



UNIVERSIDAD TECNOLÓGICA DEL NORTE DE GUANAJUATO

Tecnologías de la Información y Comunicación
Programa educativo:

TSU en Infraestructura de Redes Digitales
Área académica:

Programación de Redes
Asignatura:

Unidad III: Programación de Redes

Grupo: GIR0441

RESTCONF with Python
Laboratorio 5:

Venado Soria German Emiliano
Alumno:

Gabriel Barrón Rodríguez
Docente:

Dolores Hidalgo, C.I.N., Gto., Miércoles 14 de Diciembre de 2022
Lugar y fecha:

Lab – RESTCONF with Python

Objectives

Part 1: RESTCONF basics in Python

Part 2: Modify interface configuration with RESTCONF in Python

Background / Scenario

Following up the previous lab activity, in this lab you will learn how to execute the RESTCONF API calls using Python scripts.

Required Resources

- Python 3.x environment
- Access to a router with the IOS XE operating system version 16.6 or higher.

Instructions

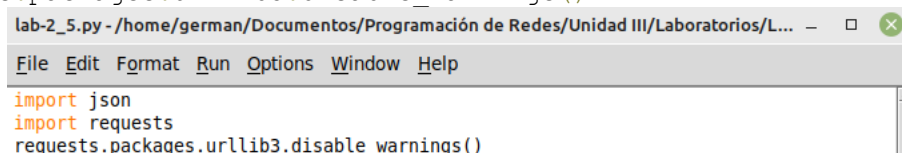
Part 1: RESTCONF in Python

In this part, you will use Python to request a RESTCONF API.

Step 1: Import modules and disable SSL warnings.

- In IDLE, click **File > New File** to open IDLE Editor.
- Save the file as **lab 2.5.py**.
- Enter the following commands to import the modules and disable SSL certificate warnings:

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



Nos pide crear un nuevo archivo en el IDLE con el nombre de “lab_2.5.py”, y posteriormente importar los módulos json y requests, y también deshabilitar las advertencias del certificado SSL.

The **json** module includes methods convert JSON data to Python objects and vice versa. The **requests** module has methods that will let us send REST requests to a URI.

Step 2: Build the request components.

Create a string variable to hold the API endpoint URI and two dictionaries, one for the request header and one for the body JSON. These are the same tasks you completed in the Postman application.

- Create a variable named **api_url** and assign the URL (adjust the IP address to match the router’s current address).

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

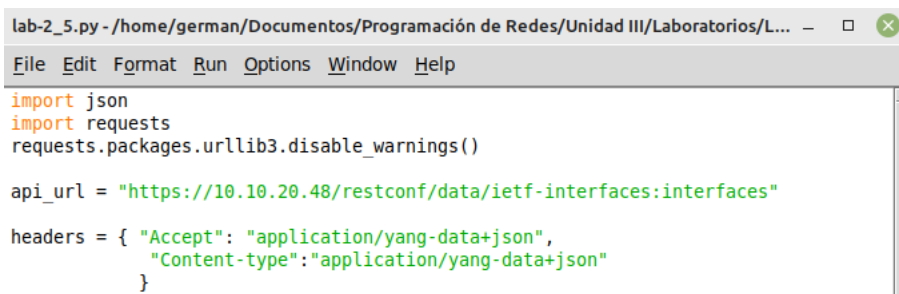


```
lab-2_5.py - /home/german/Documentos/Programación de Redes/Unidad III/Laboratorios/L...  
File Edit Format Run Options Window Help  
import json  
import requests  
requests.packages.urllib3.disable_warnings()  
  
api_url = "https://10.10.20.48/restconf/data/ietf-interfaces:interfaces"
```

Aquí se solicita crear una variable, tal cual y como no la indica, solo que cambiaremos la dirección en el parámetro del host, ahí colocaremos la dirección IP del Router remoto.

- e. Create a dictionary variable named **headers** that has keys for **Accept** and **Content-type** and assign the keys the value **application/yang-data+json**.

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

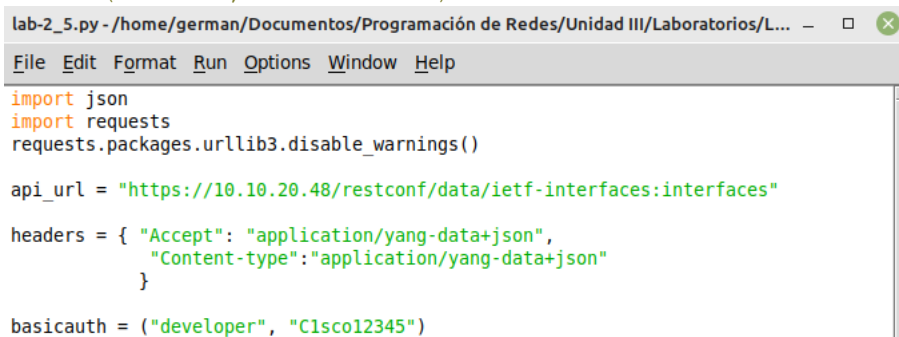


```
lab-2_5.py - /home/german/Documentos/Programación de Redes/Unidad III/Laboratorios/L...  
File Edit Format Run Options Window Help  
import json  
import requests  
requests.packages.urllib3.disable_warnings()  
  
api_url = "https://10.10.20.48/restconf/data/ietf-interfaces:interfaces"  
  
headers = { "Accept": "application/yang-data+json",  
            "Content-type": "application/yang-data+json"  
}
```

En este apartado se agrega la configuración del headers, que vienen siendo los encabezados de la solicitud API, donde dentro debe incluir el nombre y el tipo de valor de cada encabezado, en este caso se están solicitando con un formato de tipo json.

- f. Create a Python tuple variable named **basicauth** that has two keys needed for authentication, **username** and **password**.

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



```
lab-2_5.py - /home/german/Documentos/Programación de Redes/Unidad III/Laboratorios/L...  
File Edit Format Run Options Window Help  
import json  
import requests  
requests.packages.urllib3.disable_warnings()  
  
api_url = "https://10.10.20.48/restconf/data/ietf-interfaces:interfaces"  
  
headers = { "Accept": "application/yang-data+json",  
            "Content-type": "application/yang-data+json"  
}  
  
basicauth = ("developer", "Cisco12345")
```

Y en este apartado agregamos la autenticación, ingresando con el usuario y contraseña que tiene el Router, si prestamos atención, es el mismo proceso que en el POSTMAN, solo que aquí no se aprecia en un entorno gráfico, sino como modo consola.

Step 3: Send the request.

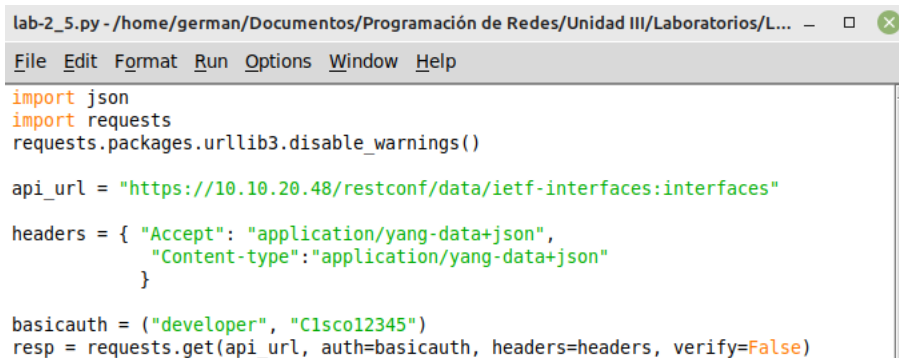
You will now use the variables created in the previous step as parameters for the **requests.get()** method. This method sends an HTTP GET request to the RESTCONF API. You will assign the result of the request to a variable name **resp**. That variable will hold the JSON response from the API. If the request is successful, the JSON will contain the returned YANG data model.

- g. Enter the following statement:

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

The various elements of this statement are:

Element	Explanation
resp	the variable to hold the response from the API.
requests.get()	the method that actually makes the GET request.
api_url	the variable that holds the URL address string
auth	the tuple variable created to hold the authentication information
headers=headers	a parameter that is assigned to the headers variable
verify=False	disables verification of the SSL certificate when the request is made



```
lab-2_5.py - /home/german/Documentos/Programación de Redes/Unidad III/Laboratorios/L...
File Edit Format Run Options Window Help

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

api_url = "https://10.10.20.48/restconf/data/ietf-interfaces:interfaces"

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

basicauth = ("developer", "C1scol2345")
resp = requests.get(api_url, auth=basicauth, headers=headers, verify=False)
```

Nos pide utilizar las variables que fueron creadas anteriormente dentro del método `request.get()`, el cual enviará una solicitud HTTP GET a la API RESTCONF, y así poder asignar el resultado de la solicitud dentro de variable llamada “resp”, que generalmente el JSON tendrá el modelo de datos YANG.

- h. Save your script and run it. There will not be any output yet but the script should run without errors. If not, review the steps and make sure your code does not contain any errors.

Step 4: Evaluate the response.

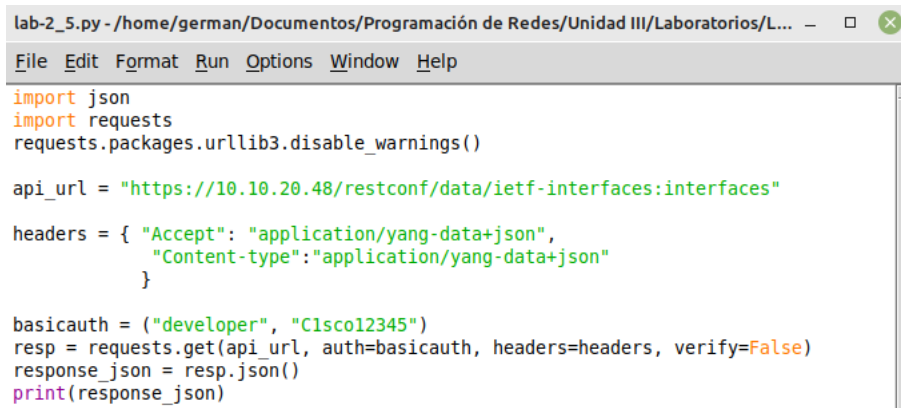
Now the YANG model response values can be extracted from the response JSON.

- i. The response JSON is not compatible with Python dictionary and list objects so it is converted to Python format. Create a new variable called **response_json** and assign the variable **resp** to it adding the **json()** method to convert the JSON. The statement is as follows:

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

- j. You can verify that your code returns the JSON in the IDLE Shell by temporarily adding a print statement to your script, as follows:

```
print(response_json)
```



```
lab-2_5.py - /home/german/Documentos/Programación de Redes/Unidad III/Laboratorios/L...
File Edit Format Run Options Window Help

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

api_url = "https://10.10.20.48/restconf/data/ietf-interfaces:interfaces"

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

basicauth = ("developer", "Cisco12345")
resp = requests.get(api_url, auth=basicauth, headers=headers, verify=False)
response_json = resp.json()
print(response_json)
```

Lo siguiente es substraer los datos del modelo YANG del json de respuesta, pero estos no son compatibles con el diccionario de Python, por lo que debemos convertirlo nosotros al formato de Python por medio de una variable asignado el valor de resp en formato json, y finalmente imprimir el resultado.

- k. Save and run your script. You should get output similar to the following although your service ticket number will be different:

```
= RESTART: /home/german/Documentos/Programación de Redes/Unidad III/Laboratorios
/Lab-5/lab-2_5.py
{'ietf-interfaces:interfaces': {'interface': [{'name': 'GigabitEthernet1', 'desc
ription': 'MANAGEMENT INTERFACE - DON'T TOUCH ME', 'type': 'iana-if-type:etherne
tCsmacd', 'enabled': True, 'ietf-ip:ipv4': {'address': [{'ip': '10.10.20.48', 'n
etmask': '255.255.255.0'}]}, 'ietf-ip:ipv6': {}}, {'name': 'GigabitEthernet2', '
description': 'Network Interface', 'type': 'iana-if-type:ethernetCsmacd', 'enabl
ed': False, 'ietf-ip:ipv4': {}, 'ietf-ip:ipv6': {}}, {'name': 'GigabitEthernet3',
'description': 'Network Interface', 'type': 'iana-if-type:ethernetCsmacd', 'enabl
ed': False, 'ietf-ip:ipv4': {}, 'ietf-ip:ipv6': {}}]}}
```

Y esta es la salida del script anterior, el cual es similar a la salida que se mostraba en esta práctica, de una forma un poco complicada de entender para los usuarios que están acostumbrados a trabajar más gráfico, solo cambian los datos propietarios como el nombre y la dirección IP del dispositivo.

- l. To prettify the output, use the `json.dumps()` function with the “indent” parameter:

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

- m. Save and run your script. If you experience errors, check the code again.

Part 2: Modify interface configuration with RESTCONF in Python

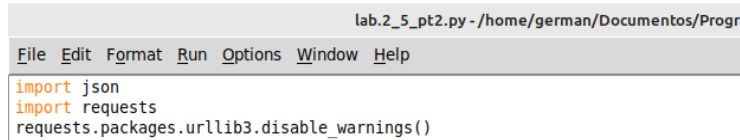
Step 1: Create the Python HTTP PUT request

In this part, you will use Python to request a RESTCONF API with a PUT method to create or modify existing configuration.

Step 2: Import modules and disable SSL warnings.

- n. In IDLE, click **File > New File** to open IDLE Editor.
- o. Save the file as **lab 2.5 part2.py**.
- p. Enter the following commands to import the modules and disable SSL certificate warnings:

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



```
lab.2_5_pt2.py - /home/german/Documentos/Progr...
File Edit Format Run Options Window Help
import json
import requests
requests.packages.urllib3.disable_warnings()
```

En esta segunda parte se va hacer lo mismo que en la parte uno, entonces creamos otro archivo en el IDLE con el nombre de “lab_2.5_p2.py”, y comenzamos a importar los módulos json y requests, y también deshabilitar las advertencias del certificado SSL.

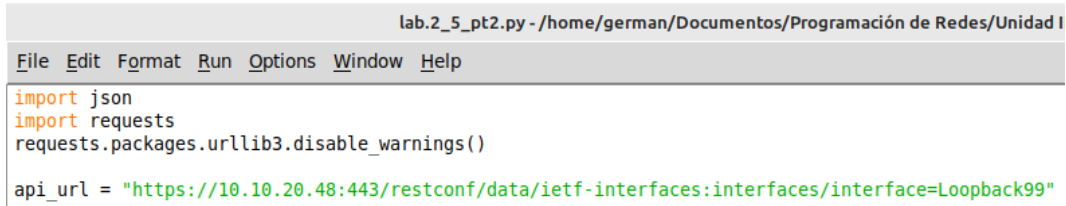
The **json** module includes methods convert JSON data to Python objects and vice versa. The **requests** module has methods that will let us send REST requests to a URI.

Step 3: Build the request components.

Create a string variable to hold the API endpoint URI and two dictionaries, one for the request header and one for the body JSON. These are the same tasks you completed in the Postman application.

- q. Create a variable named **api_url** and assign the URL that targets the **Loopback99** interface.

```
api_url = "https://192.168.56.101/restconf/data/ietf-
interfaces:interfaces/interface=Loopback99"
```



```
lab.2_5_pt2.py - /home/german/Documentos/Programación de Redes/Unidad I
File Edit Format Run Options Window Help
import json
import requests
requests.packages.urllib3.disable_warnings()

api_url = "https://10.10.20.48:443/restconf/data/ietf-interfaces:interfaces/interface=Loopback99"
```

También se solicita crear una variable para la URL, donde cambiaremos la dirección en el parámetro del host, ahí colocaremos la dirección IP del Router remoto y indicaremos al final la interfaz loopback99.

- r. Create a dictionary variable named **headers** that has keys for **Accept** and **Content-type** and assign the keys the value **application/yang-data+json**.

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



```
lab.2_5_pt2.py - /home/german/Documentos/Programación de Redes/Unidad I
File Edit Format Run Options Window Help
import json
import requests
requests.packages.urllib3.disable_warnings()

api_url = "https://10.10.20.48:443/restconf/data/ietf-interfaces:interfaces/interface=Loopback99"

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

Posteriormente se agrega la configuración del headers, que vienen siendo los encabezados de la solicitud API, donde dentro debe incluir el nombre y el tipo de valor de cada encabezado, en este caso se están solicitando con un formato de tipo json.

- s. Create a Python tuple variable named **basicauth** that has two keys needed for authentication, **username** and **password**.

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

```
lab.2_5_pt2.py - /home/german/Documentos/Programación de Redes/Unidad I
File Edit Format Run Options Window Help

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

api_url = "https://10.10.20.48:443/restconf/data/ietf-interfaces:interfaces/interface=Loopback99"

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

basicauth = ("developer", "C1sco12345")
```

Aquí se crea una variable para generar la autenticación, para poder tener acceso al Router por medio del usuario y contraseña.

- t. Create a Python dictionary variable yangConfig holding the YANG data to create new interface Loopback99 (you use here the dictionary data from the Postman lab before, be aware that the JSON's boolean **true** is in Python **True** with capital "T"):

```
yangConfig = {
    "ietf-interfaces:interface": {
        "name": "Loopback99",
        "description": "WHATEVER99",
        "type": "iana-if-type:softwareLoopback",
        "enabled": True,
        "ietf-ip:ipv4": {
            "address": [
                {
                    "ip": "99.99.99.99",
                    "netmask": "255.255.255.0"
                }
            ]
        },
        "ietf-ip:ipv6": {}
    }
}

yangConfig = {
    "ietf-interfaces:interface": {
        "name": "Loopback99",
        "description": "Venado-Lab_2.5",
        "type": "iana-if-type:softwareLoopback",
        "enabled": True,
        "ietf-ip:ipv4": {
            "address": [
                {
                    "ip": "192.168.20.210",
                    "netmask": "255.255.255.0"
                }
            ]
        },
        "ietf-ip:ipv6": {}
    }
}
```

En este paso agregamos una configuración, la cual esta práctica nos brinda para guiarnos y poder crear una segunda interfaz Loopback pero ahora llamada 99, asignando otra dirección IP diferente para que no genere ningún error.

Step 4: Send the PUT request.

You will now use the variables created in the previous step as parameters for the **requests.put()** method. This method sends an HTTP PUT request to the RESTCONF API. You will assign the result of the request to

a variable name **resp**. That variable will hold the JSON response from the API. If the request is successful, the JSON will contain the returned YANG data model.

u. Enter the following statement:

```
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 code {}, reply: {}".format(resp.status_code, resp.json()))
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 code {}, reply: {}".format(resp.status_code, resp.json()))
```

Se nos pide trabajar con las variables que fueron creadas anteriormente integrando los parámetros dentro del método `request.put()`, el cual enviará una solicitud HTTP PUT a la API RESTCONF. Para esto guardaremos el resultado de la solicitud dentro de una variable llamada “resp”, y esta tendrá la respuesta JSON contendrá el modelo de datos YANG devuelto.

The various elements of this statement are:

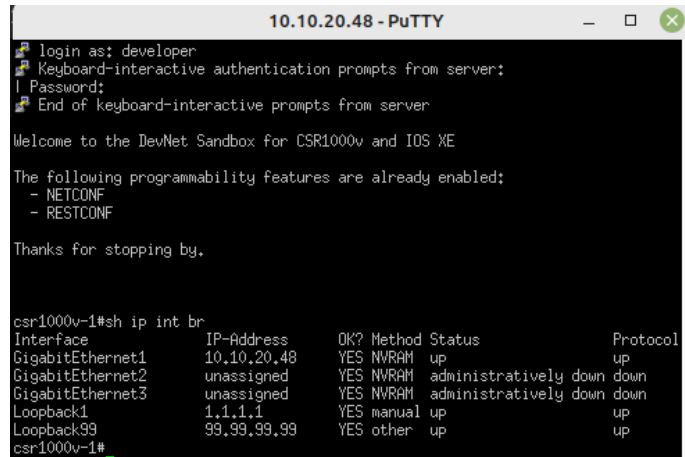
Element	Explanation
<code>resp</code>	the variable to hold the response from the API.
<code>requests.get()</code>	the method that actually makes the GET request.
<code>api_url</code>	the variable that holds the URL address string
<code>data</code>	the data to be sent to the API endpoint
<code>auth</code>	the tuple variable created to hold the authentication information
<code>headers=headers</code>	a parameter that is assigned to the headers variable
<code>verify=False</code>	disables verification of the SSL certificate when the request is made
<code>resp.status_code</code>	The HTTP status code in the API Request reply

v. Save your script and run it.

```
= RESTART: /home/german/Documentos/Programación de Redes/Unidad III/Laboratorios
/Lab-5/Lab.2_5_pt2.py
STATUS OK: 201
```

Como podemos observar, es el mismo mensaje que el resultado de esta práctica, nos manda un mensaje OK con la leyenda 201, que nos indica que fue creado correctamente la interfaz.

w. Verify using the IOS CLI that the new Loopback99 interface has been created (sh ip int brief).



```
10.10.20.48 - PuTTY
login as: developer
Keyboard-interactive authentication prompts from server:
Password:
End of keyboard-interactive prompts from server

Welcome to the DevNet Sandbox for CSR1000v and IOS XE

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

Thanks for stopping by.

csr1000v-1#sh ip int br
Interface      IP-Address      OK? Method Status      Protocol
GigabitEthernet1  10.10.20.48    YES NVRAM    up          up
GigabitEthernet2  unassigned     YES NVRAM    administratively down down
GigabitEthernet3  unassigned     YES NVRAM    administratively down down
Loopback1        1.1.1.1        YES manual  up          up
Loopback99       99.99.99.99    YES other   up          up
csr1000v-1#
```

Nos pide verificar en el entorno del Router que se aparezca la interfaz de Loopback99 que fue creada anteriormente.

- x. Modify the code to delete the interface Loopback99.
- y. What changes were applied to the code to delete the interface Loopback99?

R= Con un “`requests.delete()`”, señalando todos los parámetros como URL, headers, autenticación y verificación dentro de la función.

Investigaciones

- **¿Qué es IDLE?** Se trata de un entorno integrado de desarrollo para el aprendizaje de Python, que es una alternativa a la línea de comandos. Algunos sistemas operativos vienen con el intérprete de Python, pero en otros sistemas puede que sea necesario instalarlo. En Gnu/Linux, IDLE se distribuye como una aplicación separada que se puede instalar desde los repositorios de cada distribución.

Conclusiones

Todo este laboratorio lo controle perfectamente, pues me di cuenta que fue el mismo proceso con el que trabajamos en el POSTMAN, obviamente trabajando por una terminal de IDLE, no tenía ni la más mínima idea que fuera el mismo objetivo, pero me fui percatando por todo lo que se fue solicitando, puedo decir que ya puedo dominar las solicitudes de RESTCONF tanto en POSTMAN como en Python IDLE.