Sigla Asignatura	***DRY7122***	**Nombre de la Asignatura**	**PROGRAMACIÓN Y REDES VIRTUALIZADAS (SDN-NFV)**	**Tiempo**	***4 horas***
Experiencia de Aprendizaje N° 3	***Infraestructura automatizada y plataforma de desarrollo***				
Actividad N° 12	**Explorara los modelos de YANG - NETCONF y RESTCONF a un dispositivo IOS-XE**				
Nombre del Recurso Didáctico	***3.3.2 Modelo de YANG - NETCONF y RESTCONF***				

1. Aprendizajes e indicadores de logro

RESULTADOS DE APRENDIZAJE	Indicadores de logro
RA3. Aplica soluciones de virtualización con el fin de simplificar la administración e implementación para los requerimientos de la organización.	3.2 Compara métodos de implementación y prueba de software en entornos de automatización y virtualizado.

2. Descripción General Actividad:

1. Esta actividad tiene **carácter práctico**, es decir: es para visualizar lo que aprendes, en la directa medida que tú docente de asignatura te va retroalimentando constantemente, tanto a nivel individual como colectivo (equipo de trabajo).
2. Realizar las configuraciones solicitadas, y documentar los errores detectados.
3. Una vez finalizada la actividad, deberá ser enviada a través del AVA, según las instrucciones del docente.

3. Descripción Específica Actividad:

Esta actividad se compone de 3 laboratorios cubriendo los conceptos de esta clase, en el caso de que los alumnos no puedan por tiempo realizar cada uno en clase, se recomienda al docente, dejar como encargo los talleres faltantes y luego retroalimentar sobre los resultados obtenidos.

LABORATORIO 1

Explorar los modelos YANG

Objetivos

Parte 1: Iniciar la máquina virtual de DEVASC.

Parte 2: Explorar un modelo de YANG en GitHub.

Parte 3: Explorar un modelo YANG usando pyang.

Aspectos básicos/Situación

Los modelos YANG definen la estructura exacta, los tipos de datos, la sintaxis y las reglas de validación para el contenido de los mensajes intercambiados entre un dispositivo gestionado y otro sistema que se está comunicando con el dispositivo. Trabajar con archivos usando el lenguaje YANG puede ser un poco abrumador para el nivel de detalles de estos archivos.

En este laboratorio, aprenderá cómo utilizar la herramienta **pyang** de código abierto, para transformar los modelos de datos de YANG a partir de archivos, utilizando el lenguaje YANG en un formato mucho más fácil de leer. Usando la transformación de vista "árbol", identificará cuáles son los elementos clave del modelo YANG de interfaces ietf.

Recursos necesarios

- Un computador con el sistema operativo de su elección.
- VirtualBox o VMware.
- Máquina virtual DEVASC.

Instrucciones

Iniciar la máquina virtual de DEVASC.

Si aún no ha completado el **laboratorio - Instalar el DEVASC-LAB**, hágalo ahora. Si se ha completado ya, inicie la máquina virtual DEVASC.

Explorar un modelo de YANG en GitHub.

En esta parte, instalará el módulo pyang en su máquina virtual DEVASC y explorará como transforma los archivos YANG. Pyang simplifica el trabajo con archivos YANG. El módulo viene con una línea ejecutable de comandos pyang que transforma los archivos YANG en un formato más legible para el ser humano.

Explorar los modelos de Cisco IOS XE YANG en el repositorio de GitHub.

a. Abra Chromium y busque https://github.com/YangModels/yang.

b. En la rama **maestra**, vaya a los modelos de YANG para Cisco IOS XE (versión 16.9.3), haciendo clic en los siguientes directorios: **proveedor > cisco > xe > 1693**.

c. Desplácese por debajo de todos los modelos Cisco YANG y encuentre dónde comienzan los modelos IETF. Busque **ietf-interfaces.yang**.

d. Haga clic en **ietf-interfaces.yang** y desplácese por todos los nodos de contenedores, nodos de hoja y nodos de lista. Si está familiarizado con la salida del comando Mostrar interfaces IOS (show interfaces IOS), entonces debe reconocer algunos o todos los nodos. Por ejemplo, alrededor de la línea 221 verá la hoja habilitada.

```
        hoja habilitada {
          Tipo boolean;
          por defecto «true»;
          descripción
            «Esta hoja contiene el estado configurado y deseado del
             E&M.
             Los sistemas que implementan el IF-MIB utilizan el valor de este
             en el almacén de datos 'en ejecución' para establecer
             if-mib.ifAdminStatus a 'arriba' o 'abajo' después de una ifEntry
             se ha iniciado, como se describe en RFC 2863.
             Los cambios en esta hoja en el almacén de datos 'en ejecución'
son
             reflejado en ifAdminStatus, pero si ifAdminStatus es
             cambiado sobre SNMP, esta hoja no se va a ver afectada.«;
          referencia
```

```
            «RFC 2863: MIB del Grupo de Interfaces - IFAdminStatus»;
        }
```

Copiar el modelo ietf-interfaces.yang en una carpeta de su Máquina Virtual.

a. Abra VS Code.

b. Haga clic en **Archivo > Abrir carpeta...** y navegue al directorio **devnet-src**.

c. Haga clic en **Aceptar**.

d. Abra una ventana de terminal en VS Código: **Terminal > Nueva Terminal**.

e. Cree un subdirectorio llamado **pyang** en el directorio **/devnet-src**.

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

f. Vuelva a la pestaña de Chromium donde el modelo **ietf-interfaces.yang** sigue abierto. Desplácese hacia atrás, a la parte superior, si es necesario y haga clic en Raw para mostrar solo los datos del modelo YANG.

g. Seleccione y copie la dirección URL.

h. En la terminal, vaya a la carpeta pyang.

i. Utilice **wget** para guardar el archivo ietf-interfaces.yang sin procesar.

```
devasc @labvm: ~/labs/devnet-src/pyang$ wget
https://raw.githubusercontent.com/YangModels/yang/master/vendor/cisco/xe/1693/ietf-interfaces.yang
--2020-06-22 20:42:20--
https://raw.githubusercontent.com/YangModels/yang/master/vendor/cisco/xe/1693/ietf-interfaces.yang
Resolviendo raw.githubusercontent.com (raw.githubusercontent.com)...
151.101.0.133, 151.101.192.133, 151.101.128.133, ...
Conectando a raw.githubusercontent.com (raw.githubusercontent.com)
|151.101.0.133|:443... conectado.
HTTP request sent, awaiting response... 200 OK
Longitud: 24248 (24K) [texto/liso]
Guardar en: 'ietf-interfaces.yang'

ietf-interface 100% 23.68K --.-Kb/s en 0.05s

2020-06-22 20:42:21 (439 kB/s) - 'ietf-interfaces.yang' guardado
[24248/24248]

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

Ahora tiene una versión local del modelo **ietf-interfaces.yang** que puede manipular con **pyang**.

Explorar un modelo de YANG usando pyang.

En esta parte, usted instalará el módulo **pyang** en su máquina virtual DEVASC y explorará cómo transforma el modelo YANG que copió de GitHub. Pyang simplifica el trabajo con archivos YANG. El módulo viene con un ejecutable de línea de comandos **pyang** que transforma los archivos YANG en un formato más legible por humanos.

Verificar que pyang esté instalado y actualizado.

a. En un código VS, abra la ventana terminal

b. Verifique que pyang ya esté instalado con el comando **pyang -v**. Su número de versión puede ser diferente al que se muestra aquí. También se puede

```
devasc @labvm: ~/labs/devnet-src$ pyang -v
pyang 2.2.1
devasc@labvm:~/labs/devnet-src$
```

c. (Opcional) Usted puede verificar que tiene las últimas actualizaciones de pyang usando el siguiente comando **pip3**. Cualquier actualización después de escribir este laboratorio se descargará y se instalará.

```
devasc @labvm: ~/labs/devnet-src$ pip3 install pyang —upgrade
Requisito ya actualizado: pyang en ./.local/lib/python3.8/site-packages
(2.2.1)
Requisito ya satisfecho, omitiendo la actualización: lxml en
./.local/lib/python3.8/site-packages (de pyang) (4.5.0)
devasc@labvm:~/labs/devnet-src$
```

Transformar el modelo ietf-interfaces.yang.

a. Vaya al directorio **pyang**.

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

b. Ingrese pyang -h | para explorar más las opciones para transformar el modelo YANG. Busque la opción **-f** como se muestra a continuación. Usará la opción de formato de **árbol.**

```
devasc@labvm:~/labs/devnet-src/pyang$ pyang -h | más
Uso: pyang [opciones] [<gilename>...]

Valida el módulo YANG en <gilename> (o stdin), y todas sus dependencias.

Options
  -h, —help Mostrar este mensaje de ayuda y salir
  -v, —version Mostrar el número de versión y salir
<output omitted>
  -f FORMAT, —format=Formato
                        Convertir a FORMAT.  Los formatos admitidos son:
yang, yin,
                        dsdl, jstree, jsonxsl, capacidad, identificadores,
jtox,
                        UML, nombre, Omni, árbol, depende, sample-xml-
esqueleto
<output omitted>
devasc@labvm:~/labs/devnet-src/pyang$
```

c. Transforme el modelo **ietf-interfaces.yang** en un modelo de formato de árbol con el siguiente comando. Observe que la **hoja habilitada** es mucho más fácil de encontrar y leer en este formato.

```
devasc@labvm:~/labs/devnet-src/pyang$ pyang -f árbol ietf-
interfaces.yang
```

```
ietf-interfaces.yang:6: error: módulo «ietf-yang-types» no fue encontrado
en la ruta de búsqueda
módulo: ietf-interfaces
  interfaces +—rw
  | +--rw interface* [Nombre]
  | +—cadena de nombre rw
  | +--rw descrición? cadena
  | +--rw tipo identifiref
  | +—rw habilitado? booleano
  | +—rw link-up-down-trap-enable? enumeración {if-mib}?
  +—ro interfaces-estado
     +--ro interface* [Nombre]
        +—ro cadena de nombre
        +--ro tipo identifiref
        +—ro admin-estado enumeración {if-mib}?
        +—ro enumeración de oper-estado
        +—ro último cambio? yang: fecha y hora
        +—ro if-index int32 {if-mib}?
        +—ro phys-dirección? yang:phys-dirección
        +—ro capa superior-si* interface-state ref
        +—ro capa inferior-si* interface-state ref
        ¿Velocidad de +ro? yang:gauge64
        +--ro stadísticas
           +—ro tiempo de discontinuidad yang: fecha y hora
           +—ro en octetos? yang:counter64
           +—ro in-unicast-pkts? yang:counter64
           +—ro in-broadcast pkts? yang:counter64
           +—ro en multidifusión pkts? yang:counter64
           +—ro en descartes? yang:counter32
           +—ro en errores? yang:counter32
           +—ro in-unknown-protos? yang:counter32
           +—octetos de salida? yang:counter64
           +—ro fuera-unicast-pkts? yang:counter64
           +—ro fuera-broadcast-pkts? yang:counter64
           +—ro fuera-multicast-pkts? yang:counter64
           +—ro fuera-descartes? yang:counter32
           +—ro fuera-errores? yang:counter32
devasc@labvm:~/labs/devnet-src/pyang$
```

LABORATORIO 2

Utilizar NETCONF para acceder a un dispositivo IOS XE

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.

- Un computador 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).

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.

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: ~$
```

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 abierto la sesión SSH para la siguiente Parte.

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.

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.

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>]]>]]>
```

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#
```

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.
<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</desc
ription><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>]]>]]>
```

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>
```

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#
```

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.

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: ~$
```

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.

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$
```

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.

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)
```

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$
```

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$
```

Usar ncclient para configurar un dispositivo.

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

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.

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#
```

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
```

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)
```

LABORATORIO 3

Utilizar RESTCONF para acceder a un dispositivo IOS XE

Objetivos

Parte 1: Armar la red y verificar la conectividad.

Parte 2: Configurar un dispositivo IOS XE para el acceso RESTCONF.

Parte 3: Abrir y configurar Postman.

Parte 4: Usar Postman para enviar solicitudes GET.

Parte 5: Usar Postman para enviar solicitudes PUT.

Parte 6: Usar un script de Python para enviar solicitudes GET.

Parte 7: Usar un script de Python para enviar solicitudes PUT.

Aspectos básicos/Situación

El protocolo RESTCONF proporciona un subconjunto simplificado de características NETCONF sobre una API RESTful. RESTCONF nos permite realizar llamadas API RESTful a un dispositivo IOS XE. Los datos devueltos por la API pueden convertirse a formato XML o JSON. En la primera mitad de este laboratorio, utilizaremos el programa Postman para construir y enviar solicitudes API al servicio RESTCONF que se está ejecutando en el CSR1kV. En la segunda mitad del laboratorio, crearemos scripts de Python para realizar las mismas tareas que el programa Postman.

Recursos necesarios

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

Instrucciones

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

En esta parte, iniciaremos las dos máquinas virtuales del curso y verificaremos la conectividad. A continuación, estableceremos una conexión secure shell (SSH).

Iniciar las Máquinas Virtuales.

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

Verificar la conectividad entre las máquinas virtuales.

a. En la máquina virtual CSR1kv, presione Enter para poder empezar a ingresar comandos, luego use el comando **show ip interface brief** para verificar que la dirección IPv4 es 192.168.56.101. Si la dirección es diferente, anótela.

b. Abra el terminal en la máquina virtual DEVASC.

c. Haga ping hacia la CSR1kv para verificar la conectividad.

Ya debería haber hecho esto en los laboratorios de instalación. Si no puede hacer ping, vuelva a realizar los laboratorios mencionados anteriormente en el Paso 1 de la Parte 1.

```
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 from 192.168.56.101: icmp_seq=1 ttl=254 time=1.37 ms
64 bytes from 192.168.56.101: icmp_seq=2 ttl=254 time=1.15 ms
64 bytes from 192.168.56.101: icmp_seq=3 ttl=254 time=0.981 ms
64 bytes from 192.168.56.101: icmp_seq=4 ttl=254 time=1.01 ms
64 bytes from 192.168.56.101: icmp_seq=5 ttl=254 time=1.14 ms

--- 192.168.56.101 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 0.981/1.130/1.365/0.135 ms
devasc @labvm: ~$
```

Verificar la conectividad SSH a la máquina virtual CSR1kV.

a. En el terminal de la máquina virtual DEVASC, realice una conexión SSH hacia la máquina virtual CSR1kV con el siguiente comando:

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

Nota: La primera vez que realizamos conexión SSH hacia la CSR1kV, la máquina virtual DEVASC nos advierte sobre la autenticidad del CSR1kV. Ya que confía en el CSR1kV, responda "yes" al prompt.

```
The authenticity of host '192.168.56.101 (192.168.56.101)'
can't be established.
RSA key fingerprint is
SHA256:HYv9K5Biw7PFiXeoCDO/LTqs3EfZKBuJdiPo34VXDUY.
Are you sure you want to continue connecting
(yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.56.101' (RSA) to the list
of known hosts.
```

b. Escriba **cisco123!** como contraseña y ahora debería estar en el modo EXEC privilegiado de la línea de comandos del CSR1kv.

```
Password: <cisco123!>


CSR1kv#
```

c. Deje abierto la sesión SSH, para la siguiente Parte.

Configurar un dispositivo IOS XE para el acceso RESTCONF.

En esta parte, configuraremos la máquina virtual CSR1kV para aceptar mensajes RESTCONF. Además, iniciaremos el servicio HTTPS.

Nota: Es posible que los servicios descritos en esta parte ya se estén ejecutando en su máquina virtual. Sin embargo, asegúrese de conocer los comandos para ver los servicios en ejecución y habilitarlos.

Comprobar que se estén ejecutando los daemons de RESTCONF.

RESTCONF ya debería estar en ejecución porque forma parte de la configuración predeterminada proporcionada por NetAcad.

Desde el terminal, puede utilizar el comando **show platform software yang-management process** para ver si se están ejecutando todos los daemons asociados al servicio RESTCONF. El daemon NETCONF también puede estar en ejecución, pero no se utilizará en este laboratorio. Si uno o más de los daemons requeridos no se están ejecutando, continúe con el paso 2.

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

CSR1kv#
```

Nota: El propósito y la función de todos los daemons está más allá del enfoque de este curso.

Habilitar y verificar el servicio RESTCONF.

a. Introduzca el comando de configuración global **restconf** para habilitar el servicio RESTCONF en el CSR1kV.

```
CSR1kv#configure terminal
CSR1kv(config)# restconf
```

b. Compruebe que los daemons RESTCONF necesarios se estén ejecutando ahora. Recuerde que **ncsshd** es el servicio NETCONF, que puede estar ejecutándose en su dispositivo. No lo necesitamos para este laboratorio. Sin embargo, necesita **nginx**, que es el servidor HTTPS. El daemon nginx le permitirá realizar llamadas API REST al servicio RESTCONF.

```
CSR1kv(config)# exit
CSR1kv# show platform software yang-management process
confd : Running
nesd: Running
syncfd: Running
ncsshd : Not Running
dmiauthd: Running
nginx : Not Running
ndbmand: Running
pubd: Running
```

Habilitar y verificar el servicio HTTPS.

a. Introduzca los siguientes comandos de configuración global para habilitar el servidor HTTPS y especificar que la autenticación del servidor debe utilizar la base de datos local.

```
CSR1kv# configure terminal
CSR1kv(config)# ip http secure-server
CSR1kv(config)# ip http authentication local
```

b. Compruebe que el servidor HTTPS (nginx) se esté ejecutando ahora.

```
CSR1kv(config)# exit
CSR1kv# show platform software yang-management process
confd : Running
nesd: Running
syncfd: Running
ncsshd : Not Running
dmiauthd: Running
nginx: Running
ndbmand: Running
pubd: Running
```

Abrir y configurar Postman.

En esta parte, abriremos Postman, deshabilitaremos los certificados SSL y exploraremos la interfaz de usuario.

Abrir Postman.

a. En la **VM DEVASC**, abra la aplicación Postman.

b. Si es la primera vez que abre Postman, es posible que le pida que cree una cuenta o inicie sesión. En la parte inferior de la ventana, también puede hacer clic en "Skip" (omitir) para omitir el inicio de sesión. No es necesario iniciar sesión para utilizar esta aplicación.

Deshabilitar la verificación de certificación SSL.

De forma predeterminada, Postman tiene activada la verificación de certificación SSL. No usaremos certificados SSL con el CSR1Kv, por lo tanto, debe desactivar esta función.

a. Haga clic en **Archivo > Configuración**.

b. En la pestaña **General**, establezca **SSL certificate verification** en **OFF**.

c. Cierre el cuadro de diálogo **Configuración**.

Utilizar Postman para enviar solicitudes GET.

En esta parte, utilizaremos Postman para enviar una solicitud GET al CSR1kV para verificar si podemos conectarnos al servicio RESTCONF.

Explorar la interfaz de usuario de Postman.

a. En el centro, verá el **Launchpad**. Puede explorar esta zona si lo desea.

b. Haga clic en el signo más (+) situado junto a la pestaña **Launchpad** para abrir una **Solicitud GET sin título**. Esta es la interfaz donde realizaremos todo el trabajo en este laboratorio.

Introduzca la dirección URL del CSR1kV.

a. El tipo de solicitud ya está establecido en GET. Deje el tipo de solicitud establecido en GET.

b. En el campo "Enter request URL" (Introducir URL de solicitud), escriba la URL que se utilizará para acceder al servicio RESTCONF que se está ejecutando en el CSR1kV:

```
https://192.168.56.101/restconf/
```

Introduzca las credenciales de autenticación.

En el campo URL, hay pestañas llamadas **Parámetros**, **Autorización**, **Encabezados**, **Contenido**, **Pre-request Script**, **Test**, y **Configuraciones**. En este laboratorio, utilizaremos **Autorización**, **Encabezados** y **Contenido**.

a. Haga clic en la pestaña **Autorización**.

b. En Tipo, haga clic en la flecha abajo situada junto a "Inherit auth from parent" (Heredar autenticación del administrador) y seleccione **Autenticación básica**.

c. En **Username** y **Password**, introduzca las credenciales de autenticación local para el CSR1kV:

Username: **cisco**

Password: **cisco123!**

d. Haga clic en **Encabezados**. Luego haga clic en **7 hidden** (7 ocultos). Podemos verificar que la Clave de autorización tiene un Valor básico que se utilizará para autenticar la solicitud cuando se envíe al CSR1kV.

Establezca JSON como el tipo de datos para enviar y recibir desde el CSR1kV.

Puede enviar y recibir datos desde el CSR1kv en formato XML o JSON. Para este laboratorio, utilizaremos JSON.

a. En el área **Encabezados**, haga clic en el primer campo **Clave** en blanco y escriba **Content-Type** para el tipo de clave. En el campo **Valor**, escriba **application/yang-data+json**. Esto le dice a Postman que envíe datos JSON al CSR1kV.

b. Debajo de la clave **Content-Type**, agregue otro par de clave/valor. El campo **Clave** es **Accept** y el campo **Valor** es **application/yang-data+json**.

Nota: Puede cambiar "application/yang-data+json" a "application/yang-data+xml" para enviar y recibir datos XML en lugar de datos JSON, si fuera necesario.

Enviar la solicitud de API al CSR1kV.

Postman ahora tiene toda la información que necesita para enviar la solicitud GET. Haga clic en **Enviar**. Debajo de **Encabezados temporales** debería poder ver la siguiente respuesta JSON del CSR1kV. Si no es así, compruebe que haya completado los pasos anteriores en esta parte del laboratorio y que haya configurado correctamente el servicio RESTCONF y HTTPS en la Parte 2.

```
{
    «ietf-restconf:restconf»: {
        "data": {},
        "operations": {},
        "yang-library-version": "2016-06-21"
    }
}
```

Esta respuesta JSON verifica que Postman ahora puede enviar otras solicitudes de API REST al CSR1kV.

Utilizar una solicitud GET para recopilar la información de todas las interfaces del CSR1kv.

a. Ahora que tiene una solicitud GET exitosa, puede utilizarla como plantilla para solicitudes adicionales. En la parte superior de Postman, junto a la pestaña **Launchpad**, haga clic derecho en la pestaña **GET** que acaba de usar y seleccione **Duplicar pestaña**.

b. Utilice el modelo YANG de **ietf-interfaces** para recopilar información de la interfaz. En la URL, agregue **data/ietf-interfaces:interfaces**:

```
https://192.168.56.101/restconf/data/ietf-
interfaces:interfaces
```

c. Haga clic en **Enviar**. Debería poder ver una respuesta JSON del CSR1kv que sea similar a la salida que se muestra a continuación. La salida puede ser diferente dependiendo del router que usted esté utlizando.


```
{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "GigabitEthernet1",
        "description": "VBox",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": true,
        "ietf-ip:ipv4": {},
        "ietf-ip:ipv6": {}
      }
    ]
  }
}
```


Utilizar una solicitud GET para recopilar información de una interfaz específica del CSR1kV.

En este laboratorio, solo la interfaz GigabitEthernet1 está configurada. Para especificar solo esta interfaz, extienda la URL para solicitar información para esta interfaz solamente.

a. Duplique su última solicitud GET.

b. Agregue el parámetro **interface=** para especificar una interfaz y escriba el nombre de la misma.

```
https://192.168.56.101/restconf/data/ietf-
interfaces:interfaces/interface=GigabitEthernet1
```

Nota: Si solicita información de interfaz desde un dispositivo Cisco diferente con nombres que utilizan barras diagonales, como Gigabitethernet0/0/1, utilice el código HTML **%2F** para las barras diagonales en el nombre de la interfaz. De tal manera que, **0/0/1** se convierte en **0%2F0%2F1**.

c. Haga clic en **Enviar**. Debería poder ver una respuesta JSON del CSR1kv que sea similar a la salida que se muestra a continuación.

La salida puede ser diferente dependiendo del router que usted esté utilizando. En la configuración predeterminada de CSR1kv, no verá información de direccionamiento IP.


```
{
  "ietf-interfaces:interface": {
```

```
    "name": "GigabitEthernet1",
    "description": "VBox",
    "type": "iana-if-type:ethernetCsmacd",
    "enabled": true,
    "ietf-ip:ipv4": {},
    "ietf-ip:ipv6": {}
  }
}
```

d. Esta interfaz recibe direcciones de una plantilla de Virtual Box. Por lo tanto, la dirección IPv4 no se muestra en **show running-config**. En su lugar, verá el comando **ip address dhcp**. Esto también se puede ver en la salida show ip interface brief.

```
CSR1kv# show ip interface brief
Interface IP-Address OK? Method Status Protocol
GigabitEthernet1 192.168.56.101 YES DHCP up up
CSR1kv#
```

e. En la siguiente Parte necesitará usar la respuesta JSON desde una interfaz configurada manualmente. Abra un terminal de comandos en el CSR1kv y configure manualmente la interfaz GigabitEthernet1 con la misma dirección IPv4 que fue asignada desde Virtual Box.

```
CSR1kv# conf t
CSR1kv(config)# interface g1
CSR1kv(config-if)# ip address 192.168.56.101 255.255.255.0
CSR1kv(config-if)# end
CSR1kv# show ip interface brief
Interface IP-Address OK? Method Status Protocol
GigabitEthernet1 192.168.56.101 YES manual up up
CSR1kv#
```

f. Regrese a Postman y envíe su solicitud GET de nuevo. Ahora debería ver información de direccionamiento IPv4 en la respuesta JSON, como se muestra a continuación.

En la siguiente Parte, copiará este formato JSON para crear una nueva interfaz.

```
{
  "ietf-interfaces:interface": {
    "name": "GigabitEthernet1",
    "description": "VBox",
    "type": "iana-if-type:ethernetCsmacd",
    "enabled": true,
    "ietf-ip:ipv4": {
      "address": [
        {
          "ip": "192.168.56.101",
          "netmask": "255.255.255.0"
        }
      ]
    },
    "ietf-ip:ipv6": {}
  }
}
```

Utilizar Postman para enviar una solicitud PUT.

En esta parte, configuraremos Postman para que envíe una solicitud PUT al CSR1kv para crear una nueva interfaz de loopback.

Nota: Si ha creado una interfaz loopback en otro laboratorio, quítela ahora o cree una nueva utilizando un número diferente.

Duplicar y modificar la última solicitud GET.

a. Duplique la última solicitud GET.

b. Para el **tipo** de solicitud, haga clic en la flecha abajo situada junto a **GET** y seleccione **PUT**.

c. Cambie el parámetro **interface=** por **=Loopback1** para especificar una nueva interfaz.

```
https://192.168.56.101/restconf/data/ietf-
interfaces:interfaces/interface=Loopback1
```

Configurar el cuerpo de la solicitud especificando la información para el nuevo loopback.

a. Para enviar una solicitud PUT, debe proporcionar la información para el cuerpo de la solicitud. Junto a la pestaña **Encabezados**, haga clic en **Contenido**. Luego, haga clic en la opción **Raw**. El campo se encuentra vacío. Si hace clic en **Enviar**, obtendrá el código de error **400 Bad Request** porque Loopback1 aún no existe y no proporcionó suficiente información para crear la interfaz.

b. Rellene la sección **Contenido** con los datos JSON requeridos para crear una nueva interfaz Loopback1. Puede copiar la sección Contenido de la solicitud GET anterior y modificarla. O puede copiar lo siguiente en la sección Contenido de su solicitud PUT. Observe que el tipo de interfaz debe establecerse en **softwareLoopback**.

```
{
  "ietf-interfaces:interface": {
    "name": "Loopback1",
    "description": "My first RESTCONF loopback",
    "type": "iana-if-type:softwareLoopback",
    "enabled": true,
    "ietf-ip:ipv4": {
      "address": [
        {
          "ip": "10.1.1.1",
          "netmask": "255.255.255.0"
        }
      ]
    },
    "ietf-ip:ipv6": {}
  }
}
```

c. Haga clic en **Enviar** para enviar la solicitud PUT al CSR1kv. Debajo de la sección Contenido, debería poder ver el código de respuesta HTTP **Status: 201 Created**. Esto indica que el recurso se creó correctamente.

d. Puede verificar que se creó la interfaz. Vuelva a su sesión SSH con el CSR1kv e ingrese **show ip interface brief**. También puede ejecutar la pestaña Postman que contiene la

solicitud para obtener información sobre las interfaces del CSR1kv que fue creada en la Parte anterior de este laboratorio.

```
CSR1kv# show ip interface brief
Interface IP-Address OK? Method Status Protocol
GigabitEthernet1 192.168.56.101 YES manual up up
Loopback1 10.1.1.1 YES other up up
CSR1kv#
```

Usar un script de Python para enviar solicitudes GET.

En esta Parte, crearemos un script de Python para enviar solicitudes GET al CSR1kv.

Crear el directorio RESTCONF y empezar a crear el script.

i. Abra VS Code. Luego haga clic en **Archivo > Abrir carpeta...** y diríjase al directorio **devnet-src**. Haga clic en **Aceptar**.

j. Abra una ventana de terminal en VS Code haciendo clic en: **Terminal> Nueva Terminal.**

k. Cree un subdirectorio que se llame **restconf** en el directorio **/devnet-src**.

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

l. En el panel **EXPLORER** en **DEVNET-SRC**, haga clic derecho en el directorio **restconf** y elija **Nuevo archivo**.

m. Nombre al archivo: **restconf-get.py**.

a. Introduzca los siguientes comandos para importar los módulos necesarios y deshabilitar las advertencias de certificado SSL:

```
import json
import requests
requests.packages.urllib3.disable_warnings()
```

El módulo **json** incluye métodos para convertir datos JSON a objetos Python y viceversa. El módulo **requests** tiene métodos que le permitirán enviar solicitudes REST a una URL.

Crear las variables que serán los componentes de la solicitud.

a. Cree una variable que se llame **api_url** y asígnele la URL que accederá a la información de la interfaz del CSR1kv.

```
api_url = "https://192.168.56.101/restconf/data/ietf-interfaces:interfaces"
```

b. Cree una variable tipo diccionario que se llame **encabezados**, dentro del diccionario cree las claves **Accept** y **Content-type** y asígneles el valor **application/yang-data+json**.

```
headers = { "Accept": "application/yang-data+json",
            "Content-type":"application/yang-data+json"
          }
```

c. Cree una variable tipo tupla de Python que se llame **basicauth**, dentro de la tupla, cree las dos claves necesarias para la autenticación, **username** y **password**.

```
basicauth = ("cisco", "cisco123!")
```

Crear una variable para enviar la solicitud y almacenar la respuesta JSON.

Utilice las variables que se crearon en el paso anterior como parámetros para el método **requests.get ()**. Este método envía una solicitud HTTP GET a la API RESTCONF del CSR1kv. Asigne el resultado de la solicitud a una variable que se llame **resp**. Esa variable contendrá la respuesta JSON de la API. Si la solicitud se realiza correctamente, el JSON contendrá el modelo de datos YANG devuelto.

a. Ingrese la siguiente instrucción:

```
resp = requests.get(api_url, auth=basicauth, headers=headers, verify=False)
```

En la siguiente tabla se enumeran los diversos elementos de esta instrucción

Elemento	Explicación
`resp`	La variable que va a contener la respuesta de la AP.
`requests.get()`	El método que realiza la solicitud GET.
`api_url`	La variable que contiene el string de la dirección URL.
`auth`	La variable tipo tupla creada para contener la información de autenticación.
`headers=headers`	Parámetro al que se le asigna la variable encabezados.
`verify=False`	Desactivar la verificación del certificado SSL cuando se realiza la solicitud.

b. Para ver el código de respuesta HTTP, agregue una instrucción print.

```
print(resp)
```

c. Guarde y ejecute el script. Debería obtener la salida que se muestra a continuación. Si no es así, compruebe todos los pasos anteriores de esta parte, así como la configuración SSH y RESTCONF del CSR1kv.

```
devasc@labvm:~/labs/devnet-src$ cd restconf/
devasc@labvm:~/labs/devnet-src/restconf$ python3 restconf-get.py
<Response [200]>
devasc@labvm:~/labs/devnet-src/restconf$
```

Formatear y mostrar los datos JSON recibidos del CSR1kV.

Ahora puede extraer los valores de respuesta del modelo YANG de la respuesta JSON.

a. El JSON de respuesta no es compatible con el diccionario ni con los objetos tipo lista de Python, por lo que debe convertirse al formato Python. Cree una nueva variable llamada **response_json** y asígnele la variable **resp**. Agregue el método **json ()** para convertir el JSON. La declaración es la siguiente:

```
response_json = resp.json()
```

b. Agregue una instrucción print para mostrar los datos JSON.

```
print(response_json)
```

c. Guarde y ejecute el script. Se debería obtener una salida similar a la siguiente:

```
devasc@labvm:~/labs/devnet-src/restconf$ python3 restconf-get.py
```

```
<Response [200]>
{'ietf-interfaces:interfaces': {'interface': [{'name': 'GigabitEthernet1',
'description': 'VBox', 'type': 'iana-if-type:ethernetCsmacd', 'enabled':
True, 'ietf-ip:ipv4': {'address': [{'ip': '192.168.56.101', 'netmask':
'255.255.255.0'}]}, 'ietf-ip:ipv6': {}}, {'name': 'Loopback1',
'description': 'My first RESTCONF loopback', 'type': 'iana-if-
type:softwareLoopback', 'enabled': True, 'ietf-ip:ipv4': {'address':
[{'ip': '10.1.1.1', 'netmask': '255.255.255.0'}]}, 'ietf-ip:ipv6': {}}]}}
devasc@labvm:~/labs/devnet-src/restconf$
```

d. Para embellecer la salida, edite la instrucción print para poder usar la función **json.dumps()** con el parámetro "indent" (sangría):

```
print(json.dumps(response_json, indent=4))
```

e. Guarde y ejecute el script. Debería obtener la salida que se muestra a continuación. Esta salida es prácticamente idéntica a la salida de su primera solicitud GET de Postman.

```
devasc@labvm:~/labs/devnet-src/restconf$ python3 restconf-
get.py
<Response [200]>
{
    "ietf-interfaces:interfaces": {
        "interface": [
            {
                "name": "GigabitEthernet1",
                "description": "VBox",
                "type": "iana-if-type:ethernetCsmacd",
                "enabled": true,
                "ietf-ip:ipv4": {
                    "address": [
                        {
                            "ip": "192.168.56.101",
                            "netmask": "255.255.255.0"
                        }
                    ]
                },
                "ietf-ip:ipv6": {}
            },
            {
                "name": "Loopback1",
                "description": "My first RESTCONF loopback",
                "type": "iana-if-type:softwareLoopback",
                "enabled": true,
                "ietf-ip:ipv4": {
                    "address": [
                        {
                            "ip": "10.1.1.1",
                            "netmask": "255.255.255.0"
                        }
                    ]
                },
                "ietf-ip:ipv6": {}
```

```
                }
            ]
        }
    }
devasc@labvm:~/labs/devnet-src/restconf$
```

Usar un script de Python para enviar una solicitud PUT.

En esta parte, crearemos un script de Python para enviar una solicitud PUT al CSR1kv. Al igual que se hizo en Postman, crearemos una nueva interfaz loopback.

Importar módulos y deshabilitar las advertencias SSL.

a. En el panel **EXPLORER** en **DEVNET-SRC**, haga clic derecho en el directorio **restconf** y elija **Archivo Nuevo**.

b. Asigne un nombre al archivo **restconf-put.py**.

c. Introduzca los siguientes comandos para importar los módulos necesarios y deshabilitar las advertencias de certificado SSL:

```
import json
import requests
requests.packages.urllib3.disable_warnings()
```

Crear las variables que serán los componentes de la solicitud.

a. Cree una variable llamada **api_url** y asígnele la URL que apunta a una nueva interfaz Loopback2.

Nota: Esta especificación de variable debe estar en una línea de su script.

```
api_url = "https://192.168.56.101/restconf/data/ietf-
interfaces:interfaces/interface=Loopback2"
```

b. Cree una variable tipo diccionario que se llame **encabezados** dentro del diccionario cree las claves Accept y Content-type y asígneles el valor **application/yang-data+json**.

```
headers = { "Accept": "application/yang-data+json",
            "Content-type":"application/yang-data+json"
          }
```

c. Cree una variable tipo tupla de Python que se llame **basicauth**, dentro de la tupla, cree los dos valores necesarios para la autenticación, username y password.

```
basicauth = ("cisco", "cisco123!")
```

d. Crear una variable tipo diccionario de Python que se llame **YangConfig** la cual contendrá los datos YANG que se requieren para crear la nueva interfaz Loopback2. Puede usar el mismo diccionario que utilizó anteriormente en Postman. Sin embargo, debe cambiar el número de interfaz y la dirección. Además, tenga en cuenta que los valores booleanos deben estar en mayúsculas en Python. Por lo tanto, asegúrese de que la **T** esté mayúscula en el par clave/valor de **"enabled": True**.

```
yangConfig = {
    "ietf-interfaces:interface": {
        "name": "Loopback2",
        "description": "My second RESTCONF loopback",
        "type": "iana-if-type:softwareLoopback",
        "enabled": True,
        "ietf-ip:ipv4": {
```

```
                "address": [
                    {
                        "ip": "10.2.1.1",
                        "netmask": "255.255.255.0"
                    }
                ]
            },
            "ietf-ip:ipv6": {}
        }
    }
```

Crear una variable para enviar la solicitud y almacenar la respuesta JSON.

Utilice las variables creadas en el paso anterior como parámetros para el método **requests.put ()**. Este método envía una solicitud PUT HTTP a la API RESTCONF. Asigne el resultado de la solicitud a una variable llamada **resp**. Esa variable contendrá la respuesta JSON de la API. Si la solicitud se realiza correctamente, el JSON contendrá el modelo de datos YANG devuelto.

a. Antes de introducir declaraciones, tenga en cuenta que esta especificación de variable debe estar en una sola línea de su script. Introduzca las siguientes instrucciones:

   Nota: Esta especificación de variable debe estar en una línea de su script.

```
resp = requests.put(api_url, data=json.dumps(yangConfig), auth=basicauth, headers=headers, verify=False)
```

b. Introduzca el siguiente código para manejar la respuesta. Si la respuesta es uno de los mensajes "correcto o exitoso" 2XX de HTTP (HTTP success), se imprimirá el primer mensaje. Cualquier otro valor de código se considera un error. El código de respuesta y el mensaje de error se imprimirán en caso de que se haya detectado un error.

```
if(resp.status_code >= 200 and resp.status_code <= 299):
    print("STATUS OK: {}".format(resp.status_code))
else:
    print('Error. Status Code: {} \nError message: {}'.format(resp.status_code,resp.json()))
```

En la siguiente tabla se enumeran los diversos elementos de estas instrucciones:

Elemento	Explicación
`resp`	Variable para contener la respuesta de la API.
`requests.put()`	El método que realiza la solicitud PUT.
`api_url`	Variable que contiene el string de la dirección URL.
`data`	Los datos que se van a enviar al punto final de la API, los cuales están en formato JSON
`auth`	La variable tipo tupla creada para contener la información de autenticación
`headers=headers`	Parámetro al que se le asigna la variable headers
`verify=False`	Parámetro que desactiva la verificación de certificado SSL cuando se realiza la solicitud.
`resp.status_code`	El código de estado HTTP en la respuesta de la solicitud **PUT** de API.

c. Guarde y ejecute el script para enviar la solicitud PUT al CSR1kv. Debería recibir un mensaje que diga **201 Status Created**. Si no es así, compruebe su código y la configuración del CSR1kv.

d. Puede verificar que la interfaz fue creada, introduciendo **show ip interface brief** en el CSR1kv.

```
CSR1kv# show ip interface brief
Interface IP-Address OK? Method Status Protocol
GigabitEthernet1 192.168.56.101 YES manual up up
Loopback1 10.1.1.1 YES other up up
Loopback2 10.2.1.1 YES other up up
CSR1kv#
```

Programas utilizados en este laboratorio.

En este laboratorio se utilizaron los siguientes scripts de Python:

```
#=====================================================================
#resconf-get.py
import json
import requests
requests.packages.urllib3.disable_warnings()

api_url = "https://192.168.56.101/restconf/data/ietf-interfaces:interfaces"

headers = { "Accept": "application/yang-data+json",
            "Content-type":"application/yang-data+json"
          }

basicauth = ("cisco", "cisco123!")

resp = requests.get(api_url, auth=basicauth, headers=headers, verify=False)

print(resp)

response_json = resp.json()
print(json.dumps(response_json, indent=4))

#end of file

#=====================================================================
#resconf-put.py
import json
import requests
requests.packages.urllib3.disable_warnings()

api_url = "https://192.168.56.101/restconf/data/ietf-
interfaces:interfaces/interface=Loopback2"
```

```
headers = { "Accept": "application/yang-data+json",
            "Content-type":"application/yang-data+json"
           }

basicauth = ("cisco", "cisco123!")

yangConfig = {
    "ietf-interfaces:interface": {
        "name": "Loopback2",
        "description": "My second RESTCONF loopback",
        "type": "iana-if-type:softwareLoopback",
        "enabled": True,
        "ietf-ip:ipv4": {
            "address": [
                {
                    "ip": "10.2.1.1",
                    "netmask": "255.255.255.0"
                }
            ]
        },
        "ietf-ip:ipv6": {}
    }
}

resp = requests.put(api_url, data=json.dumps(yangConfig), auth=basicauth,
headers=headers, verify=False)

if(resp.status_code >= 200 and resp.status_code <= 299):
    print("STATUS OK: {}".format(resp.status_code))
else:
    print('Error. Status Code: {} \nError message:
{}'.format(resp.status_code,resp.json()))

#end of file
```