Discovery 5: Create an L3VPN Template Service

Introduction

In this activity, you will learn how to create an L3MPLS VPN service. After completing this activity, you will be able to meet these objectives:

- Use the Cisco NSO CLI to configure the service model in YANG
- Create the service template
- Compile and deploy the service

Job Aids

The following job aids are available to help you complete the lab activities:

- This lab guide

Job Aids Task 1

The following table describes the service characteristics and requirements. Use this information as a starting point for creating a service model.

Attribute	Description
VPN ID	Unique identifier describing an instance of a deployed service
VPN name	VPN instance name
Customer	Customer to which the VPN instance belongs
Link	Customer link
Link ID	Unique identifier describing an instance of a list link
Interface	Interface on the PE router to which the customer site is connected
Routing protocol	Routing protocol option
Pe-device	Device on which the service will be deployed
Static-route	Static routing option
Prefix	Route prefix

Attribute	Description
Mask	Route mask

Job Aids Task 2

The following data will be required for this task.

A network engineer has provided you with the actual configuration for the new service.

Cisco IOS platform:

```
vrf definition vpn10001
 description Customer ACME VPN
 rd          1:10001
 route-target export 1:10001
 route-target import 1:10001
 address-family ipv4
  exit-address-family
 !
!
ip route vrf vpn10001 192.168.11.0 255.255.255.0 172.31.1.2
interface GigabitEthernet4
 description Connection to Customer ACME
 vrf forwarding vpn10001
 ip address 172.31.1.1 255.255.255.252
exit
router bgp 1
 address-family ipv4 unicast vrf vpn10001
  neighbor 172.31.1.2 remote-as 65001
  neighbor 172.31.1.2 activate
  neighbor 172.31.1.2 allowas-in
  neighbor 172.31.1.2 as-override disable
  neighbor 172.31.1.2 default-originate
  redistribute connected
  redistribute static
  exit-address-family
 !
!
```

Device Information

Device	Description	IP Address	Credentials
Student	Linux Ubuntu VM	10.10.20.50	developer, C1sco12345
R1 (dist-rtr01)	Virtual Router, IOS XE - Amsterdam-17.3.4	10.10.20.175	cisco, cisco
R2 (dist-rtr02)	Virtual Router, IOS XE - Amsterdam-17.3.4	10.10.20.176	cisco, cisco
R3 (internet-rtr01)	Virtual Router, IOS XE - Amsterdam-17.3.4	172.21.1.181	cisco, cisco

Task 1: Design a Service Model

The first task requires you to design a service model based on the provided high-level service requirements.

Activity

Complete these steps:

Step 1 Connect to the Student DevBox.

On the topology diagram, click the **Student DevBox**.

Step 2 Open the terminal window using the Terminal icon on the bottom bar.

```
developer@devbox:~$
```

Step 3 Change your current location to the directory $HOME/nso-run/packages.

```
developer@devbox:~$ cd $HOME/nso-run/packages
```

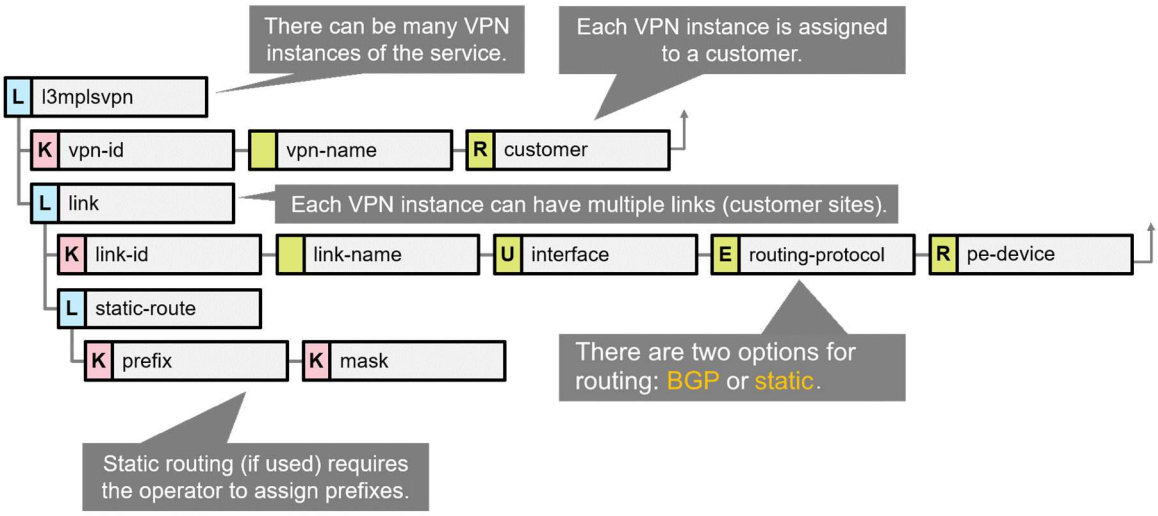
Step 4 Create a template-based service skeleton by using the **ncs-make-package** command. Use **l3vpn** as the name of the new service.

Note You can use **ncs-make-package --help** or **man ncs-make-package** to learn about all the available options.

```
developer@devbox:~/nso-run/packages$ ls
cisco-ios-cli-6.67  dns-snmp-service
developer@devbox:~/nso-run/packages$ ncs-make-package --service-skeleton
template l3vpn
developer@devbox:~/nso-run/packages$ ls
cisco-ios-cli-6.67  dns-snmp-service  l3vpn
```

Step 5 Use the following table to fine-tune the service parameters that you will use in your YANG service model.

Attribute	Name	Data Type	Restriction	Other
Service name	*l3vpn*	list	—	—
VPN ID	*vpn-id*	string		key for list *l3vpn*
VPN name				
Customer				
Link				
Link ID				
Interface				
Routing protocol				
PE - device				
Static-route				
Prefix				
Mask				



Step 6 Navigate to **l3vpn/src/yang** directory and open the YANG service model for editing with **code l3vpn.yang** command.

```
developer@devbox:~/nso-run/packages$ cd l3vpn/src/yang
developer@devbox:~/nso-run/packages/l3vpn/src/yang$ code l3vpn.yang
```

Note	You can edit the YANG service model YANG file by using Visual Studio Code or any editor of your choice. Visual Studio Code text editor on the workstation will provide you with syntax highlighting for YANG.

Step 7 Rename the namespace to *http://cisco.com/example/l3vpn* and remove the leaf-list *device* and the leaf *dummy*. They will not be used for the l3vpn service and will be replaced by other statements. Add an *augment /ncs:services* statement, to include the list *l3vpn* in the services branch of the CDB.

```
module l3vpn {
  namespace "http://cisco.com/example/l3vpn";
  prefix l3vpn;

  import ietf-inet-types {
    prefix inet;
  }
  import tailf-ncs {
    prefix ncs;
  }

  augment /ncs:services {
    list l3vpn {
      key name;

      uses ncs:service-data;
      ncs:servicepoint "l3vpn";

      leaf name {
        type string;
      }
```

```
    }
  }
}
```

Note YANG allows a module to insert additional nodes into existing data models. The "augment" statement defines the location in the data model hierarchy where new nodes are inserted. In your case, the location is under the **services** branch of the CDB.

Step 8 Rename *name* leaf to *vpn-name* leaf. Define a vpn-name leaf as a key to the service list. A list is used to support multiple instances of the l3vpn service. The list records are uniquely identified, using the key attribute (vpn-name). Add another import statement for tailf-common, to support YANG annotations, and add node descriptions for syntax help.

```
module l3vpn {
  namespace "http://cisco.com/example/l3vpn";
  prefix l3vpn;

  import ietf-inet-types {
    prefix inet;
  }
  import tailf-ncs {
    prefix ncs;
  }
  import tailf-common {
    prefix tailf;
  }

  augment /ncs:services {
    list l3vpn {
      tailf:info "L3VPN Service";
      key vpn-name;

      uses ncs:service-data;
      ncs:servicepoint "l3vpn";

      leaf vpn-name {
        tailf:info "Service Instance Name";
        type string;
      }
    }
  }
}
```

Step 9 Assign a VPN ID to each l3vpn service instance.

```
module l3vpn {

  ...

  augment /ncs:services {
    list l3vpn {
      tailf:info "L3VPN Service";
      key vpn-name;
```

```
      uses ncs:service-data;
      ncs:servicepoint "l3vpn";

      leaf vpn-name {
        tailf:info "Service Instance Name";
        type string;
      }

      leaf vpn-id {
        tailf:info "Service Instance ID";
        type uint32 {
          range "10001..19999";
        }
      }

    }
  }
}
```

Step 10 Assign a customer reference to every l3vpn service instance.

```
module l3vpn {

  ...

  augment /ncs:services {
    list l3vpn {

      ...

      leaf vpn-id {
        tailf:info "Service Instance ID";
        type uint32 {
          range "10001..19999";
        }
      }

      leaf customer {
        tailf:info "VPN Customer";
        type leafref {
          path "/ncs:customers/ncs:customer/ncs:id";
        }
      }

    }
  }
}
```

Step 11 Add a list of links. Each VPN service instance can support multiple customer links, which are represented with the list link. A link instance gets a unique link ID and a link name.

```
module l3vpn {

  ...

  augment /ncs:services {
    list l3vpn {

      ...

      leaf customer {
        tailf:info "VPN Customer";
        type leafref {
          path "/ncs:customers/ncs:customer/ncs:id";
        }
      }

      // Each VPN service instance can have one or more interfaces
      list link {
        tailf:info "PE-CE Attachment Point";
        key link-id;

        leaf link-id {
           tailf:info "Link ID";
           type uint32 {
              range "1..255";
           }
        }

        leaf link-name {
           tailf:info "Link Name";
           type string;
        }
      }

    }
  }
}
```

Step 12 Assign each link instance to the interface on the selected PE device. Add additional *unique* restriction, which will prevent duplicate entries. Interface on a specific device can be used only for one link.

```
      ...

      // Each VPN service instance can have one or more interfaces
      list link {
        tailf:info "PE-CE Attachment Point";
        key link-id;
        unique "pe-device interface";

        leaf link-id {
           tailf:info "Link ID";
           type uint32 {
```

```
            range "1..255";
          }
        }

        leaf link-name {
          tailf:info "Link Name";
          type string;
        }

        leaf pe-device {
          tailf:info "PE Router";
          type leafref {
            path "/ncs:devices/ncs:device/ncs:name";
          }
        }

        leaf interface {
          tailf:info "Customer Facing Interface";
          type string;
        }

      }

      ...
```

Step 13 Use the XPath function *starts-with()* to constrain which devices the administrator can input as *pe-device*. To get the current node value, you have to use the *current()* function, which returns the context node in the current expression. You can also add a relevant error message in case of an incorrect input.

Consider the following facts about the current model:

- *pe-device* can be any device currently added to NSO. This should be limited to include only "R" devices.
- XPath supports a wide variety of functions that include string, numbers, Booleans and nodes, such as *position(), contains(), floor(), substring(), starts-with*(), and more. By using these functions in combination with YANG statements such as *must*, we can constrain the data that is valid for a specific model. Consider which functions can be used to constrain this model.

Note You can find a list and an explanation of all the XPath functions here - https://www.w3.org/TR/1999/REC-xpath-19991116/

```
        ...

        leaf pe-device {
          tailf:info "PE Router";
          type leafref {
            path "/ncs:devices/ncs:device/ncs:name";
          }
          must "starts-with(current(),'R')" {
            error-message "Only R devices can be selected.";
          }
        }

        ...
```

Step 14 Add a routing-protocol option to each instance link. If static routing is used, route prefix and route mask attributes are used. A "*when*" statement is used to make its parent data definition conditional. The "*when*" statement's argument is an XPath expression. For this particular step, use the wrong XPath argument – "/routing-protocol='static'" to get an XPath error when compiling the YANG file. Use **type union** for the mask because Cisco IOS demands an IP address with a subnet mask and other platforms, such as Cisco IOS XR demands just an IP prefix (with CIDR annotation).

```
        ...

        leaf interface {
          tailf:info "Customer Facing Interface";
          type string;
        }

        leaf routing-protocol {
          tailf:info "Routing option on PE-CE link";
          type enumeration {
            enum bgp;
            enum static;
          }
        }

        list static-route  {
          tailf:info "Static Route";
          key prefix;
          when "/routing-protocol='static'";

          leaf prefix {
            tailf:info "Static Route Prefix";
            type inet:ipv4-address;
          }

          leaf mask {
            tailf:info "Subnet Mask or Prefix";
            type union {
                type inet:ipv4-address;
                type inet:ipv4-prefix;
            }
          }
        }

        ...
```

Step 15 Verify the service model contents and save the file.

Your YANG service model should resemble the following output:

```
module l3vpn {
  namespace "http://cisco.com/example/l3vpn";
  prefix l3vpn;

  import ietf-inet-types {
    prefix inet;
  }
  import tailf-ncs {
```

```
  prefix ncs;
}
import tailf-common {
  prefix tailf;
}

augment /ncs:services {
  list l3vpn {
    tailf:info "L3VPN Service";
    key vpn-name;

    uses ncs:service-data;
    ncs:servicepoint "l3vpn";

    leaf vpn-name {
      tailf:info "Service Instance Name";
      type string;
    }

    leaf vpn-id {
      tailf:info "Service Instance ID";
      type uint32 {
        range "10001..19999";
      }
    }

    leaf customer {
      tailf:info "VPN Customer";
      type leafref {
        path "/ncs:customers/ncs:customer/ncs:id";
      }
    }

    // Each VPN service instance can have one or more interfaces
    list link {
      tailf:info "PE-CE Attachment Point";
      key link-id;
      unique "pe-device interface";

      leaf link-id {
         tailf:info "Link ID";
         type uint32 {
            range "1..255";
         }
      }

      leaf link-name {
         tailf:info "Link Name";
         type string;
      }

      leaf pe-device {
        tailf:info "PE Router";
        type leafref {
          path "/ncs:devices/ncs:device/ncs:name";
        }
        must "starts-with(current(),'R')" {
          error-message "Only R devices can be selected.";
        }
      }
```

```
        leaf interface {
          tailf:info "Customer Facing Interface";
          type string;
        }

        leaf routing-protocol {
          tailf:info "Routing option on PE-CE link";
          type enumeration {
            enum bgp;
            enum static;
          }
        }

        list static-route  {
          tailf:info "Static Route";
          key prefix;
          when "/routing-protocol='static'";

          leaf prefix {
            tailf:info "Static Route Prefix";
            type inet:ipv4-address;
          }

          leaf mask {
            tailf:info "Subnet Mask for IOS and Prefix for IOSXR";
            type union {
                type inet:ipv4-address;
                type inet:ipv4-prefix;
            }
          }
        }

      }

    }
  }
}
```

Step 16 Use the **make** command to compile the package.

```
developer@devbox:~/nso-run/packages/l3vpn/src/yang$ cd ..
developer@devbox:~/nso-run/packages/l3vpn/src$ make
mkdir -p ../load-dir
/home/student/nso/bin/ncsc `ls l3vpn-ann.yang  > /dev/null 2>&1 && echo "-a
l3vpn-ann.yang"` \
        --fail-on-warnings \
         \
        -c -o ../load-dir/l3vpn.fxs yang/l3vpn.yang
yang/l3vpn.yang:86: error: The XPath expression references an undefined node:
the node 'routing-protocol' from module 'l3vpn' is not found
```

Step 17 You can see that the 'routing-protocol' node cannot be found because the XPath is not correct. When you are writing "when" statements, always make sure that the desired path is correct. In this specific case you are specifying that the 'routing-protocol' node is on the same level as the 'static-route' node. You must go outside the current (list static-route) node in order to address the 'routing-protocol' node. Edit the l3vpn.yang file and append "**..**" to the **/routing-protocol='static'** statement.

```
        ...

        leaf routing-protocol {
          tailf:info "Routing option on PE-CE link";
          type enumeration {
            enum bgp;
            enum static;
          }
        }

        list static-route  {
          tailf:info "Static Route";
          key prefix;
          when "../routing-protocol='static'";

          leaf prefix {
            tailf:info "Static Route Prefix";
            type inet:ipv4-address;
          }

          leaf mask {
            tailf:info "Subnet Mask or Prefix";
            type union {
                type inet:ipv4-address;
                type inet:ipv4-prefix;
            }
          }
        }

        ...
```

Step 18 Save the file. Use the **make** command to compile the package again.

```
developer@devbox:~/nso-run/packages/l3vpn/src$ make
/home/student/nso/bin/ncsc `ls l3vpn-ann.yang  > /dev/null 2>&1 && echo "-a
l3vpn-ann.yang"` \
        --fail-on-warnings \
         \
        -c -o ../load-dir/l3vpn.fxs yang/l3vpn.yang
developer@devbox:~/nso-run/packages/l3vpn/src$
```

Step 19 After the package is compiled, log in to the NSO CLI and issue the **packages reload** command.

```
developer@devbox:~/nso-run/packages/l3vpn/src$ ncs_cli -Cu admin
admin@ncs# packages reload

>>> System upgrade is starting.
>>> Sessions in configure mode must exit to operational mode.
```

```
>>> No configuration changes can be performed until upgrade has completed.
>>> System upgrade has completed successfully.
reload-result {
    package cisco-ios-cli-6.67
    result true
}
reload-result {
    package dns-snmp-service
    result true
}
reload-result {
    package l3vpn
    result true
}
admin@ncs#
System message at 2022-09-14 17:03:27...
    Subsystem stopped: ncs-dp-3-cisco-ios-cli-6.67:IOSDp
admin@ncs#
System message at 2022-09-14 17:03:27...
    Subsystem started: ncs-dp-5-cisco-ios-cli-6.67:IOSDp
admin@ncs#
```

Step 20 Check the availability of CLI commands to manage the service. For example, use the **services l3vpn** command, followed by a question mark, to investigate the new CLI options that are available for the provisioning of service instances.

```
admin@ncs# services ?
Possible completions:
  check-sync         Check if device configuration is according to the services
  dns-snmp-service   DNS and SNMP configuration service
  l3vpn              L3VPN Service
admin@ncs# services l3vpn ?
% No entries found
```

Step 21 Exit NSO CLI. Verify that the directory structure and file templates for the new service match the expected structure.

```
developer@devbox:~$ cd
developer@devbox:~$ cd nso-run/packages/
developer@devbox:~/nso-run/packages$ ls -l l3vpn/
total 20
drwxrwxr-x 2 student student 4096 Sep 14 17:02 load-dir
-rw-rw-r-- 1 student student  368 Sep 14 15:05 package-meta-data.xml
drwxr-xr-x 3 student student 4096 Sep 14 15:05 src
drwxr-xr-x 2 student student 4096 Sep 14 15:05 templates
drwxr-xr-x 3 student student 4096 Jul  8 09:56 test
developer@devbox:~/nso-run/packages$ ls -l l3vpn/src/yang
total 4
-rw-rw-r-- 1 student student 2325 Sep 14 16:53 l3vpn.yang
developer@devbox:~/nso-run/packages$ ls -l l3vpn/templates
total 4
-rw-rw-r-- 1 student student 733 Sep 14 15:05 l3vpn-template.xml
```

Activity Verification

You have completed this task when you attain this result:

- You have a directory structure and file templates for a new service that matches the expected structure.

Note	You are still unable to provision services because the service template is only a placeholder at this point. You will develop it properly in the next task.

Task 2: Create Configuration Templates

The purpose of this task is to configure the sample l3vpn service from the NSO CLI, in order to obtain the device-specific configuration in XML format. This will allow you to create a device template for the Cisco IOS and Cisco IOS XR platforms.

Activity

Complete these steps:

Step 1 Connect to the NSO CLI.

```
developer@devbox:~/nso-run/packages$ ncs_cli -Cu admin
admin@ncs#
```

Step 2 Because you are using the same sandbox environment for all the discoveries, make sure to clean up any leftover configuration to produce the correct XML configuration template output.

Run the commands in the configuration mode. Make sure you **commit** the changes.

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# no devices device R1 config interface GigabitEthernet 6
description
admin@ncs(config)# no devices device R1 config interface GigabitEthernet 6 ip
admin@ncs(config)# no devices device R2 config interface GigabitEthernet 6
description
admin@ncs(config)# no devices device R2 config interface GigabitEthernet 6 ip
admin@ncs(config)# commit
Commit complete.
admin@ncs(config)#
```

Step 3 Configure the following examples from the NSO CLI.

Use the **devices device R1** configuration command as a starting point to configure the Cisco IOS sample.

```
devices device R1
 config
  vrf definition vpn10001
   description L3VPN for Customer ACME
   rd          1:10001
   route-target export 1:10001
   route-target import 1:10001
   address-family ipv4
    exit-address-family
   !
  !
  ip route vrf vpn10001 192.168.11.0 255.255.255.0 172.31.1.2
  interface GigabitEthernet6
   description Connection to Customer ACME
   vrf forwarding vpn10001
   ip address 172.31.1.1 255.255.255.252
  exit
  router bgp 1
   address-family ipv4 unicast vrf vpn10001
    ! first
    neighbor 172.31.1.2 remote-as 65001
    neighbor 172.31.1.2 activate
    neighbor 172.31.1.2 allowas-in
    neighbor 172.31.1.2 default-originate
    redistribute connected
    redistribute static
    exit-address-family
   !
  !
 !
!
```

Note	You can configure the sample from the CLI in NSO or copy and paste the whole configuration sample. To do this, enter config mode and enter the command **load merge terminal.** Next copy and paste the entire configuration. When you are finished pasting the configuration press **Enter** to create a new line and then press **CTRL+D** on your keyboard to parse the configuration.

Step 4 Use the **commit dry-run outformat xml** command to retrieve the XML version of the configuration for the Cisco IOS and Cisco IOS XR platforms. Copy and save these outputs as you will use them later.

Note	Verify that the output contains all the configured parameters. If some or all of the changes were committed earlier, the output will be missing those parts.

```
admin@ncs(config)# commit dry-run outformat xml
result-xml {
    local-node {
        data <devices xmlns="http://tail-f.com/ns/ncs">
               <device>
                 <name>R1</name>
                 <config>
                   <vrf xmlns="urn:ios">
                     <definition>
                       <name>vpn10001</name>
```

```
      <description>L3VPN for Customer ACME</description>
      <rd>1:10001</rd>
      <route-target>
        <export>
          <asn-ip>1:10001</asn-ip>
        </export>
        <import>
          <asn-ip>1:10001</asn-ip>
        </import>
      </route-target>
      <address-family>
        <ipv4/>
      </address-family>
    </definition>
  </vrf>
  <ip xmlns="urn:ios">
    <route>
      <vrf>
        <name>vpn10001</name>
        <ip-route-forwarding-list>
          <prefix>192.168.11.0</prefix>
          <mask>255.255.255.0</mask>
          <forwarding-address>172.31.1.2</forwarding-address>
        </ip-route-forwarding-list>
      </vrf>
    </route>
  </ip>
  <interface xmlns="urn:ios">
    <GigabitEthernet>
      <name>6</name>
      <description>Connection to Customer ACME</description>
      <vrf>
        <forwarding>vpn10001</forwarding>
      </vrf>
      <ip>
        <address>
          <primary>
            <address>172.31.1.1</address>
            <mask>255.255.255.252</mask>
          </primary>
        </address>
      </ip>
    </GigabitEthernet>
  </interface>
  <router xmlns="urn:ios">
    <bgp>
      <as-no>1</as-no>
      <address-family>
        <with-vrf>
          <ipv4>
            <af>unicast</af>
            <vrf>
              <name>vpn10001</name>
              <redistribute>
                <connected/>
                <static/>
              </redistribute>
              <neighbor>
                <id>172.31.1.2</id>
                <remote-as>65001</remote-as>
```

```
                                  <activate/>
                                  <allowas-in/>
                                  <default-originate/>
                                </neighbor>
                              </vrf>
                            </ipv4>
                          </with-vrf>
                        </address-family>
                      </bgp>
                    </router>
                  </config>
                </device>
              </devices>
    }
}
admin@ncs(config)# abort
admin@ncs# exit
```

Note	You can save the output of the dry-run with by piping the output to the file. The following example will save the output to a file name **configuration.xml** in a directory from where you invoked **ncs_cli** command: **commit dry-run outformat xml | save configuration.xml**

Step 5 Go to the templates subdirectory of your package skeleton (for example, $HOME/nso-run/packages/l3vpn/templates).

```
developer@devbox:~$ cd $HOME/nso-run/packages/l3vpn/templates
developer@devbox:~/nso-run/packages/l3vpn/templates$
```

Step 6 Open the template by using the **code l3vpn-template.xml** command.

The following output shows how the template should look when you open it for the first time. The initial XML has two segments where you must include your XML configuration.

The first section describes the device that the template is configuring, and the second section contains configuration changes.

```
<config-template xmlns="http://tail-f.com/ns/config/1.0"
                 servicepoint="loopback">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <!--
          Select the devices from some data structure in the service
          model. In this skeleton the devices are specified in a leaf-list.
          Select all devices in that leaf-list:
      -->
      <name>{/device}</name>
      <config>
        <!--
            Add device-specific parameters here.
            In this skeleton the service has a leaf "dummy"; use that
            to set something on the device e.g.:
            <ip-address-on-device>{/dummy}</ip-address-on-device>
        -->
```

```
      </config>
    </device>
  </devices>
</config-template>
```

Because the service model supports two types of devices, the XML configuration changes segment must accommodate both of them. Remove the provided comments under the <device> and <config> tags and add your own (DEVICE, IOS, IOS-XR).

```
<config-template xmlns="http://tail-f.com/ns/config/1.0"
                 servicepoint="l3vpn">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <!-- DEVICE -->
    <device>
      <name>{/device}</name>
      <config>

        <!-- IOS -->


      </config>
    </device>
  </devices>
</config-template>
```

Step 7 Insert the XML configuration created with **commit dry-run outformat xml** in the modified XML skeleton sections.

Because the service model supports two types of devices, the XML configuration changes segment must accommodate both of them. Edit the segment with XML configuration code. Add both the Cisco IOS and Cisco IOS XR parts of the configuration.

```
<config-template xmlns="http://tail-f.com/ns/config/1.0" servicepoint="l3vpn">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <!-- DEVICE -->
    <device>
      <name>{/device}</name>
      <config>

        <!-- IOS -->
        <vrf xmlns="urn:ios">
          <definition>
            <name>vpn10001</name>
            <description>L3VPN for Customer ACME</description>
            <rd>1:10001</rd>
            <route-target>
              <export>
                <asn-ip>1:10001</asn-ip>
              </export>
              <import>
                <asn-ip>1:10001</asn-ip>
              </import>
            </route-target>
            <address-family>
              <ipv4/>
```

```
      </address-family>
    </definition>
  </vrf>
  <ip xmlns="urn:ios">
    <route>
      <vrf>
        <name>vpn10001</name>
        <ip-route-forwarding-list>
          <prefix>192.168.11.0</prefix>
          <mask>255.255.255.0</mask>
          <forwarding-address>172.31.1.2</forwarding-address>
        </ip-route-forwarding-list>
      </vrf>
    </route>
  </ip>
  <interface xmlns="urn:ios">
    <GigabitEthernet>
      <name>4</name>
      <description>Connection to Customer ACME</description>
      <vrf>
        <forwarding>vpn10001</forwarding>
      </vrf>
      <ip>
        <address>
          <primary>
            <address>172.31.1.1</address>
            <mask>255.255.255.252</mask>
          </primary>
        </address>
      </ip>
    </GigabitEthernet>
  </interface>
  <router xmlns="urn:ios">
    <bgp>
      <as-no>1</as-no>
      <address-family>
        <with-vrf>
          <ipv4>
            <af>unicast</af>
            <vrf>
              <name>vpn10001</name>
              <redistribute>
                <connected/>
                <static/>
              </redistribute>
              <neighbor>
                <id>172.31.1.2</id>
                <remote-as annotation="first">65001</remote-as>
                <activate/>
                <allowas-in/>
                <default-originate/>
              </neighbor>
            </vrf>
          </ipv4>
        </with-vrf>
      </address-family>
    </bgp>
  </router>
```

```
    </config>
  </device>
</devices>
</config-template>
```

Step 8 Replace all the static parameters with variables, which reference service attributes according to the hierarchy of the YANG data model. Because you are implementing a l3vpn service template, the template must be able to apply multiple devices and static routes. Use XML *foreach* tags to create the logic that will handle this. Use **string()** in XPath when referencing to *vpn-id* so that the context node will not be changed. This needs to be done only with references to (yang model) keys, because the template processor detects it as a loop.

```
<config-template xmlns="http://tail-f.com/ns/config/1.0" servicepoint="l3vpn">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <!-- DEVICE -->
    <?foreach {/link}?>
      <device>
        <name>{pe-device}</name>
        <config>

          <!-- IOS -->
          <vrf xmlns="urn:ios">
            <definition>
              <name>vpn{string(../vpn-id)}</name>
              <description>By NSO: L3VPN - {string(/customer)}</description>
              <rd>1:{string(../vpn-id)}</rd>
              <route-target>
                <export>
                  <asn-ip>1:{string(../vpn-id)}</asn-ip>
                </export>
                <import>
                  <asn-ip>1:{string(../vpn-id)}</asn-ip>
                </import>
              </route-target>
              <address-family>
                <ipv4/>
              </address-family>
            </definition>
          </vrf>
          <?if {routing-protocol='static'}?>
            <ip xmlns="urn:ios">
              <route>
                <vrf>
                  <name>vpn{string(../vpn-id)}</name>
                  <?foreach {static-route}?>
                    <ip-route-forwarding-list>
                      <prefix>{prefix}</prefix>
                      <mask>{mask}</mask>
                      <forwarding-address>172.31.{../link-id}.2</forwarding-
address>
                    </ip-route-forwarding-list>
                  <?end?>
                </vrf>
              </route>
            </ip>
          <?end?>
          <interface xmlns="urn:ios">
```

```
            <GigabitEthernet>
              <name>{interface}</name>
              <description>By NSO: L3VPN - {string(/customer)} - {string(link-
name)}</description>
              <vrf>
                <forwarding>vpn{string(../vpn-id)}</forwarding>
              </vrf>
              <ip>
                <address>
                  <primary>
                    <address>172.31.{link-id}.1</address>
                    <mask>255.255.255.252</mask>
                  </primary>
                </address>
              </ip>
            </GigabitEthernet>
          </interface>
          <router xmlns="urn:ios">
            <?if {routing-protocol='bgp'}?>
              <bgp>
                <as-no>1</as-no>
                <address-family>
                  <with-vrf>
                    <ipv4>
                      <af>unicast</af>
                      <vrf>
                        <name>vpn{string(../vpn-id)}</name>
                        <redistribute>
                          <connected/>
                          <static/>
                        </redistribute>
                        <neighbor>
                          <id>172.31.{link-id}.2</id>
                          <remote-as annotation="first">65001</remote-as>
                          <activate/>
                          <allowas-in/>
                          <default-originate/>
                        </neighbor>
                      </vrf>
                    </ipv4>
                  </with-vrf>
                </address-family>
              </bgp>
            <?end?>
          </router>

        </config>
      </device>
    <?end?>
  </devices>
</config-template>
```

Step 9 Save the file.

Step 10 Connect to the NSO CLI and reload the packages.

```
developer@devbox:~/nso-run/packages/l3vpn/templates$ ncs_cli -Cu admin
admin@ncs# packages reload
reload-result {
    package cisco-ios-cli-6.67
    result true
}
reload-result {
    package dns-snmp-service
    result true
}
reload-result {
    package l3vpn
    result true
}
```

Troubleshooting Hints

The variables must reference data according to the hierarchy of the YANG data model.

Activity Verification

You have completed this task when you attain this result:

- You have successfully developed the l3vpn-template.xml, and the **packages reload** operation was successful.

Task 3: Deploy a Service Instance

The purpose of this task is to deploy a service instance based on your newly installed service.

Activity

Complete these steps:

Step 1 Connect to the NSO CLI and enter configuration mode.

```
developer@devbox:~/nso-run/packages$ ncs_cli -Cu admin
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)#
```

Step 2 Add a new customer **ACME** and commit the changes.

```
admin@ncs(config)# customers customer ACME
admin@ncs(config-customer-ACME)# top
admin@ncs(config)# commit
Commit complete.
```

```
admin@ncs(config)#
```

Step 3 Provision the first service instance.

```
admin@ncs(config)# services l3vpn ACME vpn-id 10001
admin@ncs(config-l3vpn-ACME)# customer ACME
admin@ncs(config-l3vpn-ACME)# link 1 link-name Site1 pe-device R1 interface 6
routing-protocol bgp
admin@ncs(config-link-1)# exit
admin@ncs(config-l3vpn-ACME)# link 2 link-name Site2 pe-device R2 interface 0/1
routing-protocol static
admin@ncs(config-link-3)# static-route 192.168.1.0 mask 255.255.255.0
admin@ncs(config-static-route-192.168.1.0)# exit
admin@ncs(config-link-3)# static-route 192.168.2.0 mask 255.255.255.0
admin@ncs(config-static-route-192.168.2.0)# top
```

Step 4 After entering all service parameters, verify the configuration by using the **show configuration** command.

Your configuration should match the following output:

```
admin@ncs(config)# show configuration
services l3vpn ACME
 vpn-id   10001
 customer ACME
 link 1
  link-name        Site1
  pe-device        R1
  interface        6
  routing-protocol bgp
 !
 link 2
  link-name        Site2
  pe-device        R2
  interface        6
  routing-protocol static
  static-route 192.168.1.0
   mask 255.255.255.0
  !
  static-route 192.168.2.0
   mask 255.255.255.0
  !
 !
!
```

Note	Make sure you are at the top configuration level before running the **show configuration** command. The **show configuration** command displays only commands on the same or lower levels in the command hierarchy. You can quickly jump to the top level by issuing the **top** command.

Step 5 Preview the device changes by using the **commit dry run** command.

Your configuration should match the following output:

```
admin@ncs(config-static-route-192.168.2.0)# commit dry-run
cli {
    local-node {
        data  devices {
                  device R1 {
                      config {
                          vrf {
            +                 definition vpn10001 {
            +                     description "By NSO: L3VPN - ACME";
            +                     rd 1:10001;
            +                     route-target {
            +                         export 1:10001;
            +                         import 1:10001;
            +                     }
            +                     address-family {
            +                         ipv4 {
            +                         }
            +                     }
            +                 }
                          }
                          interface {
                              GigabitEthernet 6 {
            +                     description "By NSO: L3VPN - ACME - Site1";
                                  vrf {
            +                         forwarding vpn10001;
                                  }
                                  ip {
                                      address {
                                          primary {
            +                                 address 172.31.1.1;
            +                                 mask 255.255.255.252;
                                          }
                                      }
                                  }
                              }
                          }
                          router {
            +                 bgp 1 {
            +                     address-family {
            +                         with-vrf {
            +                             ipv4 unicast {
            +                                 vrf vpn10001 {
            +                                     redistribute {
            +                                         connected {
            +                                         }
            +                                         static {
            +                                         }
            +                                     }
            +                                     neighbor 172.31.1.2 {
            +                                         remote-as 65001;
            +                                         activate;
            +                                         allowas-in {
            +                                         }
            +                                         default-originate {
            +                                         }
```

```
+                                       }
+                                   }
+                               }
+                           }
+                       }
+                   }
               }
               device R2 {
                   config {
                       vrf {
+                          definition vpn10001 {
+                              description "By NSO: L3VPN - ACME";
+                              rd 1:10001;
+                              route-target {
+                                  export 1:10001;
+                                  import 1:10001;
+                              }
+                              address-family {
+                                  ipv4 {
+                                  }
+                              }
+                          }
                       }
                       ip {
                           route {
+                              vrf vpn10001 {
+                                  ip-route-forwarding-list 192.168.1.0
255.255.255.0 172.31.2.2;
+                                  ip-route-forwarding-list 192.168.2.0
255.255.255.0 172.31.2.2;
+                              }
                           }
                       }
                       interface {
                           GigabitEthernet 6 {
+                              description "By NSO: L3VPN - ACME - Site2";
                               vrf {
+                                  forwarding vpn10001;
                               }
                               ip {
                                   address {
                                       primary {
+                                          address 172.31.2.1;
+                                          mask 255.255.255.252;
                                       }
                                   }
                               }
                           }
                       }
                   }
               }
           }
           services {
+              l3vpn ACME {
+                  vpn-id 10001;
+                  customer ACME;
+                  link 1 {
+                      link-name Site1;
```

```
              +            pe-device R1;
              +            interface 6;
              +            routing-protocol bgp;
              +        }
              +        link 2 {
              +            link-name Site2;
              +            pe-device R2;
              +            interface 6;
              +            routing-protocol static;
              +            static-route 192.168.1.0 {
              +                mask 255.255.255.0;
              +            }
              +            static-route 192.168.2.0 {
              +                mask 255.255.255.0;
              +            }
              +        }
              +    }
               }
    }
}
```

Step 6 Commit the service instance creation, using the **commit** command.

```
admin@ncs(config)# commit
Commit complete.
admin@ncs(config)# exit
```

Step 7 Verify that the service configuration exists and that the XPath was evaluated correctly.

```
admin@ncs# show running-config services l3vpn
services l3vpn ACME
 vpn-id   10001
 customer ACME
 link 1
  link-name        Site1
  pe-device        R1
  interface        6
  routing-protocol bgp
 !
 link 2
  link-name        Site2
  pe-device        R2
  interface        6
  routing-protocol static
  static-route 192.168.1.0
   mask 255.255.255.0
  !
  static-route 192.168.2.0
   mask 255.255.255.0
  !
 !
!admin@ncs# show running-config services l3vpn | tab
VPN   VPN             LINK  LINK   PE                ROUTING
NAME  ID     CUSTOMER  ID    NAME   DEVICE  INTERFACE  PROTOCOL  PREFIX       MASK
-------------------------------------------------------------------------------------
ACME  10001  ACME      1     Site1  R1      6          bgp
```

```
                    2     Site2  R2      6          static    192.168.1.0  255.255.255.0
                                                              192.168.2.0  255.255.255.0
admin@ncs# exit
```

As you can see from the preceding output, your commit was successful, which also means that the XPath was evaluated correctly.

Step 8 You can also preview the XPath entries for the l3vpn service instance ACME you have just created.

```
admin@ncs# show running-config services l3vpn ACME | display xpath
/services/l3vpn:l3vpn[vpn-name='ACME']/vpn-id 10001
/services/l3vpn:l3vpn[vpn-name='ACME']/customer ACME
/services/l3vpn:l3vpn[vpn-name='ACME']/link[link-id='1']/link-name Site1
/services/l3vpn:l3vpn[vpn-name='ACME']/link[link-id='1']/pe-device R1
/services/l3vpn:l3vpn[vpn-name='ACME']/link[link-id='1']/interface 6
/services/l3vpn:l3vpn[vpn-name='ACME']/link[link-id='1']/routing-protocol bgp
/services/l3vpn:l3vpn[vpn-name='ACME']/link[link-id='2']/link-name Site2
/services/l3vpn:l3vpn[vpn-name='ACME']/link[link-id='2']/pe-device R2
/services/l3vpn:l3vpn[vpn-name='ACME']/link[link-id='2']/interface 6
/services/l3vpn:l3vpn[vpn-name='ACME']/link[link-id='2']/routing-protocol
static
/services/l3vpn:l3vpn[vpn-name='ACME']/link[link-id='2']/static-
route[prefix='192.168.1.0']/mask 255.255.255.0
/services/l3vpn:l3vpn[vpn-name='ACME']/link[link-id='2']/static-
route[prefix='192.168.2.0']/mask 255.255.255.0
```

Step 9 Exit NSO CLI.

```
admin@ncs# exit
developer@devbox:~$
```

Step 10 You can use the *ncs_cmd* utility using the xe command (short for XPath evaluate) to evaluate custom XPath expressions. In this example, you can try evaluating an XPath expression that references an existing vpn-id (vpn-name ACME and link-id 1) and a non-existing vpn-id (ABME).

```
developer@devbox:~$ ncs_cmd -c "xe /services/l3vpn:l3vpn[vpn-
name='ACME']/link[link-id='1']"
XPATH_TRACE: exists(/services/l3vpn:l3vpn[vpn-name='ACME']) = true

XPATH_TRACE: exists(/services/l3vpn:l3vpn[vpn-name='ACME']/link[link-id='1']) =
true

XPATH_TRACE: get_elem(/services/l3vpn:l3vpn[vpn-name='ACME']/link[link-
id='1']/link-name) = Site1

XPATH_TRACE: get_elem(/services/l3vpn:l3vpn[vpn-name='ACME']/link[link-
id='1']/pe-device) = R1

XPATH_TRACE: get_elem(/services/l3vpn:l3vpn[vpn-name='ACME']/link[link-
id='1']/interface) = 6
```

```
XPATH_TRACE: get_elem(/services/l3vpn:l3vpn[vpn-name='ACME']/link[link-
id='1']/routing-protocol) = bgp

XPATH_TRACE: get_next(/services/l3vpn:l3vpn[vpn-name='ACME']/link[link-
id='1']/static-route) = false

1Site1R16bgp

developer@devbox:~$ ncs_cmd -c "xe /services/l3vpn:l3vpn[vpn-name='ABME']"
XPATH_TRACE: exists(/services/l3vpn:l3vpn[vpn-name='ABME']) = false

developer@devbox:~$ ncs_cmd -c "xe boolean(/services/l3vpn:l3vpn[vpn-
name='ABME'])"

false
developer@devbox:~$
```

Note	If you want to test whether a node exists or not, you can test that by using the XPath function **boolean**. An empty node-set is always false, and a non-empty is always true.

Activity Verification

You have completed this task when you attain this result:

- The service has been successfully deployed.