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.

a. In IDLE, click **File > New File** to open IDLE Editor.

b. Save the file as **lab 2.5.py**.

c. Enter the following commands to import the modules and disable SSL certificate warnings:

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

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.

d. 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"
```

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"
          }
```

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

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

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

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)
```

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

```
>>>
 RESTART: B:/DevNetAcad PS-E/Workshops/ETW3 Model Driven Programmability/Python Files with Solutions/idle/lab 2.9.py
{'ietf-interfaces:interfaces': {'interface': [{'name': 'GigabitEthernet1', 'type': 'iana-if-type:ethernetCsmacd', 'enabled':
True, 'ietf-ip:ipv4': {}, 'ietf-ip:ipv6': {}}, {'name': 'Loopback1', 'description': 'WHATEVER', 'type': 'iana-if-type:softwar
eLoopback', 'enabled': True, 'ietf-ip:ipv4': {'address': [{'ip': '2.2.2.2', 'netmask': '255.255.255.0'}]}, 'ietf-ip:ipv6': {}
}, {'name': 'Loopback2', 'description': 'NEWBUTSAME', 'type': 'iana-if-type:softwareLoopback', 'enabled': True, 'ietf-ip:ipv4
': {}, 'ietf-ip:ipv6': {}}, {'name': 'Loopback99', 'description': 'WHATEVER99', 'type': 'iana-if-type:softwareLoopback', 'ena
bled': True, 'ietf-ip:ipv4': {'address': [{'ip': '99.99.99.99', 'netmask': '255.255.255.0'}]}, 'ietf-ip:ipv6': {}}, {'name':
'Loopback100', 'description': 'TEST1', 'type': 'iana-if-type:softwareLoopback', 'enabled': True, 'ietf-ip:ipv4': {'address':
[{'ip': '100.100.100.100', 'netmask': '255.255.255.0'}]}, 'ietf-ip:ipv6': {}}]}}
>>>
```

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 5: 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 6: 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()
```

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 7: 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"
```

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"
          }
```

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

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

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": {}
    }
}
```

Step 8: 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()))
```

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
`data`	the data to the sent to the API endpoint
`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
`resp.status_code`	The HTTP status code in the API Request reply

v. Save your script and run it.

```
>>>
 RESTART: B:/DevNetAcad PS-E/Workshops/ETW3 Model Driven Programmability/Python Files with Solutions/idle/lab 2.9 part2.py
STATUS OK: 201
>>>
```

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

x. Modify the code to delete the interface Loopback99.

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