attributeIdentifierValue>
                                        <attributeIdentifierValue>
                                                <identifier>relationShipType</identifier>
                                                <value>ATTACH</value>
```

```
                                        </attributeIdentifierValue>
                                        <attributeIdentifierValue>
                                                <identifier>objectType</identifier>
                                                <value>ATTACH</value>
                                        </attributeIdentifierValue>
                                        <attributeIdentifierValue>
                                                <identifier>sourceIdentifier</identifier>
                                                <value>NFVD-AGW</value>
                                        </attributeIdentifierValue>
                                </attributeList>
                        </objectInfo>
                </objectDeletion>
        </notification>
</Notifications>
```


10.10.2 State Change Notifications

NFVD generates notifications when there are changes in the Operational or Administrative states of its own components (self-management). For VNF/VMs, it is optional based on whether states are managed/updated or not (by default no state management).

In such cases, Assurance publishes alarms to the OM JMS topic named `com.hp.openmediation.alarms`, which can be consumed by OSS, Analytics tools, and so on. The body of the notification is a string that contains a well-formed and valid XML document according to the http://hp.com/openmediation/topologyNotifications/2011/06 schema.

Operational Status Change Notification:
Selectors are as below

{artifactType=artifact, NOMOriginalProvider=NFVD-AGW, OPERATION_TYPE=3, NOMType=http://hp.com/openmediation/alarms/2011/08, NOMOriginalProviderHost=15.154.112.29, sourceIdentifier=NFVD-AGW}}


```
<Notifications xmlns="http://hp.com/openmediation/topologyNotifications/2011/06">
        <notification>
                <operationalStateChange>
                        <managedObject>
                                <classInstance className="SERVER:GENERIC" instanceName="nfvdvm60"/>
                        </managedObject>
                        <eventTime>2017-03-03T12:42:30.124+05:30</eventTime>
                        <operationalStateValueChangeInfo>
                                <sourceIndicator>NFVD-AGW</sourceIndicator>
                                <attributeIdentifierList>
                                        <attributeIdentifier>d8bfde66-aaa6-3d18-b4b8-
1af675dc8b85</attributeIdentifier>
                                </attributeIdentifierList>
                                <attributeValueChangeDefinition>
                                        <attributeValueChangeEntry>
                                                <identifier>STATUS.Operational_Status</identifier>
                                                <newValue>DOWN</newValue>
                                                <oldValue>UP</oldValue>
```

```
                                        </attributeValueChangeEntry>
                                        <attributeValueChangeEntry>
                                                <identifier>STATUS.Administrative_Status</identifier>
                                                <newValue/>
                                                <oldValue/>
                                        </attributeValueChangeEntry>
                                </attributeValueChangeDefinition>
                        </operationalStateValueChangeInfo>
                </operationalStateChange>
        </notification>
</Notifications>
```


10.10.3 Operation success/failure Notifications

Assurance also generates alarms when there are operational error/success in NFVD. For example, SCALE OUT operation on VM instance.

In such cases, Assurance publishes alarms to the OM JMS topic named `com.hp.openmediation.alarms`, which can be consumed by OSS, Analytics tools, etc. The body of the notification is a string that contains a well-formed and valid XML document according to the `http://hp.com/openmediation/alarms/2011/08` schema.

The user must use JMS connection and connect to the broker running on <AA host>:10000

If the user is interested only in specific messages, then the user may use the JMS Message Selector to specify the messages of interest.

For more information about the alarm creation schema, refer to the *OSS Open Mediation Functional Specification Guide* from Open Mediation.

Sample Alarms are listed below:

a) Operation Success Alarm:


```
<AlarmCreationInterface xmlns="http://hp.com/uca/expert/x733Alarm">

  <identifier>c9065b61-15d3-4d3a-8da7-1b89e443ce99</identifier>

  <sourceIdentifier>NFVD-AGW</sourceIdentifier>

  <alarmRaisedTime>3917-02-28T10:25:01.135+05:30</alarmRaisedTime>

  <originatingManagedEntity>VNF:GENERIC VNF_SO</originatingManagedEntity>

  <alarmType>PROCESSING_ERROR_ALARM</alarmType>

  <probableCause>software error</probableCause>

  <perceivedSeverity>INDETERMINATE</perceivedSeverity>
```

```
  <networkState>NOT_CLEARED</networkState>

  <operatorState>NOT_ACKNOWLEDGED</operatorState>

  <problemState>NOT_HANDLED</problemState>

  <specificProblem>SCALE_OUT</specificProblem>



  <additionalText>(MONITORED_ENTITY_ID=2380b5c3-3463-4b2b-a5ad-
b3c19d46bf41)(MONITORED_ENTITY_FAMILY=VNF)(MONITORED_ENTITY_CATEGORY=GENERIC)(MONITORED_ENTITY
_NAME=VNF_SO)</additionalText>

  <customFields>

    <customField value="FINISH_OK" name="status"/>

  </customFields>

</AlarmCreationInterface>
```


Here <specificProblem> field represents the operation performed, in the this its SCALE OUT of a VM instance

Here <customField value="FINISH_OK" name="status"/>, means the status of the operation FINISH_OK is the value returned by the workflow on Fulfillment, FINISH_OK means “SUCCESS” operation.

b) Operation Failure Alarm:


```
<AlarmCreationInterface xmlns="http://hp.com/uca/expert/x733Alarm">

  <identifier>79b3fe58-2d1a-4753-831f-9a90d204a02c</identifier>

  <sourceIdentifier>NFVD-AGW</sourceIdentifier>

  <alarmRaisedTime>3917-02-28T06:39:26.116+05:30</alarmRaisedTime>

  <originatingManagedEntity>VNF:GENERIC VNF_Deploy</originatingManagedEntity>

  <alarmType>PROCESSING_ERROR_ALARM</alarmType>

  <probableCause>software error</probableCause>

  <perceivedSeverity>CRITICAL</perceivedSeverity>

  <networkState>NOT_CLEARED</networkState>

  <operatorState>NOT_ACKNOWLEDGED</operatorState>
```

```
<problemState>NOT_HANDLED</problemState>

<specificProblem>SCALE_UP</specificProblem>

<additionalText>(MONITORED_ENTITY_ID=2aaa9c48-808e-470f-9b48-
f9b0d221d564)(MONITORED_ENTITY_FAMILY=VNF)(MONITORED_ENTITY_CATEGORY=GENERIC)(MONITORED_ENTIT
Y_NAME=VNF_Deploy)</additionalText>

<customFields>

<customField value="FINISH_ERROR_ON_CHILD" name="status"/>

</customFields>

</AlarmCreationInterface>
```


Here <specificProblem> field represents the operation performed, in the this its SCALE UP of a VM instance

Here <customField value="FINISH_OK" name="status"/>, means the status of the operation FINISH_ERROR_ON_CHILD, is the value returned by the workflow on Fulfillment, FINISH_ERROR_ON_CHILD means “FAILURE” operation

10.10.4 Self Monitor Alarms

The sample alarm, as seen in Open Mediation (SNMP trap destination) is shown below:


```
<alarms:Alarms xmlns:alarms="http://hp.com/openmediation/alarms/2011/08">

<alarms:AlarmCreationInterface>

<alarms:identifier>10623:270ea875-3b61-4c2a-bb59-bdbda7f84a13</alarms:identifier>

<alarms:sourceIdentifier>NFVD_Source</alarms:sourceIdentifier>

<alarms:alarmRaisedTime>2017-03-03T15:07:00Z</alarms:alarmRaisedTime>

<alarms:target>SNMP-Customization-Sitescope-FlowTarget</alarms:target>

<alarms:originatingManagedEntity>VNF_COMPONENT:SOSA  SOSA2</alarms:originatingManagedEntity>

<alarms:originatingManagedEntityStructure>

<alarms:classInstance clazz="Script"
instance="SiteScope\SELF_MONITORS\RAGHAV2\SOSA\SOSA2\artifactId-7cb7c624-a624-46a4-8ffc-0590dfe69e41\Process
Linux Group\Landscape Monitor"/>

</alarms:originatingManagedEntityStructure>

<alarms:alarmType>QUALITY_OF_SERVICE_ALARM</alarms:alarmType>
```

<alarms:probableCause>THRESHOLD_CROSSED</alarms:probableCause>

<alarms:perceivedSeverity>CLEAR</alarms:perceivedSeverity>

<alarms:networkState>CLEARED</alarms:networkState>

<alarms:operatorState>NOT_ACKNOWLEDGED</alarms:operatorState>

<alarms:problemState>NOT_HANDLED</alarms:problemState>

<alarms:specificProblem>pid,</alarms:specificProblem>

<alarms:additionalText>errorOnly=|goodOnly=

pid: 21229

|warningOnly=|group=Process Linux Group|groupdescription=(MONITOR_TYPE=Process_LINUX)(MONITORED_ENTITY_ID=7cb7c624-a624-46a4-8ffc-0590dfe69e41)(MONITORED_ENTITY_FAMILY=VNF_COMPONENT)(MONITORED_ENTITY_CATEGORY=SOSA)(MONITORED_ENTITY_NAME=SOSA2)(MONITORED_ENTITY_MODE=ACTIVE)(MONITORED_ENTITY_NODE_IS_PRIMARY=true)(IS_HA_COMPONENT=true)(counterName=pid,status)(displayName=Process ID of SOSA, Status of SOSA)(unit=number, Boolean)(description=Process ID, Process Status)(template Version=16108)

|id=1|monitorUUID=99227888-ce49-480a-acc1-a72e19a7b449|monitorTypeDisplayName=Script|fullMonitorName=SiteScope\SELF_MONITORS\RAGHAV2\SOSA\SOSA2\artifactId-7cb7c624-a624-46a4-8ffc-0590dfe69e41\Process Linux Group\Landscape Monitor state=pid=21229

|time=3:07 PM 3/3/17|templateDeployPath=SELF_MONITORS/RAGHAV2/SOSA/SOSA2/artifactId-7cb7c624-a624-46a4-8ffc-0590dfe69e41/Process Linux Group</alarms: additional Text>

</alarms: AlarmCreationInterface>

</alarms:Alarms>

10.9 Extending Action capabilities using UCA-Automation

The correlation and autonomous actions correspond to correlating traps received from an NFV source and taking autonomous actions based on the NFV topology. Correlation is possible when the collection event is received from various NFV sources at the UCA-EBC value pack. Event collection to UCA-EBC depends on the generic SNMP channel adaptor configuration.

10.9.3 Correlation and autonomous process

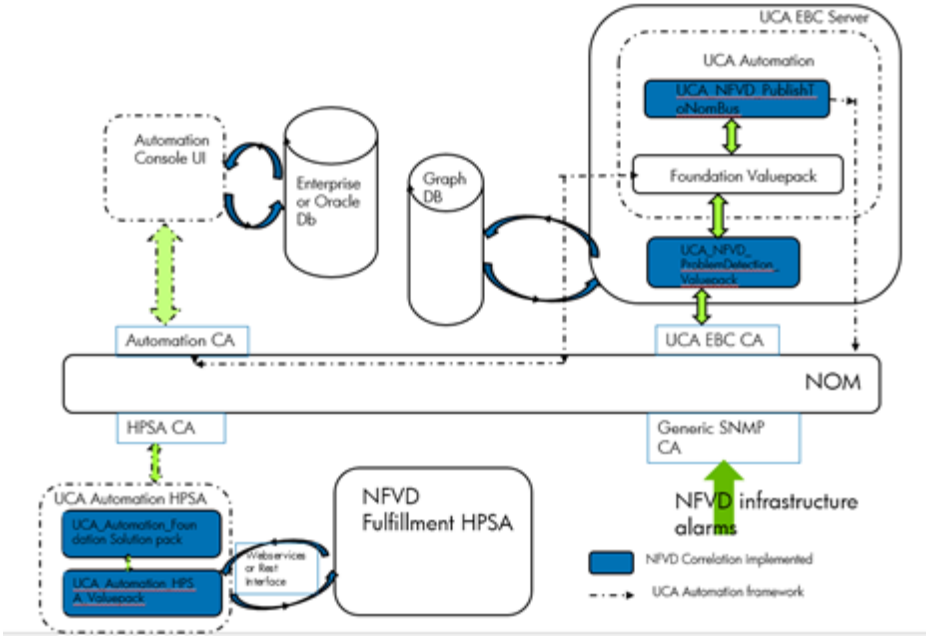


Figure 69: Correlation and autonomous action process

Correlation and Autonomous actions can be categorized as follows:

- Alarm enrichment
- Autonomous action
- Action status and reporting

10.9.3.6 Alarm enrichment

The alarms received from the NFV source should be enriched with NFV topology information and parameters required for autonomous actions. The UCA NFVD Problem detection value pack delivered with the NFV Director 4.0 enriches alarms. Depending on the action, it fetches the required parameters and publishes the alarms to Open Mediation.

The following is a sample of a raw alarm received from the NFV Director source.

```
<AlarmCreationInterface xmlns="http://hp.com/uca/expert/x733Alarm">
     <identifier>5:65d95213-00fd-4764-adfa-86b00af0881a</identifier>
      <sourceIdentifier>NFVD_Source</sourceIdentifier>
       <alarmRaisedTime>2014-06-23T11:55:00Z</alarmRaisedTime>
       <targetValuePack>SNMP-Customization-SiteScope-
FlowTarget</targetValuePack>
    <originatingManagedEntity>KVM_TestVM</originatingManagedEntity>
    <originatingManagedEntityStructure>
        <classInstance instance="SiteScope\HP\KVM VM CPU Monitor\KVM_TestVM"
clazz="Generic Hypervisor"/>
    </originatingManagedEntityStructure>
    <alarmType>QUALITY_OF_SERVICE_ALARM</alarmType>
    <probableCause>calculatedCounterValue1 == 1 error</probableCause>
    <perceivedSeverity>WARNING</perceivedSeverity>
    <networkState>NOT_CLEARED</networkState>
```

```
    <operatorState>NOT_ACKNOWLEDGED</operatorState>
    <problemState>NOT_HANDLED</problemState>
    <specificProblem>cpu_usage_medium: 1</specificProblem>
      <additionalText>_customPropertiesValues=|_httpPort=8888|
            _webserverAddress=15.154.72.226|
      alertHelpURL=http://15.154.72.226:8080/SiteScope/sisdocs/doc_lib/index.ht
m?single=false&amp;context=system_avail&amp;topic=config_sis_alert|
            diagnosticTraceRoute=|errorOnly=|goodOnly=cpu_usage_high: 0
cpu_usage_low: 0|FullGroupId=HP: KVM VM CPU Monitor|group=KVM VM CPU Monitor|
            groupdescription=CPU |
                                  groupID=201087530|
                                  id=&amp;lt;id&amp;gt;|
                                 mainStateProperties= groupID: 201087530
                       calculatedCounterValue1: 0
            calculatedCounterValue2: 1
            calculatedCounterValue3: 0|
      monitorDrilldownUrl=http://ossvm8.ind.hp.com:8080/SiteScope/servlet/Main?
activeid=201087531&amp;activerighttop=dashboard&amp;view=new&amp;dashboard_view
=Details&amp;dashboard_model=true&amp;sis_silent_login_type=encrypted&amp;login
=%28sisp%29knjxbqDESkAn5mKcvgTmj%2FyFwHH5Ke3m&amp;password=%28sisp%29EzqXbIXEFD
%2BJbE1N1T%2FZlELjja0DKaN7|

      monitorServiceId=SiteScopeMonitor:201087530:201087531|
      monitorTypeDisplayName=Generic Hypervisor|
      monitorUUID=9a145576-cefb-483f-beaa-bd3bd6f9158f|
      mountName=[/dev/mapper/VolGroup00-root_vol, /dev/mapper/VolGroup00-
root_vol (/), /dev/sda1, /dev/sda1 (/boot), /dev/mapper/VolGroup00-home_vol,
/dev/mapper/VolGroup00-home_vol (/home), /dev/mapper/VolGroup00-opt_vol,
/dev/mapper/VolGroup00-opt_vol (/opt), /dev/mapper/VolGroup00-tmp_vol,
/dev/mapper/VolGroup00-tmp_vol (/tmp), /dev/mapper/VolGroup00-usr_vol,
/dev/mapper/VolGroup00-usr_vol (/usr), /dev/mapper/VolGroup00-var_vol,
/dev/mapper/VolGroup00-var_vol (/var), /dev/mapper/VolGroup00-var_crash_vol,
/dev/mapper/VolGroup00-var_crash_vol (/var/crash), /dev/mapper/VolGroup00-
var_log_audit, /dev/mapper/VolGroup00-var_log_audit (/var/log/audit)]|

      multiViewUrl=http://15.154.72.226:8080/SiteScope/MultiView|
      fullMonitorName=SiteScope\HP\KVM VM CPU Monitor\KVM_TestVM|
      newSiteScopeURL=http://15.154.72.226:8080/SiteScope|sample=5|
      SiteScopeBaseUrl=http://ossvm8.ind.hp.com:8080|
      SiteScopeHost=ossvm8.ind.hp.com|
      SiteScopeURL=http://15.154.72.226:8080/SiteScope|
      SiteScopeuserurl=http://15.154.72.226:8080/SiteScope/userhtml/SiteScope.h
tml|
      state=Virttop Management/Domains
                Information/KVM_TestVM/Performance/%CPU=0.1, ,
cpu_usage_high=0,
                cpu_usage_medium=1, cpu_usage_low=0|
      tag=|targetHost=sheep.gre.hp.com|
      targetIP=16.16.94.139|
      targetIPVersion=IPV4|
      templateDeployPath=HP/KVM VM CPU Monitor|
      time=11:54 AM 6/23/14|
      warningOnly= cpu_usage_medium: 1|customerId=&amp;lt;customerId&amp;gt;
     </additionalText>
</AlarmCreationInterface>
```

The following is an enriched alarm after having been processed by the UCA NFVD Problem detection value pack.

```
Valuepack.
- alarmRaisedTime                = 2014-05-05T20:41:00.848+05:30
```

```
    - sourceIdentifier                = NFVD_Source
    - originatingManagedEntity        = KVM_TestVM
    - originatingManagedEntityStructure
            -> Host = ossvm1.ind.hp.com
    - alarmType                       = QUALITY_OF_SERVICE_ALARM
    - probableCause                   = UtilizationPercentage
    - perceivedSeverity               = CRITICAL
    - networkState                    = NOT_CLEARED
    - operatorState                   = NOT_ACKNOWLEDGED
    - problemState                    = NOT_HANDLED
    - problemInformation              = Attribute not available
    - specificProblem                 = ERROR
    - additionalInformation           = null
    - additionalText                  = SiteScope alarm|MONITOR.cpuMonitor-
001|CONDITION=ERROR|group=KVM VM CPU
Monitor|groupdescription=CPU|groupID=201070542|id=1
    - proposedRepairActions           = null
    - notificationIdentifier          = 0
    - correlationNotificationIdentifiers = Attribute not available
    - timeInMilliseconds              = 1399302660848 [2014/05/05 20:41:00.848
+0530]
    - targetValuePack                 = null
    - sourceScenarios                 =
[com.hp.uca.expert.vp.pd.ProblemDetection]
    - passingFilters                  = [NfvdScenario]
    - passingFiltersTags              = {NfvdScenario=[Trigger, ProblemAlarm,
SubAlarm]}
    - passingFiltersParams            = {NfvdScenario={}}
    - hasParents                      = false
    - parentsNumber                   = 0
    - parents                         = null
    - hasChildren                     = false
    - childrenNumber                  = 0
    - children                        = null
    - justInjected                    = false
    - aboutToBeRetracted              = false
    - hasStateChanged                 = false
    - stateChanges                    = none
    - hasAVCChanged                   = false
    - attributeValueChanges           = none
    - customFields
      -> userText = NFVD-PD
      -> NFVTopology =
<Start>
<VIRTUAL_MACHINE>
artifactCategory=GENERIC;
GENERAL.Description=A Virtual machine;
artifactId=KVMVM-2001;
GENERAL.Name=KVM_TestVM;
SERVICE_ACTION=CREATE;
templateId=template-512;
GENERAL.Type=VMtype;
SERVICE_TYPE=NFVD;
SERVICE_NAME=INSART;
artifactFamily=VIRTUAL_MACHINE;
GENERAL.hostname=KVM_TestVM;
GENERAL.Management_access=http://vm1.ind.com:8080;
</VIRTUAL_MACHINE>
<VNF_COMPONENT>
artifactCategory=GENERIC;
```

```
GENERAL.Description=This is VNFC component;
artifactId=VNFC-BLR1;
GENERAL.Name=vnfc-BNGALORE;
SERVICE_ACTION=CREATE;
templateId=template-8001;
lastUpdateTimestamp=15-04-2014 12:58;
creationTimestamp=15-04-2014 12:57;
SERVICE_TYPE=NFVD;
SERVICE_NAME=INSART;
artifactFamily=VNF_COMPONENT;
</VNF_COMPONENT>
<VIRTUAL_PORT>
artifactCategory=GENERIC;
artifactId=Ethe.1990;
SERVICE_ACTION=CREATE;
templateId=template-512;
lastUpdateTimestamp=15-04-2014 12:58;
SERVICE_TYPE=NFVD;
SERVICE_NAME=INSART;
artifactFamily=VIRTUAL_PORT;
INFO.Speed=10;
INFO.Name=EthernetPort-10GB;
INFO.ID=Ethe.1990;
INFO.Type=Port;
INFO.MAC=12:34:56:78:9A:BD;
</VIRTUAL_PORT>
<VIRTUAL_LUN>
artifactCategory=GENERIC;
artifactId=LUN-1001;
SERVICE_ACTION=CREATE;
templateId=template-512;
lastUpdateTimestamp=15-04-2014 12:58;
SERVICE_TYPE=NFVD;
SERVICE_NAME=INSART;
artifactFamily=VIRTUAL_LUN;
INFO.Name=LUN-1.2-BLR;
INFO.Amount=30;
INFO.ID=LUN-1001;
INFO.Type=Storage;
</VIRTUAL_LUN>
<VIRTUAL_DISK>
artifactCategory=GENERIC;
artifactId=vDisk1;
SERVICE_ACTION=CREATE;
templateId=template-512;
lastUpdateTimestamp=15-04-2014 12:58;
SERVICE_TYPE=NFVD;
SERVICE_NAME=INSART;
artifactFamily=VIRTUAL_DISK;
INFO.Name=Seagate-500GB-SATA;
INFO.Amount=2;
INFO.ID=PSATA-500;
INFO.Type=Memory;
</VIRTUAL_DISK>
<VIRTUAL_MEMORY>
artifactCategory=GENERIC;
artifactId=RAM-4001;
SERVICE_ACTION=CREATE;
templateId=template-512;
lastUpdateTimestamp=15-04-2014 12:58;
```

```
creationTimestamp=15-04-2014 12:57;
SERVICE_TYPE=NFVD;
SERVICE_NAME=INSART;
artifactFamily=VIRTUAL_MEMORY;
INFO.Name=DDR-RAM-8GB;
INFO.Amount=1;
INFO.ID=RAM-4001;
INFO.Type=Memory;
</VIRTUAL_MEMORY>
<VIRTUAL_CORE>
artifactCategory=GENERIC;
artifactId=VCore-1001;
SERVICE_ACTION=CREATE;
templateId=template-512;
lastUpdateTimestamp=15-04-2014 12:58;
SERVICE_TYPE=NFVD;
SERVICE_NAME=INSART;
artifactFamily=VIRTUAL_CORE;
INFO.Speed=2233;
INFO.Name=VCore-I5;
INFO.Amount=3;
INFO.ID=Serial-VCore-1001;
INFO.Type=Virtual Core;
</VIRTUAL_CORE>
<MONITOR>
artifactCategory=GENERIC;
GENERAL.Description=Network Monitor;
artifactId=networkMonitor-004;
GENERAL.Name=Network;
SERVICE_ACTION=CREATE;
templateId=template-004;
lastUpdateTimestamp=30-04-2014 12:57;
creationTimestamp=30-04-2014 12:57;
SERVICE_TYPE=NFVD;
SERVICE_NAME=INSART;
artifactFamily=MONITOR;
GENERAL.Frequency=30;
GENERAL.DeploymentPath=;
</MONITOR>
<MONITOR>
artifactCategory=GENERIC;
GENERAL.Description=Disk Monitor;
artifactId=diskMonitor-003;
GENERAL.Name=Disk;
SERVICE_ACTION=CREATE;
templateId=template-003;
lastUpdateTimestamp=30-04-2014 12:57;
creationTimestamp=30-04-2014 12:57;
SERVICE_TYPE=NFVD;
SERVICE_NAME=INSART;
artifactFamily=MONITOR;
GENERAL.Frequency=30;
GENERAL.DeploymentPath=HP/NFVD/NS-Routing/VNF-K/VNFC-K;
</MONITOR>
<MONITOR>
artifactCategory=GENERIC;
GENERAL.Description=Memory Monitor;
artifactId=memoryMonitor-002;
GENERAL.Name=Memory;
SERVICE_ACTION=CREATE;
```

```
templateId=template-002;
lastUpdateTimestamp=30-04-2014 12:57;
creationTimestamp=30-04-2014 12:57;
SERVICE_TYPE=NFVD;
SERVICE_NAME=INSART;
artifactFamily=MONITOR;
GENERAL.Frequency=30;
GENERAL.DeploymentPath=HP/NFVD/NS-Routing/VNF-Z/VNFC-Z;
</MONITOR>
<MONITOR>
artifactCategory=GENERIC;
GENERAL.Description=CPU Monitor;
artifactId=cpuMonitor-001;
GENERAL.Name=CPU;
SERVICE_ACTION=CREATE;
templateId=template-001;
lastUpdateTimestamp=30-04-2014 12:57;
creationTimestamp=30-04-2014 12:57;
SERVICE_TYPE=NFVD;
SERVICE_NAME=INSART;
artifactFamily=MONITOR;
GENERAL.Frequency=30;
GENERAL.DeploymentPath=HP/NFVD/NS-Routing/VNF-Y/VNFC-Y;
</MONITOR>
<End>
      -> Evp = UCA_NFVD_PublishToNomBus
      -> Evpversion = 1.0
      -> Evpscenario = publishToNomBus
      -> Problem = NFVD:SCALE_OUT_CPU
      -> Parameternames = ArtifactInstanceId
      -> Parametervalues = KVMVM-2001
      -> Resourceinstance = KVMVM-2001
      -> Resourcetype = INVENTORY
      -> Actionpreset = true
      -> Action = SCALE_OUT
```

10.9.3.7 Autonomous action

NOTE: SCALE_UP and SCALE_DOWN Actions are supported only on VNF level.

SCALE_IN and SCALE_OUT Actions are supported for VNF/VNFCOMPONENT level.

The Autonomous action of the NFV Director is based on the UCA Automation Action Framework. Refer to the *HPE UCA Automation - Administrator and User Interface Guide* for additional information on the Automation framework.

The following table includes the mandatory alarm attributes required for Autonomous action.

Table 23: Autonomous action alarm attributes

Alarm Attribute Name	Alarm Attribute Value
`Evp`	UCA_NFVD_PublishToNomBus
`Evpversion`	1.0
`Evpscenario`	publishToNomBus
`Problem`	NFVD:<Problem Name as per UCA Automation>
`Action`	One of the following values:

	• SCALE_IN • SCALE_OUT • SCALE_UP • SCALE_DOWN • SCRIPT If the `Action` parameter is not available, the Autonomous action is not triggered. The value is fetched from the Graph Database, depending on the raw alarm received.
`Parameternames`	<Value depends on Action Type>
`Parametervalues`	<Value depends on Action Type>
`Resourcetype`	`INVENTORY`
`Actionpreset`	`TRUE`

10.9.3.8 Action status and reporting

The status of action and reporting is available in the **UCA Automation console**.

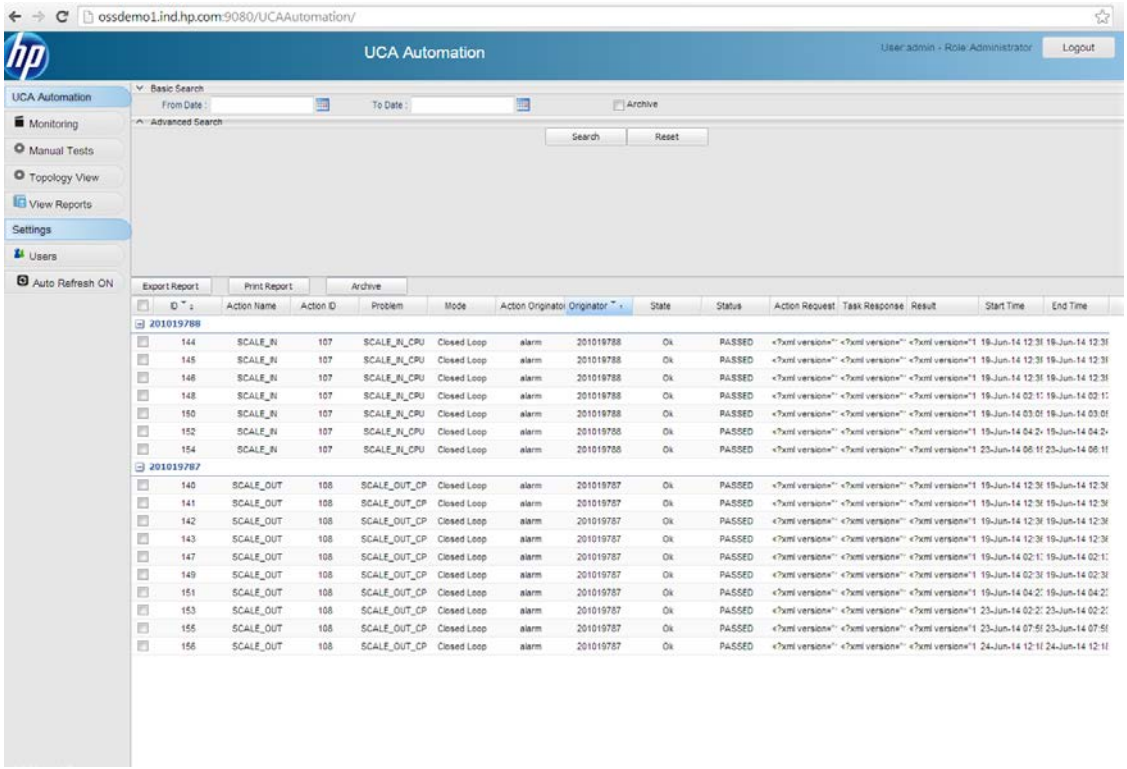
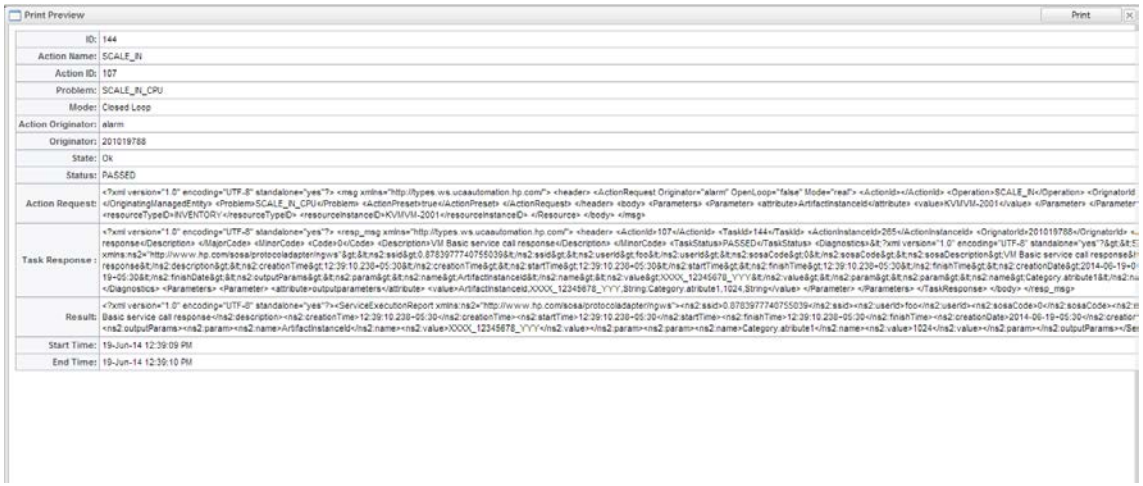


Figure 70: Action status snapshot



Print Preview

ID:	144
Action Name:	SCALE_IN
Action ID:	107
Problem:	SCALE_IN_CPU
Mode:	Closed Loop
Action Originator:	alarm
Originator:	201019788
State:	Ok
Status:	PASSED
Action Request:	<?xml version="1.0" encoding="UTF-8" standalone="yes"?> <msg xmlns="http://types.ws.ucaautomation.hp.com/"> <header> <ActionRequest Originator="alarm" OpenLoop="false" Mode="real"> <ActionId></ActionId> <Operation>SCALE_IN</Operation> <OriginatorId /> <OriginatingManagedEntity> <Problem>SCALE_IN_CPU</Problem> <ActionPreset>true</ActionPreset> </ActionRequest> </header> <body> <Parameters> <Parameter> <attribute>ArtifactInstanceId</attribute> <value>KVMVM-2001</value> </Parameter> <Parameter> <resourceTypeID>INVENTORY</resourceTypeID> <resourceInstanceID>KVMVM-2001</resourceInstanceID> </Resource> </body> </msg>
Task Response :	<?xml version="1.0" encoding="UTF-8" standalone="yes"?> <resp_msg xmlns="http://types.ws.ucaautomation.hp.com/"> <header> <ActionId>107</ActionId> <TaskId>144</TaskId> <ActionInstanceId>265</ActionInstanceId> <OriginatorId>201019788</OriginatorId> <... response</Description> </MajorCode> </MinorCode> <Code>0</Code> <Description>VM Basic service call response</Description> </MinorCode> <TaskStatus>PASSED</TaskStatus> <Diagnostics><?xml version="1.0" encoding="UTF-8" standalone="yes"?><S... xmlns:ns2="http://www.hp.com/sosa/protocoladapter/ngws"><ns2:ssid>0.8783977740755039</ns2:ssid><ns2:userid>foo</ns2:userid><ns2:sosaCode>0</ns2:sosaCode><ns2:sosaDescription>VM Basic service call response</... response</ns2:description><ns2:creationTime>12:39:10.238+05:30</ns2:creationTime><ns2:startTime>12:39:10.238+05:30</ns2:startTime><ns2:finishTime>12:39:10.238+05:30</ns2:finishTime><ns2:creationDate>2014-06-19+0... 19+05:30</ns2:finishDate><ns2:outputParams><ns2:param><ns2:name>ArtifactInstanceId</ns2:name><ns2:value>XXXX_12345678_YYY</ns2:value></ns2:param><ns2:param><ns2:name>Category.attribute1</ns2:na... </Diagnostics> <Parameters> <Parameter> <attribute>outputparameters</attribute> <value>ArtifactInstanceId,XXXX_12345678_YYY,String;Category.attribute1,1024,String</value> </Parameter> </Parameters> </TaskResponse> </body> </resp_msg>
Result:	<?xml version="1.0" encoding="UTF-8" standalone="yes"?><ServiceExecutionReport xmlns:ns2="http://www.hp.com/sosa/protocoladapter/ngws"><ns2:ssid>0.8783977740755039</ns2:ssid><ns2:userid>foo</ns2:userid><ns2:sosaCode>0</ns2:sosaCode><ns2:... Basic service call response</ns2:description><ns2:creationTime>12:39:10.238+05:30</ns2:creationTime><ns2:startTime>12:39:10.238+05:30</ns2:startTime><ns2:finishTime>12:39:10.238+05:30</ns2:finishTime><ns2:creationDate>2014-06-19+05:30</ns2:creation... <ns2:outputParams><ns2:param><ns2:name>ArtifactInstanceId</ns2:name><ns2:value>XXXX_12345678_YYY</ns2:value></ns2:param><ns2:param><ns2:name>Category.atribute1</ns2:name><ns2:value>1024</ns2:value></ns2:param></ns2:outputParams></Se...
Start Time:	19-Jun-14 12:39:09 PM
End Time:	19-Jun-14 12:39:10 PM

Figure 71: Action reporting snapshot


```
<?xml version="1.0" encoding="UTF-8" ?>
- <Alarms xmlns="http://hp.com/uca/expert/x733Alarm">
  - <AlarmCreationInterface xmlns="http://hp.com/uca/expert/x733Alarm">
      <identifier>17:ba2f6781-7187-406f-8035-5976215f9cb0</identifier>
      <sourceIdentifier>NFVD_Source</sourceIdentifier>
      <alarmRaisedTime>2016-01-05T01:26:00Z</alarmRaisedTime>
      <targetValuePack>SNMP-Customization-Sitescope-FlowTarget</targetValuePack>
      <originatingManagedEntity>VIRTUAL_MACHINE:GENERIC VM_IT</originatingManagedEntity>
    - <originatingManagedEntityStructure>
        <classInstance instance="SiteScope\HP\NFVDirector\demo\VNF_IT\VNFC_IT\VM_IT\artifactId-740274eb-bc8b-48f3-ab20-6757fc8332d6\VIRTUAL_MACHINE
          VMWARE CPU\225dd3fc-db7b-45c8-b413-15aefd73cfda" clazz="VMware" />
      </originatingManagedEntityStructure>
      <alarmType>QUALITY_OF_SERVICE_ALARM</alarmType>
      <probableCause>THRESHOLD_CROSSED</probableCause>
      <perceivedSeverity>CLEAR</perceivedSeverity>
      <networkState>CLEARED</networkState>
      <operatorState>NOT_ACKNOWLEDGED</operatorState>
      <problemState>NOT_HANDLED</problemState>
      <specificProblem>cpu_usage_average</specificProblem>
      <additionalText>errorOnly=|goodOnly= cpu_usage_average: 0|warningOnly=|group=VIRTUAL_MACHINE VMWARE CPU|groupdescription=
        (MONITOR_ARTIFACT_ID=53529662-fd60-4c25-84bd-3fce0dca119d)(MONITOR_TYPE=CPU)(MONITORED_ENTITY_ID=740274eb-bc8b-48f3-ab20-6757fc8332d6)
        (MONITORED_ENTITY_FAMILY=VIRTUAL_MACHINE)(MONITORED_ENTITY_CATEGORY=GENERIC)(MONITORED_ENTITY_NAME=VM_IT)
        (VM_HYPERVISOR_ID=225dd3fc-db7b-45c8-b413-15aefd73cfda)(counteName=cpu_usage_average) |id=1|monitorUUID=a5a032e9-506d-4027-9a11-
        2965f1f00923|monitorTypeDisplayName=VMware|fullMonitorName=SiteScope\HP\NFVDirector\demo\VNF_IT\VNFC_IT\VM_IT\artifactId-740274eb-bc8b-48f3-ab20-
        6757fc8332d6\VIRTUAL_MACHINE VMWARE CPU\225dd3fc-db7b-45c8-b413-15aefd73cfda|state=VirtualMachine/225dd3fc-db7b-45c8-b413-
        15aefd73cfda/Realtime/cpu/usage.average[]=0%, cpu_usage_average=0|time=1:26 AM
        1/5/16|templateDeployPath=HP/NFVDirector/demo/VNF_IT/VNFC_IT/VM_IT/artifactId-740274eb-bc8b-48f3-ab20-6757fc8332d6/VIRTUAL_MACHINE VMWARE
        CPU</additionalText>
  </AlarmCreationInterface>
</Alarms>
```


Customizations

10.10.5 Custom monitoring

SiteScope provides the ability to develop monitoring templates to monitor KPIs for a specific resource.

The procedure to develop customized SiteScope templates is described in the SiteScope user documents.

Once the required SiteScope template is developed, follow these steps to integrate it with the NFV Director:

- Import the SiteScope template into SiteScope.
- Modify the SiteScope template to send SNMPv2 traps to the OM bus.

- Add monitor artifact to the VNF template (using VNF designer).
- Deploy the VNF.

Once the custom monitors are imported into SiteScope and the VNF template is updated to use these new monitors, event collection and automated actions can be performed on the new custom VNF monitors. Invoking the VNF CREATE API deploys and activates the new custom monitors as part of the `VNF CREATE` process.

10.10.5.1 Creating a custom SiteScope template

NFV is a complex system that requires constant monitoring of the physical and the logical entities. The provisioning and monitoring functions can be combined through rules that define manual or autonomous scaling and placement actions, based on measured key performance indicators (KPIs).

To resolve this, the HPE NFV Director includes the agentless monitoring component. Built using HPE SiteScope, it can monitor a wide variety of monitoring points, issuing events, or executing commands when predefined thresholds are crossed. Since different virtual network functions have different monitoring needs, the monitoring points and thresholds are automatically configured by the HPE NFV Director as the VNF is provisioned or modified. As an agentless solution, the HPE NFV Director does not require the installation of monitoring agents on the target systems.

Custom templates broaden the capabilities of the regular SiteScope monitors other than the NFVD supported monitors. They help track availability and performance of monitored environments. The custom templates enable you to create your own monitor by using any existing monitors provided by SiteScope.

Follow these steps to create custom templates:

1. Select the templates context.
2. Right-click the SiteScope root node from the tree and select **New > Template Container**.

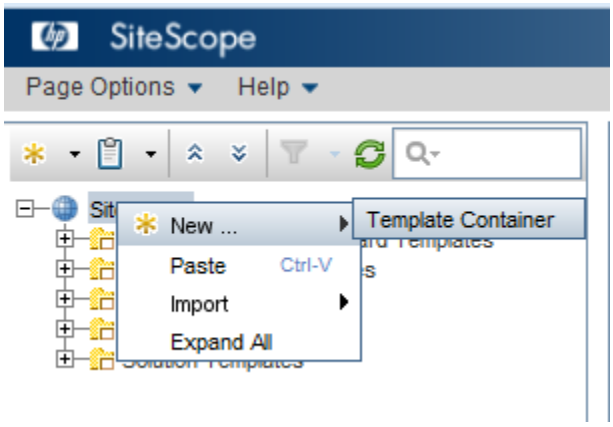


Figure 72: SiteScope - Create custom templates

3. Enter the name of the template container and click **OK**.
4. Right-click this new template container node and select **New > Template**.

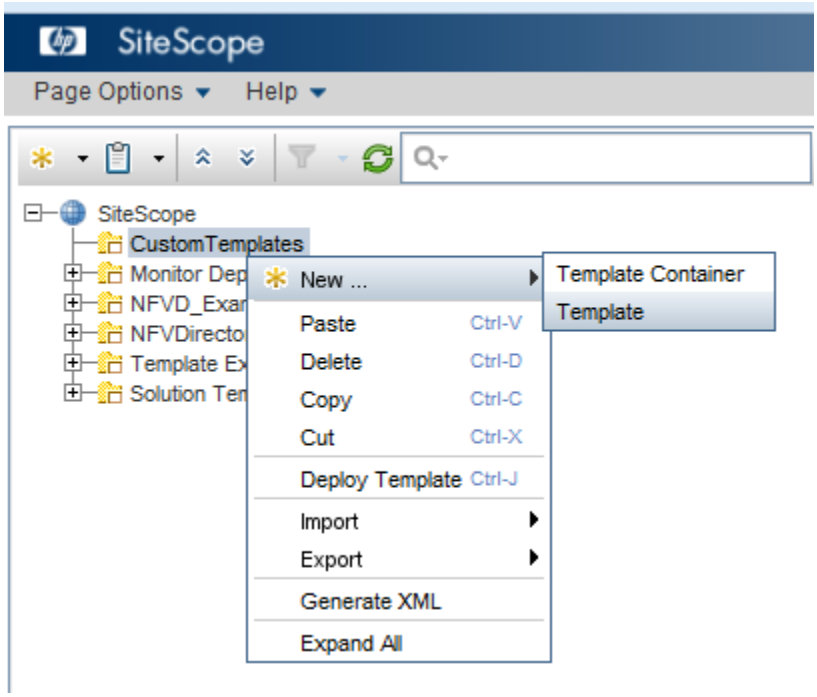


Figure 73: SiteScope - New template

5. Enter the name of the template and click **OK**.

NOTE: The example template container created in this section is called `Custom Templates`.

6. Right-click this new template node and select **New > Variable**.

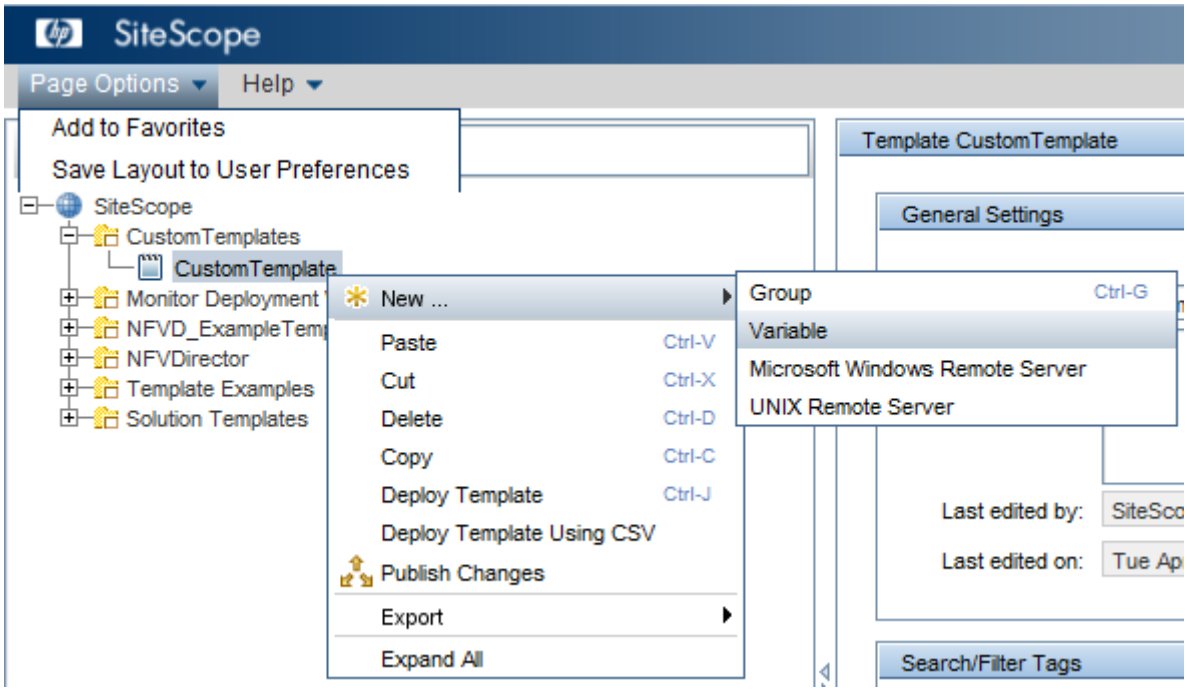


Figure 74: SiteScope - Create a new template variable

7. A pop-up will appear. Enter `groupDescription`. This variable is completed by the Assurance gateway and is mandatory if the templates need to be part of the scenario that uses the standard PD Value Pack provided out of the box.

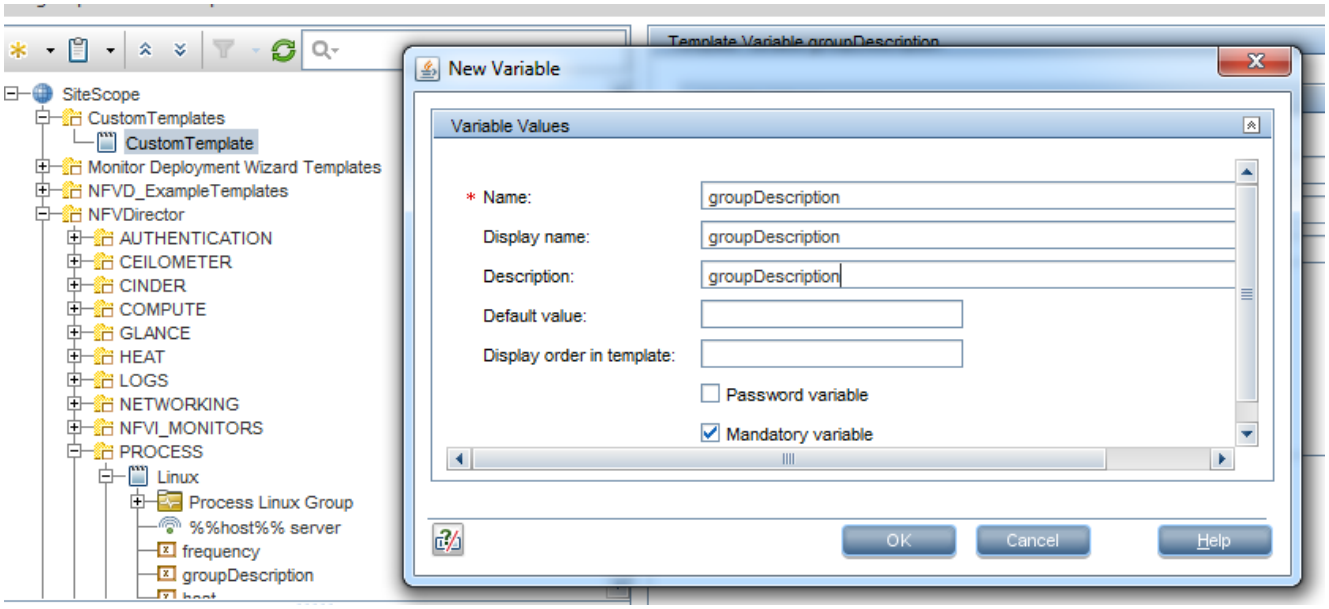


Figure 75: SiteScope - Enter groupDescription as new template variable

8. Right-click this new template node and select **New > Group**.

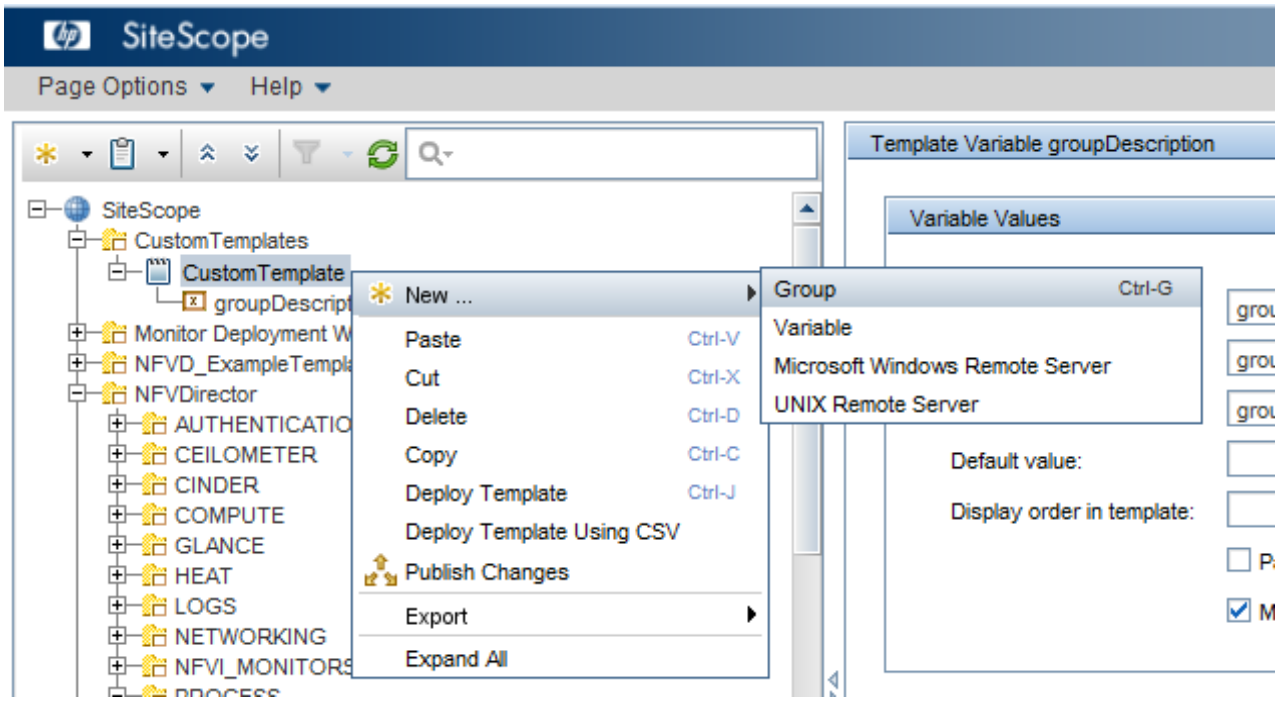


Figure 76: SiteScope - New group

9. Enter the name of the group – `groupDescription` – and click **OK**.

NOTE: The example template created in this section is called `CustomGroup`.

10. The `groupDescription` variable is configured in **Group description**. This is a variable completed by the NFVD Assurance plane. When alarms are triggered by SiteScope, this content will be part of the alarm, based on which correlation happens in the problem detection provided as part of the product. If the user wants to use some custom problem detection Value Pack, then this may not be required.
11. KPIs monitored by the templates are listed under `counterName` (name of the KPI), `displayname` (display name for GUI), `unit` (KPI measurement unit), and `description` (description of the KPI). This step is mandatory. If this step is not performed, then KPIs will not be displayed properly in the GUI, and KPI collection by the Assurance gateway will not happen. In the following illustration, there are two counters – counter1 and counter2. They would be captured when the monitor runs at regular frequency.

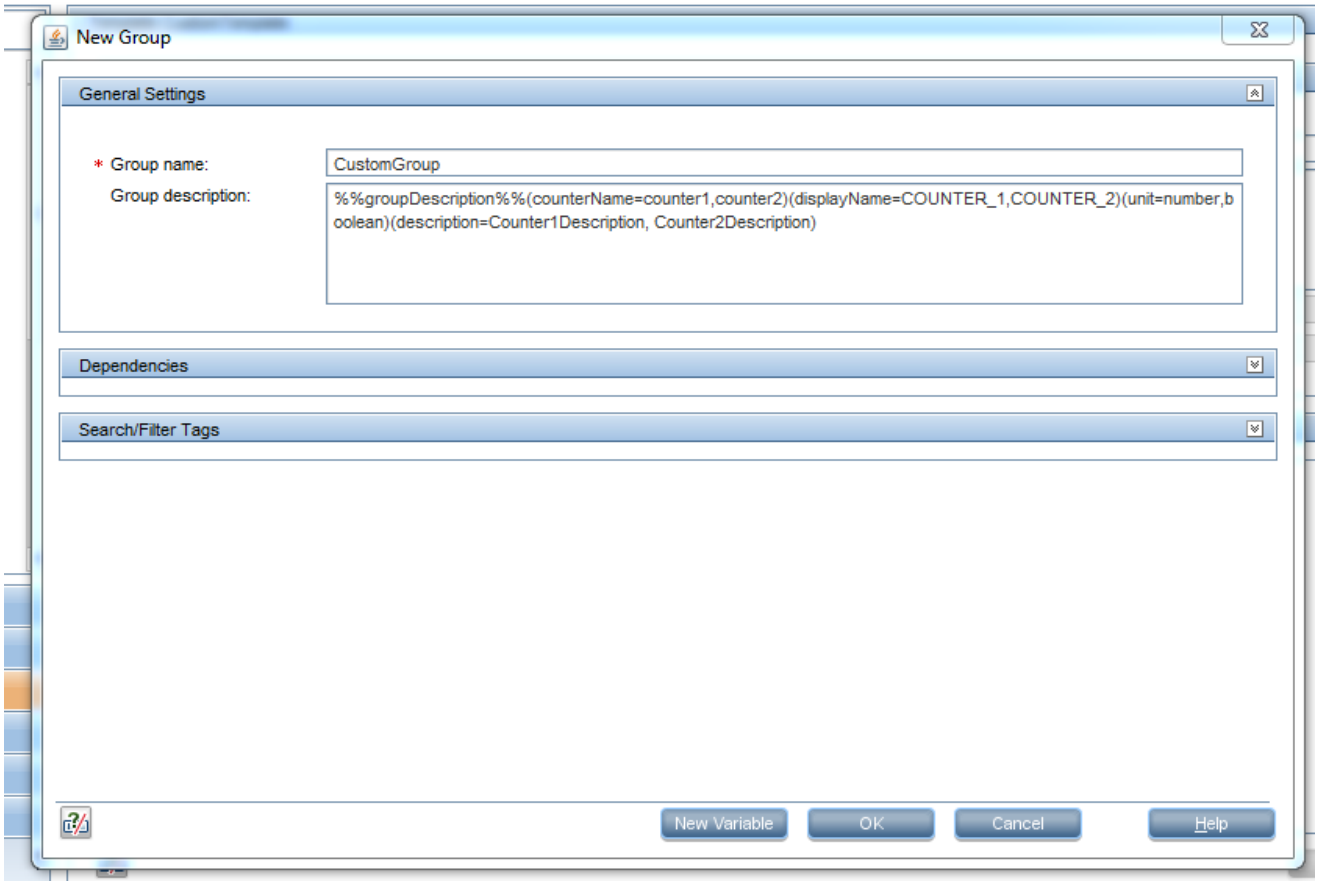


Figure 77: SiteScope - Create new group

12. Right-click this group node and select **New > Monitor**.

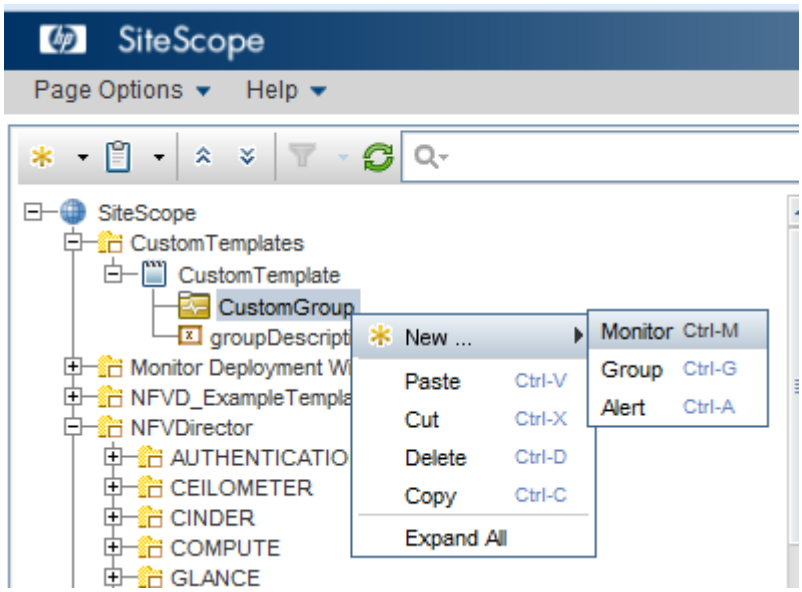


Figure 78: SiteScope - New Monitor

13. Select any of the monitors from the list.

NOTE: The example group created in this section is called `Custom Monitor`.

14. Repeat steps 6 and 7 to create all the required variables for the `Custom Monitor`.
15. Enter the details for configuring a variable to associate it with the template.

The following example describes how to configure the host variable.

After the variable is configured, it appears in the tree under the **Template** node. You can configure any number of variables. The following illustration shows how to use these variables in the specific monitor.

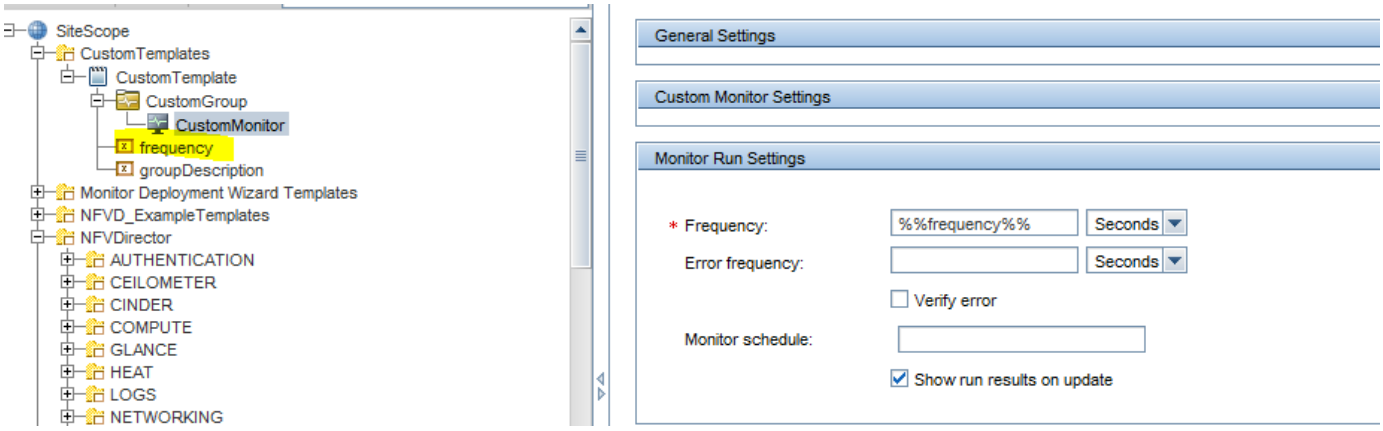


Figure 79: SiteScope - Enter variable to associate with template

The following illustration shows the complete hierarchy of the newly created Custom Template.

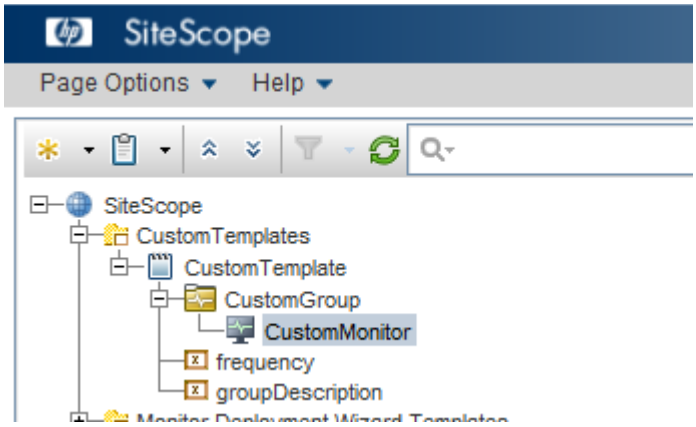
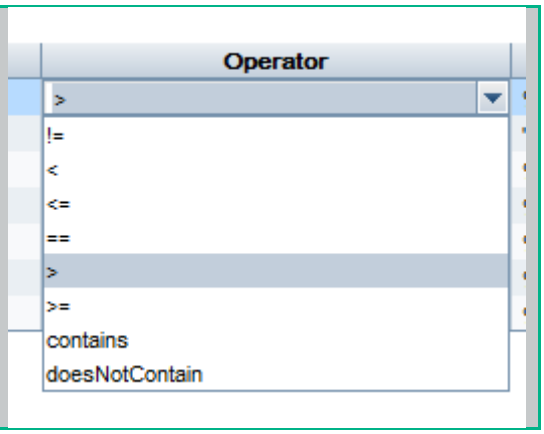


Figure 80: SiteScope – Custom template hierarchy

16. Create custom monitor variables for threshold conditions in the following format:
    a. Error Condition: `<variable>_error_<operator_type>`
    b. Warning Condition: `<variable>_warning_<operator_type>`
    c. Good Condition: `<variable>_good_<operator_type>`

Each of the operators supported by sitescope is mapped to the <operator_name> suffix, as shown in the below table:

Operator	Suffix to be used
>	gt
>=	gte
<	lt
<=	lte
==	ee
!=	ne
contains	con
doesNotContain	dnc



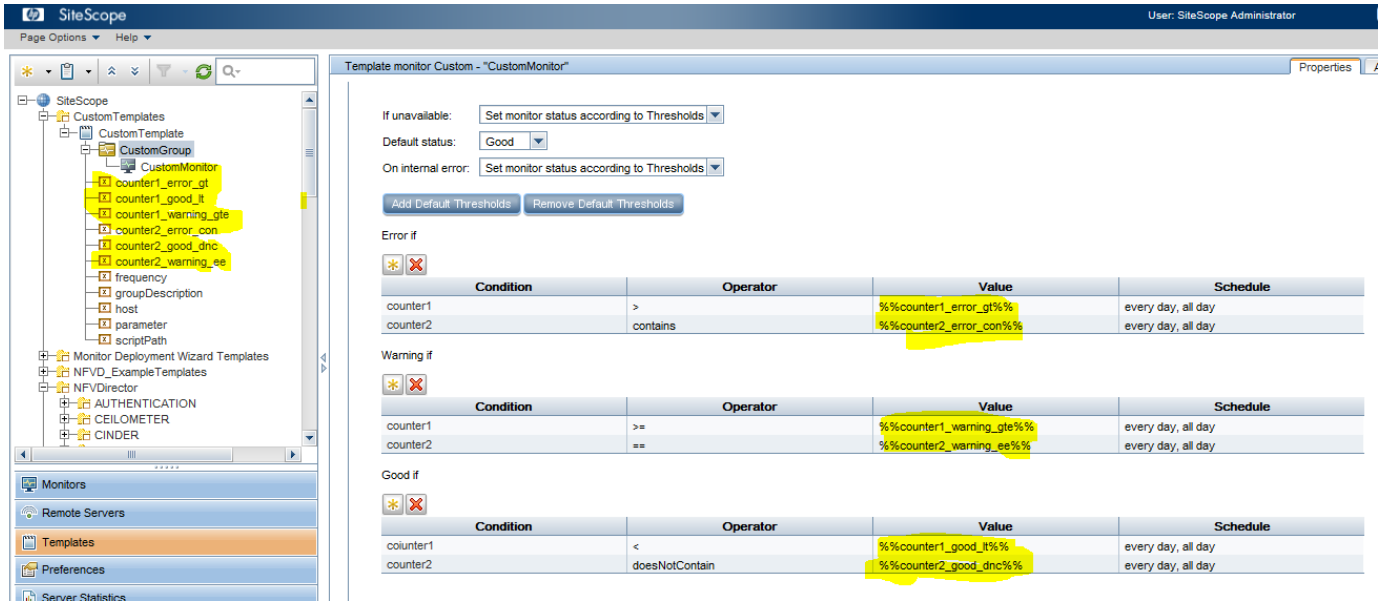


Figure 81: SiteScope - Create custom monitor variables

NOTE: In Step 12, we have provided the `counterName` as `counter1` and `counter2`. In the Condition artifact used during orchestration, we would configure the condition expression as indicated on the following illustration. During monitor deployment, the variable `counter1_error_gt` will have a value of `90`.

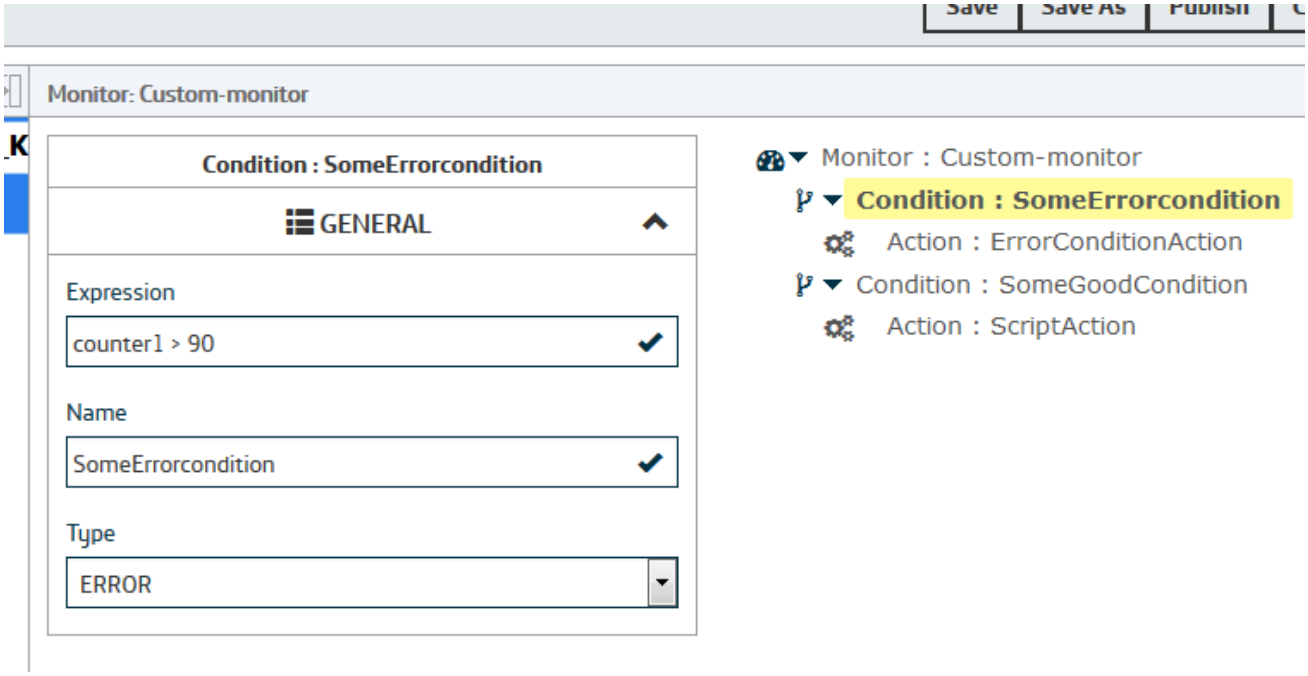


Figure 82: SiteScope - Configure condition expression

10.10.5.2 Creating a custom monitor palette for GUI

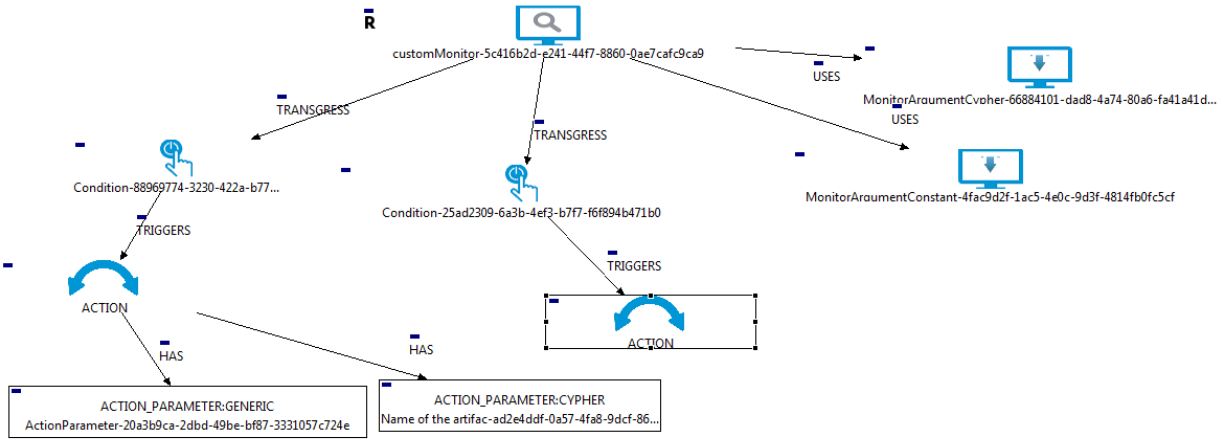


Figure 83: Create a new custom monitor palette in the GUI

10.10.5.2.1 Create new connector for standard base VM

NOTICE: The standard base VM palette is available in:

```
/opt/OV/ServiceActivator/solutions/NFVModel/etc/LoadXML/COMPONENTS/VNFC-D-vm-basic-palette.nfvd
```

Create a new VM component with connection to the custom monitor.

1. Open the Resource Modeler tool and open the standard base VM palette.
2. Select the **Components** tab, select the component, and list the **Connectors**.

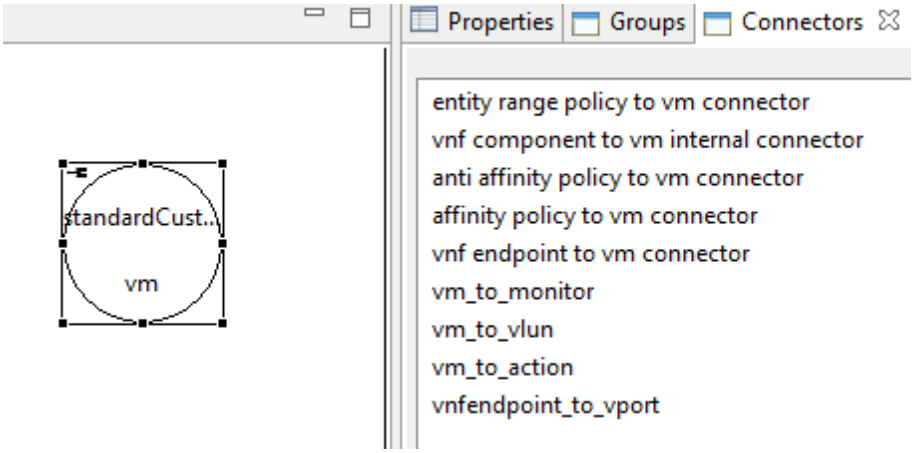


Figure 84: Standard VM palette connectors

3. Create a new connector to connect to `MONITOR:CUSTOM` and name it `vm_to_custommonitor`.

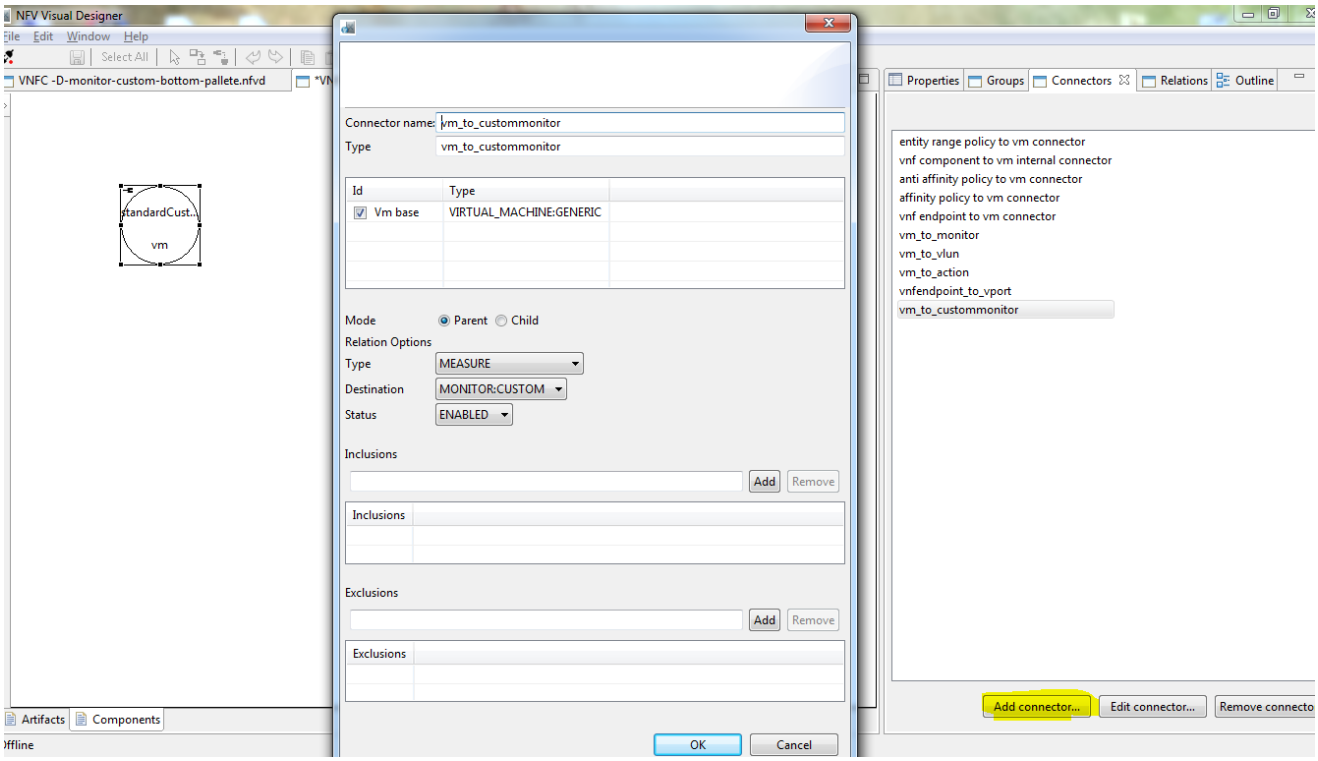


Figure 85: Create new connector for standard VM palette

4. Rename the group.

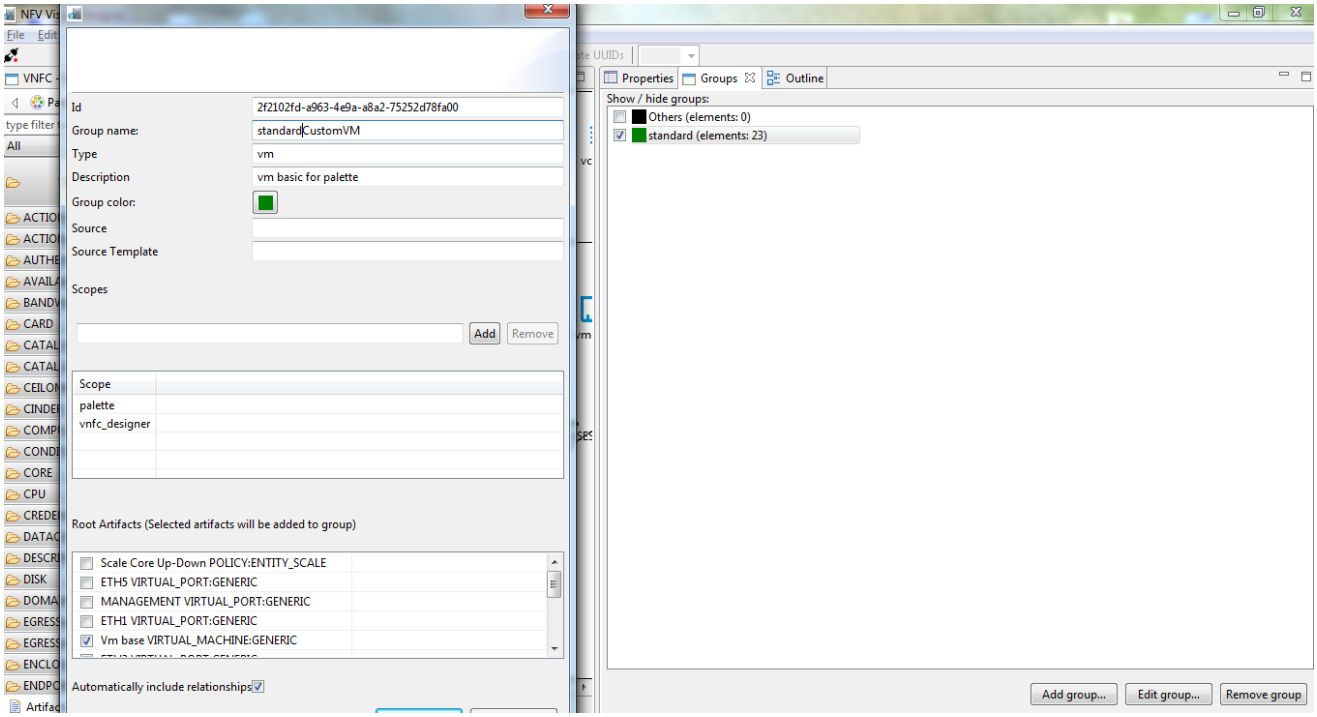


Figure 86: Rename the group

5. Regenerate the UUIDs. This is required so that the standard VM component is not replaced, rather a new VM component with a custom monitor is added. Click **Regenerate UUIDs**.

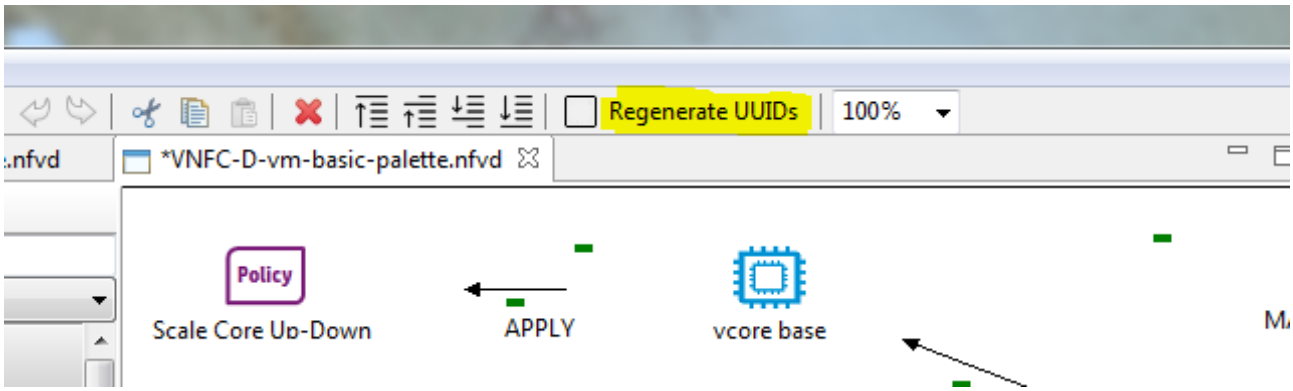


Figure 87: Regenerate UUID

6. Export the created custom VM palette.
   a. Make sure you are in offline mode.

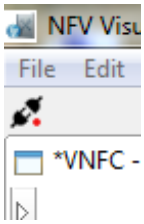


Figure 88: Export VM palette in offline mode

b. Click **File > Export**.
c. Select the Export NFVDiagram to XML wizard. Click **Next**.
d. Select the template and include all the items as indicated on the following illustration. Click **Finish** to create a loadable XML.

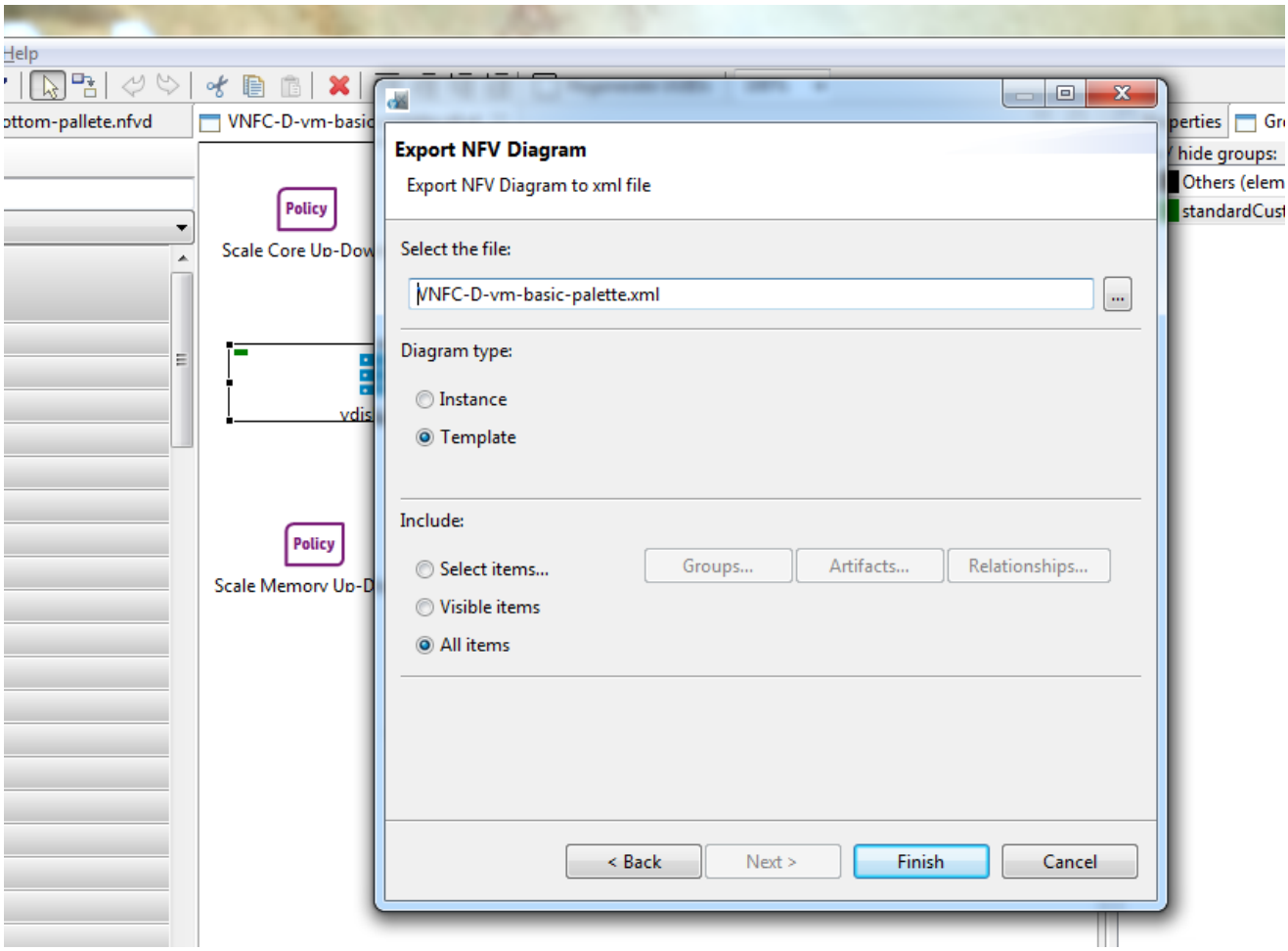


Figure 89: Export custom VM palette

e. Load the contents of the XML file using the Fulfilment Rest API.

```
http://
METHOD: POST
Content-Type: application/xml
X-Auth-Token: (Needs to be obtained by using FF rest API for a user)
```

7. After uploading the palette, the user can log into the GUI and see the new standardCustomVM.

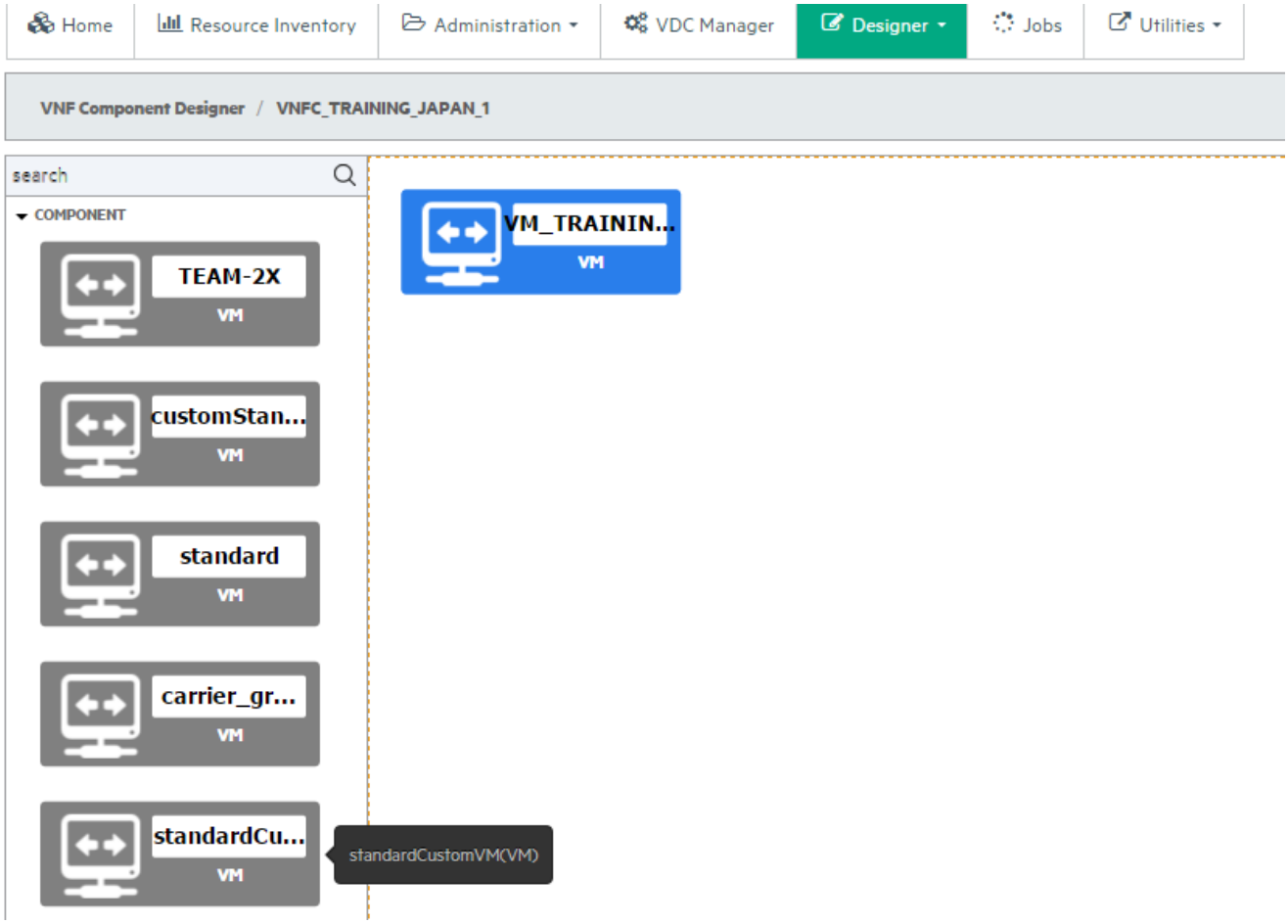


Figure 90: Search for custom VM in the GUI

10.10.5.2.2 Creating new custom monitor palette

Follow these steps to create a new custom monitor palette:

1. Open the Resource Modeler and create a new NFV Director palette.
   a. Click **File > New**.
   b. Choose NFV Diagram wizard and click **Next**.

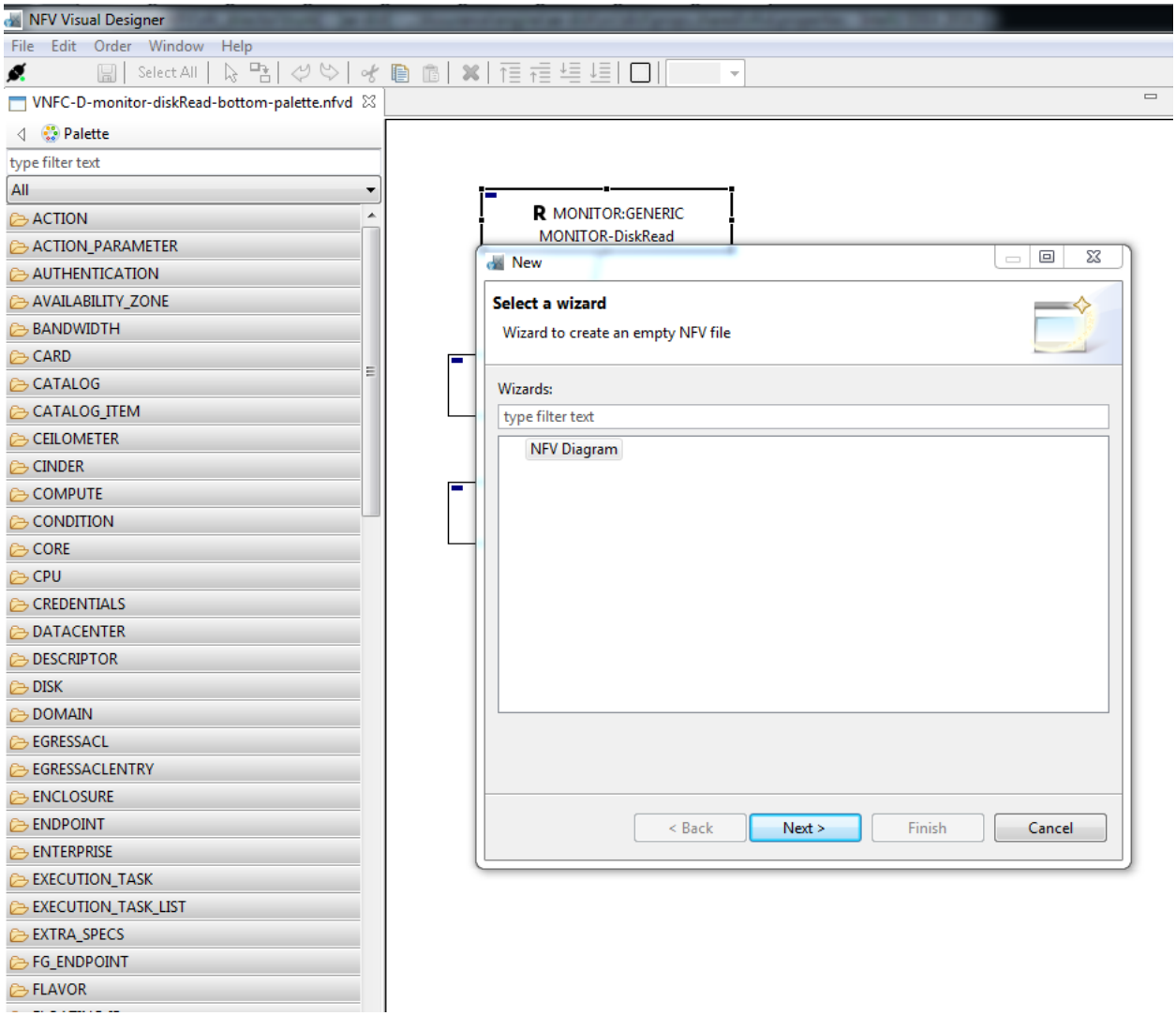


Figure 91: Launch NFV Visual Designer wizard

c. Choose the location to store the NFVD palette.
d. Click **Finish**.

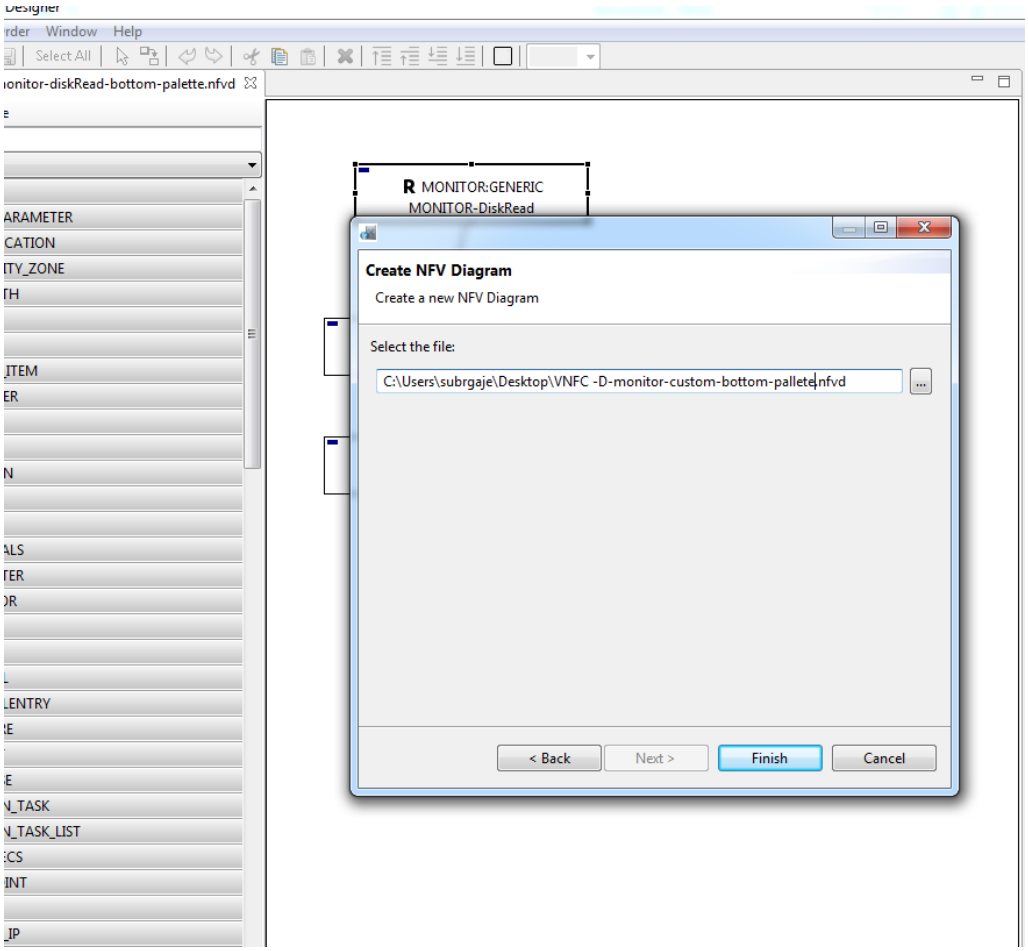


Figure 92: Name the new NFV Diagram

2. Drag `MONITOR:CUSTOM` from the definitions on the left.

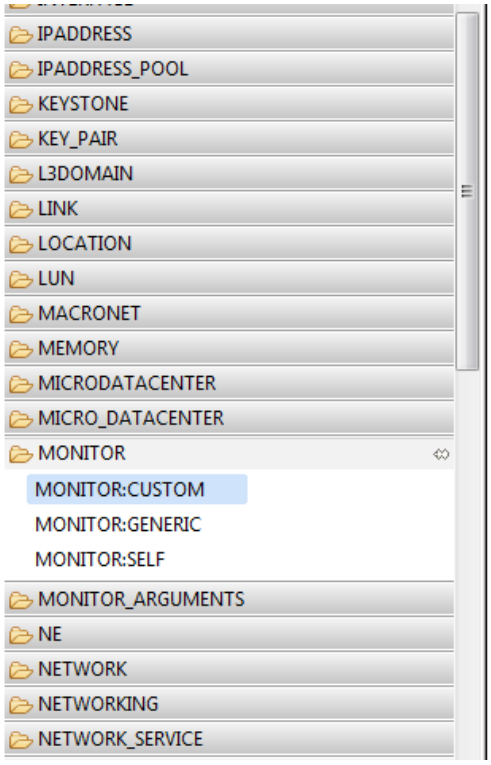


Figure 93: Drag MONITOR:CUSTOM from definitions

3. Configure the following attributes for the `MONITOR:CUSTOM` artifact:

- `TemplatePath`
- `DeploymentPath`
- `Name`
- `Frequency`
    a. Locate the `TemplatePath` attribute under the **DEPLOYMENT** category on the right.
    b. Configure the template path in SiteScope. In this case the template path is hardcoded to the one created in the previous section.

CAUTION: This is a limitation in V4.0. The template path cannot be edited in the online designer and the user must create separate monitor palettes for each custom SiteScope template.

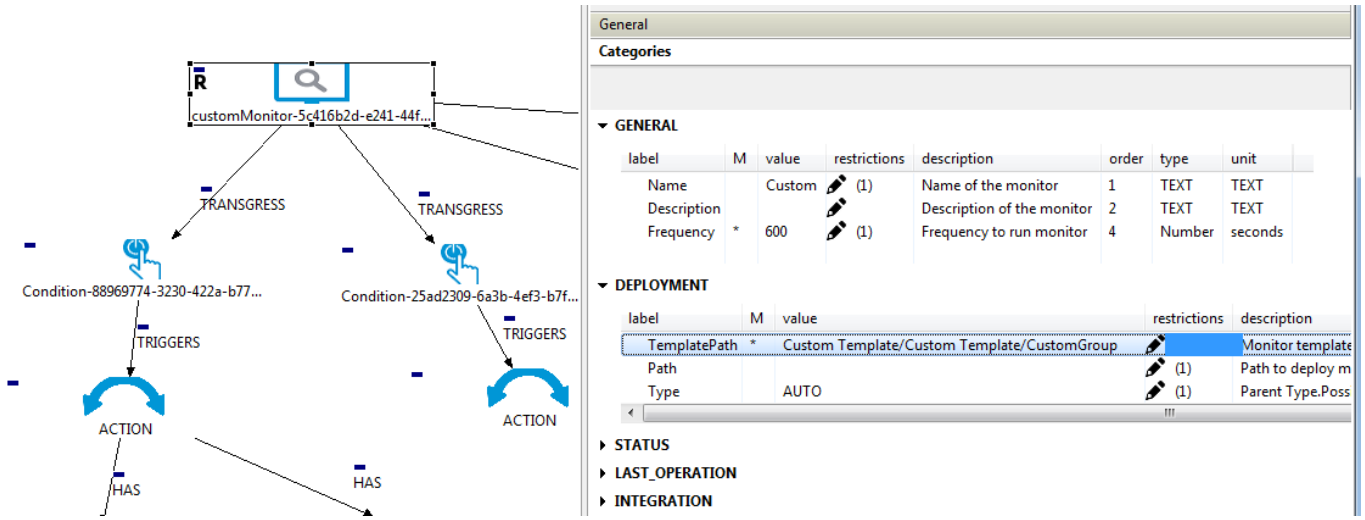


Figure 94: SiteScope - Configure template path

4. Add the following restrictions:

CAUTION: Make sure the restrictions have been entered correctly, because errors prevent the system from loading the restrictions.

Restriction	Values
`type`	`user-input`

`visible`	`vnfc_designer`
	
	`vnf_designer`
	
	`vdc_manager`

`values`	Values depend on type.

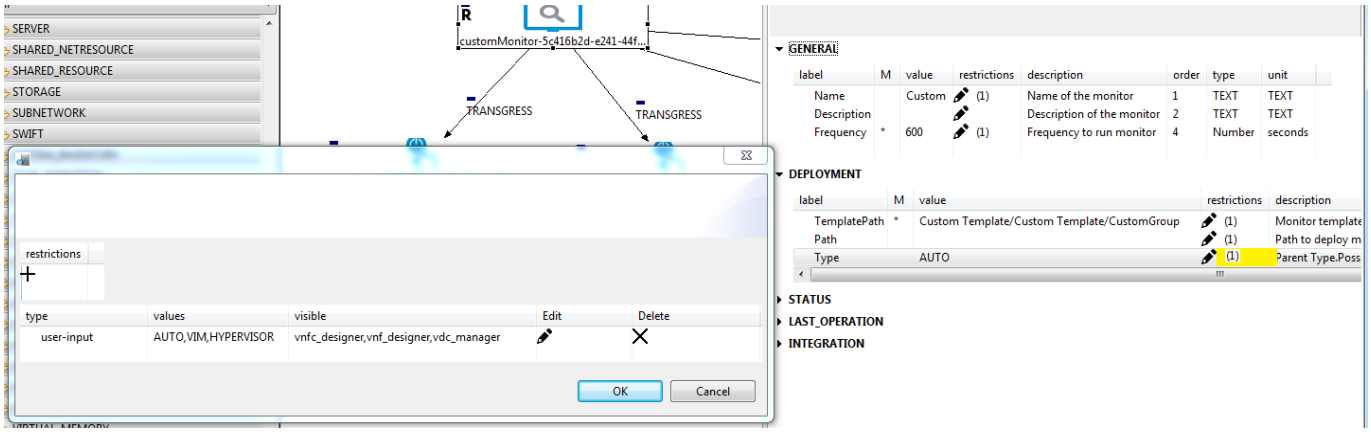


Figure 95: Add variable restrictions

a. Add restrictions for Name and Frequency under the General category.

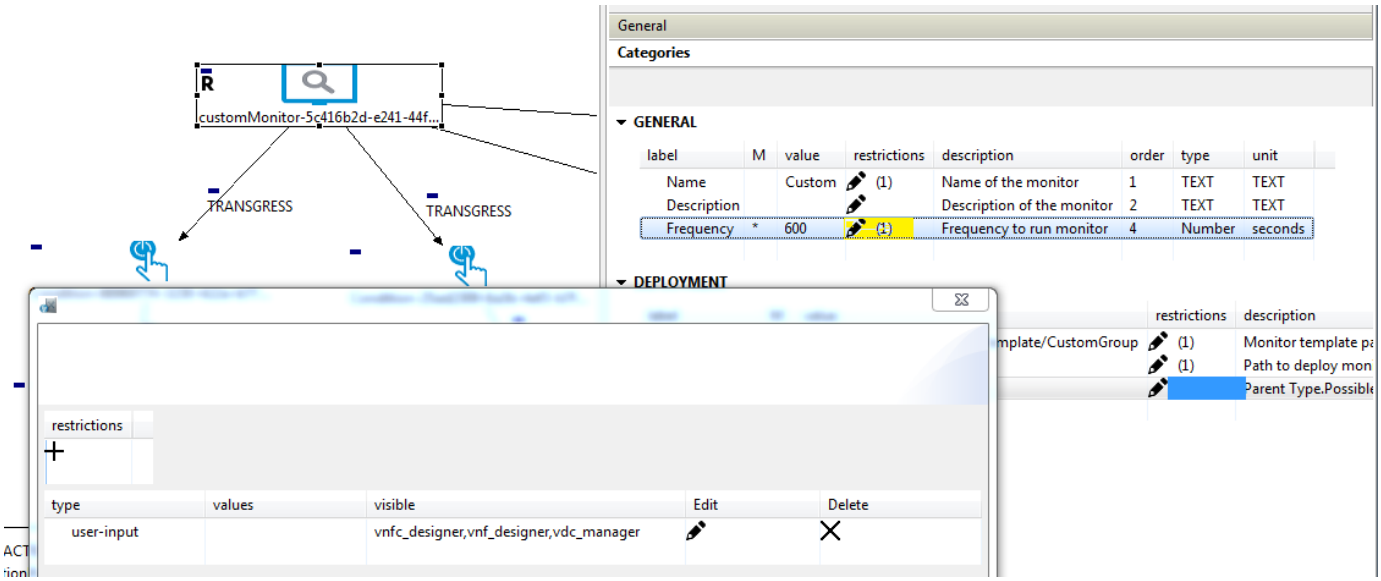


Figure 96: Add restrictions in General category

CAUTION: The next step is a limitation in V4.0.

5. Drag `MONITOR_ARGUMENTS:ARGUMENTS` and configure the restrictions according to the following illustrations. The configuration on the following illustrations would enable the user to configure constant name-value pairs for monitor deployment.
   a. Monitor arguments are not visible in the online designer. The user needs to hardcode the name-value pairs in the offline designer and then load them.
   b. Monitor arguments cannot be added in the online designer. They need to be added as part of the monitor palette in the offline designer and then uploaded. The user must create new monitor palettes for any new monitor argument or different name-value pairs.

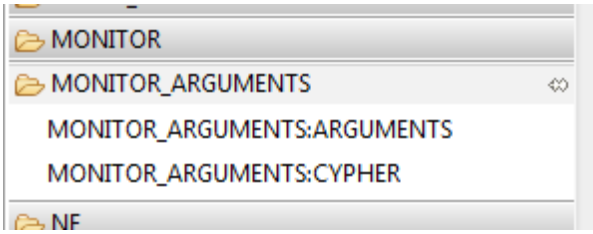


Figure 97: MONITOR_ARGUMENTS

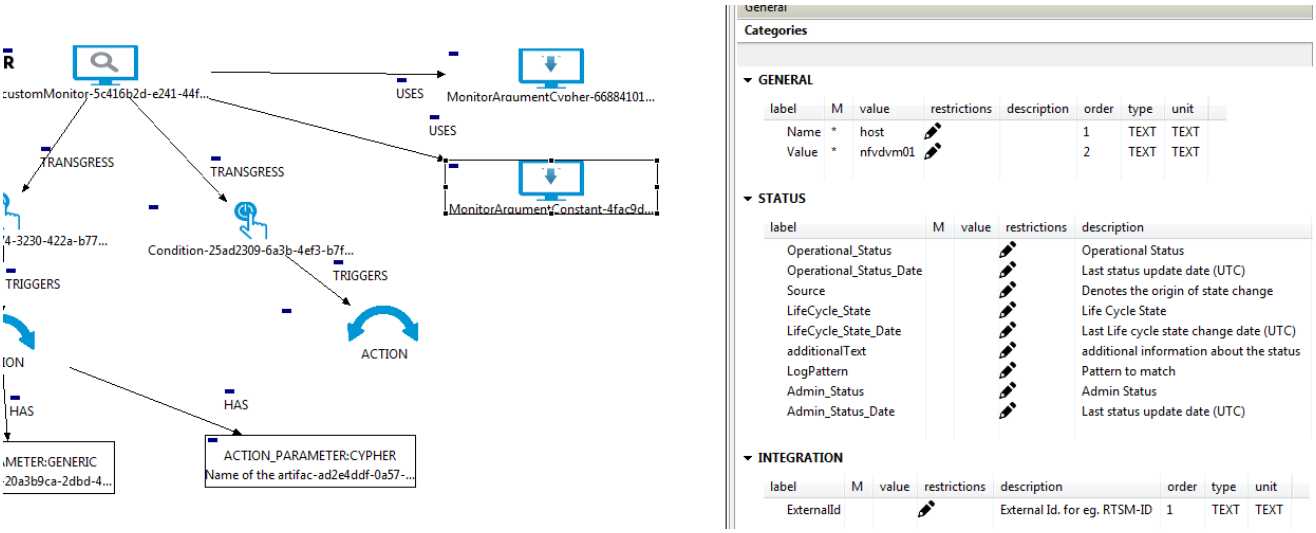


Figure 98: Create new monitor palettes

6. Drag `MONITOR_ARGUMENTS:CYPHER` and configure the restrictions according to the following illustrations. The configuration on the following illustration would enable the user to configure a cypher that can be used for monitor deployment. The limitations from the previous step are also in effect.

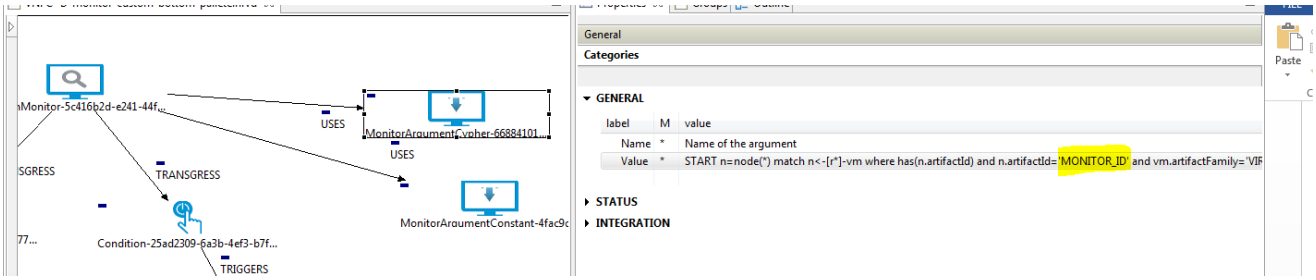


Figure 99: Drag MONITOR_ARGUMENTS:CYPHER and configure restrictions

7. Create relationships between `MONITOR:CUSTOM` to `MONITOR_ARGUMENTS:ARGUMENTS` and `MONITOR_ARGUMENTS:CYPHER`.

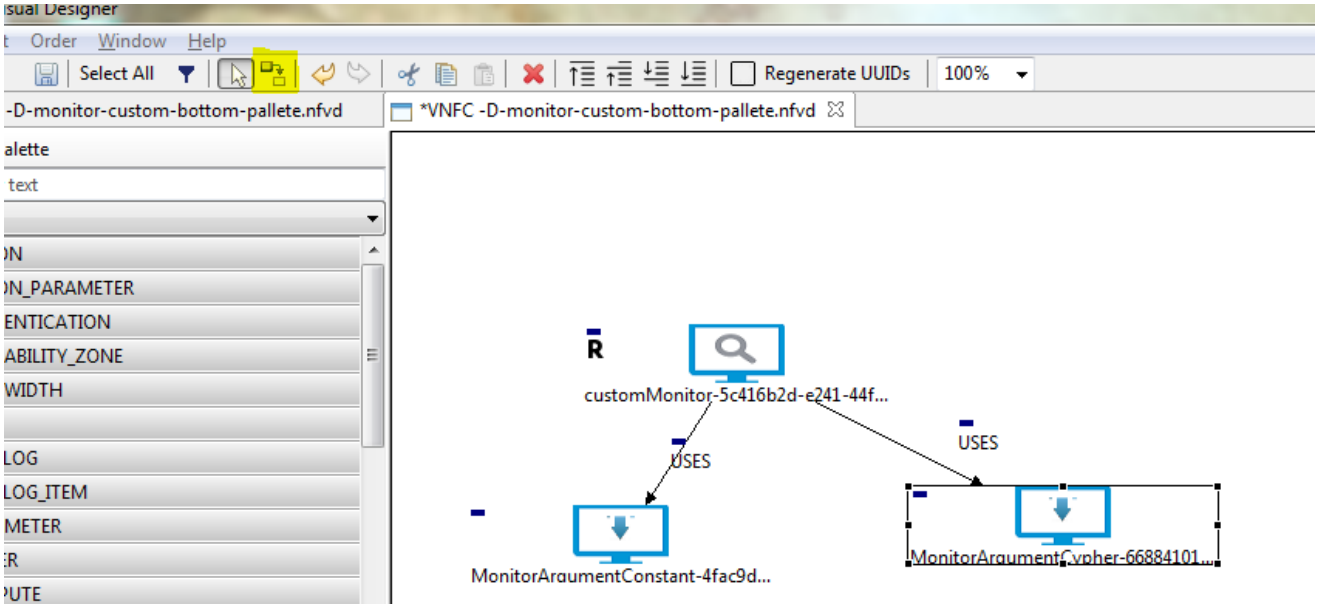


Figure 100: Create relationships between MONITOR_CUSTOM and MONITOR_ARGUMENTS:ARGUMENTS

8. Repeat steps 5-7 to create two conditions. Configure restrictions for both conditions according to the following illustration. Possible condition type values are:
   - `ERROR`
   - `WARNING`
   - `GOOD`

Condition expression is a free user text.

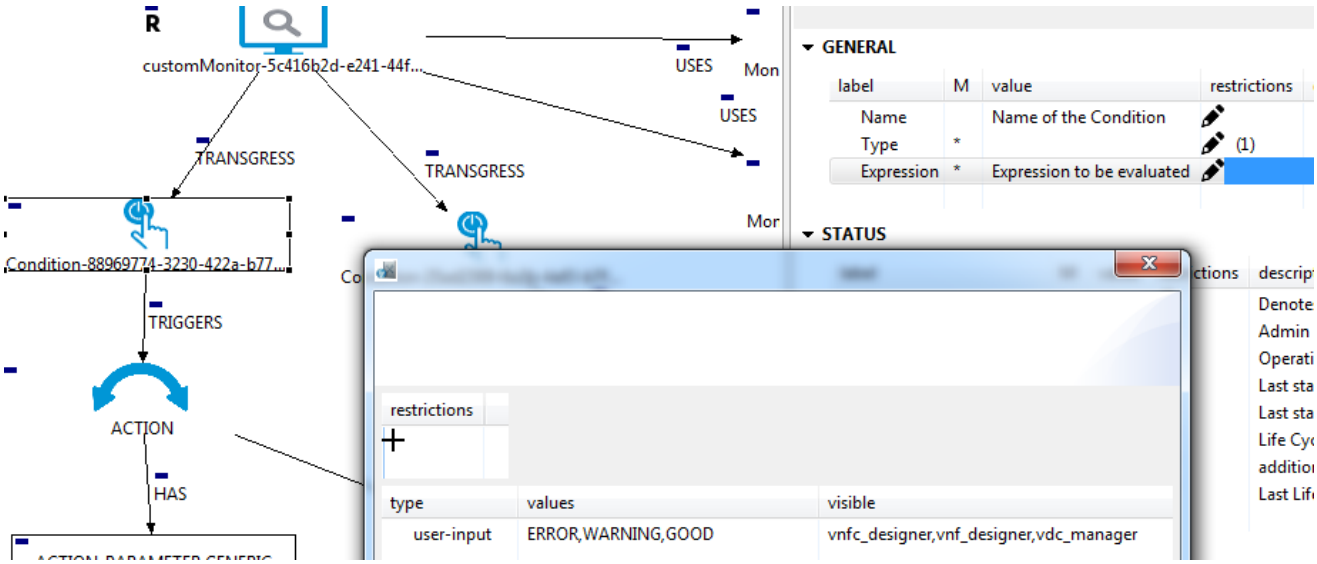


Figure 101: Adding restrictions for conditions

9. Create two actions and configure the user restrictions as indicated on the following illustration:

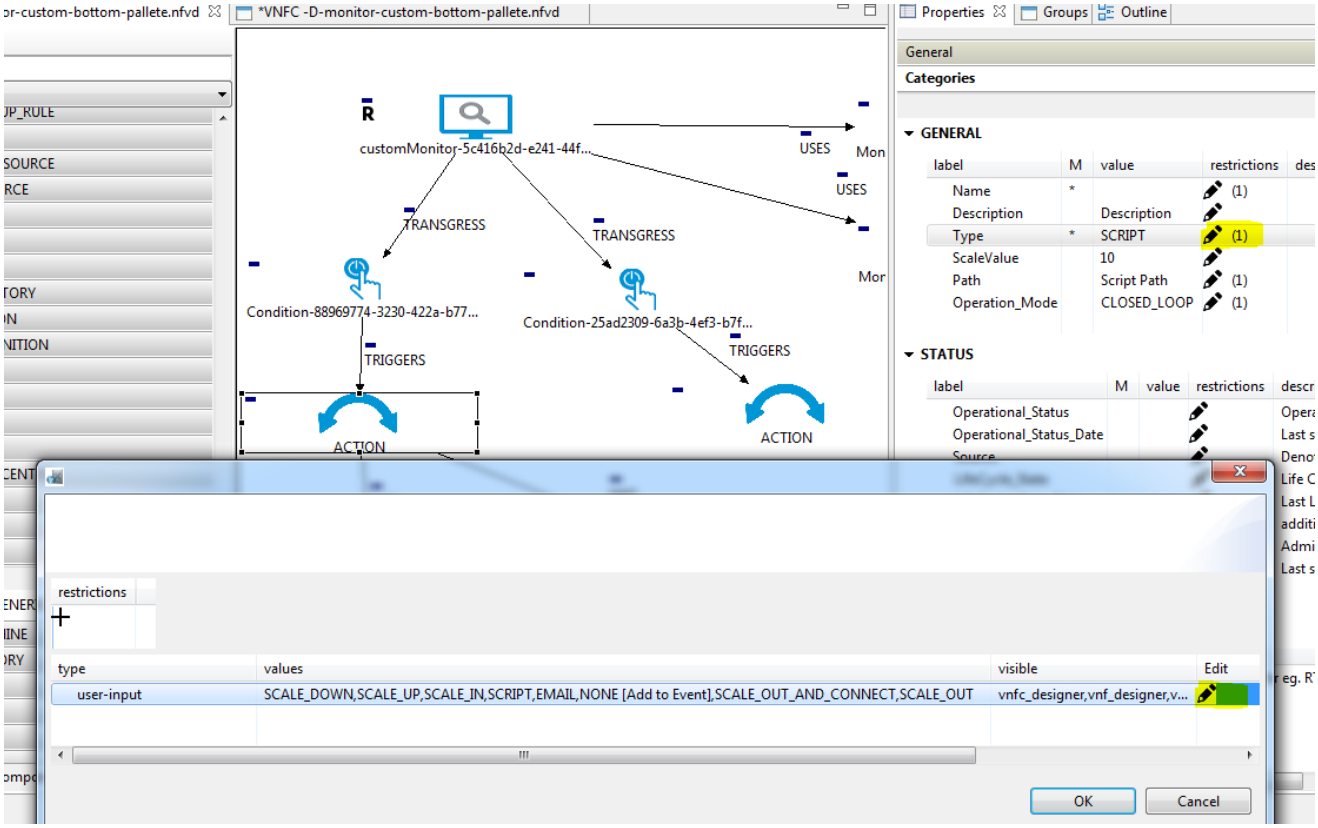


Figure 102: Adding restrictions for actions

CAUTION: The next step is a limitation in V4.0. See Step 5.

10. Create the action parameters by following the steps required to create `MONITOR_ARGUMENTS`.

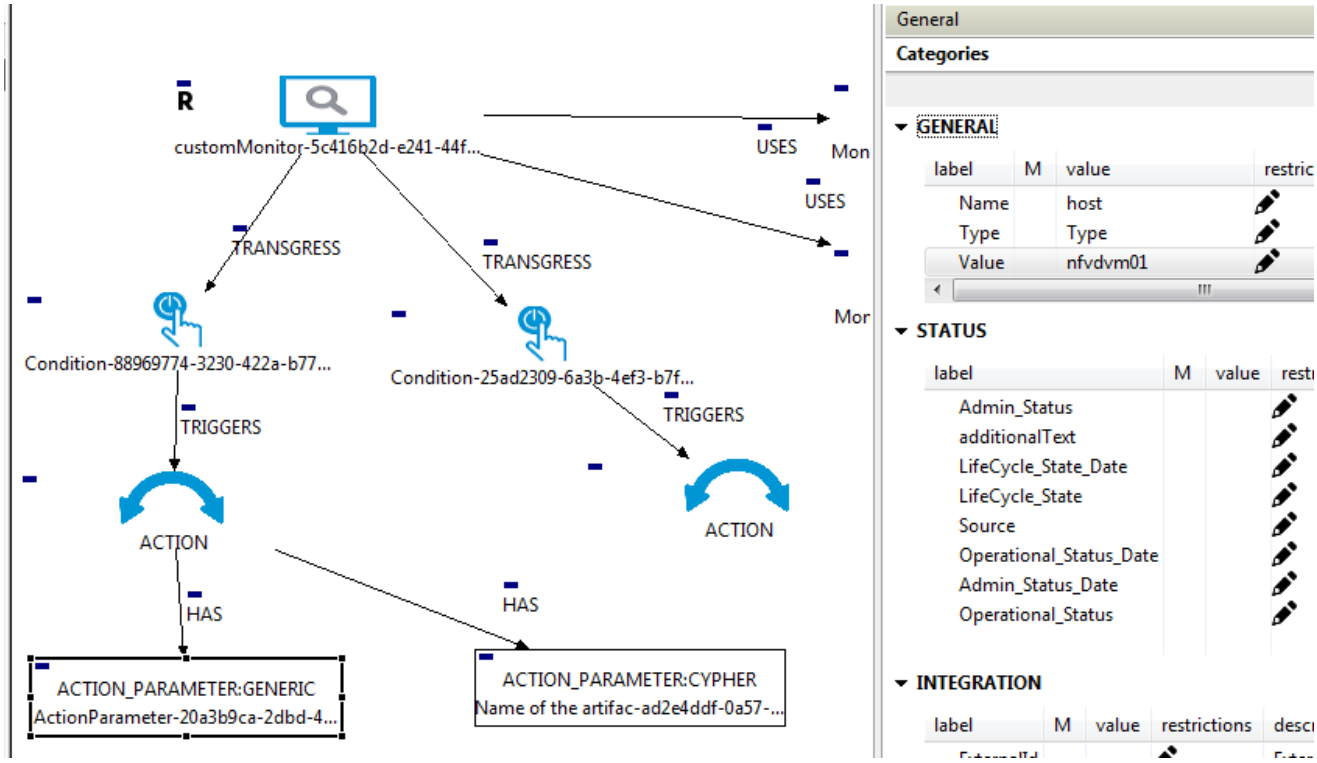


Figure 103: Create action parameters

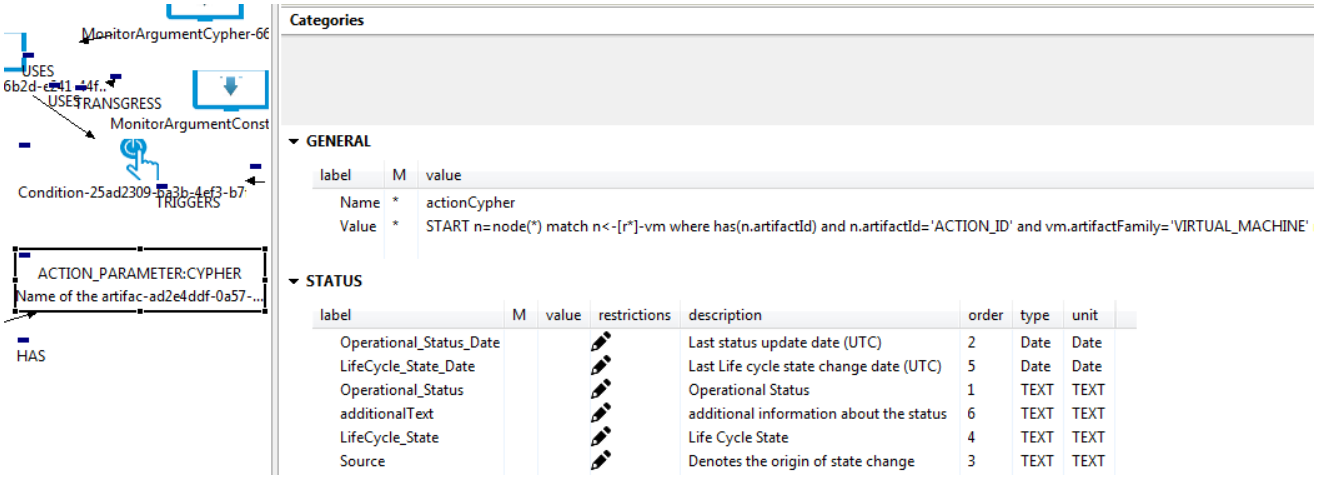


Figure 104: Create action parameter categories

11. Create a new group. Click the **Group** tab in the right pane and choose the `MONITOR:CUSTOM` artifact as the root artifact.

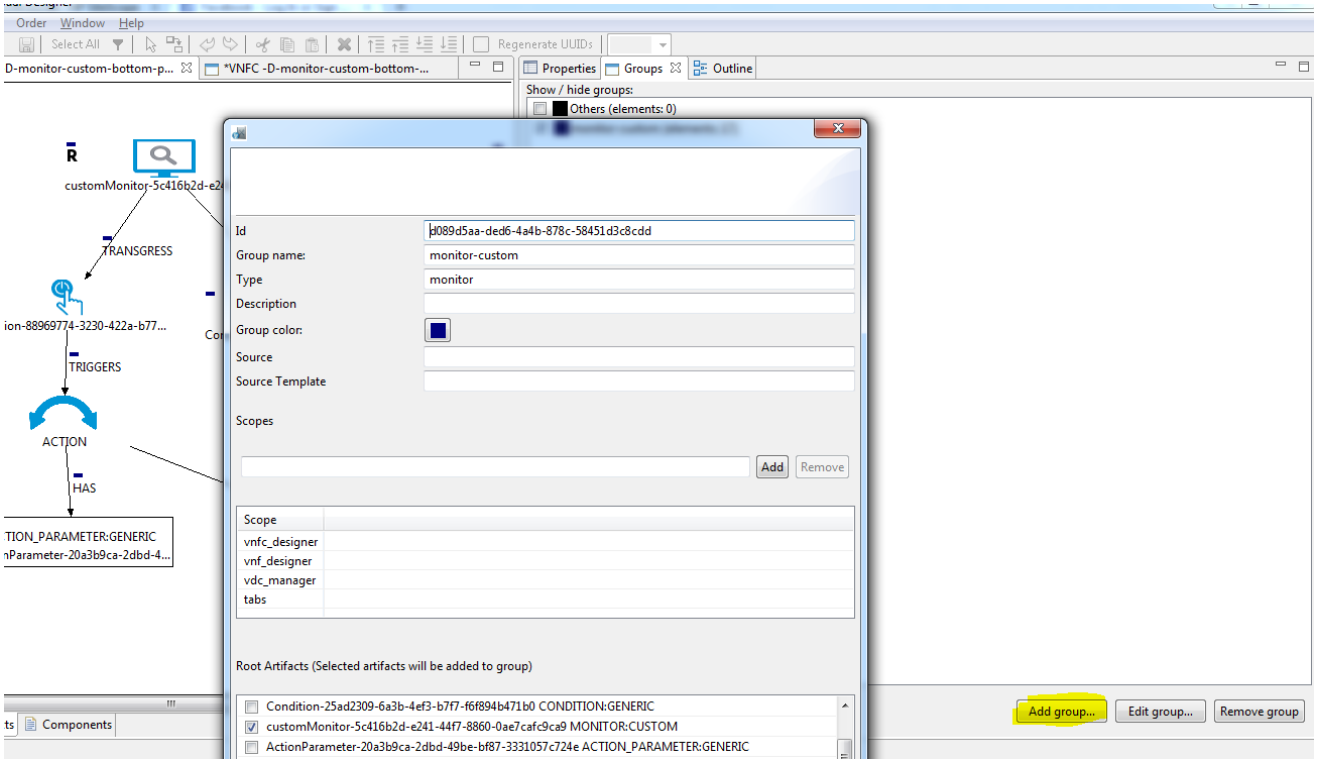


Figure 105: Create a new group for MONITOR:CUSTOM

12. Add all components to the new group.
    a. Select all components, for example with the **Ctrl+A** keyboard shortcut.
    b. Right-click and select the group.

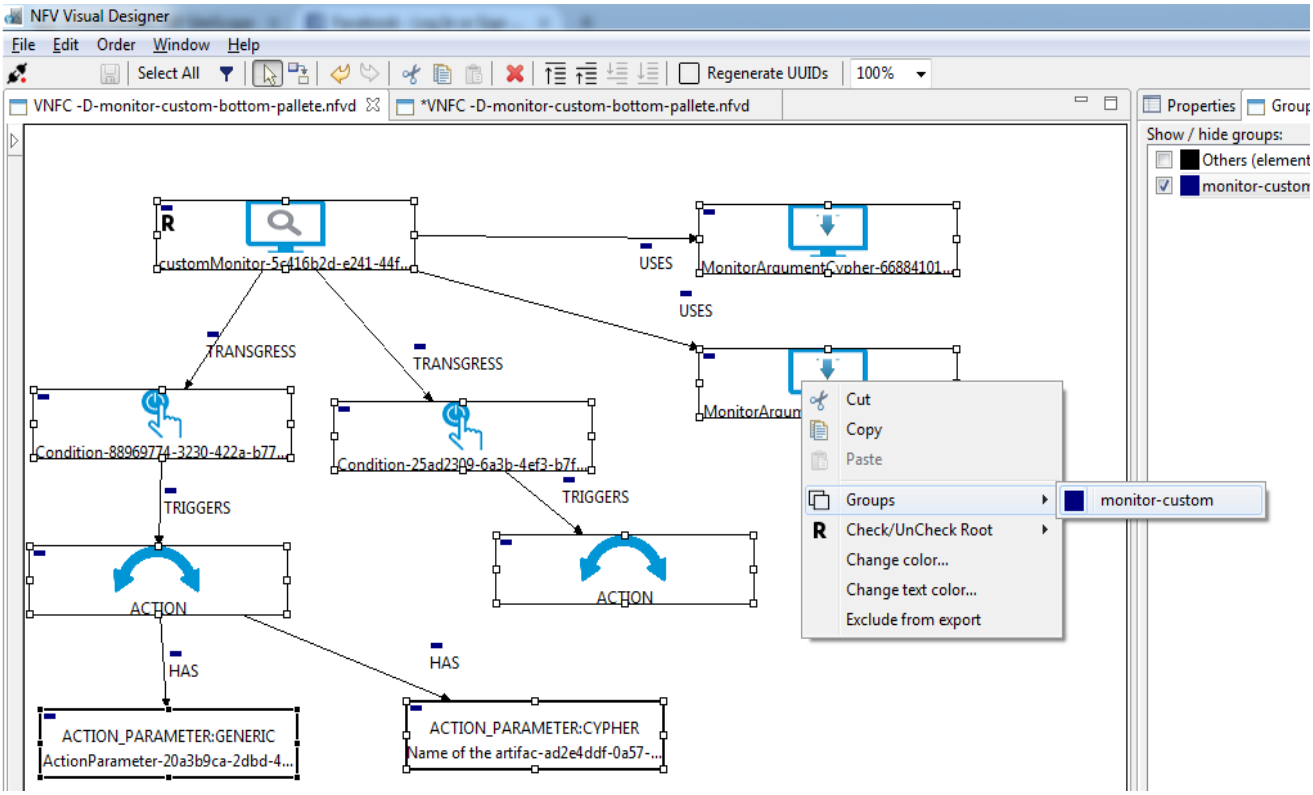


Figure 106: Add components to the group

13. Create connectors to the new component palette.
    a. Click the **Components** tab in the bottom of the middle pane.
    b. Click the **Connectors** tab in the right pane.
    c. Click **Add connector** to add a new connector.
    d. Enter the connector name/type, select the artifact and mode as indicated on the following illustration.

NOTE: The connector name and type is very important. The name/type must match the ones in the target component palette, otherwise they will not appear in the online designer and will not connect to the monitor.

NOTICE: In 4.0 release the palettes provided as part of the product only have monitor connectors for the virtual machine. This is a limitation. You can add new connectors to the standard palette and use them here.

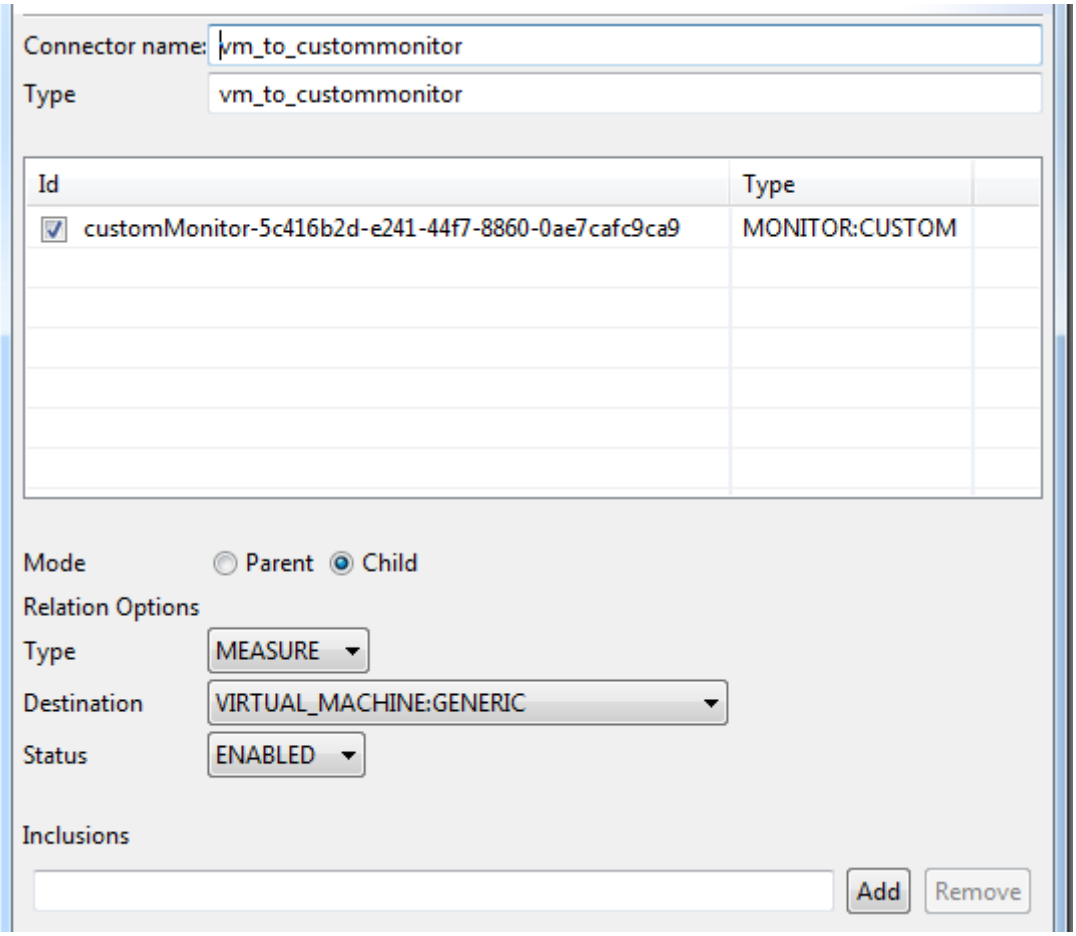


Figure 107: Create connectors for the component palettes

e. Add connectors for the action. Actions can only be connected to VNF/VNFC/Virtual_Machine. If there is a requirement for the action to be pointed to any other artifact, such as VIRTUAL_DISK, then you need to add the connectors accordingly.

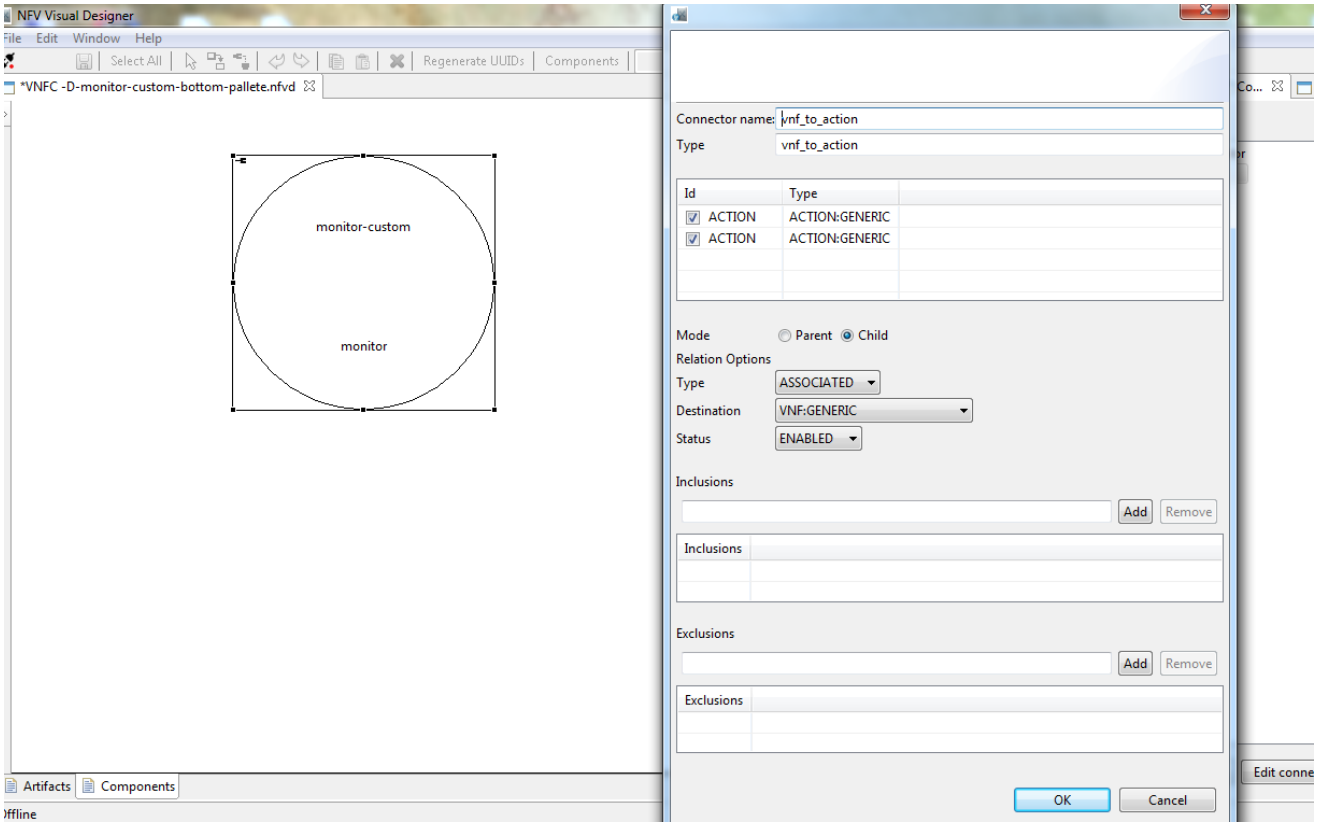


Figure 108: Create connectors from VNF to action

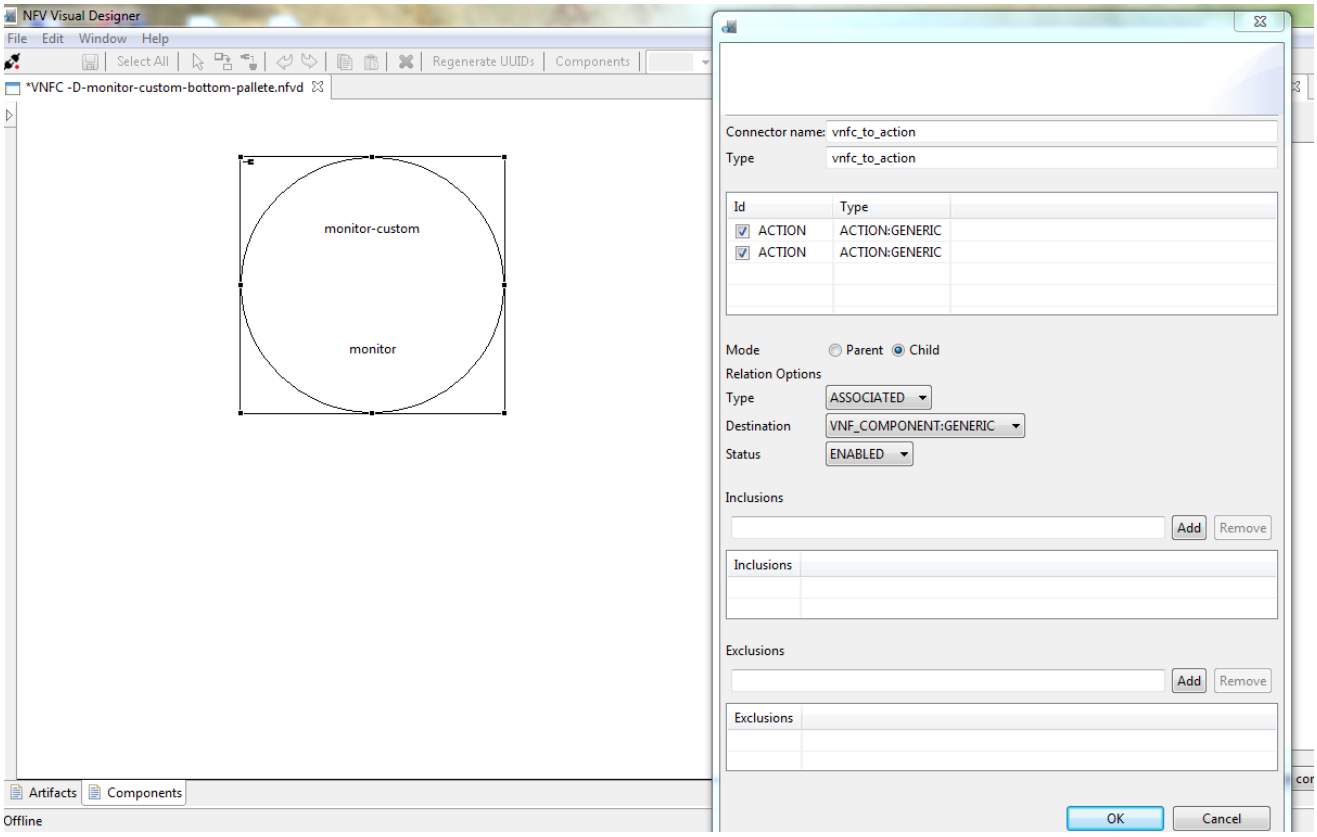


Figure 109: Create connectors from VNFC to action

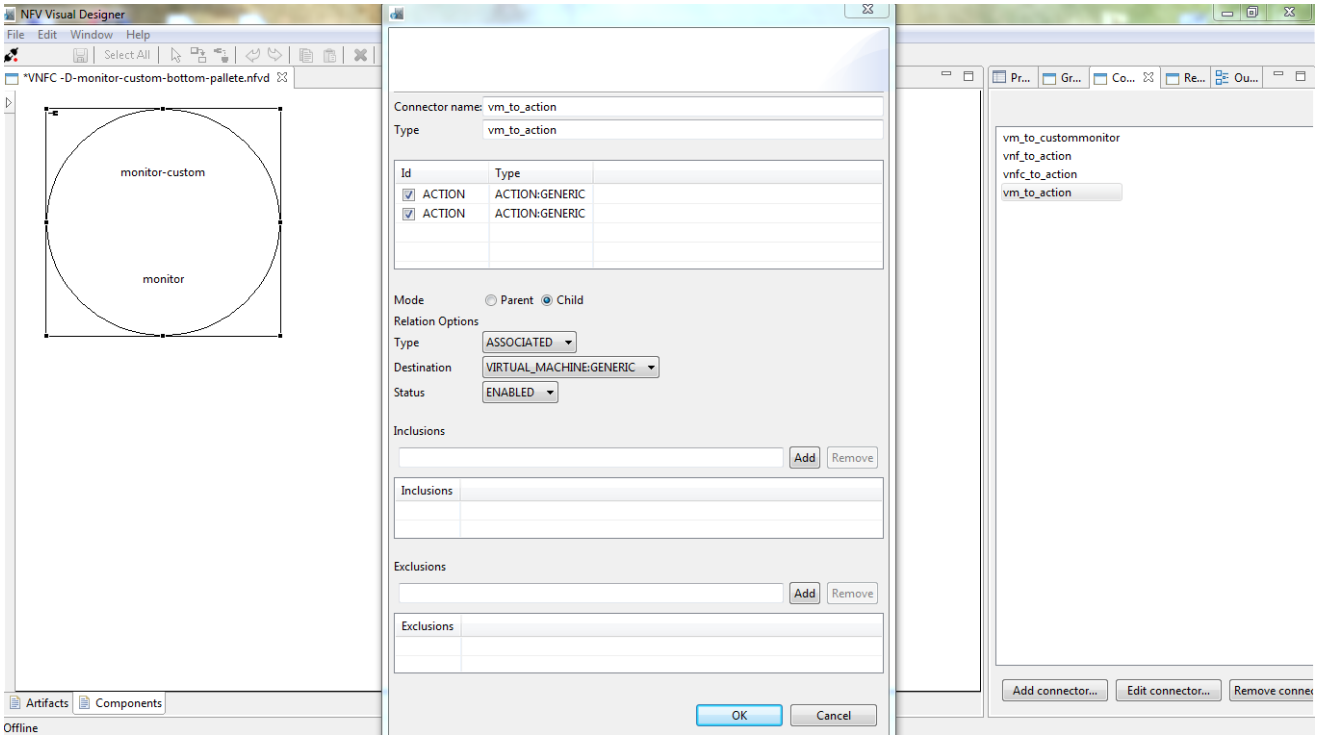


Figure 110: Create connectors from VM to action

14. The following illustration shows the final component diagram after creating all the components:

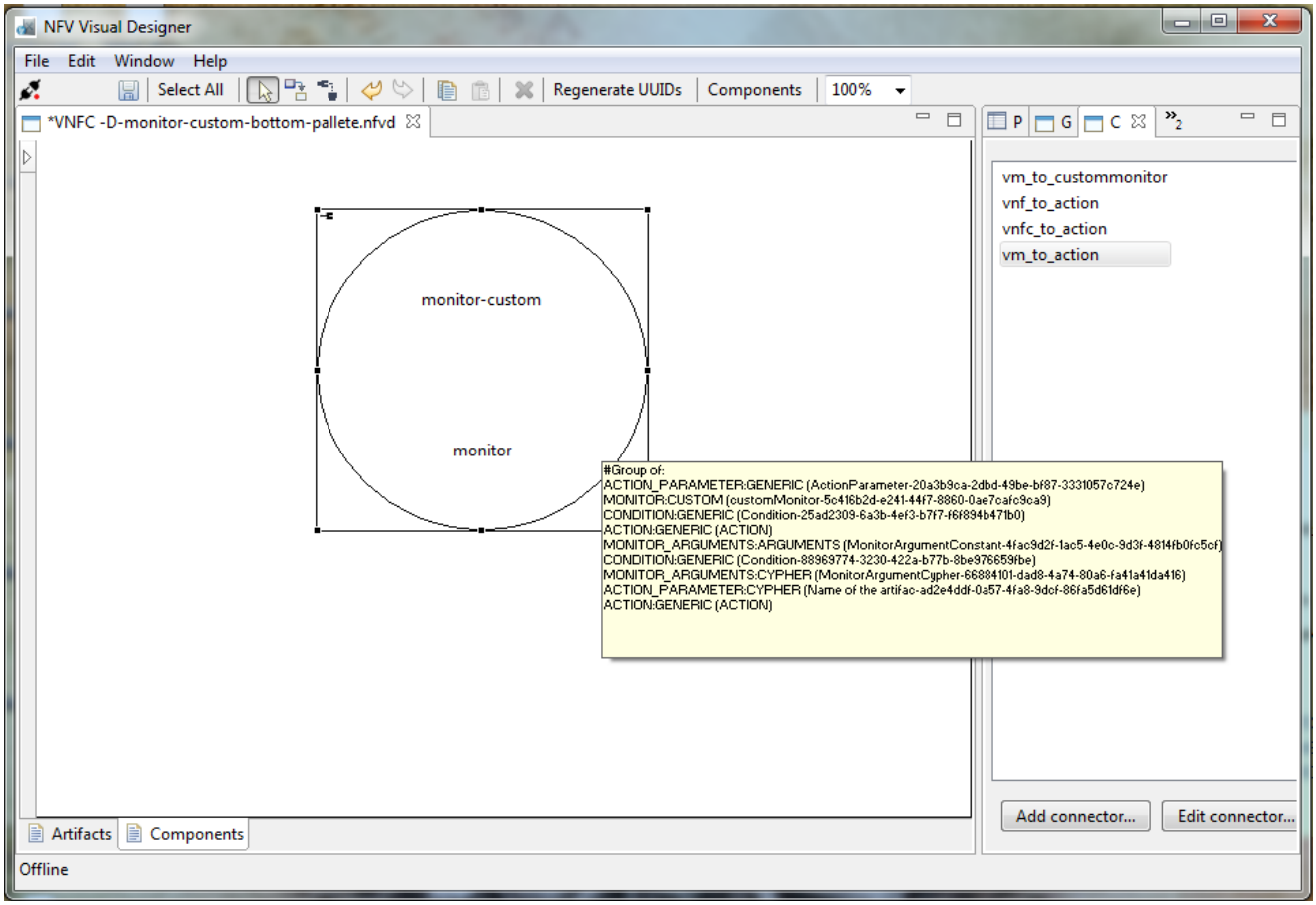


Figure 111: Final component diagram after creating all the components

15. Export the created custom monitor palette.
    a. Make sure you are in offline mode.

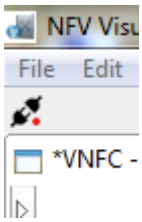


Figure 112: Export palette in offline mode

    b. Click **File > Export**.
    c. Select the **Export NFVDiagram to XML** wizard. Click **Next**.
    d. Choose the template and include all items indicated on the following illustration. Click **Finish** to create a loadable XML file.

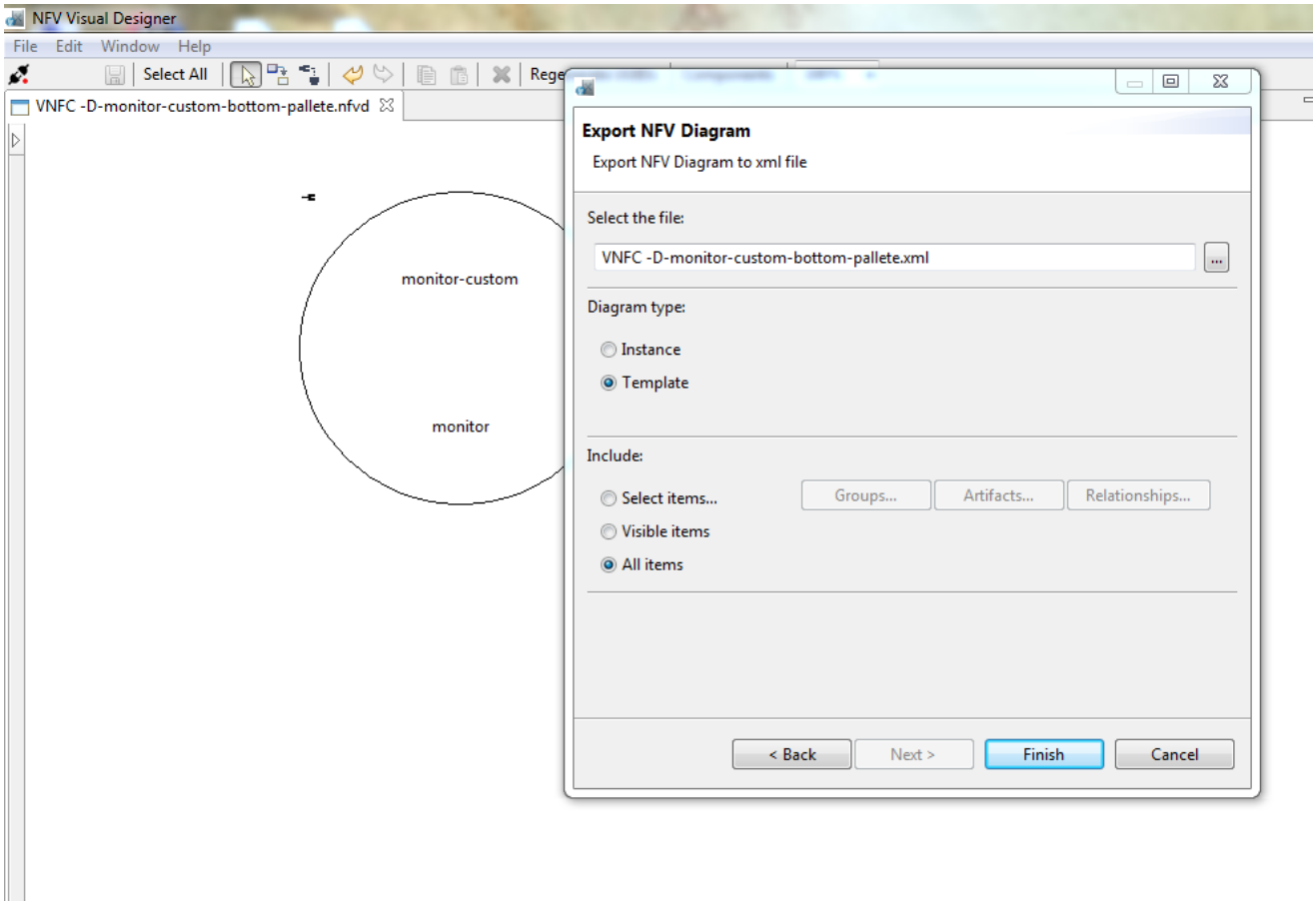


Figure 113: Export the custom monitor palette

e. Load contents of the XML file with the Fulfilment Rest API.

```
http://
METHOD: POST
Content-Type: application/xml
X-Auth-Token: (Needs to be obtained by using FF rest API for a user)
```

16. After uploading the palette, the user can log into the GUI and see the new monitor-custom.

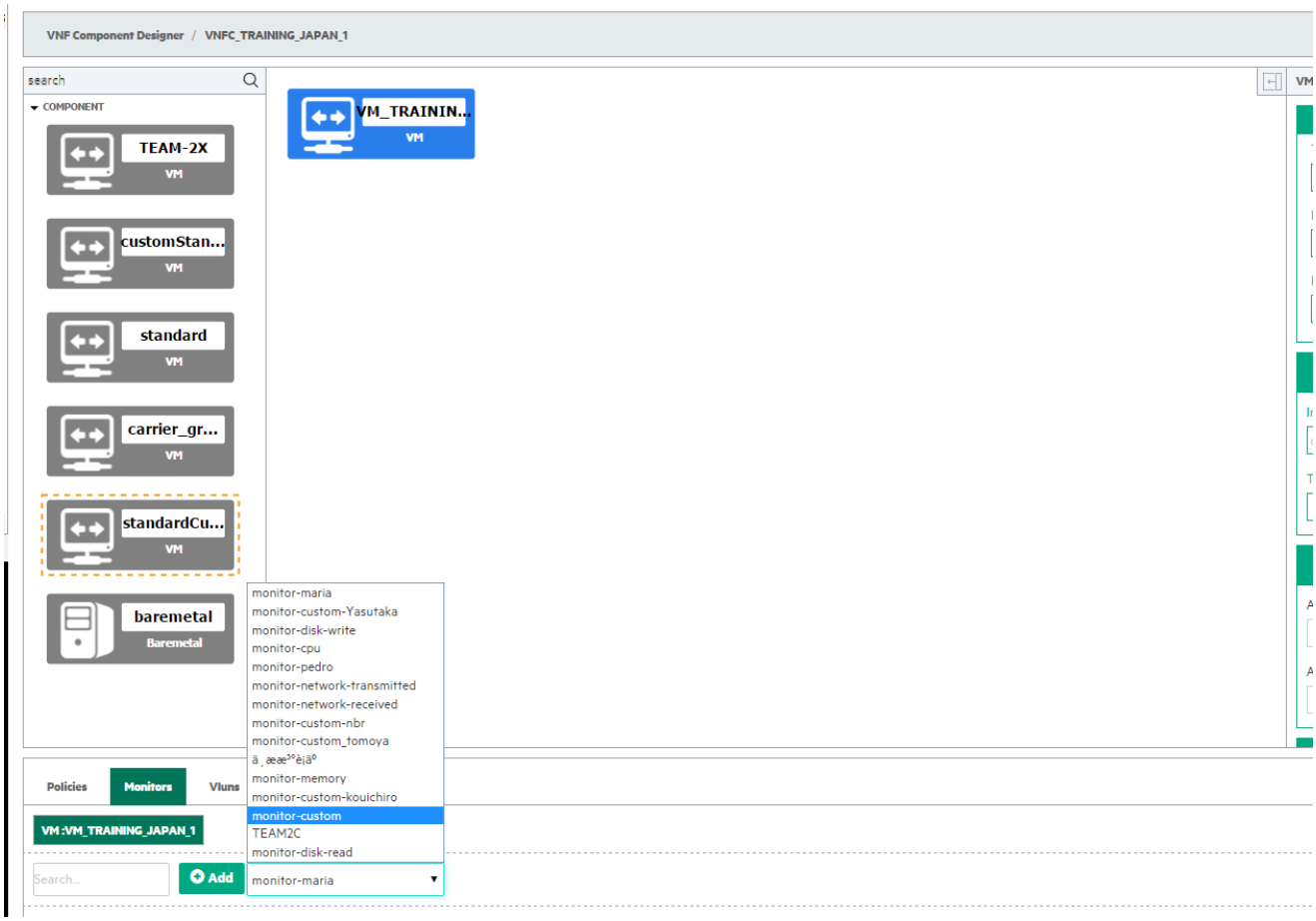


Figure 114: Search for custom monitor in the GUI

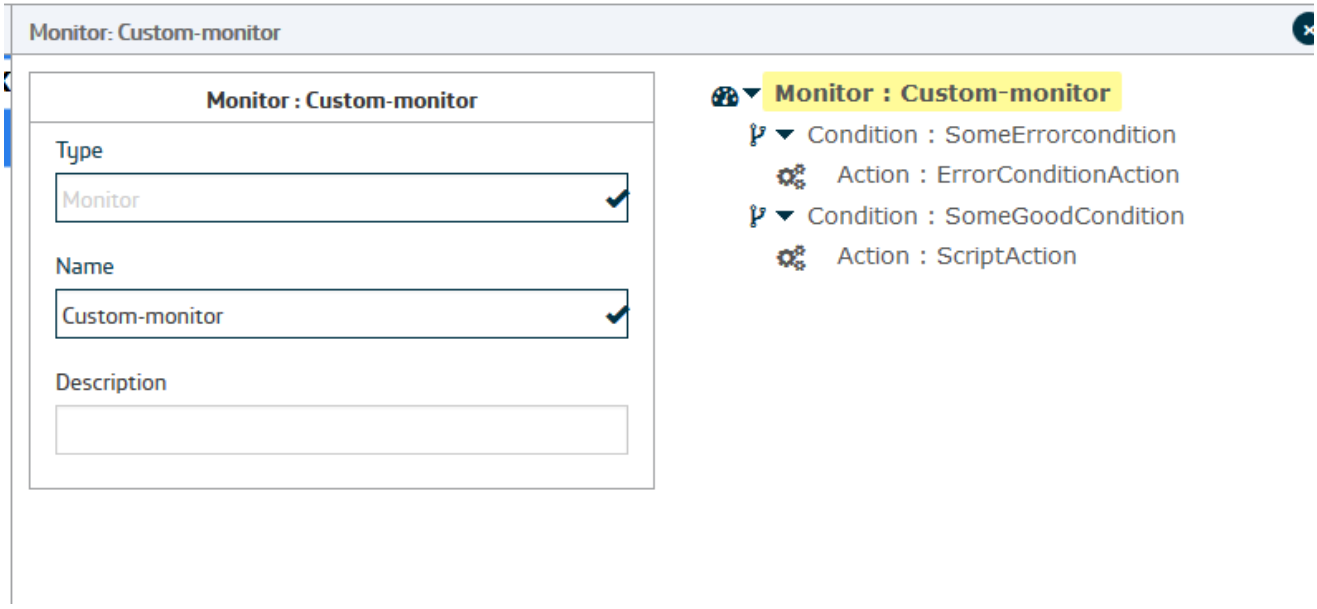


Figure 115: New custom monitor in the GUI

10.10.6 Custom event collection

Refer to the *Open Mediation 6.2 guide, OSS_Skeleton_CA_archetype_for_UCAEBC_Guide.pdf* to implement a new channel adapter for NFVD.

In NFVD 3.0 and above, the SNMP traps from NFVD monitors (SiteScope, VCenter) are translated to X.733 format by the Generic SNMP Channel Adapter. By default, SiteScope and VMware customization are delivered. The SNMP UDP port can be configured in the Generic SNMP Channel Adapter to receive SNMP traps from monitors. Configuration information is available in the *Generic SNMP Channel Adapter guide*.

Table 24: NFVD Alarm attributes

Alarm Attribute Name	Description
`<identifier>`	The value should be unique, so UUID is generated by the Channel adapter and is appended with the alert.
`<sourceIdentifier>`	The value should be set at the Channel adapter to constant `NFVD_Source`.
`<perceivedSeverity>`	ERROR monitor condition: CRITICAL. WARNING monitor condition: WARNING. GOOD monitor condition: CLEAR.
`<alarmType>`	The value should be configured at the Channel adapter to constant `QUALITY_OF_SERVICE_ALARM`.
`<additionalText>`	The value should contain `groupdescription=<MonitorName>`, where `<MonitorName>` is the value of the `GENERALName` property of the Monitor artifact.

The configurable files for a specific container instance are stored under following location:

<NOM container path>/ips/generic-snmp-ca-V20/etc.

10.10.6.1 Essential CA configuration

In order to configure a Generic SNMP CA, the config.properties file should be customized. The parameters in the file are the following:

- snmp_listen_address: Sets the IP/Port which will be used for receiving SNMP traps. OM container should have sufficient permissions in case a privileged (<1024) port is used, such as 162.
  Default value: 0.0.0.0:162
- instance_name: Name of Generic SNMP CA instance for distributed configurations.
  Default value: Generic-SNMP-CA-Instance-1

10.10.6.2 CA configuration file for OM alarm processing

The file bus-connector.xml defines the end-points and routing for Apache Camel.

10.10.6.3 Update Channel Adapter

If either of the two configuration files is updated whilethe CA is already deployed, it should be re-deployed for the changes to be applied.

Deploy CA:

```
# <NOM installation path>/bin/nom_admin --deploy-ip-in-container
<container_instance> generic-snmp-ca-V20
```

Check whether CA is deployed:

```
# <NOM installation path>/bin/nom_admin --show-ip-in-container
<container_instance> generic-snmp-ca-V20
```

Undeploy CA:

```
# <NOM installation path>/bin/nom_admin --undeploy-ip-in-container
<container_instance> generic-snmp-ca-V20
```

10.10.7 Custom automated action

Launch the UCA EBC UI, login as admin to deploy and start the UCA NFVD Problem Detection Value Pack, the UCA Automation Foundation Value Pack, and the UCA NFVD PublishToNomBus Value Pack.

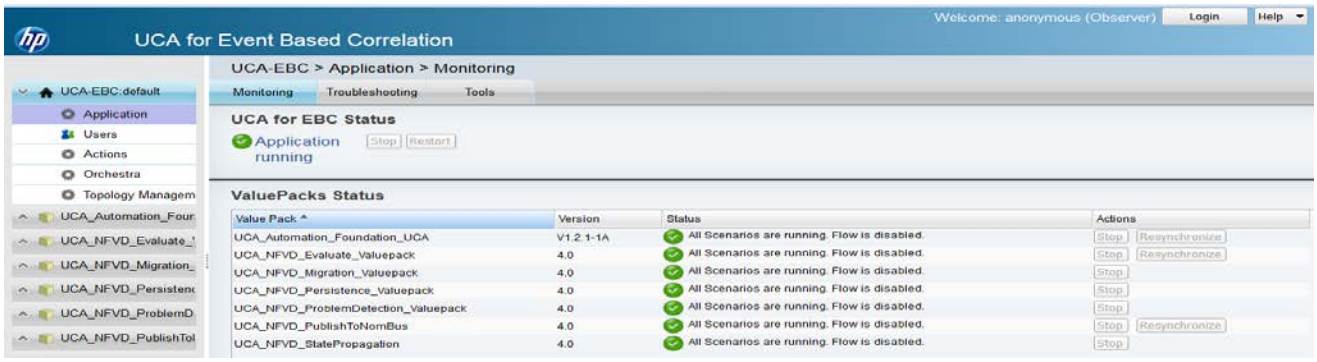


Figure 116: List of UCA Value Packs for NFV Director

10.10.8 Custom closed-loop action

This section explains how to create a new action and associate it to a template in UCA without using the decision tree. Refer to the UCA-EBC documentation on how to use the decision tree.

1. Log in to the HPSA where the UCA is installed. Locate the UCA Services as indicated on the following illustration.

NOTE: Every NFVD-related PROBLEM and ACTION is grouped under the NFVD Service. The Standard Problem detection value pack provided out of the box always groups alarms under the NFVD service.

If scenarios are to be grouped under a different service, the user must write their own problem detection value pack and group the alarms under that particular service. Refer to the UCA-EBC documentation on writing a problem detection value pack.

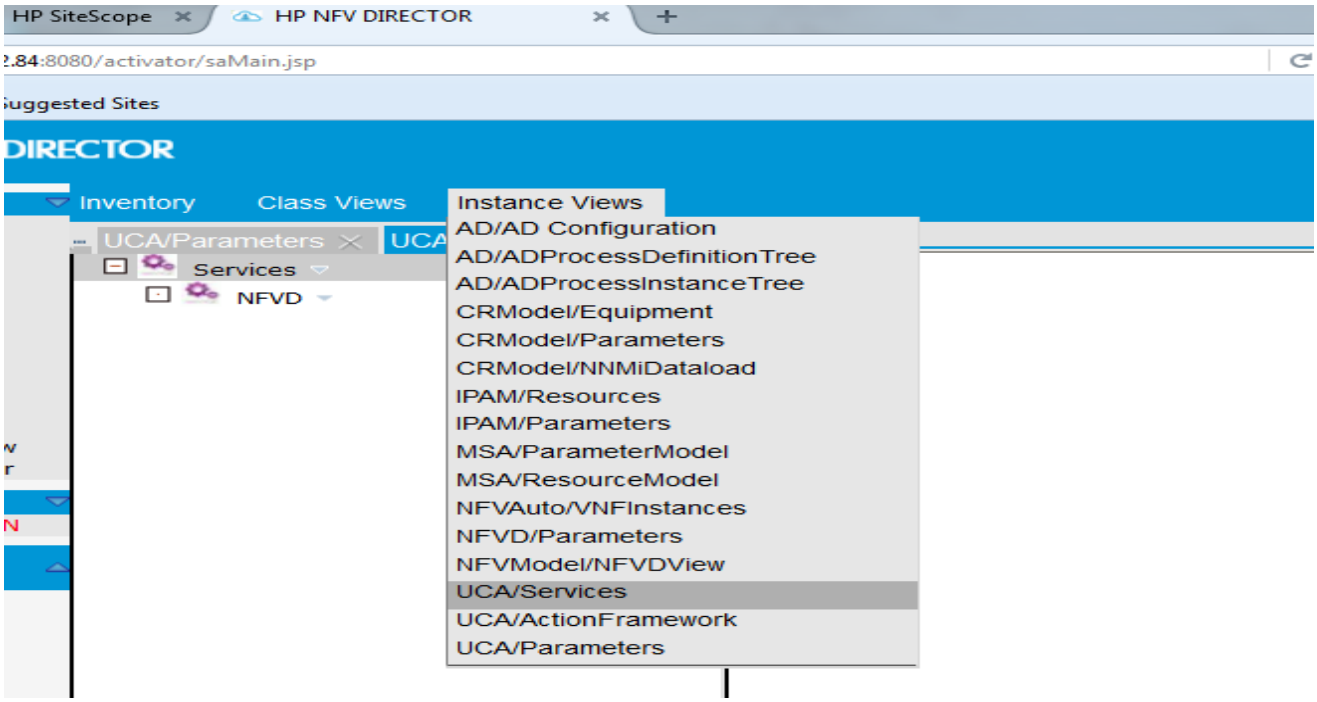


Figure 117: UCA/Service Inventory view in HPSA

2. Select the **UCA/Action Framework** to define a new custom action and problem as indicated on the following illustration:

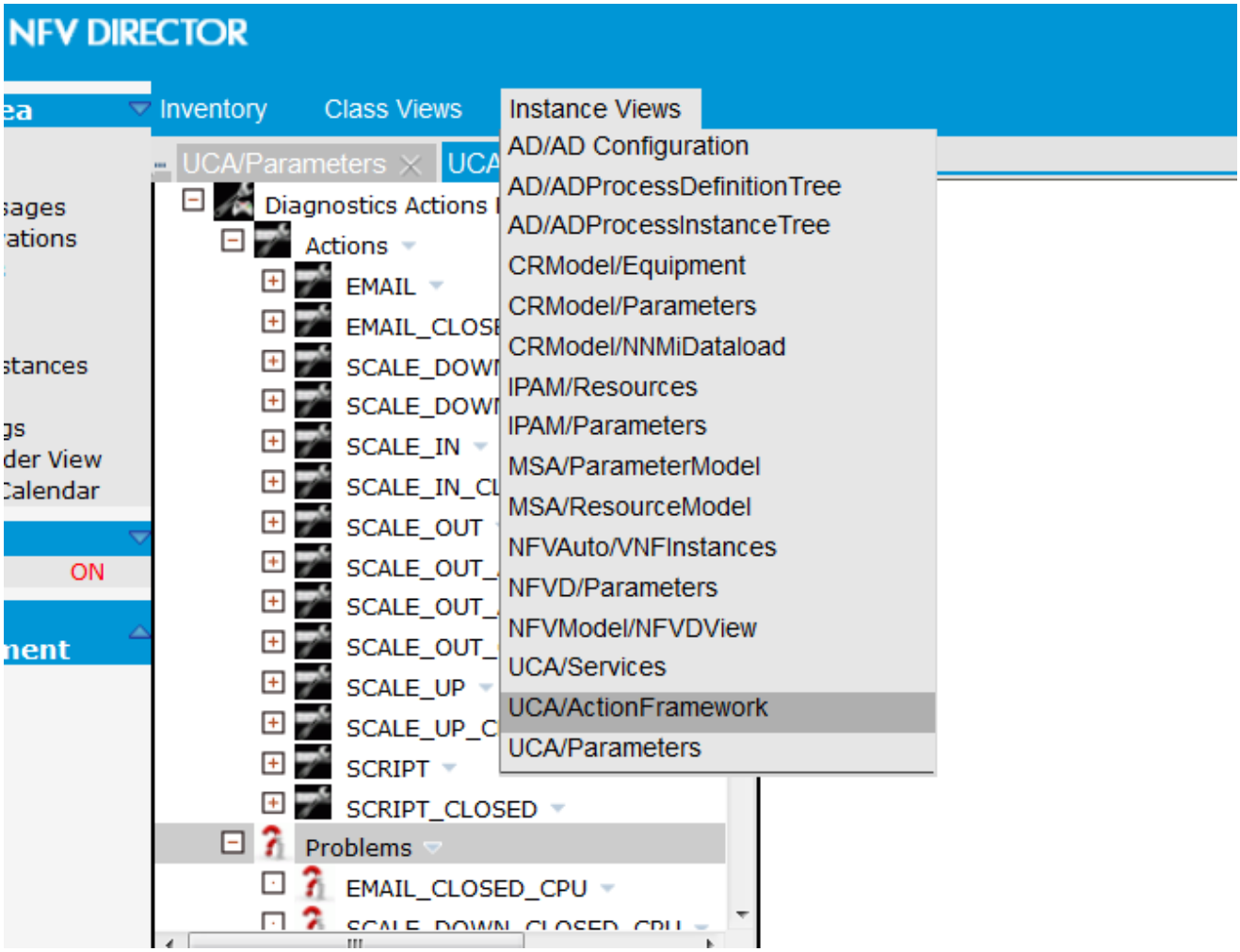


Figure 118: Create a new custom action and problem in the HPSA UCA/ActionFramework Inventory

3. Create a new Action by right-clicking **Actions** and selecting **Create New Action**.

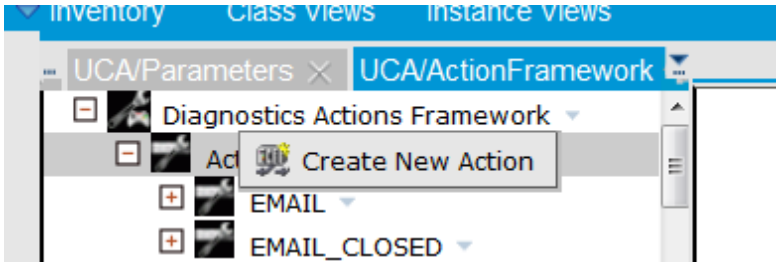


Figure 119: Create a new action

4. Create a new Open Loop Action called ROUTE.

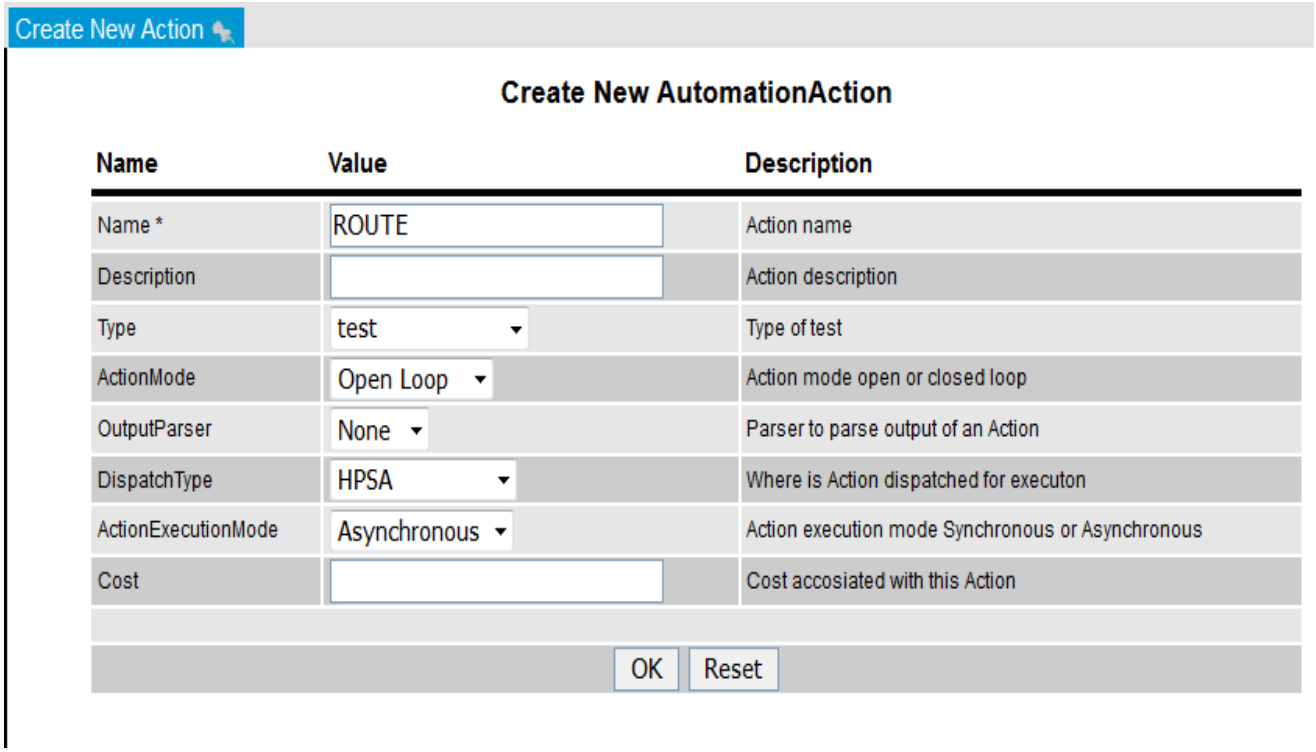


Figure 120: Provide new action parameter details with Open Loop

5. Create a new Closed Loop action called ROUTE_CLOSED. The Problem detection value pack provided as part of the NFVD attaches _CLOSED to the action name received from the topology. The user must follow the same convention if they want to use the standard Problem detection value pack.

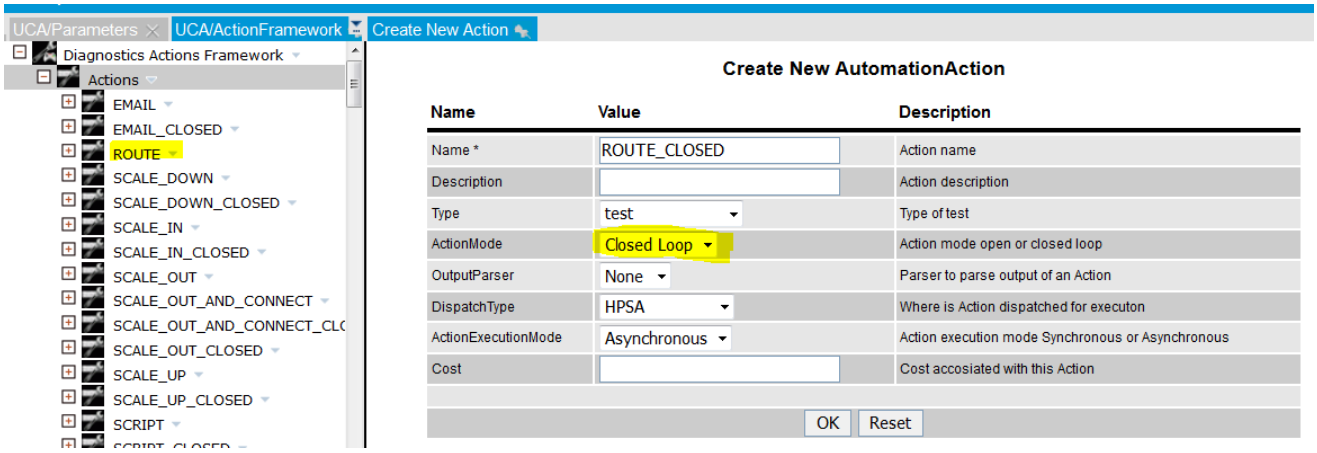


Figure 121: Create a new action with Closed Loop

6. Create a new problem by right-clicking **Problems**

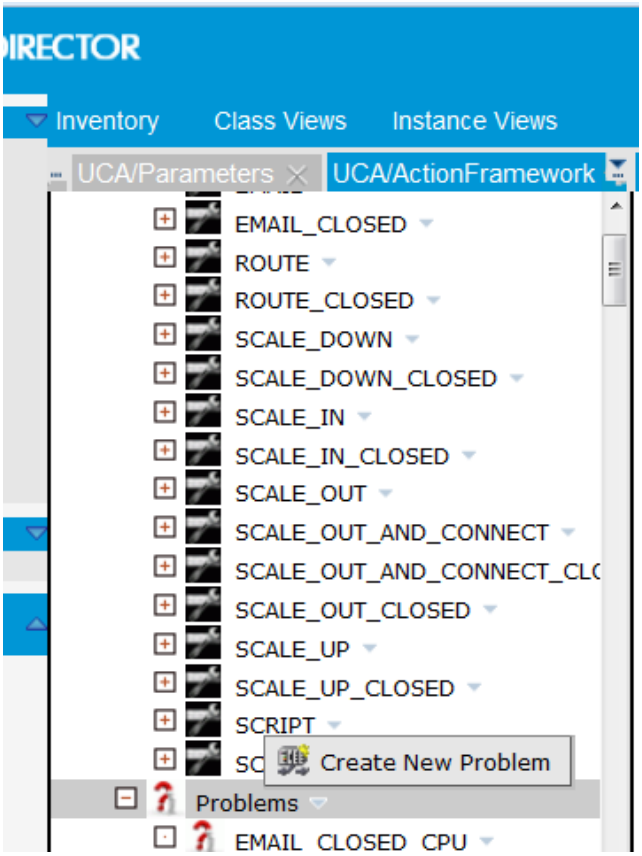


Figure 122: Create a new problem

7. Configure the new problem as indicated on the following illustrations.

NOTE: The monitor type must be appended to the ROUTE action created in the previous steps. Possible monitor types include CPU, Memory, DiskRead, DiskWrite, NetworkRx, NetworkTx, Process, Network, Logs, Storage, and CUSTOM. For example, ROUTE_CPU, ROUTE_CLOSED_CUSTOM, and so on.

NOTE: In the previous example the NFVD service was selected. You can select a different service or even disregard the naming convention, but then you have to write your own Problem detection value pack.

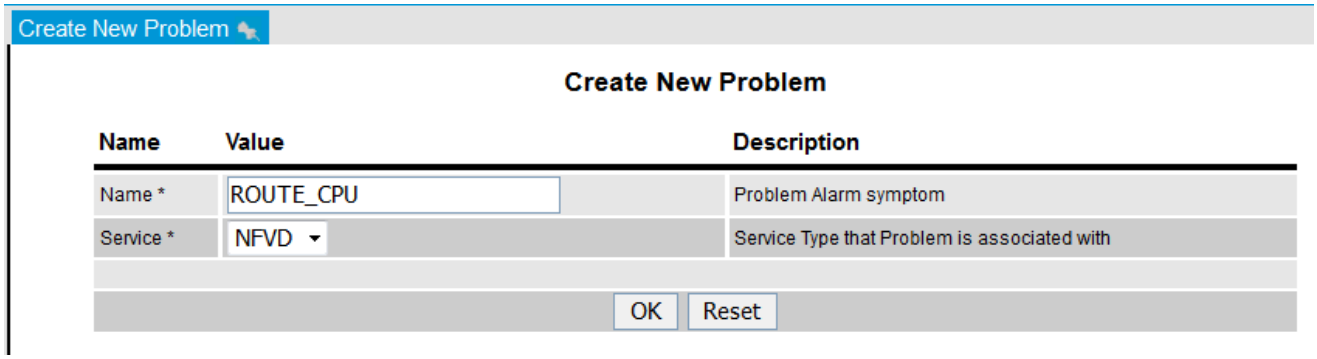


Figure 123: Enter properties for the new Open Loop problem

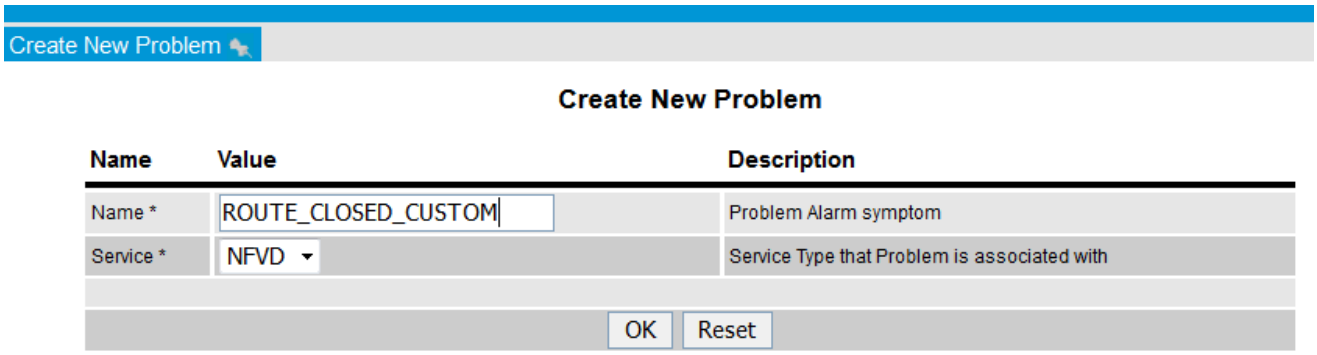


Figure 124: Enter properties for the new Closed Loop problem

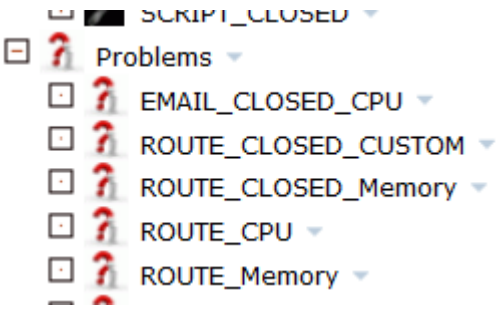


Figure 125: List the new problems

8. Locate **UCA/Parameters** to map actions to a workflow.

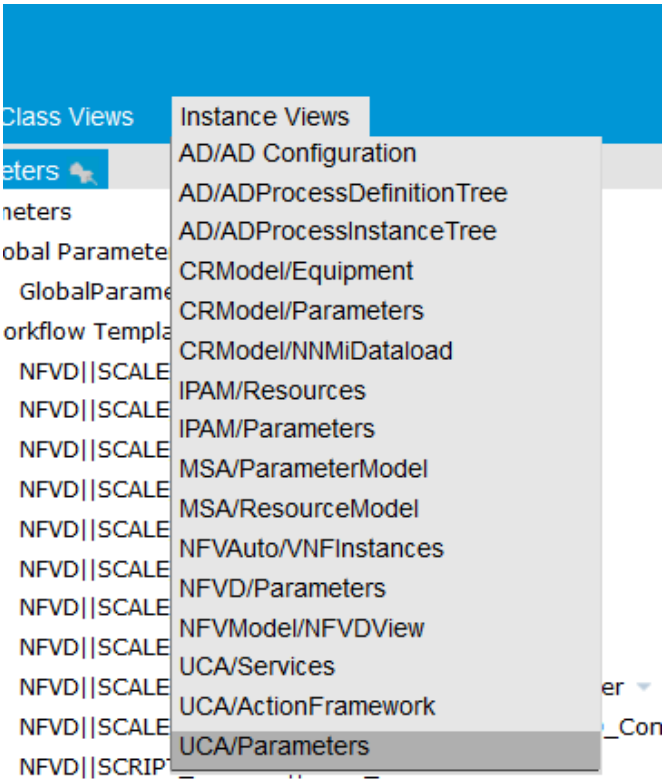


Figure 126: UCA/Parameters view in HPSA Inventory

9. Create a new workflow template to map an action to an HPSA workflow.

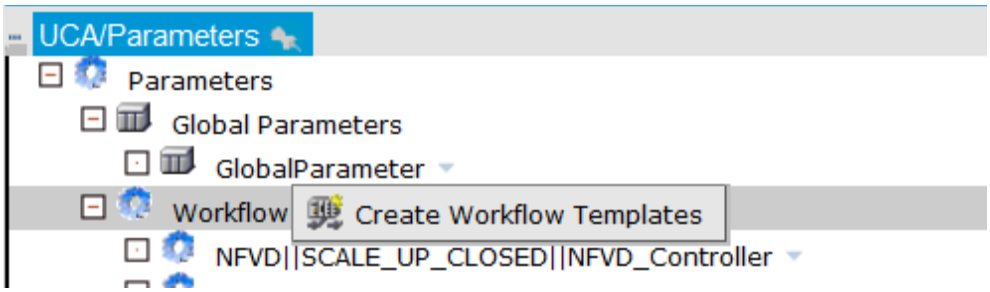


Figure 127: Create a new workflow template to map an action to an HPSA workflow

10. Configure the action to NFVD_Controller workflow as indicated on the following illustrations. The NFVD_Controller delegates the calls to other workflows which invokes rest calls on Fulfilment/VNFM or executes some script action.

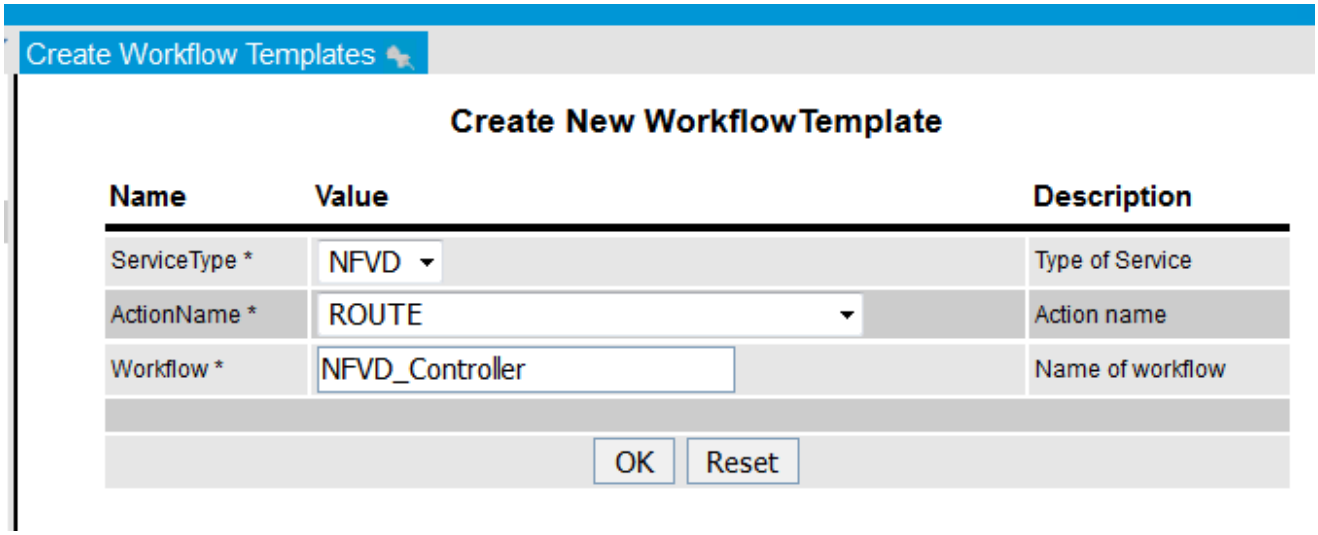


Figure 128: Enter properties for the new Open Loop action workflow template

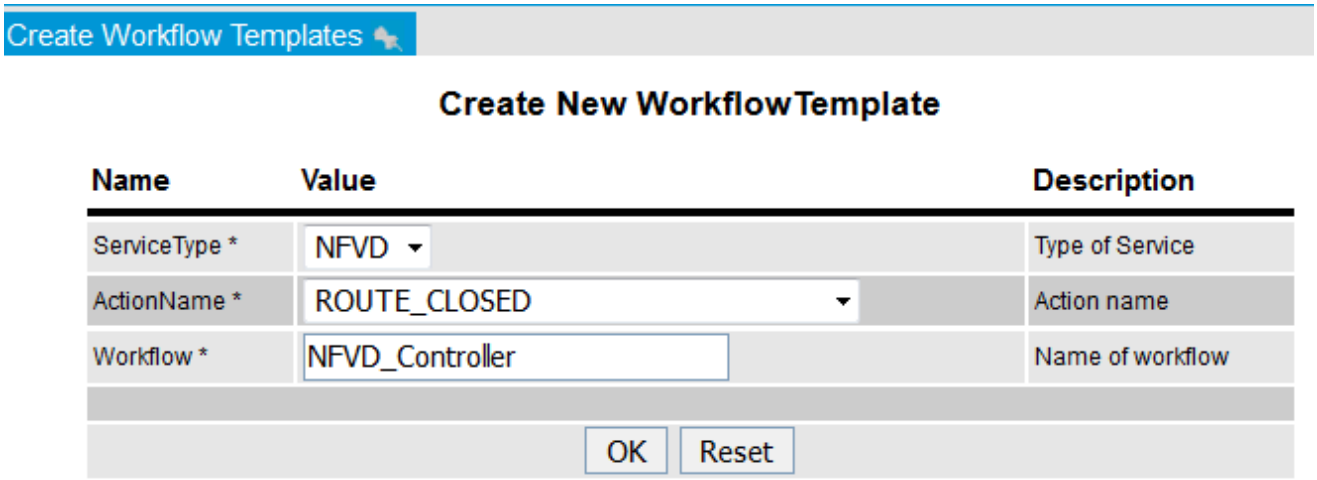


Figure 129: Enter properties for the new Closed Loop action workflow template

11. Select **NFVD/Parameters**.

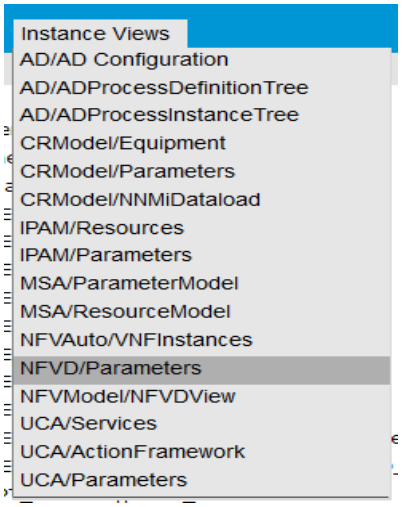


Figure 130: NFVD/Parameters view in HPSA Inventory

12. Right-click **Workflow templates** to create a new Custom workflow.

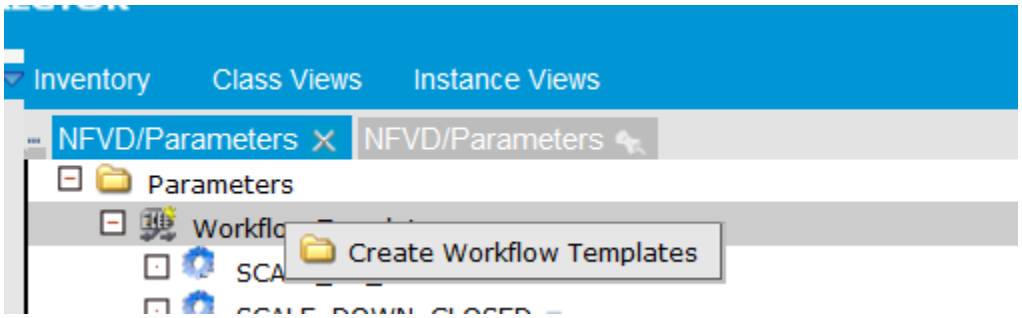


Figure 131: Select Custom workflow

13. Configure the custom workflow as below.

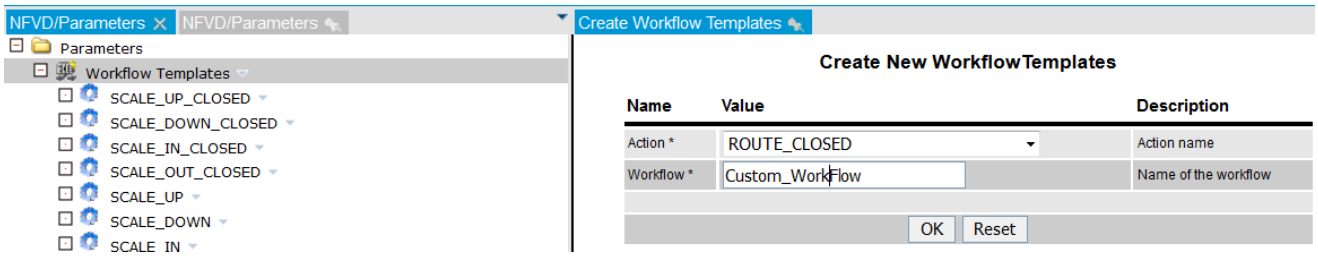


Figure 132: Creating a new custom workflow

14. The user has to develop and deploy the Custom_Workflow in HPSA. Custom_Workflow can access the action parameters, such as host and parameter that were configured as part of the topology by looking at the `parameternames` and `parametervalues` custom fields in the alarm.

NOTE: The standard Problem detection value pack sets the `actionpreset` custom field to `true`. It indicates to the UCA-EBC that the mapping between problem and action has been resolved, and it does not have to search decision trees to find the executable action for a problem.

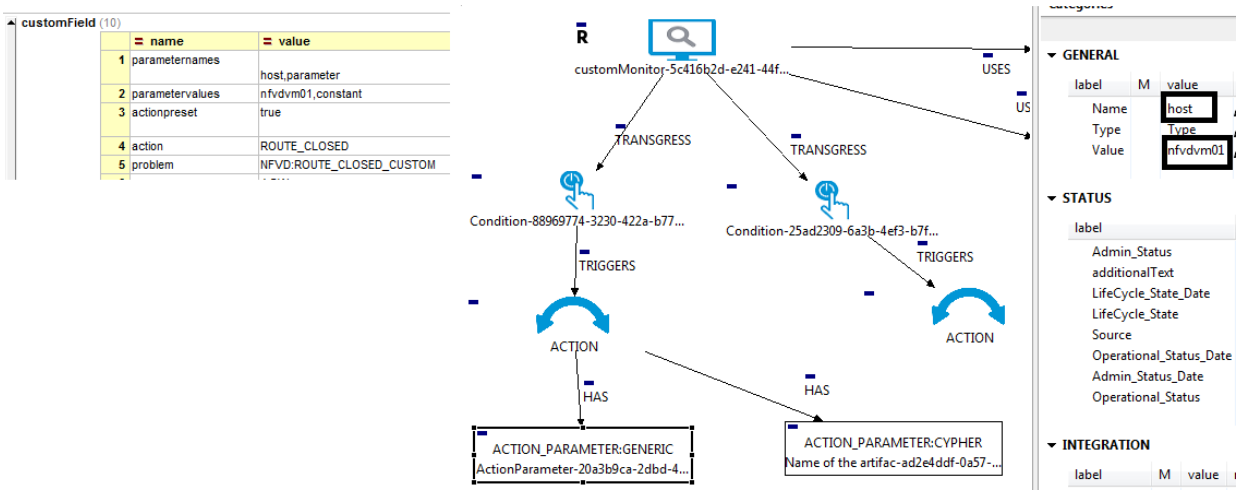


Figure 133: Set attributes to action parameters

NOTE: The standard Problem detection value pack completes the action field based on the `GENERAL.Type` and `GENERAL.Operation_Mode` parameters of the action artifact for the condition that caused the threshold breach.

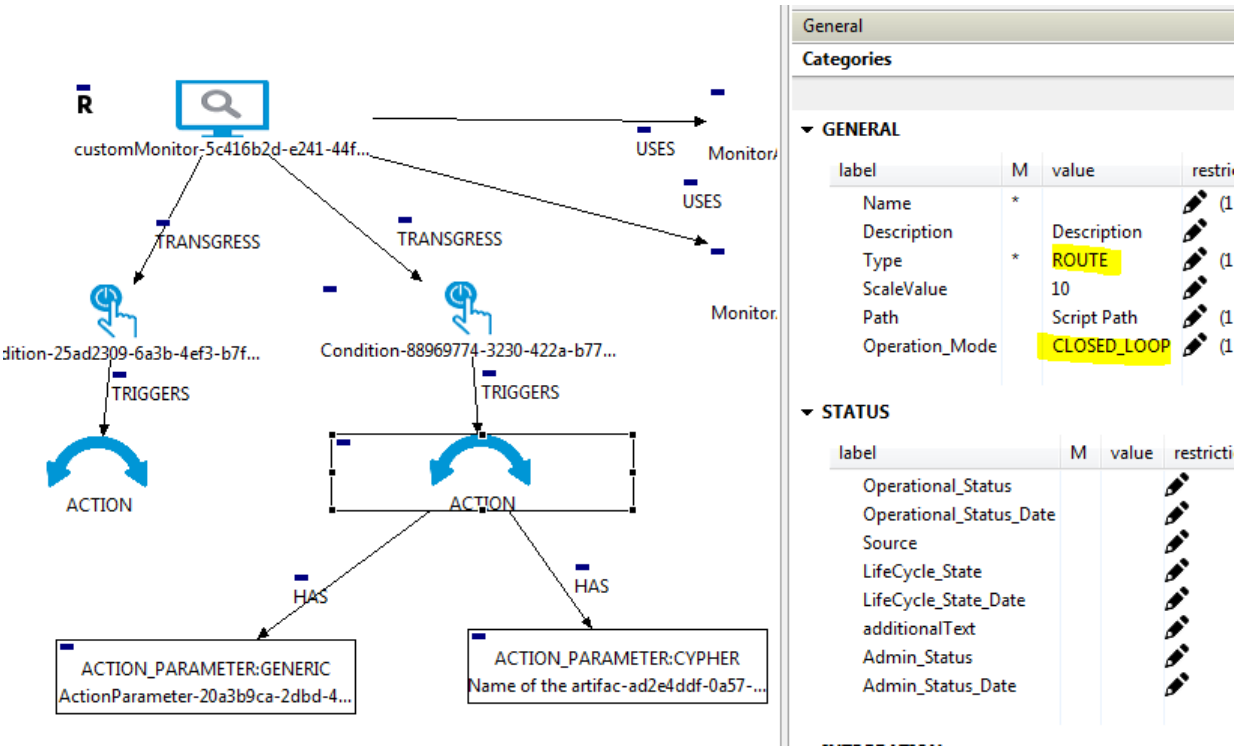


Figure 134: Map new action to the action parameter in the GUI

NOTE: The standard Problem detection value pack completes the `NFVD:ROUTE_CLOSED_CUSTOM` problem field, for example, `SERVICE:ACTION_MonitorType` .

10.10.9 Custom correlation ValuePack

This section explains how to create a custom value pack for customized state propagation.

The State propagation value pack propagates operational states for the child and all its affected parents based on **Propagation mode**. If **Propagation mode** is set to RULE_BASED, then the alarm will be orchestrated from the state propagation value pack to a custom value pack. Follow these steps to perform this orchestration.

1. Develop and deploy the **CustomRule Value pack**. Please refer to the *HPE UCA for Event Based Correlation V3.1 - Value Pack Development Guide* to create a custom value pack.

ValuePacks Status

Value Pack	Version	Status	Actions
CustomRule	1.0	All Scenarios are running. Flow is disabled.	Stop Resynchronize
UCA_Automation_Foundation_UCA	V1.2.1-1A	All Scenarios are running. Flow is disabled.	Stop Resynchronize
UCA_NFVD_Evaluate_Valuepack	4.0	All Scenarios are running. Flow is disabled.	Stop Resynchronize
UCA_NFVD_Migration_Valuepack	4.0	All Scenarios are running. Flow is disabled.	Stop
UCA_NFVD_Persistence_Valuepack	4.0	All Scenarios are running. Flow is disabled.	Stop
UCA_NFVD_ProblemDetection_Valuepack	4.0	All Scenarios are running. Flow is disabled.	Stop
UCA_NFVD_PublishToNomBus	4.0	All Scenarios are running. Flow is disabled.	Stop Resynchronize
UCA_NFVD_StatePropagation	4.0	All Scenarios are running. Flow is disabled.	Stop

Figure 135: List of UCA value packs in NFV Director

2. In case of **Rule based propagation mode**, the state propagation value pack adds a custom field parameter with `userText=toRule` value to the alarm. The user can add a filter with similar functionality in the `$UCA_EBC_DATA/instances/default/conf/OrchestraFilter.xml` file. The following illustration indicates the required parameters and values. Similar filters can be added in the rules file of the custom value pack for more specific filtering.

```
<topFilter name="stateToRule" >
   <allCondition tag="stringFilterStatement" >
      <stringFilterStatement>
         <fieldName><![CDATA[userText]]></fieldName>
         <operator>isEqual</operator>
         <fieldValue><![CDATA[toRule]]></fieldValue>
      </stringFilterStatement>
   </allCondition>
</topFilter>
```

Figure 136: Add a filter to the OrchestraFilter.xml file

3. Configure the `$UCA_EBC_DATA/instances/default/conf/OrchestraConfiguration.xml` file so that the alarms flow from the **State propagation value pack** to the **Custom Rule value pack** based on the filter stateToRule filter created in step 2.

```
<Route name="StateToCustomRule" >
   <COPY>
      <Source>
         <ValuePackNameVersion><![CDATA[UCA_NFVD_StatePropagation-4.0]]></ValuePackNameVersion>
         <ScenarioName><![CDATA[UCA_NFVD_StatePropagation.StatePropagationScenario]]></ScenarioName>
      </Source>
      <Destinations>
         <Destination>
            <Filter>
               <filterName><![CDATA[stateToRule]]></filterName>
            </Filter>
            <Target>
               <ValuePackNameVersion><![CDATA[CustomRule-1.0]]></ValuePackNameVersion>
               <ScenarioName><![CDATA[CustomRule.CustomRule]]></ScenarioName>
            </Target>
         </Destination>
      </Destinations>
   </COPY>
</Route>
```

Figure 137: Configure the OrchestraConfiguration.xml file

4. Please refer to the sample SiteScope alarm that reaches the custom value packs. SiteScope alarms have `sourceIdentifier` set as `NFVD_Source`. The `additionalText` field includes details about the `MonitorArtifactId` (for example, the `artifactId` of the monitor that caused the alarm). The `additionalText` field includes the `errorOnly`/`goodOnly`/`warningOnly` fields that describe which counter triggered the alarm. For example, SiteScope detects that the CPU average usage on the virtual machine with UUID value **225dd3fc-db7b-45c8-b413-15aefd73cfda** is 0 and triggers a good alarm.

10.10 SiteScope customizations

10.10.3 Accessing SiteScope

Enter the SiteScope address in a Web browser. The default address is `http://<server name>:<port>/SiteScope`.

NOTE: SiteScope can also be accessed from the Windows Start Menu. Click the **Start** button and select **Programs > HPE SiteScope > Open HPE SiteScope**.

Enter the login credentials and click **Login**.

10.10.4 Overview of the SiteScope dashboard

The SiteScope default dashboard is displayed after a successful login. The interface elements need to be initialized the first time SiteScope is deployed, which causes a delay. The dashboard displays current performance data for the infrastructure elements monitored by SiteScope and provides access to functions used to define filters. It also displays a table of groups and monitors for the elements highlighted in the monitor tree or listed in the path. You can double-click each group or monitor node to navigate to child nodes and monitors.

The **SiteScope dashboard** window contains the following key elements:

- The **Common toolbar** provides access to page options, documentation, and additional resources. This toolbar is located in the upper part of the window.
- The **Context toolbar** contains buttons for frequently used commands in the selected SiteScope context.
- The **Context tree** enables you to create and manage SiteScope objects in a tree structure.

- The **Context buttons** provide access to the **SiteScope Monitors**, **Remote Servers**, **Templates**, **Preferences**, **Server Statistics**, and **Diagnostic Tools**.

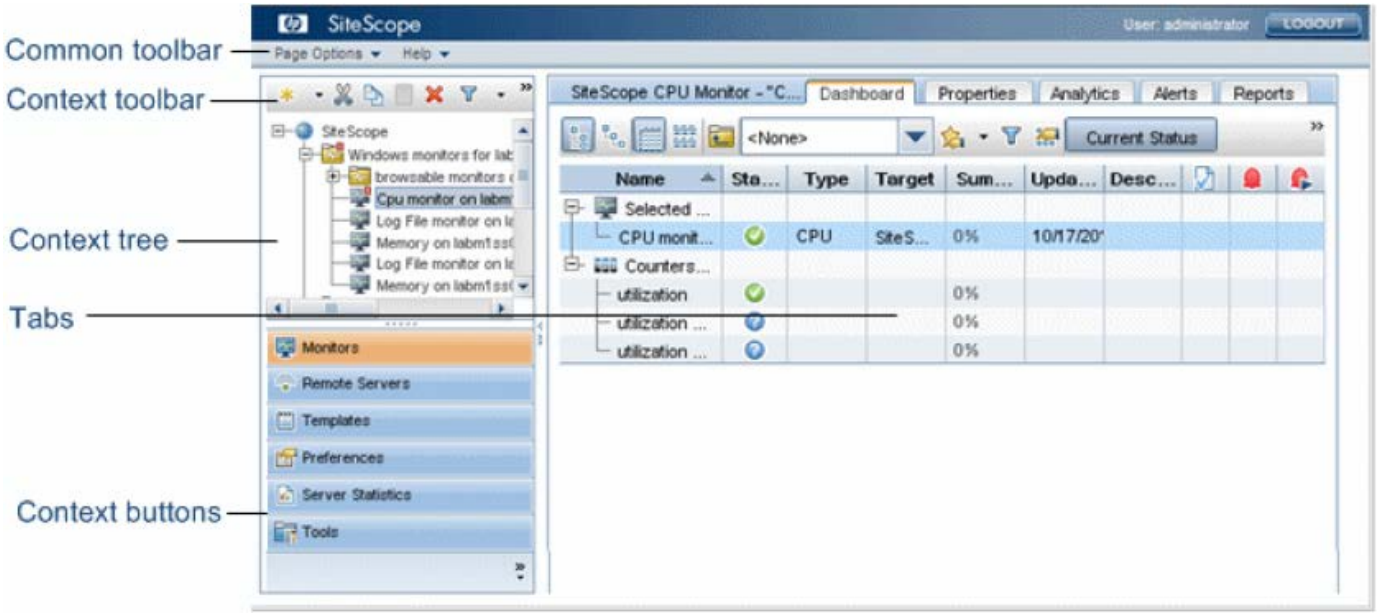


Figure 138: SiteScope dashboard

SiteScope monitoring provides a real-time picture of system availability and performance. SiteScope monitors are configured to collect metrics from a range of infrastructure components, including the Web, application, database, and firewall servers. The status and metrics are then aggregated for presentation in the SiteScope dashboard.

The dashboard is linked to the SiteScope monitor tree hierarchy. The data displayed in the dashboard represents the context selected in the monitor tree. The highest level is the SiteScope node and any applicable monitor groups. The lowest level element displayed in a dashboard view is an individual SiteScope monitor and its measurements.

The dashboard includes functions used to customize the display of monitor information. This includes defining named filter settings to limit the display of data resulting from defined criteria and select various data display options.

The dashboard also includes hyperlinks and menus used to navigate through the hierarchy of monitor elements, manually run a monitor, disable monitors, and access alert definitions.

The **SiteScope dashboard** includes the following context buttons available from the left pane:

Table 25: SiteScope dashboard context buttons

UI element	Description
Monitors	Used to create and manage SiteScope groups and monitors in a hierarchy represented by a monitor tree.
Remote Servers	Used to set up the connection properties so SiteScope can monitor systems and services running in remote Windows and UNIX environments.
Templates	Templates are used to deploy a standardized pattern of monitoring to multiple elements in the infrastructure. Preconfigured SiteScope solution templates are available, or personalized templates can be created and managed.
Preferences	Used to configure specific SiteScope administrative task properties and settings.

Server Statistics	Used to view key SiteScope server performance metrics.
Tools	Used to display diagnostic tools that help troubleshoot SiteScope problems and facilitate monitor configuration.

10.10.4.6 Analyzing data in SiteScope dashboard

This section describes how to analyze data in the SiteScope dashboard.

1. View monitor and measurement status and availability.

   When viewing SiteScope data in the **Current Status** view of the dashboard, you can explore the monitor tree to view monitor and measurement status and availability.

Example: Measurement status and availability for a monitor

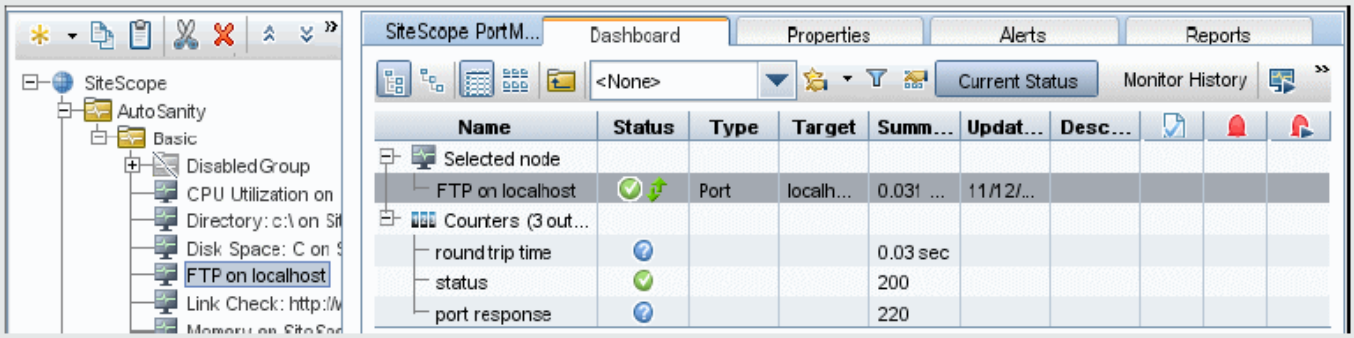


Figure 139: View monitor status in SiteScope dashboard

2. View configured and triggered alerts.

   The **Configured alerts** and **Triggered alerts** columns in the dashboard display information about the alerts. If alerts are configured for a monitor, double-clicking the **Configured Alert** icon displays a list of configured alerts. Selecting an alert lets you view or edit the alert properties.

Example: Configured alerts

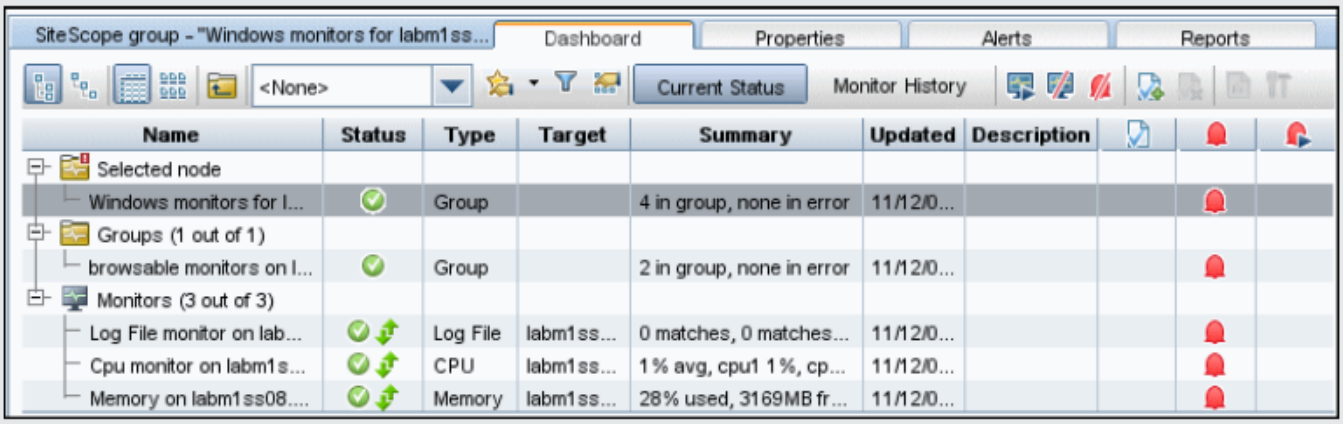


Figure 140: View configured and triggered alerts in SiteScope dashboard

3. Disable monitors in group.

   Depending on the diagnosis, you can disable a single monitor or monitors in a group, or disable the alerts associated with the monitor or group of monitors, and continue using the monitor.

4. Acknowledge monitors.

   There are two ways to acknowledge monitor status:

   a. Select a monitor or group of monitors and click the **Add Acknowledgment** icon .
   b. Select **Add Acknowledgment** from the context menu and the details in the **Acknowledge Monitors in Group** dialog box.

Example: Acknowledge Monitors In Group Dialog

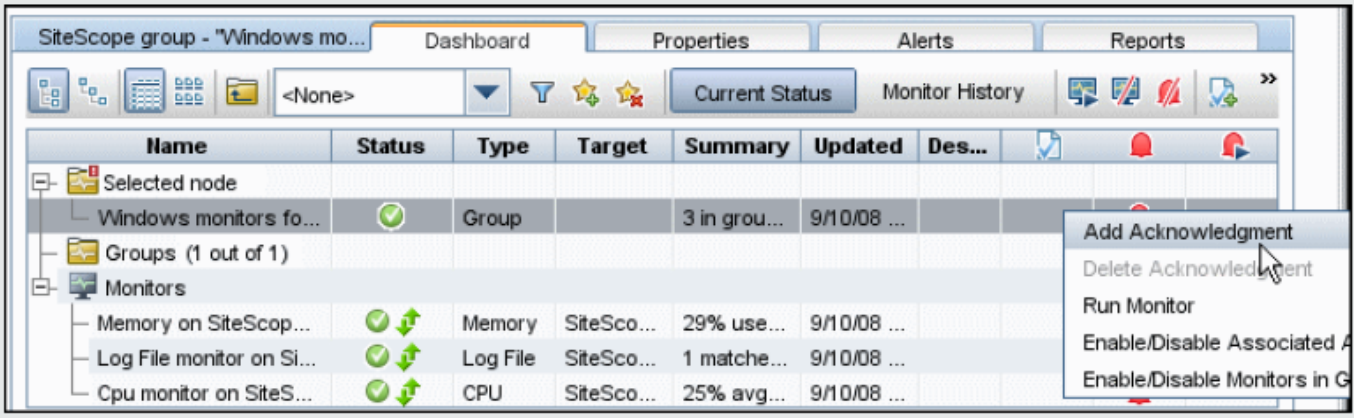


Figure 141: Acknowledging monitors in the SiteScope dashboard

NOTE: You can create a Microsoft Windows or UNIC Resources monitor to monitor the respective servers.

5. View monitor history.

   Monitor history is enabled and configured in the **General Preferences**. To view monitor history, click the **Monitor History** button in SiteScope Dashboard.

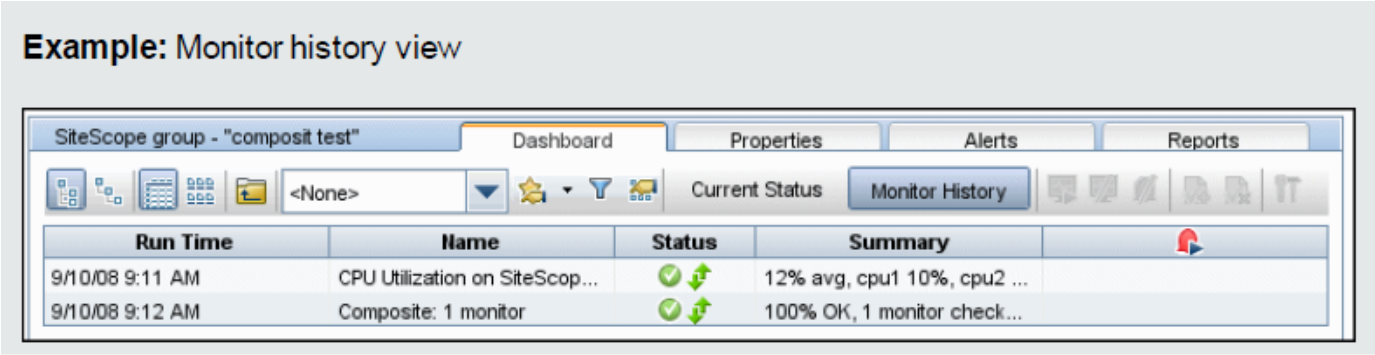


Figure 142: Viewing monitor history in the SiteScope dashboard

6. Dashboard status and availability levels

   The following table describes the different status and availability levels.

Table 26: Status and availability levels in the SiteScope dashboard

Icon	Meaning	Description
	Good status	All performance measurements are within the **Good** threshold level.
	Warning status	At least one performance measurement is within the **Warning** range, but no measurements are within the **Error** or **Poor** range.
	Error / Poor status	At least one performance measurement is within the **Error** or **Poor** range. This indicates that either: • the performance measurement has a value that is at poor quality level; or • there is no measurement value due to an error.
	Status Not Defined (no data)	There is no data for the group or monitor. This can be caused by any of the following: • A new monitor has not run yet. • Monitor counters have not yet been collected. • The monitors on which the group or monitor depend are not reporting a **Good** condition.
	No Threshold Breached status	No thresholds were defined for the monitor counter, so no status is assigned.

10.10.4.7 SiteScope templates and monitoring

SiteScope templates provide an enterprise solution for standardizing the monitoring of different IT elements in your enterprise, including servers, applications, databases, network environments, etc. You can use templates to rapidly deploy sets

of monitors that check systems in the infrastructure that share similar characteristics. You can create and customize your own templates to meet the requirements of your organization.

SiteScope templates are used to standardize a set of monitor types and configurations into a single structure. This structure can then be repeatedly deployed as a group of monitors targeting multiple elements in the network environments.

Templates speed up the deployment process of monitors across the enterprise through a single-operation deployment of groups, monitors, alerts, remote servers, and configuration settings.

IMPORTANT: Make sure that the monitor-run frequency is always greater than the time taken to scale-in/scale-out a VNF. Otherwise, multiple scale-in/scale-out requests might be sent for a single scale-in/scale-out condition.

CAUTION: In Fulfillment artifact templates, each Monitor artifact should be associated with a separate Monitor Handler artifact (even if the handler/hypervisor is the same). One-to-one mapping should be present between a Monitor artifact and a Monitor Handler artifact.

The following table describes the objects used in templates:

Table 27: SiteScope template objects

Icon	Object Type	Description
	Template Container	A template container enables you to manage your template monitoring solutions. You can only add a template to a template container.
	Template	The template contains the SiteScope group, monitors, remote servers, variable definitions, and alerts that make up the template monitoring solution.
	Template Variable	A variable is used to prompt for user input during template deployment. Template variables are either user-defined or predefined system variables.
	Template Remote Server	A template remote server is used to define Windows or UNIX remote server preferences that are created when the template is deployed.
	Template Group	A template group contains the template monitors and associated alerts. You use template groups to manage the deployment of monitors and associated alerts in your infrastructure.
	Template Monitor	Template monitors are used to define monitors that are created when the template is destroyed.
	Template Alert	Template alerts are used to define alerts on groups and monitors that are created when the template is deployed. If an alert has been set up for the template monitor or group, the alert symbol is displayed next to the monitor or group icon.

SiteScope provides template examples for monitoring in Windows and UNIX environments. These templates are available from the `Template Examples` folder in the template tree. You can use the template examples to help you use SiteScope templates.

The following illustration shows the **Windows basic template**. The template contains a template group, Windows monitors for %%host%%, two template monitors (CPU and Memory), four user-defined variables (frequency, host, password, and user), and a template remote server.

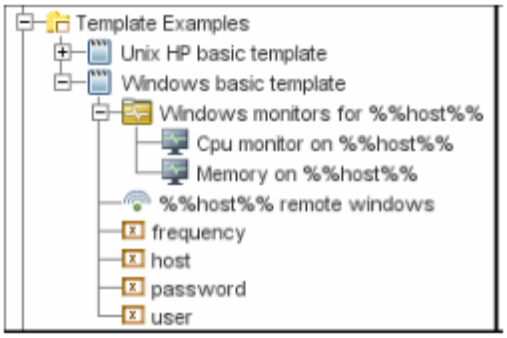


Figure 143: SiteScope sample template

A monitor template path is essential to deployment, because it decides which KPI has to be monitored. On the following illustration, the **Template Path** for CPU is `NFVDirector/VIRTUAL_MACHINE/KVM/CPU`. After triggering the respective template for deployment with the associated variables, it moves to Deployed state and this monitor can be accessed from **Monitors Context**.

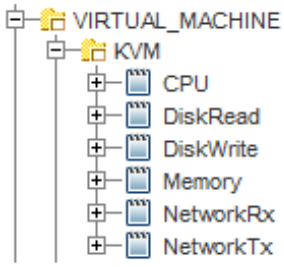


Figure 144: SiteScope monitor context

IMPORTANT: When configuring variables for Frequency and Error frequency in the **Monitor Run Settings**, the measurement unit for variable values can only be seconds.

NOTE: When a monitor is copied or moved from one template to another, any user-defined variables in the monitor are also copied or moved.

See list of Out of box monitors provided by NFV Director.

10.10.5 Alerts section

SiteScope alerts are notification actions that are triggered when the alert definition conditions are detected. You use an alert to generate a notification of an event or change of status in some element or system in your infrastructure.

Example: An alert can be triggered when a SiteScope monitor detects a change from Good to Error indicating that the monitored system has stopped responding.

Table 28: SiteScope alerts

Alert Type	Description
Error	An error alert will be triggered on breach of an error threshold condition and will be sent to the configured SNMP target if the breach has occurred at least 4 times.
Warning	A warning alert will be triggered on breach of a warning threshold condition and will be sent to the configured SNMP target if the breach has occurred at least 4 times.
Good	A good alert will be triggered on meeting a normal/safe threshold condition and will be sent to the configured SNMP target only if the monitored entity was previously in error condition.

If an alert is defined for a particular monitor, then it is activated on that monitor only. If an alert is defined for a template, then it is activated for all the monitors in the template.

`SNMPTarget` has to be configured in the **Preferences** section. Destination address and `port` have to be configured to map to the alert destination endpoint.

10.10.5.6 Configuring an alert

This task describes the steps involved in configuring an alert definition.

10.10.5.7 Prerequisites

Only a SiteScope administrator user or a user granted the appropriate alerts permissions can view, create, or edit alerts.

10.10.5.8 Creating/copying an alert

You can create a new alert or copy an existing alert into any group or monitor container in the SiteScope tree.

10.10.5.9 Creating a new alert

1. Right-click the container to which you want to associate the alert and select **New > Alert**.
2. Enter a name for the alert.
3. Select the targets to trigger the alert.
4. Configure an alert action.
5. In the **Alert Actions** panel, click **New Alert Action** to start the **Alert Action** wizard.

10.10.5.10 Copying an Alert Definition

1. In the **Alerts** tab, select the alert you want to copy.
2. Copy and paste it into the desired group or monitor container.

The alert target automatically changes to the group or monitor into which the alert is copied.

CAUTION: If you copy an alert definition from one group container to another, the Alert targets for the pasted alert are automatically reset to include all of the children of the container into which the alert is pasted. After pasting an alert, edit the alert definition properties to be sure that the assigned Alert targets are appropriate to the new alert context and your overall alerting plan.

10.10.5.11 Testing the alert

1. Select the alert in the **Alerts** tab of the monitor tree.
2. Click **Test**.
3. Select the monitor instance you want to test and click **OK**.
4. A dialog box opens with information about the alert test.

NOTE: The monitor you select does not have to be reporting the same status category that is selected to trigger the alert to test the alert. For example, the monitor does not have to currently be reporting an error to test an alert that is triggered by error conditions.

10.10.5.12 Disabling an alert - optional

You can disable alerts from the **Alerts** tab.

1. Select the alerts that you want to disable.
2. Click the **Disable** button.

Alerts disabled from the **Alerts** tab cannot be triggered; this overrides the associated alerts status set for a monitor in the monitor **Properties** tab or **Dashboard**.

10.10.6 SNMP preferences

You use **SNMP Preferences** to configure the settings SiteScope needs to communicate with an external SNMP host or management console. These are the default SNMP parameters for use with SNMP Trap alerts.

To access these parameters, select **Preferences context > SNMP Preferences**.

IMPORTANT: You must be an administrator in SiteScope, or a user granted View SNMP lists permissions to be able to view SNMP Preferences.

SNMP Preferences enable you to define settings that are used by SiteScope SNMP Trap alerts when sending data to management consoles. It also enables you to define SNMP Trap receivers, and listen to multiple local addresses and ports at the same time. SiteScope uses the SiteScope `SNMP Trap Alert type` to integrate with SNMP-based network management systems.

The following tables describe the user interface elements.

Table 29: SNMP user interface elements

UI Element	Description
General Settings	
Name	Name string assigned to the setting profile when creating a new SNMP recipient.

Description	Description for the setting profile, which appears only when editing or viewing its properties. You can include HTML tags such as , <HR>, and <B> to control display format and style. **NOTE:** HTML code entered in this box is checked for validity and security, and corrective action is taken to fix the code. For example, code is truncated if it spans more than one line. If malicious HTML code or JavaScript is detected, the entire field is rejected. The following is prohibited HTML content: • Tags: `script`, `object`, `param`, `frame`, `iframe`. • Any tag that contains an attribute string with `on` is declined. For example, `onhover`. • Any attribute with `javascript` as its value.
Preferences Settings: Main Settings Area	
Send to host	Domain name or IP address of the machine that receives all SNMP trap messages. This machine must be running an SNMP console to receive the trap message. Example: `snmp.mydomain.com` or `206.168.191.20`
SNMP port	SNMP port to which the trap is sent. Default value: `162`

Table 30: SNMP Preferences - SNMP Connection Settings

UI Element	Description
Preferences Settings: SNMP Connection Settings Area	
Timeout (seconds)	Amount of time, in milliseconds, to wait for the SNMP trap requests (including retries) to complete. Default value: 5
Number of retries	Number of times each SNMP trap GET request should be retried before SiteScope considers the request to have failed. Default value: 1
Community	Default SNMP community name used for sending traps. The community string must match the community string used by the SNMP management console. Default value: public
SNMP version	Default SNMP protocol version number to use. SNMP V1 and V2c are currently supported. Default value: V1
Authentication algorithm	Authentication algorithm used for SNMP V3. You can select MD5, SHA, or None. **NOTE:** This field is available only if SNMP V3 is selected.
User name	User name to be used for authentication is you are using SNMP V3.

	NOTE: This field is available only if SNMP V3 is selected.
Password	Password to be used for authentication if you are using SNMP V3. **NOTE:** This field is available only if SNMP V3 is selected.

Table 31: SNMP Preferences - Advanced Settings

UI Element	Description
Preference Settings: Advanced Settings Area	
SNMP trap ID	Select the type of trap to send. There are several predefined ID types for common conditions: • **Generic SNMP trap ID**. Select a generic SNMP type from the drop-down list. • **Enterprise-Specific SNMP trap ID**. To use an enterprise-specific SNMP ID type, enter the number of the specific trap type in the box. **NOTE:** When integrating SiteScope with NNMi, you must select **Enterprise-Specific SNMP trap ID** and enter **1**. SiteScope sends a different notification ID for each SNMP version: • SNMP V1: .1.3.6.1.4.1.11.15.1.4 • SNMP V2: .1.3.6.1.4.1.11.15.1.4.1
SNMP object ID	Identifies to the console the object that sent the message. • **Preconfigured SNMP object IDs**. Select one of the predefined objects from the drop-down list. • **Other SNMP object ID**. To use another object ID, enter the other object ID in the box. **NOTE:** • In SiteScope version 11.20 and later, all logged traps have an object ID that starts with a dot (.). For example, `oid=.1.3.6.1.2.1.0.1.3.6.1.4.1.11.2.17.1` • When integrating SiteScope with NNMi, select **Preconfigured SNMP object IDs** and choose **HP SiteScope Event** from the list.
Add System OID as a prefix to SNMP Trap	Adds the default system OID (1.3.6.1.2.1) as s prefix to all SNMP trap OIDs. Clear the checkbox if you do not want to use this prefix. Default value: Selected
SNMP source	The SNMP trap source: SiteScope Server or the monitor target server. Default value: Monitored Host

10.10.7 Sending SiteScope Alerts

SiteScope triggers the alert as soon as any monitor it is associated with matches the alert trigger condition. The trigger settings options in the Alert Action dialog box enable you to control when alerts are sent in relation to when a given condition is detected.

The following examples illustrate how different alert configurations send alerts after the error condition has persisted for more than one monitor run. If a monitor runs every 15 seconds and the alert is set to be sent after the third error reading, the alert is sent 30 seconds after the error was detected. If the monitor run interval is once every hour with the same alert setup, the alert is not sent until 2 hours later.

Example 1 - Always, after the condition has occurred at least N times:

Example 1a. An alert is sent for each time monitor is in error after condition persists for at least three monitor runs. Compare this with Example 1b below.

Alert setup	**Always, after the condition has occurred at least 3 times**								
sample interval	0	1	2	3	4	5	6	7	8
status	✓	✗	✗	✗	✗	✗	✓	✗	✗
count	c=0	c=1	c=2	c=3 alert!	c=4 alert!	c=5 alert!	c=0	c=1	c=2

Example 1b. An alert is sent for each time monitor is in error after condition persists for at least three monitor runs. Shows how the count is reset when the monitor returns one non-error reading between consecutive error readings. Compare this with Example 1a above.

Alert setup	**Always, after the condition has occurred at least 3 times**								
sample interval	0	1	2	3	4	5	6	7	8
status	✓	✗	✗	✓	✗	✗	✗	⚠	✓
count	c=0	c=1	c=2	c=0	c=1	c=2	c=3 alert!	c=0	c=0

Figure 145: SiteScope - send alerts always

Example 2 - Once, after the condition has occurred exactly N times:

An alert is sent only once if monitor is in error for at least three monitor runs, regardless of how long the error is returned thereafter.

Alert setup	**Once, after the condition has occurred exactly 3 times**								
sample interval	0	1	2	3	4	5	6	7	8
status	✔	✖	✖	✖	✖	✖	✖	✖	✖
count	c=0	c=1	c=2	c=3 alert!	c=4	c=5	c=6	c=7	c=8

Example 3 - Initially, after X times, and repeat every Y times:

Example 3a. An alert is sent on the fifth time monitor is in error and for every third consecutive error reading thereafter. Compare this with Example 3b below.

Alert setup	**Initially, after 5 times, and repeat every 3 times**								
sample interval	0	1	2	3	4	5	6	7	8
status	✔	✖	✖	✖	✖	✖	✖	✖	✖
count	c=0	c=1	c=2	c=3	c=4	c=5 alert!	c=6	c=7	c=8 alert!

Example 3b. An alert is sent on the third time monitor is in error and for every fifth consecutive error reading thereafter. Compare this with Example 3a above.

Alert setup	**Initially, after 3 times, and repeat every 5 times**								
sample interval	0	1	2	3	4	5	6	7	8
status	✔	✖	✖	✖	✖	✖	✖	✖	✖
count	c=0	c=1	c=2	c=3 alert!	c=4	c=5	c=6	c=7	c=8 alert!

Figure 146: SiteScope - send alert once

10.10.8 User management preferences

You can manage SiteScope user accounts from the **User Management Preferences** page. This page enables you to administer the users that are allowed access to SiteScope.

1. To access, select **Preferences context >User Management Preferences**.
2. In the **User Management Preferences** page, click the arrow next to the **New User** button and select **New User**.
3. In the **Main Settings** panel, enter the user name, login name, and password, and select the groups that can be accessed by this user profile.
4. Select the permissions to be granted to this user from the **Permissions** panel, or use the default permissions.

NOTE: All permissions are granted except the Add, Edit or Delete user permissions.

5. Click **OK**.
6. The new user profile is added to the **User Management Preferences** list.
7. The following table provides descriptions for the UI elements of the **User Management Preferences** page.

Table 32: User Management preferences

UI Element	Name	Description
	New	Click the arrow next to the button and select: • **New User**. Creates a new user profile. • **New User Role**. Creates a new user role profile.
	Edit	Enables editing the selected user or user role profile.
	Delete User/ User Role	Deletes the selected user or user role profile.
	Copy to User Role	Enables copying an existing SiteScope user's permissions to a new user role. **NOTE:** SiteScope users still need to have a user login and a security group assigned to them on the LDAP server (LDAP users have their own LDAP user name and password for logging on to SiteScope.
	Select All	Selects all listed user and user role profiles.
	Clear Selection	Clears the selection.

IMPORTANT: Only an administrator in SiteScope or a user with add, edit, or delete user preferences permissions can create or make changes to user settings and permissions for the current user or for other users.

By default, a regular user does not have add, edit or delete user preferences permissions. This means that they can view their own user properties only.

10.11