Managing VNF Lifecycle Using ETSI API

- Managing VNF Lifecycle Using ETSI API, on page 1
- VNF Lifecycle Operations Using ETSI API, on page 2

Managing VNF Lifecycle Using ETSI API

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.

VNF Lifecycle Operations Using ETSI API

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.
- The VIM connector specifies the VIM used. The VIM connector information must be available before instantiating the VNF. ESC must contain a valid, and authenticated default OpenStack VIM connector to deploy the VNFs.
- 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 ETSI MANO NFV specifications supports /vnf_packages API to allow access to the package artifacts.See chapter 10 in the *ETSI GS NFV-SOL 003 V2.4.1* specification on the ETSI website.

  The properties file provides details about the NFVO to the ETSI VNFM service.

  The properties file, *etsi-production.properties* exists under: `/opt/cisco/esc/esc_database/`

  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.

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)

```
{
    "_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": "",
    "vnfSoftwareVersion": "not-implemented",
    "vnfdId": "vnfd-88c6a03e-019f-4525-ae63-de58ee89db74",
    "vnfdVersion": ""
}
```

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 instatiation 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 Grant Permission.

Typically, the request permission, and retrieve messages are customized for the NFVO.

Method type:

```
POST
```

VNFM Endpoint:

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

HTTP Request Header:

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

Request Payload (ETSI data structure: InstantiateVnfRequest)

```
{
    "flavourId": "vdu_node1",
    "extManagedVirtualLinks": [
        {
           "id": "my-network",
            "resourceId": "RES1",
            "virtualLinkDescId": "VLD1"
        }
    ],
    "vimConnectionInfo": [
        {
            "accessInfo": {
                "resourceGroupId": "tenantName"
            },
            "id": "vimConnId",
            "vimType": "Openstack"
        }
    ]
}
```

The flavourId value must be same as the flavour_id specified in the VNFD file under properties.

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: DENY
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/457736f0-c877-4e07-8055-39dd406c616b
Content-Length: 0
Date: Thu, 04 Jan 2018 12:20:40 GMT
```

Response Body:

```
not applicable.
```

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

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>.

Method type:

```
POST
```

VNFM Endpoint:

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

HTTP Request Header:

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

Request Payload (ETSI data structure: InstantiateVnfRequest)

```
{
    "flavourId": "default",
    "additionalParams": {
        "CPUS": 2,
        "MEM_SIZE": "512 MB"
    }
}'
```

The *additionalParams* variables are merged with the VNF variables, and actual values for the variables are provided only during instantiation.

```
node_templates:
    my_server:
      type: tosca.nodes.Compute
      capabilities:
        host:
          properties:
            num_cpus: { get_input CPUS }
            mem_size: { get_input MEM_SIZE }
            disk_size: 10 GB
```

```
topology_template:
inputs:
CPUS
description: Number of CPUs
MEM_SIZE
description: Memory 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.

Note The additional variables must be initially defined in the VNFD template, so that it can be used during instantiation.

ESC supports internal virtual links while instantiating a VNF using the TOSCA VNFD template.

Example for Virtual Internal Links from a TOSCA VNFD template:

```
   esc-test-net:
      type: tosca.nodes.nfv.VnfVirtualLinkDesc
      properties:
        connectivity_type:
          layer_protocol: ipv4
        description: VDU Internal Network VL

    EtsiTestInternalVL-Net-1:
      type: tosca.nodes.nfv.VnfVirtualLinkDesc
      properties:
        connectivity_type:
          layer_protocol: ipv4
        description: VDU Internal Network VL
        externally_managed: false
```

Requesting Grant Permission

The ETSI API requests for permission from the NFVO for lifecycle management operations to complete the operations in progress for the VNF instance resources.

```
{
    "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"
        }
    }
}
```

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

NA

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

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 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 **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).

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: NA

Response Body: NA

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.
```

Scaling VNFs Using ETSI API

ESC can scale VMs using the ETSI API. The scaling workflow begins when the VNF instance is in the instantiated state. The scaling details are defined in the VNFD by the NFVO.

Scaling of a VNF instance can be achieved in two ways:

- Scale
- Scale to level

As part of scaling,

- Add VM instances to the VNF to scale out.
- Remove VM instances from the VNF to scale in.

Scale

ESC can scale a VNF incrementally. The *numberOfSteps* attribute determines the increment by which you can scale the VNF instance. By default, the number of steps is set to 1.

Method type:

```
POST
```

VNFM Endpoint:

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

HTTP Request Header:

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

Request Payload (ETSI data structure: ScaleVnfRequest)

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

Response Headers:

```
not applicable.
```

Response Body:

```
not applicable.
```

Scale to Level

Allows ESC to scale the VNF to a specified level.

Method type:

```
POST
```

VNFM Endpoint:

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

HTTP Request Header:

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

Request Payload (ETSI data structure: ScaleVnfRequest)

```
{
/* "instantiationLevelId":"id111", */
"scaleInfo": [
{ "aspectId":"processing", "scaleLevel":"3" },
{ "aspectId":"database", "scaleLevel":"2" }
```

Response Headers:

```
not applicable.
```

Response Body:

```
not applicable.
```

Auto-Scaling VNFs

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 inputs 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 overloaded or underloaded, ESC gets a notification to determine if the scaling is automatically enabled. If the scaling is not enabled, ESC generates a notification towards the ETSI API component to check the VNF's state.

  For more information, see ETSI Failure and Load Notifications for VNFs section.
- **VNFD**

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

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

If the validation passes, you can request to start the operation flow to generate the appropriate operation occurrence and associated notifications. For scaling, specified KPI data determines the scaling parameters.

- **VnfInstance Resource**

  The VNF Instance determines the scale level using the current *scaleStatus*. The processing of the request determines the number of VMs to request from ESC. 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.

Healing VNFs Using ETSI API

As part of the VNF lifecycle management, ESC heals the VNFs when there is a failure. ETSI API heals a VNF instance based on the instructions from the NFVO.

Method Type:

```
POST
```

VNFM Endpoint:

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

HTTP Request Headers:

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

Request Payload (ETSI data structure: HealVnfRequest):

```
{
"cause":"b9909dde-e21e-45ec-9cc0-9e9ae413eee0",
}
```

Response Headers:

```
not applicable.
```

Response Body:

```
not applicable.
```

Manual Recovery of a Deployment

Currently, ETSI supports recovery of a complete deployment.

Request Payload (ETSI data structure: HealVNFRequest)

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

The previous example shows a possible use for the additional parameters where it may be used, in future, to specify an overriding recovery policy, regardless of the policy configured at time of deployment. Both attributes are optional, as per the specification.

Table 1: VM and Deployment States

VM state	Deployment or service state	recovery-vm-action
error	error	supported
error	active	supported

Auto-Healing VNFs

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

- **VM Failure Notification**

  When a new ESC notification is processed, you must create the corresponding alarm resource, either by accessing the distributed alarm object directly, or using the extension API to create the alarm resource. For more information, see ETSI Failure and Load Notifications for VNFs section.

- **VNFD**

  The VNFD contains the instructions for the recovery action/time required for the *isAutohealEnabled* configurable property of the VNF.

  For more information on Manual recovery, see Manual Recovery of a Deployment section.

- **VnfInstance Resource**

  You must validate a heal payload for a VNF instance resource.

  You must validate the following:

  - Ensure that the VnfInstance state is set to INSTANTIATED

  For more information on the REST API to heal a complete deployment, see Healing VNFs Using ETSI API section.

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 Functions

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
HTTP/1.1 204
< 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: TEST
X-Frame-Options: TEST
< 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
< Date: Thu, 04 Jan 2018 12:48:59 GMT
Date: Thu, 04 Jan 2018 12:48:59 GMT
```

Response Body:

```
not applicable.
```