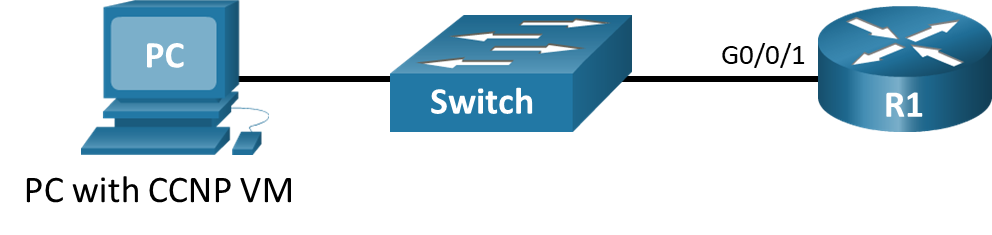
Lab - Use the Netmiko Python Module to Configure a Router

# 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: Import Netmiko Python Module

Part 3: Use Netmiko to Connect to the SSH Service

Part 4: Use Netmiko to Send Verification Commands

Part 5: Use Netmiko to Send and Verify a Configuration

Part 6: Use Netmiko to Send an Erroneous Command

Part 7: Modify the Program Used in this Lab

# Background / Scenario

With the evolution of the Python language, the **netmiko** Python module has emerged as an open source project hosted and maintained on GitHub. In this lab activity, you will use **netmiko** in a Python script to configure and verify a router.

# 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 and terminal emulation program)
* Ethernet cables as shown in the topology

# Instructions

## Build the Network and Verify Connectivity

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

### Cable the network as shown in the topology.

Connect the devices as shown in the topology diagram.

### Start the CCNP VM.

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

* + - 1. Open VirtualBox and start the **CCNP VM** virtual machine.
      2. Enter the password **StudentPass** to access the Ubuntu desktop if necessary

### Configure R1.

Console into R1 and apply the following configuration to configure basic settings and enable NETCONF, RESTCONF, and SSH.

Open configuration window

enable

configure terminal

hostname R1

no ip domain lookup

line con 0

logging synchronous

exec-timeout 0 0

logging synchronous

line vty 0 15

exec-t 0 0

logg sync

login local

transport input ssh

ip domain name example.netacad.com

crypto key generate rsa modulus 2048

username cisco priv 15 password cisco123!

interface GigabitEthernet0/0/1

description Link to PC

ip address 192.168.1.1 255.255.255.0

no shutdown

ip dhcp excluded-address 192.168.1.1 192.168.1.10

!Configure a DHCP server to assign IPv4 addressing to the CCNP VM

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

Close configuration window

### Verify the CCNP VM can ping the default gateway.

* + - 1. In the **CCNP VM**, open a terminal.
      2. Verify the CCNP VM is connected to R1 in a terminal window by either entering **ip address** to verify the CCNP VM received IP addressing from the DHCP server, or simply by pinging R1 at 192.168.1.1. Enter **Ctrl+C** to break out of the ping, as shown in the example output below.

Open terminal window

student@CCNP:~$ **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@CCNP:~$ **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

close terminal window

* + - 1. 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.

### Establish an SSH connection to R1.

* + - 1. Open the PuTTY SSH Client.
      2. Enter the IPv4 address for the default gateway, 192.168.1.1, and click **Open**.
      3. 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.
      4. Terminate your SSH session.

## Import Netmiko Python Module

In Part 2, you will install the **netmiko** module into your Python environment. Netmiko uses an SSH connection to access network devices. It has built in functionality to execute verification commands and apply new commands to the running configuration.

Explore the netmiko module on the project GitHub repository: https://github.com/ktbyers/netmiko.

In the **CCNP VM**, start IDLE and verify that netmiko is installed by importing it as shown.

Open configuration window

Python 3.6.9 (default, Nov 7 2019, 10:44:02)

[GCC 8.3.0] on linux

Type "help", "copyright", "credits" or "license()" for more information.

>>> **import netmiko**

## Use Netmiko to Connect to the SSH Service

In Part 3, you will use netmiko to connect to the SSH service running on R1.

### Create a new script file.

In IDLE, select **File > New**.

Save the file as **netmiko-script.py**.

### Create a variable with the SSH attributes.

The netmiko module includes the **ConnectHandler()** function. This function requires the following parameters for establishing an SSH connection with the IOS XE device:

* device\_type - identifies the remote device type
* host - the address (host or IP) of the remote device
* port - the remote port of the ssh service
* username - remote ssh username
* password - remote ssh password
  + - 1. In your **netmiko-script.py**, import netmiko’s **ConnectHandler()** function.

from netmiko import ConnectHandler

* + - 1. Enter the following information for the **ConnectHandler()** function.

sshCli = ConnectHandler(

device\_type = 'cisco\_ios',

host = '192.168.1.1',

port = 22,

username = 'cisco',

password = 'cisco123!'

)

* + - 1. Save the script and run it. You will see the following output if your script did not have an error:

================== RESTART: /home/student/netmiko-script.py ==================

>>>

## Use Netmiko to Send a Verification Command

In Part 4, you will use your Python script to send verification commands to router R1.

### Use the netmiko function send\_command to send a command through the SSH session.

* + - 1. Set a variable to hold the output of the **show** command you are sending. Use the **send\_command** function of the **sshCli** object, which is the SSH session previously established, to send the desired command. In this case, we are sending **sh ip int br**. Notice that the command does not have to be the full command. It can be any command that the IOS CLI would accept.

output = sshCli.send\_command("sh ip int br")

* + - 1. Run the program. You will see the following output if your script did not have an error:

================== RESTART: /home/student/netmiko-script.py ==================

>>>

### Print and format the content of the output variable.

* + - 1. The returned content from the function is stored in the **output** variable. In the interactive interpreter, enter **output** to see the content of the variable as shown in the following:

================== RESTART: /home/student/netmiko-script.py ==================

>>> **output**

'Interface IP-Address OK? Method Status Protocol\nGigabitEthernet0/0/0 unassigned YES unset administratively down down \nGigabitEthernet0/0/1 192.168.1.1 YES manual up up \nSerial0/1/0 unassigned YES unset administratively down down \nSerial0/1/1 unassigned YES unset administratively down down \nGigabitEthernet0 unassigned YES unset administratively down down '

* + - 1. The content of the **output** variable can be made readable with the **format** option of the **print()** command. Also, the **“{}\n.”** is used here to add a blank line after the output is printed.

print("{}\n".format(output))

* + - 1. Run your program now and you should get the following result, which is similar to what you would get when you enter the command directly into the IOS CLI.

================== RESTART: /home/student/netmiko-script.py ==================

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 YES unset administratively down down

Serial0/1/1 unassigned YES unset administratively down down

GigabitEthernet0 unassigned YES unset administratively down down

## Use Netmiko to Send and Verify a Configuration

In Part 5, you will send configuration commands to create a new loopback interface on R1 and then verify that the interface was created.

### Create and send a list of configuration commands to R1.

* + - 1. Add the following **config\_commmands** list variable to your **netmiko-script.py**. If multiple students are accessing R1 at the same time, use the loopback assigned to you by your instructor. Otherwise, you can use loopback 1, as shown below.

**Note**: Replace [Student Name] with your name. Leave the \ (backslash) in the **description** command. The backslash escapes the apostrophe so that Python does not read it as a closing quote, but as an apostrophe.

config\_commands = [

'int loopback 1',

'ip add 10.1.1.1 255.255.255.0',

'description [Student Name]\'s loopback'

]

* + - 1. Create a new variable called **sentConfig** to hold the results. Then use the **send\_config\_set** function of the **sshCli** object to send the commands to R1.

sentConfig = sshCli.send\_config\_set(config\_commands)

### Print and format the content of the sentConfig variable.

* + - 1. Use the **print** function to format and display what is stored in **sentConfig**. Then, resend the **send\_command** function and repeat the **print** function for the **output** variable so that you can see the new loopback interface in the **show ip interface brief** output.

print("{}\n".format(sentConfig))

output = sshCli.send\_command("sh ip int br")

print("{}\n".format(output))

* + - 1. Save and run your program. You should get output similar to the following:

================== RESTART: /home/student/netmiko-script.py ==================

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 YES unset administratively down down

Serial0/1/1 unassigned YES unset administratively down down

GigabitEthernet0 unassigned YES unset administratively down down

config term

Enter configuration commands, one per line. End with CNTL/Z.

R1(config)#int loopback 1

R1(config-if)#ip add 10.1.1.1 255.255.255.0

R1(config-if)#description [Student Name]'s loopback

R1(config-if)#end

R1#

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 YES unset administratively down down

Serial0/1/1 unassigned YES unset administratively down down

GigabitEthernet0 unassigned YES unset administratively down down

Loopback1 10.1.1.1 YES manual up up

## Use Netmiko to Send an Erroneous Command

Netmiko works similarly to copying and pasting a configuration script into the router CLI. Therefore, like when you paste a script in, netmiko will execute every command that it can. If a command fails, it will continue with the next command. Other automation tools typically will not apply any configuration changes if one or more commands are rejected. In Part 6, you will use netmiko to configure a new loopback address with a duplicate IPv4 address.

### Create a new loopback interface with the same IPv4 address.

Copy the **config\_commands** variable to the bottom of your **netmiko-script.py** script and modify it to store a new loopback interface that uses the same IPv4 address as the previous loopback interface. If multiple students are accessing R1 at the same time, add 1 to the number of the loopback assigned to you by your instructor. Otherwise, you can use loopback 2, as shown below.

config\_commands = [

'int loopback 2',

'ip add 10.1.1.1 255.255.255.0',

'description [Student Name]\'s loopback'

]

### Print and format the content of the sentConfig and output variables.

* + - 1. Send the commands and print the output like you did for the first loopback interface.

sentConfig = sshCli.send\_config\_set(config\_commands)

print("{}\n".format(sentConfig))

output = sshCli.send\_command("sh ip int br")

print("{}\n".format(output))

* + - 1. Save and run your program. You should get output similar to the output below. Notice that the new loopback interface was configured. However, it is not active because the IPv4 address duplicates the previous loopback interface.

================== RESTART: /home/student/netmiko-script.py ==================

(output from Part 4 and Part 5 omitted)

config term

Enter configuration commands, one per line. End with CNTL/Z.

R1(config)#int loopback 2

R1(config-if)#ip add 10.1.1.1 255.255.255.0

% 10.1.1.0 overlaps with Loopback1

R1(config-if)#description [Student Name]'s loopback

R1(config-if)#end

R1#

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 YES unset administratively down down

Serial0/1/1 unassigned YES unset administratively down down

GigabitEthernet0 unassigned YES unset administratively down down

Loopback1 10.1.1.1 YES manual up up

Loopback2 unassigned YES unset up up

Close configuration window

## Modify the Program Used in this Lab

The following is the complete program used in this lab. Practice your Python skills by modifying the program to send different verification and configuration commands.

from netmiko import ConnectHandler

sshCli = ConnectHandler(

device\_type = 'cisco\_ios',

host = '192.168.1.1',

port = 22,

username = 'cisco',

password = 'cisco123!'

)

output = sshCli.send\_command("sh ip int br")

print("{}\n".format(output))

config\_commands = [

'int loopback 1',

'ip add 10.1.1.1 255.255.255.0',

'description [Student Name]\'s loopback'

]

sentConfig = sshCli.send\_config\_set(config\_commands)

print("{}\n".format(sentConfig))

output = sshCli.send\_command("sh ip int br")

print("{}\n".format(output))

config\_commands = [

'int loopback 2',

'ip add 10.1.1.1 255.255.255.0',

'description [Student Name]\'s loopback'

]

sentConfig = sshCli.send\_config\_set(config\_commands)

print("{}\n".format(sentConfig))

output = sshCli.send\_command("sh ip int br")

print("{}\n".format(output))

# 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.

end of document