リソースの管理

リソースの管理

ETSI API のリソース定義

Cisco Elastic Services Controller（ESC）リソースは、イメージ、フレーバ、テナント、ボリューム、ネットワーク、およびサブネットワークで構成されます。これらのリソースは、ESC が仮想ネットワーク機能のプロビジョニングを要求するためのものです。

ETSI MANO の場合、これらのリソース定義は、VNF パッケージのオンボーディング時またはテナントのオンボーディング時に NFVO によって作成され、ESC への要求の VIM ID によって表されます。

NETCONF または REST API を使用したリソースの管理については、『Cisco Elastic Services Controller User Guide』の「Managing Resources Overview」を参照してください。

ETSI API ドキュメント

ETSI API ドキュメントには、ESC VM から直接アクセスできます。

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

ETSI API ドキュメントには、ESTI MANO インターフェイスでサポートされるさまざまな操作の詳細が記載されています。詳細については、『Cisco ETSI API Guide』も参照してください。

次の表に、VNF のインスタンス化の前に使用可能にする必要がある VIM のリソース定義を示します。

表 *1: VIM* でのリソース定義

リソースの定義	OpenStack
テナント	アウトオブバンドテナント NETCONF API、REST API、または ESC ポータルを使用してテナントを作成できます。テナントを VIM で直接作成することもできます。その後、テナントは vimConnectionInfo データ構造内で参照されます。詳細については、VIM コネクタの概要を参照してください。
イメージ	アウトオブバンドイメージ NFVO は VNF パッケージをオンボードし、VNF パッケージに含まれるイメージを抽出して VIM にオンボードします。VNFD はイメージファイルを参照しますが、イメージファイルのサイズを理由に、展開時にイメージをオンボーディングする代わりに、Grant の vimAssets が使用するイメージを指定します。
フレーバ	アウトオブバンドフレーバ VNF パッケージのオンボーディング中、NFVO は VNFD 内の各 cisco.nodes.nfv.Vdu.Compute node の機能を調べて、作成するフレーバを決定します。これはインスタンス化された後で使用できます。またはオプションで、インスタンス化したときに追加パラメータとして指定された VIM フレーバによってオーバーライドされます。 （注） ETSI 展開フレーバは、OpenStack コンピューティングフレーバとは異なる概念です。詳細については、このガイドの「用語および定義」を参照してください。
ボリューム	ESC は、ETSI のシスコの拡張仕様としてアウトオブバンドボリュームをサポートします。
外部ネットワーク（仮想リンク）	外部ネットワークは、外部接続ポイントが接続するインスタンス化ペイロードで指定されます。

リソースの定義	OpenStack
外部管理の内部仮想リンク	エフェメラルネットワークを作成する代わりに、内部仮想リンクがバインドされるインスタンス化ペイロードで指定された外部ネットワーク。
サブネットワーク	アウトオブバンドサブネット

ETSI API を使用した VNF パッケージのオンボーディングとライフサイクル操作の詳細については、VNF ライフサイクルの管理を参照してください。

リソース定義の更新

このセクションでは、ETSI API リソース定義の更新について詳しく説明します。

VNF フレーバの更新

次の TOSCA パラメータを使用して、1 つの VNFD の代替 VNF ノードと展開フレーバを定義できます。

- **Import statements**：import statement により、実行時に動的に指定できる入力値に基づいて、1 つの親の VNFD yaml ファイルに他のファイルを条件付きで含めることができます。
- **Substitution mappings**：substitution mapping は *tosca.nodes.nfv.VNF* から派生したノードタイプにのみ適用されます。他のノードタイプの値（接続ポイント、仮想リンクなど）を置き換えることはできません。

例 1：

この例では、yaml ファイルに 3 つのインポートファイルが含まれています。

3 つのファイルはすべて、インポートする親ファイルと同じ場所にある VNFD ZIP アーカイブファイルに存在する必要があります。

要件と機能は、派生した *tosca.nodes.nfv.VNF* ノードで定義されません。これらは、この VNFD を使用してインスタンス化された VNF の特性を定義するために必須です。これらはインポートされたファイル内で定義されます。

```
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
```

例 2：

VNF がインスタンス化されると、必要なフレーバがインスタンス化要求で VNFM に送信されます。TOSCA パーサーが、フレーバおよび VNF ノード名と、定義された substitution mappings との照合を試みます。これらは、VNFD 自体内でインポートまたは定義できます。たとえば、*df_silver.yaml* には次の内容が含まれています。

tosca_definitions_version: tosca_simple_yaml_1_2

説明：Silver 展開フレーバ

インポート：

```
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 は、インスタンス化要求ペイロードで渡される flavourId です。上記の親の *yaml* には、silver プロファイルからの要件で更新された空の要件セクションがあり、既存の *flavour_id* プロパティと *flavour_description* プロパティも更新されます。

```
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 は POST 要求を送信して VNF フレーバを更新します。

メソッドタイプ：

POST

VNFM エンドポイント：

```
/vnflcm/v1/vnfinstances/{vnfInstanceId}/change_flavour
```

外部 VNF 接続の更新

既存の展開で外部 VNF 接続を更新できます。API は次の変更をサポートします。

- 既存の外部仮想リンクへの既存の接続ポイント（CP）を切断し、別の仮想リンクに接続します。
- アドレスの変更を含め、既存の外部 CP の接続パラメータを変更します。

ESC は VNF 外部接続を更新するための POST 要求を送信します。

メソッドタイプ

POST

VNFM エンドポイント

```
/vnflcm/v1/vnfinstances/{vnfInstanceId}/change_ext_conn
```

要求ペイロード（データ構造 = 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"
                      }
                    ]
                  }
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

（注） extVirtualLinks の ID（上記の例では *extVL-98345443-7797-4c6d-a0ed-e18771dacf1c*）は、vnfInstance の InstantiatedVnfInof にも存在する必要があります。

マージポリシー

置換により、新しい値が VNFD にマージされます。

1. name=joe などの通常のスカラプロパティの場合、値は VNFD で置き換えられます。
2. [list, of, strings] などの配列はマージされます。新しい値が存在しない場合は、配列に追加されます。
3. キーが別のキーの下にインデントされているなどのオブジェクトは置き換えられます。一致した置換の configurable_properties オブジェクトは、VNFD で定義されたものを上書きします。

パーサーの動作

- substitution mappings が作成された後、パーサーは提供された *additionalParams* を事前入力しようとします。入力パラメータがテンプレートのパラメータと一致しない場合、コマンドは失敗します。

VNF ライフサイクル操作の詳細については、VNF ライフサイクルの管理を参照してください。

OAuth（Open Authorization）2.0 認証

ETSI NFV MANO は、SOL003 Or-Vnfm リファレンスポイントの OAuth 2.0 認証をサポートします。NFVO は、認証用のクライアント ID やクライアントシークレットなどのクライアントログイン情報を提供する ESC にトークン要求を行います。次に、ESC は要求を確認し、アクセストークンを返します。

NFVO は、プライマリ認証として clientId と secret を提供する POST 要求を行います。

メソッドタイプ

POST

URL

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

ヘッダー

```
Authorization: Basic {base 64 encoded CLIENT_ID:CLIENT_SECRET}
Accept: application/json
Content-Type: application/x-www-form-urlencoded
```

本文

```
grant_type=client_credentials
```

ESC は応答でアクセストークンを返します。

例：

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


Q.lAtre7vdCKJjgzNs7p9P3NS2qMcXegC-oWXmy5Kakn0AL95gLWF6liOqPViMZNnWZLOsG5r1kPnGoBWnN0tgIw",

    "token_type": "bearer",
    "expires_in": 600
}
```

次に、アクセストークンは or_vnfm エンドポイントにアクセスするために使用されます。

例：

方法

GET

URL

```
{apiRoot}/vnflcm/v1/subscriptions
```

ヘッダー

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

（注） ETSI サービスが再起動されると、既存のトークンは無効になります。

OAuth プロパティファイルへのアクセスと更新

ESC は、*etsi-production.properties* ファイルと同じ場所にある新しい *etsi-production.yaml* プロパティファイルにクライアント ID とシークレットを保存します。クライアント ID とシークレット値を管理するために、新しい escadm etsi コマンドを使用できます。クライアントシークレットは、既存の REST ユーザ名と同じ方法で暗号化されます。

クライアント ID を追加または更新するには

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

クライアント ID を削除するには

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

（注） OAuth 2.0 値を更新した後、ETSI サービスを再起動します。

その他のプロパティの情報については、ETSI 製品のプロパティを参照してください。

ETSI から NFVO への OAuth コール

ESCは、ETSI から NFVO への OAUTH 2.0 コールをサポートします。

次のプロパティが etsi-product.properties ファイルに追加されます。

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

クライアント ID、ClientSecret、および TokenEndpoint は、OAUTH 2.0 サーバのものと一致する必要があります。認証タイプは、ESC から NFVO への発信コールの認証を決定します。認証タイプは、BASIC または OAUTH2 のいずれかである必要があります。

NFVO からのトークンは、プロパティファイルのトークンエンドポイントに保存されます。

NFVO がコール要求を送信すると、ETSI はトークンエンドポイントに保存されているトークンをチェックします。トークンの有効期限が切れていない場合、ETSI は古いトークンを要求のヘッダーに追加し、コールを実行します。トークンの実行に失敗した場合は、新しいトークンが必要です。

トークンエンドポイントに対してトークンがない場合は、コールを実行するための新しいトークンが必要です。

OAuth 2.0 通知およびサブスクリプション

通知で OAuth 2.0 認証を有効にするには、サブスクリプション ペイロードに以下を追加する必要があります。

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