Master of Engineering in Internetworking

INWK 6312

Programming for Internetworking

Lab 5

Network Automation using Netmiko

Contents

Figures

Introduction

Netmiko is a multi-vendor SSH Python library which makes connecting to network devices via SSH easy and straightforward. The library simplifies SSH management to network devices is based on the Paramiko SSH library.

As per the documentation, the purposes of this library are the following:

- Successfully establish an SSH connection to the device
- Simplify the execution of show commands and the retrieval of output data
- Simplify execution of configuration commands including possibly commit actions
- Do the above across a broad set of networking vendors and platforms

Why Netmiko?

The question is: if we have Paramiko, why would we need a library like Netmiko. As it turns out, Paramiko can sometimes be quite hard to work with. Netmiko does hide many details of common device communication. It uses Paramiko for the underlying SSH connectivity but provides a higher abstraction for the communication with networking devices.

Netmiko Source code: https://ktbyers.github.io/netmiko/docs/netmiko/index.html

Objectives

In this lab, you will be using the Netmiko library to interact with Cisco IOS devices. You will make a connection, retrieve device information, and make push new configurations to the devices.

Lab Environment and Preparation

You will be connecting to a Linux VM and executing your code in a Python3 virtual environment.

The Linux VM Address is in the format 10.0.134.1**YY**. Where **YY** is the POD number. For example, POD 1 is 10.0.132.1**01** and Pod 56 is 10.0.132.1**56**.

Username: student
Password: Meilab123

- Connect to the Linux VM for your Pod.

```
$ ssh student@10.0.134.1YY
$ cd Desktop
```

- Start and activate a new virtual environment called `netmiko-lab`

```
$ python3 -m venv netmiko-lab
```

```
$ cd netmiko-lab
$ source bin/activate
```

- Install Netmiko, Jinja2, and Pyyaml libraries

```
(netmiko-lab)$ pip3 install netmiko
(netmiko-lab)$ pip3 install jinja2
(netmiko-lab)$ pip3 install pyyaml
```

- To check libraries installed on the virtual environment

```
(netmiko-lab)$ pip3 freeze
```

- To save installed library into a text file:

```
(netmiko-lab)$ pip3 freeze > requirements.txt
```

- Create a new folder: `src` and change directory into it:

```
(netmiko-lab)$ mkdir src && cd src
```

- Clone and enable TextFSM

```
(netmiko-lab)$ git clone https://github.com/networktocode/ntc-templates
(netmiko-lab)$ export NET_TEXTFSM=ntc-templates/ntc_templates/templates
```

Task 0 – Configure Management Interfaces

Before you can automate a device, you must configure an entry point to make a connection to that device. That entry point is also known as the Management interface. In this task, you will configure the Management interfaces of all the devices in our network.

When you're done with this task, you should be able to ping from the Linux VM to all the management interfaces of all devices. If you're unable to, troubleshoot before you continue to the remaining tasks.

Refer to **Appendix 1A and 1B** to configure the Management interfaces

Task 1 – Get User Prompt

Create a new Python file: **`get_prompt.py`** and write the following code snippet:

```
from netmiko import Netmiko

device = {
    "device_type": "cisco_ios",
    "ip": "192.168.1.101", # R1 Mgmt Interface
    "username": "student",
    "password": "Meilab123",
```

```
    "port": "22",
    }

net_connect = Netmiko(**device)
print(net_connect.find_prompt())
```

Execute the script:

```
(netmiko-lab)$ python3 get_prompt.py
 R01#
```

You'll notice that we are in enable mode.

The next code snippet will show how to issue the `disable` command while printing out the prompt. After which we will issue the `enable` command again and print out the prompt.

Update your script to include the following:

```
from netmiko import Netmiko
device = {
    "device_type": "cisco_ios",
    "ip": "192.168.1.101", # Router 1
    "username": "student",
    "password": "Meilab123",
    "secret": "cisco",
    "port": "22",
    }

net_connect = Netmiko(**device)
print(f"Default prompt: {net_connect.find_prompt()}")

net_connect.send_command_timing("disable")
print(f"Disable command: {net_connect.find_prompt()}")

net_connect.enable()
print(f"Enable command: {net_connect.find_prompt()}")
```

Note that in order for the `enable()` method to work, you need to pass the secret key in the device information. If not, you will receive an error!

Execute the script:

```
(netmiko-lab)$ python3 get_prompt.py
Default prompt: R01#
Disable command: R01>
Enable command: R01#
```

Questions:

1. Modify your script to get the prompt from all the devices in the topology

Task 2 – Retrieve Device Uptimes

We will now write another Python script that interacts with multiple routers and returns the uptime. We store all devices (routers) in a Python *list of dictionaries*. This allows us to loop over the `devices` list and execute the command for each device.

Within the for-loop, we first make a connection via Netmiko's connection method (called `Netmiko`). This is also the reason why we import the Netmiko library at the top of the script

Executing commands are easy. All we need to do is pass the command we want to execute to the `send_command` method and capture the output.

retrieve_uptimes.py

```
from netmiko import Netmiko

devices = [{"device_type": "cisco_ios",
            "ip": "192.168.1.101",
            "username": "student",
            "password": "Meilab123",
            "port": "22"},
           {"device_type": "cisco_ios",
            "ip": "192.168.1.102",
            "username": "student",
            "password": "Meilab123",
            "port": "22",
            }]

for device in devices:
  net_connect = Netmiko(**device)
  output = net_connect.send_command("show version")
  net_connect.disconnect()
  result = output.find('uptime is')
  begin = int(result)
  end = begin + 38
  print(device['ip'] + " => " + output[int(begin):int(end)])
```

Execute your script. Notice how we use Python's *String* slicing technique to extract the necessary information that we need.

Questions:

1. Modify this script to find the uptimes of all the routers in the topology.
2. Modify this script to extract the Configuration Register of the devices

Task 3 – Get Interface Description

In this example, we will retrieve the description of the various interfaces on all routers. We will be using a slightly different method compared to the task above. We will not create a list of devices, but we just define each device separately. Also note that we are importing the `ConnectHandler` class. This is just to show you the different ways you can write your script.

get_interface_description.py

```
from netmiko import ConnectHandler

r1 = {"device_type": "cisco_ios",
      "ip": "192.168.1.101",
      "username": "student",
      "password": "Meilab123",
      "port": "22"}
r2 = {"device_type": "cisco_ios",
      "ip": "192.168.1.101",
      "username": "student",
      "password": "Meilab123",
      "port": "22"}
for device in (r1, r2):
  net_connect = ConnectHandler(**device)
  output = net_connect.send_command("show interface description")
  net_connect.disconnect()
  print("-"*100)
  print(output)
  print("-"*100)
```

Execute the script

Questions:

1. Modify your script to *show the interface description* of all the routers in the topology.
2. Modify your script to print any other `show` commands that you can think of.

Task 4 – Parsing with TextFSM

In this task, we will use **TextFSM** to retrieve the output in a form that makes it easier to parse.

To understand the importance of TextFSM, we will first print out the type of the output of the "`show ip interface brief`" command.

parsed_format.py

```
from netmiko import Netmiko
device = {"device_type": "cisco_ios",
          "ip": "192.168.1.101",
          "username": "student",
          "password": "Meilab123",
          "port": "22"}
net_connect = Netmiko(**device)
output = net_connect.send_command("show ip interface brief")
```

```
net_connect.disconnect()
print(type(output))
```

Execute the script

```
(netmiko-lab)$ python3 parsed_format.py
<class 'str'>
```

The output is of type *string*. This is nice, but it's rather unpractical to parse this output. What if we could receive the output as a data structure, such as a Python list? This is possible with TextFSM. It is a module that implements a template-based state machine for parsing semi-formatted text. A large collection of TextFSM templates for quite some network vendors can be found in the GitHub repository you cloned earlier.

Modify your script to reflect the following:

```
from netmiko import Netmiko
device = {"device_type": "cisco_ios",
          "ip": "192.168.1.101",
          "username": "student",
          "password": "Meilab123",
          "port": "22"}
net_connect = Netmiko(**device)
output = net_connect.send_command("show ip interface brief", use_textfsm=True)
net_connect.disconnect()
print(type(output))

for interface in output:
  print(interface['intf'])
```

Execute the script

```
(netmiko-lab)$ python3 parsed_format.py
<class 'list'>
GigabitEthernet1
GigabitEthernet2
GigabitEthernet3
```

As you will see, we get back a Python *list* which we can easily iterate on to print for example the interface names. This gives us more flexibility to work with the output.

Questions:

1. Modify your script to find all the interface of all the routers in the topology
2. Inspect the cloned TextFSM folder and see the accept commands that can be parsed with TextFSM.

3. Modify your script to parse the "`show ip route`" command. Print the "`protocol`", "`network`", "`distance`" and "`metric`" from the output

Task 5 – Set Interface Description

In the previous tasks, we have shown examples to execute some simple "`show`" commands, in other words to read information from the device. In this task, we will write a Python script to write information to the devices in our network.

We will change the *interface description* of a particular port. It follows the same approach as the previous tasks. We store the devices in a *list of dictionaries* and we loop over them in a for-loop. Inside the for-loop, we use the `send_config_set` method.

At a minimum, we need to pass the `config_commands`, which is an iterable containing all of the configuration commands. Hence, we have created a list called `description_config` which contains a list of all the commands we want to execute (in order).

It is important to understand that Netmiko will already bring you into the global state (`conf t`). Next, we will go into 'interface GigabitEthernet3' and then set the description.

`set_interface_description.py`:

```
from netmiko import Netmiko
devices = [{"device_type": "cisco_ios",
            "ip": "192.168.1.101",
            "username": "student",
            "password": "Meilab123",
            "port": "22",}]
description = 'Description set with Netmiko'
description_config = ["interface GigabitEthernet3",
                      f"description {description}"]

for device in devices:
    net_connect = Netmiko(**device)
    output = net_connect.send_config_set(description_config)
    print(output)
    net_connect.disconnect()
```

Execute the script and check your router to make sure the changes were reflected.

```
(netmiko-lab)$ python3 set_interface_description.py
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R01(config)#interface GigabitEthernet3
R01(config-if)#description Description set with Netmiko
R01(config-if)#end
R01#
```

Questions:

1. Modify your script to add a Loopback interface to your device.

Task 6 – Variant to 'Set interface description'

In the previous use case, we have read the various commands from a list we declared in our Python script. Netmiko also supports a method called `send_config_from_file` to read these commands from a file. It's similar to the script we used in previous task.

Create a new file called "**`changes.txt`**" and write the following:

```
interface GigabitEthernet3
description This is set via Netmiko from a text file
```

Create a new script called **`set_interface_description2.py`** and write the following:

```
from netmiko import Netmiko
import logging
devices = [{"device_type": "cisco_ios",
            "ip": "192.168.1.101",
            "username": "student",
            "password": "Meilab123",
            "port": "22",}]
for device in devices:
    net_connect = Netmiko(**device)
    output = net_connect.send_config_from_file('changes.txt')
    print(output)
    net_connect.disconnect()
```

Execute the script and check your router to make sure the changes were reflected.

```
(netmiko-lab)$ python3 set_interface_description2.py
configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
71R01(config)#interface GigabitEthernet3
71R01(config-if)#description This is set via Netmiko and text file
71R01(config-if)#end
71R01#
```

Task 7 – Add Loopback interfaces using Templates

In this task, we will use some of the knowledge we learned in previous labs. We will do the following:

- Read device information from a YAML file
- Read variables (containing loopback information) from a YAML file
- Use Jinja2 template
- Use Netmiko to configure the loopback interfaces

Create a YAML file called **`interfaces.yml`**:

```
interfaces:
  - name: Loopback201
    description: Description for Loopback 2000
    ipv4_addr: 201.201.201.201
    ipv4_mask: 255.255.255.255

  - name: Loopback202
    description: Description for Loopback 2001
    ipv4_addr: 202.202.202.202
    ipv4_mask: 255.255.255.255
```

As mentioned, this file contains the required information to configure the loopback interfaces.

We will also read the host information from another YAML file.

Create another YAML file called **`hosts.yml`**:

```
hosts:
  - name: 192.168.1.101
    username: student
    password: Meilab123
    port: 22
    cmd: "show running-config"
    type: cisco_ios

  - name: 192.168.1.102
    username: student
    password: Meilab123
    port: 22
    cmd: "show running-config"
    type: cisco_ios
```

Next, we will create a *Jinja2* file, which is relative straightforward as well. Essentially, we use a for-loop to iterate over the loopback interfaces. Each time, we use the data that is passed to this template from the Python file.

Create a file called **`interfaces_config_template.j2`**:

```
{% for interface in data.interfaces %}
 interface {{interface.name}}
 description {{interface.description}}
 ip address {{interface.ipv4_addr}} {{interface.ipv4_mask}}
{% endfor %}
```

We will now write our Python Script that connects everything together. In a new file called **`set_loopback_template.py`**, wrsite the following code:

```
import yaml
from netmiko import Netmiko
from jinja2 import Environment, FileSystemLoader
```

```
hosts = yaml.load(open('hosts.yml'), Loader=yaml.SafeLoader)
interfaces = yaml.load(open('interfaces.yml'), Loader=yaml.SafeLoader)

env = Environment(loader = FileSystemLoader('.'), trim_blocks=True,
autoescape=True)
template = env.get_template('interfaces_config_template.j2')
loopback_config = template.render(data=interfaces)

for host in hosts["hosts"]:
    net_connect = Netmiko(host = host["name"],
                          username = host["username"],
                          password = host["password"],
                          port = host["port"],
                          device_type = host["type"])
    print(f"Logged into {host['name']} successfully")
    output = net_connect.send_config_set(loopback_config.split("\n"))
    print(f"Pushed config into {host['name']} successfully")
    net_connect.disconnect()

print("Done!")
```

The Python file is first reading in the host information from the `hosts.yml` file. The interface information is then read from the YAML file(`interfaces.yml`). The host and interface information are now available as Python *dictionaries*.

Next, we use the Jinja2 library to render the template with the variables. The result of this is that `loopback_config` contains the completed (absolute) file with all loopback information.

Next, we simply loop over the devices (hosts) and for each host we create an SSH connection via Netmiko. Finally, we use the `send_config_set` method to which we pass the `loopback_config` information (which contains nothing more than the commands to configure a loopback interface).

Execute the script

```
(netmiko-lab)$ python3 set_loopback_template.py
Logged into 192.168.1.101 successfully
Pushed config into 192.168.1.101 successfully
Logged into 192.168.1.102 successfully
Pushed config into 192.168.1.102 successfully
Done!
```

You can run the script "`get_interface_description.py`" from the previous tasks to see if the new loopback interfaces were added

Lab Exercise

Now that you have learnt the basics of Netmiko, in this exercise, you will use that knowledge to write a Python script that automates and configures a network Topology from one of your previous course labs.

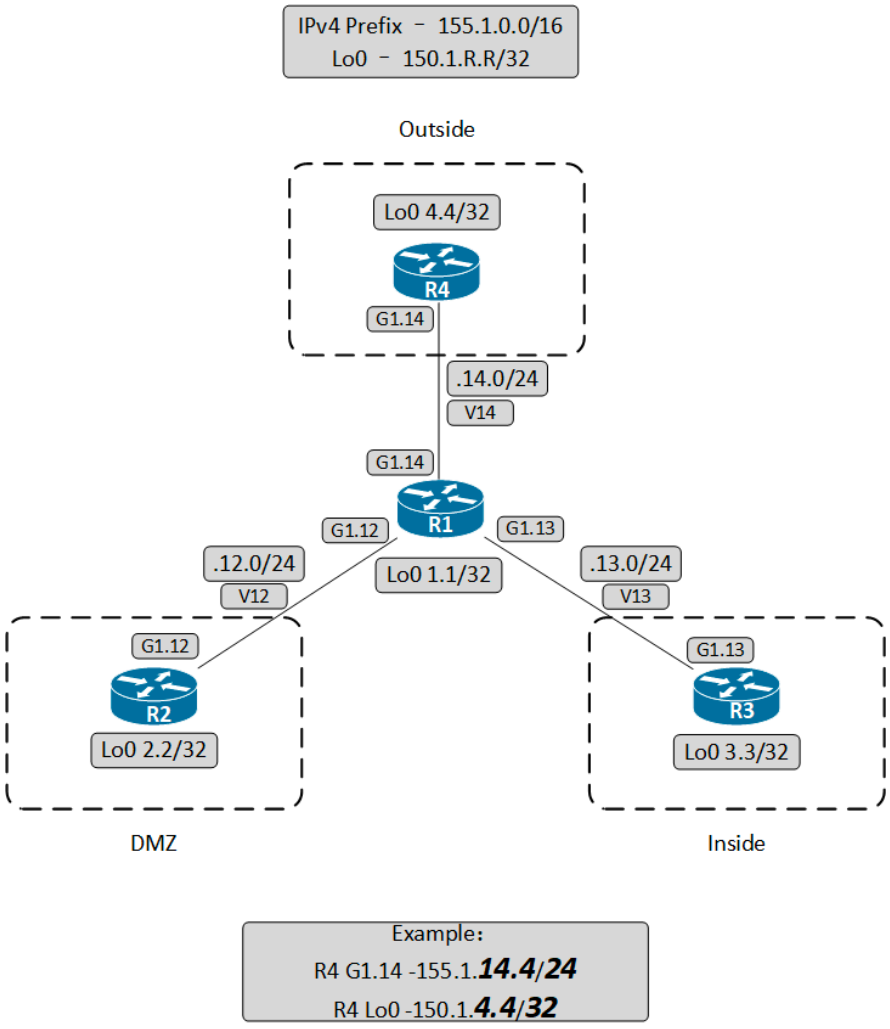

Figure 1. Network Topology for Lab Exercise

Questions:

1. Generate an appropriate YAML file that contains the necessary information about the network
2. Generate an appropriate JINJA file that contains the necessary configuration that will be pushed to the device. (See appendix B for configuration template)
3. Write a Python Program, that uses the YAML and JINJA files to push configure all the devices in the topology.
4. Improve your code to use Python Logger class to write errors and information to a log file.
5. Write another Python Script to collect the Routing Table from all routers using TextFSM.

Appendix 1A: Topology

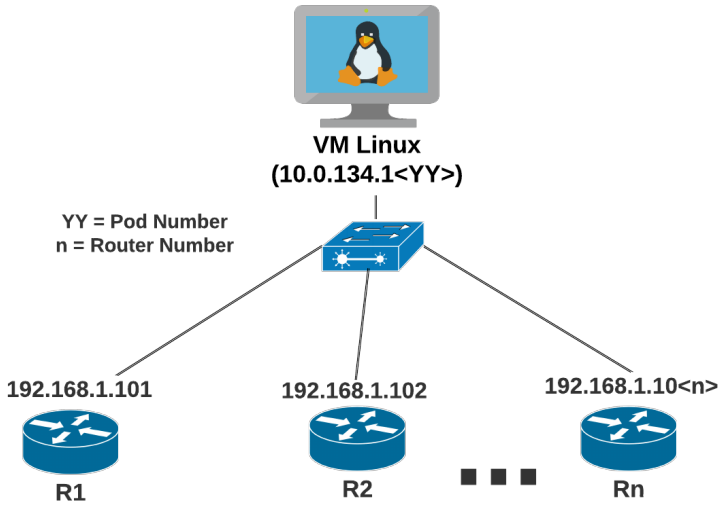

Figure 2. Network Topology showing connectivity of Management networks and Linux VM

Appendix 1B: Management Interface Configuration Template

For faster configuration, you can copy and paste the following command lines into the Cisco router after replacing the value of <X> with the router number in the two placeholders.

```
configure terminal
hostname R<X>
no ip domain lookup
enable password cisco
ip domain name inwk.local
username student privilege 15 secret Meilab123
interface gigabitEthernet 1
ip address 192.168.1.10<X> 255.255.255.0
no shutdown
ip ssh version 2
line con 0
logging synchronous
line vty 0 4
login local
transport input ssh
exit
```

Appendix 2A: Sample Configuration for Lab Exercise

use this configuration to generate configuration for the devices.

R1:

```
no ip domain lookup
hostname ???
line con 0
logging synchronous
exit
interface loopback 0
ip address 150.1.z.z 255.255.255.255
no shutdown
interface GigabitEthernet1
no shutdown
interface GigabitEthernet1.14
encapsulation dot1Q 14
ip address 155.1.x.x 255.255.255.0
no shutdown
interface GigabitEthernet1.13
encapsulation dot1Q 13
ip address 155.1.y.y 255.255.255.0
no shutdown
interface GigabitEthernet1.12
encapsulation dot1Q 12
ip address 155.1.q.q 255.255.255.0
no shutdown
router rip
version 2
network 155.1.
network 155.1.
network 155.1.
network 150.1.
no auto-summary
exit
line vty 0 4
password inwk
login
transport input all
exit
```

R2:

```
no ip domain lookup
hostname ???
line con 0
logging synchronous
exit
interface loopback 0

ip address 150.1.x.x 255.255.255.255
no shutdown
interface GigabitEthernet1
no shutdown
interface GigabitEthernet1.12
encapsulation dot1Q 12
ip address 155.1.y.y 255.255.255.0
no shutdown
router rip
version 2
network 155.1.
network 150.1.
no auto-summary
exit
line vty 0 4
password inwk
login
transport input all
exit
```

R3:

```
no ip domain lookup
hostname ???
line con 0
logging synchronous
exit
interface loopback 0
ip address 150.1.x.x 255.255.255.255
no shutdown
interface GigabitEthernet1
no shutdown
interface GigabitEthernet1.13
encapsulation dot1Q 13
ip address 155.1.y.y 255.255.255.0
no shutdown
router rip
```

R4:

```
no ip domain lookup
hostname ???
line con 0
logging synchronous
exit
interface loopback 0
ip address 150.1.x.x 255.255.255.255
no shutdown
interface GigabitEthernet1
no shutdown
interface GigabitEthernet1.14
encapsulation dot1Q 14
ip address 155.1.y.y 255.255.255.0
no shutdown
router rip
```

version 2 network 155.1. network 150.1. no auto-summary exit line vty 0 4 password inwk login transport input all exit	version 2 network 155.1. network 150.1. no auto-summary exit line vty 0 4 password inwk login transport input all exit`