

## 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.

- 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.

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

- 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)
```

- Execute the Python script and explore the output.

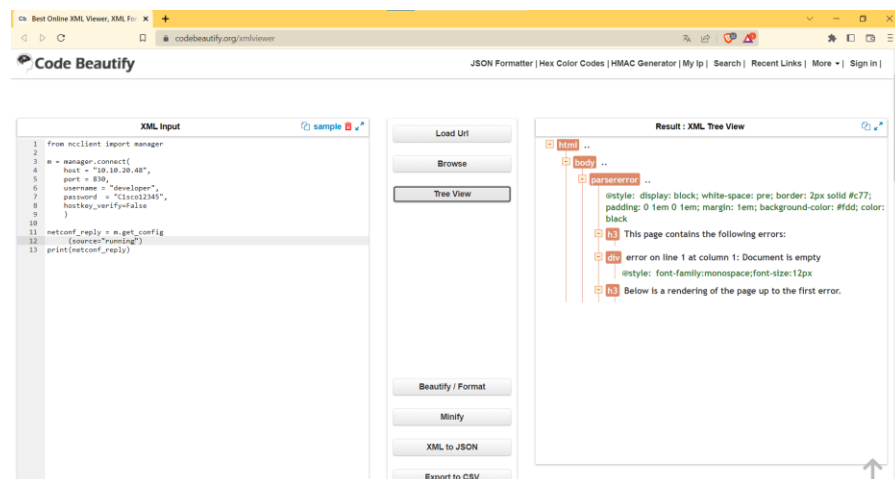
### Step 2: Use CodeBeautify.com to evaluate the response.

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

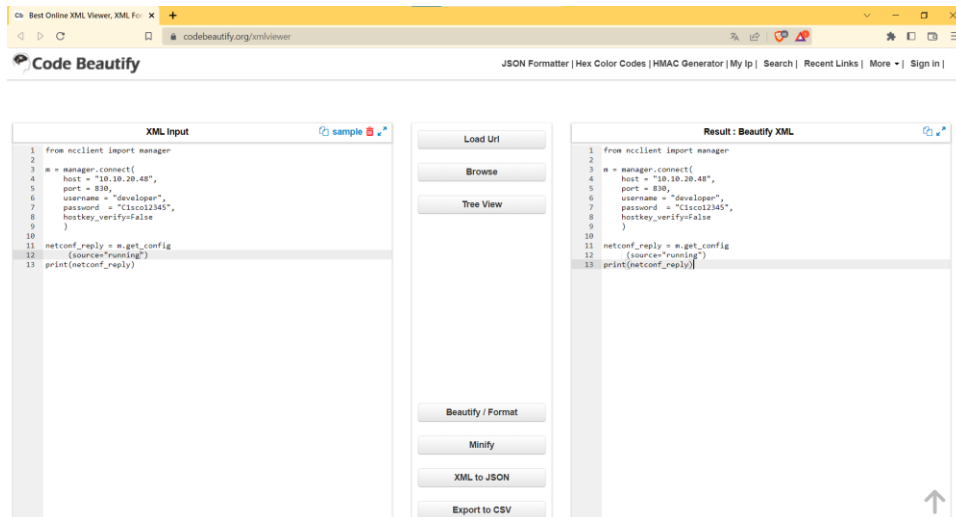
- Copy the XML from IDLE to XML Viewer.

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



## SALIDA

```

Python 3.10.9 (tags/v3.10.9:1dd9be6, Dec 6 2022, 20:01:21) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits()" or "license()" for more information.
>>>
- RESTART: D:\UTWG\UNIVERSIDAD\Cuarto cuatrimestre/Programación de redes/Unidad III/Scripts/lab2.8.py
<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:b1b4b4c6-80fb-463d-b508-0f324b33e932">
  <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>
              <sec/>
            </datetime>
          </debug>
          <log>
            <datetime>
              <sec/>
            </datetime>
          </log>
        </timestamps>
        <call-home/>
      </service>
      <platform>
        <console xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-platform">

```

### 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())

```

- Execute the updated Python script and explore the output

```

Python 3.10.9 (tags/v3.10.9:1dd9be6, Dec 6 2022, 20:01:21) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits()" or "license()" for more information.
>>>
- RESTART: D:\UTWG\UNIVERSIDAD\Cuarto cuatrimestre/Programación de redes/Unidad III/Scripts/lab2.8.py
<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:b1b4b4c6-80fb-463d-b508-0f324b33e932">
  <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>
              <sec/>
            </datetime>
          </debug>
          <log>
            <datetime>
              <sec/>
            </datetime>
          </log>
        </timestamps>
        <call-home/>
      </service>
      <platform>
        <console xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-platform">

```

## 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:

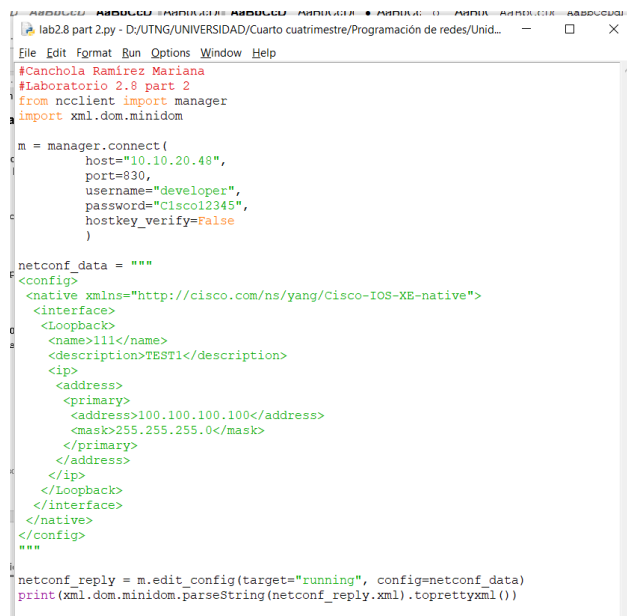
```
from ncclient import manager
import xml.dom.minidom

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

### Step 2: Change the hostname.

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



```
lab2.8 part 2.py - D:/UTNG/UNIVERSIDAD/Cuarto cuatrimestre/Programación de redes/Unid...
File Edit Format Run Options Window Help

#Canchola Ramirez Mariana
#Laboratorio 2.8 part 2
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>l11</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())
```

## SALIDA

```

Python 3.10.9 (tags/v3.10.9:1dd9be6, Dec 6 2022, 20:01:21) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:/UTNG/UNIVERSIDAD/Cuarto cuatrimestre/Programación de redes/Unidad III/Scripts/lab2.8 part 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:0edb3111-a204-4319-8076-facc8171c3b6">
  <ok/>
</rpc-reply>
>>>
  
```

```

2.py - D:/UTNG/UNIVERSIDAD/Cuarto cuatrimestre/Programación de redes/Unid...
Ramírez Mariana
io 2.8 part 2
ent import manager
.dom.minidom

t.connect(
    ost="10.10.20.48",
    ort=830,
    sername="developer",
    assword="Cisco12345",
    ostkey_verify=False

ta = ""

nlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
ce>
ck>
ll1</name>
iption>TEST1</description>

ess>
mary>
dress>100.100.100.100</address>
sk>255.255.255.0</mask>
imary>
ress>
  
```

- 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**

```

10.10.20.48 - PuTTY
End of keyboard-interactive prompts from server

Welcome to the DevNet Sandbox for CSR1000v and IOS XE

The following programmability features are already enabled:
- NETCONF
- RESTCONF

Thanks for stopping by.

csr1000v-1#
csr1000v-1#sh ip brief
^
% Invalid input detected at '^' marker.

csr1000v-1#sh ip int brief
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
Loopback111      100.100.100.100 YES other  up              up
csr1000v-1#
  
```

## Ejectando comando **show int desc**

```

GigabitEthernet3      unassigned      YES NVRAM  administratively down down
Loopback111           100.100.100.100 YES other  up              up
csr1000v-1#sh int desc
Interface      Status          Protocol Description
Gi1            up              up      MANAGEMENT INTERFACE - DO
N'T TOUCH ME
Gi2            admin down     down     Network Interface
Gi3            admin down     down     Network Interface
Lo111          up              up      TEST1
csr1000v-1#
  
```

## Conclusiones

*En este ejercicio complementé lo del ejercicio 2.7 ya que seguimos utilizando y trabajando con la librería NCCLIENT, la cual ya en esta practica pude ver y entender mejor 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")
```