Efficient Network Automation with Nornir and Napalm

Neelima Parakala

Technical Marketing Engineer

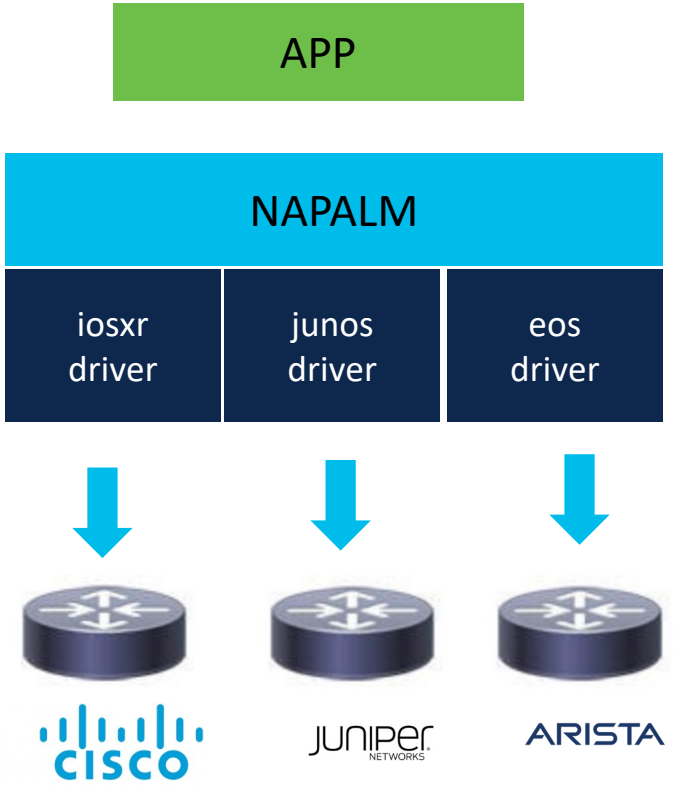
AGENDA

1. NAPALM - What, Why and How
2. Nornir - What, Why and How
3. Execute Napalm API's using Nornir framework
4. Demo

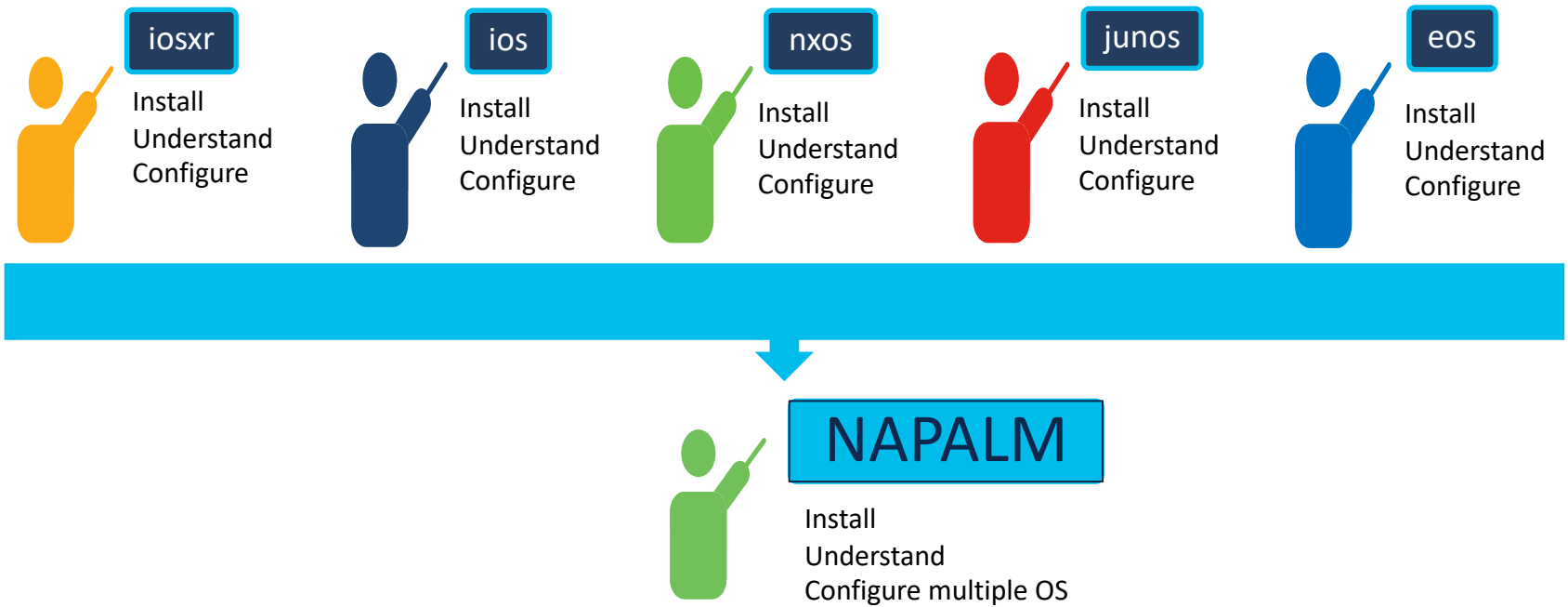
NAPALM
What, Why and How

What is NAPALM ?

- Network Automation and Programmability Abstraction Layer with Multivendor support
- Napalm is a **vendor neutral**, cross-platform open source project that provides a unified API
- NAPALM is a python library that provides a set of functions for configuration management and operational data retrieval
- Cisco IOS-XR, Cisco IOS, Cisco NX-OS, Junos and Arista EOS
- Other platforms supported by the community https://github.com/orgs/napalm-automation-community/repositories



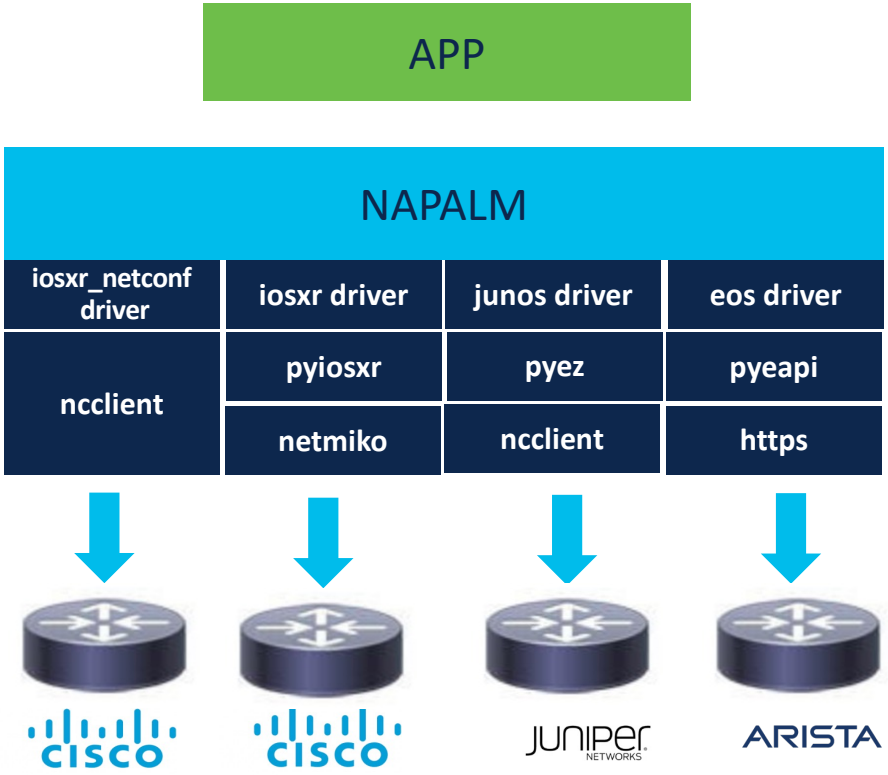
Why NAPALM ?


iosxr
Install
Understand
Configure
ios
Install
Understand
Configure
nxos
Install
Understand
Configure
junos
Install
Understand
Configure
eos
Install
Understand
Configure
NAPALM
Install
Understand
Configure multiple OS

Why is it important ?

How NAPALM works ? (1/2)

- Napalm is the base class, it defines the abstract API names and their input (API arguments) and output(API resultant data) formats
- It has multiple drivers (junos, eos, iosxr, iosxr_netconf, ios etc.) for the respective operating system of the network devices
- These drivers implement the abstract API's defined in the Napalm base class
- Drivers use their existing packages (pyiosxr, pyez, pyeapi etc.) to load and retrieve data from the network devices



How NAPALM works ? (2/2)

- Inheritance and Abstraction
- Same API's and output dictionary across the drivers
- Simple data structure and type validation for dictionaries (no formal model/schema)

IOSXR

```
{
  "uptime": 35457914,
  "vendor": "Cisco",
  "hostname": "edge01.tab",
  "fqdn": "edge01.tab01",
  "os_version": "5.3.1",
  "serial_number": "FOX171",
  "model": "ASR-9904-AC”,
  "interface_list": [
      "TenGigE0/0/0/13",
      "TenGigE0/0/0/14",
      "TenGigE0/0/0/24”
    ]
}
```

JUNOS

```
{
  "uptime": 4380,
  "vendor": "Juniper",
  "hostname": "vsrx",
  "fqdn": "vsrx",
  "os_version": "12.1X4",
  "serial_number": "beb91",
  "model": "FIREFLY",
  "interface_list": [
       "ge-0/0/0",
       "ge-0/0/1",
       "ge-0/0/2”
    ]
}
```

IOS

```
{
  "uptime": 16676160,
  "vendor": "Cisco",
  "hostname": "NS2903",
  "fqdn": "NS2903-ASW"
  "os_version": "15.0(2)”,
  "serial_number": "FOC1",
  "model": "WS-C2960G",
  "interface_list": [
      "Vlan1",
      "GigabitEthernet0/1”,
      "GigabitEthernet0/5”
    ]
}
```

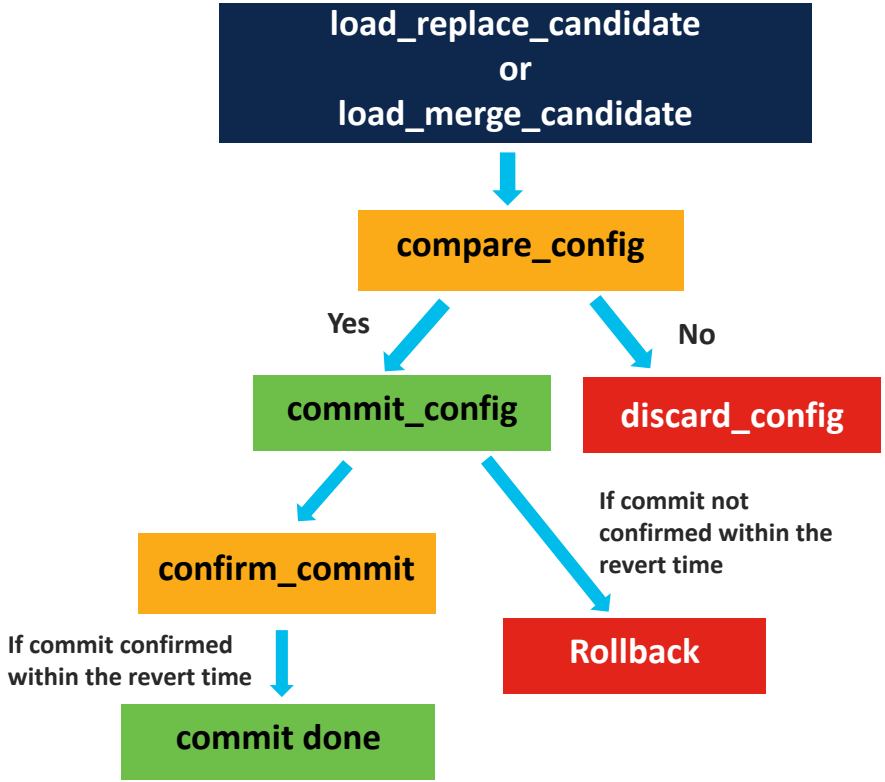
EOS

```
{
  "uptime": 123456,
  "vendor": "Arista”,
"hostname": "localhost",
  "fqdn": "localhost",
  "os_version": "4.15.5M",
  "serial_number": "",
  "model": "vEOS",
  "interface_list": [
    "Ethernet1",
    "Ethernet2",
    "Ethernet3",
    "Management1
  ],
}
```

NAPALM API Overview (1/2)

Configuration Data Management

Functions
load_replace_candidate
load_merge_candidate
compare_config
commit_config
discard_config
confirm_commit
rollback



NAPALM API Overview (2/2)

Operational Data Management

Functions	Functions	Functions	Functions
get_facts	get_route_to	get_arp_table	get_environment
get_interfaces	get_snmp_information	get_ntp_peers	cli
get_interfaces_counters	get_probes_config	get_ntp_servers	get_firewall_policies
get_interfaces_ip	get_probes_results	get_ntp_stats	get_ipv6_neighbors_table
get_bgp_config	traceroute	get_lldp_neighbors	get_network_instances
get_bgp_neighbors	get_users	get_lldp_neighbors_detail	get_optics
get_bgp_neighbors_detail	get_config	get_mac_address_table	ping

How to use NAPALM Python Library ?

1. pip install napalm
2. Write a script to retrieve or load data

Manage configuration and operational data

```
from napalm import get_network_driver

driver =
get_network_driver("driver_name")

device = driver(hostname="carreras",
                username="device",
                password= "******",
          optional_args={"port":830})

device.open()
print(device.get_interfaces())
```

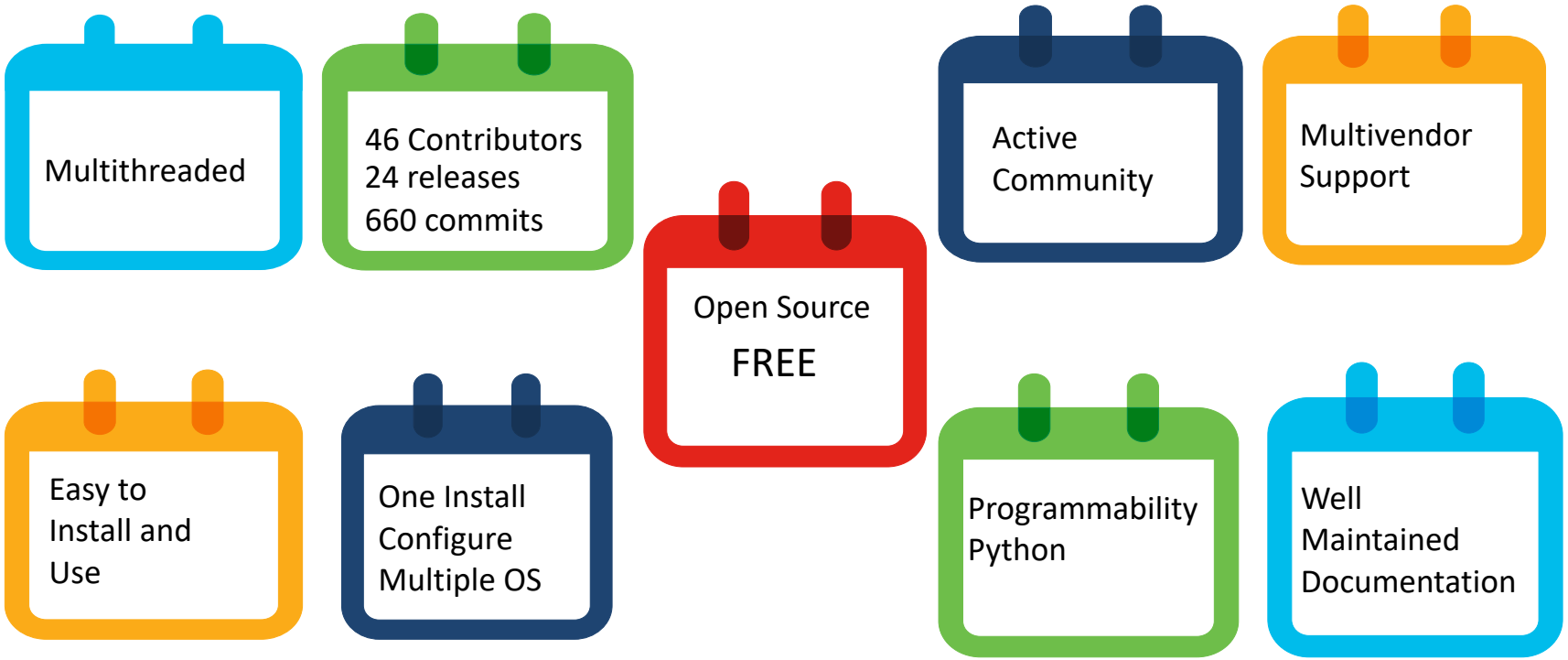

```
{
        "TenGigE0/0/0/14": {
                "is_enabled": true,
                "description": "",
                "last_flapped": -1.0,
                "is_up": false,
                "mac_address": "E0:AC:F1:64:71:52",
                "mtu": 1514,
                "speed": 10000
        },
        "TenGigE0/0/0/24": {
                "is_enabled": false,
                "description": "",
                "last_flapped": -1.0,
                "is_up": false,
                "mac_address": "E0:AC:F1:64:71:5C",
                "mtu": 1514,
                "speed": 10000
        }
}
```

Nornir
What, Why and How

What is Nornir ?

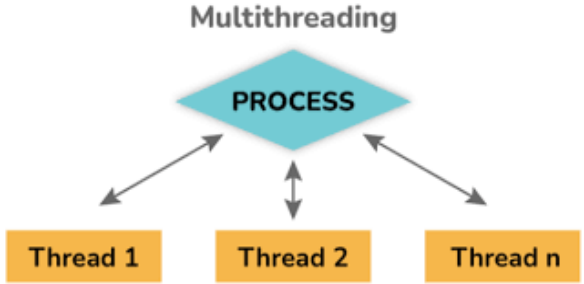
- Nornir is a multi-threaded network automation framework that abstracts inventory and task execution
- It helps to automate your network tasks efficiently
- You can execute the tasks like configuring the devices, validating the operational data, and enabling the services on the provided hosts which are part of the inventory
- It is multi-threaded and allows you to manage the configuration of multiple network devices concurrently
- It is an open-source project, completely written in python and easy to use

Why Nornir ?


Multithreaded
46 Contributors
24 releases
660 commits
Active
Community
Multivendor
Support
Open Source
FREE
Easy to
Install and
Use
One Install
Configure
Multiple OS
Programmability
Python
Well
Maintained
Documentation

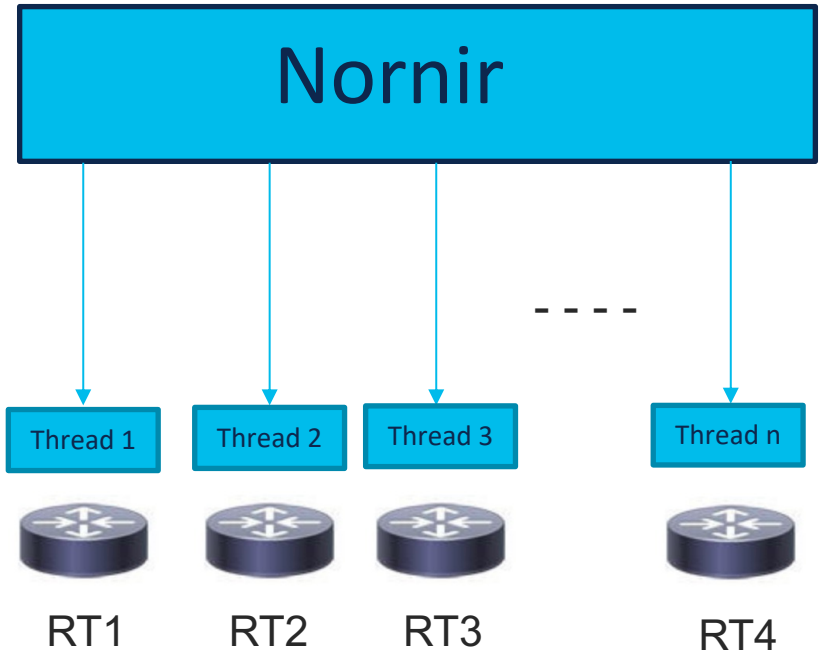
Why is it important ?

- You can develop features on top of the Nornir framework based on your requirement
- As Nornir is completely written in python, it is easy to
  - install
  - write code
  - integrate with any other python frameworks (Flask, Django, Pytest)
  - troubleshoot and debug the issues using python debug tools
- It reuses existing python libraries like Netmiko and Napalm to connect and manage the devices
- The use of multithreading greatly optimizes the execution time of the tasks
- You can effectively manage the hosts and groups separately as part of the inventory



How Nornir works ?

- Nornir works with the collection of host details
- It runs tasks against the hosts and keeps track of all the threads
- In a network environment, this typically means that you have a host inventory with data associated with each node
- You can define tasks, and those tasks use nornir-plugins to accomplish their work
- Nornir then ties everything together and lets you run those tasks against all, or a subset of, devices handling the data, concurrently and keeping track of the errors



How to use Nornir Framework ?

- **pip install nornir**
- Install Nornir plugin nornir-utils. It provides plugins like inventory, functions, processors, and tasks
  - **pip install nornir-utils**
- Once you have all the required packages installed, go ahead and write the code to retrieve, configure or validate device data
- Create the inventory files hosts.yaml, groups.yaml, and defaults.yaml
- Execute the python code to understand the schema of the objects (hosts, groups, defaults)

```
# hosts.yaml
---
rt1:
   hostname: 171.190.10.64
   groups:
        - iosxr
rt2:
   hostname: 10.30.11.170
   groups:
        - ios
```

```
# groups.yaml
---
iosxr:
    platform: 'iosxr'
ios:
    platform: 'ios'
```

```
# defaults.yaml
username: admin
password: admin
```

```
from nornir.core.inventory import Host, Group, Defaults
import json

print(json.dumps(Host.schema(), indent=4))
print(json.dumps(Group.schema(), indent=4))
print(json.dumps(Defaults.schema(), indent=4))
```

Execute Napalm API’s using Nornir framework

Prerequisites (1/3)

- Install Nornir plugin nornir-napalm
  - **pip install nornir-napalm**
- **hosts.yaml**
- **groups.yaml**
- **defaults.yaml**
- **config.yaml**
- **Python main file (nornir_main.py)**

hosts.yaml

```
# hosts.yaml
---
rt1:
   hostname: 171.190.10.64
   groups:
        - iosxr
rt2:
   hostname: 10.30.11.170
   groups:
        - ios
```

groups.yaml

```
# groups.yaml
---
iosxr:
    platform: 'iosxr'
ios:
    platform: 'ios'
```

defaults.yaml

```
# defaults.yaml
username: admin
password: admin
```

Prerequisites (2/3)

- Config file provides inventory and task concurrency information to the main file
- Nornir will use a different thread for each host to concurrently execute the tasks of the hosts
- You can provide the number of threads to be used by your code in the num_workers option of the runner plugin
- If num_workers == 1, and runner plugin is serial, then tasks of the hosts are executed sequentially
- This case helps to troubleshoot or debug the issues
- Generally, you can provide a number greater than 1 to num_workers else it defaults to 20
- In my case, I am assigning value 2 to num_workers, as I am dealing with two hosts

config.yaml

```
# config.yaml
---
inventory:
        plugin: SimpleInventory
        options:
             host_file: 'inventory/hosts.yaml'
             group_file: 'inventory/groups.yaml'
             defaults_file: 'inventory/defaults.yaml'
runner:
        plugin: threaded
        options:
             num_workers: 2
```

Prerequisites (3/3)

- This is the main file where you initialize Nornir with the InitNornir function and provide the configuration file
- In the next step, call a run method and provide the tasks to be executed, here we provided napalm_get, imported from the nornir_napalm plugin
- It executes the provided napalm getters over all the hosts provided in the inventory and returns the results

nornir_main.py

```
from nornir import InitNornir
from nornir_utils.plugins.functions import print_result
from nornir_napalm.plugins.tasks import napalm_get

nr = InitNornir(
    config_file="config.yaml", dry_run=True
)

results = nr.run(
    task=napalm_get, getters=["facts"]
)

print_result(results)
```

Execute the python file to retrieve results

- **python nornir_main.py**
- The output shows the facts (napalm getter) retrieved from the hosts provided in the inventory
- For every host, the tasks are executed separately by a thread, hence the results are shown per host
- It returns a dictionary for each host, with the key being the napalm getter name and value being the result of executing the getter method

```
napalm_get*************************************************************************
* rt1 ** changed : False ************************************************************
vvvv napalm_get ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
{ 'facts': { 'fqdn': 'pavarotti',
             'hostname': 'pavarotti',
             'interface_list': [ 'GigabitEthernet0/0/0/0',
                                 'GigabitEthernet0/0/0/1',
                                 'Loopback0',
                                 'MgmtEth0/RP0/CPU0/0',
                                 'Null0'],
             'model': 'R-IOSXRV9000-CC',
             'os_version': '6.5.3',
             'serial_number': 'E3FDA081DAC',
             'uptime': 18033322,
             'vendor': 'Cisco'}}
^^^^ END napalm_get ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* rt2 ** changed : False ************************************************************
vvvv napalm_get ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
{ 'facts': { 'fqdn': 'placido.placido.local',
             'hostname': 'placido',
             'interface_list': [ 'GigabitEthernet1',
                                 'GigabitEthernet2',
                                 'GigabitEthernet3'],
             'model': 'CSR1000V',
             'os_version': 'Virtual XE Software '
                           '(X86_64_LINUX_IOSD-UNIVERSALK9-M), Version 16.9.3, '
                           'RELEASE SOFTWARE (fc2)',
             'serial_number': '9NSHRXZD4TZ',
             'uptime': 43016280,
             'vendor': 'Cisco'}}
^^^^ END napalm_get ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Nornir-Napalm Plugins

Nornir-Napalm provides napalm connections through which you connect to the device and execute tasks like

- napalm_cli
- napalm_configure
- napalm_get
- napalm_ping
- napalm_validate

Please refer https://nornir.tech/nornir/plugins/ to learn more about Nornir plugins

Demo

Summary

- Napalm is a **vendor neutral**, cross-platform open-source project that provides a unified API to network devices
- NAPALM is free, easy to install, understand and use
- Nornir is a pluggable multi-threaded framework with inventory management to help operate collections of devices
- Nornir is multi-threaded and allows you to manage the configuration of multiple network devices concurrently
- Nornir is an open-source project, completely written in python and easy to use
- Install nornir-napalm plugin of Nornir to execute Napalm tasks concurrently, on multiple network devices

Resources

- NAPALM GitHub repository
- NAPALM documentation
- NCClient GitHub repository
- NCClient documentation
- NETCONF
- Netmiko GitHub repository
- Nornir Overview blog
- Nornir documentation
- Nornir GitHub repository

Thank You

Live in your own way with the best attitude.

- Neelima Parakala

Questions?