Perform an L3VPN Service Upgrade

Introduction

In this activity, you will create an upgrade procedure for an L3VPN service. In real life, network services tend to change over time. NSO allows you to upgrade services that are currently deployed. You will simulate a change to the service, by adding a new parameter, provided by the network engineers.

After completing this activity, you will be able to:

- Write an upgrade procedure in Python.

Job Aids

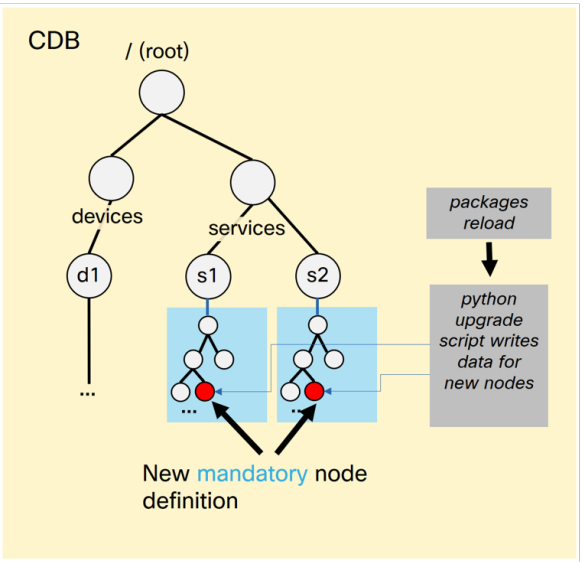
The following job aid is available to help you complete the lab activities:

- This Lab Guide

The following table contains passwords that you might need.

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

The figure below provides a visual aid for this activity.



Required Resources

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

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

Command Syntax Reference

This lab guide uses the following conventions for command syntax:

Formatting	Description and Examples
show running config	Commands in steps use this formatting.
Example	Type **show running config**
Example	Use the **name** command.
`show running config`	Commands in CLI outputs and configurations use this formatting.
highlight	CLI output that is important is highlighted.
Example	`student@student-vm:~$ ncs --version` `5.3.2`

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

Command List

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

Linux Shell:

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

NSO CLI:

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

Makefile commands for Docker environment:

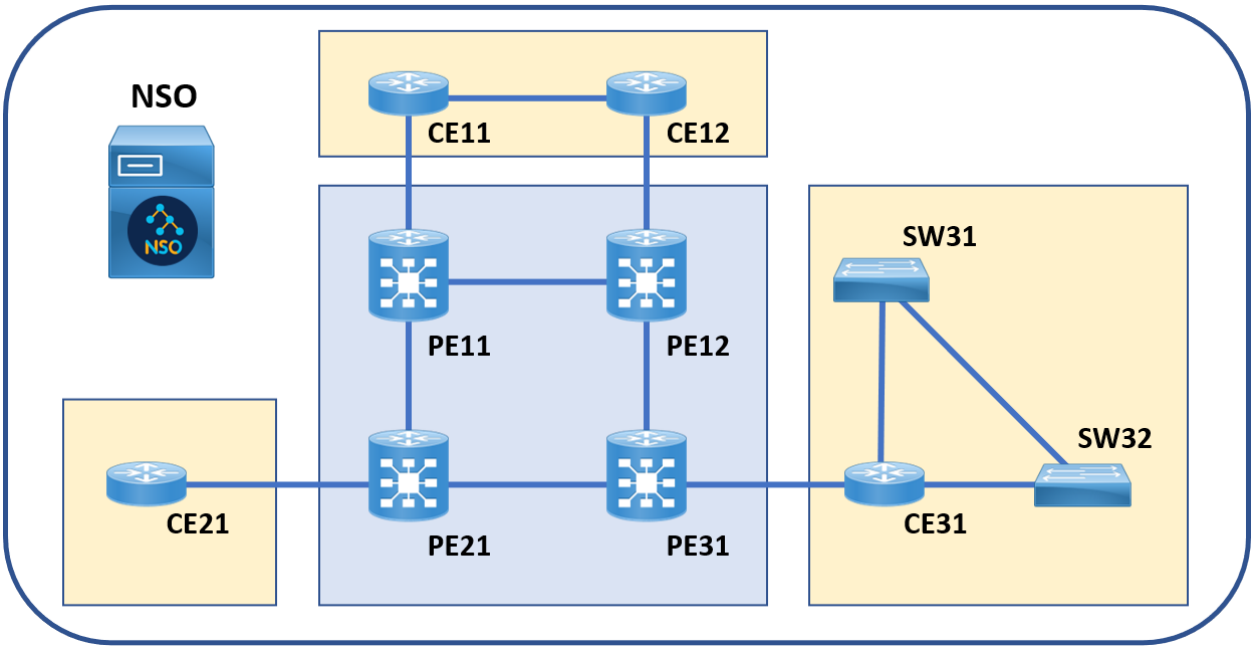
Command	Comment
make build	Builds the main NSO Docker image.
make testenv-start	Starts the NSO Docker environment.
make testenv-stop	Stops the NSO Docker environment.
make testenv-build	Recompiles and reloads the NSO packages.
make testenv-cli	Enters the NSO CLI of the NSO Docker container.
make testenv-shell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

Lab Topology Information

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

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

Topology



Task 1: Update the YANG Service Model

In this task, you will update the YANG model for l3mplsvpn service by adding a description parameter to the l3mplsvpn service.

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

Activity

Complete these steps:

Step 1

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

Step 2

Open the terminal window using the Terminal icon on the taskbar.

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

Step 3

Go to the nso300 folder.

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

Step 4

Open the service YANG model using vim.

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


You can also use an IDE tool or text editor of your choice.

This is how the YANG module should initially appear:

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

  import ietf-inet-types {
    prefix inet;
  }

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

  description "Service for L3 MPLS VPN provisioning.";

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

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

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

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

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

    ncs:component-type "ncs:self" {
      ncs:state "ncs:init";
      ncs:state "ncs:ready";
    }
    ncs:component-type "l3mplsvpn:l3mplsvpn" {
      ncs:state "ncs:init";
```

```
      ncs:state "l3mplsvpn:id-allocated" {
        ncs:create {
          ncs:nano-callback;
        }
      }

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

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

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

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

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

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

      leaf link-id {
        tailf:info "Link ID (1 to 65535)";
        type uint32 {
```

```
          range "1..65535" {
            error-message "Link ID is out of range. Should be between 1
and 65535.";
          }
        }
      }

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

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

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

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

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

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

```
        }
      }
    }
  }
```

Step 5

Add a mandatory *description* leaf, since there is a new service requirement by network engineers to include a description with the L3VPN service.

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

  import ietf-inet-types {
    prefix inet;
  }

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

  description "Service for L3 MPLS VPN provisioning.";

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

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

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

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

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

    ncs:component-type "ncs:self" {
      ncs:state "ncs:init";
      ncs:state "ncs:ready";
    }
    ncs:component-type "l3mplsvpn:l3mplsvpn" {
      ncs:state "ncs:init";
      ncs:state "l3mplsvpn:id-allocated" {
        ncs:create {
          ncs:nano-callback;
```

```
        }
      }

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

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

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

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

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

    leaf description {
      mandatory true;
      type string;
    }

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

      leaf link-id {
```

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

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

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

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

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

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

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

```
link.";
          }
        }
      }
    }
}
```

Step 6

Save the file when finished and exit the file editor.

Step 7

Compile the package. Use the **make testenv-build** command, which recompiles and reloads/redeploys the packages, used by the NSO Docker containers. Since some l3mplsvpn service instance exists and the configuration does not match the YANG model (the mandatory description parameter is not set), the packages reload command is expected to fail.

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

< … Output Omitted … >

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

>>> System upgrade is starting.
>>> Sessions in configure mode must exit to operational mode.
>>> No configuration changes can be performed until upgrade has
completed.
>>> System upgrade has been cancelled.
Error: /l3mplsvpn[name='vpn3']/description in package l3mplsvpn is not
configured
< … output omitted … >
```

Activity Verification

You have completed this task when you attain these results:

- You have successfully updated the YANG service model.

Task 2: Write an Upgrade Procedure in Python

In this task, you will write an upgrade procedure in Python, which will be used to upgrade the L3VPN service package when reloading the packages.

Activity

Complete these steps:

Step 1

Create a Python file for the upgrade procedure.

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

Step 2

Create the upgrade class skeleton.

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

class Upgrade(ncs.upgrade.Upgrade):
    """An upgrade 'class' that will be instantiated by NSO.
    """

    """ cdbsock is a CDB session attached to the pre-upgrade "running",
     trans is a MAAPI transaction attached to the upgrade transaction
    """
    def upgrade(self, cdbsock, trans):
        """The upgrade 'method' that will be called by NSO.
        """
```


Comments in the code should always explain hard to understand concepts.

Step 3

Start a *session* towards the running datastore from the Upgrade class (which is instantiated before the packages are reloaded, for example, before the description leaf is added), to get the data existing in the old schema and read the number of *l3mplsvpn* service instances.

This number will be used to loop over existing l3mplsvpn instances.

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

class Upgrade(ncs.upgrade.Upgrade):
    """An upgrade 'class' that will be instantiated by NSO.
    """

    """ cdbsock is a CDB session attached to the pre-upgrade "running",
     trans is a MAAPI transaction attached to the upgrade transaction
```

```
    """
    def upgrade(self, cdbsock, trans):
        """The upgrade 'method' that will be called by NSO.
        """
        _ncs.cdb.start_session2(cdbsock, ncs.cdb.RUNNING,
ncs.cdb.LOCK_SESSION | ncs.cdb.LOCK_WAIT)
        num = _ncs.cdb.num_instances(cdbsock, "/l3mplsvpn")
```


More details about specific classes and methods can be found in NSO Developers documentation.

Step 4

Loop through all the **l3mplsvpn** service instances and save the **name** and the **customer** leaves to variables.

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

class Upgrade(ncs.upgrade.Upgrade):
    """An upgrade 'class' that will be instantiated by NSO.
    """

    """ cdbsock is a CDB session attached to the pre-upgrade "running",
     trans is a MAAPI transaction attached to the upgrade transaction
    """
    def upgrade(self, cdbsock, trans):
        """The upgrade 'method' that will be called by NSO.
        """
        _ncs.cdb.start_session2(cdbsock, ncs.cdb.RUNNING,
ncs.cdb.LOCK_SESSION | ncs.cdb.LOCK_WAIT)
        num = _ncs.cdb.num_instances(cdbsock, "/l3mplsvpn")

        for i in range(0, num):
            name = str(_ncs.cdb.get(cdbsock, f"/l3mplsvpn[{i}]/name"))
            customer = str(_ncs.cdb.get(cdbsock, f"/l3mplsvpn[{i}]/
customer"))
```

Step 5

Create a **description** using the customer value and set the new mandatory leaf **description** using the service **name** value. Add a comment explaining the creation of the leaf. The **upgrade** function should return **True**.

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

class Upgrade(ncs.upgrade.Upgrade):
    """An upgrade 'class' that will be instantiated by NSO.
    """
```

```
    """ cdbsock is a CDB session attached to the pre-upgrade "running",
     trans is a MAAPI transaction attached to the upgrade transaction
    """
    def upgrade(self, cdbsock, trans):
        """The upgrade 'method' that will be called by NSO.
        """
        _ncs.cdb.start_session2(cdbsock, ncs.cdb.RUNNING,
ncs.cdb.LOCK_SESSION | ncs.cdb.LOCK_WAIT)
        num = _ncs.cdb.num_instances(cdbsock, "/l3mplsvpn")

        for i in range(0, num):
            name = str(_ncs.cdb.get(cdbsock, f"/l3mplsvpn[{i}]/name"))
            customer = str(_ncs.cdb.get(cdbsock, f"/l3mplsvpn[{i}]/
customer"))
  # create a mandatory leaf 'description'
            description = "VPN for " + customer
            trans.set_elem(description, f"/l3mplsvpn{{{name}}}/
description")
        return True
```

Step 6

Save the file when finished.

Step 7

Open **package-meta-data.xml** configuration file.

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

Step 8

This is how the file should initially appear:

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

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

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

```
</ncs-package>
```

Step 9

Modify the **package-meta-data.xml** file to include the **upgrade** component and change the **package-version** to **1.3**.

```
<ncs-package xmlns="http://tail-f.com/ns/ncs-packages">
  <name>l3mplsvpn</name>
  <package-version>1.3</package-version>
  <description>L3 MPLS VPN Python and Template Service</description>
  <ncs-min-version>5.3</ncs-min-version>

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

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

  <component>
    <name>Upgrade</name>
    <upgrade>
      <python-class-name>l3mplsvpn.upgrade.Upgrade</python-class-name>
    </upgrade>
  </component>

</ncs-package>
```

Step 10

Save the file when finished and exit the file editor.

Activity Verification

You have completed this task when you attain these results:

- You have successfully created and added an upgrade procedure.

Task 3: Upgrade the Service Package

In this task, you will issue the service upgrade procedure. The upgrade component will be triggered by the packages reload command.

Activity

Complete these steps:

Step 1

Compile the package. Use the **make testenv-clean-build** command, which always recompiles and reloads the packages, used by the NSO Docker containers. Make sure that no errors are present.

```
student@student-vm:~/nso300$ make testenv-clean-build
for NSO in $(docker ps --format '{{.Names}}' --filter
label=com.cisco.nso.testenv.name=testenv-nso300-5.3.2-student --filter
label=com.cisco.nso.testenv.type=nso); do \
        echo "-- Cleaning NSO: ${NSO}"; \
        docker run -it --rm -v /home/student/nso300:/src --volumes-from
${NSO} nso300.gitlab.local/cisco-nso-dev:5.3.2 bash -lc 'rsync -aEim --
delete /src/packages/. /var/opt/ncs/packages/ >/dev/null'; \
        echo "-- Copying in pristine included packages for NSO:
${NSO}"; \
        docker run -it --rm --volumes-from ${NSO} nso300.gitlab.local/
nso300/build:5.3.2-student cp -a /includes/. /var/opt/ncs/packages/; \
done
-- Cleaning NSO: testenv-nso300-5.3.2-student-nso
-- Copying in pristine included packages for NSO: testenv-nso300-5.3.2-
student-nso
-- Done cleaning, rebuilding with forced package reload...
make testenv-build PACKAGE_RELOAD="true"
make[1]: Entering directory '/home/student/nso300'
for NSO in $(docker ps --format '{{.Names}}' --filter
label=com.cisco.nso.testenv.name=testenv-nso300-5.3.2-student --filter
label=com.cisco.nso.testenv.type=nso); do \
        echo "-- Rebuilding for NSO: ${NSO}"; \
        docker run -it --rm -v /home/student/nso300:/src --volumes-from
${NSO} --network=container:${NSO} -e NSO=${NSO} -e PACKAGE_RELOAD=true
-e SKIP_LINT= -e PKG_FILE=nso300.gitlab.local/nso300/package:5.3.2-
student nso300.gitlab.local/cisco-nso-dev:5.3.2 /src/nid/testenv-build;
\
done
-- Rebuilding for NSO: testenv-nso300-5.3.2-student-nso
(^package-meta-data.xml$|\.cli$|\.yang$)
make: Entering directory '/var/opt/ncs/packages/l3mplsvpn/src'
mkdir -p ../load-dir
mkdir -p java/src//
/opt/ncs/ncs-5.3.2/bin/ncsc  `ls l3mplsvpn-ann.yang  > /dev/null 2>&1
&& echo "-a l3mplsvpn-ann.yang"` \
              -c -o ../load-dir/l3mplsvpn.fxs yang/l3mplsvpn.yang
make: Leaving directory '/var/opt/ncs/packages/l3mplsvpn/src'
make: Entering directory '/src/packages/l3mplsvpn'
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' $(ls package-meta-data.xml src/
package-meta-data.xml.in 2>/dev/null | head -n 1)); \
        export PKG_VERSION=$(xmlstarlet sel -N x=http://tail-f.com/ns/
ncs-packages -t -v '/x:ncs-package/x:package-version' $(ls package-
meta-data.xml src/package-meta-data.xml.in 2>/dev/null | head -n 1)); \
        eval "cat <<< \"$(</src/nid/build-meta-data.xml)\"" > /var/opt/
ncs/packages/l3mplsvpn//build-meta-data.xml; fi
make: Leaving directory '/src/packages/l3mplsvpn'
-- Reloading packages for NSO testenv-nso300-5.3.2-student-nso

>>> System upgrade is starting.
```

```
>>> Sessions in configure mode must exit to operational mode.
>>> No configuration changes can be performed until upgrade has
completed.
>>> System upgrade has completed successfully.
reload-result {
    package cisco-asa-cli-6.10
    result true
}
reload-result {
    package cisco-ios-cli-6.54
    result true
}
reload-result {
    package cisco-iosxr-cli-7.26
    result true
}
reload-result {
    package cisco-nx-cli-5.15
    result true
}
reload-result {
    package l3mplsvpn
    result true
}
reload-result {
    package resource-manager
    result true
}
make[1]: Leaving directory '/home/student/nso300'
student@student-vm:~/nso300$
```

Step 2

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

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

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

Step 3

Switch the CLI type.

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

Step 4

Verify that the description of the L3VPN service instance exists.

```
admin@ncs# show running-config l3mplsvpn
l3mplsvpn vpn123
 customer ACME
 description VPN for ACME
 link 1
  device    PE11
  interface 0/1
 !
 link 2
  device    PE11
  interface 0/2
 !
!
```

Activity Verification

You have completed this task when you attain these results:

- You have successfully deployed and verified a service instance using the upgrade component.