Implement Nano Services

Introduction

In this activity, you will develop a nano service. The resulting package, *l3mplsvpn,* will be a VPN service, which will require the customer to configure parameters except for the vpn-id, which will be configured by the network operator. Only when the vpn-id is assigned will the service reach the *l3mplsvpn-configured* state.

To enable this, you will delete the vpn-id leaf from the existing l3mplsvpn service. You will then define a separate list for the network operator to hold the allocated vpn-id values.

Because your service now requires two steps for successful deployment, you will change the service model to implement *nano services*, which enables you to have partially configured services that automatically transition to the next state when specific conditions are met. You decide what state constitutes the final goal, by creating a *nano service plan*.

After completing this activity, you will be able to:

- Modify an existing service to use ID allocation.
- Transform a service into a nano service.
- Deploy and track the progress of a nano service.

Job Aids

The following job aids are 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

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.

Command	Comment
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 the NSO CLI directly from the 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:

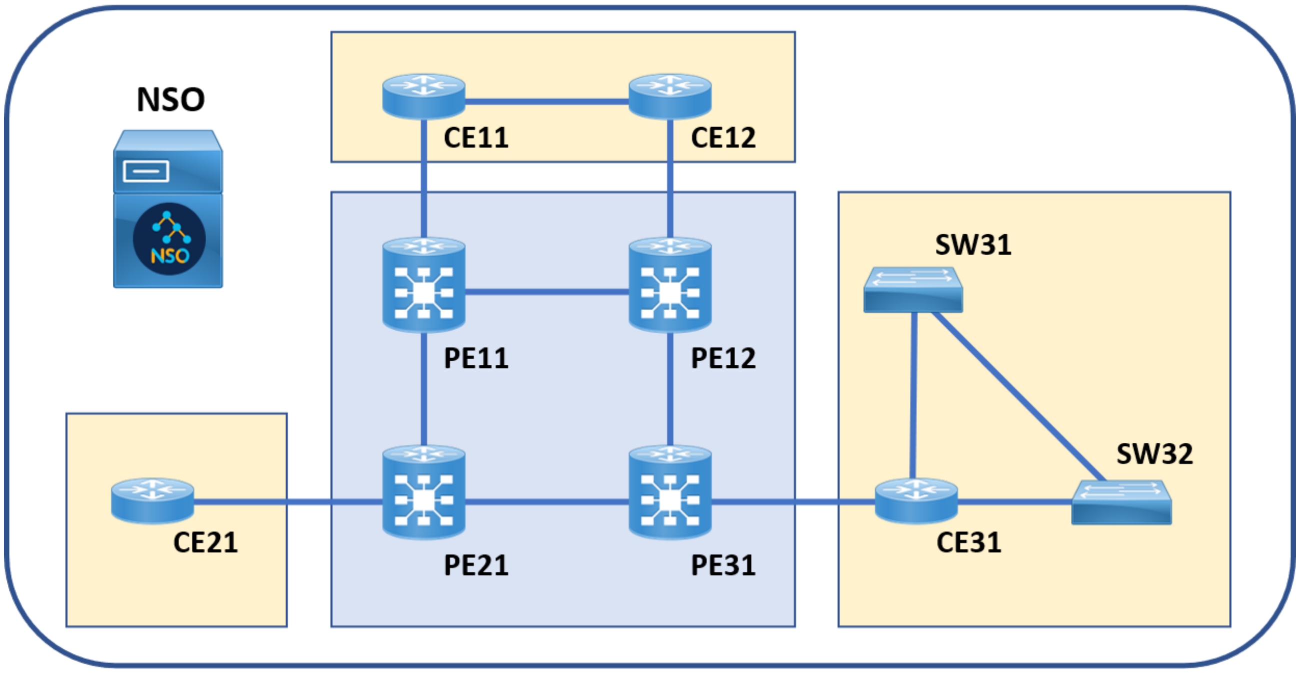
Command	Comment
make build	Builds the main NSO Docker image.
make testenv-start	Starts the NSO Docker environment.
make testenv-stop	Stops the NSO Docker environment.
make testenv-build	Recompiles and reloads the NSO packages.
make testenv-cli	Enters the NSO CLI of the NSO Docker container.
make testenv-shell	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 one 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 network will be the one that you will orchestrate with your NSO.

- The 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 configuration sent by the NSO just as real devices would.

Topology



Task 1: Modify Service Model

In this task, you will modify the existing service model in two steps. First, you will create a separate list for the **vpn-id** allocations for the network operator. Next, you will delete the existing **vpn-id** leaf.

The final solution for this lab is located in the *~/solutions/packages/l3mplsvpn/nano* directory. You can use it for copying and 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.

Step 3

Change directory to **/home/student/nso300**

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

Step 4

From the Terminal window, copy the existing **l3mplsvpn** package.

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

Step 5

List the contents of the **packages/l3mplsvpn** directory.

```
student@student-vm:~/nso300$ ls packages/l3mplsvpn
package-meta-data.xml  python  README  src  templates
```

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

Step 6

Open the service model file using a text editor of your choice.

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

Step 7

Delete the existing **vpn-id** leaf.

```
< ... Output Omitted ... >

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

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

    leaf customer {
      tailf:info "VPN Customer";

< ... Output Omitted ... >
```

Step 8

Create a list **vpn-allocations** with two leaves—**vpn-name** and **vpn-id**. Declare **vpn-name** to be used as a **key** leaf and use a unique constraint for the **vpn-id**. Make sure that **vpn-name** refers to the **name** leaf located in the **l3mplsvpn** list.

```
< ... Output Omitted ... >

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

  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";
    }
  }

< ... Output Omitted ... >
```

Step 9

Save the file and exit the text editor.

Activity Verification

You have completed this task when you attain these results:

- You have successfully copied a service package
- You have deleted the existing vpn-id leaf
- You have declared a list to store VPN allocations.

Task 2: Change to Nano Services

In this task, you will change the service model from the previous task and implement nano service. This change will consist of three key steps.

First, the service must include the **ncs:nano-plan-data** grouping which will change the execution model of your service**.**

Second, the service must define a **nano plan** declared with the **ncs:plan-outline** statement. Nano plan definitions consist of plan components and **plan state** definitions.

Finally, you must define a service **behavior tree** declared with the **ncs:service-behavior-tree** statement. The behavior tree is responsible for creating and removing plan components from instances of your service plan.

Activity

Complete these steps:

Step 1

Open the service model file using a text editor of your choice and study the service model.

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

Step 2

Find the **l3mplsvpn** list definition and declare the use of **nano-plan-data**.

```
< ... Output Omitted ... >

  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;
    }

< ... Output Omitted ... >
```

Step 3

Declare the **plan component** and **plan state** for later use in the plan outline.

```
< ... Output Omitted ... >

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

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

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

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

< ... Output Omitted ... >
```

Step 4

Define your service plan outline and add a mandatory **self** component with two mandatory states—**init** and **ready**.

```
< ... Output Omitted ... >

  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";
    }
  }

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

< ... Output Omitted ... >
```

init and **ready** states are primarily used as placeholders for timestamps.

Step 5

Define the **l3mplsvpn** component type and define two possible states. However, this time, use the **l3mplsvpn:l3mplsvpn-configured** for the second state declaration.

You have formally declared these identities previously.

```
< ... Output Omitted ... >

  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" {
      }
    }
  }

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

< ... Output Omitted ... >
```

Step 6

Finally, add a **create** nano callback and define a **pre-condition** inside the block. Write an XPath expression and pass it as an argument to the **monitor** statement. This will make sure that **l3mplsvpn-configured** state can be reached only if the **monitor** condition is satisfied. Make sure you add a **nano-callback** declaration right after the **pre-condition** block.

```
 < ... Output Omitted ... >

  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;
        }
      }
    }
  }

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

< ... Output Omitted ... >
```

Step 7

Write a behavior tree for your service by providing the name of your service point and the name of your plan outline.

```
< ... Output Omitted ... >

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

  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";
      }
    }
  }

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

< ... Output Omitted ... >
```

A behavior tree is responsible for creating components instantiated from your service plan.

Step 8

Save changes and close the file.

Step 9

Open the **l3mplsvpn.py** file using a text editor of your choice.

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

Step 10

Delete the highlighted **ServiceCallbacks** class declaration, **@Service.create** decorator, and **cb_create** method declaration along with the call to the **self.log.debug** method.

Nano services require class inheritance from a different class than standard services.

```
import ncs
import math

class ServiceCallbacks(ncs.application.Service):

    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info('Service create(service=', service._path, ')')

< ... Output Omitted ... >
```

Step 11

Declare a class **L3MPLSNanoService**, which inherits from the **ncs.application.NanoService** class. Use the **@ncs.application.NanoService.create** decorator and define the **cb_nano_create** method with a call to the **self.log.debug** method.

```
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)

< ... Output Omitted ... >
```

cb_nano_create accepts additional parameters—**plan**, **component**, **state**, and **component_proplist**.

Step 12

Delete the line with the **vpn_id** declaration. Remember that your service model no longer provides the **vpn-id**.

```
< ... Output Omitted ... >

       vpn_id = service.vpn_id

< ... Output Omitted ... >
```

Step 13

Add lines to obtain **vpn-id** from **vpn-allocations** list, and a call to **self.log.info** method. Make sure that you use underscores and not hyphen when declaring variables.

```
< ... Output Omitted ... >

        self.log.debug("NanoService create ", state)

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

< ... Output Omitted ... >
```

Step 14

Scroll down to near the end of the file and add a **return proplist** statement. Also, add a call to **self.register_nano_service** with your service parameters.

```

< ... Output Omitted ... >

            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')

< ... Output Omitted ... >
```

Step 15

Save changes and exit the text editor.

Step 16

Compile and deploy your package.

```

student@student-vm:~/nso300$ make testenv-build

< ... Output Omitted ... >

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

< ... Output Omitted ... >
```

Activity Verification

You have completed this task when you attain these results:

- You have successfully transformed the service into a nano service.

- You have successfully built the nano service package.

Task 3: Deploy Nano Services

In this task, you will deploy the newly created nano service.

Activity

Complete these steps:

Step 1

Enter the NSO CLI and switch to the Cisco mode.

```
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 ebdb0d8aa9a2
admin@ncs> switch cli
admin@ncs#
```

Step 2

Enter the configuration mode and create a new customer entry called **ACME**.

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

Step 3

Commit the configuration and return to the top.

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

Step 4

Configure a new VPN service instance for the new customer and name it **vpn512**.

```
admin@ncs(config)# l3mplsvpn vpn512 customer ACME
admin@ncs(config-l3mplsvpn-vpn512)#
```

Step 5

Configure two links, each on a different device.

```
admin@ncs(config-l3mplsvpn-vpn512)# link 1 device PE11 interface 0/1
admin@ncs(config-link-1)# exit
admin@ncs(config-l3mplsvpn-vpn512)# link 2 device PE21 interface 0/2
admin@ncs(config-link-2)# exit
admin@ncs(config-l3mplsvpn-vpn512)#
```

Step 6

Commit the configuration and exit configuration mode.

```
admin@ncs(config-l3mplsvpn-vpn512)# commit
Commit complete.
admin@ncs(config-l3mplsvpn-vpn512)# end
```

```
admin@ncs#
```

Step 7

Inspect the status of the newly configured service instance.

Plan component **l3mplsvpn** has not reached the **l3mplsvpn-configured** state because a **pre-condition** exists, which requires the presence of a **vpn-id** value in the **vpn-allocations** list.

```
admin@ncs# show l3mplsvpn vpn512

POST
                      BACK
ACTION
TYPE       NAME       TRACK  GOAL  STATE                 STATUS       WHEN                 ref
STATUS
-------------------------------------------------------------------------------------------------------
self       self       false  -     init                  reached      2020-09-15T18:10:11  -    -
                                   ready                 reached      2020-09-15T18:10:11  -    -
l3mplsvpn  l3mplsvpn  false  -     init                  reached      2020-09-15T18:10:11  -    -
                                   l3mplsvpn-configured  not-reached  -                    -    -

admin@ncs#
```

The NSO CLI adapts the output to the width of your terminal window. In case your window is too narrow, you can use the **<command> | tab** format, to force the NSO CLI to display the tabulated output.

Step 8

Enter the configuration mode.

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)#
```

Step 9

Use the **vpn-allocations** command to allocate a new **vpn-id** to the service instance.

You are taking the role of a network operator that is acting upon a new VPN service request.

```
admin@ncs(config)# vpn-allocations ?
Possible completions:
  vpn512
admin@ncs(config)# vpn-allocations vpn512 vpn-id 512
admin@ncs(config-vpn-allocations-vpn512)#
```

Step 10

Commit the configuration and exit the configuration mode.

```
admin@ncs(config-vpn-allocations-vpn512)# commit
Commit complete.
admin@ncs(config-vpn-allocations-vpn512)# end
admin@ncs#
```

Step 11

Display the status of the service instance **vpn512** one more time.

Notice that the plan component **l3mplsvpn** has now reached the **l3mplsvpn-configured** state after the **pre-condition** was satisfied.

```
admin@ncs# show l3mplsvpn vpn512
```

```
l3mplsvpn vpn512
 modified devices [ PE21 ]
 directly-modified devices [ PE21 ]
 device-list [ PE21 ]
                                                                                                     POST
                      BACK
ACTION
TYPE       NAME       TRACK  GOAL  STATE                 STATUS   WHEN                 ref
STATUS
----------------------------------------------------------------------------------------------------
self       self       false  -     init                  reached  2020-09-15T18:10:11  -    -
                                   ready                 reached  2020-09-15T18:10:11  -    -
l3mplsvpn  l3mplsvpn  false  -     init                  reached  2020-09-15T18:10:11  -    -
                                   l3mplsvpn-configured  reached  2020-09-15T18:11:53  -    -

admin@ncs#
```

Activity Verification

You have completed this task when you attain these results:

- You have successful configured service instance.
- Your plan component l3mplsvpn has reached l3mplsvpn-configured state.