UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE CIENCIAS

ÁREA DE CIENCIA DE LA COMPUTACIÓN

11° LABORATORIO - CC312

- **TÍTULO:** Utilice NETCONF para acceder a un dispositivo IOS XE
- **ALUMNO:**

JHONATAN FLORINS POMA MARTINEZ 20182729F

- **PROFESORES:** YURI JAVIER.,CCOICCA PACASI

2023

Objetivos

Parte 1: Armar la red y verificar la conectividad.

Parte 2: Utilizar una sesión de NETCONF para recopilar información.

Parte 3: Usar ncclient para conectarse a NETCONF.

Parte 4: Usar ncclient para recuperar la configuración.

Parte 5: Usar ncclient para configurar un dispositivo.

Parte 6: Desafío: Modificar el programa utilizado en este laboratorio.

Introducción:

NETCONF, (Protocolo de Configuración de Red) es un protocolo estandarizado de gestión de redes definido por el Grupo de Trabajo de Ingeniería de Internet (IETF, por sus siglas en inglés). Está diseñado para gestionar y configurar dispositivos de red, **como enrutadores**, conmutadores y firewalls, a través de una red. NETCONF opera sobre una capa de transporte segura, típicamente utilizando SSH (Secure Shell) como protocolo subyacente.
NETCONF proporciona una interfaz programática para administrar dispositivos de red, permitiendo a los administradores configurar y supervisar diversos aspectos de la red de forma remota. Utiliza XML (Lenguaje de Marcado Extensible) para la codificación de datos y admite varias operaciones, incluyendo la obtención de la configuración del dispositivo, la modificación de parámetros de configuración y la suscripción a notificaciones de eventos.

SSH, (Secure Shell) es un protocolo de red que proporciona una forma segura de acceder y administrar de manera remota dispositivos y servidores a través de una conexión cifrada.
SSH utiliza técnicas de cifrado para proteger la confidencialidad y la integridad de los datos transmitidos a través de la red. Proporciona un mecanismo de autenticación robusto para verificar la identidad del usuario y del servidor remoto.
Al establecer una conexión SSH, se crea un canal seguro entre el cliente y el servidor, a través del cual se pueden enviar comandos y transferir datos de manera segura.

SESIÓN SSH, es una conexión segura establecida entre un cliente y un servidor utilizando el protocolo SSH. Durante una sesión SSH, se crea un canal seguro a través del cual se pueden transmitir comandos y datos de forma cifrada y protegida.

- Cliente SSH: El cliente SSH es el dispositivo o sistema desde el cual se inicia la conexión y se realiza la solicitud para establecer una sesión SSH. El cliente SSH es responsable de iniciar la comunicación y enviar comandos al servidor remoto. Puede ser una computadora personal, una estación de trabajo o incluso otro servidor.
- Servidor SSH: El servidor SSH es el dispositivo o sistema que recibe la solicitud de conexión del cliente y permite que se establezca la sesión SSH. El servidor SSH está configurado para aceptar conexiones entrantes y autenticar a los clientes. Una vez que se establece la conexión, el servidor SSH procesa los comandos enviados por el cliente y devuelve las respuestas correspondientes.

El servidor SSH podría ser nuestro **router virtual CSR1kv**. El router virtual CSR1kv es un dispositivo virtualizado que puede funcionar como un enrutador en un entorno de red. Al habilitar el servidor SSH en el router virtual CSR1kv, permite que los clientes SSH se conecten a él y establezcan sesiones SSH para administrar y configurar el router de forma remota.
Nosotros, **como cliente SSH,** podemos enviar comandos al router virtual CSR1kv para realizar configuraciones, monitorear el estado del dispositivo y realizar otras tareas de administración.

RPC, (Remote Procedure Call) es un protocolo de comunicación que permite a un programa en un sistema solicitar y ejecutar un procedimiento en un sistema remoto como si fuera un procedimiento local. Es una técnica utilizada para la comunicación entre procesos distribuidos en una red.
En el modelo RPC, un programa cliente invoca una llamada a un procedimiento remoto en un sistema servidor. El cliente envía una solicitud al servidor, especificando el nombre del procedimiento y los parámetros necesarios. El servidor recibe la solicitud, ejecuta el procedimiento y envía una respuesta de vuelta al cliente con el resultado o el valor de retorno.

Dispositivos IOS XE, Los dispositivos iOS XE son una línea de productos de Cisco que ejecutan el sistema operativo Cisco IOS XE (Internetwork Operating System XE). Estos dispositivos son enrutadores y conmutadores de red de alto rendimiento utilizados en entornos empresariales y proveedores de servicios.
El **Cisco CSR1kv** (Cloud Services Router 1000V) también es un dispositivo que ejecuta el sistema operativo Cisco IOS XE. El CSR1kv es una versión virtualizada de un enrutador Cisco, diseñado para funcionar en entornos de nube y virtualización.

PARTE 1: Armar la red y verificar la conectividad.

Mi IP.

```
CSR1kv>show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.103  YES DHCP   up                    up
CSR1kv>_
```

Haciendo ping para corroborar la conexion:

```
devasc@labvm:~$ ping -c 5 192.168.56.103
PING 192.168.56.103 (192.168.56.103) 56(84) bytes of data.
64 bytes from 192.168.56.103: icmp_seq=1 ttl=255 time=427 ms
64 bytes from 192.168.56.103: icmp_seq=2 ttl=255 time=0.819 ms
64 bytes from 192.168.56.103: icmp_seq=3 ttl=255 time=0.937 ms
64 bytes from 192.168.56.103: icmp_seq=4 ttl=255 time=1.12 ms
64 bytes from 192.168.56.103: icmp_seq=5 ttl=255 time=0.677 ms

--- 192.168.56.103 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4035ms
rtt min/avg/max/mdev = 0.677/86.035/426.623/170.293 ms
devasc@labvm:~$
```

Hacemos una conexión SSH, y veremos el promt de csr1kv#
En el terminal para la máquina virtual DEVASC, conecte SSH a la máquina virtual CSR1kV con el siguiente comando: **devasc @ labvm: ~ $ ssh cisco@192.168.56.103**

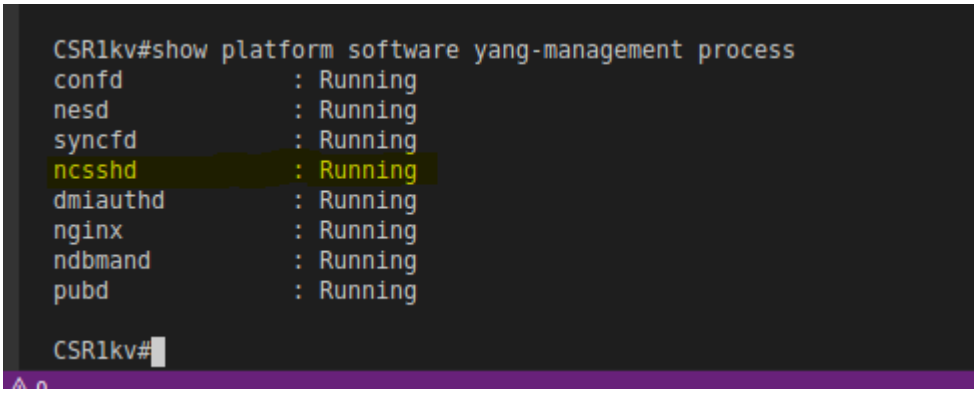
```
devasc@labvm:~$ ssh cisco@192.168.56.103
The authenticity of host '192.168.56.103 (192.168.56.103)' can't be established.
RSA key fingerprint is SHA256:ucVnFzem3xJsmdOusbtHveixQo2Qm9GNpG4IFjAZrWM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.56.103' (RSA) to the list of known hosts.
Password:
Password:

*                                        *
**                                      **
***                                    ***
***  Cisco Networking Academy          ***
***                                    ***
***  This software is provided for     ***
***   Educational Purposes             ***
***    Only in Networking Academies    ***
***                                    ***
**                                      **
*                                        *


CSR1kv#
```

Dejamos abierto la sesion SSH .

PARTE 2: Utilizar una sesión de NETCONF para recopilar información.

En esta parte, verificará que NETCONF se está ejecutando, habilitará NETCONF si no lo está y verificará que NETCONF está listo para una conexión SSH. A continuación, YSoU se conectará al proceso NETCONF, iniciará una sesión NETCONF, recopilará información de la interfaz y cerrará la sesión.

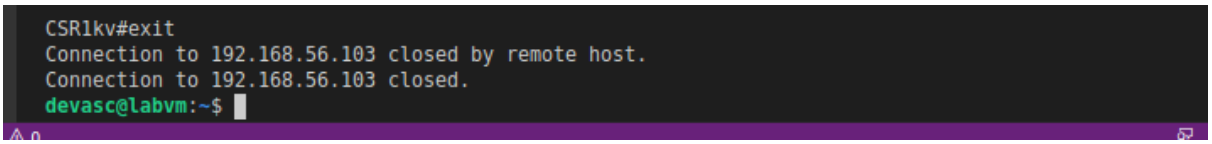
Paso 1: Comprobar si NETCONF se está ejecutando en CSR1kv.



Si NETCONF no se esta ejecutando, ingresamos el siguiente comando

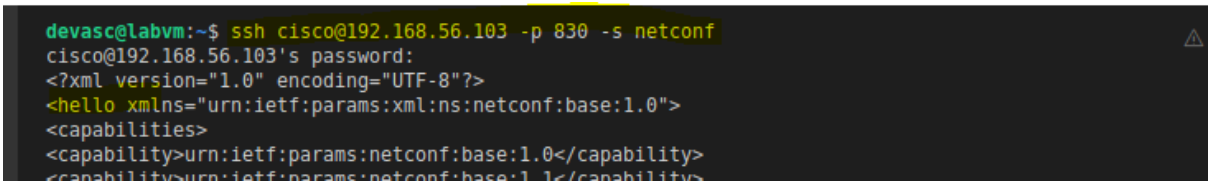
```
CSR1kv# config t
CSr1kv (config) # netconf-yang
```

Pero en nuestro caso, si se esta ejecutando, cerramos la sesion **SSH,** con el comando exit.

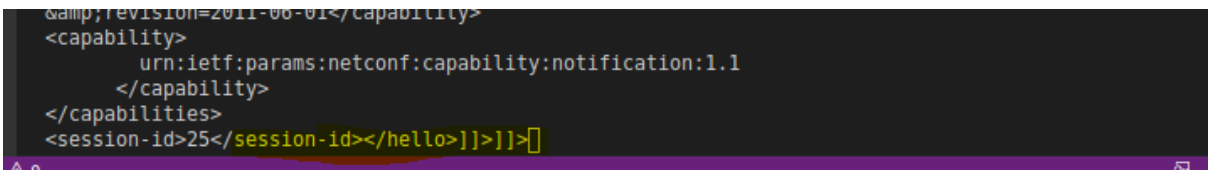


Paso 2: Acceda al proceso NETCONF, a través de un terminal SSH.
En este paso, restablecerá una sesión SSH con CSR1kV. Pero esta vez, especificará el puerto **NETCONF 830** y enviará netconf como un comando de subsistema.
Ingresamos el comando:
devasc @labvm: ~$ ssh cisco@192.168.56.103 -p 830 -s netconf
cisco@192.168.56.101's password: **cisco 123!**

El CSR1kv nos responderá con un mensaje de saludo que incluye más de 400 líneas de salida enumerando todas sus capacidades NETCONF. El final de los mensajes NETCONF se identifica con]]>]]>.



.....



Paso 3: Iniciar una sesión de NETCONF enviando un **mensaje saludo desde el cliente**.
Para iniciar una sesión NETCONF, el cliente necesita enviar su propio mensaje de saludo. El mensaje de saludo debe incluir la versión de capacidades base de NETCONF que el cliente desea utilizar.

Copie y pegue el siguiente código XML en la sesión SSH. Observe que el final del mensaje saludo del cliente se identifica con un]]>]]>.

```
<capability>urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults?module=ietf-netconf-with-defaults
&amp;revision=2011-06-01</capability>
<capability>
       urn:ietf:params:netconf:capability:notification:1.1
     </capability>
</capabilities>
<session-id>31</session-id></hello>]]>]]> <hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"> <c
apabilities> <capability>urn:ietf:params:netconf:base:1.0 </capability> </capabilities> </hello> ]]>
]]>
```

En el router virtual CSR1kv verificamos con el comando **CSR1kv# show netconf-yang sessions**, para verificar que se ha iniciado una sesion NETCONF.

```
CSR1kv#show netconf-yang sessions
R: Global-lock on running datastore
C: Global-lock on candidate datastore
S: Global-lock on startup datastore

Number of sessions : 1

session-id  transport    username              source-host           global-lock

------------------------------------------------------------------------------
31          netconf-ssh  cisco                 192.168.56.101        None

CSR1kv#_
```

Paso 4: Enviar mensajes RPC a un dispositivo IOS XE.

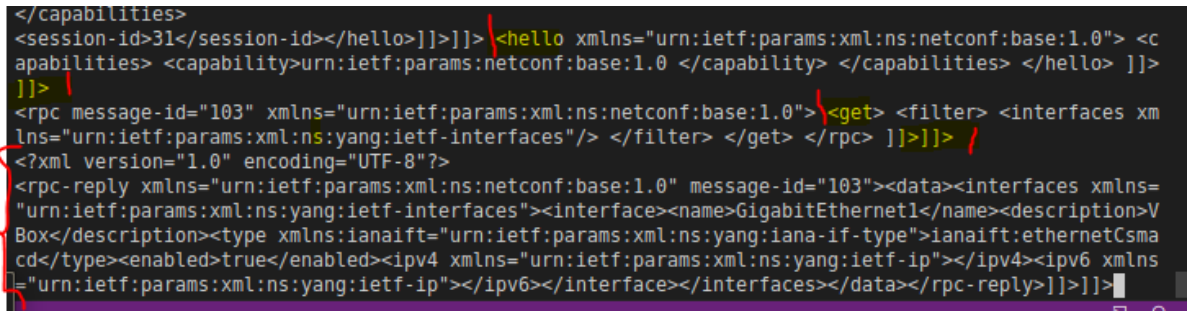
Durante una sesión SSH, un cliente NETCONF puede utilizar mensajes de llamada a procedimiento remoto (RPC) para enviar operaciones NETCONF al dispositivo IOS XE. La tabla enumera algunas de las operaciones NETCONF más comunes.

Operación	Descripción
<get>	Recupera la configuración en ejecución y la información de estado del dispositivo.
<get-config>	Recupera todo, o parte de un almacén de datos de configuración especificado.
<edit-config>	Carga toda, o parte de una configuración en el almacén de datos de configuración especificado.

Operación	Descripción
<copy-config>	Reemplaza un almacén de datos de configuración completo por otro.
<delete-config>	Elimina un almacén de datos de configuración.
<commit>	Copia data store candidato en el data store en ejecución.
<lock>/<unlock>	Bloquea o desbloquea todo el sistema de almacenamiento de datos de configuración.
<close-session>	Finaliza correctamente una sesión de NETCONF.
<kill-session>	Fuerza la finalización de una sesión de NETCONF.

a. Copie y pegue el siguiente código XML de obtener mensaje RPC en la sesión SSH de terminal para recuperar información acerca de las interfaces en R1.

```
<rpc message-id="103" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<get>
        <filter>
                <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"/>
        </filter>
</get>
</rpc> ]]>]]>
```



Copiamos lo de la llave rojo , sin el]]>]]> , y en internet buscamos para que sea mas legible:

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="103"><data><interfaces
xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
<interface><name>GigabitEthernet1</name>
<description>VBox</description><type
xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-
type">ianaift:ethernetCsmacd</type>
<enabled>true</enabled><ipv4
xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"></ipv4><ipv6
xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"></ipv6>
</interface></interfaces></data></rpc-reply>
```

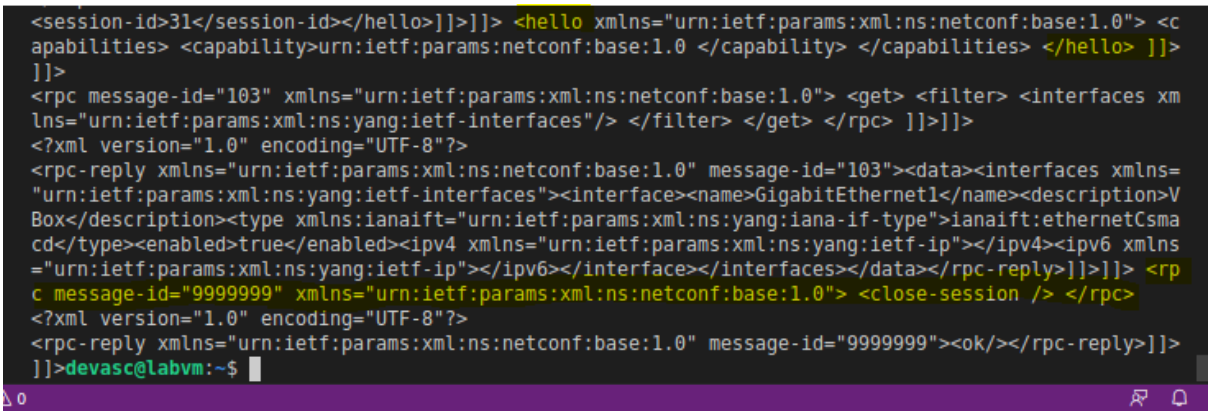
XML Pretty Printed

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="103">
  <data>
    <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
      <interface>
        <name>GigabitEthernet1</name>
        <description>VBox</description>
        <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">ianaift:ethernetCsmacd</type>
        <enabled>true</enabled>
        <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"/>
        <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"/>
      </interface>
    </interfaces>
  </data>
</rpc-reply>
```

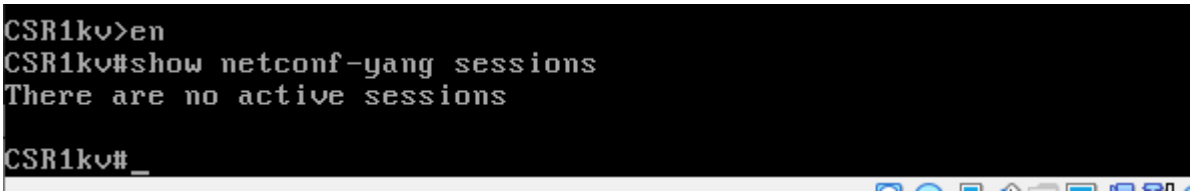
Paso 5: Cierre la sesión NETCONF.

a) Para cerrar la sesión NETCONF, el cliente necesita enviar el siguiente mensaje RPC:

```
<rpc message-id="9999999" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <close-session />
</rpc>
```



b) Verificamos en el router virtual , nuevamente con **CSR1kv# show netconf-yang sessions**

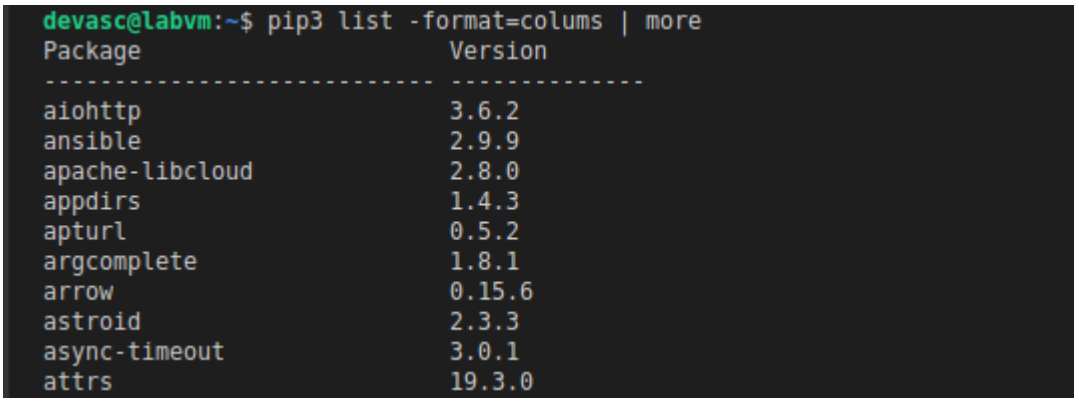


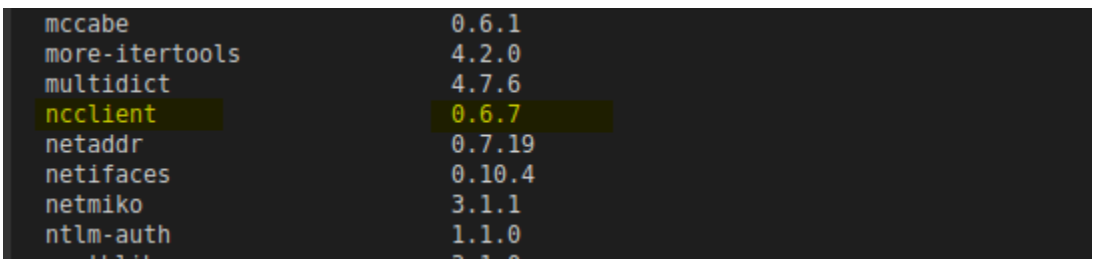
PARTE 3: USAR ncclient PARA CONECTARSE A NETCONF.

Trabajar con NETCONF no requiere trabajar con mensajes RPC NETCONF sin procesar y XML. En esta parte, aprenderá cómo usar el módulo **ncclient** Python para interactuar fácilmente con dispositivos de red utilizando NETCONF. Además, aprenderá cómo identificar qué modelos de YANG son compatibles con el dispositivo.

Esta información es útil cuando se crea un sistema de automatización de redes de producción que requiere que los modelos YANG específicos sean compatibles con el dispositivo de red dado.

Paso 1: Compruebe que ncclient está instalado y listo para su uso.





Paso 2: Cree un script para usar ncclient para conectarse al servicio NETCONF.
El **módulo ncclient** proporciona una **clase manager** con un **método connect ()** para configurar las conexiones NETCONF remotas. Después de una conexión correcta, el objeto devuelto representa la conexión NETCONF con el dispositivo remoto.
Creamos una variable **m** para representar el método connect (). El método connect () incluye toda la información necesaria para conectarse al servicio NETCONF que se ejecuta en CSR1kV. Tenga en cuenta que el puerto es 830 para NETCONF.

ncclient-netconf.py

```
netconf > ncclient-netconf.py > ...
1  from ncclient import manager
2  m = manager.connect(
3      host="192.168.56.103",
4      port=830,
5      username="cisco",
6      password="cisco123!",
7      hostkey_verify=False
8  )
```

OBS:
Si **hostkey_verify** se establece en True, CSR1kV nos pedirá que verifique la huella digital SSH. En un entorno de laboratorio, es seguro establecer este valor en False, como hemos hecho aquí.

Guarde y ejecute el programa para verificar que no haya errores. Aún no verá ninguna salida.
En DEVASC:

```
devasc@labvm:~/labs/devnet-src/netconf$
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
devasc@labvm:~/labs/devnet-src/netconf$
```

EN CSR1kv: Podemos verificar que el router aceptó la solicitud de una sesión NETCONF.

```
CSR1kv>
*Jun 23 16:30:20.864: %DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'cisco' authentic
ated successfully from 192.168.56.101:49404  for netconf over ssh. External grou
ps: PRIV15
CSR1kv>
```

Paso 3 : Agregar una función de impresión al script para que aparezcan las capacidades de NETCONF para CSR1kv.
El objeto **m** devuelto por la **función manager.connect ()** representa la sesión remota NETCONF. Como ha visto anteriormente, en cada sesión de NETCONF, el servidor envía primero sus capacidades, que es una lista, en formato XML, de los modelos YANG soportados. Con el módulo ncclient, la lista de capacidades recibida se almacena en la lista **m.server_capabilities.**
Usaremos un bucle for y una función de impresión para mostrar las capacidades del dispositivo:

ncclient-netconf.py

```
netconf > ncclient-netconf.py > ...
1  from ncclient import manager
2  m = manager.connect(
3      host="192.168.56.103",
4      port=830,
5      username="cisco",
6      password="cisco123!",
7      hostkey_verify=False
8  )
9
10 print("#Supported Capabilities (YANG models):")
11 for capability in m.server_capabilities:
12     print(capability)
13
```

Ejecutamos:

La salida es la misma salida que obtuvo al enviar el mensaje de saludo complejo anteriormente, pero sin la <capability> etiqueta XML de apertura y cierre en cada línea.

```
devasc@labvm:~/labs/devnet-src/netconf$
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
#Supported Capabilities (YANG models):
urn:ietf:params:netconf:base:1.0
urn:ietf:params:netconf:base:1.1
urn:ietf:params:netconf:capability:writable-running:1.0
urn:ietf:params:netconf:capability:xpath:1.0
urn:ietf:params:netconf:capability:validate:1.0
urn:ietf:params:netconf:capability:validate:1.1
```

.....

```
urn:ietf:params:xml:ns:yang:smiv2:VPN-TC-STD-MIB?module=VPN-TC-STD-MIB&revision=2005-11-15
urn:ietf:params:xml:ns:netconf:base:1.0?module=ietf-netconf&revision=2011-06-01
urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults?module=ietf-netconf-with-defaults&revision=20
11-06-01

        urn:ietf:params:netconf:capability:notification:1.1

devasc@labvm:~/labs/devnet-src/netconf$
```

PARTE 4: USAR ncclient PARA RECUPERAR LA CONFIGURACIÓN.

En esta parte, utilizará **NETCONF ncclient** para recuperar la configuración del CSR1kv, utilizará el **módulo xml.dom.minidom** para formatear la configuración y un filtro con **get_config()** para recuperar una parte de la configuración en ejecución.

Paso 1: Utilice la función **get_config ()** para recuperar la configuración en ejecución para R1.

Usamos el método get_config () del objeto de sesión **m NETCONF** para recuperar la configuración del CSR1kv. El único almacén de datos NETCONF actualmente en CSR1kV es el almacén de datos en ejecución. Puede verificar esto con el comando show netconf-yang datastores.

ncclient-netconf.py

```
netconf > ncclient-netconf.py > ...
1   from ncclient import manager
2   m = manager.connect(
3       host="192.168.56.103",
4       port=830,
5       username="cisco",
6       password="cisco123!",
7       hostkey_verify=False
8   )
9   '''
10  # Omitimos la salida de las capacidades de visualización (más de 400 líneas),
11  # Comentamos las instrucciones que imprimen las capacidades,
12  print("#Supported Capabilities (YANG models):")
13  for capability in m.server_capabilities:
14      print(capability)
15  '''
16  netconf_reply = m.get_config(source="running")
17  print(netconf_reply)
```

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:45a72877-11ec-4e2e-a
164-d515b2b11325" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"><data><native xmlns="http://cis
co.com/ns/yang/Cisco-IOS-XE-native"><version>16.12</version><boot-start-marker/><boot-end-marker/><b
anner><motd><banner>^C</banner></motd></banner><memory><free><low-watermark><processor>72297</proces
sor></low-watermark></free></memory><call-home><contact-email-addr xmlns="http://cisco.com/ns/yang/C
isco-IOS-XE-call-home">sch-smart-licensing@cisco.com</contact-email-addr><profile xmlns="http://cisc
```

Le damos formato usando, https://www.samltool.com/prettyprint.php

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xm
  <data>
    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-nati
      <version>16.12</version>
      <boot-start-marker/>
      <boot-end-marker/>
      <banner>
```

Paso 2: Utilice Python para petrificar el XML.

Python ha incorporado soporte para trabajar con archivos XML. El módulo xml.dom.minidom se puede utilizar para petrificar la salida con la **función toprettyxml()**.

Importamos el módulo xml.dom.minidom.

Reemplazamos en la edicion.

ncclient-netconf.py

```
netconf > ncclient-netconf.py > ...
1  import xml.dom.minidom
2  from ncclient import manager
3  m = manager.connect(
4      host="192.168.56.103",
5      port=830,
6      username="cisco",
7      password="cisco123!",
8      hostkey_verify=False
9  )
10 '''
11 # Omitimos la salida de las capacidades de visualización (más de 400 líneas),
12 # Comentamos las instrucciones que imprimen las capacidades,
13 print("#Supported Capabilities (YANG models):")
14 for capability in m.server_capabilities:
15     print(capability)
16 '''
17 netconf_reply = m.get_config(source="running")
18 #print(netconf_reply)
19 print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())
```

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.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:e1c155bf-85a4-4ea4-a00a-15d7e07118f6">
        <data>
                <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
                        <version>16.12</version>
                        <boot-start-marker/>
                        <boot-end-marker/>
                        <banner>
                                <motd>
                                        <banner>^C</banner>
                                </motd>
                        </banner>
                        <memory>
                                <free>
                                        <low-watermark>
                                                <processor>72297</processor>
                                        </low-watermark>
                                </free>
                        </memory>
                        <call-home>
```

Paso 3: Utilizar un filtro con get_config () para recuperar solo un modelo específico de YANG.

Es posible que un administrador de red solo desee recuperar una parte de la configuración en ejecución en un dispositivo. NETCONF admite devolver solo datos definidos en un parámetro de filtro de la función **get_conf()**

a) Creamos una variable llamada **netconf_filter** que solo recupere los datos definidos por el modelo YANG nativo de Cisco IOS XE.

```
21
22  netconf_filter = """
23  <filter>
24      <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native" />
25  </filter>
26  """
27  netconf_reply = m.get_config(source="running", filter=netconf_filter)
28  print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())
```

b) <native> esta vez solo se muestra el elemento XML. Anteriormente, se mostraban todos los modelos YANG disponibles en el CSR1kv

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.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:def512b5-a87f-472e-a2f4-db83dbe86da4">
        <data>
                <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
                        <version>16.12</version>
                        <boot-start-marker/>
                        <boot-end-marker/>
                        <banner>
                                <motd>
                                        <banner>^C</banner>
```

PARTE 5: USAR ncclient PARA CONFIGURAR UN DISPOSITIVO.

En esta parte, usaremos **ncclient** para configurar el CSR1kv utilizando el método **edit_config()** del módulo manager.

Paso 1: Utilizar ncclient para editar el nombre de host del CSR1kv.

Ejecutando: **#python3 ncclient-netconf.py**

Para este paso, estableceremos una variable para cambiar el **<hostname>** valor.

```
                                        <disable-kernel-core>true</disable-kernel-core>
                                </punt-keepalive>
                        </platform>
                        <hostname>CSR1kv</hostname>
                        <username>
                                <name>cisco</name>
                                <privilege>15</privilege>
                                <password>
                                        <encryption>0</encryption>
```

Para modificar la configuración de un dispositivo, definirá una <config> variable. Agregue la siguiente variable a la secuencia de comandos ncclient_netconf.py.

Puede usar NEWHOSTNAME o cualquier nombre de host que desee.

```
30  #Configuraremos usando el metodo edit_config()
31  netconf_hostname = """
32  <config>
33    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
34      <hostname>NEWHOSTNAME</hostname>
35    </native>
36  </config>
37  """
38  netconf_reply = m.edit_config(target="running", config=netconf_hostname)
39  print (xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())
```

Ejecutando:
devasc@labvm:~/labs/devnet-src/netconf$ **python3 ncclient-netconf.py**

```
                                <disable-kernel-core>true</disable-kernel-core>
                        </punt-keepalive>
                </platform>
                <hostname>NEWHOSTNAME</hostname>
                <username>
                        <name>cisco</name>
                        <privilege>15</privilege>
```

```
        </data>
</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:042cd227-3c1e-425d-9358-32066bbfa0de">
        <ok/>
</rpc-reply>
```

En el router virtual CSR1kv, cambia:

```
*                                          *
**                                        **
***                                      ***
***  Cisco Networking Academy            ***
***                                      ***
***  This software is provided for       ***
***   Educational Purposes               ***
***    Only in Networking Academies      ***
***                                      ***
**                                        **
*                                          *

NEWHOSTNAME>_
```

lo volvemos a su normalidad **CSR1kv>**, eliminados o comentamos el codigo añadido.

```
NEWHOSTNAME>
*Jun 23 19:29:17.865: %SYS-5-CONFIG_P: Configured programmatically by process io
sp_vty_100001_dmi_nesd from console as NETCONF on vty32131
*Jun 23 19:29:17.866: %DMI-5-CONFIG_I: R0/0: nesd: Configured from NETCONF/RESTC
ONF by cisco, transaction-id 76
NEWHOSTNAME>
CSR1kv>_
```

Paso 2: Utilizar ncclient para crear una nueva interfaz de bucle invertido en R1.

a) Creamos una nueva variable **<config>** para contener la configuración de una nueva interfaz de bucle invertido.La llamaremos : **netconf_loopback .** Agregue lo siguiente a su script ncclient_netconf.py.

Usamos la funcion **edit_config()** a su **ncclient_netconf.py** para enviar la nueva configuracion de bucle invertido a R1

```
ncclient-netconf.py ●
netconf > ncclient-netconf.py > ...
57
58  netconf_loopback = """
59  <config>
60   <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
61    <interface>
62     <Loopback>
63      <name>1</name>
64      <description>My first NETCONF loopback</description>
65      <ip>
66       <address>
67        <primary>
68         <address>10.1.1.1</address>
69         <mask>255.255.255.0</mask>
70        </primary>
71       </address>
72      </ip>
73     </Loopback>
74    </interface>
75   </native>
76  </config>
77  """
78  netconf_reply = m.edit_config(target="running", config=netconf_loopback)
79  print(xml.dom.minidom.parseString(netconf_reply.xml).toprettyxml())
80
81  #------------------------------------------------------------------------
```

Ejecutando:

```
hernet">
                                        <auto>true</auto>
                                </negotiation>
                        </GigabitEthernet>
                        <Loopback>
                                <name>1</name>
                                <description>My first NETCONF loopback</description>
                                <ip>
                                        <address>
                                                <primary>
                                                        <address>10.1.1.1</address>
                                                        <mask>255.255.255.0</mask>
                                                </primary>
```

```
        </data>
</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:d772773a-487a-47fb-b734-793b874871c6">
        <ok/>
</rpc-reply>

devasc@labvm:~/labs/devnet-src/netconf$
```

En CSR1kv comprobamos que se ha creado la nueva interfaz de bucle invertido.

CSR1kv# show ip interface brief

```
NEWHOSTNAME>
CSR1kv>en
CSR1kv#
CSR1kv#show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.103  YES DHCP   up                    up
Loopback1              10.1.1.1        YES other  up                    up
CSR1kv#_
```


Paso 3: Intente crear una nueva interfaz de bucle invertido con la misma dirección IPv4.

a) Cree una nueva variable llamada **netconf_newloop**. Mantendrá una configuración que crea una nueva interfaz de loopback2 pero con la misma dirección IPv4 que en el bucle de retorno **1:10.1.1.1 /24**. En la CLI del router, esto crearía un error debido al intento de asignar una dirección IP duplicada a una interfaz.

```
ncclient-netconf.py
netconf > ncclient-netconf.py > ...
81  #------------------------------------------------------------------------
82  netconf_newloop = """
83  <config>
84   <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
85    <interface>
86     <Loopback>
87      <name>2</name>
88      <description>My second NETCONF loopback</description>
89      <ip>
90       <address>
91        <primary>
92         <address>10.1.1.1</address>
93         <mask>255.255.255.0</mask>
94        </primary>
95       </address>
96      </ip>
97     </Loopback>
98    </interface>
99   </native>
100 </config>
101 """
102 netconf_reply = m.edit_config(target="running", config=netconf_newloop)
```

Guardamos y ejecutamos el programa. **Debemos obtener una salida de error similar a la siguiente con el mensaje RPC Error Device rechazó uno o más comandos.**

```
<?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:304d5d86-0f17-4b0f-a497-a3df8cc7c676">
        <ok/>
</rpc-reply>

Traceback (most recent call last):
  File "ncclient-netconf.py", line 102, in <module>
    netconf_reply = m.edit_config(target="running", config=netconf_newloop)
  File "/home/devasc/.local/lib/python3.8/site-packages/ncclient/manager.py", line 231, in execute
    return cls(self._session,
  File "/home/devasc/.local/lib/python3.8/site-packages/ncclient/operations/edit.py", line 69, in re
quest
    return self._request(node)
  File "/home/devasc/.local/lib/python3.8/site-packages/ncclient/operations/rpc.py", line 348, in _r
equest
    raise self._reply.error
ncclient.operations.rpc.RPCError: inconsistent value: Device refused one or more commands
devasc@labvm:~/labs/devnet-src/netconf$
```

NETCONF no aplicará ninguna de las configuraciones que se envíen si se rechazan uno o más comandos. Para verificar esto, ingresamos el comando **show ip interface brief** en el R1. Observe que la nueva interfaz no se ha creado.

```
CSR1kv>
CSR1kv>en
CSR1kv#show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.103  YES DHCP   up                    up
Loopback1              10.1.1.1        YES other  up                    up
CSR1kv#
```

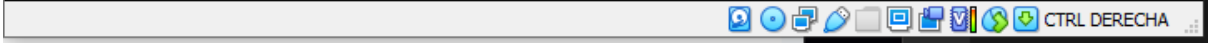
PARTE 6: Desafío: Modificar el programa utilizado en este laboratorio.

- Podemos eliminar el loopback que creamos con la operación delete.

```
105  #---------------Parte_6_DESAFIO----------------------------
106  #como creamos el loooback , podemos eliminarl
107  netconf_delete_loopback = """
108  <config>
109      <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
110          <interface>
111              <Loopback operation="delete">
112                  <name>1</name>
113              </Loopback>
114          </interface>
115      </native>
116  </config>
117  """
118  netconf_reply = m.edit_config(target="running", config=netconf_delete_loopback)
119
```

Ejcutamos: Ya no esta el loopback.

```
CSR1kv#
CSR1kv#show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.103  YES DHCP   up                    up
CSR1kv#
```



RESUMEN:

1° parte, verificamos la conectividad, y garantizamos que pudieran comunicarse entre sí. CLIENTE y SERVIDOR(CSR1kv)

2° parte, usamos una sesión de NETCONF para recopilar información. Esto implico establecer una conexión NETCONF con un dispositivo y enviar solicitudes para obtener datos operativos o de configuración.

3° parte, hacemos uso de ncclient, que es una biblioteca en Python, para poder conectarnos a NETCONF. Exploramos los conceptos básicos de la biblioteca y cómo establecer una conexión y enviar solicitudes NETCONF usando ncclient.

4° parte, hacemos uso de ncclient para recuperar la configuración de un dispositivo. Esto incluye enviar una solicitud NETCONF para obtener la configuración actual del dispositivo y procesar la respuesta para extraer los datos necesarios.

5° parte, hacemos uso de ncclient para configurar un dispositivo. Se mostró cómo enviar solicitudes NETCONF para realizar cambios en la configuración del dispositivo, como agregar interfaces.

CONCLUSION:

El servidor, es nuestro router virtual CSR1kv este permite que los clientes se conecten a él, nosotros como clientes, podemos enviar comandos al router virtual CSR1kv para realizar configuraciones, enviar solicitudes, poder administrar y configurar el router de forma remota. Para ello usaremos la biblioteca ncclient, para poder establecer sesiones NETCONF con dispositivos de red, podemos recuperar información de configuración, y realizar cambios en la configuración de los dispositivos.