



Network Automation Journey with Cisco Network Services Orchestrator (NSO)

LABOPS-1507

Speakers:

Spyriadis Spyros – Software Consulting Engineer

Overview

Network operators and service providers today are struggling to control the difference between the growth of their operating costs and their revenue. Introduction and deployment of new services is much slower compared to service demand and availability on the market. It is because of inadequate provisioning processes where services are either configured manually or hard coded inside the Operations Support Systems (OSS). Cisco Network Service Orchestrator (NSO) is the answer to the above challenge. NSO architecture decouples network services from specific components, while automatically configuring the network according to the service specifications. NSO enabled by NETCONF and YANG models, enables operators to dynamically adopt the service configuration solution according to changes in the offered service portfolio.

This session is intended to familiarize the novice NSO user with the architecture and capabilities of the platform, touching standards utilized by NSO, such as NETCONF and YANG. The session will further discuss NSO components, service and device abstraction, integration with northbound systems via Application Programming Interfaces (APIs), communication procedure with southbound devices via Network Element Drivers (NEDs), configuration compliance, and configuration data collection.

Learning Objectives

Upon completion of this lab, you will be able to:

- Automate creation and deployment of network services through Cisco Network Services Orchestrator (NSO).
- Expand current services with new parameters and device types
- Use Yang for Data/Service modelling
- Generate configuration templates in XML
- Understand Mapping Logic between data models and configuration templates
- Use NETCONF and RESTCONF to interact with NSO.

The lab has 3 main exercises:

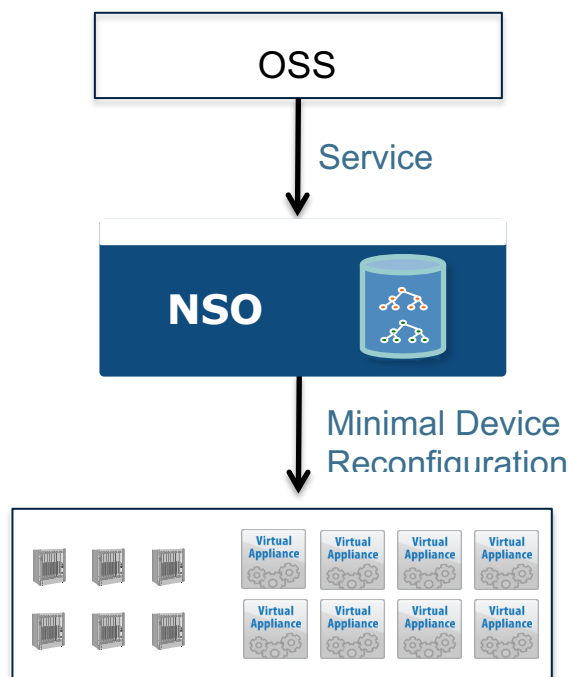
- **Exercise 1:** Basic Device Operation. It will help you get familiar with how NSO deals with devices and manages the synchronization of the configuration
- **Exercise 2:** Simple Loopback Service. You will create a very simple service to configure Loopback interfaces, but supporting 3 type of devices: IOS, IOS XR and JUNOS
- **Exercise 3 (Optional):** L3VPN. This service will take you more in deep in YANG, PYTHON, Templates, GUI, Resconf API and other concepts
- **Appendix B:** Install NSO. Will provide you instructions in case you want to install NSO from zero

Disclaimer

This training document is to familiarize with Cisco NSO for Automating your network. Although the lab design and configuration examples could be used as a reference, it's not a real design, thus not all recommended features are used, or enabled optimally. For the design related questions please contact your representative at Cisco, or a Cisco partner.

NSO Overview

Cisco® Network Services Orchestrator (NSO) enabled by Tail-f® is an industry-leading orchestration platform for hybrid networks. It provides comprehensive lifecycle service automation to enable you to design and deliver high-quality services faster and more easily.



The network is a foundation for revenue generation. Therefore, service providers must implement network orchestration to simplify the entire lifecycle management for services. For today's virtualized networks, this means transparent orchestration that spans multiple domains in your network and includes network functions virtualization (NFV) and software-defined networking (SDN) as well as your traditional physical network and all its components

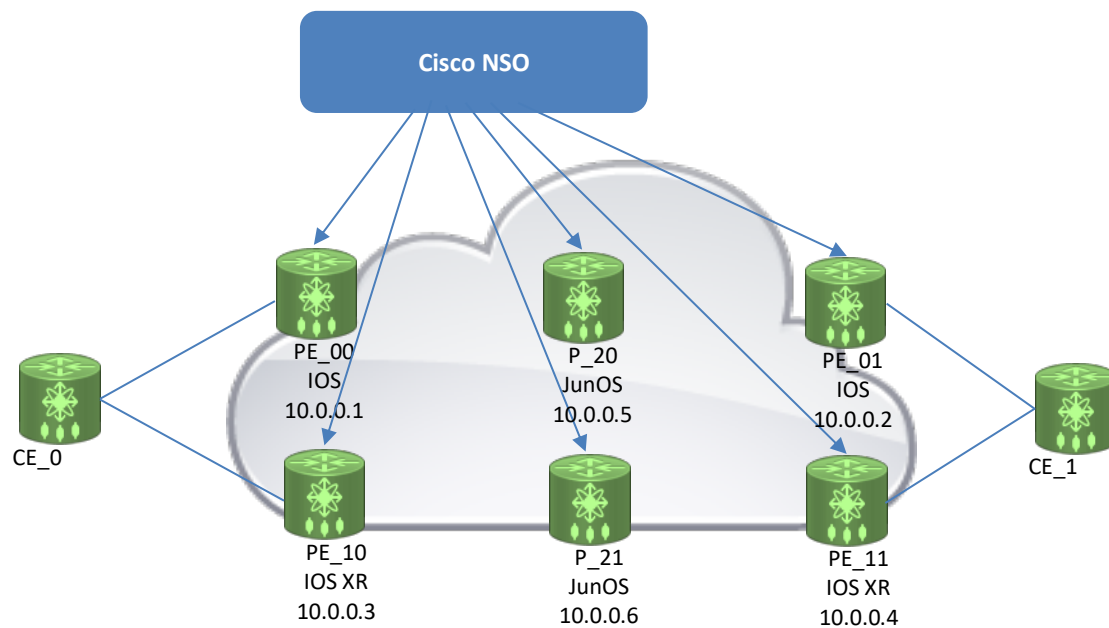
NSO is a model driven (YANG) platform for automating your network orchestration. It supports multi-vendor networks through a rich variety of Network Element Drivers (NEDs).

We support the process of validating, implementing and abstracting your network config and network services, providing support for the entire transformation into intent based networking.

Lab Introduction

Topology

NSO is installed and running six simulated devices taking the roles of Provider Edge and Provider routers in the network. Two of them run Cisco IOS, other two Cisco IOS XR software and the last two Juniper JunOS.

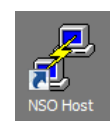
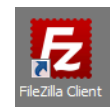
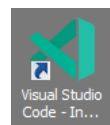


Lab environment

The lab runs inside dCloud in a Windows machine. NSO it's installed in a Linux host and can be reached through SSH, GUI and some APIs (RESTCONF will be used) from the windows machine.

Ways of development possible for this lab.

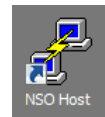
1. (preferred) Visual Studio Code – Insiders. You can find a shortcut in the desktop to this application. When you start it, you will be connected to NSO server and you will be able to view and edit files from your local Windows. A Terminal is available as well. See Appendix A for more information.
2. Edit files locally using Notepad++ or Visual Studio Code and then upload them to NSO server through SFTP. Filezilla is installed and prepared to connect to NSO, with a shortcut in the Desktop.
3. Connect to NSO host through putty and edit the files directly there by 'vim'. Desktop shortcut available.



Lab Introduction and Verification

The NSO version 5.7.2 is already installed and the required Network Element Drivers (NEDS) are loaded.

Desktop shortcut 'NSO Host' allows you to connect to the Linux host where NSO is running as user 'cisco'.



1. Double click on it to access (or open terminal in VSCode – Insiders).
2. Move to the running directory for starting NSO and simulated Netsim devices

```
cisco@nso-572i:~$ cd /home/cisco/nso572/ncs-run
```

3. Start NSO:

```
cisco@nso-572i:~/ncs-run$ ncs
```

NOTE: It can take 2 minutes. Notify the proctor if it takes more than 5min.

4. Start netsim simulated devices (see more in Appendix C):

```
cisco@ubuntu:~/nso572/ncs-run$ ncs-netsim start
DEVICE PE_00 OK STARTED
DEVICE PE_01 OK STARTED
DEVICE PE_10 OK STARTED
DEVICE PE_11 OK STARTED
DEVICE P_20 OK STARTED
DEVICE P_21 OK STARTED
cisco@ubuntu:~/nso572/ncs-run$
```

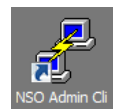
5. Connect directly to the Netsim device PE_00 CLI:

```
cisco@ubuntu:~/nso572/ncs-run$ ncs-netsim cli-i PE_00
admin connected from 198.18.133.252 using ssh on ubuntu
PE_00>
```

6. Configure IP address on interface Loopback 0. Later we will verify that NSO gets this configuration when syncing from the network device:

```
PE_00> enable
PE_00# configure
Enter configuration commands, one per line. End with CNTL/Z.
PE_00(config)# interface Loopback 0
PE_00(config-if)# ip address 10.0.0.1 255.255.255.255
PE_00(config-if)# exit
PE_00(config)# exit
PE_00# exit
```

Desktop shortcut 'NSO Admin cli' allows you to connect to NSO cli mode as user 'admin'.



7. Access to NSO CLI. There are different ways to connect to NSO CLI

- a. Double click on the icon shown above to access
- b. Run from the previous terminal 'ncs_cli -u admin' (default password for user admin is admin)

```
cisco@ubuntu:~/nso572/ncs-run$ ncs_cli -u admin
admin connected from 127.0.0.1 using console on ubuntu
admin@ncs>
```

- c. When enabled in ncs.conf file, NSO allows direct access to CLI through SSH connection.
Example:

```
cisco@ubuntu:~/nso572/ncs-run$ ssh -l admin -p 2024 localhost
The authenticity of host '[localhost]:2024 ([127.0.0.1]:2024)' can't
be established.
RSA key fingerprint is
SHA256:nzmXDxz2gP7F8r5OYnz2d6OI20uwoHTRw+sstvftHI8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:2024' (RSA) to the list of
known hosts.
admin@localhost's password:

admin connected from 127.0.0.1 using ssh on ubuntu
admin@ncs>
```

8. Cisco NSO allows 2 types of CLI to interact with it. During this workbook we will use Cisco based CLI. To use Cisco based CLI you can do it in 2 ways:

- a. You can run the command above and then 'switch cli'.

```
cisco@ubuntu:~/nso572/ncs-run$ ncs_cli -u admin

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

- b. Run the above command with the additional option -C for cisco

```
cisco@ubuntu:~/nso572/ncs-run$ ncs_cli -Cu admin

admin connected from 127.0.0.1 using console on ubuntu
admin@ncs#
```

9. Let's verify the 4 netsim devices are loaded into NSO

```
admin@ncs# show devices list
```

NAME	ADDRESS	DESCRIPTION	NED ID	ADMIN STATE
PE_00	127.0.0.1	-	cisco-ios	unlocked
PE_01	127.0.0.1	-	cisco-ios	unlocked
PE_10	127.0.0.1	-	cisco-ios-xr	unlocked
PE_11	127.0.0.1	-	cisco-ios-xr	unlocked
P_20	127.0.0.1	-	netconf	unlocked
P_21	127.0.0.1	-	netconf	unlocked

```
admin@ncs#
```

10. Now we will make sure NSO Configuration database (CDB) is synchronized with the devices. This action retrieves all configuration from the devices and stores it in NSO internal Configuration Database (CDB).

```
admin@ncs# devices sync-from
sync-result {
  device PE_00
  result true
}
sync-result {
  device PE_01
  result true
}
sync-result {
  device PE_10
  result true
}
sync-result {
  device PE_11
  result true
}
sync-result {
  device P_20
```

```

    result true
}
sync-result {
    device P_21
    result true
}
admin@ncs#

```

11. Verify that the configured interface Loopback in PE_00 is present in NSO CDB database. Note that to display device running configuration the syntax is similar to the device native CLI specifying first the path to the device we want to get the configuration for.

```

admin@ncs# show running-config devices device PE_00 config interface
Loopback 0
devices device PE_00
config
interface Loopback0
ip address 10.0.0.1 255.255.255.255
no shutdown
exit
!
!
admin@ncs#

```

12. Verify the required packages are loaded

```

admin@ncs# show packages package package-version

```

NAME	PACKAGE VERSION
cisco-ios-cli-6.80	6.80.1
cisco-iosxr-cli-7.39	7.39.2
juniper-junos-nc-4.7	4.7.1
l3vpn	1.0

```

admin@ncs# show packages package oper-status

```

FILE		PROGRAM				PACKAGE				PACKAGE META
LOAD NAME	ERROR INFO	UP	CODE ERROR	JAVA UNINITIALIZED	BAD NCS VERSION	PACKAGE NAME	PACKAGE VERSION	CIRCULAR DEPENDENCY	DATA ERROR	
cisco-ios-cli-6.80			X	-	-	-	-	-	-	
cisco-iosxr-cli-7.39			X	-	-	-	-	-	-	
juniper-junos-nc-4.7			X	-	-	-	-	-	-	
l3vpn		X	-	-	-	-	-	-	-	

```

admin@ncs#

```

NOTE: Ignore for now 'l3vpn' package, it will be used in Exercise 3.

Exercise 1: Basic device Operations

Let's start exploring how NSO synchronizes the device configuration with its internal database (CDB).

First, we will configure the device directly and see how NSO database becomes out-of-sync

1. Connect directly to the Netsim device PE_00 CLI:

```
cisco@ubuntu:~/nso572/ncs-run$ ncs-netsim cli-i PE_00

admin connected from 198.18.133.252 using ssh on ubuntu
PE_00>
```

2. Change the IP address on interface Loopback 0:

```
PE_00> enable
PE_00# configure
Enter configuration commands, one per line. End with CNTL/Z.
PE_00(config)# interface Loopback 0
PE_00(config-if)# ip address 10.0.0.2 255.255.255.255
PE_00(config-if)# exit
PE_00(config)# exit
PE_00# exit
```

3. Connect to NSO CLI and do a 'check-sync':

```
admin@ncs# devices device PE_00 check-sync
result out-of-sync
info got: 7f213b04d35d1c98f136fd3176c47747 expected:
bfe5ec360065fa2b9555e3cbc1bbbe0a

admin@ncs# *** ALARM out-of-sync: got: 7f213b04d35d1c98f136fd3176c47747
expected: bfe5ec360065fa2b9555e3cbc1bbbe0a

admin@ncs#
```

4. Perform compare-config to determine why NSO thinks it is out-of-sync with the device:

```
admin@ncs# devices device PE_00 compare-config
diff
devices {
  device PE_00 {
    config {
      interface {
        Loopback 0 {
          ip {
            address {
              primary {
-             address 10.0.0.1;
+             address 10.0.0.2;
              }
            }
          }
        }
      }
    }
  }
}

admin@ncs#
```

Above we have seen how NSO 'sync-from' operation gets whatever configuration is present in the device and writes it in its internal database. There will be case where want to discard the configuration in the device and push the one present in NSO. In that case we can use 'sync-to' operation

5. Perform 'sync-to':

```
admin@ncs# devices device PE_00 sync-to
result true
admin@ncs# devices device PE_00 check-sync
```



```
result in-sync
admin@ncs#
```

6. Verify that the device configuration has been updated by direct CLI

```
cisco@ubuntu:~/nso572/ncs-run$ ncs-netns cli-i PE_00

admin connected from 198.18.133.252 using ssh on ubuntu
PE_00> enable
PE_00# show running-config interface Loopback 0
interface Loopback0
  no shutdown
  ip address 10.0.0.1 255.255.255.255
exit
PE_00#
```

Now we will see how we can configure directly the network device from NSO

7. Assign IP address to PE_10 from NSO CLI:

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# devices device PE_10 config interface Loopback 0
admin@ncs(config-if)# ipv4 address 10.0.0.3 255.255.255.255
admin@ncs(config-if)#
```

8. Verify the configuration before commit

```
admin@ncs(config-if)# show config
devices device PE_10
config
  interface Loopback 0
    ipv4 address 10.0.0.3 255.255.255.255
    no shutdown
  exit
!
!
admin@ncs(config-if)#
```

9. Verify different commit options

- a. Dry-run (NSO format). We can see with '+' sign what NSO will configure that is not yet present in the device

```
admin@ncs(config-if)# commit dry-run
cli {
  local-node {
    data devices {
      device PE_10 {
        config {
          interface {
            + Loopback 0 {
            +   ipv4 {
            +     address {
            +       ip 10.0.0.3;
            +       mask 255.255.255.255;
            +     }
            +   }
            + }
          }
        }
      }
    }
  }
}
admin@ncs(config-if)#
```

- b. Dry-run with device native representation

```
admin@ncs(config-if)# commit dry-run outformat native
native {
  device {
    name PE_10
    data interface Loopback 0
      ipv4 address 10.0.0.3 255.255.255.255
      no shutdown
    exit
  }
}
admin@ncs(config-if)#
```

c. Dry-run in XML format (This will be very useful later to create our templates)

```
admin@ncs(config-fi)# commit dry-run outformat xml
result-xml {
  local-node {
    data <devices xmlns="http://tail-f.com/ns/ncs">
      <device>
        <name>PE_10</name>
        <config>
          <interface xmlns="http://tail-f.com/ned/cisco-ios-
xr">
            <Loopback>
              <id>0</id>
              <ipv4>
                <address>
                  <ip>10.0.0.3</ip>
                  <mask>255.255.255.255</mask>
                </address>
              </ipv4>
            </Loopback>
          </interface>
        </config>
      </device>
    </devices>
  }
}
admin@ncs(config-if)#
```

10. Perform the commit. This will cause PE_10 to be configured

```
admin@ncs(config-if)# commit
Commit complete.
admin@ncs(config-if)#
```

11. Verify the configuration directly in PE_10 cli

```
cisco@ubuntu:~/nso572/ncs-run$ ncs-netsim cli-i PE_10
admin connected from 198.18.133.252 using ssh on ubuntu
ubuntu> enable
ubuntu# show running-config interface Loopback 0
interface Loopback 0
  no shutdown
  ipv4 address 10.0.0.3 255.255.255.255
exit
ubuntu# exit
cisco@ubuntu:~/nso572/ncs-run$
```

For loading big configurations at once you can use the API's we will later explore like Restconf and Netconf, but there are other methods you can load configuration through NSO CLI. Let's explore them.

12. Load the configuration of PE_11 loopback at once with 'load merge terminal'.

- a. Copy the output of 'show config' command we run for PE_10 loopback, update the device name and the IP.

```
devices device PE_11
config
interface Loopback 0
  ipv4 address 10.0.0.4 255.255.255.255
  no shutdown
exit
!
```

- b. From config mode run 'load merge terminal', paste the above configuration at once and in a new line click Ctrl+D to load the configuration

NOTE: When you see "Loading" that means is loading mode, you don't need to wait to start pasting your configuration.

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# load merge terminal
Loading.
devices device PE_11
config
interface Loopback 0
  ipv4 address 10.0.0.4 255.255.255.255
  no shutdown
exit
!
!

0 bytes parsed in 14.85 sec (0 bytes/sec)
admin@ncs(config)#
```

13. Let's verify the configuration was loaded properly and commit it

```
admin@ncs(config)# show config
devices device PE_11
config
interface Loopback 0
  ipv4 address 10.0.0.4 255.255.255.255
  no shutdown
exit
!
!
admin@ncs(config)# commit
Commit complete.
admin@ncs(config)#
```

14. Now we will load the final configuration for all loopbacks from a file called 'ios_&_iosxr_loopback_configs.cfg'

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# load merge ios_&_iosxr_loopback_configs.cfg
Loading.
463 bytes parsed in 0.23 sec (1.91 KiB/sec)
admin@ncs(config)#
```

NOTE: We can build base files to be updated/replicated offline from NSO configuration by running 'show running-config devices ...' and the pipe with ' | save <filename>'

15. You can open the file under ncs-run directory and see that we are configuring the Loopback 0 of 4 devices, but 'show config' command will show us only what NSO will configure that is not

already present in the device. NSO only will send to the device the diff between what we want to configure and what is already configured. In this example, the Loopback for PE_01

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# load merge /home/cisco/nso572/ncs-
run/ios_&_iosxr_loopback_configs.cfg
Loading.
458 bytes parsed in 0.23 sec (1.91 KiB/sec)
admin@ncs(config)#
```

16. Commit the configuration

Exercise 2: Simple Loopback Service

Task 1: Creation of a simple service

Let's create the skeleton of our service. Notice that initially our service will only consist in a YANG service model and an XML template, for this type of service we could choose 'ncs-make-package --service-skeleton template' option, but we will create it with the skeleton for python as well for later use.

1. Create a service skeleton

```
cisco@ubuntu:~/nso572/ncs-run$ cd packages
cisco@ubuntu:~/nso572/ncs-run/packages$ ncs-make-package --service-
skeleton python-and-template loopbackbasic
cisco@ubuntu:~/nso572/ncs-run/packages$
```

2. Investigate the package structure under '/home/cisco/nso572/ncs-run/packages/loopbackbasic/'

```
3. cisco@ubuntu:~/nso572/ncs-run/packages$ tree loopbackbasic/
loopbackbasic
+ python                               ← Service python code
|   + loopbackbasic
|   |   + __init__.py
|   |   + main.py
+ src
|   + yang                             ← Service YANG models
|   |   + loopbackbasic.yang
|   + Makefile                         ← Compiles the service package
+ templates                           ← XML Servic templates
|   + loopbackbasic-template.xml
+ test
+ package-meta-data.xml               ← Package version, requirements...
+ README
```

Let's write our YANG service model now

4. Edit the loopbackbasic.yang file inside src directory to reflect the following structure:

```
module loopbackbasic {

    namespace "http://example.com/loopbackbasic";
    prefix loopbackbasic;

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

    description
        "Bla bla...";

    revision 2016-01-01 {
        description
            "Initial revision.";
    }

    augment /ncs:services {
        list loopbackbasic {
            key name;
        }
    }
}
```

```

unique 'ip-address';
unique 'device loopback-number';

uses ncs:service-data;
ncs:servicepoint "loopbackbasic-servicepoint";

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

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

leaf loopback-number {
  tailf:info "Loopback Interface Number";
  mandatory true;
  type int32;
}

leaf ip-address {
  tailf:info "Valid IP";
  mandatory true;
  type inet:ipv4-address;
}
}
}
}

```

YANG can provide a great level of control and benefits in service design. Note in the above model:

- The device name is a reference (leafref) to the devices NSO already knows about, to avoid typos and allow autocompletion
- The IP address of the loopback should be unique in the network, so in the YANG we need to make it unique. With unique 'ip-address' NSO will enforce it to avoid human error
- Similar thing for loopback-number, the loopback-number should be unique in a device. the unique statement in combination the device denotes that the pairing of device and loopback-number should be unique.

5. Compile the service

```

cisco@ubuntu:~/nso572/ncs-run/packages$ cd loopbackbasic/src/
cisco@ubuntu:~/nso572/ncs-run/packages/loopbackbasic/src$ make all
mkdir -p ../load-dir
mkdir -p java/src//
/home/cisco/nso572/nso572/bin/ncsc `ls loopbackbasic-ann.yang` >
/dev/null 2>&1 && echo "-a loopbackbasic-ann.yang" ` \
-c -o ../load-dir/loopbackbasic.fxs
yang/loopbackbasic.yang
cisco@ubuntu:~/nso572/ncs-run/packages/loopbackbasic/src$

```

6. Reload NSO packages from CLI to load the new compiled package. Then verify the result **true**.

```

admin@ncs# packages reload

>>> 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-ios-cli-6.80
  result true
}
reload-result {
  package cisco-iosxr-cli-7.39
  result true
}
reload-result {
  package juniper-junos-nc-4.7
  result true
}
reload-result {
  package l3vpn
  result true
}
reload-result {
  package loopbackbasic
  result true
}
admin@ncs#
```

7. You can already try to configure a service, but nothing will be pushed to any device as we didn't yet configure the template

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# services loopbackbasic firstLoopback device PE_00 ip-
address 1.1.1.1 loopback-number 0
admin@ncs(config-loopbackbasic-firstLoopback)# commit dry-run outformat
native
native {
}
admin@ncs(config-loopbackbasic-firstLoopback)# abort
admin@ncs#
```

Now we will prepare the template. We will need 3 parts in the template, one for IOS devices other for IOS XR and another for JunOS. A fast way to do that is to use the “show running-config devices device” together with “| display xml” and then copy the desired section to the loopbackbasic-template.xml file. After that we need to parametrize it by replacing the variables.

8. Get the XML for configured Loopbacks for IOS and IOS XR

```
admin@ncs# show running-config devices device PE_00 config interface
Loopback 0 | display xml
<config xmlns="http://tail-f.com/ns/config/1.0">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>PE_00</name>
      <config>
        <interface xmlns="urn:ios">
          <Loopback>
            <name>0</name>
            <ip>
              <address>
                <primary>
                  <address>10.0.0.1</address>
                  <mask>255.255.255.255</mask>
                </primary>
              </address>
            </ip>
          </Loopback>
        </interface>
      </config>
    </device>
  </devices>
</config>
```

```

    </interface>
  </config>
</device>
</devices>
</config>
admin@ncs#
admin@ncs#
admin@ncs# show running-config devices device PE_10 config interface
Loopback 0 | display xml
<config xmlns="http://tail-f.com/ns/config/1.0">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>PE_10</name>
      <config>
        <interface xmlns="http://tail-f.com/ned/cisco-ios-xr">
          <Loopback>
            <id>0</id>
            <ipv4>
              <address>
                <ip>10.0.0.3</ip>
                <mask>255.255.255.255</mask>
              </address>
            </ipv4>
          </Loopback>
        </interface>
      </config>
    </device>
  </devices>
</config>
admin@ncs#

```

9. For JunOS we don't have yet configured a Loopback, but we can add the configuration and get the XML by using "commit dry-run outformat xml" without updating the device.

```

admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# devices device P_20 config configuration interfaces
interface lo0 unit 0 family inet address 10.0.0.5/32
admin@ncs(config-address-10.0.0.5/32)# commit dry-run outformat xml
result-xml {
  local-node {
    data <devices xmlns="http://tail-f.com/ns/ncs">
      <device>
        <name>P_20</name>
        <config>
          <configuration
xmlns="http://xml.juniper.net/xnm/1.1/xnm">
            <interfaces>
              <interface>
                <name>lo0</name>
                <unit>
                  <name>0</name>
                  <family>
                    <inet>
                      <address>
                        <name>10.0.0.5/32</name>
                      </address>
                    </inet>
                  </family>
                </unit>
              </interface>
            </interfaces>
          </configuration>
        </config>
      </device>
    }
  }
}

```



```

        </devices>
    }
}
admin@ncs (config-address-10.0.0.5/32) # abort
admin@ncs#

```

Several things to notice when parametrizing templates are:

- Notice the different tags like the following identifying the type of device it contains configuration for. This allows us to have templates for different device types under the same XML file
- xmlns=<http://tail-f.com/ned/cisco-ios-xr>
- When referring to a variable that is part of our YANG model we can use the path of that variable in our YANG using the following format: {/ip-address}

10. Use the information from the previous step to correctly populate the loopbackbasic-template.xml template for each type of device. This ensures the service configuration is mapped to the device configuration. The final file after replacing the variables by the same YANG leafs we defined before should look like this

```

<config-template xmlns="http://tail-f.com/ns/config/1.0">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>{/device}</name>
      <config>
        <!-- Configuration for IOS devices -->
        <interface xmlns="urn:ios">
          <Loopback>
            <name>{/loopback-number}</name>
            <ip>
              <address>
                <primary>
                  <address>{/ip-address}</address>
                  <mask>255.255.255.255</mask>
                </primary>
              </address>
            </ip>
          </Loopback>
        </interface>
        <!-- Configuration for IOS XR devices -->
        <interface xmlns="http://tail-f.com/ned/cisco-ios-xr">
          <Loopback>
            <id>{/loopback-number}</id>
            <ipv4>
              <address>
                <ip>{/ip-address}</ip>
                <mask>255.255.255.255</mask>
              </address>
            </ipv4>
          </Loopback>
        </interface>
        <!-- Configuration for JunOS devices -->
        <configuration xmlns="http://xml.juniper.net/xnm/1.1/xnm">
          <interfaces>
            <interface>
              <name>lo{/loopback-number}</name>
              <unit>
                <name>0</name>
                <family>
                  <inet>
                    <address>
                      <name>{/ip-address}/32</name>
                    </address>

```

```

        </inet>
      </family>
    </unit>
  </interface>
</interfaces>
</configuration>
</config>
</device>
</devices>
</config-template>

```

11. After update the template and save it, reload NSO packages for the changes to make effect.

```

admin@ncs# packages reload
reload-result {
  package cisco-ios-cli-6.80
  result true
}
reload-result {
  package cisco-iosxr-cli-7.39
  result true
}
reload-result {
  package juniper-junos-nc-4.7
  result true
}
reload-result {
  package l3vpn
  result true
}
reload-result {
  package loopbackbasic
  result true
}
admin@ncs#
System message at 2022-06-06 15:52:56...
  Subsystem stopped: ncs-dp-4-cisco-ios-cli-6.80:IOSDp
admin@ncs#
System message at 2022-06-06 15:52:56...
  Subsystem stopped: ec_junos_ext_vlan_hook_daemon
admin@ncs#
System message at 2022-06-06 15:52:56...
  Subsystem started: ec_junos_ext_vlan_hook_daemon
admin@ncs#

```

Once you save the template, we are ready to start provisioning loopback interfaces using our service (notice that updating the template file does not require recompile the package for simple packages).

12. Let's instantiate a couple services for our Loopbacks, this time using Loopback 1

a. Create 3 services, one for each vendor

```

admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# services loopbackbasic PE_00_loop1 device PE_00
loopback-number 1 ip-address 1.1.1.1
admin@ncs(config-loopbackbasic-PE_00_loop1)# top
admin@ncs(config)# services loopbackbasic PE_10_loop1 device PE_10
loopback-number 1 ip-address 3.3.3.3
admin@ncs(config-loopbackbasic-PE_10_loop1)# top
admin@ncs(config)# services loopbackbasic PE_20_loop1 device P_20
loopback-number 1 ip-address 5.5.5.5
admin@ncs(config-loopbackbasic-PE_20_loop1)# top
admin@ncs(config)# show config
services loopbackbasic PE_00_loop1
device PE_00

```

```

loopback-number 1
ip-address      1.1.1.1
!
services loopbackbasic PE_10_loop1
device          PE_10
loopback-number 1
ip-address      3.3.3.3
!
services loopbackbasic PE_20_loop1
device          P_20
loopback-number 1
ip-address      5.5.5.5
!

```

- b. Let's see the dry-run of what would be configured on the devices, in native mode and compare the configuration used per device model

```

admin@ncs(config)# commit dry-run outformat native
native {
  device {
    name PE_00
    data interface Loopback1
      ip address 1.1.1.1 255.255.255.255
      no shutdown
    exit
  }
  device {
    name PE_10
    data interface Loopback 1
      ipv4 address 3.3.3.3 255.255.255.255
      no shutdown
    exit
  }
  device {
    name P_20
    data <rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
      message-id="1">
        <edit-config
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
          <target>
            <candidate/>
          </target>
          <test-option>test-then-set</test-option>
          <error-option>rollback-on-error</error-option>
          <config>
            <configuration
xmlns="http://xml.juniper.net/xnm/1.1/xnm">
              <interfaces>
                <interface>
                  <name>lo1</name>
                  <unit>
                    <name>0</name>
                    <family>
                      <inet>
                        <address>
                          <name>5.5.5.5/32</name>
                        </address>
                      </inet>
                    </family>
                  </unit>
                </interface>
              </interfaces>
            </configuration>
          </config>
        </edit-config>

```

```

        </rpc>
    }
}
admin@ncs (config) #

```

c. Now commit the service.

```

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

```

13. Connect directly to a device via CLI and verify the configuration has been pushed

```

cisco@ubuntu:~/nso572/ncs-run$ ncs-netsim cli-i PE_20

admin connected from 198.18.133.252 using ssh on ubuntu
ubuntu> enable
ubuntu# show running-config configuration interfaces interface lo1
configuration interfaces interface lo1
  unit 0
    family inet address 5.5.5.5/32
  !
!
!
ubuntu#

```

14. From NSO CLI let's check the service configuration

```

admin@ncs# show running-config services loopbackbasic
services loopbackbasic PE_00_loop1
  device PE_00
  loopback-number 1
  ip-address 1.1.1.1
!
services loopbackbasic PE_10_loop1
  device PE_10
  loopback-number 1
  ip-address 3.3.3.3
!
services loopbackbasic PE_20_loop1
  device P_20
  loopback-number 1
  ip-address 5.5.5.5
!
admin@ncs# show running-config services loopbackbasic | tab

```

NAME	DEVICE	LOOPBACK NUMBER	IP ADDRESS
PE_00_loop1	PE_00	1	1.1.1.1
PE_10_loop1	PE_10	1	3.3.3.3
PE_20_loop1	P_20	1	5.5.5.5

```

admin@ncs#

```

15. Let's now rollback the last configuration change, which will correspond to our 3 services

```

admin@ncs# config
Entering configuration mode terminal
admin@ncs (config) # rollback configuration
admin@ncs (config) # show config
no services loopbackbasic PE_00_loop1
no services loopbackbasic PE_10_loop1
no services loopbackbasic PE_20_loop1
admin@ncs (config) #

```

16. We can see what will be removed from the target devices before we rollback

```

admin@ncs (config) # commit dry-run outformat native

```

```

native {
  device {
    name PE_00
    data no interface Loopback1
  }
  device {
    name PE_10
    data no interface Loopback 1
  }
  device {
    name P_20
    data <rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
      message-id="1">
        <edit-config
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
          <target>
            <candidate/>
          </target>
          <test-option>test-then-set</test-option>
          <error-option>rollback-on-error</error-option>
          <config>
            <configuration
xmlns="http://xml.juniper.net/xnm/1.1/xnm">
              <interfaces>
                <interface
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
                  nc:operation="delete">
                  <name>lo1</name>
                </interface>
              </interfaces>
            </configuration>
          </config>
        </edit-config>
      </rpc>
    }
  }
}
admin@ncs (config) #

```

17. Commit the rollback

```

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

```

18. Verify the configuration has been removed from the device

```

cisco@ubuntu:~/nso572/ncs-run$ ncs-netsim cli-i P_20

admin connected from 198.18.133.252 using ssh on ubuntu
ubuntu> enable
ubuntu# show running-config configuration interfaces interface lo1
-----^
syntax error: element does not exist
ubuntu#

```

Task 2: Simple Python-based service

We can add python (or Java) code into a service to derive parameters, integrate with other systems and many other applications. In the current example we will just assign value to a new variable to see how we can pass variables and data from Python code to the template

1. Update the main.py file with the following code
(packages/loopbackbasic/python/loopbackbasic/main.py)

```
class ServiceCallbacks(Service):  
  
    # The create() callback is invoked inside NCS FASTMAP and  
    # must always exist.  
    @Service.create  
    def cb_create(self, tctx, root, service, proplist):  
        self.log.info('Service create(service=', service._path, ')')  
  
        vars = ncs.template.Variables()  
        vars.add('MASK', '255.255.255.255')  
        vars.add('CIDR', '32')  
        template = ncs.template.Template(service)  
        template.apply('loopbackbasic-template', vars)
```

2. To refer to the above MASK and CIDR variables in the template use “{\$MASK}” and “{\$CIDR}”

```
<config-template xmlns="http://tail-f.com/ns/config/1.0">  
  <devices xmlns="http://tail-f.com/ns/ncs">  
    <device>  
      <name>{/device}</name>  
      <config>  
        <!-- Configuration for IOS devices -->  
        <interface xmlns="urn:ios">  
          <Loopback>  
            <name>{/loopback-number}</name>  
            <ip>  
              <address>  
                <primary>  
                  <address>{/ip-address}</address>  
                  <mask>{$MASK}</mask>  
                </primary>  
              </address>  
            </ip>  
          </Loopback>  
        </interface>  
        <!-- Configuration for IOS XR devices -->  
        <interface xmlns="http://tail-f.com/ned/cisco-ios-xr">  
          <Loopback>  
            <id>{/loopback-number}</id>  
            <ipv4>  
              <address>  
                <ip>{/ip-address}</ip>  
                <mask>{$MASK}</mask>  
              </address>  
            </ipv4>  
          </Loopback>  
        </interface>  
        <!-- Configuration for JunOS devices -->  
        <configuration xmlns="http://xml.juniper.net/xnm/1.1/xnm">  
          <interfaces>  
            <interface>  
              <name>lo{/loopback-number}</name>  
              <unit>  
                <name>0</name>  
                <family>
```

```

        <inet>
        <address>
        <name>{/ip-address}/{$CIDR}</name>
        </address>
        </inet>
    </family>
</unit>
</interface>
</interfaces>
</configuration>
</config>
</device>
</devices>
</config-template>

```

3. Compile the service

```

cisco@ubuntu:~/nso572/ncs-run/packages/loopbackbasic/src$ make clean all
rm -rf ../load-dir java/src//
mkdir -p ../load-dir
mkdir -p java/src//
/home/cisco/nso572/nso572/bin/ncsc `ls loopbackbasic-ann.yang` >
/dev/null 2>&1 && echo "-a loopbackbasic-ann.yang" ` \
    -c -o ../load-dir/loopbackbasic.fxs
yang/loopbackbasic.yang
cisco@ubuntu:~/nso572/ncs-run/packages/loopbackbasic/src$

```

4. Reload the packages

```

admin@ncs# packages reload
reload-result {
    package cisco-ios-cli-6.80
    result true
}
reload-result {
    package cisco-iosxr-cli-7.39
    result true
}
reload-result {
    package juniper-junos-nc.4.7
    result true
}
reload-result {
    package l3vpn
    result true
}
reload-result {
    package loopbackbasic
    result true
}
admin@ncs#

```

5. Instantiate some service and explore some debug options

a. Configure the services

```

admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# services loopbackbasic PE_00_loop1 device PE_00
loopback-number 1 ip-address 1.1.1.1
admin@ncs(config-loopbackbasic-PE_00_loop1)# top
admin@ncs(config)# services loopbackbasic PE_10_loop1 device PE_10
loopback-number 1 ip-address 3.3.3.3
admin@ncs(config-loopbackbasic-PE_10_loop1)# top
admin@ncs(config)# services loopbackbasic PE_20_loop1 device P_20
loopback-number 1 ip-address 5.5.5.5
admin@ncs(config-loopbackbasic-PE_20_loop1)# top

```

```
admin@ncs (config) #
```

b. Debug the use of the template

```
admin@ncs (config) # commit dry-run | debug ?
Possible completions:
  dependencies    Display dependency debug info
  kicker          Display kicker debug info
  service         Display service debug info
  template        Display template debug info
  xpath           Display XPath debug info
admin@ncs (config) # commit dry-run | debug template
Evaluating "/device" (from file "loopbackbasic-template.xml", line 4)
Context node:
/services/loopbackbasic:loopbackbasic[name='PE_00_loop1']
Result:

...

Result:
For /services/loopbackbasic:loopbackbasic[name='PE_00_loop1'], it
evaluates to "1.1.1.1"
Setting
/devices/device[name='PE_00']/config/ios:interface/Loopback[name='1']
/ip/address/primary/address to "1.1.1.1"
Operation 'merge' on non-existing node:
/devices/device[name='PE_00']/config/ios:interface/Loopback[name='1']
/ip/address/primary/mask (from file "loopbackbasic-template.xml",
line 14)
Evaluating "$MASK" (from file "loopbackbasic-template.xml", line 14)
Context node:
/services/loopbackbasic:loopbackbasic[name='PE_00_loop1']
Result: "255.255.255.255"
Setting
/devices/device[name='PE_00']/config/ios:interface/Loopback[name='1']
/ip/address/primary/mask to "255.255.255.255"

...

Result:
For /services/loopbackbasic:loopbackbasic[name='PE_20_loop1'], it
evaluates to "5.5.5.5"
Fetching literal "/" (from file "loopbackbasic-template.xml", line
42)
Evaluating "$CIDR" (from file "loopbackbasic-template.xml", line 42)
Context node:
/services/loopbackbasic:loopbackbasic[name='PE_20_loop1']
Result: "32"
Operation 'merge' on non-existing node:
/devices/device[name='P_20']/config/junos:configuration/interfaces/in
terface[name='lo1']/unit[name='0']/family/inet/address[name='5.5.5.5/
32'] (from file "loopbackbasic-template.xml", line 42)
cli {

...

```

c. Debug the service with the refcounts (points to how many service instances are using the same configuration line)

```
admin@ncs (config) # commit dry-run | debug service

Service: /services/loopbackbasic:loopbackbasic[name='PE_00_loop1']
shared_create
/devices/device[name='PE_00']/config/ios:interface/Loopback[name='1']
, refcount: 1

```



```

shared_set
/devices/device[name='PE_00']/config/ios:interface/Loopback[name='1']
/ip/address/primary/mask: 255.255.255.255, refcount: 1
shared_set
/devices/device[name='PE_00']/config/ios:interface/Loopback[name='1']
/ip/address/primary/address: 1.1.1.1, refcount: 1
Service done

Service: /services/loopbackbasic:loopbackbasic[name='PE_10_loop1']
shared_create /devices/device[name='PE_10']/config/cisco-ios-
xr:interface/Loopback[id='1'], refcount: 1
shared_set /devices/device[name='PE_10']/config/cisco-ios-
xr:interface/Loopback[id='1']/ipv4/address/mask: 255.255.255.255,
refcount: 1
shared_set /devices/device[name='PE_10']/config/cisco-ios-
xr:interface/Loopback[id='1']/ipv4/address/ip: 3.3.3.3, refcount: 1
Service done

Service: /services/loopbackbasic:loopbackbasic[name='PE_20_loop1']
shared_create
/devices/device[name='P_20']/config/junos:configuration/interfaces/in
terface[name='lo1'], refcount: 1
shared_create
/devices/device[name='P_20']/config/junos:configuration/interfaces/in
terface[name='lo1']/unit[name='0'], refcount: 1
shared_create
/devices/device[name='P_20']/config/junos:configuration/interfaces/in
terface[name='lo1']/unit[name='0']/family/inet, refcount: 1
shared_create
/devices/device[name='P_20']/config/junos:configuration/interfaces/in
terface[name='lo1']/unit[name='0']/family/inet/address[name='5.5.5.5/
32'], refcount: 1
Service done

cli {
...

```

d. Commit

```

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

```

6. Verify the device backpointers and refcounts

- a. PE_10 (IOS XR) Loopback 0 has not Refcount as it was not configured through a service, but Loopback 1 Refcount is 1 and Backpointer tells you the service instance that is using it.

```

admin@ncs# show running-config devices device PE_10 config interface
Loopback | display service-meta-data
devices device PE_10
config
interface Loopback 0
ipv4 address 10.0.0.3 255.255.255.255
no shutdown
exit
! Refcount: 1
! Backpointer: [
/ncs:services/loopbackbasic:loopbackbasic[loopbackbasic:name='PE_10_1
oop1'] ]
interface Loopback 1
! Refcount: 1

```

```

    ipv4 address 3.3.3.3 255.255.255.255
    no shutdown
    exit
    !
    !
admin@ncs#

```

b. P_20 (JunOS)

- i. Juniper has a default configuration for Loopback 0. You can see it is not owned by any service

```

admin@ncs# show running-config devices device P_20 config
configuration interfaces interface lo0 | display service-meta-
data
devices device P_20
config
  configuration interfaces interface lo0
    unit 0
      family inet address 192.168.1.1/32
    !
  !
  !
  !
  !
admin@ncs#

```

- ii. Now let's configure a service that will overwrite it and see the refcounts

```

admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# services loopbackbasic P_20_loop0 device
P_20 loopback-number 0 ip-address 6.6.6.6
admin@ncs(config-loopbackbasic-P_20_loop0)# commit
Commit complete.
admin@ncs(config-loopbackbasic-P_20_loop0)# exit
admin@ncs(config)# exit
admin@ncs# show running-config devices device P_20 config
configuration interfaces interface lo0 | display service-meta-
data
devices device P_20
config
  ! Refcount: 2
  ! Backpointer: [
/ncs:services/loopbackbasic:loopbackbasic[loopbackbasic:name='P
_20_loop0'] ]
  configuration interfaces interface lo0
    ! Refcount: 2
    ! Backpointer: [
/ncs:services/loopbackbasic:loopbackbasic[loopbackbasic:name='P
_20_loop0'] ]
    unit 0
      ! Refcount: 1
      ! Backpointer: [
/ncs:services/loopbackbasic:loopbackbasic[loopbackbasic:name='P
_20_loop0'] ]
      ! Refcount: 2
(/devices/device{P_20}/config/junos:configuration/interfaces/in
terface{lo0}/unit{0}/family/inet)
      ! Backpointer: [
/ncs:services/loopbackbasic:loopbackbasic[loopbackbasic:name='P
_20_loop0'] ]
(/devices/device{P_20}/config/junos:configuration/interfaces/in
terface{lo0}/unit{0}/family/inet)
      family inet address 6.6.6.6/32
    !
  !

```

```
!  
!  
!  
!  
admin@ncs#
```

For JunOS note that:

- the refcount under “interface lo0” or “unit 0 family inet” is owned both by the service and by the configuration that was there before present (notice backpointers don’t show any path if the other owner is the configuration of the device before any service),
- but the refcount for assigning the IP address value is only 1, as that value is only owned by our service. Here a rollback would restore the original default value.

Task 3: Automation

Python (or Java) can be used on top of NSO as a tool to find/discover details of the network onboarded in NSO, by accessing the database (CDB). The script under “scripts/get-config.py” connect to NSO CDB and returns all Loopback interfaces and IP addresses across for PE_10.

```
import ncs

# This first lines open a read transaction to NSO CDb
with ncs.maapi.Maapi() as m:
    with ncs.maapi.Session(m, 'admin', 'python'):
        with m.start_read_trans() as t:
            root = ncs.maagic.get_root(t)
            # From this point "root" is the base of our CDB to navigate

            device_name = "PE_10"

            dev = root.devices.device[device_name]
            hostname = dev.config.cisco_ios_xr_hostname;
            print("{} hostname: {}".format(device_name, str(hostname)))

            # The following returns parameters for all configured Loopback
            interfaces
            for Loopback in dev.config.cisco_ios_xr_interface.Loopback:
                print("Loopback" + str(Loopback.id) + " IP address: " +
                    str(Loopback.ipv4.address.ip))
```

1. Set a hostname for device PE_10

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# devices device PE_10 config hostname PE_10
admin@ncs(config-config)# commit
Commit complete.
admin@ncs(config-config)#
```

2. Run the python script

```
cisco@ubuntu:~/nso572/ncs-run/scripts$ python get-config.py
PE_10 hostname: PE_10
Loopback0 IP address: 10.0.0.3
Loopback1 IP address: 3.3.3.3
cisco@ubuntu:~/nso572/ncs-run/scripts$
```

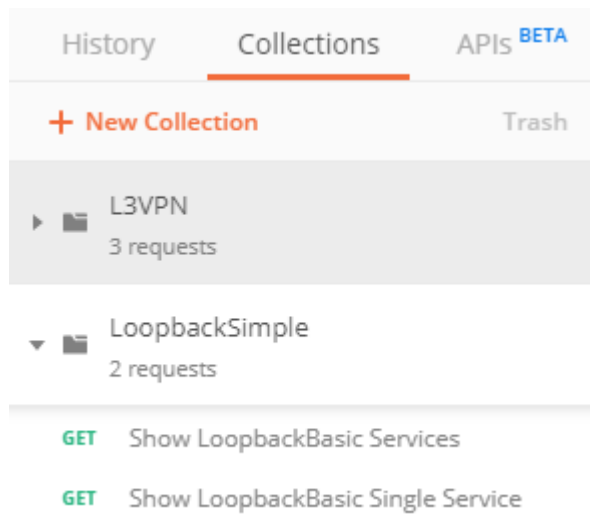
Task 4: Resconf API

During the exercises above we saw how we can interact with NSO through CLI and GUI, but generally NSO will have to integrate with other applications and tools, or with Operations and Business portals (OSS, BSS). NSO provides several APIs to integrate among REST, Restconf and Netconf. For the next verification we will guide you to use Restconf to configure a new service and see its configuration.

1. Open Postman application from the Desktop



2. You will find on the left panel a collection of already prepared calls to interact with NSO loopbackbasic service called 'LoopbackSimple'. Click the arrow to expand it



3. Open 'Show LoopbackBasic Services' This Restconf call request the configuration of all configured loopback services (as indicated by the URI). Click send button.
4. If you left some of the previous services you will be able to see in bottom section of the screen the response from NSO

NOTE: You can practice more RESTCONF calls in Exercise 3 L3VPN.

Exercise 3: L3VPN

NOTE: The following service is independent from Exercise 2 and can be run individually. You will note that some concepts are repeated here in case you decide to start with this exercise.

Task 1: Get familiar with the L3VPN Service

In this task we will explore the current working L3VPN service:

- Service Model in YANG
- Configuration Template in XML
- Mapping through Python (direct mapping)

Step 1: Review the YANG Service Model

Our L3VPN Service Model require the following parameters:

Attribute	YANG name	Description
VPN ID	vpn-id	Unique identifier describing an instance of a deployed service
VPN name	vpn-name	VPN instance name
Customer	customer	Customer to which the VPN instance belongs
Link	link	Customer link (each VPN instance can have multiple links)
Link ID	link-id	Unique identifier describing an instance of a list link
Interface	interface	Interface on the PE router to which the customer site is connected
Routing protocol	routing-protocol	Routing protocol option. Currently Static and RIP supported
Device	device	Device on which the service will be deployed
Static	static	Static routing option
Prefix	prefix	Route prefix (only if routing-protocol is Static)
Mask	mask	Route mask (only if routing-protocol is Static)

Let's explore the current YANG model to identify the most important details. You can find the YANG model under `ncs-run/packages/l3vpn/src/yang/l3vpn.yang`

- A general variable parameter is defined as a 'leaf' of our Service Model tree structure.

- For every parameter we need to indicate the type, for 'vpn-name' this is 'type string'

```
augment /ncs:services {
  list l3vpn {
    description "L3VPN service";

    key vpn-name;

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

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

- In the above section we can see that our L3VPN service uses the 'vpn-name' as key for every instance, that implies that this parameter needs to be unique.
- We have a special type of leaf called 'leafref'. It indicates that they only allow values, already configured in other Service Models under NSO structure. The following indicates that we can only select customers that are already configured in NSO under '/customers/customer/id'

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

- Each VPN service instance can support multiple customer links, which are represented with the list link. A link instance gets a unique link ID and a link name. Each link points to a specific interface in a device.

```
list link {
  tailf:info "PE-CE Attachment Point";
  key link-id;
  unique "device interface";
  leaf link-id {
    tailf:info "Link ID";
    type uint32 {
      range "1..255";
    }
  }
  leaf link-name {
    tailf:info "Link Name";
    type string;
  }
  leaf device {
    tailf:info "PE Router";
    type leafref {
      path "/ncs:devices/ncs:device/ncs:name";
    }
  }
  leaf interface {
    tailf:info "Customer Facing Interface";
    type string;
  }
}
```

- Notice that link-id is restricted to be between 1 and 255.

- Each link instance gets a routing-protocol option. If static routing is used, route prefix and route mask attributes are used. This is specified with a 'when' condition that points through xpath to our routing-protocol variable.

```

leaf routing-protocol {
  tailf:info "Routing option on PE-CE link";
  type enumeration {
    enum rip;
    enum static;
  }
}
list static {
  tailf:info "Static Route";
  key prefix;
  when "../routing-protocol='static'";
  leaf prefix {
    tailf:info "Static Route Prefix";
    type inet:ipv4-address;
  }
  leaf mask {
    tailf:info "Static Route Subnet Mask";
    type inet:ipv4-address;
  }
}

```

Step 2: Review the XML template for PE (IOS)

Currently the service is only supporting PE devices running Cisco IOS software. This will be later updated in following tasks to add support for Cisco IOS XR PE routers.

Templates are built using XML. NSO allows you to take an already present configuration in a device and display it in XML for easy copy/paste into the template file and start parametrizing your variables. We will practice this in following tasks.

Locate the template under ncs-run/packages/l3vpn/l3vpn-template.xml

Refer to the following device configuration to compare it with the content of the template:

```

vrf definition vpn10001
description Customer ACME VPN
rd 1:10001
route-target export 1:10001
route-target import 1:10001
!
ip route vrf vpn10001 192.168.11.0 255.255.255.0 172.31.1.2
!
interface GigabitEthernet4
description Connection to Customer ACME - Site 5
vrf forwarding vpn10001
ip address 172.31.1.1 255.255.255.252
exit
!
router bgp 1
address-family ipv4 unicast vrf vpn10001
redistribute connected
redistribute static
exit-address-family
!
!
router rip
address-family ipv4 vrf vpn10001
network 0.0.0.0
default-information originate

```



```
exit-address-family
!  
!
```

l3vpn-template.xml

```
<config-template xmlns="http://tail-f.com/ns/config/1.0">  
  <devices xmlns="http://tail-f.com/ns/ncs">  
    <device>  
      <name>{/link/device}</name>  
      <config>  
        <!-- IOS -->  
        <vrf xmlns="urn:ios">  
          <definition>  
            <name>vpn{string(..../vpn-id)}</name>  
            <rd>1:{string(..../vpn-id)}</rd>  
            <route-target>  
              <export>  
                <asn-ip>1:{string(..../vpn-id)}</asn-ip>  
              </export>  
              <import>  
                <asn-ip>1:{string(..../vpn-id)}</asn-ip>  
              </import>  
            </route-target>  
          </definition>  
        </vrf>  
        <?if {routing-protocol='static'}?>  
          <ip xmlns="urn:ios">  
            <route>  
              <vrf>  
                <name>vpn{string(..../vpn-id)}</name>  
                <ip-route-forwarding-list>  
                  <prefix>{string(static/prefix)}</prefix>  
                  <mask>{string(static/mask)}</mask>  
                  <forwarding-address>172.31.{string(link-  
id)}.2</forwarding-address>  
                </ip-route-forwarding-list>  
              </vrf>  
            </route>  
          </ip>  
          <?end?>  
          <interface xmlns="urn:ios">  
            <GigabitEthernet>  
              <name>{interface}</name>  
              <description>Connection to Customer ACME - Site  
5</description>  
              <vrf>  
                <forwarding>vpn{string(..../vpn-id)}</forwarding>  
              </vrf>  
              <ip>  
                <address>  
                  <primary>  
                    <address>172.31.{link-id}.1</address>  
                    <mask>255.255.255.252</mask>  
                  </primary>  
                </address>  
              </ip>  
            </GigabitEthernet>  
          </interface>  
          <router xmlns="urn:ios">  
            <?if {routing-protocol='rip'}?>  
              <rip>  
                <address-family>  
                  <ipv4>
```

```

        <vrf>
            <name>vpn{string(..../vpn-id)}</name>
            <network>
                <ip>0.0.0.0</ip>
            </network>
            <default-information>
                <originate/>
            </default-information>
        </vrf>
    </ipv4>
</address-family>
</rip>
<?end?>
</router>
</config>
</device>
</devices>
</config-template>

```

- Notice that all configuration is placed inside '`<config-template> <devices> <device> <config>`' where we will have as many `<device>` as network elements we need to configure at once for a service instance (one for this service).
- The template already points to the variables our YANG service model. For example, our variable containing the device name is part of the list 'link', so the template refers to it with XPATH as '/link/device'
- Some sections are dependent on the value or a variable. In our service depending on the 'routing-protocol' selected some Static or RIP configuration will be present. These sections are contained in structures like the following:

```

<?if {routing-protocol='static'}?>
<...>
<?end?>

```

- The xml tag 'xmlns="urn:ios"' indicates NSO that this part of the template is specific for IOS devices. We will use this later to distinguish between IOS and IOS XR configurations in the same template file.

Step 3: Review the Mapping from Service Model to Template

In its simplest the mapping between template and YANG model will be in a 1:1 form like in the initial setup of this service. That means that every parameter used in the template comes directly from YANG and we don't require any additional processing.

NSO allows advances conditions and verifications to be in both YANG model and XML template, allowing as adding conditions and validations to our variables and include loops.

For any other data processing, to get some parameters from external databases or sources, to integrate with other platforms and services or more complex implementations we can use Python and JAVA to develop the required actions.

In this lab we will use Python. The main file we will use can be located under `ncs-run/packages/l3vpn/python/l3vpn/main.py`

We will only need to look at this section for this lab:

```

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

    ## Here we will add our validations and calculations

```

```
vars = ncs.template.Variables()
vars.add('DUMMY', '127.0.0.1')
template = ncs.template.Template(service)
template.apply('l3vpn-template', vars)
```

This function gets the parameters used as input for creating a service instance, that we can use and modify before passing them to the template. Currently we are not doing anything and every parameter in the template comes directly from our YANG model.

The line `vars.add('DUMMY', '127.0.0.1')` is an example on how to pass a parameter from python to the template, which later will be referred in the template as `{DUMMY}`.

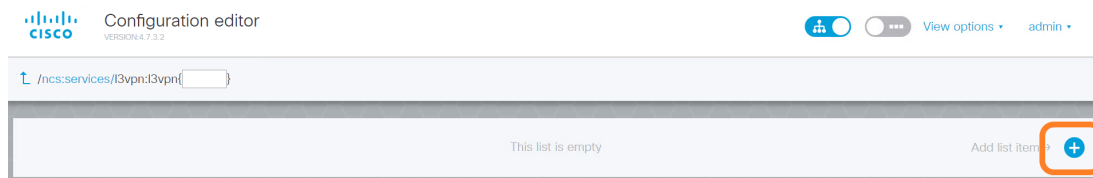
Step 4: Test provisioning of a L3VPN Service

First time we will use GUI to provision L3VPN to get familiar, but during next sections the outputs from CLI be shown for faster provisioning. You can always choose to come to the GUI and do the same there.

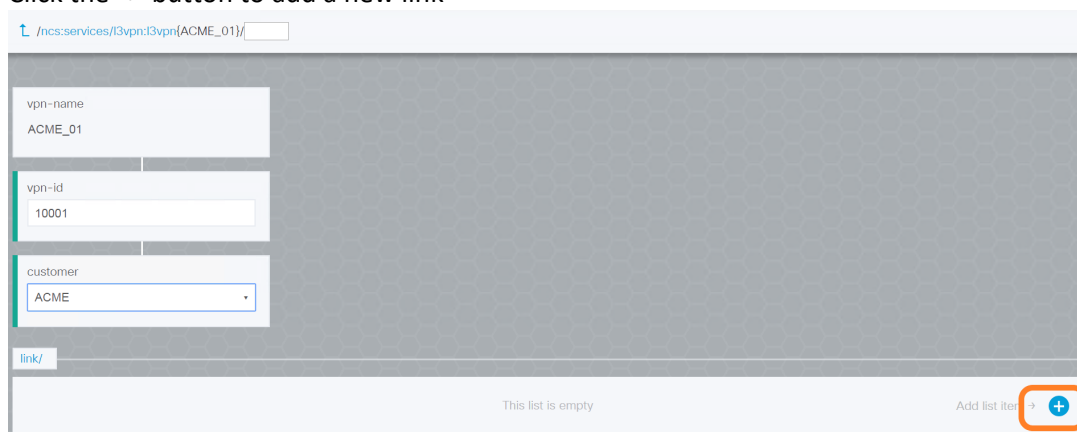
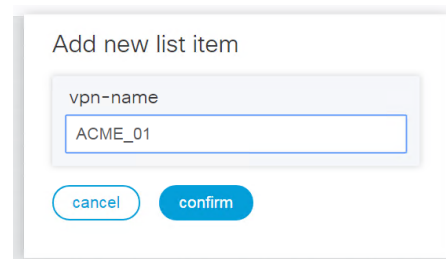
1. Go to NSO GUI (<http://198.18.134.28:8080>), login as admin/admin.
2. Click in 'Configuration Editor'. L3VPN service is nested under 'ncs:services' Module, click on it and navigate to the L3VPN section.

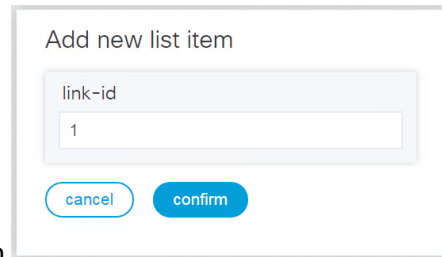
NOTE: We have added a bookmark to go directly to L3VPN service in Chrome.

3. Click the "+" button to add a new L3VPN instance



4. Add the vpn-name 'ACME_01' and click 'confirm' button.
5. Click in the list the name of your vpn-name to start editing it.
6. Add 'vpn-id' 10001 and select customer 'ACME'
7. Click the '+' button to add a new link





Add new list item

link-id

1

cancel confirm

8. Identify this link with 'link-id' 1 and click confirm button

9. Click in the 'link-id' 1 from the table to start editing it

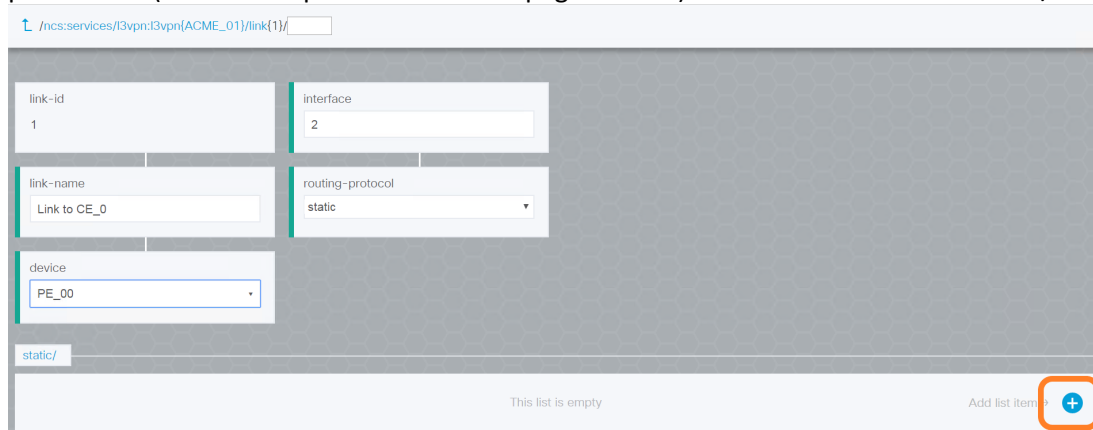
10. Name this link as 'link-name' 'Link to CE_0'

11. Select the PE device where to configure the L3VPN PE_00

12. Use 'interface' 2, this will configure PE_00 GigabitEthernet 2

13. Select 'routing-protocol' 'Static'

14. Because we have selected 'Static' a new section appears to provision "Static" specific parameters (if it doesn't please refresh the page with F5). Click the '+' button on 'static/' section.



/ncs:services/l3vpn{l3vpn{ACME_01}}/link{1}/

link-id 1	interface 2
link-name Link to CE_0	routing-protocol static
device PE_00	

static/

This list is empty

Add list item +

15. Introduce 'prefix' '192.168.11.0' and click confirm button

16. Click in the prefix in the list to edit it.

17. Add 'mask' '255.255.255.0'

18. Once all parameters have been introduced click in the bottom of the page the 'Commit Manager' button. Notice it has an '*' symbol indicating there are some changes pending to be committed.

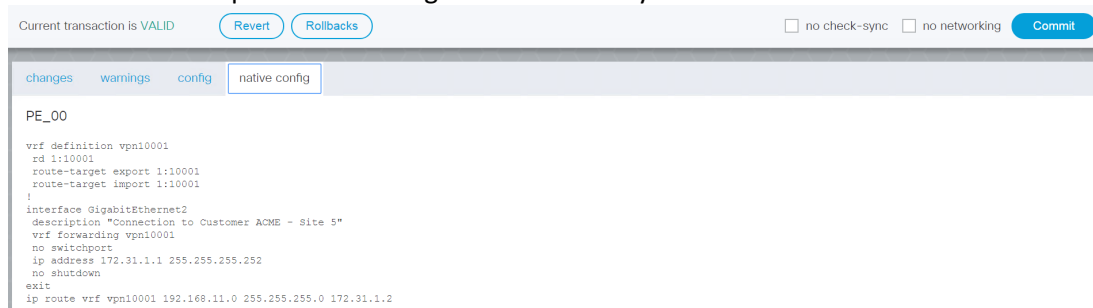


19. NSO performs a validation to make sure the parameters introduced are ok. If you get any error please go back to the service and update as required

20. The 'config' tab shows you the configuration that NSO will add for the service and the device configured. This is equivalent of doing in cli 'commit dry-run'



21. The 'native config' tab will show you the configuration that will be pushed into PE_00 showed in native cli. This is equivalent of doing in cli 'commit dry-run outformat native'.



22. Click 'commit' button to provision the device.
23. You can then go to 'Device Manager' button to explore the device configuration, but this is much faster to verify from cli.
24. What we did above is equivalent of writing this in cli from config mode:

```
admin@ncs(config)# services 13vpn ACME_01 vpn-id 10001 customer ACME
link 1 link-name "Link to CE_0" device PE_00 interface 2 routing-
protocol static static 192.168.11.0 mask 255.255.255.0
admin@ncs(config)# commit
```

25. Use the following command to verify the configuration that was pushed to PE_00

```
admin@ncs# show running-config devices device PE_00 config vrf
definition vpn10001
devices device PE_00
config
vrf definition vpn10001
rd 1:10001
route-target export 1:10001
route-target import 1:10001
!
!
!
admin@ncs# show running-config devices device PE_00 config ip route vrf
vpn10001
devices device PE_00
config
ip route vrf vpn10001 192.168.11.0 255.255.255.0 172.31.1.2
!
!
admin@ncs# show running-config devices device PE_00 config interface G
Possible completions:
GigabitEthernet - GigabitEthernet IEEE 802.3z
```

```
Group-Async      - Async Group interface
admin@ncs# show running-config devices device PE_00 config interface
GigabitEthernet 2
devices device PE_00
config
interface GigabitEthernet2
description Connection to Customer ACME - Site 5
no switchport
vrf forwarding vpn10001
ip address 172.31.1.1 255.255.255.252
no shutdown
exit
!
!
admin@ncs#
```

Task 2: Update the Service Model

During this task we will make a couple of enhancements and expansions to our L3VPN service

- A. Restrict vpn-id values to 10001-19999
- B. Configure description for 'vrf definition'
- C. Add BGP as an additional PE-CPE routing-protocol

Step 1: Restrict vpn-id values

New requirements restrict vpn-id values to be between 10001-19999.

Verify current behaviour: No restriction for vpn-id value

If we review our YANG we will not find any restrictions for vpn-id values. We can verify this by updating the value from the service we configured above.

```
admin@ncs(config)# services l3vpn ACME_01 vpn-id ?
Description: Service Instance ID
Possible completions:
<unsignedInt>[10001]
admin@ncs(config)# services l3vpn ACME_01 vpn-id 12345
admin@ncs(config-l3vpn-ACME_01)# commit dry-run outformat native
native {
    device {
        name PE_00
        data no ip route vrf vpn10001 192.168.11.0 255.255.255.0 172.31.1.2
          vrf definition vpn12345
            rd 1:12345
            route-target export 1:12345
            route-target import 1:12345
          !
        interface GigabitEthernet2
          vrf forwarding vpn12345
        exit
        no vrf definition vpn10001
        ip route vrf vpn12345 192.168.11.0 255.255.255.0 172.31.1.2
    }
}
admin@ncs(config-l3vpn-ACME_01)#
```

Update YANG model

YANG modelling language allows specifying details for each parameter that allows us to validate the data and help introducing it for a new instance. Some possibilities are: when, must, range, path

For more check the [YANG RFC](#) and the [YANG 1.1 RFC](#).

In this exercise we will edit our l3vpn.yang file adding a range for 'vpn-id'. Update it as follows:

```
leaf vpn-id {
    tailf:info "Service Instance ID";
    type uint32 {
        range "10001..19999";
    }
}
```

Step 2: Configure description for VRF definition

Current exercise will add a description for every VRF definition configured in the device with the customer name, for visual identification.

Verify current behaviour: No VRF description is configured

Currently we don't configure any description for 'vrf definition'. You can verify the current configuration for our service

```
admin@ncs# show running-config devices device PE_00 config vrf definition
vpn10001
devices device PE_00
  config
    vrf definition vpn10001
      rd 1:10001
      route-target export 1:10001
      route-target import 1:10001
    !
  !
!
```

Update XML template

Update l3vpn-template.xml as follows:

```
<vrf xmlns="urn:ios">
  <definition>
    <name>vpn{string(..../vpn-id)}</name>
    <description>L3VPN for customer {/customer}</description>
    <rd>1:{string(..../vpn-id)}</rd>
    <route-target>
```

Step 3: BGP as an additional PE-CPE routing-protocol

In order to expand our offer of PE-CE protocols to our customers we will allow them to connect to the SP core by BGP protocol.

Verify current behaviour: Only RIP and Static are options for routing-protocol

As we show in Task 1, we only allow to configure CE-PE communication to be Static or RIP

```
admin@ncs(config)# services l3vpn ACME_01 vpn-id 10001 customer ACME link 1
link-name "Link to CE_1" device PE_00 interface 3 routing-protocol ?
Description: Routing option on PE-CE link
Possible completions:
  [static]  bgp  rip  static
admin@ncs(config)# services l3vpn ACME_01 vpn-id 10001 customer ACME link 1
link-name "Link to CE_1" device PE_00 interface 3 routing-protocol
```

Update YANG model

The variable 'routing-protocol' indicates the possible options for PE-CE protocols. This variable is of type enum, allowing only the options indicated per 'enum' line. Let's update it to allow 'bgp' as protocol.

```
leaf routing-protocol {
  tailf:info "Routing option on PE-CE link";
  type enumeration {
    enum bgp;
    enum rip;
    enum static;
  }
}
```

Update XML template

The following is an example of the PE configuration for BGP

```
router bgp 1
```



```

address-family ipv4 unicast vrf vpn10001
neighbor 172.31.1.2 remote-as 65001
neighbor 172.31.1.2 activate
neighbor 172.31.1.2 allowas-in
neighbor 172.31.1.2 as-override disable
neighbor 172.31.1.2 default-originate
redistribute connected
redistribute static
exit-address-family
!

```

In order to obtain the XML equivalent to add to our template, let's configure it into a device through NSO first.

```

admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# devices device PE_00 config
admin@ncs(config-config)# router bgp 1
admin@ncs(config-router)# address-family ipv4 unicast vrf vpn10001
admin@ncs(config-router-af)# neighbor 172.31.1.2 remote-as 65001
admin@ncs(config-router-af)# neighbor 172.31.1.2 activate
admin@ncs(config-router-af)# neighbor 172.31.1.2 allowas-in
admin@ncs(config-router-af)# neighbor 172.31.1.2 as-override disable
admin@ncs(config-router-af)# neighbor 172.31.1.2 default-originate
admin@ncs(config-router-af)# redistribute connected
admin@ncs(config-router-af)# redistribute static
admin@ncs(config-router-af)# exit-address-family
admin@ncs(config-router)#

```

Use the **commit dry-run outformat xml** command to retrieve the XML version of the configuration for the Cisco IOS

```

admin@ncs(config-router)# commit dry-run outformat xml
result-xml {
  local-node {
    data <devices xmlns="http://tail-f.com/ns/ncs">
      <device>
        <name>PE_00</name>
        <config>
          <router xmlns="urn:ios">
            <bgp>
              <as-no>1</as-no>
              <address-family>
                <with-vrf>
                  <ipv4>
                    <af>unicast</af>
                    <vrf>
                      <name>vpn10001</name>
                      <redistribute>
                        <connected/>
                        <static/>
                      </redistribute>
                      <neighbor>
                        <id>172.31.1.2</id>
                        <remote-as>65001</remote-as>
                        <activate/>
                        <allowas-in/>
                        <as-override>
                          <disable/>
                        </as-override>
                        <default-originate/>
                      </neighbor>
                    </vrf>

```

```

        </ipv4>
      </with-vrf>
    </address-family>
  </bgp>
</router>
</config>
</device>
</devices>
}
admin@ncs(config-router)#

```

The '<bgp>' section it is what we need to add to our template

Replace all the static parameters with variables, which reference service attributes according to the hierarchy of the YANG data model. And add an 'if' condition so this configuration will only be applied when 'routing-protocol' is selected to be 'bgp'.

After adding it to our l3vpn-template.xml it should look like this.

```

</interface>
<router xmlns="urn:ios">
  <?if {routing-protocol='bgp'}?>
    <bgp>
      <as-no>1</as-no>
      <address-family>
        <with-vrf>
          <ipv4>
            <af>unicast</af>
            <vrf>
              <name>vpn{string(..../vpn-id)}</name>
              <redistribute>
                <connected/>
                <static/>
              </redistribute>
              <neighbor>
                <id>172.31.{link-id}.2</id>
                <remote-as>65001</remote-as>
                <activate/>
                <allowas-in/>
                <as-override>
                  <disable/>
                </as-override>
                <default-originate/>
              </neighbor>
            </vrf>
          </ipv4>
        </with-vrf>
      </address-family>
    </bgp>
  <?end?>
  <?if {routing-protocol='rip'}?>
    <rip>

```

Step 4: Verify the new Service

First let's load the new version of our L3VPN service

1. Save the YANG and XML files
2. If you have worked on the files locally, use Filezilla to SFTP your updated package to NSO host and replace the one under /home/cisco/nso572/ncs-run/packages

3. Connect to NSO Host and go to /home/cisco/nso572/ncs-run/packages/l3vpn/src directory to compile the new l3vpn package

```
cisco@nso-572i:~/ncs-run$ cd /home/cisco/nso572/ncs-run/packages/l3vpn/src/
cisco@nso-572i:~/ncs-run/packages/l3vpn/src$
cisco@nso-572i:~/ncs-run/packages/l3vpn/src$ make clean all
rm -rf ../load-dir java/src//
mkdir -p ../load-dir
mkdir -p java/src//
/home/cisco/nso-5.7.2/bin/ncsc `ls l3vpn-ann.yang` > /dev/null 2>&1 &&
echo "-a l3vpn-ann.yang" `ls` \
    -c -o ../load-dir/l3vpn.fxs yang/l3vpn.yang
cisco@nso-572i:~/ncs-run/packages/l3vpn/src$
```

4. Login to NSO CLI and reload the packages (it will take some time)

```
admin@ncs# packages reload

>>> 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-ios-cli-6.80
  result true
}
reload-result {
  package cisco-iosxr.cli-7.39
  result true
}
reload-result {
  package l3vpn
  result true
}
admin@ncs#
System message at 2019-05-26 21:45:38...
  Subsystem stopped: ncs-dp-2-cisco-ios:IOSDp
admin@ncs#
System message at 2019-05-26 21:45:38...
  Subsystem started: ncs-dp-3-cisco-ios:IOSDp
admin@ncs#
```

After the package is successfully compiled and reloaded let's verify the new behaviour by configuring an instance. You can do this both by GUI and CLI. Here we will continue with CLI mode.

5. Verify that the vpn-id is only allowed between 10001 and 19999

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# services l3vpn ACME_02 vpn-id ?
Description: Service Instance ID
Possible completions:
  <unsignedInt, 10001 .. 19999>
admin@ncs(config)# services l3vpn ACME_02 vpn-id 1234
-----^
syntax error: "1234" is out of range.
admin@ncs(config)# services l3vpn ACME_02 vpn-id
```

6. Verify that 'bgp' protocol is allowed

```
admin@ncs(config)# services l3vpn ACME_02 vpn-id 10001 customer ACME
link 1 link-name "Link to CE_1" device PE_01 interface 2 routing-
protocol ?
Description: Routing option on PE-CE link
```

```

    bgp rip static
admin@ncs(config)# services l3vpn ACME_02 vpn-id 10001 customer ACME
link 1 link-name "Link to CE_1" device PE_01 interface 2 routing-
protocol bgp

```

- ```

admin@ncs(config)# services l3vpn ACME_02 vpn-id 10001 customer ACME
link 1 link-name "Link to CE_1" device PE_01 interface 2 routing-
protocol bgp
admin@ncs(config-link-1)# commit dry-run
cli {
 local-node {
 data devices {
 device PE_01 {
 config {
 vrf {
 +
 + definition vpn10001 {
 + + description "L3VPN for customer ACME";
 + + rd 1:10001;
 + + route-target {
 + + export 1:10001;
 + + import 1:10001;
 + + }
 + + }
 }
 interface {
 +
 + GigabitEthernet 2 {
 + + description "Connection to Customer
ACME - Site 5";
 + + vrf {
 + + forwarding vpn10001;
 + + }
 + + ip {
 + + address {
 + + primary {
 + + address 172.31.1.1;
 + + mask 255.255.255.252;
 + + }
 + + }
 + + }
 }
 router {
 +
 + bgp 1 {
 + + address-family {
 + + with-vrf {
 + + ipv4 unicast {
 + + vrf vpn10001 {
 + + redistribute {
 + + connected {
 + + }
 + + static {
 + + }
 + + }
 + + }
 + + neighbor 172.31.1.2 {
 + + remote-as 65001;
 + + activate;
 + + allowas-in {
 + + }
 + + as-override {
 + + disable;
 + + }
 + + }
 + + }
 + + }
 + + }
 }
 }
 }
 }
 }
}

```

```
+ default-originate
{
+
+ }
+ }
+ }
+ }
+ }
+ }
+ }
 services {
+ l3vpn ACME_02 {
+ vpn-id 10001;
+ customer ACME;
+ link 1 {
+ link-name "Link to CE_1";
+ device PE_01;
+ interface 2;
+ routing-protocol bgp;
+ }
+ }
}

admin@ncs(config-link-1)#
```

8. Verify that the description for the VRF is present and the router bgp configuration
9. You can connect to the simulated devices from NSO host by running the following command, to verify that we don't have this yet configured (remember that it is a simulated device and most of the show commands won't be available)

```
cisco@nso-572i:~$ cd /home/cisco/nso572/ncs-run
cisco@nso-572i:~/ncs-run$ ncs-netsim cli-c PE_01

admin connected from 198.18.133.252 using ssh on nso-572i
PE_01# show running-config | begin router
PE_01#
```

- ## 10. Commit the service

```
admin@ncs(config-link-1)# commit
Commit complete.
admin@ncs(config-link-1)#
```

11. Verify that the device configuration was pushed

```
PE_01# show running-config | begin router
router bgp 1
 address-family ipv4 unicast vrf vpn10001
 redistribute connected
 redistribute static
 neighbor 172.31.1.2 remote-as 65001
 neighbor 172.31.1.2 activate
 neighbor 172.31.1.2 allowas-in
 neighbor 172.31.1.2 as-override disable
 neighbor 172.31.1.2 default-originate
 exit-address-family
!
!
```



### Task 3: Add support for a new PE device type

We are upgrading some of our PE devices to ASR9K running IOS XR software. NSO L3VPN service should be able to know the vendor and configure it correctly without Operator intervention.

The following is the example configuration to use for the template

```
vrf vpn10001
 description Customer ACME VPN
 address-family ipv4 unicast
 import route-target
 1:10001
 exit
 export route-target
 1:10001
 exit
exit
interface GigabitEthernet 0/0/0/1
 description Connection to Customer ACME - Site 9
 ipv4 address 172.31.1.1 255.255.255.252
 vrf vpn10001
exit
router static
 address-family ipv4 unicast
 192.168.21.0/24 GigabitEthernet0/0/0/1 172.31.1.2
 exit
exit
router bgp 1
 vrf vpn10001
 rd 1:10001
 address-family ipv4 unicast
 redistribute connected
 redistribute static
 exit
 neighbor 172.31.1.2
 address-family ipv4 unicast
 route-policy in
 as-override
 default-originate
 exit
 exit
 exit
exit
exit
```

#### Step 1: Create XML Template for new device type

Let's get the XML template for IOS XR router by configuring it into a device and use "display xml" NSO function.

To add the configuration into NSO, you can:

- Add by hand all the configuration below line by line, which it is not practical.
- Paste it directly at once but notice that NSO does not like the spaces in front of the commands. Additionally, this method is not very reliable.
- Load from terminal using 'load merge terminal' command to be able to copy/paste/load all the configuration at once.
- Load from file using 'load merge <file>' where <file> is the relative path to a file local in NSO.

Here we will use 'load merge terminal method'.

1. Go into NSO CLI configuration mode
2. Run 'load merge terminal' command

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# load merge terminal
Loading.
```

3. After that paste all the configurations found below

```
devices device PE_10
config
 vrf vpn10001
 description Customer ACME VPN
 address-family ipv4 unicast
 import route-target
 1:10001
 exit
 export route-target
 1:10001
 exit
 exit
exit
interface GigabitEthernet 0/0/0/1
 description Connection to Customer ACME - Site 9
 vrf vpn10001
 ipv4 address 172.31.1.1 255.255.255.252
 no shutdown
exit
route-policy PASS
 end-policy
!
router static
 address-family ipv4 unicast
 192.168.21.0/24 GigabitEthernet0/0/0/1 172.31.1.2
 exit
exit
router bgp 1
 vrf vpn10001
 rd 1:10001
 address-family ipv4 unicast
 redistribute connected
 redistribute static
 exit
 neighbor 172.31.1.2
 address-family ipv4 unicast
 route-policy PASS in
 as-override
 default-originate
 exit
 exit
exit
exit
!
```

4. Make sure you are in a new line or press enter and then Ctrl+D to exit the insert mode. You can then verify that all the configuration went in by using command 'show config'
5. Let's do a Dry-run now forcing NSO to display the configuration that would be pushed into the device in XML format

```
admin@ncs(config)# show configuration | display xml
```



```

<devices xmlns="http://tail-f.com/ns/ncs">
 <device>
 <name>PE_10</name>
 <config>
 <vrf xmlns="http://tail-f.com/ned/cisco-ios-xr">
 <vrf-list>
 <name>vpn10001</name>
 <description>Customer ACME VPN</description>
 <address-family>
 <ipv4>
 <unicast>
 <import>
 <route-target>
 <address-list>
 <name>1:10001</name>
 </address-list>
 </route-target>
 </import>
 <export>
 <route-target>
 <address-list>
 <name>1:10001</name>
 </address-list>
 </route-target>
 </export>
 </unicast>
 </ipv4>
 </address-family>
 </vrf-list>
 </vrf>
 <interface xmlns="http://tail-f.com/ned/cisco-ios-xr">
 <GigabitEthernet>
 <id>0/0/0/1</id>
 <description>Connection to Customer ACME - Site
9</description>
 <vrf>vpn10001</vrf>
 <ipv4>
 <address>
 <ip>172.31.1.1</ip>
 <mask>255.255.255.252</mask>
 </address>
 </ipv4>
 </GigabitEthernet>
 </interface>
 <route-policy xmlns="http://tail-f.com/ned/cisco-ios-xr">
 <name>PASS</name>
 </route-policy>
 <router xmlns="http://tail-f.com/ned/cisco-ios-xr">
 <static>
 <address-family>
 <ipv4>
 <unicast>
 <routes>
 <net>192.168.21.0/24</net>
 <interface>GigabitEthernet0/0/0/1</interface>
 <address>172.31.1.2</address>
 </routes>
 </unicast>
 </ipv4>
 </address-family>
 </static>
 <bgp>
 <bgp-no-instance>
 <id>1</id>

```

```

 <vrf>
 <name>vpn10001</name>
 <rd>1:10001</rd>
 <address-family>
 <ipv4>
 <unicast>
 <redistribute>
 <connected/>
 <static/>
 </redistribute>
 </unicast>
 </ipv4>
 </address-family>
 </neighbor>
 <id>172.31.1.2</id>
 <address-family>
 <ipv4>
 <unicast>
 <route-policy>
 <direction>in</direction>
 <name>PASS</name>
 </route-policy>
 <as-override/>
 <default-originate/>
 </unicast>
 </ipv4>
 </address-family>
 </neighbor>
</vrf>
</bgp-no-instance>
</bgp>
</router>
</config>
</device>
</devices>
admin@ncs (config) #

```

Notice that to assign a route-policy it must be created first, so we have added it only temporarily, it will not appear in our template

We could have both IOS and IOS XR templates in a single XML file. As described before 'xmlns="urn:ios"' in some of the template lines indicate this section is exclusive for IOS devices and it will be ignored for non-IOS devices. In the same manner 'xmlns=<http://tail-f.com/ned/cisco-ios-xr>' indicate lines exclusive of IOS XR devices.

Nevertheless, when a template starts to be bigger and contain mix of device types, configurations and so on, it gets harder to maintain and update. That's why in this exercise we will separate it in two templates and we will leave up to python code to decide when to call one or the other.

Let's rename our previous template 'l3vpn-ios-template.xml' and create a new one (you can copy previous one and replace the '<config>' section) called l3vpn-iosxr-template.xml.

It is time to parametrize the new template:

```

<config-template xmlns="http://tail-f.com/ns/config/1.0">
 <devices xmlns="http://tail-f.com/ns/ncs">
 <device>
 <name>{/link/device}</name>
 <config>
 <vrf xmlns="http://tail-f.com/ned/cisco-ios-xr">
 <vrf-list>
 <name>vpn{string(..vpn-id)}</name>
 <description>L3 VPN for customer
{/customer}</description>

```

```

 <address-family>
 <ipv4>
 <unicast>
 <import>
 <route-target>
 <address-list>
 <name>1:{string(..../vpn-id)}</name>
 </address-list>
 </route-target>
 </import>
 <export>
 <route-target>
 <address-list>
 <name>1:{string(..../vpn-id)}</name>
 </address-list>
 </route-target>
 </export>
 </unicast>
 </ipv4>
 </address-family>
 </vrf-list>
 </vrf>
 <interface xmlns="http://tail-f.com/ned/cisco-ios-xr">
 <GigabitEthernet>
 <id>{interface}</id>
 <description>Connection to Customer ACME - Site
9</description>

 <vrf>vpn{string(..../vpn-id)}</vrf>
 <ipv4>
 <address>
 <ip>172.31.{link-id}.1</ip>
 <mask>255.255.255.252</mask>
 </address>
 </ipv4>
 </GigabitEthernet>
 </interface>
 <router xmlns="http://tail-f.com/ned/cisco-ios-xr">
 <?if {routing-protocol='static'}?>
 <static>
 <address-family>
 <ipv4>
 <unicast>
 <routes>
 <net>{string(static/prefix)}</net>

<interface>GigabitEthernet{interface}</interface>
 <address>172.31.{string(link-
id)}.2</address>

 </routes>
 </unicast>
 </ipv4>
 </address-family>
 </static>
 <?end?>
 <?if {routing-protocol='bgp'}?>
 <bgp>
 <bgp-no-instance>
 <id>1</id>
 <vrf>
 <name>vpn{string(..../vpn-id)}</name>
 <rd>1:{string(..../vpn-id)}</rd>
 <address-family>
 <ipv4>
 <unicast>

```

```

 <redistribute>
 <connected/>
 <static/>
 </redistribute>
 </unicast>
</ipv4>
</address-family>
<neighbor>
 <id>172.31.{link-id}.2</id>
 <address-family>
 <ipv4>
 <unicast>
 <route-policy>
 <direction>in</direction>
 <name>PASS</name>
 </route-policy>
 <as-override/>
 <default-originate/>
 </unicast>
 </ipv4>
 </address-family>
</neighbor>
</vrf>
</bgp-no-instance>
</bgp>
<?end?>
</router>
</config>
</device>
</devices>
</config-template>

```

## Step 2: Update Mapping with Python

As we have two different templates for IOS and IOS XR devices, inside the python main Service Callback we will have to find out for the selected the device the platform and then apply the correct template.

Open for edit your main.py (ncs-run/packages/l3vpn/python/l3vpn/main.py).

When a new service creation calls the class 'ServiceCallbacks' all parameters introduced by user as passed to the db\_create function inside argument 'service'. Additionally, a pointer to the root path of our database is passed to this function, allowing us to get any element from inside it.

The following code:

- iterates through all links configured for our 'service' (each link has a device associated, that can be IOS or IOS XR)
- gets the 'device' name from the link and identifies what is the device type. The possible values for 'device\_type' are:
- Cisco IOS: 'ios-id:cisco-ios'
- Cisco IOS XR: 'cisco-ios-xr-id:cisco-ios-xr'
- Depending on the device type the appropriate template is applied

```

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

 vars = ncs.template.Variables()

```

```

vars.add('DUMMY', '127.0.0.1')
template = ncs.template.Template(service)

for link in service.link:
 device_type =
root.devices.device[link.device].device_type.cli.ned_id
 if device_type == 'ios-id:cisco-ios':
 template.apply('l3vpn-ios-template', vars)
 elif device_type == 'cisco-ios-xr-id:cisco-ios-xr':
 template.apply('l3vpn-iosxr-template', vars)

```

### Step 3: Verify the new Service

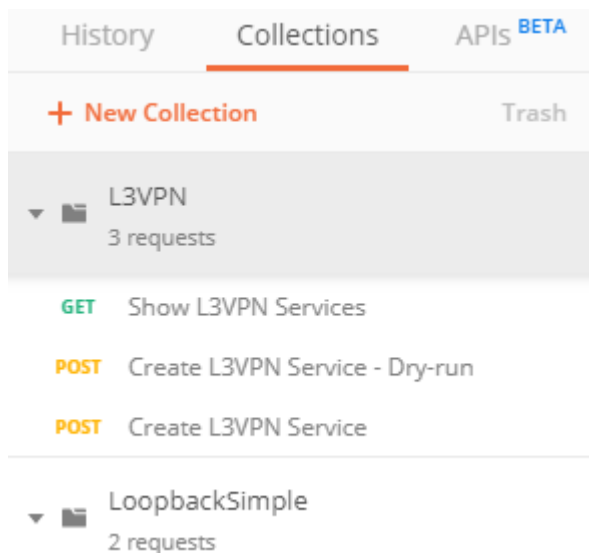
First let's load the new version of our L3VPN service

1. Save the Python and XML files
2. Follow the same steps 2-4 from Task 2 Step 4: Verify the new Service

After compile the package and reload it let's configure a new service for IOS XR device.

During the exercises above we saw how we can interact with NSO through CLI and GUI, but generally NSO will have to integrate with other applications and tools, or with Operations and Business portals (OSS, BSS). NSO provides several APIs to integrate among REST, Restconf and Netconf. For the next verification we will guide you to use Restconf to configure a new service and see its configuration.

1. Open Postman application from the Desktop
2. You will find on the left panel a collection of already prepared calls to interact with NSO L3VPN service called 'L3VPN'. Click the arrow to expand it



3. Open 'Show L3VPN Services' This Restconf call request the configuration of all configured L3VPN services (as indicated by the URI). Click send button.
4. If you left some of the previous services you will be able to see in bottom section of the screen the response from NSO
5. We will now create a new service instance on an IOS XR device, PE\_10, open the call 'Create L3VPN Service – Dry-run'

6. In the tab called 'Body' you can see already a payload with the configuration that will be pushed to NSO. Review it and update it if desired.
7. Notice that this call performs a 'dry-run', that means that we will see the configuration that would be pushed into the devices, but nothing will be configured. This is very useful to review what will be configured in advance.
8. Click the Send button and wait for the 201 response. It should look like this:

```
{
 "dry-run-result": {
 "native": {
 "device": [
 {
 "name": "PE_01",
 "data": "\nvrf definition vpn10010\n description\n\"L3VPN for customer ACME\"\n\n rd 1:10010\n route-target export\n1:10010\n route-target import 1:10010\n!\ninterface GigabitEthernet2\n description \"Connection to Customer ACME - Site 5\"\n vrf forwarding\nvpn10010\n no switchport\n ip address 172.31.2.1 255.255.255.252\n no\nshutdown\n\nexit\n\nrouter bgp 1\n address-family ipv4 unicast vrf\nvpn10010\n redistribute connected\n redistribute static\n neighbor\n172.31.2.2 remote-as 65001\n neighbor 172.31.2.2 activate\n neighbor\n172.31.2.2 allowas-in\n neighbor 172.31.2.2 as-override disable\n\nneighbor 172.31.2.2 default-originate\n exit-address-family\n !\n!\n\"
 },
 {
 "name": "PE_10",
 "data": "\nvrf vpn10010\n description \"L3 VPN for\n customer ACME\"\n\n address-family ipv4 unicast\n import route-target\n1:10010\n exit\n export route-target\n 1:10010\n exit\n\nexit\n\nexit\n\ninterface GigabitEthernet 2\n description \"Connection to\n Customer ACME - Site 9\"\n vrf\n vpn10010\n ipv4 address\n172.31.1.1 255.255.255.252\n no shutdown\n\nexit\n\nrouter bgp 1\n vrf\nvpn10010\n rd 1:10010\n address-family ipv4 unicast\n redistribute\nconnected\n redistribute static\n exit\n neighbor 172.31.1.2\n\naddress-family ipv4 unicast\n route-policy PASS in\n as-override\n\n default-originate\n exit\n exit\n exit\n\nexit\n\nexit\n\n\"
 }
]
 }
 }
}
```

9. You can see how we would configure both devices and each of them has the configuration specific for their device types IOS and IOS XR.
10. Doing the same from cli we can see a more readable output

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# load merge terminal
Loading.

services l3vpn ACME_10
vpn-id 10010
customer EMCA
link 1
link-name "Link to CE_0"
device PE_10
interface 2
routing-protocol bgp
!
link 2
link-name "Link to CE_1"
```

```

device PE_01
interface 2
routing-protocol bgp
!
!
admin@ncs(config)# commit dry-run outformat native
native {
 device {
 name PE_01
 data vrf definition vpn10010
 description "L3VPN for customer EMCA"
 rd 1:10010
 route-target export 1:10010
 route-target import 1:10010
 !
 interface GigabitEthernet2
 description "Connection to Customer ACME - Site 5"
 vrf forwarding vpn10010
 no switchport
 ip address 172.31.2.1 255.255.255.252
 no shutdown
 exit
 router bgp 1
 address-family ipv4 unicast vrf vpn10010
 redistribute connected
 redistribute static
 neighbor 172.31.2.2 remote-as 65001
 neighbor 172.31.2.2 activate
 neighbor 172.31.2.2 allowas-in
 neighbor 172.31.2.2 as-override disable
 neighbor 172.31.2.2 default-originate
 exit-address-family
 !
 !
 }
 device {
 name PE_10
 data vrf vpn10010
 description "L3 VPN for customer EMCA"
 address-family ipv4 unicast
 import route-target
 1:10010
 exit
 export route-target
 1:10010
 exit
 exit
 exit
 interface GigabitEthernet 2
 description "Connection to Customer ACME - Site 9"
 vrf vpn10010
 ipv4 address 172.31.1.1 255.255.255.252
 no shutdown
 exit
 router bgp 1
 vrf vpn10010
 rd 1:10010
 address-family ipv4 unicast
 redistribute connected
 redistribute static
 exit
 neighbor 172.31.1.2
 address-family ipv4 unicast
 route-policy PASS in

```

```
 as-override
 default-originate
 exit
exit
exit
exit
}
}
admin@ncs (config) #
```

11. You could choose to commit the configuration from CLI, but let's use Postman instead. Send the call "Create L3VPN Service"
12. After getting the "Status: 201 Created" you can send again "Show L3VPN Services" to see the configured service.



## Next Steps

This Lab guide works with two simple use cases, nevertheless these services could keep expanding, adding more options, device types and simplify the service provisioning by many ways, like:

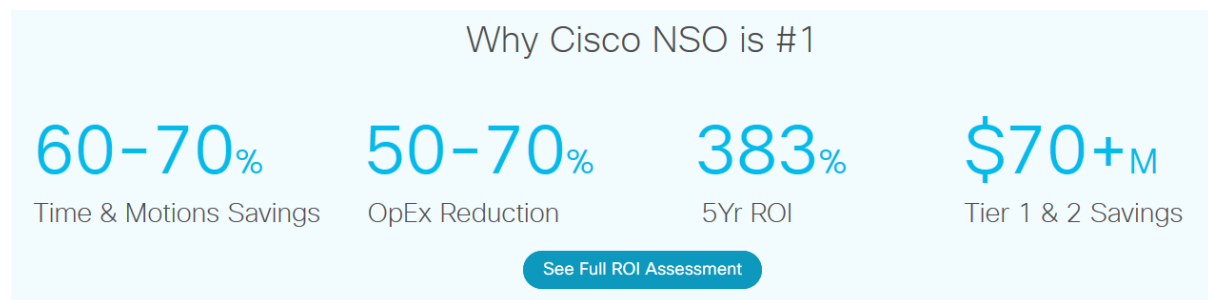
- Having a Topology service with the information about all nodes and parameters specific to them to avoid the operator having to introduce them.
- Configure many sites at the same time
- Use local or remote Resource Managers to administrate used VLAN, VRF, IP networks...
- Integrate with a northbound provisioning portal, or offer some self-service portal for customers to modify some parameters

For most of the complex services that we can automate in our networks the creation of an instance does not happen in a single step. It often requires several teams involved providing input, approving phases, testing and so on. NSO can play a key role in these environments and integrate with Workflow Managers for reducing the time between steps, that often is costlier to companies than the time spent configuring the devices.

## Conclusion

Deliver high-quality services faster and more easily through network automation. Cisco Network Services Orchestrator (NSO) is industry-leading software for automating services across traditional and virtualized networks. Use NSO to add, change, and delete services without disrupting overall service, and help ensure that services are delivered in real time.

NSO is now [free to download](#) for non-production use! Download NSO to evaluate and learn how to automate your network and orchestrate your services using NETCONF and YANG today.



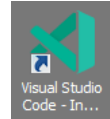
Ref.: <https://www.cisco.com/c/en/us/solutions/service-provider/solutions-cloud-providers/network-services-orchestrator-solutions.html>

## Appendix A: Use Visual Studio Code – Insiders

Notice that we connect to a Windows Host, but Cisco NSO software runs in a remote server reachable through IP 198.18.134.28.

You can find the initial packages and other required files locally in Windows under “C:\dcloud\HOLOPS-1806” (reachable through shortcut in Desktop) and as well inside NSO Host under “/home/cisco/nso572/ncs-run”.

To avoid having to edit every file locally and then upload it to NSO host using SFTP (Filezilla configured for it) the lab comes with ‘Visual Studio Code – Insiders’ installed (available from the Desktop) with a plugin that automatically connects to NSO and allows you to work with the remote files locally.



When you start ‘Visual Studio Code – Insiders’ you can verify that you are connected to NSO by looking at the bottom left corner for ‘SSH: NSO-Host’. You should see as well in the left panel the /home/cisco directory with all the main files for NSO in directory /home/cisco/nso572/ncs-run. From there you can view, edit and save all the required files

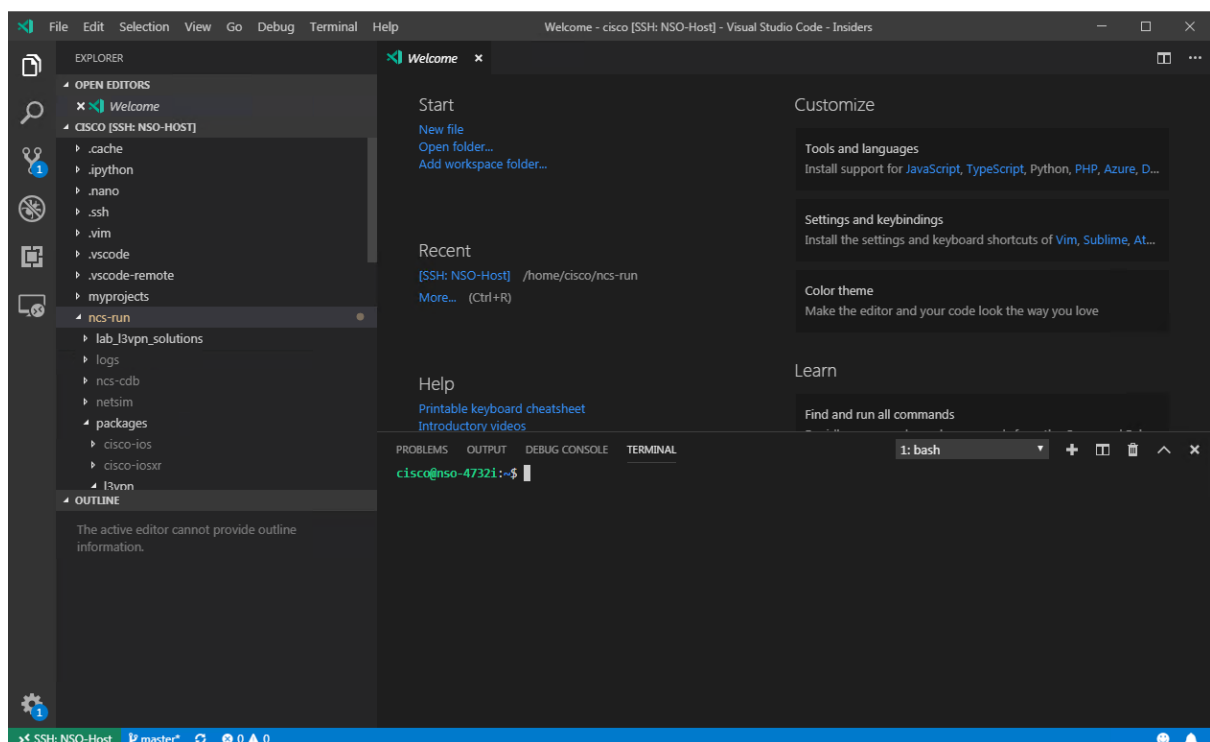


To open a Terminal to NSO Host go to Terminal > New Terminal.

From the terminal you can run the required commands in NSO Host. Or you can run the following command to connect to NSO CLI

```
cisco@nso-572i:~/ncs-run$ ncs_cli -C -u admin
```

Where ‘-C’ is for Cisco CLI mode



**NOTE:** Do not confuse ‘Visual Studio Code – Insiders’ with ‘Visual Studio Code’, that it is as well installed in Windows, but doesn’t have a plugin to edit remote files

## Appendix B: Install NSO

This section will help you understand how to install NSO Software in your laptop/machine for development and learning purpose. The installation is called local-install.

For production environment NSO Software installation should be done on a server with recommended specification. This installation is called system-install.

Download free NSO: <https://developer.cisco.com/docs/nso/#!getting-nso/getting-nso>

Note: At the time of writing this section NSO 5.2.1 was available in the above link.

In this lab you can find it downloaded in NSO under /home/cisco/Downloads

Note: Make sure the user of the system has root privileges. If not then use command with sudo in case of any issues.

### Install NSO 5.2.1

1. Extract the installation file:

```
developer@nso-host:~/Downloads$ sh nso-5.2.1.linux.x86_64.signed.bin
Unpacking...
Verifying signature...
Downloading CA certificate from
http://www.cisco.com/security/pki/certs/crcam2.cer ...
Successfully downloaded and verified crcam2.cer.
Downloading SubCA certificate from
http://www.cisco.com/security/pki/certs/innerspace.cer ...
Successfully downloaded and verified innerspace.cer.
Successfully verified root, subca and end-entity certificate chain.
Successfully fetched a public key from tailf.cer.
Successfully verified the signature of nso-
5.2.1.linux.x86_64.installer.bin using tailf.cer
nso@ubuntu:~/Downloads$
```

2. Install NSO

```
developer@nso-host:~/Downloads$ sh nso-5.2.1.linux.x86_64.installer.bin
/home/cisco/nso521/nso521 --local-install
INFO Using temporary directory /tmp/ncs_installer.11530 to stage NCS
installation bundle
INFO Unpacked ncs-5.2.1 in /home/cisco/nso521/nso521/
INFO Found and unpacked corresponding DOCUMENTATION_PACKAGE
INFO Found and unpacked corresponding EXAMPLE_PACKAGE
INFO Generating default SSH hostkey (this may take some time)
INFO SSH hostkey generated
INFO Environment set-up generated in /home/cisco/nso521/nso521//ncsrc
INFO NCS installation script finished
INFO Found and unpacked corresponding NETSIM_PACKAGE
INFO NCS installation complete
```

3. Source the binaries:

```
developer@nso-host:~/Downloads$ cd ..
developer@nso-host:~$ source nso521/nso521/ncsrc
```

This sets up paths and environment variables in order to run NSO.

As this must be done before running NSO. It is recommended to put it in your profile.

4. Create running directory:

```
developer@nso-host:~$ ncs-setup --dest nso521/ncs-run-521
```

The runtime directory should look similar to below:

```
developer@nso-host:~/nso521/ncs-run-521$ ls -lrt
total 36
drwxrwxr-x 2 developer developer 4096 Nov 27 03:05 state
drwxrwxr-x 2 developer developer 4096 Nov 27 03:05 packages
drwxrwxr-x 2 developer developer 4096 Nov 27 03:05 ncs-cdb
drwxrwxr-x 2 developer developer 4096 Nov 27 03:05 logs
drwxrwxr-x 4 developer developer 4096 Nov 27 03:05 scripts
-rw-rw-r-- 1 developer developer 9951 Nov 27 03:05 ncs.conf
-rw-rw-r-- 1 developer developer 636 Nov 27 03:05 README.ncs
```

## Install the NEDs

1. Link NED packages from install directory to running directory (in our lab this is already done):

```
nso@ubuntu:~/ncs-run-5.2.1$ cd ../Downloads/
nso@ubuntu:~/Downloads$ ls -lrt
total 449576
-rwxr-xr-x 1 nso nso 183649615 Nov 30 09:13 nso-5.2.1.linux.x86_64.installer.bin
-rw-r--r-- 1 nso nso 1383 Nov 30 09:13 tailf.cer
-rw-r--r-- 1 nso nso 10696 Nov 30 09:13 cisco_x509_verify_release.py
-rw-r--r-- 1 nso nso 256 Nov 30 09:13 nso-5.2.1.linux.x86_64.installer.bin.signature
-rw-r--r-- 1 nso nso 1822 Nov 30 09:13 README.signature
-rw-rw-r-- 1 nso nso 183661414 Jan 19 02:15 nso-5.2.1.linux.x86_64.signed.bin
-rw-rw-r-- 1 nso nso 32476407 Jan 19 02:17 ncs-5.2.1-cisco-ios-6.39.signed.bin
-rw-rw-r-- 1 nso nso 24417815 Jan 19 02:17 ncs-5.2.1-cisco-iosxr-7.17.signed.bin
-rw-rw-r-- 1 nso nso 36116185 Jan 19 02:18 ncs-5.2.1-cisco-nx-5.13.signed.bin
nso@ubuntu:~/Downloads$
```

2. Extract the NEDs:

```
developer@nso-host:~/Downloads$ sh ncs-5.2.1-cisco-ios-6.39.signed.bin
&& sh ncs-5.2.1-cisco-iosxr-7.17.signed.bin && sh ncs-5.2.1-cisco-nx-
5.13.signed.bin
developer@nso-host:~/Downloads$ ls -lrt
total 419972
-rw-r--r-- 1 developer developer 1383 May 6 2019 tailf.cer
-rw-r--r-- 1 developer developer 12381 May 6 2019 cisco_x509_verify_release.py
-rw-r--r-- 1 developer developer 36267747 May 16 2019 ncs-5.2.1-cisco-iosxr-7.17.tar.gz
-rw-r--r-- 1 developer developer 39632795 May 16 2019 ncs-5.2.1-cisco-nx-5.13.tar.gz
-rw-r--r-- 1 developer developer 46103905 May 16 2019 ncs-5.2.1-cisco-ios-6.39.tar.gz
-rw-r--r-- 1 developer developer 256 May 16 2019 ncs-5.2.1-cisco-iosxr-
7.17.tar.gz.signature
-rw-r--r-- 1 developer developer 256 May 17 2019 ncs-5.2.1-cisco-ios-
6.39.tar.gz.signature
-rw-r--r-- 1 developer developer 256 May 17 2019 ncs-5.2.1-cisco-nx-
5.13.tar.gz.signature
-rw-r--r-- 1 developer developer 1832 May 17 2019 README.signature
-rwxrwxrwx 1 developer developer 186085011 Nov 27 02:38 nso-
5.2.1.linux.x86_64.installer.bin
-rw-r--r-- 1 developer developer 46077897 Nov 27 03:33 ncs-5.2.1-cisco-ios-
6.39.signed.bin
-rw-r--r-- 1 developer developer 36256645 Nov 27 03:40 ncs-5.2.1-cisco-iosxr-
7.17.signed.bin
-rw-r--r-- 1 developer developer 39577109 Nov 27 05:08 ncs-5.2.1-cisco-nx-5.13.signed.bin
```

3. Untar the NEDs:

```
developer@nso-host:~/Downloads$ tar -xvf ncs-5.2.1-cisco-iosxr-
7.17.tar.gz && tar -xvf ncs-5.2.1-cisco-nx-5.13.tar.gz && tar -xvf ncs-
5.2.1-cisco-ios-6.39.tar.gz
```

```
developer@nso-host:~/Downloads$ ls -lrt
total 419984
-rw-r--r-- 1 developer developer 1383 May 6 2019 tailf.cer
-rw-r--r-- 1 developer developer 12381 May 6 2019 cisco_x509_verify_release.py
-rw-r--r-- 1 developer developer 36267747 May 16 2019 ncs-5.2.1-cisco-iosxr-7.17.tar.gz
drwxr-xr-x 9 developer developer 4096 May 16 2019 cisco-iosxr-cli-7.17
-rw-r--r-- 1 developer developer 39632795 May 16 2019 ncs-5.2.1-cisco-nx-5.13.tar.gz
-rw-r--r-- 1 developer developer 46103905 May 16 2019 ncs-5.2.1-cisco-ios-6.39.tar.gz
drwxr-xr-x 6 developer developer 4096 May 16 2019 cisco-nx-cli-5.13
drwxr-xr-x 8 developer developer 4096 May 16 2019 cisco-ios-cli-6.39
-rw-r--r-- 1 developer developer 256 May 16 2019 ncs-5.2.1-cisco-iosxr-
7.17.tar.gz.signature
```

```
-rw-r--r-- 1 developer developer 256 May 17 2019 ncs-5.2.1-cisco-ios-6.39.tar.gz.signature
-rw-r--r-- 1 developer developer 256 May 17 2019 ncs-5.2.1-cisco-nx-5.13.tar.gz.signature
-rw-r--r-- 1 developer developer 1832 May 17 2019 README.signature
-rwxrwxrwx 1 developer developer 186085011 Nov 27 02:38 nso-5.2.1.linux.x86_64.installer.bin
-rw-r--r-- 1 developer developer 46077897 Nov 27 03:33 ncs-5.2.1-cisco-ios-6.39.signed.bin
-rw-r--r-- 1 developer developer 36256645 Nov 27 03:40 ncs-5.2.1-cisco-iosxr-7.17.signed.bin
-rw-r--r-- 1 developer developer 39577109 Nov 27 05:08 ncs-5.2.1-cisco-nx-5.13.signed.bin
developer@nso-host:~/Downloads$
nso@ubuntu:~/Downloads$
```

4. Link the NEDs to NSO runtime directory packages folder:

```
developer@nso-host:~/nso521$ cd ncs-run-521/packages/
developer@nso-host:~/nso521/ncs-run-521/packages$ ln -s
~/Downloads/cisco-iosxr-cli-7.17/ /home/cisco/nso521/ncs-run-521/packages/
developer@nso-host:~/nso521/ncs-run-521/packages$ ln -s
~/Downloads/cisco-ios-cli-6.39/ /home/cisco/nso521/ncs-run-521/packages/
developer@nso-host:~/nso521/ncs-run-521/packages$ ln -s
~/Downloads/cisco-nx-cli-5.13/ /home/cisco/nso521/ncs-run-521/packages/
```

## Start NSO and verify status

1. Start NSO from within running-directory:

```
developer@nso-host:~/nso521/ncs-run-521/packages$ cd
/home/cisco/nso521/ncs-run-521/
developer@nso-host:~/nso521/ncs-run-521$ ncs
```

2. Verify NSO status:

```
developer@nso-host:~/nso521/ncs-run-521$ ncs --status
vsn: 5.2.1
SMP support: yes, using 4 threads
Using epoll: yes
available modules: backplane,netconf,cdb,cli,snmp,webui
running modules: backplane,netconf,cdb,cli,snmp,webui
status: started
...
```

3. Check the ncs.conf file for default NSO configuration parameters:

```
developer@nso-host:~/nso521/ncs-run-521$ cat ncs.conf

<!-- -*- nxml -*- -->
<!-- Example configuration file for ncs. -->

<ncs-config xmlns="http://tail-f.com/yang/tailf-ncs-config">

 <!-- NCS can be configured to restrict access for incoming connections
 -->
 <!-- to the IPC listener sockets. The access check requires that -->
 <!-- connecting clients prove possession of a shared secret. -->
 <ncs-ipc-access-check>
 <enabled>false</enabled>
 <filename>${NCS_DIR}/etc/ncs/ipc_access</filename>
 </ncs-ipc-access-check>

 <!-- Where to look for .fxs and snmp .bin files to load -->
 ...
```

4. Login to NSO CLI Juniper mode (default password for user admin is admin):

```
developer@nso-host:~/nso521/ncs-run-521$ ncs_cli -u admin
```

```
admin connected from 192.168.234.3 using ssh on nso-host
admin@ncs>
```

OR

```
developer@nso-host:~/nso521/ncs-run-521$ ssh -l admin -p 2024 localhost
The authenticity of host '[localhost]:2024 ([127.0.0.1]:2024)' can't be
established.
RSA key fingerprint is
SHA256:ZLWvfBSWDj4yqS1a68ZpTT4nTVsrrCC8CVTB1DPJu00.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:2024' (RSA) to the list of known
hosts.
admin@localhost's password:

admin connected from 127.0.0.1 using ssh on nso-host
admin@ncs>
```

##### 5. Switch to Cisco style CLI:

```
admin@ncs> switch cli
admin@ncs# << Cisco style CLI>>
admin@ncs#

admin@ncs#
admin@ncs# switch cli
[ok][2019-11-27 05:38:02]
admin@ncs> << Juniper Style>>
admin@ncs>
admin@ncs>
```

OR

```
Cisco Style CLI

developer@nso-host:~/nso521/ncs-run-521$ ncs_cli -u admin -C

admin connected from 192.168.234.3 using ssh on nso-host
admin@ncs#

Juniper Style CLI

developer@nso-host:~/nso521/ncs-run-521$ ncs_cli -u admin -J

admin connected from 192.168.234.3 using ssh on nso-host
admin@ncs>
```

##### 6. Explore different show commands from NSO CLI

```
show packages packages package-version
show devices list
show running-config
```

##### 7. Observe NSO startup process through different logs: ncs-java-vm.log

```
developer@nso-host:~/nso521/ncs-run-521$
developer@nso-host:~/nso521/ncs-run-521$ cd logs/
developer@nso-host:~/nso521/ncs-run-521/logs$ ls -lrt
total 240
-rw-rw-r-- 1 developer developer 0 Nov 27 05:30 netconf.log
-rw-rw-r-- 1 developer developer 0 Nov 27 05:30 snmp.log
-rw-rw-r-- 1 developer developer 13 Nov 27 05:30 ncserr.log.siz
-rw-rw-r-- 1 developer developer 18 Nov 27 05:30 ncserr.log.idx
-rw-rw-r-- 1 developer developer 8 Nov 27 05:30 ncserr.log.1
-rw-rw-r-- 1 developer developer 0 Nov 27 05:31 ncs-python-vm.log
-rw-rw-r-- 1 developer developer 826 Nov 27 05:31 rollback10001
```

```
-rw-rw-r-- 1 developer developer 0 Nov 27 05:31
localhost:8080.access
-rw-rw-r-- 1 developer developer 7274 Nov 27 05:31 ncs-java-vm.log
-rw-rw-r-- 1 developer developer 21401 Nov 27 05:32 ncs.log
-rw-rw-r-- 1 developer developer 53451 Nov 27 05:40 xpath.trace
-rw-rw-r-- 1 developer developer 127736 Nov 27 05:40 devel.log
-rw-rw-r-- 1 developer developer 6097 Nov 27 05:40 audit.log
developer@nso-host:~/nso521/ncs-run-521/logs$
```

## Appendix C: Manage Netsim Devices

NSO already has 6 simulated devices (called Netsim devices across the lab guide) configured and ready to start from `/home/cisco/nso572/ncs-run` directory.

During Lab Introduction and Verification section you are asked to start them by running command 'ncs-netsim start' from that directory. If everything goes well, you will see an output like the following:

```
cisco@ubuntu:~/nso572/ncs-run$ ncs-netsim start
DEVICE PE_00 OK STARTED
DEVICE PE_01 OK STARTED
DEVICE PE_10 OK STARTED
DEVICE PE_11 OK STARTED
DEVICE P_20 OK STARTED
DEVICE P_21 OK STARTED
cisco@ubuntu:~/nso572/ncs-run$
```

If see the following output, everything it's OK, that means the devices had already been started.

```
cisco@ubuntu:~/nso572/ncs-run$ ncs-netsim start
Cannot bind to internal socket 127.0.0.1:5010 : address already in use
Daemon died status=20
DEVICE PE_00 FAIL
Cannot bind to internal socket 127.0.0.1:5011 : address already in use
Daemon died status=20
DEVICE PE_01 FAIL
Cannot bind to internal socket 127.0.0.1:5012 : address already in use
Daemon died status=20
DEVICE PE_10 FAIL
Cannot bind to internal socket 127.0.0.1:5013 : address already in use
Daemon died status=20
DEVICE PE_11 FAIL
Cannot bind to internal socket 127.0.0.1:5014 : address already in use
Daemon died status=20
DEVICE P_20 FAIL
Cannot bind to internal socket 127.0.0.1:5015 : address already in use
Daemon died status=20
DEVICE P_21 FAIL
cisco@ubuntu:~/nso572/ncs-run$
```

In case you are experiencing some issues, you can stop and start again the devices. To stop the devices run the following command.

```
cisco@nso-572i:~/ncs-run$ ncs-netsim stop
DEVICE PE_00 STOPPED
DEVICE PE_01 STOPPED
DEVICE PE_10 STOPPED
DEVICE PE_11 STOPPED
DEVICE P_20 STOPPED
DEVICE P_21 STOPPED
cisco@nso-572i:~/ncs-run$
```

If by mistake you have deleted the devices from NSO, once they have been started you can add them back by running the following command from `/home/cisco/nso572/ncs-run`.

```
cisco@nso-572i:~/ncs-run$ ncs-netsim ncs-xml-init > devices.xml
cisco@nso-572i:~/ncs-run$ ncs_load -l -m devices.xml
```

If you have to re-create the netsim devices please do as follows:

```
cisco@ubuntu:~/nso572/ncs-run$ cd /home/cisco/nso572/ncs-run
```



```
cisco@ubuntu:~/nso572/ncs-run$ ncs-netsim create-network packages/cisco-ios
2 PE_0
DEVICE PE_00 CREATED
DEVICE PE_01 CREATED
cisco@ubuntu:~/nso572/ncs-run$ ncs-netsim add-to-network packages/cisco-
iosxr 2 PE_1
DEVICE PE_10 CREATED
DEVICE PE_11 CREATED
cisco@ubuntu:~/nso572/ncs-run$ ncs-netsim add-to-network packages/juniper-
junos 2 P_2
DEVICE P_20 CREATED
DEVICE P_21 CREATED
cisco@ubuntu:~/nso572/ncs-run$ ncs-netsim start
DEVICE PE_00 OK STARTED
DEVICE PE_01 OK STARTED
DEVICE PE_10 OK STARTED
DEVICE PE_11 OK STARTED
DEVICE P_20 OK STARTED
DEVICE P_21 OK STARTED
cisco@ubuntu:~/nso572/ncs-run$ ncs-netsim ncs-xml-init > devices.xml
cisco@ubuntu:~/nso572/ncs-run$ ncs_load -l -m devices.xml
```