Cisco Elastic Services Controller 5.0 ETSI NFV MANO User Guide

First Published: 2019-08-09

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 its VNFs are deployed in a Virtual Infrastructure Manager (VIM). Currently OpenStack, VMware vCenter, VMware vCloud Director, CSP 2100 / 5000, and Amazon Web Services (AWS) are the supported VIMs. The administrator must be familiar with the VIM layer, vCenter, OpenStack and AWS resources, 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
AWS	Amazon Web Services (AWS) is a secure cloud services platform, offering compute, database storage, content delivery and other functionalities.
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.

Terms	Definitions
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.
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.

Terms	Definitions
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

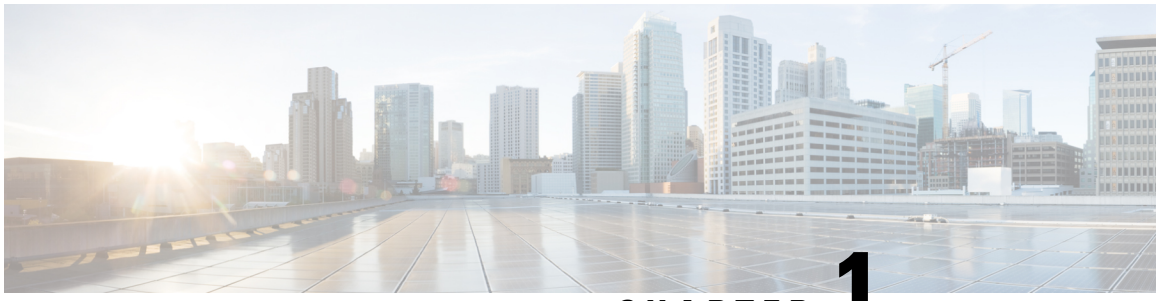
The Cisco ESC doc set comprises of the following guides to help you perform installation, configuration; the lifecycle management operations, healing, scaling, monitoring and maintenance of the VNFs using different APIs.

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 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 informaiton 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	v0.10.0	Format and structure for the VNF Descriptor
SOL003	v2.4.1	Defines all interactions over the Or-Vnfm reference point

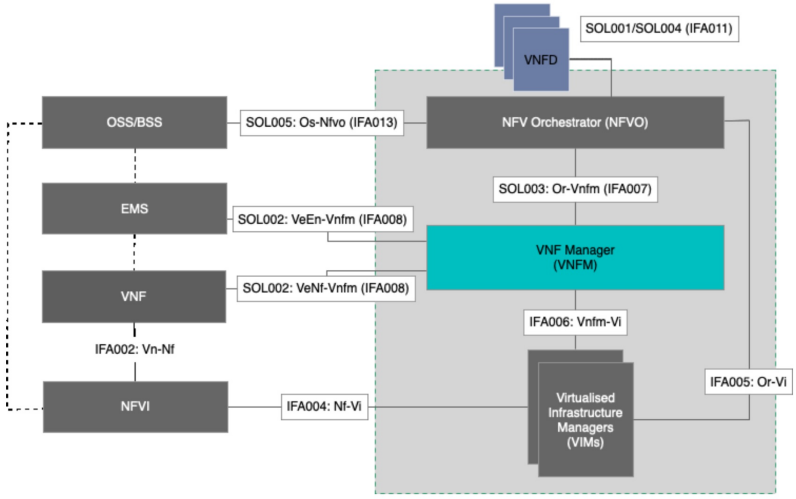
The initial implementation of ETSI standard supports ETSI MANO API over Or-Vnfm reference point, which is the interface between ESC and NFVO. The Or-Vnfm reference point details the interactions to onboard ETSI compliant VNF packages, manage resources, and VNF lifecycle management (LCM) operations.

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.

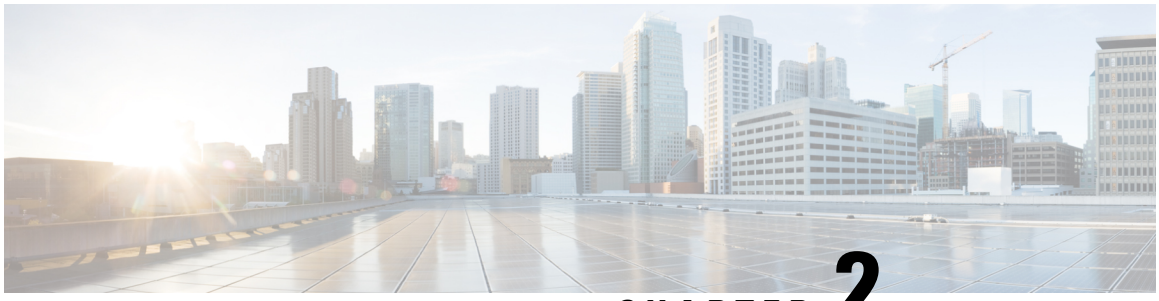
For more information on Or-Vnfm reference point, see the *ETSI Group Specification document* on the ETSI website. The figure below represents the NFV MANO architecture with the Or-Vnfm reference point.

ETSI NFV MANO Northbound API Overview

Figure 1: NFV MANO Architecture with Reference Points



For information on managing resources, see Resource Definitions for ETSI API, on page 3.



CHAPTER 2

Managing Resources

Managing Resources

Resource Definitions for ETSI API

Cisco Elastic Services Controller (ESC) resources comprises 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 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 11.
Images	Out of band images 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 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. **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 out-of-band volumes as Cisco extension.
External Networks (Virtual Link)	External networks specified in the instantiation payload to which external connection points will connect.
Externally Managed Internal Virtual Links	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 VNF Lifecycle On Or-Vnfm Reference Point, on page 21.

Updating Resource Definitions

This section provides details about updating ETSI API resource definitions.

Updating the VNF Flavour

A VNF flavour can be updated by updating the TOSCA parameters in the VNFD template. 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_2
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:
# None

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_2

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_0_0
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/v1/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/v1/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": [
            {
              "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 instantiatedVnfInof 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

- If the VNF is instantiated by the Test Harness, the flavour is persisted and used in the subsequent LCM operations so that the VNFD always uses the same mappings.
- If the instantiate operation is called from elsewhere (such as cURL or Postman) then the flavour contained in the grant is persisted and used in the subsequent LCM operations.
- If the Test Harness receives a grant request for a VNF not instantiated through it, then the grant is most likely to fail if no substitution mapping occurs.
- 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 VNF Lifecycle On Or-Vnfm Reference Point, on page 21.

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.

The NFVO makes a POST request providing the clientId 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/v1/subscriptions
```

Headers

```
Authorization: Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJjaHJpcyIsImlzcyI6IkVUU0k
tVk5GTSIsImlhdCI6MTU1ODYwMzk2NiwiZXhwIjoxNTU4NjA0NTY2fQ.lAtre7vdCKJjgzNs7p9P3NS2qMcXegC-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

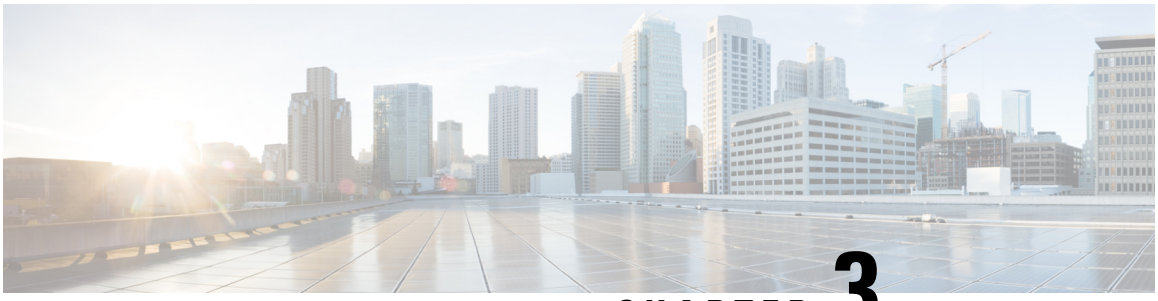
```
sudo escadm etsi oauth2_clients --set <CLIENT_ID>:<CLIENT_SECRET>
```

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 87.



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 a 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.

VimConnectionInfo in OpenStack:

```
{
  "id": "435456",
  "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"
  }
}
```

VimConnectionInfo in vCloud Director:

```
{
  "id": "435456",
  "vimType": "VMWARE_VCD",
  "interfaceInfo": {
    "endpoint": "https://10.85.103.150"
  },
  "accessInfo": {
    "username": "admin@cisco",
    "password": "bmkQJtyDrbPFnJT8ENdZw2Maw",
    "vim_project": "cbamnso",
    "vim_vdc": "vdc1"
  }
}
```

Creating New VIM Connectors

During the ETSI LCM operation, ESC checks each VimConnectionInfo against the VnfInstance to see if an existing VIM connector is available. 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 12.

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
  - vCloud Director
    - 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:

OpenStack

- username
- password
- project
- projectDomain
- userDomain
- vim_project

vCloud Director

- username
- password
- vim_project
- vim_vdc

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 12.

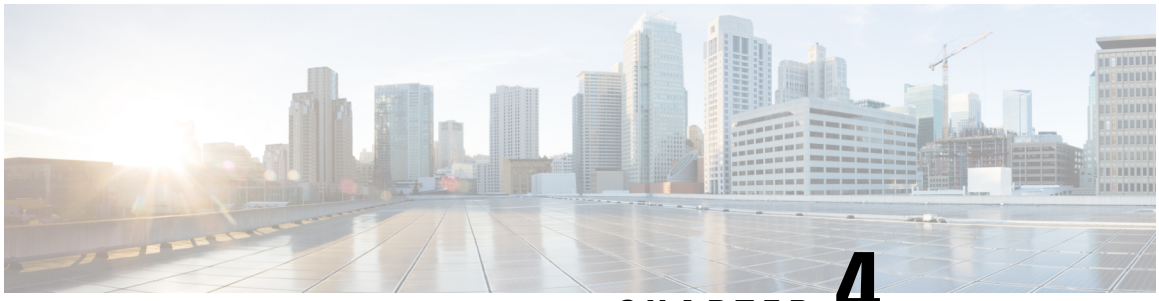
Updating the VIM Connector

The ETSI API updates the existing VimConnectionInfo, and the associated VIM connector via the Modifying Virtual Network Functions, on page 33 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 ang 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 12.



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 properties. The VNFD conforms to the *GS NFV-SOL 001* specifications and standards specified by ETSI.

The VNFD file describes the instantiation parameters and operational behaviors of the VNFs. It 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 Or-Vnfm Reference Point, on page 22.

Defining Extensions to the Virtual Network Function Descriptor

ESC implements extensions to the VNFD defined by Cisco to expose the more advanced concepts supported by ESC, but missing in the ETSI standards. These extensions are strongly typed in the Cisco types definition to describe the overridden data, node, and interface types.

VNF Configurable Properties

The VNF node type is always customized for each VNF. The Cisco extensions provide the ability to specify the recovery policy and time to wait for the VNF to recover before ESC considers any mitigating action.

For example:

```
vnf:
      type: cisco.VPC.1_0.1_0
      properties:
        descriptor_id: b98450dd-f532-4a42-8419-e3dc04327318
        descriptor_version: '3.8'
        provider: cisco
        product_name: VPC
        software_version: 1.0
        product_info_name: 'Virtual Packet Core (VPC); 32 vCPUs, 64Gb RAM, 66Gb vStorage'
        vnfm_info:
          - '9:Cisco Elastic Services Controller:v04.04.01'
```

```
    configurable_properties:
      is_autoscale_enabled: false
      is_autoheal_enabled: false
    lcm_operations_configuration:
      heal:
        recovery_action: REBOOT_THEN_REDEPLOY
        recovery_wait_time: 0
    flavour_id: default
    flavour_description: 'Default VNF Deployment Flavour'
```

Compute

The Cisco Compute node allows for many of the ESC features to be exposed via the extended ETSI data model. 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.

For example:

```
vdu1:
  type: cisco.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
    name_override: my-vdu-1
    vdu_profile:
      min_number_of_instances: 1
      max_number_of_instances: 1
      static_ip_address_pool:
        network: esc-net
        ip_address_range:
          start: { get_input: VDU1_NETWORK_START }
          end: { get_input: VDU1_NETWORK_END }
        ip_addresses: { get_input: VDU1_SCALE_IP_LIST }
    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 }
  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
```

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

Connection Point

The Cisco extensions to the VduCp node type mainly allows for improved IP addressing capabilities and accessibility to the interface. The features added the connection point are as follows:

- Overriding the automatically generated name for a port on the VIM
- Static IP Addresses (and pools for scaling)
- 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
- Whether port security is enabled

For example:

```
vdu1_nic0:
      type: cisco.nodes.nfv.VduCp
      properties:
        layer_protocols: [ ipv6 ]
        protocol:
          - associated_layer_protocol: ipv6
```

```
      trunk_mode: false
      order: 0
      nw_card_model: virtio
      iface_type: direct
      management: true
      name_override: my-vdu1-nic0
      ip_subnet:
        - ip_address: { get_input: VDU1_NIC0_IP }
      allowed_address_pairs:
        - ip_address: { get_input: VDU1_NIC0_AADR_PAIRS }
      port_security_enabled: false
    requirements:
      - virtual_binding: vdu1
```

Volume

ESC supports out-of-band volume as a Cisco extension. This allows the specification of the persistent volume UUID as the resourceId property against the cisco.nodes.nfv.Vdu.VirtualBlockStorage node to be used in place of the ephemeral volume defined in the VNFD. Instead of adding extra properties, ESC allows 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: cisco.nodes.nfv.Vdu.VirtualBlockStorage
    properties:
      resource_id: { get_input: VDU1_BOOT_VOL_UUID }
      virtual_block_storage_data:
        size_of_storage: 4GB
        vdu_storage_requirements:
          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
```

To specify the out-of-band resource in place of ephemeral resource, ESC allows you to use the incoming request to match tags in the VNFD during instantiation. A new data structure is appended to the existing InstantiateVnfRequest.

For example,

```
{
  "flavourId": "default",
  "instantiationLevelId": "default",
  "extVirtualLinks": [{}],
  "extManagedVirtualLinks": [{}],
  "extManagedVolumes": [
        {
            "virtualStorageDescId": "cf-cdr1-volume",
            "resourceId": "vol123"
        },
```

```
        {
            "virtualStorageDescId": "cf-boot1-volume",
            "resourceId": "vol456"
        }
    ],
  ...
}
```

Security Group Rule

As per the handling 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.

For example:

```
    - NETWORK_ORCH_SEC_GRP_1:
        type: cisco.policies.nfv.SecurityGroupRule
        group_name: { get_input: VIM_NETWORK_ORCH_SEC_GRP_1 }
        targets: [ vdu1_nic0 ]
```

Custom VM Name

The Cisco extension allows you to customize the VNFC (VM) name in a deployment using additional parameters. The ESC ETSI includes the additional parameters to customize VM names.

To configure the VM name on the VIM, you must first define the data type and then extend the Cisco node type for the compute node:

```
tosca_definitions_version: tosca_simple_yaml_1_2
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
```

These definitions allow the VNFD node_templates to use the inputs to map to the Compute node:

```
topology_template:
  inputs:
    ...

  node_templates:

#####################
# VDU configuration #
#####################
    c1:
      type: cisco.nodes.nfv.Vdu.Compute
      properties:
        name: control-function 1
        description: Vdu1 of an active:standby (1:1) redundant pair of CF VMs
        ...
```

```
      configurable_properties:
        additional_vnfc_configurable_properties:
          vim_flavor: { get_input: CF_FLAVOR }
          bootup_time: { get_input: BOOTUP_TIME_CF }
          vm_name_override: { get_input: VIM_C1_VM_NAME ] }
      ...
    capabilities:
      virtual_compute:
        properties:
          virtual_cpu:
            num_virtual_cpu: 8
          virtual_memory:
            virtual_mem_size: 16 GiB
    requirements:
      - virtual_storage:  cf-cdr1-volume
      - virtual_storage:  cf-boot1-volume
```

Specify *vm_name_override* under configurable properties of the compute node. 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.

For information on lifecycle management operations, see Managing VNF Lifecycle On Or-Vnfm Reference Point, on page 21.

SR-IOV

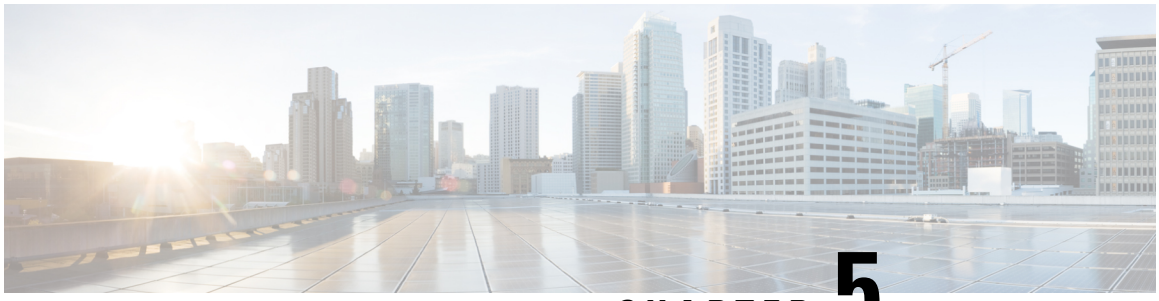
ESC ETSI NFV MANO supports SR-IOV properties using the Cisco data types. You can configure the interface to associate the VNFC with an SR-IOV pass through adapter.

Cisco data type:

```
cisco.datatypes.nfv.L2ProtocolData:
  derived_from: tosca.datatypes.nfv.L2ProtocolData
  properties:
    segmentation_id:
      type: integer
      required: false
```

Example VNFD:

```
  virtual_link_protocol_data:
  - associated_layer_protocol: ethernet
    l2_protocol_data:
      network_type: vlan
      physical_network: vlan_network
      segmentation_id: { get_input: VL1_SEG_ID }
```



CHAPTER 5

Managing VNF Lifecycle Operations

Managing VNF Lifecycle On Or-Vnfm Reference Point

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. 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.

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

VNF Lifecycle Operations On Or-Vnfm Reference Point

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 two options with regards to connecting 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) or created as part of the request if new vimConnectionInfo is supplied. See VIM Connectors Overview, on page 11.

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 The initial implementation of the ETSI MANO API supports only a single VIM. The tenant/project is currently specified using the resourceGroupId.

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

- The chosen deployment flavour
- The VIM connection details (vimConnectionInfo)
- 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

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/v1/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/v1/vnf_instances/14924fca-fb10-45da-bcf5-59c581d675d8/instantiate"

        },
        "self": {
            "href":
"http://localhost:8250/vnflcm/v1/vnf_instances/14924fca-fb10-45da-bcf5-59c581d675d8"
        }
    }
}
```

For instantiating VNFs, see Instantiating Virtual Network Functions, on page 24.

Instantiating Virtual Network Functions

The instantiation request triggers a number of message exchanges, which allows the call flow to be completed in order to instantiate a VNF instance. The resources are allocated when the VNF instance is instantiated. It requires the VNF instance identifier, returned by the create VNF request, encoded into the URL to which the request is posted.

The instantiation request sub-tasks within the flow include:

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

Method type:

```
POST
```

VNFM Endpoint:

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

HTTP Request Header:

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

Request Payload (ETSI data structure: InstantiateVnfRequest)

```
{
    "flavourId": "default",
    "extManagedVirtualLinks": [
        {
            "id": "my-network",
            "resourceId": "93fb90ae-0ec1-4a6e-8700-bf109a0f4fba",
            "virtualLinkDescId": "VLD1"
        }
    ],
    "vimConnectionInfo": [
        {
            "accessInfo": {
                "password": "P@55w0rd!",
                "username": "admin",
                "vim_project": "tenantName"
            },
            "extra": {
                "name": "esc"
            },
            "id": "default_openstack_vim",
            "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 same as a single flavour_id specified in the VNFD.

You can customize the VNF before instantiation by adding variables to the VNFD template. Specify the variables in the *additionalParams* field of the LCM request. The variables are name-value pairs, where the value can be either string, numeric or boolean. In the example below, the *cpus*, and *mem_size additionalParams* are defined in the VNFD template using the get_input: <TOSCA method>.

Note If there are multiple vm groups within the VNFD in a single ETSI deployment, they must all use the same VIM.

When this template is submitted to the VNFM, the variables are merged into the same VNF instance. The *additionalParams* variables are merged with the VNF variables, and actual values for the variables are provided only during instantiation.

The list of parameters supplied are 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:

```
tosca_definitions_version: tosca_simple_yaml_1_2

imports:
  - cisco_nfv_sol001_types.yaml
  - etsi_nfv_sol001_vnfd_0_10_0_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"

node_templates:

    vdu1:
      type: cisco.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 }
```

If further LCM requests with *additionalParams* variables are submitted for the same VNF, then the new variables overwrite the existing variables. The VNFM uses the new variables for instantiation.

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
```

To specify an external virtual link to be used in place of automation_net in the VNF deployment, the following data structure must be 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 have implemented an extension to support this. The resource Id of the persistent volume can be supplied as an additionalParam and tied to a volume in the VNFD using an optional property, as per the following example:

```
       example-volume:
      type: cisco.nodes.nfv.Vdu.VirtualBlockStorage
      properties:
        resource_id: { get_input: EX_VOL_UUID }
        virtual_block_storage_data:
          size_of_storage: 200 GB
          vdu_storage_requirements:
            vol_id: 1
            bus: ide
            type: LUKS
```

Requesting Permission via Grant

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

```
{
  "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/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 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": [
        {
            "id": "esc-005e4412-e056-43a9-8bc0-d6699c968a3c",
            "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.

Retrieving the Deployment Descriptor from ESC

The NFVO can retrieve the ESC datamodel instance in the form of a deployment descriptor. The NFVO can view all the inputs provided at the time of instantiation and changes made later to the deployment descriptor.

To retrieve the deployment descriptor, you must:

- Create the VNF
- Provide the vnfinstanceId

Method Type

```
GET
```

VNFM Endpoint

```
/vnflcm/v1/ext/vnfinstances/{vnfInstanceId}/deployment
```

HTTP Request Header

```
content-Type:application/xml
```

Request Payload

not applicable.

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 isntance.

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/v1/vnf_instances/14924fca-fb10-45da-bcf5-59c581d675d8/instantiate"

    },
    "self": {
      "href":
"http://localhost:8250/vnflcm/v1/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",
"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": [
{
```

```
"extLinkPorts": [
{
"cpInstanceId": "extCp-4143f7d4-f581-45fc-a730-568435dfdb4f",
"id": "extLP-4143f7d4-f581-45fc-a730-568435dfdb4f",
"resourceHandle": {
"resourceId": "d6a4c231-e77c-4d1f-a6e2-d3f463c4ff72",
"vimConnectionId": "default_openstack_vim"
}
}
],
"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"
}
]
}
]
}
```

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",
    "vnfPkgId": "pkg-xyzzy-123",
    "vnfConfigurableProperties": {
        "isAutoscaleEnabled": "true"
    },
    "metadata": {
        "serialRange": "ab123-cc331",
        "manufacturer": "Cisco"
    },
    "extensions": {
        "testAccess": "false",
        "ipv6Interface": "false"
    },
    "vimConnectionInfo": [
        {
            "id": "vci1",
            "vimType": "openstack",
            "interfaceInfo": {
                "uri": "http://172.16.14.27:35357/v3"
            },
            "accessInfo": {
                "domainName": "default",
                "projectName": "admin",
                "userName": "default"
            }
        }
    ]
}
```

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/v1/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.
```

Terminating Virtual Network Functions

The terminating VNF request terminates a VNF instance. The resources are deallocated but remain reserved for this instance until it is deleted. Permission is required from the NFVO (bi-directional Grant flow) for this operation. The VNF instance can be decommissioned gracefully or forcefully.

Note The OpenStack API supports only forceful termination.

As per the Instantiate VNF Request, the terminate VNF request requires the VNF instance identifier encoded into the URL to which the request is posted.

Method Type:

```
POST
```

VNFM Endpoint:

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

HTTP Request Headers:

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

Request Payload (ETSI data structure: TerminateVnfRequest)

```
{
  "terminationType":"FORCEFUL",
  }
```

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/v1/vnf_lcm_op_occs/dae25dbc-fcde-4ff9-8fd6-31797d19dbc1
Content-Length: 0
Date: Thu, 04 Jan 2018 12:45:59 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:

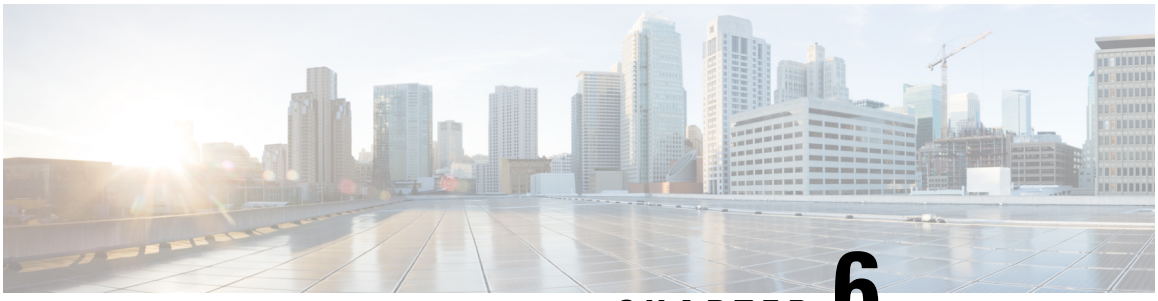
```
not applicable.
```

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.
```



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:

```
vdu2:
      type: cisco.nodes.nfv.Vdu.Compute
      properties:
        name: Example VDU 2
        description:  Example VDU
        ...
        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
        property_list:
            - name: vmname
              value: vdu2
            - name: status
            value: ERROR
  admin_rules:
    VM_ALIVE-1:
      event_name: 'VM_ALIVE-1'
      action:
        - 'ALWAYS log'
        - 'FALSE recover autohealing'
        - 'TRUE esc_vm_alive_notification'
        property_list:
            - name: vmname
              value: vdu2
            - name: status
            value: SUCCESS
...
```

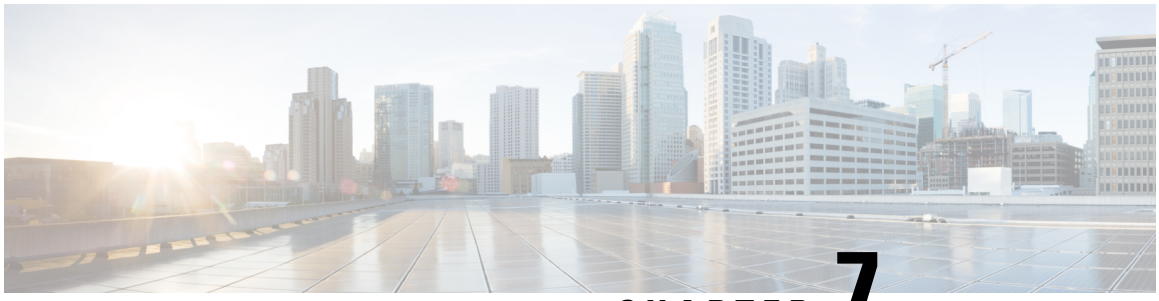
The kpi_data shown above is the default KPI required that is required in all deployments at a minimum so that the VM_ALIVE message is generated to tell ESC Manager that the VNFC has been deployed successfully; it consists of the KPI, how it is collected and the actions to be executed when the KPI is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

Cisco data structure properties

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

Data Type	Property Name	Description	Values
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		

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



CHAPTER 7

Monitoring VNF Using D-MONA

Onboarding D-MONA

ETSI NFV MANO 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.
- Only ESC Active/Active deployment is supported by D-MONA.

For information on deploying D-MONA, see Deploying D-MONA, on page 43.

Deploying D-MONA

ESC supports 1:1 D-MONA deployment for a VIM. A single D-MONA instance monitors VNF on a single VIM.

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 their respective liveness.

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

For information on deploying VNFs using D-MONA, see Deploying VNF Using D-MONA, on page 44.

Configuring D-MONA

D-MONA reuses the ESC 5.0 image. 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 configuration:

```
<configuration>
 <dst>--user-data</dst>
 <file>file:///opt/cisco/esc/esc-config/dmona/iser-data.template</file>
 <variable>
  <name>vm_credentials</name>
  <val>REPLACED_WITH_GENERATED_PWD</val>
 </variable>
</configuration>
```

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

The following example shows the D-MONA ESC certificate configuration:

```
<configuration>
 <dst>/opt/cisco/esc/moan/dmona.crt</dst>
 <data>$DMONA_CERT</data>
</configuration>
```

For monitoring using D-MONA, see Monitoring Using D-MONA, on page 45.

Deploying VNF Using D-MONA

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 true within the same `vim_connector`. When ESC detects D-MONA, monitoring of the VNF is assigned to that D-MONA, otherwise the local MONA handles the monitoring.

The following example shows the D-MONA VNFD:

```
tosca_definitions_version: tosca_simple_yaml_1_2
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'
```

Monitoring Using D-MONA

To monitor the VNFs using D-MONA, you must deploy the ESTI VNFD D-MONA and then deploy the ESTI VNFD monitored by D-MONA. For information on deploying D-MONA, see Deploying VNF Using D-MONA, on page 44.

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.

Table 4: Input Parameters for D-MONA Deployment

Parameter	Description
SW_IMAGE_NAME	The name of ESC image
DMONA_CERT	The HTTPS certificate
DMONA_AGENT_ID	The URL or ID of the monitoring agent that will monitor the VM
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:

- monitoring_agent—value defined for DMONA_AGENT_ID in the input parameter.
- property_list
  - name—protocol
  - value—https
  - name—port
  - value—8443

- name—path
- value—mona/v1/health/status

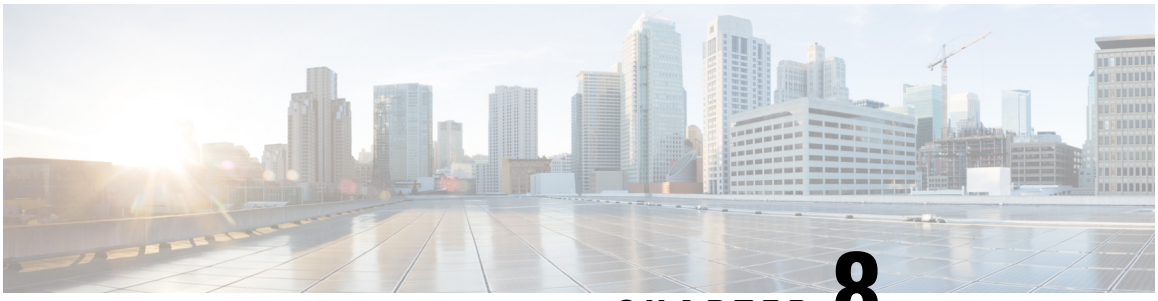
Config data parameters:

- 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—true

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: true
```



CHAPTER 8

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, 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 along with rules. The rules define the action to be taken (including events that are triggered) as a result of these KPI conditions to heal a VNF.

ESC ETSI 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
    ...
    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'
...
```

This 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.

There are three 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 will:

- 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
- REDEPLOY_ONLY—only attempt to redeploy the VM

The recovery policy is configured at a VNF-level, and applies to each VNFC contained within. 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).

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 alternatively modified against the VNF instance prior to instantiation.

The recovery of the VNF is to request action against the affected VNFCs. If the service fails to deploy, then the lifecycle management operation fails, if ESC cannot manage to recover the service using the defined policy after the initial deployment operation times out.

To modify the autoheal flag (*isAutohealEnabled*) VNF instance resource, see Modifying Virtual Network Functions, on page 33.

If autoheal is *not enabled*, only the alarm is dispatched to all the subscribers. The subscriber can initiate a manual HealVnfRequest. The data structures are available for any VNF specific actions. There are no mandatory parameters.

For example:

```
Request Payload (ETSI data structure: HealVNFRequest)
POST /vnf_instances/{vnfInstanceId}/heal
{
    "cause": "b9909dde-e21e-45ec-9cc0-9e9ae413eee0",
}
```

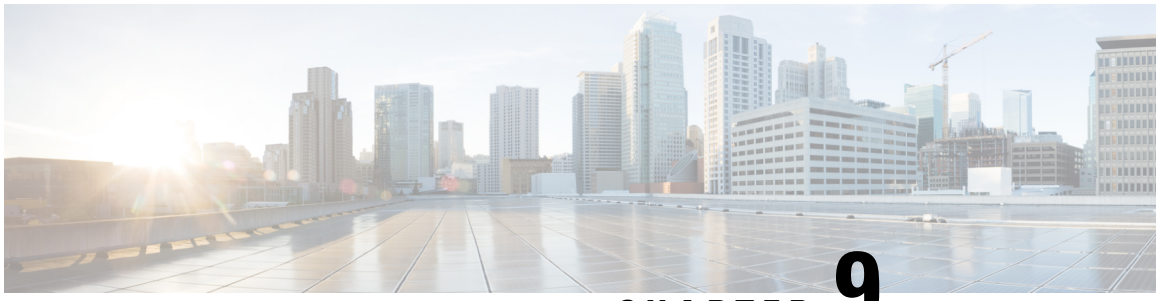
Additional parameters can be used to specify an overriding recovery policy, regardless of the policy configured at the time of deployment.

The recovery policy can be specified at VNFC level using additional parameters. This will override the values set at the VNF level. If the recovery policy is not specified at VNFC level, then ESC will inherit the properties from the VNF level recovery policy.

An optional additional parameter is added to the cisco.datatypes.nfv.VnfcAdditionalConfigurableProperties data type to support VNFC level recovery.

```
cisco.datatypes.nfv.VnfcAdditionalConfigurableProperties:
  derived_from: tosca.datatypes.nfv.VnfcAdditionalConfigurableProperties
  properties:
    ...
    is_vnfc_autoheal_enabled:
      type: boolean
     description: It permits to enable (TRUE)/disable (FALSE) the auto-healing functionality.
 If the properties is not present for configuring, then VNF-level property is used instead
      required: false
    recovery_action:
      type: string
      required: false
      constraints:
        - valid_values: [ REBOOT_THEN_REDEPLOY, REDEPLOY_ONLY, REBOOT_ONLY ]
```

For information on monitoring, see Monitoring Virtual Network Functions Using ETSI API, on page 39.



CHAPTER 9

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.

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 be scaled as there are no further IP addresses to assign to the connection points on the newly spun up VNFC instances. Instead, a pool of further static IP addresses can be specified. This is an extension to the ETSI specification.

The following example explains how to create a static IP pool using a list of IP addresses, IP ranges or a gateway with netmask (one or a combination of more than one can be specified):

```
    vdu2:
      type: cisco.nodes.nfv.Vdu.Compute
      properties:
        name: 'Webserver1'
        description: 'Webserver VNFC'
        vdu_profile:
          min_number_of_instances: 1
          max_number_of_instances: 6
          static_ip_address_pool:
            network: network1
            ip_addresses:
              - ip_address: 192.168.100.0
              - ip_address: 192.168.100.1
              - ip_address: 192.168.100.2
              - ip_address: 192.168.100.3
            ip_address_range:
              - start: 172.16.233.10
```

```
            end: 172.16.233.15
          - start: 172.16.233.20
            end: 172.16.233.25
        gateway: 172.10.11.0
        netmask: 255.255.255.0
```

The scaled out VNFC instance that has connection points with static IP addresses is assigned to a network. This is the key to identify which IP address pool to use when the scaled out instance is deployed. The static IPs are specified at deployment as part of the inputs in the InstantiateVnfRequest. For information on instantiating VNFs, see Instantiating VNFs.

The inputs are provided as part of the additionalParams through the VNFD.

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*.

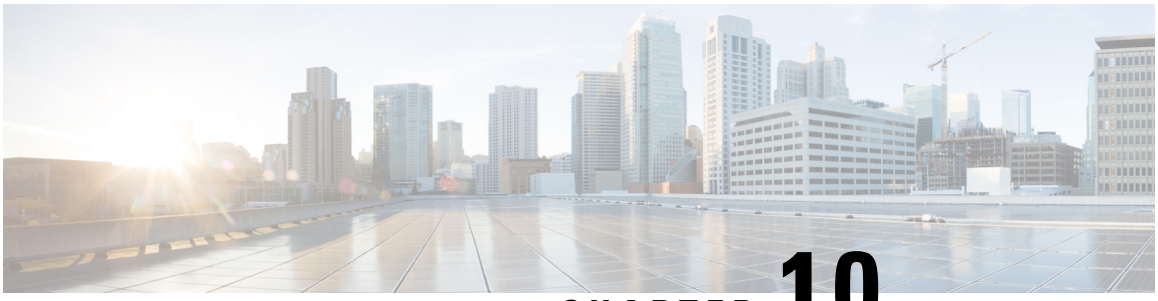
Autoscaling of VNFs

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.



CHAPTER 10

Error Handling Procedures

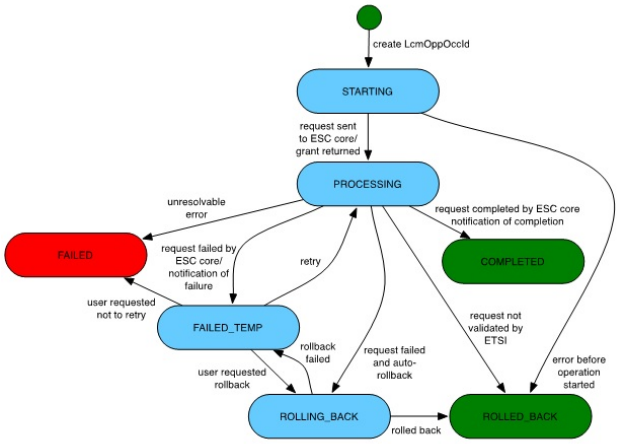
VNF Lifecycle Management Error Handling Procedures

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.

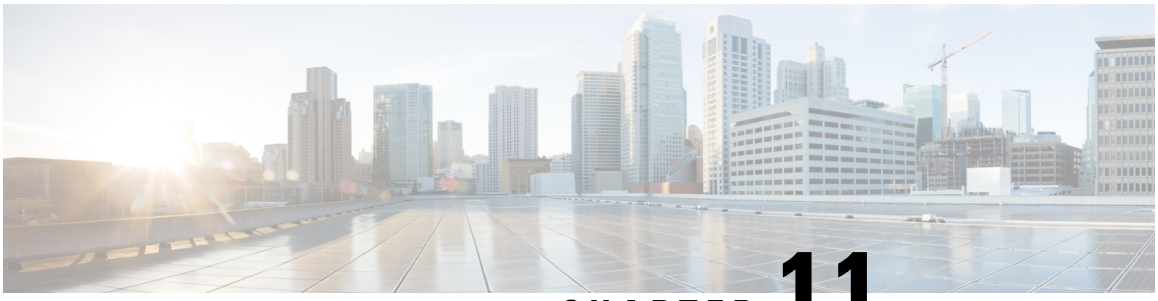
Error Handling Procedures for ETSI VNF Lifecycle Operations

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 Or-Vnfm Reference Point, on page 22.



CHAPTER 11

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:

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

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 value is ACKNOWLEDGE. 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/v1/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 overheading",
    "isRootCause": "false"
}
```

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/subscriptions
```

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 63.

  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>
```

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:

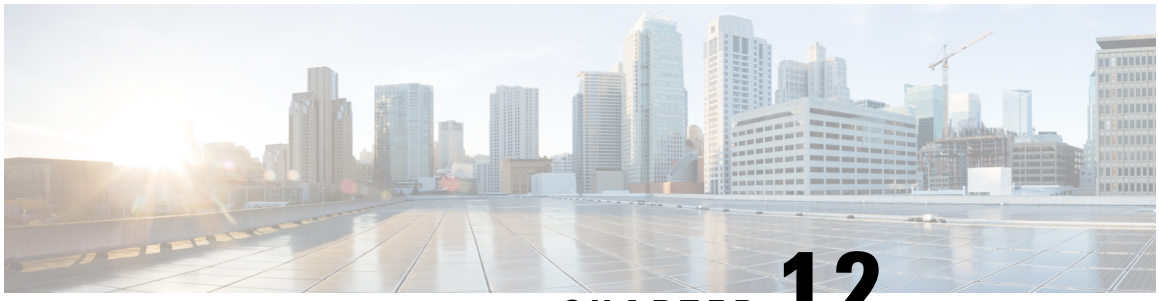
```
{
  /* "instantiationLevelId":"id111", */
  "scaleInfo": [
    { "aspectId":"processing", "scaleLevel":"3" },
    { "aspectId":"database",   "scaleLevel":"2" }
  ]
  "additionalParams": {
    "password": "pass1234",
    "username": "admin"
  },
  "action": "scale_to_level"
}
```

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 12

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
- Manage subscriptions, query, subscribe or terminate subscriptions

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.

Method Type:

POST

VNFM Endpoint:

```
{api_root}/vnfpm/v1/pm_jobs (Data structure=CreatePmJobRequest)
```

Request Payload:

```
{
  "objectInstanceIds": [
    "cc6a34e5-0463-459a-b367-493ba997775f"
  ],
  "criteria": {
    "performanceMetric": [
      "default"
    ],
    "performanceMetricGroup": [
      "default"
    ],
    "collectionPeriod": 3600,
    "reportingPeriod": 14400
  }
}
```

Response Payload:

```
{
  "id": "13963644-11b0-4302-a13b-26ca3d9eb8f8",
  "objectInstanceIds": [
    "cc6a34e5-0463-459a-b367-493ba997775f "
  ],
  "criteria": {
    "performanceMetric": [
      "default"
    ],
    "performanceMetricGroup": [
      "default"
    ],
    "collectionPeriod": 3600,
    "reportingPeriod": 14400
  },
  "_links": {
    "self": {
      "href": "http://host:port/vnfpm/v1/pm_jobs/13963644-11b0-4302-a13b-26ca3d9eb8f8"
    },
    "objects": [
      {
        "href":
"http://host:port/vnflcm/v1/vnf_instances/cc6a34e5-0463-459a-b367-493ba997775f"
      }
    ]
  }
}
```

Query an Individual Performance Management Job

The NFVO queries for the individual performance management job.

Method Type:

GET

VNFM Endpoint:

`{api_root}/vnfpm/v1/pm_jobs/{pmJobId} or GET {api_root}/vnfpm/v1/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/v1/pm_jobs/13963644-11b0-4302-a13b-26ca3d9eb8f8"
    },
    "objects": [
      {
        "href":
"http://host:port/vnflcm/v1/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/v1/pm_jobs or GET {api_root}/vnfpm/v1/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/v1/pm_jobs/13963644-11b0-4302-a13b-26ca3d9eb8f8"
    },
    "objects": [
      {
        "href":
"http://host:port/vnflcm/v1/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.

Delete a Performance Management Job

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

```
DELETE {api_root}/vnfpm/v1/pm_jobs/{pmJobId}
```

The NFVO is notified using the `PerformanceInformationAvailableNotification` notification.

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.

Method Type:

POST

VNFM Endpoint:

```
{api_root}/vnfpm/v1/thresholds (Datastructure=CreateThresholdRequest)
```

Request Payload:

```
{
  "objectInstanceId": "cc6a34e5-0463-459a-b367-493ba997775f",
  "criteria": {
    "performanceMetric": "default",
    "thresholdType": "SIMPLE",
    "simpleThresholdDetails": {
      "thresholdValue": 0.8,
      "hysteresis": 0.9
    }
  }
}
```

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/v1/thresholds/23f52511-9f72-4797-881b-c0f72e60a052"
    },
    "object": [
      {
        "href":
"http://host:port/vnflcm/v1/vnf_instances/cc6a34e5-0463-459a-b367-493ba997775f"
      }
    ]
  }
}
```

Query an Individual Threshold

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

GET

VNFM Endpoint:

```
{api_root}/vnfpm/v1/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/v1/thresholds/23f52511-9f72-4797-881b-c0f72e60a052"
    },
    "object": [
      {
        "href":
"http://host:port/vnflcm/v1/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/v1/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/v1/thresholds/23f52511-9f72-4797-881b-c0f72e60a052"
    },
    "object": [
      {
        "href":
"http://host:port/vnflcm/v1/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.

Delete a Threshold

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

```
DELETE {api_root}/vnfpm/v1/thresholds/{thresholdId}
```

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

Subscribing to Performance Management Job

This section describes subscribing to the performance management jobs.

Create a Performance Management Subscription

The NFVO can subscribe to the performance management jobs.

Method Type:

POST

VNFM Endpoint:

```
{api_root}/vnfpm/v1/subscriptions(DataStructure=PmSubscriptionRequest)
```

Example 1:

Request Payload:

```
{
          "callbackUri": "http://host:port/notification",
          "filter": {
                    "notificationTypes": ["ThresholdCrossedNotification",
"PerformanceInformationAvailableNotification"],
                    "vnfInstanceSubscriptionFilter": {
                          "vnfdIds": ["25ec9e1c-ad9e-4613-9280-411920f3649a"],
                           "vnfInstanceIds": ["cc6a34e5-0463-459a-b367-493ba997775f"]
                         }
           }
}
```

Response Payload:

```
{
       "id": "4fba7dcb-e015-4674-9c50-8cee7059eb91"
       "callbackUri": "http://host:port/notification",
           "filter": {
                              "notificationTypes": ["ThresholdCrossedNotification",
PerformanceInformationAvailableNotification"],
                               "vnfInstanceSubscriptionFilter": {
```

```
                                        "vnfdIds": ["25ec9e1c-ad9e-4613-9280-411920f3649a"],
                                         "vnfInstanceIds":
["cc6a34e5-0463-459a-b367-493ba997775f"]                }
          },
         "_links": {
                    "self": {
                                      "href":
"http://host:port/vnfpm/v1/subscriptions/4fba7dcb-e015-4674-9c50-8cee7059eb91"
                    }
           }
}
```

Example 2:

Request Payload:

```
{
      "callbackUri": "http://host:port/notification",
      "filter": {
                 "notificationTypes": ["ThresholdCrossedNotification",
"PerformanceInformationAvailableNotification"],
                  "vnfInstanceSubscriptionFilter": {
                                  "vnfProductsFromProviders": [{
                                                        "vnfProvider": "Cisco",
                                                         "vnfProducts": [{
                                                              "vnfProductName": "vnfd-1VDU",

                                                                          "versions": [{

"vnfSoftwareVersion": "1.3.1",
                                                                              "vnfdVersions":
 ["1.0", "1.1", "1.2"]
                                                                    }]
                                                    }]
                                     }],
                                "vnfInstanceNames":
["kaswaczy-TestETSIPmSubscriptionGet-114113"]
                   }
                }
}
```

Response Payload:

```
{
      "id": "4fba7dcb-e015-4674-9c50-8cee7059eb92"
       "callbackUri": "http://host:port/notification",
         "filter": {
                    "notificationTypes": ["ThresholdCrossedNotification",
"PerformanceInformationAvailableNotification"],
                      "vnfInstanceSubscriptionFilter": {
                      "vnfProductsFromProviders": [{
                                                        "vnfProvider": "Cisco",
                                                         "vnfProducts": [{
                                                                "vnfProductName":
 "vnfd-1VDU",
                                                                "versions": [{

"vnfSoftwareVersion": "1.3.1",

"vnfdVersions": ["1.0", 1.1", 1.2"]
                                                                    }]
                                                    }]
                                     }],
```

```
                                "vnfInstanceNames":
["kaswaczy-TestETSIPmSubscriptionGet-114113"]
                            }
             },
             "_links": {
                            "self": {
                               "href":
"http://host:port/vnfpm/v1/subscriptions/4fba7dcb-e015-4674-9c50-8cee7059eb92"
                            }
           }
}
```

Note

- The vnfdIds and vnfProductsFromProviders attributes are mutually exclusive. Only one of them can be provided in a single create request.
- The vnfInstanceIds and vnfInstanceNames attributes are mutually exclusive. Only one of them can be provided in a single create request.
- If the specified callbackUri and filter precisely matches an existing subscription, the create subscription operation will fail with an error message stating duplicate subscriptions are not allowed.

Query an Individual Performance Management Subscription

Method Type:

GET

VNFM Endpoint:

```
{api_root}/vnfpm/v1/subscriptions/{subscriptionId}
```

Request Payload: NA

Response Payload:

```
{
                "id": "4fba7dcb-e015-4674-9c50-8cee7059eb91"
                "callbackUri": "http://host:port/notification",
                "filter": {
                                "notificationTypes": ["ThresholdCrossedNotification",
"PerformanceInformationAvailableNotification"],
                                "vnfInstanceSubscriptionFilter": {
                                                "vnfdIds":
["25ec9e1c-ad9e-4613-9280-411920f3649a"],
                                                "vnfInstanceIds":
["cc6a34e5-0463-459a-b367-493ba997775f"]                }
                },
                "_links": {
                                "self": {
                                                "href":
"http://host:port/vnfpm/v1/subscriptions/4fba7dcb-e015-4674-9c50-8cee7059eb91"
                                }
                }
}
```

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

Query all Performance Management Subscriptions

Method Type:

GET

```
{api_root}/vnfpm/v1/subscriptions
```

Request Payload: NA

Response Payload:

```
{
                "_embedded": {
                                "pmSubscriptions": [{
                                                "id":
"4fba7dcb-e015-4674-9c50-8cee7059eb91""callbackUri": "http://host:port/notification",
                                                "filter": {
                                                                "notificationTypes":
["ThresholdCrossedNotification", "PerformanceInformationAvailableNotification"],

"vnfInstanceSubscriptionFilter": {
                                                                                "vnfdIds":
 ["25ec9e1c-ad9e-4613-9280-411920f3649a"],

"vnfInstanceIds": ["cc6a34e5-0463-459a-b367-493ba997775f"]
                                                                }
                                                },
                                                "_links": {
                                                                "self": {
                                                                                "href":
"http://host:port/vnfpm/v1/subscriptions/4fba7dcb-e015-4674-9c50-8cee7059eb91"
                                                                }
                                                }
                                },
                                {
                                                "id":
"4fba7dcb-e015-4674-9c50-8cee7059eb92""callbackUri": "http://host:port/notification",
                                                "filter": {
                                                                "notificationTypes":
["ThresholdCrossedNotification", "PerformanceInformationAvailableNotification"],

"vnfInstanceSubscriptionFilter": {

"vnfProductsFromProviders": [{

       "vnfProvider": "Cisco",

       "vnfProducts": [{

                       "vnfProductName": "vnfd-1VDU",

                       "versions": [{

                                       "vnfSoftwareVersion": "1.3.1",

                                       "vnfdVersions": ["1.0", "1.1", "1.2"]
```

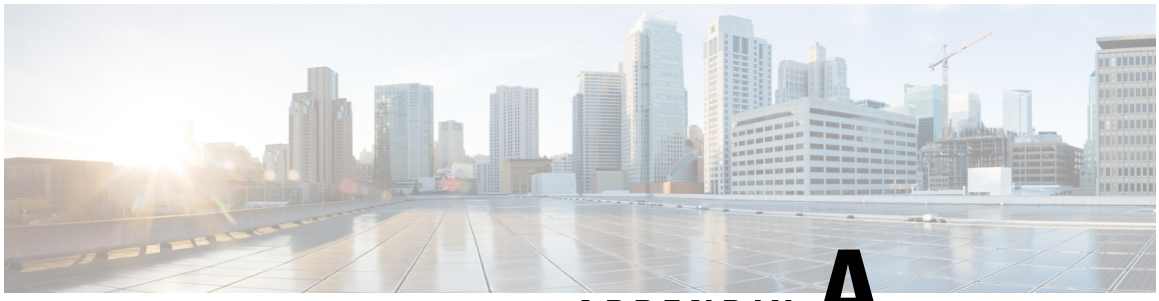
```
                        }]

        }]
                                                                                }],

"vnfInstanceNames": ["kaswaczy-TestETSIPmSubscriptionGet-114113"]
                                                                }
                                                    },
                                                    "_links": {
                                                                "self": {
                                                                                "href":
"http://host:port/vnfpm/v1/subscriptions/4fba7dcb-e015-4674-9c50-8cee7059eb92"
                                                                }
                                                    }
                                    }]
                    }
}
```

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

Terminate a Performance Management Subscription

The NFVO can terminate a subscription.

```
DELETE {api_root}/vnfpm/v1/subscriptions/{subscriptionId}
```



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 5: 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`
`server.port`	The port used to communicate over HTTP.	Integer	`8250`

Property Name	Description	Type	Default Value
`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. Allows for looser validation, 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. Valid values: http, 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`
`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`

Property Name	Description	Type	Default Value
`nfvo.httpScheme`	The HTTP scheme used for communicating with the NFVO. Valid values: `http, 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. Valid values: `http, 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`
`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	

Property Name	Description	Type	Default Value
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
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

Property Name	Description	Type	Default Value
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

For information on resource definitions, see Resource Definitions for ETSI API, on page 3.

ETSI Production Properties