



ETSI MANO Northbound API Overview

- [ETSI MANO Northbound API Overview, on page 1](#)
- [Resource Definitions for ETSI API, on page 4](#)

ETSI MANO Northbound API Overview

The ETSI 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). The API contains its own datamodel designed around the ETSI MANO specifications that abstract away from the ESC core datamodel.

Table 1: 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

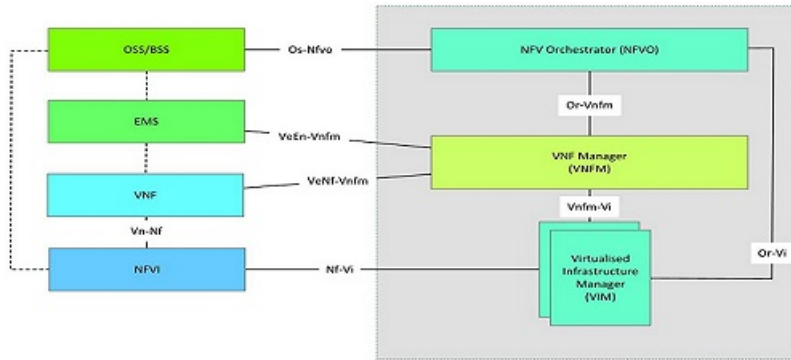
The initial implementation of ETSI standard supports ETSI MANO API over Or-Vnfm reference point, which is the interface between ESC and NFVO. The Or-Vnfm reference point details the interactions to onboard ETSI compliant VNF packages, manage resources, and VNF lifecycle management (LCM) operations.



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.

Figure 1: NFV MANO Architecture with Reference Points



Production Properties

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 2: 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
server.port.https	The port used to communicate over HTTPS.	Integer	8251

Property Name	Description	Type	Default Value
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 <code>sudo escadm etsi set --rest_user <username>:<password></code> and should be synchronized here.	String	
nfvo.apiRoot	Mandatory. The apiRoot for the NFVO.	String	localhost:8280
nfvo.httpScheme	The HTTP scheme used for communicating with the NFVO. Valid values: http, https.	Enum	http

Property Name	Description	Type	Default Value
<code>nfvo.isPackageNotificationSupported</code>	Determine if the VNFM will attempt to subscribe to package notifications.	Boolean	false
<code>nfvo.callback.httpScheme</code>	The HTTP scheme used for communicating with the NFVO when polling for responses. Valid values: <code>http</code> , <code>https</code> .	Enum	<code>https</code>
<code>nfvo.username</code>	The username for NFVO credentials.	String	
<code>nfvo.password</code>	The password for NFVO credentials, required in plain text.	String	
<code>retryTemplate.exponential.retryPolicy.maxAttempt</code>	The number of retries for the exponential retry mechanism.	Integer	1000
<code>retryTemplate.exponential.backOffPolicy.interval.initial</code>	The starting interval for the exponential retry mechanism.	Integer	1000
<code>retry.simple.maxAttempt</code>	The number of retries for the simple retry mechanism.	Integer	50
<code>retry.simple.backOff.delay</code>	The interval for the simple retry mechanism.	Integer	1000

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](#).

To access ETSI MANO API documentation directly from the ESC VM, see [ETSI MANO Northbound API](#).

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	<p>Out of band tenants</p> <p>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 Using ETSI API.</p>
Images	<p>Out of band images</p> <p>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.</p>
Flavors	<p>Out of band flavors</p> <p>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 overridden by a VIM flavor supplied at instantiation time as an additional parameter.</p> <p>Note ETSI deployment flavor is a different concept than OpenStack compute flavour. For more information, see <i>Terms and Definitions</i> in Preface.</p>
Volumes	ETSI does not support Out of band volumes.
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 MANO API, see [VNF Lifecycle Operations Using ETSI API](#).

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: [ vml_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-e18771dacflc",
      "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-e18771dacflc* in the above example, must also exist in the instantiatedVnfInof in the vnflInstance.

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.

