



## Configuring a Service Graph

---

- [About Service Graphs, page 1](#)
- [About Function Nodes, page 1](#)
- [About Function Node Connectors, page 2](#)
- [About Service Graph Connections, page 2](#)
- [About Terminal Nodes, page 2](#)
- [About Service Graph Template Configuration Parameters, page 2](#)
- [Configuring Service Graph Templates Using the GUI, page 2](#)
- [Creating a Service Graph Template Using the REST APIs, page 2](#)
- [Configuring a Service Graph Using the NX-OS-Style CLI, page 3](#)

### About Service Graphs

A service graph is an ordered set of function nodes between a set of terminals, which identifies a set of network service functions that are required by an application. Service functions within a graph are automatically provisioned on a service device that is based on an application's requirements.

You can define a service graph by using the GUI, CLI, or the Application Policy Infrastructure Controller (APIC). Configuring a service device through the APIC does not require changes on a service device.

### About Function Nodes

A function node represents a single service function. A function node has function node connectors, which represent the network requirement of a service function.

A function node within a service graph can require one or more parameters. The parameters can be specified by an endpoint group (EPG), an application profile, or a tenant context. Parameters can also be assigned at the time that you define a service graph. The parameter values can be locked to prevent any additional changes.

## About Function Node Connectors

A function node connector connects a function node to the service graph and is associated with the appropriate bridge domain and connections based on the graph's connector's subset. Each connector is associated with a VLAN or Virtual Extensible LAN (VXLAN). Each side of a connector is treated as an endpoint group (EPG), and whitelists are downloaded to the switch to enable communication between the two function nodes.

## About Service Graph Connections

A service graph connection connects one function node to another function node.

## About Terminal Nodes

Terminal nodes connect a service graph with the contracts. You can insert a service graph for the traffic between two application endpoint groups (EPGs) by connecting the terminal node to a contract. Once connected, traffic between the consumer EPG and provider EPG of the contract is redirected to the service graph.

## About Service Graph Template Configuration Parameters

A service graph template can have configuration parameters, which are specified by the device package. Configuration parameters can also be specified by an EPG, application profile, or tenant context. A function node within a service graph template can require one or more configuration parameters. The parameter values can be locked to prevent any additional changes.

When you configure a service graph template and specify the values of the configuration parameters, the Application Policy Infrastructure Controller (APIC) passes the parameters to the device script that is within the device package. The device script converts the parameter data to the configuration that is downloaded onto the device.

## Configuring Service Graph Templates Using the GUI

You can configure the service graph templates using the GUI.

See [Using the GUI](#) for the procedure for configuring the service graph templates.

## Creating a Service Graph Template Using the REST APIs

You can create a service graph template using the following REST API:

```
<polUni>
  <fvTenant name="acme">
    <vnsAbsGraph name="G1">
      <vnsAbsTermNodeCon name="Input1">
        <vnsAbsTermConn name="C1">
          </vnsAbsTermConn>
        </vnsAbsTermNodeCon>
      <vnsAbsNode name="Node" funcType="GoTo">
```

```

<vnsRsDefaultScopeToTerm
  tDn="uni/tn-acme/AbsGraph-G1/AbsTermNodeProv-Output1/outtmnl"/>
<vnsAbsFuncConn name="inside">
  <vnsRsMConnAtt
    tDn="uni/infra/mDev-Insieme-Generic-1.0/mFunc-SubnetFunc/mConn-external"/>
</vnsAbsFuncConn>
<vnsAbsFuncConn name="outside">
  <vnsRsMConnAtt
    tDn="uni/infra/mDev-Insieme-Generic-1.0/mFunc-SubnetFunc/mConn-internal"/>
</vnsAbsFuncConn>
<vnsAbsDevCfg>
  <vnsAbsFolder key="oneFolder" name="f1">
    <vnsAbsParam key="oneParam" name="p1" value="v1"/>
  </vnsAbsFolder>
</vnsAbsDevCfg>
<vnsAbsFuncCfg>
  <vnsAbsFolder key="folder" name="folder1" devCtxLbl="C1">
    <vnsAbsParam key="param" name="param" value="value"/>
  </vnsAbsFolder>
  <vnsAbsFolder key="folder" name="folder2" devCtxLbl="C2">
    <vnsAbsParam key="param" name="param" value="value"/>
  </vnsAbsFolder>
</vnsAbsFuncCfg>
<vnsRsNodeToMFunc tDn="uni/infra/mDev-Insieme-Generic-1.0/mFunc-SubnetFunc"/>
</vnsAbsNode>
<vnsAbsTermNodeProv name="Output1">
  <vnsAbsTermConn name="C6">
    </vnsAbsTermConn>
</vnsAbsTermNodeProv>
<vnsAbsConnection name="CON1">
  <vnsRsAbsConnectionConns
    tDn="uni/tn-acme/AbsGraph-G1/AbsTermNodeCon-Input1/AbsTConn"/>
  <vnsRsAbsConnectionConns tDn="uni/tn-acme/AbsGraph-G1/AbsNode-Node/AbsFConn-inside"/>
</vnsAbsConnection>
<vnsAbsConnection name="CON3">
  <vnsRsAbsConnectionConns tDn="uni/tn-acme/AbsGraph-G1/AbsNode-Node/AbsFConn-outside"/>
  <vnsRsAbsConnectionConns
    tDn="uni/tn-acme/AbsGraph-G1/AbsTermNodeProv-Output1/AbsTConn"/>
</vnsAbsConnection>
</vnsAbsGraph>
</fvTenant>
</polUni>

```

## Configuring a Service Graph Using the NX-OS-Style CLI

You can configure a service graph using the NX-OS-style CLI.

**Step 1** Enter the configure mode.

**Example:**

```
apic1# configure
```

**Step 2** Enter the configure mode for a tenant.

```
tenant tenant_name
```

**Example:**

```
apicl(config)# tenant t1
```

**Step 3**

Add a service graph.

```
1417 graph graph_name [contract contract_name]
```

Parameter	Description
graph	Name of the service graph.
contract	Name of the contract that is associated with this service graph instance. Specify the contract only if you want to create the service graph instance. You can simply configure a service graph (equivalent to the service graph template) without instantiating it.

**Example:**

```
apicl(config-tenant)# 1417 graph G2 contract C2
```

**Step 4**

Add a node (service) in the service graph.

```
service node_name [device-cluster-tenant tenant_name] [device-cluster device_name] [mode deployment_mode]
```

Parameter	Description
service	The name of the service node to add.
device-cluster-tenant	The tenant from which to import the device cluster. Specify this only if the device-cluster is not in the same tenant in which the graph is being configured.
device-cluster	Name of the device cluster to use for this service node.
mode	<p>The deployment mode. Possible values are:</p> <ul style="list-style-type: none"> <li>• ADC_ONE_ARM—Specifies one-arm mode.</li> <li>• ADC_TWO_ARM—Specifies two-arm mode.</li> <li>• FW_ROUTED—Specifies routed (GoTo) mode.</li> <li>• FW_TRANS—Specifies transparent (GoThrough) mode.</li> <li>• OTHERS—Specifies any other deployment mode.</li> </ul> <p>If the mode is not specified, then a deployment mode is not used.</p>

**Example:**

The following example adds node N1 to the device cluster D4, which is from tenant t1:

```
apicl(config-graph)# service N1 device-cluster-tenant t1 device-cluster D4
```

The following example adds node `N1` to the device cluster `D4`, which is from tenant `t1`, and uses the routed deployment mode:

```
apicl(config-graph)# service N1 device-cluster-tenant t1 device-cluster D4 mode FW_ROUTED
```

**Step 5**

Add the consumer connector.

```
connector connector_type [cluster-interface interface_type]
```

Parameter	Description
connector	The type of the connector in the service graph. Possible values are: <ul style="list-style-type: none"> <li>• provider</li> <li>• consumer</li> </ul>
cluster-interface	The type of the device cluster interface. Possible values are: <ul style="list-style-type: none"> <li>• provider</li> <li>• consumer</li> </ul> Do not specify this parameter if you are a service graph template in tenant <code>Common</code> .

**Example:**

```
apicl(config-service)# connector consumer cluster-interface consumer
```

**Step 6**

Configure the bridge domain for the connectors by specifying the bridge domain information and tenant where the bridge domain is present.

```
bridge-domain tenant tenant_name name bridge_domain_name
```

Parameter	Description
tenant	Tenant that owns the bridge domain. You can only specify a bridge domain from same tenant or tenant <code>Common</code> . For example if you are in tenant <code>t1</code> , then you cannot specify the bridge domain from tenant <code>t2</code> .
name	Name of the bridge domain.

**Example:**

```
apicl(config-connector)# bridge-domain tenant t1 name bd2
```

**Step 7**

(Optional) Configure the direct server return (DSR) virtual IP address (VIP) for the connector.

```
dsr-vip ip_address
```

If you specify the DSR VIP, the Application Policy Infrastructure Controller (APIC) does not learn the VIP.

Parameter	Description
dsr-vip	The virtual IP address of the DSR for the connector.

**Example:**

```
apicl(config-connector)# dsr-vip 192.168.10.100
```

**Step 8**

Configure connections for the consumer and provider and exit the service graph configuration mode.

```
connection connection_name {terminal terminal_type service node_name connector connector_type} |
  {intra_service service1 node_name connector1 connector_type service2 node_name connector2
  connector_type}
exit
```

Parameter	Description
connection	The name of the connection.
terminal	Connects a service node to the terminal. Specifies the type of the terminal. Possible values are: <ul style="list-style-type: none"> <li>• provider</li> <li>• consumer</li> </ul>
service service1 service2	The name of the service node to add. service is used only with terminal. service1 and service2 are used only with intra_service.
connector connector1 connector2	The type of the connector. Possible values are: <ul style="list-style-type: none"> <li>• provider</li> <li>• consumer</li> </ul> connector is used only with terminal. connector1 and connector2 are used only with intra_service.
intra_service	Connects a service node to another node.

**Example:**

The following example configures the connections of a single node graph:

```
apicl(config-graph)# connection CON1 terminal consumer service N1 connector consumer
apicl(config-graph)# connection CON2 terminal provider service N2 connector provider
apicl(config-graph)# exit
```

The following example configures the connections of a two node graph:

```
apicl(config-graph)# connection CON1 terminal consumer service N1 connector consumer
apicl(config-graph)# connection CON2 intra_service service1 N1 connector1 provider service2 N2
connector2 consumer
apicl(config-graph)# connection CON3 terminal provider service N2 connector provider
apicl(config-graph)# exit
```

**Step 9**

Exit the configuration mode.

**Example:**

```
apic1(config-tenant)# exit
apic1(config)# exit
```

---

