



Efficient Network Automation with Nornir and Napalm

Neelima Parakala

Technical Marketing Engineer

AGENDA

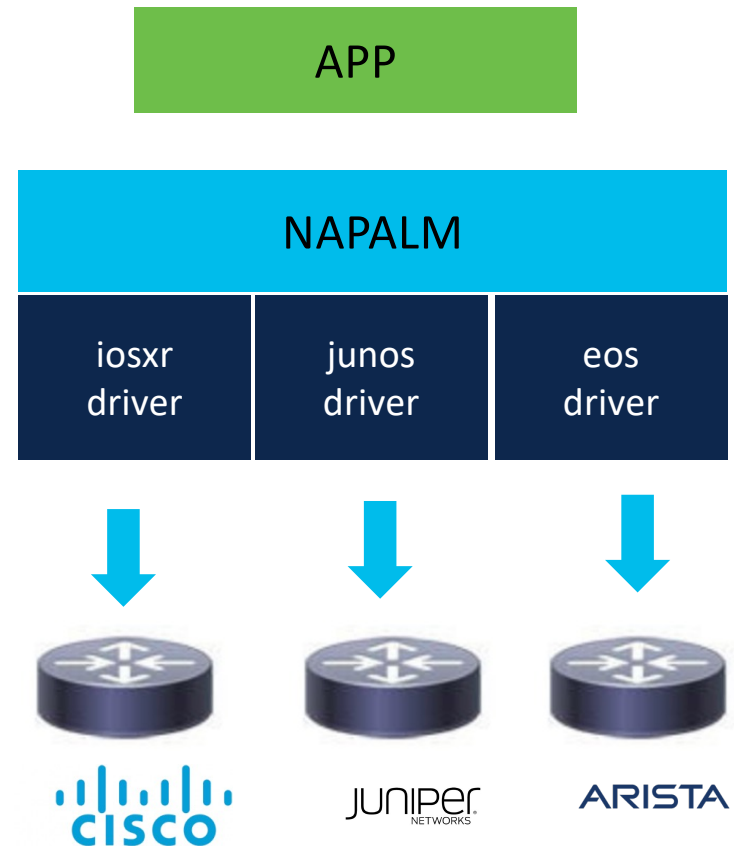
- 1 NAPALM - What, Why and How ?
- 2 Nornir - What, Why and How ?
- 3 Execute NAPALM API's using Nornir framework
- 4 Demo

NAPALM

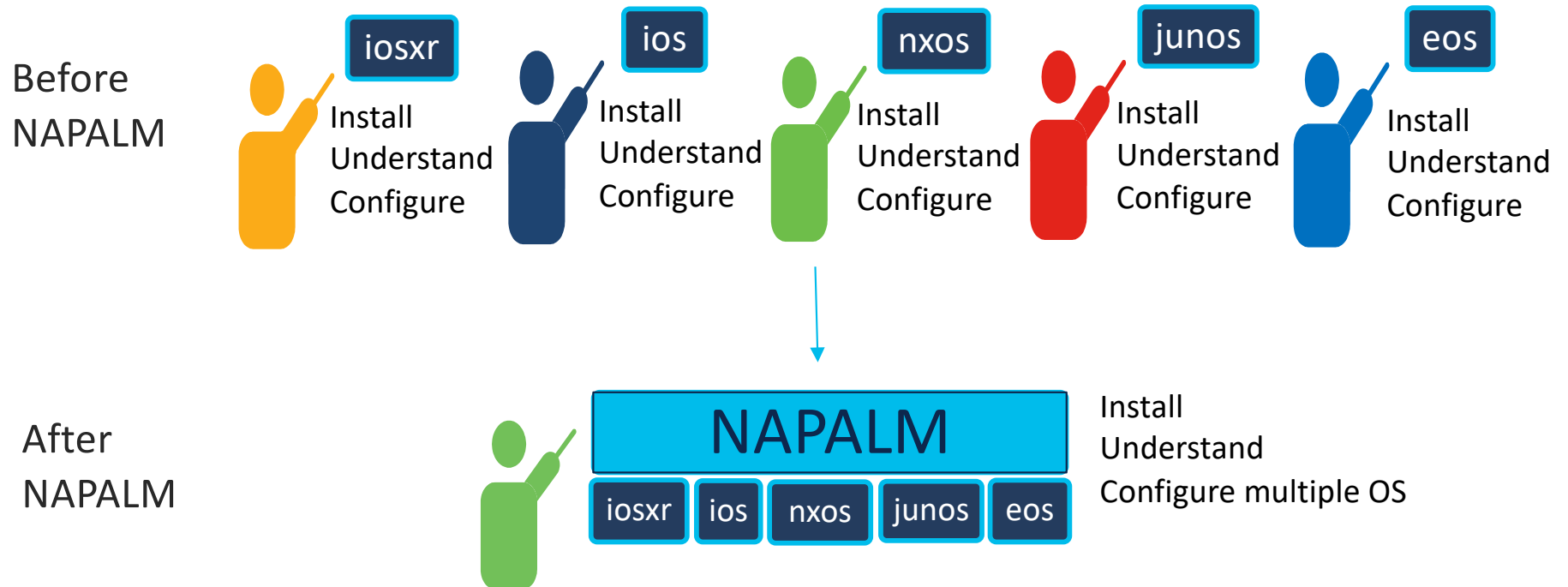
What, Why and How ?

What is NAPALM ?

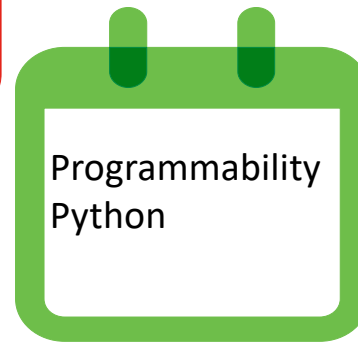
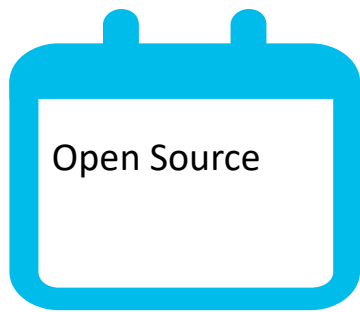
- Network Automation and **Programmability Abstraction Layer** with Multivendor support
- NAPALM is a **vendor neutral**, cross-platform open source project that provides a **unified API**
- NAPALM is a **python library** that provides a set of functions for configuration management and operational data 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 ?

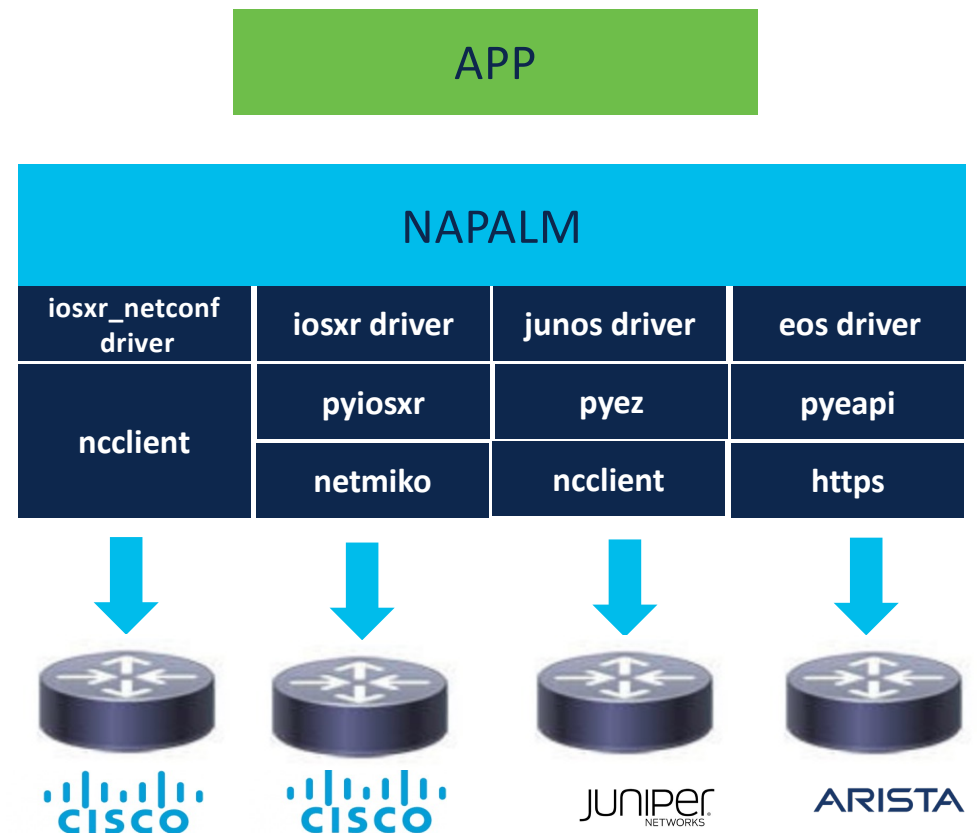


Why is it important ?



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** 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)

IOS-XR

```
{
  "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"
  ]
}
```

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"
  ]
}
```

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"
  ]
}
```

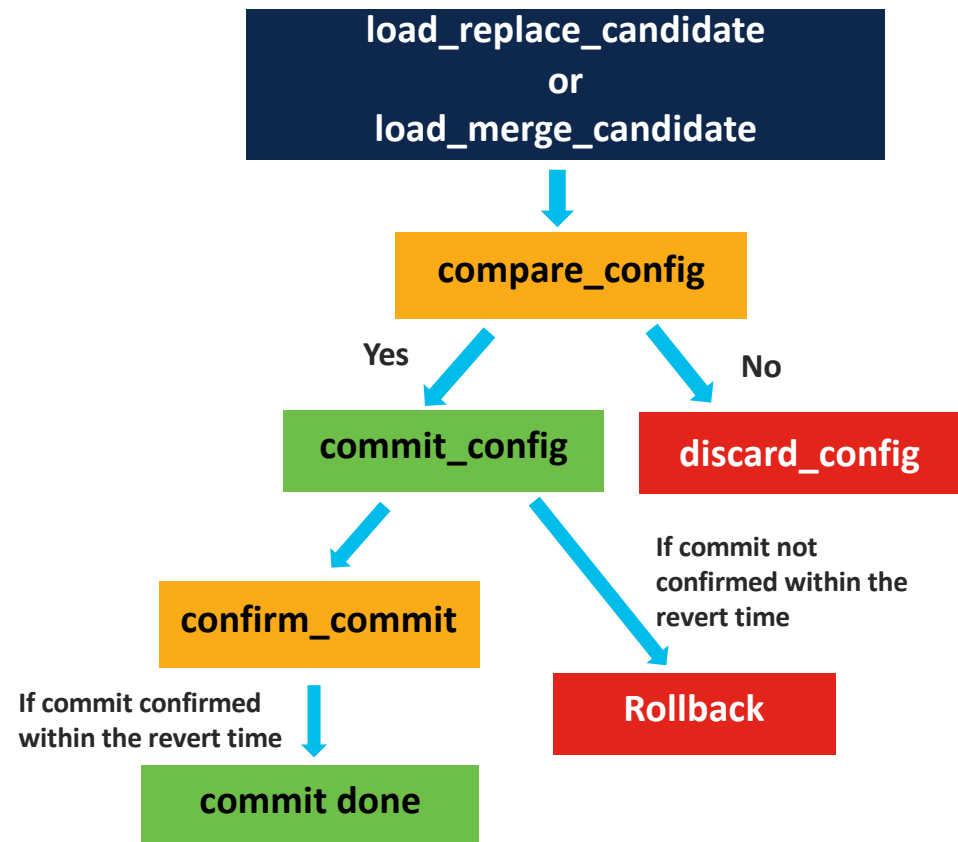
EOS

```
{
  "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 **network automation framework** that abstracts inventory and task execution
- It helps to automate your network tasks efficiently
- You can **configure** the devices, **validate** the operational data, and **enable** 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 ?

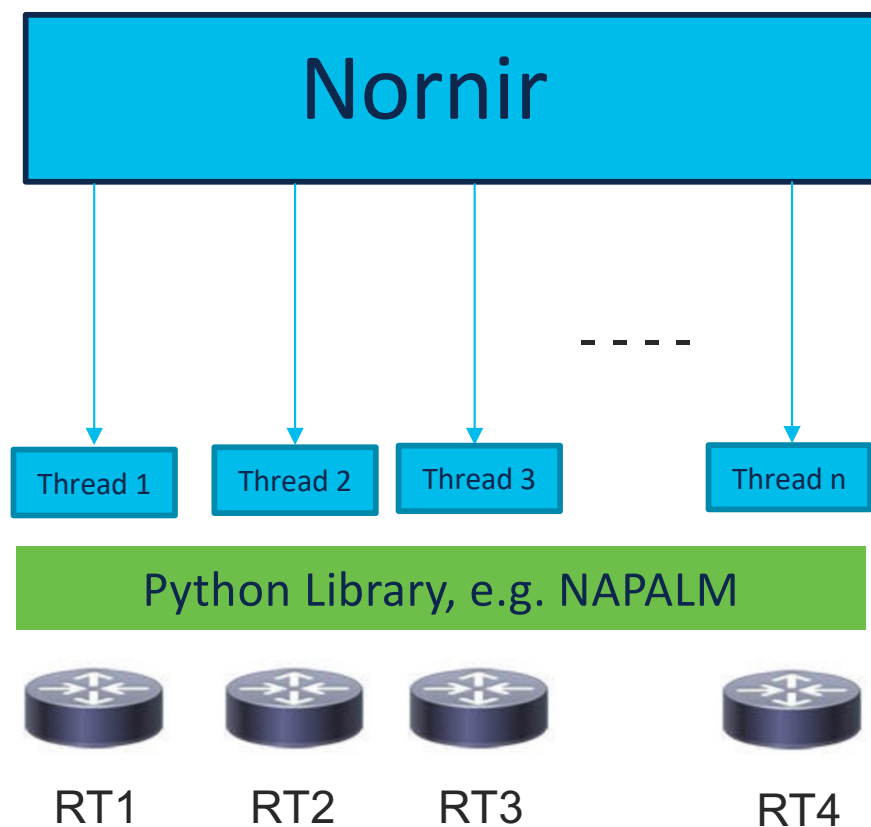


Why is it important ?

- 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 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 a collection of **hosts**
- 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 execute tasks against the 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

Pre-requisites (1/4)

- 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
rt3:
  hostname: localhost
  platform: 'nx-os'
  username: user
  password: pwd
```

groups.yaml

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

defaults.yaml

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

Pre-requisites (2/4)

- 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

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
```

Pre-requisites (3/4)

- If `num_workers == 1`, and runner plugin is `serial`, then tasks of the hosts are executed `sequentially`
- `Serial` 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`

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
```

Pre-requisites (4/4)

- Main file initialize Nornir with `InitNornir` function and provide the config file
- Call `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
)

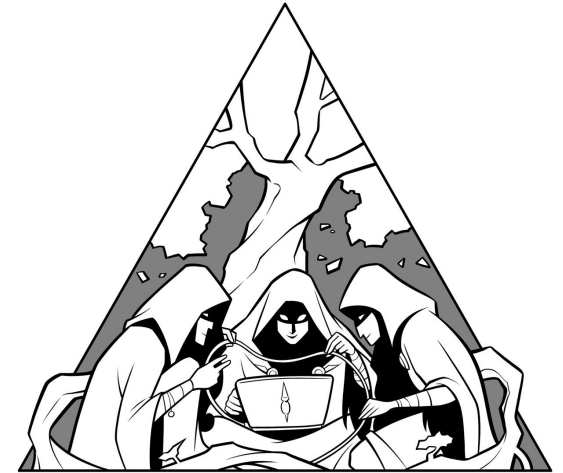
results = nr.run(
    task=napalm_get, getters=["facts"]
)

print_result(results)
```


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`



Refer <https://nornir.tech/nornir/plugins/> to learn more about Nornir plugins

Demo

```
(base) neparaka@NEPARAKA-M-F176:~$ source ~/nornir_napalm_venv/bin/activate
(nornir_napalm_venv) (base) neparaka@NEPARAKA-M-F176:~$ python --version
Python 3.8.5
(nornir_napalm_venv) (base) neparaka@NEPARAKA-M-F176:~$ pip list | grep nornir
nornir                3.3.0
nornir-napalm        0.3.0
nornir-netmiko       0.2.0
nornir-utils         0.2.0
(nornir_napalm_venv) (base) neparaka@NEPARAKA-M-F176:~$ pip list | grep napalm
napalm                4.0.0
nornir-napalm        0.3.0
(nornir_napalm_venv) (base) neparaka@NEPARAKA-M-F176:~$ c
```

Summary

- NAPALM is a **vendor neutral**, cross-platform open-source project that provides a unified API to network devices
- NAPALM is **open-source** project, completely written in **python**, easy to install, understand and use
- Nornir is a **network automation 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?

