How to Migrate from a CLI NED to a NETCONF NED

Overview

This document describes how to migrate from a CLI NED to a NETCONF NED for an existing Cisco Network Services Orchestrator (NSO) service application and is divided into 4 sections: “Why should you use a NETCONF NED?”, “Steps for migrating from a CLI NED to a NETCONF NED,” “Examples for demonstrating the CLI NED to NETCONF NED migration procedure,” and “Troubleshooting.”

“Why should you use a NETCONF NED?” describes the benefits for interfacing to a network element using NETCONF instead of CLI through NSO.

“Steps for migrating from a CLI NED to a NETCONF NED” describes the procedure on how to migrate from a CLI NED to a NETCONF NED for an existing NSO service application.

“Examples for demonstrating the CLI NED to NETCONF NED migration procedure” describes how to work with two service examples from the NSO distribution and migrate one of the PE devices from CLI NED to NETCONF NED. The first example uses an official NETCONF NED for IOS-XR released by Cisco. The second example shows you how to build your own IOS-XR NETCONF NED to be used for the NED migration.

“Troubleshooting” contains various tips and tricks on how to debug the service templates and NETCONF NED building issues.

Contents

1 Why should you use a NETCONF NED?

NSO can speak southbound through the NED architecture to an arbitrary management interface supported by the device. NSO knows how to automatically communicate southbound to NETCONF enabled devices. By supplying NSO with the YANG data models of a NETCONF device, NSO knows the data models of the device, and, through the NETCONF protocol, knows exactly how to manipulate the device configuration.

Unfortunately, the majority of devices supported by earlier versions of NSO didn't speak NETCONF. By far, the most common way to configure network devices has been through the CLI. For NSO to speak to Cisco style CLI devices, the process is not entirely automatic like with NETCONF, and, depending on the type of interface the device has for configuration, this may involve some programming. Devices with a Cisco style CLI can be managed by writing YANG data models which describe the data in CLI and a relatively thin layer of Java code to handle the communication to the devices. Other types of devices will likely require more coding.

One of the benefits with NETCONF is that all device features can be supported on day 1 without any development effort on the NETCONF NED. With CLI NEDs, only a subset of CLI commands that are needed for the use case by the service application will be modeled and coded. This reduces the upfront work. When additional CLI commands are needed, more development work is required.

The other benefit with NETCONF is the set of standards-based YANG data models that are being supported by multiple device vendors. The service to device mapping template work only needs to be done once and can be reused by all device vendors that support the same standards-based YANG data models.

From a runtime standpoint, NETCONF is optimized for machine-to-machine communication and has much better performance as compared to a CLI which is optimized for human operators.From a testability standpoint, NETCONF allows for systematic testing which can be easily automated. CLI NEDs are hard to test and require re-testing for minor upgrades.

Another important benefit with NETCONF is that it comes with support for network-wide transactions which significantly simplifies service deployment across multiple devices in the entire network and provides the ability for the devices to automatically rollback their configuration if anything fails. As a result, the network is never left in an inconsistent state. This is something that isn't available through the CLI NED.

2 Steps for migrating from a CLI NED to a NETCONF NED

For an existing NSO service application that has devices in the network which are using a CLI NED, you can select a CLI-based device that also supports NETCONF to be used for the migration. You can choose to work with either pre-built NETCONF NEDs, if available, or build your own NETCONF NED for the selected device. You can get pre-built NETCONF NEDs either from Cisco or the device vendor. Pre-built NETCONF NEDs come with the benefit that they should have already been validated to work with NSO. If a NETCONF NED isn't available for your device, you can refer to the "NETCONF & YANG Automation Testing User Guide v3" available for download at https://info.tail-f.com/netconf_yang_automation_testing for information on how to build your own NETCONF NED and validate it. An alternative is to request your device vendor to begin to provide a NETCONF NED.

It is always a good idea to update the CLI NED to make sure that it is compatible with the NETCONF NED. Any configuration changes made through CLI should be properly reflected through NETCONF. In order for the service application to be able to work with both CLI and NETCONF NEDs for the same device, the following are the general steps to take:

1. Set up NSO to communicate with both the CLI and NETCONF interfaces of the device (only the CLI NED is being used by the service application)
   a. Upgrade the device to use the latest CLI NED (if necessary)
   b. Update the service template as necessary due to structural changes of the CLI commands
   c. Add a device to NSO using the NETCONF NED
      i. Either a pre-built NETCONF NED supplied by Cisco or the device vendor
      ii. Or build your own NETCONF NED
2. Generate the service template for the NETCONF NED
   a. Create a service instance using the CLI NED
   b. View the configuration changes in XML for the NETCONF NED using the compare-config command in ncs_cli
   c. Take this XML diffs from the pervious step as input to a new section in the service templates
   d. Fill in variables in the NETCONF template as used by the CLI version of the template
3. Verify the service mapping after switching to the NETCONF NED
   a. Switch to the NETCONF NED for the device
   b. Re-deploy dry-run to see that nothing is forgotten
   c. Un-deploy the service
   d. Re-deploy the service
   e. Compare the configuration

3 Examples for demonstrating the CLI NED to NETCONF NED migration procedure

Two service application examples that come with the NSO distribution will be used as a basis to demonstrate how to migrate from the CLI NED for a IOS-XR based PE device to a NETCONF NED. The two examples are stored under $NCS_DIR/examples.ncs/service-provider. They are called simple-mpls-vpn and mpls-vpn. The simple-mpls-vpn example only uses service templates for implementing the service to device configuration mapping. The mpls-vpn example uses both Java code and configuration templates for implementing the service application mapping. You will see that the migration process works similarly for both examples.

A pre-built IOS-XR NETCONF NED released by Cisco will be used for the simple-mpls-vpn example to illustrate the NED migration procedure. A custom IOS-XR NETCONF NED will be built using the NETCONF NED Builder for the mpls-vpn example to illustrate the NED migration procedure. You are free to go through both examples or pick one that is more relevant to what you plan on doing.

3.1 The simple-mpls-vpn example

Let's first make a fresh copy of simple-mpls-vpn from the NSO 5.3.1.1 distribution:

```
nso-dd$ cp -a $NCS_DIR/examples.ncs/service-provider/simple-mpls-vpn .
nso-dd$ cd simple-mpls-vpn
simple-mpls-vpn$ make clean all
```

In order to allow copy and paste in ncs_cli, the following two XML fragments will need to be added to the cli block inside of ncs.conf in the current directory:

```
<ignore-leading-whitespace>true</ignore-leading-whitespace>
<auto-wizard><enabled>false</enabled></auto-wizard>
```

After the above lines have been added to ncs.conf, we are ready to start the simple-mpls-vpn project:

```
simple-mpls-vpn$ make start
```

At this point, all netsim (simulated) devices for the project and NSO have been started.

3.1.1 Update the CLI NED and install a NETCONF NED

Download an updated CLI NED and the NETCONF NED for IOS-XR from the NSO delivery server and copy them to the packages directory.

```
simple-mpls-vpn$ cp -a ~/Downloads/cisco-iosxr-cli-7.25 packages
simple-mpls-vpn$ cp -a ~/Downloads/cisco-iosxr-nc-7.0 packages
simple-mpls-vpn$ ncs_cli -C -u admin

admin connected from 127.0.0.1 using console on WAITAI-M-72J2
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-3.0
    result true
}
reload-result {
    package cisco-iosxr-cli-3.0
    result true
}
reload-result {
    package cisco-iosxr-cli-7.25
    result true
}
reload-result {
    package cisco-iosxr-nc-7.0
    result true
}
reload-result {
    package l3vpn
    result true
}
```

3.1.2 Connect service to a real PE device

For the IOS-XR based PE devices used in the example, the "volvo" l3vpn service only touches "pe2". For this demo, a real IOS-XR device is connected as "pe2" and the latest CLI NED is used for it. In this case, a Cisco IOS XRv 9000 virtual PE router is used. Here are the steps to upgrade the CLI NED and connect it to a real IOS-XR device:

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# devices device pe2
admin@ncs(config-device-pe2)# device-type cli ned-id cisco-iosxr-cli-7.25
admin@ncs(config-device-pe2)# address 10.147.46.208
admin@ncs(config-device-pe2)# port 33881
admin@ncs(config-device-pe2)# commit
Commit complete.
admin@ncs(config-device-pe2)# ssh fetch-host-keys

result updated
fingerprint {
    algorithm ssh-rsa
    value fb:44:4a:d7:e5:74:87:12:cc:72:7a:cb:ff:53:a0:0f
}
fingerprint {
    algorithm ssh-dss
    value 0c:3d:69:eb:e2:95:b4:b4:d2:89:6f:44:92:a3:9c:db
}
```

Because the configuration for "pe2" in NSO is empty at this point, I was able to modify its ned-id. Otherwise, I would first need to perform a "no config" on "pe2" followed by a "commit" or "commit no-networking" before I can perform the above changes to "pe2".

Just to show you the version information of the IOS-XR router that I have connected to, the following is its "show version" CLI output:

```
simple-mpls-vpn $ ssh admin@10.147.46.208 -p 33881
Password:

RP/0/RP0/CPU0:xrv9000#show version
Sun Jun 07 22:20:42.379 UTC
Cisco IOS XR Software, Version 7.0.2
Copyright (c) 2013-2020 by Cisco Systems, Inc.

Build Information:
 Built By     : ahoang
 Built On     : Fri Mar 13 22:27:54 PDT 2020
 Built Host   : iox-ucs-029
 Workspace    : /auto/srcarchive15/prod/7.0.2/xrv9k/ws
 Version      : 7.0.2
 Location     : /opt/cisco/XR/packages/
 Label        : 7.0.2

cisco IOS-XRv 9000 () processor
System uptime is 5 weeks 5 days 20 hours 22 minutes

RP/0/RP0/CPU0:xrv9000#
```

3.1.3 Connect to the NETCONF interface of the real PE device

It will become handy later on to have a NETCONF session connected to the same pe device and allow the result of the CLI configuration changes to be captured through NSO in the NETCONF format. Let's set up NSO to connect to the NETCONF interface of the real pe router using the previously loaded NETCONF NED and call it "pe2-nc":

```
admin@ncs(config)# devices device pe2-nc
admin@ncs(config-device-pe-nc)# address 10.147.46.208 port 33880
admin@ncs(config-device-pe-nc)# authgroup default
admin@ncs(config-device-pe-nc)# device-type netconf ned-id cisco-iosxr-nc-7.0
admin@ncs(config-device-pe-nc)# state admin-state unlocked
admin@ncs(config-device-pe-nc)# commit
Commit complete.
admin@ncs(config-device-pe-nc)# ssh fetch-host-keys
result updated
fingerprint {
    algorithm ssh-rsa
    value fb:44:4a:d7:e5:74:87:12:cc:72:7a:cb:ff:53:a0:0f
}
fingerprint {
    algorithm ssh-dss
    value 0c:3d:69:eb:e2:95:b4:b4:d2:89:6f:44:92:a3:9c:db
}
```

We then synchronize all the device configurations into NSO:

```
admin@ncs# devices sync-from
sync-result {
    device ce0
    result true
}
sync-result {
    device ce1
    result true
}
…
```

3.1.4 Set up a l3vpn service with the CLI-based IOS-XR router

Let's set up the l3vpn service named "volvo" as described in the README file:

```
admin@ncs(config)# vpn l3vpn volvo
admin@ncs(config-l3vpn-volvo)# endpoint c1
admin@ncs(config-endpoint-c1)# as-number 65001
admin@ncs(config-endpoint-c1)# ce device ce0
admin@ncs(config-endpoint-c1)# ce local interface-name GigabitEthernet
admin@ncs(config-endpoint-c1)# ce local interface-number 0/9
admin@ncs(config-endpoint-c1)# ce local ip-address 192.168.0.1
admin@ncs(config-endpoint-c1)# ce link interface-name GigabitEthernet
admin@ncs(config-endpoint-c1)# ce link interface-number 0/2
admin@ncs(config-endpoint-c1)# ce link ip-address 10.1.1.1
```

```
admin@ncs(config-endpoint-c1)# pe device pe2
admin@ncs(config-endpoint-c1)# pe link interface-name GigabitEthernet
admin@ncs(config-endpoint-c1)# pe link interface-number 0/0/0/1
admin@ncs(config-endpoint-c1)# pe link ip-address 10.1.1.2
admin@ncs(config-endpoint-c1)#  !
admin@ncs(config-endpoint-c1)#  endpoint c2
admin@ncs(config-endpoint-c2)# as-number 65001
admin@ncs(config-endpoint-c2)# ce device ce2
admin@ncs(config-endpoint-c2)# ce local interface-name GigabitEthernet
admin@ncs(config-endpoint-c2)# ce local interface-number 0/3
admin@ncs(config-endpoint-c2)# ce local ip-address 192.168.1.1
admin@ncs(config-endpoint-c2)# ce link interface-name GigabitEthernet
admin@ncs(config-endpoint-c2)# ce link interface-number 0/1
admin@ncs(config-endpoint-c2)# ce link ip-address 10.2.1.1
admin@ncs(config-endpoint-c2)# pe device pe2
admin@ncs(config-endpoint-c2)# pe link interface-name GigabitEthernet
admin@ncs(config-endpoint-c2)# pe link interface-number 0/0/0/2
admin@ncs(config-endpoint-c2)# pe link ip-address 10.2.1.2
admin@ncs(config-endpoint-c2)# !
admin@ncs(config-endpoint-c2)#
admin@ncs(config-endpoint-c2)# top
admin@ncs(config)#commit
Error: External error in the NED implementation for device pe2: Wed May 27
05:25:51.557 UTC

% Failed to commit one or more configuration items during a pseudo-atomic opera-
tion. All changes made have been reverted.
 !! SEMANTIC ERRORS: This configuration was rejected by
!! the system due to semantic errors. The individual
!! errors with each failed configuration command can be
!! found below.


router bgp 100
 vrf volvo
  address-family ipv4 unicast
!!% 'BGP' detected the 'warning' condition 'The parent address family has not
been initialized'
  !
 !
!
end
admin@ncs(config)#
```

The above error indicates that the service template that was developed for the original example isn't compatible with the virtual XRv 9000 PE router that is being used here. If you search on the Internet for reasons of this warning, you will discover that "address-family vpnv4 unicast" has to be configured on the parent container named BGP before it can be used in a vrf. This requires modifying the service template located at packages/l3vpn/templates/l3vpn.xml. The following block needs to be added to "PE template for Cisco IOS-XR routers" under the path of "router/bgp/bgp-no-instance":

```
<address-family>
  <vpnv4>
    <unicast>
    </unicast>
  </vpnv4>
</address-family>
```

In order to be able to work with both versions of the CLI NED, I'll add the following conditional constructs based on ned-ids to the template:

```
<?if-ned-id cisco-iosxr-cli-3.0:cisco-iosxr-cli-3.0?>
  <router xmlns="http://tail-f.com/ned/cisco-ios-xr" tags="merge">
  <bgp>
    <bgp-no-instance>
      <id>100</id>
      <vrf tags="merge">
        <name>{string(/name)}</name>
        <?set-context-node {..}?>
        <rd>{as-number}:1</rd>
        <address-family>
          <ipv4>
            <unicast>
            </unicast>
          </ipv4>
        </address-family>
  …
<?elif-ned-id cisco-iosxr-cli-7.25:cisco-iosxr-cli-7.25?>
  <router xmlns="http://tail-f.com/ned/cisco-ios-xr" tags="merge">
  <bgp>
    <bgp-no-instance>

      <id>100</id>
      <address-family>
        <vpnv4>
          <unicast>
          </unicast>
        </vpnv4>
      </address-family>
      <vrf tags="merge">
        <name>{string(/name)}</name>
        <?set-context-node {..}?>
        <rd>{as-number}:1</rd>
        <address-family>
          <ipv4>
            <unicast>
            </unicast>
          </ipv4>
        </address-family>
  …
<?end?>
```

After applying the above changes to l3vpn.xml, the packages need to be reloaded:

```
simple-mpls-vpn$ ncs_cli -C -u admin

admin connected from 127.0.0.1 using console on WAITAI-M-72J2
admin@ncs# packages reload
reload-result {
    package cisco-ios-cli-3.0
    result true
}
reload-result {
    package cisco-iosxr-cli-3.0
    result true
}
reload-result {
    package cisco-iosxr-cli-7.25
    result true
}
reload-result {
    package cisco-iosxr-nc-7.0
    result true
}
reload-result {
    package l3vpn
    result true
}
admin@ncs#
```

Let's attempt to create the l3vpn service again:

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# vpn l3vpn volvo
admin@ncs(config-l3vpn-volvo)# endpoint c1
…
admin@ncs(config-endpoint-c2)# !
dmin@ncs(config)# commit
Commit complete.
```

This time the commit operation has succeeded and the l3vpn service called "volvo" is now up and running with mostly netsim (simulated) devices and one real IOS-XR CLI-based device named "pe2".

3.1.5 Generate a service template for the NETCONF device

Because the above service was only associated with the CLI interface of the XRv 9000 “pe2” device, the “pe2-nc” device that was created for its NETCONF interface did not get its configuration datastore in NSO synchronized when the above service configuration was committed. We’ll take advantage of this fact to identify the configuration which needs to be applied through NETCONF when the equivalent CLI configuration was applied to the device by the service.

We’ll use the compare-config command to generate the NETCONF payload in XML format:

```
admin@ncs(config)# devices device pe2-nc compare-config outformat xml
diff
<devices xmlns="http://tail-f.com/ns/ncs">
  <device>
    <name>pe2-nc</name>
    <config>
      <interface-configurations
          xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
        <interface-configuration>
          <active>act</active>
          <interface-name>GigabitEthernet0/0/0/1</interface-name>
          <description>link to CE</description>
          <vrf
              xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-rsi-cfg">
            volvo
          </vrf>
          <ipv4-network
              xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-io-cfg">
            <addresses>
              <primary>
                <address>10.1.1.2</address>
                <netmask>255.255.255.252</netmask>
              </primary>
            </addresses>
          </ipv4-network>
        </interface-configuration>
        …
```

The XML diffs generated by the compare-config operation is what the “volvo” service needs to push out to the IOS-XR device over NETCONF. Since the original service was based on an XML template towards the CLI NED, all we need to do is to add these XML diffs for NETCONF to the service template, and parameterize it using the same variables used by the CLI template.

When you look through the service template at packages/l3vpn/templates/l3vpn.xml, you will see the following high-level structure:

```
<config-template xmlns="http://tail-f.com/ns/config/1.0"
  servicepoint="l3vpn-template">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <?foreach {endpoint/ce}?>
    <device tags="nocreate">
      <name>{device}</name>
      <config tags="merge">
        <!-- CE template for Cisco IOS routers -->
        ...

        <!-- PE template for Cisco IOS-XR routers -->
        ...
      </config>
    </device>
    <?end?>
  </devices>
</config-template>
```

What we need to do is to add a 3rd block for a PE template for “Cisco IOS-XR routers over NETCONF” after the “PE template for Cisco IOS-XR routers” block and simply paste the XML diff we generated earlier into the template file before the </config> tags. We don’t want every line of the diff, only the part that’s inside the diff’s <config>...</config> tags. Also, remember to add tags=”merge” on each top level node. Another thing to note here is that the list instances are merged during the service to device configuration mapping process. We only need a single instance from a list to be present in the template. There are two of these in the XML diffs and they are the <interface-configuration></interface-configuration> and <vrf-neighbor></vrf-neighbor> tags.

Parameterizing the template means looking for hard-coded values in the XML diffs and replacing them with XPATH expressions in {...} format. Having a ready-made CLI template here makes the process rather trivial.

The first XML block of the final template should look something like the following:

```
<interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-
cfg" tags="merge">
  <interface-configuration tags="merge">
    <active>act</active>
    <interface-name>GigabitEthernet{link/interface-number}</interface-name>
    <description>link to CE</description>
    <vrf xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-rsi-cfg">
      {string(/name)}
    </vrf>
    <ipv4-network xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-io-cfg">
      <addresses>
        <primary>
          <address>{ip-address}</address>
          <netmask>255.255.255.252</netmask>
        </primary>
      </addresses>
    </ipv4-network>
  </interface-configuration>
</interface-configurations>
…
```

You want to make sure that the context node is at the right level after the XPATH expression has been evaluated. You can't always use the variables exactly the same way they are used in the CLI template as their structures may be different. Sometimes, you have to add constructs such as "../" to the path of the variable in order to get to the parent of the context node. Refer to the NSO Development Guide for full documentation on how to write templates.

The complete NETCONF template for IOS-XR can be found in Appendix A.

3.1.6 Re-deploy service to use the NETCONF device

After adding the NETCONF template for the IOS-XR router, the "packages reload" command needs to be executed for the new l3vpn.xml to take effect. Before we change the device configuration for "pe2" to point to its NETCONF port, we'll first un-deploy the service to unconfigure the device. This will make the service re-deployment more interesting. Otherwise, no device configuration may be needed to be pushed out upon a re-deploy. The other benefit of performing the un-deploy first is that the configuration of "pe2" is now empty. I can now change its ned-id without first having to unconfigure it. The device configuration for "pe2" is now changed to point to the NETCONF port and the use of netconf and the IOS-XR NETCONF NED as its device-type and ned-id respectively. The commands are as follows:

```
admin@ncs# packages reload
…
reload-result {
    package l3vpn
    result true
}
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# vpn l3vpn volvo un-deploy
admin@ncs(config)# devices device pe2
admin@ncs(config-device-pe2)# port 33880
admin@ncs(config-device-pe2)# device-type netconf ned-id cisco-iosxr-nc-7.0
admin@ncs(config-device-pe2)# commit
Commit complete.
admin@ncs(config-device-pe2)# ssh fetch-host-keys
result unchanged
fingerprint {
    algorithm ssh-rsa
    value fb:44:4a:d7:e5:74:87:12:cc:72:7a:cb:ff:53:a0:0f
}
fingerprint {
    algorithm ssh-dss
    value 0c:3d:69:eb:e2:95:b4:b4:d2:89:6f:44:92:a3:9c:db
}
admin@ncs(config-device-pe2)# sync-from
result true
admin@ncs(config-device-pe2)# top
```

A service check-sync performed now will find the service to be out-of-sync as we have previously un-deployed the service. We'll perform a "re-deploy dry-run" of the service to check out the configuration that would be be pushed out to "pe2" in the NETCONF format:

```
admin@ncs(config)# vpn l3vpn volvo check-sync
in-sync false
admin@ncs(config)# vpn l3vpn volvo re-deploy dry-run
cli {
  local-node {
    data devices {
          device ce0 {
           …
          device ce2 {
           …
          device pe2 {
            config {
              interface-configurations {
                interface-configuration act GigabitEthernet0/0/0/1 {
        +         description "link to CE";
        +         vrf volvo;
                  ipv4-network {
                    addresses {
        +             primary {
        +               address 10.1.1.2;
        +               netmask 255.255.255.252;
                      }
                    }
                  }
                }
                …
            }
          }
        }

  }
}
```

When the dry-run output is as expected from comparing to the NETCONF template, we can perform the re-deploy operation. We can then do an un-deploy to insure that the service can also be un-deployed successfully.

```
admin@ncs(config)# vpn l3vpn volvo re-deploy
admin@ncs(config)#
System message at 2020-06-07 18:00:07...
Commit performed by admin via console using cli.
admin@ncs(config)# vpn l3vpn volvo un-deploy
admin@ncs(config)# exit
```

If you want to cross check that the NETCONF configuration being pushed out will cause the same configuration through the CLI, you can add a device in NSO to point to the CLI port and use compare-config after going from un-deploy to re-reploy of the service through NETCONF. We have now successfully completed the CLI NED to NETCONF NED migration for this l3vpn service. It is rather simple!

3.2 The mpls-vpn example

First, make a fresh copy of the mpls-vpn example from the NSO 5.3.1 distribution:

```
nso-dd$ cp -a $NCS_DIR/examples.ncs/service-provider/mpls-vpn .
nso-dd$ cd mpls-vpn
```

Then, make a copy of the downloaded CLI NED for IOS-XR from the last example into the packages directory of your NSO installation:

```
mpls-vpn$ cp -a ~/Downloads/cisco-iosxr-cli-7.25 $NCS_DIR/packages
```

Then, we modify the example to use the downloaded CLI NED for IOS-XR and build the project:

```
mpls-vpn$ sed -i '' 's/3.5/7.25/g' Makefile
mpls-vpn$ sed -i '' 's/3.5/7.25/g' initial_data/template.xml
mpls-vpn$ make clean all
```

In order to allow copy and paste in ncs_cli, the following two XML fragments will need to be added to the cli block inside of ncs.conf in the current directory:

```
<ignore-leading-whitespace>true</ignore-leading-whitespace>
<auto-wizard><enabled>false</enabled></auto-wizard>
```

After the above lines have been added to ncs.conf, we are ready to start the simple-mpls-vpn project:

```
mpls-vpn$ make start
```

At this point, all simulated devices for the project and NSO have been started. Let's synchronize all devices' configuration data into NSO:

```
mpls-vpn$ ncs_cli -C -u admin
admin connected from 127.0.0.1 using console on WAITAI-M-72J2
admin@ncs# devices sync-from
sync-result {
    device ce0
    result true
}
…
sync-result {
    device pe3
    result true
}
```

3.2.1 Set up two l3vpn services

Now we set up the l3vpn services named "volvo" and "ford" as described in the README file by copying and pasting it into ncs_cli:

```
admin@ncs(config)# vpn l3vpn volvo
admin@ncs(config-l3vpn-volvo)# route-distinguisher 999
admin@ncs(config-l3vpn-volvo)# endpoint main-office
admin@ncs(config-endpoint-main-office)# ce-device ce6
admin@ncs(config-endpoint-main-office)# ce-interface GigabitEthernet0/11
admin@ncs(config-endpoint-main-office)# ip-network   10.10.1.0/24
admin@ncs(config-endpoint-main-office)# as-number    65101
admin@ncs(config-endpoint-main-office)# bandwidth    12000000
admin@ncs(config-endpoint-main-office)# !
admin@ncs(config-endpoint-main-office)# endpoint branch-office1
```

```
admin@ncs(config-endpoint-branch-office1)# ce-device ce1
admin@ncs(config-endpoint-branch-office1)# ce-interface GigabitEthernet0/11
admin@ncs(config-endpoint-branch-office1)# ip-network   10.7.7.0/24
admin@ncs(config-endpoint-branch-office1)# as-number    65102
admin@ncs(config-endpoint-branch-office1)# bandwidth    6000000
admin@ncs(config-endpoint-branch-office1)# !
admin@ncs(config-endpoint-branch-office1)# endpoint branch-office2
admin@ncs(config-endpoint-branch-office2)# ce-device ce4
admin@ncs(config-endpoint-branch-office2)# ce-interface GigabitEthernet0/18
admin@ncs(config-endpoint-branch-office2)# ip-network   10.8.8.0/24
admin@ncs(config-endpoint-branch-office2)# as-number    65103
admin@ncs(config-endpoint-branch-office2)# bandwidth    300000
admin@ncs(config-endpoint-branch-office2)# !
admin@ncs(config-endpoint-branch-office2)# top
admin@ncs(config)# commit dry-run outformat native
native {
    device {
        name ce1
        data policy-map volvo
        …
    }
    device {
        name ce4
        data policy-map volvo
        …
    }
    device {
        name ce6
        data policy-map volvo
        …
    }
    device {
        name pe0
        data vrf volvo
        …
    }
    device {
        name pe2
        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">
        …
}
device {
        name pe3
        data configure
             qos
             …
}
}
admin@ncs(config)# commit
Commit complete.
admin@ncs(config)# vpn l3vpn ford
admin@ncs(config-l3vpn-ford)# route-distinguisher 777
```

```
admin@ncs(config-l3vpn-ford)# endpoint main-office
admin@ncs(config-endpoint-main-office)# ce-device ce2
admin@ncs(config-endpoint-main-office)# ce-interface GigabitEthernet0/5
admin@ncs(config-endpoint-main-office)# ip-network   192.168.1.0/24
admin@ncs(config-endpoint-main-office)# as-number    65201
admin@ncs(config-endpoint-main-office)# bandwidth    10000000
admin@ncs(config-endpoint-main-office)# !
admin@ncs(config-endpoint-main-office)# endpoint branch-office1
admin@ncs(config-endpoint-branch-office1)# ce-device ce3
admin@ncs(config-endpoint-branch-office1)# ce-interface GigabitEthernet0/5
admin@ncs(config-endpoint-branch-office1)# ip-network   192.168.2.0/24
admin@ncs(config-endpoint-branch-office1)# as-number    65202
admin@ncs(config-endpoint-branch-office1)# bandwidth    5500000
admin@ncs(config-endpoint-branch-office1)# !
admin@ncs(config-endpoint-branch-office1)# endpoint branch-office2
admin@ncs(config-endpoint-branch-office2)# ce-device ce5
admin@ncs(config-endpoint-branch-office2)# ce-interface GigabitEthernet0/5
admin@ncs(config-endpoint-branch-office2)# ip-network   192.168.7.0/24
admin@ncs(config-endpoint-branch-office2)# as-number    65203
admin@ncs(config-endpoint-branch-office2)# bandwidth    1500000
admin@ncs(config-endpoint-branch-office2)# !
admin@ncs(config-endpoint-branch-office2)# top
admin@ncs(config)# commit dry-run
…
admin@ncs(config)# commit
Commit complete.
admin@ncs(config)# exit
admin@ncs# exit
```

It is good practice to inspect the output of a “commit dry-run” during development to insure that the expected configuration changes are being sent to the proper devices. Most of the dry-run output has been left out to conserve space in this document. Both l3vpn services are now up and running just like they were in the original example.

3.2.2 Connect service to a real PE device

For the IOS-XR based PE devices in this example, both the “volvo” and “ford” l3vpn services will touch pe0’s device configuration. A real IOS-XR device will be connected “pe0”. The same Cisco IOS XRv 9000 virtual PE router as used in the previous example will be used here. Here are the steps to connect to the real IOS-XR device:

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# devices device pe0
admin@ncs(config-device-pe2)# address 10.147.46.208
admin@ncs(config-device-pe2)# port 33881
admin@ncs(config-device-pe2)# commit
Commit complete.
admin@ncs(config-device-pe2)# ssh fetch-host-keys
result updated
fingerprint {
    algorithm ssh-rsa
    value fb:44:4a:d7:e5:74:87:12:cc:72:7a:cb:ff:53:a0:0f
}
fingerprint {
    algorithm ssh-dss
    value 0c:3d:69:eb:e2:95:b4:b4:d2:89:6f:44:92:a3:9c:db
}
admin@ncs(config-device-pe0)# sync-from
result true
admin@ncs(config-device-pe0)# top
```

Do a re-deploy to push out the proper configuration to the real IOS-XR device:

```
admin@ncs(config)# vpn l3vpn volvo check-sync
in-sync false
admin@ncs(config)# vpn l3vpn ford check-sync
in-sync false
admin@ncs(config)# vpn l3vpn volvo re-deploy dry-run
…
admin@ncs(config)# vpn l3vpn volvo re-deploy
Error: External error in the NED implementation for device pe0: Mon Jun  1
16:44:16.615 UTC

% Failed to commit one or more configuration items during a pseudo-atomic opera-
tion. All changes made have been reverted.
!! SEMANTIC ERRORS: This configuration was rejected by
!! the system due to semantic errors. The individual
!! errors with each failed configuration command can be
!! found below.


router bgp 100
 vrf volvo
  address-family ipv4 unicast
!!% 'BGP' detected the 'warning' condition 'The parent address family has not
been initialized'
  !
  neighbor 192.168.1.5
   address-family ipv4 unicast
```

```
    route-policy volvo in
!!% 'BGP' detected the 'warning' condition 'The address family has not been
initialized'
    route-policy volvo out
!!% 'BGP' detected the 'warning' condition 'The address family has not been
initialized'
   !
  !
 !
!
end
admin@ncs(config)# exit
admin@ncs# exit
```

The above error is the same as the one we ran into during the last example. In this example, the PE specific service templates can be found in packages/l3vpn/templates/l3vpn-pe.xml. The following block needs to be added to the “Cisco ios xr” section under the path of “router/bgp/bgp-no-instance”:

```
<address-family>
  <vpnv4>
    <unicast>
    </unicast>
  </vpnv4>
</address-family>
```

Perform the “packages reload” command followed by a re-deploy of the “vpn l3vpn volvo” service:

```
mpls-vpn$ ncs_cli -C -u admin

admin connected from 127.0.0.1 using console on WAITAI-M-72J2
admin@ncs# packages reload
…
dmin@ncs(config)# vpn l3vpn volvo re-deploy
Error: External error in the NED implementation for device pe0: Tue Jun  2
23:21:35.992 UTC

% Failed to commit one or more configuration items during a pseudo-atomic opera-
tion. All changes made have been reverted.
 !! SEMANTIC ERRORS: This configuration was rejected by
!! the system due to semantic errors. The individual
!! errors with each failed configuration command can be
!! found below.


interface GigabitEthernet0/0/0/3.77
 service-policy output volvo-ce1
!!% 'qos-ea' detected the 'warning' condition 'shape average in child level is
not supported in this hardware version'
!
end
admin@ncs(config)#
```

Since the service-policy in our service template isn't supported by our real IOS-XR device, we will simply comment it out from the service template:

```
<!-- not supported in real IOS-XR device
<service-policy>
  <output>
    <name>{/name}-{$CE}</name>
  </output>
</service-policy>
-->
```

With the above changes, let's repeat the process again:

```
mpls-vpn$ ncs_cli -C -u admin

admin connected from 127.0.0.1 using console on WAITAI-M-72J2
admin@ncs# packages reload
…
admin@ncs(config)# vpn l3vpn volvo re-deploy
admin@ncs(config)#
System message at 2020-06-02 16:34:58...
Commit performed by admin via console using cli.
admin@ncs(config)# vpn l3vpn ford re-deploy
admin@ncs(config)#
System message at 2020-06-02 16:35:47...
Commit performed by admin via console using cli.
```

Both services are now up and running on a real IOS-XR CLI-based PE device.

3.2.3 Build a NETCONF NED

Instead of installing a pre-built NETCONF NED for IOS-XR as was done for the last example, let's go ahead and build one from scratch using the NETCONF Builder feature of NSO. I'll first add a device to NSO for the NETCONF interface of "pe0" which will be called "pe0-nc":

```
admin@ncs(config)# devices device pe0-nc
admin@ncs(config-device-pe0-nc)# authgroup default
admin@ncs(config-device-pe0-nc)# address 10.147.46.208
admin@ncs(config-device-pe0-nc)# port 33880
admin@ncs(config-device-pe0-nc)# state admin-state unlocked
admin@ncs(config-device-pe0-nc)# device-type netconf ned-id netconf
admin@ncs(config-device-pe0-nc)# commit
Commit complete.
admin@ncs(config-device-pe0-nc)# ssh fetch-host-keys
result updated
fingerprint {
    algorithm ssh-rsa
    value fb:44:4a:d7:e5:74:87:12:cc:72:7a:cb:ff:53:a0:0f
}
fingerprint {
    algorithm ssh-dss
    value 0c:3d:69:eb:e2:95:b4:b4:d2:89:6f:44:92:a3:9c:db
}
admin@ncs(config-device-pe0-nc)# top
```

The ned-id of netconf is being used temporarily to allow a NETCONF NED to be built. Without assigning a ned-id of a real NETCONF NED, NSO won't be able to do much else with the NETCONF device.

We'll now turn to the NETCONF NED Builder to build a mini version of the IOS-XR NETCONF NED with only a subset of the YANG modules supported by the IOS-XR device. Since I know which YANG modules are being used by the service template, I will only select those to be included in the NETCONF NED. By default, all dependent YANG modules will also be automatically included in the NETCONF NED.

```
admin@ncs# devtools true
admin@ncs# config
dmin@ncs(config)# netconf-ned-builder project cisco-xr-mini 1.0 device pe0-nc
local-user admin vendor Cisco
admin@ncs(config-project-xr-mini/1.0)# commit
Commit complete.
admin@ncs(config-project-xr-mini/1.0)# top
admin@ncs(config)# exit
admin@ncs# show netconf-ned-builder project cisco-xr-mini
netconf-ned-builder project cisco-xr-mini 1.0
 download-cache-path /Users/waitai1/Tail-f/nso-dd/temp-2/mpls-vpn/state/net-
conf-ned-builder/cache/cisco-xr-mini-nc-1.0
 ned-directory-path  /Users/waitai1/Tail-f/nso-dd/temp-2/mpls-vpn/state/net-
conf-ned-builder/cisco-xr-mini-nc-1.0
admin@ncs# netconf-ned-builder project cisco-xr-mini 1.0 fetch-module-list
admin@ncs# show netconf-ned-builder project cisco-xr-mini 1.0 module
module CISCO-ENTITY-FRU-CONTROL-MIB 2003-11-24
 namespace http://tail-f.com/ns/mibs/CISCO-ENTITY-FRU-CONTROL-MIB/200311240000Z
 location  [ NETCONF ]
module Cisco-IOS-XR-Subscriber-infra-subdb-oper 2019-04-05
 namespace http://cisco.com/ns/yang/Cisco-IOS-XR-Subscriber-infra-subdb-oper
 location  [ NETCONF ]
 submodule Cisco-IOS-XR-Subscriber-infra-subdb-oper-sub1 2019-04-05
  location [ NETCONF ]
 submodule Cisco-IOS-XR-Subscriber-infra-subdb-oper-sub2 2019-04-05
location [ NETCONF ]
...
admin@ncs# netconf-ned-builder project cisco-xr-mini 1.0 module Cisco-IOS-XR-
ifmgr-cfg 2019-04-05 select
admin@ncs# netconf-ned-builder project cisco-xr-mini 1.0 module Cisco-IOS-XR-
infra-rsi-cfg 2019-10-31 select
admin@ncs# netconf-ned-builder project cisco-xr-mini 1.0 module Cisco-IOS-XR-
ipv4-io-cfg 2019-04-05 select
admin@ncs# netconf-ned-builder project cisco-xr-mini 1.0 module Cisco-IOS-XR-
l2-eth-infra-cfg 2019-04-05 select
admin@ncs# netconf-ned-builder project cisco-xr-mini 1.0 module Cisco-IOS-XR-
ipv4-bgp-cfg 2019-08-31 select
admin@ncs# netconf-ned-builder project cisco-xr-mini 1.0 module Cis-
co-IOS-XR-policy-repository-cfg 2019-04-05 select
admin@ncs# show netconf-ned-builder project xr-mini 1.0 module status
NAME                                  REVISION    STATUS
-----------------------------------------------------------------------
Cisco-IOS-XR-ifmgr-cfg                2019-04-05  selected,downloaded
Cisco-IOS-XR-infra-rsi-cfg            2019-10-31  selected,downloaded
Cisco-IOS-XR-ipv4-bgp-cfg             2019-08-31  selected,downloaded
Cisco-IOS-XR-ipv4-bgp-datatypes       2019-08-31  selected,downloaded
```

```
Cisco-IOS-XR-ipv4-io-cfg              2019-04-05  selected,downloaded
Cisco-IOS-XR-l2-eth-infra-cfg         2019-04-05  selected,downloaded
Cisco-IOS-XR-l2-eth-infra-datatypes   2019-04-05  selected,downloaded
Cisco-IOS-XR-l2vpn-cfg                2019-12-20  selected,downloaded
Cisco-IOS-XR-policy-repository-cfg    2019-04-05  selected,downloaded
Cisco-IOS-XR-snmp-agent-cfg           2019-10-31  selected,downloaded
Cisco-IOS-XR-types                    2019-12-03  selected,downloaded
cisco-semver                          2019-03-13  selected,downloaded
ietf-inet-types                       2013-07-15  selected,downloaded
ietf-yang-types                       2013-07-15  selected,downloaded

admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# netconf-ned-builder project cisco-xr-mini 1.0 build-ned
admin@ncs(config)# exit
admin@ncs# show netconf-ned-builder project cisco-xr-mini 1.0 build-status
build-status success
admin@ncs# netconf-ned-builder project cisco-xr-mini 1.0 export-ned to-directo-
ry /tmp
tar-file /tmp/ncs-5.3.1.1-cisco-xr-mini-nc-1.0.tar.gz
admin@ncs# exit
mpls-vpn$ mv /tmp/ncs-5.3.1.1-cisco-xr-mini-nc-1.0.tar.gz packages
mpls-vpn$ ncs_cli -C -u admin

admin connected from 127.0.0.1 using console on WAITAI-M-72J2
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-xr-mini-nc-1.0
    result true
}
```

After the NETCONF NED has been loaded, the ned-id of “pe0-nc” needs to be updated:

```
admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# devices device pe0-nc
admin@ncs(config-device-pe0-nc)# device-type netconf ned-id cisco-xr-mini-nc-1.0
admin@ncs(config-device-pe0-nc)# commit
Commit complete.
admin@ncs(config-device-pe0-nc)# sync-from
result true
admin@ncs(config-device-pe0-nc)# top
```

If you have been following along in your own setup, you have seen that it only took a couple of minutes to build a NETCONF NED including some manual typing. NSO can now communicate with the real IOS-XR device using the NETCONF NED and perform both edit and get operations on the YANG data models that have been included in the NED.

3.2.4 Generate a service template for the NETCONF device

Following the same approach as the last example, we will take advantage of the fact that changes made to the CLI based "pe0" aren't being synchronized to the NETCONF based "pe0-nc". I will first perform an un-deploy to remove the service related configuration and then perform a re-deploy to set up the service related configuration on "pe0". I'll then use compare-config to determine the NETCONF payload that is required to set up the equivalent configuration.

```
admin@ncs(config)# vpn l3vpn volvo check-sync
in-sync true
admin@ncs(config)# vpn l3vpn volvo un-deploy
admin@ncs(config)# devices device pe0-nc sync-from
admin@ncs(config)# vpn l3vpn volvo re-deploy
admin@ncs(config)#
System message at 2020-06-01 10:21:02...
Commit performed by admin via console using cli.
admin@ncs(config)# devices device pe0-nc compare-config outformat xml
diff
<devices xmlns="http://tail-f.com/ns/ncs">
  <device>
    <name>pe0-nc</name>
    <config>
      <interface-configurations
          xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
        <interface-configuration>
          <active>act</active>
          <interface-name>GigabitEthernet0/0/0/3.77</interface-name>
…
```

The process here is the same as the last example. The XML diffs generated by the compare-config operation is essentially what the "volvo" service needs to push out to the "pe0-nc" device over NETCONF. Since the original service was based on an XML template towards the CLI NED, all I need to do is to add these XML diffs to that template and parameterize it using the same variables as used by the CLI template. The difference this time is that we will be working with a PE specific service template called l3vpn-pe.xml. There will be two types of parameters that will replace the hardcoded values. They are the ones populated by the service code in Java in the form of ($) and the service mapping logic in the form of XPath expressions enclosed within { }. An "IOS-XR NETCONF" specific chunk of XML blocks will then be added to l3vpn-pe.xml.

The complete NETCONF template for IOS-XR can be found in Appendix B

3.2.5 Reconfigure the l3vpn service to use the NETCONF device

After adding the new service template to cover the NETCONF interface for "pe0-nc", the "packages reload" command needs to be performed for the new l3vpn-pe.xml template to take effect. For this example, we'll then update "pe0" to connect to the NETCONF interface of the real IOS-XR device. Before we do that, we will first un-deploy both services to allow the services related configuration to be unconfigured on the device. Since NSO has configuration data for "pe0" in its datastore, "pe0"'s configuration needs to be wiped clean in NSO before its ned-id can be updated.

```
admin@ncs(config)# vpn l3vpn volvo un-deploy
admin@ncs(config)# vpn l3vpn ford un-deploy
admin@ncs(config)# devices device pe0
admin@ncs(config-device-pe0)# no config
admin@ncs(config-device-pe0)# commit no-networking
Commit complete.
admin@ncs(config-device-pe0)# port 33880
admin@ncs(config-device-pe0)# device-type netconf ned-id cisco-xr-mini-nc-1.0
admin@ncs(config-device-pe0)# commit
Commit complete.
admin@ncs(config-device-pe0)# ssh fetch-host-keys
result unchanged
fingerprint {
    algorithm ssh-rsa
    value fb:44:4a:d7:e5:74:87:12:cc:72:7a:cb:ff:53:a0:0f
}
fingerprint {
    algorithm ssh-dss
    value 0c:3d:69:eb:e2:95:b4:b4:d2:89:6f:44:92:a3:9c:db
}
admin@ncs(config-device-pe0)# sync-from
result true
admin@ncs(config-device-pe0)# exit
```

We can now re-deploy both services to verify that they are working properly:

```
admin@ncs(config)# vpn l3vpn volvo re-deploy dry-run
…
admin@ncs(config)# vpn l3vpn volvo re-deploy
admin@ncs(config)#
System message at 2020-06-01 10:34:16...
Commit performed by admin via console using cli.
admin@ncs(config)# vpn l3vpn ford check-sync
in-sync false
admin@ncs(config)# vpn l3vpn ford re-deploy dry-run
…
admin@ncs(config)# vpn l3vpn ford re-deploy
```

Inspect the output of “re-deploy dry-run” to make sure that the necessary configuration is being sent to the NETCONF flavor of “pe0”. You can perform an un-deploy followed by a re-deploy to verify that there are no errors. As an exercise for the reader, you can also create a device in NSO for the CLI interface of the real PE device and use that to confirm the device configuration being updated by the service application on the NETCONF device by using the compare-config function when going from a un-deployed state to a deployed state.

The NED migration is now done. It’s almost as simple as the last example.

4 Troubleshooting

There are 2 areas in the documented NED migration procedure where you may require some help. They are the parameterization of NETCONF templates and the building of NETCONF NEDs.

4.1 Debugging templates

To debug service templates, there is a cli pipe command called “debug” that can be applied on either “template” or “xpath”:

```
admin@ncs(config)# commit dry-run | debug template
admin@ncs(config)# vpn l3vpn re-deploy dry-run | debug xpath
```

For all templates invoked, “debug template” will output XPath expression results from the template, under which context it is evaluated, what operation is used, and how it affects the configuration. The command can be narrowed down to only show debugging information for a specific template:

```
admin@ncs(config)# commit dry-run | debug template l3vpn
```

“debug xpath” will output all XPath evaluations for the transaction and is not limited to the XPath expressions inside templates.

Template and XPath debugging can be combined:

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

4.2 Troubleshooting NETCONF NED building issues

You can refer to section 5.3 of the “NETCONF & YANG Automation Testing User Guide v3” available for download at https://info.tail-f.com/netconf_yang_automation_testing for information on how to troubleshoot NETCONF NED building issues.

4.3 Downloading of YANG modules from IOS-XR issue

If you observe the following alarm message in ncs_cli:

```
*** ALARM connection-failure: Failed to authenticate towards device pe0-nc: SSH
subsystem not supported
```

while the YANG modules are being downloaded from the IOS-XR device using the NETCONF NED Builder, change the device configuration for the IOS-XR device to match the following:

```
RP/0/RP0/CPU0:xrv9000#show running-config ssh
Fri Jun 12 20:56:28.860 UTC
ssh timeout 120
ssh server rate-limit 600
ssh server session-limit 100
```

For More Information

More information about NSO:
https://cisco.com/go/nso

NSO on DevNet:
https://developer.cisco.com/nso

NSO on GitHub:
https://github.com/NSO-developer

Appendix A NETCONF Template for the simple-mpls-vpn example

```
<!-- PE Template for IOS-XR over NETCONF -->
<interface-configurations
    xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg"
    tags="merge">
  <interface-configuration tags="merge">
    <active>act</active>
    <interface-name>GigabitEthernet{link/interface-number}</interface-name>
    <description>link to CE</description>
    <vrf xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-rsi-cfg">
      {string(/name)}
    </vrf>
    <ipv4-network xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-io-cfg">
      <addresses>
        <primary>
          <address>{ip-address}</address>
          <netmask>255.255.255.252</netmask>
        </primary>
      </addresses>
    </ipv4-network>
  </interface-configuration>
</interface-configurations>
<vrfs xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-rsi-cfg"
    tags="merge">
  <vrf>
    <vrf-name>{string(/name)}</vrf-name>
    <create/>
    <afs>
      <af>
        <af-name>ipv4</af-name>
        <saf-name>unicast</saf-name>
        <topology-name>default</topology-name>
        <create/>
        <bgp xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-bgp-cfg">
          <import-route-targets>
            <route-targets>
              <route-target>
                <type>as</type>
                <as-or-four-byte-as>
                  <as-xx>0</as-xx>
                  <as>{../as-number}</as>
                  <as-index>1</as-index>
                  <stitching-rt>0</stitching-rt>
                </as-or-four-byte-as>
              </route-target>
            </route-targets>
          </import-route-targets>
          <export-route-targets>
            <route-targets>
              <route-target>
                <type>as</type>
                <as-or-four-byte-as>
                  <as-xx>0</as-xx>
```

```
                    <as>{../as-number}</as>
                    <as-index>1</as-index>
                    <stitching-rt>0</stitching-rt>
                  </as-or-four-byte-as>
                </route-target>
              </route-targets>
            </export-route-targets>
          </bgp>
        </af>
      </afs>
    </vrf>
  </vrfs>
  <bgp
      xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-bgp-cfg"
      tags="merge">
    <instance>
      <instance-name>default</instance-name>
      <instance-as>
        <as>0</as>
        <four-byte-as>
          <as>100</as>
          <vrfs tags="merge">
            <vrf>
              <vrf-name>{string(/name)}</vrf-name>
              <vrf-global>
                <route-distinguisher>
                  <type>as</type>
                  <as-xx>0</as-xx>
                  <as>{../as-number}</as>
                  <as-index>1</as-index>
                </route-distinguisher>
                <vrf-global-afs>
                  <vrf-global-af>
                    <af-name>ipv4-unicast</af-name>
                    <enable/>
                  </vrf-global-af>
                </vrf-global-afs>
                <exists/>
              </vrf-global>
              <vrf-neighbors>
                <vrf-neighbor>
                  <neighbor-address>
                    {../ce/link/ip-address}
                  </neighbor-address>
                  <vrf-neighbor-afs>
                    <vrf-neighbor-af>
                      <af-name>ipv4-unicast</af-name>
                      <as-override>true</as-override>
                      <activate/>
                    </vrf-neighbor-af>
                  </vrf-neighbor-afs>
                  <remote-as>
                    <as-xx>0</as-xx>
                    <as-yy>{../../as-number}</as-yy>
```

```
                </remote-as>
              </vrf-neighbor>
            </vrf-neighbors>
          </vrf>
        </vrfs>
        <default-vrf>
          <global>
            <global-afs>
              <global-af>
                <af-name>vpnv4-unicast</af-name>
                <enable/>
              </global-af>
            </global-afs>
          </global>
        </default-vrf>
        <bgp-running/>
      </four-byte-as>
    </instance-as>
  </instance>
</bgp>
<!-- End of PE Template for IOS-XR over NETCONF -->
```

Appendix B NETCONF Template for the mpls-vpn example

```
<!—PE Template for IOS-XR over NETCONF -->
          <interface-configurations
              xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
            <interface-configuration>
              <active>act</active>
              <interface-name>
                GigabitEthernet{substring($PE_INT_NAME,16)}.{$VLAN_ID}
              </interface-name>
              <interface-mode-non-physical>
                Default
              </interface-mode-non-physical>
              <description>Link to CE / {$CE} - {$CE_INT_NAME}</description>
              <vrf xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-rsi-cfg">
                {string(/name)}
              </vrf>
              <ipv4-network
                  xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-io-cfg">
                <addresses>
                  <primary>
                    <address>{$LINK_PE_ADR}</address>
                    <netmask>{$LINK_MASK}</netmask>
                  </primary>
                </addresses>
              </ipv4-network>
              <vlan-sub-configuration
                  xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-l2-eth-infra-cfg">
                <vlan-identifier>
                  <vlan-type>vlan-type-dot1q</vlan-type>
                  <first-tag>{$VLAN_ID}</first-tag>
                </vlan-identifier>
              </vlan-sub-configuration>
            </interface-configuration>
          </interface-configurations>
          <vrfs xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-rsi-cfg">
            <vrf>
              <vrf-name>{/name}</vrf-name>
              <create/>
              <afs>
                <af>
                  <af-name>ipv4</af-name>
                  <saf-name>unicast</saf-name>
                  <topology-name>default</topology-name>
                  <create/>
                  <bgp xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-bgp-cfg">
                    <import-route-targets>
                      <route-targets>
                        <route-target>
                          <type>as</type>
                          <as-or-four-byte-as>
                            <as-xx>0</as-xx>
                            <as>{/route-distinguisher}</as>
                            <as-index>1</as-index>
```

```
                    <stitching-rt>0</stitching-rt>
                  </as-or-four-byte-as>
                </route-target>
              </route-targets>
            </import-route-targets>
            <export-route-targets>
              <route-targets>
                <route-target>
                  <type>as</type>
                  <as-or-four-byte-as>
                    <as-xx>0</as-xx>
                    <as>{/route-distinguisher}</as>
                    <as-index>1</as-index>
                    <stitching-rt>0</stitching-rt>
                  </as-or-four-byte-as>
                </route-target>
              </route-targets>
            </export-route-targets>
          </bgp>
        </af>
      </afs>
    </vrf>
  </vrfs>
  <bgp xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-bgp-cfg">
    <instance>
      <instance-name>default</instance-name>
      <instance-as>
        <as>0</as>
        <four-byte-as>
          <as>100</as>
          <vrfs>
            <vrf>
              <vrf-name>{/name}</vrf-name>
              <vrf-global>
                <route-distinguisher>
                  <type>as</type>
                  <as-xx>0</as-xx>
                  <as>{/route-distinguisher}</as>
                  <as-index>1</as-index>
                </route-distinguisher>
                <vrf-global-afs>
                  <vrf-global-af>
                    <af-name>ipv4-unicast</af-name>
                    <enable/>
                  </vrf-global-af>
                </vrf-global-afs>
                <exists/>
              </vrf-global>
              <vrf-neighbors>
                <vrf-neighbor>
                  <neighbor-address>{$LINK_CE_ADR}</neighbor-address>
                  <vrf-neighbor-afs>
                    <vrf-neighbor-af>
                      <af-name>ipv4-unicast</af-name>
```

```
                        <as-override>true</as-override>
                        <activate/>
                        <route-policy-in>{/name}</route-policy-in>
                        <route-policy-out>{/name}</route-policy-out>
                      </vrf-neighbor-af>
                    </vrf-neighbor-afs>
                    <remote-as>
                      <as-xx>0</as-xx>
                      <as-yy>{$CE_AS_NUM}</as-yy>
                    </remote-as>
                  </vrf-neighbor>
                </vrf-neighbors>
              </vrf>
            </vrfs>
            <default-vrf>
              <global>
                <global-afs>
                  <global-af>
                    <af-name>vpnv4-unicast</af-name>
                    <enable/>
                  </global-af>
                </global-afs>
              </global>
            </default-vrf>
            <bgp-running/>
          </four-byte-as>
        </instance-as>
      </instance>
    </bgp>
    <routing-policy
        xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-policy-repository-cfg">
      <route-policies>
        <route-policy>
          <route-policy-name>{/name}</route-policy-name>
          <rpl-route-policy>
            route-policy {/name}
            pass
            end-policy
          </rpl-route-policy>
        </route-policy>
      </route-policies>
    </routing-policy>
    <!-- End of PE Template for IOS-XR over NETCONF -->
```