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?

Before NAPALM
- Install
- Understand
- Configure

After NAPALM
- Install
- Understand
- Configure multiple OS
Why is it important?

Open Source

90 Contributors
30 releases
4200 commits

FREE

Active Community

Well Maintained Documentation

Easy to Install and Use

One Install Configure Multiple OS

Programmability Python

Multivendor Support
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)

```json
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

<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>load_replace_candidate</td>
</tr>
<tr>
<td>load_merge_candidate</td>
</tr>
<tr>
<td>compare_config</td>
</tr>
<tr>
<td>commit_config</td>
</tr>
<tr>
<td>discard_config</td>
</tr>
<tr>
<td>confirm_commit</td>
</tr>
<tr>
<td>rollback</td>
</tr>
</tbody>
</table>

- load_replace_candidate or load_merge_candidate
- compare_config
  - Yes
    - commit_config
  - No
    - discard_config
      - If commit not confirmed within the revert time
        - confirm_commit
          - If commit confirmed within the revert time
            - Rollback
          - commit done
NAPALM API Overview (2/2)

### Operational Data Management

<table>
<thead>
<tr>
<th>Functions</th>
<th>Functions</th>
<th>Functions</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_facts</td>
<td>get_route_to</td>
<td>get_arp_table</td>
<td>get_environment</td>
</tr>
<tr>
<td>get_interfaces</td>
<td>get_snmp_information</td>
<td>get_ntp_peers</td>
<td>cli</td>
</tr>
<tr>
<td>get_interfaces_counters</td>
<td>get_probes_config</td>
<td>get_ntp_servers</td>
<td>get_firewall_policies</td>
</tr>
<tr>
<td>get_interfaces_ip</td>
<td>get_probes_results</td>
<td>get_ntp_stats</td>
<td>get_ipv6_neighbors_table</td>
</tr>
<tr>
<td>get_bgp_config</td>
<td>traceroute</td>
<td>get_lldp_neighbors</td>
<td>get_network_instances</td>
</tr>
<tr>
<td>get_bgp_neighbors</td>
<td>get_users</td>
<td>get_lldp_neighbors_detail</td>
<td>get_optics</td>
</tr>
<tr>
<td>get_bgp_neighbors_detail</td>
<td>get_config</td>
<td>get_mac_address_table</td>
<td>ping</td>
</tr>
</tbody>
</table>
How to use NAPALM Python Library?

1. **pip install napalm**

2. Write a script to retrieve or load data

Manage configuration and operational data

```python
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())
```
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?

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

Python Library, e.g. NAPALM
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
  - **pip install nornir-utils**

- 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)

```python
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
  
  ```bash
  pip install nornir-napalm
  ```

- hosts.yaml
- groups.yaml
- defaults.yaml
- config.yaml
- Python main file - nornir_main.py

```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
---
iosxr:
    platform: 'iosxr'
ios:
    platform: 'ios'
xo-os:
    platform: 'nx-os'

# 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.

```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

```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

```
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

```python
napalm_get**************************************************
********************
* rt1 ** changed : False
******************************************************
vnvn napalm_get ** changed : False
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvnvvvvvvvvvvvvvvvvvvvv
INFO
{ 'facts': { 'fqdn': 'pavarotti',
            'hostname': 'pavarotti',
            'interface_list': ['GigabitEthernet0/0/0/0',
                              'GigabitEthernet0/0/0/1',
                              'Loopback0',
                              'MgmtEth0/RP0/CPU0/0',
                              'Null0'],
            'model': 'R- IOSXR V9000- CC',
            'os_version': '6.5.3',
            'serial_number': 'E3FDA081DAC',
            'uptime': 18033322,
            'vendor': 'Cisco'}}
```
Execute the python file to retrieve results

- 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.

```python
* rt2 ** changed : False
*******************************************************
vvvv napalm_get ** changed : False
vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv 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`

Refer [https://nornir.tech/nornir/plugins/](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 an 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?