Cisco live!
June 9-13, 2019 · San Diego, CA

Automating Services with NSO
LABNMS-1009

Proctors:

Sunpreetsingh Arora

Bartosz Luraniec

Hector Oses

Table of Contents

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

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.
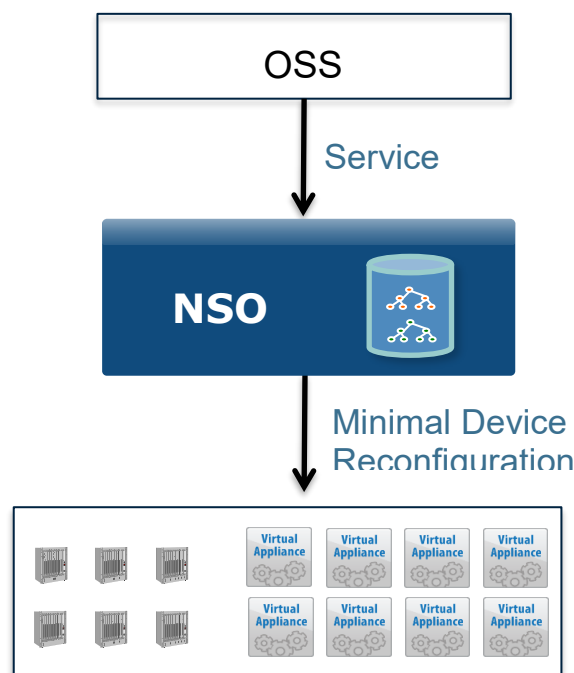
The network is a foundation for revenue generation. Therefore, service providers must implement network orchestration to simplify the entire lifecycle management for services. For today's virtualized networks, this means transparent orchestration that spans multiple domains in your network and includes 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.
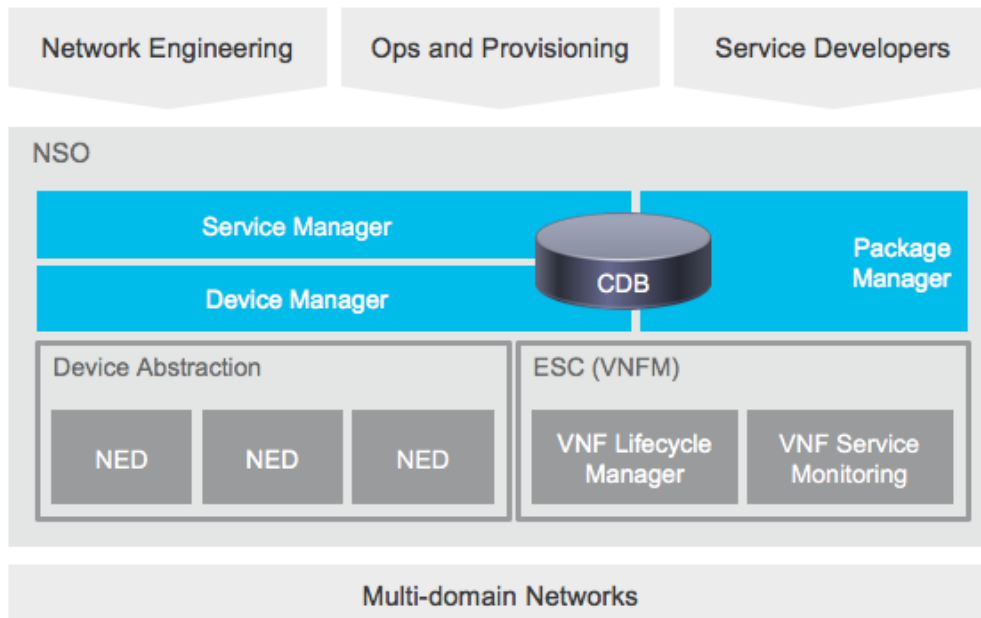
Topology

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

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.

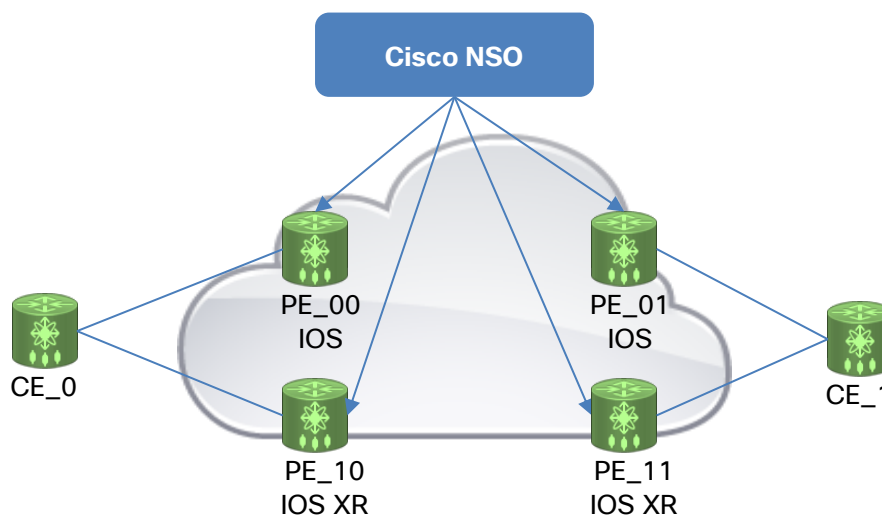1. (preferred) Visual Studio Code – Insiders. You can find a shortcut in the desktop to this application. When you start it, you will be connected to NSO server and you will be able to view and edit files from your local Windows. A Terminal is available as well. See Appendix A for more information.
2. 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.
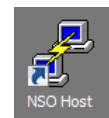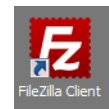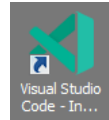
Lab Verification

The NSO version 4.7.3.2 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.
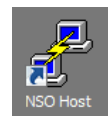
1. Double click on it to access (or open terminal in VSCode – Insiders).
2. Move to the running directory for starting NSO and simulated Netsim devices

```
cisco@nso-4732i:~$ cd /home/cisco/ncs-run
```
3. Start NSO:

```
cisco@nso-4732i:~/ncs-run$ ncs
```
4. Start netsim devices:

```
cisco@nso-4732i:~/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
cisco@nso-4732i:~/ncs-run$
```

Desktop shortcut 'NSO Admin cli' allows you to connect to NSO cli mode as user 'admin'.
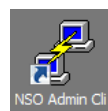
5. Double click on it to access
6. Cisco NSO allows 2 types of cli to interact with it. During this workbook we will use Cisco based cli.

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

7. 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
admin@ncs#
```

8. Now we will make sure NSO Configuration database (CDB) is synchronized with the devices

```
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
}
```

9. Verify that we have 2 customers configured in NSO:

```
admin@ncs# show customers
      CUSTOMER
ID    SERVICE
----------------
ACME  -
EMCA  -

admin@ncs#
```

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

```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.
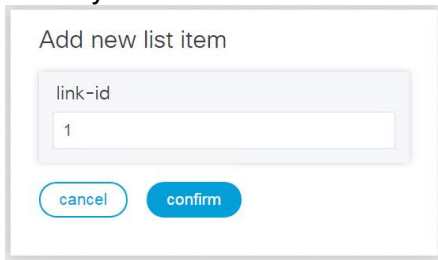
1. Go to NSO GUI (http://198.18.134.28:8080), login as admin/admin and navigate to the L3VPN section.
2. Click the "+" button to add a new L3VPN instance
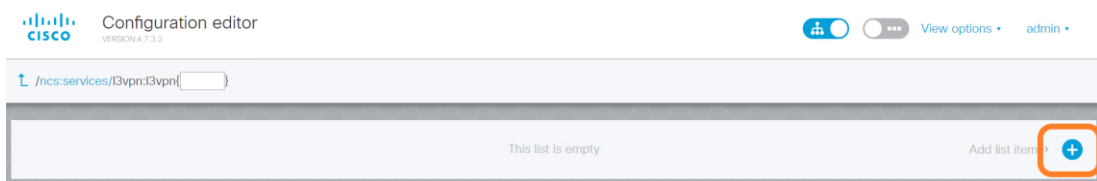
3. Add the vpn-name 'ACME_01' and click 'confirm' button.
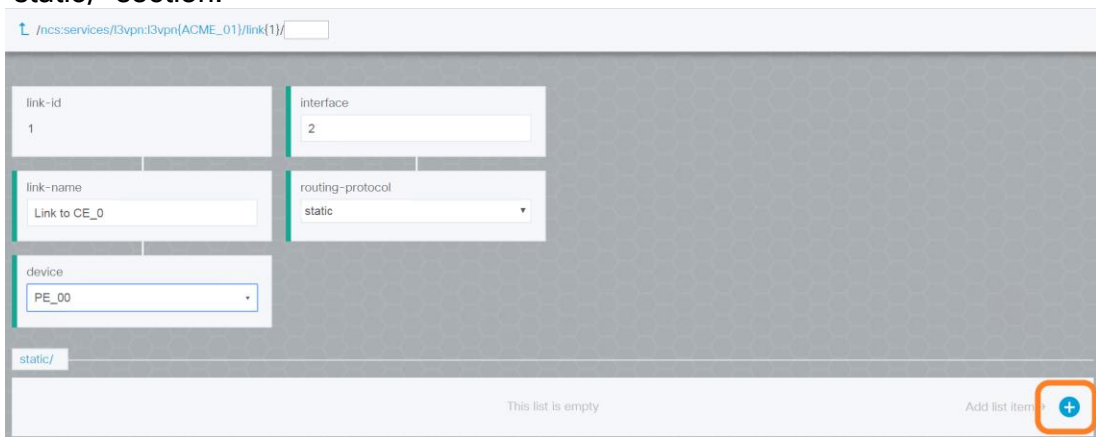
4. Click in the list the name of your vpn-name to start editing it.
5. Add 'vpn-id' 10001 and select customer 'ACME'
6. Click the '+' button to add a new link

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

8. Click in the 'link-id' 1 from the table to start editing it
9. Name this link as 'link-name' 'Link to CE_0'
10. Select the PE device where to configure the L3VPN PE_00
11. Use 'interface' 2, this will configure PE_00 GigabitEthernet 2
12. Select 'routing-protocol' 'Static'
13. Because we have selected 'Static' a new section appears to provision "Static" specific parameters (if it doesn't please refresh the page). Click the '+' button on 'static/' section.

14. Introduce 'prefix' '192.168.11.0' and click confirm button
15. Click in the prefix in the list to edit it.
16. Add 'mask' '255.255.255.0'
17. 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.
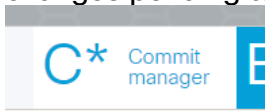
18. NSO performs a validation to make sure the parameters introduced are ok. If you get any error please go back to the service and update as required
19. The 'config' tab shows you the configuration that NSO will add for the service and the device configured. This is equivalent of doing in cli 'commit dry-run'

20. 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'.


```
Current transaction is VALID     Revert    Rollbacks                    □ no check-sync  □ no networking   Commit

changes   warnings   config   native config

PE_00

vrf definition vpn10001
 rd 1:10001
 route-target export 1:10001
 route-target import 1:10001
!
interface GigabitEthernet2
 description "Connection to Customer ACME - Site 5"
 vrf forwarding vpn10001
 no switchport
 ip address 172.31.1.1 255.255.255.252
 no shutdown
exit
ip route vrf vpn10001 192.168.11.0 255.255.255.0 172.31.1.2
```

21. Click 'commit' button to provision the device.

22. You can then go to 'Device Manager' button to explore the device configuration, but this is much faster to verify from cli.

23. What we did above is equivalent of writing this in cli from config mode:

```
admin@ncs(config)# services l3vpn 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
```

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

```
admin@ncs# show running-config devices device PE_00 config ios:vrf
definition vpn10001
devices device PE_00
 config
  ios: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 ios:ip
route vrf vpn10001
devices device PE_00
 config
  ios: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
ios:interface G
Possible completions:
  GigabitEthernet - GigabitEthernet IEEE 802.3z
  Group-Async     - Async Group interface
admin@ncs# show running-config devices device PE_00 config
ios:interface GigabitEthernet 2
devices device PE_00
 config
  ios: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 ios:vrf
definition vpn10001
devices device PE_00
 config
  ios: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)# ios: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/ncs-run/packages

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

```
cisco@nso-4732i:~/ncs-run$ cd /home/cisco/ncs-run/packages/l3vpn/src/
cisco@nso-4732i:~/ncs-run/packages/l3vpn/src$
cisco@nso-4732i:~/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-4.7.3.2/bin/ncsc  `ls l3vpn-ann.yang  > /dev/null
2>&1 && echo "-a l3vpn-ann.yang"` \
                -c -o ../load-dir/l3vpn.fxs yang/l3vpn.yang
cisco@nso-4732i:~/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
    result true
}
reload-result {
    package cisco-iosxr
    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
Possible completions:
  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
```

7. Configure all parameters and do a dry-run to see the configuration that will be pushed.

```
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 {
                        ios:vrf {
        +                   definition vpn10001 {
        +                       description "L3VPN for customer
ACME";
        +                       rd 1:10001;
        +                       route-target {
        +                           export 1:10001;
        +                           import 1:10001;
        +                       }
        +                   }
                        }
                        ios: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;
        +                               }
        +                           }
        +                       }
        +                   }
                        }
                        ios:router {
        +                   bgp 1 {
        +                       address-family {
        +                           with-vrf {
        +                               ipv4 unicast {
        +                                   vrf vpn10001 {
        +                                       redistribute {
        +                                           connected {
        +                                           }
        +                                           static {
        +                                           }
```

Cisco live!

```
                 +                                    }
                 +                                    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-4732i:~$ cd /home/cisco/ncs-run
cisco@nso-4732i:~/ncs-run$ ncs-netsim cli-c PE_01

admin connected from 198.18.133.252 using ssh on nso-4732i
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
```

Cisco live!

```
  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
 !
!
PE_01#
```

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
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
```

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:
  a. Add by hand all the configuration below line by line, which it is not practical.
  b. 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.
  c. Load from terminal using 'load merge terminal' command to be able to copy/paste/load all the configuration at once.
  d. 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 configuration found below

```
devices device PE_10
 config
  cisco-ios-xr:vrf vpn10001
   description Customer ACME VPN
   address-family ipv4 unicast
    import route-target
     1:10001
    exit
    export route-target
     1:10001
    exit
   exit
  exit
  cisco-ios-xr: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
  cisco-ios-xr:route-policy PASS
   end-policy
  !
  cisco-ios-xr:router static
   address-family ipv4 unicast
    192.168.21.0/24 GigabitEthernet0/0/0/1 172.31.1.2
   exit
  exit
  cisco-ios-xr: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>
```

```xml
            <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>
```

```xml
                            <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:

    o Cisco IOS: 'ios-id:cisco-ios'
    o 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 'LABNMS-1009'. Click the arrow to expand it
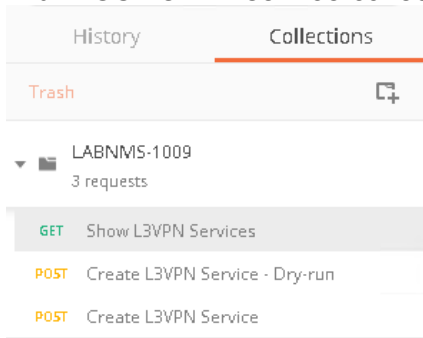
3. Open 'Show L3VPN Services' This Restconf call request the configuration of all configured L3VPN services (as indicated by the URI). Click send button.
4. If you left some of the previous services you will be able to see in bottom section of the screen the response from NSO
5. We will now create a 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
\"L3VPN for customer ACME\"\n rd            1:10010\n route-target
export 1:10010\n route-target import 1:10010\n!\ninterface
GigabitEthernet2\n description \"Connection to Customer ACME - Site
5\"\n vrf forwarding vpn10010\n no switchport\n ip address 172.31.2.1
255.255.255.252\n no shutdown\nexit\nrouter bgp 1\n address-family
ipv4 unicast vrf vpn10010\n  redistribute connected\n  redistribute
static\n  neighbor 172.31.2.2 remote-as 65001\n  neighbor 172.31.2.2
activate\n  neighbor 172.31.2.2 allowas-in\n  neighbor 172.31.2.2 as-
override disable\n  neighbor 172.31.2.2 default-originate\n  exit-
address-family\n !\n!\n"
                },
                {
                    "name": "PE_10",
                    "data": "vrf vpn10010\n description \"L3 VPN for
customer ACME\"\n address-family ipv4 unicast\n  import route-
target\n   1:10010\n  exit\n  export route-target\n   1:10010\n
exit\n exit\nexit\ninterface GigabitEthernet 2\n description
\"Connection to Customer ACME - Site 9\"\n vrf            vpn10010\n
ipv4 address 172.31.1.1 255.255.255.252\n no shutdown\nexit\nrouter
bgp 1\n vrf vpn10010\n  rd 1:10010\n  address-family ipv4 unicast\n
redistribute connected\n   redistribute static\n  exit\n  neighbor
172.31.1.2\n   address-family ipv4 unicast\n    route-policy PASS
in\n    as-override\n    default-originate\n   exit\n  exit\n
exit\nexit\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 a simple use case, nevertheless this service could keep expanding, adding more options, device types and simplify the service provisioning by many ways, like:
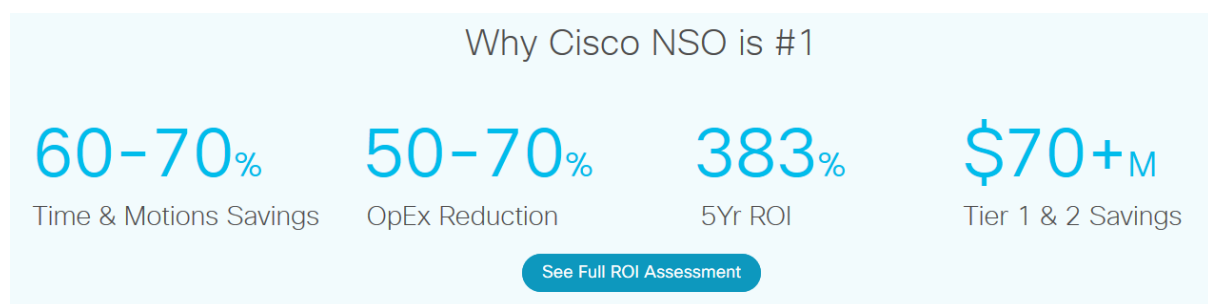
- Having a Topology service with the information about all nodes and parameter specific to them to avoid the operator having to introduce them.
- Configure many sites at the same time
- Use local or remote Resource Managers to administrate used VLAN, VRF, IP networks...
- Integrate with a northbound 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.

Why Cisco NSO is #1			
60-70%	50-70%	383%	$70+M
Time & Motions Savings	OpEx Reduction	5Yr ROI	Tier 1 & 2 Savings

See Full ROI Assessment

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\LABNMS-1009" (reachable through shortcut in Desktop) and as well inside NSO Host under "/home/cisco/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.
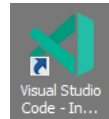
When you start 'Visual Studio Code – Insiders' you can verify that you are connected to NSO by looking at the bottom left corner for 'SSH: NSO-Host'. You should see as well in the left panel the /home/cisco directory with all the main files for NSO in directory /home/cisco/ncs-run. From there you can 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-4732i:~/ncs-run$ ncs_cli -C -u admin
```
Where '-C' is for Cisco CLI mode
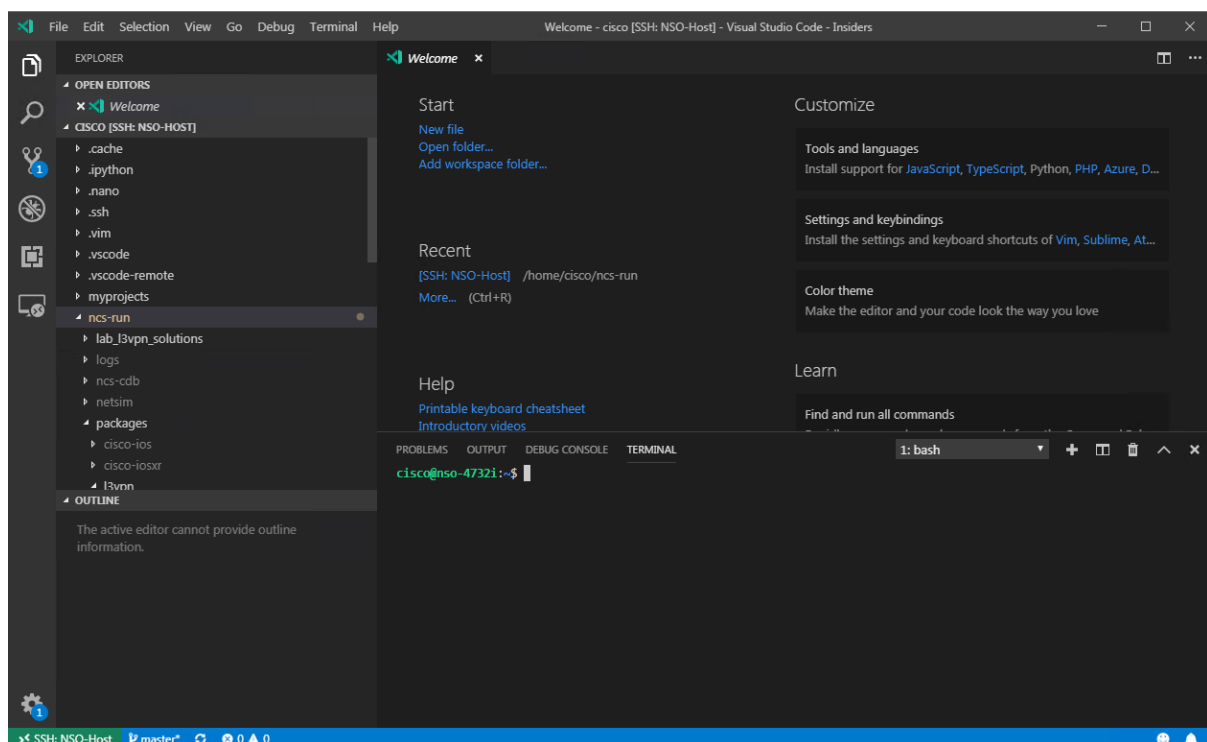
NOTE: Do not confuse 'Visual Studio Code – Insiders' with 'Visual Studio Code', that it is as well installed in Windows, but doesn't have a plugin to edit remote files.

Appendix B: Manage Netsim Devices

NSO already has 4 simulated devices (called Netsim devices across the lab guide) configured and ready to start from /home/cisco/ncs-run directory.

During Lab 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@nso-4732i:~/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
cisco@nso-4732i:~/ncs-run$
```

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

```
cisco@nso-4732i:~/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
cisco@nso-4732i:~/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-4732i:~/ncs-run$ ncs-netsim stop
DEVICE PE_00 STOPPED
DEVICE PE_01 STOPPED
DEVICE PE_10 STOPPED
DEVICE PE_11 STOPPED
cisco@nso-4732i:~/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/ncs-run.

```
cisco@nso-4732i:~/ncs-run$ ncs-netsim ncs-xml-init > devices.xml
cisco@nso-4732i:~/ncs-run$ ncs_load -l -m devices.xml
```

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

```
cisco@ubuntu:~/nso4732/ncs-run$ cd /home/cisco/nso4732/ncs-run
```

```
cisco@ubuntu:~/nso4732/ncs-run$ ncs-netsim create-network packages/cisco-
ios 2 PE_0
DEVICE PE_00 CREATED
DEVICE PE_01 CREATED
cisco@ubuntu:~/nso4732/ncs-run$ ncs-netsim add-to-network packages/cisco-
iosxr 2 PE_1
DEVICE PE_10 CREATED
DEVICE PE_11 CREATED
cisco@ubuntu:~/nso4732/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
cisco@ubuntu:~/nso4732/ncs-run$ ncs-netsim ncs-xml-init > devices.xml
cisco@ubuntu:~/nso4732/ncs-run$ ncs_load -l -m devices.xml
```