Managing Resources

Managing Resources

Resource Definitions for ETSI API

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

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

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

ETSI API Documentation

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

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

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

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

Table 1: 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.
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 in-band volumes of type VirtualBlockStorage, as required by a deployment. It also supports out-of-band volumes as a Cisco extension to the ETSI specification.
External Networks (Virtual Link)	External networks are specified in the instantiation payload to which external connection points will connect.
Externally Managed Internal Virtual Links	Networks internal to the VNF are supported, as well as external networks specified in the instantiation payload to which internal virtual links will be bound instead of creating ephemeral networks.
Subnetworks	Out-of-band subnets

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

Updating Resource Definitions

This section provides details about updating ETSI API resource definitions.

Updating the VNF Flavour

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

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

Example1:

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

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

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

```
tosca_definitions_version: tosca_simple_yaml_1_3
 description: Substitution Mapping Example

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

. . .

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

. . .

topology_template:

. . .

########################
# Substitution Mapping #
########################
substitution_mappings:
node_type: my-vnf
requirements:
# 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_3*

description: Silver Deployment Flavour

imports:

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

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

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

```
            scaling_aspect:
              - default_scaling_aspect
            cisco_esc_properties:

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

ESC sends a POST request to update the VNF flavour:

Method Type:

POST

VNFM Endpoint:

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

Updating the External VNF Connectivity

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

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

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

Method Type

POST

VNFM Endpoint

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

Request Payload (Data structure = ChangeExtVnfConnectivityRequest)

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

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

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

Merging Policy

The substitution merges the new values into the VNFD.

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

Parser Behaviour

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

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

OAuth (Open Authorization) 2.0 Authentication

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

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

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

Method Type

POST

URL

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

Header

```
Authorization: Basic {base 64 encoded CLIENT_ID:CLIENT_SECRET}
```

```
Accept: application/json
Content-Type: application/x-www-form-urlencoded
```

Body

```
grant_type=client_credentials
```

ESC returns the access token in response.

Example:

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


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

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

Example:

Method

GET

URL

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

Headers

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

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

Accessing and Updating the OAuth Properties File

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

To add or update a client id

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

To remove a client id

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

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

For information on other properties, see ETSI Production Properties.

OAuth Calls from ETSI to the NFVO

ESC supports OAUTH 2.0 calls from ETSI to the NFVO.

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

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

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

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

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

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

OAuth 2.0 Notification and Subscription

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

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