Lab – NETCONF w/Python: Device Configuration

Objectives

Part 1: Retrieve the IOS XE VMs' existing running configuration

Part 2: Update the device's configuration

Background / Scenario

In this lab, you will learn how to use the NETCONF ncclient to retrieve the device's configuration, update and create new interface configuration. You will also learn why the transactional support of NETCONF is important for getting consistent network changes.

Required Resources

- Access to a router with the IOS XE operating system version 16.6 or higher
- Python 3.x environment

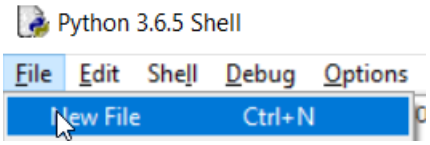
Part 1: Retrieve the IOS XE VMs' existing running configuration

In this part, you will use the ncclient module to retrieve the device's running configuration. The data are returned back in XML form that in the following steps is being transformed into more human readable format.

Step 1: Use ncclient to retrieve the device's running configuration.

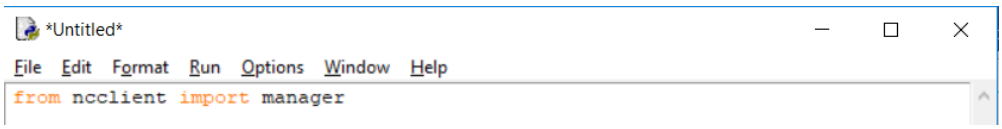
The ncclient module provides a "`manager`" class with "`connect()`" function to setup the remote NETCONF connection. After a successful connection, the returned object represents the NETCONF connection to the remote device.

a. In Python IDLE, create a new Python script file:



b. In the new Python script file editor, import the "`manager`" class from the ncclient module:

```
from ncclient import manager
```



c. Setup an `m` connection object using the `manager.connect()` function to the IOS XE device.

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

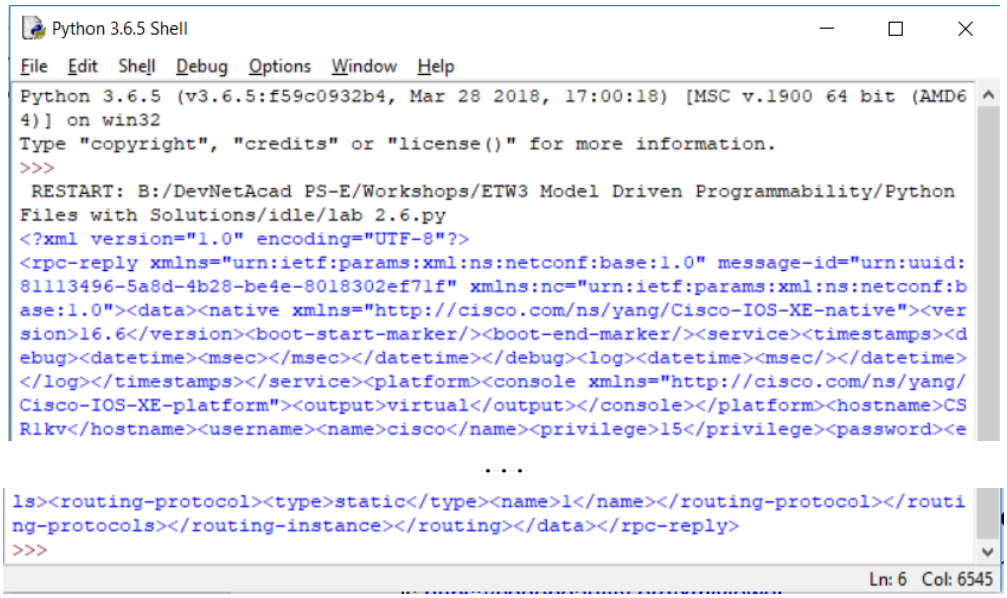
The parameters of the `manager.connect()` function are:

- `host` – the address (host or IP) of the remote device (adjust the IP address to match the router's current address)
- `port` – the remote port of the ssh service
- `username` – remote ssh username (in this lab "cisco" for that was setup in the IOS XE VM)
- `password` – remote ssh password (in this lab "cisco123!" for that was setup in the IOS XE VM)
- `hostkey_verify` – whether to verify the ssh fingerprint (in lab it is safe to set to False, in production environments you should always verify the ssh fingerprints)

d. After a successful NETCONF connection, using the "`get_config()`" function of the "m" NETCONF session object retrieve and print the device's running configuration. The `get_config()` function expects a "source" string parameter that defines the source NETCONF data-store.

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

e. Execute the Python script and explore the output:

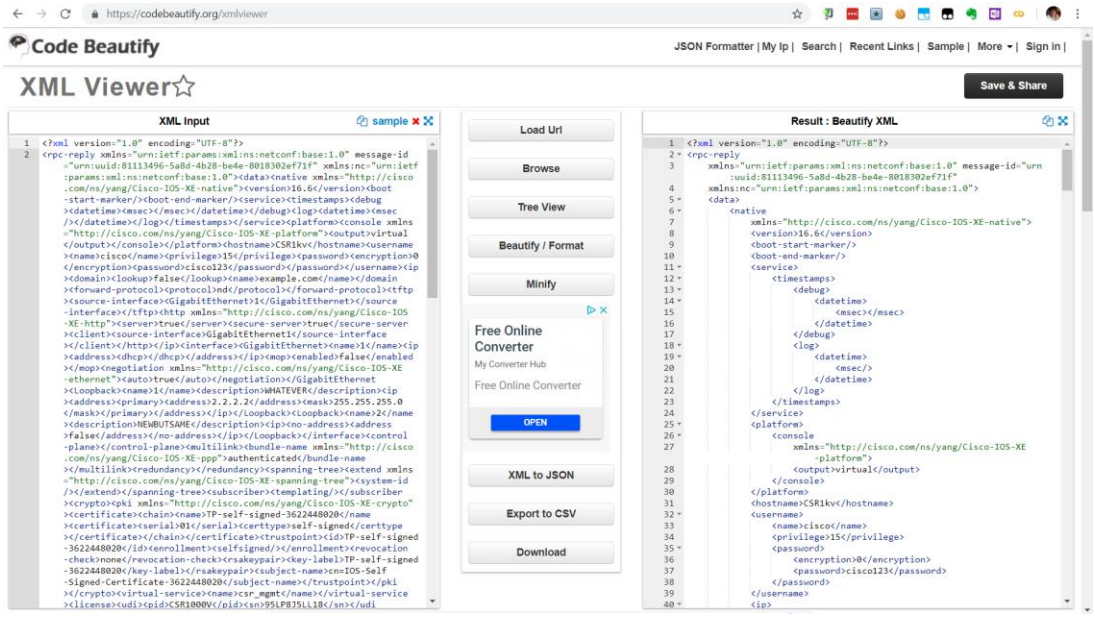


Step 2: Use CodeBeautfiy.com to evaluate the response

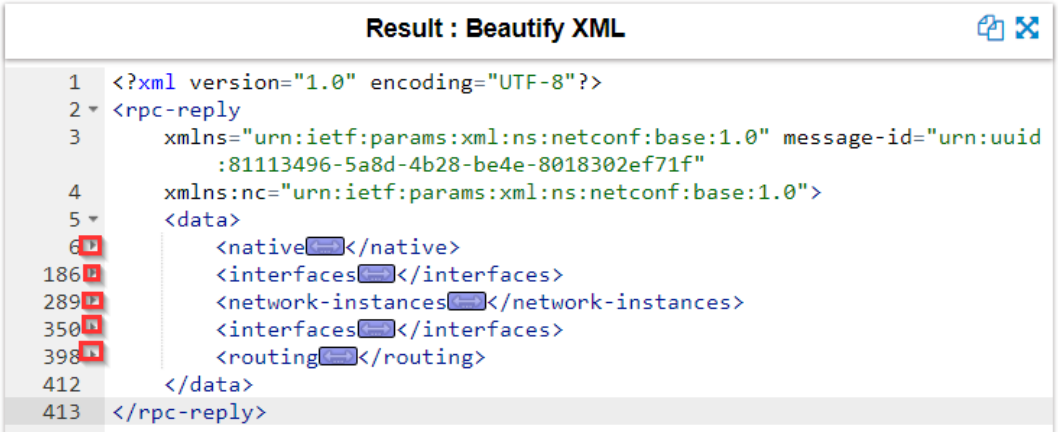
Code Beautify maintains a website for viewing code in a more human readable format. The XML viewer URL is https://codebeautify.org/xmlviewer

f. Copy the XML from IDLE to XML Viewer.

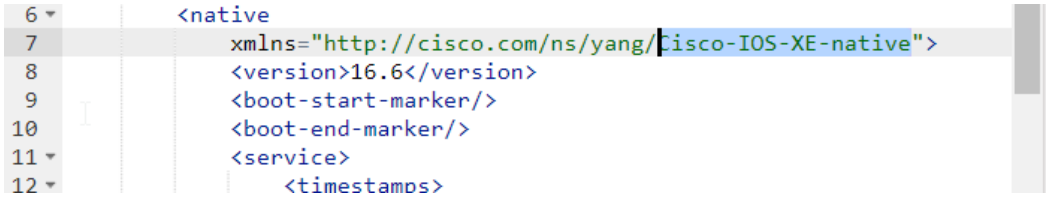
g. Click **Tree View** or **Beautify / Format** to render the raw XML output into a more human readable format.



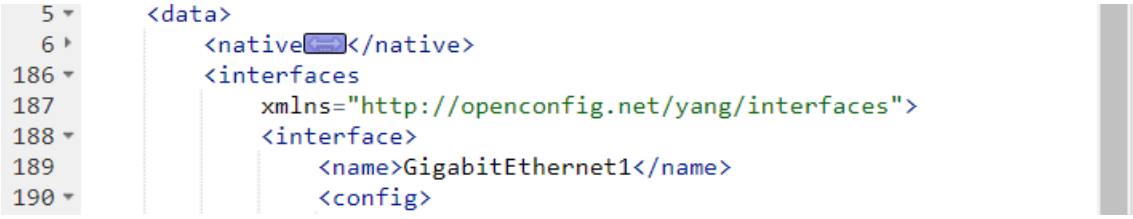
h. To simplify the view, close the XML elements that are under the rpc-reply/data structure:



i. Note that the rpc-reply/data/native element when opened, it contains an attribute xmlns that points to "Cisco-IOS-XE-native" YANG model. That means this part of the configuration is Cisco Native for IOS XE.



j. Also note that there are two "interfaces" elements – one with xmlns pointing to "http://openconfig.net/yang/interfaces" YANG model, while the other pointing to "ietf-interfaces" YANG model.



…

```
350        <interfaces
351            xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
352            <interface>
353                <name>GigabitEthernet1</name>
354                <type
355                    xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana
                        -if-type">ianaift:ethernetCsmacd
356                </type>
357                <enabled>true</enabled>
```

Both are used to describe the configuration of the interfaces, with a difference that the openconfig.net YANG model does support sub-interfaces, while the ietf-interfaces YANG model does not.

Step 3: Use toprettyxml() function to prettify the output.

a. Python has built in support to work with XML files. The "`xml.dom.minidom`" module can be used to prettify the output with the `toprettyxml()` function.

b. Import the "`xml.dom.minidom`" module:

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

c. 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() )
```

d. Execute the updated Python script and explore the output.

Step 4: Use filters to retrieve a configuration defined by a specific YANG model

a. NETCONF has support to return only data that are defined in a filter element.

b. Create the following `netconf_filter` variable that contains an XML NETCONF filter element to only retrieve 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>
"""
```

c. Include the `netconf_filter` variable in the get_config() call using the "filter" parameter:

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

e. Execute the updated Python script and explore the output:

```
 RESTART: B:/DevNetAcad PS-E/Workshops/ETW3 Model Driven Programmability/Python
Files with Solutions/idle/lab 2.6.py
<?xml version="1.0" ?>
<rpc-reply message-id="urn:uuid:7a860e08-0447-4482-9ce6-7ed0efe2f24a" xmlns="urn
:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:b
ase:1.0">
        <data>
                <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
                        <version>16.6</version>
                        <boot-start-marker/>
                        <boot-end-marker/>
                        <service>
                                <timestamps>
                                        <debug>
                                                <datetime>
                                                        <msec/>
                                                </datetime>
                                        </debug>
                                        <log>
                                                <datetime>
                                                        <msec/>
                                                </datetime>
                                        </log>
                                </timestamps>
                        </service>
```

…

```
                                <vty>
                                        <first>5</first>
                                        <last>15</last>
                                        <login>
                                                <local/>
                                        </login>
                                </vty>
                        </line>
                        <diagnostic xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE
-diagnostics">
                                <bootup>
                                        <level>minimal</level>
                                </bootup>
                        </diagnostic>
                </native>
        </data>
</rpc-reply>

>>>
```

Part 2: Update the device's configuration

Step 1: Create a new Python script file

a. In IDLE create a new Python script file

b. Import the required modules and setup the NETCONF session:

```
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
        )
```

Step 2: Change the hostname

f. In order to update an existing setting in the configuration, you can extract the setting location from the configuration retrieved in Step 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn
        :uuid:ffad8001-6b82-4094-acc4-6f456ba9e088"
    xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
    <data>
        <native
            xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
            <version>16.6</version>
            <boot-start-marker/>
            <boot-end-marker/>
            <service>
                <timestamps>
                    <debug>
                        <datetime>
                            <msec></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>
```

g. The configuration update is always enclosed in a "`config`" XML element that includes a tree of XML elements that require update.

h. Create a `netconf_data` variable that holds a configuration update for the hostname element as defined in the Cisco IOS XE Native YANG Model:

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

```
"""
```

i. Edit the existing device configuration with the "`edit_config()`" function of the "`m`" NETCONF session object. The `edit_config()` function expects two parameters:

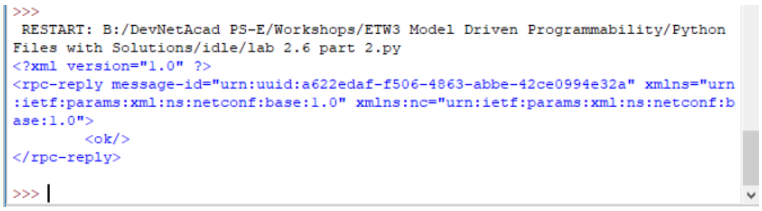
- `target` – the target netconf data-store to be updated
- `config` – the configuration update

The `edit_config()` function returns an XML object containing information about the change success. After editing the configuration, print the returned value:

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

j. Before executing the new Python script, check the current hostname by connecting to the console of the IOS XE VM.

k. Execute the Python script and explore the output:



l. After executing the Python script, if the reply contained the `<ok/>` element, verify whether current hostname has been changed by connecting to the console of the IOS XE VM.

Step 3: Create a loopback interface

m. Update the `netconf_data` variable to hold a configuration update that creates a new loopback **100** interface with the IP address **100.100.100.100**/24:

```
netconf_data = """
<config>
 <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
   <Loopback>
    <name>100</name>
    <description>TEST1</description>
    <ip>
     <address>
      <primary>
       <address>100.100.100.100</address>
       <mask>255.255.255.0</mask>
      </primary>
     </address>
    </ip>
   </Loopback>
  </interface>
 </native>
</config>
"""
```

n. Add the new loopback 100 interface by editing the existing device configuration using the "`edit_config()`" function:

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

o. Before executing the updated Python script, check using "show ip int brief" and "show int desc" the existing loopback interface by connecting to the console of the IOS XE VM.

```
NEWHOSTNAME#sh ip int brie
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       10.0.2.15       YES DHCP   up                    up
Loopback1              2.2.2.2         YES manual up                    up
Loopback2              unassigned      YES unset  up                    up
NEWHOSTNAME#
NEWHOSTNAME#sh int desc
Interface                      Status         Protocol Description
Gi1                            up             up
Lo1                            up             up       WHATEVER
Lo2                            up             up       NEWBUTSAME
NEWHOSTNAME#
```

p. Execute the Python script and explore the output:

```
>>>
 RESTART: B:/DevNetAcad PS-E/Workshops/ETW3 Model Driven Programmability/Python
Files with Solutions/idle/lab 2.6 part 2.py
<?xml version="1.0" ?>
<rpc-reply message-id="urn:uuid:a622edaf-f506-4863-abbe-42ce0994e32a" xmlns="urn
:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:b
ase:1.0">
        <ok/>
</rpc-reply>

>>> |
```

q. After executing the Python script, if the reply contained the `<ok/>` element, verify whether current loopback interfaces have changed by connecting to the console of the IOS XE VM.

```
NEWHOSTNAME#sh ip int brie
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       10.0.2.15       YES DHCP   up                    up
Loopback1              2.2.2.2         YES manual up                    up
Loopback2              unassigned      YES unset  up                    up
Loopback100            100.100.100.100 YES other  up                    up
NEWHOSTNAME#sh int desc
Interface                      Status         Protocol Description
Gi1                            up             up
Lo1                            up             up       WHATEVER
Lo2                            up             up       NEWBUTSAME
Lo100                          up             up       TEST1
NEWHOSTNAME#
```

Step 4: Attempt to create a new loopback interface with a conflicting IP address

a. Update the `netconf_data` variable to hold a configuration update that creates a new loopback **111** interface with the same IP address as on loopback 100: 100.100.100.100/32:

```
netconf_data = """
<config>
 <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
   <Loopback>
    <name>111</name>
    <description>TEST1</description>
    <ip>
     <address>
      <primary>
       <address>100.100.100.100</address>
```

```
        <mask>255.255.255.0</mask>
       </primary>
      </address>
     </ip>
    </Loopback>
   </interface>
  </native>
</config>
"""
```

b. Attempt to add the new loopback 111 interface by editing the existing device configuration using the "`edit_config()`" function:

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

c. Before executing the updated Python script, check using "show ip int brief" and "show int desc" the existing loopback interface by connecting to the console of the IOS XE VM.

```
NEWHOSTNAME#sh ip int brie
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       10.0.2.15       YES DHCP   up                    up
Loopback1              2.2.2.2         YES manual up                    up
Loopback2              unassigned      YES unset  up                    up
NEWHOSTNAME#
NEWHOSTNAME#sh int desc
Interface                      Status         Protocol Description
Gi1                            up             up
Lo1                            up             up       WHATEVER
Lo2                            up             up       NEWBUTSAME
NEWHOSTNAME#
```

d. Execute the Python script and explore the output:

```
>>>
 RESTART: B:/DevNetAcad PS-E/Workshops/ETW3 Model Driven Programmability/Python
Files with Solutions/idle/lab 2.6 part 2.py
Traceback (most recent call last):
  File "B:/DevNetAcad PS-E/Workshops/ETW3 Model Driven Programmability/Python Fi
les with Solutions/idle/lab 2.6 part 2.py", line 42, in <module>
    netconf_reply = m.edit_config(target="running", config=netconf_data)
  File "C:\Users\jjanitor\AppData\Local\Programs\Python\Python36\lib\site-packag
es\ncclient\manager.py", line 216, in execute
    raise_mode=self._raise_mode).request(*args, **kwds)
  File "C:\Users\jjanitor\AppData\Local\Programs\Python\Python36\lib\site-packag
es\ncclient\operations\edit.py", line 67, in request
    return self._request(node)
ncclient.operations.rpc.RPCError: inconsistent value: Device refused one or more
 commands
>>>
```

The device has refused one or more configuration settings. With NETCONF, thanks to the transactional behavior, no partial configuration change has been applied but the whole transaction was canceled.

e. After executing the Python script, verify that no configuration changes, not even partial have been applied:

```
NEWHOSTNAME#sh ip int brie
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       10.0.2.15       YES DHCP   up                    up
Loopback1              2.2.2.2         YES manual up                    up
Loopback2              unassigned      YES unset  up                    up
Loopback100            100.100.100.100 YES other  up                    up
NEWHOSTNAME#sh int desc
Interface                      Status         Protocol Description
Gi1                            up             up
Lo1                            up             up       WHATEVER
Lo2                            up             up       NEWBUTSAME
Lo100                          up             up       TEST1
NEWHOSTNAME#_
```