# ETSI MANO Northbound API Overview



# ETSI MANO Northbound API Overview

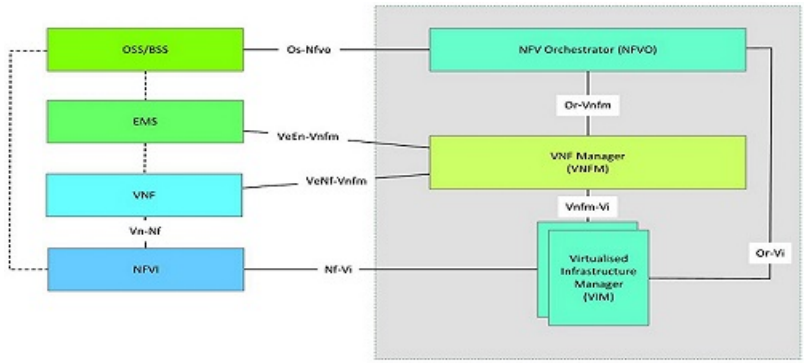
The ETSI MANO API is another programmatic interface to ESC that uses the REST architecture. The ETSI MANO adheres to the standards defined by the European Telecommunications Standards Institute (ETSI), specifically around Management and Orchestration (MANO). The API accepts and returns HTTP messages that contain JavaScript Object Notation (JSON). The API contains its own datamodel designed around the ETSI MANO specifications (*ETSI GS NFV-SOL 003 V2.4.1*) that abstract away from the ESC core datamodel.

The initial implementation of ETSI standard supports ETSI MANO API over Or-Vnfm reference point, which is the interface between ESC and NFV MANO. 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***



# 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.

**Note** Virtual network function lifecycle operation using ETSI MANO API is supported on OpenStack only.

***Table 1: Resource Definitions on VIM***

| Resource Definitions | OpenStack |
|---|---|
| Tenants | Out of band tenants<br>You can create a tenant using NETCONF API, REST API, or the ESC portal. You can also create a tenant directly on the VIM. The tenant is then referred to as the resourceGroupId. |
| Images | Out of band images<br>The NFVO onboards a VNF package and then onboards the image contained within the VNF package on to the VIM. This can then be referenced in the sw_image attribute. |
| Flavours | Out of band flavours<br>During onboarding of the VNF Package, the NFVO looks at the tosca.nodes.nfv.VNF.CiscoESC node in the VNFD to determine the flavour to be created. This is available later at the time of instantiation.<br>**Note** ETSI deployment flavour is a different concept than OpenStack compute flavor. For more information, see *Terms and Definitions* in Preface. |
| Volumes | Out of band volumes<br>The out of band volume requirements are detailed in the tosca.nodes.nfv.VDU.Compute node in the VNFD. |
| External Networks (Virtual Link) | Out of band networks |

| Resource Definitions | OpenStack |
|---|---|
| Externally Managed Internal Virtual Links | The externally managed internal virtual links are defined in the VNFD, or referred to if the resource is out of band. |
| 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: [ 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.