pandas

Network Analysis Stack /w Nornir, NAPALM + Pandas!

PACKET CODERS

CODE - LABS - COMMUNITY

Agenda

- Quick introduction
- Aim of todays talk
- NAPALM
- Nornir
- Pandas
- E2e Demo

Quick Introduction

Myself

- Rick Donato (@rickjdon)
- Founder of **Packet Coders**

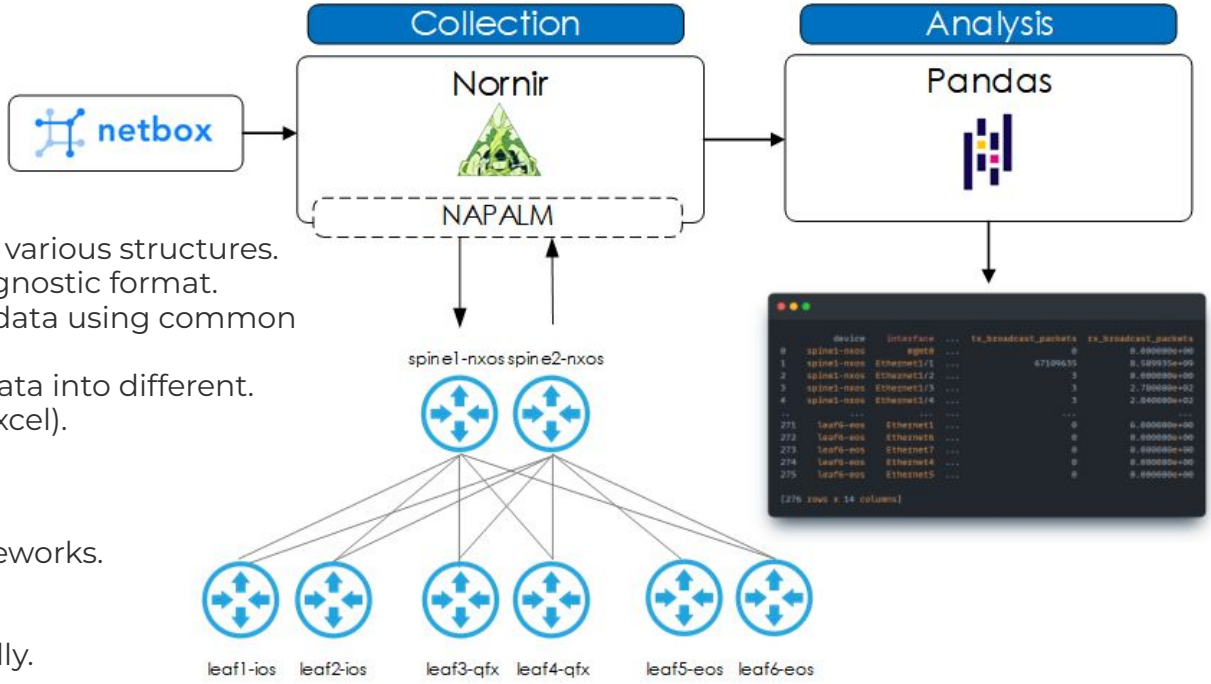
Packet Coders

- **Network Automation learning platform**
- We provide:
  - ✅ Courses
  - ✅ Instructor-led training
  - ✅ Membership
  - ✅ Newsletter
  - ✅ Blog
- https://packetcoders.io

Aim of Todays Talk

Walk through the building of an:

- network analysis stack, with
- open-source Python ingredients,



What does this mean?

- Take data from various vendors in various structures.
- Unify them into a single vendor agnostic format.
- Ask questions and work with the data using common methods.
- Ability to output the results and data into different. formats (JSON, Markdown, CSV, Excel).

The Why?

- Prime data ready for testing frameworks.
- Find issues in the network.
- Reporting.
- Great to know each tool individually.

NAPALM

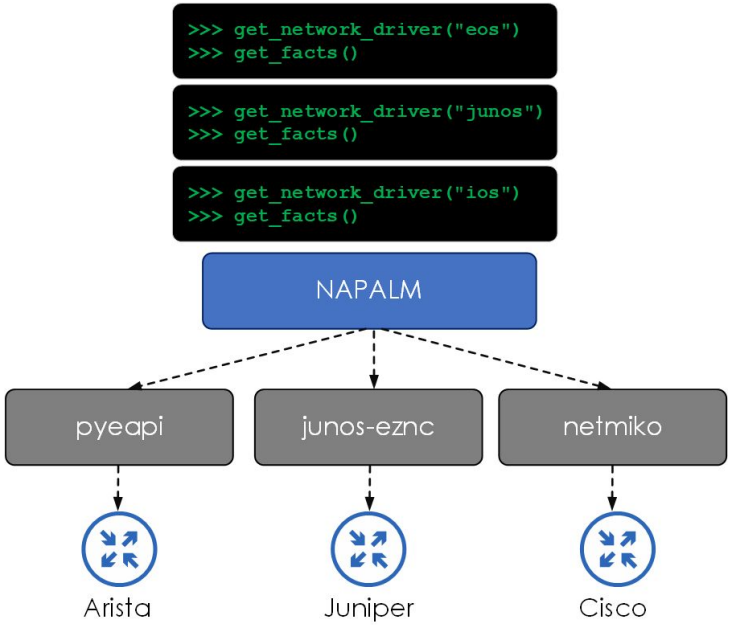
What is NAPALM?

NAPALM is a:

- NAPALM (Network Automation and Programmability Abstraction Layer with Multivendor support)
- Abstracts the lower-level semantics of device interaction
- Presents a common set of methods that are agnostic of the platform type.

Key Features

- Supports the 5 main vendors (i.e Arista, Cisco IOS, XR, NXOS, Juniper)
- Provides **getters** (get_routes, get_interfaces etc)
- Provides **configuration operation** methods (replace, merge etc)
- Compliance reporting (validating network against “states” in YAML)



Using NAPALM

NAPALM is installed via Pip

To use NAPALM against a device we:

1. **Create a NAPALM driver** instance (based on platform)
2. **Create a connection** to the device (via the driver instance)
3. **Do “things”** to the device over the **NAPALM connection**. (Config ops, getters etc.).

```
$ pip install napalm
---
from napalm import get_network_driver

napalm_driver = get_network_driver("eos")

device = {
    "hostname": "leaf5.lab.packetcoders.io",
    "username": os.getenv("DEVICE_USERNAME"),
    "password": os.getenv("DEVICE_PASSWORD"),
}

with napalm_driver(**device) as napalm_conn:
    # Get the interfaces counters
    napalm_conn.get_interfaces_counters()

    # Get the BGP neighbors
    napalm_conn.get_bgp_neighbors()

    # Get the device version and platform
    napalm_conn.get_facts()

    # Get the CPU and memory utilization
    napalm_conn.get_environment()
```

```
{
    'global': {
        'peers': {
            '11.11.11.11': {
                'is_up': True,
                'is_enabled': True,
                'uptime': 1468335,
                'description': '',
                'remote_as': 65000,
                'remote_id': '11.11.11.11',
                'local_as': 65013,
                'address_family': {
                    'ipv4': {'sent_prefixes': 0, ...
                    'ipv6': {'sent_prefixes': 0, ...
                }
            },
            '22.22.22.22': {
                'is_up': False,
                'is_enabled': True,
                'uptime': 1042620,
                'description': '',
  ...
```

Over to the Terminal

- Collect data from a single device.
- Getter for BGP neighbours info
- Getter for interface counters
- Getter for facts (NOS version)
- Getter for Environment (CPU, Memory)

Great! But...
What about concurrency, more then 1 device etc. ?

Nornir

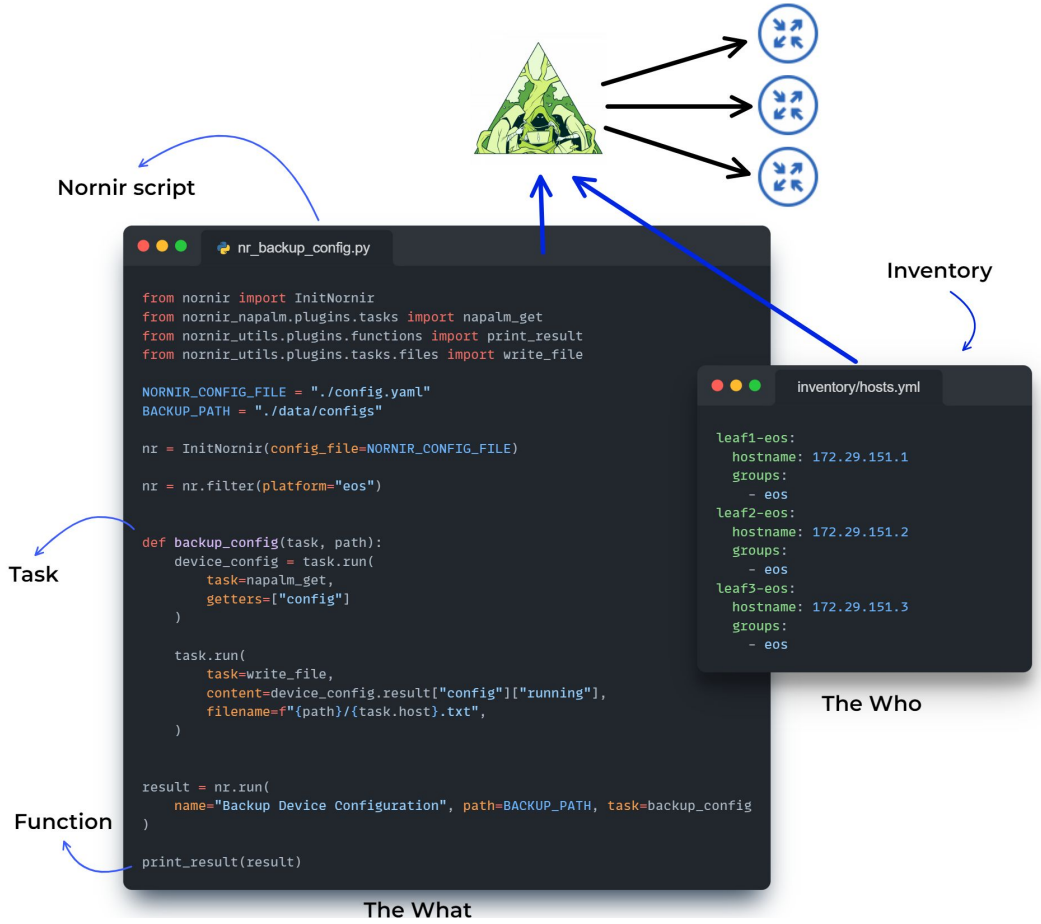
What is Nornir?

Nornir is an:

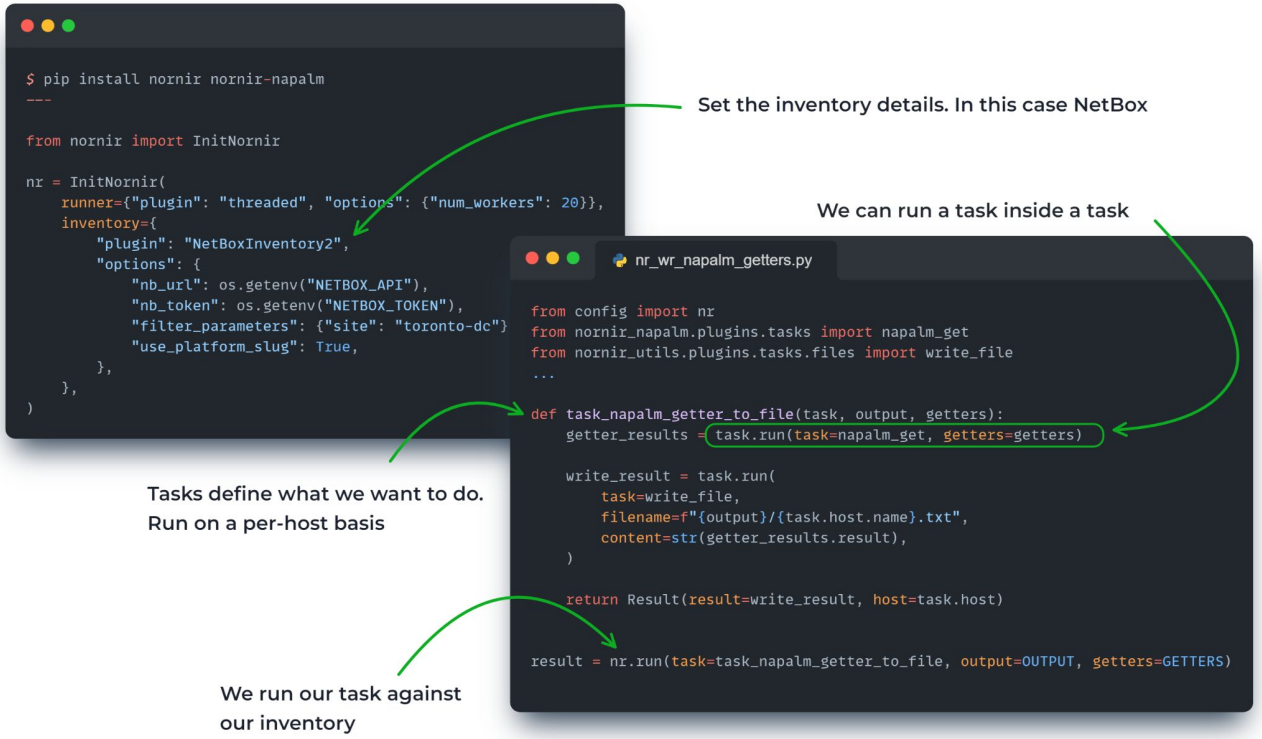
- Open-source tool/framework
- Community-led
- Networking focused for
- Managing groups of devices.

Key Features

- Pluggable (lean/lightweight)
- Multi-threaded
- Framework
- Inventory Management
- Fast
- 100% Python (great for debugging)



Using Nornir



Over to the Terminal

- Run our NAPALM getters against the network.
- Output the data (Python dicts) to a file.

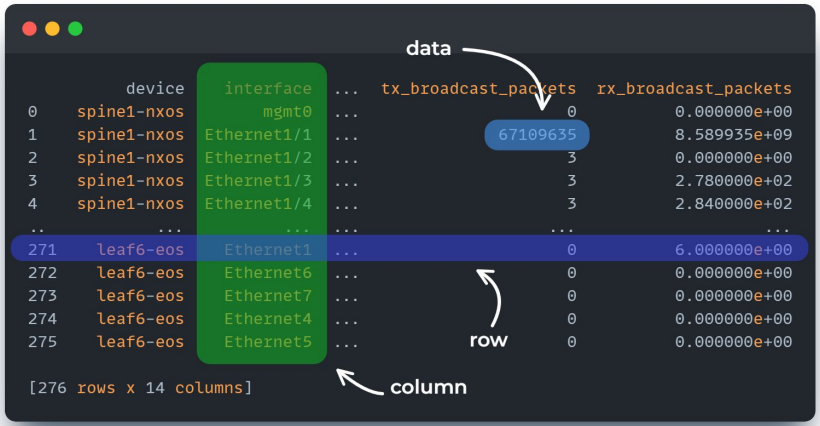
Now we have the data ...
What next?

Pandas

What is Pandas?

Pandas is a:

- Popular **Python library** used in the field of **data science**.
- Gaining adoption within the networking space.
- Provides various tools for data **exploration. manipulation** and **analysis**.
- **DataFrame** data structure:
  - Consists of rows and columns.
  - Similar to a spreadsheet (but easier to work with programmatically).
  - Extensive range of filtering, merging, and data processing methods.
  - Great support for converting data between data formats.



Working with Pandas

Importing

- Dict, CSV, Excel, JSON, Nested JSON, SQL

Analyze

- Filter, merge, grouping, aggregation

Exporting

- CSV, Excel, JSON, CSV, Markdown, HTML.

Input

```
$ pip install pandas
---
import pandas as pd

data = {
    "device": ["rtr001", "rtr002", "rtr003"],
    "network": ["10.1.1.0/24", "10.1.2.0/24", "10.1.3.0/24"],
    "version": ["12.1", "12.2", "12.3"],
    "vrf": ["mgmt", "mgmt", "default"],
}

# Import data
df = pd.DataFrame.from_dict(data)
# pd.DataFrame.from_dict(data, orient=...)
# pd.DataFrame.read_csv(csv)
# pd.DataFrame.read_json(json)
# pd.json_normalize(data, ...)

print(df)
...
   device      network version      vrf
0  rtr001  10.1.1.0/24    12.1     mgmt
1  rtr002  10.1.2.0/24    12.2     mgmt
2  rtr003  10.1.3.0/24    12.3  default
```

Analysis

```
# Index location referencing
df.iloc[0]
...
device          rtr001
network    10.1.1.0/24
version           12.1
vrf               mgmt
Name: 0, dtype: object

# Group by column name and show group totals
df.groupby(["version"]).size()
...
version
12.1    1
12.2    1
12.3    1

# Query DataFrame
df.query("vrf == 'default' & version > '12.1'")
...
   device      network version      vrf
2  rtr003  10.1.3.0/24    12.3  default
```

Output

```
# Output the df to various formats
df.to_excel("examples/output/output.xlsx", index=False)
df.to_markdown("examples/output/output.md", index=False)
```

Over to the Terminal

- Show basic importing into Pandas
- Show common methods used against data

E2E Demo

THANK YOU !