EETS 8353 NETWORK AUTOMATION

NETWORK AUTOMATION WITH NETMIKO

PALAK GANDHI

48900571

SPRING 2023

SMU

Agenda

Introduction & Features

Netmiko is a powerful open-source library for network automation that simplifies the process of managing network devices such as routers, switches, and firewalls using Python.

- **Vendor-agnostic:** Netmiko supports a wide range of network devices from various vendors, making it a vendor-agnostic tool. This means that you can use the same code to interact with devices from different vendors, simplifying the automation process.

- **SSH and Telnet support:** Netmiko supports both SSH and Telnet protocols for connectivity to network devices. This allows you to use the most appropriate protocol for your environment.

KIRK BYERS

https://github.com/ktbyers/netmiko

Introduction & Features

- **Command execution:** Netmiko allows you to execute commands on network devices and retrieve the output. This can be used for various tasks such as retrieving device information, debugging, and troubleshooting.

- **Configuration management:** Netmiko allows you to manage device configurations by sending commands in configuration mode. This can be used to automate configuration changes, backups, and rollbacks.

- **File transfer:** Netmiko allows you to transfer files to and from network devices. This can be used to automate tasks such as software upgrades and configuration backups.

- **Advanced features:** Netmiko provides advanced features such as enabling and disabling interfaces, configuring VLANs, and sending multiple commands in a single session. This allows you to automate complex network management tasks.

https://github.com/ktbyers/netmiko

Installation & Setup

1. Build the Network and Verify Connectivity through SSH

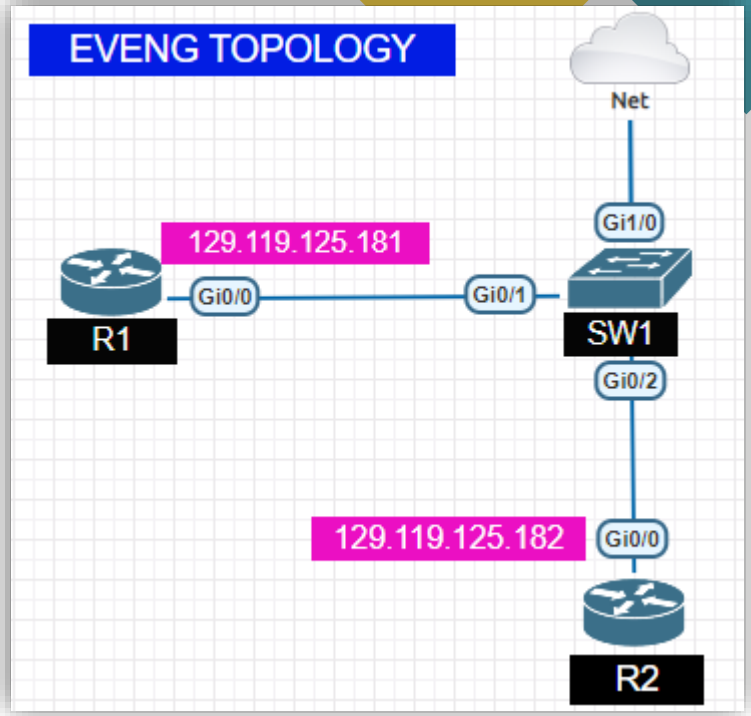
```
PS C:\Users\Navkar> ping 129.119.125.181

Pinging 129.119.125.181 with 32 bytes of data:
Reply from 129.119.125.181: bytes=32 time=26ms TTL=254
Reply from 129.119.125.181: bytes=32 time=21ms TTL=254
Reply from 129.119.125.181: bytes=32 time=30ms TTL=254
Reply from 129.119.125.181: bytes=32 time=34ms TTL=254

Ping statistics for 129.119.125.181:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 21ms, Maximum = 34ms, Average = 27ms
PS C:\Users\Navkar> ping 129.119.125.182

Pinging 129.119.125.182 with 32 bytes of data:
Reply from 129.119.125.182: bytes=32 time=25ms TTL=254
Reply from 129.119.125.182: bytes=32 time=24ms TTL=254
Reply from 129.119.125.182: bytes=32 time=37ms TTL=254
Reply from 129.119.125.182: bytes=32 time=28ms TTL=254

Ping statistics for 129.119.125.182:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 24ms, Maximum = 37ms, Average = 28ms
PS C:\Users\Navkar>
```



Installation & Setup

2. Import Netmiko Python Module

- Check Python version
- Check availability of Netmiko
- Install Netmiko using pip
- Verify Netmiko

```
PS C:\Users\Navkar>
PS C:\Users\Navkar> python --version
Python 3.10.7
PS C:\Users\Navkar>
PS C:\Users\Navkar> pip show netmiko
WARNING: Package(s) not found: netmiko
PS C:\Users\Navkar>
PS C:\Users\Navkar> pip install netmiko
Collecting netmiko
  Using cached netmiko-4.1.2-py3-none-any.whl (196 kB)
Requirement already satisfied: textfsm==1.1.2 in c:\use
Requirement already satisfied: scp>=0.13.3 in c:\users\
```

```
[notice] A new release of pip is available: 23.0.1 -> 23.1
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\Users\Navkar>
PS C:\Users\Navkar> pip show netmiko
Name: netmiko
Version: 4.1.2
Summary: Multi-vendor library to simplify legacy CLI connections to
Home-page: https://github.com/ktbyers/netmiko
Author: Kirk Byers
Author-email: ktbyers@twb-tech.com
License: MIT
Location: c:\users\navkar\appdata\local\programs\python\python310\l
Requires: ntc-templates, paramiko, pyserial, pyyaml, scp, setuptool
Required-by:
PS C:\Users\Navkar>
```

Installation & Setup

3. Create a dictionary representing the device.

   By creating this dictionary, the Netmiko library can use these values to establish a connection to the specified network device. The values for these keys will vary depending on the specific device. The supported devices can be found on:

   netmiko/PLATFORMS.md at develop · ktbyers/netmiko · GitHub

```
cisco_device = {
    'device_type': 'cisco_ios',
    'host': '129.119.125.181',
    'username': 'admin',
    'password': 'cisco',
    'port': 22,
}
```

Installation & Setup

3. Use Netmiko to Connect to the SSH Service

ConnectHandler is the main entry point into the library. It picks the right class, creates a Netmiko object based on that class, and establishes an SSH connection to the remote device.

We create an instance of the **ConnectHandler** class, passing in the **cisco_device** dictionary using the ** notation to unpack its contents. The resulting object, **connection**, represents the connection to the device.

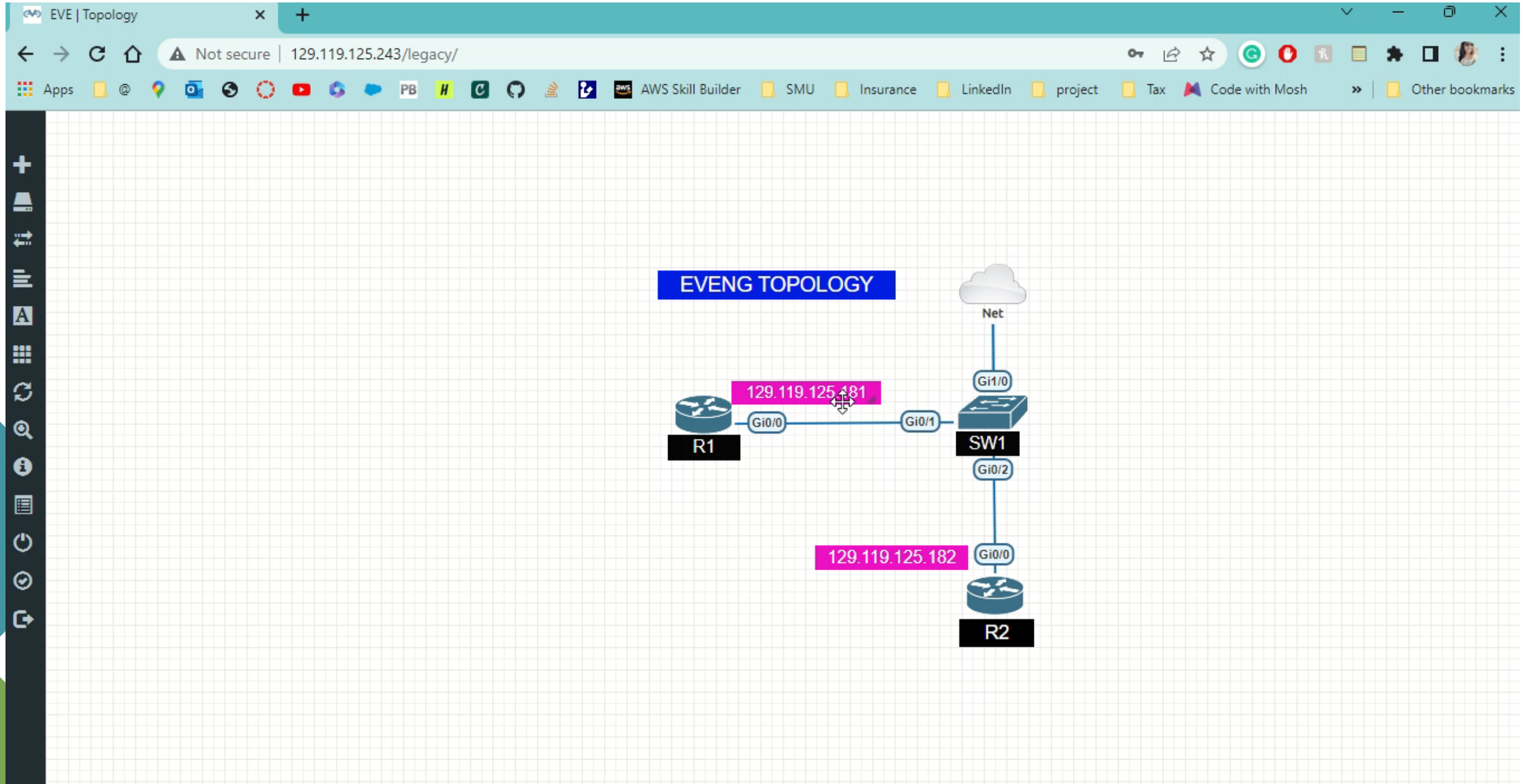
```
# creating an SSH connection to the Cisco device
connection = ConnectHandler(**cisco_device)
```

Use Cases

1. Check installation
2. Connect and verify using single command
3. Run multiple commands on single device
4. Configuration using txt file
5. Run multiple commands for multiple devices
6. How to backup
7. How to handle troubleshoot

Use Cases

1. Check installation



Use Cases

2. Connect and verify using single command

File Edit Selection View Go Run Terminal Help • 1_connect_run.py - NA_Project_2023 - Visual Studio Code

EXPLORER

NA_PROJECT_2023
- .vscode
- project-files
- 0_check_installati...
- 1_connect_run.py
- 2_multiple_comm...
- 3_using_file.py
- 4_multiple_comm...
- 5_automating_bac...
- 6_troubleshootin...
- devices.txt
- ospf.txt
- Router1.txt
- Router2.txt
- z_NA_Project_202...

0_check_installation.py | 1_connect_run.py | 2_multiple_commands.py | 3_using_file.py

1_connect_run.py > ...

```python
from netmiko import ConnectHandler

# creating a dictionary with device specification
cisco_device = {
    'device_type': 'cisco_ios',
    'host': '129.119.125.181',
    'username': 'admin',
    'password': 'cisco',
    'port': 22,                        # optional, default 22
}

# creating an SSH connection to the Cisco device
connection = ConnectHandler(**cisco_device)


# retrieving the running configuration
output = connection.send_command("sh run")
print(output)


# closing connection
print('Closing connection')
connection.disconnect()
```

OUTLINE

TIMELINE

0 0 Quokka Ln 24, Col 1 Spaces: 4 UTF-8 CRLF Python 3.10.7 64-bit Prettier

Use Cases

3. Run multiple commands on single device

2_multiple_commands.py - NA_Project_2023 - Visual Studio Code

```python
from netmiko import ConnectHandler

cisco_device = {
    'device_type': 'cisco_ios',
    'host': '129.119.125.181',
    'username': 'admin',
    'password': 'cisco',
    'port': 22
}

connection = ConnectHandler(**cisco_device)


# checking the prompt, if user in user-exac mode change to priviledge mode
prompt = connection.find_prompt()
if '>' in prompt:
    print('Entering the enable mode...')
    connection.enable()


# checking the prompt, and switching to global configuration mode.
if not connection.check_config_mode():
    connection.config_mode()


commands = ['int loopback 0', 'ip address 3.3.3.3 255.255.255.255',
            'exit']

# displaying the commands
output = connection.send_config_set(commands)

# priting the output
```

Use Cases

4. Configuration using txt file

3_using_file.py - NA_Project_2023 - Visual Studio Code

EXPLORER

NA_PROJECT_2023
- .vscode
- project-files
- 0_check_installati...
- 1_connect_run.py
- 2_multiple_comm...
- 3_using_file.py
- 4_multiple_comm...
- 5_automating_bac...
- 6_troubleshootin...
- devices.txt
- ospf.txt
- Router1.txt
- Router2.txt
- z_NA_Project_202...

0_check_installation.py | 1_connect_run.py | 2_multiple_commands.py | 3_using_file.py | ospf.txt

3_using_file.py > ...

```python
from netmiko import ConnectHandler

cisco_device = {
    'device_type': 'cisco_ios',
    'host': '129.119.125.181',
    'username': 'admin',
    'password': 'cisco',
    'port': 22
}

connection = ConnectHandler(**cisco_device)
print('Entering the enable mode...')
connection.enable


# using file, sending the commands
print('sending commands from file..')
output = connection.send_config_from_file('ospf.txt')
print(output)


# connection.send_command('write memory')
print('Closing connection')
connection.disconnect()
```

OUTLINE

TIMELINE

Use Cases

5. Run commands for multiple devices

4_multiple_commands_file.py - NA_Project_2023 - Visual Studio Code

```python
from netmiko import ConnectHandler

#opening a text file "device.txt" and reading it in a list
with open('devices.txt')as f:
    devices = f.read().splitlines()

#creating an empty list,containing a dictionary for each router
device_list = list()


# iterating each devices mentioned in the device.txt to create a list
for ip in devices:
    cisco_device = {
        'device_type': 'cisco_ios',
        'host': ip,
        'username': 'admin',
        'password': 'cisco',
        'port': 22
    }

    device_list.append(cisco_device)
print(device_list)


# iterating over device list.
for device in device_list:
    connection = ConnectHandler(**device)

    # entering the enable mode.
    print('Entering the enable mode...')
    connection.enable()
```

Use Cases

6./7. How to backup & troubleshoot

5_automating_backup.py - NA_Project_2023 - Visual Studio Code

EXPLORER

NA_PROJECT_2023

- .vscode
- project-files
- 0_check_installation.py
- 1_connect_run.py
- 2_multiple_commands.py
- 3_using_file.py
- 4_multiple_commands_file.py
- 5_automating_backup.py
- 6_troubleshooting.py
- devices.txt
- ospf.txt
- Router1.txt
- Router2.txt
- z_NA_Project_2023.gns3

5_automating_backup.py > ...

```python
print('Entering the enable mode...')
connection.enable()


# getting the hostname from the device
output = connection.send_command('show run')
prompt = connection.find_prompt()
hostname = prompt[0:-1]

# getting currwent date and time
now = datetime.now()
year = now.year
month = now.month
day = now.day

# creating a file backup of the device configuration
# using hostname + current date
filename = f'{hostname}-{year}-{month}-{day}-backup.txt'
with open(filename, 'w') as backup:
    backup.write(output)
    print(f'Backup of {hostname} completed successfully.')


print('Closing connection')
connection.disconnect()
```

OUTLINE

TIMELINE

0 0 Quokka Ln 39, Col 1 Spaces: 4 UTF-8 CRLF Python 3.10.7 64-bit Prettier

Limitations

Unsupported devices: Netmiko supports a wide range of network devices from various vendors, but there may be some devices that are not yet supported. If the device is not supported by Netmiko, we may need to use another library or write custom code to automate its configuration.

Performance limitations: Netmiko relies on SSH or Telnet connections to communicate with network devices, which can introduce some performance overhead. If we need to perform high-volume or real-time data processing, we may need to consider other approaches, such as using SNMP or a dedicated network management system.

Unmanageable for large networks: Netmiko may not be suitable for extremely large or complex networks, as it may become cumbersome to manage and scale.

Security concerns: While Netmiko provides secure SSH connections to network devices, there may be security risks associated with using Python scripts for network automation. In some cases, we may need to use more secure and auditable approaches, such as using configuration management tools or running automation scripts in a dedicated environment.

Limited functionality: While Netmiko provides a rich set of features for interacting with network devices, it may not cover every possible use case or scenario. In some cases, we may need to use other libraries or tools to complement Netmiko's functionality.

References

- https://github.com/ktbyers/netmiko
- https://pyneng.readthedocs.io/en/latest/book/18_ssh_telnet/netmiko.html
- https://ktbyers.github.io/netmiko/docs/netmiko/index.html
- https://www.youtube.com/watch?v=76MFWqhwFfs&ab_channel=NetworktoCode
- https://pypi.org/project/netmiko/

Thank you