Canchola Ramírez Mariana

Lab – NETCONF w/Python: Device Configuration

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
        )
```

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.

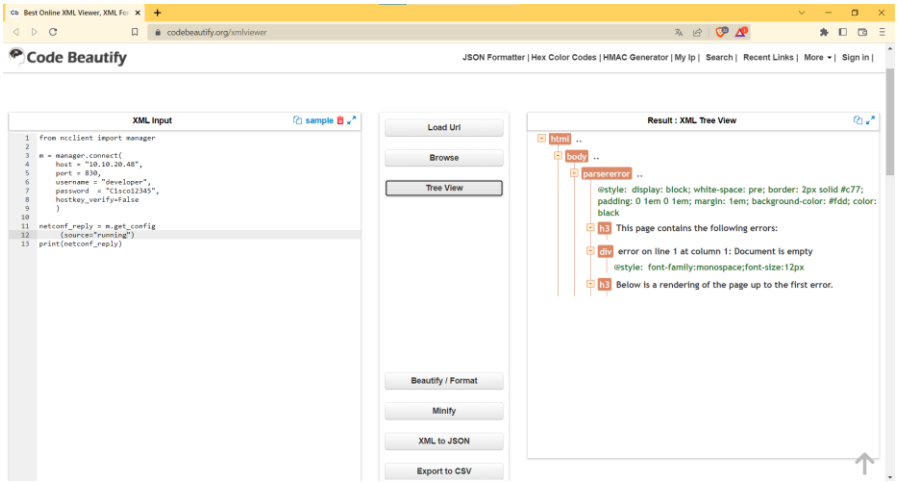
Step 2: Use CodeBeautfiy.com to evaluate the response.

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

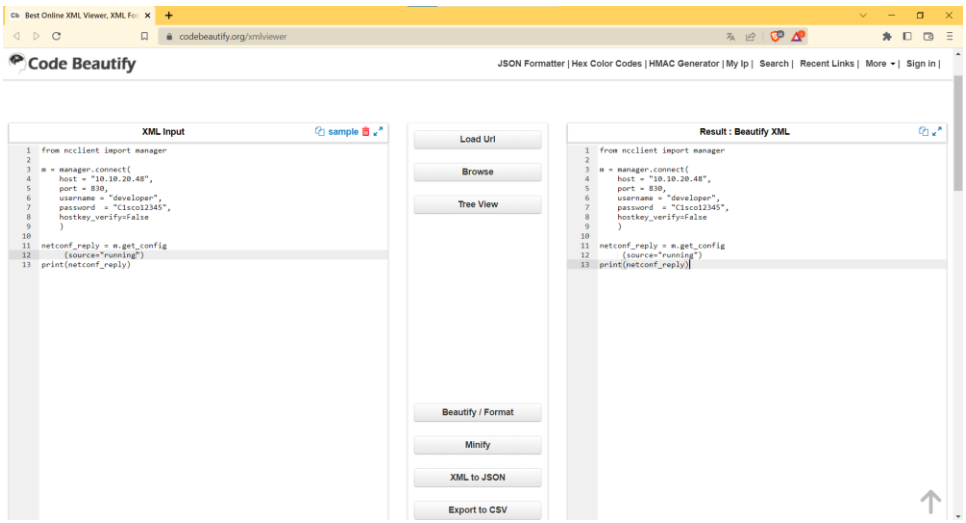
f. Copy the XML from IDLE to XML Viewer.

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

Tree View



Beautify/Format



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

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

SCRIPT


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

netconf_filter = """
<filter>
    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native" />
</filter>
"""

m = manager.connect(
    host = "10.10.20.48",
    port = 830,
    username = "developer",
    password  = "C1sco12345",
    hostkey_verify=False
    )

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

SALIDA

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-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())
```

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
        )
```

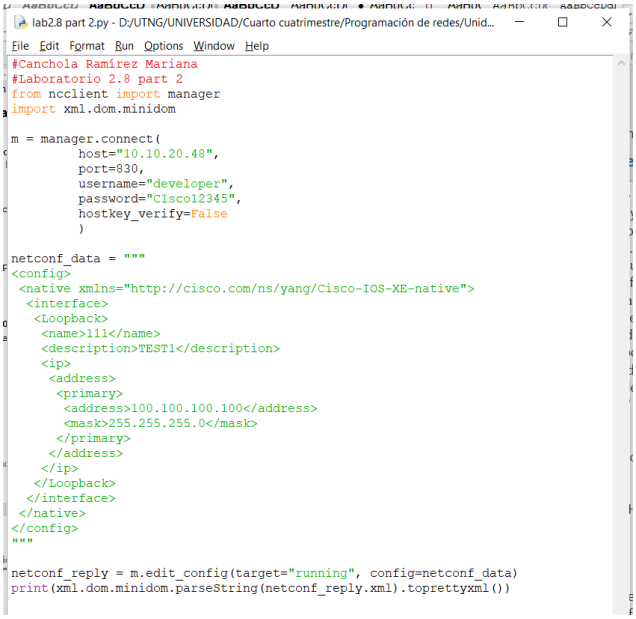
Step 2: Change the hostname.

e. 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.

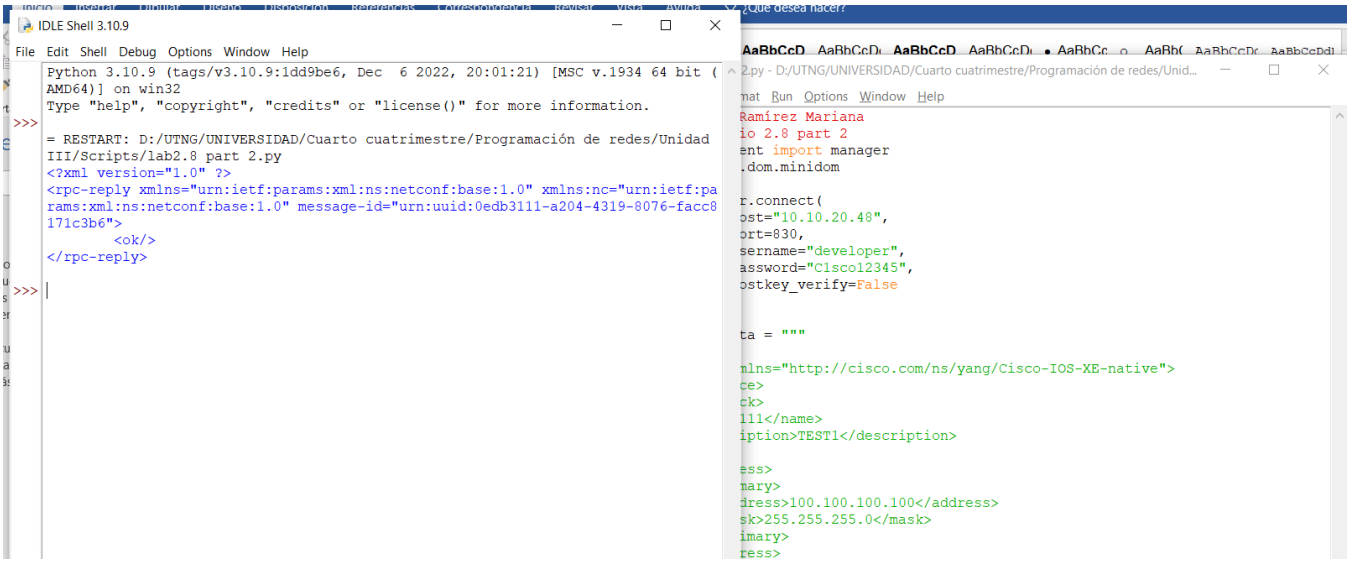
f. Execute the Python script and explore the output

g. 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 3: Attempt to create a new loopback interface with a conflicting IP address.

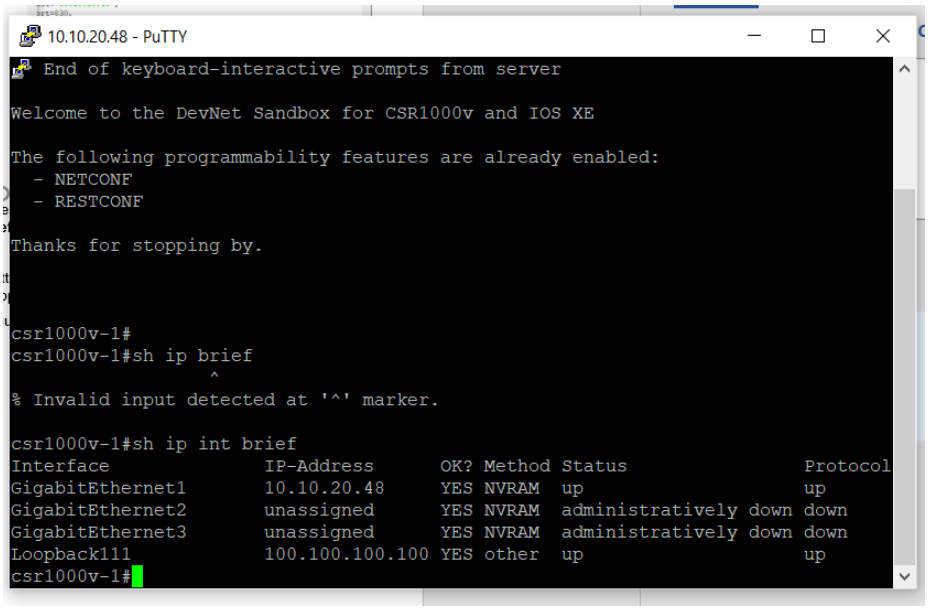


SALIDA

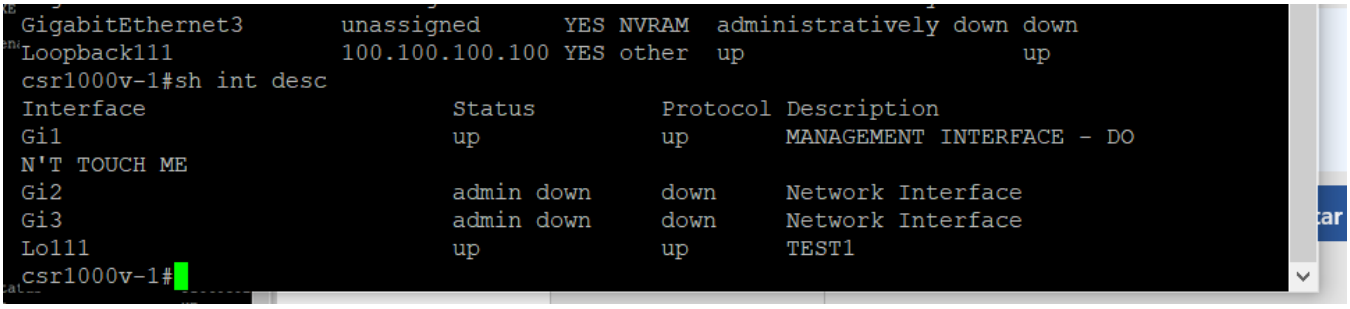


h. Before executing the updated Python script, check the existing loopback interface by connecting to the console of the IOS XE VM using the **show ip int brief** and **show int desc** commands.

Ejecutando comando **show ip int brief**



Ejectando comando **show int desc**



Conclusiones

En este ejercicio complementé lo del ejercicio 2.7 ya que seguimos utlizando y trabajando con la librería NCCLIENT, la cual ya en esta practica pude ver y entender major su funcionamiento. Y algunos de sus comandos pude ver su funcionalidad y la manera en que tornan las cosas diferentes en cada salida cada que configuramos o hacemos cambios con estos comandos en nuestro código.

Como ya sabemos NCCLIENT proporciona API´s intuitivas que puedan ser mapeadas de manera inteligente, así mismo proporciona y facilita la administración de aplicaciones con secuencias de comandos de la red.

ncclient fue desarrollado por Shikar Bhushan <http://schmizz.net>.

Algunas de sus características son:

- Admite todas las operaciones y capacidades definidas en RFC 6241.
- Solicitud de canalización.
- Solicitudes RPC asíncronas.
- Mantener XML fuera del camino a menos que sea realmente necesario.
- Extensible. Se pueden agregar fácilmente nuevas asignaciones de transporte y capacidades/operaciones.

La mejor manera de presentar es a través de un ejemplo de código simple:

```
from ncclient import manager

# use unencrypted keys from ssh-agent or ~/.ssh keys, and rely on known_hosts
with manager.connect_ssh("host", username="user") as m:
    assert(":url" in m.server_capabilities)
    with m.locked("running"):
        m.copy_config(source="running", target="file:///new_checkpoint.conf")
        m.copy_config(source="file:///old_checkpoint.conf", target="running")
```