



UNIVERSIDAD TECNOLÓGICA DEL NORTE DE GUANAJUATO

Tecnologías de la Información y Comunicación
Programa educativo:

TSU en Infraestructura de Redes Digitales
Área académica:

Programación de Redes
Asignatura:

Unidad III: Programación de Redes

Grupo: GIR0441

NETCONF w/Python: Device Configuration
Laboratorio 8:

Venado Soria German Emiliano
Alumno:

Gabriel Barrón Rodríguez
Docente:

Dolores Hidalgo, C.I.N., Gto., Miércoles 14 de Diciembre de 2022
Lugar y fecha:

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.

- In Python IDLE, create a new Python script file:
- In the new Python script file editor, import the “manager” class from the ncclient module:

```
from ncclient import manager
```
- Using the `manager.connect ()` function, set up an `m` connection object to the IOS XE device.

```
practica.py - /home/german/Documentos/Programación
File Edit Format Run Options Window Help
from ncclient import manager

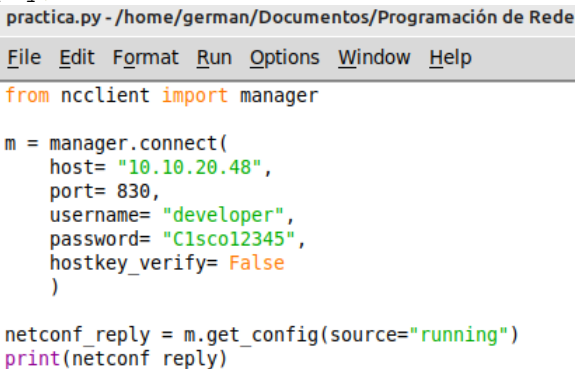
m = manager.connect(
    host= "10.10.20.48",
    port= 830,
    username= "developer",
    password= "Cisco12345",
    hostkey_verify= False
)
```

En estos primeros puntos de indica que debemos crear un nuevo archivo Python en IDLE, y comenzar a importar la clase “manager” del módulo ncclient, y posteriormente generar la sesión para tener la conexión remota hacia al Router.

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)
```



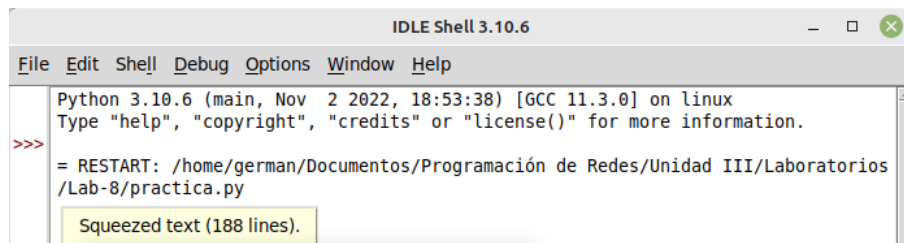
```
practica.py - /home/german/Documentos/Programación de Rede
File Edit Format Run Options Window Help
from ncclient import manager

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

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

Hay que utilizar la función **get_config()** del objeto de sesión NETCONF "m" para poder devolver e imprimir la configuración en ejecución del dispositivo. La función **get_config()** se complementa con un parámetro de cadena de "source" que define el almacén de datos NETCONF de origen.

- e. Execute the Python script and explore the output.



```
>>>
= RESTART: /home/german/Documentos/Programación de Redes/Unidad III/Laboratorios
/Lab-8/practica.py
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:
20dc30a4-c812-47d1-8b57-ee3fab011b0a" xmlns:nc="urn:ietf:params:xml:ns:netconf:b
ase:1.0"><data><native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native"><ver
sion>16.11</version><boot-start-marker/><boot-end-marker/><banner><motd><banner>
^C</banner></motd></banner><memory><free><low-watermark><processor>80557</proces
sor></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</ Contac
t-email-addr></profile xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-call-home"><p
rofile-name>CiscoTAC-1</profile-name><active>true</active></profile></call-home>
<service><timestamps><debug><datetime><msec></msec></datetime></log><date
time><msec></datetime></log></timestamps><call-home/></service><platform><conso
le xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-platform"><output>virtual</outpu
t></console></platform><hostname>csr1000v-1</hostname><enable><secret><type>9</t
ype><secret>$9$4fdKs44Vxjq2x.$Zahb8Zy6RQwM90NMWkNq4k3igSv4v2KkH/UUuqr09s</secre
t></secret></enable><username><name>cisco</name></privilege>15</privilege><secret
><encryption>9</encryption><secret>$9$yGG2/zjAgFbMt.$UnZG70.Ahe5Q6dkKyWuaIDmmuc
5.4ax8uQxE7zpzQU</secret></secret></username><username><name>developer</name><pr
ivilege>15</privilege><secret><encryption>9</encryption><secret>$9$hXxkjjeovP1Ns
E$djrwUSyHgy2dj/VbEM3Ce.wwesn89i0UyQubBy76dX.</secret></secret></username><usern
ame><name>root</name></privilege>15</privilege><secret><encryption>9</encryption>
<secret>$9$98fva3XgX2LKbE$NtueqkfF8CJ8eMs8w4ZrVCcqD30Ws3tVfFRPoeTgl6</secret></
secret></username><ip><domain><name>abc.inc</name></domain><forward-protocol><pr
otocol>nd</protocol></forward-protocol><route><ip><route><interface><forwarding-lis
t><prefix>0.0.0.0</prefix><mask>0.0.0.0</mask><fwd-list><fwd>GigabitEthernet1</f
wd></interface><next-hop><ip><address>10.10.20.254</ip></address></interface><next-hop
></fwd-list></ip><route><interface><forwarding-list></route><scp><server><enable></
server></scp><ssh><rsa><keypair><name>ssh-key</keypair></rsa><version>2</ve
rsion></ssh><access-list><extended xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-
acl"><name>meraki-fqdn-dns</name></extended></access-list><http xmlns="http://ci
sco.com/ns/yang/Cisco-IOS-XE-http"><authentication><local></authentication><ser
ver>true</server><secure-server>true</secure-server></http></ip></interface><Giga
bitEthernet><name>1</name><description>MANAGEMENT INTERFACE - DON'T TOUCH ME</de
```

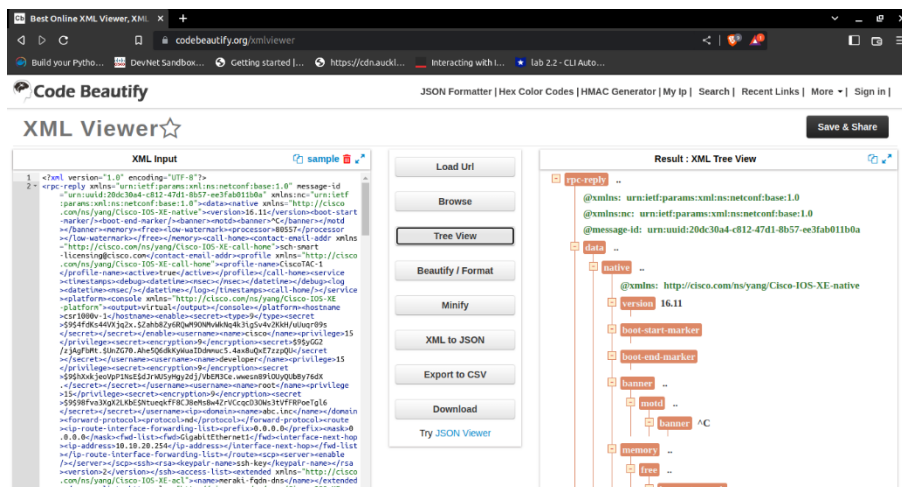
Esta es salida del script ejecutado, el cual nos comparte información en formato XML de la configuración del Router, aunque lo muestra de una manera un poco desordenada, y de esta manera puede complicar la lectura a los usuarios.

Step 2: Use CodeBeautify.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.



En este apartado se solicita copiar la salida del IDLE, que viene en formato XML, y pegarla en la página de **XML Viewer**, y dar clic “Tree View”, esto nos arrojará una salida en la parte derecha, con la misma información, pero de manera organizada, tal y como lo dice su opción, vista de árbol, esto para que sea mejor visualizado y leído por los usuarios.

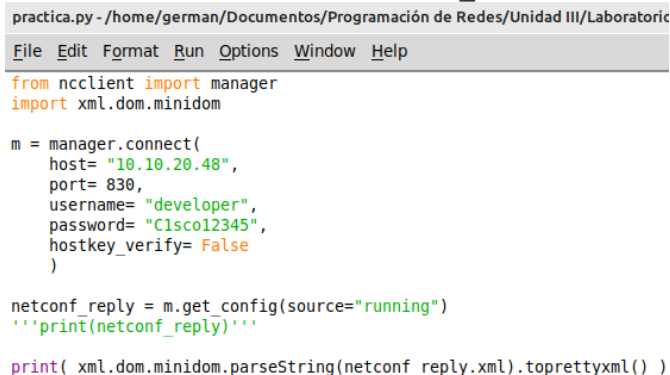
- h. To simplify the view, close the XML elements that are under the rpc-reply/data structure.
- i. Note that the opened rpc-reply/data/native element contains an attribute xmlns that points to “Cisco-IOS-XE-native” YANG model. That means this part of the configuration is Cisco Native for IOS XE.
- j. 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())
```



```
practica.py - /home/german/Documentos/Programación de Redes/Unidad III/Laboratorio
File Edit Format Run Options Window Help
from ncclient import manager
import xml.dom.minidom

m = manager.connect(
 host= "10.10.20.48",
 port= 830,
 username= "developer",
 password= "Cisco12345",
 hostkey_verify= False
)

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

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

En el tercer paso se indica que debemos importar otro módulo llamado “xml.dom.minidom” y reemplazar la impresión por: “print(xml.dom.minidom.parseString(netconf\_reply.xml).toprettyxml())”

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

```
= RESTART: /home/german/Documentos/Programación de Redes/Unidad III/Laboratorios/Lab-8/practica.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:2cd588c2-b706-42b2-ad94-1a0e31763202">
 <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><</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>
 <mssec/>
 </datetime>
 </debug>
 </timestamps>
 </service>
 </native>
 </data>
</rpc-reply>
```

Esta es la segunda salida después de haber importado el nuevo modulo y modificar la función de salida, como podemos apreciar, nos comparte la misma información pero ahora de una manera más organizada y estructurada.

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

- NETCONF has support to return only data that are defined in a filter element.
- 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>
"""
```

- 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())
```

practica.py - /home/german/Documentos/Programación de Redes/Unidad III/Laboratorio

File Edit Format Run Options Window Help

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

```
m = manager.connect(
 host= "10.10.20.48",
 port= 830,
 username= "developer",
 password= "Cisco12345",
 hostkey_verify= False
)
```

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

```
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())
```

En este paso se indica que utilizemos una especie de filtro para devolver una configuración específica definida por un modelo YANG. Esto lo vamos a poder obtener al momento de crear una variable (`netconf_filter`) la cual integra un elemento de filtro XML NETCONF que está diseñado para recuperar solo datos definidos por el modelo YANG nativo.

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

```
> RESTART: /home/german/Documentos/Programación de Redes/Unidad III/Laboratorios/Lab-8/practica.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:4757256f-6735-4b14-8398-61be0ba3f1d8">
 <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>08557</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>
 <sec/>
 </datetime>
 </debug>
 </timestamps>
 </service>
 </native>
 </data>
</rpc-reply>
```

Esta es otra de las salidas, usando el filtro, y como podemos observar, los datos específicos que solicitamos nos lo da de la misma forma organizada que el paso anterior, ya que se sigue empleando la misma estructura de la función de salida, solo agregado el filtro.

## Part 2: Update the Device's Configuration

### Step 1: Create a new Python script file.

- In IDLE, create a new Python script file.
- 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
)
```

```
parte_2.py - /home/german/Documentos/Pro
File Edit Format Run Options Window
from ncclient import manager
import xml.dom.minidom

m = manager.connect(
 host="10.10.20.48",
 port=830,
 username="developer",
 password="Cisco12345",
 hostkey_verify=False
)
```

En este apartado dejamos el mismo proceso de importar los módulos y la configuración de la sesión remota.

### Step 2: Change the hostname.

- In order to update an existing setting in the configuration, you can extract the setting location from the configuration retrieved in Step 1.
- The configuration update is always enclosed in a “config” XML element. This element includes a tree of XML elements that require updates.
- 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>
"""
```

```
parte_2.py - /home/german/Documentos/Programación de Redes/Unidad III/Labora
File Edit Format Run Options Window Help
from ncclient import manager
import xml.dom.minidom

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

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

Al mismo archivo se sugiere que agreguemos el código de configuración que esta misma practica nos brinda, para poder modificar el nombre del dispositivo. Para esto debemos utilizar la misma ubicación de configuración, y ahí dentro ingresaremos el nuevo nombre del host.

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

Además nos pide que modifiquemos la función para poder efectuar la configuración, en el cual incluye dos parámetros uno que se llama `target`, que es el almacén de datos de netconf de destino que deseamos actualizar, y el otro `config`, que es la actualización de la configuración.

- 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.

```
= RESTART: /home/german/Documentos/Programación de Redes/Unidad III/Laboratorios/Lab-8/parte_2.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:6d6256d6-a4e4-4799-b31c-7a0b73d4bcef">
 <ok/>
</rpc-reply>
```

Y esta es la salida esperada, la cual nos manda un OK, el cual representa que la configuración se actualizado correctamente.

- 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>
"""
```



The screenshot shows a code editor window titled 'parte\_2.py - /home/german/Documentos/Programación de Redes/Unidad III/Laboratorios/...'. The editor contains Python code that imports 'ncclient' and 'xml.dom.minidom', connects to a device, and defines the 'netconf\_data' variable. The XML content of 'netconf\_data' is displayed in a green monospace font, matching the code block above. The XML structure is as follows:

```
<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>
```

Vamos a seguir conservando la importación de los módulos y la sesión remota, solo modificaremos la configuración para poder crear una interfaz Loopback con el nombre “100”, añadiendo su dirección IP.

- 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())

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

Como siguiente estaremos agregando la interfaz creada anteriormente (Loopback100) editando la configuración del dispositivo existente empleando la función “edit\_config()”.

- 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

```
= RESTART: /home/german/Documentos/Programación de Redes/Unidad III/Laboratorios/Lab-8/parte_2.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:57866bb6-390f-4db0-ae93-b4a1fb915b1d">
 <ok/>
</rpc-reply>
```

Se muestra la salida esperada (OK), que nos hace saber que la configuración ha sido exitosa.

- 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>
"""
```

```

parte_2.py - /home/german/Documentos/Programación de Redes/Unidad III/Laboratorios/...
File Edit Format Run Options Window Help

from ncclient import manager
import xml.dom.minidom

m = manager.connect(
 host="10.10.20.48",
 port=830,
 username="developer",
 password="Cisco12345",
 hostkey_verify=False
)

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>
"""

```

En este ultimo estaremos repitiendo el mismo proceso que hemos venido haciendo desde antes, intentaremos crear otra interfaz Loopback con la misma dirección IP pero con distinto nombre, para ver si es posible o no la creación de una interfaz como ésta.

- 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())

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

```

Empleamos la misma función para generar la configuración, y veremos si nos da una salida exitosa o no.

- 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.

```
= RESTART: /home/german/Documentos/Programación de Redes/Unidad III/Laboratorios
/Lab-8/parte_2.py
Traceback (most recent call last):
 File "/usr/lib/python3.10/idlelib/run.py", line 578, in runcode
 exec(code, self.locals)
 File "/home/german/Documentos/Programación de Redes/Unidad III/Laboratorios/La
b-8/parte_2.py", line 33, in <module>
 netconf_reply = m.edit_config(target="running", config=netconf_data)
 File "/home/german/.local/lib/python3.10/site-packages/ncclient/manager.py", l
ine 246, in execute
 return cls(self._session,
 File "/home/german/.local/lib/python3.10/site-packages/ncclient/operations/edi
t.py", line 76, in request
 return self._request(node)
 File "/home/german/.local/lib/python3.10/site-packages/ncclient/operations/rpc
.py", line 375, in _request
 raise self._reply.error
ncclient.operations.rpc.RPCError: inconsistent value: Device refused one or more
commands
```

Como podemos ver, la salida no tuvo éxito, esto debido a que posiblemente no se pueden tener interfaces con una misma dirección IP.

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.

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

## Investigaciones

- **¿Por qué no se puede asignar la misma dirección IP a dos o más interfaces?**

Esto provoca que uno de esos equipos no pueda acceder a la red o que incluso afecte a los dos. Ya que las direcciones IP Públicas son siempre únicas, es decir, no se pueden repetir.

## Conclusiones

Como conclusión me llevo la misma idea e impresión que la vez que trabaje con POSTMAN, pues si observamos es muy similar el proceso de solicitudes para generar las actualizaciones de la configuración del Router.