Lab - Use RESTCONF to Access an IOS XE Device

Objectives

Part 1: Build the Network and Verify Connectivity

Part 2: Configure an IOS XE Device for RESTCONF Access

Part 3: Open and Configure Postman

Part 4: Use Postman to Send GET Requests

Part 5: Use Postman to Send a PUT Request

Part 6: Use a Python Script to Send GET Requests

Part 7: Use a Python Script to Send a PUT Request

Background / Scenario

The RESTCONF protocol provides a simplified subset of NETCONF features over a RESTful API. RESTCONF allows you to make RESTful API calls to an IOS XE device. The data that is returned by the API can be formatted in either XML or JSON. In the first half of this lab, you will use the Postman program to construct and send API requests to the RESTCONF service that is running on the CSR1kv. In the second half of the lab, you will create Python scripts to perform the same tasks as your Postman program.

Required Resources

- 1 PC with operating system of your choice
- Virtual Box or VMWare
- DEVASC Virtual Machine
- CSR1kv Virtual Machine

Instructions

Part 1: Launch the VMs and Verify Connectivity

In this Part, you launch the two VMs for course and verify connectivity. You will then establish a secure shell (SSH) connection.

Step 1: Launch the VMs

If you have not already completed the **Lab - Install the Virtual Machine Lab Environment** and the **Lab - Install the CSR1kv VM**, do so now. If you have already completed these labs, launch both the DEVASC VM and CSR1000v VM now.

Step 2: Verify connectivity between the VMs.

a. In the CSR1kv VM, press Enter to get a command prompt and then use **show ip interface brief** to verify that the IPv4 address is 192.168.56.101. If the address is different, make a note of it.

b. Open terminal in the DEVASC VM.

c. Ping the CSR1kv to verify connectivity. You should have already done this in the installation labs. If you are not able to ping, then revisit those labs listed above in Part 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 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:~$
```

Step 3: Verify SSH connectivity to the CSR1kv VM.

a. In the terminal for the DEVASC VM, SSH to the CSR1kv VM with the following command:

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

Note: The first time you SSH to the CSR1kv, your DEVASC VM warns you about the authenticity of the CSR1kv. Because you trust the CSR1kv, answer yes to the 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. Enter **cisco123!** as the password and you should now be at the privileged EXEC command prompt for the CSR1kv.

```
Password: <cisco123!>


CSR1kv#
```

c. Leave the SSH session open for the next Part.

Part 2: Configure an IOS XE Device for RESTCONF Access

In this Part, you will configure the CSR1kv VM to accept RESTCONF messages. You will also start the HTTPS service.

Note: The services discussed in this Part may already be running on your VM. However, make sure you know the commands to view the running services and to enable them.

Step 1: Verify that the RESTCONF daemons are running.

RESTCONF should already be running because it was part of the default configuration provided by NetAcad. From the terminal, you can use the **show platform software yang-management process** command to see if all the daemons associated with the RESTCONF service are running. The NETCONF daemon may also be running, but it won't be used in this lab. If one or more of the required daemons are not running, proceed to Step 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#
```

Note: The purpose and function of all the daemons is beyond the scope of this course.

Step 2: Enable and verify the RESTCONF service.

a. Enter the global configuration command **restconf** to enable the RESTCONF service on the CSR1kv.

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

b. Verify that the required RESTCONF daemons are now running. Recall that **ncsshd** is the NETCONF service, which may be running on your device. We do not need it for this lab. However, you do need **nginx**, which is the HTTPS server. This will allow you to make REST API calls to the RESTCONF service.

```
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
```

Step 3: Enable and verify the HTTPS service.

a. Enter the following global configuration commands to enable the HTTPS server and specify that server authentication should use the local database.

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

b. Verify that the HTTPS server (nginx) is now running.

```
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
```

Part 3: Open and Configure Postman

In this Part, you will open Postman, disable SSL certificates, and explore the user interface.

Step 1: Open Postman.

a. In the **DEVASC VM**, open the Postman application.

b. If this is the first time you have opened Postman, it may ask you to create an account or sign in. At the bottom of the window, you can also click the “Skip” message to skip signing in. Signing in is not required to use this application.

Step 2: Disable SSL certification verification.

By default, Postman has SSL certification verification turned on. You will not be using SSL certificates with the CSR1kv; therefore, you need to turn off this feature.

a. Click **File > Settings**.

b. Under the **General** tab, set the **SSL certificate verification** to **OFF**.

c. Close the **Settings** dialog box.

Part 4: Use Postman to Send GET Requests

In this Part, you will use Postman to send a GET request to the CSR1kv to verify that you can connect to the RESTCONF service.

Step 1: Explore the Postman user interface.

a. In the center, you will see the **Launchpad**. You can explore this area if you wish.

b. Click the plus sign (+) next to the **Launchpad** tab to open a **GET Untitled Request**. This interface is where you will do all of your work in this lab.

Step 2: Enter the URL for the CSR1kv.

a. The request type is already set to GET. Leave the request type set to GET.

b. In the “Enter request URL” field, type in the URL that will be used to access the RESTCONF service that is running on the CSR1kv:

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

Step 3: Enter authentication credentials.

Under the URL field, there are tabs listed for **Params**, **Authorization**, **Headers**, **Body**, **Pre-request Script**, **Test**, and **Settings**. In this lab, you will use **Authorization**, **Headers**, and **Body**.

a. Click the **Authorization** tab.

b. Under Type, click the down arrow next to “Inherit auth from parent” and choose **Basic Auth**.

c. For **Username** and **Password**, enter the local authentication credentials for the CSR1kv:

Username: **cisco**

Password: **cisco123!**

d. Click **Headers**. Then click the **7 hidden**. You can verify that the Authorization key has a Basic value that will be used to authenticate the request when it is sent to the CSR1kv.

Step 4: Set JSON as the data type to send to and receive from the CSR1kv.

You can send and receive data from the CSR1kv in XML or JSON format. For this lab, you will use JSON.

a. In the **Headers** area, click in the first blank **Key** field and type **Content-Type** for the type of key. In the **Value** field, type **application/yang-data+json**. This tells Postman to send JSON data to the CSR1kv.

b. Below your **Content-Type** key, add another key/value pair. The **Key** field is **Accept** and the **Value** field is **application/yang-data+json**.

Note: You can change application/yang-data+json to application/yang-data+xml to send and receive XML data instead of JSON data, if necessary.

Step 5: Send the API request to the CSR1kv.

Postman now has all the information it needs to send the GET request. Click **Send**. Below **Temporary Headers**, you should see the following JSON response from the CSR1kv. If not, verify that you completed the previous steps in this part of the lab and correctly configured RESTCONF and HTTPS service in Part 2.

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

This JSON response verifies that Postman can now send other REST API requests to the CSR1kv.

Step 6: Use a GET request to gather the information for all interfaces on the CSR1kv.

a. Now that you have a successful GET request, you can use it as a template for additional requests. At the top of Postman, next to the **Launchpad** tab, right-click the **GET** tab that you just used and choose **Duplicate Tab**.

b. Use the **ietf-interfaces** YANG model to gather interface information. For the URL, add **data/ietf-interfaces:interfaces**:

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

c. Click **Send**. You should see a JSON response from the CSR1kv that is similar to the output shown below. Your output may be different depending on your particular router.

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

Step 7: Use a GET request to gather information for a specific interface on the CSR1kv.

In this lab, only the GigabitEthernet1 interface is configured. To specify just this interface, extend the URL to only request information for this interface.

a. Duplicate your last GET request.

b. Add the **interface=** parameter to specify an interface and type in the name of the interface.

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

Note: If you request interface information from a different Cisco device with names that use forward slashes, such as GigabitEthernet0/0/1, use the HTML code **%2F** for the forward slashes in the interface name. So, **0/0/1** becomes **0%2F0%2F1**.

c. Click **Send**. You should see a JSON response from the CSR1kv that is similar to output below. Your output may be different depending on your particular router. In the default CSR1kv setup, you will not see IP addressing information.

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

d. This interface receives addressing from a Virtual Box template. Therefore, the IPv4 address is not shown under **show running-config**. Instead, you will see the **ip address dhcp** command. You can see this also in the show ip interface brief output.

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

e. In the next Part you will need to use the JSON response from a manually configurated interface. Open a command terminal with the CSR1kv and manually configure the GigabitEthernet1 interface with the same IPv4 address currently assigned to it by 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. Return to Postman and send your GET request again. You should now see IPv4 addressing information in the JSON response, as shown below. In the next Part, you will copy this JSON format to create a new interface.

```
{
  "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": {}
    }
}
```

Part 5: Use Postman to Send a PUT Request

In this Part, you will configure Postman to send a PUT request to the CSR1kv to create a new loopback interface.

Note: If you created a Loopback interface in another lab, either remove it now or create a new one by using a different number.

Step 1: Duplicate and modify the last GET request.

a. Duplicate the last GET request.

b. For the **Type** of request, click the down arrow next to **GET** and choose **PUT**.

c. For the **interface=** parameter, change it to **=Loopback1** to specify a new interface.

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

Step 2: Configure the body of the request specifying the information for the new loopback.

a. To send a PUT request, you need to provide the information for the body of the request. Next to the **Headers** tab, click **Body**. Then click the **Raw** radio button. The field is currently empty. If you click **Send** now, you will get error code **400 Bad Request** because Loopback1 does not exist yet and you did not provide enough information to create the interface.

b. Fill in the **Body** section with the required JSON data to create a new Loopback1 interface. You can copy the Body section of the previous GET request and modify it. Or you can copy the following into the Body section of your PUT request. Notice that the type of interface must be set to **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. Click **Send** to send the PUT request to the CSR1kv. Below the Body section, you should see the HTTP response code **Status: 201 Created**. This indicates that the resource was created successfully.

d. You can verify that the interface was created. Return to your SSH session with the CSR1kv and enter **show ip interface brief**. You can also run the Postman tab that contains the request to get information about the interfaces on the CSR1kv that was created in the previous Part of this lab.

```
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#
```

Part 6: Use a Python script to Send GET Requests

In this Part, you will create a Python script to send GET requests to the CSR1kv.

Step 1: Create the RESTCONF directory and start the script.

a. Open VS code. Then click **File > Open Folder...** and navigate to the **devnet-src** directory. Click **OK**.

b. Open a terminal window in VS Code: **Terminal > New Terminal**.

c. Create a subdirectory called **restconf** in the **/devnet-src** directory.

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

d. In the **EXPLORER** pane under **DEVNET-SRC**, right-click the **restconf** directory and choose **New File**.

e. Name the file **restconf-get.py**.

a. Enter the following commands to import the modules that are required and disable SSL certificate warnings:

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

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

Step 2: Create the variables that will be the components of the request.

a. Create a variable named **api_url** and assign it the URL that will access the interface information on the CSR1kv.

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

b. 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"
          }
```

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

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

Step 3: Create a variable to send the request and store the JSON response.

Use the variables that were created in the previous step as parameters for the **requests.get()** method. This method sends an HTTP GET request to the RESTCONF API on the CSR1kv. Assign the result of the request to a variable named **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.

a. Enter the following statement:

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

The table below lists the various elements of this statement:

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 the headers variable
`verify=False`	Disables verification of the SSL certificate when the request is made

b. To see the HTTP response code, add a print statement.

```
print(resp)
```

c. Save and run your script. You should get the output shown below. If not, verify all previous steps in this part as well as the SSH and RESTCONF configuration for the 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$
```

Step 4: Format and display the JSON data received from the CSR1kv.

Now you can extract the YANG model response values from the response JSON.

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

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

b. Add a print statement to display the JSON data.

```
print(response_json)
```

c. Save and run your script. You should get output similar to the following:

```
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. To prettify the output, edit your print statement to use the **json.dumps()** function with the “indent” parameter:

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

e. Save and run your script. You should get the output shown below. This output is virtually identical to the output of your first Postman GET request.

```
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$
```

Part 7: Use a Python Script to Send a PUT Request

In this Part, you will create a Python script to send a PUT request to the CSR1kv. As was done in Postman, you will create a new loopback interface.

Step 1: Import modules and disable SSL warnings.

a. In the **EXPLORER** pane under **DEVNET-SRC**, right-click the **restconf** directory and choose **New File**.

b. Name the file **restconf-put.py**.

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

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

Step 2: Create the variables that will be the components of the request.

a. Create a variable named **api_url** and assign it the URL that targets a new Loopback2 interface.

Note: This variable specification should be on one line in your script.

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

b. 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"
           }
```

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

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

d. Create a Python dictionary variable **yangConfig** that will hold the YANG data that is required to create the new interface Loopback2. You can use the same dictionary that you used previously in Postman. However, change the interface number and address. Also, be aware that Boolean values must be capitalized in Python. Therefore, make sure that the **T** is capitalized in the key/value pair for **"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": {}
    }
}
```

Step 3: Create a variable to send the request and store the JSON response.

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. Assign the result of the request to a variable named **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.

a. Before entering statements, please note that this variable specification should be on only one line in your script. Enter the following statements:

Note: This variable specification should be on one line in your script.

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

b. Enter the code below to handle the response. If the response is one of the HTTP success messages, the first message will be printed. Any other code value is considered an error. The response code and error message will be printed in the event that an error has been detected.

```
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()))
```

The table below lists the various elements of these statements:

Element	Explanation
`resp`	The variable to hold the response from the API.
`requests.put()`	The method that makes the PUT request.
`api_url`	The variable that holds the URL address string.
`data`	The data to be sent to the API endpoint, which is formatted as JSON.
`auth`	The tuple variable created to hold the authentication information.
`headers=headers`	A parameter that is assigned the headers variable.
`verify=False`	A parameter that disables verification of the SSL certificate when the request is made.
`resp.status_code`	The HTTP status code in the API **PUT** request reply.

c. Save and run the script to send the PUT request to the CSR1kv. You should get a **201 Status Created** message. If not, check your code and the configuration for the CSR1kv.

d. You can verify that the interface was created by entering **show ip interface brief** on the 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#
```

Programs Used in this Lab

The following Python scripts were used in this lab:

```
#======================================================================
#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
```