2022

2.8 Lab – NETCONF w/Python: Device Configuration

Programación de Redes

UNIVERSIDAD TECNLOGICA DEL NORTE DE GUANAJUATO

NOMBRE

JUAN PABLO PALMA APODERADO

CARRERA

TSU EN INFRAESTRUCTURA DE REDES DIGITALES

DOCENTE

BARRON RODRIGUEZ GABRIEL

NO. CONTROL

1221100259

FECHA

14 DE OCTUBRE DEL 2022

GRUPO

GIR0441

2.8 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.

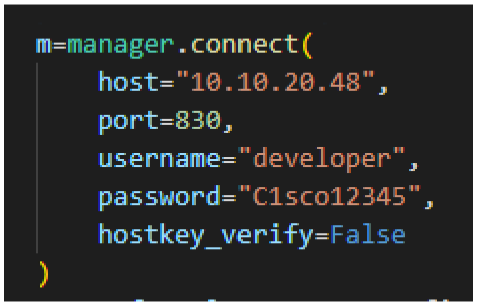
The ncclient module provides a "`manager`" class with "`connect()`" function to setup the remote NETCONF connection. After a successful connection, the returned object represents the NETCONF connection to the remote device.

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.



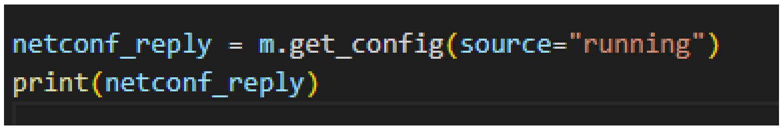
```
m=manager.connect(
    host="10.10.20.48",
    port=830,
    username="developer",
    password="C1sco12345",
    hostkey_verify=False
)
```

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.).

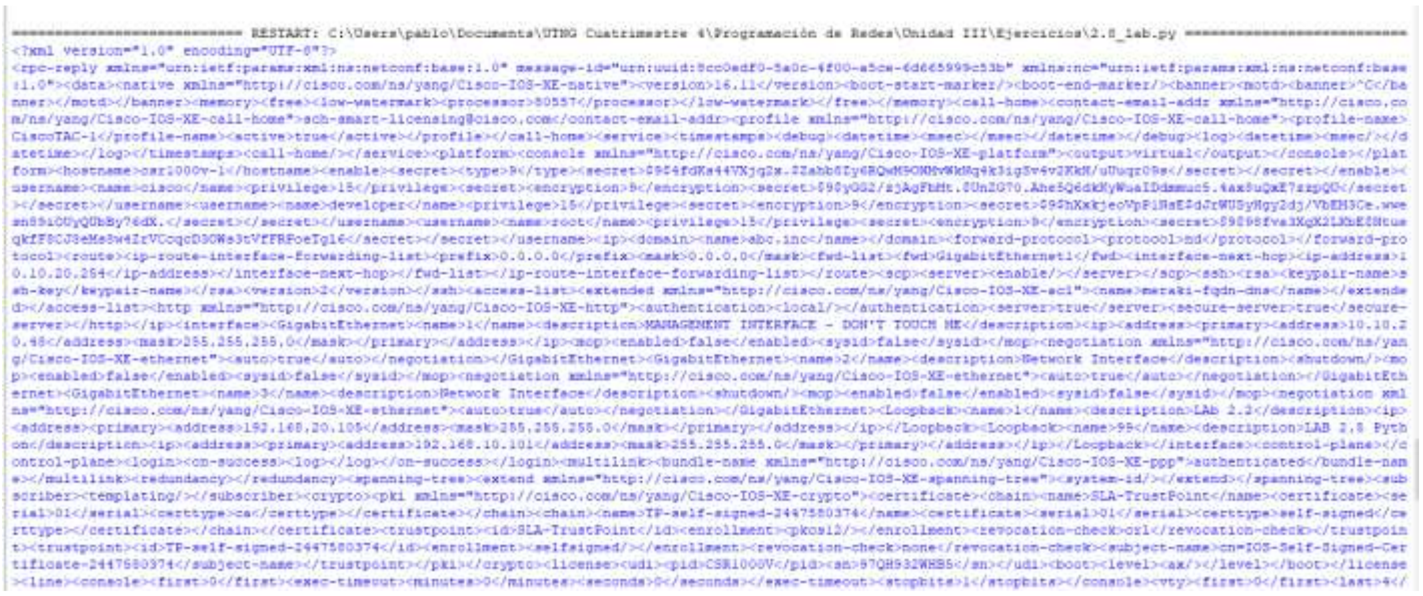
- **`port`** – This is the remote port of the SSH service.
- **`username`** – This is the remote SSH username (In this lab, use “cisco” because that was set up in the IOS XE VM.).
- **`password`** – This is the remote SSH password (In this lab, use “cisco123!” because that was set up in the IOS XE VM.).
- **`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.



```
============================ RESTART: C:\Users\pablo\Documents\UTNG Cuatrimestre 4\Programación de Redes\Unidad III\Ejercicios\2.8_lab.py ============================
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:8cc0edf0-5a0c-4f00-a5ce-6d665993c53b" xmlns:nc="urn:ietf:params:xml:ns:netconf:base
:1.0"><data><native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native"><version>16.11</version><boot-start-marker/><boot-end-marker/><banner><motd><banner>^C</ba
nner></motd></banner><memory><free><low-watermark><processor>80557</processor></low-watermark></free></memory><call-home><contact-email-addr xmlns="http://cisco.co
m/ns/yang/Cisco-IOS-XE-call-home">sch-smart-licensing@cisco.com</contact-email-addr><profile xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-call-home"><profile-name>
CiscoTAC-1</profile-name><active>true</active></profile></call-home><service><timestamps><debug><datetime><msec></msec></datetime></debug><log><datetime><msec/></d
atetime></log></timestamps><call-home/></service><platform><console xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-platform"><output>virtual</output></console></plat
form><hostname>csr1000v-1</hostname><enable><secret><type>9</type><secret>$9$4fdKs44VXjq2x.ZZabb6Iy6RQwM5OMMvWkMq4k3ig5v4v2KkM/uUuqr09s</secret></secret></enable><
username><name>cisco</name><privilege>15</privilege><secret><encryption>9</encryption><secret>$9$yOG2/sjAgFbMt.8Un2GT0.Ahe5Q6dkKyWuaIDdmmuc5.4ax8uQxE7zzpQ0</secret
></secret></username><username><name>developer</name><privilege>15</privilege><secret><encryption>9</encryption><secret>$9$hXxkjeoVpP1NsE#dJrWUSyHgy2dj/VbEH3Ce.wwe
sn89iOUyQUbBy76dX.</secret></secret></username><username><name>root</name><privilege>15</privilege><secret><encryption>9</encryption><secret>$9$95fva3XgX2LXbE8Ntue
qkfF8CJ3eMe8w4IrVCoqcD3OWe3tVfFRFoeTgl6</secret></secret></username><ip><domain><name>abc.inc</name></domain><forward-protocol><protocol>nd</protocol></forward-pro
tocol><route><ip-route-interface-forwarding-list><prefix>0.0.0.0</prefix><mask>0.0.0.0</mask><fwd-list><fwd>GigabitEthernet1</fwd><interface-next-hop><ip-address>1
0.10.20.254</ip-address></interface-next-hop></fwd-list></ip-route-interface-forwarding-list></route><scp><server><enable/></server></scp><ssh><rsa><keypair-name>s
sh-key</keypair-name></rsa><version>2</version></ssh><access-list><extended xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-acl"><name>meraki-fqdn-dns</name></extende
d></access-list><http xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-http"><authentication><local/></authentication><server>true</server><secure-server>true</secure-
server></http></ip><interface><GigabitEthernet><name>1</name><description>MANAGEMENT INTERFACE - DON'T TOUCH ME</description><ip><address><primary><address>10.10.2
0.48</address><mask>255.255.255.0</mask></primary></address></ip><mop><enabled>false</enabled><sysid>false</sysid></mop><negotiation xmlns="http://cisco.com/ns/yan
g/Cisco-IOS-XE-ethernet"><auto>true</auto></negotiation></GigabitEthernet><GigabitEthernet><name>2</name><description>Network Interface</description><shutdown/><mo
p><enabled>false</enabled><sysid>false</sysid></mop><negotiation xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-ethernet"><auto>true</auto></negotiation></GigabitEth
ernet><GigabitEthernet><name>3</name><description>Network Interface</description><shutdown/><mop><enabled>false</enabled><sysid>false</sysid></mop><negotiation xml
ns="http://cisco.com/ns/yang/Cisco-IOS-XE-ethernet"><auto>true</auto></negotiation></GigabitEthernet><Loopback><name>1</name><description>LAb 2.2</description><ip>
<address><primary><address>192.168.20.105</address><mask>255.255.255.0</mask></primary></address></ip></Loopback><Loopback><name>99</name><description>LAB 2.8 Pyth
on</description><ip><address><primary><address>192.168.10.101</address><mask>255.255.255.0</mask></primary></address></ip></Loopback></interface><control-plane></c
ontrol-plane><login><on-success><log></log></on-success></login><multilink><bundle-name xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-ppp">authenticated</bundle-nam
e></multilink><redundancy></redundancy><spanning-tree><extend xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-spanning-tree"><system-id/></extend></spanning-tree><sub
scriber><templating/></subscriber><crypto><pki xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-crypto"><certificate><chain><name>SLA-TrustPoint</name><certificate><se
rial>01</serial><certtype>ca</certtype></certificate></chain><chain><name>TP-self-signed-2447580374</name><certificate><serial>01</serial><certtype>self-signed</ce
rttype></certificate></chain></certificate><trustpoint><id>SLA-TrustPoint</id><enrollment><pkcs12/></enrollment><revocation-check>crl</revocation-check></trustpoin
t><trustpoint><id>TP-self-signed-2447580374</id><enrollment><selfsigned/></enrollment><revocation-check>none</revocation-check><subject-name>cn=IOS-Self-Signed-Cer
tificate-2447580374</subject-name></trustpoint></pki></crypto><license><udi><pid>CSR1000V</pid><sn>97QH932WHB5</sn></udi><boot><level><ax/></level></boot></license
><line><console><first>0</first><exec-timeout><minutes>0</minutes><seconds>0</seconds></exec-timeout><stopbits>1</stopbits></console><vty><first>0</first><last>4</
```

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-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. 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-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()
)
```

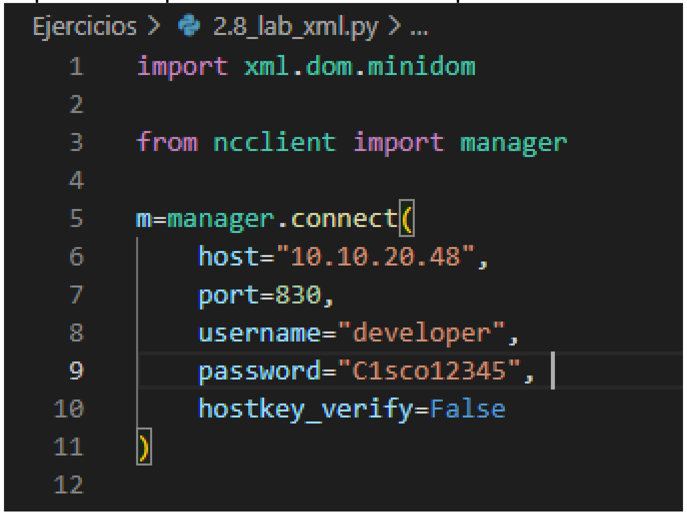
e. Execute the updated Python script and explore the output

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:



Ejercicios > 2.8_lab_xml.py > ...

```
import xml.dom.minidom

from ncclient import manager

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

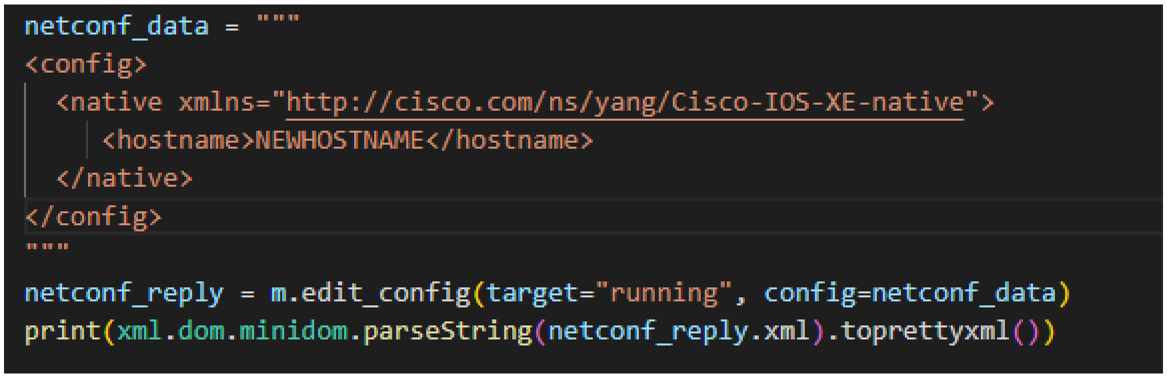
```
netconf_data = """
<config>
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
     <hostname>NEWHOSTNAME</hostname>
  </native>
</config>
"""
```

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())
```



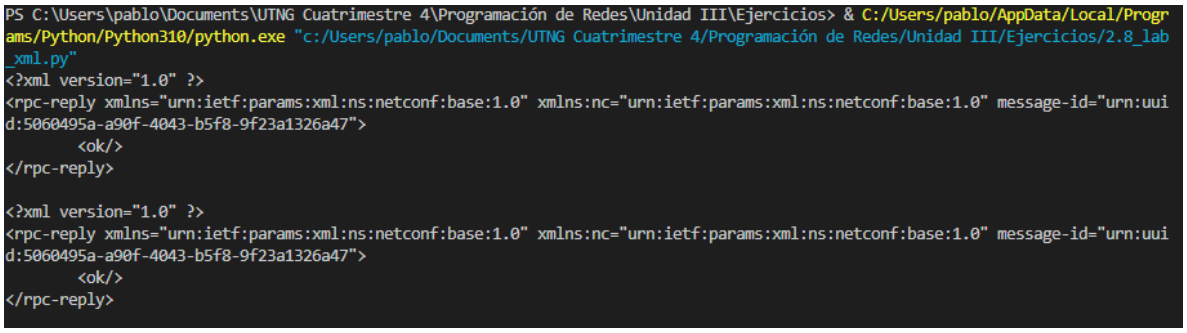
```
netconf_data = """
<config>
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
    <hostname>NEWHOSTNAME</hostname>
  </native>
</config>
"""

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.

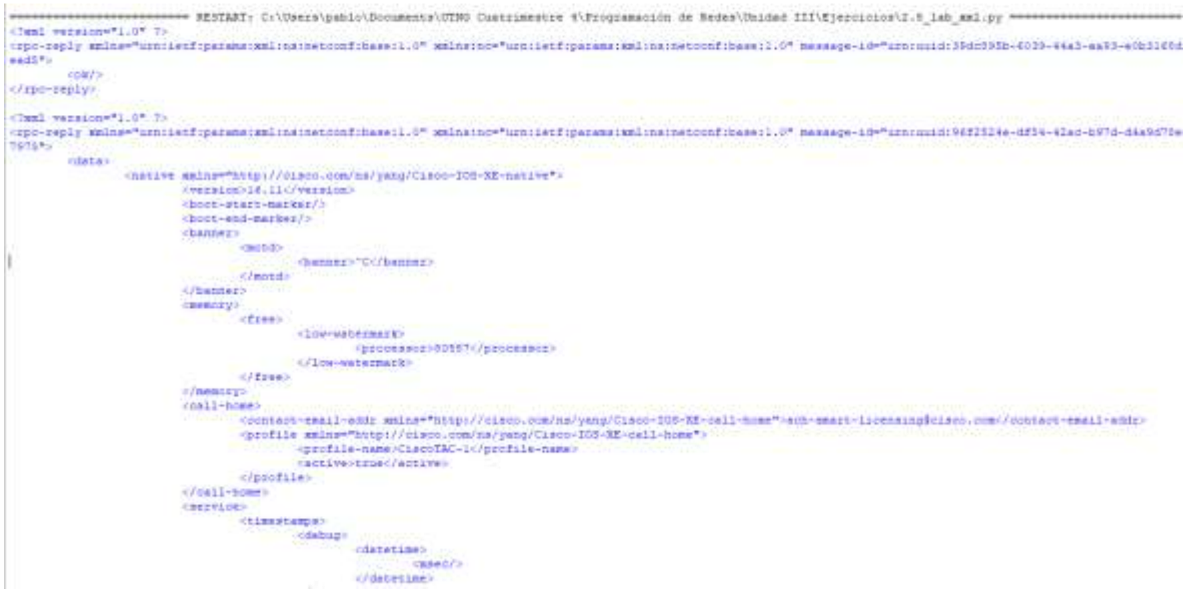
k. Execute the Python script and explore the output.

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.



```
PS C:\Users\pablo\Documents\UTNG Cuatrimestre 4\Programación de Redes\Unidad III\Ejercicios> & C:/Users/pablo/AppData/Local/Progr
ams/Python/Python310/python.exe "c:/Users/pablo/Documents/UTNG Cuatrimestre 4/Programación de Redes/Unidad III/Ejercicios/2.8_lab
_xml.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:uui
d:5060495a-a90f-4043-b5f8-9f23a1326a47">
        <ok/>
</rpc-reply>

<?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:uui
d:5060495a-a90f-4043-b5f8-9f23a1326a47">
        <ok/>
</rpc-reply>
```



```
========================= RESTART: C:\Users\pablo\Documents\UTNG Cuatrimestre 4\Programación de Redes\Unidad III\Ejercicios\2.8_lab_xml.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:35dc595b-6039-44a3-aa93-e0b3160d
eed5">
        <ok/>
</rpc-reply>

<?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:96f2524e-df34-42ad-b97d-d4a9d70e
7978">
        <data>
                <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
                        <version>16.11</version>
                        <boot-start-marker/>
                        <boot-end-marker/>
                        <banner>
                                <motd>
                                        <banner>^C</banner>
                                </motd>
                        </banner>
                        <memory>
                                <free>
                                        <low-watermark>
                                                <processor>80557</processor>
                                        </low-watermark>
                                </free>
                        </memory>
                        <call-home>
                                <contact-email-addr xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-call-home">sch-smart-licensing@cisco.com</contact-email-addr>
                                <profile xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-call-home">
                                        <profile-name>CiscoTAC-1</profile-name>
                                        <active>true</active>
                                </profile>
                        </call-home>
                        <service>
                                <timestamps>
                                        <debug>
                                                <datetime>
                                                        <msec/>
                                                </datetime>
```

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

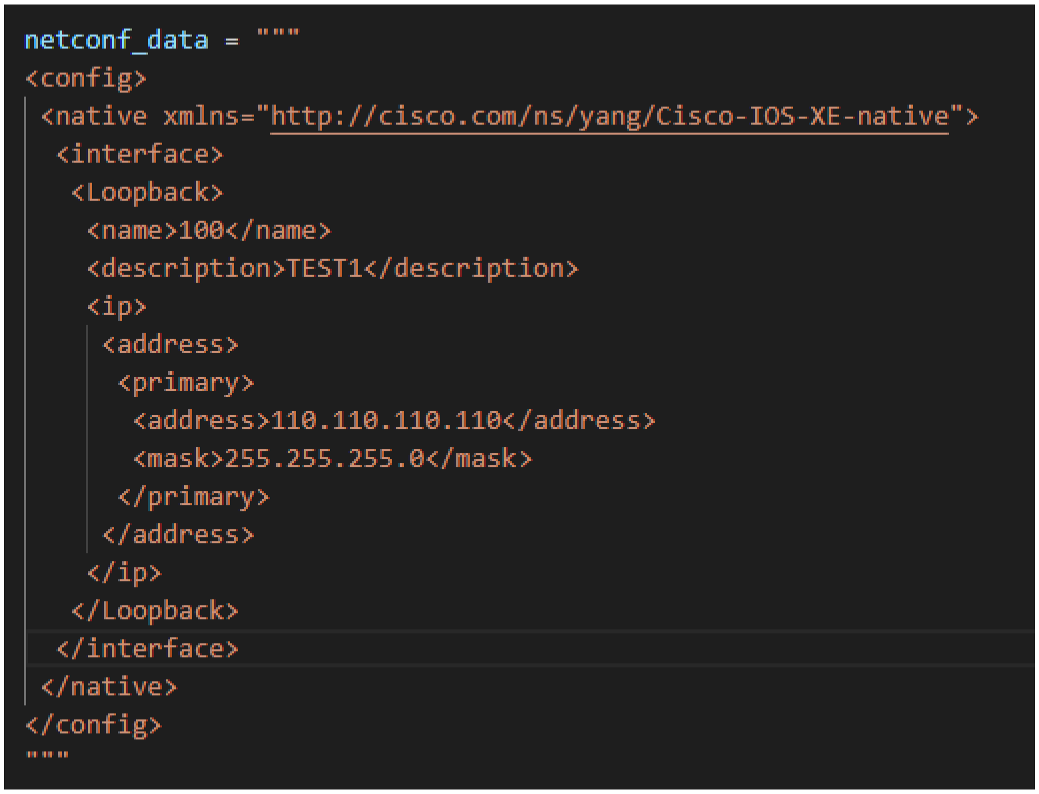
q. After executing the Python script, if the reply contained the `<ok/>` element, verify whether the current loopback interfaces have changed by connecting to the console of the IOS XE VM.

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>
```

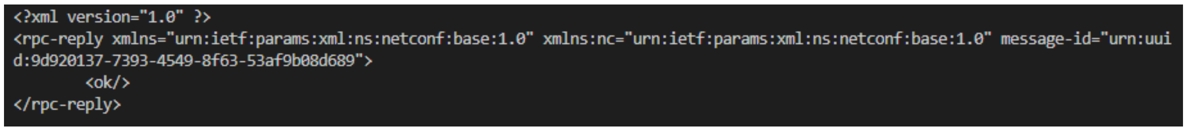
```
      <primary>
       <address>100.100.100.100</address>
       <mask>255.255.255.0</mask>
      </primary>
     </address>
    </ip>
   </Loopback>
  </interface>
 </native>
</config>
"""
```



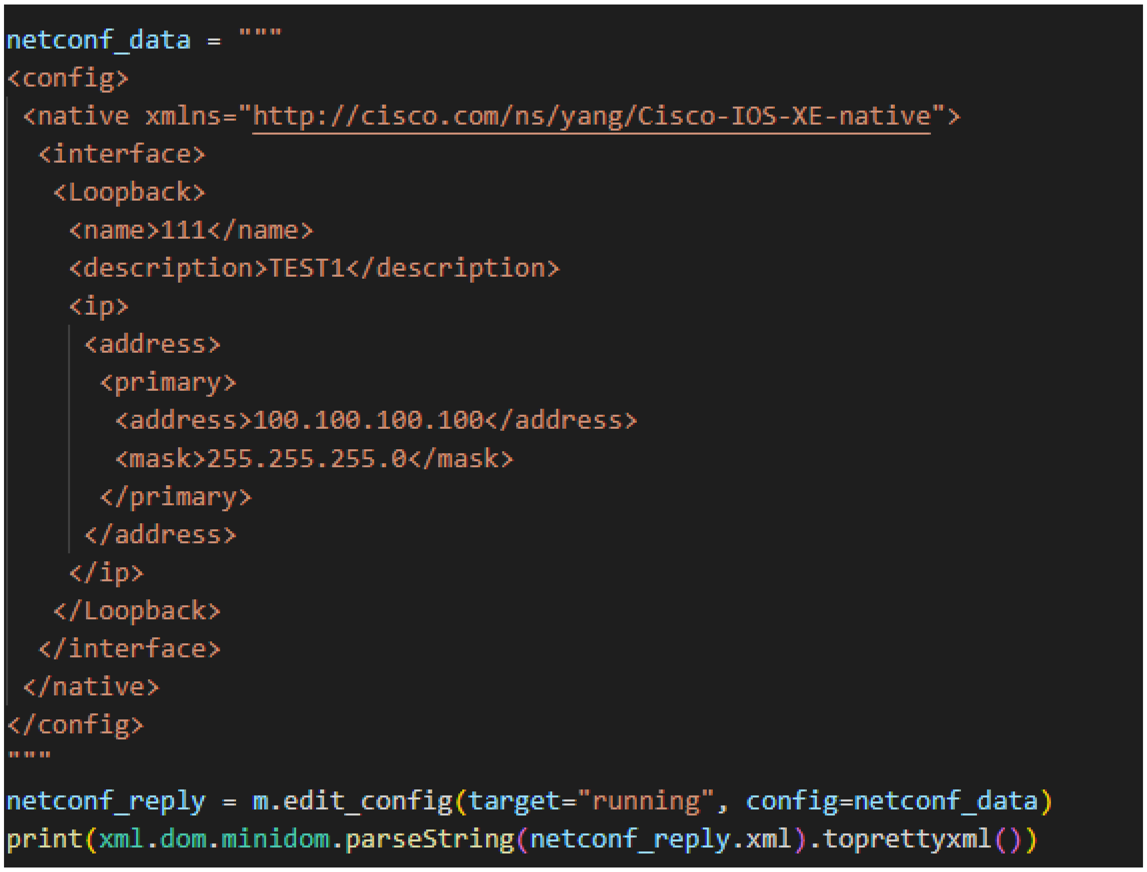
```
netconf_data = """
<config>
 <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
   <Loopback>
    <name>100</name>
    <description>TEST1</description>
    <ip>
     <address>
      <primary>
       <address>110.110.110.110</address>
       <mask>255.255.255.0</mask>
      </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())
```

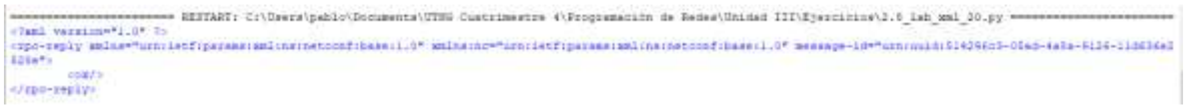


```
<?xml version="1.0" ?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uui
d:9d920137-7393-4549-8f63-53af9b08d689">
        <ok/>
</rpc-reply>
```



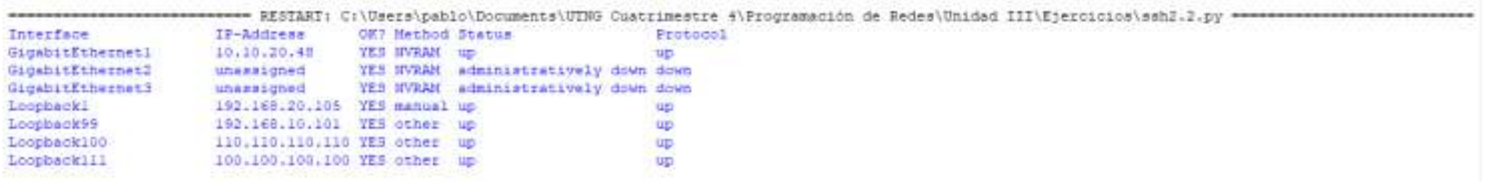
```
netconf_data = """
<config>
 <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
   <Loopback>
    <name>111</name>
    <description>TEST1</description>
    <ip>
     <address>
      <primary>
       <address>100.100.100.100</address>
       <mask>255.255.255.0</mask>
      </primary>
     </address>
    </ip>
   </Loopback>
  </interface>
 </native>
</config>
"""

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



```
====================== RESTART: C:\Users\pablo\Documents\UTNG Cuatrimestre 4\Programación de Redes\Unidad III\Ejercicios\2.6_lab_xml_20.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:51429[illegible]-0[illegible]-4a[illegible]-[illegible]-[illegible]
[illegible]">
        <ok/>
</rpc-reply>
```

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.



```
========================== RESTART: C:\Users\pablo\Documents\UTNG Cuatrimestre 4\Programación de Redes\Unidad III\Ejercicios\ssh2.2.py ==========================
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       10.10.20.48     YES NVRAM  up                    up
GigabitEthernet2       unassigned      YES NVRAM  administratively down down
GigabitEthernet3       unassigned      YES NVRAM  administratively down down
Loopback1              192.168.20.105  YES manual up                    up
Loopback99             192.168.10.101  YES other  up                    up
Loopback100            110.110.110.110 YES other  up                    up
Loopback111            100.100.100.100 YES other  up                    up
```

Conclusión

Dentro del código que teníamos que ejecutar fue relativamente fácil por que ya venia el código dentro del documento viene una variable con el netconf_date que es codijo XML para introducir en el router por medio de manager.connect() , hasta ahora no hemos tenido ningún problema si no que acepta el tipo XML que estamos introduciendo en el código , lo toma y se ejecuta un OK como debe de ser.

En el script que ejecutaremos tenemos que conectarnos por medio de su dirección , host , username y password , con una variable netconf_date que es código XML que se ejecutara dentro del router por medio del puerto 830 que es donde el código XML para después mostrar el código sin se ejecutó el programa con un OK dentro de un código XML.

NCCLIENT

Python dispone de una librería llamada ncclient (Netconf client), que nos abstrae gran parte de la complejidad y nos permite trabajar con mayor comodidad. Veremos a continuación como utilizarla para interactuar con los routers de forma programática.

¿Como funciona RPC dentro de NETCONF?

El protocolo NETCONF utiliza RPC para facilitar la comunicación entre el cliente (NETCONF administrador o aplicación) y el servidor (agente NETCONF o dispositivo administrado). Un cliente codifica un RPC en XML y lo envía a un servidor mediante una sesión segura orientada a la conexión. El servidor responde con una respuesta codificada en XML.

La comunicación entre el cliente y el servidor consiste en una serie de solicitudes alternas y responder mensajes. Los pares de NETCONF usan elementos <rpc> y <rpc-reply> para proporcionar transporte enmarcado independiente del protocolo de solicitudes y respuestas de NETCONF. El servidor NETCONF procesa las solicitudes de RPC secuencialmente en el orden en que se reciben.

¿Cómo funciona RCP con NETCONF y YANG?

NETCONF es un protocolo basado en estándar y codificado en lenguaje de marcado extensible (XML) que proporciona el transporte para comunicar la configuración con formato YANG o la solicitud de datos operativos de una aplicación que se ejecuta en una plataforma de administración centralizada al dispositivo Cisco desde el que un usuario desea configurar o solicitar datos operativos (comando show). Proporciona servicios basados en transacciones, como anular la solicitud de configuración completa cuando falla una parte de dicha solicitud de configuración.

¿Qué es rpc-reply/data/native?

Se envía un elemento <rpc-reply> en respuesta a cada solicitud de RPC. El elemento <rpc-reply> contiene el atributo obligatorio message-id copiado de la solicitud RPC correspondiente, junto con cualquier atributos adicionales que están presentes en la solicitud RPC.

Para las solicitudes <get> o <get-config> procesadas con éxito, los datos de respuesta se codifican como el contenido del elemento <rpc-reply>. Para solicitudes procesadas con éxito <edit-config> o <close-session>, el elemento <ok> es codificado como el contenido del elemento <rpc-reply>.

Para solicitudes RPC fallidas, uno o más elementos <rpc-error> están codificados dentro del elemento <rpc-reply>.