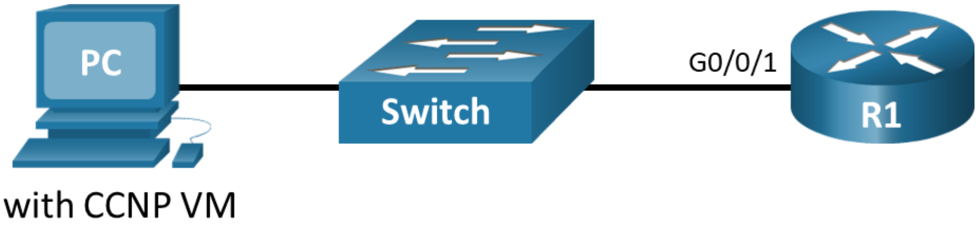
Lab - Use RESTCONF to Access an IOS XE Router (Instructor Version)

Instructor Note: Red font color or gray highlights indicate text that appears in the instructor copy only.

Answers: 28.1.5 Lab - Use RESTCONF to Access an IOS XE Device

Topology



Addressing Table

Device	Interface	IP Address	Subnet Mask
R1	G0/0/1	192.168.1.1	255.255.255.0
PC	NIC	DHCP	DHCP

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 is a subset of NETCONF. RESTCONF allows you to make RESTful API calls to an IOS XE device. The data that is returned by the API can be 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 R1. In the second half of the lab, you will create Python scripts to perform the same tasks as your Postman program.

Required Resources

- 1 Router (Cisco 4221 with Cisco IOS XE Release 16.9.4 universal image or comparable)
- 1 Switch (Optional: any switch available for connecting R1 and the PC)

- 1 PC (Choice of operating system with Cisco Networking Academy CCNP VM running in a virtual machine client and terminal emulation program)
- Ethernet cables as shown in the topology

Instructions

Part 1: Build the Network and Verify Connectivity

In Part 1, you will cable the devices, start the Python VM, and configure R1 for NETCONF and RESTCONF access over an SSH connection. You will then verify connectivity between the Python VM and R1, and test an SSH connection to R1.

Step 1: Cable the network as shown in the topology.

Attach the devices as shown in the topology diagram, and cable as necessary.

Step 2: Start the CCNP VM.

Note: If you have not completed **Lab - Install the CCNP Virtual Machine**, do so now before continuing with this lab.

a. Open VirtualBox. Start the **CCNP VM** virtual machine.

b. Enter the password **StudentPass** to log into the VM if prompted.

Step 3: Configure R1.

Connect to the console of R1 and configure basic settings for the lab as follows:

```
enable
configure terminal
hostname R1
no ip domain lookup
line con 0
logging synchronous
 exec-timeout 0 0
line vty 0 15
 exec-t 0 0
 logg sync
 login local
 transport input ssh
!Configure SSH which is required for NETCONF and RESTCONF access
ip domain name example.netacad.com
crypto key generate rsa modulus 2048
username cisco privilege 15 password cisco123!
interface GigabitEthernet0/0/1
 description Link to PC
 ip address 192.168.1.1 255.255.255.0
 no shutdown
!Configure a DHCP server to assign IPv4 addressing to the CCNP VM
ip dhcp excluded-address 192.168.1.1 192.168.1.10
ip dhcp pool LAN
```

```
 network 192.168.1.0 /24
 default-router 192.168.1.1
 domain-name example.netacad.com
end
copy run start
```

Step 4: Verify the CCNP VM can ping the default gateway.

a. In the **CCNP VM**, open a terminal.

b. Verify that the **CCNP VM** is connected to R1 by either entering **ip address** to verify that the CCNP VM received IP addressing from the DHCP server, or by pinging R1 at 192.168.1.1. Enter **Ctrl+C** to break out of the ping, as shown in the example output below.

```
student@Ubuntu-Master:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
default qlen 1000
    link/ether 00:50:56:b3:72:3b brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.15/24 brd 192.168.1.255 scope global dynamic noprefixroute ens160
       valid_lft 79564sec preferred_lft 79564sec
    inet6 fe80::1ae4:952f:402d:6b1/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: ens192: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
default qlen 1000
    link/ether 00:50:56:b3:26:b6 brd ff:ff:ff:ff:ff:ff
    inet 192.168.50.183/24 brd 192.168.50.255 scope global dynamic noprefixroute
ens192
       valid_lft 70687sec preferred_lft 70687sec
    inet6 fe80::4c87:a2b3:aa9:5470/64 scope link noprefixroute
       valid_lft forever preferred_lft forever

student@Ubuntu-Master:~$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=255 time=0.703 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=255 time=0.748 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=255 time=0.757 ms
^C
--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2033ms
rtt min/avg/max/mdev = 0.703/0.736/0.757/0.023 ms
```

c. If the **CCNP VM** has not received IPv4 addressing, check your physical connections between the host PC and R1. Also, verify that R1 is configured correctly according to the previous step.

Step 5: Establish an SSH Connection to R1.

a. Open the PuTTY SSH Client.

b. Enter the IPv4 address for the default gateway, 192.168.1.1, and click **Open**.

c. You should be able to login to R1 with the username **cisco** and password **cisco123!**. If not, verify that your SSH configuration is correct on R1.

Part 2: Configure an IOS XE Device for RESTCONF Access

In Part 2, you will configure R1 to accept RESTCONF messages. You will also start the HTTPS service on R1.

Step 1: Verify that RESTCONF is running on R1.

RESTCONF may already be running if another student enabled it, or if a later IOS version enables it by default. From the PuTTY 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.

```
R1# show platform software yang-management process
confd                : Not Running
nesd                 : Not Running
syncfd               : Not Running
ncsshd               : Not Running
dmiauthd             : Not Running
nginx                : Not Running
ndbmand              : Not Running
pubd                 : Not Running
```

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 R1.

```
R1(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.

```
R1(config)# exit
R1# 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.

```
R1# configure terminal
R1(config)# ip http secure-server
R1(config)# ip http authentication local
```

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

```
R1(config)# exit
R1# 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 Part 3, you will open Postman, disable SSL certificates, and explore the user interface.

Step 1: Open Postman.

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

b. The first time you open Postman, it will 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 R1; 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 Part 4, you will use Postman to send a GET request to R1 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 a R1.

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 R1:

```
https://192.168.1.1/restconf/
```

Step 3: Enter authentication credentials.

Under the URL field, there are tabs listed for **Params**, **Authorization**, **Headers**, **Body**, **Pre-request Script**, and **Test**. 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 that were configured on R1 in Part 1:

Username: **cisco**

Password: **cisco123!**

Step 4: Add authentication credentials to the request headers.

a. Click the **Preview Request** button. This will add the authentication credentials to the request header.

b. Click **Headers**. Then expand the **Temporary Headers** field. You can verify that the Authorization key has a Basic value that will be used to authenticate the request when it is sent to R1.

Step 5: Set JSON as the data type to send to and receive from R1.

You can send and receive data from R1 in XML or JSON format. For this lab, you will use JSON.

a. Above the Temporary Headers area is the **Headers** area. Click in the **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 R1.

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 6: Send the API request to R1.

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 R1. If not, verify that you completed the previous steps in this part of the lab and correctly configured SSH in Part 1 and RESTCONF 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 R1.

Step 7: Use a GET request to gather the information for all interfaces on R1.

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 Current Tab**.

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

```
https://192.168.1.1/restconf/data/ietf-interfaces:interfaces
```

c. Click **Send**. You should see a JSON response from R1 that is similar to the output shown below. Your output may be different depending on your particular router.

```
{
  "ietf-interfaces:interfaces": {
    "interface": [
      {
        "name": "GigabitEthernet0",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": false,
        "ietf-ip:ipv4": {},
        "ietf-ip:ipv6": {}
      },
      {
        "name": "GigabitEthernet0/0/0",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": false,
        "ietf-ip:ipv4": {},
        "ietf-ip:ipv6": {}
      },
      {
        "name": "GigabitEthernet0/0/1",
        "description": "Link to PC",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": true,
        "ietf-ip:ipv4": {
          "address": [
            {
              "ip": "192.168.1.1",
              "netmask": "255.255.255.0"
            }
          ]
        },
        "ietf-ip:ipv6": {}
      }
    ]
  }
}
```

Step 8: Use a GET request to gather information for a specific interface on R1.

In this lab, only the GigabitEthernet 0/0/0 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. However, notice that you need to use the HTML code **%2F** for the forward slashes in the interface name. So, 0/0/1 becomes 0%2F0%2F1.

```
https://192.168.1.1/restconf/data/ietf-interfaces:interfaces/interface=GigabitEthernet0%2F0%2F1
```

Note: The above URL should be on one line.

c. Click **Send**. You should see a JSON response from R1 that is similar to output below. Your output may be different depending on your particular router.

```
{
  "ietf-interfaces:interface": {
    "name": "GigabitEthernet0/0/1",
    "description": "Link to PC",
    "type": "iana-if-type:ethernetCsmacd",
    "enabled": true,
    "ietf-ip:ipv4": {
      "address": [
        {
          "ip": "192.168.1.1",
          "netmask": "255.255.255.0"
        }
      ]
    },
    "ietf-ip:ipv6": {}
  }
}
```

Part 5: Use Postman to Send a PUT Request

In Part 5, you will configure Postman to send a PUT request to R1 to create a new loopback interface.

Instructor Note: If you are in a lab environment with multiple students completing this lab at the same time, use a scheme to assign students a unique loopback interface and IPv4 address. The scheme should not conflict with the previous lab loopback assignments.

Student Name	Loopback	IPv4 Address
	200	10.200.1.1/24
	202	10.202.1.1/24
	204	10.204.1.1/24
	206	10.206.1.1/24

Note: If students proceed directly to other labs in this chapter, it is fine if these loopbacks still exist in the configuration.

Step 1: Duplicate and modify the last GET request.

Note: For this step, use the loopback interface information that was assigned to you by your instructor. If you were not assigned loopback information, and no other students are accessing your R1 router at the same time as you, then you can use the loopback information specified below.

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://192.168.1.1/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. Replace the text in brackets in the description field with your name. Remove the brackets. This will add your name to the interface description.

c. 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": "[Student Name]'s 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": {}
  }
}
```

d. Click **Send** to send the PUT request to R1. In the response field, you should get a **201 Created** HTTP response code. This indicates that the resource was created successfully.

e. You can verify that the interface was created by entering **show ip interface brief** on R1. You can also run the Postman tab that contains the request to get information about the interfaces on R1 that was created in Part 4 of this lab.

```
R1# show ip interface brief
Any interface listed with OK? value "NO" does not have a valid configuration

Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0/0   unassigned      YES unset  administratively down down
GigabitEthernet0/0/1   192.168.1.1     YES manual up                    up
Serial0/1/0            unassigned      NO  unset  down                  down
Serial0/1/1            unassigned      NO  unset  down                  down
GigabitEthernet0       unassigned      YES unset  administratively down down
Loopback1              10.1.1.1        YES other  up                    up
```

Part 6: Use a Python script to Send GET Requests

In Part 6, you will create a Python script to send GET requests to R1.

Step 1: Import modules and disable SSL warnings.

a. Open IDLE. Then click **File > New File** to open IDLE Editor.

b. Save the file as **restconf-get.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()
```

The **json** module includes methods to 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: Create the variables that will be the components of the request.

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

a. Create a variable named **api_url** and assign it the URL that will access the interface information on R1.

```
api_url = "https://192.168.1.1/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 R1. 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 R1.

```
================= RESTART: /home/student/restconf-script.py =================
<Response [200]>
>>>
```

Step 4: Format and display the JSON data received from R1.

Now the YANG model response values can be extracted 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:

```
================= RESTART: /home/student/restconf-script.py =================
<Response [200]>
{'ietf-interfaces:interfaces': {'interface': [{'name': 'GigabitEthernet0', 'type':
'iana-if-type:ethernetCsmacd', 'enabled': False, 'ietf-ip:ipv4': {}, 'ietf-ip:ipv6':
{}}, {'name': 'GigabitEthernet0/0/0', 'type': 'iana-if-type:ethernetCsmacd',
'enabled': False, 'ietf-ip:ipv4': {}, 'ietf-ip:ipv6': {}}, {'name':
'GigabitEthernet0/0/1', 'description': 'Link to PC', 'type': 'iana-if-
type:ethernetCsmacd', 'enabled': True, 'ietf-ip:ipv4': {'address': [{'ip':
'192.168.1.1', 'netmask': '255.255.255.0'}]}, 'ietf-ip:ipv6': {}}, {'name':
'Loopback1', 'description': "[Student's Name] 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': {}}]}}
>>>
```

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.

Note: The following output is truncated to display only interfaces with a configuration.

```
================= RESTART: /home/student/restconf-script.py =================
<Response [200]>
{
    "ietf-interfaces:interfaces": {
        "interface": [
(output omitted)
                "name": "GigabitEthernet0/0/1",
                "description": "Link to PC",
                "type": "iana-if-type:ethernetCsmacd",
                "enabled": true,
                "ietf-ip:ipv4": {
                    "address": [
                        {
                            "ip": "192.168.1.1",
                            "netmask": "255.255.255.0"
                        }
                    ]
                },
                "ietf-ip:ipv6": {}
            },
            {
                "name": "Loopback1",
                "description": "[Student's Name] 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": {}
            }
        ]
    }
}
```

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

In Part 7, you will create a Python script to send a PUT request to R1. As was done in Postman, you will create a new loopback interface.

Instructor Note: If you are in a lab environment with multiple students completing this lab at the same time, they should each increment the Loopback number and 2nd octet of the IP address assigned to them in Part 5.

Step 1: Import modules and disable SSL warnings.

Note: For this step, use the loopback interface information that was assigned to you by your instructor. Add 1 to the interface number and the 2nd octet of the IP address. For example, if you were assigned Loopback 200 and 10.200.1.1/24 for configuring the loopback address in Postman, use Loopback 201 and 10.201.1.1/24 for this step. If you were not assigned loopback information and no other students are accessing your R1 router at the same time as you, then you can use the loopback information specified below.

a. Open IDLE. Then click **File > New File** to open IDLE Editor.

b. Save the file as **restconf-put.py**.

c. Enter the following commands to import the modules 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.

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

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://192.168.1.1/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 new interface Loopback2. You can use the same dictionary that you used in Part 5 for 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": "[Student\’s Name] loopback interface",
        "type": "iana-if-type:softwareLoopback",
        "enabled": True,
        "ietf-ip:ipv4": {
            "address": [
                {
                    "ip": "10.2.1.1",
                    "netmask": "255.255.255.0"
                }
            ]
        },
        "ietf-ip:ipv6": {}
    }
}
```

Note: If using an apostrophe in the description text string, then you must escape it with the backslash, as shown above.

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 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 actually 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. Run the script to send the PUT request to R1. You should get a **201 Status Created** message. If not, check your code and the configuration for R1.

d. You can verify that the interface was created by entering **show ip interface brief** on R1.

```
R1# show ip interface brief
Any interface listed with OK? value "NO" does not have a valid configuration

Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0/0   unassigned      YES unset  administratively down down
GigabitEthernet0/0/1   192.168.1.1     YES manual up                    up
Serial0/1/0            unassigned      NO  unset  down                  down
Serial0/1/1            unassigned      NO  unset  down                  down
GigabitEthernet0       unassigned      YES unset  administratively down down
Loopback1              10.1.1.1        YES other  up                    up
Loopback2              10.2.1.1        YES other  up                    up
```

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://192.168.1.1/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))

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

api_url = "https://192.168.1.1/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": "[Student\'s Name] loopback interface",
        "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()))
```

Router Interface Summary Table

Router Model	Ethernet Interface #1	Ethernet Interface #2	Serial Interface #1	Serial Interface #2
1800	Fast Ethernet 0/0 (F0/0)	Fast Ethernet 0/1 (F0/1)	Serial 0/0/0 (S0/0/0)	Serial 0/0/1 (S0/0/1)
1900	Gigabit Ethernet 0/0 (G0/0)	Gigabit Ethernet 0/1 (G0/1)	Serial 0/0/0 (S0/0/0)	Serial 0/0/1 (S0/0/1)
2801	Fast Ethernet 0/0 (F0/0)	Fast Ethernet 0/1 (F0/1)	Serial 0/1/0 (S0/1/0)	Serial 0/1/1 (S0/1/1)
2811	Fast Ethernet 0/0 (F0/0)	Fast Ethernet 0/1 (F0/1)	Serial 0/0/0 (S0/0/0)	Serial 0/0/1 (S0/0/1)
2900	Gigabit Ethernet 0/0 (G0/0)	Gigabit Ethernet 0/1 (G0/1)	Serial 0/0/0 (S0/0/0)	Serial 0/0/1 (S0/0/1)
4221	Gigabit Ethernet 0/0/0 (G0/0/0)	Gigabit Ethernet 0/0/1 (G0/0/1)	Serial 0/1/0 (S0/1/0)	Serial 0/1/1 (S0/1/1)
4300	Gigabit Ethernet 0/0/0 (G0/0/0)	Gigabit Ethernet 0/0/1 (G0/0/1)	Serial 0/1/0 (S0/1/0)	Serial 0/1/1 (S0/1/1)

Note: To find out how the router is configured, look at the interfaces to identify the type of router and how many interfaces the router has. There is no way to effectively list all the combinations of configurations for each router class. This table includes identifiers for the possible combinations of Ethernet and Serial interfaces in the device. The table does not include any other type of interface, even though a specific router may contain one. An example of this might be an ISDN BRI interface. The string in parenthesis is the legal abbreviation that can be used in Cisco IOS commands to represent the interface.

Device Configs – Final

Router R1

```
R1# show run
Building configuration...


Current configuration : 2080 bytes
!
version 16.9
service timestamps debug datetime msec
service timestamps log datetime msec
```

```
platform qfp utilization monitor load 80
no platform punt-keepalive disable-kernel-core
!
hostname R1
!
boot-start-marker
boot-end-marker
!
no aaa new-model
!
no ip domain lookup
ip domain name example.netacad.com
ip dhcp excluded-address 192.168.1.1 192.168.1.10
!
ip dhcp pool LAN
 network 192.168.1.0 255.255.255.0
 default-router 192.168.1.1
 domain-name example.netacad.com
!
login on-success log
!
subscriber templating
!
multilink bundle-name authenticated
!
spanning-tree extend system-id
!
restconf
!
username cisco privilege 15 password 0 cisco123!
!
redundancy
 mode none
!
interface Loopback1
 description My Loopback
 ip address 10.1.1.1 255.255.255.0
!
interface Loopback2
 description My loopback interface
 ip address 10.2.1.1 255.255.255.0
!
interface GigabitEthernet0/0/0
 no ip address
 shutdown
 negotiation auto
!
interface GigabitEthernet0/0/1
 description Link to PC
 ip address 192.168.1.1 255.255.255.0
 negotiation auto
!
interface Serial0/1/0
 no ip address
 shutdown
!
interface Serial0/1/1
 no ip address
 shutdown
!
ip forward-protocol nd
ip http server
ip http authentication local
ip http secure-server
!
control-plane
!
line con 0
 exec-timeout 0 0
 logging synchronous
 transport input none
 stopbits 1
line aux 0
 stopbits 1
line vty 0 4
 exec-timeout 0 0
 logging synchronous
 login local
 transport input ssh
line vty 5 15
 exec-timeout 0 0
 logging synchronous
 login local
 transport input ssh
!
end
```