# Discovery 8: Create an L3VPN Python-and-Template Service

## Introduction

In this activity, you will learn how to create an L3MPLS VPN service with Python programming language support. 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
- Create the Python code that will apply variables to feature templates using conditional statements and loops
- 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 | A 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 | A unique identifier describing an instance of a list link |
| Interface | An interface on the PE router to which the customer site is connected |
| Routing protocol | Routing protocol option |
| Pe-device | A device on which the service will be deployed |
| Static-route | Static routing option |

| Attribute | Description |
|---|---|
| Prefix | Route prefix |
| 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.

**Note** The service model parameters are the same as the ones in the template-based l3vpn service from one of the previous discoveries. You may copy the contents of the service model **l3vpn-python.yang** from **/home/developer/nso/examples/l3vpn-python/src/yang**. In that case, you may continue with Step 16 (to compile the service package).

### *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 *Python-and-template*-based service skeleton by using the **ncs-make-package** command. Use **l3vpn-python** as the name of the new service.

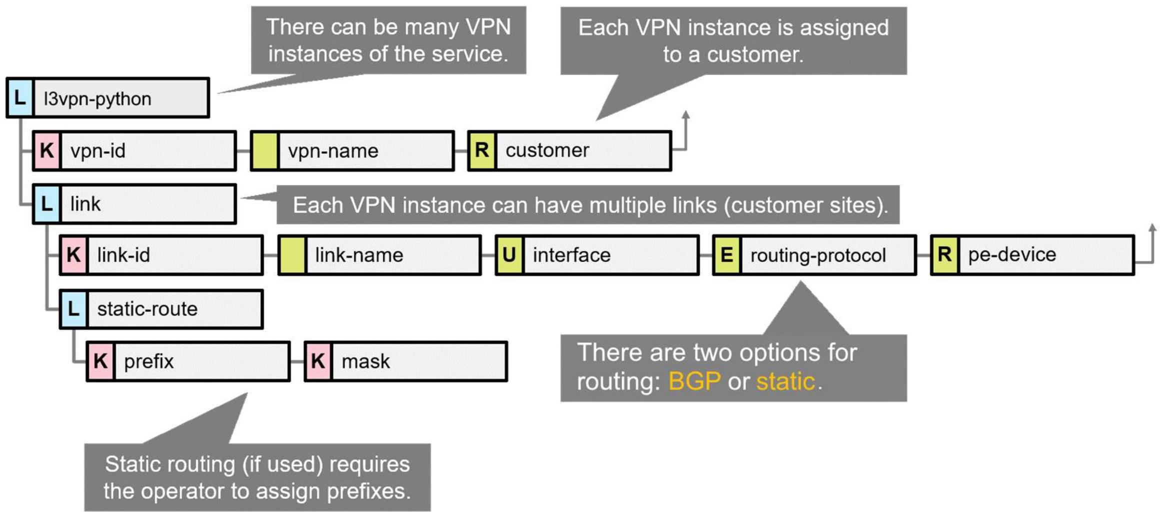
**Note** You can use **ncs-make-package --help** or **man ncs-make-package** to learn about all the available options.

```
developer@devbox:~/nso-run/packages$ ls
cisco-ios-cli-6.21 cisco-ios-cli-6.67  dns-snmp-service  l3vpn
developer@devbox:~/nso-run/packages$ ncs-make-package --service-skeleton
python-and-template l3vpn-python
developer@devbox:~/nso-run/packages$ ls
cisco-ios-cli-6.67  dns-snmp-service  l3vpn  l3vpn-python
```

**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-python* | list | — | — |
| VPN ID | vpn-id | uint32 | Range: 10001 – 19999 | key for list *l3vpn* |
| VPN name | vpn-name | string | — | — |
| Customer | customer | leafref | — | reference to NSO customers |
| Link | link | list | — | — |
| Link ID | link-id | uint32 | Range: 1 – 255 | key for list *link* |
| Interface | interface | string | — | *pe-device* and *interface* combination must be unique |
| Routing protocol | routing-protocol | enumeration | Options: *bgp* *static* | — |
| PE device | pe-device | leafref | Only PE devices may be selected | *pe-device* and *interface* combination must be unique |
| Static route | static-route | list | Applicable only when static routing is selected | — |
| Prefix | prefix | inet:ipv4-address | — | key for list *static-route* |

| Attribute | Name | Data Type | Restriction | Other |
|---|---|---|---|---|
| Mask | mask | union of<br>inet:ipv4-address and inet:ipv6address | — | — |


There can be many VPN instances of the service.
Each VPN instance is assigned to a customer.
L l3vpn-python
K vpn-id
vpn-name
R customer
L link
Each VPN instance can have multiple links (customer sites).
K link-id
link-name
U interface
E routing-protocol
R pe-device
L static-route
K prefix
K mask
There are two options for routing: BGP or static.
Static routing (if used) requires the operator to assign prefixes.

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

```
developer@devbox:~/nso-run/packages$ cd l3vpn-python/src/yang
developer@devbox:~/nso-run/packages/l3vpn-python/src/yang$ code l3vpn-
python.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://www.cisco.com/example/l3vpn-python* and remove the leaf-list *device*, the leaf *dummy*, and the *description*. They will not be used for the l3vpn-python service and will be replaced by other statements. Add an *augment /ncs:services* statement, to include the list *l3vpn-python* in the services branch of the CDB. Also, add header data and a revision statement.

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

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

  organization "CISCO SYSTEMS";
  contact "CISCO SYSTEMS; support@cisco.com";
  description "The module to showcase NSO capabilities.";
  revision 2022-10-01 {
    description "version 1.0.0.0, see README file.";
  }

  augment /ncs:services {
    list l3vpn-python {
      description "This is an RFS skeleton service";

      key name;
      leaf name {
        tailf:info "Unique service id";
        tailf:cli-allow-range;
        type string;
      }

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

    }
  }

}
```

**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-python service. The list records are uniquely identified, using the key attribute (vpn-name). Add node descriptions for syntax help.

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

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

  organization "CISCO SYSTEMS";
  contact "CISCO SYSTEMS; support@cisco.com";
  description "The module to showcase NSO capabilities.";
  revision 2022-10-01 {
    description
      "Initial revision of L3vpn Python service.";
  }

  augment /ncs:services {
    list l3vpn-python {
      tailf:info "L3VPN Service - Python based";

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

      uses ncs:service-data;
      ncs:servicepoint l3vpn-python-servicepoint;    }
  }
}
```

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

```
module l3vpn-python {

  ...

  augment /ncs:services {
    list l3vpn-python {
      tailf:info "L3VPN Service - Python based";

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

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

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

    }
  }
}
```

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

```
module l3vpn-python {

  ...

  augment /ncs:services {
    list l3vpn-python {

      ...

      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-python {

  ...

  augment /ncs:services {
    list l3vpn-python {

      ...

      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 a *unique* restriction, which will prevent duplicate entries. An 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 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. 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, demand 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-python {
  namespace "http://www.cisco.com/example/l3vpn-python";
  prefix l3vpn-python;

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

  organization "CISCO SYSTEMS";
  contact "CISCO SYSTEMS; support@cisco.com";
  description "The module to showcase NSO capabilities.";
  revision 2022-10-01 {
    description
      "Initial revision of L3vpn Python service.";
  }

  augment /ncs:services {
    list l3vpn-python {
      tailf:info "L3VPN Service - Python based";

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

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

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

**Step 17** After the package is compiled, log in to the NSO CLI and issue the **packages reload** command.

```
developer@devbox:~/nso-run/packages/l3vpn-python/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
}
reload-result {
    package l3vpn-python
    result true
}
admin@ncs#
System message at 2022-10-01 17:03:27...
```

```
    Subsystem stopped: ncs-dp-3-cisco-ios-cli-6.67:IOSDp
admin@ncs#
System message at 2022-10-01 17:03:27...
    Subsystem started: ncs-dp-5-cisco-ios-cli-6.67:IOSDp
admin@ncs#
```

**Step 18** Check the availability of CLI commands to manage the service. For example, use the **services l3vpn-python** 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
  l3vpn-python        L3VPN Service - Python based
admin@ncs# services l3vpn-python ?
% No entries found
```

**Step 19** 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-python/
total 8
drwxr-xr-x 2 developer docker  30 Oct 25 12:05 load-dir
-rw-r--r-- 1 developer docker 392 Oct 25 11:44 package-meta-data.xml
drwxr-xr-x 3 developer docker  26 Oct 25 11:44 python
-rw-r--r-- 1 developer docker 751 Oct 25 11:44 README
drwxr-xr-x 4 developer docker  46 Oct 25 12:05 src
drwxr-xr-x 2 developer docker  39 Oct 25 11:44 templates
drwxr-xr-x 3 developer docker  38 Sep 14  2020 test
developer@devbox:~/nso-run/packages$ ls -l l3vpn-python/src/yang
total 4
-rw-r--r-- 1 developer docker 2652 Oct 25 12:05 l3vpn-python.yang
developer@devbox:~/nso-run/packages$ ls -l l3vpn-python/templates
total 4
-rw-rw-r-- 1 student student 733 Sep 14 15:05 l3vpn-python-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 and also add Python code.

# Task 2: Create Feature Templates

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

## *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)# devices sync-from
sync-result {
    device R1
    result true
}
sync-result {
    device R2
    result true
}
admin@ncs(config)# no services l3vpn
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 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-python/templates).

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

**Step 6** Rename the file *l3vpn-python-template.xml* to *l3vpn-common.xml.*

```
developer@devbox:~/nso-run/packages/l3vpn-python/templates$ mv l3vpn-python-
template.xml l3vpn-common.xml
```

**Step 7** Open the template by using the **code l3vpn-common.xml** command.

Remove the provided comments under the <device> and <config> tags and add your own (IOS).

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

        <!-- IOS -->


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

**Step 8** Save the file *l3vpn-common.xml* and switch back to the terminal window. Create two copies of the *l3vpn-common.xml* template to create additional feature templates: l3vpn-bgp.xml for BGP configuration, and l3vpn-static.xml for static routing configuration. You will drive all three templates from your Python code.

```
developer@devbox:~/nso-run/packages/l3vpn-python/templates$ cp l3vpn-common.xml
l3vpn-bgp.xml
developer@devbox:~/nso-run/packages/l3vpn-python/templates$ cp l3vpn-common.xml
l3vpn-static.xml
developer@devbox:~/nso-run/packages/l3vpn-python/templates$ ls
l3vpn-bgp.xml  l3vpn-common.xml  l3vpn-static.xml
```

**Step 9** Switch back to the **l3vpn-common.xml** template. Insert ONLY the routing-protocol independent part of the XML configuration you previously created with **commit dry-run outformat xml** into the XML skeleton section (vrf, interface).

```
<config-template xmlns="http://tail-f.com/ns/config/1.0">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <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>
        <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>

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

**Step 10** Replace all the static parameters with variables we will later apply in Python code. Compared to the template-based service, any conditional statements and loops will also be implemented in Python.

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

        <!-- IOS-->
        <vrf xmlns="urn:ios">
          <definition>
            <name>{$VRF}</name>
            <description>{$VRF_DESCRIPTION}</description>
            <rd>{$RD}</rd>
            <route-target>
              <export>
                <asn-ip>{$ASN_IP}</asn-ip>
              </export>
              <import>
                <asn-ip>{$ASN_IP}</asn-ip>
              </import>
            </route-target>
            <address-family>
              <ipv4/>
            </address-family>
          </definition>
        </vrf>
        <interface xmlns="urn:ios">
          <GigabitEthernet>
            <name>{$INTERFACE}</name>
            <description>{$INTERFACE_DESC}</description>
            <vrf>
              <forwarding>{$VRF}</forwarding>
            </vrf>
            <ip>
              <address>
                <primary>
                  <address>{$INTERFACE_IP}</address>
                  <mask>255.255.255.252</mask>
                </primary>
              </address>
            </ip>
          </GigabitEthernet>
        </interface>
      </config>
    </device>
  </devices>
</config-template>
```

**Step 11** Save and close the file. Switch back to the terminal window, and open the *l3vpn-bgp.xml* template.

```
developer@devbox:~/nso-run/packages/l3vpn-python/templates$ code l3vpn-bgp.xml
```

**Step 12** Insert only the BGP part of the XML configuration created with **commit dry-run outformat xml** into the XML skeleton section. You will apply this configuration from Python only when BGP will be selected as a routing protocol.

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

        <!-- IOS -->
        <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>
</config-template>
```

**Step 13** Replace all the static parameters with variables you will later apply in Python code. Compared to the template-based service, any conditional statements and loops will also be implemented in Python.

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

        <!-- IOS-->
        <router xmlns="urn:ios">
          <bgp>
            <as-no>1</as-no>
            <address-family>
              <with-vrf>
                <ipv4>
                  <af>unicast</af>
                  <vrf>
                    <name>{$VRF}</name>
                    <redistribute>
                      <connected/>
                      <static/>
                    </redistribute>
                    <neighbor>
                      <id>{$BGP_NEIGHBOR_IP}</id>
                      <remote-as>65001</remote-as>
                      <activate/>
                      <allowas-in/>
                      <default-originate/>
                    </neighbor>
                  </vrf>
                </ipv4>
              </with-vrf>
            </address-family>
          </bgp>
        </router>

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

**Step 14** Save and close the file. Switch back to the terminal window, and open the *l3vpn-static.xml* template.

```
developer@devbox:~/nso-run/packages/l3vpn-python/templates$ code l3vpn-
static.xml
```

**Step 15** Insert only the static routing part of the XML configuration created with **commit dry-run outformat xml** into the XML skeleton section. You will apply this configuration from Python only when static routing will be selected as a routing protocol.

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

```
        <!-- IOS -->
        <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>

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

**Step 16** Replace all the static parameters with variables you will later apply in Python code. Compared to the template-based service, any conditional statements and loops will also be implemented in Python.

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

        <!-- IOS-->
        <ip xmlns="urn:ios">
          <route>
            <vrf>
              <name>{$VRF}</name>
              <ip-route-forwarding-list>
                <prefix>{$STATIC_ROUTE_IP}</prefix>
                <mask>{$STATIC_ROUTE_MASK}</mask>
                <forwarding-address>{$STATIC_ROUTE_FWD_IP}</forwarding-address>
              </ip-route-forwarding-list>
            </vrf>
          </route>
        </ip>

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

**Step 17** Save and close the file. Switch back to the terminal window and connect to the NSO CLI.

```
developer@devbox:~/nso-run/packages/l3vpn-python/templates$ ncs_cli -Cu admin
```

**Step 18** Reload the packages and make sure the *l3vpn-python* service package reloads successfully.

```
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
}
reload-result {
    package l3vpn-python
    result true
}
```

## Activity Verification

You have completed this task when you attain this result:

- You have successfully developed the templates, and the **packages reload** operation was successful.

## Task 3: Write Python Mapping Code

In this task, you will write Python code for service attribute mapping.

### *Activity*

Complete these steps:

**Step 1** Open the Terminal application.

**Step 2** Go to the *python/l3vpn_python* subfolder inside the l3vpn-python package.

```
student@student-vm:~$ cd $HOME/nso-run/packages/l3vpn-
python/python/l3vpn_python
student@student-vm:~/nso-run/packages/l3vpn-python/python/l3vpn_python$ ls
__init__.py  main.py
```

**Step 3** Open the Python code for editing by the **code main.py** command.

```
student@student-vm:~/nso-run/packages/l3vpn-python/python/l3vpn_python$ code
main.py
```

This is how the main.py Python code should look when you open it for the first time:

```
# -*- mode: python; python-indent: 4 -*-
import ncs
from ncs.application import Service


# ------------------------
# SERVICE CALLBACK EXAMPLE
# ------------------------
class ServiceCallbacks(Service):

    # The create() callback is invoked inside NCS FASTMAP and
    # must always exist.
    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info('Service create(service=', service._path, ')')

        vars = ncs.template.Variables()
        vars.add('DUMMY', '127.0.0.1')
        template = ncs.template.Template(service)
        template.apply('l3vpn-python-template', vars)

    # The pre_modification() and post_modification() callbacks are optional,
    # and are invoked outside FASTMAP. pre_modification() is invoked before
    # create, update, or delete of the service, as indicated by the enum
    # ncs_service_operation op parameter. Conversely
    # post_modification() is invoked after create, update, or delete
    # of the service. These functions can be useful e.g. for
    # allocations that should be stored and existing also when the
    # service instance is removed.

    # @Service.pre_lock_create
    # def cb_pre_lock_create(self, tctx, root, service, proplist):
    #     self.log.info('Service plcreate(service=', service._path, ')')

    # @Service.pre_modification
    # def cb_pre_modification(self, tctx, op, kp, root, proplist):
    #     self.log.info('Service premod(service=', kp, ')')

    # @Service.post_modification
    # def cb_post_modification(self, tctx, op, kp, root, proplist):
    #     self.log.info('Service premod(service=', kp, ')')


# ---------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ---------------------------------------------
class Main(ncs.application.Application):
    def setup(self):
        # The application class sets up logging for us. It is accessible
        # through 'self.log' and is a ncs.log.Log instance.
        self.log.info('Main RUNNING')

        # Service callbacks require a registration for a 'service point',
        # as specified in the corresponding data model.
        #
        self.register_service('l3vpn-python-servicepoint', ServiceCallbacks)

        # If we registered any callback(s) above, the Application class
        # took care of creating a daemon (related to the service/action point).

        # When this setup method is finished, all registrations are
        # considered done and the application is 'started'.

    def teardown(self):
        # When the application is finished (which would happen if NCS went
        # down, packages were reloaded or some error occurred) this teardown
        # method will be called.

        self.log.info('Main FINISHED')
```

**Note** Indentation is important in Python. Indentation refers to the spaces at the beginning of a code line. Python uses indentation to indicate a block of code. The number of spaces is up to you as a programmer, but it has to be at least one. You have to use the same number of spaces in the same block of code, otherwise, Python will give you an error. The number of spaces is usually indicated at the beginning of the Python file in a form of a comment: **# -*- mode: python; python-indent: 4 -*-** .

**Step 4** Remove the dummy code. Create a log entry and calculate values common to all three feature templates.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
from ncs.application import Service

# ------------------------
# SERVICE CALLBACK EXAMPLE
# ------------------------
class ServiceCallbacks(Service):

    # The create() callback is invoked inside NCS FASTMAP and
    # must always exist.
    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info(f'Configuring: {service.vpn_name} - {service.vpn_id}')

        # define common variables
        vrf = f'vpn{service.vpn_id}'
        vrf_description = f'By NSO: L3VPN - {service.customer}'
        rd = f'1:{service.vpn_id}'
        asn_ip = f'1:{service.vpn_id}'

    # The pre_modification() and post_modification() callbacks are optional,
    # and are invoked outside FASTMAP. pre_modification() is invoked before
    ...
```

**Note** You can enable whitespace rendering in Visual Studio Code to have better control over the indentation. Spaces will appear as dots. This feature can be enabled in the **View, Render Whitespaces** menu.

**Step 5** Create a Python "*for*" loop that will loop over links. Calculate values specific to the *l3vpn-common.xml* template and assign values for the variables, defined in the template. Finally, apply the configuration by using the *l3vpn-common.xml* template.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
from ncs.application import Service


# ------------------------
# SERVICE CALLBACK EXAMPLE
# ------------------------
class ServiceCallbacks(Service):
    # The create() callback is invoked inside NCS FASTMAP and
    # must always exist.
    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info(f'Configuring: {service.vpn_name} - {service.vpn_id}')

        # define common variables
        vrf = f'vpn{service.vpn_id}'
        vrf_description = f'By NSO: L3VPN - {service.customer}'
        rd = f'1:{service.vpn_id}'
        asn_ip = f'1:{service.vpn_id}'

        for lnk in service.link:
            # calculate variables
            interface_ip = f'172.31.{lnk.link_id}.1'
            interface_desc = f'By NSO: L3VPN - {service.customer} - {lnk.link_name}'

            # apply to variables object
            vars = ncs.template.Variables()
            vars.add('PE-DEVICE', lnk.pe_device)
            vars.add('VRF', vrf)
            vars.add('VRF_DESCRIPTION', vrf_description)
            vars.add('RD', rd)
            vars.add('ASN_IP', asn_ip)
            vars.add('INTERFACE', lnk.interface)
            vars.add('INTERFACE_IP', interface_ip)
            vars.add('INTERFACE_DESC', interface_desc)

            # apply to common template
            template = ncs.template.Template(service)
            template.apply('l3vpn-common', vars)

    # The pre_modification() and post_modification() callbacks are optional,
    # and are invoked outside FASTMAP. pre_modification() is invoked before
    ...
```

**Step 6** In the code, check if the BGP routing protocol is set. Only if it is set, you may apply the BGP configuration. You must calculate values specific to the *l3vpn-bgp.xml* template and assign values for variables, defined in the template. Finally, apply the configuration by using the *l3vpn-bgp.xml* template.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
from ncs.application import Service


# ------------------------
# SERVICE CALLBACK EXAMPLE
# ------------------------
class ServiceCallbacks(Service):

    # The create() callback is invoked inside NCS FASTMAP and
    # must always exist.
    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info(f'Configuring: {service.vpn_name} - {service.vpn_id}')

        # define common variables
        vrf = f'vpn{service.vpn_id}'
        vrf_description = f'By NSO: L3VPN - {service.customer}'
        rd = f'1:{service.vpn_id}'
        asn_ip = f'1:{service.vpn_id}'

        for lnk in service.link:
            # calculate variables
            interface_ip = f'172.31.{lnk.link_id}.1'
            interface_desc = f'By NSO: L3VPN - {service.customer} - {lnk.link_name}'

            # apply to variables object
            vars = ncs.template.Variables()
            vars.add('PE-DEVICE', lnk.pe_device)
            vars.add('VRF', vrf)
            vars.add('VRF_DESCRIPTION', vrf_description)
            vars.add('RD', rd)
            vars.add('ASN_IP', asn_ip)
            vars.add('INTERFACE', lnk.interface)
            vars.add('INTERFACE_IP', interface_ip)
            vars.add('INTERFACE_DESC', interface_desc)

            # apply to common template
            template = ncs.template.Template(service)
            template.apply('l3vpn-common', vars)

            if lnk.routing_protocol == 'bgp':
                # calculate variables
                forwarding_address = f'172.31.{lnk.link_id}.2'

                # apply to variables object
                vars_bgp = ncs.template.Variables()
                vars_bgp.add('PE-DEVICE', lnk.pe_device)
                vars_bgp.add('VRF', vrf)
                vars_bgp.add('BGP_NEIGHBOR_IP', forwarding_address)

                # apply to template for bgp config
                template = ncs.template.Template(service)
                template.apply('l3vpn-bgp', vars_bgp)

    # The pre_modification() and post_modification() callbacks are optional,
    # and are invoked outside FASTMAP. pre_modification() is invoked before
    ...
```

**Note** This snippet of code replaces the 'bgp' routing protocol condition you have defined in your l3vpn template-based service.

**Step 7** In the code, check if the static routing protocol is set. Only if it is set, you may apply the static routing configuration. You need to loop through all the static route entries, calculate values specific to the *l3vpn-static.xml* template and assign values for variables, defined in the template. Finally, apply the configuration by using the *l3vpn-static.xml* template.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
from ncs.application import Service


# ------------------------
# SERVICE CALLBACK EXAMPLE
# ------------------------
class ServiceCallbacks(Service):

    # The create() callback is invoked inside NCS FASTMAP and
    # must always exist.
    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info(f'Configuring: {service.vpn_name} - {service.vpn_id}')

        ...

        for lnk in service.link:

            ...

            # apply to common template
            template = ncs.template.Template(service)
            template.apply('l3vpn-common', vars)


            if lnk.routing_protocol == 'bgp':
                # calculate variables
                forwarding_address = f'172.31.{lnk.link_id}.2'

                # apply to variables object
                vars_bgp = ncs.template.Variables()
                vars_bgp.add('PE-DEVICE', lnk.pe_device)
                vars_bgp.add('VRF', vrf)
                vars_bgp.add('BGP_NEIGHBOR_IP', forwarding_address)

                # apply to template for bgp config
                template = ncs.template.Template(service)
                template.apply('l3vpn-bgp', vars_bgp)


            if lnk.routing_protocol == 'static':
                for route in lnk.static_route:
                    # calculate variables
                    forwarding_address = f'172.31.{lnk.link_id}.2'

                    # apply to variables object
                    vars_static = ncs.template.Variables()
                    vars_static.add('PE-DEVICE', lnk.pe_device)
                    vars_static.add('VRF', vrf)
                    vars_static.add('STATIC_ROUTE_IP', route.prefix)
                    vars_static.add('STATIC_ROUTE_MASK', route.mask)
                    vars_static.add('STATIC_ROUTE_FWD_IP', forwarding_address)

                    # apply to template for static routing
                    template = ncs.template.Template(service)
                    template.apply('l3vpn-static', vars_static)

    # The pre_modification() and post_modification() callbacks are optional,
    # and are invoked outside FASTMAP. pre_modification() is invoked before
    ...
```

**Note** This snippet of code replaces the 'static' routing protocol condition you have defined in your l3vpn template-based service.

**Step 8** Finally, add a log entry notifying the log facility the processing is complete.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
from ncs.application import Service


# ------------------------
# SERVICE CALLBACK EXAMPLE
# ------------------------
```

```python
class ServiceCallbacks(Service):

    # The create() callback is invoked inside NCS FASTMAP and
    # must always exist.
    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info(f'Configuring: {service.vpn_name} - {service.vpn_id}')

        # define common variables
        vrf = f'vpn{service.vpn_id}'
        vrf_description = f'By NSO: L3VPN - {service.customer}'
        rd = f'1:{service.vpn_id}'
        asn_ip = f'1:{service.vpn_id}'

        for lnk in service.link:
            # calculate variables
            interface_ip = f'172.31.{lnk.link_id}.1'
            interface_desc = f'By NSO: L3VPN - {service.customer} - {lnk.link_name}'

            # apply to variables object
            vars = ncs.template.Variables()
            vars.add('PE-DEVICE', lnk.pe_device)
            vars.add('VRF', vrf)
            vars.add('VRF_DESCRIPTION', vrf_description)
            vars.add('RD', rd)
            vars.add('ASN_IP', asn_ip)
            vars.add('INTERFACE', lnk.interface)
            vars.add('INTERFACE_IP', interface_ip)
            vars.add('INTERFACE_DESC', interface_desc)

            # apply to common template
            template = ncs.template.Template(service)
            template.apply('l3vpn-common', vars)

            if lnk.routing_protocol == 'bgp':
                # calculate variables
                forwarding_address = f'172.31.{lnk.link_id}.2'

                # apply to variables object
                vars_bgp = ncs.template.Variables()
                vars_bgp.add('PE-DEVICE', lnk.pe_device)
                vars_bgp.add('VRF', vrf)
                vars_bgp.add('BGP_NEIGHBOR_IP', forwarding_address)

                # apply to template for bgp config
                template = ncs.template.Template(service)
                template.apply('l3vpn-bgp', vars_bgp)

            if lnk.routing_protocol == 'static':
                for route in lnk.static_route:
                    # calculate variables
                    forwarding_address = f'172.31.{lnk.link_id}.2'

                    # apply to variables object
                    vars_static = ncs.template.Variables()
                    vars_static.add('PE-DEVICE', lnk.pe_device)
                    vars_static.add('VRF', vrf)
                    vars_static.add('STATIC_ROUTE_IP', route.prefix)
                    vars_static.add('STATIC_ROUTE_MASK', route.mask)
                    vars_static.add('STATIC_ROUTE_FWD_IP', forwarding_address)

                    # apply to template for static routing
                    template = ncs.template.Template(service)
                    template.apply('l3vpn-static', vars_static)

        self.log.info(f'Configuring DONE: {service.vpn_name} - {service.vpn_id}')
    # The pre_modification() and post_modification() callbacks are optional,
    # and are invoked outside FASTMAP. pre_modification() is invoked before
    ...
```

**Note** Log entry is the last line of code in your **cb_create** method. Indentation is important; in the case of troubleshooting, it has 8 spaces from the beginning – the same as the *for* loop statement.

**Step 9** Save the file and switch back to the terminal window.

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

```
student@student-vm:~/nso-run/packages/l3vpn-python/python/l3vpn_python$ 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
}
reload-result {
    package l3vpn-python
    result true
}
admin@ncs#
```

## Activity Verification

You have completed this task when you attain these results:

- You have successfully developed the main.py, and the packages reload operation was successful.

# Task 4: 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 the configuration mode.

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

**Step 2** Enter the configuration mode and add a new customer **ACME.** 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-python ACME vpn-id 10001
admin@ncs(config-l3vpn-python-ACME)# customer ACME
admin@ncs(config-l3vpn-python-ACME)# link 1 link-name Site1 pe-device R1
interface 6 routing-protocol bgp
admin@ncs(config-link-1)# exit
admin@ncs(config-l3vpn-python-ACME)# link 2 link-name Site2 pe-device R2
interface 6 routing-protocol static
admin@ncs(config-link-2)# 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-2)# 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-python 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(config)#
```

**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)# 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 {
                                      no-address {
             -                            address false;
                                      }
                                      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 {
                        no-address {
-                           address false;
                        }
```

```
                                      address {
                                          primary {
             +                                address 172.31.2.1;
             +                                mask 255.255.255.252;
                                          }
                                      }
                                  }
                              }
                          }
                      }
                  }
              }
              services {
             +    l3vpn-python 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 using the **commit** command.

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

**Step 7** Verify that the service configuration exists.

```
admin@ncs# show running-config services l3vpn-python
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-python | 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#
```

## Activity Verification

You have completed this task when you attain this result:

- The service has been successfully deployed.