Lab - Use NETCONF to Access an IOS XE Device (Instructor Version)

Instructor Note: Red font color or gray highlights indicate text that appears in the instructor copy only.

Objectives

Part 1: Build the Network and Verify Connectivity

Part 2: Use a NETCONF Session to Gather Information

Part 3: Use ncclient to Connect to NETCONF

Part 4: Use ncclient to Retrieve the Configuration

Part 5: Use ncclient to Configure a Device

Part 6: Challenge: Modify the Program Used in This Lab

Background / Scenario

The Network Configuration Protocol (NETCONF), defined in RFCs 4741 and 6241, uses YANG data models to communicate with various devices on the network. YANG (yet another next generation) is a data modeling language. This language defines the data that is sent over network management protocols, like NETCONF. When using NETCONF to access an IOS XE device, the data is returned in XML format.

In this lab, you will use a NETCONF client, ncclient, which is a Python module for client-side scripting. You will use ncclient to verify NETCONF is configured, retrieve a device configuration, and modify a device configuration.

Required Resources

- 1 PC with operating system of your choice
- Virtual Box or VMWare
- DEVASC Virtual Machine
- CSR1kv Virtual Machine

Instructions

Part 1: Launch the VMs and Verify Connectivity

In this Part, you launch the two VMs and verify connectivity. You will then establish a secure shell (SSH) connection.

Step 1: Launch the VMs

If you have not already completed the **Lab - Install the Virtual Machine Lab Environment** and the **Lab - Install the CSR1kv VM**, do so now. If you have already completed these labs, launch both the DEVASC VM and CSR1000v VM now.

Step 2: Verify connectivity between the VMs.

a. In the CSR1kv VM, press Enter to get a command prompt and then use **show ip interface brief** to verify that the IPv4 address is 192.168.56.101. If the address is different, make a note of it.

b. Open a terminal in VS Code in the DEVASC VM.

c. Ping the CSR1kv to verify connectivity. You should have already done this previously in the installation labs. If you are not able to ping, then revisit those labs listed above in Part 1a.

```
devasc@labvm:~$ ping -c 5 192.168.56.101
PING 192.168.56.101 (192.168.56.101) 56(84) bytes of data.
64 bytes from 192.168.56.101: icmp_seq=1 ttl=254 time=1.37 ms
64 bytes from 192.168.56.101: icmp_seq=2 ttl=254 time=1.15 ms
64 bytes from 192.168.56.101: icmp_seq=3 ttl=254 time=0.981 ms
64 bytes from 192.168.56.101: icmp_seq=4 ttl=254 time=1.01 ms
64 bytes from 192.168.56.101: icmp_seq=5 ttl=254 time=1.14 ms

--- 192.168.56.101 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 0.981/1.130/1.365/0.135 ms
devasc@labvm:~$
```

Step 3: Verify SSH connectivity to the CSR1kv VM.

a. In the terminal for the DEVASC VM, SSH to the CSR1kv VM with the following command:

```
devasc@labvm:~$ ssh  cisco@192.168.56.101
```

Note: The first time you SSH to the CSR1kv, your DEVASC VM warns you about the authenticity of the CSR1kv. Because you trust the CSR1kv, answer yes to the prompt.

```
The authenticity of host '192.168.56.101 (192.168.56.101)' can't be
established.
RSA key fingerprint is SHA256:HYv9K5Biw7PFiXeoCDO/LTqs3EfZKBuJdiPo34VXDUY.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.56.101' (RSA) to the list of known hosts.
```

b. Enter **cisco123!** as the password and you should now be at the privileged EXEC command prompt for the CSR1kv.

```
Password:


CSR1kv#
```

c. Leave the SSH session open for the next Part.

Part 2: Use a NETCONF Session to Gather Information

In this Part, you will verify that NETCONF is running, enable NETCONF if it is not, and verify that NETCONF is ready for an SSH connection. YSou will then connect to the NETCONF process, start a NETCONF session, gather interface information, and close the session.

Step 1: Check if NETCONF is running on the CSR1kv.

a. NETCONF may already be running if another student enabled it, or if a later IOS version enables it by default. From your SSH session with the CSR1kv, use the **show platform software yang-management process** command to see if the NETCONF SSH daemon (**ncsshd**) is running.

```
CSR1kv# show platform software yang-management process
confd            : Running
nesd             : Running
syncfd           : Running
ncsshd           : Running
dmiauthd         : Running
```

```
nginx            : Running
ndbmand          : Running
pubd             : Running

CSR1kv#
```

b. If NETCONF is not running, as shown in the output above, enter the global configuration command **netconf-yang**.

```
CSR1kv# config t
CSR1kv (config)# netconf-yang
```

c. Type **exit** to close the SSH session.

Step 2: Access the NETCONF process through an SSH terminal.

In this step, you will re-establish an SSH session with the CSR1kv. But this time, you will specify the NETCONF port 830 and send **netconf** as a subsystem command.

Note: For more information on these options, explore the manual pages for SSH (**man ssh**).

a. Enter the following command in a terminal window. You can use the up arrow to recall the latest SSH command and just add the **-p** and **-s** options as shown. Then, enter **cisco123!** as the password.

```
devasc@labvm:~$ ssh cisco@192.168.56.101 -p 830 -s netconf
cisco@192.168.56.101's password:
```

b. The CSR1kv will respond with a **hello** message that includes over 400 lines of output listing all of its NETCONF capabilities. The end of NETCONF messages are identified with **]]>]]>**.

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
<capability>urn:ietf:params:netconf:base:1.0</capability>
<capability>urn:ietf:params:netconf:base:1.1</capability>
<capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
<capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
(output omitted)
      </capability>
</capabilities>
<session-id>20</session-id></hello>]]>]]>
```

Step 3: Start a NETCONF session by sending a hello message from the client.

To start a NETCONF session, the client needs to send its own hello message. The hello message should include the NETCONF base capabilities version the client wants to use.

a. Copy and paste the following XML code into the SSH session. Notice that the end of the client hello message is identified with a **]]>]]>**.

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <capabilities>
   <capability>urn:ietf:params:netconf:base:1.0</capability>
 </capabilities>
</hello>
]]>]]>
```

b. Switch to the CSR1kv VM and use the **show netconf-yang sessions** command to verify that a NETCONF session has been started. If the CSR1kv VM screen is dark, press **Enter** to wake it up.

```
CSR1kv> en
CSRk1v# show netconf-yang sessions
R: Global-lock on running datastore
C: Global-lock on candidate datastore
S: Global-lock on startup datastore

Number of sessions : 1

session-id  transport    username             source-host           global-lock
--------------------------------------------------------------------------------
20          netconf-ssh  cisco                192.168.56.1          None

CSR1kv#
```

Step 4: Send RPC messages to an IOS XE device.

During an SSH session, a NETCONF client can use Remote Procedure Call (RPC) messages to send NETCONF operations to the IOS XE device. The table lists some of the more common NETCONF operations.

Operation	Description
<get>	Retrieve running configuration and device state information
<get-config>	Retrieve all or part of a specified configuration data store
<edit-config>	Loads all or part of a configuration to the specified configuration data store
<copy-config>	Replace an entire configuration data store with another
<delete-config>	Delete a configuration data store
<commit>	Copy candidate data store to running data store
<lock> / <unlock>	Lock or unlock the entire configuration data store system
<close-session>	Graceful termination of NETCONF session
<kill-session>	Forced termination of NETCONF session

a. Copy and paste the following RPC **get** message XML code into the terminal SSH session to retrieve information about the interfaces on R1.

```
<rpc message-id="103" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <get>
  <filter>
    <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"/>
   </filter>
 </get>
</rpc>
]]>]]>
```

b. Recall that XML does not require indention or white space. Therefore, the CSR1kv will return a long string of XML data.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-
id="103"><data><interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-
interfaces"><interface><name>GigabitEthernet1</name><description>VBox</description><ty
pe xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-
type">ianaift:ethernetCsmacd</type><enabled>true</enabled><ipv4
xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"></ipv4><ipv6
xmlns="urn:ietf:params:xml:ns:yang:ietf-
ip"></ipv6></interface></interfaces></data></rpc-reply>]]>]]>
```

c. Copy the XML that was returned, but do not include the final “]]>]]>” characters. These characters are not part of the XML that is returned by the router.

d. Search the internet for “prettify XML”. Find a suitable site and use its tool to transform your XML into a more readable format, such as the following:

```
<?xml version="1.0"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="103">
  <data>
    <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
      <interface>
        <name>GigabitEthernet1</name>
        <description>VBox</description>
        <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-
type">ianaift:ethernetCsmacd</type>
        <enabled>true</enabled>
        <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"/>
        <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"/>
      </interface>
    </interfaces>
  </data>
</rpc-reply>
```

Step 5: Close the NETCONF session.

a. To close the NETCONF session, the client needs to send the following RPC message:

```
<rpc message-id="9999999" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <close-session />
</rpc>
```

b. After a few seconds, you will be returned to the terminal prompt. Return to the CSR1kv prompt and show the open netconf sessions. You will see that the session has been closed.

```
CSR1kv# show netconf-yang sessions
There are no active sessions

CSR1kv#
```

Part 3: Use ncclient to Connect to NETCONF

Working with NETCONF does not require working with raw NETCONF RPC messages and XML. In this Part, you will learn how to use the **ncclient** Python module to easily interact with network devices using NETCONF. In addition, you will learn how to identify which YANG models are supported by the device. This information is helpful when building a production network automation system that requires specific YANG models to be supported by the given network device.

Step 1: Verify that ncclient is installed and ready for use.

In a DEVASC-VM terminal, enter the command **pip3 list --format=columns** to see all Python modules currently installed. Pipe the output to **more**. Your output may differ from the following. But you should see **ncclient** listed, as shown. If not, use the **pip3 install ncclient** command to install it.

```
devasc@labvm:~$ pip3 list --format=columns | more
Package               Version
--------------------- -----------
ansible               2.9.6
apache-libcloud       2.8.0
appdirs               1.4.3
argcomplete           1.8.1
astroid               2.3.3
(output omitted)
ncclient              0.6.7
netaddr               0.7.19
netifaces             0.10.4
netmiko               3.1.0
ntlm-auth             1.1.0
oauthlib              3.1.0
(output omitted)
xmltodict             0.12.0
zipp                  1.0.0
devasc@labvm:~$
```

Step 2: Create a script to use ncclient to connect to the NETCONF service.

The ncclient module provides a **manager** class with a **connect()** method to setup the remote NETCONF connections. After a successful connection, the returned object represents the NETCONF connection to the remote device.

a. In VS code, click **File > Open Folder...** and navigate to the **devnet-src** directory. Click **OK**.

b. Open a terminal window in VS Code: **Terminal > New Terminal**.

c. Create a subdirectory called **netconf** in the **/devnet-src** directory.

```
devasc@labvm:~/labs/devnet-src$ mkdir netconf
devasc@labvm:~/labs/devnet-src$
```

d. In the **EXPLORER** pane under **DEVNET-SRC**, right-click the **netconf** directory and choose **New File**.

e. Name the file **ncclient-netconf.py**.

f. In your script file, import the manager class from the **ncclient** module. Then create a variable **m** to represent the **connect()** method. The **connect()** method includes all the information that is required to connect to the NETCONF service running on the CSR1kv. Note that the port is 830 for NETCONF.

```
from ncclient import manager

m = manager.connect(
    host="192.168.56.101",
    port=830,
    username="cisco",
    password="cisco123!",
```

```
    hostkey_verify=False
    )
```

If the **hostkey_verify** is set to True, the CSR1kv will ask you to verify the SSH fingerprint. In a lab environment, it is safe to set this value to False, as we have done here.

g. Save and run the program to verify that there are no errors. You will not see any output yet.

```
devasc@labvm:~/labs/devnet-src$ cd netconf/
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
devasc@labvm:~/labs/devnet-src/netconf$
```

h. You can verify that the CSR1kv accepted the request for a NETCONF session. There should be an **%DMI-5-AUTH_PASSED** syslog message in the CSR1kv VM. If the screen is black, press **Enter** to wake the router. The syslog message can be seen above the banner.

Step 3: Add a print function to the script so that the NETCONF capabilities for the CSR1kv are listed.

The **m** object returned by the **manager.connect()** function represents the NETCONF remote session. As you saw previously, in every NETCONF session, the server first sends its capabilities, which is a list, in XML format, of supported YANG models. With the **ncclient** module, the received list of capabilities is stored in the **m.server_capabilities** list.

a. Use a **for** loop and a **print** function to display the device capabilities:

```
print("#Supported Capabilities (YANG models):")
for capability in m.server_capabilities:
    print(capability)
```

b. Save and run the program. The output is the same output you got from sending the complex **hello** message in previously, but without the opening and closing <capability> XML tag on each line.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
#Supported Capabilities (YANG models):
urn:ietf:params:netconf:base:1.0
urn:ietf:params:netconf:base:1.1
urn:ietf:params:netconf:capability:writable-running:1.0
urn:ietf:params:netconf:capability:xpath:1.0
<output omitted>
urn:ietf:params:xml:ns:netconf:base:1.0?module=ietf-netconf&revision=2011-06-01
urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults?module=ietf-netconf-with-
defaults&revision=2011-06-01

        urn:ietf:params:netconf:capability:notification:1.1

devasc@labvm:~/labs/devnet-src/netconf$
```

Part 4: Use ncclient to Retrieve the Configuration

In this Part, you will use the NETCONF **ncclient** to retrieve the configuration for the CSR1kv, use the **xml.dom.minidom** module to format the configuration, and use a filter with get_config() to retrieve a portion of the running configuration.

Step 1: Use the get_config() function to retrieve the running configuration for R1.

a. If you want to skip displaying capabilities output (400+ lines), comment out the block of statements that print the capabilities, as shown in the following:

```
'''
print("#Supported Capabilities (YANG models):")
for capability in m.server_capabilities:
    print(capability)
'''
```

b. You can use the **get_config()** method of the **m** NETCONF session object to retrieve the configuration for the CSR1kv. The **get_config()** method expects a source string parameter that specifies the source NETCONF datastore. Use a print function to display the results. The only NETCONF datastore currently on the CSR1kv is the **running** datastore. You can verify this with the **show netconf-yang datastores** command.

```
netconf_reply = m.get_config(source="running")
print(netconf_reply)
```

c. Save and run your program. The output will be well over 100 lines, so IDLE may compress them. Double-click the **Squeezed text** message in the IDLE shell window to expand the output.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-
id="urn:uuid:3f31bedc-5671-47ca-9781-4d3d7aadae24"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"><data><native
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native"><version>16.9</version><boot-
start-marker/><boot-end-marker/><banner><motd><banner>
(output omitted)
devasc@labvm:~/labs/devnet-src/netconf$
```

d. Notice that the returned XML is not formatted. You can copy it out to the same site you found earlier to prettify the XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:3f31bedc-5671-
47ca-9781-4d3d7aadae24">
  <data>
    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
      <version>16.9</version>
      <boot-start-marker/>
      <boot-end-marker/>
      <banner>
        <motd>
          <banner>^C</banner>
        </motd>
      </banner>
      <service>
        <timestamps>
          <debug>
            <datetime>
              <msec/>
            </datetime>
          </debug>
          <log>
            <datetime>
              <msec/>
```

```
            </datetime>
          </log>
        </timestamps>
      </service>
      <platform>
        <console xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-platform">
          <output>virtual</output>
        </console>
      </platform>
      <hostname>CSR1kv</hostname>
(output omitted)
```

Step 2: Use Python to prettify the XML.

Python has built in support for working with XML files. The **xml.dom.minidom** module can be used to prettify the output with the **toprettyxml()** function.

a. At the beginning of your script, add a statement to import the **xml.dom.minidom** module.

```
import xml.dom.minidom
```

b. Replace the simple print function **print(netconf_reply)** with a version that prints prettified XML output.

```
print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())
```

c. Save and run your program. XML is displayed in a more readable format.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
<?xml version="1.0" ?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:3a5f6abc-76b4-
436d-9e9a-7758091c28b7">
        <data>
                <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
                        <version>16.9</version>
                        <boot-start-marker/>
                        <boot-end-marker/>
                        <banner>
                                <motd>
                                        <banner>^C</banner>
                                </motd>
                        </banner>
(output omitted)
        </data>
</rpc-reply>

devasc@labvm:~/labs/devnet-src/netconf$
```

Step 3: Use a filter with get_config() to only retrieve a specific YANG model.

A network administrator may only want to retrieve a portion of the running configuration on a device. NETCONF supports returning only data that is defined in a filter parameter of the **get_conf()** function.

a. Create a variable called **netconf_filter** that only retrieves data defined by the Cisco IOS XE Native YANG model.

```
netconf_filter = """
```

```
<filter>
    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native" />
</filter>
"""
netconf_reply = m.get_config(source="running", filter=netconf_filter)
print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())
```

b. Save and run your program. The start of the output is the same, as shown below. However, only the <native> XML element is displayed this time. Previously, all the YANG models available on the CSR1kv were displayed. Filtering the retrieved data to only display the native YANG module significantly reduces your output. This is because the native YANG module only includes a subset of all the Cisco IOX XE YANG models.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
<?xml version="1.0" ?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:4da5b736-1d33-
47c3-8e3c-349414be0958">
        <data>
                <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
                        <version>16.9</version>
                        <boot-start-marker/>
                        <boot-end-marker/>
                        <banner>
                                <motd>
                                        <banner>^C</banner>
                                </motd>
                        </banner>
                        <service>
                                <timestamps>
                                        <debug>
                                                <datetime>
                                                        <msec/>
                                                </datetime>
                                        </debug>
                                        <log>
                                                <datetime>
                                                        <msec/>
                                                </datetime>
                                        </log>
                                </timestamps>
                        </service>
                        <platform>
                                <console xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-
platform">
                                        <output>virtual</output>
                                </console>
                        </platform>
                        <hostname>CSR1kv</hostname>
(output omitted)
                </native>
```

```
        </data>
</rpc-reply>

devasc@labvm:~/labs/devnet-src/netconf$
```

Part 5: Use ncclient to Configure a Device

In this Part, you will use **ncclient** to configure the CSR1kv using the **edit_config()** method of the **manager** module.

Step 1: Use ncclient to edit the hostname for the CSR1kv.

To update an existing setting in the configuration for the CSR1kv, you can extract the setting location from the configuration retrieved previously. For this step, you will set a variable to change the **<hostname>** value.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
<?xml version="1.0" ?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:4da5b736-1d33-
47c3-8e3c-349414be0958">
        <data>
                <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
(output omitted)
                        <hostname>CSR1kv</hostname>
(output omitted)
```

a. Previously, you defined a <filter> variable. To modify a device configuration, you will define a <config> variable. Add the following variable to your **ncclient_netconf.py** script. You can use NEWHOSTNAME or whatever hostname you wish.

```
netconf_hostname = """
<config>
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
     <hostname>NEWHOSTNAME</hostname>
  </native>
</config>
"""
```

b. Use the **edit_config()** function of the **m** NETCONF session object to send the configuration and store the results in the **netconf_reply** variable so that they can be printed. The parameters for the **edit_config()** function are as follows:

- **target** - the targeted NETCONF datastore to be updated
- **config** - the configuration modification that is to be sent

```
netconf_reply = m.edit_config(target="running", config=netconf_hostname)
```

c. The **edit_config()** function returns an XML RPC reply message with **<ok/>** indicating that the change was successfully applied. Repeat the previous print statement to display the results.

```
print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())
```

d. Save and run your program. You should get output similar to the output displayed below. You can also verify that the hostname has changed by switching to the CSR1kv VM.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
(output omitted)
<?xml version="1.0" ?>
```

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:e304b225-7951-
4029-afd5-59e8e7edbaa0">
        <ok/>
</rpc-reply>

devasc@labvm:~/labs/devnet-src/netconf$
```

e. Edit your script to change hostname back to CSR1kv. Save and run your program. You can also just comment out the code from the previous step if you want to avoid changing the hostname again.

Step 2: Use ncclient to create a new loopback interface on R1.

a. Create a new <config> variable to hold the configuration for a new loopback interface. Add the following to your **ncclient_netconf.py** script.

Note: You can use whatever **description** you want. However, only use alphanumeric characters or you will need to escape them with the backslash (\ ).

```
netconf_loopback = """
<config>
 <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
   <Loopback>
    <name>1</name>
    <description>My first NETCONF loopback</description>
    <ip>
     <address>
      <primary>
       <address>10.1.1.1</address>
       <mask>255.255.255.0</mask>
      </primary>
     </address>
    </ip>
   </Loopback>
  </interface>
 </native>
</config>
"""
```

b. Add the following **edit_config()** function to your **ncclient_netconf.py** to send the new loopback configuration to R1 and then print out the results.

```
netconf_reply = m.edit_config(target="running", config=netconf_loopback)
print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())
```

c. Save and run your program. You should get output similar to the following:

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
(output omitted)
<?xml version="1.0" ?>
```

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:98437f47-7a93-
4cac-9b9e-9bc8afc9dfa1">
       <ok/>
</rpc-reply>

devasc@labvm:~/labs/devnet-src/netconf$
```

d. On the CSR1kv, verify that the new loopback interface was created.

```
CSR1kv>en
CSR1kv# show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.101  YES DHCP   up                    up
Loopback1              10.1.1.1        YES other  up                    up
CSR1kv# show run | section interface Loopback1
interface Loopback1
 description My first NETCONF loopback
 ip address 10.1.1.1 255.255.255.0
CSR1kv#
```

Step 3: Attempt to create a new loopback interface with the same IPv4 address.

a. Create a new variable called **netconf_newloop**. It will hold a configuration that creates a new loopback 2 interface but with the same IPv4 address as on loopback 1: 10.1.1.1 /24. At the router CLI, this would create an error because of the attempt to assign a duplicate IP address to an interface.

```
netconf_newloop = """
<config>
 <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
   <Loopback>
    <name>2</name>
    <description>My second NETCONF loopback</description>
    <ip>
     <address>
      <primary>
       <address>10.1.1.1</address>
       <mask>255.255.255.0</mask>
      </primary>
     </address>
    </ip>
   </Loopback>
  </interface>
 </native>
</config>
"""
```

b. Add the following **edit_config()** function to your **ncclient_netconf.py** to send the new loopback configuration to the CSR1kv. You do not need a print statement for this step.

```
netconf_reply = m.edit_config(target="running", config=netconf_newloop)
```

c. Save and run the program. You should get error output similar to the following with the RPCError message **Device refused one or more commands**.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
Traceback (most recent call last):
  File "ncclient-netconf.py", line 80, in <module>
    netconf_reply = m.edit_config(target="running", config=netconf_newloop)
  File "/home/devasc/.local/lib/python3.8/site-packages/ncclient/manager.py", line
231, in execute
    return cls(self._session,
  File "/home/devasc/.local/lib/python3.8/site-packages/ncclient/operations/edit.py",
line 69, in request
    return self._request(node)
  File "/home/devasc/.local/lib/python3.8/site-packages/ncclient/operations/rpc.py",
line 348, in _request
    raise self._reply.error
ncclient.operations.rpc.RPCError: inconsistent value: Device refused one or more
commands
devasc@labvm:~/labs/devnet-src/netconf$
```

d. NETCONF will not apply any of the configuration that is sent if one or more commands are rejected. To verify this, enter the **show ip interface brief** command on R1. Notice that your new interface was not created.

```
CSR1kv# show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.101  YES DHCP   up                    up
Loopback1              10.1.1.1        YES other  up                    up
```

Part 6: Challenge: Modify the Program Used in This Lab

The following is the complete program that was created in this lab with none of the code commented out so you can run the script without error. Your script may look different. Practice your Python skills by modifying the program to send different verification and configuration commands.

```
from ncclient import manager
import xml.dom.minidom

m = manager.connect(
    host="192.168.56.101",
    port=830,
    username="cisco",
    password="cisco123!",
    hostkey_verify=False
    )

print("#Supported Capabilities (YANG models):")
for capability in m.server_capabilities:
    print(capability)
```

```
netconf_reply = m.get_config(source="running")
print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())

netconf_filter = """
<filter>
    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native" />
</filter>
"""
netconf_reply = m.get_config(source="running", filter=netconf_filter)
print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())

netconf_hostname = """
<config>
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
     <hostname>CSR1kv</hostname>
  </native>
</config>
"""
netconf_reply = m.edit_config(target="running", config=netconf_hostname)
print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())

netconf_loopback = """
<config>
 <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
   <Loopback>
    <name>1</name>
    <description>My NETCONF loopback</description>
    <ip>
     <address>
      <primary>
       <address>10.1.1.1</address>
       <mask>255.255.255.0</mask>
      </primary>
     </address>
    </ip>
   </Loopback>
  </interface>
 </native>
</config>
"""
netconf_reply = m.edit_config(target="running", config=netconf_loopback)
print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())

netconf_newloop = """
```

```
<config>
 <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
   <Loopback>
    <name>2</name>
    <description>My second NETCONF loopback</description>
    <ip>
     <address>
      <primary>
       <address>10.1.1.1</address>
       <mask>255.255.255.0</mask>
      </primary>
     </address>
    </ip>
   </Loopback>
  </interface>
 </native>
</config>
"""
netconf_reply = m.edit_config(target="running", config=netconf_newloop)
```