Create an L3VPN Service Using Dynamic ID Allocation

Introduction

In this activity, you will modify the l3mplsvpn nanoservice. You will change it to use an id-allocator for dynamic service parameter requesting and allocation instead of relying on service configuration. This way, your NSO services can integrate with services and code that are located outside your package.

This ID allocation simulates an external allocation service, which is often a part of more complex NSO services.

After completing this activity, you will be able to:

- Use an id-allocator package for dynamic ID requests and allocation.
- Modify an existing service to use a dynamic ID allocator.
- Deploy a service and observe how IDs are allocated dynamically.

Job Aids

The following job aid is available to help you complete the lab activities:

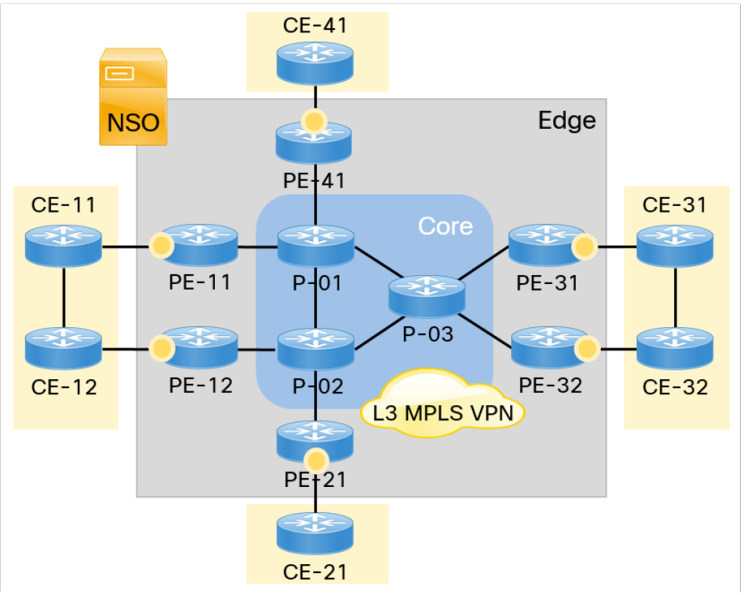
- This Lab Guide

The following table contains passwords that you might need.

Device	Username	Password
Student-VM	student	1234QWer
NSO application	admin	admin

Job Aids for Task 1

The figure provides a visual aid for this activity.



Required Resources

The following resources and equipment are required for completing the activities in this lab guide:

- PC or laptop with a web browser
- Access to the Internet

Command Syntax Reference

This lab guide uses the following conventions for command syntax:

Formatting	Description and Examples
show running config	Commands in steps use this formatting.
Example	Type **show running config**
Example	Use the **name** command.
`show running config`	Commands in CLI outputs and configurations use this formatting.
highlight	CLI output that is important is highlighted.
Example	`student@student-vm:~$ ncs --version` `5.3.2`
Example	Save your current configuration as the default **startup config**. `Router Name# copy running startup`
brackets ([ ])	Indicates optional element. You can choose one of the options.
Example:	`(config-if)# frame-relay lmi-type {ansi|cisco|q933a}`
italics font	Arguments for which you supply values.
Example	Open file **ip tcp window-size** *bytes*
angle brackets (<>)	In contexts that do not allow italics, arguments for which you supply values are enclosed in angle brackets [<>]. Do not type the brackets when entering the command.
Example	If the command syntax is **ping** <*ip_address*>, you enter ping *192.32.10.12*
string	A non-quoted set of characters. Type the characters as-is.
Example	(config)# **hostname MyRouter**
vertical line (|)	Indicates that you enter one of the choices. The vertical line separates choices. Do not type the vertical line when entering the command.
Example	If the command syntax is **show ip route|arp**, you enter either **show ip route** or **show ip arp**, but not both.

Command List

The following are the most common commands that you will need.

Linux Shell:

Command	Comment
source /opt/ncs/ncs-5.3.2/ncsrc	Source NSO environmental variable in Docker container.
ls|ll	Display contents of the current directory.
cd	Move directly to user home directory.
cd ..	Exit out of current directory.
cd test	Move into folder "test" which is a subfolder of the current directory.
cd /home/student/nso300	Move into folder "nso300" by specifying direct path to it starting from the root of directory system.
ncs_cli -C -u admin	Log in to NSO CLI directly from local server.

NSO CLI:

Command	Comment
switch cli	Change CLI style.
show ?	Display all command options for current mode.
configure	Enter configuration mode.
commit	Commit new configuration (configuration mode only command).
show configuration	Display new configuration that has not yet been committed (configuration mode only command).

Makefile commands for Docker environment:

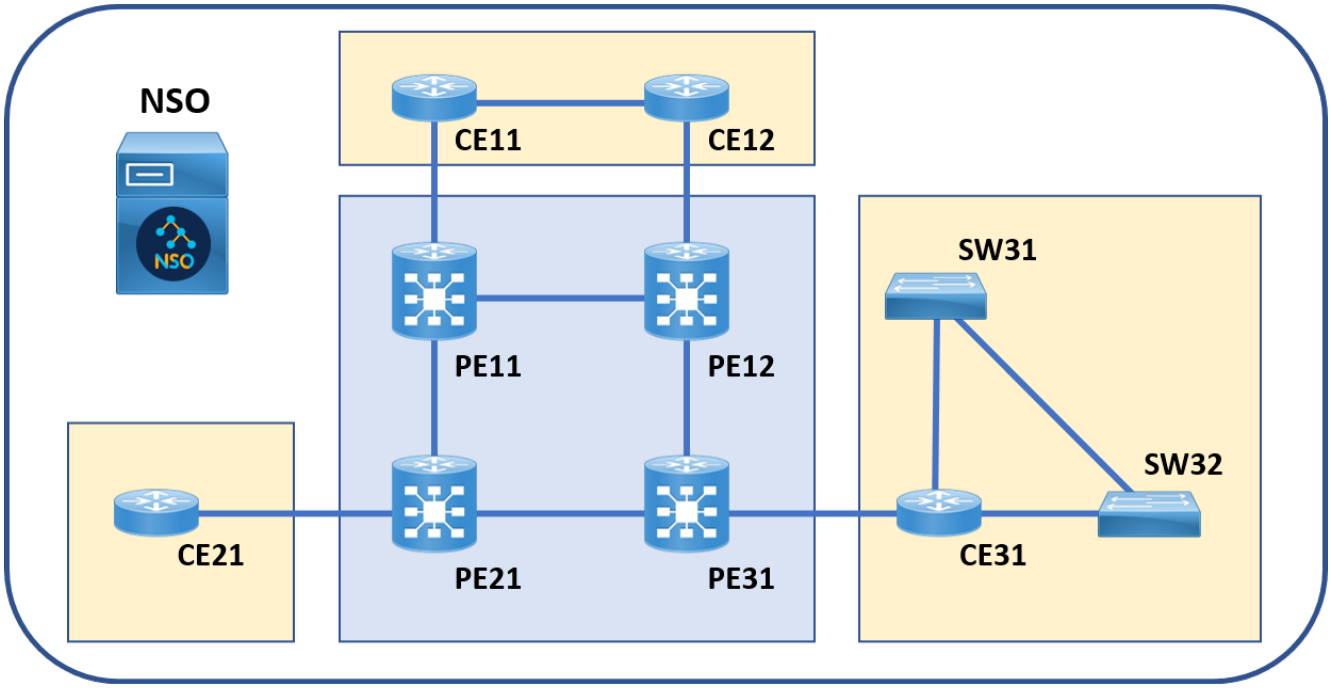
Command	Comment
make build	Builds the main NSO Docker image.
make testenv-start	Starts the NSO Docker environment.
make testenv-stop	Stops the NSO Docker environment.
make testenv-build	Recompiles and reloads the NSO packages.
make testenv-cli	Enters the NSO CLI of the NSO Docker container.
make testenv-shell	Enters the Linux shell of the NSO Docker container.
make dev-shell	Enters the Linux shell of the NSO Docker development container.

Lab Topology Information

Your lab session is your own personal sandbox. Whatever you do in your session will not be reflected in anyone else's session. There are two topologies. The general topology is your lab environment with a Linux server (Student-VM). On the Linux server within that topology is your second topology—a Docker environment, which consists of an NSO Docker image with your NSO installation, together with numerous Docker images of NetSim routers and switches that are logically grouped into a network topology. This will be the network that you will orchestrate with your NSO.

- Network topology is designed in a way to cover both Service Provider and Enterprise use cases. It is a simulated NetSim network—devices have no Control or Data Plane. Devices will, however, accept or reject configurations sent by the NSO, just as real devices would.

Topology



Task 1: Use the Resource-Manager Package for ID Allocation

In this task, you will modify the l3mplsvpn nano service to use the resource-manager package for ID allocation inside the service. This package simulates an external ID allocation service that is commonly used to allocate parameters (such as IP addresses, customer data, dynamic network data).

The final solutions for all labs, including this lab, are located in the ~/solutions directory. You can use them for copy-pasting longer pieces of code and as a reference point for troubleshooting your packages.

Activity

Complete these steps:

Step 1

Connect to the Student-VM server by clicking the icon labelled NSO in the topology.

Step 2

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

```
student@student-vm:$
```

Step 3

Go to the *nso300* folder.

```
student@student-vm:~$ cd nso300
student@student-vm:~/nso300$
```

Step 4

Enter the development NSO Docker container shell with the **make dev-shell** command and enter the **/src/packages** directory.

```
student@student-vm:~/nso300$ make dev-shell
Docker run -it -v $(pwd):/src nso300.gitlab.local/cisco-nso-dev:5.3.2
root@2fa6a6e21168:/# cd src/packages/
root@2fa6a6e21168:/src/packages#
```

Step 5

Create a new l3mplsvpn package using the **ncs-make-package** command.

```
root@2fa6a6e21168:/src/packages# ncs-make-package --service-skeleton python-and-template --component-class l3mplsvpn.L3MplsVpn
l3mplsvpn
```

Step 6

Change the ownership of the package.

```
root@2fa6a6e21168:/src/packages# chown -Rv 1000:1000 l3mplsvpn
changed ownership of 'l3mplsvpn/test/internal/Makefile' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/test/internal/lux/Makefile' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/test/internal/lux/service/Makefile' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/test/internal/lux/service/dummy-service.xml' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/test/internal/lux/service/run.lux' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/test/internal/lux/service/dummy-device.xml' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/test/internal/lux/service/pyvm.xml' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/test/internal/lux/service' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/test/internal/lux' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/test/internal' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/test/Makefile' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/test' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/python/l3mplsvpn/__init__.py' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/python/l3mplsvpn/l3mplsvpn.py' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/python/l3mplsvpn' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/python' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/README' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/templates/l3mplsvpn-template.xml' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/templates' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/src/Makefile' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/src/yang/l3mplsvpn.yang' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/src/yang' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/src' from root:root to 1000:1000
changed ownership of 'l3mplsvpn/package-meta-data.xml' from root:root to 1000:1000
changed ownership of 'l3mplsvpn' from root:root to 1000:1000
root@2fa6a6e21168:/src/packages#
```

Step 7

Open and study the **package-meta-data.xml** file of your new service package.

```
root@2fa6a6e21168:/src/packages# cd l3mplsvpn
root@2fa6a6e21168:/src/packages/l3mplsvpn# vi package-meta-data.xml
```

Step 8

This is how the package-meta-data.xml file should initially appear:

```
<ncs-package xmlns="http://tail-f.com/ns/ncs-packages">
  <name>l3mplsvpn</name>
  <package-version>1.0</package-version>
  <description>Generated Python package</description>
  <ncs-min-version>5.3</ncs-min-version>

  <component>
    <name>l3mplsvpn</name>
    <application>
      <python-class-name>l3mplsvpn.l3mplsvpn.L3MplsVpn</python-class-name>
    </application>
  </component>
</ncs-package>
```

Step 9

Change the package description and component name to better represent the service purpose.

```
<ncs-package xmlns="http://tail-f.com/ns/ncs-packages">
  <name>l3mplsvpn</name>
  <package-version>1.0</package-version>
  <description>L3 MPLS VPN Python and Template Service</description>
  <ncs-min-version>5.3</ncs-min-version>

  <component>
    <name>L3 MPLS VPN</name>
    <application>
      <python-class-name>l3mplsvpn.l3mplsvpn.L3MplsVpn</python-class-name>
    </application>
  </component>
</ncs-package>
```

Step 10

Save the file and exit the file editor.

Step 11

Exit the development container.

```
root@2fa6a6e21168:/src/packages/l3mplsvpn# exit
logout
student@student-vm:~/nso300$
```

Step 12

List the contents of the *packages* folder. The resource-manager package is not in the packages folder but is instead included as your Docker image dependency. Locate the resource-manager as a dependency in the **includes** folder.

```
student@student-vm:~/nso300$ ls packages
l3mplsvpn
student@student-vm:~/nso300$ ls includes
ned-asa  ned-ios  ned-iosxr  ned-nx  resource-manager
```

Step 13

Add **resource-manager** as a required package in your **l3mplsvpn** package-meta-data.xml file and change the **package-version** to **1.1** and save the file. This prevents the service from working if the required package is missing.

```
student@student-vm:~/nso300$ vi packages/l3mplsvpn/package-meta-data.xml

<ncs-package xmlns="http://tail-f.com/ns/ncs-packages">
  <name>l3mplsvpn</name>
  <package-version>1.1</package-version>
  <description>L3 MPLS VPN Python and Template Service</description>
  <ncs-min-version>5.3</ncs-min-version>

  <component>
    <name>L3 MPLS VPN</name>
    <application>
      <python-class-name>l3mplsvpn.l3mplsvpn.L3MplsVpn</python-class-name>
    </application>
  </component>

  <required-package>
    <name>resource-manager</name>
  </required-package>

</ncs-package>
```

Step 14

Create a new folder inside the **l3mplsvpn** service package for the Python code that will read and request the IDs.

```
student@student-vm:~/nso300$ mkdir packages/l3mplsvpn/python/l3mplsvpn/resource_manager
```

Step 15

For import statements to work correctly, create an empty **__init__.py** file in **l3mplsvpn/python/resource_manager**.

```
student@student-vm:~/nso300$ touch packages/l3mplsvpn/python/l3mplsvpn/resource_manager/__init__.py
```

Step 16

Copy the **id-allocator.py** code from the **id-allocator** package into the **l3mplsvpn** service.

```
student@student-vm:~/nso300$ cp ~/packages/resource-manager/python/resource_manager/id_allocator.py packages/l3mplsvpn/python/
l3mplsvpn/resource_manager
```

Step 17

Open the **id-allocator.py** file and study it.

You will use the *id_request()* and *id_read()* functions to allocate IDs, using the resource-manager package.

```
student@student-vm:~/nso300$ cat packages/l3mplsvpn/python/l3mplsvpn/resource_manager/id_allocator.py
import ncs
import ncs.maapi as maapi
import ncs.maagic as maagic


def id_request(service, svc_xpath, username,
               pool_name, allocation_name, sync, requested_id=-1):
    """Create an allocation request.

    After calling this function, you have to call response_ready
    to check the availability of the allocated ID.

    Example:
    import resource_manager.id_allocator as id_allocator
    pool_name = "The Pool"
    allocation_name = "Unique allocation name"

    # This will try to allocate the value 20 from the pool named 'The Pool'
    # using allocation name: 'Unique allocation name'
    id_allocator.id_request(service,
                            "/services/vl:loop-python[name='%s']" % (service.name),
                            tctx.username,
                            pool_name,
                            allocation_name,
                            False,
                            20)
```

```
    id = id_allocator.id_read(tctx.username, root,
                              pool_name, allocation_name)

    if not id:
        self.log.info("Alloc not ready")
        return

    print ("id = %d" % (id))

    Arguments:
    service -- the requesting service node
    svc_xpath -- xpath to the requesting service
    username -- username to use when redeploying the requesting service
    pool_name -- name of pool to request from
    allocation_name -- unique allocation name
    sync -- sync allocations with this name across pools
    requested_id -- a specific ID to be requested
    """
    template = ncs.template.Template(service)
    vars = ncs.template.Variables()
    vars.add("POOL", pool_name)
    vars.add("ALLOCATIONID", allocation_name)
    vars.add("USERNAME", username)
    vars.add("SERVICE", svc_xpath)
    vars.add("SYNC", sync)
    vars.add("REQUESTEDID", requested_id)
    template.apply('resource-manager-id-allocation', vars)


def id_read(username, root, pool_name, allocation_name):
    """Returns the allocated ID or None

    Arguments:
    username -- the requesting service's transaction's user
    root -- a maagic root for the current transaction
    pool_name -- name of pool to request from
    allocation_name -- unique allocation name
    """

    # Look in the current transaction
    id_pool_l = root.ralloc__resource_pools.id_pool

    if pool_name not in id_pool_l:
        raise LookupError("ID pool %s does not exist" % (pool_name))

    id_pool = id_pool_l[pool_name]

    if allocation_name not in id_pool.allocation:
        raise LookupError("allocation %s does not exist in pool %s" %
                          (allocation_name, pool_name))

    # Now we switch from the current trans to actually see if
    # we have received the alloc
    with maapi.single_read_trans(username, "system",
                                 db=ncs.OPERATIONAL) as th:
        id_pool_l = maagic.get_root(th).ralloc__resource_pools.id_pool

        if pool_name not in id_pool_l:
            return None

        id_pool = id_pool_l[pool_name]

        if allocation_name not in id_pool.allocation:
            return None

        alloc = id_pool.allocation[allocation_name]

        if alloc.response.id:
            return alloc.response.id
        elif alloc.response.error:
            raise LookupError(alloc.response.error)
        else:
            return None
```

Step 18

Copy the **l3mplsvpn** nanoservice from **~/solutions/packages/l3mplsvpn/nano** to the **~/nso300/packages/l3mplsvpn** folder.

```
student@student-vm:~/nso300$ cp -r ../solutions/packages/l3mplsvpn/nano/* packages/l3mplsvpn/
student@student-vm:~/nso300$
```

Step 19

Open the **l3mplsvpn.yang** YANG model.

```
student@student-vm:~/nso300$ vi packages/l3mplsvpn/src/yang/l3mplsvpn.yang
```

Step 20

Remove the **vpn-allocations** list from the **l3mplsvpn** YANG service model. The vpn ID will be allocated dynamically, and it not required to be

specified it in the service configuration.

The following text should be removed:

```
list vpn-allocations {
    key "vpn-name";
    unique "vpn-id";

    leaf vpn-name {
      type leafref {
        path /l3mplsvpn/name;
      }
    }

    leaf vpn-id {
      type uint16;
      tailf:info "Unique VPN ID";
    }
  }
```

The edited file should look like this:

```
module l3mplsvpn {
  namespace "http://example.com/l3mplsvpn";
  prefix l3mplsvpn;

  import ietf-inet-types {
    prefix inet;
  }

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

  description "Service for L3 MPLS VPN provisioning.";

  revision 2020-06-04 {
    description
      "Initial revision.";
  }

  identity l3mplsvpn {
    base ncs:plan-component-type;
  }

  identity l3mplsvpn-configured {
    base ncs:plan-state;
  }

  ncs:plan-outline l3mplsvpn-plan {
    description "L3 MPLS VPN Plan";

    ncs:component-type "ncs:self" {
      ncs:state "ncs:init";
      ncs:state "ncs:ready";
    }
    ncs:component-type "l3mplsvpn:l3mplsvpn" {
      ncs:state "ncs:init";
      ncs:state "l3mplsvpn:l3mplsvpn-configured" {
        ncs:create {
          ncs:pre-condition {
            ncs:monitor "/vpn-allocations[vpn-name=$SERVICE/name]/vpn-id";
          }
          ncs:nano-callback;
        }
      }
    }
  }

  ncs:service-behavior-tree l3mplsvpn-servicepoint {
    description "L3 MPLS VPN behavior tree";
    ncs:plan-outline-ref "l3mplsvpn:l3mplsvpn-plan";
    ncs:selector {
      ncs:create-component "'self'" {
        ncs:component-type-ref "ncs:self";
      }
      ncs:create-component "'l3mplsvpn'" {
        ncs:component-type-ref "l3mplsvpn:l3mplsvpn";
      }
    }
  }

  list l3mplsvpn {
    uses ncs:service-data;
    uses ncs:nano-plan-data;
    ncs:servicepoint l3mplsvpn-servicepoint;
    key name;

    leaf name {
```

```
      tailf:info "Service Instance Name";
      type string;
    }

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

    list link {
      tailf:info "PE-CE Attachment Point";
      key link-id;
      min-elements 1;

      leaf link-id {
        tailf:info "Link ID (1 to 65535)";
        type uint32 {
          range "1..65535" {
            error-message "Link ID is out of range. Should be between 1 and 65535.";
          }
        }
      }

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

      leaf routing-protocol {
        tailf:info "Routing option for the PE-CE link";
        type enumeration {
          enum bgp;
          enum rip;
        }
        default bgp;
      }

      leaf pe-ip {
        tailf:info "PE-CE Link IP Address";
        type inet:ipv4-address {
            pattern "172.(1[6-9]|2[0-9]|3[0-1])..*" {
              error-message "Invalid IP address. IP address should be in the 172.16.0.0/12 range.";
            }
          }
      }

      leaf ce-ip {
        tailf:info "CE Neighbor IP Address";
        when "../routing-protocol='bgp'";
          type inet:ipv4-address;
      }

      leaf rip-net {
        tailf:info "IP Network for RIP";
        when "../routing-protocol='rip'";
          type inet:ipv4-address;
      }

      leaf interface{
        tailf:info "Customer Facing Interface";
        mandatory true;
        type string;
        must "count(../../../l3mplsvpn[name != current()/../../name]/link[device = current()/../device]/interface = current())
= 0" {
            error-message "Interface is already used for another link.";
          }
        }
      }
    }
  }
}
```

Step 21

Add the id-allocated nano service state and define a create callback.

```
module l3mplsvpn {
  namespace "http://example.com/l3mplsvpn";
  prefix l3mplsvpn;

  import ietf-inet-types {
    prefix inet;
  }

  import tailf-ncs {
    prefix ncs;
```

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

  description "Service for L3 MPLS VPN provisioning.";

  revision 2020-06-04 {
    description
      "Initial revision.";
  }

  identity l3mplsvpn {
    base ncs:plan-component-type;
  }

  identity id-allocated {
    base ncs:plan-state;
  }

  identity l3mplsvpn-configured {
    base ncs:plan-state;
  }

  ncs:plan-outline l3mplsvpn-plan {
    description "L3 MPLS VPN Plan";

    ncs:component-type "ncs:self" {
      ncs:state "ncs:init";
      ncs:state "ncs:ready";
    }
    ncs:component-type "l3mplsvpn:l3mplsvpn" {
      ncs:state "ncs:init";
      ncs:state "l3mplsvpn:id-allocated" {
        ncs:create {
          ncs:nano-callback;
        }
      }
      ncs:state "l3mplsvpn:l3mplsvpn-configured" {
        ncs:create {
          ncs:pre-condition {
            ncs:monitor "/vpn-allocations[vpn-name=$SERVICE/name]/vpn-id";
          }
          ncs:nano-callback;
        }
      }
    }
  }

  ncs:service-behavior-tree l3mplsvpn-servicepoint {
    description "L3 MPLS VPN behavior tree";
    ncs:plan-outline-ref "l3mplsvpn:l3mplsvpn-plan";
    ncs:selector {
      ncs:create-component "'self'" {
        ncs:component-type-ref "ncs:self";
      }
      ncs:create-component "'l3mplsvpn'" {
        ncs:component-type-ref "l3mplsvpn:l3mplsvpn";
      }
    }
  }

  list l3mplsvpn {
    uses ncs:service-data;
    uses ncs:nano-plan-data;
    ncs:servicepoint l3mplsvpn-servicepoint;
    key name;

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

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

    list link {
      tailf:info "PE-CE Attachment Point";
      key link-id;
      min-elements 1;

      leaf link-id {
        tailf:info "Link ID (1 to 65535)";
        type uint32 {
          range "1..65535" {
            error-message "Link ID is out of range. Should be between 1 and 65535.";
          }
        }
      }

      leaf device {
```

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

      leaf routing-protocol {
        tailf:info "Routing option for the PE-CE link";
        type enumeration {
          enum bgp;
          enum rip;
        }
        default bgp;
      }

      leaf pe-ip {
        tailf:info "PE-CE Link IP Address";
        type inet:ipv4-address {
            pattern "172.(1[6-9]|2[0-9]|3[0-1])..*" {
              error-message "Invalid IP address. IP address should be in the 172.16.0.0/12 range.";
            }
          }
      }

      leaf ce-ip {
        tailf:info "CE Neighbor IP Address";
        when "../routing-protocol='bgp'";
          type inet:ipv4-address;
      }

      leaf rip-net {
        tailf:info "IP Network for RIP";
        when "../routing-protocol='rip'";
          type inet:ipv4-address;
      }

      leaf interface{
        tailf:info "Customer Facing Interface";
        mandatory true;
        type string;
        must "count(../../../l3mplsvpn[name != current()/../../name]/link[device = current()/../device]/interface = current())
= 0" {
            error-message "Interface is already used for another link.";
        }
      }
    }
  }
}
```

Step 22

Modify the pre-condition for the l3mplsvpn-initialized state. The pre-condition should monitor the resource-pool ID for the current service for changes. When the ID is set, the nano plan detects that the previous state has been reached (a specific ID has been allocated) and that it can trigger a callback for the current state.

```
module l3mplsvpn {
  namespace "http://example.com/l3mplsvpn";
  prefix l3mplsvpn;

  import ietf-inet-types {
    prefix inet;
  }

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

  description "Service for L3 MPLS VPN provisioning.";

  revision 2020-06-04 {
    description
      "Initial revision.";
  }

  identity l3mplsvpn {
    base ncs:plan-component-type;
  }

  identity id-allocated {
    base ncs:plan-state;
  }

  identity l3mplsvpn-configured {
    base ncs:plan-state;
  }
```

```
ncs:plan-outline l3mplsvpn-plan {
  description "L3 MPLS VPN Plan";

  ncs:component-type "ncs:self" {
    ncs:state "ncs:init";
    ncs:state "ncs:ready";
  }
  ncs:component-type "l3mplsvpn:l3mplsvpn" {
    ncs:state "ncs:init";
    ncs:state "l3mplsvpn:id-allocated" {
      ncs:create {
        ncs:nano-callback;
      }
    }
    ncs:state "l3mplsvpn:l3mplsvpn-configured" {
      ncs:create {
        ncs:pre-condition {
          ncs:monitor "/resource-pools/id-pool[name='vpn-id']/allocation[id=$SERVICE/name]/response/id";
        }
        ncs:nano-callback;
      }
    }
  }
}

ncs:service-behavior-tree l3mplsvpn-servicepoint {
  description "L3 MPLS VPN behavior tree";
  ncs:plan-outline-ref "l3mplsvpn:l3mplsvpn-plan";
  ncs:selector {
    ncs:create-component "'self'" {
      ncs:component-type-ref "ncs:self";
    }
    ncs:create-component "'l3mplsvpn'" {
      ncs:component-type-ref "l3mplsvpn:l3mplsvpn";
    }
  }
}

list l3mplsvpn {
  uses ncs:service-data;
  uses ncs:nano-plan-data;
  ncs:servicepoint l3mplsvpn-servicepoint;
  key name;

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

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

  list link {
    tailf:info "PE-CE Attachment Point";
    key link-id;
    min-elements 1;

    leaf link-id {
      tailf:info "Link ID (1 to 65535)";
      type uint32 {
        range "1..65535" {
          error-message "Link ID is out of range. Should be between 1 and 65535.";
        }
      }
    }

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

    leaf routing-protocol {
      tailf:info "Routing option for the PE-CE link";
      type enumeration {
        enum bgp;
        enum rip;
      }
      default bgp;
    }

    leaf pe-ip {
      tailf:info "PE-CE Link IP Address";
      type inet:ipv4-address {
          pattern "172.(1[6-9]|2[0-9]|3[0-1])..*" {
            error-message "Invalid IP address. IP address should be in the 172.16.0.0/12 range.";
```

```
            }
          }
      }

      leaf ce-ip {
        tailf:info "CE Neighbor IP Address";
        when "../routing-protocol='bgp'";
          type inet:ipv4-address;
      }

      leaf rip-net {
        tailf:info "IP Network for RIP";
        when "../routing-protocol='rip'";
          type inet:ipv4-address;
      }

      leaf interface{
        tailf:info "Customer Facing Interface";
        mandatory true;
        type string;
        must "count(../../../l3mplsvpn[name != current()/../../name]/link[device = current()/../device]/interface = current())
= 0" {
            error-message "Interface is already used for another link.";
        }
      }
    }
  }
}
```

Step 23

Save the file and exit the file editor.

Step 24

Reload the packages using the **make testenv-build** command.

```
student@student-vm:~/nso300$ make testenv-build
for NSO in $(Docker ps --format '{{.Names}}' --filter label=testenv-nso300-5.3.2-student --filter label=nidtype=nso); do \
        echo "-- Rebuilding for NSO: ${NSO}"; \
        Docker run -it --rm -v /home/student/nso300:/src --volumes-from ${NSO} --network=container:${NSO} -e NSO=${NSO} -e
PACKAGE_RELOAD= -e SKIP_LINT= -e PKG_FILE=nso300.gitlab.local/nso300/package:5.3.2-student nso300.gitlab.local/cisco-nso-
dev:5.3.2 /src/nid/testenv-build; \
done
-- Rebuilding for NSO: testenv-nso300-5.3.2-student-nso

< … Output Omitted … >

-- Reloading packages for NSO testenv-nso300-5.3.2-student-nso

>>> 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-asa-cli-6.10
    result true
}
reload-result {
    package cisco-ios-cli-6.54
    result true
}
reload-result {
    package cisco-iosxr-cli-7.26
    result true
}
reload-result {
    package cisco-nx-cli-5.15
    result true
}
reload-result {
    package l3mplsvpn
    result true
}
reload-result {
    package resource-manager
    result true
}
student@student-vm:~/nso300$
```

Step 25

Enter the NSO CLI, switch the CLI mode, and show the packages operational status.

```
student@student-vm:~/nso300$ make testenv-cli
Docker exec -it testenv-nso300-5.3.2-student-nso bash -lc 'ncs_cli -u admin'

admin connected from 127.0.0.1 using console on 387b47a68937
admin@ncs> switch cli
admin@ncs# show packages package oper-status | tab
                                                                                                    PACKAGE
                          PROGRAM                                                                   META     FILE
```

```
                      CODE     JAVA           PYTHON         BAD NCS  PACKAGE  PACKAGE  CIRCULAR    DATA     LOAD   ERROR
NAME                  UP  ERROR    UNINITIALIZED  UNINITIALIZED  VERSION  NAME     VERSION  DEPENDENCY  ERROR    ERROR  INFO
-------------------------------------------------------------------------------------------------------------------------------
cisco-asa-cli-6.10    X   -        -              -              -        -        -        -           -        -      -
cisco-ios-cli-6.54    X   -        -              -              -        -        -        -           -        -      -
cisco-iosxr-cli-7.26  X   -        -              -              -        -        -        -           -        -      -
cisco-nx-cli-5.15     X   -        -              -              -        -        -        -           -        -      -
l3mplsvpn             X   -        -              -              -        -        -        -           -        -      -
resource-manager      X   -        -              -              -        -        -        -           -        -      -
```

Step 26

Configure a pool of IDs from which you will allocate and deallocate vpn-ids. Go to the NSO CLI and configure a new ID pool:

```
admin@ncs# config
admin@ncs(config)# resource-pools id-pool vpn-id range start 100 end 199
admin@ncs(config-id-pool-vpn-id)# exclude range 101 110
admin@ncs(config-id-pool-vpn-id)# commit
Commit complete.
admin@ncs(config-id-pool-vpn-id)# exit
admin@ncs(config)# exit
admin@ncs# exit
student@student-vm:~/nso300$
```

Activity Verification

You have completed this task when you attain these results:

- You have successfully added a resource-manager package.
- You have successfully modified the YANG model.
- You have created a vpn-id resource pool.

Task 2: Modify Python Mapping Code

In this task, you will modify the Python mapping code for the l3mplsvpn nano service by adding a class for ID allocation requests.

Activity

Complete these steps:

Step 1

Edit the Python service code to match the changes made to the YANG model. Open the l3mplsvpn.py file.

```
student-vm:~/nso300$ vi packages/l3mplsvpn/python/l3mplsvpn/l3mplsvpn.py
```

This is how the file should appear when you open it for the first time.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
import math

class L3MPLSNanoService(ncs.application.NanoService):

    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state, proplist, component_proplist):
        self.log.debug("NanoService create ", state)

        vpn_id = root.vpn_allocations[service.name].vpn_id
        self.log.info(f'Vpn ID read: {vpn_id}')

        for link in service.link:

            vpn_link = {'pe-ip': link.pe_ip, 'ce-ip': link.ce_ip, 'rip-net': link.rip_net}

            # Calculate IP address from unique link ID: 172.x.y.z
            pe_ip_o2 = 16 + math.ceil(link.link_id / 4096)         # Second octet
            pe_ip_o3 = math.ceil((link.link_id % 4096) / 16)       # Third octet
            pe_ip_o4 = (link.link_id % 16) * 16 + 1                # Fourth octet
            ce_ip_o4 = pe_ip_o4 + 1                                # Fourth octet for CE side ( = PE + 1 )

            if not link.pe_ip:
                vpn_link['pe-ip'] = f'172.{pe_ip_o2}.{pe_ip_o3}.{pe_ip_o4}'
            if not link.ce_ip:
                vpn_link['ce-ip'] = f'172.{pe_ip_o2}.{pe_ip_o3}.{ce_ip_o4}'
            if not link.rip_net:
                vpn_link['rip-net'] = f'172.{pe_ip_o2}.0.0'

            tvars = ncs.template.Variables()
            template = ncs.template.Template(service)
            tvars.add('VPNID', vpn_id)
            tvars.add('DEVICE', link.device)
            tvars.add('PEIP', vpn_link['pe-ip'])
            tvars.add('CEIP', vpn_link['ce-ip'])
            tvars.add('ROUTING-PROTOCOL', link.routing_protocol)

            if link.routing_protocol == 'rip':
```

```
                tvars.add('RIP-NET', vpn_link['rip-net'])
            else:
                tvars.add('RIP-NET', '')

            tvars.add('INTERFACE', link.interface)
            self.log.info(f'Service create(applying template for device {link.device})')
            template.apply('l3mplsvpn-template', tvars)

        return proplist

# ------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ------------------------------------------
class L3MplsVpn(ncs.application.Application):
    def setup(self):
        self.log.info('L3MplsVpn RUNNING')
        self.register_nano_service('l3mplsvpn-servicepoint', 'l3mplsvpn:l3mplsvpn', 'l3mplsvpn:l3mplsvpn-configured',
L3MPLSNanoService)

    def teardown(self):
        self.log.info('L3MplsVpn FINISHED')
```

Step 2

Import the **id_allocator** from the **resource_manager** module.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
import math
from .resource_manager import id_allocator

< … Output Omitted … >
```

Step 3

Add the nano create callback functions for **id-alloated** state and register it as a component.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
import math
from .resource_manager import id_allocator

class IdAllocationNanoService(ncs.application.NanoService):

    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state, proplist, component_proplist):
        self.log.debug("NanoService create ", state)


class L3MPLSNanoService(ncs.application.NanoService):

    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state, proplist, component_proplist):
        self.log.debug("NanoService create ", state)

        vpn_id = root.vpn_allocations[service.name].vpn_id
        self.log.info(f'Vpn ID read: {vpn_id}')

        for link in service.link:

            vpn_link = {'pe-ip': link.pe_ip, 'ce-ip': link.ce_ip, 'rip-net': link.rip_net}

            # Calculate IP address from unique link ID: 172.x.y.z
            pe_ip_o2 = 16 + math.ceil(link.link_id / 4096)         # Second octet
            pe_ip_o3 = math.ceil((link.link_id % 4096) / 16)       # Third octet
            pe_ip_o4 = (link.link_id % 16) * 16 + 1                # Fourth octet
            ce_ip_o4 = pe_ip_o4 + 1                                # Fourth octet for CE side ( = PE + 1 )

            if not link.pe_ip:
                vpn_link['pe-ip'] = f'172.{pe_ip_o2}.{pe_ip_o3}.{pe_ip_o4}'
            if not link.ce_ip:
                vpn_link['ce-ip'] = f'172.{pe_ip_o2}.{pe_ip_o3}.{ce_ip_o4}'
            if not link.rip_net:
                vpn_link['rip-net'] = f'172.{pe_ip_o2}.0.0'

            tvars = ncs.template.Variables()
            template = ncs.template.Template(service)
            tvars.add('VPNID', vpn_id)
            tvars.add('DEVICE', link.device)
            tvars.add('PEIP', vpn_link['pe-ip'])
            tvars.add('CEIP', vpn_link['ce-ip'])
            tvars.add('ROUTING-PROTOCOL', link.routing_protocol)

            if link.routing_protocol == 'rip':
                tvars.add('RIP-NET', vpn_link['rip-net'])
            else:
                tvars.add('RIP-NET', '')

            tvars.add('INTERFACE', link.interface)
            self.log.info(f'Service create(applying template for device {link.device})')
            template.apply('l3mplsvpn-template', tvars)
```

```
        return proplist

# ------------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ------------------------------------------------
class L3MplsVpn(ncs.application.Application):
    def setup(self):
        self.log.info('L3MplsVpn RUNNING')
        self.register_nano_service('l3mplsvpn-servicepoint', 'l3mplsvpn:l3mplsvpn', 'l3mplsvpn:id-allocated',
IdAllocationNanoService)
        self.register_nano_service('l3mplsvpn-servicepoint', 'l3mplsvpn:l3mplsvpn', 'l3mplsvpn:l3mplsvpn-configured',
L3MPLSNanoService)

    def teardown(self):
        self.log.info('L3MplsVpn FINISHED')
        return proplist
```

Step 4

Add the code that uses the *id_allocator.id_request()* function to allocate a new ID to the ID allocation nano create callback method.

```
# -*- mode: python; python-indent: 4 -*-
import ncs
import math
from .resource_manager import id_allocator

class IdAllocationNanoService(ncs.application.NanoService):

    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state, proplist, component_proplist):
        self.log.debug("NanoService create ", state)

        service_path = f"/l3mplsvpn[name='{service.name}']"
        id_allocator.id_request(service, service_path, tctx.username, 'vpn-id', service.name, False)


class L3MPLSNanoService(ncs.application.NanoService):

    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state, proplist, component_proplist):
        self.log.debug("NanoService create ", state)

        vpn_id = root.vpn_allocations[service.name].vpn_id
        self.log.info(f'Vpn ID read: {vpn_id}')

        for link in service.link:

            vpn_link = {'pe-ip': link.pe_ip, 'ce-ip': link.ce_ip, 'rip-net': link.rip_net}

            # Calculate IP address from unique link ID: 172.x.y.z
            pe_ip_o2 = 16 + math.ceil(link.link_id / 4096)         # Second octet
            pe_ip_o3 = math.ceil((link.link_id % 4096) / 16)       # Third octet
            pe_ip_o4 = (link.link_id % 16) * 16 + 1                # Fourth octet
            ce_ip_o4 = pe_ip_o4 + 1                                # Fourth octet for CE side ( = PE + 1 )

            if not link.pe_ip:
                vpn_link['pe-ip'] = f'172.{pe_ip_o2}.{pe_ip_o3}.{pe_ip_o4}'
            if not link.ce_ip:
                vpn_link['ce-ip'] = f'172.{pe_ip_o2}.{pe_ip_o3}.{ce_ip_o4}'
            if not link.rip_net:
                vpn_link['rip-net'] = f'172.{pe_ip_o2}.0.0'

            tvars = ncs.template.Variables()
            template = ncs.template.Template(service)
            tvars.add('VPNID', vpn_id)
            tvars.add('DEVICE', link.device)
            tvars.add('PEIP', vpn_link['pe-ip'])
            tvars.add('CEIP', vpn_link['ce-ip'])
            tvars.add('ROUTING-PROTOCOL', link.routing_protocol)

            if link.routing_protocol == 'rip':
                tvars.add('RIP-NET', vpn_link['rip-net'])
            else:
                tvars.add('RIP-NET', '')

            tvars.add('INTERFACE', link.interface)
            self.log.info(f'Service create(applying template for device {link.device})')
            template.apply('l3mplsvpn-template', tvars)

        return proplist

# ---------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ---------------------------------------------
class L3MplsVpn(ncs.application.Application):
    def setup(self):
        self.log.info('L3MplsVpn RUNNING')
        self.register_nano_service('l3mplsvpn-servicepoint', 'l3mplsvpn:l3mplsvpn', 'l3mplsvpn:id-allocated',
IdAllocationNanoService)
        self.register_nano_service('l3mplsvpn-servicepoint', 'l3mplsvpn:l3mplsvpn', 'l3mplsvpn:l3mplsvpn-configured',
L3MPLSNanoService)

    def teardown(self):
        self.log.info('L3MplsVpn FINISHED')
```

Step 5

Modify the way that the **vpn_id** is read within the nano callback for the **l3mplsvpn-configured** state. It should now be allocated by the *Resource-Manager* package using the *id_read* function.

```
  # -*- mode: python; python-indent: 4 -*-
import ncs
import math
from .resource_manager import id_allocator

class IdAllocationNanoService(ncs.application.NanoService):

    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state, proplist, component_proplist):
        self.log.debug("NanoService create ", state)

        service_path = f"/l3mplsvpn[name='{service.name}']"
        id_allocator.id_request(service, service_path, tctx.username, 'vpn-id', service.name, False)
```

```
class L3MPLSNanoService(ncs.application.NanoService):

    @ncs.application.NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state, proplist, component_proplist):
        self.log.debug("NanoService create ", state)

        vpn_id  = id_allocator.id_read(tctx.username, root, 'vpn-id', service.name)
        self.log.info(f'Vpn ID read: {vpn_id}')

        for link in service.link:

            vpn_link = {'pe-ip': link.pe_ip, 'ce-ip': link.ce_ip, 'rip-net': link.rip_net}

            # Calculate IP address from unique link ID: 172.x.y.z
            pe_ip_o2 = 16 + math.ceil(link.link_id / 4096)         # Second octet
            pe_ip_o3 = math.ceil((link.link_id % 4096) / 16)       # Third octet
            pe_ip_o4 = (link.link_id % 16) * 16 + 1                # Fourth octet
            ce_ip_o4 = pe_ip_o4 + 1                                # Fourth octet for CE side ( = PE + 1 )

            if not link.pe_ip:
                vpn_link['pe-ip'] = f'172.{pe_ip_o2}.{pe_ip_o3}.{pe_ip_o4}'
            if not link.ce_ip:
                vpn_link['ce-ip'] = f'172.{pe_ip_o2}.{pe_ip_o3}.{ce_ip_o4}'
            if not link.rip_net:
                vpn_link['rip-net'] = f'172.{pe_ip_o2}.0.0'

            tvars = ncs.template.Variables()
            template = ncs.template.Template(service)
            tvars.add('VPNID', vpn_id)
            tvars.add('DEVICE', link.device)
            tvars.add('PEIP', vpn_link['pe-ip'])
            tvars.add('CEIP', vpn_link['ce-ip'])
            tvars.add('ROUTING-PROTOCOL', link.routing_protocol)

            if link.routing_protocol == 'rip':
                tvars.add('RIP-NET', vpn_link['rip-net'])
            else:
                tvars.add('RIP-NET', '')

            tvars.add('INTERFACE', link.interface)
            self.log.info(f'Service create(applying template for device {link.device})')
            template.apply('l3mplsvpn-template', tvars)

        return proplist

# ---------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ---------------------------------------------
class L3MplsVpn(ncs.application.Application):
    def setup(self):
        self.log.info('L3MplsVpn RUNNING')
        self.register_nano_service('l3mplsvpn-servicepoint', 'l3mplsvpn:l3mplsvpn', 'l3mplsvpn:id-allocated',
IdAllocationNanoService)
        self.register_nano_service('l3mplsvpn-servicepoint', 'l3mplsvpn:l3mplsvpn', 'l3mplsvpn:l3mplsvpn-configured',
L3MPLSNanoService)

    def teardown(self):
        self.log.info('L3MplsVpn FINISHED')
```

Step 6

Save and exit the file.

Activity Verification

You have completed this task when you attain this result:

- You have successfully modified the nano service.

Task 3: Compile and Deploy the Service

In this task, you will compile and deploy the nano service created in the previous tasks and observe how the resource-manager ID allocation works.

Activity

Complete these steps:

Step 1

Reload the packages again, by using the **make testenv-build** command.

```
student@student-vm:~/nso300$ make testenv-build
for NSO in $(Docker ps --format '{{.Names}}' --filter label=testenv-nso300-5.3.2-student --filter label=nidtype=nso); do \
        echo "-- Rebuilding for NSO: ${NSO}"; \
        Docker run -it --rm -v /home/student/nso300:/src --volumes-from ${NSO} --network=container:${NSO} -e NSO=${NSO} -e
PACKAGE_RELOAD= -e SKIP_LINT= -e PKG_FILE=nso300.gitlab.local/nso300/package:5.3.2-student nso300.gitlab.local/cisco-nso-
dev:5.3.2 /src/nid/testenv-build; \
done
-- Rebuilding for NSO: testenv-nso300-5.3.2-student-nso

< … Output Omitted … >
```

```
student@student-vm:~/nso300$
```

Step 2

Connect to the NSO Docker CLI by using the **make testenv-cli** command.

```
student@student-vm:~/nso300$ make testenv-cli
Docker exec -it testenv-nso300-5.3.2-student-nso bash -lc 'ncs_cli -u admin'

admin connected from 127.0.0.1 using console on 387b47a68937
admin@ncs>
```

Step 3

Switch the CLI type.

```
admin@ncs> switch cli
admin@ncs#
```

Step 4

Create a new customer ACME.

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# customers customer ACME
admin@ncs(config-customer-ACME)# top
admin@ncs(config)#
```

Step 5

Configure a new L3VPN service instance.

```
admin@ncs(config)# l3mplsvpn vpn3 customer ACME
admin@ncs(config-l3mplsvpn-vpn3)# link 1 device PE11 interface 0/1
admin@ncs(config-link-1)# exit
admin@ncs(config-l3mplsvpn-vpn3)# link 2 device PE12 interface 0/1
admin@ncs(config-link-2)# top
```

Step 6

Commit the transaction and verify the status of the nano plan with the **show l3mpls plan** command.

You can see that the plan has been successfully started and finished. Both of your custom plan states have been reached. Note that the plan was executed only for the latest service instance, since the others were created without nanoservices.

```
admin@ncs(config)# commit
Commit complete.
admin@ncs(config)# exit
admin@ncs# show l3mplsvpn plan | tab
                                                                                                                                                POST
                      LOG                           BACK
ACTION
NAME  FAILED  MESSAGE  ENTRY  TYPE       NAME       TRACK  GOAL  STATE                 STATUS   WHEN                 ref
STATUS  ID
-------------------------------------------------------------------------------------------------------------------------------------------------
vpn3  -       -        -      self       self       false  -     init                  reached  2020-07-23T12:54:49  -    -
                                                                  ready                 reached  2020-07-23T12:54:49  -    -
                             l3mplsvpn  l3mplsvpn  false  -     init                  reached  2020-07-23T12:54:49  -    -
                                                                  id-allocated          reached  2020-07-23T12:54:49  -    -
                                                                  l3mplsvpn-configured  reached  2020-07-23T12:54:49  -    -
admin@ncs#
```

Step 7

Check which vpn-id has been assigned to the vpn3 service instance.

You can see that the newly assigned ID is 100.

```
admin@ncs# show resource-pools id-pool
NAME    ID    ERROR  ID
-------------------------
vpn-id  vpn3  -      100
admin@ncs#
```

Step 8

Verify that the configuration for vpn-id 100 exists on devices PE11 and PE12.

```
admin@ncs# config
admin@ncs(config)# show full-configuration devices device PE11 config interface GigabitEthernet 0/1
```

```
devices device PE11
 config
  interface GigabitEthernet0/1
   no switchport
   vrf forwarding vpn100
   ip address 172.17.1.17 255.255.255.252
   no shutdown
  exit
 !
!
admin@ncs(config)# show full-configuration devices device PE11 config router
devices device PE11
 config
  router bgp 1
   address-family ipv4 unicast vrf vpn100
    redistribute connected
    redistribute static
    neighbor 172.17.1.18 remote-as 65001
    exit-address-family
   !
 !
!
admin@ncs(config)# show full-configuration devices device PE12 config interface GigabitEthernet 0/1
devices device PE12
 config
  interface GigabitEthernet0/1
   no switchport
   vrf forwarding vpn100
   ip address 172.17.1.33 255.255.255.252
   no shutdown
  exit
 !
!
admin@ncs(config)# show full-configuration devices device PE12 config router
devices device PE12
 config
  router bgp 1
   address-family ipv4 unicast vrf vpn100
    redistribute connected
    redistribute static
    neighbor 172.17.1.34 remote-as 65001
    exit-address-family
   !
  !
 !
!
```

Step 9

Configure another L3VPN service instance.

```
admin@ncs(config)# l3mplsvpn vpn4 customer ACME
admin@ncs(config-l3mplsvpn-vpn4)# link 1 device PE11 interface 0/0
admin@ncs(config-link-1)# exit
admin@ncs(config-l3mplsvpn-vpn4)# link 2 device PE12 interface 0/0
admin@ncs(config-link-2)# top
admin@ncs(config)# commit
Commit complete.
admin@ncs(config)# exit
```

Step 10

Verify that the allocated ID range has been excluded from the selection in the vpn-id resource pool.

The ID allocated for vpn4 should be 111, because you excluded a range of 101-110 from selection.

```
admin@ncs# show resource-pools id-pool vpn-id
NAME    ID    ERROR  ID
--------------------------
vpn-id  vpn3  -      100
        vpn4  -      111
admin@ncs#
```

Activity Verification

You have completed this task when you attain this result:

- You have successfully deployed and verified service instances using nano services.