Cisco Networking Academy® Mind Wide Open™

Práctica de laboratorio 11: Utilice NETCONF para acceder a un dispositivo IOS XE

Versión en inglés:

https://itexamanswers.net/8-3-6-lab-use-netconf-to-access-an-ios-xe-device-answers.html

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.

Aspectos básicos/Situación

El Protocolo de configuración de red (NETCONF), definido en RFC 4741 y 6241, utiliza modelos de datos YANG para comunicarse con varios dispositivos de la red.

YANG es un lenguaje de modelado de datos. Este lenguaje define los datos que se envían a través de protocolos de administración de red, como NETCONF. Cuando se utiliza NETCONF para acceder a un dispositivo IOS XE, los datos se devuelven en formato XML.

En este laboratorio, utilizará un cliente NETCONF, ncclient, que es un módulo de Python para secuencias de comandos del lado del cliente. Usará ncclient para verificar que NETCONF está configurado, recuperar una configuración de dispositivo y modificar una configuración de dispositivo.

Recursos necesarios.

- Una computadora con el sistema operativo de su elección.
- Virtual Box o VMWare.
- Máquina virtual DEVASC.
- CSR1kv Máquina Virtual.

Instrucciones.

Parte 1: Iniciar las máquinas virtuales y verificar la conectividad.

En esta parte, inicie las dos máquinas virtuales y verifique la conectividad. A continuación, establecerá una conexión segura de Shell (SSH).

Paso 1: Iniciar las máquinas virtuales.

Si aún no ha completado el **Laboratorio - Instalar el entorno de laboratorio de máquina virtual** y el **laboratorio - Instalar la máquina virtual CSR1kV**, hágalo ahora. Si ya ha completado estos laboratorios, inicie la máquina virtual DEVASC y la CSR1000v ahora.

Paso 2: Verifique la conectividad entre las VM.

a. En la máquina virtual CSR1kv, presione **Entrar** para obtener un símbolo del sistema y, a continuación, use **show ip interface brief** para verificar que la dirección IPv4 es 192.168.56.101. Si la dirección es diferente, anótela.

b. Abra una terminal en VS Code en la máquina virtual DEVASC.

c. Haga ping al CSR1kv para verificar la conectividad. Ya debería haber hecho esto anteriormente en los laboratorios de instalación. Si no puede hacer ping, vuelva a visitar los laboratorios enumerados anteriormente en la Parte 1a.

```
devasc @labvm: ~$ ping -c 5 192.168.56.101
PING 192.168.56.101 (192.168.56.101) 56(84) bytes of data.
64 bytes de 192.168.56.101: icmp_seq=1 ttl=254 tiempo=1.37 ms
64 bytes de 192.168.56.101: icmp_seq=2 ttl=254 tiempo=1.15 ms
64 bytes de 192.168.56.101: icmp_seq=3 ttl=254 tiempo=0.981
64 bytes de 192.168.56.101: icmp_seq=4 ttl=254 tiempo=1.01 ms
64 bytes de 192.168.56.101: icmp_seq=5 ttl=254 tiempo=1.14 ms

— 192.168.56.101 estadísticas de ping —
5 paquetes transmitidos, 5 recibidos, 0% de pérdida de paquetes, tiempo 4006ms
rtt min/avg/max/mdev = 0.981/1.130/1.365/0.135 ms
devasc @labvm: ~$
```

Paso 3: Verificar la conectividad SSH a la máquina virtual CSR1kV.

a. En el terminal para la máquina virtual DEVASC, conecte SSH a la máquina virtual CSR1kV con el siguiente comando:

```
devasc @ labvm: ~ $ ssh cisco@192.168.56.101
```

Nota: La primera vez que SSH a CSR1kV, su máquina virtual DEVASC le advierte sobre la autenticidad del CSR1kV. Debido a que confía en CSR1kV, responda sí al prompt.

```
No se puede establecer la autenticidad del host '192.168.56.101
(192.168.56.101)'.
La huella digital de la clave RSA es
Sha256:Hyv9K5Biw7PFixeocdo/LTQS3EFZKBujDipo34VXduy.
¿Está seguro de que desea continuar con la conexión  (yes/no/[fingerprint])?
si
Warning: Permanently added '192.168.56.101' (RSA) to the list of known hosts.
```

b. ¡Entra en **cisco123!** como contraseña y ahora debería estar en el símbolo del sistema EXEC privilegiado para CSR1kV.

```
Contraseña:


CSR1kv#
```

c. Deje abierta la sesión SSH para la siguiente Parte.

Parte 2: Usar 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.

a. Es posible que NETCONF ya se esté ejecutando si otro estudiante lo habilitó, o si una versión posterior de IOS lo habilita de forma predeterminada. Desde su sesión SSH con CSR1kV, utilice el comando **show platform software yang-management process** para ver si se está ejecutando el daemon SSH de NETCONF (**ncsshd**).

```
CSR1kv# show platform software yang-management process
confd : Running
nesd: Running
syncfd: Running
ncsshd: Running
dmiauthd: Running
nginx: Running
ndbmand: Running
pubd: Running

CSR1kv#
```

b. Si NETCONF no se está ejecutando, como se muestra en la salida anterior, ingrese el comando de configuración global **netconf-yang**.

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

c. Escriba **exit** para cerrar la sesión SSH.

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.

Nota: Para obtener más información sobre estas opciones, explore las páginas del manual de SSH (**man ssh**).

a. Ingrese el siguiente comando en una ventana de terminal. Puede usar la flecha arriba para recuperar el último comando SSH y simplemente agregar las opciones **-p** y **-s** como se muestra. Entonces, ingrese ¡**cisco123!** como contraseña.

```
devasc @labvm: ~$ ssh cisco @192 .168.56.101 -p 830 -s netconf
cisco@192.168.56.101's password:
```

b. El CSR1kv 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**]]>]]>**.

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
<capabilities>
<capability>urn:ietf:params:netconf:base:1.0 </capability>
<capability>urn:ietf:params:netconf:base:1.1 </capability>
<capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
<capability>urn:ietf:params:netconf:capacidad:xpath:1.0 </capability>
<capability>urn:ietf:params:netconf:capacidad:validar:1.0 </capability>
```

```
<capability>urn:ietf:params:netconf:capacidad:validar:1.1 </capability>
(output omitted)
      </capability>
</capabilities>
<session-id>20</session-id></hello>]]>]]>
```

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.

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

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
 <capabilities>
   <capability>urn:ietf:params:netconf:base:1.0 </capability>
 </capabilities>
</hello>
]]>]]>
```

b. Cambie a la máquina virtual CSR1kv y utilice el comando **show netconf-yang sessions** para verificar que se ha iniciado una sesión NETCONF. Si la pantalla CSR1kV VM está oscura, presione **Entrar** para activarla.

```
CSr1kv> es
CSrk1v# show netconf-yang sesiones
R: Bloqueo global en el almacén de datos en ejecución
C: Bloqueo global en el almacén de datos candidato
S: Bloqueo global en el almacén de datos de inicio

Número de sesiones: 1

sesion-id transport nombre de usuario source-host global-lock
-------------------------------------------------------------------------------------------------------------
20 netconf-ssh cisco 192.168.56.1 Ninguno

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>
]]>]]>
```

b. Recuerde que XML no requiere sangría o espacio en blanco. Por lo tanto, el CSR1kv devolverá una cadena larga de datos 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><ty
pe 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>]]>]]>
```

c. Copie el XML devuelto, pero no incluya los caracteres finales «]]>]]>». Estos caracteres no forman parte del XML devuelto por el router.

d. Busque en Internet “prettify XML”. Encuentre un sitio adecuado y utilice su herramienta para transformar su XML en un formato más legible, como el siguiente:

```
<?xml version="1.0"?>
<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. Después de unos segundos, volverá al símbolo del terminal. Vuelva al símbolo del sistema CSR1kv y muestre las sesiones netconf abiertas. Verá que la sesión ha sido cerrada.

```
CSR1kv# show netconf-yang sesiones
No hay sesiones activas

CSR1kv#
```

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.

En un terminal de la máquina virtual DEVASC, ingrese el comando **pip3 list —format=columns** para ver todos los módulos de Python instalados actualmente. Conduzca la salida a **más**. Su salida puede diferir de la siguiente. Pero debería ver **ncclient** en la lista, como se muestra. Si no es así, utilice el comando **pip3 install ncclient** para instalarlo.

```
devasc @labvm: ~$ pip3 list —format=columns | more
Versión del paquete
------------------------------ ---------------
ansible 2.9.6
apache-libcloud 2.8.0
appdirs 1.4.3
argcomplete 1.8.1
astroide 2.3.3
(output omitted)
ncclient 0.6.7
netaddr 0.7.19
netifaces 0.10.4
netmiko 3.1.0
ntlm-auth 1.1.0
oauthlib 3.1.0
(output omitted)
xmltodict 0.12.0
zipp 1.0.0
devasc @labvm: ~$
```

Paso 2: Cree un script para usar ncclient para conectarse al servicio NETCONF.

El módulo ncclient proporciona una clase de **administrador** 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.

a. En el código VS, haga clic en **Archivo > Abrir carpeta...** y navegue al directorio **devnet-src**. Haga clic en **Aceptar**.

b. Abra una ventana de terminal en VS Code: **Terminal> Nueva Terminal.**

c. Cree un subdirectorio llamado **netconf** en el directorio **/devnet-src**.

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

d. En el panel **EXPLORER** en **DEVNET-SRC**, haga clic con el botón derecho en el directorio **netconf** y elija **Nuevo archivo**.

e. Asigne un nombre al archivo **ncclient-netconf.py**.

f. En su archivo de script, importe la clase de administrador desde el módulo **ncclient**. A continuación, cree 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.

```
desde el administrador de importación ncclient

m = manager.connect (
    host="192.168.56.101",
    port=830,
    username="cisco»,
    password="cisco123!",
    hostKey_verify=false
    ).
```

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

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

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

h. Puede verificar que el CSR1kV aceptó la solicitud de una sesión NETCONF. Debe haber un mensaje%DMI-5-auth_passed **syslog** en la máquina virtual CSR1kv. Si la pantalla es negra, presione **Entrar** para activar el router. El mensaje syslog se puede ver encima del banner.

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**.

a. Utilice un bucle **for** y una función de **impresión** para mostrar las capacidades del dispositivo:

```
print (» #Supported Capacidades (modelos YANG):»)
para la capacidad en m.server_capabilities:
```

```
    imprimir (capacidad)
```

b. Guarde y ejecute el programa. 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$ python3 ncclient-netconf.py
Capacidades #Supported (modelos YANG):
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
<output omitted>
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=2011-06-01

        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.

a. Si desea omitir la salida de las capacidades de visualización (más de 400 líneas), comente el bloque de instrucciones que imprimen las capacidades, como se muestra en el siguiente ejemplo:

```
'''
print (» #Supported Capacidades (modelos YANG):»)
para la capacidad en m.server_capabilities:
    print(capability)
'''
```

b. Puede utilizar el método **get_config ()** del objeto de sesión **m** NETCONF para recuperar la configuración del CSR1kV. El método **get_config ()** espera un parámetro de cadena de origen que especifique el almacén de datos NETCONF de origen. Utilice una función de impresión para mostrar los resultados. 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**.

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

c. Guarde y ejecute su programa. La salida será más de 100 líneas, por lo que IDLE puede comprimirlas. Haga doble click en el mensaje de **texto Exprimido** en la ventana del shell IDLE para expandir la salida.

```
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:3f31bedc-5671-47ca-9781-4d3d7aadae24"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"> <data> <native
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native"> <version>16.9 </version><boot-
start-marker/> <boot-end-marker/> <banner> <motd> <banner>
(output omitted)
```

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

d. Observe que el XML devuelto no tiene formato. Puede copiarlo en el mismo sitio que encontró anteriormente para envolver el XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<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:3f31bedc-5671-
47ca-9781-4d3d7aadae24">
  <data>
    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
      <version>16.9 </version>
      <boot-start-marker/>
      <boot-end-marker/>
      <banner>
        <motd>
          <banner>^C </banner>
        </motd>
      </banner>
      <service>
        <timestamps>
          <debug>
            <datetime>
              <msec/>
            </datetime>
          </debug>
          <log>
            <datetime>
              <msec/>
            </datetime>
          </log>
        </timestamps>
      </service>
      <platform>
        <console xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-platform">
          <output>Virtual</output>
        </console>
      </platform>
      <hostname>CSR1kV </hostname>
(output omitted)
```

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 ()**.

a. Al principio de su script, agregue una instrucción para importar el módulo **xml.dom.minidom**.

```
importar xml.dom.minidom
```

b. Reemplace la impresión simple de la función de **impresión (netconf_reply)** por una versión que imprima la salida XML prettified.

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

c. Guarde y ejecute su programa. XML se muestra en un formato más legible.

```
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:3a5f6abc-76b4-
436d-9e9a-7758091c28b7">
        <data>
                <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
                        <version>16.9 </version>
                        <boot-start-marker/>
                        <boot-end-marker/>
                        <banner>
                                <motd>
                                        <banner>^C </banner>
                                </motd>
                        </banner>
(output omitted)
        </data>
</rpc-reply>

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

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

```
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 ())
```

b. Guarde y ejecute su programa. El inicio de la salida es el mismo, como se muestra a continuación. Sin embargo, <native> esta vez solo se muestra el elemento XML. Anteriormente, se mostraban todos los modelos YANG disponibles en el CSR1kV.

Filtrar los datos recuperados para mostrar solo el módulo nativo de YANG reduce significativamente la salida. Esto se debe a que el módulo YANG nativo solo incluye un subconjunto de todos los modelos IOX XE YANG de Cisco.

```
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:4da5b736-1d33-
47c3-8e3c-349414be0958">
        <data>
                <nativo xmlns=» http://cisco.com/ns/yang/Cisco-IOS-XE-native «
```

```
                        <version>16.9 </version>
                        <boot-start-marker/>
                        <boot-end-marker/>
                        <banner>
                                <motd>
                                        <banner>^C </banner>
                                </motd>
                        </banner>
                        <service>
                                <timestamps>
                                        <debug>
                                                <datetime>
                                                        <msec/>
                                                </datetime>
                                        </debug>
                                        <log>
                                                <datetime>
                                                        <msec/>
                                                </datetime>
                                        </log>
                                </timestamps>
                        </service>
                        <platform>
                                <console xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-
platform">
                                        <output>Virtual</output>
                                </console>
                        </platform>
                        <hostname>CSR1kV </hostname>
(output omitted)
                </native>
        </data>
</rpc-reply>

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

Parte 5: Usar ncclient para configurar un dispositivo.

En esta parte, usará **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.

Para actualizar una configuración existente en la configuración para CSR1kV, puede extraer la ubicación de configuración de la que fue recuperada anteriormente. Para este paso, establecerá una variable para cambiar el <hostname> valor.

```
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:4da5b736-1d33-
47c3-8e3c-349414be0958">
```

```
        <data>
                <nativo xmlns=» http://cisco.com/ns/yang/Cisco-IOS-XE-native «
(output omitted)
                        <hostname>CSR1kV </hostname>
(output omitted)
```

a. Anteriormente, definía una <gilter> variable. 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.

```
netconf_hostname = «"»
<config>
  <nativo xmlns=» http://cisco.com/ns/yang/Cisco-IOS-XE-native «
     <hostname>NOMBRE NOMBRE </hostname>
  </native>
</config>
"""
```

b. Utilice la función **edit_config ()** del objeto de sesión **m** NETCONF para enviar la configuración y almacenar los resultados en la variable **netconf_reply** para que puedan imprimirse. Los parámetros para la función **edit_config ()** son los siguientes:

- **target** : El almacén de datos NETCONF objetivo que se actualizará.
- **config**: La modificación de configuración que se va a enviar.

```
netconf_reply = m.edit_config (target="running», config=netconf_hostname)
```

c. La función **edit_config ()** devuelve un mensaje de respuesta XML RPC que <ok/> indica que el cambio se ha aplicado correctamente. Repita la instrucción de impresión anterior para mostrar los resultados.

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

d. Guarde y ejecute su programa. Debe obtener una salida similar a la salida que se muestra a continuación. También puede verificar que el nombre de host ha cambiado cambiando a la máquina virtual CSR1kV.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
(output omitted)
<?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:e304b225-7951-
4029-afd5-59e8e7edbaa0">
        <ok/>
</rpc-reply>

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

e. Edite su script para cambiar el nombre de host de nuevo a CSR1kV. Guarde y ejecute su programa. También puede comentar el código del paso anterior si desea evitar cambiar el nombre de host nuevamente.

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

a. Cree una nueva variable <config> para contener la configuración de una nueva interfaz de bucle invertido. Agregue lo siguiente a su script **ncclient_netconf.py**.

Nota: Puede usar **la descripción** que desee. Sin embargo, solo use caracteres alfanuméricos o tendrá que separarlos con la barra invertida (\).

```
netconf_loopback = «"»
<config>
 <nativo xmlns=» http://cisco.com/ns/yang/Cisco-IOS-XE-native «
  <interface>
   <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>
     </address>
    </ip>
   </Loopback>
  </interface>
 </native>
</config>
"""
```

b. Agregue la siguiente función **edit_config ()** a su **ncclient_netconf.py** para enviar la nueva configuración de bucle invertido a R1 y luego imprima los resultados.

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

c. Guarde y ejecute su programa.

Se debería obtener una salida similar a la siguiente:

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
(output omitted)
<?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:98437f47-7a93-
4cac-9b9e-9bc8afc9dfa1">
        <ok/>
</rpc-reply>

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

d. En CSR1kV, compruebe que se ha creado la nueva interfaz de bucle invertido.

```
CSR1kv>es
CSR1kv# show ip interface brief
¿Interface IP-Address OK? Método de protocolo de estado
GigabitEthernet1 192.168.56.101 YES DHCP up up
Loopback1 10.1.1.1 SÍ otras arriba
CSR1kv# show run | sección interfaz Loopback1
interface Loopback1
```

```
 descripción Mi primer bucle de retorno NETCONF
 ip address 10.1.1.1 255.255.255.0
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 loopback 2 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.

```
netconf_newloop = «"»
<config>
 <nativo xmlns=» http://cisco.com/ns/yang/Cisco-IOS-XE-native «
  <interface>
   <Loopback>
    <name>2 </name>
    <description>My second NETCONF loopback</description>
    <ip>
     <address>
      <primary>
       <address>10.1.1.1</address>
       <mask>255.255.255.0 </mask>
      </primary>
     </address>
    </ip>
   </Loopback>
  </interface>
 </native>
</config>
"""
```

b. Agregue la siguiente función **edit_config ()** a su **ncclient_netconf.py** para enviar la nueva configuración de bucle invertido al CSR1kv. No necesita una declaración de impresión para este paso.

```
netconf_reply = m.edit_config (target="running», config=netconf_newloop)
```

c. Guarde y ejecute el programa. Debe obtener una salida de error similar a la siguiente con el mensaje **RPC Error Device rechazó uno o más comandos**.

```
devasc@labvm:~/labs/devnet-src/netconf$ python3 ncclient-netconf.py
Traceback (última llamada más reciente):
  File "ncclient-netconf.py", line 80, 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 request
    return self._request(node)
  File "/home/devasc/.local/lib/python3.8/site-packages/ncclient/operations/rpc.py",
line 348, in _request
```

```
    raise self._reply.error
ncclient.operations.rpc.RPCError: inconsistent value: Device refused one or more
commands
devasc @labvm: ~/labs/devnet-src/netconf$
```

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

```
CSR1kv# show ip interface brief
¿Interface IP-Address OK? Método de protocolo de estado
GigabitEthernet1 192.168.56.101 YES DHCP up up
Loopback1 10.1.1.1 YES other up up
```

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

El siguiente paso es el programa completo que se creó en este laboratorio sin ningún código comentado para que pueda ejecutar el script sin error. Su guion puede verse diferente. Practique sus habilidades de Python modificando el programa para enviar diferentes comandos de verificación y configuración.

```
desde el administrador de importación ncclient
importar xml.dom.minidom

m = manager.connect (
    host="192.168.56.101",
    port=830,
    username="cisco»,
    password="cisco123!",
    hostKey_verify=false
    ).

print (» #Supported Capacidades (modelos YANG):»)
para la capacidad en m.server_capabilities:
    print(capability)

netconf_reply = m.get_config(source="running»)
print (Xml.dom.minidom.parseString (netconf_reply.xml) .toprettyxml ())

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 ())

netconf_hostname = «"»
<config>
  <nativo xmlns=» http://cisco.com/ns/yang/Cisco-IOS-XE-native «
```

```
      <hostname>CSR1kV </hostname>
  </native>
</config>
"""
netconf_reply = m.edit_config (target="running», config=netconf_hostname)
print (Xml.dom.minidom.parseString (netconf_reply.xml) .toprettyxml ())

netconf_loopback = «"»
<config>
 <nativo xmlns=» http://cisco.com/ns/yang/Cisco-IOS-XE-native «
  <interface>
   <Loopback>
    <name>1 </name>
    <description>Mi bucle invertido NETCONF </description>
    <ip>
     <address>
      <primary
       <address>10.1.1.1</address>
       <mask>255.255.255.0 </mask>
      </primary>
     </address>
    </ip>
   </Loopback>
  </interface>
 </native>
</config>
"""
netconf_reply = m.edit_config (target="running», config=netconf_loopback)
print (Xml.dom.minidom.parseString (netconf_reply.xml) .toprettyxml ())

netconf_newloop = «"»
<config>
 <nativo xmlns=» http://cisco.com/ns/yang/Cisco-IOS-XE-native «
  <interface>
   <Loopback>
    <name>2 </name>
    <description>My second NETCONF loopback</description>
    <ip>
     <address>
      <primary
       <address>10.1.1.1</address>
       <mask>255.255.255.0 </mask>
      </primary>
     </address>
    </ip>
```

```
    </Loopback>
   </interface>
  </native>
</config>
"""
netconf_reply = m.edit_config (target="running», config=netconf_newloop)
```