# Cisco Elastic Services Controller 6.0 ETSI NFV MANO User Guide

**First Published:** 2023-10-26

**Americas Headquarters**
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

# CONTENTS

# About This Guide

This guide helps you to perform tasks such as lifecycle management operations, monitoring, healing and scaling of the VNFs using the ETSI APIs.


- Audience, on page vii


# Audience

This guide is designed for network administrators responsible for provisioning, configuring, and monitoring VNFs. Cisco Elastic Services Controller (ESC) and the VNFs whose lifecycle it manages are deployed in a Virtual Infrastructure Manager (VIM). Currently OpenStack, VMware vCenter, VMware vCloud Director, and VMware NSX-T are the supported VIMs. The administrator must be familiar with the VIM layer, vCenter, OpenStack, and the commands used.

Cisco ESC is targeted for Service Providers (SPs) and Large Enterprises. ESC helps SPs reduce cost of operating the networks by providing effective and optimal resource usage. For Large Enterprises, ESC automates provisioning, configuring and monitoring of network functions.

## Terms and Definitions

The below table defines the terms used in this guide.

***Table 1: Terms and Definitions***

| Terms | Definitions |
|---|---|
| ESC | Elastic Services Controller (ESC) is a Virtual Network Function Manager (VNFM), performing lifecycle management of Virtual Network Functions. |
| ETSI | European Telecommunications Standards Institute (ETSI) is an independent standardization organization that has been instrumental in developing standards for information and communications technologies (ICT) within Europe. |
| ETSI Deployment Flavour | A deployment flavour definition contains information about affinity relationships, scaling, min/max VDU instances, and other policies and constraints to be applied to the VNF instance. The deployment flavour defined in the VNF Descriptor (VNFD) must be selected by passing the *flavour_id* attribute in the InstantiateVNFRequest payload during the instantiate VNF LCM operation. |

| Terms | Definitions |
|---|---|
| HA | ESC High Availability (HA) is a solution for preventing single points of ESC failure and achieving minimum ESC downtime. |
| KPI | Key Performance Indicator (KPI) measures performance management. KPIs specify what, how and when parameters are measured. KPI incorporates information about source, definitions, measures, calculations for specific parameters. |
| MSX | Cisco Managed Services Accelerator (MSX) is a service creation and delivery platform that enables fast deployment of cloud-based networking services for both Enterprises and Service Providers customers. |
| NFV | Network Function Virtualization (NFV) is the principle of separating network functions from the hardware they run on by using virtual hardware abstraction. |
| NFVO | NFV Orchestrator (NFVO) is a functional block that manages the Network Service (NS) lifecycle and coordinates the management of NS lifecycle, VNF lifecycle (supported by the VNFM) and NFVI resources (supported by the VIM) to ensure an optimized allocation of the necessary resources and connectivity. |
| NSO | Cisco Network Services Orchestrator (NSO) is an orchestrator for service activation which supports pure physical networks, hybrid networks (physical and virtual) and NFV use cases. |
| OpenStack Compute Flavor | Flavors define the compute, memory, and storage capacity of nova computing instances. A flavor is an available hardware configuration for a server. It defines the *size* of a virtual server that can be launched. |
| Service | A service consists of a single or multiple VNFs. |
| VDU | The Virtualisation Deployment Unit (VDU) is a construct that can be used in an information model, supporting the description of the deployment and operational behaviour of a subset of a VNF, or the entire VNF if it was not componentized in subsets. |
| VIM | The Virtualized Infrastructure Manager (VIM) adds a management layer for the data center hardware. Its northbound APIs are consumed by other layers to manage the physical and virtual resources for instantiation, termination, scale in and out procedures, and fault & performance alarms. |
| VM | A Virtual Machine (VM) is an operating system OS or an application installed on a software, which imitates a dedicated hardware. The end user has the same experience on a virtual machine as they would have on dedicated hardware. |
| VNF | A Virtual Network Function (VNF) consists of a single or a group of VMs with different software and processes that can be deployed on a Network Function Virtualization (NFV) Infrastructure. |
| VNFC | A Virtual Network Function Component is (VNFC) a composite part of the VNF, synonymous with a VDU, which could be implemented as a VM or a container. |
| VNFM | Virtual Network Function Manager (VNFM) manages the life cycle of a VNF. |

# Related Documentation

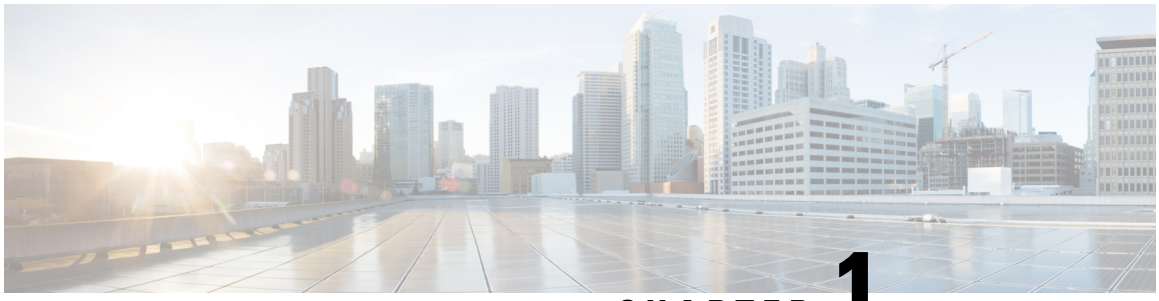
The Cisco ESC doc set comprises of the following guides to help you perform installation, configuration; the lifecycle management operations, healing, scaling, monitoring and maintenance of the VNFs using different APIs.

| Guide | Information Provided in This Guide |
|---|---|
| Cisco Elastic Services Controller Release Notes | Includes new features and bugs, known issues. |
| Cisco Elastic Services Controller Install and Upgrade Guide | Includes procedure for new installation and upgrade scenarios, pre and post installation tasks, and procedure for ESC High Availability (HA) deployment. |
| Cisco Elastic Services Controller User Guide | Includes lifecycle management operations, monitoring, healing and scaling of the VNFs. |
| Cisco Elastic Services Controller ETSI NFV MANO User Guide | Includes lifecycle management operations, monitoring, healing and scaling of the VNFs using the ETSI APIs. |
| Cisco Elastic Services Controller Administration Guide | Includes maintenance, monitoring the health of ESC, and information on system logs generated by ESC. |
| Cisco Elastic Services Controller NETCONF API Guide | Information on the Cisco Elastic Services Controller NETCONF northbound API, and how to use them. |
| Cisco Elastic Services Controller REST API Guide | Information on the Cisco Elastic Services Controller RESTful northbound API, and how to use them. |
| Cisco Elastic Services Controller ETSI REST API Guide | Includes information on the Cisco Elastic Services Controller ETSI APIs, and how to use them. |
| Cisco Elastic Services Controller Deployment Attributes | Includes information about deployment attributes used in a deployment datamodel. |
| Cisco Elastic Services Controller Open Source | Includes information on licenses and notices for open source software used in Cisco Elastic Services Controller. |

## Obtaining Documentation Request

For information on obtaining documentation, using the Cisco Bug Search Tool (BST), submitting a service request, and gathering additional information, see *What's New in Cisco Product Documentation*, at: http://www.cisco.com/c/en/us/td/docs/general/whatsnew/whatsnew.html.

Subscribe to *What's New in Cisco Product Documentation*, which lists all new and revised Cisco technical documentation, as an RSS feed and deliver content directly to your desktop using a reader application. The RSS feeds are a free service.



CHAPTER 1

# ETSI NFV MANO Northbound API Overview




# ETSI NFV MANO Northbound API Overview


The ETSI NFV MANO API (ETSI API) is another programmatic interface to ESC that uses the REST architecture. The ETSI MANO adheres to the standards defined by the European Telecommunications Standards Institute (ETSI), specifically around Management and Orchestration (MANO). The API accepts and returns HTTP messages that contain JavaScript Object Notation (JSON) payloads. The API contains its own datamodel designed around the ETSI MANO specifications that abstract away from the ESC core datamodel.

For information on VNF lifecycle management operations using the REST/NETCONF APIs, see the Cisco Elastic Services Controller User Guide.

***Table 2: ETSI MANO Specifications***

| Specification | Version Support | Description |
|---|---|---|
| SOL001 | v3.3.1 | Format and structure for the VNF Descriptor |
| SOL002 | v3.3.1 | Defines all interactions over the Ve-Vnfm reference point |
| SOL003 | v3.3.1 | Defines all interactions over the Or-Vnfm reference point |

**Note** The terminology used in the ETSI-specific sections of the user guide align to the ETSI MANO standards defined in the ETSI documentation. For more information, see the ETSI website.

For an orchestrator to check the versions of the APIs supported by a VNFM, a request can be made to the `/api_versions` endpoints. A version takes the form *MAJOR.MINOR.PATCH*; although only the MAJOR version appears in the URIs presented by the VNFM, the full version indicates the data model that the VNFM has implemented.

The Operations supported are:

ETSI NFV MANO Northbound API Overview

- Retrieve all supported versions for the given API
- Retrieve all supported versions for the given API, filtered on the major version

**Retrieve all supported versions :**

The request returns the version for the apiName supplied, showing the version, whether the version is deprecated, and optionally when the version will be retired.

Method Type:

POST

VNFM endpoint:

```
{apiRoot}/{apiName}/api_versions
```

HTTP Request Headers:

```
Content-Type:application/json
```

Response Body (ETSI data structure: ApiVersionInformation)

For example, for `vnffm`:

```
{
   "uriPrefix" : "localhost:8251/vnffm",
   "apiVersions" : [
       {
              "version" : "1.0.0",
              "isDeprecated" : true,
              "retirementDate" : "13-Jan-22"
       },
       {
              "version" : "1.3.0",
              "isDeprecated" : false
       }
   ]
}
```

**Retrieve all supported versions for a given major version:**

The request returns the version for the apiName supplied, showing the version, whether the version is deprecated and optionally when the version retires, filtered by the major version.

Method type:

```
POST
```

VNFM endpoint:

```
{apiRoot}/{apiName}/{apiMajorVersion}/api_versions
```

HTTP Request Headers:

```
Content-Type:application/json
```

Response Body (ETSI data structure: ApiVersionInformation)

For example, for `vnflcm` and major version=`2`:

```
{
             "uriPrefix" : "localhost:8251/vnflcm/v2",
             "apiVersions" : [
                {
```

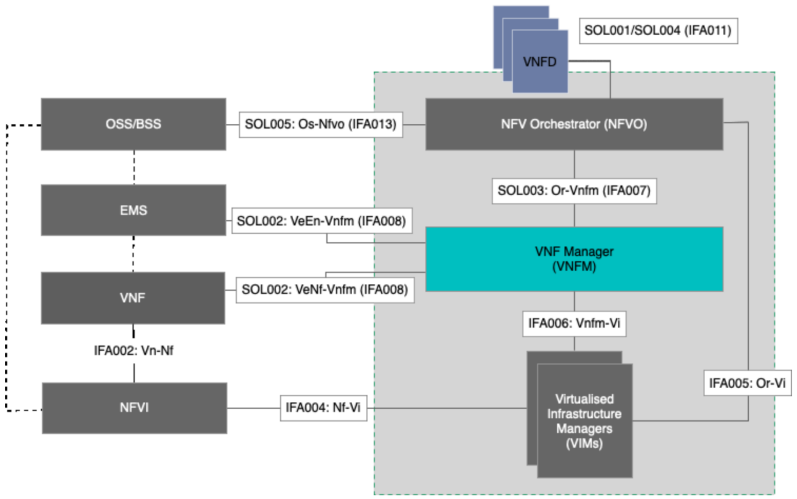
ETSI NFV MANO Northbound API Overview

```
                "version" : "2.0.0",
                "isDeprecated" : false
            }
        ]
    }
```

The current implementation of the ETSI NFV MANO standards consists of the Or-Vnfm and Ve-Vnfm reference points, which are the interfaces between the NFVO and VNFM, and the EM and the VNFM respectively. Both of these allow for the onboarding of ETSI-compliant CSAR packages, management of virtualized resources, and VNF lifecycle management (LCM) operations.

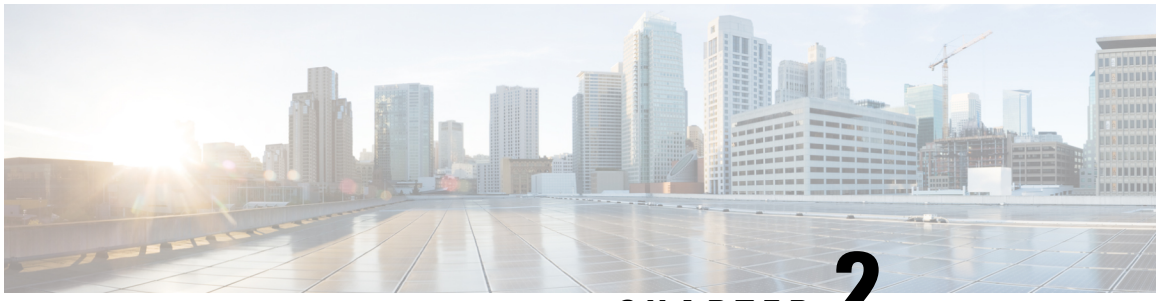
For more information on Or-Vnfm and Ve-Vnfm reference points, see the *ETSI Group Specification document* on the ETSI website. The figure below represents the NFV MANO architecture for all reference points.

***Figure 1: NFV MANO Architecture with Reference Points***




For information on managing resources, see Managing Resources, on page 5.

ETSI NFV MANO Northbound API Overview



CHAPTER 2

# Managing Resources



# Managing Resources

# Resource Definitions for ETSI API

Cisco Elastic Services Controller (ESC) resources comprise of images, flavours, tenants, volumes, networks, and subnetworks. These resources are the ones that ESC requests to provision a Virtual Network Function.

For ETSI MANO, these resource definitions are created by NFVO either at the time of onboarding the VNF package or onboarding the tenant, and represented by the VIM identifiers in the request to ESC.

For information on managing resources using NETCONF or REST APIs, see Managing Resources Overview in the Cisco Elastic Services Controller User Guide.

**ETSI API Documentation**

You can access the ETSI API documentation directly from the ESC VM:

```
http://[ESC VM IP]:8250/API
```

The ETSI API documentation provides details about all the various operations supported through the ESTI MANO interface.You can also see the Cisco ETSI API Guide for more information.

The following table lists the resource definitions on the VIM that must be made available before VNF instantiation.

*Table 3: Resource Definitions on VIM*

| Resource Definitions | OpenStack |
|---|---|
| Tenants | Out of band tenants<br>You can create a tenant using NETCONF API, REST API, or the ESC portal. You can also create a tenant directly on the VIM. The tenant is then referred to within the vimConnectionInfo data structure. For more information, see VIM Connectors Overview, on page 13. |
| Images | Out of band images<br>The NFVO onboards a VNF package, extracts and then onboards the image contained within the VNF package on to the VIM. Though the VNFD refers to the image file, because of the size of the image file, instead of onboarding the image at the time of deployment, the vimAssets in the Grant stipulates the image to be used. |
| Flavors | Out of band flavors<br>During onboarding of the VNF package, the NFVO looks at each cisco.nodes.nfv.Vdu.Compute node's capabilities in the VNFD to determine the flavor to be created. This is available later at the time of instantiation, or optionally overriden by a VIM flavor supplied at instantiation time as an additional parameter.<br>**Note** ETSI deployment flavour is a different concept than OpenStack compute flavor. For more information, see *Terms and Definitions* in About This Guide. |
| Volumes | ESC supports in-band volumes of type VirtualBlockStorage, as required by a deployment. It also supports out-of-band volumes as a Cisco extension to the ETSI specification. |
| External Networks (Virtual Link) | External networks are specified in the instantiation payload to which external connection points will connect. |
| Externally Managed Internal Virtual Links | Networks internal to the VNF are supported, as well as external networks specified in the instantiation payload to which internal virtual links will be bound instead of creating ephemeral networks. |
| Subnetworks | Out-of-band subnets |

For information on onboarding VNF packages and lifecycle operations using the ETSI API, see Managing the VNF Lifecycle, on page 25.

# Updating Resource Definitions

This section provides details about updating ETSI API resource definitions.

### Updating the VNF Flavour

You can define the alternate VNF nodes and deployment flavours for a single VNFD using the following TOSCA parameters:

- **Import statements**—The import statement allows a single, parent VNFD yaml file to conditionally include other files based on an input value which can be specified dynamically, at run time.
- **Substitution mappings**—The substitution mapping applies only to the node types derived from the *tosca.nodes.nfv.VNF*. You cannot substitute values of other node types that is, Connection Points, Virtual Links and so on.

Example1:

In this example, the yaml file contains three import files.

All three files must exist in the VNFD ZIP archive file in the same location as the parent file importing them.

The *requirements* and *capabilities* are not defined in the derived *tosca.nodes.nfv.VNF* node. These are mandatory for defining characteristics of VNFs instantiated using this VNFD. They are defined within the imported files.

```
tosca_definitions_version: tosca_simple_yaml_1_3
 description: Substitution Mapping Example

imports:
- df_default.yaml
- df_silver.yaml
- df_gold.yaml

. . .

node_types:
my-vnf:
derived_from: tosca.nodes.nfv.VNF

. . .

topology_template:

. . .

########################
# Substitution Mapping #
########################
substitution_mappings:
node_type: my-vnf
requirements:
 virtual_link: [ vm1_nic1, virtual_link ]

node_templates:

vnf:
type: my-vnf
```

```
properties:
descriptor_id: 8717E6CC-3D62-486D-8613-F933DE1FB3A0

. . .

flavour_id: default
flavour_description: Default VNF Deployment Flavour
```

Example 2:

When the VNF is instantiated, the required flavour is sent in the Instantiate request to the VNFM. The TOSCA parser tries to match the flavour and the VNF node name with the defined substitution mappings. These may be imported or defined within the VNFD itself. For example, the *df_silver.yaml* contains the following:

tosca_definitions_version: *tosca_simple_yaml_1_3*

description: Silver Deployment Flavour

imports:

```
topology_template:
substitution_mappings:
node_type: my-vnf
properties:
flavour_id: silver
flavour_description: Silver VNF Deployment Flavour
requirements:
- virtual_link: [ vm1_nic1, virtual_link ]
```

*silver* is the flavourId passed in the Instantiate Request payload. The parent *yaml* shown above has its empty *requirements* section updated with the *requirements* from the silver profile, and the existing *flavour_id* and *flavour_description* properties are updated as well.

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_3
description: Deployment Flavour SILVER
topology_template:
  substitution_mappings:
    node_type: tosca.nodes.nfv.VNF.CiscoESC
    requirements:
      virtual_link: [ anECP, external_virtual_link ]
    capabilities:
        deployment_flavour:
          properties:
            flavour_id: silver
            description: 'SILVER Deployment Flavour'
            vdu_profile:
              vdu_node_1:
                min_number_of_instances: 2
                max_number_of_instances: 2
            instantiation_levels:
              default:
                description: 'Default Instantiation Level'
                vdu_levels:
                  vdu_node_1:
                    number_of_instances: 1
                scale_info:
                  default_scaling_aspect:
                    scale_level: 2
              silver_level:
                description: 'SILVER Instantiation Level'
                vdu_levels:
                  vdu_node_1:
                    number_of_instances: 2
```

```
            scale_info:
              default_scaling_aspect:
                scale_level: 2
          default_instantiation_level_id: default
          vnf_lcm_operations_configuration: {}
          scaling_aspect:
            - default_scaling_aspect
          cisco_esc_properties:

    description: "SILVER: This is substituted if not already defined"
```

ESC sends a POST request to update the VNF flavour:

Method Type:

**POST**

VNFM Endpoint:

```
/vnflcm/v2/vnfinstances/{vnfInstanceId}/change_flavour
```

### Updating the External VNF Connectivity

You can update the external VNF connectivity in an existing deployment. The API supports the following changes:

- Disconnect the existing connection points (CPs) to the existing external virtual link and connect to a different virtual link.
- Change the connectivity parameters of the existing external CPs, including changing the addresses.

ESC sends a POST request to update the VNF external connectivity:

Method Type

**POST**

VNFM Endpoint

```
/vnflcm/v2/vnfinstances/{vnfInstanceId}/change_ext_conn
```

Request Payload (Data structure = ChangeExtVnfConnectivityRequest)

```
{
  "extVirtualLinks": [
    {
      "id": "extVL-98345443-7797-4c6d-a0ed-e18771dacf1c",
      "resourceId": "node_1_ecp",
      "extCps": [
        {
          "cpdId": "node_1_ecp",
          "cpConfig": {
              cp1 : {
              "cpProtocolData": [
                {
                  "layerProtocol": "IP_OVER_ETHERNET",
                  "ipOverEthernet": {
                    "ipAddresses": [
                      {
                        "type": "IPV4",
                        "numDynamicAddresses": 2,
                        "subnetId": "esc-subnet"
                      }
```

```
                        ]
                    }
                }
            ]
        }
    }
  }
 ]
}
```

**Note** The id in the extVirtualLinks, *extVL-98345443-7797-4c6d-a0ed-e18771dacf1c* in the above example, must also exist in the instantiatedVnfInfo in the vnfInstance.

### Merging Policy

The substitution merges the new values into the VNFD.

1. For regular scalar properties such as name=joe, the value is replaced in the VNFD.
2. Arrays such as [list, of, strings] are merged. The new values are added into the array, if they do not exist.
3. Objects such as where a key is indented under another key, are replaced. The configurable_properties object in the matched substitution will overwrite that defined in the VNFD.

### Parser Behaviour

- After the substitution mappings are made, the parser tries to populate any *additionalParams* provided. Note that the command fails if the input parameters do not match those in the template.

For more information on VNF lifecycle operations, see Managing the VNF Lifecycle, on page 25.

# OAuth (Open Authorization) 2.0 Authentication

The ETSI NFV MANO supports OAuth 2.0 authentication for SOL003 Or-Vnfm reference point. The NFVO makes a token request to ESC providing the client credentials such as client id and client secret for authentication. In turn, ESC verifies the request and returns the access token.

**Note** ETSI supports both basic authentication as well as subscriptions for NFVO connections over SOL003.

The NFVO makes a POST request providing the client id and secret as primary authentication.

Method Type

**POST**

URL

```
{apiRoot}/oauth2/token
```

Header

```
Authorization: Basic {base 64 encoded CLIENT_ID:CLIENT_SECRET}
Accept: application/json
Content-Type: application/x-www-form-urlencoded
```

Body

```
grant_type=client_credentials
```

ESC returns the access token in response.

Example:

```
{
    "access_token":
"eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJjaHJpcyIsImlzcyI6IkVUU0ktVk5GTSIsImlhdCI6MTU1ODYwMzk2NiwiZXhwIjoxNTU4NjA0NTY2f


Q.lAtre7vdCKJjgzNs7p9P3NS2qMcXegC-oWXmy5Kakn0AL95gLWF6liOqPViMZNnWZLOsG5r1kPnGoBWnN0tgIw",
    "token_type": "bearer",
    "expires_in": 600
}
```

The access token is then used to access the or_vnfm endpoints.

Example:

Method

**GET**

URL

```
{apiRoot}/vnflcm/v2/subscriptions
```

Headers

```
Authorization: Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJjaHJpcyIsImlzcyI6IkVUU0k
tVk5GTSIsImlhdCI6MTU1ODYwMzk2NiwiZXhwIjoxNTU4NjA0NTY2fQ.lAtre7vdCKJjgzNs
7p9P3NS2qMcXegC-oWXmy5Kakn0AL95gLWF6liOqPViMZNnWZLOsG5r1kPnGoBWnN0tgIw
```

**Note** The existing tokens become invalid if the ETSI service is restarted.

### Accessing and Updating the OAuth Properties File

ESC stores the client id and secret in the new *etsi-production.yaml* properties file in the same location as the *etsi-production.properties* file. The new escadm etsi commands are available to maintain the client id and secret values. The client secret is encrypted the same way as the existing rest username.

#### To add or update a client id

```
sudo escadm etsi oauth2_clients --set <CLIENT_ID>:<CLIENT_SECRET>
```

#### To remove a client id

```
sudo escadm etsi oauth2_clients --remove <CLIENT_ID>
```

**Note** Restart the ETSI services after updating the OAuth 2.0 values.

For information on other properties, see ETSI Production Properties, on page 127.

### OAuth Calls from ETSI to the NFVO

ESC supports OAUTH 2.0 calls from ETSI to the NFVO.

The following properties are added to the etsi-production.properties file:

```
nfvo.clientID=<YourClientID>
nfvo.clientSecret=<YourClientSecret>
nfvo.tokenEndpoint=<Your NFVO Token Endpoint>
nfvo.authenticationType=OAUTH2
```

The Client id, ClientSecret and TokenEndpoint must match that of the OAUTH 2.0 Server. The authentication type determines authentication of the outgoing calls from ESC to the NFVO. The authentication type must be either BASIC, or OAUTH2.

The tokens from the NFVO are stored against the token endpoint in the properties file.

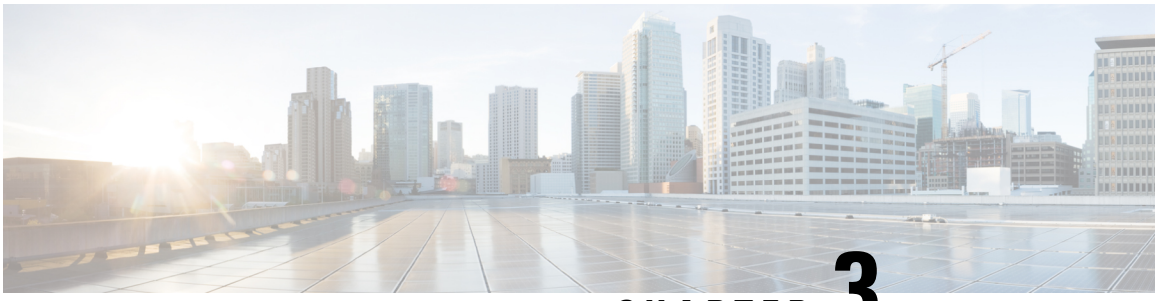
When the NFVO sends a call request, ETSI checks for the tokens stored against the token endpoint. If the token has not expired, then ETSI adds the old token to the header of the request and executes the call. A new token is required if the token fails to execute.

If there are no tokens against the token endpoint, then new tokens are required to execute the call.

### OAuth 2.0 Notification and Subscription

The subscription payloads must add the following to enable OAuth 2.0 authentication with the notifications:

```
{
  "authentication": {
    "authType": [
      "OAUTH2_CLIENT_CREDENTIALS"
    ],
    "paramsOauth2ClientCredentials": {
      "clientId": <client_id>,
      "clientPassword": <client_secret>,
      "tokenEndpoint": <token_endpoint>
    }
  }
}
```



CHAPTER 3

# Managing VIM Connectors



## VIM Connectors Overview

The ETSI API creates VIM connectors during the processing of an LCM operation or uses an existing connector.

The Grant response or the LCM operation request from the NFVO supplies new VimConnectionInfo to the VnfInstance. During the processing of the LCM operation, ETSI synchronizes the new *VimConnectionInfo* with the VIM connectors in ESC.

A *VimConnectionInfo* is new if the VnfInstance does not have an existing VimConnectionInfo with the same id. Any *VimConnectionInfo* supplied that matches an existing *VimConnectionInfo* id stored against any VnfInstance as part of an LCM request uses the existing connector and ignore any changes submitted in the new request.

ESC creates a new VIM connector only if a matching VIM connector is not available.

The ETSI API allows only the existing *VimConnectionInfo*, and the associated VIM connector, to be updated via the Modify VNF information operation.

The Grant from the NFVO specifies the vimConnectionId for each resource. This value identifies the VimConnectionInfo and the associated VIM connector for creating the locator for each resource. The VIM specific *VimConnectionInfo.accessInfo* properties are set as additional properties in the locator.

Example for VimConnectionInfo in OpenStack:

```
{
  ,
  "vimType": "OPENSTACK_V3",
  "interfaceInfo": {
    "endpoint": "https://10.18.54.42:13001/v3/"
  },
  "accessInfo": {
    "username": "admin",
    "password": "bmkQJtyDrbPFnJT8ENdZw2Maw",
    "project": "cbamnso",
    "projectDomain": "Default",
    "userDomain": "Default",
```

```
    "vim_project": "cbamnso"
  }
}
```

Example for VimConnectionInfo in VMware Cloud Director:

```
{
  ,
  "vimType": "VMWARE_VCD",
  "interfaceInfo": {
    "endpoint": "https://10.85.103.150"
  },
  "accessInfo": {
    "username": "admin@cisco",
    "password": "bmkQJtyDrbPFnJT8ENdZw2Maw",
    "vim_project": "cbamnso",
    "vim_vdc": "vdc1"
  }
}
```

Example for VimConnectionInfo in VMware vCenter

```
{

  "vimType": "VMWARE_VSPHERE",
  "interfaceInfo": {
    "endpoint": "https://10.85.103.21"
  },
  "accessInfo": {
    "username": "admin@vsphere.local",
    "password": "bmkQJtyDrbPFnJT8ENdZw2Maw",
    "vim_project": "cbamnso",
    "vim_vdc": "vdc1"
  }
}
```

For VIM Connector Status and SNMP Trap Notifications, see the Cisco Elastic Services Controller Administration Guide.

# Creating New VIM Connectors

During the ETSI LCM operation, ESC checks each *VimConnectionInfo* against the existing VIM connector records. If an existing VIM connector is not available, ESC creates a new VIM connector.

If the *VimConnectionInfo.vimId* is supplied, then this value is used as the id of the new VIM connector. If the *VimConnectionInfo.vimId* is not supplied, then an id is generated for the new VIM connector and this value is also set as the *VimConnectionInfo.vimId*.

To use an existing VIM connector, see Using an Existing VIM Connector, on page 14.

VIM connectors to many VIMs of different types are supported by a single instance of ESC.

# Using an Existing VIM Connector

During an ETSI LCM operation, ESC checks for an existing vimConnectionInfo with a matching identifier stored against any VnfInstance.

Existing VIM connectors are found by:

- Matching the *VimConnectionInfo.vimId*, if supplied, to the id of a VIM connector.
- Matching the VIM specific properties of the *VimConnectionInfo* to a VIM connector.
  - OpenStack
    - vimType
    - interfaceInfo.endpoint
    - accessInfo.project
  - VMware Cloud Director or vCenter
    - vimType
    - interfaceInfo.endpoint

If a matching VIM connector is found, and the *VimConnectionInfo.vimId* is not set, then the *VimConnectionInfo.vimId* is set to the id of the VIM connector.

If an NFVO provides a VimConnectionInfo with accessInfo to stipulate some of the connection properties, we use the following keys to configure the VIM connectors:

| accessInfo Property | OpenStack | Cloud Director | vCenter |
|---|---|---|---|
| username | Yes | Yes | Yes |
| password | Yes | Yes | Yes |
| project | Yes | | |
| vim_project | Yes | Yes | Yes |
| projectDomain | Yes | | |
| userDomain | Yes | | |
| vim_vdc | | Yes | Yes |

The ETSI specifications does not specify the keys to be used as part of the accessInfo attribute. In order to ease integration, in the event that an NFVO uses different keys, the properties file allows the user to specify a mapping from the third party keys to the ones that ESC understands.

```
mapping.vimConnectionInfo.accessInfo.username
mapping.vimConnectionInfo.accessInfo.password
mapping.vimConnectionInfo.accessInfo.project
mapping.vimConnectionInfo.accessInfo.projectDomain
mapping.vimConnectionInfo.accessInfo.userDomain
mapping.vimConnectionInfo.accessInfo.vim_project
mapping.vimConnectionInfo.accessInfo.vim_vdc
```

To create a new VIM connector, see Creating New VIM Connectors, on page 14.

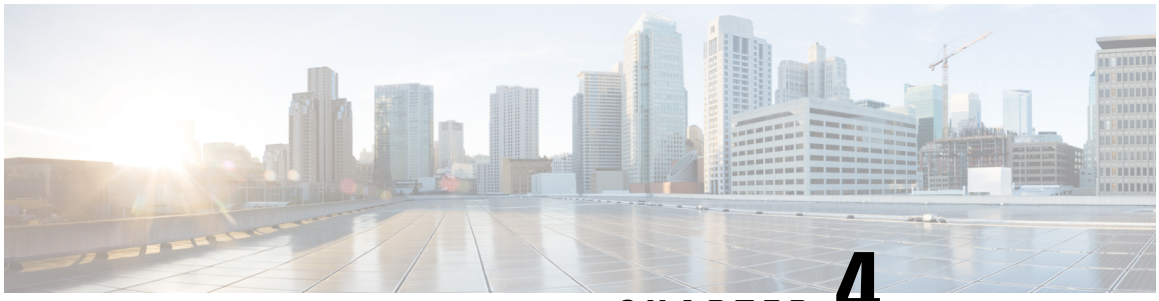
# Updating the VIM Connector

The ETSI API updates the existing VimConnectionInfo, and the associated VIM connector via the Modifying Virtual Network Functions, on page 41 operation. The *VimConnectionInfo* in the modify request payload is compared to the existing *VimConnectionInfo* stored against the VnfInstance.

If an existing VimConnectionInfo stored against any VnfInstance with a matching id is not found, then then VimConnectionInfo is added to the VnfInstance.

If an existing *VimConnectionInfo* stored against any VnfInstance with a matching id is found, then the VimConnectionInfo is updated. If the VimConnectionInfo has been modified and it has an associated VIM connector, then the VIM connector is also updated.

To create new VIM connectors, see Creating New VIM Connectors, on page 14.



CHAPTER 4

# Understanding Virtual Network Function Descriptors



## Virtual Network Function Descriptor Overview

ESC supports a TOSCA-based Virtual Network Function Descriptor (VNFD) to describe the VNF characteristics. The VNFD conforms to the *GS NFV-SOL 001 v.3.3.1* specifications and standard specified by ETSI (which in turn implements *TOSCA Simple Profile in YAML Version 1.3*).

The VNFD file describes the instantiation parameters and operational behaviors of the VNFs, their internal topology as well as their external connectivity. It also contains KPIs and other key requirements that can be used in the process of onboarding and managing the lifecycle of a VNF.

For VNF Lifecycle operations, see VNF Lifecycle Operations, on page 26.

## Defining Extensions to the Virtual Network Function Descriptor

The VNFM implements extensions to the VNFD to expose the more advanced concepts supported by ESC that are not explicitly defined in the ETSI standards. These extensions have been implemented in an ETSI-compliant way to ensure maximum compatibility with other ETSI NFV MANO components.

If there is a requirement to control these properties on a per-deployment basis, then replace the hard-coded values with inputs in the VNFD that can be supplied as *additionalParams* in the incoming request.

**VNFCs (tosca.nodes.nfv.Vdu.Compute)**

The Compute node allows for many of the ESC features to be exposed via the extended `tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties`. This includes the following:

- Overriding the automatically generated name for a VNFC on the VIM.
- VIM flavor (overriding the ETSI capabilities specified for a VNFC).
- Supplying ESC with an expected bootup time to prevent further actions being taken until this timer has expired.

- Providing Day-0 configuration blocks to execute/store on the VNFC once deployed.
- Specifying KPI parameters and associated rules to configure the monitoring agent.
- Intra-VM Group placement rules.

Here is the data type extension definition:

```
data_types:
  ...
  cisco.datatypes.nfv.VnfcAdditionalConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties
    properties:
      vim_flavor:
        type: string
        required: true
      bootup_time:
        type: integer
        required: true
      vm_name_override:
        type: string
        required: false
      recovery_action:
        type: string
        required: true
      recovery_wait_time:
        type: integer
        required: true
      monitor_on_error:
        type: boolean
        description: Continue monitoring of VNFC on error state.
        required: false
      max_retries:
        type: integer
        description: The number of recovery attempts
        required: false
      kpi_data:
        type: map # key: event_name
        description: The different KPIs applicable to this VDU
        required: false
        entry_schema:
          type: cisco.datatypes.nfv.data.Kpi
          description: A single KPI
      admin_rules:
        type: map # key: event_name
        description: Actions for events
        required: false
        entry_schema:
          type: cisco.datatypes.nfv.data.Admin_rules
          description: Define actions for events
      name_override:
        type: string
        description: An optional custom name that can be configured on the VIM
        required: false
      vendor_section:
        type: cisco.datatypes.nfv.VendorExtension
        required: false

  cisco.datatypes.nfv.VnfcConfigurableProperties:
    derived_from: tosca.datatypes.nfv.VnfcConfigurableProperties
    properties:
      additional_vnfc_configurable_properties:
        type: cisco.datatypes.nfv.VnfcAdditionalConfigurableProperties
        required: false
```

```
node_types:
  cisco.nodes.nfv.Vdu.Compute:
    derived_from: tosca.nodes.nfv.Vdu.Compute
    properties:
      configurable_properties:
        type: cisco.datatypes.nfv.VnfcConfigurableProperties
        description: Describes the configurable properties of all VNFC instances based on
this VDU
        required: false
```

For example:

```
vdu1:
      type: tosca.nodes.nfv.Vdu.Compute
      properties:
        name: Example VDU1
        description: Example VDU
        boot_order: true
                  configurable_properties:
          additional_vnfc_configurable_properties:
            vim_flavor: Automation-Cirros-Flavor
            bootup_time: 1800
            vm_name_override: my-vdu-1
            recovery_action: REBOOT_THEN_REDEPLOY
            recovery_wait_time: 100
            monitor_on_error: false
            max_retries: 2
            kpi_data:
              VM_ALIVE-1:
                event_name: 'VM_ALIVE-1'
                  metric_value: 1
                  metric_cond: 'GT'
                  metric_type: 'UINT32'
                  metric_occurrences_true: 1
                  metric_occurrences_false: 30
                  metric_collector:
                    type: 'ICMPPing'
                    nicid: 1
                    poll_frequency: 10
                    polling_unit: 'seconds'
                    continuous_alarm: false
            admin_rules:
              VM_ALIVE-1:
                event_name: 'VM_ALIVE-1'
                action:
                  - 'ALWAYS log'
                  - 'FALSE recover autohealing'
                  - 'TRUE esc_vm_alive_notification'
            placement_type: zone
            placement_target: nova
            placement_enforcement: strict
            vendor_section:
              cisco_esc:
                config_data:
                  example.txt:
                    file: ../Files/Scripts/example.txt
                      variables:
                        DOMAIN_NAME: { get_input: DOMAIN_NAME }
                        NAME_SERVER: { get_input: NAME_SERVER }
                        VIP_ADDR: { get_input: VIP_ADDR }
                        VIP_PREFIX: { get_input: VIP_PREFIX }
        vdu_profile:
          min_number_of_instances: 1
          max_number_of_instances: 1
```

```
        capabilities:
    virtual_compute:
      properties:
        virtual_cpu:
          num_virtual_cpu: 8
        virtual_memory:
          virtual_mem_size: 16
  requirements:
    - virtual_storage:  cdr1-volume
    - virtual_storage:  boot1-volume
```

If *vm_name_override* is not specified, ESC will auto-generate the VM names.

ESC stores the VNFC specific value in *VnfInstance.instantiatedVnfInfo.vnfcResourceInfo.metadata.vim_vm_name* for the VNFC identified by the vduId, which matches the label given to the Compute node representing the VNFC.

**Note** You can supply a high number of input parameters, allowing the use of a single template for multiple deployments.

**Connection Points (tosca.nodes.nfv.VduCp)**

The Cisco extensions to the VduCp node type mainly allows for defining the interface requirements map. The features added to the connection point are as follows:

- Overriding the automatically generated name for a port on the VIM
- Identification of whether the port is a management port (i.e. used for monitoring)
- Allowed Address Pairs
- Support for specific network card types and interface types, e.g. SR-IOV
- Support for port binding profiles
- Whether port security is enabled

For example:

```
vdu1_nic0:
      type: tosca.nodes.nfv.VduCp
      properties:
        layer_protocols: [ ipv6 ]
        protocol:
          - associated_layer_protocol: ipv6
        trunk_mode: false
        order: 0
        virtual_network_interface_requirements:
          - support_mandatory: true
            network_interface_requirements:
                    allowed_address_pairs: { get_input: VDU1_NIC0_AADR_PAIRS }
                    nw_card_model: virtio
                    iface_type: direct
                    management: true
                    name_override: my-vdu1-nic0
                    port_security_enabled: false
                    binding_profile:
                      trusted: true
        requirements:
         - virtual_binding: vdu1
```

ESC supports SR-IOV properties using the network interface requirements. You can configure the interface to associate the VNFC with an SR-IOV pass-through adapter by specifying the type as direct, as per the previous example.

If there is a requirement to control these properties on a per-deployment basis, then replace the hard-coded values with inputs in the VNFD that can be supplied as *additionalParams* in the incoming request, as per the allowed address pairs above.

**Note** The port binding profile is available for Pike and later versions of OpenStack.

**Volumes (tosca.nodes.nfv.Vdu.VirtualBlockStorage)**

ESC supports out-of-band volumes as a Cisco extension. This allows the specification of the persistent volume UUID as the `resourceId` property against the `VirtualBlockStorage` node to be used in place of the ephemeral volume defined in the VNFD. ESC allows the request to override the volume specified in the VNFD and supplies its own persistent (deployed out-of-band) storage by identifying it with a UUID from the VIM.

For example:

```
boot1-volume:
      type: tosca.nodes.nfv.Vdu.VirtualBlockStorage
      properties:
        virtual_block_storage_data:
          size_of_storage: 4GB
          vdu_storage_requirements:
            resource_id: { get_input: VDU1_BOOT_VOL_UUID }
            vol_id: 1
            bus: ide
            type: LUKS
        sw_image_data:
          name: 'Automation_Cirros'
          version: '1.0'
          checksum: 9af30fce37a4c5c831e095745744d6d2
          container_format: bare
          disk_format: qcow2
          min_disk: 2 GB
          size: 2 GB
      artifacts:
        sw_image:
          type: tosca.artifacts.nfv.SwImage
          file: ../Files/Images/Automation-Cirros.qcow2
```

**Note** The VNFD accepts the volume or software image size in mebibyte-based units such as MiB, GiB or TiB equivalent. If the volume or software image size is in megabyte-based units such as MB, GB or TB, ESC converts the size to mebibyte-based equivalent and adjusts to the nearest value. Ensure you use mebibyte-based units for volume or software image size for clarity.

**Security Group Rule (tosca.nodes.nfv.VduCp)**

As per the handling a persistent of the volume above, ESC provides the ability to specify an out-of-band security group instead of configuring one in the VNFD. This is because the verbs used to describe the security group in the standards documentation are too simplistic for a very complicated configuration. Since the security group is being specified for use on a connection point, this is where it is defined in the VNFD.

For example:

```
c1_nic0:
  type: tosca.nodes.nfv.VduCp
  properties:
    order: 0
    layer_protocols: [ ipv6 ]
    protocol:
      - associated_layer_protocol: ipv6
    trunk_mode: false
    virtual_network_interface_requirements:
      - support_mandatory: true
        network_interface_requirements:
          management: "false"
          iface_type: "virtual"
  metadata:
    security_groups: { get_input: VIM_NETWORK_SEC_GRP_0 }
  requirements:
    - virtual_binding: c1
```

**Virtual Links (tosca.nodes.nfv.VnfVirtualLink)**

The virtual links defined in the VNFD can be used to define those physical provider networks.

For example:

```
 vpc-di-internal1:
  type: tosca.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4 ]
    description: DI Internal 1 Network VL
    vl_profile:
      max_bitrate_requirements:
        root: 100000
      min_bitrate_requirements:
        root: 0
      virtual_link_protocol_data:
        - associated_layer_protocol: ethernet
          l2_protocol_data:
            vlan_transparent: yes
            network_type: vlan
            segmentation_id: { get_input: VL1_SEG_ID }
            physical_network: vlan_network
```

They can also be used to specify the IP subnets that an internal connection point may use when using DHCP to assign an address to them.

For example:

```
vpc-di-internal2:
  type: tosca.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ipv4 ]
    description: DI Internal 1 Network VL
    vl_profile:
      max_bitrate_requirements:
        root: 100000
      min_bitrate_requirements:
        root: 0
      virtual_link_protocol_data:
        - associated_layer_protocol: ipv4
          l3_protocol_data:
            ip_version: ipv4
```

```
            cidr: 1.180.10.0/29
            dhcp_enabled: true
```

For information on lifecycle management operations, see Managing the VNF Lifecycle.

**Note** The previous versions of ESC supported Cisco-only extensions to support the above functionality. These extensions were outside of the specification and although now these extensions are largely conformant with the SOL001 standard, the previous definitions are still supported by ESC for backward compatibility. For more information, see the Cisco Elastic Services Controller 5.5 documentation.

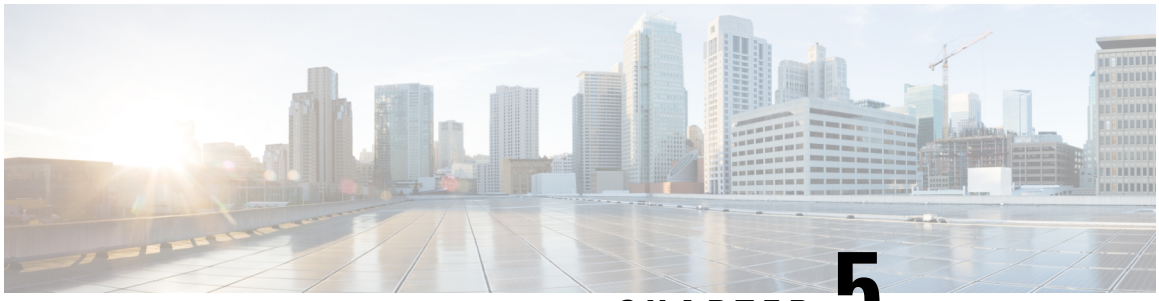
### Package Change Policy (tosca.policies.nfv.VnfPackageChange)

In order to support the change of a VNF package without redeploying the VNF instance, ETSI defines the Change Current VNF Package endpoint which allows the VNF package to be swapped for a new one. For example, a SOL004 package is immutable once created. The following policies ensure that only desired upgrades and downgrades pass the validation in the VNFM.

```
type: tosca.policies.nfv.VnfPackageChange
        properties:
          selector:
            source_descriptor_id: f5699972-3d35-4679-b2e7-19633154bd8d2
            destination_descriptor_id: 0628204d-3a29-4133-9f2b-7b26f76ef88d
            source_flavour_id: default
          modification_qualifier: up
          destination_flavour_id: small

type: tosca.policies.nfv.VnfPackageChange
        properties:
          selector:
            source_descriptor_id: 0628204d-3a29-4133-9f2b-7b26f76ef88d
            destination_descriptor_id: f5699972-3d35-4679-b2e7-19633154bd8d2
            source_flavour_id: small
          modification_qualifier: down
          destination_flavour_id: default
```

Although the source and destination descriptor IDs are validated, no other constructs are considered as part of the validation of the current implementation of this API.



CHAPTER 5

# Managing VNF Lifecycle Operations



## Managing the VNF Lifecycle

The NFVO communicates with ESC using the ETSI MANO API for lifecycle management of a VNF. A configuration template, the Virtual Network Function Descriptor (VNFD) file describes the deployment parameters and operational behaviors of a VNF type. The VNFD is used in the process of deploying a VNF and managing the lifecycle of a VNF instance.

The lifecycle operations of a VNF instance is as follows:

1. **Create a VNF Identifier**—ESC generates a new VNF Instance Id (a universally unique identifier) that is subsequently used as a handle to reference the instance upon which to execute further operations.
2. **Instantiate / Deploy VNF**—As part of VNF instantiation, ESC instantiates a new VNF instance in the VIM. ESC receives a request to instantiate a VNF instance from NFVO. The instantiate request contains resource requirements, networking and other service operational behaviors. All these requirements along with the VNFD and the grant information provides all the necessary information to instantiate the VNF.
3. **Operate VNF**—ESC allows you to start and stop a VNF instance. The resources are not released or changed, but the VNF instance in the VIM is toggled between these two states.
4. **Query VNF**—To query one or more VNF instances known to ESC. This is a specific REST end point that can be filtered to find specific instances. The instances can be filtered using the VNF Instance Id.

   Also, a separate REST end point allows the NFVO to query the status of one or more lifecycle operation occurrences associated with a VNF. The lifecycle operations can be filtered using a specific occurrence identifier.
5. **Modify VNF**—ESC allows you to modify the properties of a single VNF instance. The instantiated VNF is updated, and the lifecycle management operation occurrence sends notification to the NFVO about the status of the VNF.
6. **Scale and Scale to Level VNF**—ESC allows you to scale VNFs in two ways. You can scale a VNF incrementally, or to a specific level.
7. **Heal VNF**—ESC heals the VNF when there is a failure.

8. **Terminate / Undeploy VNF**—To terminate the VNF instance in the VIM. The resources themselves remain reserved for the VNF instance, however the VNF itself is undeployed.
9. **Delete VNF Identifier**—The resources are fully released in the VIM and in ESC and the associated VNF instance identifer is also released.
10. **Change Current VNF Package** – To change the package upon which the VNF instance is defined without a redeploy of the instance.

For VNF lifecycle operations using REST and NETCONF APIs, see Configuring Deployment Parameters in the Cisco Elastic Services Controller User Guide.

The ESC health monitor API can determine the connectivity of ESC to the NFVO and send appropriate status notifications. For more details, see *Monitoring ESC Health* in the Cisco Elastic Services Controller Administration Guide.

# VNF Lifecycle Operations

### VNFM Prerequisites

The following prerequisites must be met for VNF lifecycle operations:

- The resource definitions must be created out of band and must be available before VNF instantiation.
- There are a few options with respect to specifying connections to the VIM. The VIM Connector specifies how ESC connects to the VIM and may be created and validated in advance of deploying a VNF (and identified by name), created as part of the request if new vimConnectionInfo is supplied or as part of the Grant response (all have a common source - the NFVO). See VIM Connectors Overview, on page 13.

### NFVO Prerequisites

- The VNF to be instantiated has to be onboarded to the NFVO within an ETSI compliant VNF package.
  - The NFVO must provide ETSI compliant VNF Packages to ESC.
  - The VNF package must contain a VNF Descriptor (VNFD) file.

  The NFVO must support the /vnf_packages API to allow access to the package artifacts.See chapter 10 in the *ETSI GS NFV-SOL 003* specification on the ETSI website for details.
- Update the properties file, *etsi-production.properties* under: `/opt/cisco/esc/esc_database/`. The properties file provides details about the NFVO to ESC.

  The single property *nfvo.apiRoot* allows specification of the NFVO host and port. For example, `nfvo.apiRoot=localhost:8280`.

**Note** For notes on ESC in HA mode, enabled with ETSI service, see the Cisco Elastic Services Controller Install and Upgrade Guide.

### Deployment Request

The deployment request includes the following tasks:

The VNFD provides a description of the following constructs (see *ETSI GS NFV-SOL 001* specification on the ETSI website for details)

- The deployment level configuration such as deployment flavours and external connections
- The VDU configuration, including any applicable images (Compute)
- The internal connection points (VduCp)
- Any volumes to be created, including any applicable images (VirtualBlockStorage)
- The internal virtual links (VnfVirtualLink)
- Policies and groups for placement, scaling and security

The InstantiateVnfRequest:

- The chosen deployment flavour
- The VIM connection details (vimConnectionInfo - Or-Vnfm only)
- Any external networks to which to connect the external connection points (extVirtualLinks)
- Any external networks that may be bound to for internal virtual links (extManagedVirtualLinks)
- A list of key-value pairs to provide deployment specific variables for the deployment (additionalParams)

The Grant from the NFVO (see *ETSI GS NFV-SOL 003* specification on the ETSI website for details):

- Approved and/or updated resources to be added, updated or removed (UUIDs)
- Confirmed placement information

Each lifecycle management request is submitted to the VNFM through the Ve-Vnfm or Or-Vnfm reference points, SOL002 or SOL003 respectively. In order to invoke the correct API, the {apiRoot} is constructed of the following elements:

```
[http_protocol]://[esc_ip]:[esc_port]/[ve_vnfm|or_vnfm]
```

and is followed by the apiName and operations, as per the following sections.

# Creating the VNF Identifier

Creating the VNF Identifier is the first request for any VNF instance. This identifier is used for all further LCM operations executed by the ETSI API. Resources are neither created nor reserved at this stage.

ESC sends a POST request to create VNF instances:

Method Type:

```
POST
```

VNFM Endpoint:

```
 /vnf_instances/
```

HTTP Request Headers:

```
 Content-Type:application/json
```

Request Payload (ETSI data structure: CreateVnfRequest):

```
{
    "vnfInstanceName": "Test-VNf-Instance",
    "vnfdId": "vnfd-88c6a03e-019f-4525-ae63-de58ee89db74"
}
```

Response Headers:

```
HTTP/1.1 201
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Application-Context: application:8250
Accept-Ranges: none
Location: http://localhost:8250/vnflcm/v2/vnf_instances/14924fca-fb10-45da-bcf5-59c581d675d8
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Thu, 04 Jan 2018 12:18:13 GMT
```

Response Body (ETSI Data structure:VnfInstance)

```
{
    "id": "14924fca-fb10-45da-bcf5-59c581d675d8",
    "instantiationState": "NOT_INSTANTIATED",
    "onboardedVnfPkgInfoId": "vnfpkg-bb5601ef-cae8-4141-ba4f-e96b6cad0f74",
    "vnfInstanceName": "Test-VNf-Instance",
    "vnfProductName": "vnfd-1VDU",
    "vnfProvider": "Cisco",
    "vnfSoftwareVersion": "1.1",
    "vnfdId": "vnfd-88c6a03e-019f-4525-ae63-de58ee89db74",
    "vnfdVersion": "1.3",
    "_links": {
        "instantiate": {
            "href":
"http://localhost:8250/vnflcm/v2/vnf_instances/14924fca-fb10-45da-bcf5-59c581d675d8/instantiate"

        },
        "self": {
            "href":
"http://localhost:8250/vnflcm/v2/vnf_instances/14924fca-fb10-45da-bcf5-59c581d675d8"
        }
    }
}
```

For instantiating VNFs, see Instantiating Virtual Network Functions, on page 28.

# Instantiating Virtual Network Functions

The instantiation request triggers several message exchanges, which allow the call flow to deploy a VNF instance. The resources for the VNF are only allocated when the VNF instance is instantiated. The request requires the VNF instance identifier, returned by the Create VNF request to be encoded into the URL to which the request is posted.

The instantiation sub-tasks within the flow include:

1. Retrieving the VNF Descriptor (VNFD) template from the NFVO.
2. Requesting permission from the NFVO (bi-directional Grant flow). For more information see, Requesting Permission via Grant.

Example for *SOL003*:

Method type:

```
POST
```

VNFM Endpoint:

```
/vnf_instances/{vnfInstanceId}/instantiate
```

HTTP Request Header:

```
Content-Type:application/json
```

Request Payload (ETSI data structure: InstantiateVnfRequest)

```
{
"flavourId": "default",
"extVirtualLinks": [
   {
      "id": "extVL-dbf477ad-199a-47ff-939a-cb0101c92585",
      "resourceId": "ext-net",
      "extCps": [
        {
          "cpdId": "ecp_1_vdu_node_1",
          "cpConfig": {
            "cp1": {
              "cpProtocolData": [
                {
                  "layerProtocol": "IP_OVER_ETHERNET",
                  "ipOverEthernet": {
                    "ipAddresses": [
                      {
                        "numDynamicAddresses": "1",
                        "subnetId": "23bb3-742aa-8213eb-dded2"
                        "type": "IPV4"
                      }
                    ]
                  }
                }
              ]
            }
          }
        }
    ],
    "extManagedVirtualLinks": [
        {
            "id": "my-network",
            "resourceId": "93fb90ae-0ec1-4a6e-8700-bf109a0f4fba",
            "virtualLinkDescId": "VLD1"
        }
    ],
    "vimConnectionInfo": {
    "default_openstack_vim": {
            "accessInfo": {
                "password": "*******,
                "username": "admin",
                "vim_project": "tenantName"
            },
            "extra": {
                "name": "esc"
            },
               "interfaceInfo": {
                "baseUrl": "http://localhost:8080"
            },
```

```
            "vimId": "default_openstack_vim",
            "vimType": "OPENSTACK"
        }
    }
    "additionalParams": {
        "CPUS": 2,
        "MEM_SIZE": "512 MB",
        "VIM_FLAVOR": "Automation-Cirros-Flavor",
        "BOOTUP_TIME": "1800"
    }
}
```

The *flavourId* value must be the same as a single `flavour_id` specified in the VNFD.

The previous example also includes an external connection point with a subnet defined. The IP addresses are allocated from that subnet. For information on fixed IP or MAC addresses, see Scaling Virtual Network Functions Using ETSI API, on page 79.

**Note** The Grant response from the NFVO provides the vimConnectionInfo. It is not provided in the *SOL002* payload. This is required in some cases since the SOL002 payloads do not include the vimConnectionInfo information.

You can customize the VNF before instantiation by adding variables to the VNFD template. The values that map to those variables are supplied in the *additionalParams* field of the LCM request. The variables are key-value pairs, where the value can be either a list, string, numeric or boolean.

When the VNFD is retrieved by the VNFM, the *additionalParams* variables are merged into the VNF instance data from the original request received to form instance-specific data.

The list of parameters supplied is driven by the contents of the VNFD; the *additionalParams* specified in the request are used by the VNFD using the get_input TOSCA method within the VNFD. For example, the `cpus`, and `mem_size` variables are merged with the placeholders within the VNFD. For example:

```
tosca_definitions_version: tosca_simple_yaml_1_3

imports:
  - cisco_nfv_sol001_types.yaml
  - etsi_nfv_sol001_vnfd_3_3_1_types.yaml

metadata:
  template_name: Example
  template_author: Cisco Systems
  template_version: '1.0'

topology_template:
  inputs:

    CPUS:
      description: Number of CPUs
      type: string
      default: "2"
    MEM_SIZE:
      description: Memory size
      type: string
      default: "512 MB"
    VIM_FLAVOR:
      description: VIM Flavor
      type: string
      default: "Automation-Cirros-Flavour"
    BOOTUP_TIME:
      description: Time taken to boot the VNF
```

```
      type: string
      default: "1800"

  substitution_mappings:
    node_type: cisco.1VDU.1_0.1_0
    requirements:
      - virtual_link: [ node_1_nic0, virtual_link ]

node_templates:

    vdu1:
      type: tosca.nodes.nfv.Vdu.Compute
      properties:
        name: vdu1
        description: Example
        configurable_properties:
          additional_vnfc_configurable_properties:
            vim_flavor: { get_input: VIM_FLAVOR }
            bootup_time: { get_input: BOOTUP_TIME }
            ...
        vdu_profile:
          min_number_of_instances: 1
          max_number_of_instances: 1
      capabilities:
        virtual_compute:
          properties:
            virtual_cpu:
              num_virtual_cpu: { get_input: CPUS }
            virtual_memory:
              virtual_mem_size: { get_input: MEM_SIZE }

node_1_nic0:
      type: tosca.nodes.nfv.VduCp
      properties:
        order: 0
        layer_protocols: [ ipv4 ]
        protocol:
          - associated_layer_protocol: ipv4
        trunk_mode: false
        virtual_network_interface_requirements:
          - support_mandatory: true
            network_interface_requirements:
              management: "false"
              name_override: { get_input: SRIOV_A_INT_NAME }
              iface_type: "direct"
      requirements:
        - virtual_binding: vdu_1
```

If a modification request with new *additionalParams* variables is submitted for the same VNF instance, then the new variables overwrites the existing values for those keys. The VNFM uses the new variables for deployment.

Although internal links are designed to be ephemeral, in some deployment scenarios they can be bound to external links that outlive the VNF. Consider the following example VNFD fragment:

```
     automation_net:
      type: tosca.nodes.nfv.VnfVirtualLink
      properties:
        connectivity_type:
          layer_protocols: [ ipv4 ]
        description: Internal Network VL
        vl_profile:
          max_bitrate_requirements:
            root: 10000
```

```
          min_bitrate_requirements:
            root: 0
          virtual_link_protocol_data:
            - associated_layer_protocol: ipv4
              l3_protocol_data:
              ip_version: ipv4
              cidr: 1.180.10.0/29
              dhcp_enabled: true
```

To specify an external virtual link to be used in place of automation_net in the VNF deployment, the following data structure is used as part of the instantiation request:

```
...
"extManagedVirtualLinks": [
      {
            "id": "net-5ddc8435-9d85-4560-8b95-bfcd3369c5c2",
            "resourceId": "esc-net2",
            "vimConnectionId":"default_openstack_vim",
            "virtualLinkDescId": "automation_net"
      }
],
...
```

Although the ETSI specifications only support the concept of ephemeral volumes, many vendors require the specification of a persistent volume and so Cisco has implemented an extension to support this. The VIM resource Id of the persistent volume can be supplied as an *additionalParams* key (that matches the get_input in the VNFD) and replace a volume in the VNFD using an optional property, as per the following example:

```
    example-volume:
      type: tosca.nodes.nfv.Vdu.VirtualBlockStorage
      properties:
        virtual_block_storage_data:
          size_of_storage: 200 GB
          vdu_storage_requirements:
            resource_id: { get_input: EX_VOL_UUID }
            vol_id: "0"
            bus: ide
            type: LUKS
```

### Requesting Permission via Grant

The ETSI API requests permission from the NFVO to complete lifecycle management operations for the VNF instance resources and gets resource Ids for any resources pre-provisioned. Following is an example of GrantRequest:

```
{
  "flavourId": "default",
  "instantiationLevelId": "default",
  "isAutomaticInvocation": false,
  "operation": "INSTANTIATE",
  "vnfInstanceId": "e426a94e-7963-430c-96ee-778dde5bd021",
  "vnfLc mOpOccId": "06fe989b-7b0b-40dc-afb3-de26c18651ae",
  "vnfdId": "6940B47B-B0D0-48CB-8920-86BC23F91B16",
  "addResources":
  [
    {
      "id": "res-1abb1609-a1f3-418a- a7a0-2692a5e53311",
      "resourceTemplateId": "vdu1",
      "type": "COMPUTE",
      "vduId": "vdu1"
    },
```

```
    {
      "id": "res-c5ece35c-89e3-4d29-b594-ee9f6591f061",
      "resourceTemplateI d": "node_1_nic0",
      "type": "LINKPORT",
      "vduId": "vdu1"
    },
    {
      "id": "res-e88d8461-5f5a-4dba-af14-def82ce894e5",
      "resourceTemplateId": "automation_net",
      "type": "VL"
    }
  ],
  "_links":
  {
    "vnfInstance":
    {
      "href": "https://172.16
.255.8:8251/vnflcm/v2/vnf_instances/14924fca-fb10-45da-bcf5-59c581d675d8"
    },
    "vnfLcmOpOcc":
    {
      "href":
"https://172.16.255.8:8251/vnflcm/v2/vnf_lcm_op_occs/457736f0-c877-4e07-8055-39dd406c616b"
    }
  }
}
```

The corresponding grant returned may look like the following:

```
{
    "id": "grant-0b7d3420-e6ee-4037-b116-18808dea4e2a",
    "vnfInstanceId": "14924fca-fb10-45da-bcf5-59c581d675d8",
    "vnfLcmOpOccId": "457736f0-c877-4e07-8055-39dd406c616b",
    "addResources": [
        {
            "resourceDefinitionId": "res-1abb1609-a1f3-418a-a7a0-2692a5e53311",
            "vimConnectionId": "esc-005e4412-e056-43a9-8bc0-d6699c968a3c"
        },
        {
            "resourceDefinitionId": "res-c5ece35c-89e3-4d29-b594-ee9f6591f061",
            "vimConnectionId": "esc-005e4412-e056-43a9-8bc0-d6699c968a3c"
        },
        {
            "resourceDefinitionId": "res-e88d8461-5f5a-4dba-af14-def82ce894e5",
            "vimConnectionId": "esc-005e4412-e056-43a9-8bc0-d6699c968a3c"
        }
    ],
    "vimAssets": {
        "computeResourceFlavours": [
            {
                "vimConnectionId": "esc-005e4412-e056-43a9-8bc0-d6699c968a3c",
                "vimFlavourId": "Automation-Cirros-Flavor",
                "vnfdVirtualComputeDescId": "vdu1"
            }
        ],
        "softwareImages": [
            {
                "vimConnectionId": "esc-005e4412-e056-43a9-8bc0-d6699c968a3c",
                "vimSoftwareImageId": "Automation-Cirros-DHCP-2-IF",
                "vnfdSoftwareImageId": "vdu1"
            }
        ]
    },
    "vimConnections": {
    "default_openstack_vim": {
```

```
            "vimId": "default_openstack_vim",
            "vimType": "OPENSTACK",
            "accessInfo": {
                "vim_project": "admin"
            }
        }
    },
    "zones": [
        {
            "id": "zone-c9f79460-7a23-43e4-bb6d-0683e2cdb3d4",
            "vimConnectionId": "default_openstack_vim",
            "zoneId": "default"
        },
        {
            "id": "zone-4039855e-a2cb-48f8-996d-b328cdf9889a",
            "vimConnectionId": "default_openstack_vim",
            "zoneId": "nova"
        }
    ],
    "_links": {
        "self": {
            "href":
"http://localhost:8280/grant/v1/grants/grant-0b7d3420-e6ee-4037-b116-18808dea4e2a"
        },
        "vnfInstance": {
            "href": "https://172.16
.255.8:8251/vnflcm/v1/vnf_instances/14924fca-fb10-45da-bcf5-59c581d675d8"
        },
        "vnfLcmOpOcc": {
            "href":
"https://172.16.255.8:8251/vnflcm/v1/vnf_lcm_op_occs/457736f0-c877-4e07-8055-39dd406c616b"
        }
    }
}
```


The grant request is accepted only if all the requested resources have been granted, else the grant is rejected.

# Querying Virtual Network Functions

Querying VNFs does not affect the state of any VNF instance. This operation simply queries ESC for all the VNF instances it knows about, or a specific VNF instance.

Method Type:

```
GET
```

VNFM Endpoint:

```
/vnf_instances/vnf_instances/{vnfInstanceId}
```

HTTP Request Header:

```
Content-Type: application/json
```

Request Payload:

```
not applicable.
```

Response Headers:

```
< HTTP/1.1 200
HTTP/1.1 200
< X-Content-Type-Options: nosniff
X-Content-Type-Options: nosniff
```

```
< X-XSS-Protection: 1; mode=block
X-XSS-Protection: 1; mode=block
< Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
< Pragma: no-cache
Pragma: no-cache
< Expires: 0
Expires: 0
< X-Frame-Options: DENY
X-Frame-Options: DENY
< Strict-Transport-Security: max-age=31536000 ; includeSubDomains
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
< X-Application-Context: application:8250
X-Application-Context: application:8250
< Accept-Ranges: none
Accept-Ranges: none
< ETag: "2"
ETag: "2"
< Content-Type: application/json;charset=UTF-8
Content-Type: application/json;charset=UTF-8
< Transfer-Encoding: chunked
Transfer-Encoding: chunked
< Date: Thu, 04 Jan 2018 12:25:32 GMT
Date: Thu, 04 Jan 2018 12:25:32 GMT
```

Response Body for a single VNF Instance (ETSI Data structure:VnfInstance)

**Note** The ETag response header is only returned for a single VNF query (that is, one with the VNF Instance ID specified). The ETag value is conditionally used during any subsequent VNF modify operations.

```
{
  "_links": {
    "instantiate": {
      "href":
"http://localhost:8250/vnflcm/v2/vnf_instances/14924fca-fb10-45da-bcf5-59c581d675d8/instantiate"

    },
    "self": {
      "href":
"http://localhost:8250/vnflcm/v2/vnf_instances/14924fca-fb10-45da-bcf5-59c581d675d8"
    }
  },
  "id": "14924fca-fb10-45da-bcf5-59c581d675d8",
  "instantiationState": "NOT_INSTANTIATED",
  "onboardedVnfPkgInfoId": "vnfpkg-bb5601ef-cae8-4141-ba4f-e96b6cad0f74",
  "vnfInstanceName": "Test-VNf-Instance",
  "vnfProductName": "vnfd-1VDU",
  "vnfProvider": "Cisco",
  "vnfSoftwareVersion": "1.1",
  "vnfdId": "vnfd-88c6a03e-019f-4525-ae63-de58ee89db74",
  "vnfdVersion": "2.1"
}
```

The query VNF operation output shows the instantiated state of the VNF. The *InstantiatedVnfInfo* element shows the VIM resource information for all the VNFs.

For example:

```
{
  "instantiatedVnfInfo": {
```

```
"extCpInfo": [
  {
    "cpProtocolInfo": [
      {
        "ipOverEthernet": {
          "ipAddresses": [
            {
              "addresses": [
                "172.16.235.19"
              ],
              "isDynamic": false,
              "type": "IPV4"
            }
          ],
          "macAddress": "fa:16:3e:4b:f8:03"
        },
        "layerProtocol": "IP_OVER_ETHERNET"
      }
    ],
    "cpdId": "anECP",
    "id": "extCp-4143f7d4-f581-45fc-a730-568435dfdb4f"
  }
],
"extManagedVirtualLinkInfo": [
  {
    "id": "net-d39bc4de-285c-4056-8113-24eccf821ebc",
    "networkResource": {
      "resourceId": "my-network",
      "vimConnectionId": "esc-b616e5be-58ce-4cfc-8eee-e18783c5ae5d"
    },
    "vnfLinkPorts": [
      {
        "cpInstanceId": "vnfcCp-9b24c9e0-1b28-4aba-a9df-9bfc786bfaed",
        "cpInstanceType": "EXT_CP"
        "id": "vnfLP-9b24c9e0-1b28-4aba-a9df-9bfc786bfaed",
        "resourceHandle": {
          "resourceId": "926b7748-61d9-4295-b9ff-77fceb05589a",
          "vimConnectionId": "esc-b616e5be-58ce-4cfc-8eee-e18783c5ae5d"
        }
      }
    ],
    "vnfVirtualLinkDescId": "my-network"
  }
],
"extVirtualLinkInfo": {
   "id": "extLP-4143f7d4-f581-45fc-a730-568435dfdb4f",
   "resourceHandle": {
        "resourceId": "d6a4c231-e77c-4d1f-a6e2-d3f463c4ff72"
   },
   "extLinkPorts": {
        "id": "extLP-4143f7d4-f581-45fc-a730-568435dfdb4f",
        "resourceHandle": {
             "resourceId": "d6a4c231-e77c-4d1f-a6e2-d3f463c4ff72 "
        }
   },
   "currentVnfExtCpData": [
        {
             "cpdId": "extCp-4143f7d4-f581-45fc-a730-568435dfdb4f",
             "cpConfig": {
                  "vm1_nic0": {
                  "linkPortId": "extLP-4143f7d4-f581-45fc-a730-568435dfdb4f"
                  }
             }
        }
```

```
    ]
}

      "id": "extVL-b9bd55a9-4bd9-4ad8-bf67-ba1e7b82aca6",
        "resourceHandle": {
        "resourceId": "anECP",
        "vimConnectionId": "esc-b616e5be-58ce-4cfc-8eee-e18783c5ae5d"
      }
    }
  ],
  "flavourId": "bronze",
  "scaleStatus": [
    {
      "aspectId": "default_scaling_aspect",
      "scaleLevel": 1
    }
  ],
  "vnfState": "STARTED",
  "vnfcResourceInfo": [
    {
      "computeResource": {
        "resourceId": "a21f0b15-ec4b-4968-adce-1ccfad118caa",
        "vimConnectionId": "default_openstack_vim"
      },
      "id": "res-89a669bb-fef4-4099-b9fe-c8d2e465541b",
      "vduId": "vdu_node_1",
      "vnfcCpInfo": [
        {
          "cpProtocolInfo": [
            {
              "ipOverEthernet": {
                "ipAddresses": [
                  {
                    "addresses": [
                      "172.16.235.19"
                    ],
                    "isDynamic": false,
                    "type": "IPV4"
                  }
                ],
                "macAddress": "fa:16:3e:4b:f8:03"
              },
              "layerProtocol": "IP_OVER_ETHERNET"
            }
          ],
          "cpdId": "node_1_nic0",
          "id": "vnfcCp-c09d5cf2-8727-400e-8845-c4d5cb479db8",
          "vnfExtCpId": "extCp-4143f7d4-f581-45fc-a730-568435dfdb4f"
        },
        {
          "cpProtocolInfo": [
            {
              "ipOverEthernet": {
                "ipAddresses": [
                  {
                    "addresses": [
                      "172.16.235.16"
                    ],
                    "isDynamic": false,
                    "type": "IPV4"
                  }
                ],
                "macAddress": "fa:16:3e:94:b3:91"
              },
```

```
                "layerProtocol": "IP_OVER_ETHERNET"
              }
            ],
            "cpdId": "node_1_nic1",
            "id": "vnfcCp-9b24c9e0-1b28-4aba-a9df-9bfc786bfaed"
          }
        ]
      }
    ]
  }
}
```


### Selecting Attributes for VNF Query

You can select the attributes to appear in the VNF Query response using the attribute selector. You can mark the attributes for including or excluding from a query. You can exclude some of the attributes that are not required, for example attributes with a lower bound of zero on their cardinality (e.g. 0..1, 0..N) and that are not mandatory (subject to certain conditions).

By selecting only the necessary attributes in the query reduces the amount of data exchanged over the interface and processed by the API consumer application.

The table lists the URI query parameters for selecting attributes for the GET Request.

***Table 4: Selecting Attributes for GET Request***

| Parameter | Definition |
|---|---|
| all_fields | Requests all complex attributes included in the response, including those suppressed by exclude_default. It is opposite to the *exclude_default* parameter. The API producer supports the all_fields parameter for certain resources.<br>**Note** The complex attributes are structured attributes or arrays. |
| fields | Requests to include only the listed complex attributes in the response.<br>The parameter is formatted as a list of attribute names. An attribute name can either be the name of an attribute, or a path consisting of the names of multiple attributes with parent-child relationship, separated by "/". The attribute names in the list can be separated by comma (","). The valid attribute names for a particular GET request are the names of all complex attributes in the expected response that have a lower cardinality bound of 0 and that are not conditionally mandatory.<br>The API producer supports the fields parameter for certain resources. The details are defined in the clauses specifying the actual resources.<br>The "/" and "~" characters in attribute names in an attribute selector will be escaped according to the IETF standards.<br>The "," character in attribute names in an attribute selector will be escaped by replacing it with "~a".<br>Further, percent-encoding applies to the characters that are not allowed in a URI query part according to the IETF standards. |

| Parameter | Definition |
|---|---|
| exclude_fields | Requests to exclude the listed complex attributes from the response. For the format, eligible attributes and support by the API producer, the provisions defined for the "fields" parameter will apply. |
| exclude_default | Requests to exclude a default set of complex attributes from the response. Not every resource has a default set. Only complex attributes with a lower cardinality bound of zero that are not conditionally mandatory can be included in the set.<br>The API producer supports this parameter for certain resources.<br>The exclude_default parameter is a flag and has no value.<br>If a resource supports attribute selector, and none of the attribute selector parameters is specified in a GET request, then the **exclude_default** parameter becomes the default. To emulate the original behaviour of GET Request, you can either supply the **all_fields** flag or set the ETSI property **attribute.selector.default.all_fields** to true which changes the behaviour, when no attribute selectors are provided, to all_fields. |

The GET Response validates the parameter combinations in the GET Request.The table defines the valid parameter combinations.

***Table 5: Parameter combinations for Get Response***

| Parameter Combination | GET Response |
|---|---|
| (none) | Includes same as exclude_default. |
| all_fields | Includes all the attributes. |
| fields=<list> | Includes all the attributes except all complex attributes with minimum cardinality of zero that are not conditionally mandatory, **and** that are not provided in <list>. |
| exclude_fields=<list> | Includes all attributes **except** those complex attributes with a minimum cardinality of zero that are not conditionally mandatory, **and** that are provided in <list>. |
| exclude_default | Includes all attributes **except** those complex attributes with a minimum cardinality of zero that are not conditionally mandatory, **and** that are part of the *default exclude set* defined in the present document for the particular resource. |
| exclude_default and fields=<list> | Includes all attributes **except** those complex attributes with a minimum cardinality of zero that are not conditionally mandatory **and** that are part of the *default exclude set* defined in the present document for the particular resource, **but** that are not part of <list>. |

The GET Request for resources such as VNF Instances, VNF LCM Operation Occurrences, and PM Jobs supports the selection of attributes.

***Table 6: Resources supporting the selection of attributes***

| Name | Cardinality | Description |
|---|---|---|
| VNF Instances | | |
| exclude_default | 0..1 | Indicates to exclude the following complex attributes from the response.<br>The following attributes are excluded from the VnfInstance structure in the response body if this parameter is provided, or none of the parameters ( all_fields, fields, exclude_fields, exclude_default) are provided:<br>• vnfConfigurableProperties<br>• vimConnectionInfo<br>• instantiatedVnfInfo<br>• metadata<br>• extension |
| VNF LCM operation occurrences | | |
| exclude_default | 0..1 | The following attributes are excluded from the VnfLcmOpOcc structure in the response body if this parameter is provided, or none of the parameters (all_fields, fields, exclude_fields, exclude_default) are provided:<br>• operationParams<br>• error<br>• resourceChanges<br>• changedInfo<br>• changedExtConnectivity |
| PM Jobs | | |

| Name | Cardinality | Description |
|---|---|---|
| exclude_default | 0..1 | The following attributes are excluded from the PmJob structure in the response body if this parameter is provided, or none of the parameters (all_fields, fields, exclude_fields, exclude_default) are provided:<br>• Reports |

For information on VNF lifecycle operations, see VNF Lifecycle Operations, on page 26.

# Modifying Virtual Network Functions

You can modify or update the properties of a VNF instance, which is in the NOT_INSTANTIATED state, using the modify VNF lifecycle operation. ESC receives a PATCH request from NFVO to modify a single VNF instance.

A JSON merge algorithm is applied from the input payload against the stored data to modify the VNF instance.

**Note** Modifying VNF operation updates only the properties, but not the functionality of the VNF. The modify operation is only valid on a VNF instance resource that is NOT_INSTANTIATED.

The following properties of an existing VNF instance can be modified:

- vnfInstanceName
- vnfInstanceDescription
- onboardedVnfPkgInfoId (null value is not allowed)
- vnfConfigurableProperties
- metadata
- extensions
- vimConnectionInfo

Method Type

```
PATCH
```

VNFM Endpoint

```
/vnf_instances/{vnfInstanceId}
```

HTTP Request Header

```
Content-Type: application/merge-patch+json
If-Match: ETag value
```

**Note** The ETag, if specified, is validated against the ETag value stored against the VNF instance resource. If the values do not match, the modify request will be rejected.

Request Payload (ETSI data structure: VnfInfoModifications)

```
{
    "vnfInstanceName": "My NEW VNF Instance Name",
    "vnfInstanceDescription": "My NEW VNF Instance Description",
        "vnfConfigurableProperties": {
        "isAutoscaleEnabled": "true"
    },
    "metadata": {
        "serialRange": "ab123-cc331",
        "manufacturer": "Cisco"
    },
    "extensions": {
        "testAccess": "false",
        "ipv6Interface": "false"
    },
    "vimConnectionInfo": {
      "default_openstack_vim": {
            "vimType": "openstack",
            "interfaceInfo": {
                "uri": "http://172.16.14.27:35357/v3"
            },
            "accessInfo": {
                "domainName": "default",
                "projectName": "admin",
                "userName": "default"
            }
        }
    }
}
```

**Note** The Grant response from the NFVO provides the vimConnectionInfo instead of the *SOL002* payload. The *SOL002* request contains some attributes that affect the VNF resource at a finer VNFC-level such as vnfcInfoModifications. See *SOL002* on the *ETSI website* for more details.

Response Header:

```
not applicable.
```

Response Body:

```
not applicable.
```

When the PATCH operation is complete, the VNF instance is modified, and the details are sent to the NFVO through the notification.

# Operating Virtual Network Functions

You can start or stop a VNF instance using the operate lifecycle management operation. The VNF instance can be stopped gracefully or forcefully.

**Note** The OpenStack API supports only forceful stop.

The *changeStateTo* field must have the value STARTED or STOPPED in the request payload, to start or stop a VNF instance.

Permission is also required from the NFVO (bi-directional Grant flow) for this operation. See Requesting Grant Permission for more informaiton.

Method Type:

```
POST
```

VNFM Endpoint:

```
/vnf_instances/{vnfInstanceId}/operate
```

HTTP Request Headers:

```
Content-Type:application/json
```

Response Headers:

```
HTTP/1.1 202
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: TEST
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Application-Context: application:8250
Accept-Ranges: none
Location: http://localhost:8250/vnflcm/v2/vnf_lcm_op_occs/e775aad5-8683-4450-b260-43656b6b13e9
Content-Length: 0
Date: Thu, 04 Jan 2018 12:40:27 GMT
```

Response Body:

```
not applicable.
```

# Deleting Virtual Network Function Resource Identifier

Deleting VNF operation releases the VIM resources reserved for the VNF instance as well as deletes the VNF instance identifier. Upon deletion, the VNF instance identifier is no longer available. So, no further lifecycle management operations are possible using this identifier.

Method Type:

```
DELETE
```

VNFM Endpoint:

```
/vnf_instances/{vnfInstanceId}
```

HTTP Request Headers:

```
Content-Type:application/json
```

Request Payload:

```
not applicable.
```

Response Headers:

```
HTTP/1.1 204
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: TEST
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Application-Context: application:8250
Accept-Ranges: none
Date: Thu, 04 Jan 2018 12:48:59 GMT
```

Response Body:

```
not applicable.
```

# Changing the VNF Package

Changing the VNF package operation allows the package which is immutible upon which an instance is modelled to change into a new package.There are cases for this operation such as a software upgrade or fixing defects in the original package. Validate the change as per the policies defined to describe the allowable upgrade or downgrade.

**Note** ESC 5.8 supports only a change in the software image which is either supplied in the request or in the grant as part of the execution of this API.

Method Type:

```
POST
```

VNFM Endpoint:

```
/vnf_instances/{vnfInstanceId}/change_vnfpkg
```

HTTP Request Headers:

```
Content-Type:application/json
```

Request Payload:

```
{
    "vnfdId": "CE2F2413-5723-4661-8EC0-6A8FD7562892",
    "extVirtualLinks": [{}],
    "extManagedVirtualLinks": "[{}]",
    "vimConnectionInfo": {[
        {
            "id": "vci1",
            "vimType": "OPENSTACK_V3",
            "interfaceInfo": {
                "uri": "http://10.51.14.27:35357/v3"
            },
            "accessInfo": {
                "domainName": "default",
                "projectName": "admin",
                "userName": "default"
            }
        }
```

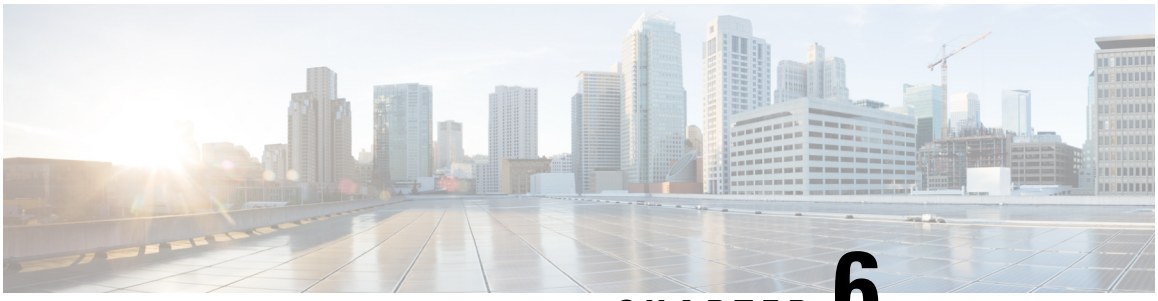
```
    ]},
    "vnfConfigurableProperties": {},
    "additionalParams": {
      "SOFTWARE_IMAGE": "NEW_IMAGE_NAME"
    },
    "extensions": {}
}
```

Response Headers:

```
HTTP/1.1 202
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: TEST
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Application-Context: application:8250
Accept-Ranges: none
Date: Thu, 04 Jan 2018 12:48:59 GMT
```

Response Body:

```
Not applicable.
```



CHAPTER 6

# Monitoring Virtual Network Functions



# Monitoring Virtual Network Functions Using ETSI API

During the deployment of a VNF, metrics must be defined to instruct the ESC monitoring agent component (MONA) how to determine if the VNF is healthy. The definition of metrics is within the Key Performance Indicator (KPI) section of the VNFD and allow MONA to periodically monitor the VNF to check its health and workload, defined on a per-VNFC basis. Actions are then associated with these KPIs and executed when the appropriate conditions are met.

There are several built-in monitoring methods such as ICMP Ping and SNMP. Some of the metrics to monitor on the constituent VNFCs include:

- reachability
- resource usage (such as CPU, memory, disk and network throughput)

The following pre-requisites must be met for the deployed VNFCs to be monitored:

- The deployed VNFCs must be alive
- Monitoring is enabled
- KPIs must be configured

Example:

```
vdu1:
      type: tosca.nodes.nfv.Vdu.Compute
      properties:
        name: Example VDU1
        description: Example VDU
        boot_order:
          - boot1-volume
        configurable_properties:
          additional_vnfc_configurable_properties:
            vim_flavor: Automation-Cirros-Flavor
            bootup_time: 1800
            vm_name_override: my-vdu-1
            recovery_action: REBOOT_THEN_REDEPLOY
            recovery_wait_time: 100
```

```
      monitor_on_error: false
      max_retries: 2
      kpi_data:
        VM_ALIVE-1:
          event_name: 'VM_ALIVE-1'
            metric_value: 1
            metric_cond: 'GT'
            metric_type: 'UINT32'
            metric_occurrences_true: 1
            metric_occurrences_false: 30
            metric_collector:
              type: 'ICMPPing'
              nicid: 1
              poll_frequency: 10
              polling_unit: 'seconds'
              continuous_alarm: false
      admin_rules:
        VM_ALIVE-1:
          event_name: 'VM_ALIVE-1'
          action:
            - 'ALWAYS log'
            - 'FALSE recover autohealing'
            - 'TRUE esc_vm_alive_notification'
      placement_type: zone
      placement_target: nova
      placement_enforcement: strict
      vendor_section:
        cisco_esc:
          config_data:
            example.txt:
              file: ../Files/Scripts/example.txt
                variables:
                  DOMAIN_NAME: { get_input: DOMAIN_NAME }
                  NAME_SERVER: { get_input: NAME_SERVER }
                  VIP_ADDR: { get_input: VIP_ADDR }
                  VIP_PREFIX: { get_input: VIP_PREFIX }
  vdu_profile:
    min_number_of_instances: 1
    max_number_of_instances: 1
         capabilities:
  virtual_compute:
    properties:
      virtual_cpu:
        num_virtual_cpu: 8
      virtual_memory:
        virtual_mem_size: 16
requirements:
  - virtual_storage:  cdr1-volume
  - virtual_storage:  boot1-volume
```

The kpi_data shown above is the default KPI required that is required in all deployments at a minimum so that the VM_ALIVE message is generated to tell ESC Manager that the VNFC has been deployed successfully; it consists of the KPI, how it is collected and the actions to be executed when the KPI is met.

Cisco data structure properties

| Data Type | Property Name | Description | Values |
|---|---|---|---|
| cisco.datatypes.nfv.data.Kpi | KPI label | Unique user-defined KPI name | Any |

| Data Type | Property Name | Description | Values |
|---|---|---|---|
| cisco.datatypes.nfv.data.Kpi | monitoring_agent | Specifies the monitoring agent for a VNF, for example, local or distributed MONA | URI for the agent |
| cisco.datatypes.nfv.data.Kpi | event_name | | |
| cisco.datatypes.nfv.data.Kpi | metric_value | | |
| cisco.datatypes.nfv.data.Kpi | metric_cond | | |
| cisco.datatypes.nfv.data.Kpi | metric_type | | |
| cisco.datatypes.nfv.data.Kpi | metric_occurrences_true | | |
| cisco.datatypes.nfv.data.Kpi | metric_occurrences_false | | |
| cisco.datatypes.nfv.metric.Collector | type | See the NETCONF API Guide | See the NETCONF API Guide |
| cisco.datatypes.nfv.metric.Collector | nicid | | |
| cisco.datatypes.nfv.metric.Collector | poll_frequency | | |
| cisco.datatypes.nfv.metric.Collector | polling_unit | | |
| cisco.datatypes.nfv.metric.Collector | continuous_alarm | | |
| cisco.datatypes.nfv.metric.Collector | property_list | | |
| cisco.datatypes.nfv.data.Admin_rules | Rule label | Unique user-defined name | Any |
| cisco.datatypes.nfv.data.Admin_rules | event_name | This value must match a Kpi event_name | |
| cisco.datatypes.nfv.data.Admin_rules | action | | |
| cisco.datatypes.nfv.data.Admin_rules | property_list | | |

The following extract is from the ETSI properties file, which allows the subscription to an extension notification type:

```
# For notificationType "InfrastructureOperationOccurrenceNotification"
subscription.notifications.infra.filter.operationTypes=MONITORING_MIGRATION
subscription.notifications.infra.filter.operationStates=COMPLETED,FAILED_TEMP,FAILED,ROLLED_BACK
subscription.notifications.infra.callbackUri=http://<nfvoHost>:<nfvoPort>/monitoring/migration/notification
 # Full URL where the notification will be sent
subscription.notifications.infra.authentication.authType=BASIC # or OAUTH2_CLIENT_CREDENTIALS

# Basic Auth credentials (based on authType)
subscription.notifications.infra.authentication.paramsBasic.userName=nfvo
subscription.notifications.infra.authentication.paramsBasic.password=mypw
```

```
# Alternatively, OAUTH 2.0 credentials (based on authType)
#subscription.notifications.infra.authentication.paramsOauth2ClientCredentials.clientId=
#subscription.notifications.infra.authentication.paramsOauth2ClientCredentials.clientPassword=
#subscription.notifications.infra.authentication.paramsOauth2ClientCredentials.tokenEndpoint=
```

**Note** If the previous properties are not set, then these notifications are sent to the subscribers where all notifications types are applicable.

For more information on KPIs and Rules, see the *Cisco Elastic Services Controller User Guide*.

# VM Monitoring Operations

You can set and unset monitoring of VMs using RESTful interface.

The operation is defaults to asynchronous, you must set `sync.supported=true` to use this functionaity in a synchronous way.

A payload is required to monitor VMs:

Method type

`POST`

VNFM Endpoint

Example for SOL003:

`{apiRoot}/or_vnfm/vnflcm/v2/ext/vnf_instances/{vnfInstanceId}/monitoring/operations`

Example for SOL002:

`{apiRoot}/ve_vnfm/vnflcm/v2/ext/vnf_instances/{vnfInstanceId}/monitoring/operations`

To start and stop monitoring operation on a specified VM, set the `vnfcInstanceIds`

with payload:

```
{
    "vnfcInstanceIds": ["vnfcInstanceId1","vnfcInstanceId2",...,"vnfcInstaceIdN"],    ##
optional
    "operation": "ENABLE_MONITOR",                                                    ##
mandatory ENABLE_MONITOR, DISABLE_MONITOR, REBOOT
    "additionalParams": []                                                            ##
optional - for future use :-)
}
```

To start and stop monitoring operation on the entire VNF, do not set the `vnfcInstanceIds`.

You must mention enable_monitoring to set VM monitoring, and disable_monitoring to unset VM monitoring in the operation field.

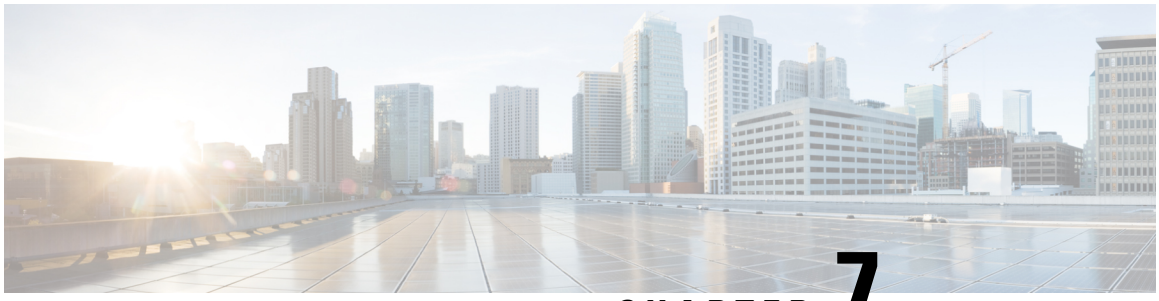
**Note** When a user reboots the VM from the ESC ETSI interface, the monitoring is automatically enabled.

# Notification for VM Monitoring Status

ETSI NFV MANO provides status notifications for VM Monitoring. You can enable, disable, and reboot the VMs on a particular VNF or on a particular VM of a VNF using payload.

ETSI NFV MANO sends the following `[notifications-per-operation]` when setting, unsetting, or rebooting the VMs:

```
[notifications-per-operation]
-------------------------------------
VM_MONITOR_SET notification when enabling a monitor
VM_MONITOR_UNSET notification when disabling a monitor
VM_REBOOTED notification when rebooting
```



CHAPTER 7

# Monitoring VNFs Using D-MONA



## Onboarding D-MONA

Cisco Elastic Services Controller supports Distributed Monitoring and Actions (D-MONA) for effective monitoring of the VNFs. D-MONA is a standalone monitoring application. For more information, see Monitoring VNFs Using D-MONA in the Cisco Elastic Services Controller User Guide.

To onboard D-MONA, you must fulfill the prerequisites and prepare the deployment data model:

**Prerequisites**

- Ensure connectivity between ESC and D-MONA.
- Ensure connectivity between D-MONA and the deployed VNFs.

**Note** Monitoring of D-MONA by another D-MONA is not supported.

For information on deploying D-MONA, see Deploying D-MONA, on page 53.

## Deploying D-MONA

From ESC 5.3 or later, 1:1 mapping is not required. It supports explicit D-MONA deployment.

- In this scenarios, multiple D-MONA Instances can be deployed.
- VNFs can be deployed under, or migrated to specified monitoring agent.

For more information on deploying the VNFs with explicit D-MONA mapping, see the Deploying VNFs with Explicit D-MONA Mapping chapter in the Cisco Elastic Services Controller User Guide.

For using D-MONA in your infrastructure, you must:

1. Deploy the D-MONA with the monitoring infrastructure.
2. Deploy the VNFs using the D-MONA for monitoring.

After deployment, D-MONA is monitored by the local MONA running on the ESC VM.

The following example shows the D-MONA VNFD:

```
tosca_definitions_version: tosca_simple_yaml_1_3
description: D-MONA VNFD (SOL001 v0.10.0)

imports:
  - cisco_nfv_sol001_types.yaml
  - etsi_nfv_sol001_vnfd_0_10_0_types.yaml

metadata:
  template_name: D-MONA
  template_author: Cisco Systems
  template_version: '1.0'

dsl_definitions:
  descriptor_id: &descriptor_id f5b37b47-d9bd-4605-afb0-30c0d659a3c2
  provider: &provider cisco
  product_name: &product_name D-MONA
  software_version: &software_version '1.0'
  descriptor_version: &descriptor_version '1.0'
  flavour_id: &flavour_id default
  flavour_description: &flavour_description 'Default VNF Deployment Flavour'
  vnfm: &vnfm '9:Cisco Elastic Services Controller:v04.04.01'
```

For information on deploying VNFs using D-MONA, see Using D-MONA for a Deployed VNF, on page 56.

***Table 7: Input Parameters for D-MONA Deployment***

| Parameter | Description |
|---|---|
| SW_IMAGE_NAME | The name of ESC image |
| DMONA_CERT | The HTTPS certificate |
| ADMIN_PASSWORD | The admin user password |
| SECURITY_BASIC_ENABLED | A flag that indicates whether basic security is enabled or not |
| SECURITY_USER_NAME | A security user to communicate with ESCManager |
| SECURITY_USER_PASSWORD | A security user's password used to communicate with ESCManager |

KPI data:

- property_list
  - name—protocol

- value—https
- name—port
- value—8443
- name—path
- value—mona/v1/health/status
- name—application_startup_timevalue—true

Config data parameters:

- user-data.txt

  admin_password—value defined for ADMIN_PASSWORD in input parameter
- application—dmona.template
  - monitoring.agent—true
  - security_basic_enabled—value defined for SECURITY_BASIC_ENABLED in input parameter
  - security_user_name—value defined for SECURITY_USER_NAME in input parameter
  - security_user_password—value defined for SECURITY_USER_PASSWORD in input parameter
  - monitoring.agent.vim.mapping—false

Example payload:

```
config_data:
    '--user-data':
        file: ../Files/Scripts/user-data.txt
        variables:
             admin_password: { get_input: ADMIN_PASSWORD }
    '/opt/cisco/esc/mona/dmona.crt':
        data: { get_input: DMONA_CERT }
    '/opt/cisco/esc/mona/config/application-dmona.properties':
        file: ../Files/Scripts/application-dmona.template
        variables:
             monitoring.agent: true
             security_basic_enabled: { get_input: SECURITY_BASIC_ENABLED }
             security_user_name: { get_input: SECURITY_USER_NAME }
            security_user_password: { get_input: SECURITY_USER_PASSWORD }
            monitoring.agent.vim.mapping: false
```

The following table lists the D-MONA VM flavors for large scale deployments:

| Deployment | Number of VMs | Virtual CPU per VM | Virtual Memory (GB) per VM | Virtual Hard Disk (GB)per VM | Number of total VMs Supported |
|---|---|---|---|---|---|
| D-MONA | 1 | 4 | 8 | 40 | 1500 |

# Configuring D-MONA

While configuring D-MONA, you can view two types of runtime behavior; one from a typical ESC deployment, and the other one with capabilities provided by D-MONA.

**D-MONA Day Zero Configuration**

The D-MONA runtime behavior is controlled by the day 0 configuration provided to the VM at the time of deployment.

The following example shows D-MONA SSH access and D-MONA ESC certificate configuration:

```
            config_data:
              '--user-data':
                file: file:///opt/cisco/esc/esc-config/dmona/user-data.template
                variables:
                  # This is the SHA-512 hashed password for 'C1sco@123'
                  vm_credentials:
$6$rounds=4096$6YN5.SHEdfa6v$t6tkvtIrEZv9xpFlLIKkkU2CBq6G2rtObztMqui4Y7uRUBDU62T0NIeDpMn4/TPMsbiBL8CHjdjZaj/5HlwIo/

              '/opt/cisco/esc/mona/dmona.crt':
                data: { get_input: DMONA_CERT }
              '/opt/cisco/esc/mona/config/application-dmona.properties':
                file: file:///opt/cisco/esc/esc-config/dmona/application-dmona.template
```

The `vm_credentials` passes the encrypted password to admin for SSH access to D-MONA.

For monitoring using D-MONA, see Monitoring Using D-MONA, on page 58.

# Using D-MONA for a Deployed VNF

For deploying the VNFs using D-MONA for monitoring, you must have the D-MONA with the `monitoring.agent.vim.mapping day-0` variable set to false. When ESC detects D-MONA, monitoring of the VNF is assigned to that D-MONA, otherwise the local MONA handles the monitoring.

# Specifying D-MONA Monitoring Agent through ETSI ESC Interface

Use the following to specify the monitoring agent:

1. Only specify the monitoring agent (via additionalParams), or you can have the monitoring agent under KPI section of VNFD.

   Here, the specified agent is used to populate the deployment model processed by ESC.

   NFVO or EM sends the POST request.

   Method Type:

   POST

   VNFM Endpoint:

   `vnflcm/v2/vnf_instances/$vnf_instance_id/instantiate`

Example:

```
InstantiateVnfRequest with only the monitoring agent specified (additionalParams)

############################
# Instantiate VNF Request #
############################

#POST https://localhost:8251/vnflcm/v2/vnf_instances/$vnf_instance_id/instantiate
{
    "flavorId": "default",
    "instantiationLevelId": "default",
    "vimConnectionInfo": {
        "default_openstack_vim": {
            "accessInfo": {
                "password": "VIM-password",
                "project": "Project_001",
                "projectDomain": "default",
                "region": "regionOne",
                "userDomain": "VIM-user-uuid",
                "username": "VIM-user"
            },
               "interfaceInfo": {
                "endpoint": "http://openstack_vim:5000/v3/auth"
            },
            "vimId": "VIM-001",
            "vimType": "OPENSTACK_V3"
        }
    },
    "extVirtualLinks": [
        {
            "id": "Network0",
            "extCps": [
                {
                    "cpConfig": {
                      "cp1": {
                            "cpProtocolData": [
                                {
                                    "ipOverEthernet": {
                                        "ipAddresses": [
                                            {
                                                "subnetId":
"654c5793-c74b-4e78-8bd5-2162ec3f9f3e",
                                                "type": "IPV4"
                                            }
                                        ]
                                    },
                                    "layerProtocol": "IP_OVER_ETHERNET"
                                }
                            ]
                        }
                    ],
                    "cpdId": "VDU_1:port_1"
                }
            ],
            "resourceId": "3ecaeb96-f2f5-4eed-b51f-8a69e80748f3",
            "resourceProviderId": "3ecaeb96-f2f5-4eed-b51f-8a69e80748f3",
            "vimConnectionId": "string"
        }
    ],
    "additionalParams": {
        "CF1_SLOT_CARD_NUMBER": "1",
        "CF2_SLOT_CARD_NUMBER": "2",
        "CF_CARD_TYPE_NUM": "0x40010100",
```

```
            "CF_DOMAIN_NAME": "cisco.com",
            "CF_NAME_SERVER": "171.70.168.183",
            "CF_STAROS_CONFIG_URL": "../Files/Scripts/control-function/staros_config.txt",
            "CF_STAROS_PARAM_URL": "../Files/Scripts/control-function/staros_param_cf.cfg",

            "CF_VIP_ADDR": "172.77.11.6",
            "CHASSIS_KEY": "164c03a0-eebb-44a8-87fa-20c791c0aa6d",
            "SF1_SLOT_CARD_NUMBER": "3",
            "SF2_SLOT_CARD_NUMBER": "4",
            "SF_CARD_TYPE_NUM": "0x42030100",
            "SF_STAROS_PARAM_URL": "../Files/Scripts/session-function/staros_param_sf.cfg",

            "VIM_NETWORK_DI_INTERNAL1": "etsi-vpc-di-internal1",
            "VIM_NETWORK_DI_INTERNAL2": "etsi-vpc-di-internal2",
            "VIM_NETWORK_MANAGEMENT": "DualStack-KPI-M-Test-Net",
            "VIM_NETWORK_ORCHESTRATION": "esc-net",
            "VIM_NETWORK_SERVICE1": "etsi-vpc-service1",
            "VIM_NETWORK_SERVICE2": "etsi-vpc-service2",
            "VNFM_PROXY_ADDRS": "172.77.12.106,172.77.12.104,172.77.12.105",
            "VNFM_MONITORING_AGENT": "dmonaName://dm1-agent"
        }
    }
```

The single agent specified in the API request is mapped to the variable wherever it is used in the VNFD and is converted to the appropriate data model.

# Monitoring Using D-MONA

To monitor the VNFs using D-MONA, you must deploy the ETSI VNFD D-MONA and then deploy the ETSI VNFD monitored by D-MONA.

The D-MONA parameters are defined within the VNFD, or provided as additionalparams in the instantiate D-MONA VNF payload.

An ETSI compliant VNFD is used for the deployment of D-MONA.

The input parameters, KPI data, and config paramters are required for instantiation of D-MONA deployment.

The input parameters are either defined within the VNFD or provided as additionalParams section of instantiate D-MONA VNF payload.

```
"VNFM_MONITORING_AGENT": "dmonaName://<dmona_instance__name>"
```

# Resetting the Monitoring Rules for D-MONA

ESC can now detect rebooting of the D-MONA application by monitoring the startup time.

The local MONA monitors D-MONA by performing a HTTP(S) call to the D-MONA health API and keeps track of the last known startup time of the polled D-MONA process. Upon successful request (status code = 200), local MONA compares the last known startup time with the returned startup time from the polled application.

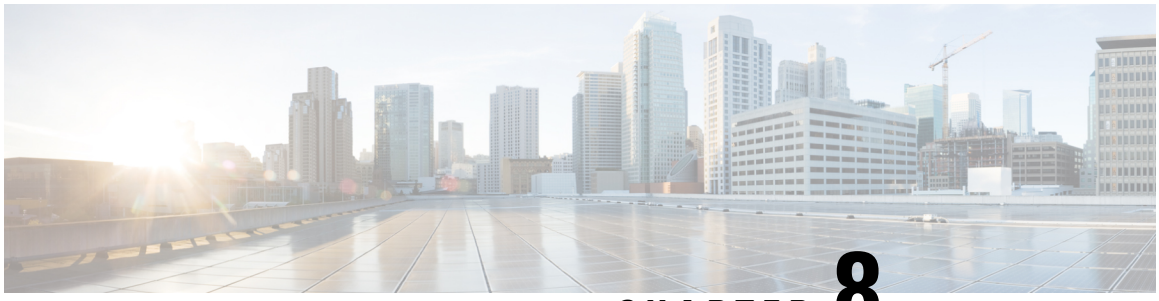
To enable the startup time check, you must set `application_startup_time` to true in KPI section of VNFD yaml.

However, if the `application_startup_time` is not present or set to false, then DMONA reboot check is disabled. You must set this property for D-MONA reboot.

**Note** The application startup time is not backward compatible. It is available from ESC release 5.3 onwards.

Following is a sample KPI section of D-MONA VNFD:

```
 VM_ALIVE-1:
                event_name: 'VM_ALIVE'
                metric_value: 1
                metric_cond: 'GT'
                metric_type: 'UINT32'
                metric_occurrences_true: 1
                metric_occurrences_false: 30
                metric_collector:
                  type: 'HTTPGET'
                  nicid: 0
                  address_id: 0
                  poll_frequency: 3
                  polling_unit: 'seconds'
                  continuous_alarm: false
                  property_list:
                    - name: protocol
                      value: https
                    - name: port
                      value: 8443
                    - name: path
                      value: mona/v1/health/status
                    - name: application_startup_time # Set to true to enable start time
check
                      value: true
```



CHAPTER 8

# Migrating the Monitoring Agent



## Migrating the Monitoring Agent

Each ESC instance has an agent to monitor it to enable ESC to control recovery and scaling operations. Following are the various scenarios that need migration of the monitoring agent:

1. Migrating from **local** to **distributed**

   For example:

   When introducing a new D-MONA into a data center.

2. Migrating from **distributed** to **local**

   For example:

   When performing a software upgrade.

3. Migrating from **distributed** to **distributed**

   For example:

   When performing load balancing.

4. Migrating many instances in quick succession from **distributed** to **distributed**

   For example:

   Disaster recovery

This section covers API that will enable migrating the monitoring agent without impacting the primary function of the VNF instance and also minimizing the impact on virtualisation (recovery/scaling).

The following three steps are performed by this API to process the monitoring update:

- Disable monitoring
- Service model update

- Re-enable monitoring

# Executing the Monitoring Migration API

Method Type:

GET

VNFM Endpoint:

```
{http_scheme}://{api_root}/vnflcm/v2/ext/vnf_instances/{vnfInstanceId}/monitoring/migrate
```

HTTP Request Header:

```
Content-Type: application/json
```

Following are the examples for JSON payload:

Sample VnfMonitoring payload for migrating monitoring to a D-MONA instance (dmona1):

```
{
  "monitoring_agent": "dmona://dmona1",
  "key": "MONITORING_AGENT"
}
```

Sample for VnfMonitoring payload migrating monitoring to local MONA

```
{
  "monitoringAgent": "dmonaName://local_mona",
  "key": "MONITORING_AGENT"
}
```

**Note** A new string value is introduced to represent the central MONA component within ESC. It is used for the migration to local MONA by the previous API.

The following are the supported attribute names and data types for the migration request:

***Table 8:***

| Attribute Name | Data Type | Description |
|---|---|---|
| `monitoring_agent` | Identifier | Deployment identifier of the monitoring agent. In the event the agent is local to ESC, the string must be set to `dmonaName://local_mona`. |

| Attribute Name | Data Type | Description |
|---|---|---|
| `key` | IdentifierInVnfd | This is the key in which the value for the monitoring agent should be stored. It **must** match the key used to identify the monitoring agent in the initial deployment. However, if the VNFD contained no agent definition then the key will reference a new KeyValue pair against which the agent reference should be stored, else update the existing value.<br>**Note** If the key supplied does not match the initial Key used to specify a monitoring agent, a new key will be created to store the new value against the VnfInstance. If the deployment is terminated and then re-instantiated without a new value for the monitoring agent, then the old value is used, which may not be the required outcome. |

# VNF Notifications During Migration

Once a request received for migration, ESC sends notifications for LCM operations for a particular VNF.

Following is the example for Starting Notification:

```
{
    "vnfInstanceId": "fd0bcc11-3f22-4c91-b363-1def72619db8",
    "timeStamp": "2020-07-23T08:38:47.876Z",
    "isAutomaticInvocation": false,
    "notificationType": "InfrastructureOperationOccurrenceNotification",
    "operationState": "STARTING",
    "notificationStatus": "START",
    "vnfLcmOpOccId": "143cfc34-cc14-414d-9374-d70d01ae7b5a",
    "_links": {
        "vnfInstance": {
            "href":
"https://172.16.235.30:8251/vnflcm/v2/vnf_instances/fd0bcc11-3f22-4c91-b363-1def72619db8"
        },
        "vnfLcmOpOcc": {
```

```
            "href":
"https://172.16.235.30:8251/vnflcm/v2/vnf_lcm_op_occs/143cfc34-cc14-414d-9374-d70d01ae7b5a"

        },
        "subscription": {
            "href":
"https://172.16.235.30:8251/vnflcm/v2/subscriptions/e54d546a-6753-4f35-86fa-6ef8ac07a9de"
        }
    },
    "subscriptionId": "e54d546a-6753-4f35-86fa-6ef8ac07a9de",
    "operation": "MONITORING_MIGRATION",
    "id": "6b737d3f-a485-46d9-9276-6802eb48decd"
}
```

If required, you can subscribe for other notifications.

**Note** The migration API is an extension for the existing subscription endpoint, `VNFM-preferred` for all other LCM operations .

For more information on the Subscription, see the Subscribing to Notifications section in the Alarms and Notifications for ETSI LCM Operations chapter.

# Error Scenarios

ETSI invokes the following error handling procedures for all its ETSI VNF lifecycle management (LCM) operations:

For more information on the VNF Lifecycle Management Error Handling Procedures, see Error Handling Procedures chapter.

A new property, `monitorMigration.terminalStateOnError`, is added to the ETSI service to determine what happens in the event of an error when ESC is performing the migration.

| Error / Interrupt | ESC Behaviour | ETSI-VNFM Behaviour | Resulting LcmOpOcc state | ETSI-VNFM Behaviour Resulting LcmOpOcc state with * [1] |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| Validation Failure | • Send validation error<br>• Rejects service update request | • Move operation to FAILED_TEMP<br>• Send notification with problem details containing error message from ESC Manager. | FAILED_TEMP | **ETSI-VNFM Behaviour**<br>• Move operation to FAILED<br>• Send notification with problem details containing error message from ESC.<br>**Resulting LcmOpOcc state**<br>FAILED |
| Monitoring already unset | • ESCManager will reject service update for monitoring migration if any of the VM is in VM_MONITOR_UNSET_STATE | • Move operation to FAILED_TEMP<br>• Send notification with problem details containing error message from ESC Manager. | FAILED_TEMP | **ETSI-VNFM Behaviour**<br>• Move operation to FAILED<br>• Send notification with problem details containing error message from ESC Manager.<br>**Resulting LcmOpOcc state**<br>FAILED |

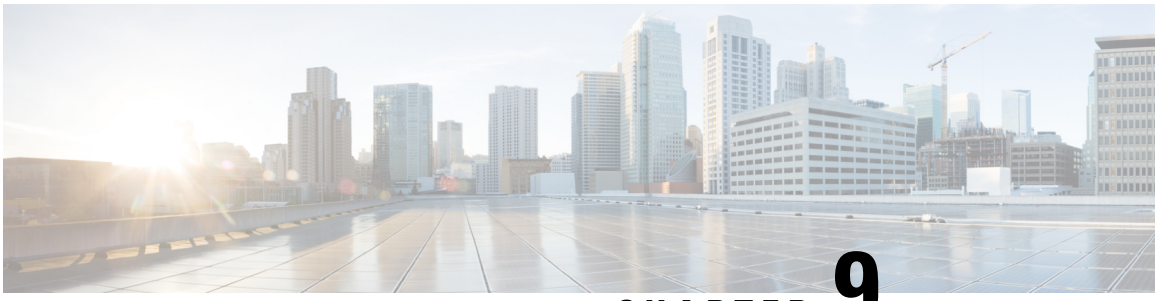
| | | | | |
|---|---|---|---|---|
| Unset monitor fails | • Unset monitor fails silently.<br>• Deleting rule from existing monitoring agent failed.<br>• Update deployment.<br>• Sends service update success notification.<br>• Set monitor on the new monitoring agent.<br>• Send VM_SET_MONITOR_STATUS and SVC_SET_MONITOR_STATUS notifications. | • Move operation to COMPLETED<br>• Send notification | COMPLETED | **ETSI-VNFM Behaviour**<br>• Move operation to COMPLETED<br>• Send notification<br>**Resulting LcmOpOcc state**<br>COMPLETED |
| Service Update fails | • Unset monitor on existing monitoring agent.<br>• Deployment update failed.<br>• Send service update failure notification.<br>• Set monitor on the existing/previous monitoring agent based on if the deployment was actually updated.<br>• Send VM_SET_MONITOR_STATUS notification.<br>• Send SVC_SET_MONITOR_STATUS notification. | • Move operation to FAILED_TEMP<br>• Send notification with problem details containing error message from ESC Manager. | FAILED_TEMP | **ETSI-VNFM Behaviour**<br>• Send notification with problem details containing error message from ESC Manager.<br>• Start rollback process (ROLLING_BACK)<br>**Resulting LcmOpOcc state**<br>ROLLING_BACK → ROLLED_BACK |

| | | | | |
|---|---|---|---|---|
| Set monitor fails | • Unset monitor from existing monitoring agent.<br>• Update deployment.<br>• Send service update success notification.<br>• Set monitor failed - Adding rule to new monitoring agent failed.<br>• Send VM_SET_MONITOR_STATUS notification with failure state.<br>• Skips set monitor for other VMs with same monitoring agent.<br>• Send SVC_SET_MONITOR_STATUS notification with partial failure/failure notification. | • Move operation to FAILED_TEMP<br>• Send notification with problem details containing error message from ESC Manager. | FAILED_TEMP | • Send notification with problem details containing error message from ESC Manager.<br>• Start rollback process (ROLLING_BACK)<br>**Resulting LcmOpOcc state**<br>ROLLING_BACK → ROLLED_BACK |
| Unset monitor fails (rollback) | • ETSI should not rollback on unset monitor failure. | N/A | N/A | N/A |

| | | | | |
|---|---|---|---|---|
| Service Update fails (rollback) | • If the deployment config was updated with the new monitoring agent during the service update failure, then a service update rollback will restore the previous monitoring agent and a set monitor is attempted on the previous monitoring agent.<br>• If the deployment config was not updated due to service update failure, then a service update rollback will not be accepted by ESCManager (service update will not be accepted unless there is something to be updated). | • Move operation to FAILED_TEMP<br>• Send notification with problem details containing error message from ESC Manager. | FAILED_TEMP | **ETSI-VNFM Behaviour**<br>• Move operation to FAILED_TEMP<br>• Send notification with problem details containing error message from ESC Manager.<br>**Resulting LcmOpOcc state**<br>FAILED_TEMP |

| | | | | |
|---|---|---|---|---|
| Set monitor fails (rollback) | • Unset monitor on new monitoring agent (because deployment config was already updated successfully).<br>• Update deployment with the previous monitoring agent.<br>• Send service update success.<br>• Set monitor on the previous monitoring agent.<br>• Send VM_SET_MONITOR_STATUS notification with success/failure state.<br>• Send SVC_SET_MONITOR_STATUS notification with success/failure/partial-failure state. | • Move operation to ROLLED_BACK<br>• Send notification<br>**Note:**<br>Rollback only checks for the service update notification not the service level set monitor notification. | ROLLED_BACK | • Move operation to ROLLED_BACK<br>• Send notification<br>**Note:**<br>Rollback only checks for the service update notification not the service level set monitor notification.<br>**Resulting LcmOpOcc state**<br>ROLLED_BACK |
| Cancel operation (during unset) | Since the request to ESC Core is atomic, cancel cannot be serviced. | N/A | N/A | N/A |
| Cancel operation (during service update) | Since the request to ESC Core is atomic, cancel cannot be serviced. | N/A | N/A | N/A |
| Cancel operation (during set) | Since the request to ESC Core is atomic, cancel cannot be serviced. | N/A | N/A | N/A |

[1] monitorMigration.terminalStateOnErrorOutcome flag true



CHAPTER 9

# Healing Virtual Network Functions



## Healing Virtual Network Functions Using ETSI API

As part of life cycle management, ESC heals the VNFs when there is a failure. The recovery policy specified during deployment controls the recovery. ESC supports recovery using the policy-driven framework, for more information, see Configuring a Recovery Policy Using the Policy-driven Framework in the Cisco Elastic Services Controller User Guide.

The healing parameters define the behavior that is monitored to trigger a notification to heal a VNF. These parameters are configured in the KPI section of each compute node in the VNFD with rules. The rules define the action as a result of these KPI conditions to heal a VNF.

The ETSI VNFM configures monitoring using the following two sections:

- kpi_data—defines the type of monitoring, events, polling interval, and other parameters
- admin_rules—defines the actions when the KPI monitoring events are triggered

Example:

```
vdu1:
  type: cisco.nodes.nfv.Vdu.Compute
  properties:
    name: Example VDU1
    description: Example VDU
    ...
     configurable_properties:
      additional_vnfc_configurable_properties:
        vim_flavor: { get_input: VIM_FLAVOR }
        bootup_time: { get_input: BOOTUP_TIME }
        vm_name_override: { get_input: VDU1_VM_NAME}
        recovery_action: REBOOT_THEN_REDEPLOY
        recovery_wait_time: 1
        kpi_data:
          VM_ALIVE-1:
            event_name: 'VM_ALIVE'
            metric_value: 1
            metric_cond: 'GT'
```

```
      metric_type: 'UINT32'
      metric_occurrences_true: 1
      metric_occurrences_false: 30
      metric_collector:
        type: 'ICMPPing'
        nicid: 1
        address_id: 0
        poll_frequency: 10
        polling_unit: 'seconds'
        continuous_alarm: false
admin_rules:
  VM_ALIVE:
    event_name: 'VM_ALIVE'
    action:
      - 'ALWAYS log'
      - 'FALSE recover autohealing'
      - 'TRUE esc_vm_alive_notification'
```

The previous example shows the default KPI and rule to support the service alive notification required to complete the deployment in ESC. For more information on KPI, rules, and the underlying data model that is exposed in the VNFD, see KPIs, Rules and, Metrics in the Cisco Elastic Services Controller User Guide.

The recovery of the VNF is to request action against the affected VNFCs determined by the recovery policy defined during the initial deployment or in the recovery request.

There are four types of actions for recovery. When an event denoting that an instance requires attention is received, a timer expires, or a manual recovery request is received. The healing workflow by default uses the recovery policy configured at either the VNF-level or at the VNFC-level within the VNFD. The supported policies are:

- REBOOT_THEN_REDEPLOY—first attempt to reboot the affected VNFCs; if this fails, then it attempts to redeploy the affected VNFCs (on the same host)
- REBOOT_ONLY—only attempt to reboot the VM
- RESET_THEN_REBOOT—reset the state of the VM (Openstack only) and then attempt to reboot the VM
- REDEPLOY_ONLY—only attempt to redeploy the VM

If the recovery policy is configured at a VNF-level, the policy applies to each constituent VNFC. If it is specified at VNFC-level, then that policy prevails. The monitoring agent monitors each VNFC and when a recovery situation arises, the message is converted to an alarm and sent to any subscribed consumers (e.g. an NFVO or Element Manager).

The HealVnfRequest contains a *cause* parameter that triggers different behaviors within the VNFM while processing the recovery request. If the *cause* is one of the values supported by the VNFM (and listed in the VNFD for the deployment as a supported cause) then certain *additionalParams* keys are activated to support the desired recovery action, as mentioned in the following table. If the NFVO supports the *cause*, the grant receives the *additionalParams* and allows the inputs to be modified before executing the recovery request.

If the *cause* is not one of the overriding causes supported by ESC, then it is assumed that the value provided is simply metadata and ignored; the VNFM would then use the recovery policy configured at the time of deployment. If the cause is supported by ESC, but not listed in the VNFD, then the request is rejected.

*Table 9: HealVnfRequest causes*

| Cause | additionalParams keys | Recovery behavior |
|---|---|---|
| APPLICATION_FAILURE | *Optional:*<br>vnfcInstanceId | The recovery attempts to reboot the entire VNF unless vnfcInstanceId is populated with a list of valid identifiers for VNFC instance(s) which constrains the recovery to those VNFCs only. For example:<br>{<br>...<br>"vnfcInstanceId": [<br>"resId1",<br>"resId2"<br>]<br>...<br>} |
| VIRTUALISATION_FAILURE | *Optional:*<br>vnfcInstanceId<br>resourceId<br>virtualStorageDescId | The treatment of the vnfcInstanceId is as per APPLICATION_FAILURE .<br>In addition, if there is a persistent volume to be replaced in the same request, the identifier for the volume in the VNFD and the VIM is supplied to avoid multiple requests. However, the VNFC to which the volume is attached must be in the list of VNFCs to be healed. **This persistent volume update is only applicable to Openstack VIMs.**<br>Any ephemeral ports and volumes managed by VNFM that are faulty or deleted will be recreated and attached to ensure the recovery is successful. |
| APPLICATION_OR_VIRTUALISATION_FAILURE | *Optional:*<br>vnfcInstanceId | As per APPLICATION_FAILURE.<br>Any ephemeral ports and volumes managed by VNFM that are faulty or deleted will be recreated and attached to ensure the recovery is successful if the VMs are redeployed. |
| INVALID_VM_STATE | *Optional:*<br>vnfcInstanceId | As per APPLICATION_FAILURE. |

| Cause | additionalParams keys | Recovery behavior |
|---|---|---|
| PERSISTENT_VOLUME_FAILURE | *Mandatory:*<br>resourceId<br>virtualStorageDescId<br>*Optional:*<br>vnfcInstanceId | The treatment of the vnfcInstanceId is as per APPLICATION_FAILURE. The mandatory keys allow a new persistent volume to replace the existing volume without redeploying the VM. Once the data model is updated and the volume is replaced, and the VM is rebooted. **This is only applicable to Openstack VIMs.** |
| CHANGE_PERSISTENT_VOLUME | *Mandatory:*<br>resourceId<br>virtualStorageDescId | The mandatory keys allow a new persistent (including multi-attach) volume to replace the existing volume without redeploying the VM. Once the data model is updated and the volume is replaced, the VM is rebooted. **This is only applicable to Openstack VIMs.** |
| VIM_FAILURE | *None* | No additionalParams keys are activated, however the Grant from the NFVO **must** include new vimConnectionInfo to redeploy the VNF on a VIM that is available else the recovery request is rejected.<br>**Note** The old deployment is not removed since the VIM is assumed to be unavailable if this cause is used; it needs to be manually removed once the VIM is reachable again. |

If autoheal is *enabled* on the VNF instance, then ESC automatically attempts to recover the VNF based on the recovery policy configured on deployment. This may be configured in the VNFD or modified against the VNF instance before instantiation.

To modify the autoheal flag (*isAutohealEnabled*) VNF instance resource, see Modifying Virtual Network Functions, on page 41.

If autoheal is *not enabled*, only the alarm is dispatched to all the subscribers. The subscriber can initiate a manual HealVnfRequest, as per the following examples. The parameters are optional by default but subject to the rules in table 9 for the different causes.

Example for *SOL003*:

Method type:

```
POST
```

VNFM Endpoint:

```
/vnf_instances/{vnfInstanceId}/heal
```

HTTP Request Header:

```
Content-Type:application/json
```

Request Payload (ETSI data structure: HealVnfRequest)

```
{
"cause":"VIRTUALISATION_FAILURE",
"additionalParams": {
 "virtualStorageDescId": "cf-cdr1-vol",
 "resourceId": " d8771acb-a32f-66dg-7bc2-8f4ec333ccb8"
},
"vnfcInstanceId": [b9909dde-e21e-45ec-9cc0-9e9ae413eee0"]
}
```

Example for *SOL002*:

```
POST /vnf_instance/{vnfInstanceId}/heal
 {
   "vnfcInstanceId": ["b9909dde-e21e-45ec-9cc0-9e9ae413eee0"],
   "cause": "b9909dde-e21e-45ec-9cc0-9e9ae413eee0"
 }
```

The list of `vnfcInstanceIds` constrains recovery to the required VNFCs. However, the absence of this list means the request applies to the entire VNF.

The cause in the SOL002 HealVnfRequest has the same behavior as in the SOL003 API.

For information on monitoring, see Monitoring Virtual Network Functions Using ETSI API, on page 47.

# Recovering VM During Healing

If the recovery action is `REDEPLOY_ONLY` or `REBOOT_THEN_REDEPLOY` and VM needs to be redeployed during SOL002 and SOL003 heal operation, check whether the:

- ephemeral volume is missing or in error state; and recreate them.
- ephemeral neutron port is missing or in error state; and recreate them.

**Note** SOL002 heal is constrained to specific VNFCs, if `vnfcInstanceIds` are supplied in the heal payload.

# Updating an Existing Deployment During Healing

After a deployment is created successfully, the resources within it can be updated. As part of deployment management, you can add or remove resources, or update the configuration of the existing resources. These updates can be carried out in a running deployment. The resources are updated as part of the recovery process.

You can update an existing deployment (provisioned through the ETSI NFV MANO API) during the healing workflow. During the Heal request, the existing image and Day-0 parameters are compared and updated to the new ones provided as part of a subsequent Heal request.

The healing workflow allows:

- Updating the deployment model with the new image and Day-0 configuration
- Re-applying new or existing configuration data to the VNFC when healing with an upgraded image

**Note** You must redeploy the VNF after any update to the data model *if* the change is not carried out directly on the VIM.

After supplying new *additionalParams* via the HealVnfRequest, if the Grant response (from the NFVO) also supplies a new image or new *additionalParams*, this would also trigger a service update.

If the NFVO determines that the deployment should be moved as part of a redeployment, then the Grant provides a new *zoneId* to reflect the new placement of the resources.

The recovery action takes place after the service update is complete. In the event of a redeploy, it considers the up-to-date deployment model to ensure that any deployed updates are not reverted.

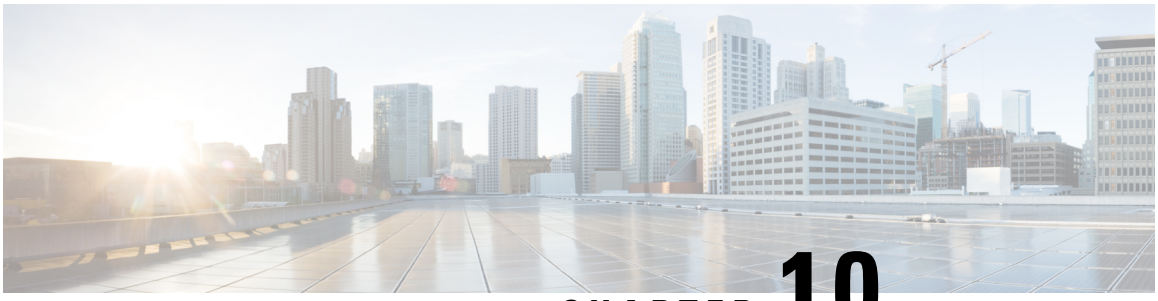
The following example shows the details NFVO returns in the Grant to trigger a service update with new *additionalParams* and/or a new *vimSoftwareImageId*.

Example:

```
{
  "headers" : {
    "Content-Type" : [ "application/json" ],
    "Location" : [
"http://{nfvoApiRoot}/sol003/default/grant/v1/grants/38ba2103-dab3-450e-992b-ee85aad6c899"
 ],
    "Content-Length" : [ "22935" ],
  },
  "body" : {
    "id" : "38ba2103-dab3-450e-992b-ee85aad6c899",
    "vnfInstanceId" : "6aaf527c-0093-49c3-ba2e-49fc6d8a4f71",
    "vnfLcmOpOccId" : "cdc5d9b3-81a0-400b-a4d9-97d1b3e117d9",
    "_links" : {
      "self" : {
        "href" :
"http://{nfvoApiRoot}/sol003default/grant/v1/grants/38ba2103-dab3-450e-992b-ee85aad6c899"
      },
      "vnfLcmOpOcc" : {
        "href" :
"https://{vnfmApiRoot}/vnflcm/v2/vnf_lcm_op_occs/cdc5d9b3-81a0-400b-a4d9-97d1b3e117d9"
      },
      "vnfInstance" : {
        "href" :
"https://{vnfmApiRoot}/vnflcm/v2/vnf_instances/6aaf527c-0093-49c3-ba2e-49fc6d8a4f71"
      }
    },
    "vimConnections" : {
    "default_openstack_vim": {
      "vimType" : "OPENSTACK_V3",
      "vimId" : "595b0bc2-8dad-4087-abdf-ebe3b0b14d96",
      "interfaceInfo" : {
```

```
        "endpoint" : "https://{vimApiRoot}/v3"
      },
      "accessInfo" : {
        "password" : "********",
        "project" : "cisco",
        "projectDomain" : "demo",
        "region" : "RegionOne",
        "userDomain" : "demo",
        "username" : "********"
      }
    } },
    "zones" : [{
      "id" : "1773873a-ab15-4a7b-b024-bc338425ed24",
      "zoneId" : "nova"
    },{
      "id" : "1773873a-ab15-4a7b-b024-bc555555ed55",
      "zoneId" : "nova2"
    }],
    "addResources" : [{
      "resourceDefinitionId" : "res-a6252dbf-b418-4f88-b8a9-14d8f3942938",
      "vimConnectionId" : "myVimConnection",
      "zoneId" : "1773873a-ab15-4a7b-b024-bc555555ed55"
    }],
    "vimAssets" : {
      "softwareImages" : [ {
        "vnfdSoftwareImageId" : "s3",
        "vimSoftwareImageId" : "3a609da7-e2b2-4e27-91b6-7bcabe902820",
        "vimConnectionId" : "myVimConnection"
      }, {
        "vnfdSoftwareImageId" : "s4",
        "vimSoftwareImageId" : "3a609da7-e2b2-4e27-91b6-7bcabe902820",
        "vimConnectionId" : "myVimConnection"
      } ]
    }
  },
  "additionalParams": [
    ...
    /* changed additionalParams */
    "CF_VIP_ADDR": "10.123.23.4",
    "SF_VIP_ADDR": "10.123.24.4",
    ...
  ],
  "statusCode" : "CREATED",
  "statusCodeValue" : 201
}
```

For more information on healing, see Healing Virtual Network Functions Using ETSI API, on page 71.



CHAPTER 10

# Scaling Virtual Network Functions



# Scaling Virtual Network Functions Using ETSI API

One of the main benefits of ESC is its capability to elastically scale a service. This allows a VNFC that performs a particular role or aspect within the VNF to be able to service requests and scale out to meet high demand or scale in when being under utilized. This aspect may span across multiple VNFCs.

The scaling requests may be manual or automatic. The different approaches to accomplishing scaling are detailed below.

For more details on these concepts and specification, please see Annex B of *ETSI GS NFV-SOL 003*.

For information on Scaling VNFs using REST and NETCONF APIs, see the *Cisco Elastic Services Controller User Guide*.

**Scale**

The Scale VNF request uses the *scaleStatus*, an attribute found as part of the instantiatedVnfInfo when querying a VnfInstance resource. This attribute describes the current scale level of each aspect in the VNF, for example:

```
"scaleInfo": [
  {
    "aspectId": "webserver", "scaleLevel": "4"
  },
  {
    "aspectId": "processing", "scaleLevel": "2"
  }
]
```

This forms the starting point for a Scale VNF request, which allows a single aspect to be scaled horizontally (i.e. adding or removing VNFCs) relative to the current *scaleLevel* for that dimension of the VNF. Any scaling operation on an aspect will be applied to each VNFC that supports that aspect.

**Note** The current specification does not support vertical scaling (adding/removing resources to/from existing VNFC instances) at this time.

Request Payload (ETSI data structure: ScaleVNFRequest)

```
{
    "type": "SCALE_OUT",
    "aspectId": "processing",
    "numberOfSteps": 1,
    "additionalParams": {}
}
```

The above payload results in the *scaleStatus* example above being updated to and the addition of the number of VNFCs for this step required to scale out to scaleLevel 3:

```
"scaleInfo": [
  {
    "aspectId": "webserver", "scaleLevel": "4"
  },
  {
    "aspectId": "processing", "scaleLevel": "3"
  }
]
```

To understand the scaling steps and other related policies configured to support scaling, see the VNFD Policies for Scaling.

### Scale To Level

The Scale VNF To Level request, rather than the relative scaling that Scale VNF offers, specifies the absolute scale result desired and so some aspects may be scaled out and others scaled in. This option uses one of the two approaches to define the scaling required:

- instantiation level
- scale level

These are mutually exclusive and allow for more than one aspect to be scaled in a single request.

#### Instantiation Level

An Instantiation level is a predefined size for each aspect, where each level has a scale level associated with each aspect. There is no further granularity offered and so the entire VNF (that is, all aspects) is scaled according to the instantiation level requested.

Example:

Request Payload (ETSI data structure: ScaleVNFToLevelRequest)

```
{
    "instantiationLevelId": "premium"
}
```

See the VNFD Policies for the definition of instantiation levels.

#### Scale Level

The Scale Level is also a pre-defined size for each aspect where each aspect has target VNFCs, defined step_deltas (since each scaling step may not be uniform) and a maximum scale level. The policies that define this option allow the different targets to have different scaling outcomes.

**Note** The scale level does not represent the number of VMs; for example scaleLevel=0 means the initial number of instances (initial delta) for that aspect on the target VNFC and scaleLevel=1 is the initial delta plus the first scaling step defined for that aspect and VNFC tuple.

Request Payload (ETSI data structure: ScaleVNFToLevelRequest)

```
{
    "scaleInfo": [
      {
            "aspectId": "processing",
            "scaleLevel": "2"
      },
      {
            "aspectId": "webserver",
            "scaleLevel": "3"
      }
    ]
}
```

For information on definition of scale levels, See the VNFD Policies for Scaling.

### ESC ETSI Support for Trunks and Subports

**ETSI VLAN Trunk:**

Introduction:

For OpenStack VIMs, starting from 5.8, ESC supports trunks and VLANs. The initial release was limited to the ESC Netconf/APIs and trunk enabled VNFs were not scalable. The introduction of TOSCA SOL003 3.5.1 version provided new node types allowing an ETSI VNFD to define trunks and subports. With the ESC 5.9 release, the ETSI VNFM supports scalable trunks and subports.

**Defining a Trunk in the VNFD:**

The TOSCA type *tosca.nodes.nfv.VduSubCp* is available from SOL001 3.5.1. Use the VNFD version that is SOL001 3.5.1 or higher.

Apply the ETSI Trunk Mode to CPs that is Connection Points between the VDU which is Virtualisation Deployment Unit and VL which is the Virtual Link or network. For a given CP, setting a trunk_mode property value as *true* signifies it as being the parent port for a trunk.

Example Payload:

```
s3_nic0:

type: tosca.nodes.nfv.VduCp
  properties:
    layer_protocols: [ ipv4 ]
    protocol:
      - associated_layer_protocol: ipv4
    trunk_mode: true  # denotes the parent port
    order: 0
    management: false
    allowed_address_pairs:
      - ip_address: 192.168.0.0/18
  requirements:
    - virtual_binding: s3
```

Setting the *trunk_mode* property creates a trunk. The CP is the primary port for the VDU linked by *virtual_binding*. The trunk name is generated in the format "trunk-" + VDU name + "-" + index number. The index is based on the number of CPs in trunk mode for the current VDU. Note that setting *trunk_mode* can be done at instantiation time.

**Defining Subports in the VNFD:**

To make a trunk useful, the trunk needs to connect to other networks through subports. A subport is defined with a node of type *tosca.nodes.nfv.VduSubCp* as follows:

Sample Payload:

```
s3_nic0_1:
  type: tosca.nodes.nfv.VduSubCp
  properties:
    layer_protocols: [ ethernet, ipv4 ]
    segmentation_type: vlan
    segmentation_id: 303
    management: false
  requirements:
    - trunk_binding: s3_nic0
    - virtual_link: a_vlan_VL
```

Here the segmentation type and ID are configured. The requirements properties have two links:

- trunk_binding: The node name of the CP where the primary port is defined that is trunk_mode set to true
- virtual_link: The name of the VL node to which this subport will be connected.

Example Payload for VL that is type tosca.nodes.nfv.VnfVirtualLink:

```
a_vlan_VL:
  type: tosca.nodes.nfv.VnfVirtualLink
  properties:
    connectivity_type:
      layer_protocols: [ ethernet ]
    description: subport VL
    vl_profile:
      max_bitrate_requirements:
        root: 100000
      min_bitrate_requirements:
        root: 0
      virtual_link_protocol_data:
      - associated_layer_protocol: ethernet
        l2_protocol_data:
          vlan_transparent: false
          segmentation_id: 303
```

**Note** Configure the subports with the JSON payload at instantiation time together with user data that are input variables.

The following shows the traditional dep.xml produced by ETSI constructs:

```
 <trunk>
    <name>trunk-name-0</name> <!-- Derived from VDU name and index -->
    <parent_nicid>0</parent_nicid> <!-- Primary port -->
    <subports>
        <subport>
            <name>trunk-name-0-subport-0</name> <!-- Derived from trunk name and subport
index -->
```

```
            <network>child-net</network>
            <segmentation_type>vlan</segmentation_type>
            <segmentation_id>500</segmentation_id>
            <binding_profile>
                <property>
                    <name>physical_network</name>
                    <value>physnet_tenant1</value>
                </property>
                <property>
                    <name>trusted</name>
                    <value>true</value>
                </property>
            </binding_profile>
        </subport>
    </subports>
</trunk>
```

**ETSI VNF SCALING:**

Trunk and subports scale automatically depending on the policy defined in the VNFD. As ESC scales the VNF up and down, additional trunks and subports are created or deleted as necessary. These are managed by ESC. ESC ensures VIM resources are cleaned up during LCM operations that are modify, delete.

**Note** During scaling, ESC duplicates the trunk and port names and relies on resource IDs when updating or deleting.

For ETSI, scaling is controlled according to the scaling policies.

```
######
# VM #
######
    - vm_initial_delta:
        type: tosca.policies.nfv.VduInitialDelta
        properties:
          initial_delta:
            number_of_instances: 2
        targets: [ s3_nic0 ]

    - vm_instantiation_levels:
        type: tosca.policies.nfv.VduInstantiationLevels
        properties:
          levels:
            default:
              number_of_instances: 2
        targets: [ s3_nic0 ]

    - vm_scaling_aspect_deltas:
        type: tosca.policies.nfv.VduScalingAspectDeltas
        properties:
          aspect: default_scaling_aspect
          deltas:
            delta_1:
              number_of_instances: 2
            delta_2:
              number_of_instances: 3
        targets: [ s3_nic0 ]
```

The following shows the traditional dep.xml produced by ETSI constructs:

```
<scaling>
    <min_active>1</min_active>
```

```
    <max_active>2</max_active>
</scaling>
```

And the appropriate VM group blocks are created.

**Scaling Behaviour within ESC:**

When a VMGroup is scaled up, then the corresponding trunks and subports are created, and the deployment detail queries through REST or Netconf APIs show the trunk and subport details.

When a VMGroup is scaled down, then the corresponding trunks and subports are deleted from the VIM, and deployment detail queris through REST or Netconf show the new trunk and subport details.

### Updating SOL001 Parser to Support The trunk_mode Property for the Connection Points

The interfaces currently configured by ESC are not trunk ports, and so they do not support the definition of sub-ports. To use the networks more efficiently, segment the network using VLANs to connect multiple Layer 2 networks to a single pass-through interface. The following data model supports this configuration.

The following is an extract of a VNFD for a VPC-DI, with a parent port shown to be a trunk port, with 2 subports defined - one with an external VL connection that is exposed as an external connection through `substitution_mappings` and the other connected to an internal VL that is both of which specify their own segmentation Id.

```
s3_nic0:
      type: tosca.nodes.nfv.VduCp
      properties:
        layer_protocols: [ ipv4 ]
        protocol:
          - associated_layer_protocol: ipv4
        trunk_mode: true  # denotes the parent port
        order: 0
        management: false
        allowed_address_pairs:
          - ip_address: 192.168.0.0/18
      requirements:
        - virtual_binding: vdu_node_1

    s3_nic0_1:
      type: tosca.nodes.nfv.VduSubCp
      properties:
        layer_protocols: [ ipv4 ]
        protocol:
          - associated_layer_protocol: ipv4
        trunk_mode: false
        segmentation_type: vlan
        segmentation_id: 303
        management: false
      requirements:
        - trunk_binding: s3_nic0
        - virtual_link: a_vlan_VL
```

**Note** The `trunk_mode` is set to `true`, indicating that when the port is created, it is used as a trunk port and sub-ports are configured within the trunk network.

This results in the following deployment XML:

```
<trunks>
                    <trunk>
                       <name>trunk-vdu_node_1-0</name>
```

```
            <parent_nicid>0</parent_nicid>
            <subports>
               <subport>
                  <name>trunk-vdu_node_1-0-subport-0</name>
                  <network>a_vlan_VL</network>
                  <segmentation_type>vlan</segmentation_type>
                  <segmentation_id>303</segmentation_id>
               </subport>
               <subport>
                  <name>trunk-vdu_node_1-0-subport-1</name>
                  <network>a_vlan_VL</network>
                  <segmentation_type>vlan</segmentation_type>
                  <segmentation_id>304</segmentation_id>
               </subport>
            </subports>
         </trunk>
      </trunks>
```

# VNFD Policies for Scaling

There are a number of policies that make up the overall scaling behavior of a VNF. These policies will support the various scaling approaches described above. The first policy defines the aspects that may be scaled (or not):

```
policies:
    - scaling_aspects:
        type: tosca.policies.nfv.ScalingAspects
        properties:
          aspects:
            webserver:
              name: 'webserver'
              description: 'The webserver cluster.'
              max_scale_level: 5
              step_deltas:
                - delta_1
            processing:
              name: 'processing'
              description: 'An example processing function'
              max_scale_level: 3
              step_deltas:
                - delta_1
                - delta_2
                - delta_1
            database:
              name: 'database'
              description: 'A test database'
              max_scale_level: 0
```

You can see in this example that the database aspect has a max_scale_level of 0, which denotes that it cannot be scaled out - this does not mean 0 instances of that aspect - see the algorithm below to see why. The webserver aspect only has a single step_delta, meaning that all scaling steps are uniform whereas the processing aspect has different step_deltas specified for each scaling step. This is called non-uniform scaling. This is only the declaration of the aspects of this VNF, and this is one of the policies used to perform the validation when a scaling request is received.

Next, they must be applied to VNFCs to control their behavior:

```
- db_initial_delta:
        type: tosca.policies.nfv.VduInitialDelta
        properties:
          initial_delta:
```

```
          number_of_instances: 1
      targets: [ vdu1 ]

  - ws_initial_delta:
      type: tosca.policies.nfv.VduInitialDelta
      properties:
        initial_delta:
          number_of_instances: 1
      targets: [ vdu2, vdu4 ]

  - pc_initial_delta:
      type: tosca.policies.nfv.VduInitialDelta
      properties:
        initial_delta:
          number_of_instances: 1
      targets: [ vdu3 ]

  - ws_scaling_aspect_deltas:
      type: tosca.policies.nfv.VduScalingAspectDeltas
      properties:
        aspect: webserver
        deltas:
          delta_1:
            number_of_instances: 1
      targets: [ vdu2, vdu4 ]

  - pc_scaling_aspect_deltas:
      type: tosca.policies.nfv.VduScalingAspectDeltas
      properties:
        aspect: processing
        deltas:
          delta_1:
            number_of_instances: 1
          delta_2:
            number_of_instances: 2
      targets: [ vdu2, vdu4 ]
```

In the examples above, the VNFCs are identified as targets; the aspects could have different behaviours on different VNFCS, but this is not shown here. The definition of the step_deltas are also shown here which are used in the validation and generation of scaling requests (these steps are inferred by the scale level requested). The minimum number of instances of a VNFC is always assumed to be 0 and the maximum number is calculated by the following algorithm:

initial_delta plus the number of instances for each step up to the max_scale_level.

These policies are considered for the scale-level based scaling. There are similar constructs used for instantiation-level based scaling.

```
- instantiation_levels:
        type: tosca.policies.nfv.InstantiationLevels
        properties:
          levels:
            default:
              description: 'Default instantiation level'
              scale_info:
                database:
                  scale_level: 0
                webserver:
                  scale_level: 0
                processing:
                  scale_level: 0
            premium:
              description: 'Premium instantiation level'
              scale_info:
```

```
            database:
              scale_level: 0
            webserver:
              scale_level: 2
            processing:
              scale_level: 3
        default_level: default
```

Similar to the scaling aspects, the first part of the definition of instantiation levels is just their declaration. Here each aspect must already be declared and then each aspect's scale_level is declared for the instantiation level; a default instantiation level is also stipulated in the event that no other is specified. What each scale_level means for each VNFC is further elaborated upon in the VduInstantiationLevels policies, for example:

```
- ws_instantiation_levels:
        type: tosca.policies.nfv.VduInstantiationLevels
        properties:
          levels:
            default:
              number_of_instances: 1
        targets: [ vdu2, vdu4 ]
```

So these policies together state that the default instantiation level is 'default' which will result in the webserver aspect being instantiated at scale_level 0 which is 1 VNFC instance.

# Dependencies on Multiple IP Addresses

### Static IP Addresses

If the VNFC has connection points configured with a static IP address, the VNFC cannot scale as there are no further IP addresses to assign to the connection points on the newly spun up VNFC instances. Instead, you can specify a pool of static IP addresses in the instantiate request or Grant response (in the extVirtualLinks element) as a list:

- in fixedAddresses in a single cpProtocolData
- of individual fixedAddresses in multiple cpProtocolData

**Note** A list of ipAddresses in a single cpProtocolData assigns all the IP addresses to a single port on a single VNFC instance.

Alternatively, a contiguous range can also be supplied in an ipAddresses entry, as an addressRange. If the specific IP addresses need not be stipulated, then a subnetId can be used, as per the example in Instantiating Virtual Network Functions, on page 28.

The following example explains how to create a static IP pool with four IP addresses by specifying them as a list in fixedAddresses in a single cpProtocolData:

```
    {
…
"extVirtualLinks": [
  {
    "id": "extVL-dbf477ad-199a-47ff-939a-cb0101c92585",
    "resourceId": "ext-net",
    "extCps": [
      {
        "cpdId": "ecp_1_vdu_node_1",
```

```
          "cpConfig": {
            "cp1": {
              "cpProtocolData": [
                {
                  "layerProtocol": "IP_OVER_ETHERNET",
                  "ipOverEthernet": {
                    "ipAddresses": [
                      {
                        "type": "IPV4",
                        "fixedAddresses": [
                          "172.16.0.10",
                          "172.16.0.11",
                          "172.16.0.12",
                          "172.16.0.13"
                        ]
                      }
                    ]
                  }
                }
              }
            }
          ]
        }
      ]
    }
  ]
  …
}
```

The same pool of IP addresses can also be created by specifying them as individual fixedAddresses in multiple cpProtocolData:

```
{
  …
  "extVirtualLinks": [
    {
      "id": "extVL-dbf477ad-199a-47ff-939a-cb0101c92585",
      "resourceId": "ext-net",
      "extCps": [
        {
          "cpdId": "ecp_1_vdu_node_1",
          "cpConfig": {
            "cp1": {
              "cpProtocolData": [
                {
                  "layerProtocol": "IP_OVER_ETHERNET",
                  "ipOverEthernet": {
                    "ipAddresses": [
                      {
                        "type": "IPV4",
                        "fixedAddresses": [
                          "172.16.0.10"
                        ]
                      }
                    ]
                  }
                },
                {
                  "layerProtocol": "IP_OVER_ETHERNET",
                  "ipOverEthernet": {
                    "ipAddresses": [
                      {
                        "type": "IPV4",
                        "fixedAddresses": [
```

```
                        "172.16.0.11"
                      }
                    }
                  ]
                }
              },
              {
                "layerProtocol": "IP_OVER_ETHERNET",
                "ipOverEthernet": {
                  "ipAddresses": [
                    {
                      "type": "IPV4",
                      "fixedAddresses": [
                        "172.16.0.12"
                      ]
                    }
                  ]
                }
              },
              {
                "layerProtocol": "IP_OVER_ETHERNET",
                "ipOverEthernet": {
                  "ipAddresses": [
                    {
                      "type": "IPV4",
                      "fixedAddresses": [
                        "172.16.0.13"
                      ]
                    }
                  ]
                }
              }
            ]
          }
        ]
      }
    ]
  ]
  …
}
```

The same pool of IP addresses created using an addressRange:

```
{
  …
  "extVirtualLinks": [
    {
      "id": "extVL-dbf477ad-199a-47ff-939a-cb0101c92585",
      "resourceId": "ext-net",
      "extCps": [
        {
          "cpdId": "ecp_1_vdu_node_1",
          "cpConfig": {
            "cp1": {
              "cpProtocolData": [
                {
                  "layerProtocol": "IP_OVER_ETHERNET",
                  "ipOverEthernet": {
                    "ipAddresses": [
                      {
                        "type": "IPV4",
                        "addressRange": {
                          "minAddress": "172.16.0.10",
```

```
                        "maxAddress": "172.16.0.13"
                      }
                    }
                  ]
                }
              }
            ]
          }
        }
      }
    ]
  }
  ]
  …
}
```

The implementation of these IP address pools conforms to the *ETSI NFV MANO SOL003* specification, *chapter 4.4.1.10*.

### Static MAC Addresses

If the VNFC has connection points configured with a static MAC address, the VNFC cannot scale as there are no further MAC addresses to assign to the connection points on the newly spun up VNFC instances. Instead, a pool of further static MAC addresses can be specified in the instantiate request or grant response.

Static MAC address pools can be created in the extVirtualLinks element of the instantiate request or grant response by specifying the macAddress in multiple cpProtocolData.

The following example shows how to create a static MAC pool with four MAC addresses by specifying them in multiple cpProtocolData:

```
{
  …
  "extVirtualLinks": [
    {
      "id": "extVL-dbf477ad-199a-47ff-939a-cb0101c92585",
      "resourceId": "ext-net",
      "extCps": [
        {
          "cpdId": "ecp_1_vdu_node_1",
          "cpConfig": {
            "cp1": {
              "cpProtocolData": [
                {
                  "layerProtocol": "IP_OVER_ETHERNET",
                  "ipOverEthernet": {
                    "macAddress": "fa:16:3e:0b:10:10",
                    "ipAddresses": [
                      {
                        "type": "IPV4",
                        "fixedAddresses": [
                          "172.16.0.10"
                        ]
                      }
                    ]
                  }
                },
                {
                  "layerProtocol": "IP_OVER_ETHERNET",
                  "ipOverEthernet": {
                    "macAddress": "fa:16:3e:0b:10:11",
                    "ipAddresses": [
                      {
```

```
                        "type": "IPV4",
                        "fixedAddresses": [
                          "172.16.0.11"
                        }
                      }
                    ]
                  }
                },
                {
                  "layerProtocol": "IP_OVER_ETHERNET",
                  "ipOverEthernet": {
                    "macAddress": "fa:16:3e:0b:10:12",
                    "ipAddresses": [
                      {
                        "type": "IPV4",
                        "fixedAddresses": [
                          "172.16.0.12"
                        ]
                      }
                    ]
                  }
                },
                {
                  "layerProtocol": "IP_OVER_ETHERNET",
                  "ipOverEthernet": {
                    "macAddress": "fa:16:3e:0b:10:13",
                    "ipAddresses": [
                      {
                        "type": "IPV4",
                        "fixedAddresses": [
                          "172.16.0.13"
                        ]
                      }
                    ]
                  }
                }
              ]
            }
          ]
        }
      ]
    }
  ]
  …
}
```

### Day Zero Configuration

After deploying the VNFs, day 0 variables are configured in the VNFC instance for the deployment service. In most cases, the values for the day 0 configuration is constant. In other cases, there is a resource pool of values supplied to the day 0 parameter to allow new values to be assigned to the new VNFC instances.

Day 0 configuration within the vendor_section of the VNFD:

```
    vdu3:
      type: cisco.nodes.nfv.Vdu.Compute
      properties:
        name: 'Processing1'
        description: 'Processing VNFC'
        vdu_profile:
          min_number_of_instances: 1
          max_number_of_instances: 5
        vendor_section:
          cisco_esc:
```

```
                config_data:
                  '/tmp/OSRESTTestETSIDay0_Inline_data.cfg':
                    data: |
                      NODE_NAME $NODE_NAME
                      NUM_OF_CPU $NUM_OF_CPU
                      MEM_SIZE $MEM_SIZE
                      PROXY_ADDRS $PROXY_ADDRS
                      SPECIAL_CHARS $SPECIAL_CHARS
                    variables:
                      NODE_NAME: vdu_node_1
                      NUM_OF_CPU: 1
                      MEM_SIZE: 1GB
                      PROXY_ADDRS: ["1.1.1.1", "1.1.2.1", "1.1.3.1", "1.1.4.1", "1.1.5.1",
"1.1.6.1", "1.1.7.1"]
                      SPECIAL_CHARS: '`~!@#$%^&*()-_=+[{]}|;:<.>/?'
```

In the above example the day 0 configuration is specified inline, with velocity variables defined in the target configuration. Each of these variables are supported by a variable with one or more values. In order to support multiple values for the $PROXY_ADDRS variable, a list of values are provided. These values are used to populate subsequent uses of the variable on new instances of the VNFC.

For information on day 0 configuration in the deployment data model, see Day Zero Configuration in the *Cisco Elastic Services Controller User Guide*.

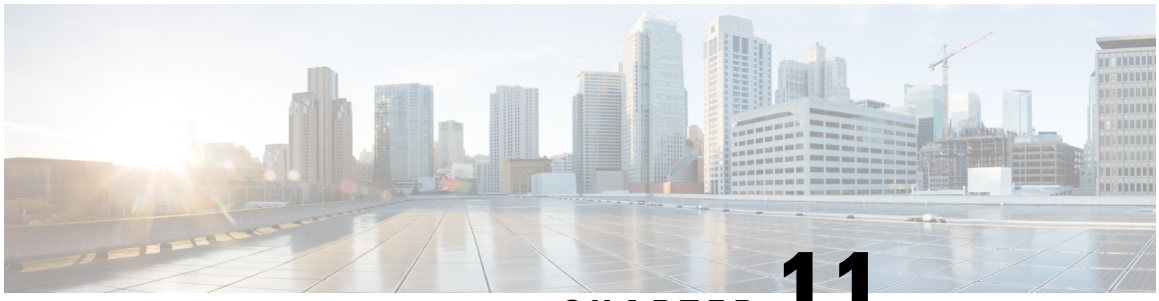
# Autoscaling of VNFs

KPIs, rules and actions defined in the VNFD determine the conditions under which scaling must be considered. The details are provided in Monitoring Virtual Network Functions. The scaling policies are also defined in the VNFD using several policy types that control the allowed scaling boundaries. These policy items are described below.

After deployment, ESC configures a monitoring agent (this may be the centralised or distributed instance) with the KPIs to monitor each VNFC. The scaling workflow begins if a KPI reaches its threshold; based on the action defined, ESC performs scale in or scale out and generates appropriate notifications and event logs. This is subject to some built-in functions that can be specified such as `log` or an onboarded script.

ESC sends appropriate notifications to the subscribed consumers. At this time, ESC interrogates the VNF instance resource for the *isAutoscaleEnabled* flag (this is set initially by the value in the VNFD but can be modified after creation). If this flag is set to true, ESC invokes the scaling workflow (instigated using a *ScaleVnfToLevelRequest* to request the scaling of multiple aspects in a single request). If the *isAutoscaleEnabled* is set to false, then the control is with an external system such as an NFVO or EM to trigger the desired action using the requests described above.

**Note** While creating an auto scaling or auto healing request, any new external requests are blocked. The user is notified of the corresponding response and problem details of the blocked request.



CHAPTER 11

# Managing VNF Snapshot



# Managing VNF Snapshots

A snapshot is a mechanism that allows the creation of a new image on OpenStack from a running Instance. VNF Snapshots mainly serves two purposes:

- As a backup mechanism: Save the main disk of the instance to an image and later boot a new instance from this image with saved data.
- As a templating mechanism: Customize a base image and save it to use as a template for new instances.

The full lifecycle of a VNF snapshot can be managed using ETSI-defined APIs.

**Notes and Limitations:**

Before using the ETSI APIs for VNF Snapshots, it is important to understand the following points:

- There are no changes required for the VNF Descriptor files to use VNF snapshots.Snapshot functionality exists for VNFs deployed against an OpenStack VIM. If a snapshot is attempted for a VNF deployed on a non-OpenStack VIM such as CVIM or VMWare, then the appropriate error message is generated.
- As per ETSI specifications, the API root is only available under the new, "v2" URL, that is http://192.168.201.33:8250/or_vnfm/vnflcm/v2/vnf_snapshots for SOL003 APIs or http://192.168.201.33:8250/ve_vnfm/vnflcm/v2/vnf_snapshots for SOL002 APIs.
- If a VNF uses one or more volumes that are either VNF-managed volumes or out-of-band volumes, then a resultant snapshot of the VNF results in image and volume snapshot resources generated on OpenStack.
- Deletion of a VNF within ETSI does not trigger deletion of any previous snapshots taken of the VNF. Therefore, delete the VNF Snapshots before deletion of the parent VNF.

**API Resources for Snapshot Management:**

Create, Query, Revert and Delete the VNF Snapshots using the ETSI APIs.

**VNF Snapshot Creation:**

The creation of a snapshot with the associated resources generated on OpenStack is a two-step process:

- Creating a snapshot resource

- Creating the snapshot given a snapshot resource ID and existing VNF Instance ID

**API Execution**

The following shows the operations, sample payloads, and the API responses using Linux curl as a client, executing the APIs on the ESC VM that is a local host itself:

- Create a snapshot resource - note the returned "id" value

```
[admin@host]$ curl -s --user 'admin:*******' -X POST --data {} -H
'Content-Type:application/json' http://localhost:8250/or_vnfm/vnflcm/v2/vnf_snapshots  |
python -m json.tool
{
    "_links": {
        "self": {
            "href":
"http://localhost:8250/or_vnfm/vnflcm/v2/vnf_snapshots/fc7f055c-a541-4801-9295-299ce806763f"

        }
    },
    "id": "fc7f055c-a541-4801-9295-299ce806763f"
}
```

- Create the snapshot given snapshot resource ID and an existing VNF Instance ID

```
[admin@host]$ cat create_snapshot.json
{
    "vnfSnapshotInfoId": "fc7f055c-a541-4801-9295-299ce806763f"
}

[admin@host]$ curl -s --user 'admin:*******' -X POST --data @create_snapshot.json  -H
'Content-Type:application/json'
http://localhost:8250/or_vnfm/vnflcm/v2/vnf_instances/c9cdf5c8-3681-4641-ba7e-df40539815b5/create_snapshot
```

The payload must contain the VNF Snapshot ID from the earlier operation, and the VNF Instance ID in the URL must refer to an INSTANTIATED VNF.

**Error Conditions:**

- An error returns if the VNF Snapshot ID or the VNF Instance ID are invalid.
- OpenStack-specific errors return if the OpenStack is unreachable or if the resource quotas exceed.
- The ETSI services rely on all other ESC services to operate, otherwise there are connectivity-related errors.

**Notifications Generated:**

There are no notifications generated when the snapshot resource is created.

When the snapshot is created on OpenStack, three notifications are generated for the three operational states namely: STARTING, PROCESSING, and COMPLETED as shown:

```
{
    "vnfInstanceId": "c9cdf5c8-3681-4641-ba7e-df40539815b5",
    "timeStamp": "2022-07-20T15:08:43.089Z",
    "isAutomaticInvocation": false,
    "notificationType": "VnfLcmOperationOccurrenceNotification",
    "operationState": "STARTING",
    "notificationStatus": "START",
    "vnfLcmOpOccId": "ecbbdc92-a38a-4aed-bc7c-acf0df1a5b92",
```

```
    "_links": {
        "vnfInstance": {
            "href":
"https://192.168.10.50:8251/or_vnfm/vnflcm/v2/vnf_instances/c9cdf5c8-3681-4641-ba7e-df40539815b5"

        },
        "vnfLcmOpOcc": {
            "href":
"https://192.168.10.50:8251/or_vnfm/vnflcm/v2/vnf_lcm_op_occs/ecbbdc92-a38a-4aed-bc7c-acf0df1a5b92"

        },
        "subscription": {
            "href":
"https://192.168.10.50:8251/or_vnfm/vnflcm/v2/subscriptions/900c511f-27e7-4819-aa8d-1fae527caa85"

        }
    },
    "subscriptionId": "900c511f-27e7-4819-aa8d-1fae527caa85",
    "operation": "CREATE_SNAPSHOT",
    "id": "640804b1-2564-4020-af72-16b70d6ac83d"
}

{
    "vnfInstanceId": "c9cdf5c8-3681-4641-ba7e-df40539815b5",
    "timeStamp": "2022-07-20T15:08:43.798Z",
    "isAutomaticInvocation": false,
    "notificationType": "VnfLcmOperationOccurrenceNotification",
    "operationState": "PROCESSING",
    "notificationStatus": "START",
    "vnfLcmOpOccId": "ecbbdc92-a38a-4aed-bc7c-acf0df1a5b92",
    "_links": {
        "vnfInstance": {
            "href":
"https://192.168.10.50:8251/or_vnfm/vnflcm/v2/vnf_instances/c9cdf5c8-3681-4641-ba7e-df40539815b5"

        },
        "vnfLcmOpOcc": {
            "href":
"https://192.168.10.50:8251/or_vnfm/vnflcm/v2/vnf_lcm_op_occs/ecbbdc92-a38a-4aed-bc7c-acf0df1a5b92"

        },
        "subscription": {
            "href":
"https://192.168.10.50:8251/or_vnfm/vnflcm/v2/subscriptions/900c511f-27e7-4819-aa8d-1fae527caa85"

        }
    },
    "subscriptionId": "900c511f-27e7-4819-aa8d-1fae527caa85",
    "operation": "CREATE_SNAPSHOT",
    "id": "6907ac6f-41e4-4bb6-9d31-83f9e809b933"
}

{
    "vnfInstanceId": "c9cdf5c8-3681-4641-ba7e-df40539815b5",
    "timeStamp": "2022-07-20T15:09:02.773Z",
    "isAutomaticInvocation": false,
    "notificationType": "VnfLcmOperationOccurrenceNotification",
    "operationState": "COMPLETED",
    "notificationStatus": "RESULT",
    "vnfLcmOpOccId": "ecbbdc92-a38a-4aed-bc7c-acf0df1a5b92",
    "_links": {
        "vnfInstance": {
            "href":
"https://192.168.10.50:8251/or_vnfm/vnflcm/v2/vnf_instances/c9cdf5c8-3681-4641-ba7e-df40539815b5"
```

```
        },
        "vnfLcmOpOcc": {
            "href":
"https://192.168.10.50:8251/or_vnfm/vnflcm/v2/vnf_lcm_op_occs/ecbbdc92-a38a-4aed-bc7c-acf0df1a5b92"

        },
        "subscription": {
            "href":
"https://192.168.10.50:8251/or_vnfm/vnflcm/v2/subscriptions/900c511f-27e7-4819-aa8d-1fae527caa85"

        }
    },
    "subscriptionId": "900c511f-27e7-4819-aa8d-1fae527caa85",
    "operation": "CREATE_SNAPSHOT",
    "id": "de25c769-4264-4fa3-a61f-2aae960c6b60"
}
```


**OpenStack Resources Generated:**

Upon successful completion of the operation and receiving the final notification, the following resources are created in OpenStack:

IMAGE

Create an image for every VM within the VNF. For example, if the VNF contains two VDUs, then two images are created in OpenStack.

The images have the name of the auto-generated VNFC Snapshot, a UUID-type value. For example

```
[admin@host]$ openstack image list
+--------------------------------------+--------------------------------------------------------+--------+
| ID                                   | Name
 | Status |
+--------------------------------------+--------------------------------------------------------+--------+
| 92e144ae-24fc-49a5-8622-bb224f1e55cd | eac61a66-51d2-47dd-b8f4-289f38203eff
 | active |
```

**Note** Note: Find both the image ID and its UUID-like name in the VNF Snapshot query output, explained in the “Query VNF Snapshot” section

**VOLUME SNAPSHOT:**

Create a volume snapshot for every volume within the VNF. For example, if the VNF contains two VDUs within two volumes each, then four volume snapshots are created in OpenStack.

The volume snapshots have the name of the auto-generated VNFC Snapshot which is a UUID type value prepended by “snapshot for “. For example:

```
[admin@host]$ openstack volume snapshot list
+------------------------------+--------------------------------------+-----------+---------+-----+
| ID                                   | Name
 | Description | Status    | Size |
+------------------------------+--------------------------------------+-----------+---------+-----+
| 503c348d-94f1-4351-85ec-686b4a21589c | snapshot for eac61a66-51d2-47dd-b8f4-289f38203eff
 | None        | available |    1 |
```

**Note** Find both the volume snapshot ID and the UUID portion of its name in the VNF Snapshot query output, explained in the “Query VNF Snapshot” section

**Query VNF Snapshot:**

Use these two main queries to return ETSI VNF Snapshot information:

- Query all VNF Snapshots
- Query a specific VNF Snapshot

**API Execution**

The following shows both these operations and the API responses, using Linux curl as a client, executing the APIs on the ESC VM that is, the localhost itself:

- Query all VNF Snapshots - an array is returned

```
[admin@host]$ curl -s --user 'admin*******' -X GET -H 'Content-Type:application/json'
http://localhost:8250/or_vnfm/vnflcm/v2/vnf_snapshots | python -m json.tool
[
    {
        "_links": {
            "self": {
                "href":
"http://localhost:8250/or_vnfm/vnflcm/v2/vnf_snapshots/fc7f055c-a541-4801-9295-299ce806763f"

            },
            "takenFrom": {
                "href":
"http://localhost:8250/or_vnfm/vnflcm/v2/vnf_instances/c9cdf5c8-3681-4641-ba7e-df40539815b5"

            }
        },
        "id": "fc7f055c-a541-4801-9295-299ce806763f",
        "vnfSnapshot": {
            "creationFinishedAt": "2022-07-20T15:09:02.588Z",
            "creationStartedAt": "2022-07-20T15:08:43.966Z",
            "id": "0e61b4f8-b347-4d48-80e1-b7a1d28196ef",
            "vnfInstanceId": "c9cdf5c8-3681-4641-ba7e-df40539815b5",
            "vnfdId": "9fb7e4ee-2db1-4aef-bc62-98a2d35d1fa0"
        }
    }
]
```

- Query a specific VNF Snapshot - a single snapshot is returned

```
[admin@host]$ curl -s --user 'admin:cisco123' -X GET -H 'Content-Type:application/json'
http://localhost:8250/or_vnfm/vnflcm/v2/vnf_snapshots/fc7f055c-a541-4801-9295-299ce806763f
 | python -m json.tool
{
    "_links": {
        "self": {
            "href":
"http://localhost:8250/or_vnfm/vnflcm/v2/vnf_snapshots/fc7f055c-a541-4801-9295-299ce806763f"

        },
        "takenFrom": {
            "href":
```

```
"http://localhost:8250/or_vnfm/vnflcm/v2/vnf_instances/c9cdf5c8-3681-4641-ba7e-df40539815b5"

        }
    },
    "id": "fc7f055c-a541-4801-9295-299ce806763f",  <!-- THE VNF SNAPSHOT ID -->
    "vnfSnapshot": {
        "creationFinishedAt": "2022-07-20T15:09:02.588Z",
        "creationStartedAt": "2022-07-20T15:08:43.966Z",
        "id": "0e61b4f8-b347-4d48-80e1-b7a1d28196ef",
        "vnfInstance": {
            "id": "c9cdf5c8-3681-4641-ba7e-df40539815b5",
            "instantiatedVnfInfo": {

<!-- Data deleted as identical to the output from a VNF Instance query -->

        },
        "vnfInstanceId": "c9cdf5c8-3681-4641-ba7e-df40539815b5",  <!-- THE VNF INSTANCE ID
 -->
        "vnfcSnapshots": [
            {
                "computeSnapshotResource": {
                    "resourceId": "92e144ae-24fc-49a5-8622-bb224f1e55cd"  <!-- THE IMAGE
ID -->
                },
                "creationFinishedAt": "2022-07-20T15:09:02.588Z",
                "creationStartedAt": "2022-07-20T15:08:43.966Z",
                "id": "eac61a66-51d2-47dd-b8f4-289f38203eff",   <!-- THE IMAGE NAME AND
VOLUME SNAPSHOT NAME -->
                "storageSnapshotResources": [
                    {
                        "storageResourceId": "res-cfd9a704-0cae-43e2-9880-0b1ba41f2615",
                        "storageSnapshotResource": {
                            "resourceId": "503c348d-94f1-4351-85ec-686b4a21589c"  <!-- THE
 VOLUME SNAPSHOT ID -->
                        }
                    }
                ],
                "vnfcInstanceId": "res-9f5401e3-0129-4657-8ef7-18da424fd369",  <!-- NEEDED
 IF USING THE SOL002 API -->
                "vnfcResourceInfoId": "res-9f5401e3-0129-4657-8ef7-18da424fd369"
            },
        ],
        "vnfdId": "9fb7e4ee-2db1-4aef-bc62-98a2d35d1fa0"
    }
}
```

**Reverting to VNF Snapshot using SOL002 or SOL003 APIs:**

Users can perform a revert to VNF snapshot lifecycle management operation to return to a previous version of the VNF.

**API Execution:**

The following shows a sample payload to revert to VNF snapshot:

- Revert to VNF snapshot giving snapshot resource id and the VNF Instance ID

```
[admin@host]$ cat revert_snapshot.json
{
    "vnfSnapshotInfoId": "fc7f055c-a541-4801-9295-299ce806763f"
}

[admin@host]$ curl -s --user 'admin:*******' -X POST --data @revert_snapshot.json -H
```

```
'Content-Type:application/json'
http://localhost:8250/or_vnfm/vnflcm/v2/vnf_instances/c9cdf5c8-3681-4641-ba7e-df40539815b5/revert_to_snapshot
```

**Note** The SOL002 API root uses ve_vnfm, not or_vnfm.

The following sample payload, restricts the revert to a single VNFC

**SOL002:**

```
{
    "vnfSnapshotInfoId": "fc7f055c-a541-4801-9295-299ce806763f",
    "vnfcInstanceId": "res-9f5401e3-0129-4657-8ef7-18da424fd369",
    "vnfcSnapshotInfoId": "eac61a66-51d2-47dd-b8f4-289f38203eff"
}
```

**SOL003 using Additional Parameters:**

```
{
    "vnfSnapshotInfoId": "fc7f055c-a541-4801-9295-299ce806763f",
    "additionalParams": {
        "vnfcInstanceId": "res-9f5401e3-0129-4657-8ef7-18da424fd369",
        "vnfcSnapshotInfoId": "eac61a66-51d2-47dd-b8f4-289f38203eff"
    }
}
```

**Notifications Generated:**

When the snapshot is reverted on OpenStack, three notifications are generated for the three operational states namely: STARTING, PROCESSING, and COMPLETED.

**Notes and Limitations:**

- Revert to snapshot does not currently support snapshots of VM without a bootable volume.
- It is not possible to revert a snapshot with OOB volumes.
- OpenStack prevents the deletion of volumes if they have a volume snapshot, ESC attempts to delete the volumes during the revert but these are left on the VIM.

**VNF Snapshot Deletion:**

Deleting a VNF snapshot involves the single URL as shown:

```
[admin@host]$ curl --user 'admin:********' -X DELETE
http://localhost:8250/or_vnfm/vnflcm/v2/vnf_snapshots/fc7f055c-a541-4801-9295-299ce806763f
```

**Note** The VNF Snapshot Deletion is a synchronous operation, that is, the API call does not return until the entire workflow in ESC has finished. Deletion takes some time if the VNF Snapshot has to delete multiple VDUs and volumes.

**Error Conditions**

- A suitable error returns if the VNF Snapshot ID is invalid.
- OpenStack-specific errors return if the OpenStack is unreachable or if the resource quotas exceed.

- The ETSI services rely on all other ESC services to operate, otherwise there are connectivity-related errors.

**Notifications Generated:**

As per the ETSI Specification, no notifications are generated for a VNF Snapshot delete operation due to its synchronous nature.

**Creating a VNF Snapshot using SOL002 APIs:**

Create a VNF Snapshot using the SOL002 API, which allows the specification of an individual VNFC ID.

Individual VDUs (VNFCs) within a VNF have snapshots created for them as opposed to taking a VNF Snapshot of the entire VNF.

```
[admin@host]$ cat create_snapshot.json
{
    "vnfSnapshotInfoId": "fc7f055c-a541-4801-9295-299ce806763f",
    "additionalParams": {
        "vnfcInstanceId": "res-9f5401e3-0129-4657-8ef7-18da424fd369"
    }
}
[admin@host]$ curl --user 'admin:********' -X POST --data @create_snapshot.json -H
'Content-Type:application/json'
http://localhost:8250/ve_vnfm/vnflcm/v2/vnf_instances/c9cdf5c8-3681-4641-ba7e-df40539815b5/create_snapshot
```

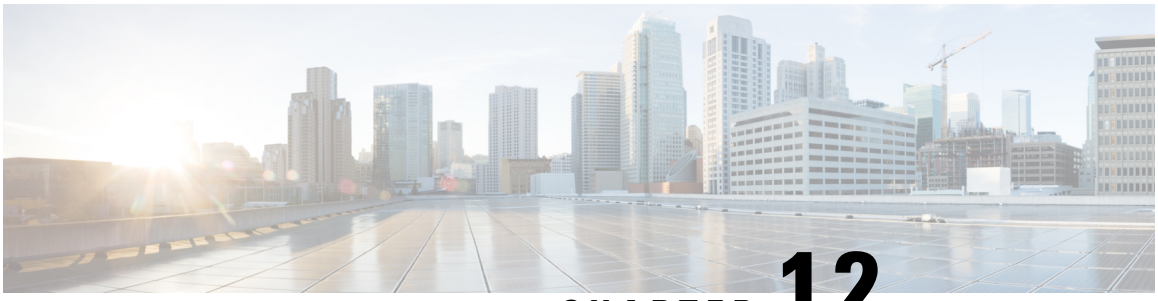
**Note** The SOL002 API root uses *ve_vnfm*, not *or_vnfm*.

**Error Conditions:**

- An error returns if the VNF Snapshot ID, the VNF Instance ID, or the VNFC Instance ID are invalid.
- OpenStack-specific errors return if the OpenStack is unreachable or if the resource quotas exceed.
- The ETSI services rely on all other ESC services to operate, otherwise there are connectivity-related errors.

**Notifications generated:**

The identical notifications for the SOL003 VNF Snapshot Create are generated.



CHAPTER 12

# Error Handling Procedures



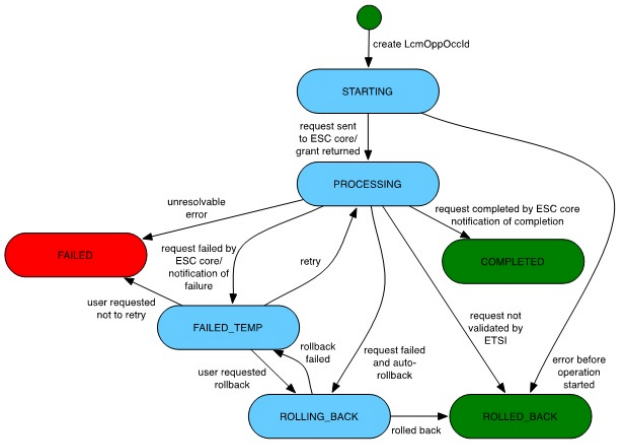
# VNF Lifecycle Management Error Handling Procedures

ETSI invokes the following error handling procedures for all its ETSI VNF lifecycle management (LCM) operations:

- Retry
- Rollback
- Fail
- Cancel

The image below represents the transitional states of the VNF lifecycle management operational occurrence.

***Figure 2: VNF Lifecycle Management Transitional States***



**Note** The *vnfLcmOpOccId* is encoded into the URI, which is the primary key to retrieve the request details.

The retry, rollback and fail requests are rejected if the LCM operation is in any other state other than the FAILED_TEMP state. This error returns HTTP code 409.

The retry, rollback, fail and cancel requests are not supported for the particular VNF LCM operation for the particular VNF. This error returns HTTP code 404.

An error occurs if the *vnfLcmOpOccId* does not exist in the ETSI database. This error returns HTTP code 404.

### Retry

A retry request is applicable if there is a possibility of the LCM operation to succeed. The operation should be (pre-condition) in the FAILED_TEMP state for a retry request. You can send several retry requests, as long as the operation is in the FAILED_TEMP state.

| | |
|---|---|
| Precondition | FAILED_TEMP state |
| Request | `POST {api_root}/vnf_lcm_op_occs/{vnfLcmOpOccId}/retry()` |
| Postcondition | PROCESSING state |

Upon successful retry, ESC sends a START or PROCESSING notification. If the retry request fails, then ESC sends a notification to the NFVO with the details.

### Rollback

A rollback request is made if it is not possible for the operation to succeed even after a retry request.

Set the *rollback_required* flag to true. If this is not set to true, then rollback is not performed.

| | |
|---|---|
| Precondition | FAILED_TEMP state |
| Request | `POST {api_root}/vnf_lcm_op_occs/{vnfLcmOpOccId}/rollback()` |
| Postcondition | ROLLED_BACK |

Upon successful rollback, the LCM operation is rolled back. If the rollback request fails, then the LCM operation is back to the failed_temp sate.

### Fail

When an LCM operation does not require a retry request, or a clean up, a fail request allows you to free up resources for a subsequent request.

If the *rollback_required* flag is set to true, a fail request cannot be made.

| | |
|---|---|
| Precondition | FAILED_TEMP state |
| Request | `POST {api_root}/vnf_lcm_op_occs/{vnfLcmOpOccId}/fail()` |
| Postcondition | FAILED state |

Upon successful execution of this request, the LCM operation is in FAILED state.

### Cancel

A cancel request is possible if the operation is in STARTING state.

**Note** A cancel request is currently possible in the STARTING or PROCESSING state for Instantiate, but only STARTING for all other LCM operations.

| | |
|---|---|
| Precondition | STARTING state |
| Request | `POST {api_root}/vnf_lcm_op_occs/{vnfLcmOpOccId}/cancel(CancelMode)` |
| Postcondition | ROLLED_BACK |

The cancel request is Forceful.

**Note** ETSI supports canceling an LCM operation in starting state only. The cancel request for LCM operations in processing or rolling back states are currently not supported.

Example JSON payload (CancelMode):

```
{
  "cancelMode": "FORCEFUL",
  "action": "cancel"
}
```

Set the *IsCancelPending* attribute of the *VnfLcmOpOcc* to true. This will stop the processing request, and move the LCM operation to ROLLED_BACK state.

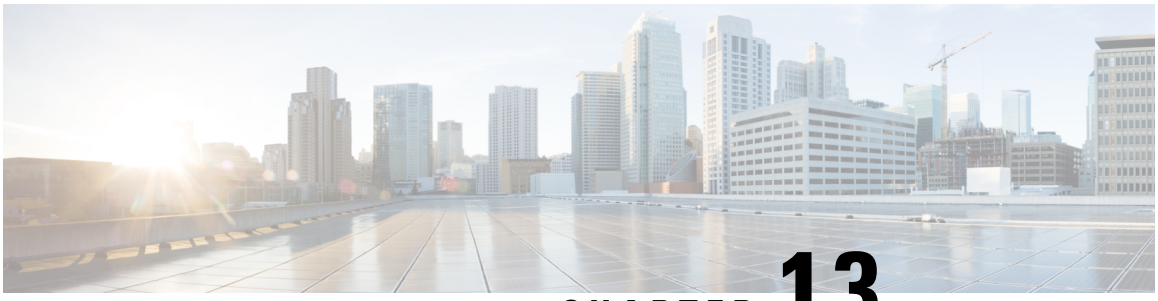
### Error Handling Procedures for ETSI VNF Lifecycle Operations

If the LCM operation for a VNF instance fails, the operation moves to the FAILED_TEMP state according to the state machine. To complete the intended operation, you must either run the retry or rollback request.

- If creating a VNF identifier fails, then no further action is required. The rollback request is not supported.
- If instantiating the VNF fails, then ESC terminates the request, and sends a new instantiation request.
- If operating the VNF fails, then no further action is required.
- If terminating the VNF fails, you must retry the operation, as rollback is not supported.
- If deleting the VNF operation fails, then no further action is required. Th erollback request is not supported.

**Note** The error handling requests do not impact the operating VNF lifecycle operation.

For information on VNF lifecycle operations, see VNF Lifecycle Operations, on page 26.



CHAPTER 13

# Alarms and Notifications for ETSI LCM Operations



## ETSI Alarms

ESC provides alarms and notifications to the NFVO. The NFVO has to subscribe to these alarms and notifications and send requests to ESC.

The NFVO can receive information about the alarms in the following ways:

### Query All Alarms

The NFVO can get a list of all the alarms from the alarms resource.

Method Type:

```
GET
```

VNFM Endpoint:

```
/vnffm/v1/alarms
```

HTTP Request Header:

```
Accept:application/json
```

For example, to query all alarms with the event type as ENVIRONMENTAL_ALARM

Method Type:

```
GET
```

VNFM Endpoint:

```
http://localhost:8250/vnffm/v1/alarms?eventType="ENVIRONMENTAL_ALARM"
```

HTTP Request Headers:

```
Accept:application/json
```

While querying for multiple alarms, the NFVO can use the URI query parameters to filter the results. The following attribute names are supported for the URI query of the alarms:

- id
- managedObjectId
- rootCauseFaultyResource.faultyResourceType
- eventType
- perceivedSeverity
- probableCause

**Note** The URI query parameters are for querying multiple alarms only.

### Query an Individual Alarm

The NFVO can query a particular alarm from the *alarmId* resource.

Method Type:

```
GET
```

VNFM Endpoint

```
/vnffm/v1/alarms/{alarmId}
```

HTTP Request Header:

```
Accept:application/json
```

### Modify an Individual Alarm

To modify an alarm, the NFVO must send a PATCH request to the *AlarmModifications* resource.

Method Type:

```
PATCH
```

VNFM Endpoint:

HTTP Request Header:

```
Content-Type: application/merge-patch+json
If-Match: ETag value
```

**Note** **If-Match:** is optional. If specified, its value is validated against the ETag value stored against the VNF (and returned from a single VNF query).

The supported attribute is ackState, and the supported attribute values are ACKNOWLEDGED and UNACKNOWLEDGED. All other modification payloads are rejected.

### VNF Failure and Load Alarms

The following alarms are created for ETSI VNF failure and load notifications.

- Failure Alarm—ESC generates the failure alarms when one of the compute resources within the VNF becomes unreachable based upon the VM_ALIVE KPI configuration of the VFND. For more information, see ETSI Failure and Load Notifications for VNFs.

  Example:

  Method Type

  ```
  POST
  ```

  VNFM Endpoint

  ```
  /vnffm/v1/extension/alarms
  ```

  HTTP Request Header

  ```
  Content-Type:application/json
  ```

  Request Payload:

  ```
  {
    "externalAlarmId" : "26bf1e3d-cefa-4f59-88ea-210a29358a5c", #generated value
    "alarmSource" : "MONA", #hard-coded
    "managedObjectId" : "08733ef2-319b-46ce-9d8d-95730306bd1a", #external_deployment_id
   "rootCauseFaultyResource" : "chrimann-dep_g1_0_212da327-0573-421b-ae37-057f6b1a6aef",
  #vm_name
    "alarmRaisedTime" : "$timestamp", #generated value
    "ackState" : "UNACKNOWLEDGED", #hard-coded
    "perceivedSeverity" : "CRITICAL", #hard-coded
    "eventTime" : "2018-05-08T00:59:32.571+00:00", #do we have the eventTime?
    "eventType" : "EQUIPMENT_ALARM", #hard-coded
    "faultType" : "COMPUTE", #hard-coded
    "probableCause" : "VM_MANUAL_RECOVERY_NEEDED", #event_name
    "isRootCause" : "TRUE", #hard-coded
    "links" : {
      "objectInstance" :
  "{http_scheme}://{api_root}/vnflcm/v2/vnf_instances/08733ef2-319b-46ce-9d8d-95730306bd1a"

    }
  }
  ```

- Load Alarm—ESC generates the load alarms when one of the compute resources within the VNF becomes over or under loaded based upon the related KPI configurations of the VFND. ESC creates these alarms after receiving notifications from the NFVO. For more information, see ETSI Failure and Load Notifications for VNFs.

  Example:

  Method Type

  ```
  POST
  ```

  VNFM Endpoint

  ```
  /vnffm/v1/extension/alarms
  ```

  HTTP Request Header

  ```
  Content-Type:application/json
  ```

  Request Payload

### Alarm Extensions

ETSI provides an extension for the alarms to interact with the third party tools. You must send a POST request to create the alarms.

Method Type

```
POST
```

VNFM Endpoint

```
/vnffm/v1/extension/alarms
```

HTTP Request Header

```
Content-Type:application/json
```

Request Payload

```
[admin@davwebst-esc-4-2-0-49-keep ETSI]$ cat CreateAlarm.json
{
    "id": "alm87032",
    "externalAlarmId": "ext-id-xx11214",
    "managedObjectId": "930fb087-c1b9-4660-bec8-2a8d97dc1df5",
    "rootCauseFaultyResource": {
        "id": "fres7629",
        "faultyResource": {
            "resourceId": "res7727"
        },
        "faultyResourceType": "NETWORK"
    },
    "alarmRaisedTime": "2018-05-30T13:55:15.645000+00",
    "ackState": "UNACKNOWLEDGED",
    "perceivedSeverity": "MAJOR",
    "eventTime": "2018-05-30T13:55:15.645000+00",
    "eventType": "ENVIRONMENTAL_ALARM",
    "probableCause": "Server room overheating",
    "isRootCause": "false",
    "vnfInstanceIds": [
        "res-a3023a03-fc73-430a-a983-5e9439011e45"

}
```

# Subscribing to Notifications

The NFVO can subscribe to the ETSI notifications related to fault management from ESC.

### Create a Subscription

The NFVO sends a POST request to subscribe to the notifications.

Method Type:

```
POST
```

VNFM Endpoint:

```
/vnffm/v1
```

Response Payload:

```
{
  "filter" : {
```

```
    "notificationTypes" : [
      "AlarmNotification",
      "AlarmClearedNotification",
      "AlarmListRebuiltNotification"
    ],
    "perceivedSeverities" : [
      "CRITICAL",
      "MAJOR"
    ]
  },
  "callbackUri" : "https://nfvo.endpoint.listener",
  "authentication" : {
    "authType" : "BASIC",
    "paramsBasic" : {
      "userName" : "admin",
      "password" : "pass123"
    }
  }
}
```

This creates a new subscription resource and a new identifier. The callbackUri is the only mandatory parameter. The others are all optional. You can verify if the callbackuri is valid and reachable by sending a GET request.

### Query all Subscriptions

The NFVO can query information about its subscriptions by sending a GET request to the *subscriptions* resource.

Method Type:

```
GET
```

VNFM Endpoint:

```
/vnffm/v1/subscriptions
```

HTTP Request Header:

```
Accept:application/json
```

For example, to query all alert subscriptions, when the callbackUri is

```
http://10.10.1.44:9202/alerts/subscriptions/callback
```

```
GET
```

VNFM Endpoint

```
http://localhost:8250/vnffm/v1/subscriptions?callbackUri="http://10.10.1.44:9202/alerts/subscriptions/callback"
```

HTTP Request Header

```
Accept:application/json
```

The NFVO can use the URI query parameters to filter the results. The following attribute names are supported for the URI query of the subscriptions:

- id
- filter
- callbackUri

**Note** The URI query parameters are for querying multiple subscriptions only.

### Query an Individual Subscription

You must know the subscription ID to query an individual subscription.

Method Type:

```
GET
```

VNFM Endpoint:

```
/vnffm/v1/subscriptions/{subscriptionId}
```

HTTP Request Header:

```
Accept:application/json
```

### Delete a Subscription

You can delete a subscription if the NFVO does not need it. Send a delete request to the individual subscription.

Method Type:

```
DELETE
```

VNFM Endpoint:

```
/vnffm/v1/subscriptions/{subscriptionId}
```

HTTP Request Header:

```
http://localhost:8250/vnffm/v1/subscriptions/682791f8-34ad-487e-811a-553036bf49b2
```

# ETSI Failure and Load Notifications for VNFs

ESC generates notifications for the following:

- **VM Failure**

  The NFVO receives failure notifications from ESC, when the VMs within the deployed VNFs fail. After receiving the notifications, alarms are generated. For more information on alarms, see ETSI Alarms, on page 105.

  The NFVO must subscribe to the ESC for notifications.

  Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_event xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <deployment_name>sample-dep</deployment_name>
  <event_name>MY_VM_UNDERLOADED</event_name>
  <event_type>VM_UNDERLOADED</event_type>
  <external_deployment_id>e911eecf-5f3f-456c-9c80-d99aca2416da</external_deployment_id>

  <external_tenant_id>etsi_tenant</external_tenant_id>
  <internal_deployment_id>99f7629f-98d3-40f5-ad68-7addcfe07006</internal_deployment_id>

  <internal_tenant_id>etsi_tenant</internal_tenant_id>
  <vm_source>
```

```
<generated_vm_name>sample-dep_vm1_0_fbc3da46-e0c6-40dc-91c8-70b1a88857de</generated_vm_name>

    <interfaces>
      <addresses>
        <address>
          <address_id>0</address_id>
          <gateway>172.16.0.1</gateway>
          <ip_address>172.16.0.0</ip_address>
          <dhcp_enabled>true</dhcp_enabled>
          <prefix>20</prefix>
          <subnet>365a0884-fdb3-424c-afe9-2deb3b39baae</subnet>
        </address>
      </addresses>
      <network_uuid>c7fafeca-aa53-4349-9b60-1f4b92605420</network_uuid>
      <mac_address>fa:16:3e:38:1d:6c</mac_address>
      <nic_id>0</nic_id>
      <port_forwarding/>
      <port_uuid>0aeb9585-5190-4f3b-b1aa-495e09c56b7d</port_uuid>
      <security_groups/>
      <subnet_uuid>none</subnet_uuid>
      <type>virtual</type>

<vim_interface_name>sample-dep_vm1_0_fbc3da46-e0c6-40dc-91c8-70b1a88857de</vim_interface_name>

    </interfaces>
    <vim_id>default_openstack_vim</vim_id>
    <vim_project>admin</vim_project>
    <vim_project_id>c12f013306d849e5b1bbf257c54d5891</vim_project_id>
    <host_uuid>6b8cf361c5ff08a5a886e26f591b8087dadcf2d2b34fb3b5d2772a8d</host_uuid>
    <host_name>my-server</host_name>
    <vm_uuid>9fea3fe7-9417-4734-b962-b24340941ef3</vm_uuid>
    <vm_group_name>vm1</vm_group_name>
    <vm_name>sample-dep_vm1_0_fbc3da46-e0c6-40dc-91c8-70b1a88857de</vm_name>
  </vm_source>
</esc_event>
```

- **VM Overload and Underload**

  Similarly, the NFVO receives an overload or underload notification for a VM.

  If scaling is not enabled automatically, ESC generates a notification depending on the state of the VM.

  Examples:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_event xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <deployment_name>sample-dep</deployment_name>
  <event_name>MY_VM_UNDERLOADED</event_name>
  <event_type>VM_UNDERLOADED</event_type>
  <external_deployment_id>e911eecf-5f3f-456c-9c80-d99aca2416da</external_deployment_id>

  <external_tenant_id>etsi_tenant</external_tenant_id>
  <internal_deployment_id>99f7629f-98d3-40f5-ad68-7addcfe07006</internal_deployment_id>

  <internal_tenant_id>etsi_tenant</internal_tenant_id>
  <vm_source>

<generated_vm_name>sample-dep_vm1_0_fbc3da46-e0c6-40dc-91c8-70b1a88857de</generated_vm_name>

    <interfaces>
      <addresses>
        <address>
          <address_id>0</address_id>
          <gateway>172.16.0.1</gateway>
```

```
            <ip_address>172.16.0.0</ip_address>
            <dhcp_enabled>true</dhcp_enabled>
            <prefix>20</prefix>
            <subnet>365a0884-fdb3-424c-afe9-2deb3b39baae</subnet>
          </address>
        </addresses>
        <network_uuid>c7fafeca-aa53-4349-9b60-1f4b92605420</network_uuid>
        <mac_address>fa:16:3e:38:1d:6c</mac_address>
        <nic_id>0</nic_id>
        <port_forwarding/>
        <port_uuid>0aeb9585-5190-4f3b-b1aa-495e09c56b7d</port_uuid>
        <security_groups/>
        <subnet_uuid>none</subnet_uuid>
        <type>virtual</type>

<vim_interface_name>sample-dep_vm1_0_fbc3da46-e0c6-40dc-91c8-70b1a88857de</vim_interface_name>

    </interfaces>
    <vim_id>default_openstack_vim</vim_id>
    <vim_project>admin</vim_project>
    <vim_project_id>c12f013306d849e5b1bbf257c54d5891</vim_project_id>
    <host_uuid>6b8cf361c5ff08a5a886e26f591b8087dadcf2d2b34fb3b5d2772a8d</host_uuid>
    <host_name>my-server</host_name>
    <vm_uuid>9fea3fe7-9417-4734-b962-b24340941ef3</vm_uuid>
    <vm_group_name>vm1</vm_group_name>
    <vm_name>sample-dep_vm1_0_fbc3da46-e0c6-40dc-91c8-70b1a88857de</vm_name>
  </vm_source>
</esc_event>
```

VM underload example:

```
<?xml version="1.0" encoding="UTF-8"?>
<esc_event xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <deployment_name>sample-dep</deployment_name>
  <event_name>MY_VM_OVERLOADED</event_name>
  <event_type>VM_OVERLOADED</event_type>
  <external_deployment_id>e911eecf-5f3f-456c-9c80-d99aca2416da</external_deployment_id>

  <external_tenant_id>etsi_tenant</external_tenant_id>
  <internal_deployment_id>99f7629f-98d3-40f5-ad68-7addcfe07006</internal_deployment_id>

  <internal_tenant_id>etsi_tenant</internal_tenant_id>
  <vm_source>

<generated_vm_name>sample-dep_vm1_0_fbc3da46-e0c6-40dc-91c8-70b1a88857de</generated_vm_name>

    <interfaces>
      <addresses>
        <address>
          <address_id>0</address_id>
          <gateway>172.16.0.1</gateway>
          <ip_address>172.16.0.0</ip_address>
          <dhcp_enabled>true</dhcp_enabled>
          <prefix>20</prefix>
          <subnet>365a0884-fdb3-424c-afe9-2deb3b39baae</subnet>
        </address>
      </addresses>
      <network_uuid>c7fafeca-aa53-4349-9b60-1f4b92605420</network_uuid>
      <mac_address>fa:16:3e:38:1d:6c</mac_address>
      <nic_id>0</nic_id>
      <port_forwarding/>
      <port_uuid>0aeb9585-5190-4f3b-b1aa-495e09c56b7d</port_uuid>
      <security_groups/>
      <subnet_uuid>none</subnet_uuid>
      <type>virtual</type>
```

```
<vim_interface_name>sample-dep_vm1_0_fbc3da46-e0c6-40dc-91c8-70b1a88857de</vim_interface_name>

    </interfaces>
    <vim_id>default_openstack_vim</vim_id>
    <vim_project>admin</vim_project>
    <vim_project_id>c12f013306d849e5b1bbf257c54d5891</vim_project_id>
    <host_uuid>6b8cf361c5ff08a5a886e26f591b8087dadcf2d2b34fb3b5d2772a8d</host_uuid>
    <host_name>my-server</host_name>
    <vm_uuid>9fea3fe7-9417-4734-b962-b24340941ef3</vm_uuid>
    <vm_group_name>vm1</vm_group_name>
    <vm_name>sample-dep_vm1_0_fbc3da46-e0c6-40dc-91c8-70b1a88857de</vm_name>
  </vm_source>
</esc_event>
```

**Note** ETSI only generates an alarm for a VNFC that exists in instantiatedVnfInfo.vnfcResourceInfo when the notification from ESC is received.

# Auto-Scaling VNFs Using KPI Instructions

ESC can auto-scale VMs using the KPI instructions. The scaling workflow begins when the VNF instance is in the instantiated state.The NFVO enables and disables the auto-scaling while modifying *isAutoscaleEnabled* configurable property of the VNF.

Following are the events that trigger an ETSI-compliant auto-scale, which requires an instigation of a *ScaleVnfToLevelRequest:* functionality.

- **Overload and Underload**

  If the state of a VM changes and it is under or overloaded, ESC gets a notification to determine if the scaling is automatically enabled. If it is not, ESC generates a notification towards the ETSI-VNFM component to check the VNF's state.

  The following example shows underloaded notification from ESC:

```
Headers:
  esc-status-code = 200
  esc-status-message = VM [sample-dep_vm1_0_fbc3da46-e0c6-40dc-91c8-70b1a88857de]
underloaded.
Body:
<?xml version="1.0" encoding="UTF-8"?>
<esc_event xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <deployment_name>sample-dep</deployment_name>
  <event_name>MY_VM_UNDERLOADED</event_name>
  <event_type>VM_UNDERLOADED</event_type>
  <external_deployment_id>e911eecf-5f3f-456c-9c80-d99aca2416da</external_deployment_id>

  <external_tenant_id>etsi_tenant</external_tenant_id>
  <internal_deployment_id>99f7629f-98d3-40f5-ad68-7addcfe07006</internal_deployment_id>

  <internal_tenant_id>etsi_tenant</internal_tenant_id>
  <vm_source>

<generated_vm_name>sample-dep_vm1_0_fbc3da46-e0c6-40dc-91c8-70b1a88857de</generated_vm_name>

    <interfaces>
      <addresses>
        <address>
```

```
            <address_id>0</address_id>
            <gateway>172.24.0.1</gateway>
            <ip_address>172.24.0.37</ip_address>
            <dhcp_enabled>true</dhcp_enabled>
            <prefix>20</prefix>
            <subnet>365a0884-fdb3-424c-afe9-2deb3b39baae</subnet>
          </address>
        </addresses>
        <network_uuid>c7fafeca-aa53-4349-9b60-1f4b92605420</network_uuid>
        <mac_address>fa:16:3e:38:1d:6c</mac_address>
        <nic_id>0</nic_id>
        <port_forwarding/>
        <port_uuid>0aeb9585-5190-4f3b-b1aa-495e09c56b7d</port_uuid>
        <security_groups/>
        <subnet_uuid>none</subnet_uuid>
        <type>virtual</type>

<vim_interface_name>sample-dep_vm1_0_fbc3da46-e0c6-40dc-91c8-70b1a88857de</vim_interface_name>

    </interfaces>
    <vim_id>default_openstack_vim</vim_id>
    <vim_project>admin</vim_project>
    <vim_project_id>c12f013306d849e5b1bbf257c54d5891</vim_project_id>
    <host_uuid>6b8cf361c5ff08a5a886e26f591b8087dadcf2d2b34fb3b5d2772a8d</host_uuid>
    <host_name>my-server-65</host_name>
    <vm_uuid>9fea3fe7-9417-4734-b962-b24340941ef3</vm_uuid>
    <vm_group_name>vm1</vm_group_name>
    <vm_name>sample-dep_vm1_0_fbc3da46-e0c6-40dc-91c8-70b1a88857de</vm_name>
  </vm_source>
</esc_event>
```

The following example shows overloaded notification from ESC:

```
Headers:
  esc-status-code = 200
  esc-status-message = VM [sample-dep_vm1_0_fbc3da46-e0c6-40dc-91c8-70b1a88857de]
overloaded.
Body:
<?xml version="1.0" encoding="UTF-8"?>
<esc_event xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <deployment_name>sample-dep</deployment_name>
  <event_name>MY_VM_OVERLOADED</event_name>
  <event_type>VM_OVERLOADED</event_type>
  <external_deployment_id>e911eecf-5f3f-456c-9c80-d99aca2416da</external_deployment_id>

  <external_tenant_id>etsi_tenant</external_tenant_id>
  <internal_deployment_id>99f7629f-98d3-40f5-ad68-7addcfe07006</internal_deployment_id>

  <internal_tenant_id>etsi_tenant</internal_tenant_id>
  <vm_source>

<generated_vm_name>sample-dep_vm1_0_fbc3da46-e0c6-40dc-91c8-70b1a88857de</generated_vm_name>

    <interfaces>
      <addresses>
        <address>
          <address_id>0</address_id>
          <gateway>172.24.0.1</gateway>
          <ip_address>172.24.0.37</ip_address>
          <dhcp_enabled>true</dhcp_enabled>
          <prefix>20</prefix>
          <subnet>365a0884-fdb3-424c-afe9-2deb3b39baae</subnet>
        </address>
      </addresses>
      <network_uuid>c7fafeca-aa53-4349-9b60-1f4b92605420</network_uuid>
```

```
      <mac_address>fa:16:3e:38:1d:6c</mac_address>
      <nic_id>0</nic_id>
      <port_forwarding/>
      <port_uuid>0aeb9585-5190-4f3b-b1aa-495e09c56b7d</port_uuid>
      <security_groups/>
      <subnet_uuid>none</subnet_uuid>
      <type>virtual</type>

<vim_interface_name>sample-dep_vm1_0_fbc3da46-e0c6-40dc-91c8-70b1a88857de</vim_interface_name>

    </interfaces>
    <vim_id>default_openstack_vim</vim_id>
    <vim_project>admin</vim_project>
    <vim_project_id>c12f013306d849e5b1bbf257c54d5891</vim_project_id>
    <host_uuid>6b8cf361c5ff08a5a886e26f591b8087dadcf2d2b34fb3b5d2772a8d</host_uuid>
    <host_name>my-server-65</host_name>
    <vm_uuid>9fea3fe7-9417-4734-b962-b24340941ef3</vm_uuid>
    <vm_group_name>vm1</vm_group_name>
    <vm_name>sample-dep_vm1_0_fbc3da46-e0c6-40dc-91c8-70b1a88857de</vm_name>
  </vm_source>
</esc_event>
```

- **VNFD**

  The VNFD notification contains the instructions for the scale action required for *isAutoscaleEnabled* configurable property of the VNF operation flow.

  If the scaling is not enabled automatically, you can instigate the manual LCM operations using the KPI instructions. It is instigated by processing the ESC notification stream. You must validate the notification once you receive the KPI events.

  You must take the following actions:

  - Find the matching VNF instance
  - Validate that the appropriate configuration property is set to enable the automated operation

  If the validation passes then you can request to instigate the operation flow to generate the appropriate operation occurrence and associated notifications. For scaling, any specified KPI data determines the scaling parameters. The properties file includes the following new attributes:

```
external.scaling.decision = 1
  #external.scaling.window = 120
  external.healing.decision = 1
  #external.healing.window = 120
```

- **VnfInstance resource**

  The VNFD determines the scale level using the current scaleStatus. The processing of the request determines the number of VMs to request from ESCManager. The request only supplies a relative number of increments (SCALE_IN or SCALE_OUT).

  You can call the *ScaleVnfToLevel* endpoint with the following payload, using *vnfInstanceId* from the vnfInstance resource of the VNF to be scaled.

  Ensure that the VnfLcmOpOcc.isAutomaticInvocation is set to true.

  The following eexample shows JSON payload:

```
{
  /* "instantiationLevelId":"id111", */
  "scaleInfo": [
    { "aspectId":"processing", "scaleLevel":"3" },
```

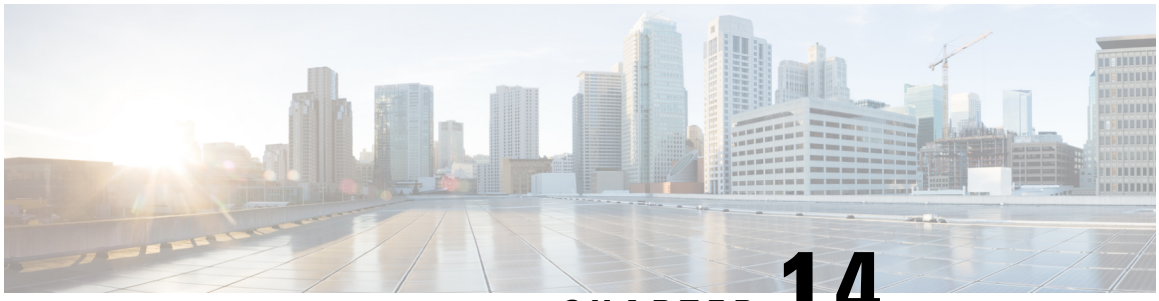
```
      { "aspectId":"database",   "scaleLevel":"2" }
    ]
    "additionalParams": {
      "password": "pass1234",
      "username": "admin"
    },
    "action": "scale_to_level"
  }
```

# Healing VNFs Using KPI Instructions

ESC can auto-heal VMs using the KPI instructions. The NFVO enables and disables the auto-healing while modifying *isAutohealEnabled* configurable property of the VNF.

The *isAutohealEnabled* property permits to enable (TRUE)/disable (FALSE) the auto-healing functionality.

-



CHAPTER 14

# Administering ESC



# ETSI Performance Reports

ESC allows you to collect the performance information of the VNFs such as metrics and notifications using the performance management job functionality. You must first create a performance management (PM) job. After creating the PM job, you can perform the following tasks:

- Query, delete, or notify performance management jobs
- Read an individual report, or obtain the performance reports
- Configure the threshold of the performance management jobs
- Query, delete or notify the threshold of the performance management jobs
- Create or update subscriptions belonging to a Performance Management job or Threshold

# Performance Management Jobs

This section describes the performance management jobs.

### Create Performance Management Job

You must create a performance management job to further query and run reports.

As a part of the job creation, subscription details should be provided to receive any notification.

The NFVO is notified using the `PerformanceInformationAvailableNotification` notification.

Method Type:

POST

VNFM Endpoint:

```
{api_root}/vnfpm/v2/pm_jobs (Data structure=CreatePmJobRequest)
```

Request Payload:

```
{
  "objectInstanceIds": ["9d20a459-b3ff-4d1c-9b63-0dae7444b645"],
  "subObjectInstanceIds": ["9d20a459-b3ff-4d1c-9b63-0dae7444b645"],
  "objectType": "XYZ",
  "callbackUri": "http://localhost:45247/notification",
  "authentication": {
    "authType": ["BASIC"],
     "paramsBasic": {
        "userName": "admin",
        "password": "P@55w0rd!"
      }
  },
  "criteria": {
    "collectionPeriod": 60,
    "reportingPeriod": 3600,
    "reportingBoundary": "2020-08-01T00:00:00.000Z",
    "performanceMetric": [
      "Gold",
      "Silver"
    ],
    "performanceMetricGroup": [
      "VIP",
      "Europe"
    ]
  }
}
```

Response Payload:

```
{
    "id": "b375b81c-3236-4b1c-9c47-61455bf5bc74",
    "objectType": "XYZ",
    "callbackUri": "http://localhost:45248/notification",
    "objectInstanceIds": [
        "9d20a459-b3ff-4d1c-9b63-0dae7444b645"
    ],
    "subObjectInstanceIds": [
        "07775e8b-1279-4338-a643-be283d36fa98"
    ],
    "criteria": {
        "collectionPeriod": 60,
        "reportingPeriod": 3600,
        "performanceMetric": [
            "Gold",
            "Silver"
        ],
        "performanceMetricGroup": [
            "VIP",
            "Europe"
        ],
        "reportingBoundary": "2020-08-01T00:00:00.000Z"
    },
    "reports": [
        {
            "href":
"http://localhost:8250/or_vnfm/vnfpm/v2/pm_jobs/b375b81c-3236-4b1c-9c47-61455bf5bc74/reports/1c787c0d-69a5-4ade-b5ca-43f80e17bd58",

            "readyTime": "2022-02-28T07:29:45.609Z"
        }
    ],
    "_links": {
```

```
        "self": {
            "href":
"http://localhost:8250/or_vnfm/vnfpm/v2/pm_jobs/b375b81c-3236-4b1c-9c47-61455bf5bc74"
        },
        "objects": [
            {
                "href":
"http://localhost:8250/or_vnfm/vnflcm/v2/vnf_instances/9d20a459-b3ff-4d1c-9b63-0dae7444b645"

            }
        ]
    }
}
```

**Note** **Update the same response for *Query and Individual Performance Management Job* and *Query All Performance Management jobs.***

### Query an Individual Performance Management Job

The NFVO queries for the individual performance management job.

Method Type:

GET

VNFM Endpoint:

```
{api_root}/vnfpm/v2/pm_jobs/{pmJobId}
```

Request Payload:

NA.

Response Payload:

```
{
  "id": "13963644-11b0-4302-a13b-26ca3d9eb8f8",
  "objectInstanceIds": [
    "cc6a34e5-0463-459a-b367-493ba997775f "
  ],
  "criteria": {
    "performanceMetric": [
      "default"
    ],
    "performanceMetricGroup": [
      "default"
    ],
    "collectionPeriod": 3600,
    "reportingPeriod": 14400,
    "reports": [
      {
        "href": "uri_where_report_can_be_obtained",
        "readyTime": "2018-08-20T06:17:35.081+0000",
        "expiryTime": "2018-10-20T06:17:35.081+0000",
        "fileSize": "5000"
      }
    ]
  },
  "_links": {
    "self": {
```

```
      "href": "http://host:port/vnfpm/v2/pm_jobs/13963644-11b0-4302-a13b-26ca3d9eb8f8"
    },
    "objects": [
      {
        "href":
"http://host:port/vnflcm/v2/vnf_instances/cc6a34e5-0463-459a-b367-493ba997775f"
      }
    ]
  }
}
```

**Note** A reports section is added to the response payload (as shown above) only if a report is available.

All the attribute names and the data types referenced from the attribute names in the response payload are supported in the attribute-based filtering.

### Query All Performance Management Jobs

The NFVO gets the list of all the performance management jobs.

Method Type:

GET

VNFM Endpoint:

```
{api_root}/vnfpm/v2/pm_jobs
```

Request Payload:

NA.

Response Payload:

```
{
  "id": "13963644-11b0-4302-a13b-26ca3d9eb8f8",
  "objectInstanceIds": [
    "cc6a34e5-0463-459a-b367-493ba997775f "
  ],
  "criteria": {
    "performanceMetric": [
      "default"
    ],
    "performanceMetricGroup": [
      "default"
    ],
    "collectionPeriod": 3600,
    "reportingPeriod": 14400,
    "reports": [
      {
        "href": "uri_where_report_can_be_obtained",
        "readyTime": "2018-08-20T06:17:35.081+0000",
        "expiryTime": "2018-10-20T06:17:35.081+0000",
        "fileSize": "5000"
      }
    ]
  },
  "_links": {
    "self": {
      "href": "http://host:port/vnfpm/v2/pm_jobs/13963644-11b0-4302-a13b-26ca3d9eb8f8"
    },
```

```
    "objects": [
      {
        "href":
"http://host:port/vnflcm/v2/vnf_instances/cc6a34e5-0463-459a-b367-493ba997775f"
      }
    ]
  }
}
```

**Note** A reports section is added to the response payload (as shown above) only if a report is available.

All the attribute names in the response payload and data types referenced from the attribute names are supported in the attribute-based filtering.

### Update a Performance Management Job

The NFVO updates the callbackUri and associated authentication of the individual performance management job.

Method Type:

PATCH

VNFM Endpoint:

```
http://localhost:8250/or_vnfm/vnfpm/v2/pm_jobs/{pmJobId}
```

Request Payload:

```
{
  "callbackUri": "http://localhost:45248/notification",
  "authentication": {
    "authType": ["BASIC"],
     "paramsBasic": {
        "userName": "admin",
        "password": "P@55w0rd!"
      }
  }
 }
```

Response Payload:

```
{
    "callbackUri": "http://localhost:45248/notification"
}
```

### Delete a Performance Management Job

The NFVO sends a delete request to the existing performance management job.

```
DELETE {api_root}/vnfpm/v2/pm_jobs/{pmJobId}
```

# Configuring Threshold for Performance Management Job

This seciton describes how to set the threshold for the performance management jobs.

### Create a Threshold

The NFVO sends a create request to create a threshold for the performance management job.

As part of the threshold creation, subscription details should be provided to receive any notification.

The NFVO receives the `ThresholdCrossedNotification` if ESC crosses a configured threshold.

Method Type:

POST

VNFM Endpoint:

```
{api_root}/vnfpm/v2/thresholds (Datastructure=CreateThresholdRequest)
```

Request Payload:

```
{
  "objectInstanceId": "9d20a459-b3ff-4d1c-9b63-0dae7444b645",
  "thSubObjectInstanceIds": ["9d20a459-b3ff-4d1c-9b63-0dae7444b645"],
  "objectType": "THRESHOLDJOB",
  "callbackUri": "http://localhost:45247/notification",
  "authentication": {
    "authType": ["BASIC"],
    "paramsBasic": {
      "userName": "admin",
      "password": "P@55w0rd!"
 }
  },
  "criteria": {
    "performanceMetric" : "uptime",
    "thresholdType" : "SIMPLE",
    "simpleThresholdDetails" : {
      "thresholdValue" : "74400.0",
      "hysteresis" : "10.0"
    }
  }
}
```

Response Payload:

```
{
    "id": "0341d294-f8db-408a-a68b-64b1db306304",
    "objectInstanceId": "9d20a459-b3ff-4d1c-9b63-0dae7444b645",
    "criteria": {
        "performanceMetric": "uptime",
        "thresholdType": "SIMPLE",
        "simpleThresholdDetails": {
            "thresholdValue": 74400.0,
            "hysteresis": 10.0
        }
    },
    "objectType": "THRESHOLDJOB",
    "callbackUri": "http://localhost:45247/notification",
    "thSubObjectInstanceIds": [
        "9d20a459-b3ff-4d1c-9b63-0dae7444b645"
    ],
    "_links": {
        "self": {
            "href":
"http://localhost:8250/or_vnfm/vnfpm/v2/thresholds/0341d294-f8db-408a-a68b-64b1db306304"
        },
        "object": {
            "href":
```

```
"http://localhost:8250/or_vnfm/vnflcm/v2/vnf_instances/9d20a459-b3ff-4d1c-9b63-0dae7444b645"

        }
    }
}
```

**Note** **Same Response Payload for *Query an individual threshold* and *Query all thresholds***

### Query an Individual Threshold

The NFVO can query the threshold of a performance management job.

GET

VNFM Endpoint:

```
{api_root}/vnfpm/v2/thresholds/{thresholdId}
```

Request Payload: NA

Response Payload:

```
{
  "id": "23f52511-9f72-4797-881b-c0f72e60a052",
  "objectInstanceId": "cc6a34e5-0463-459a-b367-493ba997775f",
  "criteria": {
    "performanceMetric": "default",
    "thresholdType": "SIMPLE",
    "simpleThresholdDetails": {
      "thresholdValue": 0.8,
      "hysteresis": 0.9
    }
  },
  "_links": {
    "self": {
      "href": "http://host:port/vnfpm/v2/thresholds/23f52511-9f72-4797-881b-c0f72e60a052"
    },
    "object": [
      {
        "href":
"http://host:port/vnflcm/v2/vnf_instances/cc6a34e5-0463-459a-b367-493ba997775f"
      }
    ]
  }
}
```

**Note** Attribute-based filtering is not possible when specifying a threshold id.

### Query All Thresholds

The NFVO can query the threshold of a performance management job.

Method Type:

GET

VNFM Endpoint:

```
{api_root}/vnfpm/v2/thresholds
```

Request Payload: NA

Response Payload:

```
{
  "id": "23f52511-9f72-4797-881b-c0f72e60a052",
  "objectInstanceId": "cc6a34e5-0463-459a-b367-493ba997775f",
  "criteria": {
    "performanceMetric": "default",
    "thresholdType": "SIMPLE",
    "simpleThresholdDetails": {
      "thresholdValue": 0.8,
      "hysteresis": 0.9
    }
  },
  "_links": {
    "self": {
      "href": "http://host:port/vnfpm/v2/thresholds/23f52511-9f72-4797-881b-c0f72e60a052"
    },
    "object": [
      {
        "href":
"http://host:port/vnflcm/v2/vnf_instances/cc6a34e5-0463-459a-b367-493ba997775f"
      }
    ]
  }
}
```

**Note** All the attribute names in the response payload and data types referenced from the attribute names are supported in the attribute-based filtering.

### Update a Threshold

The NFVO sends a update request to update a threshold for the performance management job.

Method Type:

PATCH

VNFM Endpoint:

```
http://localhost:8250/or_vnfm/vnfpm/v2/thresholds/{thresholdId}
```

Request Payload:

```
{
  "callbackUri": "http://localhost:45248/notification",
  "authentication": {
    "authType": ["BASIC"],
     "paramsBasic": {
        "userName": "admin",
        "password": "P@55w0rd!"
      }
  }
 }
```

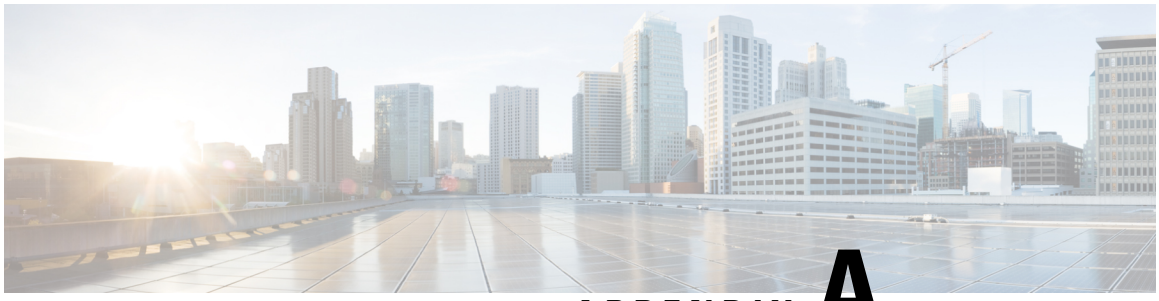
Response Payload:

```
{
    "callbackUri": "http://localhost:45248/notification"
}
```

### Delete a Threshold

The NFVO sends a delete request to delete the threshold configuration of the existing performance management job.

```
DELETE {api_root}/vnfpm/v2/thresholds/{thresholdId}
```



APPENDIX A

# ETSI Production Properties



# ETSI Production Properties

There are many properties that can be set to determine the behaviour of ESC. These properties enable integration of ESC with the NFVO in the system architecture.

You can access the properties file in the following location:

`/opt/cisco/esc/esc_database/etsi-production.properties`

The following table describes the parameters that can be used to control the behaviour of ESC acting as a VNFM within the ETSI NFV MANO stack.

***Table 10: ETSI Production Properties***

| Property Name | Description | Type | Default Value |
|---|---|---|---|
| `server.host` | The host IP address on which the ETSI service is located. This is a mandatory property if the server has multiple IP addresses, or if the deployment is configured for High Availability (it should then be set to the VIP). | String | |
| `server.host.preferInet6` | Where there are multiple IP address types assigned to the server, use the IPv6 address over any IPv4 address. | Boolean | `false` |

| Property Name | Description | Type | Default Value |
|---|---|---|---|
| `server.port` | The port used to communicate over HTTP. | Integer | `8250` |
| `server.port.https` | The port used to communicate over HTTPS. | Integer | `8251` |
| `certificate.validation` | Determine whether to validate a host in any certificate presented when using HTTPS. Relaxes the validation of certificates, especially useful in testing. | Boolean | `true` |
| `notification.maxThreads` | The maximum number of threads utilised for the notification service. | Integer | `3` |
| `notification.subscription.test` | Upon creating a new subscription, determine whether to test | Boolean | `true` |
| `notification.links.httpScheme` | The HTTP scheme used for communicating with the NFVO for notifications. The valid values are http and https. | Enum | `https` |
| `notification.retry.maxAttempt` | The number of retries for the notification retry mechanism. | Integer | `5` |
| `notification.retry.backOff.delay` | The interval for the notification retry mechanism. | Integer | `1000` |

| Property Name | Description | Type | Default Value |
|---|---|---|---|
| security.user.name | **Mandatory**: This is the REST API username. It is set by sudo escadm etsi set --rest_user <username>:<password> and should be synchronized here. | String | |
| nfvo.apiRoot | **Mandatory**: The apiRoot for the NFVO. | String | localhost:8280 |
| nfvo.httpScheme | The HTTP scheme used for communicating with the NFVO. The valid values are http and https. | Enum | http |
| nfvo.isPackageNotificationSupported | Determine if the VNFM will attempt to subscribe to package notifications. | Boolean | true |
| nfvo.callback.httpScheme | The HTTP scheme used for communicating with the NFVO when polling for responses. The valid values are http and https. | Enum | https |
| nfvo.userName | The username for NFVO credentials. | String | |
| nfvo.password | The password for NFVO credentials, required in plain text. | String | |
| retryTemplate.expotential.retryPolicy.maxAttempt | The number of retries for the exponential retry mechanism. | Integer | 1000 |
| retryTemplate.expotential.backOffPolicy.interval.initial | The starting interval for the exponential retry mechanism. | Integer | 1000 |

| Property Name | Description | Type | Default Value |
|---|---|---|---|
| retry.simple.maxAttempt | The number of retries for the simple retry mechanism. | Integer | 50 |
| retry.simple.backOff.delay | The interval for the simple retry mechanism. | Integer | 1000 |
| nfvo.allPackagesFilter | The value to use to filter packages on the NFVO when querying for packages. | String | |
| mapping.vimConnectionInfo.accessInfo.username | Provide an alternate attribute name when specifying the username in accessInfo. | String | username |
| mapping.vimConnectionInfo.accessInfo.password | Provide an alternate attribute name when specifying the password in accessInfo. | String | password |
| mapping.vimConnectionInfo.accessInfo.project | Provide an alternate attribute name when specifying the project in accessInfo. | String | project |
| mapping.vimConnectionInfo.accessInfo.projectDomain | Provide an alternate attribute name when specifying the projectDomain in accessInfo. | String | projectDomain |
| mapping.vimConnectionInfo.accessInfo.userDomain | Provide an alternate attribute name when specifying the userDomain in accessInfo. | String | userDomain |
| mapping.vimConnectionInfo.accessInfo.vim_project | Provide an alternate attribute name when specifying the vim_project in accessInfo. | String | vim_project |

| Property Name | Description | Type | Default Value |
|---|---|---|---|
| mapping.vimConnectionInfo.accessInfo.vim_vdc | Provide an alternate attribute name when specifying the vim_vdc in accessInfo. | String | vim_vdc |
| nfvo.grantRequest.retry.maxAttempt | The number of retries for failed GrantRequest attempts. | Integer | 5 |
| nfvo.grantRequest.retry.backOff.delay | The interval for the retries for failed GrantRequest attempts. | Integer | 1000 |
| spring.jackson.date-format | A string to represent a date format to allow for varying NFVO implementations to read dates correctly. | String | yyyy-MM-dd'T'HH:mm:ss.SSSXXX |
| nfvo.authenticationType | Setting the authentication type of the NFVO that is being used. Required property. Valid options are "BASIC", "OAUTH2", "OFF". All other Strings will be treated the same as "OFF". Use this to enable Basic and OAuth2 authentication. | String | |
| nfvo.clientID | For NFVO OAuth2 Authentication. Client ID. | String | |
| nfvo.clientSecret | For NFVO OAuth2 Authentication. Client Secret. | String | |
| nfvo.tokenEndpoint | For NFVO OAuth2 Authentication. The endpoint for ETSI to retrieve a OAuth2 token from the NFVO. | String | |

| Property Name | Description | Type | Default Value |
|---|---|---|---|
| rate.limit.capacity.read | Set the bucket capacity for read (GET) requeusts to the ETSI REST API.<br>By default this is disabled. | Integer | |
| rate.limit.perSecond.read | Sets the rate (per second) at which the bucket empties for the read (GET) requests to the ETSI REST API.<br>By default this is disabled. | Double | |
| rate.limit.capacity.write | Set the bucket capacity for write (POST, PUT, PATCH, DELETE) requeusts to the ETSI REST API.<br>By default this is disabled. | Integer | |
| rate.limit.perSecond.write | Sets the rate (per second) at which the bucket empties for the write (POST, PUT, PATCH, DELETE) requests to the ETSI REST API.<br>By default this is disabled. | Double | |
| log.multiple.query | The flag to enable logging response for query multiple VNF instances and response for query multiple VNF lifecycle management operation occurrences. | Boolean | false |

| Property Name | Description | Type | Default Value |
|---|---|---|---|
| scheduled.cleanup[vnfLcmOpOcc].interval.value | Set the interval value for the VnfLcmOpOcc cleanup task.<br>The combination of interval.value and interval.unit will determine the frequency that the cleanup task is executed. | Integer | 1 |
| scheduled.cleanup[vnfLcmOpOcc].interval.unit | Set the interval unit for the VnfLcmOpOcc cleanup task.<br>The combination of interval.value and interval.unit will determine the frequency that the cleanup task is executed.<br>Valid values:<br>`NANOS, MICROS, MILLIS, SECONDS, MINUTES, HOURS, HALF_DAYS, DAYS` | | DAYS |
| scheduled.cleanup[vnfLcmOpOcc].age.value | Set the age value for the VnfLcmOpOcc cleanup task.<br>The combination of age.value and age.unit will determine the age of orphan records to be deleted. | Integer | 60 |

| Property Name | Description | Type | Default Value |
|---|---|---|---|
| scheduled.cleanup[vnfLcmOpOcc].age.unit | Set the age unit for the VnfLcmOpOcc cleanup task.<br>The combination of age.value and age.unit will determine the age of orphan records to be deleted.<br>Valid values:<br>`NANOS, MICROS, MILLIS, SECONDS, MINUTES, HOURS, HALF_DAYS, DAYS` | | DAYS |
| paging.size | Setting a value > 0 turns on paging for query endpoints.<br>This value represents the number of results to be included per page.<br>If a response is paged and there are further pages then the response will include a header named "Link" with rel="next" for example:<br><http://example.com<br>/resources?nextpage_<br>opaque_marker=abc123>; rel="next"<br>The link url will retrieve the next page.<br>If there are no further pages to be retrieved then the Link header will be omitted. | Integer | 0 |

| Property Name | Description | Type | Default Value |
|---|---|---|---|
| attribute.selector.default.all_fields | Setting the value to true will change the behaviour of ETSI query endpoints to return the full set of attributes if an attribute selector is not provided (all_fields). | Boolean | false |
| monitorMigration.terminalStateOnError | Defines whether the lifecycle operation will move to terminal state automatically on error during monitoring migration. | Boolean | false |
| sync.supported | Allows the operations ENABLE/DISABLE MONITOR to be performed synchronously. Note: This is only supported in a non cloud native environment. | Boolean | false |
| `subscription.notifications.infra.filter.operationTypes` | Part of the properties used to register the infrastructure notifications. Defines the operation types filter.<br>**Note** If these properties are set, then infrastructure notifications will not be sent to the other subscriptions. | `MONITORING_MIGRATION`<br>`MONITORING_OPERATION` | |

| Property Name | Description | Type | Default Value |
|---|---|---|---|
| `subscription.notifications.infra.filter.operationStates` | Part of the properties used to register the infrastructure notifications.<br>Defines the operation states filter.<br>**Note** If these properties are set, then infrastructure notifications will not be sent to the other subscriptions. | STARTING<br>PROCESSING<br>COMPLETED<br>FAILED_TEMP<br>FAILED<br>ROLLING_BACK<br>ROLLED_BACK | |
| `subscription.notifications.infra.callbackUri` | Part of the properties used to register the infrastructure notifications.<br>Defines the callback URI to send the notifications. This is the full URI including the scheme, host and port.<br>**Note** If these properties are set, then infrastructure notifications will not be sent to the other subscriptions. | String | |

| Property Name | Description | Type | Default Value |
|---|---|---|---|
| `subscription.notifications.infra.authentication.authType` | Part of the properties used to register the infrastructure notifications.<br>Defines the authentication type for the notification.<br>**Note** If these properties are set, then infrastructure notifications will not be sent to the other subscriptions. | `BASIC`<br>`OAUTH2_CLIENT_CREDENTIALS` | |
| `subscription.notifications.infra.authentication.paramsBasic.userName` | Part of the properties used to register the infrastructure notifications.<br>Defines the BASIC authType username.<br>**Note** If these properties are set, then infrastructure notifications will not be sent to the other subscriptions. | String | |

| Property Name | Description | Type | Default Value |
|---|---|---|---|
| `subscription.notifications.infra.authentication.paramsBasic.password` | Part of the properties used to register the infrastructure notifications.<br>Defines the BASIC authType password.<br>**Note** If these properties are set, then infrastructure notifications will not be sent to the other subscriptions. | String | |
| `subscription.notifications.infra.authentication.paramsOauth2ClientCredentials.clientId` | Part of the properties used to register the infrastructure notifications.<br>Defines the OAUTH2_CLIENT_CREDENTIALS authType client id.<br>**Note** If these properties are set, then infrastructure notifications will not be sent to the other subscriptions. | String | |

| Property Name | Description | Type | Default Value |
|---|---|---|---|
| `subscription.notifications.infra.authentication.paramsOauth2ClientCredentials.clientPassword` | Part of the properties used to register the infrastructure notifications.<br>Defines the `OAUTH2_CLIENT_CREDENTIALS` authType client password.<br>**Note** If these properties are set, then infrastructure notifications will not be sent to the other subscriptions. | String | |
| `subscription.notifications.infra.authentication.paramsOauth2ClientCredentials.tokenEndpoint` | Part of the properties used to register the infrastructure notifications.<br>Defines the `OAUTH2_CLIENT_CREDENTIALS` authType token endpoint. This is the full URI including the scheme, host and port.<br>**Note** If these properties are set, then infrastructure notifications will not be sent to the other subscriptions. | String | |

For information on resource definitions, see Resource Definitions for ETSI API, on page 5.

ETSI Production Properties