Deploy LSA Service

Introduction

In this activity, you will deploy a *l3vpn* service, which uses a layered services architecture. You will also learn how to configure a basic LSA environment, which consists of a customer-facing NSO and two additional resource (devices)-facing NSO servers, in this case, one for each geographical region (AM – Americas, EMEAR – Europe, Middle East, Africa, and Russia).

After completing this activity, you will be able to:

- Configure a basic NSO LSA environment
- Deploy an LSA service
- Understand LSA implications

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

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**

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

NSO CLI:

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

Makefile commands for Docker environment:

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

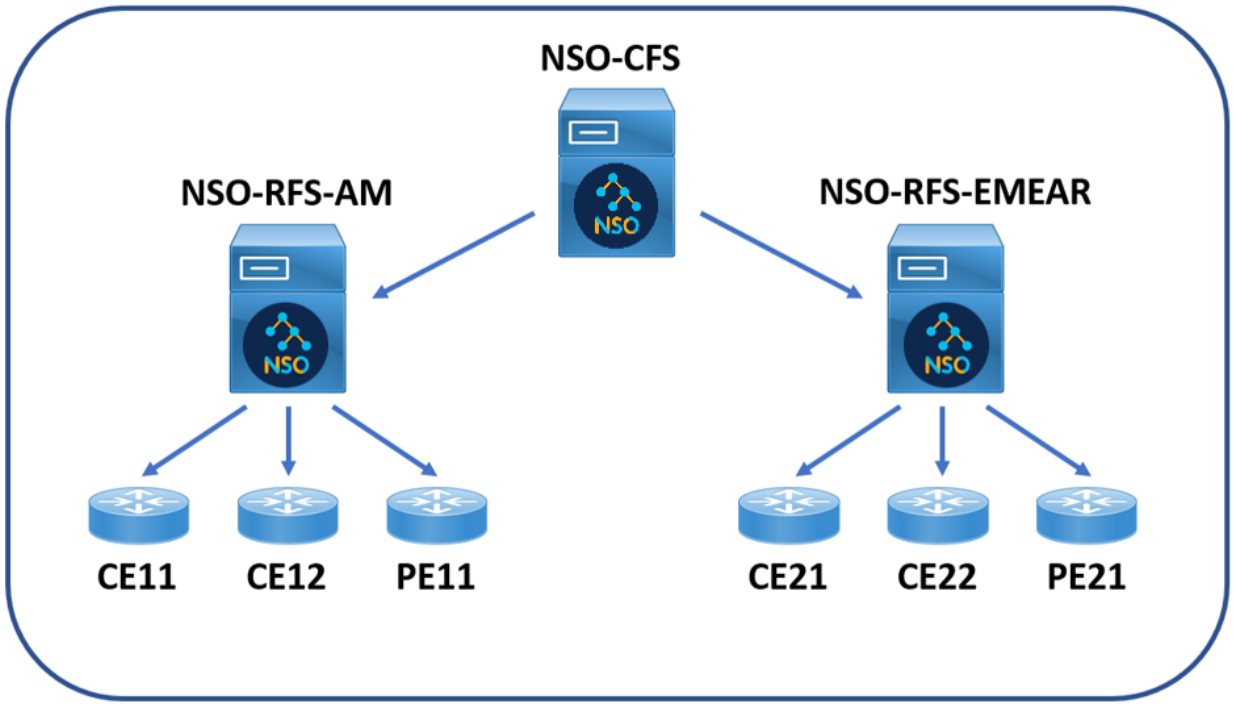
Command	Comment
make testenv-shell	Enters the Linux shell of the NSO Docker container.
make testenv-start	Starts the NSO Docker environment.
make testenv-stop	Stops the NSO Docker environment.

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: Configure RFS Nodes

In this task, you will configure and explore the resource-facing NSO nodes. These nodes are responsible for deploying the device configuration to the actual network devices.

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

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:~$
```

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

There are 3 NSO containers and 6 NetSim device containers. The **testenv-nso-cfs-5.3.2-student-nso** container hosts the upper customer-facing node, and the ***t*estenv-nso-rfs-5.3.2-student-nso-am** and **testenv-nso-rfs-5.3.2-student-nso-emear** containers host the lower resource-facing nodes. Each RFS node is designated to a specific geographical region (*am*—Americas, *emear*—Europe, Middle East, Africa, and Russia).

```
student@student-vm:~$ docker ps -a
CONTAINER ID        IMAGE
COMMAND             CREATED             STATUS                      PORTS
NAMES
b47c1dbee77f        nso300.gitlab.local/ned-iosxr/netsim:5.3.2-student
"/run-netsim.sh"    10 minutes ago      Up 10 minutes
testenv-PE21-emear-5.3.2-student
67a8556d16f9        nso300.gitlab.local/ned-ios/netsim:5.3.2-student
"/run-netsim.sh"    10 minutes ago      Up 10 minutes
testenv-CE22-emear-5.3.2-student
4f424cdcf849        nso300.gitlab.local/ned-ios/netsim:5.3.2-student
"/run-netsim.sh"    10 minutes ago      Up 10 minutes
testenv-CE21-emear-5.3.2-student
dc6201c86501        nso300.gitlab.local/ned-iosxr/netsim:5.3.2-student
"/run-netsim.sh"    10 minutes ago      Up 10 minutes
testenv-PE11-am-5.3.2-student
f2603a7193e3        nso300.gitlab.local/ned-ios/netsim:5.3.2-student
"/run-netsim.sh"    10 minutes ago      Up 10 minutes
testenv-CE12-am-5.3.2-student
7e2caf861c90        nso300.gitlab.local/ned-ios/netsim:5.3.2-student
"/run-netsim.sh"    10 minutes ago      Up 10 minutes
testenv-CE11-am-5.3.2-student
ab5dcd0880a7        nso300.gitlab.local/nso-rfs/nso:5.3.2-student
```

```
"/run-nso.sh"       10 minutes ago      Up 10 minutes (healthy)   22/
tcp, 80/tcp, 443/tcp, 830/tcp, 4334/tcp   testenv-nso-rfs-5.3.2-
student-nso-emear
9be403da125e        nso300.gitlab.local/nso-rfs/nso:5.3.2-student
"/run-nso.sh"       10 minutes ago      Up 10 minutes (healthy)   22/
tcp, 80/tcp, 443/tcp, 830/tcp, 4334/tcp   testenv-nso-rfs-5.3.2-
student-nso-am
6752ef552cf4        nso300.gitlab.local/nso-cfs/nso:5.3.2-student
"/run-nso.sh"       10 minutes ago      Up 10 minutes (healthy)   22/
tcp, 80/tcp, 443/tcp, 830/tcp, 4334/tcp   testenv-nso-cfs-5.3.2-
student-nso
student@student-vm:~$
```

Step 3

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=emear**).

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

Step 4

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

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

Step 5

Enter the Docker container shell with the **make testenv-shell NSO=am** command.

```
student@student-vm:~/nso-lsa/nso-rfs$ make testenv-shell NSO=am
docker exec -it testenv-nso-rfs-5.3.2-student-nso-am bash -l
root@a0f0d24a825f:/#
```

Step 6

Display the NSO configuration that is located at **/etc/ncs/ncs.conf**. Make sure that the northbound NETCONF communication is enabled.

If this parameter is set to false, the upper (CFS) NSO is not able to communicate with

the lower (RFS) NSOs.

```
root@a0f0d24a825f:/# cat /etc/ncs/ncs.conf

< … Output Omitted … >

  <netconf-north-bound>
    <enabled>true</enabled>
    <transport>
      <ssh>
        <enabled>true</enabled>
        <ip>0.0.0.0</ip>
        <port>830</port>
        <extra-listen>
          <ip>::</ip>
          <port>830</port>
        </extra-listen>
      </ssh>
      <tcp>
        <enabled>false</enabled>
        <ip>127.0.0.1</ip>
        <port>2023</port>
      </tcp>
    </transport>
  </netconf-north-bound>

< … Output Omitted … >
```

Step 7

Exit the NSO Docker shell.

```
root@a0f0d24a825f:/# exit
logout
student@student-vm:~/nso-lsa/nso-rfs $
```

Step 8

Copy the **l3vpn-rfs** package from **~/packages/l3vpn-rfs** to the **packages** directory

```
student@student-vm:~/nso-lsa/nso-rfs $ cp -r ~/packages/l3vpn-rfs
packages
student@student-vm:~/nso-lsa/nso-rfs $
```

Step 9

Open the YANG service model for this service.

The following output shows how the RFS service model should appear when you open it for the first time. Examine the model. You will later compare it to the CFS model.

```
student@student-vm:~/nso-lsa/nso-rfs$ cat packages/l3vpn-rfs/src/yang/
l3vpn-rfs.yang
module l3vpn-rfs {
  namespace "http://example.com/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 name;
    leaf 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;
          }
      }
  }
}
```

Step 10

Use the **make testenv-build** command to reload the packages. This command builds and reloads the packages for both RFS containers.

Make sure that the packages are successfully reloaded. You will not directly interact with this service, though. It will be invoked by the CFS node.

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

< Output Omitted … >
-- Reloading packages for NSO testenv-nso-rfs-5.3.2-student-nso-emear

>>> 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 l3vpn-rfs
    result true
}
-- Rebuilding for NSO: testenv-nso-rfs-5.3.2-student-nso-am

< Output Omitted … >

-- Reloading packages for NSO testenv-nso-rfs-5.3.2-student-nso-am

>>> 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 l3vpn-rfs
    result true
}
student@student-vm:~/nso-lsa/nso-rfs$
```

Step 11

Enter the NSO CLI of the *am* RFS node and display the devices that are added to NSO.

As the name denotes, these devices are located in the AM (Americas) region. This NSO node manages only the devices located in this region. LSA nodes are not necessarily divided geographically. Different nodes can take care of different customers, vendors, versions, and so on. They can also simply be used to divide the load between several nodes.

```
student@student-vm:~/nso-lsa/nso-rfs$ make testenv-cli NSO=am
docker exec -it testenv-nso-rfs-5.3.2-student-nso-am bash -lc 'ncs_cli
-u admin'

admin connected from 127.0.0.1 using console on a0f0d24a825f
admin@ncs> switch cli
admin@ncs# show devices brief
NAME     ADDRESS  DESCRIPTION  NED ID
-------------------------------------------------
CE11-AM  CE11-AM  -            cisco-ios-cli-6.54
CE12-AM  CE12-AM  -            cisco-ios-cli-6.54
PE11-AM  PE11-AM  -            cisco-iosxr-cli-7.26
admin@ncs# exit
student@student-vm:~/nso-lsa/nso-rfs$
```

Step 12

Display the devices added to the *emear* RFS node.

```
student@student-vm:~/nso-lsa/nso-rfs$ make testenv-cli NSO=emear
docker exec -it testenv-nso-rfs-5.3.2-student-nso-emear bash -lc
'ncs_cli -u admin'

admin connected from 127.0.0.1 using console on 0c8c2159821b
admin@ncs> switch cli
admin@ncs# show devices brief
NAME         ADDRESS     DESCRIPTION  NED ID
-----------------------------------------------------------
CE21-EMEAR   CE21-EMEAR   -            cisco-ios-cli-6.54
CE22-EMEAR   CE22-EMEAR   -            cisco-ios-cli-6.54
PE21-EMEAR   PE21-EMEAR   -            cisco-iosxr-cli-7.26
admin@ncs# exit
student@student-vm:~/nso-lsa/nso-rfs$
```

Activity Verification

You have completed this task when you attain these results:

- You have configured both RFS nodes.
- l3vpn-rfs packages are successfully reloaded.
- You have verified that each RFS node contains its own device.

Task 2: Configure CFS Node

In this task, you will configure the customer-facing node for the LSA architecture. This is the "upper" node that an NSO administrator interacts with and is responsible for deploying service configuration to the resource-facing nodes below. You will also create a dispatch map for the service, so that the CFS nodes correctly interpret to which RFS node to dispatch the configuration.

Activity

Complete these steps:

Step 1

Navigate to the **~/nso-lsa/nso-cfs** directory.

```
student@student-vm:~/nso-lsa/nso-rfs$ cd ../nso-cfs/
student@student-vm:~/nso-lsa/nso-cfs$
```

Step 2

Use the **make testenv-shell** command to enter the Docker container shell.

```
student@student-vm:~/nso-lsa/nso-cfs$ make testenv-shell
docker exec -it testenv-nso-cfs-5.3.2-student-nso bash -l
root@e89d67ae48d2:/#
```

Step 3

Enable the LSA within the **/etc/ncs/ncs.conf** configuration file, save it, and exit the file editor.

```
root@e89d67ae48d2:/# vi /etc/ncs/ncs.conf

< … Output Omitted … >

  <large-scale>
    <lsa>
      <!-- Enable Layered Service Architecture, LSA. This requires
           a separate Cisco Smart License.
      -->
      <enabled>true</enabled>
    </lsa>
  </large-scale>

< … Output Omitted … >
```

Step 4

Reload the *ncs* service with the **ncs –reload** command and exit the container.

This action applies the configuration changes done to the **ncs.conf** file that you just edited. However, be careful; this configuration change only applies to the current NSO Docker environment because the configuration file does not persist outside of it. If the environment is stopped, the configuration changes are lost.

```
root@e89d67ae48d2:/# ncs --reload
root@e89d67ae48d2:/# exit
```

Step 5

Copy the CFS service package to packages directory. The package is located in **~/packages/l3vpn-cfs**.

```
student@student-vm:~/nso-lsa/nso-cfs$ cp -r ~/packages/l3vpn-cfs
packages
```

Step 6

Open and study the YANG model for the l3vpn-cfs service.

Examine the model. Note how it is very similar to the RFS model and how the device parameter is a leaf and not a leafref. The reason is because the network devices are added to RFS nodes and therefore cannot be referenced from the CFS node.

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

```
  prefix l3vpn-cfs;

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

  augment /ncs:services {
    list l3vpn-cfs {
      description "This is an L3VPN cfs service model.";

      key name;
      leaf name {
        tailf:info "Unique L3 VPN cfs service instance 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;
            }

      }
    }
  }
}
```

Step 7

Create a list **dispatch-map**. This list is used to map the lower RFS nodes to the

dispatch criteria of your choice.

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

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

  list dispatch-map {}

  augment /ncs:services {
    list l3vpn-cfs {
      description "This is an L3VPN cfs service model.";

      key name;
      leaf name {
        tailf:info "Unique L3 VPN cfs service instance 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;
            }

      }
    }
```

```
    }
  }
```

Step 8

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.

```
module l3vpn-cfs {
  namespace "http://example.com/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;

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

  augment /ncs:services {
    list l3vpn-cfs {
      description "This is an L3VPN cfs service model.";

      key name;
      leaf name {
        tailf:info "Unique L3 VPN cfs service instance 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;
            }

      }
    }
  }
}
```

Step 9

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

```
module l3vpn-cfs {
  namespace "http://example.com/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";
      }
    }
  }

  augment /ncs:services {
    list l3vpn-cfs {
      description "This is an L3VPN cfs service model.";
```

```
      key name;
      leaf name {
        tailf:info "Unique L3 VPN cfs service instance 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;
            }

      }
    }
  }
}
```

Step 10

Save the file and exit the file editor.

Step 11

Display the XML template for the **l3vpn-cfs** service.

The following outputs show how the file should appear when you open it for the first time. In addition to the normal service parameters, you need to set the correct RFS-NODE to which the service configuration is dispatched.

```
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>{$RFS-NODE}</name>
      <config>
        <l3vpn-rfs xmlns="http://example.com/l3vpn-rfs">
          <name>{string(../name)}</name>
          <link>
            <id>{$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@student-vm:~/nso-lsa/nso-cfs$
```

Step 12

Open the main Python file for the **l3vpn-cfs** service.

The following output shows how the file should appear when you open it for the first time. The code that takes care of the RFS node assignment is missing.

```
student@student-vm:~/nso-lsa/nso-cfs$ vim 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()
            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)

# ---------------------------------------------
```

```
# 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

Add a **get_rfs_node** method that finds an RFS node, which maps to the region, contained in the device name. The dispatch map is accessible directly from the *root* variable.

```
# -*- 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('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 14

Use the method you created in the previous step to assign an RFS node to the RFS-NODE template variable.

```
# -*- 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 15

Save the file and exit the file editor.

Step 16

Navigate back to the RFS node home directory (for example, **~/nso-lsa/nso-rfs**), and use the **make dev-shell** command to create and enter an RFS development container.

```
student@student-vm:~/nso-lsa/nso-cfs$ cd ../nso-rfs
student@student-vm:~/nso-lsa/nso-rfs$ make dev-shell
docker run -it -v $(pwd):/src nso300.gitlab.local/cisco-nso-dev:5.3.2
root@065ce2545a3c:/#
```

Step 17

Enter the **/src/packages** directory. It should contain a bind-mounted **l3vpn-rfs** package.

```
root@065ce2545a3c:/# cd src/packages
root@065ce2545a3c:/src/packages# ls
l3vpn-rfs
root@065ce2545a3c:/src/packages#
```

Step 18

Create a netconf NED package **l3vpn-rfs-ned** for the **l3vpn-rfs** service with the **ncs-make-package** command. No errors should be present and the package should be created in the **/src/packages** folder.

The netconf NED package is required for the CFS node to be able to communicate with RFS nodes.

```
root@065ce2545a3c:/src/packages# ncs-make-package --lsa-netconf-ned
l3vpn-rfs/src/yang l3vpn-rfs-ned
root@065ce2545a3c:/src/packages# ls
l3vpn-rfs  l3vpn-rfs-ned
root@065ce2545a3c:/src/packages#
```

Step 19

Change the file permissions for the **l3vpn-rfs-ned** package because the package will not be used in this container.

```
root@065ce2545a3c:/src/packages# chown -Rv 1000:1000 l3vpn-rfs-ned
changed ownership of 'l3vpn-rfs-ned/python/l3vpn_rfs_ned' from
root:root to 1000:1000
changed ownership of 'l3vpn-rfs-ned/python' from root:root to 1000:1000
changed ownership of 'l3vpn-rfs-ned/src/Makefile' from root:root to
1000:1000
changed ownership of 'l3vpn-rfs-ned/src/yang/l3vpn-rfs.yang' from
```

```
root:root to 1000:1000
changed ownership of 'l3vpn-rfs-ned/src/yang' from root:root to
1000:1000
changed ownership of 'l3vpn-rfs-ned/src/package-meta-data.xml.in' from
root:root to 1000:1000
changed ownership of 'l3vpn-rfs-ned/src/java/src/com/example/
l3vpnrfsned/namespaces' from root:root to 1000:1000
changed ownership of 'l3vpn-rfs-ned/src/java/src/com/example/
l3vpnrfsned' from root:root to 1000:1000
changed ownership of 'l3vpn-rfs-ned/src/java/src/com/example' from
root:root to 1000:1000
changed ownership of 'l3vpn-rfs-ned/src/java/src/com' from root:root to
1000:1000
changed ownership of 'l3vpn-rfs-ned/src/java/src' from root:root to
1000:1000
changed ownership of 'l3vpn-rfs-ned/src/java/build.xml' from root:root
to 1000:1000
changed ownership of 'l3vpn-rfs-ned/src/java' from root:root to
1000:1000
changed ownership of 'l3vpn-rfs-ned/src' from root:root to 1000:1000
changed ownership of 'l3vpn-rfs-ned/netsim/start.sh' from root:root to
1000:1000
changed ownership of 'l3vpn-rfs-ned/netsim/Makefile' from root:root to
1000:1000
changed ownership of 'l3vpn-rfs-ned/netsim/confd.conf.netsim' from
root:root to 1000:1000
changed ownership of 'l3vpn-rfs-ned/netsim' from root:root to 1000:1000
changed ownership of 'l3vpn-rfs-ned' from root:root to 1000:1000
root@065ce2545a3c:/src/packages#
```

Step 20

Exit the development container.

```
root@065ce2545a3c:/src/packages# exit
logout
student@student-vm:~/nso-lsa/nso-rfs$
```

Step 21

Move (do not copy) the **l3vpn-rfs-ned** to the **packages** folder from **nso-cfs** project.

```
student@student-vm:~/nso-lsa/nso-rfs$ mv packages/l3vpn-rfs-ned ../nso-
cfs/packages
student@student-vm:~/nso-lsa/nso-rfs$
```

Step 22

Return to the *nso-cfs* folder and reload the packages. Make sure that there are no errors present and both **l3vpn-cfs** and **l3vpn-rfs-ned** packages are successfully reloaded.

```
student@student-vm:~/nso-lsa/nso-rfs$ cd ../nso-cfs
student@student-vm:~/nso-lsa/nso-cfs$ make testenv-build
for NSO in $(docker ps --format '{{.Names}}' --filter label=testenv-
nso-cfs-5.3.2-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=nso300.gitlab.local/nso-cfs/
package:5.3.2-student nso300.gitlab.local/cisco-nso-dev:5.3.2 /src/nid/
testenv-build; \
done
-- Rebuilding for NSO: testenv-nso-cfs-5.3.2-student-nso

< … Output Omitted … >

-- Reloading packages for NSO testenv-nso-cfs-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 l3vpn-cfs
    result true
}
reload-result {
    package l3vpn-rfs-ned
    result true
}
student@student-vm:~/nso-lsa/nso-cfs$
```

Step 23

Enter the NSO CLI and switch the CLI mode.

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

admin connected from 127.0.0.1 using console on e89d67ae48d2
admin@ncs> switch cli
admin@ncs#
```

Step 24

Add both of the RFS nodes as devices. You can use the *lsa* authentication group for remote authentication.

```
admin@ncs# config
admin@ncs(config-device-NSO-RFS-AM)# state admin-state unlocked
admin@ncs(config-device-NSO-RFS-AM)# top
admin@ncs(config)# devices device NSO-RFS-EMEAR address NSO-RFS-EMEAR
port 830 authgroup lsa device-type netconf ned-id lsa-netconf
```

```
admin@ncs(config-device-NSO-RFS-EMEAR)# state admin-state unlocked
admin@ncs(config-device-NSO-RFS-EMEAR)# top
admin@ncs(config)# commit
Commit complete.
admin@ncs(config)#
```

Step 25

Exit the configuration mode, fetch the device SSH keys, and perform a **sync-from devices**. Make sure that the synchronization is successful.

```
admin@ncs(config)# exit
admin@ncs# devices fetch-ssh-host-keys
fetch-result {
    device NSO-RFS-AM
    result updated
    fingerprint {
        algorithm ssh-rsa
        value 5b:66:c3:16:7a:11:db:16:8e:49:5f:e8:44:20:45:84
    }
}
fetch-result {
    device NSO-RFS-EMEAR
    result updated
    fingerprint {
        algorithm ssh-rsa
        value 80:85:18:6f:e4:cd:6c:f7:17:91:9c:c5:87:19:d7:2e
    }
}
admin@ncs# devices sync-from
sync-result {
    device NSO-RFS-AM
    result true
}
sync-result {
    device NSO-RFS-EMEAR
    result true
}
admin@ncs#
```

Activity Verification

You have completed this task when you attain these results:

- You have configured both RFS nodes.
- l3vpn-rfs packages are successfully reloaded.
- You have verified that each RFS node contains its own device.

Task 3: Deploy an LSA Service Instance

In this task, you will create a dispatch map and deploy an LSA service instance, using the CFS and RFS nodes you configured in the previous tasks.

Activity

Complete these steps:

Step 1

Create a dispatch map entry for each of the regions.

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# dispatch-map AM NSO-RFS-AM
admin@ncs(config)# dispatch-map EMEAR NSO-RFS-EMEAR
admin@ncs(config)# commit
Commit complete.
admin@ncs(config)#
```

Step 2

Configure an L3VPN service instance. Do a dry run of the commit and observe on which LSA RFS node the request has been dispatched. Then commit the transaction.

```
admin@ncs(config)# services l3vpn-cfs vpn10 link customer1 device CE11-
AM interface 0/1 ip-address 10.100.0.1 mask 255.255.255.255
admin@ncs(config-link-customer1)# commit dry-run
cli {
    local-node {
        data  devices {
                  device NSO-RFS-AM {
                      config {
             +            l3vpn-rfs vpn10 {
             +                link customer1 {
             +                    device CE11-AM;
             +                    interface 0/1;
             +                    ip-address 10.100.0.1;
             +                    mask 255.255.255.255;
             +                }
             +            }
                      }
                  }
              }
              services {
             +    l3vpn-cfs vpn10 {
             +        link customer1 {
             +            device CE11-AM;
             +            interface 0/1;
             +            ip-address 10.100.0.1;
             +            mask 255.255.255.255;
             +        }
             +    }
              }
    }
    lsa-node {
        name NSO-RFS-AM
        data  devices {
                  device CE11-AM {
```

```
                        config {
                            vrf {
                +               definition vpn10 {
                +                   rd 65000:30292;
                +                   route-target {
                +                       export 65000:30292;
                +                       import 65000:30292;
                +                   }
                +               }
                            }
                            interface {
                                GigabitEthernet 0/1 {
                                    vrf {
                +                       forwarding vpn10;
                                    }
                                    ip {
                                        no-address {
                -                           address false;
                                        }
                                        address {
                                            primary {
                +                               address 10.100.0.1;
                +                               mask 255.255.255.255;
                                            }
                                        }
                                    }
                                }
                            }
                            router {
                +               bgp 65000 {
                +                   address-family {
                +                       with-vrf {
                +                           ipv4 unicast {
                +                               vrf vpn10 {
                +                                   neighbor 10.100.0.1 {
                +                                       remote-as 65001;
                +                                   }
                +                               }
                +                           }
                +                       }
                +                   }
                            }
                        }
                    }
                }
                +l3vpn-rfs vpn10 {
                +    link customer1 {
                +        device CE11-AM;
                +        interface 0/1;
                +        ip-address 10.100.0.1;
                +        mask 255.255.255.255;
                +    }
                +}
            }
        }
admin@ncs(config-link-customer1)# top
admin@ncs(config)# commit
```

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

Step 3

Configure an extra L3VPN service instance. This time, use an EMEAR device and verify the commit with a dry run.

```
admin@ncs(config)# services l3vpn-cfs vpn10 link customer2 device CE21-
EMEAR interface 0/1 ip-address 10.200.0.1 mask 255.255.255.255
admin@ncs(config-link-customer2)# commit dry-run
cli {
    local-node {
        data  devices {
                  device NSO-RFS-EMEAR {
                      config {
              +           l3vpn-rfs vpn10 {
              +               link customer2 {
              +                   device CE21-EMEAR;
              +                   interface 0/1;
              +                   ip-address 10.200.0.1;
              +                   mask 255.255.255.255;
              +               }
              +           }
                      }
                  }
               }
               services {
                      l3vpn-cfs vpn10 {
              +           link customer2 {
              +               device CE21-EMEAR;
              +               interface 0/1;
              +               ip-address 10.200.0.1;
              +               mask 255.255.255.255;
              +           }
                      }
               }
    }
    lsa-node {
        name NSO-RFS-EMEAR
        data  devices {
                  device CE21-EMEAR {
                      config {
                          vrf {
              +               definition vpn10 {
              +                   rd 65000:56450;
              +                   route-target {
              +                       export 65000:56450;
              +                       import 65000:56450;
              +                   }
              +               }
                          }
                          interface {
                              GigabitEthernet 0/1 {
                                  vrf {
              +                       forwarding vpn10;
```

```
                                }
                                ip {
                                    no-address {
                -                       address false;
                                    }
                                    address {
                                        primary {
                +                           address 10.200.0.1;
                +                           mask 255.255.255.255;
                                        }
                                    }
                                }
                            }
                        }
                        router {
                +           bgp 65000 {
                +               address-family {
                +                   with-vrf {
                +                       ipv4 unicast {
                +                           vrf vpn10 {
                +                               neighbor 10.200.0.1 {
                +                                   remote-as 65001;
                +                               }
                +                           }
                +                       }
                +                   }
                +               }
                            }
                        }
                    }
                }
                +l3vpn-rfs vpn10 {
                +    link customer2 {
                +        device CE21-EMEAR;
                +        interface 0/1;
                +        ip-address 10.200.0.1;
                +        mask 255.255.255.255;
                +    }
                +}
    }
}
admin@ncs(config-link-customer2)# top
admin@ncs(config)# commit
Commit complete.
admin@ncs(config)#
```

Step 4

Exit the CFS NSO CLI and connect to CLI of the RFS-AM node.

```
admin@ncs(config)# exit
admin@ncs# exit
student@student-vm:~/nso-lsa/nso-cfs$ cd ../nso-rfs
student@student-vm:~/nso-lsa/nso-rfs$ make testenv-cli NSO=am
docker exec -it testenv-nso-rfs-am-5.3.2-student-nso bash -lc 'ncs_cli
```

```
-u admin'

admin connected from 127.0.0.1 using console on a0f0d24a825f
admin@ncs>
```

Step 5

Verify that the configuration has been applied to the device through an RFC node.

```
admin@ncs> switch cli
admin@ncs# show running-config devices device CE11-AM config router bgp
devices device CE11-AM
 config
  router bgp 65000
   address-family ipv4 unicast vrf vpn10
    neighbor 10.100.0.1 remote-as 65001
    exit-address-family
   !
  !
 !
!
admin@ncs#
```

Step 6

Exit the NSO CLI and enter the container shell.

```
admin@ncs# exit
student@student-vm:~/nso-lsa/nso-rfs$ make testenv-shell NSO=am
docker exec -it testenv-nso-rfs-am-5.3.2-student-nso bash -l
root@a0f0d24a825f:/#
```

Step 7

Use SSH to connect to CE11-AM device.

```
root@a0f0d24a825f:/# ssh admin@CE11-AM
The authenticity of host 'ce11-am (172.17.0.5)' can't be established.
RSA key fingerprint is
SHA256:zhkMRdv3HHyrS2jE2vxBW0CSAC0r45Kznjrjtt3tjD8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ce11-am,172.17.0.5' (RSA) to the list of
known hosts.
admin@ce11-am's password:

admin connected from 172.17.0.3 using ssh on fe3217b73437
dev>
```

Step 8

Verify that the configuration is present on the device.

```
dev> enable
dev# show running-config router bgp
router bgp 65000
 address-family ipv4 unicast vrf vpn10
  neighbor 10.100.0.1 remote-as 65001
  exit-address-family
 !
!
dev#
```

Activity Verification

You have completed this task when you attain these results:

- You have deployed and verified several service instances using LSA.