Discovery 7: Migrate a Monolithic Service to LSA

Introduction

In this activity, you will migrate an existing monolithic L3VPN service to an LSA service.

You will create three different packages from the original service—a customer-facing service (*cfs*), a resource-facing service (*rfs*), and a resource-facing NED package (*rfs-ned*). The cfs and the rfs packages are needed for each of the corresponding LSA nodes, and the rfs-ned package is needed for the upper node to talk to the lower nodes.

After completing this activity, you will be able to meet these objectives:

- Modify a service to become a resource-facing service.
- Create a NETCONF NED from a resource-facing service.
- Modify a service to become a customer-facing service.

Job Aid

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-server	student	1234QWer

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 List

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

Linux Shell:

Command	Comment
source /opt/ncs/ncs-6.1/ncsrc	Source NSO environmental variable in Docker container.
ls|ll	Display contents of the current directory.

Command	Comment
cd	Move directly to user home directory.
cd ..	Exit out of current directory.
cd test	Move into the "test" folder, which is a subfolder of the current directory.
cd /home/student	Move into the "nso300" folder by specifying the direct path to it starting from the root of the directory system.
ncs_cli -C	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.

Command Syntax Reference

This lab guide uses the following conventions for command syntax:

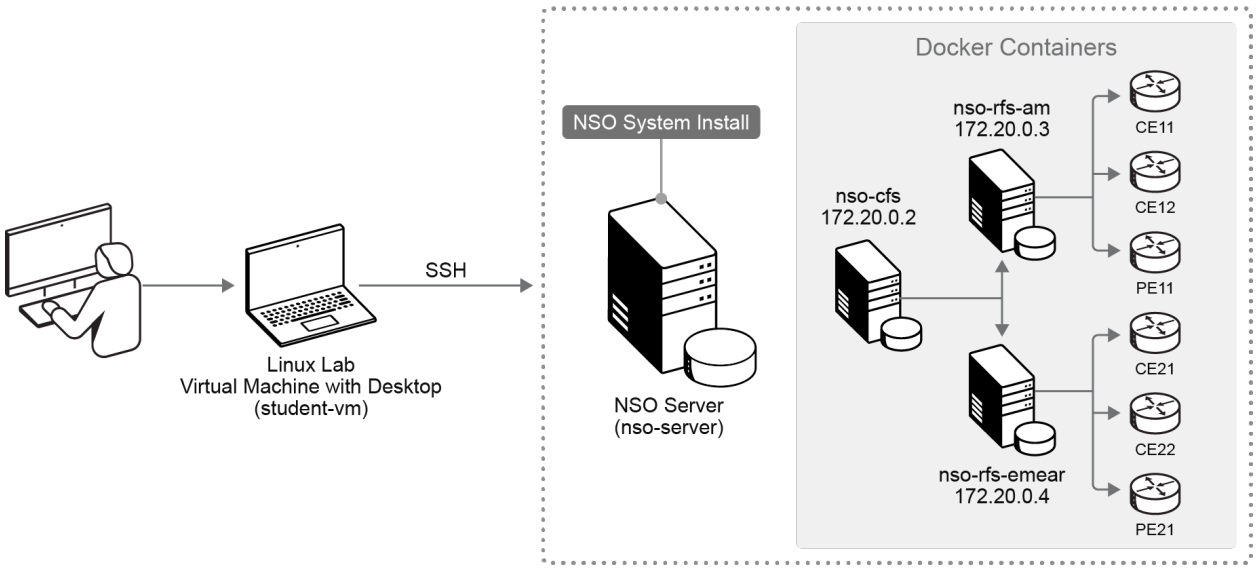
Formatting	Description and Examples
show running config	Commands in steps use this formatting.
Example	Type **show running config**
Example	Use the **name** command.

Formatting	Description and Examples
`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` `6.1`
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 *10.0.0.102*
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.

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. Your lab environment is a Linux server (Student-VM) acting as a jumphost, and a Linux server (NSO-server) acting as an Docker environment that consists of three NSO Docker containers with your NSO CFS and RFS servers, and six Docker containers with your NetSim devices.

Topology



Task 1: Create a l3vpn-rfs Package

In this task, you will modify an existing L3VPN service and transform it into a resource-facing service. The resource-facing service is generally the same as the existing service, and can function on its own, without the customer-facing service.

> The final solutions for this lab are in the **~/packages** directory. You can use it for copy-pasting longer pieces of code and as a reference point for troubleshooting your packages.

Activity

Step 1

Connect to the Student-VM.

You can connect to the server either by choosing the **Student-VM** from the device list or by clicking on the **Student-VM** icon in the topology map.

Step 2

Open the terminal window.

Open the terminal window by clicking the **Terminal** icon in the bottom bar.

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

Step 3

Connect to the **nso-server** NSO server.

Connect to the **nso-server** NSO server with the **student** user using the SSH client. The authentication is already preconfigured with public key authentication, therefore the password is not needed. The prompt will change, stating that you are now

connected to the nso-server.

```
student@student-vm:~$ ssh student@nso-server
Last login: Tue Oct  3 09:14:42 2023 from 10.0.0.102
student@nso-server:~$
```

Step 4

Display the Docker containers currently running with the **docker ps -a** command.

There are three NSO containers, together with six NetSim device containers. The **testenv-nso-cfs-6.1-student-nso container** hosts the upper, customer-facing node, and **the testenv-nso-rfs-6.1-student-nso-am** and **testenv-nso-rfs-6.1-student-nso-emea** containers host the lower, resource-facing nodes. Each RFS node is designated to a specific geographical region (*am*—Americas, *emea*—Europe, Middle East, and Africa).

```
student@nso-server:~$ docker ps -a
CONTAINER ID   IMAGE
COMMAND            CREATED          STATUS                     PORTS

NAMES
155c8add696f   nso303.gitlab.local/ned-iosxr/netsim:6.1-student   "/
run-netsim.sh"   40 minutes ago   Up 40 minutes

testenv-PE21-emea-6.1-student
a5d61b8d275e   nso303.gitlab.local/ned-ios/netsim:6.1-student     "/
run-netsim.sh"   40 minutes ago   Up 40 minutes

testenv-CE22-emea-6.1-student
4166a267f7f5   nso303.gitlab.local/ned-ios/netsim:6.1-student     "/
run-netsim.sh"   40 minutes ago   Up 40 minutes

testenv-CE21-emea-6.1-student
56430efd52b7   nso303.gitlab.local/ned-iosxr/netsim:6.1-student   "/
run-netsim.sh"   40 minutes ago   Up 40 minutes

testenv-PE11-am-6.1-student
5abb0fc584da   nso303.gitlab.local/ned-ios/netsim:6.1-student     "/
run-netsim.sh"   40 minutes ago   Up 40 minutes

testenv-CE12-am-6.1-student
8c4a2d6bef00   nso303.gitlab.local/ned-ios/netsim:6.1-student     "/
run-netsim.sh"   40 minutes ago   Up 40 minutes

testenv-CE11-am-6.1-student
5bede4191199   nso303.gitlab.local/nso-rfs/nso:6.1-student        "/
run-nso.sh"      40 minutes ago   Up 40 minutes (healthy)   22/tcp, 80/
tcp, 443/tcp, 830/tcp, 4334/tcp
testenv-nso-rfs-6.1-student-nso-emea
4cb432374e30   nso303.gitlab.local/nso-rfs/nso:6.1-student        "/
run-nso.sh"      40 minutes ago   Up 40 minutes (healthy)   22/tcp, 80/
tcp, 443/tcp, 830/tcp, 4334/tcp
```

```
testenv-nso-rfs-6.1-student-nso-am
e91348045b91   nso303.gitlab.local/nso-cfs/nso:6.1-student        "/
run-nso.sh"      40 minutes ago   Up 40 minutes (healthy)   80/tcp,
443/tcp, 830/tcp, 4334/tcp, 0.0.0.0:8022->22/tcp, :::8022->22/tcp
testenv-nso-cfs-6.1-student-nso
student@nso-server:~$
```

Step 5

Navigate to the **nso-lsa** directory and display its contents with the **ls** command.

The *nso-lsa* folder contains two different NSO Docker System projects: one for the customer-facing node (*nso-cfs*) and one for resource-facing nodes (*nso-rfs*). You can use the standard **make** commands for NSO Docker that you have learned so far, such as **make testenv-build** and **make testenv-cli**. Because two different docker containers are built from the nso-rfs image, you can specify the container by adding an NSO parameter together with the region acronym to the command (for example, **make testenv-cli NSO=am** or **make testenv-cli NSO=emea**).

The NSO Docker project does not yet feature an NSO LSA skeleton.

```
student@nso-server:~$ cd nso-lsa
student@nso-server:~/nso-lsa$ ls
nso-cfs  nso-rfs
student@nso-server:~/nso-lsa$
```

Step 6

Navigate to the **nso-rfs** directory.

Use the **cd nso-rfs** command.

```
student@nso-server:~/nso-lsa$ cd nso-rfs
student@nso-server:~/nso-lsa/nso-rfs$
```

Step 7

Enter the shell of the development container.

Use the **make dev-shell** command.

```
student@nso-server:~/nso-lsa/nso-rfs$ make dev-shell
docker run -it -v $(pwd):/src nso303.gitlab.local/cisco-nso-dev:6.1
root@5984bb12d155:/#
```

Step 8

Navigate to the **/src/packages** folder, which is mapped to the host machine, and create a new Python and template-based NSO package.

Name the package **nso-rfs**, using the **ncs-make-package** command.

```
root@7620438f8070:/# cd src/packages/
root@7620438f8070:/src/packages# ncs-make-package --service-skeleton
python-and-template l3vpn-rfs
root@7620438f8070:/src/packages#
```

Step 9

Change the ownership of the package to a non-root user.

The owner with the UID 1000 is the user 'student' in this case.

```
root@7620438f8070:/src/packages# chown -Rv 1000:1000 l3vpn-rfs/
changed ownership of 'l3vpn-rfs/test/internal/lux/service/dummy-
service.xml' from root:root to 1000:1000
changed ownership of 'l3vpn-rfs/test/internal/lux/service/pyvm.xml'
from root:root to 1000:1000
changed ownership of 'l3vpn-rfs/test/internal/lux/service/dummy-
device.xml' from root:root to 1000:1000
changed ownership of 'l3vpn-rfs/test/internal/lux/service/run.lux' from
root:root to 1000:1000
changed ownership of 'l3vpn-rfs/test/internal/lux/service/Makefile'
from root:root to 1000:1000
changed ownership of 'l3vpn-rfs/test/internal/lux/service' from
root:root to 1000:1000
changed ownership of 'l3vpn-rfs/test/internal/lux/Makefile' from
root:root to 1000:1000
changed ownership of 'l3vpn-rfs/test/internal/lux' from root:root to
1000:1000
changed ownership of 'l3vpn-rfs/test/internal/Makefile' from root:root
to 1000:1000
changed ownership of 'l3vpn-rfs/test/internal' from root:root to
1000:1000
changed ownership of 'l3vpn-rfs/test/Makefile' from root:root to
1000:1000
changed ownership of 'l3vpn-rfs/test' from root:root to 1000:1000
changed ownership of 'l3vpn-rfs/README' from root:root to 1000:1000
changed ownership of 'l3vpn-rfs/templates/l3vpn-rfs-template.xml' from
root:root to 1000:1000
changed ownership of 'l3vpn-rfs/templates' from root:root to 1000:1000
changed ownership of 'l3vpn-rfs/package-meta-data.xml' from root:root
to 1000:1000
changed ownership of 'l3vpn-rfs/python/l3vpn_rfs/main.py' from
root:root to 1000:1000
changed ownership of 'l3vpn-rfs/python/l3vpn_rfs/__init__.py' from
root:root to 1000:1000
changed ownership of 'l3vpn-rfs/python/l3vpn_rfs' from root:root to
1000:1000
changed ownership of 'l3vpn-rfs/python' from root:root to 1000:1000
changed ownership of 'l3vpn-rfs/src/yang/l3vpn-rfs.yang' from root:root
to 1000:1000
changed ownership of 'l3vpn-rfs/src/yang' from root:root to 1000:1000
changed ownership of 'l3vpn-rfs/src/Makefile' from root:root to
1000:1000
```

```
changed ownership of 'l3vpn-rfs/src' from root:root to 1000:1000
changed ownership of 'l3vpn-rfs/' from root:root to 1000:1000
root@7620438f8070:/src/packages#
```

Step 10

Exit the development container.

Use the **exit** command.

```
root@7620438f8070:/src/packages# exit
logout
student@nso-server:~/nso-lsa/nso-rfs$
```

Step 11

Replace the generated YANG model with the existing model from a monolithic l3vpn service.

Copy the YANG model and rename it to **l3vpn-rfs.yang** in the process. You can find the existing **l3vpn** service in the **~/packages** folder.

```
student@nso-server:~/nso-lsa/nso-rfs$ cp -r ~/packages/l3vpn/src/yang/
l3vpn.yang packages/l3vpn-rfs/src/yang/l3vpn-rfs.yang
student@nso-server:~/nso-lsa/nso-rfs$
```

Step 12

Open and edit the copied YANG model.

Rename the module, namespace, service list, and prefix to **l3vpn-rfs**. Rename the servicepoint to **l3vpn-rfs-servicepoint**. Optionally, modify the model description as well.

```
student@nso-server:~/nso-lsa/nso-rfs$ nano packages/l3vpn-rfs/src/yang/
l3vpn-rfs.yang
module l3vpn-rfs {
  namespace "http://cisco.com/example/l3vpn-rfs";
  prefix l3vpn-rfs;

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

  list l3vpn-rfs {
    description "This is an L3VPN resource facing service model.";

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

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

    list link {
      key id;

      leaf id {
        tailf:info "Unique L3VPN link id";
        tailf:cli-allow-range;
        type string;
      }

      leaf device {
        tailf:info "Device";
        type leafref {
          path "/ncs:devices/ncs:device/ncs:name";
        }
      }


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

      leaf ip-address {
        tailf:info "Remote IP Address";
        type inet:ipv4-address;
      }

      leaf mask {
        tailf:info "Subnet mask for Remote IP Address";
        type inet:ipv4-address;
      }
    }
  }
}
student@nso-server:~/nso-lsa/nso-rfs$
```

Step 13

Save the file and exit the file editor.

Use the **Ctrl+X** keys to exit the file editor. Confirm saving changes with **Yes**.

Step 14

Copy and rename the XML template and the **main.py** Python file from the existing **l3vpn** package, which is located in the **~/packages** folder, and replace the corresponding files in your **l3vpn-rfs** package.

Use the following commands:

```
student@nso-server:~/nso-lsa/nso-rfs$ cp ~/packages/l3vpn/templates/
l3vpn-template.xml packages/l3vpn-rfs/templates/l3vpn-rfs-template.xml
student@nso-server:~/nso-lsa/nso-rfs$ cp ~/packages/l3vpn/python/l3vpn/
```

```
main.py packages/l3vpn-rfs/python/l3vpn_rfs/main.py
student@nso-server:~/nso-lsa/nso-rfs$
```

Step 15

Open the copied Python file and change the servicepoint registration to **l3vpn-rfs-servicepoint**, and the template used in the service callback to **l3vpn-rfs-template**.

Use the **nano packages/l3vpn-rfs/python/l3vpn_rfs/main.py** command to open the file.

```
 student@nso-server:~/nso-lsa/nso-rfs$ nano packages/l3vpn-rfs/python/
l3vpn_rfs/main.py
# -*- mode: python; python-indent: 4 -*-
import ncs
from ncs.application import Service
import string
import random


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

    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info(f"Service create(service='{service._path}')")
        for link in service.link:
          vars = ncs.template.Variables()
          vars.add('ID', link.id)
          vars.add('NAME', service.vpn_name)
          vars.add('DEVICE', link.device)
          vars.add('IP-ADDRESS', link.ip_address)
          vars.add('INTERFACE', link.interface)
          vars.add('MASK', link.mask)
          vars.add('RT', "65000:" + str(random.randrange(1, 65535)))
          template = ncs.template.Template(service)
          template.apply('l3vpn-rfs-template', vars)

# ---------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ---------------------------------------------
class Main(ncs.application.Application):
    def setup(self):
        self.log.info('Main RUNNING')
        self.register_service('l3vpn-rfs-servicepoint',
ServiceCallbacks)

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

Step 16

Save the file and exit the file editor.

Use the **Ctrl+X** keys to exit the file editor. Confirm saving changes with **Yes**.

Step 17

Recompile and reload the package.

Use the **make testenv-build** command. The package should be reloaded successfully.

```
student@student-vm:~/nso-lsa/nso-rfs$ make testenv-build
for NSO in $(docker ps --format '{{.Names}}' --filter label=testenv-
nso-rfs-6.1-student --filter label=nidtype=nso)>        echo "--
Rebuilding for NSO: ${NSO}"; \
        docker run -it --rm -v /home/student/nso-lsa/nso-rfs:/src --
volumes-from ${NSO} --network=container:${NSO} >done
-- Rebuilding for NSO: testenv-nso-rfs-6.1-student-nso-emea
(package-meta-data.xml|\.cli1|\.yang1)
...
...
======== MAKE TIDY
make: Leaving directory '/var/opt/ncs/packages/cisco-iosxr-cli-7.41/
src'
make: Entering directory '/src/packages/cisco-iosxr-cli-7.41'
if [ ! -f build-meta-data.xml ]; then \
        export PKG_NAME=$(xmlstarlet sel -N x=http://tail-f.com/ns/ncs-
packages -t -v '/x:ncs-package/x:name' $(ls >        export
PKG_VERSION=$(xmlstarlet sel -N x=http://tail-f.com/ns/ncs-packages -t
-v '/x:ncs-package/x:package->        eval "cat <<< \"$(</src/nid/
build-meta-data.xml)\"" > /var/opt/ncs/packages/cisco-iosxr-cli-7.41/
build-meta>make: Leaving directory '/src/packages/cisco-iosxr-cli-7.41'
make: Entering directory '/var/opt/ncs/packages/l3vpn-rfs/src'
make: Nothing to be done for 'all'.
make: Leaving directory '/var/opt/ncs/packages/l3vpn-rfs/src'
make: Entering directory '/src/packages/l3vpn-rfs'
if [ ! -f build-meta-data.xml ]; then \
        export PKG_NAME=$(xmlstarlet sel -N x=http://tail-f.com/ns/ncs-
packages -t -v '/x:ncs-package/x:name' $(ls >        export
PKG_VERSION=$(xmlstarlet sel -N x=http://tail-f.com/ns/ncs-packages -t
-v '/x:ncs-package/x:package->        eval "cat <<< \"$(</src/nid/
build-meta-data.xml)\"" > /var/opt/ncs/packages/l3vpn-rfs/build-meta-
data.xml; >make: Leaving directory '/src/packages/l3vpn-rfs'
-- Reloading packages for NSO testenv-nso-rfs-6.1-student-nso-am
reload-result {
    package cisco-ios-cli-6.85
    result true
}
reload-result {
    package cisco-iosxr-cli-7.41
    result true
}
reload-result {
    package l3vpn-rfs
    result true
}student@student-vm:~/nso-lsa/nso-rfs$
```

The **make testenv-build** command rebuilds the packages for both of the RFS nodes.

Activity Verification

You have completed this task when you attain these results:

- You have successfully modified and built the **l3vpn-rfs** package.

Task 2: Create a l3vpn-rfs-ned Package

In this task, you will create a *l3vpn-rfs-ned* package from your *l3vpn-rfs* service. This package is an LSA NETCONF NED package that is used by the CFS node to send the service configuration to the RFS node.

Activity

Step 1

Copy the **l3vpn-rfs** package to the **packages** folder of the **~/nso-lsa/nso-cfs** directory.

Use the **cp -r packages/l3vpn-rfs ../nso-cfs/packages/** command.

```
student@nso-server:~/nso-lsa/nso-rfs$ cp -r packages/l3vpn-rfs ../nso-
cfs/packages/
student@nso-server:~/nso-lsa/nso-rfs$
```

Step 2

Navigate to the **~/nso-lsa/nso-cfs** folder and enter the shell of the development container for the CFS node. Once there, navigate to the **/src/packages folder**.

Use the commands provided in the following output.

```
student@nso-server:~/nso-lsa/nso-rfs$ cd ../nso-cfs/
student@nso-server:~/nso-lsa/nso-cfs$ make dev-shell
docker run -it -v $(pwd):/src nso303.gitlab.local/cisco-nso-dev:6.1
root@ef32a3f30ade:/# cd src/packages/
root@ef32a3f30ade:/src/packages#
```

Step 3

Use the **ncs-make-package** command to create an LSA NETCONF NED. The **--lsa-netconf-ned** parameter requires the path to the resource-facing services **yang** folder.

When the service model is updated, the LSA NETCONF NED needs to be re-created.

```
root@ef32a3f30ade:/src/packages# ncs-make-package --lsa-netconf-ned
l3vpn-rfs/src/yang l3vpn-rfs-ned
root@ef32a3f30ade:/src/packages#
```

Step 4

Delete the **l3vpn-rfs** service package.

The service package it is not needed on a CFS node. Use the **rm -rf l3vpn-rfs** command.

```
root@ef0df7368810:/src/packages# rm -rf l3vpn-rfs
root@ef0df7368810:/src/packages#
```

Step 5

Exit the development container.

Use the **exit** command to exit.

```
root@ef32a3f30ade:/src/packages# exit
logout
student@nso-server:~/nso-lsa/nso-cfs$
```

Step 6

Open and edit the NED YANG model.

Change any device references to string data types.

```
student@nso-server:~/nso-lsa/nso-cfs$ sudo nano packages/l3vpn-rfs-ned/
src/yang/l3vpn-rfs.yang
module l3vpn-rfs {
  namespace "http://cisco.com/example/l3vpn-rfs";
  prefix l3vpn-rfs;

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

  list l3vpn-rfs {
    description "This is an L3VPN resource facing service model.";

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

    uses ncs:service-data;
```

```
    ncs:servicepoint l3vpn-rfs-servicepoint;

    list link {
      key id;

      leaf id {
        tailf:info "Unique L3VPN link id";
        tailf:cli-allow-range;
        type string;
      }

      leaf device {
        tailf:info "Device";
        type string;
      }

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

    leaf ip-address {
      tailf:info "Remote IP Address";
      type inet:ipv4-address;
    }

    leaf mask {
      tailf:info "Subnet mask for Remote IP Address";
      type inet:ipv4-address;
    }
  }
  }
}
student@nso-server:~/nso-lsa/nso-cfs$
```

Step 7

Save the file and exit the file editor.

Use the **Ctrl+X** keys to exit the file editor. Confirm saving the changes with **Yes**.

Step 8

Recompile and reload the package.

Use the **make testenv-build** command. The package should reload successfully.

```
student@nso-server:~/nso-lsa/nso-cfs$ make testenv-build
for NSO in $(docker ps --format '{{.Names}}' --filter label=testenv-
nso-cfs-6.1-student --filter label=nidtype=nso); do \
        echo "-- Rebuilding for NSO: ${NSO}"; \
        docker run -it --rm -v /home/student/nso-lsa/nso-cfs:/src --
volumes-from ${NSO} --network=container:${NSO} -e NSO=${NSO} -e
PACKAGE_RELOAD= -e SKIP_LINT= -e PKG_FILE=nso303.gitlab.local/nso-cfs/
package:6.1-student nso303.gitlab.local/cisco-nso-dev:6.1 /src/nid/
testenv-build; \
done
```

```
-- Rebuilding for NSO: testenv-nso-cfs-6.1-student-nso
(package-meta-data.xml|\.cli1|\.yang1)
make: Entering directory '/var/opt/ncs/packages/l3vpn-rfs-ned/src'
...

BUILD SUCCESSFUL
...
...
make: Leaving directory '/src/packages/l3vpn-rfs-ned'
-- Reloading packages for NSO testenv-nso-cfs-6.1-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 l3vpn-rfs-ned
    result true
}
student@nso-server:~/nso-lsa/nso-cfs$
```

Activity Verification

You have completed this task when you attain these results:

- You have successfully created and built a **l3vpn-rfs-ned** LSA NETCONF NED package.

Task 3: Create a l3vpn-cfs Package

In this task, you will modify an existing L3VPN service and transform it into a customer-facing service. In addition to the l3vpn logic, the customer-facing service will also include the dispatch logic for the lower layer nodes.

Activity

Step 1

Enter the shell of the development container.

Using the **make dev-shell** command.

```
student@nso-server:~/nso-lsa/nso-cfs$ make dev-shell
docker run -it -v $(pwd):/src nso303.gitlab.local/cisco-nso-dev:6.1
root@04169f5eb568:/#
```

Step 2

Navigate to the **/src/packages** folder, which is mapped to the host machine, and create a new Python and template-based NSO package, named **nso-cfs**.

To create a new package, use the **ncs-make-package** command.

```
root@04169f5eb568:/# cd src/packages/
root@04169f5eb568:/src/packages# ncs-make-package --service-skeleton
python-and-template l3vpn-cfs
root@04169f5eb568:/src/packages#
```

Step 3

Change the ownership of the package to a non-root user.

The owner with the UID 1000 is the user 'student' in this case.

```
root@04169f5eb568:/src/packages# chown -Rv 1000:1000 l3vpn-cfs/
changed ownership of 'l3vpn-cfs/test/internal/lux/service/dummy-
service.xml' from root:root to 1000:1000
changed ownership of 'l3vpn-cfs/test/internal/lux/service/pyvm.xml'
from root:root to 1000:1000
changed ownership of 'l3vpn-cfs/test/internal/lux/service/dummy-
device.xml' from root:root to 1000:1000
changed ownership of 'l3vpn-cfs/test/internal/lux/service/run.lux' from
root:root to 1000:1000
changed ownership of 'l3vpn-cfs/test/internal/lux/service/Makefile'
from root:root to 1000:1000
changed ownership of 'l3vpn-cfs/test/internal/lux/service' from
root:root to 1000:1000
changed ownership of 'l3vpn-cfs/test/internal/lux/Makefile' from
root:root to 1000:1000
changed ownership of 'l3vpn-cfs/test/internal/lux' from root:root to
1000:1000
changed ownership of 'l3vpn-cfs/test/internal/Makefile' from root:root
to 1000:1000
changed ownership of 'l3vpn-cfs/test/internal' from root:root to
1000:1000
changed ownership of 'l3vpn-cfs/test/Makefile' from root:root to
1000:1000
changed ownership of 'l3vpn-cfs/test' from root:root to 1000:1000
changed ownership of 'l3vpn-cfs/README' from root:root to 1000:1000
changed ownership of 'l3vpn-cfs/templates/l3vpn-cfs-template.xml' from
root:root to 1000:1000
changed ownership of 'l3vpn-cfs/templates' from root:root to 1000:1000
changed ownership of 'l3vpn-cfs/package-meta-data.xml' from root:root
to 1000:1000
changed ownership of 'l3vpn-cfs/python/l3vpn_cfs/main.py' from
root:root to 1000:1000
changed ownership of 'l3vpn-cfs/python/l3vpn_cfs/__init__.py' from
root:root to 1000:1000
changed ownership of 'l3vpn-cfs/python/l3vpn_cfs' from root:root to
1000:1000
changed ownership of 'l3vpn-cfs/python' from root:root to 1000:1000
changed ownership of 'l3vpn-cfs/src/yang/l3vpn-cfs.yang' from root:root
to 1000:1000
changed ownership of 'l3vpn-cfs/src/yang' from root:root to 1000:1000
changed ownership of 'l3vpn-cfs/src/Makefile' from root:root to
1000:1000
changed ownership of 'l3vpn-cfs/src' from root:root to 1000:1000
changed ownership of 'l3vpn-cfs/' from root:root to 1000:1000
root@04169f5eb568:/src/packages#
```

Step 4

Exit the development container.

Use the **exit** command to exit.

```
root@04169f5eb568:/src/packages# exit
logout
student@nso-server:~/nso-lsa/nso-cfs$
```

Step 5

Replace the generated YANG model with the existing model from a monolithic **l3vpn** service.

Copy the YANG model and rename it to **l3vpn-cfs.yang** in the process. You can find the existing l3vpn service in the **~/packages** folder.

```
student@nso-server:~/nso-lsa/nso-cfs$ cp -r ~/packages/l3vpn/src/yang/
l3vpn.yang packages/l3vpn-cfs/src/yang/l3vpn-cfs.yang
student@nso-server:~/nso-lsa/nso-cfs$
```

Step 6

Open and edit the copied YANG model.

Rename the module, namespace, and prefix to **l3vpn-cfs**, and rename the service point to **l3vpn-cfs-servicepoint**. Also, change any device references to string data types.

```
student@nso-server:~/nso-lsa/nso-cfs$ nano packages/l3vpn-cfs/src/yang/
l3vpn-cfs.yang
module l3vpn-cfs {
  namespace "http://cisco.com/example/l3vpn-cfs";
  prefix l3vpn-cfs;

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

  list l3vpn-cfs {
    description "This is an L3VPN service model.";

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

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

    list link {
```

```
      key id;

      leaf id {
        tailf:info "Unique L3VPN link id";
        tailf:cli-allow-range;
        type string;
      }

      leaf device {
        tailf:info "Device";
        type string;
      }

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

      leaf ip-address {
        tailf:info "Remote IP Address";
        type inet:ipv4-address;
      }

      leaf mask {
        tailf:info "Subnet mask for Remote IP Address";
        type inet:ipv4-address;
      }
    }
  }
}
student@nso-server:~/nso-lsa/nso-cfs$
```

Step 7

Add a **dispatch-map** list below the imports.

Map the RFS nodes within the dispatch map by geographical region. The region should be represented by any string and the RFS node should reference an existing device added to the CFS NSO.

Optionally optimize the dispatch map CLI syntax by dropping the **rfs-node** command from the CLI terminal with the use of the **cli-drop-node-name** statement on rfs-node and **cli-suppress-mode** on the dispatch map list. This action allows you to enter the RFS device directly after specifying the region.

```
student@nso-server:~/nso-lsa/nso-cfs$ nano packages/l3vpn-cfs/src/yang/
l3vpn-cfs.yang
module l3vpn-cfs {
  namespace "http://cisco.com/example/l3vpn-cfs";
  prefix l3vpn-cfs;

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

```
  list dispatch-map {
    key region;
    tailf:cli-suppress-mode;

    leaf region {
      type string;
    }
    leaf rfs-node {
      tailf:cli-drop-node-name;
      type leafref {
        path "/ncs:devices/ncs:device/ncs:name";
      }
    }
  }

  list l3vpn-cfs {
    description "This is an L3VPN service model.";

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

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

    list link {
      key id;

      leaf id {
        tailf:info "Unique L3VPN link id";
        tailf:cli-allow-range;
        type string;
      }

      leaf device {
        tailf:info "Device";
        type string;
      }

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

      leaf ip-address {
        tailf:info "Remote IP Address";
        type inet:ipv4-address;
      }

      leaf mask {
        tailf:info "Subnet mask for Remote IP Address";
        type inet:ipv4-address;
      }
    }
  }
```

```
}
student@nso-server:~/nso-lsa/nso-cfs$
```

Step 8

Save the file and exit the file editor.

Use the **Ctrl+X** keys to exit the file editor. Confirm saving the changes with **Yes**.

Step 9

Copy the already prepared service XML template file.

The file is already created for your convenience.

```
student@nso-server:~/nso-lsa/nso-cfs$ cp -r ~/packages/l3vpn-cfs/
templates/l3vpn-cfs-template.xml packages/l3vpn-cfs/templates/l3vpn-
cfs-template.xml
student@nso-server:~/nso-lsa/nso-cfs$
```

Step 10

Review the **l3vpn-cfs** service XML template file.

Mapping of the parameters from the CFS service to the RFS service is created. Each parameter maps into itself, with the addition of the device name being **RFS-NODE**. This way, the same exact service configuration gets created, just on the appropriate RFS node.

```
student@student-vm:~/nso-lsa/nso-cfs$ cat packages/l3vpn-cfs/templates/
l3vpn-cfs-template.xml
<config-template xmlns="http://tail-f.com/ns/config/1.0">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>{$RFS-NODE}</name>
      <config>
        <l3vpn-rfs xmlns="http://cisco.com/example/l3vpn-rfs">
          <vpn-name>{string(../vpn-name)}</vpn-name>
          <link>
            <id>{$ID}</id>
            <device>{$DEVICE}</device>
            <interface>{$INTERFACE}</interface>
            <ip-address>{$IP-ADDRESS}</ip-address>
            <mask>{$MASK}</mask>
          </link>
        </l3vpn-rfs>
      </config>
    </device>
  </devices>
</config-template>
student@nso-server:~/nso-lsa/nso-cfs$
```

Step 11

Copy the already prepared **main.py** Python file.

Copy the file from the existing **l3vpn-cfs** package, which is located in the **~/packages** folder for your convenience. This will replace the corresponding files in your **l3vpn-cfs** package.

```
student@nso-server:~/nso-lsa/nso-cfs$ cp ~/packages/l3vpn-cfs/python/
l3vpn_cfs/main.py packages/l3vpn-cfs/python/l3vpn_cfs/main.py
```

Step 12

Open the copied Python file and review the changes.

The service point registration is changed from l3vpn-servicepoint to **l3vpn-cfs-servicepoint**, and the template used in the service callback to **l3vpn-cfs-template**. Also, service logic is added that finds an RFS node, which maps to the region contained in the device name. The device name represents the RFS node. The dispatch map is accessible directly from the *root* variable. The added method is *get_rfs_node*.

```
student@nso-server:~/nso-lsa/nso-cfs$ cat packages/l3vpn-cfs/python/
l3vpn_cfs/main.py
# -*- mode: python; python-indent: 4 -*-
import ncs
from ncs.application import Service
import string
import random


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

    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info(f"Service create(service='{service._path}')")
        for link in service.link:
            vars = ncs.template.Variables()
            rfs_node = get_rfs_node(root, link.device)
            vars.add('RFS-NODE', rfs_node)
            vars.add('ID', link.id)
            vars.add('DEVICE', link.device)
            vars.add('IP-ADDRESS', link.ip_address)
            vars.add('INTERFACE', link.interface)
            vars.add('MASK', link.mask)
            template = ncs.template.Template(link)
            template.apply('l3vpn-cfs-template', vars)




    def get_rfs_node(root, device):
        for mapping in root.dispatch_map:
```

```
            if mapping.region in device:
                return root.dispatch_map[mapping.region].rfs_node

# ---------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ---------------------------------------------
class Main(ncs.application.Application):
    def setup(self):
        self.log.info('Main RUNNING')
        self.register_service('l3vpn-cfs-servicepoint',
ServiceCallbacks)

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

Step 13

Build the package and make sure that it is successfully loaded.

Use the **make testenv-build** command.

```
student@nso-server:~/nso-lsa/nso-cfs$ make testenv-build
for NSO in $(docker ps --format '{{.Names}}' --filter label=testenv-
nso-cfs-6.1-student --filter label=nidtype=nso); do \
        echo "-- Rebuilding for NSO: ${NSO}"; \
        docker run -it --rm -v /home/student/nso-lsa/nso-cfs:/src --
volumes-from ${NSO} --network=container:${NSO} -e NSO=${NSO} -e
PACKAGE_RELOAD= -e SKIP_LINT= -e PKG_FILE=nso303.gitlab.local/nso-cfs/
package:6.1-student nso303.gitlab.local/cisco-nso-dev:6.1 /src/nid/
testenv-build; \
done
...
...

>>> 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 l3vpn-cfs
    result true
}
reload-result {
    package l3vpn-rfs-ned
    result true
}
student@nso-server:~/nso-lsa/nso-cfs$
```

Activity Verification

You have completed this task when you attain these results:

- You have successfully created and built a **l3vpn-cfs** package with LSA dispatch logic.

Which functionality determines to which RFS node the service instance configuration of a device must be sent from the CFS node?

- No additional code is needed
- The dispatching logic in the LSA NED
- The dispatching logic on the CFS node
- The dispatching logic on the RFS node