Developing Device Scripts

- About Device Scripts, page 1
- Guidelines for Creating Device Scripts, page 2
- Sample Script, page 23

About Device Scripts

The device script acts as an adapter between the Application Policy Infrastructure Controller (APIC) and the network service by converting calls to the APIC service API into device-specific calls.

Figure 1: Device Script Model

The device script runs in the context of a `ScriptWrapper`, which is an environment that handles calls for each device type. A `ScriptWrapper` runs within a namespace that limits CPU, file, and socket resource consumption.

Uploading a device package creates a `ScriptWrapper` that imports the module from the device script file. The module exposes the functions described in the API.

The device scripts must be stateless and idempotent (producing the same result if run more than once). No file I/O operations are permitted within a script, except for generating a temporary state in the `/tmp` directory. You should not use any file that is created within the `/tmp` directory for storing a persistent state. The APIC can be deployed in a cluster. The device script can get invoked from any one of the APIC instances. Any data stored in the `/tmp` directory is not guaranteed to be available across two API calls. A script must not store its own state in any file.

The APIC requires scripts to be developed for Python 2.7. Other than standard libraries that are available in Python 2.7, the script environment provides Python Requests library v1.2.3. If the device package requires any other libraries, the device package developer can bundle those libraries in the device script's `zip` file.

Note The script must be thread-safe, which means that for any instance, multiple threads can invoke the same function in the script in order to configure different devices. The script should execute in the invoking thread context and must not spawn any new threads as part of its execution.

Guidelines for Creating Device Scripts

You must implement all the scripting APIs to establish the adapters between the Application Policy Infrastructure Controller (APIC) and the network services. The APIs receive the Python dictionaries that correspond to the device specification hierarchy. You must convert the Python dictionaries to the internal format that is needed by the specific device. Similarly, when a device returns a value, the API must convert the return value to the format needed by the APIC. For examples, see the specification in Developing Device Specifications and the sample script at the end of this section.

Device Script APIs

The device script APIs are divided into four categories:

- Device
- Cluster
- Service
- Endpoint and Network Event

The Application Policy Infrastructure Controller (APIC) requires users to register one or more device cluster within a tenant. All service functions are applied to a cluster. A cluster can contain one or more network service devices. A device can be deployed in standalone mode without any redundancy by defining a cluster with a single device. A device can be deployed in active-standby HA mode by registering two devices configured as active-standby peers within a cluster. Similarly devices can be deployed in active-active mode by registering multiple devices configured as active peers within a cluster. The devices registered within a cluster are assumed to have active-active or active-standby pairing. The HA configuration of devices within a cluster can be pushed by way of APIC or it could be done out-of-band directly on the device prior to registering the devices with the APIC.

The configuration on the device is split into three categories:

- Service function specific configuration
- Device specific configuration
- Cluster specific configuration

The service configuration is pushed by way of the service APIs, the device configuration is pushed by way of the device APIs, and the cluster configuration is pushed by way of the cluster APIs.

Device APIs

The following APIs are called for each device registered within a cluster with APIC:

```
def deviceValidate( device, version )
def deviceModify( device, interfaces, configuration)
def deviceAudit( device, interfaces, configuration )
def deviceHealth( device, interfaces, configuration )
def deviceCounters( device, interfaces, configuration )
```

The configuration dictionary passed in these APIs contains any device-specific configuration that is done on the APIC.

Note The APIC does not pass any cluster-level or service function configuration information during device API callouts.

The device APIs are assumed to act on any device specific configuration and should not reference or affect cluster-level configuration or affect service functions.

Typically, configurations that are device-specific, such as link bundling (LACP), can be done in the deviceModify() and deviceAudit() callouts.

The configuration passed in the dictionary will be an instance of any folder and parameter that is defined under `vnsDevCfg` in the device Model.

Cluster APIs

The following APIs are called for each device cluster that is registered with the APIC:

```
def clusterModify( device, interfaces, configuration )
def clusterAudit( device, interfaces, configuration )
```

The configuration dictionary contains any cluster configuration that is done on the APIC.

Note The APIC does not pass any device configuration or service configuration information during cluster API callouts.

The cluster APIs are assumed to act on any cluster-level configuration and should not reference or affect device-specific of function specific configuration.

The cluster APIs are assumed to act on any cluster-level configuration. A cluster API should not reference nor change a device-specific or function-specific configuration.

A cluster-level configuration, such as for an NTP server or a global syslog server, should be done through the cluster API.

The configuration passed in the dictionary is an instance of any folder and parameter defined directly under `vnsClusterCfg` in the device Model.

Service APIs

The following APIs are called for any service function that is rendered on the device:

```
def serviceModify( device, configuration )
def serviceAudit( device, configuration )
def serviceHealth( device, configuration )
def serviceCounters( device, configuration )
```

The configuration dictionary contains an instance of parameters and folders that are defined under `vnsMDevCfg`, `vnsGrpCfg`, or `vnsMFunc`.

Note An instance of folders or parameters defined under `vnsMDevCfg` within a device model is passed in a service API callout if and only if an instance of a function in the service graph has a reference to these parameters. Not all parameter and folder instances defined under `vnsMDevCfg` are passed to the service function. The APIC passes only those parameters and folders that are used by a specific function instance that is being referenced in the service API callout.

Endpoint and Network Event

The APIC provides device script with information about the endpoints within an endpoint group (EPG) and the subnet associated with the endpoint group. This information is passed through explicit API calls, as shown in the following example:

```
def attachEndpoint(device, configuration, endpoint)
def detachEndpoint(device, configuration, endpoint)
def attachNetwork(device, configuration, network)
def detachNetwork(device, configuration, network)
```

With controller version 1.1 and later, the APIC also conveys the endpoint and network information within the configuration dictionary that is passed in the serviceModify() API and serviceAudit() API.

The service functions defined by a device package have two connectors. A user can define a service graph by attaching one or more service functions through the connectors. A service graph can be deployed by associating the graph to a pair of EPGs. The connectors of end node functions in the graph are attached to one of the EPGs. Once a service graph is deployed between a pair of EPGs, each function connector within the graph is associated to one of the EPGs. For each function connector within the graph, the APIC provides endpoint and network information for the EPG associated with the connector. The endpoint and network information is conveyed to the device script only after the following criteria are satisified:

1. The device model indicates that notification is supported by the device script for the function connector.

   ```
   <vnsMConn ... epNotifications="endpoint">
   ```

   or

   ```
   <vnsMConn ... epNotifications="endpoint, subnet">
   ```

Note If a device package uses the epNotifications attribute, the APIC supports notification of both the subnet and endpoint on the same connector. If the device package uses the notifications attribute, the APIC does not support notifications for both the endpoint and subnet on same connector.

2. The user enables notification on the connector while defining a service graph template.

Endpoint and network information allows scripts to modify a configuration dynamically on the service device. An example use case for using the attachEndpoint() and detachEndpoint() API calls for notification is with a load balancing function so that the function can dynamically update servers within a pool by using endpoint information. An example use case for subnet for notification is with a firewall so that the firewall can dynamically add access rules for subnets that are associated with an EPG. Such dynamic updating of the configuration can eliminate user error and provide automation through the APIC.

Script Framework

The following two modules must be imported by a device script:

- Import Insieme.Logger
- Import Insieme.Config

Logging

The `Insieme.Logger` module defines a logging utility. A device script can use this utility to log debug information. The logging utility writes the configuration API logs to a file called `debug.log`. This file is included in any technical support data that is collected from the Application Policy Infrastructure Controller (APIC). A device script developer should log as much information as possible to help debug any script issues.

Logs for periodic APIs, such as `serviceHealth`() and `serviceCounters`(), are redirected to the `periodic.log`. The `debug.log` and `periodic.log` files can be accessed as a fabric administrator on the APIC under the `/data/devicescript/`*`vendor_model_version`*`/logs` directory.

The logging function is similar to the Python logging function. The logs can be split into the following categories:

- CRIT
- ERROR
- WARN
- INFO
- DEBUG
- DEBUG2
- DEBUG3
- DEBUG4

The script can invoke the API as follows:

```
Logger.log( level,  Log String)
```

The following example invokes the API:

```
Logger.log( Logger.DEBUG, 'Connection to device failed')
```

Constants

The `Insieme.Config` module defines constants that can be used for parsing the dictionary:

```
Type = Insieme.Fwk.Enum(
    DEV=0,
    GRP=1,
    CONN=2,
    FUNC=3,
    FOLDER=4,
    PARAM=5,
    RELATION=6,
    ENCAP=7,
    ENCAPASS=8,
    ENCAPREL=9,
    VIF=10,
```

```
    CIF=11,
    LIF=12,
    OSPFDEV=13,
    OSPFFUNC=14,
    OSPFCONN=15,
    OSPFVENCAPASC=16,
    BGPDEV=17,
    BGPFUNC=18,
    BGPCONN=19,
    BGPVENCAPASC=20,
    DEVMGR=21,
    ENDPOINT=22,
    SUBNET=23,
)

State = Insieme.Fwk.Enum(
    UNCHANGED=0,
    NEW=1,
    CHANGED=2,
    DELETED=3,
)

Result = Insieme.Fwk.Enum(
    SUCCESS=0,
    TRANSIENT=1,
    PERMANENT=2,
    AUDIT=3,
)
```

Note Device package developers should use the `enums` defined in the `Insieme.Fwk` Python module. The integer values shown above might have changed in the final implementation. Using the absolute values shown in this document can lead to unexpected behavior.

Configuration Dictionary Format

The configuration dictionary that is passed in the cluster API, device API, and service API follows the same structure as defined in the device specification file. The configuration is passed as a hierarchy of dictionaries, with each level identifying a folder. The dictionary format is as follows:

```
(type,  key, name) : { 'state': ...
                       'transaction': ...
                       'connector': ...
                       'value': ...
                       'target': ...
                       'device': ...
                      }
```

The fields are as follows:

Field	Description

type	Identifies the type of the object represented by the dictionary. The field can have one of the following values:
	`DEV=0,`
	`GRP=1,`
	`CONN=2,`
	`FUNC=3,`
	`FOLDER=4,`
	`PARAM=5,`
	`RELATION=6,`
	`ENCAP=7,`
	`ENCAPASS=8,`
	`ENCAPREL=9,`
	`VIF=10,`
	`CIF=11,`
	`LIF=12,`
	`OSPFDEV=13,`
	`OSPFFUNC=14,`
	`OSPFCONN=15,`
	`OSPFVENCAPASC=16,`
	`BGPDEV=17,`
	`BGPFUNC=18,`
	`BGPCONN=19,`
	`BGPVENCAPASC=20,`
	`DEVMGR=21,`
	`ENDPOINT=22,`
	`SUBNET=23,`

Key	Specifies the key or name attribute that is defined in the device specification file for the object.
Name	Specifies the parameter or folder instance name that is provided by the user.

State	Identifies the object's state. This field can have one of the following values:
	`UNCHANGED=0,`
	`NEW=1,`
	`CHANGED=2,`
	`DELETED=3,`

Connector	Specifies the name of the connect instance that is resolved according to the relations that are defined the specification file. This field is populated for a folder or a relation dictionary only if the corresponding `vnsMFolder` object or `vnsMRel` object has `vnsRsConnector` relations defined in the device specification file.
Value	Defines the value for the object. In the case of a folder, this field can contain another dictionary. A relations object does not contain a value element, and instead has a target element. A value for a parameter object cannot exceed 512 characters.

Field	Description
Target	Defines the target folder to which a relations object is resolved. This element is populated only for a relations object.
Transaction	Contains the Application Policy Infrastructure Controller (APIC) transaction ID that resulted in a specific API callout. The transaction ID is used for correlating request/response between APIC and the device script. It is used primarily for debugging convenience. A script can ignore this value.
Device	Any configuration passed in a cluster API or service API is typically applied to the cluster and it is in effect on all devices within the cluster. However, there can be cases in which the configuration must be applied to a specific device within the cluster, such as when configuring the interface IP address for devices within the cluster. Each device can be assigned a unique IP address. As a result, the interface IP address configuration must be applied to one specific device within the cluster. You accomplish this by using device context labels on the APIC. When you configure the parameter on the APIC, you can associate a device context label that identifies a device within the cluster on which the configuration should be applied. If a parameter is tied to a specific device context within a cluster, the APIC instantiates a device key in the dictionary with device name as its value. The script can lookup the device name in the device dictionary passed in the callout. A script can apply the parameter configuration to a specific device identified by device field.

For more information about the configuration dictionary format, see Sample Script, on page 23. For an example dictionary for connector, encapsulation, and interface information, see Fabric Connectivity

API Return Value

The APIs return a dictionary with the following format:

```
{ 'state':
  'health': []
  'fault': []
}
```

The state returns one of the following values:

```
SUCCESS=0
TRANSIENT=1
PERMANENT=2
AUDIT=3
```

For information about the health, see Health Monitoring. For information about faults, see Faults Codes.

You should configure the device script to set a timeout of at least 30 seconds to establish a connection with the device. If the device script fails to establish network connectivity within the time interval, it returns a `TRANSIENT (1)` state in the return dictionary. The APIC retries the transaction until the Transient state is cleared.

Transient faults indicate failures that don't require immediate user attention to resolve the issue. It could be a temporary event that prevents a script from pushing the configuration. APIC will retry pushing the configuration aggressively till the fault is cleared. If a transient fault fails to clear after multiple (5) retries, APIC marks the failure as permanent.

A device script can request for an audit call by returning the `AUDIT` state in the return dictionary. APIC will trigger a clusterAudit(), deviceAudit(), or serviceAudit() call depending on whether the cluster API, device API, or service API returned an `AUDIT` state. The script can request for an audit in the event it detects a configuration mismatch between the APIC and a device which cannot be resolved within the current API call

A device script returns a `PERMANENT` fault if the parameter values or configuration has an issue. User intervention might be required to resolve the problem.

A persistent transient fault may translate to a permanent fault. APIC will continue to periodically push the configuration till the fault is resolved. The retry is spaced at larger interval. Some faults might require a user intervention to clear, such as an invalid parameter value.

The **scriptwrapper** process that invokes the device script API expects the API to return within 120 seconds. If the script takes longer than 120 seconds, the **scriptwrapper** process terminates and restarts. Any outstanding transactions are replayed after the restart.

Service Configuration

The Application Policy Infrastructure Controller (APIC) creates an instance of a metadevice (`MDev`) for each tenant context. An `MDev` instance is referred to as a virtual device, or `vDev`. All service configuration instances for a tenant are rooted under a `vDev`. The APIC generates a unique id for identifying each `vDev`. The APIC also generates a unique ID for each graph instance, which is represented as `vGrp`. The group configuration and function configuration are rooted under this `vGrp` instance that identifies a specific graph instance.

The configuration dictionary that is passed in the serviceAudit(), **serviceModify**(), serviceHealth(), and serviceCounter() APIs always contains a `vDev` object and a `vGrp` object. A multi-context device script should use the `vDev` object to identify a tenant context uniquely, which could map to a specific routing domain or context.

Parameter Instance Name on Device

Any folders and parameters that are defined under `vnsMDevCfg` are instantiated under `vDev`. Either a multi-context device must create a configuration folder for each `vDev` ID, or the device or device script must concatenate the `vDev` ID that is passed in the configuration dictionary to generate a unique name across multiple contexts.

The following example shows a dictionary with a global folder and parameter for a function:

```
{
  (0,'',4304): {
     'state': 1,
     'transaction': 10000,
     'value': {
          (4,'Server','webserver1'): {
             'state': 1,
              'transaction': 10000,
              'value': {
              (5,'ipaddress','ipaddress'): {
                 'state': 1,
                 'transaction': 10000,
                 'value': '192.168.100.2'
               },
               (5,'servername','servername'): {
                 'state': 1,
                 'transaction': 10000,
                 'value': 'webserver1'
               }
          }
      }
}
```

A device script must make sure that the servername instance is uniquely identified across different contexts. Because the names can overlap across different contexts that are configured on the same device, a device script can append the `vDev` ID to the servername value to make the parameter and folder name unique across different contexts. The following examples are unique servername values:

```
4304_webserver1
webserver1_4304
webserver1.4304
```

Another method for creating unique servername values is by creating a folder called `4304`, and then creating the `webserver1` instance under the `4304` folder.

A single context device can ignore the `vDev` argument that is passed in the configuration dictionary. Similarly, a single context or multi-context device must append the group ID to keep the parameter and folder name that are configured for a graph instance unique from another graph instance that is rendered on the same device, as shown in the following example:

```
(0,'',4304): {
 'state': 1,
 'transaction': 10000,
 'value': {
    (1,'',4368): {
       'state': 1,
       'transaction': 10000,
       'value': {
          (3,'SLB','SLB'): {
             'state': 1,
              'transaction': 10000,
              'value': {
                  (5,'servicename','servicename'): {
                     'state': 1,
                     'transaction': 10000,
                     'value': 'webservice'
                  },
              }
          }
      }
    }
  }
}
```

The device script must ensure that the servicename instance that is created for this graph instance does not overlap with another graph instance that is rendered on the same device. The reason is because the parameter and folder that are defined under a Group or Function configuration in the device specification is unique for

a graph instance or function instance within a graph. Such parameter or folder names might not be unique across different instances of the graph or instances of a function within a graph, respectively. The device script can append the group ID or the group ID and function name that is passed in the device dictionary to make the folder parameter name unique across graphs or functions within a graph, as shown in the following example:

```
4368.SLB.webservice
```

If the device supports creating folders, the script can create a folder for the graph identified by the vgrp ID passed in the configuration dictionary and group specific parameters can be instantiated under the vgrp folder. Similarly function configuration can created under a function specific folder within the group folder, thus maintaining uniqueness of each parameter/folder across multiple graph instances.

API Callouts

Cluster Configuration

Registering the first concrete device within a cluster with Application Policy Infrastructure Controller (APIC) triggers the following sequence of API callouts:

1. `deviceValidate`— This API validates whether a device version registered with the APIC can be supported by the device package.
2. `deviceAudit`— This APIC call clears any device global configuration that is not pushed from the APIC. The device script does not clear the management IP address, login credentials, and any configuration that the device needs to be operational that is not supported through the APIC. The deviceAudit call needs to be selective in clearing the configuration. Only the configuration that can be pushed from the APIC is cleared.
   deviceAudit() should not modify any service function parameter/configuration or cluster level configuration. The service functions should not be affected on a deviceAudit() call. The purpose of the deviceAudit() call is to bring the device level configuration in sync with the APIC. The script should bring the device in sync with minimal disruption to data path. It should identify the configuration found on the device which was not pushed by APIC, such a configuration should be removed.

Note This should be done only for a configuration that can be managed through the device package.

   The script should push any missing configuration or configuration that is not in sync to the device.
3. `clusterAudit`—The APIC calls this API when the first device is added to the logical device (device cluster). This API clears any configuration from the cluster that is not pushed by the APIC. Similar to the `deviceAudit`() call, this API clears only the configuration that can be supported through the APIC.
   The clusterAudit() call should not modify any service function parameter/configuration or device specific parameters. The service functions should not be affected on a clusterAudit() call. The purpose of the clusterAudit() call is to bring the cluster level configuration in sync with the APIC. The script should bring the device in sync with minimal disruption to the data path. It should identify the configuration found on the device which was not pushed by APIC, such a configuration should be removed.

Note This should be done only for configuration that can be managed through a device package.

   A script should push any missing configuration or configuration that is not in sync to the device.

Device packages that support multiple contexts must use the clusterAudit() call for clearing any unwanted tenant configurations from the device. The APIC allocates a unique `VDev` instance for each tenant context. A device can use the APIC for isolating tenant-specific configurations. With the 1.1 release, the APIC provides a list of `VDev` IDs during the clusterAudit() call. The `VDev` ID list is passed in the device dictionary. The device script can use the `VDev` ID list information to identify contexts that should be cleared from the device.

The following example shows a `VDev` ID list that was passed in the device dictionary:

```
{'creds': {'password': '<hidden>', 'username': 'nsroot'},
 'devs': {'Generic1': {'creds': {'password': '<hidden>',
                                 'username': 'nsroot'},
                       'host': '42.42.42.100',
                       'port': 80,
                       'version': '1.0',
                       'virtual': True},
          'Generic2': {'creds': {'password': '<hidden>',
                                 'username': 'nsroot'},
                       'host': '42.42.42.101',
                       'port': 80,
                       'version': '1.0',
                       'virtual': True}},
 'host': '42.42.42.99',
 'name': 'InsiemeCluster',
 'port': 80,
 'vdevs': [{'ctxName': 'cokectx1', 'id': 9597, 'tenant': 'coke'}],
 'virtual': True}
```

Note The APIC generates a serviceAudit() for each `VDev` ID and provides a complete configuration. The configuration within the context must be synchronized during the serviceAudit() call.

4. `clusterModify`—This API is called for any device that is registered and added to a logical device (device cluster). The call results in configuring the cluster configuration.
5. `serviceAudit`—This API is called with function configuration applied by the user on APIC. The script should push any service functions passed in the configuration. If the device has any service function that is not configured on APIC but could be managed through the API, the script should remove such a configuration. The purpose of the serviceAudit() call is to make sure service specific functionality on the device is in sync with the APIC.

Note A device cluster state maintained within the APIC is changed to be operationally up when clusterAudit() returns success. All service level APIs are invoked only after cluster is operational.

The cluster can be marked operational once a single device within the cluster is operational.

Registering additional devices within a cluster triggers the following calls:

- deviceValidate()
- deviceAudit()
- clusterAudit()

Note clusterAudit() should not disrupt devices that are operational within the cluster. Besides service functions that are deployed on the cluster others should not be impacted by adding more devices (such as **deviceAudit**() or clusterAudit()) should not impact the service functions that are in operational state on the cluster.

After the device is registered, APIC periodically calls the following APIs:

- `deviceHealth`
- `deviceCounter`

Removing a device within a cluster results in calling clusterAudit(). If the last device is removed from the cluster and the cluster state changes to operationally down, APIC calls serviceModify() to remove any service specific configuration.

Service Graph Configuration

When you instantiate a graph by associating an abstract graph to a contract that is bound to an endpoint group (EPG), the APIC calls the following API:

- `serviceModify`—This API instantiates service functions on the device.

After a service has been rendered, the APIC periodically calls the following APIs:

- serviceHealth
- serviceCounter

Audit Calls

APIC will trigger a serviceAudit() when the APIC cluster changes while there is an outstanding transaction with the device. Since the cluster change can cause a different APIC to resume communication with the device, a previous configuration transaction may not have completed causing the device and the APIC cluster to go out of sync. The serviceAudit() call issued by a new master ensures the device state is kept in sync with the APIC.

APIC passes the entire service configuration that is associated with a given device in a single serviceAudit() call. A script is required to process serviceAudit() with a minimum disruption to services configured on the

device. The serviceAudit() call should not result in clearing any device specific global configuration or cluster configuration.

Similar to serviceAudit(), APIC triggers a deviceAudit() and clusterAudit() depending on whether there were any outstanding device configurations or cluster configuration transactions in progress.

The device script can trigger an audit call when it detects that the configuration on the device and the APIC has changed and the script cannot resolve the difference from within the API call.

The `clusterAudit`() call can be triggered by returning "3" (AUDIT) as the return state for the `clusterModify`() call. The APIC passes the entire cluster configuration that is defined by the folders and parameters under `vnsClusterCfg` in `clusterAudit`(). The service function configuration is not passed in `clusterAudit`(). The script must apply the configuration that is passed in the dictionary and remove any configuration that is not defined by the APIC. The script removes only unwanted configurations that are found on the device that can be managed by the APIC and is defined in a device specification file under `vnsClusterCfg`.

The `deviceAudit`() call can be triggered by returning "3" (AUDIT) as the return state for the `deviceModify`(), `deviceHealth`(), or `deviceCounter`() call. The APIC passes the entire device configuration that is defined by the folders and parameters under `vnsDevCfg` in the `deviceAudit`() call. The service function configuration is not passed in `deviceAudit`(). The script must apply the configuration that is passed in the dictionary and remove any device configuration that is not defined by the APIC. The script removes only a configuration that is defined on the APIC and can be managed through the APIC.

The `serviceAudit`() can be triggered by returning '3' as the return state for the `serviceModify`(), `serviceHealth`(), or `serviceCounter`() call. The APIC passes all service function configurations across all `vDev` (tenant) and graph instances that are rendered on the device cluster. The device removes any configuration that is not configured by the APIC and can be managed through the APIC.

Passing Parameters

The following example shows a configuration dictionary that is passed for service APIs:

```
Configuration = {
   (0, mDev-key, mDev-Instance-Name) : {
       'state': state
       'value': {
            (1, '', functionGroup-Instance) : {
               'state': state
               'value': {
                   (3, mDevFunction-Key, mDev-Function-Instance-Name): {
                       'state': state,
                       'value': {
                            (2, mDevFunction-Connector-Key, InstanceName): {
                               'state': state
                               'value': {
                                  'CDev-Instance-Name': 'EncapAssociation-Instance',
                                   ...
                               }
                            }
                            (4, mFolder-Key, mFolder-Instance): {
                                  'state': state
                                  'value': {
                                       (5, mParam-Key, mParam-Instance): {
                                            'state': state
                                             'device': cluster-node-instance
                                              'value': {
                                              }
                                       }
                                  }
                            }
                       }
```

```
                        }
                    }
                }
                (4, mFolder-Key, mFolder-Instance): {
                      'state': state
                      'value': {
                           (5, mParam-Key, mParam-Instance): {
                                'state': state
                                 'value': {
                                 }
                           }
                      }
                },
                (7, '', <encap-instance>): {'state': 1, 'tag': TagValue, 'type': TagType },
                (8, '', <encap-association-Instance>): {
                     'state': 1,
                     'encap': <encapInstance>,
                     'vif': <LogicalInterfaceInstanceID>
                },
                (10, '', <LogicalInterfaceInstance>): {
                     'state': 0,
                     'cifs': {'
                        'state': 0,
                        'value': <Interface Value>
                      }
                }
          }
    },
}         '

Device = {
   'devs': {
      cluster-node-Instance: {
            'host': cluster-node-ipaddress
            'port': cluster-node-port-number,
            'creds': {
               'username': username,
               'password': password
            }
      }
    },
    'name': cluster-name,
    'host': cluster-ipaddress
    'port': cluster-port-number,
    'creds': {
        'username': username,
        'password': password
    }
}
```

Note All parameters are strings.

The following example shows parameters that are passed in the deviceValidate() callout.

```
Device =  {
           'creds': {
                  'password': '<password>',
                   'username': 'admin'
             },
            'host': '10.0.0.2',
            'port': 443,
            'virtual': False
       }

version: '1.0'
```

The following is an example of a parameter dictionary that is passed in deviceAudit(), deviceModify(), deviceHealth(), and deviceCounter() call outs. This is parameter structure is identical to a configuration dictionary that is passed in the service or cluster API call outs.

```
devices: = {
          'host': '10.0.0.2',
          'port': 443,
          'creds': {'username': 'admin',
                    'password': '<password>'},
          'virtual': False},

interfaces =  {
      (11, '', 'eth1_0'): {'state': 0, 'label': ''},
      (11, '', 'eth1_1'): {'state': 0, 'label': ''}
},
```

The following is an example of parameters that are passed in the clusterAudit() and clusterModify() APIs. The configuration dictionary format is identical to the example that is shown in the service call out.

```
Device =  {
      'name': 'LB',
      'virtual': False,
      'devs': {
            'Device1': {
                  'host': '10.0.0.2',
                  'port': 443,
                  'creds': {
                        'username': 'admin',
                        'password': 'password'
                   },
                   'virtual': False}
               },
         'host': '10.0.0.1',
         'port': 443,
         'creds': {
               'username': 'admin',
               'password': '<password>'
         }
  },

Interfaces =  {
    (12, '', 'internal'): {
             'state': 0,
             'cifs': {
                   'Device1': 'eth1_1'
             },
             'label': ''
     },
     (12, '', 'external'): {
               'state': 0,
                'cifs': {
                      'Device1': 'eth1_0'
                 },
               'label': ''
       }
 }
```

Device Identification

The device parameter is a simple dictionary that contains the device configuration and the credentials required to access the device. Most of the functions in the device script take a device parameter that identifies the device intended for modification, as shown in the following example:

```
{
  'creds': {
    'password': 'admin',
    'username': 'admin'
  },
```

```
  'host': '10.30.13.153',
  'port': 443
}
```

The Application Policy Infrastructure Controller (APIC) stores credentials in an encrypted partition.

The script must not store the credentials in temporary files and should not print the credential in debug logs.

Endpoint and Network Event Callouts

If the service function connector requires notification and you have enabled notification, the Application Policy Infrastructure Controller (APIC) calls the attachEndpoint() and detachEndpoint() APIs with endpoint information for an endpoint group (EPG) that is associated with a given connector. The APIC calls the APIs for each function connector that supports notification and for which you have administratively enabled notification. The endpoint is passed as a list.

The APIC controller version 1.1 and later supports passing endpoint information by using the serviceModify() and serviceAudit() call. The device script must use endpoint and network information that is passed in serviceModify() or serviceAudit() callout instead of the attachEndpoint() and detachEndpoint() callouts and the attachNetwork() and detachNetwork() callouts. The serviceAudit() callout provides the capability to audit and keep dynamically added endpoint and subnet information in sync with the APIC. The attachEndpoint(), detachEndpoint(), attachNetwork(), and detachNetwork() APIs alone do not support audit capability.

For controller version 1.1 and later, if the device script supports handling endpoint or network information through the service API callouts, the device script should not implement the attachEndpoint(), detachEndpoint(), attachNetwork(), and detachNetwork() APIs. If these APIs are also defined, the APIC calls the attachEndpoint(), detachEndpoint(), and serviceModify() APIs on the endpoint event. Similarly, the APIC calls the attachEndpoint(), detachEndpoint(), and serviceModify() APIs on the network event.

Controller version 1.0 does not provide endpoint and network information in the serviceAudit() and serviceModify() APIs.

The endpoint information in the serviceModify() and serviceAudit() APIs is passed as a new object type: 22. The endpoint information is contained under the connector within a function. The endpoint information for an EPG is contained under the function connector that is associated with the EPG.

The network information in the serviceModify() and serviceAudit() APIs is passed as a new object type: 23. The network information is contained under the connector within a function. The network information for an EPG is contained under the function connector that is associated with the EPG.

Following is a sample configuration dictionary for conveying endpoint information in a serviceModify() API call:

```
{'args': ({
    (0, '', 5348): {
        'ackedState': 0,
        'ctxName': 'tenant1ctx1',
        'state': 2,
        'tenant': 'tenant1',
        'transaction': 0,
        'txid': 10000,
        'value': {
            (1, '', 5661): {
                'absGraph': 'G1',
                'ackedState': 0,
                'state': 2,
                'transaction': 0,
                'value': {
                    (3, 'SubnetFunc', 'Node'): {
                        'ackedState': 0,
                        'state': 2,
                        'transaction': 0,
```

```
                        'value': {
                            (2, 'external', 'inside'): {
                                'ackedState': 0,
                                'state': 2,
                                'transaction': 0,
                                'value': {
                                    (9, '', 'InsiemeCluster_inside_2621440_32773'): {
                                        'ackedState': 0,
                                        'state': 0,
                                        'target': 'InsiemeCluster_inside_2621440_32773',
                                        'transaction': 0},
                                    (23, '', u'10.10.10.0/24'): {
                                        'ackedState': 0,
                                        'state': 0,
                                        'epg': 'app',
                                        'transaction': 0},
                                    (23, '', u'10.10.11.0/24'): {
                                        'ackedState': 0,
                                        'state': 0,
                                        'epg': 'app',
                                        'transaction': 0},
                                    (22, '', u'10.10.12.0/24'): {
                                        'ackedState': 0,
                                        'state': 0,
                                        'epg': 'app',
                                        'transaction': 0}}},
                            (2, 'internal', 'outside'): {
                                'ackedState': 0,
                                'state': 2,
                                'transaction': 0,
                                'value': {
                                    (9, '', 'InsiemeCluster_outside_2621440_16388'): {
                                        'ackedState': 0,
                                        'state': 0,
                                        'target': 'InsiemeCluster_outside_2621440_16388',
                                        'transaction': 0}}},
                            (4, 'folder', 'folder1'): {
                                'ackedState': 0,
                                'device': 'Generic1',
                                'state': 0,
                                'transaction': 0,
                                'value': {
                                    (5, 'param', 'param'): {
                                        'ackedState': 0,
                                        'state': 0,
                                        'transaction': 0,
                                        'value': 'value'}}},
                            (4, 'folder', 'folder2'): {
                                'ackedState': 0,
                                'device': 'Generic2',
                                'state': 0,
                                'transaction': 0,
                                'value': {
                                    (5, 'param', 'param'): {
                                        'ackedState': 0,
                                        'state': 0,
                                        'transaction': 0,
                                        'value': 'value'}}}}}}},
(4, 'oneFolder', 'f1'): {
    'ackedState': 0,
    'state': 0,
    'transaction': 0,
    'value': {
        (5, 'oneParam', 'p1'): {
            'ackedState': 0,
            'state': 0,
            'transaction': 0,
            'value': 'v1'}}},
(7, '', '2621440_16388'): {
    'ackedState': 0,
    'state': 0,
    'tag': 237,
    'transaction': 0,
```

```
                'type': 1},
            (7, '', '2621440_32773'): {
                'ackedState': 0,
                'state': 0,
                'tag': 238,
                'transaction': 0,
                'type': 1},
            (8, '', 'InsiemeCluster_inside_2621440_32773'): {
                'ackedState': 0,
                'encap': '2621440_32773',
                'state': 0,
                'transaction': 0,
                'vif': 'InsiemeCluster_inside'},
            (8, '', 'InsiemeCluster_outside_2621440_16388'): {
                'ackedState': 0,
                'encap': '2621440_16388',
                'state': 0,
                'transaction': 0,
                'vif': 'InsiemeCluster_outside'},
            (10, '', 'InsiemeCluster_inside'): {
                'OspfVEncapAscCfg': {},
                'ackedState': 0,
                'cifs': {'Generic1': 'ext',
                'Generic2': 'ext'},
                'state': 0,
                'transaction': 0},
            (10, '', 'InsiemeCluster_outside'): {
                'OspfVEncapAscCfg': {},
                'ackedState': 0,
                'cifs': {'Generic1': 'int',
                'Generic2': 'int'},
                'state': 0,
                'transaction': 0}}}},),
    'device': {
        'creds': {'password': '<hidden>', 'username': 'nsroot'},
        'devs': {
            'Generic1': {
                'creds': {
                    'password': '<hidden>',
                    'username': 'nsroot'},
                'host': '42.42.42.100',
                'port': 80,
                'version': '1.0',
                'virtual': True},
            'Generic2': {
                'creds': {
                    'password': '<hidden>',
                    'username': 'nsroot'},
                'host': '42.42.42.101',
                'port': 80,
                'version': '1.0',
                'virtual': True}},
        'host': '42.42.42.99',
        'name': 'InsiemeCluster',
        'port': 80,
        'virtual': True}}
```

Following is a sample configuration dictionary for conveying network information in a serviceModify() API call:

```
{'args': ({
    (0, '', 5348): {
        'ackedState': 0,
        'ctxName': 'tenant1ctx1',
        'state': 2,
        'tenant': 'tenant1',
        'transaction': 0,
        'txid': 10000,
        'value': {
            (1, '', 5661): {
                'absGraph': 'G1',
                'ackedState': 0,
                'state': 2,
```

```
    'transaction': 0,
    'value': {
        (3, 'SubnetFunc', 'Node'): {
            'ackedState': 0,
            'state': 2,
            'transaction': 0,
            'value': {
                (2, 'external', 'inside'): {
                    'ackedState': 0,
                    'state': 2,
                    'transaction': 0,
                    'value': {
                        (9, '', 'InsiemeCluster_inside_2621440_32773'): {
                            'ackedState': 0,
                            'state': 0,
                            'target': 'InsiemeCluster_inside_2621440_32773',
                            'transaction': 0},
                        (22, '', u'10.10.10.11'): {
                            'ackedState': 0,
                            'state': 0,
                            'transaction': 0},
                        (22, '', u'10.10.10.12'): {
                            'ackedState': 0,
                            'state': 0,
                            'transaction': 0},
                        (22, '', u'10.10.10.13'): {
                            'ackedState': 0,
                            'state': 0,
                            'transaction': 0}}},
                (2, 'internal', 'outside'): {
                    'ackedState': 0,
                    'state': 2,
                    'transaction': 0,
                    'value': {
                        (9, '', 'InsiemeCluster_outside_2621440_16388'): {
                            'ackedState': 0,
                            'state': 0,
                            'target': 'InsiemeCluster_outside_2621440_16388',
                            'transaction': 0}}},
                (4, 'folder', 'folder1'): {
                    'ackedState': 0,
                    'device': 'Generic1',
                    'state': 0,
                    'transaction': 0,
                    'value': {
                        (5, 'param', 'param'): {
                            'ackedState': 0,
                            'state': 0,
                            'transaction': 0,
                            'value': 'value'}}},
                (4, 'folder', 'folder2'): {
                    'ackedState': 0,
                    'device': 'Generic2',
                    'state': 0,
                    'transaction': 0,
                    'value': {
                        (5, 'param', 'param'): {
                            'ackedState': 0,
                            'state': 0,
                            'transaction': 0,
                            'value': 'value'}}}}}}},
(4, 'oneFolder', 'f1'): {
    'ackedState': 0,
    'state': 0,
    'transaction': 0,
    'value': {
        (5, 'oneParam', 'p1'): {
            'ackedState': 0,
            'state': 0,
            'transaction': 0,
            'value': 'v1'}}},
(7, '', '2621440_16388'): {
    'ackedState': 0,
```

```
                'state': 0,
                'tag': 237,
                'transaction': 0,
                'type': 1},
            (7, '', '2621440_32773'): {
                'ackedState': 0,
                'state': 0,
                'tag': 238,
                'transaction': 0,
                'type': 1},
            (8, '', 'InsiemeCluster_inside_2621440_32773'): {
                'ackedState': 0,
                'encap': '2621440_32773',
                'state': 0,
                'transaction': 0,
                'vif': 'InsiemeCluster_inside'},
            (8, '', 'InsiemeCluster_outside_2621440_16388'): {
                'ackedState': 0,
                'encap': '2621440_16388',
                'state': 0,
                'transaction': 0,
                'vif': 'InsiemeCluster_outside'},
            (10, '', 'InsiemeCluster_inside'): {
                'OspfVEncapAscCfg': {},
                'ackedState': 0,
                'cifs': {
                    'Generic1': 'ext',
                    'Generic2': 'ext'},
                'state': 0,
                'transaction': 0},
            (10, '', 'InsiemeCluster_outside'): {
                'OspfVEncapAscCfg': {},
                'ackedState': 0,
                'cifs': {
                    'Generic1': 'int',
                    'Generic2': 'int'},
                'state': 0,
                'transaction': 0}}}},),
    'device': {
        'creds': {'password': '<hidden>', 'username': 'nsroot'},
        'devs': {
            'Generic1': {
                'creds': {
                    'password': '<hidden>',
                    'username': 'nsroot'},
                'host': '42.42.42.100',
                'port': 80,
                'version': '1.0',
                'virtual': True},
            'Generic2': {
                'creds': {
                    'password': '<hidden>',
                    'username': 'nsroot'},
                'host': '42.42.42.101',
                'port': 80,
                'version': '1.0',
                'virtual': True}},
        'host': '42.42.42.99',
        'name': 'InsiemeCluster',
        'port': 80,
        'virtual': True}}
```

The state field that is associated with the endpoint (22) object and network (23) object has the following meanings:

State	Description
0	No change to the endpoint or network. The endpoint or network continues to be associated with the EPG.

State	Description
1	Identifies an attach event. A new endpoint or network was associated to the EPG.
2	This state typically does not occur. This state can be treated the same as a new endpoint or network association (state 1).
3	Identifies a detach event. An endpoint or network was dissociated from the EPG.

Handling Script Failures

The Application Policy Infrastructure Controller (APIC) services integration model is based on the promise theory, in which individual agents join in a system of voluntary cooperation. The APIC pushes the intended state to the script and provides an API to raise faults on parts of the configuration.

Failures can occur when parameter values change or when a device error occurs. In the case of an APIC failure, the new APIC determines whether to reissue the serviceModify call or audit the function groups. The APIs can return a fault list that provides details about the cause for the fault and the potential corrective action for resolving the fault.

The APIC does not maintain a transaction history. The state of an object within the configuration dictionary is determined based on the state of the managed object information tree. The object state is marked "NEW (1)" when a new object is inserted in the managed object tree. Any subsequent API callouts from the APIC sets that objects state to "UNCHANGED (0)" till either the object value is changed by an explicit user configuration or an implicit the APIC event that causes the object value to change.

On a change in objects value, the state of the object in the next modify() API call is set to "CHANGED(2)". If the object is deleted, the APIC indicates the deletion by setting the object state to "DELETED (3)".

If the API returns the "PERMANENT(2)" state, which indicates that the script encountered an error while processing the configuration, the APIC retries the configuration until it receives the "SUCCESS(0)" state from the script. If the user has not changed the value of any object between retries, the APIC sets the object state to "UNCHANGED(0)" during these retries. The state of the object is set to "NEW(1)", "CHANGED(2)", or "DELETED(3)" only if a new object was created, an existing object was modified, or an existing object was deleted between the retries.

The following example shows a case that can occur due to a lack of transaction history on the APIC. The device package developer should be aware of such issues and address the issues in the device script.

Assume that a device package has parameters A, B, and C, where A depends on B and B depends on C.

- When A, B, and C are created on the APIC, this causes the APIC to call a device script with the "create (1)" state for all three parameters, such that A=a(1) -> B=b (1) -> C=c (1).

  The script raises a fault for B and configures C, but does not configure A and returns the "PERMANENT" state. The end of this callout device has only "c".

  The APIC raises a fault on B because the configuration for object B was invalid.

- When the value of B is changed to b', this resolves the configuration issue. This change in configuration results in the APIC calling the device script with the modification. The parameter state during the subsequent API callout is as follows:

  A=a(0) -> B=b'(2) -> C=c(0)

  The APIC does not maintain a transaction history. The APIC does not have any knowledge that A=a was not applied on the device during the previous transaction. The APIC retries the configuration and updates the parameter state for B, as it was the only parameter that changed between retries.

  The script tries to update parameter B to b'. The script should be able to identify that parameter B does not exist on the device and is being modified before the create operation. Ideally, the modify request to the device should return an error.

  If the device is capable of identifying a modify operation before a create operation as an error, the script should handle such an error from the device using one of the following options:

  - Option 1

    Create the parameter B with value b' on the device and resolve any dependent objects that could be missing from the device. The script must walk through the configuration that is passed in the dictionary and check if the objects that depend on the modified object were created on the device. If the objects were not created, the script should create the object. In this example, the script should create A=a that depends on B=b' because object A=a is not found on the device.

  - Option 2

    Return the AUDIT(3) state in the return response. The APIC replays the entire configuration.

    

    Note Requesting an audit call can be an expensive operation because the APIC passes the entire configuration.

    The device script needs to identify the missing configuration and apply any missing configuration parameters. This option should be used only when a modified object in the dictionary has dependent objects in the configuration.

  If the device is not capable of returning an error when a parameter is modified before a create operation, the device script should read the object that is being modified before performing a modify operation on it. This approach of reading the configuration from the device before modifying can be expansive and impact performance. To keep the solution optimal, reading the configuration from the device should be done only if the object being modified could have other objects depending on it.

Sample Script

The following example shows a device script in Python:

```
import pprint
import sys
import Insieme.Logger as Logger

#
# Infra API
#

def deviceValidate( device,version ):
```

```
    return {
        'state': 0,'version': '1.0'
        }

def deviceModify( device,interfaces,configuration):
    return {
        'state': 0,'faults': [],'health': []
        }

def deviceAudit( device,interfaces,configuration ):
    return {
        'state': 0,'faults': [],'health': []
        }

def deviceHealth( device,interfaces,configuration ):
    return {
        'state': 0,'faults': [],'health': [([], 100)]
        }

def deviceCounters( device,interfaces,configuration ):
    return {
        'state': 0,'counters': [
            ( [(11,'','eth0')],{
                'rxpackets':100,
                'rxerrors':101,
                'rxdrops':102,
                'txpackets':200,
                'txerrors':201,
                'txdrops':202
            }
          )
        ]
    }

def clusterModify( device,interfaces,configuration ):
    return {
        'state': 0,'faults': [],'health': []
        }

def clusterAudit( device,interfaces,configuration ):
    return {
        'state': 0,'faults': [],'health': []
        }

#
# FunctionGroup API
#

def serviceModify( device,configuration ):
    return {
        'state': 0,'faults': [],'health': []
       }

def serviceAudit( device,configuration ):
    return {
        'state': 0,'faults': [],'health': []
        }

def serviceHealth( device,configuration ):
    return {
        'state': 0,'faults': [],'health': []
        }

def serviceCounters( device,configuration ):
    externalIntferface, = [
        (0, 'Firewall', 4384),
        (1, '', 4432),
        (3, 'Firewall-Func', 'FW-1'),
        (2, 'external', 'external1')
    ]
    internalInterface = [
        (0, 'Firewall', 4384)
        (1, '', 4432)
```

```
        (3, 'Firewall-Func', 'FW-1'),
        (2, 'internal','internal1')
    ]
    Firewall-1-External-Counters = (externalInterface, {
        'rxpackets': 100,
        'rxerrors': 0,
        'rxdrops': 0
        'txpackets': 100
        'txerrors': 4
        'txdrops': 2
        }
    )
    Firewall-1-Internal-Counters = (internalInterface, {
        'rxpackets': 100,
        'rxerrors': 0,
        'rxdrops': 0
        'txpackets': 100
        'txerrors': 4
        'txdrops': 2
        }
    )
    Counters = [ Firewall-1-External-Counters, Firewall-1-Internal-Counters ]
        return {
            'state': 0,
            'counters': Counters
        }

#
# EndPoint/Network API
#

def attachEndpoint( device,
                    configuration,
                    endpoints ):
    return {
        'state': 0,
        'faults': [],
        'health': [],
        }

def detachEndpoint( device,
                    configuration,
                    endpoints ):
    return {
        'state': 0,
        'faults': [],
        'health': [],
        }

def attachNetwork( device,
                   configuration,
                   networks ):
    return {
        'state': 0,
        'faults': [],
        'health': [],
        }

def detachNetwork( device,
                   configuration,
                   networks ):
    return {
        'state': 0,
        'faults': [],
        'health': [],
        }
```

The following is an example invocation:

```
Function: deviceValidate

Arguments:
```

```
(
    {
    'creds': {
        'password': 'admin',
        'username': 'admin'
    },
    'host': '10.30.13.153','port': 443
}
u'1.0'
)
```

Function: deviceAudit

Arguments:

```
(
    {
    'host': '10.30.13.153',
    'port': 443,
    'creds': {
        'username': 'admin','password': 'admin'
    }
    },
    {
        (11,'','1_1'): {
        'state': 0,
        'label': 'int'
    },
    (11,'','1_2'): {
        'state': 0,
        'label': 'ext'
    },
    (11,'','1_3'): {
        'state': 0,
        'label': 'mgmt'
    }
    },
    {
        (4,'HighAvailabilityCfg','HA'): {
            'state': 2,'value': {
                (5,'peerIP','peerip'): {
                    'state': 2,
                    'value': '10.30.13.154'
                }
            }
        }
    }
)
```

Function: deviceCounters

Arguments:

```
(
    {
    'creds': {
        'password': 'insieme',
        'username': 'admin'
    },
    'host': '10.0.0.2',
    'port': 443,
    'virtual': False
    },
    {
    (11, '', 'eth1_0'): {
        'label': '',
        'state': 0
    },
    (11, '', 'eth1_1'): {
        'label': '',
        'state': 0
    }
    },
```

```
    {
    (4, 'HighAvailability', 'HA'): {
        'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': {
           (5, 'id', 'id'): {
               'ackedState': 0,
               'state': 1,
               'transaction': 0,
               'value': '1'
           },
           (5, 'ipaddress', 'ipaddress'): {
               'ackedState': 0,
               'state': 1,
               'transaction': 0,
               'value': '10.0.0.3'
           },
           (5, 'netmask', 'netmask'): {
               'ackedState': 0,
               'state': 1,
               'transaction': 0,
               'value': '255.255.255.0'
           }
       }
   }
}
)
```

Function: clusterAudit

Arguments:

```
(
    {
    'name': 'Cluster1',
    'virtual': False,
    'devs': {
        'SampleDevice1': {
            'host': '10.30.13.153',
            'port': 443,
            'creds': {
                'username': 'admin',
                'password': 'admin'
            }
        }
    },
    'host': '10.30.13.153',
    'port': 443,
    'creds': {
        'username': 'admin',
        'password': 'admin'
    }
    },
    {
    (12, '', 'internal'): {
        'state': 0,
        'label': 'int'
    },
    (12, '', 'external'): {
        'state': 0,
        'label': 'ext'
    }
    },
    {
    (4,'SyslogConfig','syslogconfig'): {
        'state': 2,
        'value': {
            (5,'ipaddress','syslogip'): {
                'state': 2,
                'value': '10.168.62.100'
            }
        }
```

```
        },
        (4,'NTPConfig','ntpconfig'): {
            'state': 2,
            'value': {
                (5,'ipaddress','syslogip'): {
                    'state': 2,
                    'value': '10.168.62.1'
                }
            }
        }
    }
)
```

Function: clusterModify

Arguments:

```
(
    {
    'name': 'Cluster1',
    'virtual': False,
    'devs': {
        'SampleDevice1': {
            'host': '10.30.13.153',
            'port': 443,
            'creds': {
                'username': 'admin',
                'password': 'admin'
            }
        }
    },
    'host': '10.30.13.153',
    'port': 443,
    'creds': {
        'username': 'admin',
        'password': 'admin'
    }
    },
    {
    (12, '', 'internal'): {
        'state': 0,
        'label': 'int'
    },
    (12, '', 'external'): {
        'state': 0,
        'label': 'ext'
    }
    },
    {
    (4,'SyslogConfig','syslogconfig'): {
        'state': 2,
        'value': {
            (5,'ipaddress','syslogip'): {
                'state': 2,
                'value': '10.168.62.100'
            }
        }
    },
    (4,'NTPConfig','ntpconfig'): {
        'state': 2,
        'value': {
            (5,'ipaddress','syslogip'): {
                'state': 2,
                'value': '10.168.62.1'
            }
        }
    }
  }
)
```

Function: serviceModify

Arguments:

```
(
    {
    'Device1': {
        'creds': {
            'password': 'insieme',
            'username': 'admin'
        },
        'host': '10.0.0.2',
        'port': 443,
        'virtual': False
    }
    },
    'host': '10.0.0.1',
    'name': 'LB',
    'port': 443,
    'virtual': False
    },
    {
    (0, '', 5539): {
        'ackedState': 0,
        'ctxName': 'TenantActx1',
        'state': 1,
        'tenant': 'TenantA',
        'transaction': 0,
        'txid': 10000,
        'value': {
            (1, '', 4411): {
                'absGraph': 'WebGraph',
                'ackedState': 0,
                'state': 1,
                'transaction': 0,
                'value': {
                    (3, 'LoadBalancing', 'SLB'): {
                        'ackedState': 0,
                        'state': 1,
                        'transaction': 0,
                        'value': {
                            (2, 'external', 'external'): {
                                'ackedState': 0,
                                'state': 1,
                                'transaction': 0,
                                'value': {
                                    (9, '', 'LB_external_2424832_32771'): {
                                        'ackedState': 0,
                                        'state': 1,
                                        'target': 'LB_external_2424832_32771',
                                        'transaction': 0
                                    }
                                }
                            },
                            (2, 'internal', 'internal'): {
                                'ackedState': 0,
                                'state': 1,
                                'transaction': 0,
                                'value': {
                                    (9, '', 'LB_internal_2424832_49154'): {
                                        'ackedState': 0,
                                        'state': 1,
                                        'target': 'LB_internal_2424832_49154',
                                        'transaction': 0
                                    }
                                }
                            },
                            (4, 'externalNetwork', 'externalNetwork'): {
                                'ackedState': 0,
                                'connector': 'external',
                                'state': 1,
                                'transaction': 0,
                                'value': {
                                    (6, 'externalNetworkRel', 'externalNetworkRel'): {
                                        'ackedState': 0,
```

```
                                'state': 1,
                                'target': 'network/externalIP',
                                'transaction': 0
                            }
                        }
                    },
                    (4, 'internalNetwork', 'internalNetwork'): {
                        'ackedState': 0,
                        'connector': 'internal',
                        'state': 1,
                        'transaction': 0,
                        'value': {
                            (6, 'internalNetworkRel', 'internalNetworkRel'): {
                                'ackedState': 0,
                                'state': 1,
                                'target': 'network/internalIP',
                                'transaction': 0
                            }
                        }
                    },
                    (4, 'lbvserverCfg', 'lbvserverCfg'): {
                        'ackedState': 0,
                        'connector': 'external',
                        'state': 1,
                        'transaction': 0,
                        'value': {
                            (6, 'lbvserverRel', 'lbvserverRel'): {
                                'ackedState': 0,
                                'state': 1,
                                'target': 'lbvserver',
                                'transaction': 0
                            }
                        }
                    },
                    (4, 'serverpoolCfg', 'serverpoolCfg'): {
                        'ackedState': 0,
                        'state': 1,
                        'transaction': 0,
                        'value': {
                            (6, 'serverpoolRel', 'serverpoolRel'): {
                                'ackedState': 0,
                                'state': 1,
                                'target': 'serverpool',
                                'transaction': 0
                            }
                        }
                    },
                    (4, 'vipCfg', 'vipcfg'): {
                        'ackedState': 0,
                        'state': 1,
                        'transaction': 0,
                        'value': {
                            (6, 'vipRel', 'vipRel'): {
                                'ackedState': 0,
                                'state': 1,
                                'target': 'network/vipaddress',
                                'transaction': 0
                            }
                        }
                    }
                }
            }
        },
        (4, 'Network', 'network'): {
            'ackedState': 0,
            'state': 1,
            'transaction': 0,
            'value': {
                (4, 'ip', 'externalIP'): {
                    'ackedState': 0,
                    'state': 1,
                    'transaction': 0,
                    'value': {
```

```
                        (5, 'ipaddress', 'ipaddress'): {
                            'ackedState': 0,
                            'state': 1,
                            'transaction': 0,
                            'value': '20.0.0.1'
                        },
                        (5, 'netmask', 'netmask'): {
                            'ackedState': 0,
                            'state': 1,
                            'transaction': 0,
                            'value': '255.255.255.0'
                        }
                    }
                },
                (4, 'ip', 'internalIP'): {
                    'ackedState': 0,
                    'state': 1,
                    'transaction': 0,
                    'value': {
                        (5, 'ipaddress', 'ipaddress'): {
                            'ackedState': 0,
                            'state': 1,
                            'transaction': 0,
                            'value': '30.0.0.1'
                        },
                        (5, 'netmask', 'netmask'): {
                            'ackedState': 0,
                            'state': 1,
                            'transaction': 0,
                            'value': '255.255.255.0'
                        }
                    }
                },
                (4, 'ip', 'vipaddress'): {
                    'ackedState': 0,
                    'state': 1,
                    'transaction': 0,
                    'value': {
                        (5, 'ipaddress', 'ipaddress'): {
                            'ackedState': 0,
                            'state': 1,
                            'transaction': 0,
                            'value': '100.0.0.1'
                        },
                        (5, 'netmask', 'netmask'): {
                            'ackedState': 0,
                            'state': 1,
                            'transaction': 0,
                            'value': '255.255.255.255'
                        }
                    }
                }
            }
        },
        (4, 'lbvserver', 'lbvserver'): {
            'ackedState': 0,
            'state': 1,
            'transaction': 0,
            'value': {
                (5, 'ipmask', 'ipmask'): {
                    'ackedState': 0,
                    'state': 1,
                    'transaction': 0,
                    'value': '255.255.255.255'
                },
                (5, 'ipv4', 'ipv4'): {
                    'ackedState': 0,
                    'state': 1,
                    'transaction': 0,
                    'value': '100.0.0.1'
                },
                (5, 'name', 'name'): {
                    'ackedState': 0,
```

```
                'state': 1,
                'transaction': 0,
                'value': 'vserver1'
            },
            (5, 'port', 'port'): {
                'ackedState': 0,
                'state': 1,
                'transaction': 0,
                'value': '80'
            },
            (5, 'servicetype', 'servicetype'): {
                'ackedState': 0,
                'state': 1,
                'transaction': 0,
                'value': 'http'
            }
        }
    },
    (4, 'serverpool', 'serverpool'): {
        'ackedState': 0,
        'state': 1,
        'transaction': 0,
        'value': {
            (4, 'server', 'server1'): {
                'ackedState': 0,
                'state': 1,
                'transaction': 0,
                'value': {
                    (5, 'ip', 'ip'): {
                        'ackedState': 0,
                        'state': 1,
                        'transaction': 0,
                        'value': '30.0.0.2'
                    },
                    (5, 'port', 'port'): {
                        'ackedState': 0,
                        'state': 1,
                        'transaction': 0,
                        'value': '80'
                    }
                }
            },
            (5, 'serverpoolname', 'serverpoolname'): {
                'ackedState': 0,
                'state': 1,
                'transaction': 0,
                'value': 'webpool'
            },
            (5, 'type', 'type'): {
                'ackedState': 0,
                'state': 1,
                'transaction': 0,
                'value': 'http'
            }
        }
    },
    (7, '', '2424832_32771'): {
        'ackedState': 0,
        'state': 1,
        'tag': 436,
        'transaction': 0,
        'type': 1
    },
    (7, '', '2424832_49154'): {
        'ackedState': 0,
        'state': 1,
        'tag': 370,
        'transaction': 0,
        'type': 1
    },
    (8, '', 'LB_external_2424832_32771'): {
        'ackedState': 0,
        'encap': '2424832_32771',
```

```
                        'state': 1,
                        'transaction': 0,
                        'vif': 'LB_external'
                    },
                    (8, '', 'LB_internal_2424832_49154'): {
                        'ackedState': 0,
                        'encap': '2424832_49154',
                        'state': 1,
                        'transaction': 0,
                        'vif': 'LB_internal'
                    },
                    (10, '', 'LB_external'): {
                        'ackedState': 0,
                        'cifs': {
                            'Device1': 'eth1_0'
                        },
                        'state': 1,
                        'transaction': 0
                    },
                    (10, '', 'LB_internal'): {
                        'ackedState': 0,
                        'cifs': {
                            'Device1': 'eth1_1'
                        },
                        'state': 1,
                        'transaction': 0
                    }
                }
            }
        }
)
Function: serviceCounters

Arguments:

(
{'creds': {
    'password': 'insieme',
    'username': 'admin'
    },
    'devs': {
        'Device1': {
            'creds': {
                'password': 'insieme',
                'username': 'admin'
            },
            'host': '10.0.0.2',
            'port': 443,
            'virtual': False
        }
    },
    'host': '10.0.0.1',
    'name': 'LB',
    'port': 443,
    'virtual': False
},
{
(0, '', 5539): {
    'ctxName': 'TenantActx1',
    'state': 2,
    'tenant': 'TenantA',
    'value': {
        (1, '', 4411): {
            'absGraph': 'WebGraph',
            'state': 2,
            'value': {
            (3, 'LoadBalancing', 'SLB'): {
                'state': 2, 'value': {
                    (2, 'external', 'external'): {
                        'state': 2, 'value': {
                            (9, '', 'LB_external_2424832_32771'): {
                                'state': 0, 'target': 'LB_external_2424832_32771'
                            }
```

```
                        }
                    },
                    (2, 'internal', 'internal'): {
                        'state': 2,
                        'value': {
                            (9, '', 'LB_internal_2424832_49154'): {
                                'state': 0,
                                'target': 'LB_internal_2424832_49154'
                            }
                        }
                    },
                    (4, 'externalNetwork', 'externalNetwork'): {
                        'connector': 'external',
                        'state': 0,
                        'value': {
                            (6, 'externalNetworkRel', 'externalNetworkRel'): {
                                'state': 0,
                                'target': 'network/externalIP'
                            }
                        }
                    },
                    (4, 'internalNetwork', 'internalNetwork'): {
                        'connector': 'internal',
                        'state': 0,
                        'value': {
                            (6, 'internalNetworkRel', 'internalNetworkRel'): {
                                'state': 0,
                                'target': 'network/internalIP'
                            }
                        }
                    },
                    (4, 'lbvserverCfg', 'lbvserverCfg'): {
                        'connector': 'external',
                        'state': 0,
                        'value': {
                            (6, 'lbvserverRel', 'lbvserverRel'): {
                                'state': 0,
                                'target': 'lbvserver'
                            }
                        }
                    },
                    (4, 'serverpoolCfg', 'serverpoolCfg'): {
                        'state': 0,
                        'value': {
                            (6, 'serverpoolRel', 'serverpoolRel'): {
                                'state': 0,
                                'target': 'serverpool'
                            }
                        }
                    },
                    (4, 'vipCfg', 'vipcfg'): {
                        'state': 0,
                        'value': {
                            (6, 'vipRel', 'vipRel'): {
                                'state': 0,
                                'target': 'network/vipaddress'
                            }
                        }
                    }
                }
            }
        }
    },
    (4, 'Network', 'network'): {
        'state': 0,
        'value': {
        (4, 'ip', 'externalIP'): {
            'state': 0,
            'value': {
                (5, 'ipaddress', 'ipaddress'): {
                    'state': 0,
                    'value': '20.0.0.1'
                },
```

```
                (5, 'netmask', 'netmask'): {
                    'state': 0,
                    'value': '255.255.255.0'
                }
            }
        },
        (4, 'ip', 'internalIP'): {
            'state': 0,
            'value': {
                (5, 'ipaddress', 'ipaddress'): {
                    'state': 0,
                    'value': '30.0.0.1'
                },
                (5, 'netmask', 'netmask'): {
                    'state': 0,
                    'value': '255.255.255.0'
                }
            }
        },
        (4, 'ip', 'vipaddress'): {
            'state': 0,
            'value': {
            (5, 'ipaddress', 'ipaddress'): {
                'state': 0,
                'value': '100.0.0.1'
            },
            (5, 'netmask', 'netmask'): {
                'state': 0,
                'value': '255.255.255.255'
            }
        }
    }
}
},
(4, 'lbvserver', 'lbvserver'): {
    'state': 0,
    'value': {
    (5, 'ipmask', 'ipmask'): {
        'state': 0,
        'value': '255.255.255.255'
    },
    (5, 'ipv4', 'ipv4'): {
        'state': 0,
        'value': '100.0.0.1'
    },
    (5, 'name', 'name'): {
        'state': 0,
        'value': 'vserver1'
    },
    (5, 'port', 'port'): {
        'state': 0,
        'value': '80'
    },
    (5, 'servicetype', 'servicetype'): {
        'state': 0,
        'value': 'http'
    }
}
},
(4, 'serverpool', 'serverpool'): {
    'state': 0,
    'value': {
        (4, 'server', 'server1'): {
            'state': 0,
            'value': {
                (5, 'ip', 'ip'): {
                    'state': 0,
                    'value': '30.0.0.2'
                },
                (5, 'port', 'port'): {
                    'state': 0,
                    'value': '80'
                }
```

```
                }
            },
            (5, 'serverpoolname', 'serverpoolname'): {
                'state': 0,
                'value': 'webpool'
            },
            (5, 'type', 'type'): {
                'state': 0,
                'value': 'http'
            }
        }
},
(7, '', '2424832_32771'): {
    'state': 0,
    'tag': 436,
    'type': 1
},
(7, '', '2424832_49154'): {
    'state': 0,
    'tag': 370,
    'type': 1
},
(8, '', 'LB_external_2424832_32771'): {
    'encap': '2424832_32771',
    'state': 0,
    'vif': 'LB_external'
},
(8, '', 'LB_internal_2424832_49154'): {
    'encap': '2424832_49154',
    'state': 0,
    'vif': 'LB_internal'
},
(10, '', 'LB_external'): {
    'cifs': {
        'Device1': 'eth1_0'
    },
    'state': 0
},
(10, '', 'LB_internal'): {
    'cifs': {
        'Device1': 'eth1_1'
    },
    'state': 0
}
}
}
)
```

Function: serviceHealth

Arguments:

```
(
{'creds': {
    'password': 'insieme',
    'username': 'admin'
},
'devs': {
    'Device1': {
        'creds': {
            'password': 'insieme',
            'username': 'admin'
        },
        'host': '10.0.0.2',
        'port': 443,
        'virtual': False
    }
    },
    'host': '10.0.0.1',
    'name': 'LB',
    'port': 443,
    'virtual': False
    },
    {(0, '', 5539): {
```

```
'ctxName': 'TenantActx1',
'state': 2,
'tenant': 'TenantA',
'value': {
    (1, '', 4411): {
        'absGraph': 'WebGraph',
        'state': 2,
        'value': {
            (3, 'LoadBalancing', 'SLB'): {
                'state': 2,
                'value': {
                    (2, 'external', 'external'): {
                        'state': 2,
                        'value': {
                            (9, '', 'LB_external_2424832_32771'): {
                                'state': 0,
                                'target': 'LB_external_2424832_32771'
                            }
                        }
                    },
                    (2, 'internal', 'internal'): {
                        'state': 2,
                        'value': {
                            (9, '', 'LB_internal_2424832_49154'): {
                                'state': 0,
                                'target': 'LB_internal_2424832_49154'
                            }
                        }
                    },
                    (4, 'externalNetwork', 'externalNetwork'): {
                        'connector': 'external',
                        'state': 0,
                        'value': {
                            (6, 'externalNetworkRel', 'externalNetworkRel'): {
                                'state': 0,
                                'target': 'network/externalIP'
                            }
                        }
                    },
                    (4, 'internalNetwork', 'internalNetwork'): {
                        'connector': 'internal',
                        'state': 0,
                        'value': {
                            (6, 'internalNetworkRel', 'internalNetworkRel'): {
                                'state': 0,
                                'target': 'network/internalIP'
                            }
                        }
                    },
                    (4, 'lbvserverCfg', 'lbvserverCfg'): {
                        'connector': 'external',
                        'state': 0,
                        'value': {
                            (6, 'lbvserverRel', 'lbvserverRel'): {
                                'state': 0,
                                'target': 'lbvserver'
                            }
                        }
                    },
                    (4, 'serverpoolCfg', 'serverpoolCfg'): {
                        'state': 0,
                        'value': {
                            (6, 'serverpoolRel', 'serverpoolRel'): {
                                'state': 0,
                                'target': 'serverpool'
                            }
                        }
                    },
                    (4, 'vipCfg', 'vipcfg'): {
                        'state': 0,
                        'value': {
                            (6, 'vipRel', 'vipRel'): {
                                'state': 0,
```

```
                                'target': 'network/vipaddress'
                            }
                        }
                    }
                }
            }
        }
    },
    (4, 'Network', 'network'): {
        'state': 0,
        'value': {
            (4, 'ip', 'externalIP'): {
                'state': 0,
                'value': {
                    (5, 'ipaddress', 'ipaddress'): {
                        'state': 0,
                        'value': '20.0.0.1'
                    },
                    (5, 'netmask', 'netmask'): {
                        'state': 0,
                        'value': '255.255.255.0'
                    }
                }
            },
            (4, 'ip', 'internalIP'): {
                'state': 0,
                'value': {
                    (5, 'ipaddress', 'ipaddress'): {
                        'state': 0,
                        'value': '30.0.0.1'
                    },
                    (5, 'netmask', 'netmask'): {
                        'state': 0,
                        'value': '255.255.255.0'
                    }
                }
            },
            (4, 'ip', 'vipaddress'): {
                'state': 0,
                'value': {
                    (5, 'ipaddress', 'ipaddress'): {
                        'state': 0,
                        'value': '100.0.0.1'
                    },
                    (5, 'netmask', 'netmask'): {
                        'state': 0,
                        'value': '255.255.255.255'
                    }
                }
            }
        }
    },
    (4, 'lbvserver', 'lbvserver'): {
        'state': 0,
        'value': {
            (5, 'ipmask', 'ipmask'): {
                'state': 0,
                'value': '255.255.255.255'
            },
            (5, 'ipv4', 'ipv4'): {
                'state': 0,
                'value': '100.0.0.1'
            },
            (5, 'name', 'name'): {
                'state': 0,
                'value': 'vserver1'
            },
            (5, 'port', 'port'): {
                'state': 0,
                'value': '80'
            },
            (5, 'servicetype', 'servicetype'): {
                'state': 0,
```

```
                        'value': 'http'
                    }
                }
            },
            (4, 'serverpool', 'serverpool'): {
                'state': 0,
                'value': {
                    (4, 'server', 'server1'): {
                        'state': 0,
                        'value': {
                            (5, 'ip', 'ip'): {
                                'state': 0,
                                'value': '30.0.0.2'
                            },
                            (5, 'port', 'port'): {
                                'state': 0,
                                'value': '80'
                            }
                        }
                    },
                    (5, 'serverpoolname', 'serverpoolname'): {
                        'state': 0,
                        'value': 'webpool'
                    },
                    (5, 'type', 'type'): {
                        'state': 0,
                        'value': 'http'
                    }
                }
            },
            (7, '', '2424832_32771'): {
                'state': 0,
                'tag': 436,
                'type': 1
            },
            (7, '', '2424832_49154'): {
                'state': 0,
                'tag': 370,
                'type': 1
            },
            (8, '', 'LB_external_2424832_32771'): {
                'encap': '2424832_32771',
                'state': 0,
                'vif': 'LB_external'
            },
            (8, '', 'LB_internal_2424832_49154'): {
                'encap': '2424832_49154',
                'state': 0,
                'vif': 'LB_internal'
            },
            (10, '', 'LB_external'): {
                'cifs': {
                    'Device1': 'eth1_0'
                },
                'state': 0
            },
            (10, '', 'LB_internal'): {
                'cifs': {
                    'Device1': 'eth1_1'
                },
                'state': 0
            }
        }
    }
}
)
```

Function: attachEndpoint

Arguments:

```
(
```

```
{'creds': {
    'password': 'insieme', 'username': 'admin'
},
'devs': {
    'Device1': {
        'creds': {
        'password': 'insieme', 'username': 'admin'
    },
    'host': '10.0.0.2',
    'port': 443,
    'virtual': False
}
},
'host': '10.0.0.1',
'name': 'LB',
'port': 443,
'virtual': False
},

{(0, '', 5539): {
    'ctxName': 'TenantActx1',
    'state': 2,
    'tenant': 'TenantA',
    'value': {
        (1, '', 4411): {
            'absGraph': 'WebGraph',
            'state': 2,
            'value': {
                (3, 'LoadBalancing', 'SLB'): {
                    'state': 2,
                    'value': {
                        (2, 'external', 'external'): {
                            'state': 2,
                            'value': {
                                (9, '', 'LB_external_2424832_32771'): {
                                    'state': 0,
                                    'target': 'LB_external_2424832_32771'
                                }
                            }
                        },
                        (2, 'internal', 'internal'): {
                            'state': 2,
                            'value': {
                                (9, '', 'LB_internal_2424832_49154'): {
                                    'state': 0,
                                    'target': 'LB_internal_2424832_49154'
                                }
                            }
                        },
                        (4, 'externalNetwork', 'externalNetwork'): {
                            'connector': 'external',
                            'value': {
                                (6, 'externalNetworkRel', 'externalNetworkRel'): {
                                    'state': 0,
                                    'target': 'network/externalIP'
                                }
                            }
                        },
                        (4, 'internalNetwork', 'internalNetwork'): {
                            'connector': 'internal',
                            'state': 0,
                            'value': {
                                (6, 'internalNetworkRel', 'internalNetworkRel'): {
                                    'state': 0,
                                    'target': 'network/internalIP'
                                }
                            }
                        },
                        (4, 'lbvserverCfg', 'lbvserverCfg'): {
                            'connector': 'external',
                            'state': 0,
                            'value': {
                                (6, 'lbvserverRel', 'lbvserverRel'): {
```

```
                                    'state': 0,
                                    'target': 'lbvserver'
                                }
                            }
                        },
                        (4, 'serverpoolCfg', 'serverpoolCfg'): {
                            'state': 0,
                            'value': {
                            (6, 'serverpoolRel', 'serverpoolRel'): {
                                'state': 0,
                                'target': 'serverpool'
                            }
                            }
                        },
                        (4, 'vipCfg', 'vipcfg'): {
                            'state': 0,
                            'value': {
                                (6, 'vipRel', 'vipRel'): {
                                    'state': 0,
                                    'target': 'network/vipaddress'
                                }
                            }
                        }
                    }
                }
            }
        },
        (4, 'Network', 'network'): {
            'state': 0,
            'value': {
                (4, 'ip', 'externalIP'): {
                    'state': 0,
                    'value': {
                        (5, 'ipaddress', 'ipaddress'): {
                            'state': 0,
                            'value': '20.0.0.1'
                        },
                        (5, 'netmask', 'netmask'): {
                            'state': 0,
                            'value': '255.255.255.0'
                        }
                    }
                },
                (4, 'ip', 'internalIP'): {
                    'state': 0,
                    'value': {
                        (5, 'ipaddress', 'ipaddress'): {
                            'state': 0,
                            'value': '30.0.0.1'
                        },
                        (5, 'netmask', 'netmask'): {
                            'state': 0,
                            'value': '255.255.255.0'
                        }
                    }
                },
                (4, 'ip', 'vipaddress'): {
                    'state': 0,
                    'value': {
                        (5, 'ipaddress', 'ipaddress'): {
                            'state': 0,
                            'value': '100.0.0.1'
                        },
                        (5, 'netmask', 'netmask'): {
                            'state': 0,
                            'value': '255.255.255.255'
                        }
                    }
                }
            }
        },
        (4, 'lbvserver', 'lbvserver'): {
            'state': 0,
```

```
        'value': {
            (5, 'ipmask', 'ipmask'): {
                'state': 0,
                'value': '255.255.255.255'
            },
            (5, 'ipv4', 'ipv4'): {
                'state': 0,
                'value': '100.0.0.1'
            },
            (5, 'name', 'name'): {
                'state': 0,
                'value': 'vserver1'
            },
            (5, 'port', 'port'): {
                'state': 0,
                'value': '80'
            },
            (5, 'servicetype', 'servicetype'): {
                'state': 0,
                'value': 'http'
            }
        }
    },
    (4, 'serverpool', 'serverpool'): {
        'state': 0,
        'value': {
            (4, 'server', 'server1'): {
                'state':
                'value': {
                    (5, 'ip', 'ip'): {
                        'state': 0,
                        'value': '30.0.0.2'
                    },
                    (5, 'port', 'port'): {
                        'state': 0,
                        'value': '80'
                    }
                }
            },
            (5, 'serverpoolname', 'serverpoolname'): {
                'state': 0,
                'value': 'webpool'
            },
            (5, 'type', 'type'): {
                'state': 0,
                'value': 'http'
            }
        }
    },
    (7, '', '2424832_32771'): {
        'state': 0,
        'tag': 436,
        'type': 1
    },
    (7, '', '2424832_49154'): {
        'state': 0,
        'tag': 370,
        'type': 1
    },
    (8, '', 'LB_external_2424832_32771'): {
        'encap': '2424832_32771',
        'state': 0,
        'vif': 'LB_external'
    },
    (8, '', 'LB_internal_2424832_49154'): {
        'encap': '2424832_49154',
        'state': 0,
        'vif': 'LB_internal'
    },
    (10, '', 'LB_external'): {
        'cifs': {
            'Device1': 'eth1_0'
        },
```

```
        'state': 0
    },
    (10, '', 'LB_internal'): {
        'cifs': {
            'Device1': 'eth1_1'
        },
        'state': 0
    }
}
}
},
[
    {
    'addr': '34.34.34.12',
    'conn': 'internal'
    }
]
)
```

The `endpoints` dictionary in the API callout contains the following attributes:

- `'addr'`—The IP address of the endpoint that attached to an EPG.
- `'conn'`—The connector to which the EPG is attached directly or indirectly through other function nodes.

```
Function: attachNetwork
Arguments:

(
{'creds': {
    'password': 'insieme',
    'username': 'admin'
},
'devs': {
    'Device1': {
        'creds': {
            'password': 'insieme',
            'username': 'admin'
        },
        'host': '10.0.0.2',
        'port': 443,
        'virtual': False
    }
},
'host': '10.0.0.1',
'name': 'LB',
'port': 443,
'virtual': False
},

{(0, '', 5539): {
    'ctxName': 'TenantActx1',
    'state': 2,
    'tenant': 'TenantA',
    'value': {
        (1, '', 4411): {
            'absGraph': 'WebGraph',
            'state': 2,
            'value': {
                (3, 'LoadBalancing', 'SLB'): {
                    'state': 2,
                    'value': {
                        (2, 'external', 'external'): {
                            'state': 2,
                            'value': {
                                (9, '', 'LB_external_2424832_32771'): {
                                    'state': 0,
                                    'target': 'LB_external_2424832_32771'
                                }
                            }
                        },
```

```
                    (2, 'internal', 'internal'): {
                        'state': 2,
                        'value': {
                            (9, '', 'LB_internal_2424832_49154'): {
                                'state': 0,
                                'target': 'LB_internal_2424832_49154'
                            }
                        }
                    },
                    (4, 'externalNetwork', 'externalNetwork'): {
                        'connector': 'external',
                        'state': 0,
                        'value': {
                            (6, 'externalNetworkRel', 'externalNetworkRel'): {
                                'state': 0,
                                'target': 'network/externalIP'
                            }
                        }
                    },
                    (4, 'internalNetwork', 'internalNetwork'): {
                        'connector': 'internal',
                        'state': 0,
                        'value': {
                            (6, 'internalNetworkRel', 'internalNetworkRel'): {
                                'state': 0,
                                'target': 'network/internalIP'
                            }
                        }
                    },
                    (4, 'lbvserverCfg', 'lbvserverCfg'): {
                        'connector': 'external',
                        'state': 0,
                        'value': {
                            (6, 'lbvserverRel', 'lbvserverRel'): {
                                'state': 0,
                                'target': 'lbvserver'
                            }
                        }
                    },
                    (4, 'serverpoolCfg', 'serverpoolCfg'): {
                        'state': 0,
                        'value': {
                            (6, 'serverpoolRel', 'serverpoolRel'): {
                                'state': 0,
                                'target': 'serverpool'
                            }
                        }
                    },
                    (4, 'vipCfg', 'vipcfg'): {
                        'state': 0,
                        'value': {
                            (6, 'vipRel', 'vipRel'): {
                                'state': 0,
                                'target': 'network/vipaddress'
                            }
                        }
                    }
                }
            }
        }
    },
    (4, 'Network', 'network'): {
        'state': 0,
        'value': {
            (4, 'ip', 'externalIP'): {
                'state': 0,
                'value': {
                    (5, 'ipaddress', 'ipaddress'): {
                        'state': 0,
                        'value': '20.0.0.1'
                    },
                    (5, 'netmask', 'netmask'): {
                        'state': 0,
```

```
                        'value': '255.255.255.0'
                    }
                }
            },
            (4, 'ip', 'internalIP'): {
                'state': 0,
                'value': {
                    (5, 'ipaddress', 'ipaddress'): {
                        'state': 0,
                        'value': '30.0.0.1'
                    },
                    (5, 'netmask', 'netmask'): {
                        'state': 0,
                        'value': '255.255.255.0'
                    }
                }
            },
            (4, 'ip', 'vipaddress'): {
                'state': 0,
                'value': {
                    (5, 'ipaddress', 'ipaddress'): {
                        'state': 0,
                        'value': '100.0.0.1'
                    },
                    (5, 'netmask', 'netmask'): {
                        'state': 0,
                        'value': '255.255.255.255'
                    }
                }
            }
        }
    },
    (4, 'lbvserver', 'lbvserver'): {
        'state': 0,
        'value': {
            (5, 'ipmask', 'ipmask'): {
                'state': 0,
                'value': '255.255.255.255'
            },
            (5, 'ipv4', 'ipv4'): {
                'state': 0,
                'value': '100.0.0.1'
            },
            (5, 'name', 'name'): {
                'state': 0,
                'value': 'vserver1'
            },
            (5, 'port', 'port'): {
                'state': 0,
                'value': '80'
            },
            (5, 'servicetype', 'servicetype'): {
                'state': 0,
                'value': 'http'
            }
        }
    },
    (4, 'serverpool', 'serverpool'): {
        'state': 0,
        'value': {
            (4, 'server', 'server1'): {
                'state': 0,
                'value': {
                    (5, 'ip', 'ip'): {
                        'state': 0,
                        'value': '30.0.0.2'
                    },
                    (5, 'port', 'port'): {
                        'state': 0,
                        'value': '80'
                    }
                }
            },
```

```
                (5, 'serverpoolname', 'serverpoolname'): {
                    'state': 0,
                    'value': 'webpool'
                },
                (5, 'type', 'type'): {
                    'state': 0,
                    'value': 'http'
                }
            }
        },
        (7, '', '2424832_32771'): {
            'state': 0,
            'tag': 436,
            'type': 1
        },
        (7, '', '2424832_49154'): {
            'state': 0,
            'tag': 370,
            'type': 1
        },
        (8, '', 'LB_external_2424832_32771'): {
            'encap': '2424832_32771',
            'state': 0,
            'vif': 'LB_external'
        },
        (8, '', 'LB_internal_2424832_49154'): {
            'encap': '2424832_49154',
            'state': 0,
            'vif': 'LB_internal'
        },
        (10, '', 'LB_external'): {
            'cifs': {
                'Device1': 'eth1_0'
            },
            'state': 0
        },
        (10, '', 'LB_internal'): {
            'cifs': {
                'Device1': 'eth1_1'
            },
            'state': 0
        }
    }
}
},
[
   {
    'addr': '34.34.34.0/24',
    'conn': 'internal'
   }
]
)
```

The `networks` dictionary in the API callout contains the following attributes:

- `'addr'`—Identifies the subnet configured in the bridge domain or EPG associated with the connector.
- `'conn'`—The connector to which the EPG is attached directly or indirectly through other function nodes.