# UNIVERSIDAD NACIONAL DE INGENIERÍA

# FACULTAD DE CIENCIAS


## ÁREA DE CIENCIA DE LA COMPUTACIÓN

## ADMINISTRACIÓN DE REDES

## 12° LABORATORIO

- **TÍTULO:** Utilizar RESTCONF para acceder a un dispositivo IOS XE.

- **ALUMNO:**

JHONATAN FLORINS POMA MARTINEZ 20182729F

- **PROFESORES:** YURI JAVIER CCOICCA PACASI

## 2023

**Objetivos**

**Parte 1: Armar la red y verificar la conectividad.**

**Parte 2: 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.**

**Introducción:**

**RESTCONF:** Es un protocolo de gestión de dispositivos de red basado en REST (Representational State Transfer) que permite la configuración y el monitoreo de dispositivos de red a través de interfaces web. RESTCONF se basa en el protocolo HTTP (Hypertext Transfer Protocol) y utiliza los principios de la arquitectura REST para proporcionar operaciones estándar y uniformes para interactuar con dispositivos de red. Mediante RESTCONF, los administradores de red pueden acceder a recursos y datos específicos de los dispositivos de red, como la configuración, el estado operativo, las estadísticas y las capacidades. El protocolo utiliza URI (Uniform Resource Identifier) para identificar y acceder a los recursos de los dispositivos.
RESTCONF utiliza métodos HTTP como GET, PUT, POST y DELETE para realizar operaciones de lectura, escritura, actualización y eliminación de datos de configuración en dispositivos de red. También utiliza formatos de datos estándar como JSON (JavaScript Object Notation) o XML (eXtensible Markup Language) para representar y transmitir información.
Este enfoque basado en estándares y protocolos web conocidos hace que RESTCONF sea fácilmente integrable con otras tecnologías y aplicaciones. Además, su arquitectura basada en REST permite una mayor flexibilidad y escalabilidad en comparación con los enfoques más tradicionales de gestión de dispositivos de red.

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

**Daemons de Restconf,** los daemons, o demonios, son los componentes de software que implementan y gestionan el protocolo RESTCONF en un dispositivo de red.
Los daemons de RESTCONF suelen ser procesos en ejecución que se encargan de recibir, procesar y responder a las solicitudes RESTCONF enviadas por los clientes. Estos daemons se comunican con otros componentes del sistema, como el motor de configuración, la base de datos de configuración o los controladores de dispositivos, para obtener y aplicar la configuración solicitada.

## Parte 1: Armar la red y verificar la conectividad.

En el enrutador verificamos la interfaz: “show ip interface brief”

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

Hacemos un PING, para verificar la conectividad entre DEVASC y CSR1kv.

```
devasc@labvm:~$ ping -c 5 192.168.56.103
PING 192.168.56.103 (192.168.56.103) 56(84) bytes of data.
64 bytes from 192.168.56.103: icmp_seq=1 ttl=255 time=1.20 ms
64 bytes from 192.168.56.103: icmp_seq=2 ttl=255 time=0.855 ms
64 bytes from 192.168.56.103: icmp_seq=3 ttl=255 time=0.756 ms
64 bytes from 192.168.56.103: icmp_seq=4 ttl=255 time=1.05 ms
64 bytes from 192.168.56.103: icmp_seq=5 ttl=255 time=0.609 ms

--- 192.168.56.103 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4076ms
rtt min/avg/max/mdev = 0.609/0.893/1.200/0.209 ms
devasc@labvm:~$
```

Hacemos una conexión SSH , en nuestro DEVASC con el CSR1kv.

```
devasc@labvm:~$ ssh cisco@192.168.56.103
Password:
Password:

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


CSR1kv#
```

## Parte 2: 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.

**Paso 1: Comprobar que se estén ejecutando los daemons de RESTCONF.**

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

CSR1kv#
```

**Paso 2: Habilitar y verificar el servicio RESTCONF.**

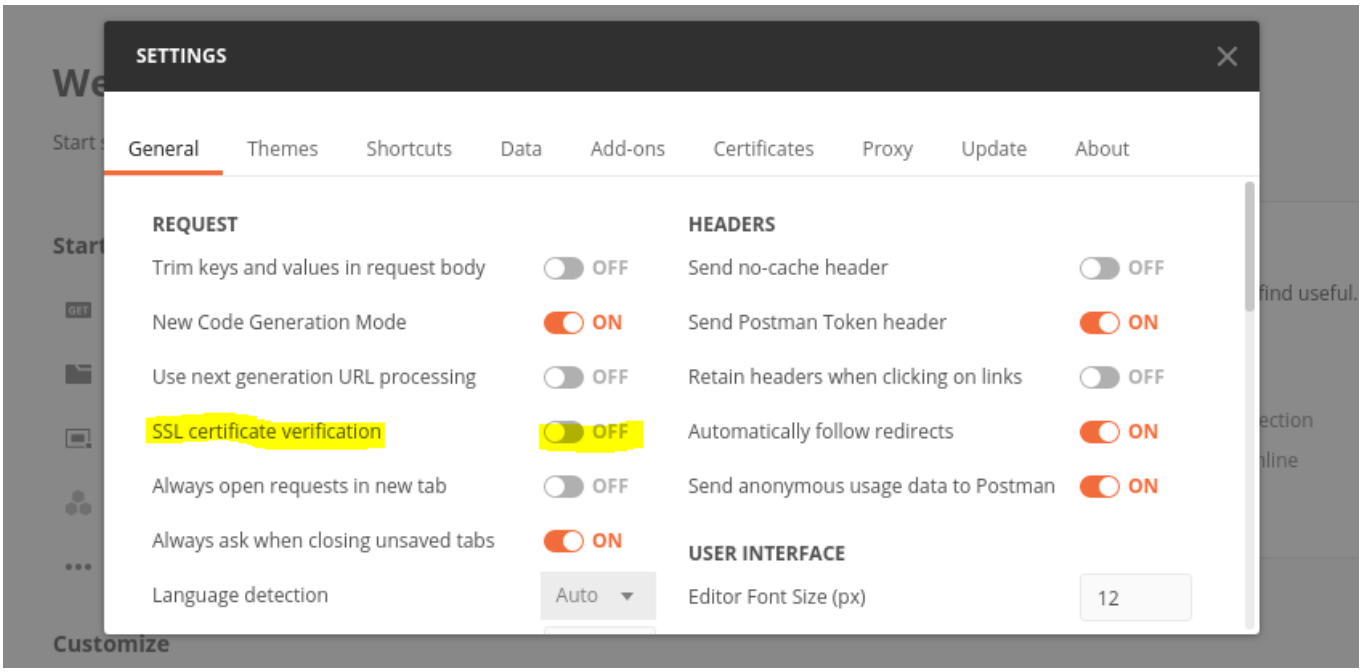
```
CSR1kv#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
CSR1kv(config)#restconf
CSR1kv(config)#exit
CSR1kv#show platform software yang-management process
confd           : Running
nesd            : Running
syncfd          : Running
ncsshd          : Running
dmiauthd        : Running
nginx           : Not Running
ndbmand         : Running
pubd            : Running

CSR1kv#
```

NETCONF

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

## Parte 3: Abrir y configurar Postman.

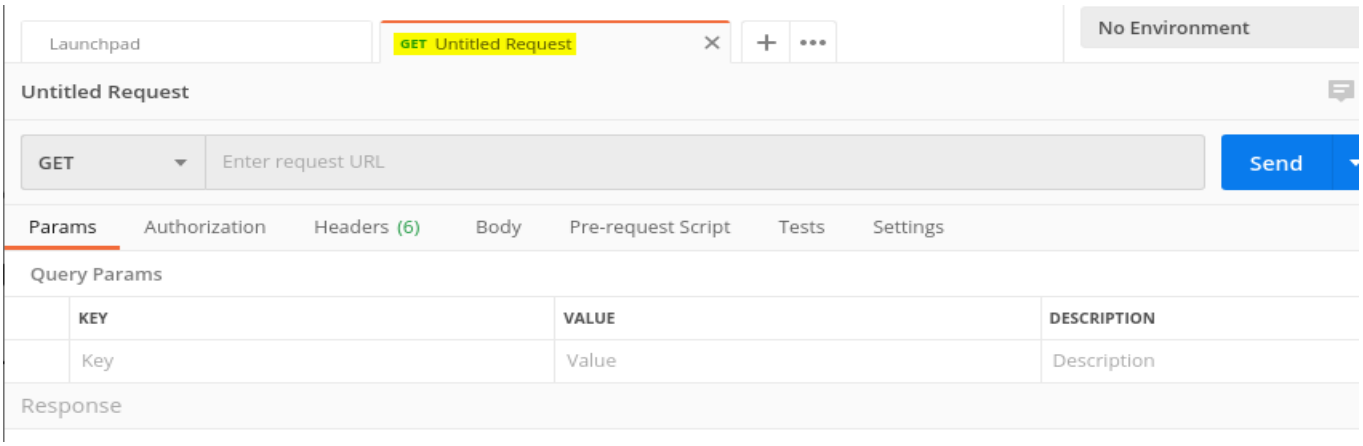
En esta parte, abriremos Postman, deshabilitamos los certificados SSL y exploramos la interfaz de usuario.



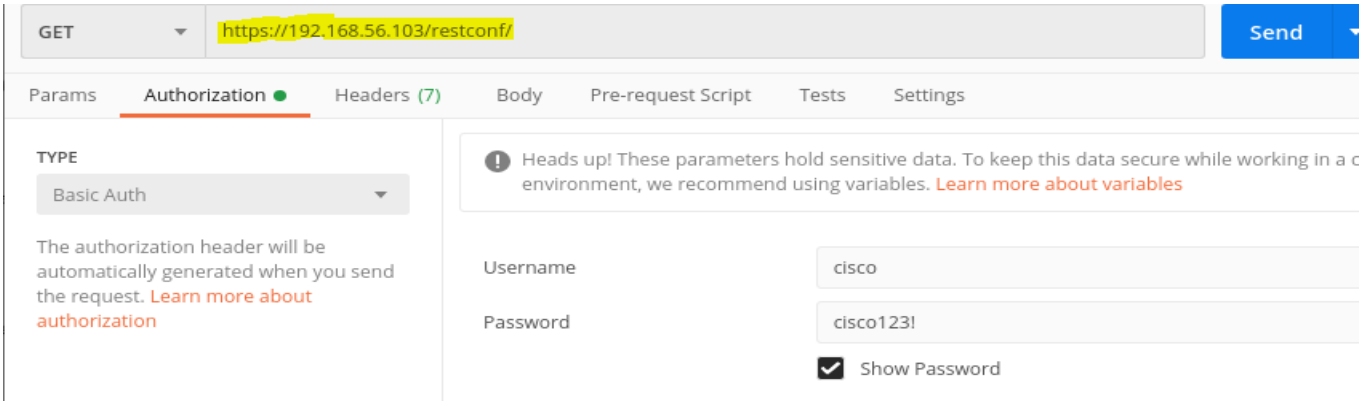

## Parte 4: Usar Postman para enviar solicitudes GET.

En esta parte, utilizaremos Postman para enviar una solicitud GET al CSR1kv para verificar si podemos conectarnos al servicio RESTCONF.

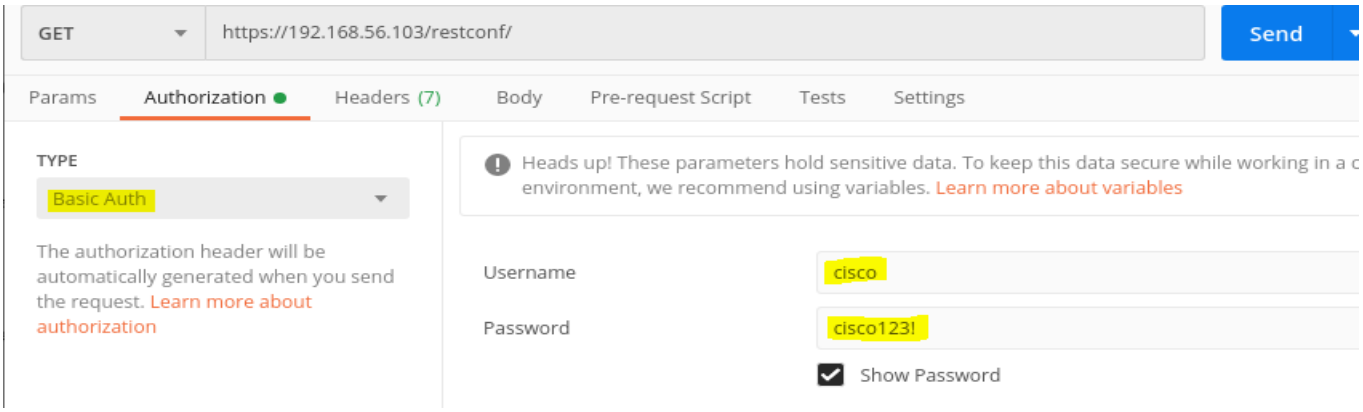
**Paso 1: Explorar la interfaz de usuario de Postman.**



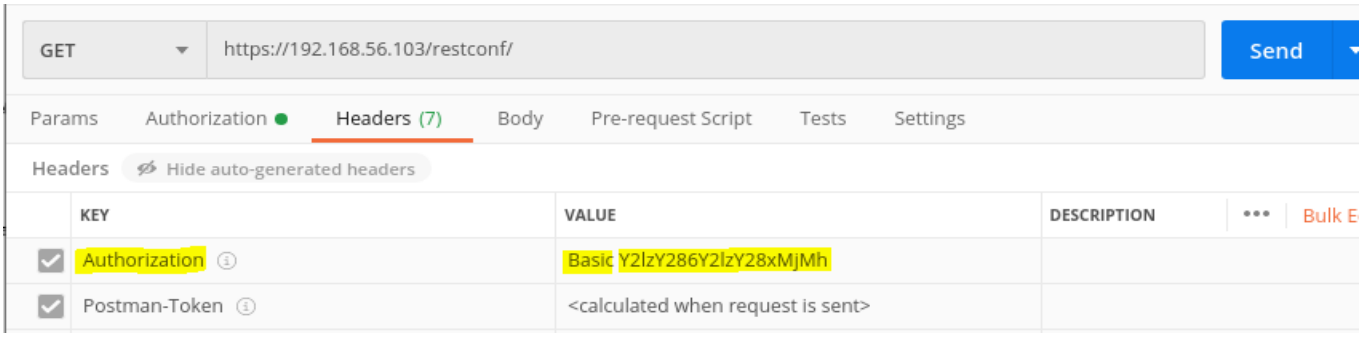

**Paso 2: Introduzca la dirección URL del CSR1kv. https://192.168.56.103/restconf/**




**Paso 3: Introduce las credenciales de autenticación. User : cisco, Pass : cisco123!**

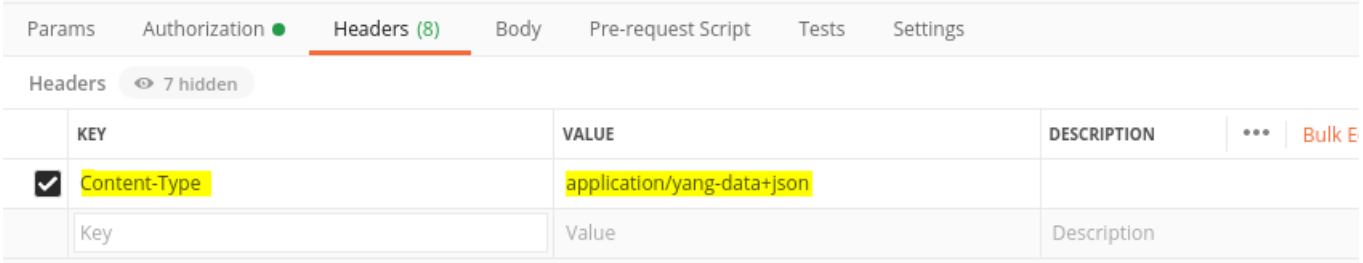



Podemos verificar que la Clave de autorización tiene un Valor básico que se utilizará para autenticar la solicitud cuando se envíe al CSR1kv.

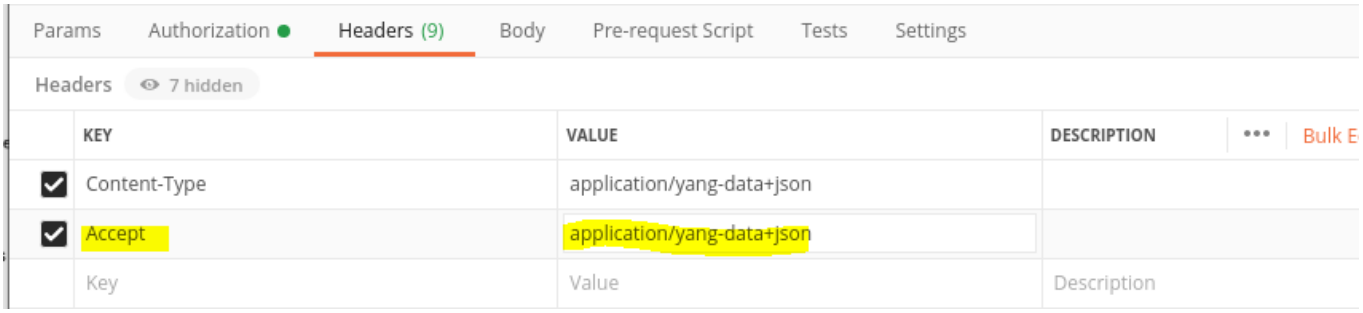


**Paso 4: Establezca JSON como el tipo de datos para enviar y recibir desde el CSR1kv.**
Podemos enviar y recibir datos desde el CSR1kv en formato **XML** o **JSON**. Para este laboratorio, utilizaremos JSON.

En el campo Clave(KEY) en blanco escriba **Content-Type** para el tipo de clave. En el campo Valor(value), escriba **application/yang-data+json**. Esto le dice a Postman que envíe datos JSON al CSR1kv.

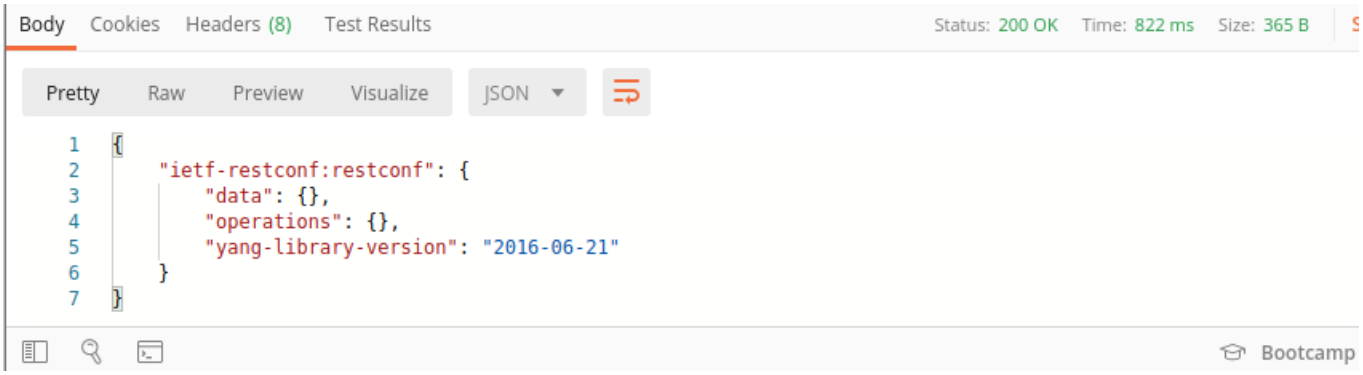



Agregamos otro par de clave/valor. El campo Clave es **Accept** y el campo Valor es **application/yang-data+json**.




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

**Paso 5: Enviar la solicitud de API al CSR1kv.**
Postman ahora tiene toda la información que necesita para enviar la solicitud GET. Haga clic en Enviar (Send). Deberíamos poder ver la siguiente **respuesta JSON del CSR1kv**.



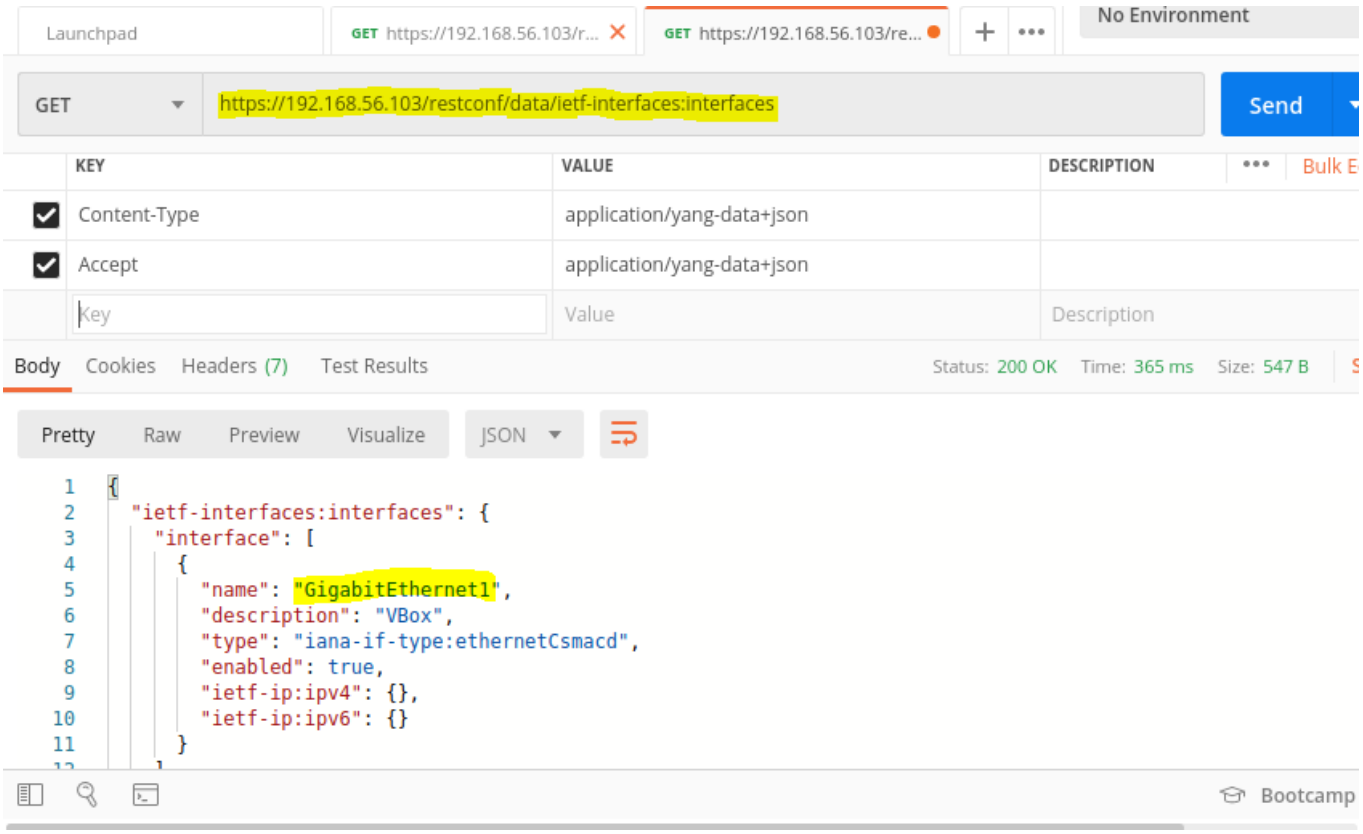

Esta respuesta JSON verifica que Postman ahora puede enviar otras solicitudes de API REST al CSR1kv.

**Paso 6: Utilizar una solicitud GET para recopilar la información de todas las interfaces del router CSR1kv.**

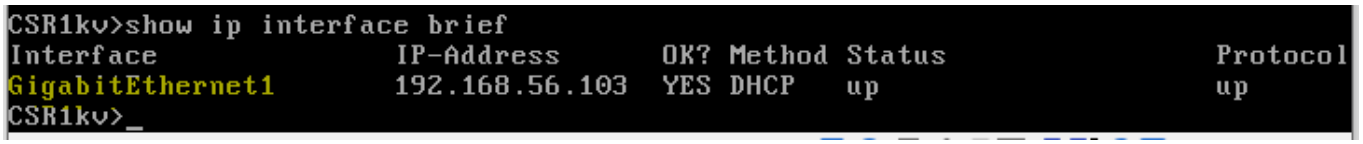
Duplicamos la pestaña GET que acabamos de crear, en ella Utilizamos el modelo YANG de **ietf-interfaces** para recopilar información de la interfaz.

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

**Esto nos muestra las interfaces del router CSR1kv, usando Postman, en nuestro caso solo nos muestra la interfaz GigabitEthernet.**




**Corroboramos en nuestro router csr1kv:**

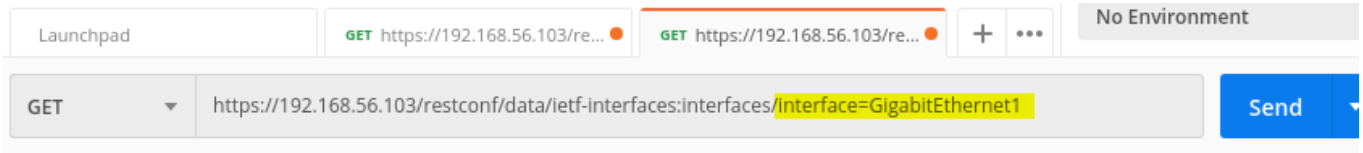



**Paso 7: Utilizar una solicitud GET para recopilar información de una interfaz específica del CSR1kv.**

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

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

En el laboratorio 11 creamos una interfaz loopback, pero yo lo elimine, como practicando usar las configuraciones por eso me aparece solo una interfaz.




hacemos click en Send:

```
{
  "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       192.168.56.103  YES DHCP   up                    up
CSR1kv>_
```

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

```
CSR1kv#
CSR1kv#config terminal
Enter configuration commands, one per line.  End with CNTL/Z.
CSR1kv(config)#interface g1
CSR1kv(config-if)#ip address 192.168.56.103 255.255.255.0
CSR1kv(config-if)#end
CSR1kv#show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.103  YES manual up                    up
CSR1kv#
```

En el router csr1kv:

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

Ahora en Postman enviamos las solicitud GET, Ahora deberíamos ver la información de direccionamiento IPv4 en la respuesta JSON, como se muestra a continuación.

**ANTERIOR:**


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


**DESPUÉS:**

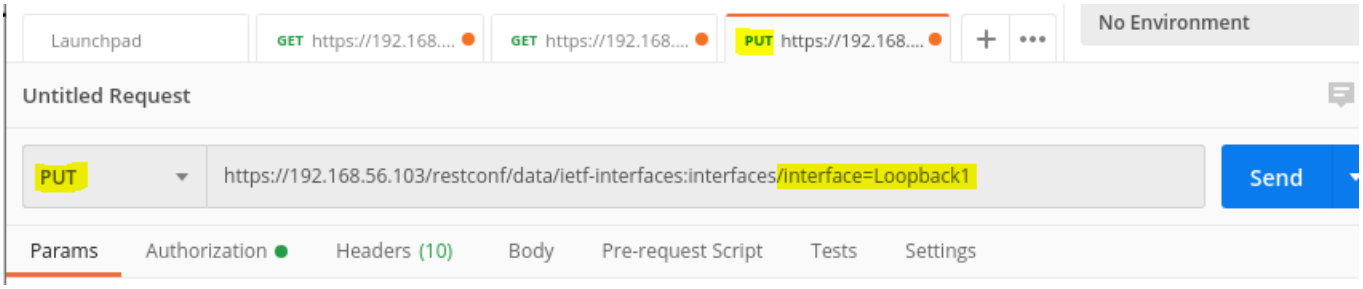

```
{
  "ietf-interfaces:interface": {
    "name": "GigabitEthernet1",
    "description": "VBox",
    "type": "iana-if-type:ethernetCsmacd",
    "enabled": true,
    "ietf-ip:ipv4": {
      "address": [
        {
          "ip": "192.168.56.103",
          "netmask": "255.255.255.0"
        }
      ]
    },
    "ietf-ip:ipv6": {}
  }
}
```


ipv4

## Parte 5: Usar Postman para enviar solicitudes PUT.

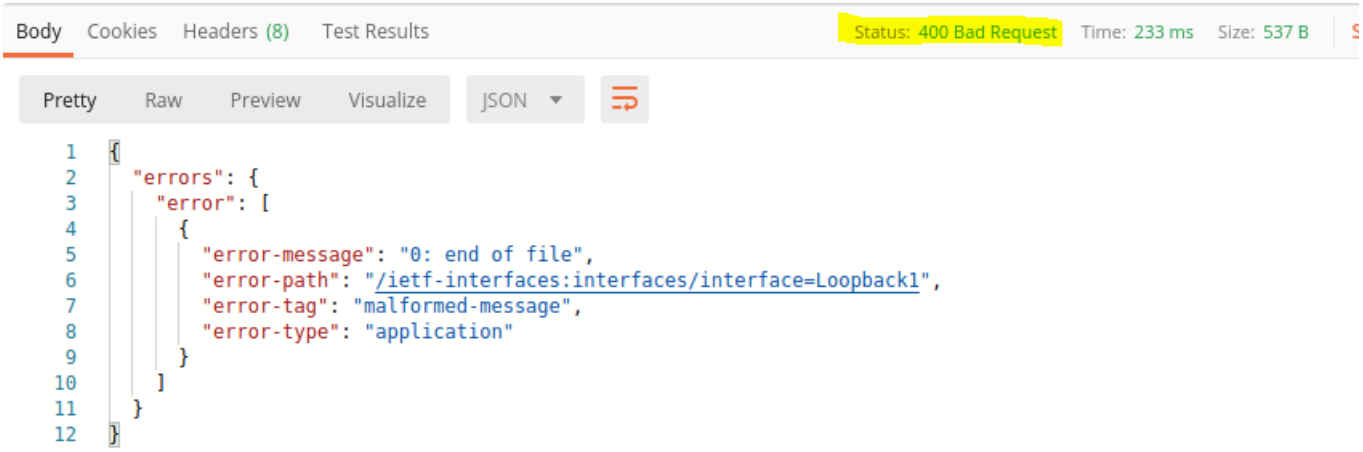
En esta parte, configuraremos Postman para que envíe una **solicitud PUT** al CSR1kv para **crear una nueva interfaz de loopback.**

**Paso 1: Duplicar y modificar la última solicitud GET. En el duplicado cambiamos a PUT, también cambiamos el parámetro interface= por =Loopback1 para especificar una nueva interfaz.**



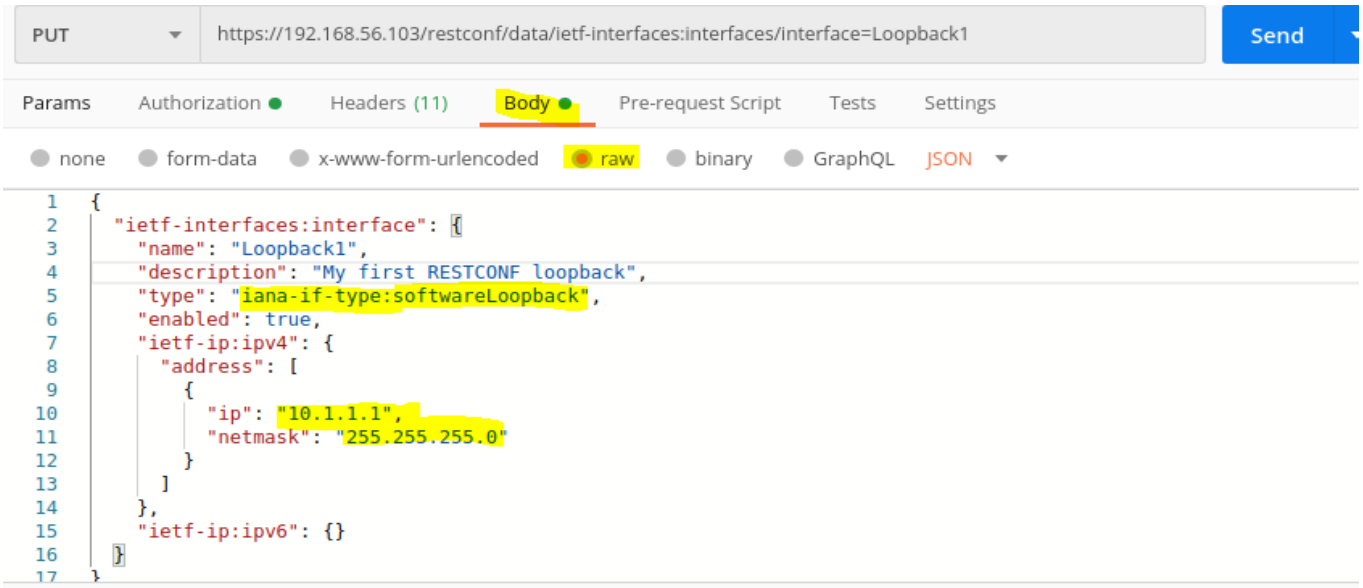

**Paso 2: Configurar el cuerpo de la solicitud especificando la información para el nuevo loopback.**

Si hacemos click en Send, nos saldrá un error porque aún no especificamos nada de la interface loopback1.

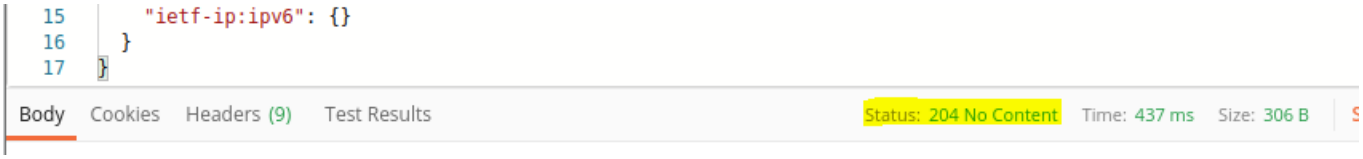



Para enviar una solicitud PUT, debe proporcionar la información para el cuerpo de la solicitud. En la pestaña Body haga click. Luego, haga clic en la opción Raw.

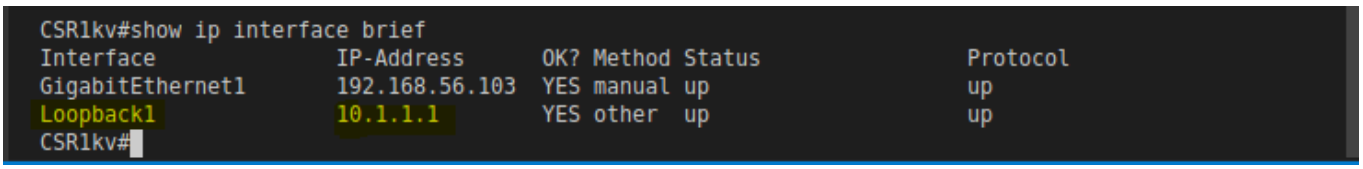
Rellenamos la sección Body con los datos JSON requeridos para crear una **nueva interfaz Loopback1**. en la sección Body de su solicitud PUT. Observe que el tipo de interfaz debe establecerse en softwareLoopback.



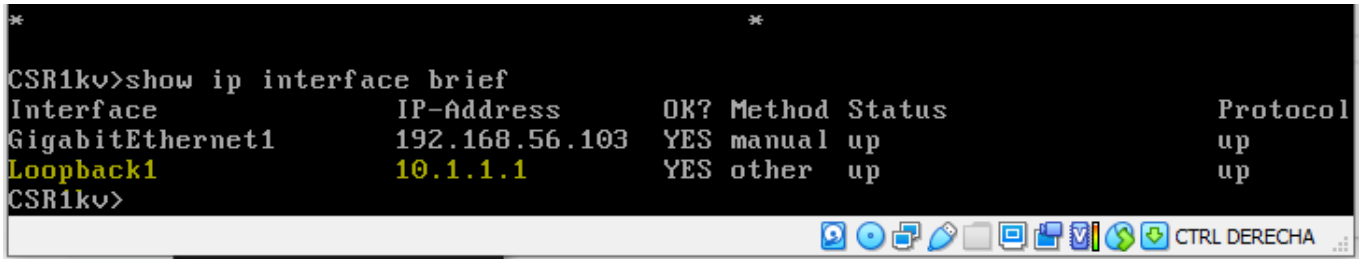
Hacemos click en SEND,
Me sale **204 No Content**, esto nos indica que la solicitud se ha procesado correctamente, pero no hay contenido para devolver como respuesta.




Verificamos en el terminal DEVASC.



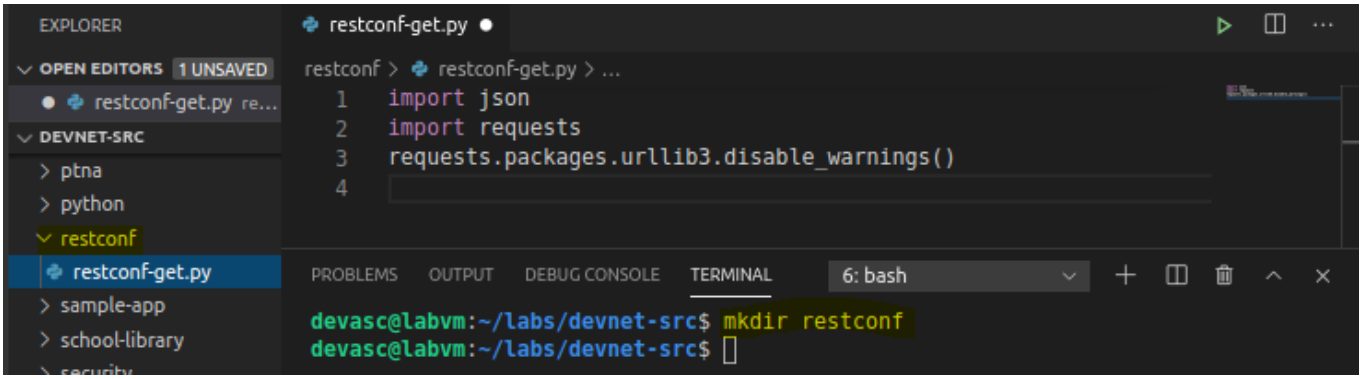

EN EL ROUTER CSR1kv:




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

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

**Paso 1: Crear el directorio RESTCONF y empezar a crear el script.**
Creamos una carpeta **restconf** en ella creamos un archivo **restconf-get.py**
Ingresamos los siguientes comandos para importar los módulos necesarios y deshabilitar las advertencias de certificado SSL.
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.




**Paso 2: Crear las variables que serán los componentes de la solicitud.**

a) Creamos una variable que se llame **api_url** y asígnele la URL que accederá a la información de la interfaz del CSR1kv.
b) Creamos una variable tipo diccionario que se llame **headers**, dentro del diccionario cree las claves Accept y Content-type y asígneles el valor application/yang-data+json.
c) Creamos 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.

```
restconf-get.py ●
restconf > restconf-get.py > ...
1  import json
2  import requests
3  requests.packages.urllib3.disable_warnings()
4
5  api_url = "https://192.168.56.103/restconf/data/ietf-interfaces:interfaces"
6  headers = { "Accept": "application/yang-data+json",
7              "Content-type":"application/yang-data+json"
8            }
9  basicauth = ("cisco", "cisco123!")
10
```

**Paso 3: Crear una variable para enviar la solicitud y almacenar la respuesta JSON.**
Usamos las variables que creamos en el paso anterior como parámetros para el método **requests.get()** Este método envía una solicitud HTTP GET a la API RESTCONF del CSR1kv.
Asignamos 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.

```
restconf-get.py ●
restconf > restconf-get.py > ...
1  import json
2  import requests
3  requests.packages.urllib3.disable_warnings()
4
5  api_url = "https://192.168.56.103/restconf/data/ietf-interfaces:interfaces"
6  headers = { "Accept": "application/yang-data+json",
7              "Content-type":"application/yang-data+json"
8            }
9  basicauth = ("cisco", "cisco123!")
10
11 resp = requests.get(api_url, auth=basicauth, headers=headers, verify=False)
12
```

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. |

Agregamos un print(resp) para ver en la salida **el código de respuesta HTTP.**

```
11 resp = requests.get(api_url, auth=basicauth, headers=headers, verify=False)
12 print(resp)
13
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL        6: bash
devasc@labvm:~/labs/devnet-src$ mkdir restconf
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$
```

El código <Response [200]> indica que la solicitud ha tenido éxito y que el contenido esperado se ha devuelto correctamente en la respuesta

**Paso 4: Formatear y mostrar los datos JSON recibidos del CSR1kv.**
Ahora podemos extraer los valores de respuesta del modelo YANG de la respuesta JSON.

El JSON de respuesta no es compatible con el diccionario ni con los objetos tipo lista de Python, por lo que debemos convertir al formato Python. Creamos una nueva variable llamada **response_json** y le asignamos la variable resp. Agregue el método json() para convertir el JSON.

```
11  resp = requests.get(api_url, auth=basicauth, headers=headers, verify=False)
12  print(resp)
13
14  response_json = resp.json()
15  print(response_json)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                    6: bash

devasc@labvm:~/labs/devnet-src$ mkdir restconf
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$ python3 restconf-get.py
<Response [200]>
{'ietf-interfaces:interfaces': {'interface': [{'name': 'GigabitEthernet1', 'description': 'VBox', 't
ype': 'iana-if-type:ethernetCsmacd', 'enabled': True, 'ietf-ip:ipv4': {'address': [{'ip': '192.168.5
6.103', '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$
```

Pero aun el JSON no se ve bien, para embellecer la salida, editamos la instrucción print para poder usar la función **json.dumps()** con el parámetro "indent" (sangría):

```
11  resp = requests.get(api_url, auth=basicauth, headers=headers, verify=False)
12  print(resp)
13
14  response_json = resp.json()
15  #print(response_json)
16  print(json.dumps(response_json,indent=2))
17
```

```
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": "192.168.56.103",
              "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$
```

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

**Paso 1: Importar módulos y deshabilitar las advertencias SSL.**
En nuestra capeta **restconf,** creamos un archivo **restconf-put.py**.

**Paso 2: Crear las variables que serán los componentes de la solicitud.**

a) Creamos una variable llamada **api_url** y asígnele la URL que apunta a una nueva interfaz **Loopback2**.
b) Creamos una variable tipo diccionario que se llame **headers** dentro del diccionario cree las claves **Accept y Content-type** y asígneles el valor **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.
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, pero modificando el nombre.

restconf-get.py | restconf-put.py

restconf > restconf-put.py > ...

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

#Creamos una nueva interfaz Loopback2
api_url = "https://192.168.56.103/restconf/data/ietf-interfaces:interfaces/interface=Loopback2"

headers = { "Accept": "application/yang-data+json",
            "Content-type":"application/yang-data+json"
          }
basicauth = ("cisco", "cisco123!")

#Creamos un diccionario, la cual tendra los datos YANG para la nueva interfaz
#(asi como creamos en RAW en postman), Loopback2.
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": {}
  }
}
```

**Paso 3: 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.

Usaremos este IF, 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. Ya usamos en la data el  **json.dumps().**

```
33  resp = requests.put(api_url, data=json.dumps(yangConfig), auth=basicauth, headers=headers, verify=False)
34
35  if(resp.status_code >= 200 and resp.status_code <= 299):
36      print("STATUS OK: {}".format(resp.status_code))
37  else:
38      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. |

Guardamos el script y ejecutamos.

```
*                                        *


CSR1kv#show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.103  YES manual up                    up
Loopback1              10.1.1.1        YES other  up                    up
CSR1kv#exit
Connection to 192.168.56.103 closed.
devasc@labvm:~/labs/devnet-src$ cd restconf
devasc@labvm:~/labs/devnet-src/restconf$ python3 restconf-put.py
STATUS OK: 201
devasc@labvm:~/labs/devnet-src/restconf$ ssh cisco@192.168.56.103
Password:

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


CSR1kv#show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.103  YES manual up                    up
Loopback1              10.1.1.1        YES other  up                    up
Loopback2              10.2.1.1        YES other  up                    up
CSR1kv#
```

En el router CSR1kv:

```
CSR1kv>show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.56.103  YES manual up                    up
Loopback1              10.1.1.1        YES other  up                    up
Loopback2              10.2.1.1        YES other  up                    up
CSR1kv>
```

CTRL DERECHA

## RESUMEN:

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

**2° parte,** configuramos la máquina virtual CSR1kv, para que acepte mensajes RESTCONF.

**3° parte,** Abrimos y exploramos Postman, deshabilitamos los certificados SSL esto para omitir la validación de certificados y permitir la conexxion a servidores con certificados no confiables. Pero no es recomendable hacer, como estamos haciendo pruebas lo deshabilitamos.

**4° parte,** Usaremos Postman para hacer una solicitud GET(solicitamos al servidor que nos envíe el contenido de un recurso específico), le enviamos la direccion URL de nuestro router virtual, ingresamos tambien las credenciales para autenticarnos, establecemos el tipo de dato JSON (también podemos usar XML), para enviar y recibir los datos que enviamos al csr1kv.
Y al hacer la consulta debemos ver en postman las interfaces que hay.

**5° parte,** Usaremos Postman para hacer una solicitud PUT(para actualizar o crear un recurso en un servidor). En este caso crearemos una nueva interfaz, le enviamos la direccion URL de nuestro router virtual + /interface=Loopback1, ingresamos tambien las credenciales para autenticarnos, establecemos el tipo de dato JSON (tambien podemos usar XML), Configuramos el cuerpo del nuevo interfaz.
Guardamos y enviamos, y cuando hacemo una consulta veremos que ya se creo la nueva interfaz.

**6° parte,** Lo mismo que hicimos en postman, pero ahora lo haremos usando scripts en python.
Enviaremos solicitudes GET, importamos json, y los request, creamos variables para acceder a la URL, al igual que el tipo de dato que recibiremos y enviaremos que sera JSON. Tambien la ingresamos una variable para la autenticacion.
Y cuando ejecutamos el script, veremos en formato JSON las interfaces que tenemos.

**7° parte,** Lo mismo que en postman, pero ahora crearemos usando un script en python, un nuevo interfaz Loopback2.

## CONCLUSIONES:

Hicimos un proceso de establecimiento de la comunicación, configuración y realización de solicitudes utilizando RESTCONF a través de herramientas como Postman y scripts en Python. Esto nos permite interactuar con el dispositivo IOS XE (CSR1kv) y realizar acciones como obtener información de las interfaces existentes y crear nuevas interfaces.