Las Vegas + Digital | June 12-16, 2022

Programming for Network Engineers LABATO-1010

Speakers:

Ambili Sasidharan – Leader, Customer Delivery

Senthil Palanisamy - Principal Architect, CX

Jagadesh Nagaraj - Consulting Engineer, CX

Table of Contents

Learning Objectives

Upon completion of this lab, you will be able to:

- Understand various use cases and examples of network programmability and automation.
- Understand the basic concepts on how to build python scripts and automate network tasks.
- Interact with Cisco devices using python programming language.
- Use python SSH libraries to communicate with cisco devices.
- Build and leverage simple network automation tools for day-to-day operations like backup device configurations and create inventory reports.

Prerequisites

- It is strongly recommended to read through the lab guide and complete all lab tasks, which will give you a good foundation for understanding automation concepts and interacting with cisco devices using Python.
- Basic Python programming and Linux skills
    - Setup Python on your system (Linux, Windows or Mac)
    - Install Virtualenv and Python packages
    - Understand basic Python concepts such as:
        - Variables
        - Functions
        - Imports

Disclaimer

This training document is to get you familiarize with network programming and automation concepts. Although the lab design, configuration and python demo scripts could be used as a reference, it's not a real design, thus not all recommended features are used, or enabled optimally to use it in production networks. For the design related questions please contact your representative at Cisco, or a Cisco partner. Else feel free to reach out to us. We will be glad to answer.

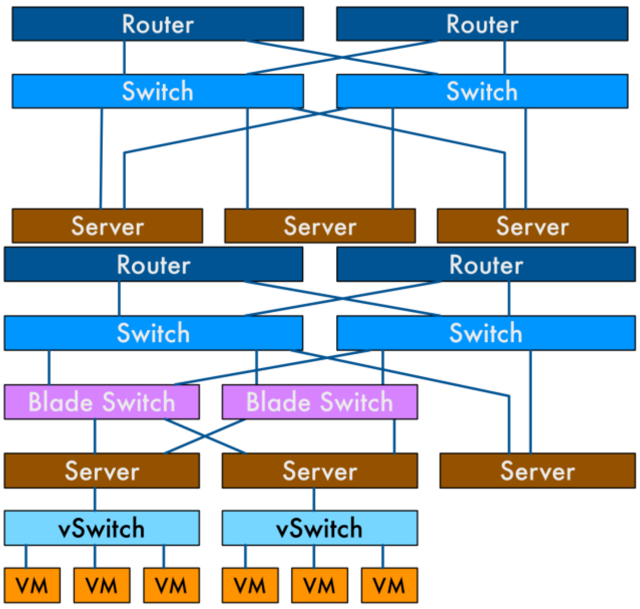
Pre-Configuration

Due to limited duration of walk-in labs, we have the devices preconfigured to save time.

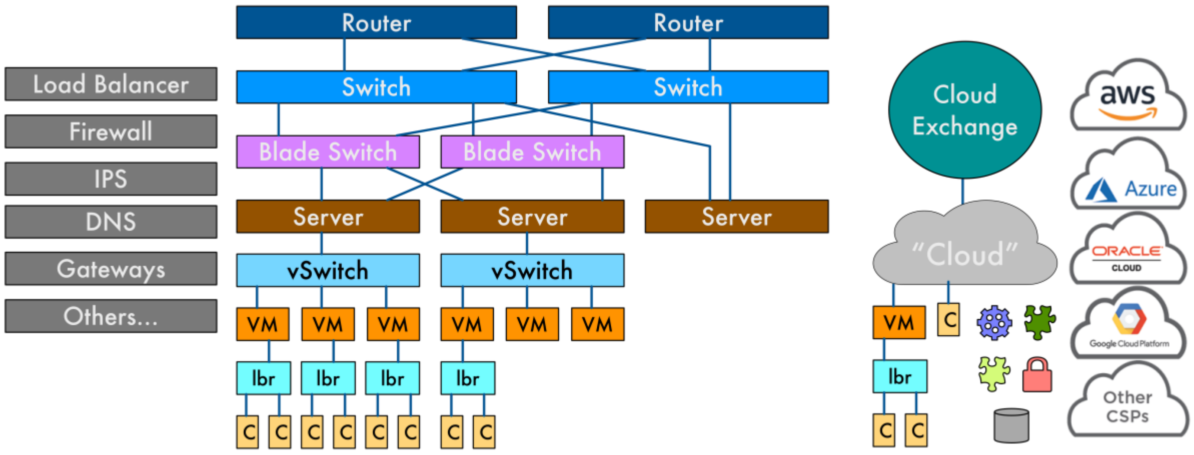
Introduction

In today' modern Multicloud and Hybrid network environment, the roles and responsibilities of the Network Engineers are drastically changing. Reducing Opex and Faster time to rollout services are the key business drivers for most of the IT organizations today. As the networks grow, it has increased the network management and operations complexity to a very high level.

The state of the network has changed from a simple three tier architecture to complex Multicloud and Hybrid network architecture. Especially Software as a service (or SaaS), secure access service edge (SASE), distributed computing, microservices and container technologies has revolutionized the landscape of IT.



Simple Network



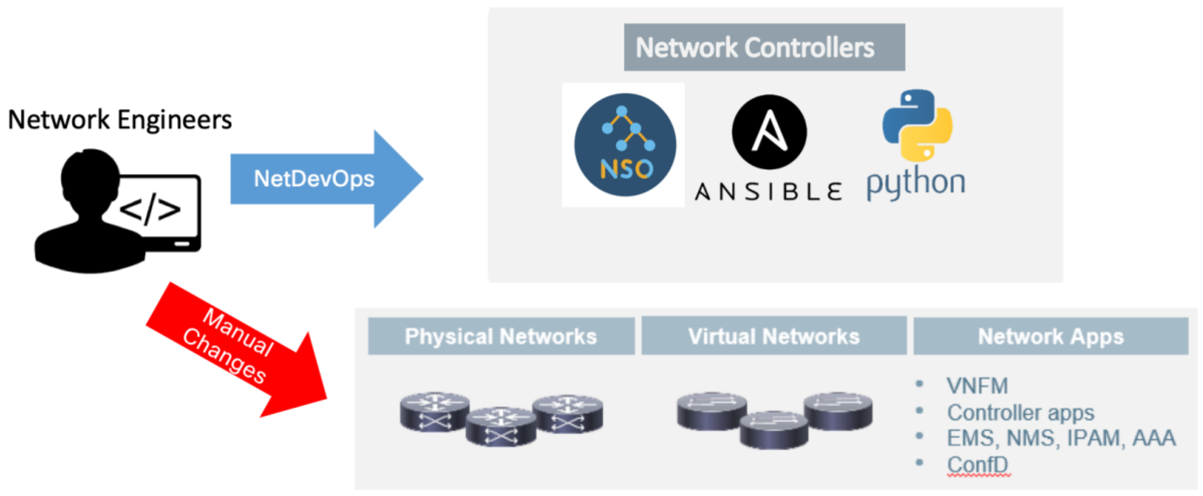
Multicloud Hybrid Network

Network Programmability is the new way of communicating with the network devices and services. The main reason to learn python is to automate provisioning, push configuration changes, network monitoring and troubleshoot complex network issues. Additionally, leveraging automation leads to a more predictable and uniform network as a whole and streamlines the process for all users supporting a given network environment.

Manual vs Automation

CLI is still a critical tool in the network engineers tool kit, because it is readily available, easy to use and grants valuable insights about the state of network. Unfortunately, CLI is not without its flaws, especially when it comes to network programmability and automation. CLI is inherently not scalable, and it is designed to have 1:1 human interaction which makes it slow. Another issue is that the CLI outputs or data passed over CLI are unstructured raw data which are traditionally designed in a human readable fashion. This makes screen scraping increasingly complex when interacting with multiple different hardware platforms and software versions.

As today's networks are constantly growing in size every day, manually making changes or managing device by device will become a huge daily challenge. So, the solution is to adopt NetDevops concepts for managing the network and move away from making manual changes to the network. By moving towards a devops model to manage the network, we can build networks that can scale better and helps to reduce human errors, configuration drift and faster troubleshooting. Consistency, Reliability and Scalability are the three main pillars that defines the modern network. The general rule here is Use Programmability and Automation whenever possible and fallback to using CLI for the tasks that cannot be automated.



The analogy here is that you don't need to know all the parts that goes inside a car to drive a car. This lab is designed in the same way, as a Network Engineer who is getting started with programming you don't need to spend countless hours of time in educating yourself with learning all the basics of programming.

Instead create a lab infrastructure, leverage the work that is already done by the networking community, play and learn along the way. Start with collecting show command outputs, inventory details and making small changes that is not going to break your network. For Ex: Collecting facts, backup configs, add or update syslog, snmp or ntp servers, add dummy loopback or acl's. These non-intrusive tasks will help you to build and run automation tools without impacting the network.

Why Python

Python is a very easy and fun programming language to learn. Rather than having to jump into strict syntax rules, Python reads like English and is simple to understand for someone who's new to programming. Apart from the networking use-cases, we can pretty much program or automate anything using python. If you dedicate 20-30mins to practice python programming every day, you will become an expert python programmer in no time.

Python Programming skillset is mandatory for today's Network engineers, one needs to know and feel comfortable with how to read, write, edit, modify and expand python scripts. It is specifically around the operations of a network that learning to read and write some code starts to make sense.

The way we manage the network haven't changed much for many years. As network engineers we have done the same repeated tasks thousands of times over and over again. Using python programming we can automate most of the repeated network admin tasks any number of times without any typo or human errors. Another benefit is you can save lot of time which is spent on doing boring repeated/manual network admin tasks by automating it.

The main benefit of using python for network management is, we don't need to reinvent the wheel. There are lot of open-source and community driven python libraries that we can leverage to quickly get started and bring our self upto speed in a very short time.

There are two types of tasks that you can try automating:

1) Frequently done small tasks
2) One-time bulk changes

One final warning - and this is a warning about network automation in general. Error prone scripts can break the network very fast and can cause major business impact. You potentially might be changing fifty devices instead of one; you could be changing devices in multiple locations. The consequences of mistakes are greatly increased.

When it comes to programming or automation, don't have an all or nothing mindset. Consequently, start small. Follow the Crawl, Walk, Run approach. Start with collecting show commands instead of configuration changes. When making config changes practice good operational procedures, know what you are doing. Deploy the change to a single device, then expand it to a small set of devices; consider the consequences if things go wrong and how you will correct them. Determine whether this change is something that makes sense to do programmatically.

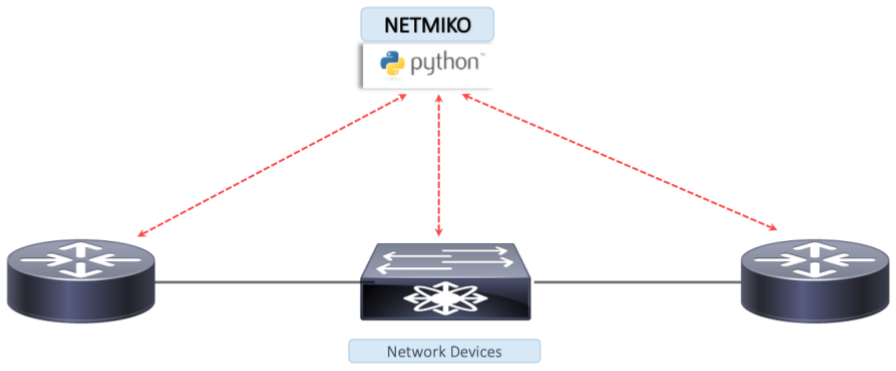
Test the changes first, in a physical lab or a virtual lab using a network simulator like CML, GNS3 or EVE-NG. This lab is built using a CentOS VM and Cisco Modelling Labs (CML) platform.

For this lab you will be learning about two very popular python libraries named Netmiko and Nornir.

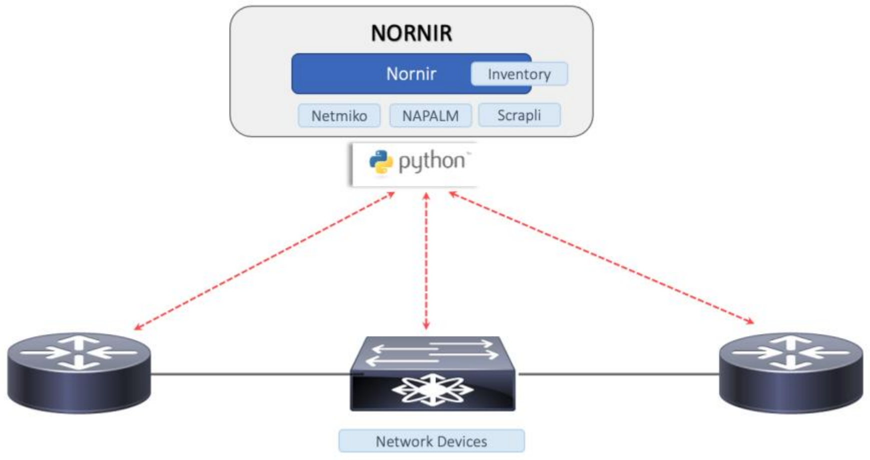
Netmiko is an open-source python library based on Paramiko SSH library. It provides a uniform programming interface across a broad set of network devices. It also handles many of the low-level SSH details that can be time consuming and problematic when you are a beginner and getting started with programming.

The purpose of this library is:

- 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
- Support for multivendor networking platforms



Nornir is a multi-threaded network automation framework that abstracts inventory and task execution. Is operates very similar to Ansible, instead of writing playbooks in YAML syntax, nornir uses python syntax. So, it is easy to execute the tasks like configuring the devices, validating the operational data, and enabling the services on the managed hosts which are part of the inventory. As it is multithreaded, it allows you to manage the configuration of multiple network devices concurrently. It is also an open-source project, completely written in python and easy to use.



The advantages of nornir are:

- Provides an automation framework in python
- Easy management and troubleshooting of the automation scripts
- Inbuilt inventory and multithreading capabilities to scale for large networks
- It reuses existing python libraries like Netmiko and Napalm to connect and manage the devices.

There are several other features and functionalities available in Netmiko, Nornir or python programming itself, we are not covering all the topics in this lab. Similarly, there are lot of vendor and community developed python libraries and SDK's that you might come across which does similar functions too. The goal here is to educate Network engineers about how easy it is to get started with network programmability and automation without going too much deeper into advanced python programming concepts.

For beginners learning about programming and automation can be very intimidating. Don't worry there are many sessions in Ciscolive that you can attend and get yourself educated. Please check the Devnet zone, breakout session catalog and reference section to learn more about Python programming, Network Programmability and Automation.

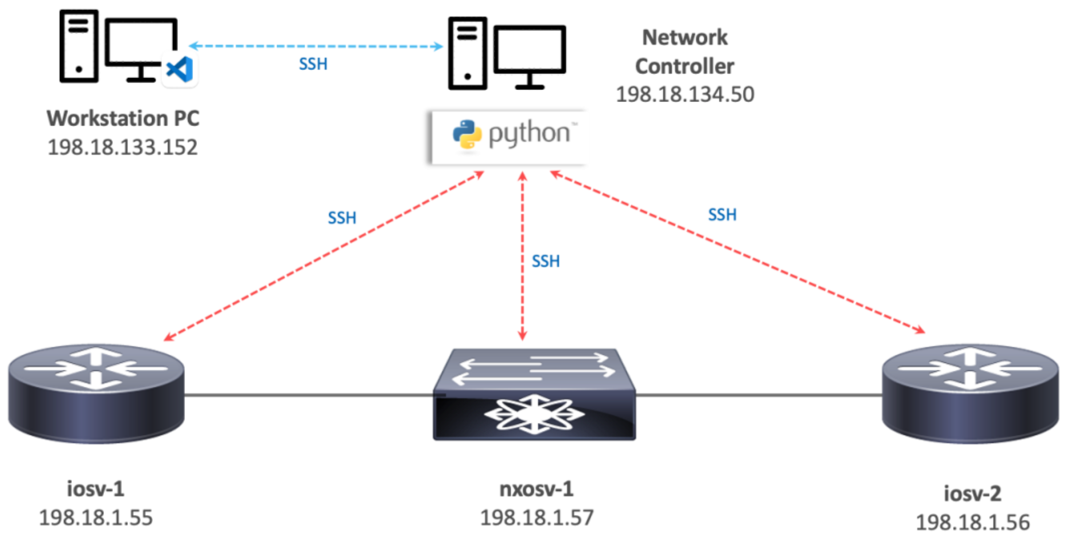
Some of the recommended Devnet sessions that you can attend this week are:

Intro to Python - DEVWKS-1175

Python Advanced - DEVWKS-2860

Note: Search for "devnet coding" or "devnet python" on youtube. You can get access to the previous recording of coding sessions.

Lab Topology



Note: The workstation provides all the required access to the lab devices

Connection Details:

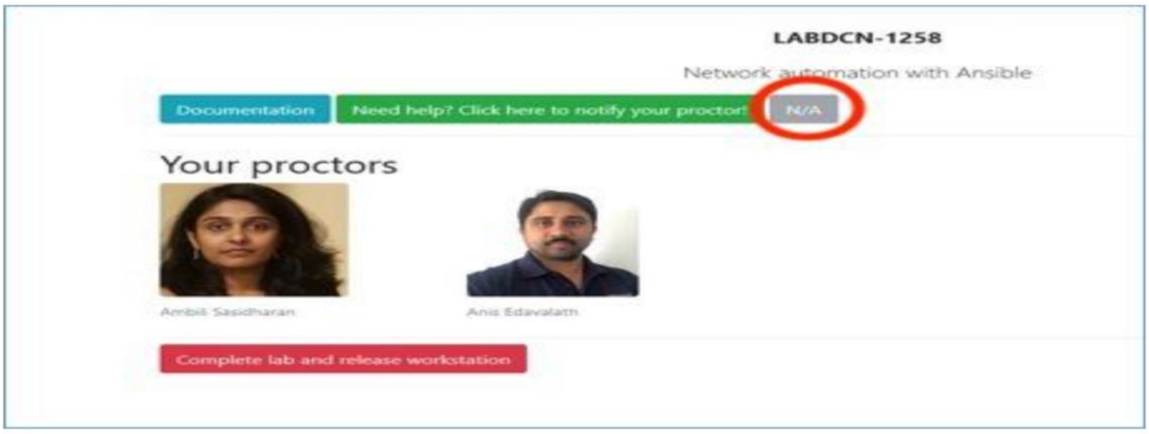
Hostname	IP	Username	Password	Connection	Port
iosv-1				SSH	22
iosv-2				SSH	22
nxosv-1				SSH	22
Network Controller				SSH	22
Workstation				RDP	3389

Code repository:

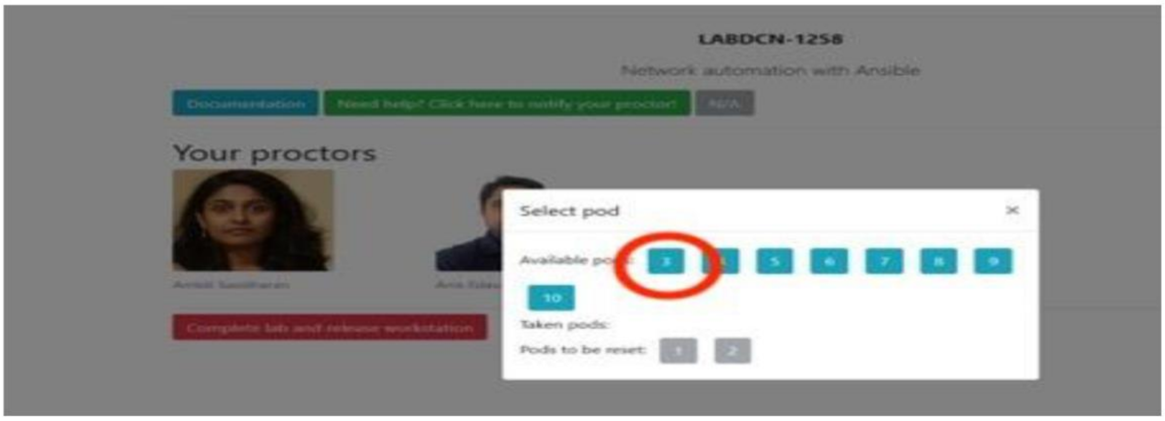
https://github.com/jagadnag/cl_python_2022.git

Connecting to the WIL Assistant and accessing the Lab

Step 1: From the WIL assistance home page @ www.wilassiatance.com , click on START and search for the lab 2658 and click on START. You will see the page as below. Click on the buton N/A to select the Pod.


LABDCN-1258
Network automation with Ansible
Documentation
Need help? Click here to notify your proctor!
N/A
Your proctors
Ambili Sasidharan
Anis Edavalath
Complete lab and release workstation

Step 2: Please select the Pod that is assigned to you on the hand out given to you or the lab proctor.


LABDCN-1258
Network automation with Ansible
Documentation
Need help? Click here to notify your proctor!
N/A
Your proctors
Select pod
Available pods:
3
4
5
6
7
8
9
10
Taken pods:
Pods to be reset:
1
2
Complete lab and release workstation

Step 3: Open the lab guide and follow the lab guide to connect to the RDP session and proceed to the lab exercises.

Connect to dCloud lab network

This lab leverages dCloud, which uses virtual servers and switches that runs on a simulator. These devices have most of the functions of actual hardware based devices, but there is no data plane traffic.

You need to VPN into the dCloud environment in order to access devices and complete this lab.

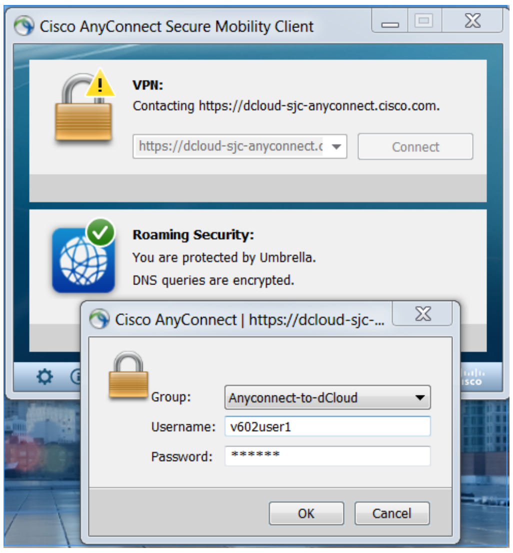
Click on the Cisco AnyConnect client and check to see if the client is connected to another VPN session. If yes, go ahead and disconnect.

Now type or copy paste the VPN server details:

dcloud-sjc-anyconnect.cisco.com

For the vpn credentials, use the username and password that is attached to the lab card. Input the username and password. Click OK.

If you need any additional assistance, please use the WISP lab assistant to get help from lab proctors.


Cisco AnyConnect Secure Mobility Client

VPN:
Contacting https://dcloud-sjc-anyconnect.cisco.com.

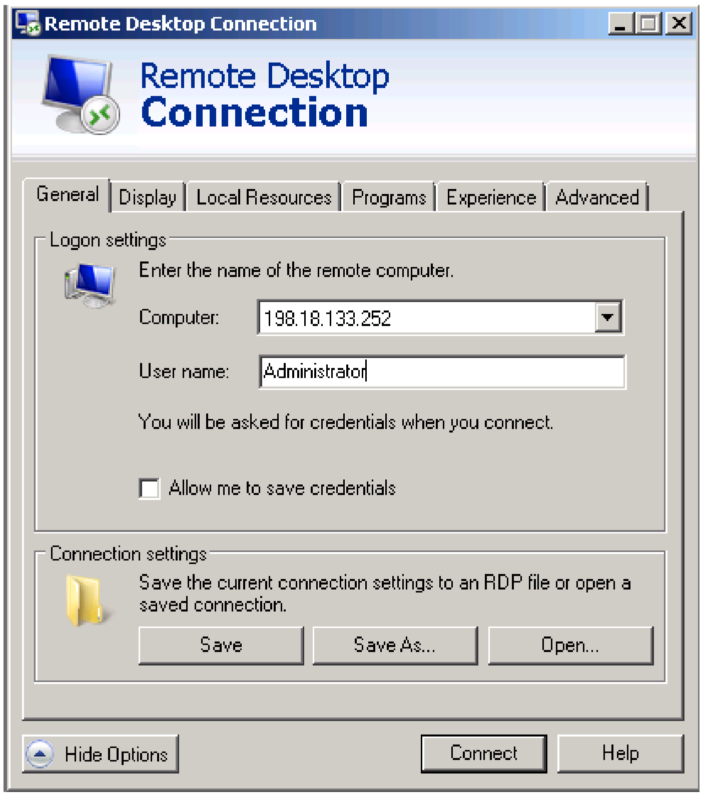
Roaming Security:
You are protected by Umbrella.
DNS queries are encrypted.

Cisco AnyConnect | https://dcloud-sjc-...

Group: Anyconnect-to-dCloud
Username: v602user1
Password: ******

Once VPN connection is successful, you should be able to access the lab environment.

Now open a Remote Desktop Connection (RDP) to the lab workstation using the RDP client.


Remote Desktop Connection

Remote Desktop Connection

Logon settings

Enter the name of the remote computer.

Computer: 198.18.133.252

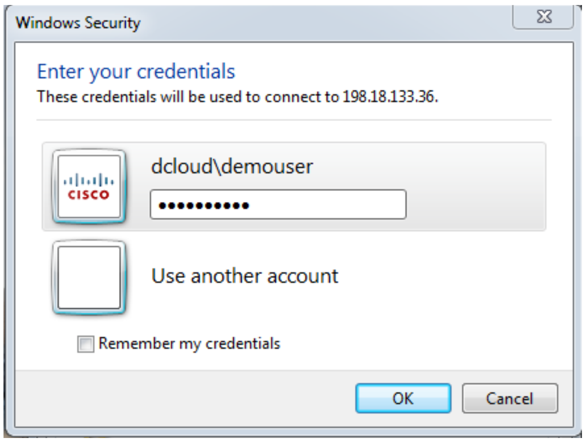
User name: Administrator

You will be asked for credentials when you connect.

☐ Allow me to save credentials

Connection settings

Save the current connection settings to an RDP file or open a saved connection.


Windows Security

Enter your credentials

These credentials will be used to connect to 198.18.133.36.

dcloud\demouser

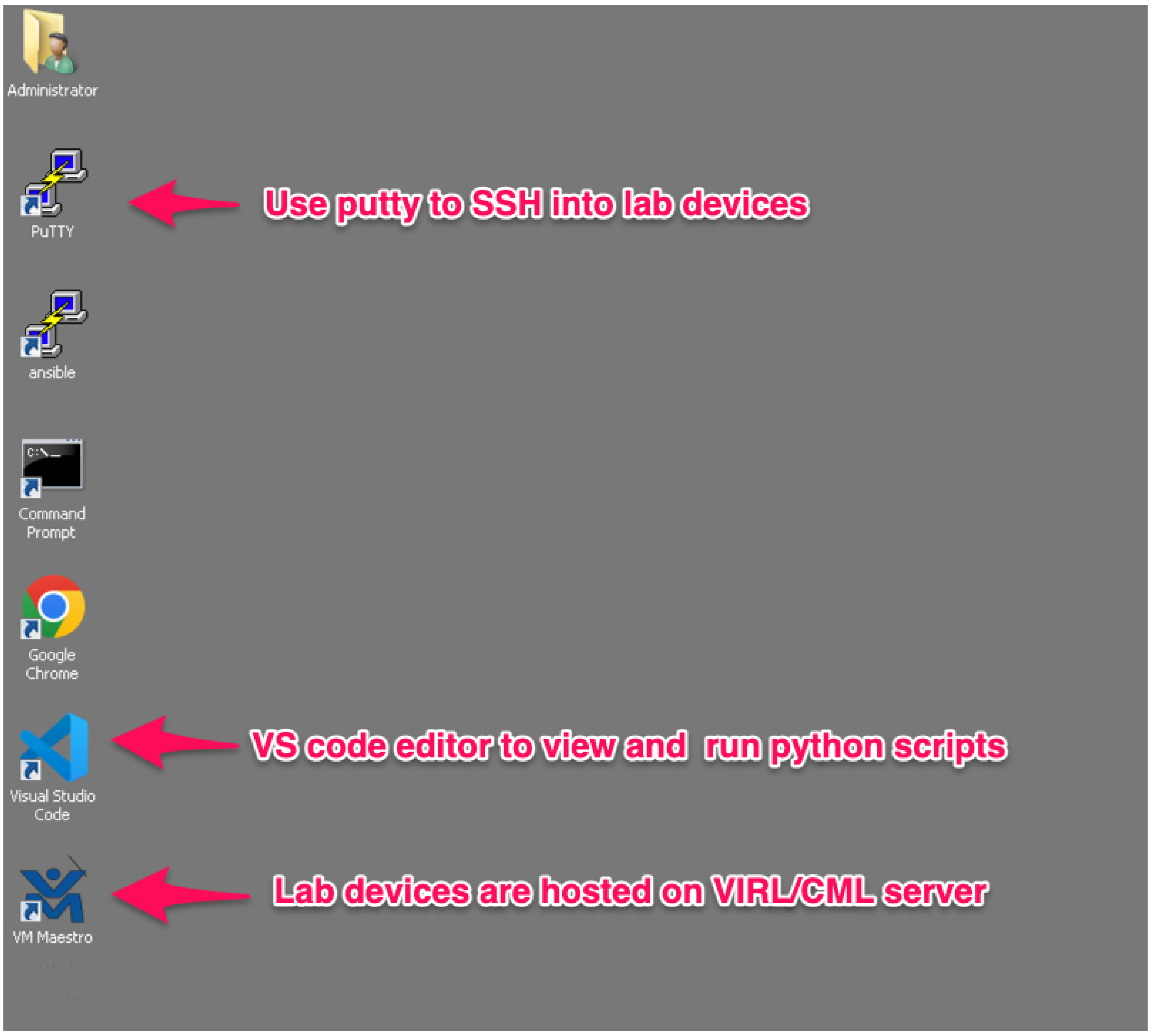
••••••••••

Use another account

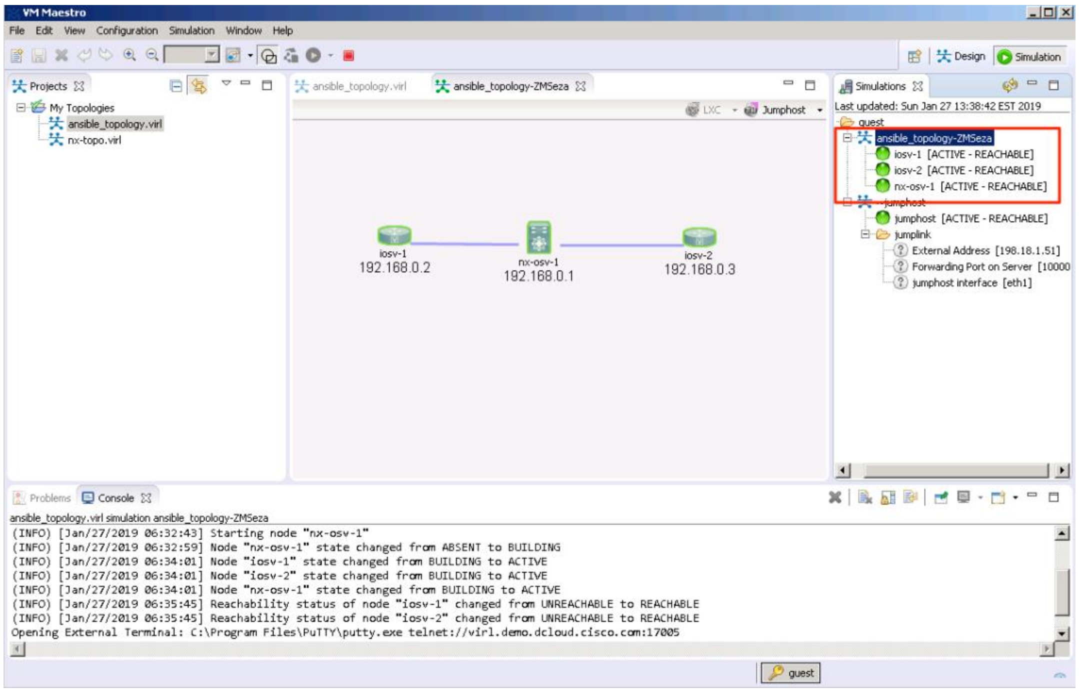
☐ Remember my credentials

Enter the password for RDP connection –

After establishing the RDP connection, you will see a similar desktop screen setup as shown below. Use the tools as needed to complete the lab tasks.


Administrator
PuTTY
Use putty to SSH into lab devices
ansible
Command Prompt
Google Chrome
Visual Studio Code
VS code editor to view and run python scripts
VM Maestro
Lab devices are hosted on VIRL/CML server

Open VM Mastero application and ensure all the lab devices are in green (active state). You can also use putty to ssh into the lab devices and check their status.



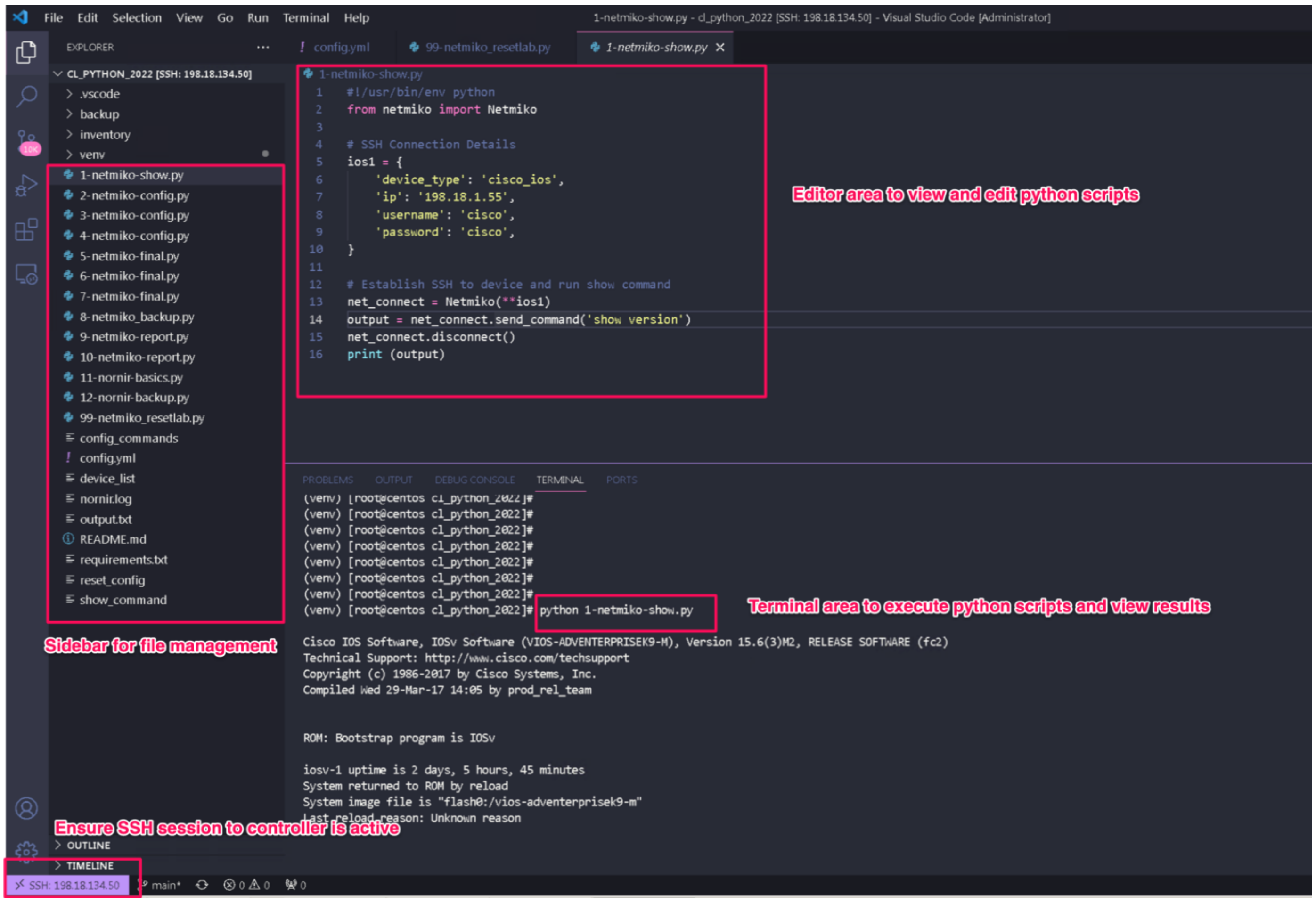
Last updated: Sun Jan 27 13:38:42 EST 2019

```
guest
  ansible_topology-ZMSeza
    iosv-1  [ACTIVE - REACHABLE]
    iosv-2  [ACTIVE - REACHABLE]
    nx-osv-1  [ACTIVE - REACHABLE]
  --jumphost
    jumphost  [ACTIVE - REACHABLE]
    jumplink
      External Address  [198.18.1.51]
      Forwarding Port on Server  [10000
      jumphost interface  [eth1]
```

ansible_topology.virl simulation ansible_topology-ZMSeza

```
(INFO) [Jan/27/2019 06:32:43] Starting node "nx-osv-1"
(INFO) [Jan/27/2019 06:32:59] Node "nx-osv-1" state changed from ABSENT to BUILDING
(INFO) [Jan/27/2019 06:34:01] Node "iosv-1" state changed from BUILDING to ACTIVE
(INFO) [Jan/27/2019 06:34:01] Node "iosv-2" state changed from BUILDING to ACTIVE
(INFO) [Jan/27/2019 06:34:01] Node "nx-osv-1" state changed from BUILDING to ACTIVE
(INFO) [Jan/27/2019 06:35:45] Reachability status of node "iosv-1" changed from UNREACHABLE to REACHABLE
(INFO) [Jan/27/2019 06:35:45] Reachability status of node "iosv-2" changed from UNREACHABLE to REACHABLE
Opening External Terminal: C:\Program Files\PuTTY\putty.exe telnet://virl.demo.dcloud.cisco.com:17005
```

By default, beginner programmers will use python shell and IDLE for learning purposes. Once after getting some hands-on experience with IDLE, you will be using a Code editor to develop and test python scripts. For this lab we use Microsoft Visual Studio (VS) Code application which is very user friendly and feature rich. It is free and available for macOS, Linux, and Windows operating systems. For more information, please refer to: https://code.visualstudio.com/



```python
#!/usr/bin/env python
from netmiko import Netmiko

# SSH Connection Details
ios1 = {
    'device_type': 'cisco_ios',
    'ip': '198.18.1.55',
    'username': 'cisco',
    'password': 'cisco',
}

# Establish SSH to device and run show command
net_connect = Netmiko(**ios1)
output = net_connect.send_command('show version')
net_connect.disconnect()
print (output)
```

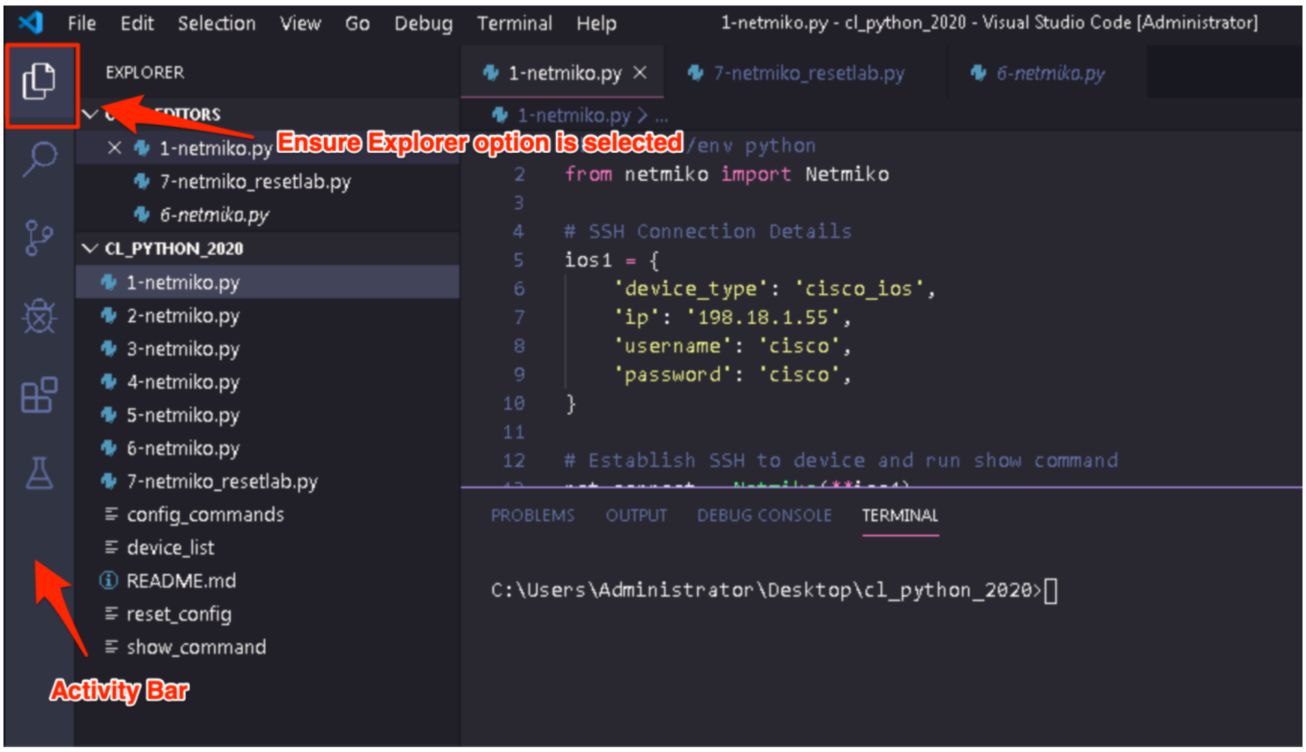
```
(venv) [root@centos cl_python_2022]#
(venv) [root@centos cl_python_2022]#
(venv) [root@centos cl_python_2022]#
(venv) [root@centos cl_python_2022]#
(venv) [root@centos cl_python_2022]#
(venv) [root@centos cl_python_2022]#
(venv) [root@centos cl_python_2022]#
(venv) [root@centos cl_python_2022]# python 1-netmiko-show.py

Cisco IOS Software, IOSv Software (VIOS-ADVENTERPRISEK9-M), Version 15.6(3)M2, RELEASE SOFTWARE (fc2)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2017 by Cisco Systems, Inc.
Compiled Wed 29-Mar-17 14:05 by prod_rel_team


ROM: Bootstrap program is IOSv

iosv-1 uptime is 2 days, 5 hours, 45 minutes
System returned to ROM by reload
System image file is "flash0:/vios-adventerprisek9-m"
Last reload reason: Unknown reason
```

Figure: Visual Studio Code Editor Layout

This section is for your reference only, the lab tasks start from page 19, if you are stuck with any errors or issues while running the python scripts. Please request the lab proctor to help.

Here are some tips that will get you started with VS Code editor.

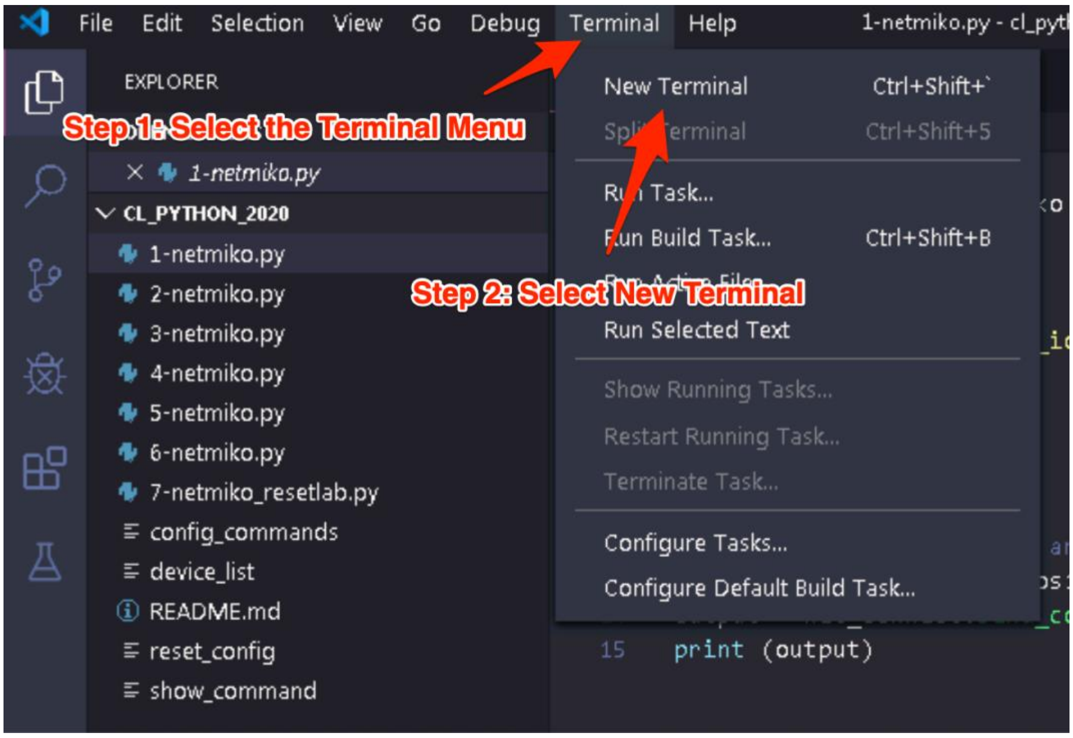
Ensure Explorer option is selected



```
from netmiko import Netmiko

# SSH Connection Details
ios1 = {
    'device_type': 'cisco_ios',
    'ip': '198.18.1.55',
    'username': 'cisco',
    'password': 'cisco',
}

# Establish SSH to device and run show command
```

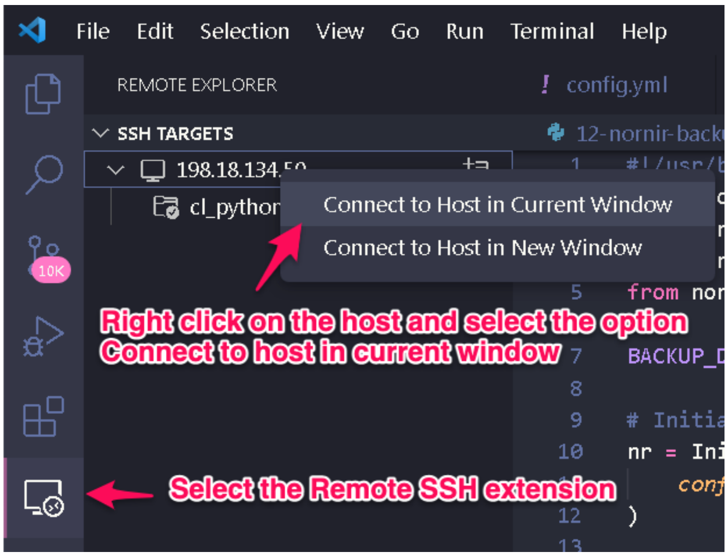
```
C:\Users\Administrator\Desktop\cl_python_2020>
```

How to open the Terminal window:

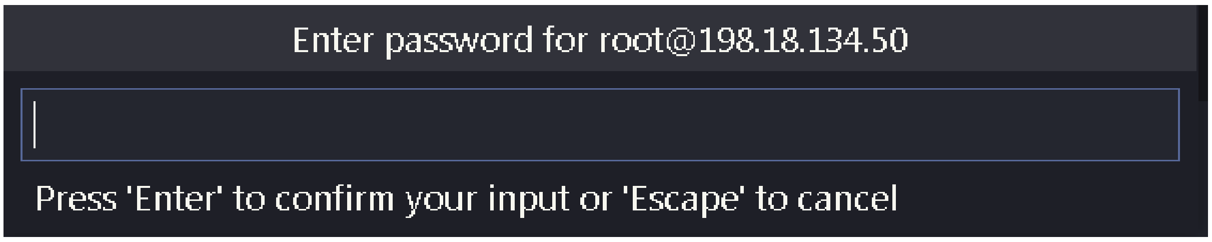

File Edit Selection View Go Debug Terminal Help
Step 1: Select the Terminal Menu
New Terminal Ctrl+Shift+`
Split Terminal Ctrl+Shift+5
Run Task...
Run Build Task... Ctrl+Shift+B
Run Active File
Run Selected Text
Step 2: Select New Terminal
Show Running Tasks...
Restart Running Task...
Terminate Task...
Configure Tasks...
Configure Default Build Task...

```
print (output)
```

How to SSH into the network controller from VScode and access the scripts stored in the network controller

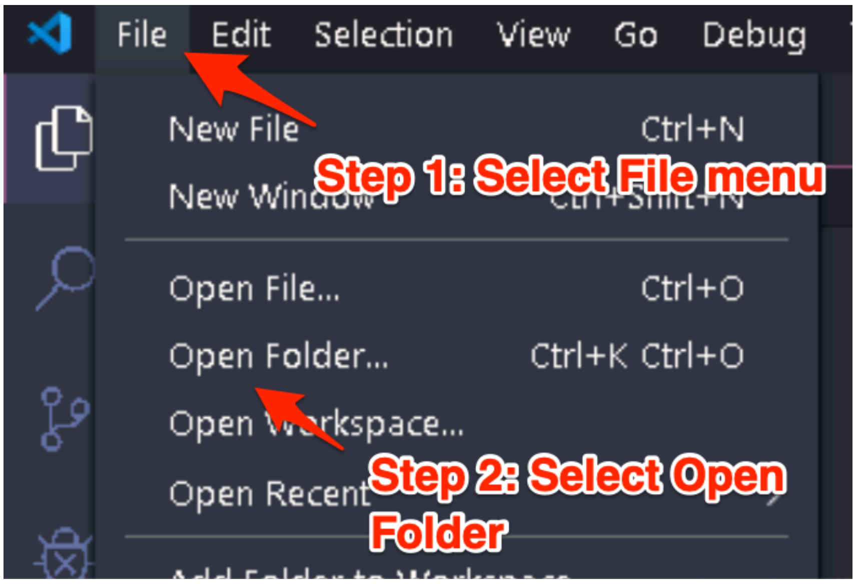

File
Edit
Selection
View
Go
Run
Terminal
Help
REMOTE EXPLORER
config.yml
SSH TARGETS
12-nornir-back
198.18.134.50
cl_python
Connect to Host in Current Window
Connect to Host in New Window
Right click on the host and select the option Connect to host in current window
Select the Remote SSH extension

If password is requested, enter the password

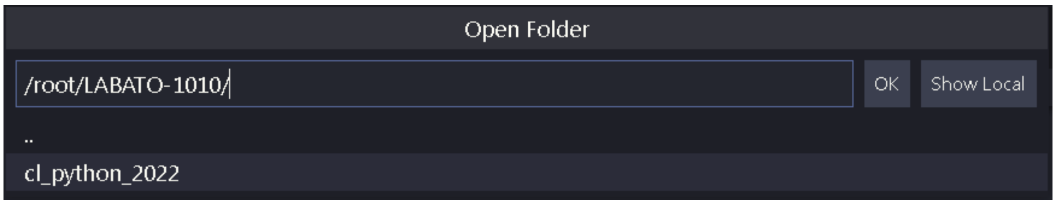

Enter password for root@198.18.134.50

Press 'Enter' to confirm your input or 'Escape' to cancel

How to open the scripts folder:



Browse to folder path `/root/LABATO-1010/cl_python_2022` and select OK.



Navigating to python scripts folder

All python scripts are preloaded in the network controller folder path

`/root/LABATO-1010/cl_python_2022`

All python modules are installed inside the (venv) python virtual environment. Ensure the network controller prompt is

`(venv) [root@centos cl_python_2022]#`

If you see [root@centos]# without (venv) in the front of the prompt, or in wrong path, use the commands

```
cd /root/LABATO-1010/cl_python_2022 && source  venv/bin/activate
```

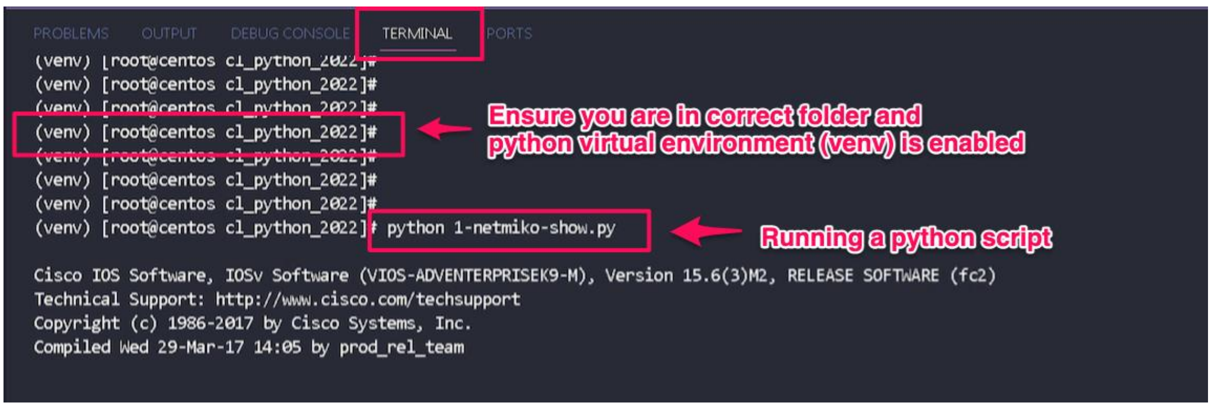
to activate the python virtual environment and verify you are in right path using 'pwd' command

```
(venv) [root@centos cl_python_2022]# pwd

/root/LABATO-1010/cl_python_2022
```

Note: Lab device ip and login credentials are available in the table Connection Details (page 9)

How to execute python scripts from VS Code:

To execute the scripts, go to the Terminal window, type "`python`" and name of the script and press enter. If python requests, you to enter some inputs (for ex: username or password) type the requested information and press enter.


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Ensure you are in correct folder and python virtual environment (venv) is enabled
Running a python script

```
(venv) [root@centos cl_python_2022]#
(venv) [root@centos cl_python_2022]#
(venv) [root@centos cl_python_2022]#
(venv) [root@centos cl_python_2022]#
(venv) [root@centos cl_python_2022]#
(venv) [root@centos cl_python_2022]#
(venv) [root@centos cl_python_2022]#
(venv) [root@centos cl_python_2022]# python 1-netmiko-show.py

Cisco IOS Software, IOSv Software (VIOS-ADVENTERPRISEK9-M), Version 15.6(3)M2, RELEASE SOFTWARE (fc2)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2017 by Cisco Systems, Inc.
Compiled Wed 29-Mar-17 14:05 by prod_rel_team
```

Note: Please verify the path first before executing the scripts, if you are not in the right folder where the scripts are stored (`/root/LABATO-1010/cl_python_2022`) Python will throw a "`No such file or directory error`".

LAB TASKS

Building blocks of Netmiko python script

In this lab exercise you will learn how a basic netmiko python script is constructed. Think of it like a Lego block, first we will start with basic building blocks and quickly iterate through it to build complex programming logics.

On a very high level you will see three process that are being repeated for every script.

1) Import the required python modules
2) Feed the input data to the script, either we hardcode the information inside the program like username, ip address, filename or request the user to provide that information
3) Then we will build some logic on how to work with the data and execute the programmed tasks
4) Finally, we will print the output to screen or store the processed output to a file

Let's dive into the basic building blocks. First, we import the Netmiko connection libraries

```
from netmiko import Netmiko
```

The connection parameters are collected in a Python dictionary. The connection parameters provide Netmiko with everything it needs to create the SSH connection. In the below example we have shown how to connect to an IOS device

```
ios1 = {
    'device_type': 'cisco_ios',
    'ip': '192.18.1.55',
    'username': 'cisco',
    'password': 'cisco',
```

Netmiko is a function that calls the necessary connection parameters and device type (cisco_ios, cisco_xr, cisco_nxos, etc.) Once connection parameters are loaded, the script will launch a SSH connection to login into the device.

```
net_connect = Netmiko(**ios1)
```

.send_command() method is used to send show commands over the channel and receive the output back. Here, we are reading the output of 'show version' command and storing it in a variable named 'output'.

```
output = net_connect.send_command('show version’)
```

Using the .send_config_set() method, we can program the network device to enter into configuration mode and make configuration changes. After executing the config_commands the script will exit the configuration mode.

```
output = net_connect.send_config_set(config_commands)
```

We can send either only one command or multiple lines of commands by converting it into a simple list. If we are sending a big configuration block, it is recommended to use the .send_config_from_file( ) method.

```
output = net_connect.send_config_from_file('more_config')
```

We use the .save_config( ) function to save the configuration to device.

```
net_connect.save_config()
```

We use the .disconnect( ) function to close the SSH session to the device.

```
net_connect.disconnect()
```

All of the session output is stored in an output variable and then printed out in the screen using print( ) function for our reference.

For more information on the all the connection methods available with Netmiko, pls refer the documentation

https://pynet.twb-tech.com/blog/netmiko-python-library.html

https://github.com/ktbyers/netmiko/blob/develop/EXAMPLES.md

Task 1: Executing a show command on a network device

The python script will

- login to the iosv-1 device via SSH
- run the show version command
- capture the output
- print the command output to the screen.

Let's execute our first netmiko python script. For executing the python script, type the command - `python <script_name.py>` in the Terminal window. The script names are constructed in way to support tab or auto completion. You can type `python 1-(press tab key)` the script name will autocomplete. Then you can press enter key to run the script.

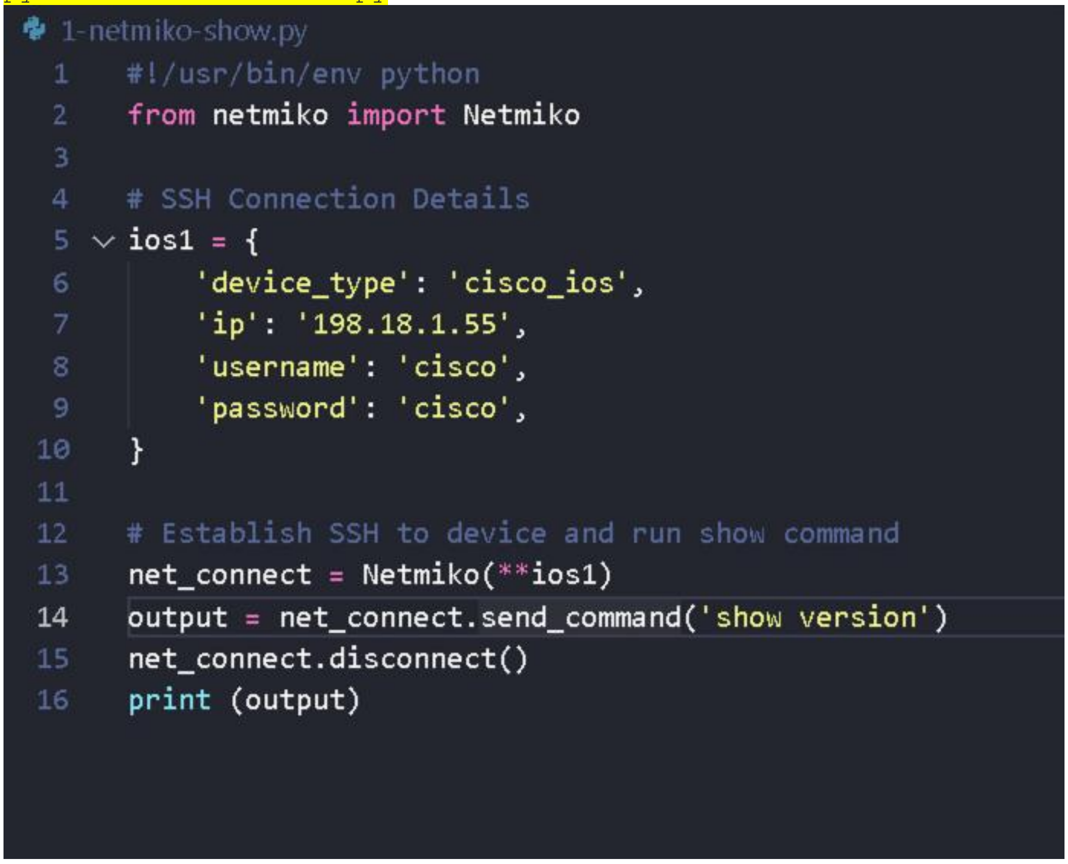
Note: The script output will get printed on the Terminal window itself. On an average it will take 5-10 seconds for the script to complete the tasks in the background and print results to the terminal window.

For testing the scripts outside ciscolive days, please use the free IOS-XE (CSR1000v) Always on Cisco Devnet Sandboxes. Login credentials are provided in the sandbox topology page.

https://devnetsandbox.cisco.com/RM/Topology

Launch the python script from the VS Code Terminal window

```
python 1-netmiko-show.py
```



1-netmiko-show.py

```python
#!/usr/bin/env python
from netmiko import Netmiko

# SSH Connection Details
ios1 = {
    'device_type': 'cisco_ios',
    'ip': '198.18.1.55',
    'username': 'cisco',
    'password': 'cisco',
}

# Establish SSH to device and run show command
net_connect = Netmiko(**ios1)
output = net_connect.send_command('show version')
net_connect.disconnect()
print (output)
```

Verify the "show version" command output printed in the Terminal.

Now you can see how easy it is to get started with python programming. Even though it is a very simple python program with only few lines of code, in the following tasks we will quickly iterate through it to send multiple lines of config commands or retrieve multiple show command output from one device or multiple devices.

The idea here is to understand the basic programmability logic first. Build a base skeleton for the code and quickly iterate through it to build more complex tasks on top of it and automate day to day network admin tasks.

Task 2: Configuring a network device

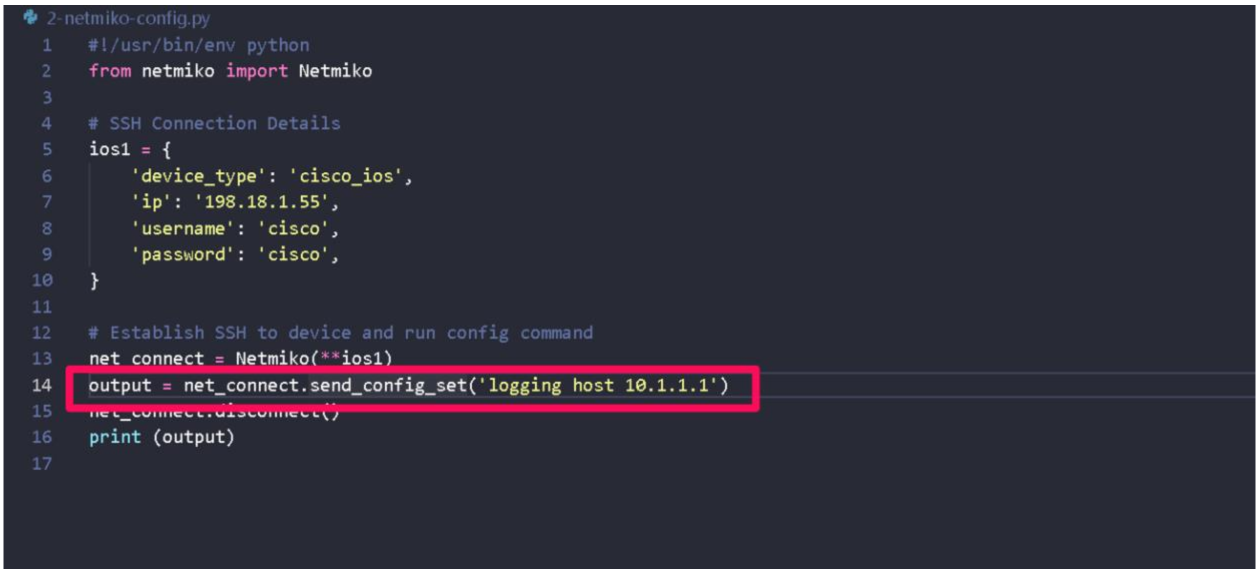
The python script will

- login to the iosv-1 device via SSH
- make a configuration change "logging host 10.1.1.1"
- capture the output
- print the command output to the screen.

Notice we are reusing the same code, that we used in previous task. Only change is, instead of net_connect.send_command() we are using net_connect.send_config_set(). Rest of the script remains the same. Moving forward we will use the same logic, iterate through the code we have already written and add additional logic on top of it to develop our final scripts.

Launch the python script from the VS Code Terminal window

```
python 2-netmiko-config.py
```



2-netmiko-config.py

```
#!/usr/bin/env python
from netmiko import Netmiko

# SSH Connection Details
ios1 = {
    'device_type': 'cisco_ios',
    'ip': '198.18.1.55',
    'username': 'cisco',
    'password': 'cisco',
}

# Establish SSH to device and run config command
net_connect = Netmiko(**ios1)
output = net_connect.send_config_set('logging host 10.1.1.1')
net_connect.disconnect()
print (output)
```

Verify the configuration changes made by the script. The output is printed to the Terminal.

Task 3: Configuring multiple network devices

The python script will

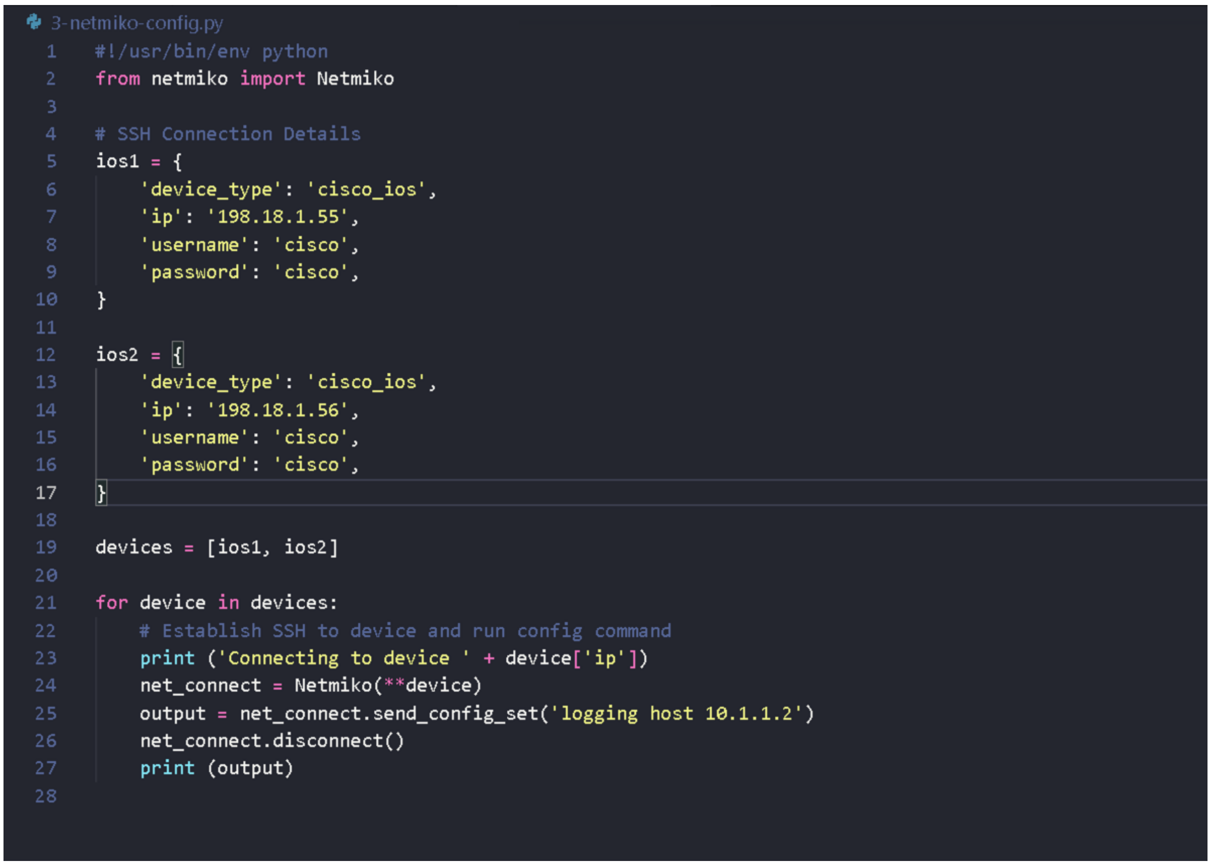
- login to the iosv-1 and iosv-2 devices via SSH
- make a configuration change "logging host 10.1.1.2"
- capture the output
- print the command output to the screen.

To configure multiple devices, we must create multiple SSH connection profiles and add it to a python list. Then add a for loop to iterate through the connection profiles and make config changes to the IOS devices. The code logic will look like this

```
device_list = [ios1, ios2]
for device in device_list:
    ** Netmiko config code block **
```

Launch the python script from the VS Code Terminal window

```
python 3-netmiko-config.py
```



3-netmiko-config.py

```
#!/usr/bin/env python
from netmiko import Netmiko

# SSH Connection Details
ios1 = {
    'device_type': 'cisco_ios',
    'ip': '198.18.1.55',
    'username': 'cisco',
    'password': 'cisco',
}

ios2 = {
    'device_type': 'cisco_ios',
    'ip': '198.18.1.56',
    'username': 'cisco',
    'password': 'cisco',
}

devices = [ios1, ios2]

for device in devices:
    # Establish SSH to device and run config command
    print ('Connecting to device ' + device['ip'])
    net_connect = Netmiko(**device)
    output = net_connect.send_config_set('logging host 10.1.1.2')
    net_connect.disconnect()
    print (output)

```

Verify the configuration changes made by the script. The output is printed to the Terminal.

Task 4: Pushing large configurations across multiple devices

Now we have a solid understanding of how to write basic scripts. Next step is to make our code modular. For that we should remove all hardcoded variables from the script. The variables that are used in the script has to be provided by the user who runs the script or from an external file. This will help you as network engineer / programmer to share your scripts with the extended team, so others can reuse your scripts.

The python script will

- Load the device ip details from a text file named 'device_list'
- Load the configuration commands from a text file named 'config_commands'
- Requests the login credentials from the user
- Uses getpass() module to encrypt the user provided password
- Login to each device in the 'device_list' and configure the commands given in 'config_commands' file.
- Print the output

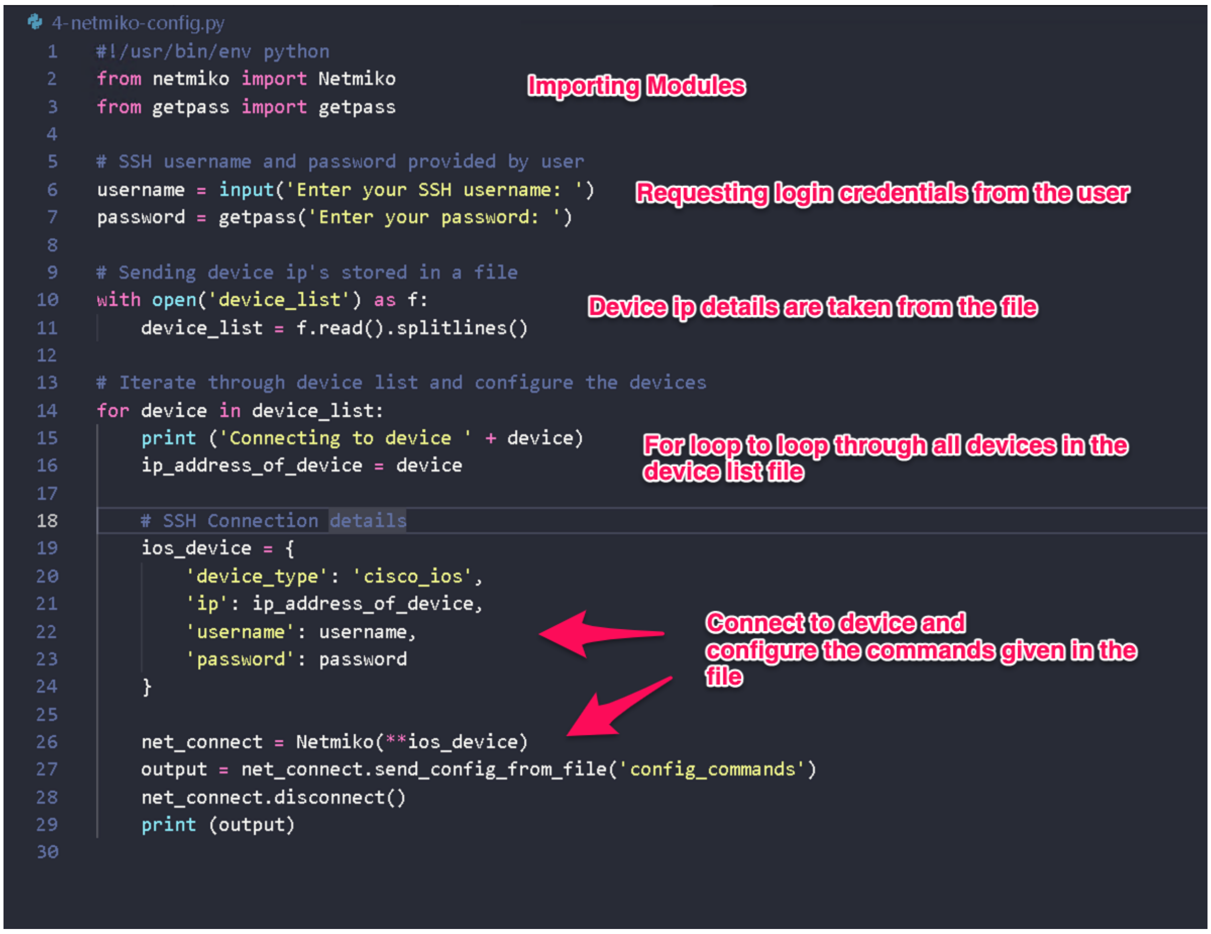
First we use the input() and getpass() modules to collect the login credentials. Next, read the contents of the file using inbuilt python file module - with open(). After that we loop through the device_list and configure the devices. The code logic is given below

```
username = input()
password = getpass()

with open('file_name') as f:
    device_list = f.read().splitlines()

for devices in device_list:
    ** Netmiko config code block **
```

Launch the python script from the VS Code Terminal window

`python 4-netmiko-config.py`

provide the login credentials for the script:
username: cisco
password: cisco



4-netmiko-config.py

```python
#!/usr/bin/env python
from netmiko import Netmiko
from getpass import getpass

# SSH username and password provided by user
username = input('Enter your SSH username: ')
password = getpass('Enter your password: ')

# Sending device ip's stored in a file
with open('device_list') as f:
    device_list = f.read().splitlines()

# Iterate through device list and configure the devices
for device in device_list:
    print ('Connecting to device ' + device)
    ip_address_of_device = device

    # SSH Connection details
    ios_device = {
        'device_type': 'cisco_ios',
        'ip': ip_address_of_device,
        'username': username,
        'password': password
    }

    net_connect = Netmiko(**ios_device)
    output = net_connect.send_config_from_file('config_commands')
    net_connect.disconnect()
    print (output)
```

Importing Modules
Requesting login credentials from the user
Device ip details are taken from the file
For loop to loop through all devices in the device list file
Connect to device and configure the commands given in the file

Verify the configuration changes made by the script. The output is printed to the Terminal.

Task 5: Error handling and verification

In this task we will demonstrate how to enable error handling for our scripts. The idea behind error handling is to catch any exceptions that occurs during the execution of the script. Without error handling when an exception is detected the python script will terminate execution and reports error.

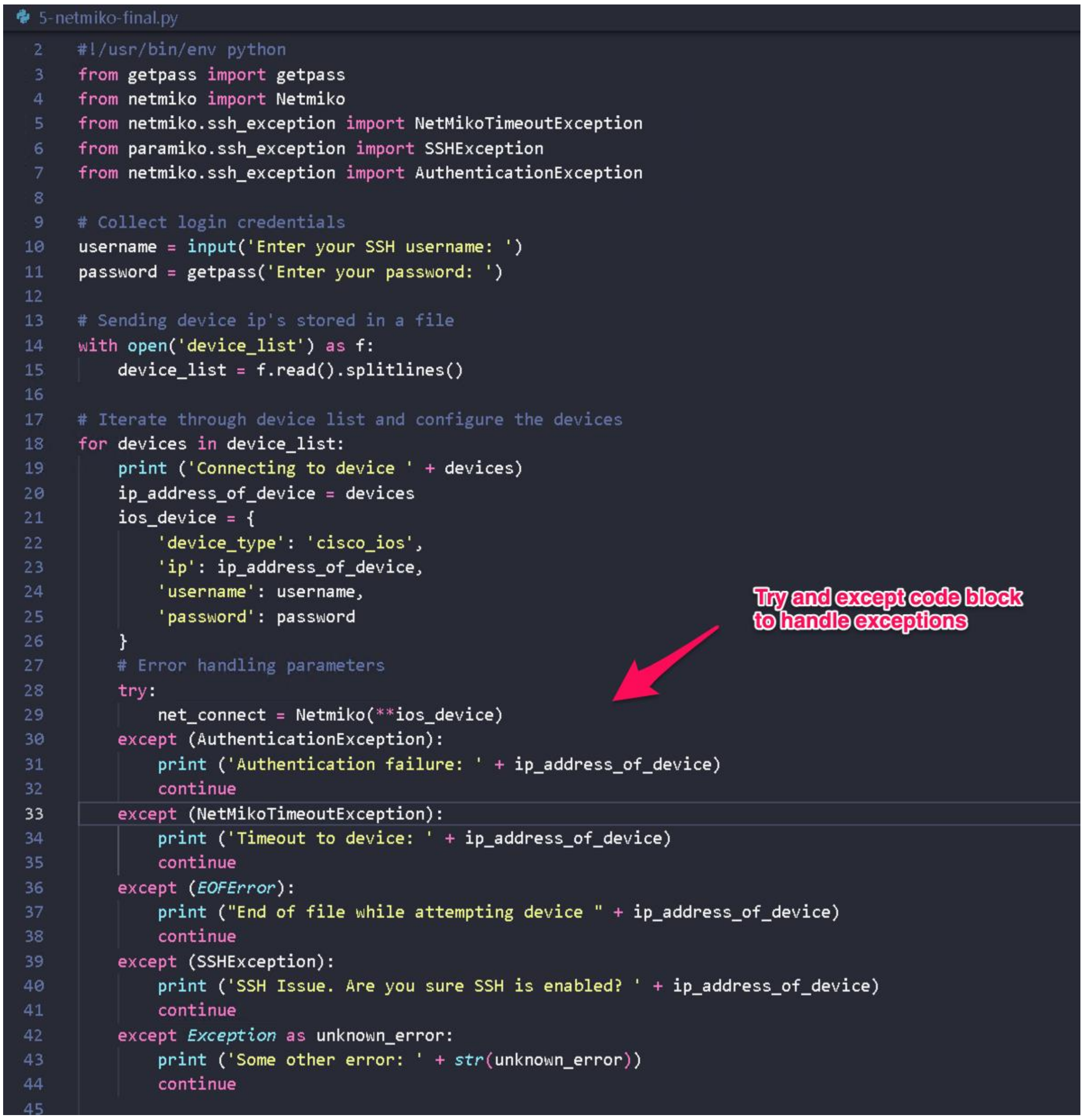
Try and expect code blocks will help us to catch any errors during program execution. We have added different exceptions that can be triggered during the execution. For example: device timeout, reachability issues, wrong user credential errors, etc. If an exception is detected, the script will move on to the next device and complete the task.

```
username = input()
password = getpass()

with open('file_name') as f:
    device_list = f.read().splitlines()

for devices in device_list:
    try:
        ** Netmiko connection **
    except:
        ** Error condition **
        continue
    ** Netmiko config code block **
```

Note: To test the exceptions, try providing wrong username and password to check whether the scripts catch and report the exceptions.

Launch the python script from the VS Code Terminal window

```
python 5-netmiko-final.py
```



```python
#!/usr/bin/env python
from getpass import getpass
from netmiko import Netmiko
from netmiko.ssh_exception import NetMikoTimeoutException
from paramiko.ssh_exception import SSHException
from netmiko.ssh_exception import AuthenticationException

# Collect login credentials
username = input('Enter your SSH username: ')
password = getpass('Enter your password: ')

# Sending device ip's stored in a file
with open('device_list') as f:
    device_list = f.read().splitlines()

# Iterate through device list and configure the devices
for devices in device_list:
    print ('Connecting to device ' + devices)
    ip_address_of_device = devices
    ios_device = {
        'device_type': 'cisco_ios',
        'ip': ip_address_of_device,
        'username': username,
        'password': password
    }
    # Error handling parameters
    try:
        net_connect = Netmiko(**ios_device)
    except (AuthenticationException):
        print ('Authentication failure: ' + ip_address_of_device)
        continue
    except (NetMikoTimeoutException):
        print ('Timeout to device: ' + ip_address_of_device)
        continue
    except (EOFError):
        print ("End of file while attempting device " + ip_address_of_device)
        continue
    except (SSHException):
        print ('SSH Issue. Are you sure SSH is enabled? ' + ip_address_of_device)
        continue
    except Exception as unknown_error:
        print ('Some other error: ' + str(unknown_error))
        continue
```

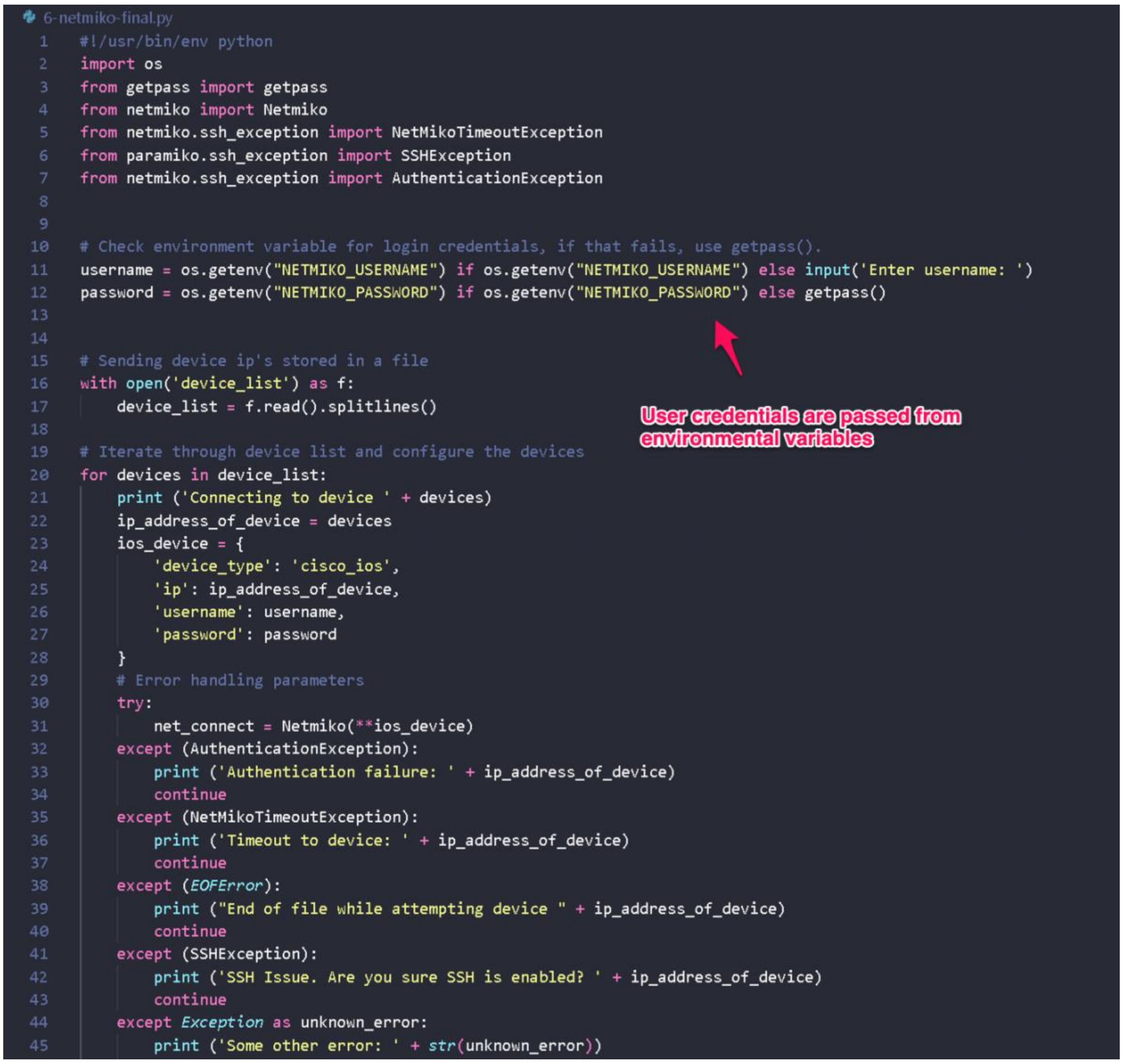
Provide wrong or random credentials and verify script is not breaking in middle of execution. The output is printed to the Terminal.

Task 6: Passing user credentials using environmental variables

In this task we will pass the user credentials using environmental variables. These credentials are stored in the user .bash_profile and loaded by default. By passing the credentials this way, we can safely run the scripts in lab or production networks for network backup jobs or collecting facts, inventory and operational data.

Launch the python script from the VS Code Terminal window

```
python 6-netmiko-final.py
```



6-netmiko-final.py

```python
#!/usr/bin/env python
import os
from getpass import getpass
from netmiko import Netmiko
from netmiko.ssh_exception import NetMikoTimeoutException
from paramiko.ssh_exception import SSHException
from netmiko.ssh_exception import AuthenticationException


# Check environment variable for login credentials, if that fails, use getpass().
username = os.getenv("NETMIKO_USERNAME") if os.getenv("NETMIKO_USERNAME") else input('Enter username: ')
password = os.getenv("NETMIKO_PASSWORD") if os.getenv("NETMIKO_PASSWORD") else getpass()


# Sending device ip's stored in a file
with open('device_list') as f:
    device_list = f.read().splitlines()

# Iterate through device list and configure the devices
for devices in device_list:
    print ('Connecting to device ' + devices)
    ip_address_of_device = devices
    ios_device = {
        'device_type': 'cisco_ios',
        'ip': ip_address_of_device,
        'username': username,
        'password': password
    }
    # Error handling parameters
    try:
        net_connect = Netmiko(**ios_device)
    except (AuthenticationException):
        print ('Authentication failure: ' + ip_address_of_device)
        continue
    except (NetMikoTimeoutException):
        print ('Timeout to device: ' + ip_address_of_device)
        continue
    except (EOFError):
        print ("End of file while attempting device " + ip_address_of_device)
        continue
    except (SSHException):
        print ('SSH Issue. Are you sure SSH is enabled? ' + ip_address_of_device)
        continue
    except Exception as unknown_error:
        print ('Some other error: ' + str(unknown_error))
```

Verify script can run without providing login credentials. The output is printed to the Terminal.

If the script still requests for username and password, please reload the environmental variables from the bash profile.

```
Run the command source ~/.bash_profile and rerun python 6-netmiko-final.py
You can view the file details using command cat ~/.bash_profile
```

Task 7: Use case to collect show command for network validation

Now we have a complete python script that can login into the device and collect data from the device or make configuration changes, based on the Netmiko methods we call. This script helps us to quickly login into multiple devices and collect the show commands for network validation or pre-post command validation during change windows.

Launch the python script from the VS Code Terminal window

```
python 7-netmiko-final.py
```

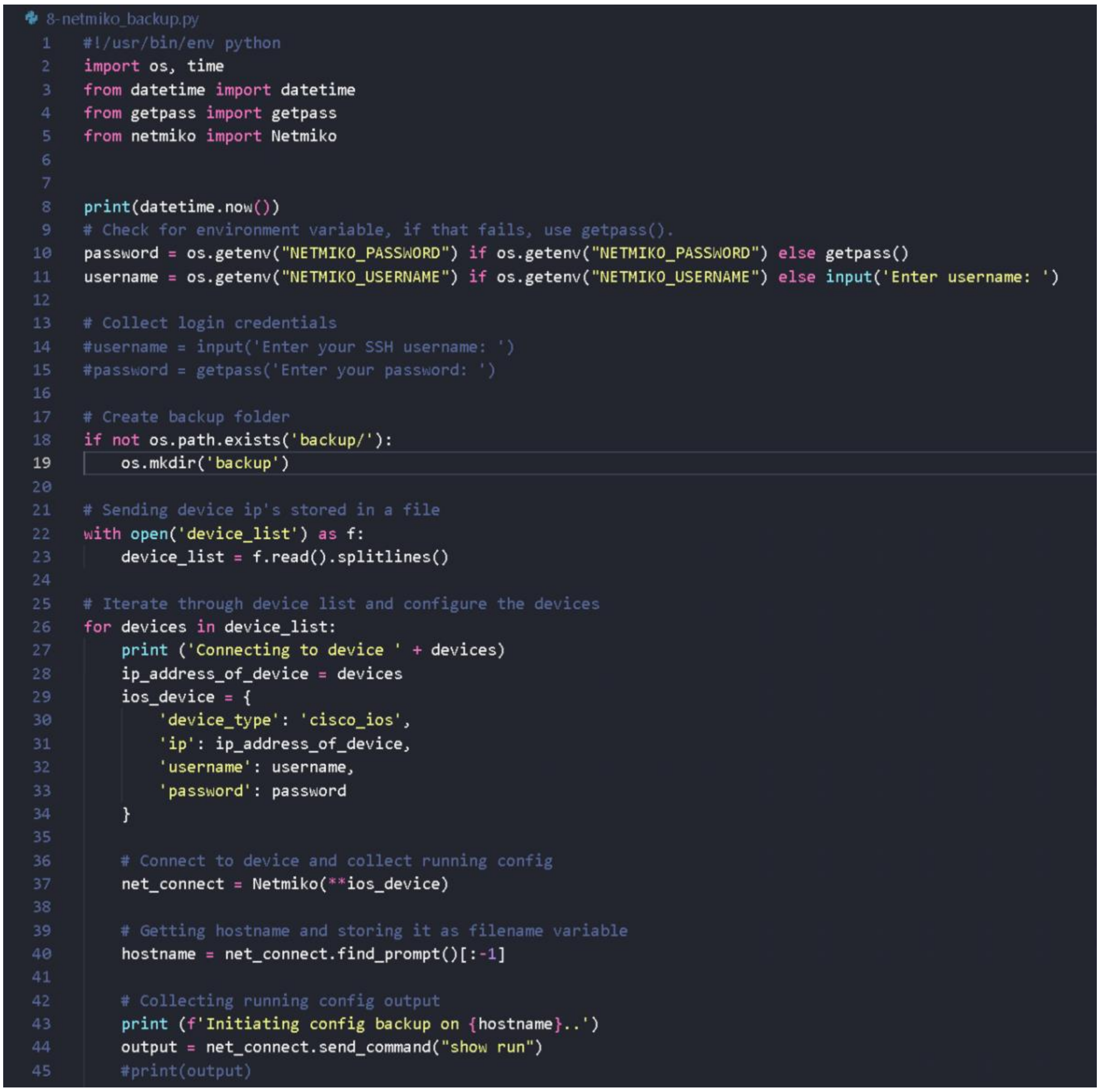
Verify the show command output printed to the Terminal.

Task 8: Use case - Automate network device configuration backup

Automating network configuration backup is one of the primary day-to-day tasks for any network engineer. This python script will help to achieve that task. We are reusing most of the code that we created in previous examples. The script uses os module to create a backup folder and save the running configuration to individual files with device hostname as filename.

Launch the python script from the VS Code Terminal window

```
python 8-netmiko-backup.py
```



8-netmiko_backup.py

```python
#!/usr/bin/env python
import os, time
from datetime import datetime
from getpass import getpass
from netmiko import Netmiko


print(datetime.now())
# Check for environment variable, if that fails, use getpass().
password = os.getenv("NETMIKO_PASSWORD") if os.getenv("NETMIKO_PASSWORD") else getpass()
username = os.getenv("NETMIKO_USERNAME") if os.getenv("NETMIKO_USERNAME") else input('Enter username: ')

# Collect login credentials
#username = input('Enter your SSH username: ')
#password = getpass('Enter your password: ')

# Create backup folder
if not os.path.exists('backup/'):
    os.mkdir('backup')

# Sending device ip's stored in a file
with open('device_list') as f:
    device_list = f.read().splitlines()

# Iterate through device list and configure the devices
for devices in device_list:
    print ('Connecting to device ' + devices)
    ip_address_of_device = devices
    ios_device = {
        'device_type': 'cisco_ios',
        'ip': ip_address_of_device,
        'username': username,
        'password': password
    }

    # Connect to device and collect running config
    net_connect = Netmiko(**ios_device)

    # Getting hostname and storing it as filename variable
    hostname = net_connect.find_prompt()[:-1]

    # Collecting running config output
    print (f'Initiating config backup on {hostname}..')
    output = net_connect.send_command("show run")
    #print(output)
```

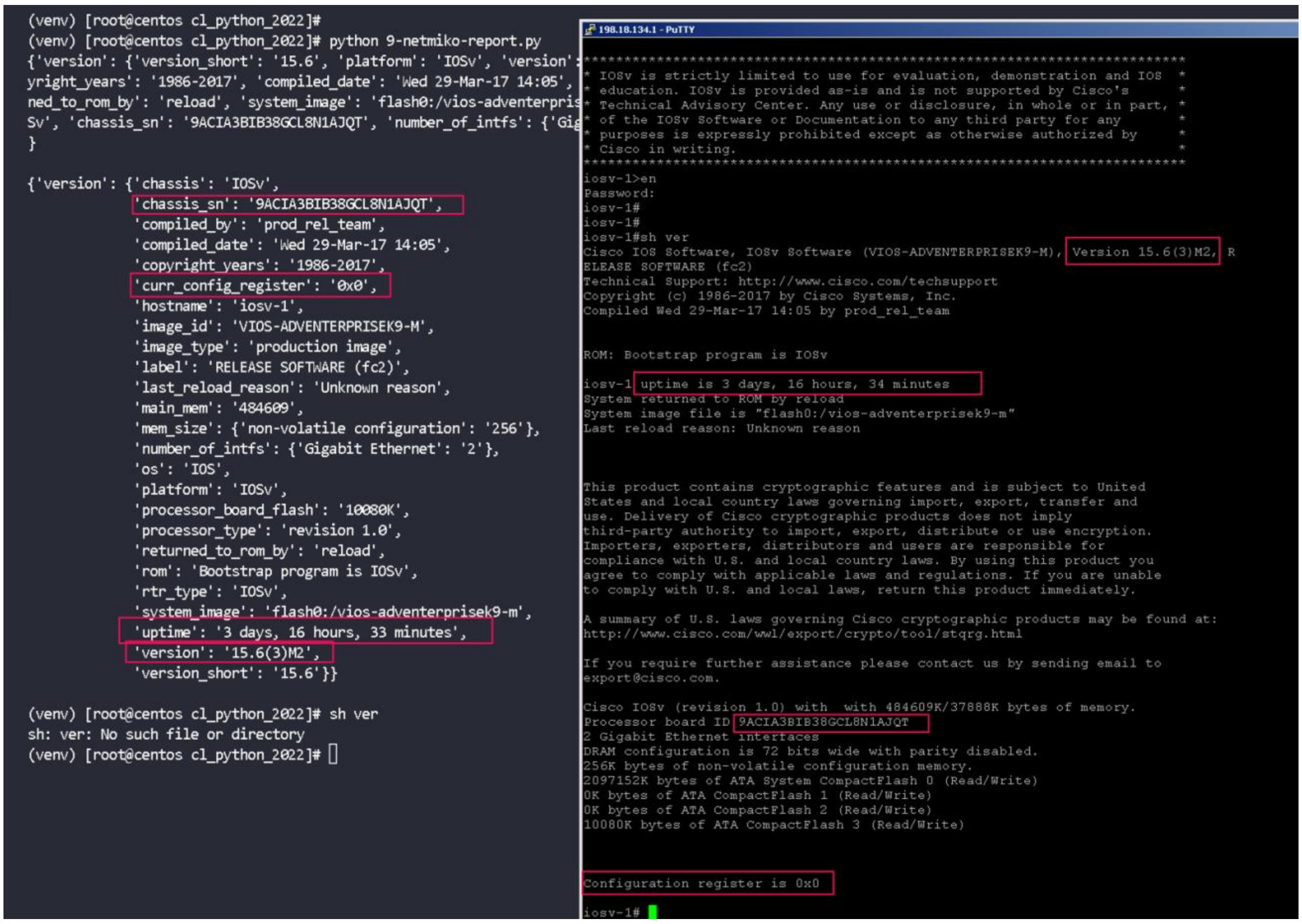
Verify the configuration backup files created by the script in the “backup” folder. The files will have a .cfg extension

Task 9: Using pyats / genie parsers

In this python script will use cisco pyats and genie parsers to extract valuable data out of show command outputs. Genie python library provides structured data output by parsing devices configuration and operation data for more than 1000+ cli commands. Netmiko natively supports genie parsers, so it makes our job easier to build network inventories and reports by leveraging genie, instead writing our own regex parsers to extract key:value data out of command output.

Launch the python script from the VS Code Terminal window

```
python 9-netmiko-report.py
```



```
(venv) [root@centos cl_python_2022]#
(venv) [root@centos cl_python_2022]# python 9-netmiko-report.py
{'version': {'version_short': '15.6', 'platform': 'IOSv', 'version':
yright_years': '1986-2017', 'compiled_date': 'Wed 29-Mar-17 14:05',
ned_to_rom_by': 'reload', 'system_image': 'flash0:/vios-adventerpris
Sv', 'chassis_sn': '9ACIA3BIB38GCL8N1AJQT', 'number_of_intfs': {'Gig
}

{'version': {'chassis': 'IOSv',
             'chassis_sn': '9ACIA3BIB38GCL8N1AJQT',
             'compiled_by': 'prod_rel_team',
             'compiled_date': 'Wed 29-Mar-17 14:05',
             'copyright_years': '1986-2017',
             'curr_config_register': '0x0',
             'hostname': 'iosv-1',
             'image_id': 'VIOS-ADVENTERPRISEK9-M',
             'image_type': 'production image',
             'label': 'RELEASE SOFTWARE (fc2)',
             'last_reload_reason': 'Unknown reason',
             'main_mem': '484609',
             'mem_size': {'non-volatile configuration': '256'},
             'number_of_intfs': {'Gigabit Ethernet': '2'},
             'os': 'IOS',
             'platform': 'IOSv',
             'processor_board_flash': '10080K',
             'processor_type': 'revision 1.0',
             'returned_to_rom_by': 'reload',
             'rom': 'Bootstrap program is IOSv',
             'rtr_type': 'IOSv',
             'system_image': 'flash0:/vios-adventerprisek9-m',
             'uptime': '3 days, 16 hours, 33 minutes',
             'version': '15.6(3)M2',
             'version_short': '15.6'}}

(venv) [root@centos cl_python_2022]# sh ver
sh: ver: No such file or directory
(venv) [root@centos cl_python_2022]# 
```

```
*************************************************************************
* IOSv is strictly limited to use for evaluation, demonstration and IOS  *
* education. IOSv is provided as-is and is not supported by Cisco's      *
* Technical Advisory Center. Any use or disclosure, in whole or in part, *
* of the IOSv Software or Documentation to any third party for any       *
* purposes is expressly prohibited except as otherwise authorized by     *
* Cisco in writing.                                                      *
*************************************************************************
iosv-1>en
Password:
iosv-1#
iosv-1#
iosv-1#sh ver
Cisco IOS Software, IOSv Software (VIOS-ADVENTERPRISEK9-M), Version 15.6(3)M2, R
ELEASE SOFTWARE (fc2)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2017 by Cisco Systems, Inc.
Compiled Wed 29-Mar-17 14:05 by prod_rel_team


ROM: Bootstrap program is IOSv

iosv-1 uptime is 3 days, 16 hours, 34 minutes
System returned to ROM by reload
System image file is "flash0:/vios-adventerprisek9-m"
Last reload reason: Unknown reason



This product contains cryptographic features and is subject to United
States and local country laws governing import, export, transfer and
use. Delivery of Cisco cryptographic products does not imply
third-party authority to import, export, distribute or use encryption.
Importers, exporters, distributors and users are responsible for
compliance with U.S. and local country laws. By using this product you
agree to comply with applicable laws and regulations. If you are unable
to comply with U.S. and local laws, return this product immediately.

A summary of U.S. laws governing Cisco cryptographic products may be found at:
http://www.cisco.com/wwl/export/crypto/tool/stqrg.html

If you require further assistance please contact us by sending email to
export@cisco.com.

Cisco IOSv (revision 1.0) with  with 484609K/37888K bytes of memory.
Processor board ID 9ACIA3BIB38GCL8N1AJQT
2 Gigabit Ethernet interfaces
DRAM configuration is 72 bits wide with parity disabled.
256K bytes of non-volatile configuration memory.
2097152K bytes of ATA System CompactFlash 0 (Read/Write)
0K bytes of ATA CompactFlash 1 (Read/Write)
0K bytes of ATA CompactFlash 2 (Read/Write)
10080K bytes of ATA CompactFlash 3 (Read/Write)


Configuration register is 0x0

iosv-1#
```

The screenshot shows how genie parsers work in the background to extract the values that is readily available for us to build report or validate operational data. If you try to write you own regex parsers to extract hostname, serial no, uptime, software version out of the raw text-based show command output, the python script can become very big and tedious to manage. To call the parsers written in genie, all we want to do is add the keyword use_genie=True to our script.

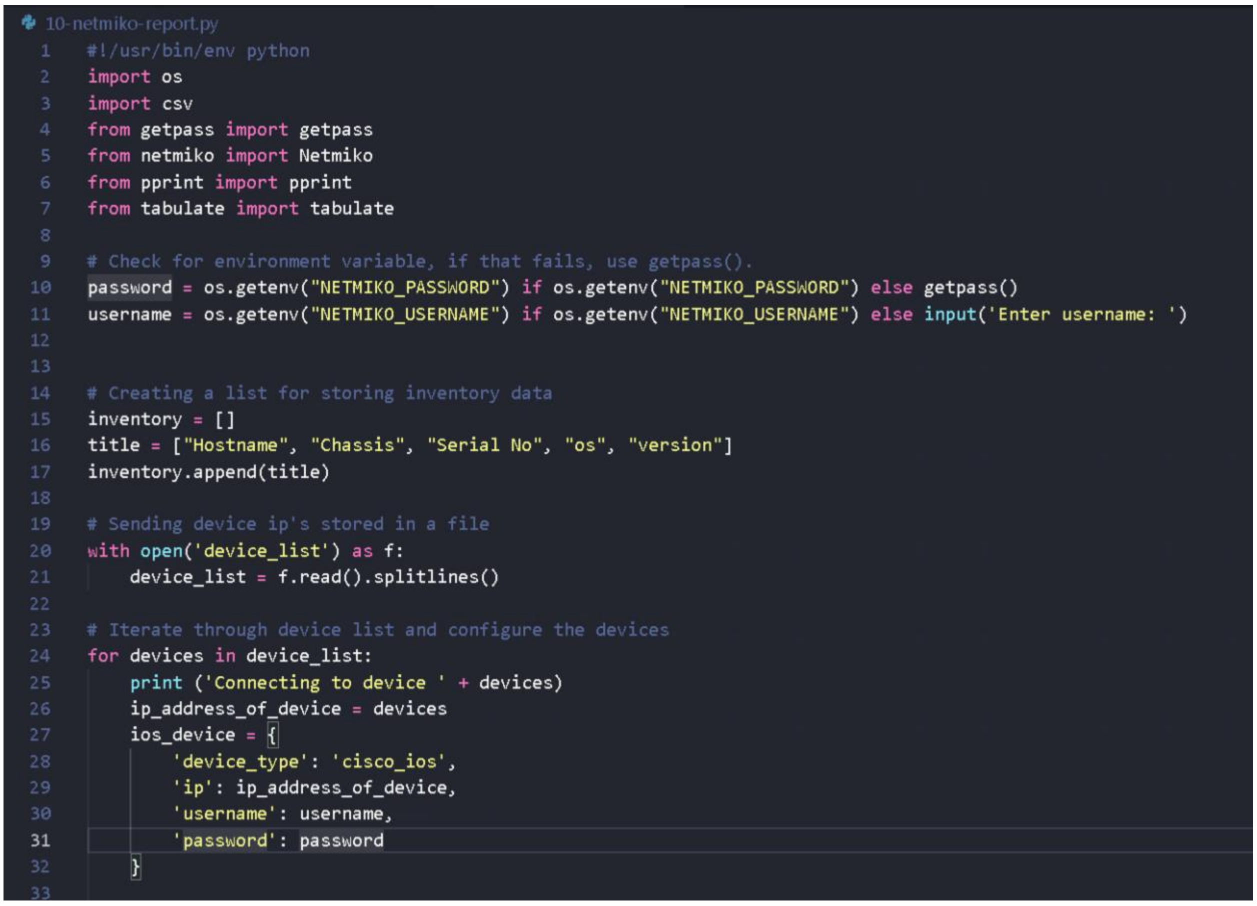
```
output = net_connect.send_command('show version', use_genie=True)
```

Task 10: Use case – Using PYATS / Genie parsers to generate reports

Keeping the network inventory information up to date or Automating network report creation is another use case that can greatly help network engineers. In the previous example we saw how to leverage genie parsers and extract the data out of show commands. In this example we will use the output from genie to build inventory reports or quickly validate operational data from multiple devices within minutes.

Launch the python script from the VS Code Terminal window

```
python 10-netmiko-report.py
```



10-netmiko-report.py

```
#!/usr/bin/env python
import os
import csv
from getpass import getpass
from netmiko import Netmiko
from pprint import pprint
from tabulate import tabulate

# Check for environment variable, if that fails, use getpass().
password = os.getenv("NETMIKO_PASSWORD") if os.getenv("NETMIKO_PASSWORD") else getpass()
username = os.getenv("NETMIKO_USERNAME") if os.getenv("NETMIKO_USERNAME") else input('Enter username: ')


# Creating a list for storing inventory data
inventory = []
title = ["Hostname", "Chassis", "Serial No", "os", "version"]
inventory.append(title)

# Sending device ip's stored in a file
with open('device_list') as f:
    device_list = f.read().splitlines()

# Iterate through device list and configure the devices
for devices in device_list:
    print ('Connecting to device ' + devices)
    ip_address_of_device = devices
    ios_device = {
        'device_type': 'cisco_ios',
        'ip': ip_address_of_device,
        'username': username,
        'password': password
    }

```

Verify the script output printed to the Terminal. First, we have built a table to easily view the data. Second, we have used the same data to create an inventory report. Open the inventory.csv file and verify the data.

Building blocks of Nornir python framework

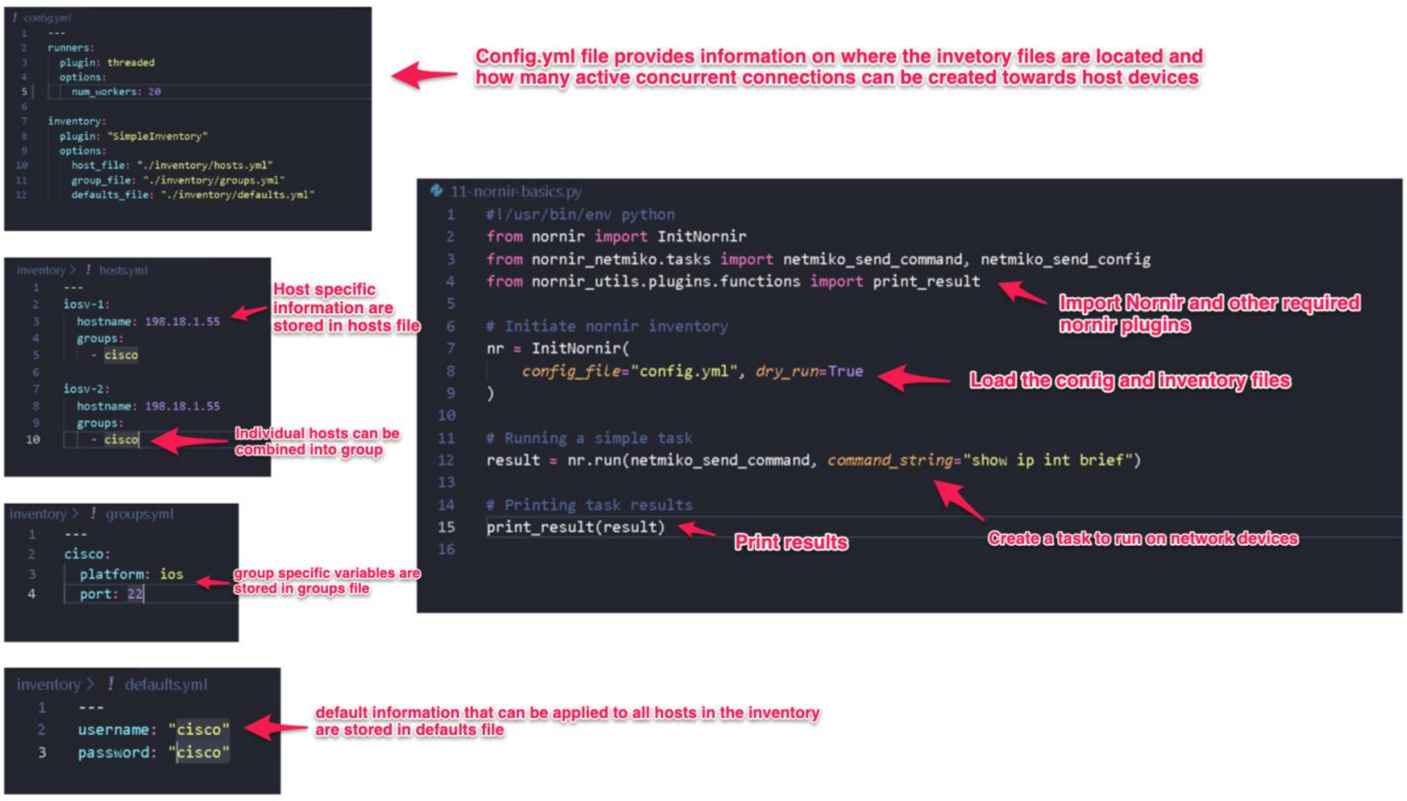
Nornir is a python based multithreaded framework with inbuilt inventory management. It is very similar to Ansible in the way it operates. The inventory details are stored in YAML file format, it is agentless, but instead of writing playbooks in YAML format, we write the nornir scripts in pure python.

To run Nornir we just run the Nornir Python script via Python. At this point, Nornir will connect to the devices within the inventory and perform the necessary actions that have been defined within our Nornir tasks.

The main advantages for using nornir are:

- the readily available inventory system, which helps us to scale better and maintain without much effort.
- Parallel execution of the jobs against many devices will greatly save time, when you are trying to automate a usecase across hundreds of devices.

Below is an overview of how Nornir script operates


Config.yml file provides information on where the invetory files are located and how many active concurrent connections can be created towards host devices
Host specific information are stored in hosts file
Individual hosts can be combined into group
group specific variables are stored in groups file
default information that can be applied to all hosts in the inventory are stored in defaults file
Import Nornir and other required nornir plugins
Load the config and inventory files
Print results
Create a task to run on network devices

config.yml
```
---
runners:
  plugin: threaded
  options:
    num_workers: 20

inventory:
  plugin: "SimpleInventory"
  options:
    host_file: "./inventory/hosts.yml"
    group_file: "./inventory/groups.yml"
    defaults_file: "./inventory/defaults.yml"
```

inventory > hosts.yml
```
---
iosv-1:
  hostname: 198.18.1.55
  groups:
    - cisco

iosv-2:
  hostname: 198.18.1.55
  groups:
    - cisco
```

inventory > groups.yml
```
---
cisco:
  platform: ios
  port: 22
```

inventory > defaults.yml
```
---
username: "cisco"
password: "cisco"
```

11-nornir-basics.py
```
#!/usr/bin/env python
from nornir import InitNornir
from nornir_netmiko.tasks import netmiko_send_command, netmiko_send_config
from nornir_utils.plugins.functions import print_result

# Initiate nornir inventory
nr = InitNornir(
    config_file="config.yml", dry_run=True
)

# Running a simple task
result = nr.run(netmiko_send_command, command_string="show ip int brief")

# Printing task results
print_result(result)
```

For more information on nornir, please refer to nornir documentation
https://nornir.readthedocs.io/en/latest/

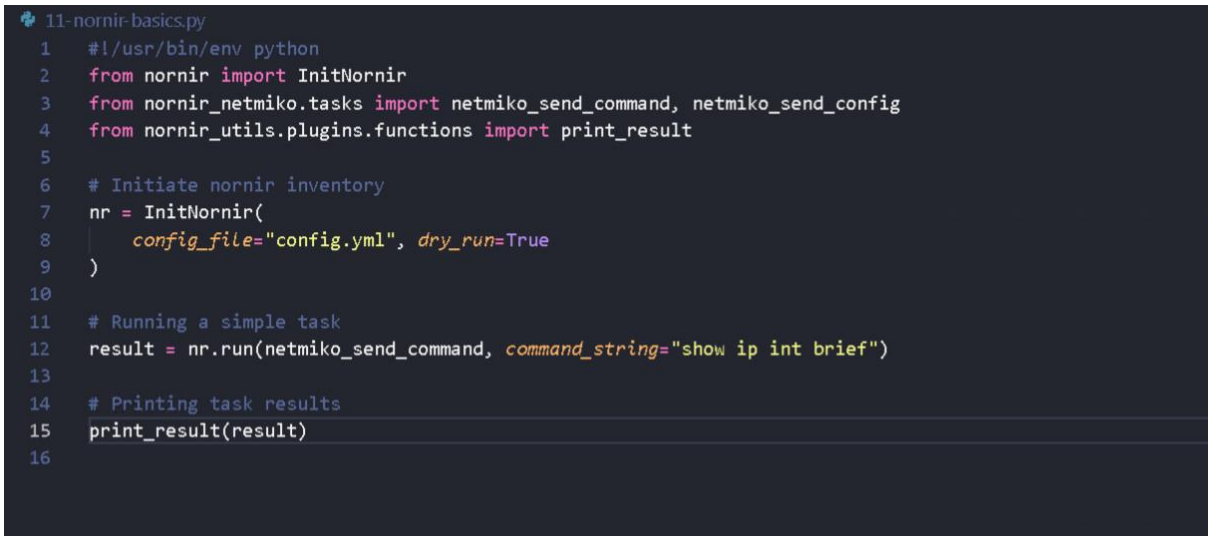
Task 11: Execute basic Nornir script

The last two scripts on this lab shows how to leverage nornir framework and automate network tasks. The scripts are very similar to how we developed the pervious Netmiko scripts. We have three import statements, which loads the required plugins for nornir to operate. We are using Netmiko as a plugin in nornir, there are many plugins we can call into Nornir to extends its functionality.

Next we initiate the nornir related configurations using InitNornir( ) function. Use the .run( ) function to execute the tasks and print the results to terminal using print_result( ) function.

Launch the python script from the VS Code Terminal window

`python 11-nornir-basics.py`



11-nornir-basics.py

```python
#!/usr/bin/env python
from nornir import InitNornir
from nornir_netmiko.tasks import netmiko_send_command, netmiko_send_config
from nornir_utils.plugins.functions import print_result

# Initiate nornir inventory
nr = InitNornir(
    config_file="config.yml", dry_run=True
)

# Running a simple task
result = nr.run(netmiko_send_command, command_string="show ip int brief")

# Printing task results
print_result(result)
```

Verify the show command output printed to the Terminal. Notice Nornir is executing the tasks in parallel, it is not working in serially by logging in one device at a time and printing the results.

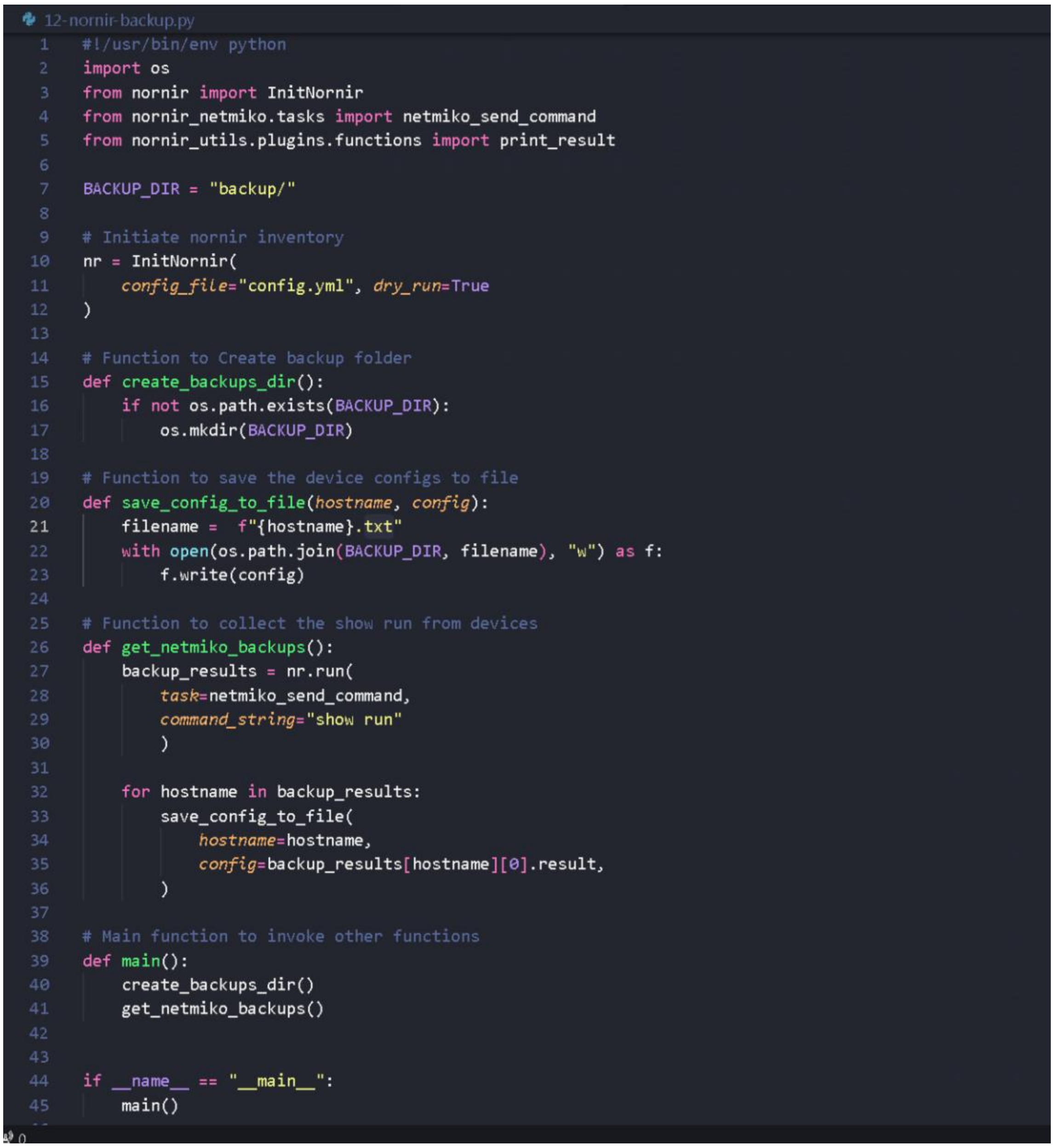
Task 12: Execute final Nornir script

The final script shows an example of how a real-world modular python script looks like. It will have multiple import statements, a main function to invoke all child functions and the child functions will complete the tasks it is programmed to do. This script is developed to automate the network configuration backup task, the primary difference between Netmiko vs Nornir version of backup script is scalability and speed of execution. The python script will

- Create a backup folder, if one does not exist
- Login into the host devices and collect the show running config concurrently
- Save the output to a file in the backup folder

Launch the python script from the VS Code Terminal window

```
python 12-nornir-backup.py
```


12-nornir-backup.py

```python
#!/usr/bin/env python
import os
from nornir import InitNornir
from nornir_netmiko.tasks import netmiko_send_command
from nornir_utils.plugins.functions import print_result

BACKUP_DIR = "backup/"

# Initiate nornir inventory
nr = InitNornir(
    config_file="config.yml", dry_run=True
)

# Function to Create backup folder
def create_backups_dir():
    if not os.path.exists(BACKUP_DIR):
        os.mkdir(BACKUP_DIR)

# Function to save the device configs to file
def save_config_to_file(hostname, config):
    filename =  f"{hostname}.txt"
    with open(os.path.join(BACKUP_DIR, filename), "w") as f:
        f.write(config)

# Function to collect the show run from devices
def get_netmiko_backups():
    backup_results = nr.run(
        task=netmiko_send_command,
        command_string="show run"
        )

    for hostname in backup_results:
        save_config_to_file(
            hostname=hostname,
            config=backup_results[hostname][0].result,
        )

# Main function to invoke other functions
def main():
    create_backups_dir()
    get_netmiko_backups()


if __name__ == "__main__":
    main()
```

Verify the config backup files created in the backup folder with .txt extension

Conclusion

You have successfully completed the lab. Now you should be able to realize the power of python programming, where you initially started with very simple python scripts, then quickly modified it to build additional logic and error handling to automate the day-to-day network admin tasks.

We also saw couple of real-world network use-cases on how to leverage python to automate network configuration backup and create inventory reports. This is just an introduction to python, the reference materials will help you to become a better programmer; learn and understand the programming and automation concepts in detail. Hope we have given you a head start with your Network automation and programmability journey.

Thanks for attending this lab, please share your valuable feedback.

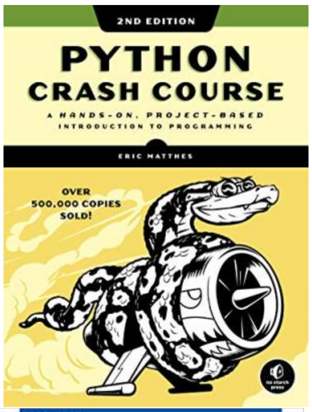
Note: Please don't log off the windows user session. Just close the RDP window

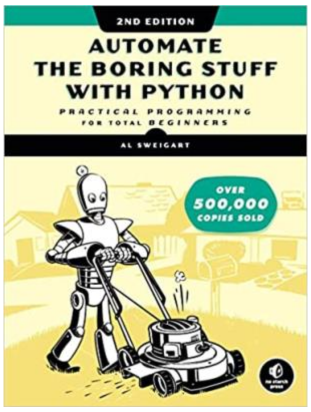
Run the 99-netmiko-resetlab.py script to reset the lab back to default config

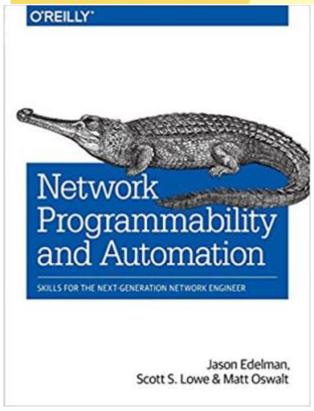
Reference

Additional Learning and Reference materials:
https://developer.cisco.com/video/net-prog-basics/
https://developer.cisco.com/learning/modules/dne-intro-python/
https://pynet.twb-tech.com/free-python-course.html
https://my.thisisit.io/p/52-weeks-of-python

Recommended Reading:
https://www.amazon.com/Network-Programmability-Automation-Next-Generation-Engineer/dp/1491931256
https://www.amazon.com/Python-Crash-Course-2nd-Edition/dp/1593279280
https://www.amazon.com/Automate-Boring-Stuff-Python-2nd/dp/1593279922


2ND EDITION
PYTHON CRASH COURSE
A HANDS-ON, PROJECT-BASED INTRODUCTION TO PROGRAMMING
ERIC MATTHES
OVER 500,000 COPIES SOLD!


2ND EDITION
AUTOMATE THE BORING STUFF WITH PYTHON
PRACTICAL PROGRAMMING FOR TOTAL BEGINNERS
AL SWEIGART
OVER 500,000 COPIES SOLD


O'REILLY
Network Programmability and Automation
SKILLS FOR THE NEXT-GENERATION NETWORK ENGINEER
Jason Edelman, Scott S. Lowe & Matt Oswalt

Note: Most of the famous python books will be available in your local community library to borrow and read