

Lab – CLI Automation with Python using netmiko

Objectives

- Part 1: Install the netmiko Python module
- Part 2: Connect to IOS XE's SSH service using netmiko
- Part 3: Use netmiko to gather information from the device
- Part 4: Use netmiko to alter configuration on the device

Background / Scenario

For simple network automation using a remote telnet or ssh based command line, network administrators have been using various screen scraping techniques for a long period of time. Initially the “expect” based scripts we utilized to automate entering commands when a specific expected string appeared on the command line. With the evolution of the Python language, the netmiko Python module has emerged as an open source project hosted and maintained on GitHub.com that provides a simple network automation interface using similar techniques like the “expect” based scripts.

In this lab activity, you will identify the potential but also the limitations of using netmiko to transport CLI commands for network automation.

Required Resources

- Access to a router with the IOS XE operating system version 16.6 or higher.
- Access to the Internet
- Python 3.x environment

Part 1: Install the netmiko Python module

In this part, you will install netmiko module into your Python environment. Netmiko is a python module that simplifies ssh CLI connection to network devices. It has built in functionality to identify to execute “exec mode” commands, as well as apply new commands in the running configuration.

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

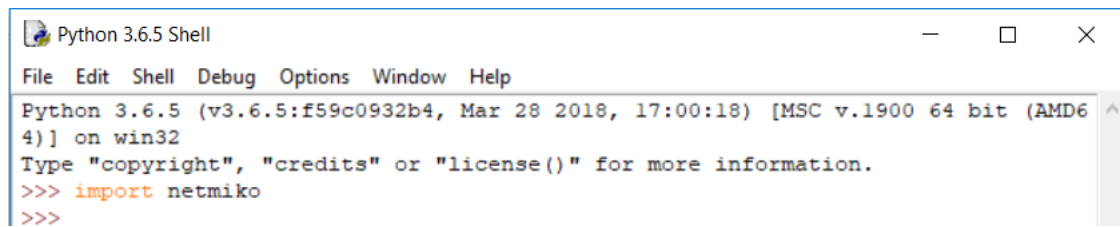
Step 1: Use pip to install netmiko.

- a. Start a new Windows command prompt (cmd).
- b. Install netmiko using pip in the Windows command prompt:

```
pip install netmiko
```

- c. Verify that netmiko has been successfully installed. Start Python IDLE and in the interactive shell try to import the netmiko module:

```
import netmiko
```



A screenshot of a Python 3.6.5 Shell window. The window title is "Python 3.6.5 Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area shows the following output and input:

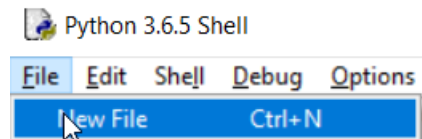
```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import netmiko
>>>
```

Part 2: Connect to IOS XE's SSH service using netmiko

Connect to IOS XE's SSH service using netmiko.

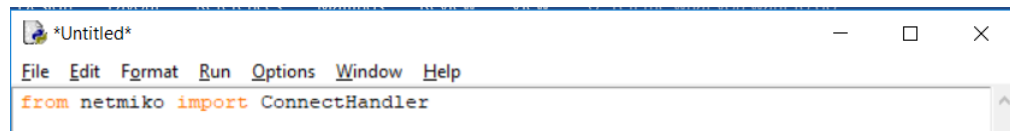
The netmiko module provides a `ConnectHandler()` function to setup the remote ssh connection. After a successful connection, the returned object represents the ssh cli connection to the remote device.

- a. In Python IDLE, create a new Python script file:



- b. In the new Python script file editor, import the `ConnectHandler()` function from the netmiko module:

```
from netmiko import ConnectHandler
```



- c. Setup a `sshCli` connection object using the `ConnectHandler()` function to the IOS XE device.

```
sshCli = ConnectHandler(  
    device_type='cisco_ios',  
    host='192.168.56.101',  
    port=22,  
    username='cisco',  
    password='cisco123!'  
)
```

The parameters of the `ConnectHandler()` function are:

- `device_type` – identifies the remote device type
- `host` – the address (host or IP) of the remote device (adjust the IP address “192.168.56.101” to match your router’s current address)
- `port` – the remote port of the ssh service
- `username` – remote ssh username (in this lab “cisco” for that was setup in the IOS XE VM)
- `password` – remote ssh password (in this lab “cisco123!” for that was setup in the IOS XE VM)

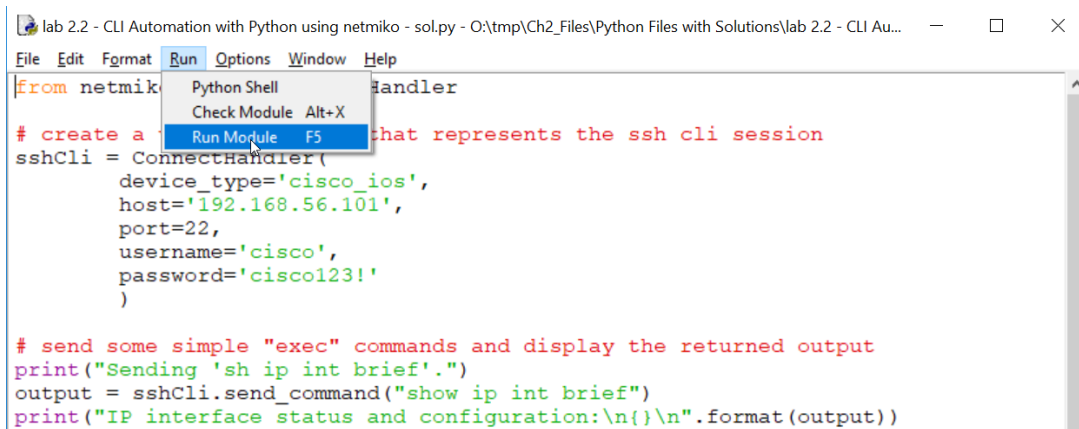
Part 3: Use netmiko to gather information from the device

Send show commands and display the output

- a. Using the `sshCli` object, returned by the `ConnectHandler()` function that represents the ssh cli remote session, send some “show” command and print the output. Use the `send_command()` function of the `sshCli` object with a string parameter that represents the command you wish to execute in the exec mode:

```
output = sshCli.send_command("show ip int brief")
print("show ip int brief:\n{}\n".format(output))
```

- b. Execute the Python script file to see the results:



```
lab 2.2 - CLI Automation with Python using netmiko - sol.py - O:\tmp\Ch2_Files\Python Files with Solutions\lab 2.2 - CLI Au...
File Edit Format Run Options Window Help
from netmiko Python Shell handler
Check Module Alt+X
Run Module F5
# create a ConnectHandler object that represents the ssh cli session
sshCli = ConnectHandler(
    device_type='cisco_ios',
    host='192.168.56.101',
    port=22,
    username='cisco',
    password='cisco123!'
)

# send some simple "exec" commands and display the returned output
print("Sending 'sh ip int brief'.")
output = sshCli.send_command("show ip int brief")
print("IP interface status and configuration:\n{}\n".format(output))
```

If you have not saved the script file yet, you will be prompted to save it before it is executed.

- c. Verify the results:

```
>>>
RESTART: O:\tmp\Ch2_Files\Python Files with Solutions\lab 2.2 - CLI Automation
with Python using netmiko - sol.py
Sending 'sh ip int brief'.
IP interface status and configuration:
Interface          IP-Address      OK? Method Status          Protocol
GigabitEthernet1   192.168.56.101 YES DHCP    up                up
>>>
```

- d. Verify the data type of the “output” variable. How would you extract the IP address and the Interface Name into variables? What is there were multiple interfaces?

Part 4: Use netmiko to alter configuration on the device

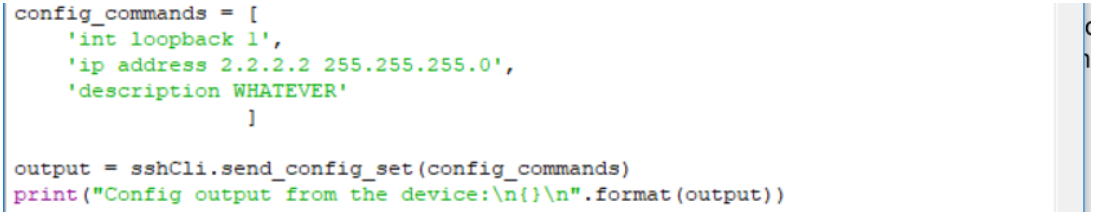
In the following steps, you will alter the configuration of the device by creating new loopback interfaces.

Create a new loopback interface

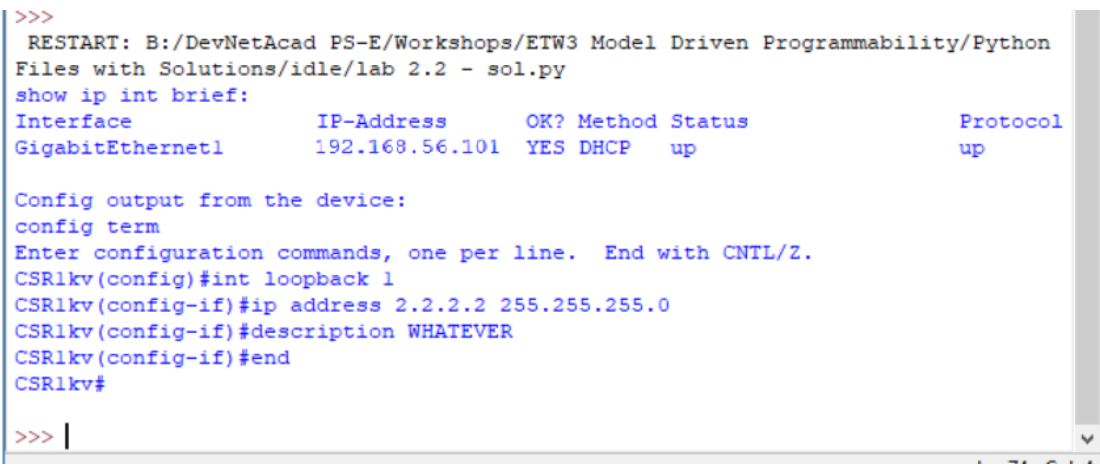
- a. Using the `sshCli` object, returned by the `ConnectHandler()` function that represents the ssh cli remote session, send some configuration command and print the output. Use the `send_config_set()` function of the `sshCli` object with a list parameter including the configuration commands as strings you wish to execute in the exec mode:

```
config_commands = [
    'int loopback 1',
    'ip address 2.2.2.2 255.255.255.0',
    'description WHATEVER'
]

output = sshCli.send_config_set(config_commands)
```



- b. Execute the Python script file and verify the results:



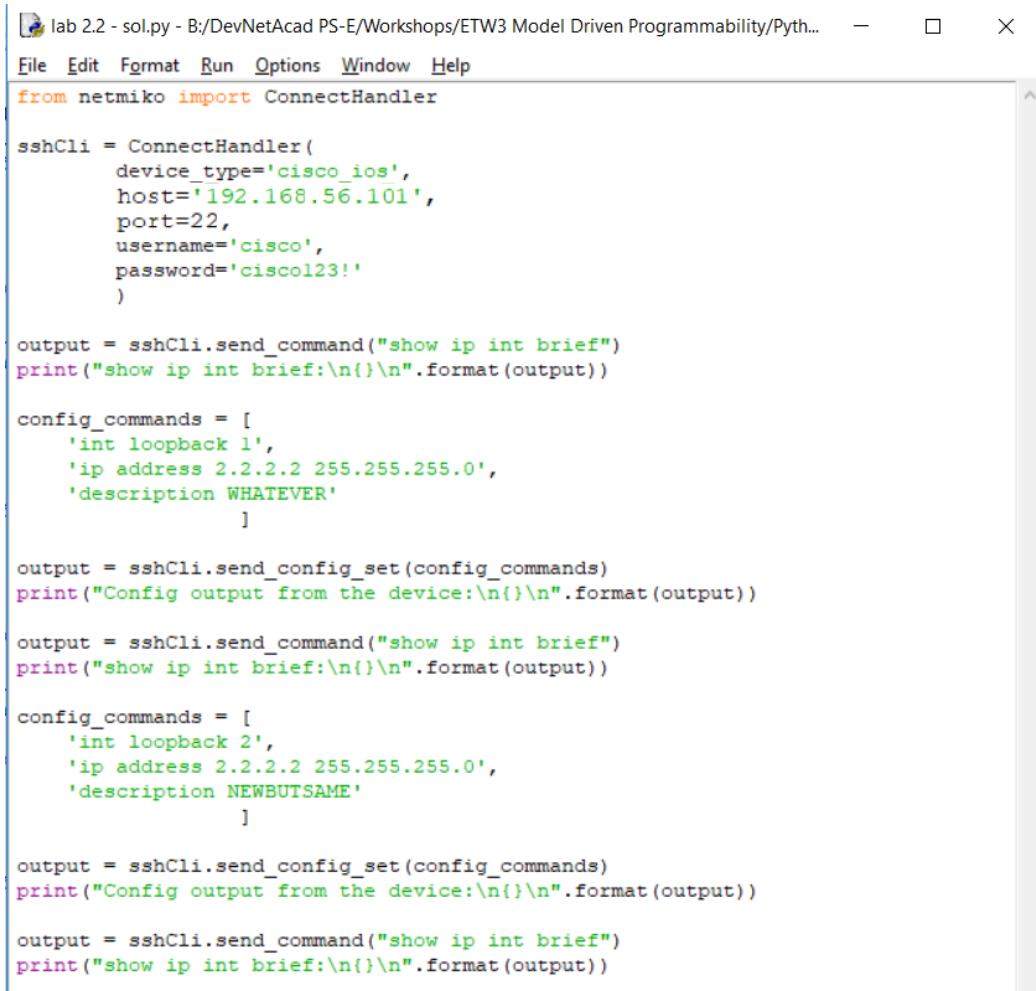
```
>>>
RESTART: B:/DevNetAcad PS-E/Workshops/ETW3 Model Driven Programmability/Python
Files with Solutions/idle/lab 2.2 - sol.py
show ip int brief:
Interface          IP-Address      OK? Method Status          Protocol
GigabitEthernet1   192.168.56.101 YES DHCP    up              up

Config output from the device:
config term
Enter configuration commands, one per line.  End with CNTL/Z.
CSR1kv(config)#int loopback 1
CSR1kv(config-if)#ip address 2.2.2.2 255.255.255.0
CSR1kv(config-if)#description WHATEVER
CSR1kv(config-if)#end
CSR1kv#

>>> |
```

- c. Why does the output from “show ip int brief” not include the “loopback1” interface?
- _____
- _____
- _____
- d. How to execute and display the output from the “show ip int brief” command after the loopback interfaces was created?
- _____
- _____
- _____
- e. Add code to create a new loopback interface (loopback2) with the same IP address as on the existing loopback interface, only with a different description.

- f. Execute the Python script file and verify the results:



```
lab 2.2 - sol.py - B:/DevNetAcad PS-E/Workshops/ETW3 Model Driven Programmability/Pyth...
File Edit Format Run Options Window Help

from netmiko import ConnectHandler

sshCli = ConnectHandler(
    device_type='cisco_ios',
    host='192.168.56.101',
    port=22,
    username='cisco',
    password='cisco123!'
)

output = sshCli.send_command("show ip int brief")
print("show ip int brief:\n{}\n".format(output))

config_commands = [
    'int loopback 1',
    'ip address 2.2.2.2 255.255.255.0',
    'description WHATEVER'
]

output = sshCli.send_config_set(config_commands)
print("Config output from the device:\n{}\n".format(output))

output = sshCli.send_command("show ip int brief")
print("show ip int brief:\n{}\n".format(output))

config_commands = [
    'int loopback 2',
    'ip address 2.2.2.2 255.255.255.0',
    'description NEWBUTSAME'
]

output = sshCli.send_config_set(config_commands)
print("Config output from the device:\n{}\n".format(output))

output = sshCli.send_command("show ip int brief")
print("show ip int brief:\n{}\n".format(output))
```

```

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help

>>>
RESTART: B:/DevNetAcad PS-E/Workshops/ETW3 Model Driven Programmability/Python
Files with Solutions/idle/lab 2.2 - sol.py
show ip int brief:
Interface          IP-Address      OK? Method Status          Protocol
GigabitEthernet1   192.168.56.101  YES DHCP    up              up
Loopback1          2.2.2.2         YES manual  up              up

Config output from the device:
config term
Enter configuration commands, one per line. End with CNTL/Z.
CSR1kv(config)#int loopback 1
CSR1kv(config-if)#ip address 2.2.2.2 255.255.255.0
CSR1kv(config-if)#description WHATEVER
CSR1kv(config-if)#end
CSR1kv#

show ip int brief:
Interface          IP-Address      OK? Method Status          Protocol
GigabitEthernet1   192.168.56.101  YES DHCP    up              up
Loopback1          2.2.2.2         YES manual  up              up

Config output from the device:
config term
Enter configuration commands, one per line. End with CNTL/Z.
CSR1kv(config)#int loopback 2
CSR1kv(config-if)#ip address 2.2.2.2 255.255.255.0
% 2.2.2.0 overlaps with Loopback1
CSR1kv(config-if)#description NEWBUTSAME
CSR1kv(config-if)#end
CSR1kv#

show ip int brief:
Interface          IP-Address      OK? Method Status          Protocol
GigabitEthernet1   192.168.56.101  YES DHCP    up              up
Loopback1          2.2.2.2         YES manual  up              up
Loopback2          unassigned      YES unset   up              up

>>> |
  
```

g. Was the new loopback2 interface successfully created?

h. Was the new configuration change accepted, partially accepted or rejected?
