Lab – NETCONF w/Python: Device Configuration

Objectives

Part 1: Retrieve the IOS XE VM's 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, and update and create a 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

Instructions

Part 1: Retrieve the IOS XE VM's 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. In the following steps, this data will be transformed into a 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. Using the `manager.connect()` function, set up an **`m`** connection object 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:

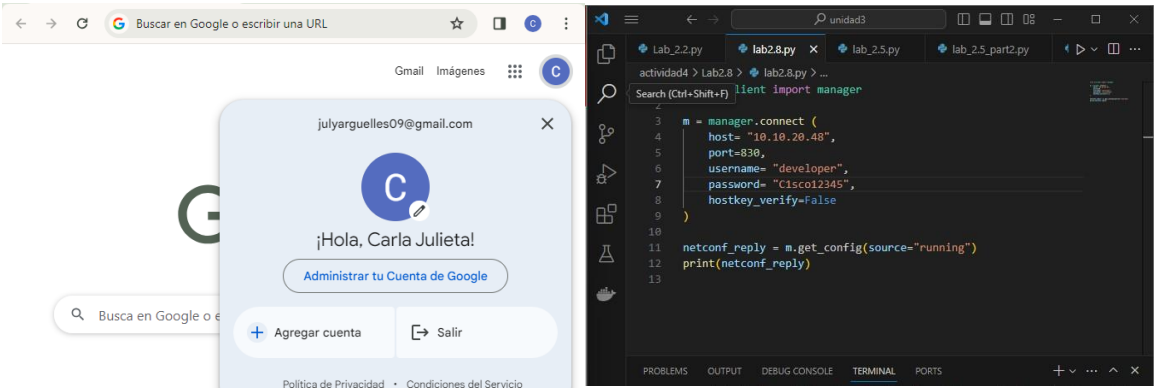
- **`host`** – This is the address (host or IP) of the remote device (Adjust the IP address to match the router's current address.).

- **port** – This is the remote port of the SSH service.
- **username** – This is the remote SSH username (In this lab, use "cisco" because that was set up in the IOS XE VM.).
- **password** – This is the remote SSH password (In this lab, use "cisco123!" because that was set up in the IOS XE VM.).
- **hostkey_verify** – Use this to verify the SSH fingerprint (In this lab, it is safe to set to False; however, in production environments you should always verify the SSH fingerprints.).

d. After a successful NETCONF connection, use the "`get_config()`" function of the "**m**" NETCONF session object to 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 opened rpc-reply/data/native element contains an attribute xmlns that points to "Cisco-IOSXE-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. The one with xmlns is pointing to the "http://openconfig.net/yang/interfaces" YANG model, while the other is pointing to the "ietf-interfaces" YANG model.

Both are used to describe the configuration of the interfaces. The difference is 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.

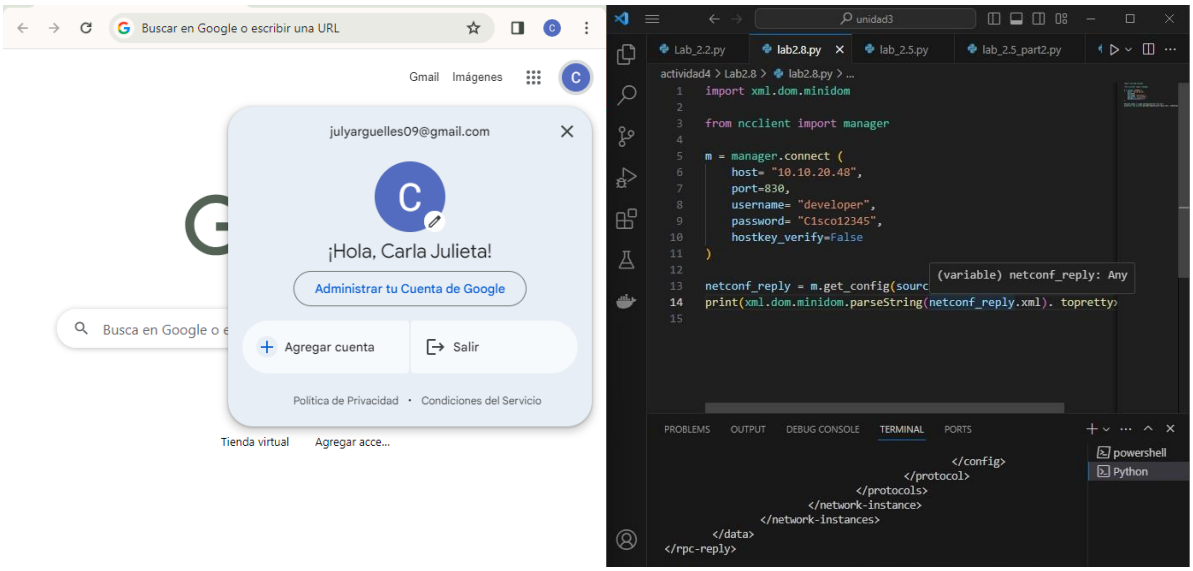
b. Import the "`xml.dom.minidom`" module:

```
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 containing an XML NETCONF filter element that is designed to retrieve only data that is defined by the Cisco IOS XE Native YANG model:

```
netconf_filter = """
<filter>
    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XI-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())
```

d. Execute the updated Python script and explore the output

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 set up 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.

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

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`** – This is the target netconf data-store to be updated.
- **`config`** – This is the configuration update.

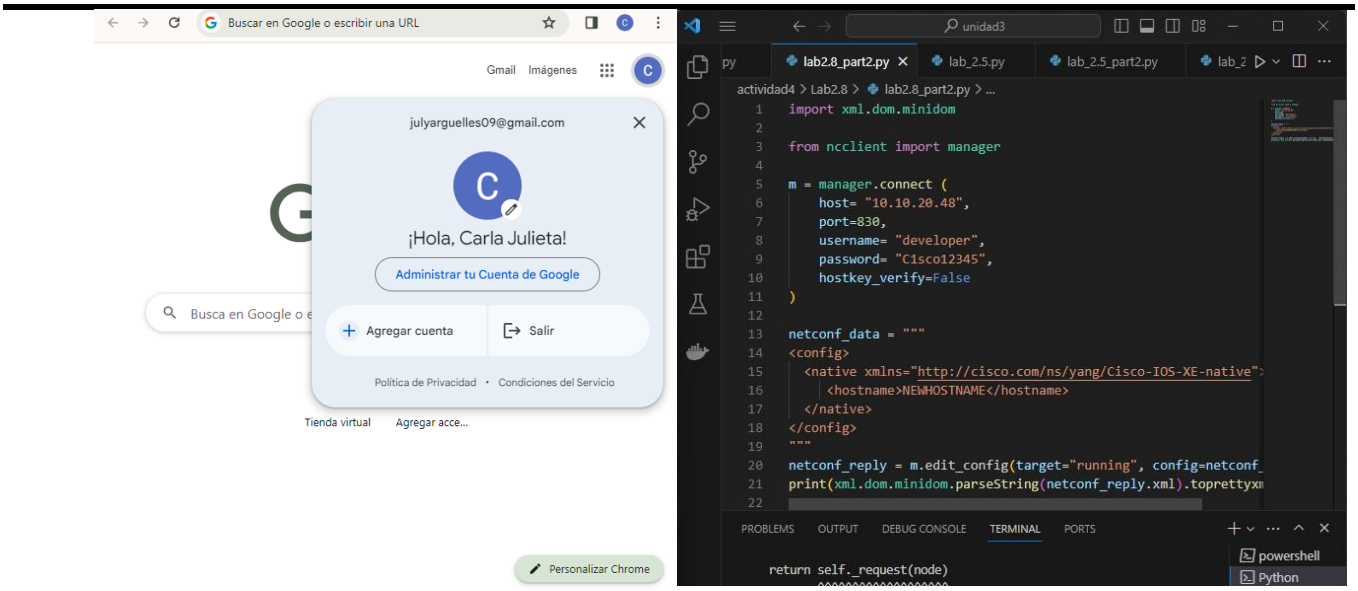
The `edit_config()` function returns an XML object containing information about the success of the change. After editing the configuration, print the returned value:

```
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 the 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>
"""
```

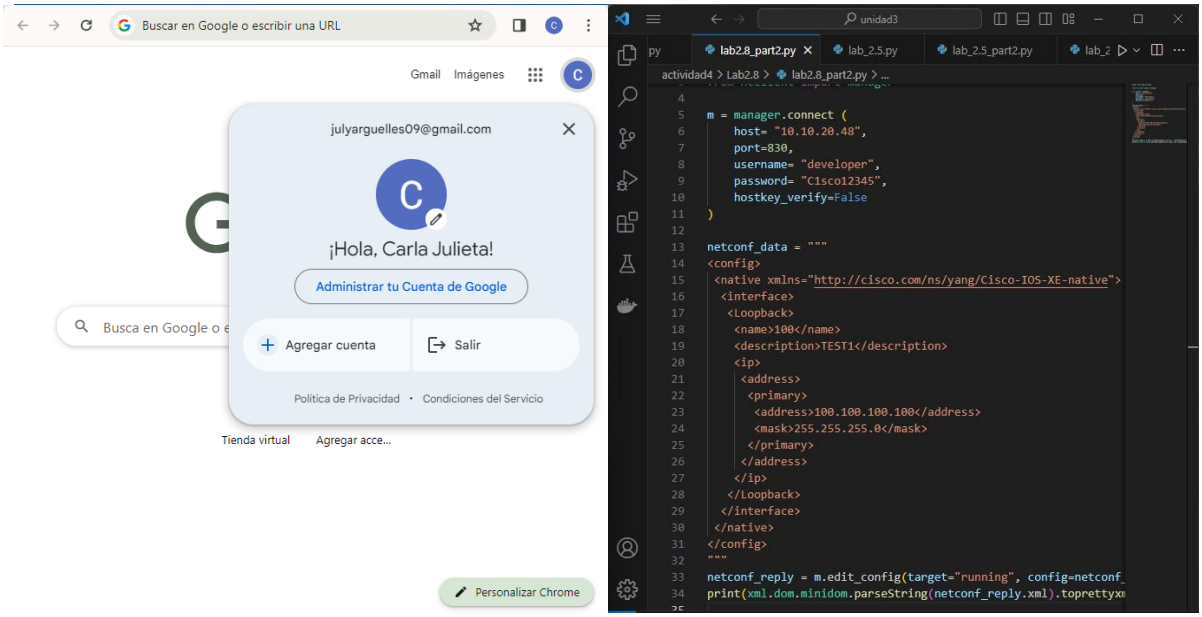
n. Add the new loopback 100 interface by editing the existing device configuration using the "`edit_config()`" function:

```
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 the existing loopback interface by connecting to the console of the IOS XE VM using the **show ip int brief** and **show int desc** commands.

p. Execute the Python script and explore the output

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



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 the existing loopback interface by connecting to the console of the IOS XE VM using the **show ip int brief** and **show int desc** commands.

d. Execute the Python script and explore the output.

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.