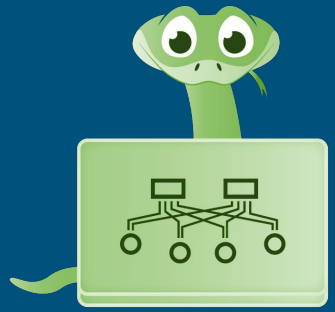


PYTHON

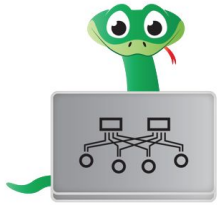
FOR NETWORK ENGINEERS

Onsite Training Session
July 2020

Day5 Schedule - Nornir

- Nornir Overview
- Inventory
- Nornir Tasks
- Results
- Nornir and Networking Plugins
- Custom Tasks and Grouped Tasks
- Nornir + Jinja2
- Using Netmiko and NAPALM Directly in Tasks
- Failed Tasks
- Debugging

PYTHON
FOR NETWORK ENGINEERS



PYTHON
FOR NETWORK ENGINEERS

Why was Nornir created? What problem is it trying to solve?

1. Systematic approach to inventory and data.
2. Concurrency
3. Using all Python

Why use a framework?

* Systematic Inventory Management
* Modular integration to other libraries
* Integrated Concurrency
* Systematizes automation in your organization

Ansible - Nornir Comparisons

Ansible Pluses

+ Easy getting started path
+ Use of SSH as a primary transport (familiarity)
+ Large community of network engineers using Ansible
+ Large organization behind it (Red Hat)

Nornir Pluses

+ All Python - Single, general purpose language
+ Easier debugging/troubleshooting
+ Use of Python Tool Chain (linters, debuggers, code testing)
+ Good performance
+ Tighter integrations to NAPALM and Netmiko

Ansible - Nornir Comparisons

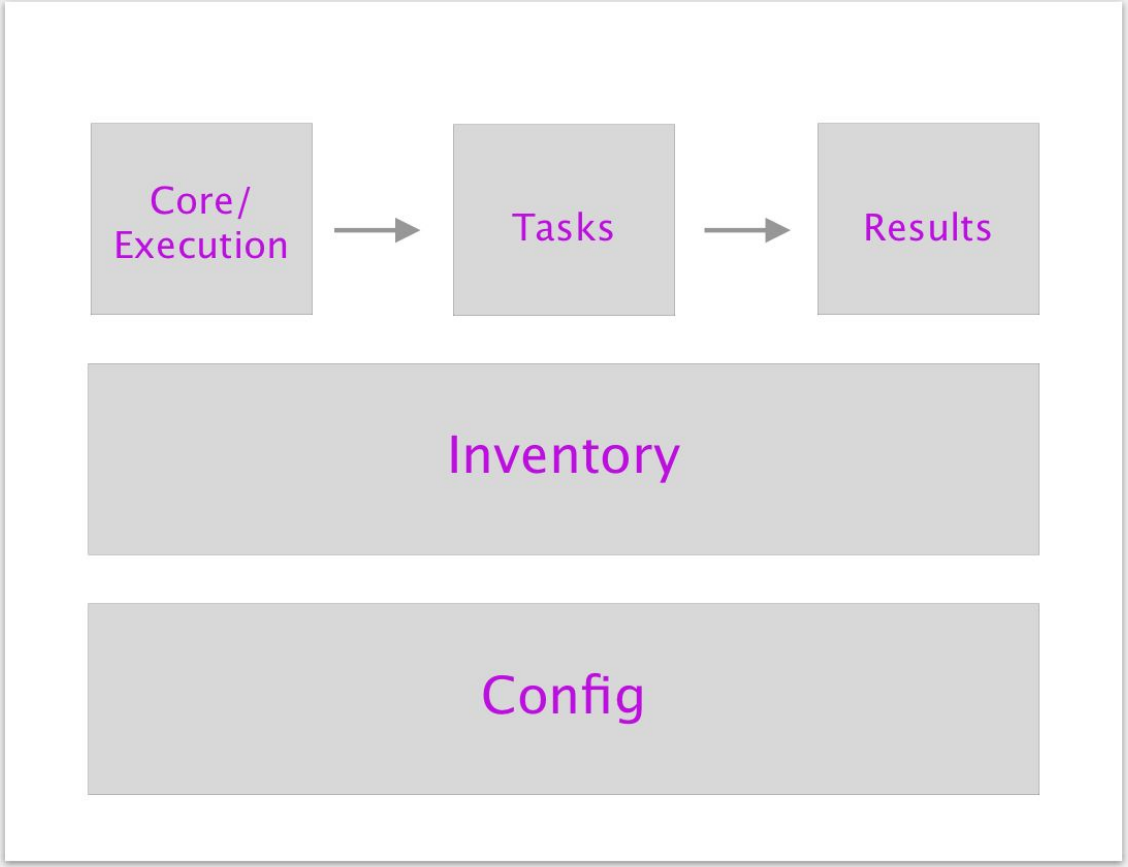
Ansible Minuses

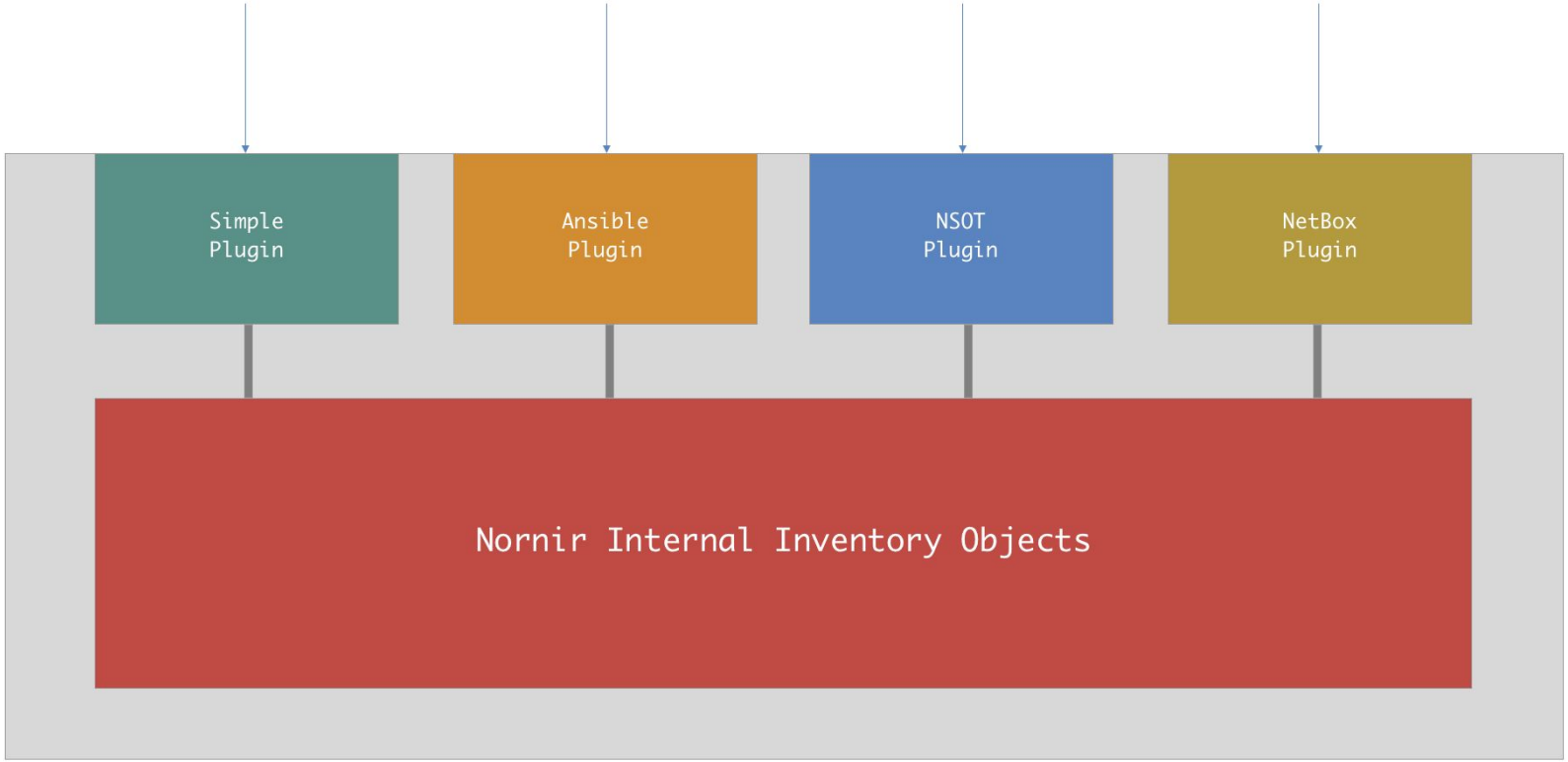
- Complex logic in Ansible is very painful
- Complex, nested data structures in Ansible are painful
- Troubleshooting can be unnecessarily difficult
- Easy things are easy, but somewhat difficult tasks get very hard quickly

Nornir Minuses

- You need to know Python
- Relatively new project/relatively small number of developers working on it
- Smaller community

Nornir Components




Simple
Plugin
Ansible
Plugin
NSOT
Plugin
NetBox
Plugin
Nornir Internal Inventory Objects

Nornir
SimpleInventory

```
logging:
  enabled: True
inventory:
  plugin: nornir.plugins.inventory.simple.SimpleInventory
  options:
    host_file: "~/nornir_inventory/hosts.yaml"
    group_file: "~/nornir_inventory/groups.yaml"
    defaults_file: "~/nornir_inventory/defaults.yaml"
```

Nornir SimpleInventory

```
sros1:
    hostname: sros.lasthop.io
    platform: nokia_sros
    connection_options:
      netmiko:
        extras:
          port: 2211
    groups:
      - sros
```

Inventory Attribute Traversal

Host -> Group1 -> Group2 -> GroupN -> Defaults

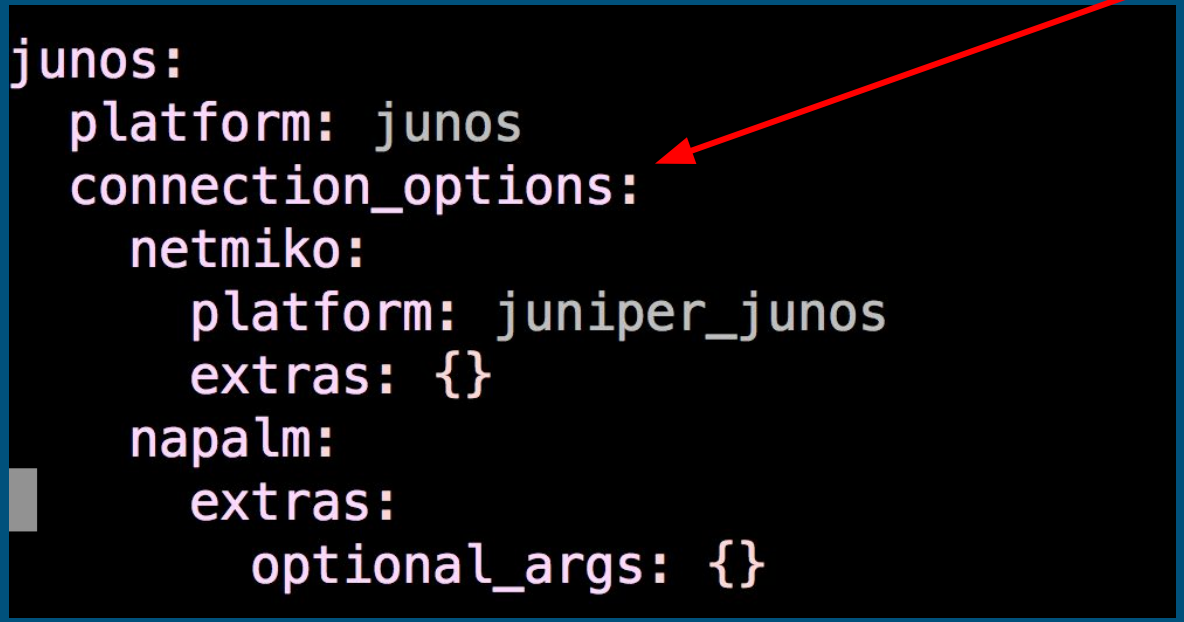
* the "data" exception

The *data inventory exception

```
ipdb> nr.inventory.hosts['sros2']['site']
'Freemont DC'
ipdb> nr.inventory.hosts['sros2'].data
{}
ipdb>
```

connection_options

What is going on here?



```
junos:
  platform: junos
  connection_options:
    netmiko:
      platform: juniper_junos
      extras: {}
    napalm:
      extras:
        optional_args: {}
```

Inventory filtering

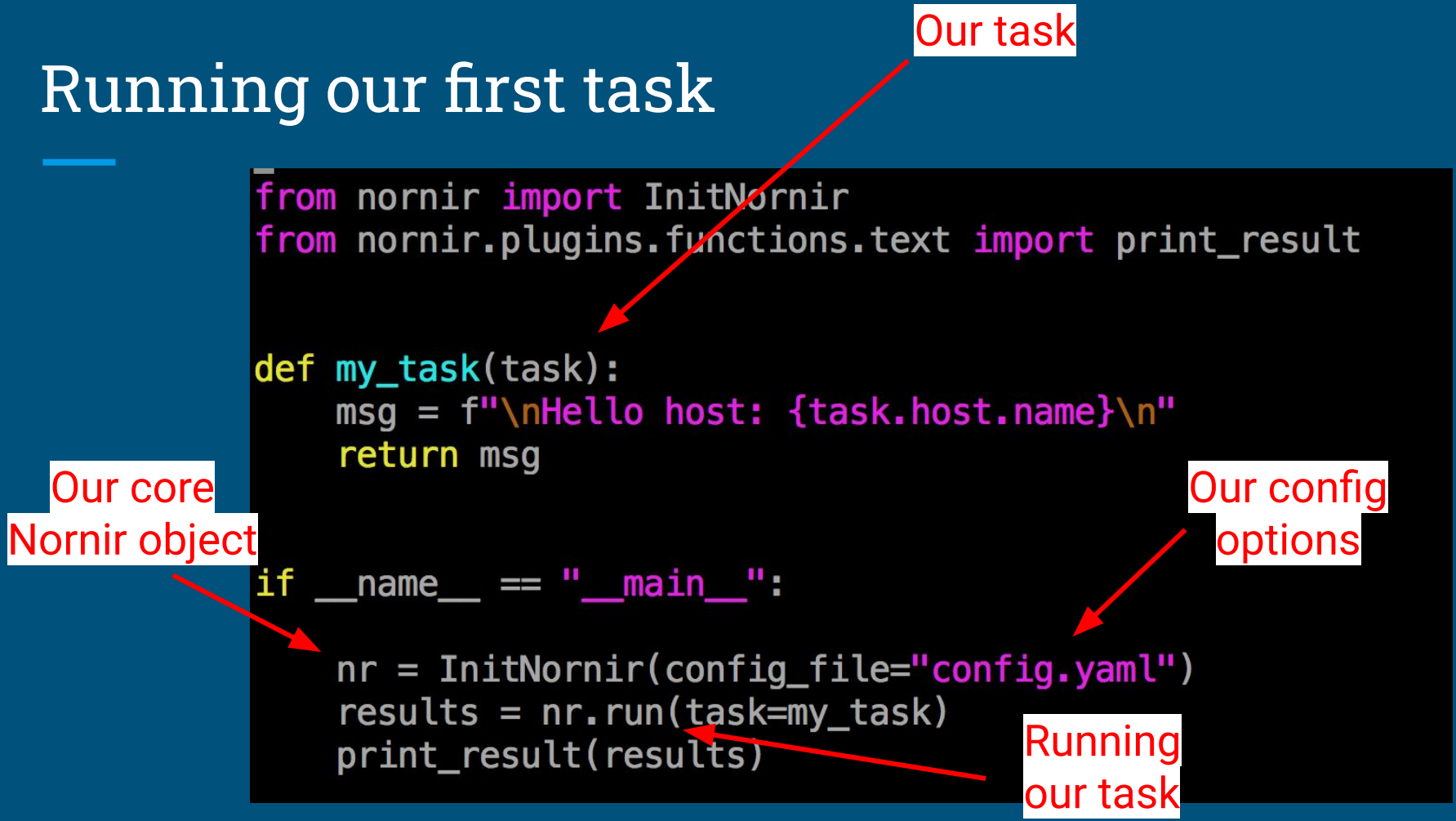
```
nr = InitNornir(config_file="config.yaml")
tmp_nr = nr.filter(name="sros1")
tmp_nr = nr.filter(platform="nokia_sros")
tmp_nr = nr.filter(hostname="vmx1.lasthop.io")

sros = nr.filter(F(groups__contains="sros"))

all_devices = nr.filter(F(groups__contains="sros") | F(groups__contains="junos"))
```

Exercises:
./day4/nornir_inventory/ex1.txt
./day4/nornir_inventory/ex2.txt
./day4/nornir_inventory/ex3.txt

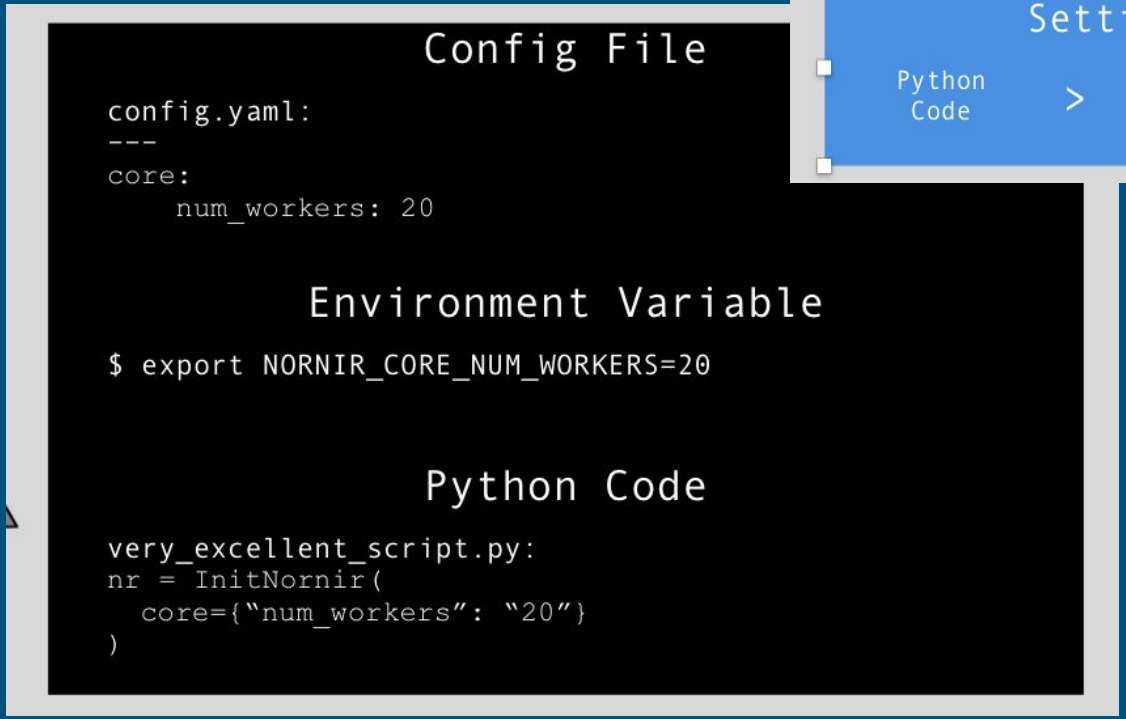
Running our first task



What can we specify in config.yaml?

```yaml
core:
  num_workers: 20
logging:
  enabled: True
inventory:
  plugin: nornir.plugins.inventory.simple.SimpleInventory
  options:
    host_file: "~/nornir_inventory/hosts.yaml"
    group_file: "~/nornir_inventory/groups.yaml"
    defaults_file: "~/nornir_inventory/defaults.yaml"
```

How can we specify configuration options?



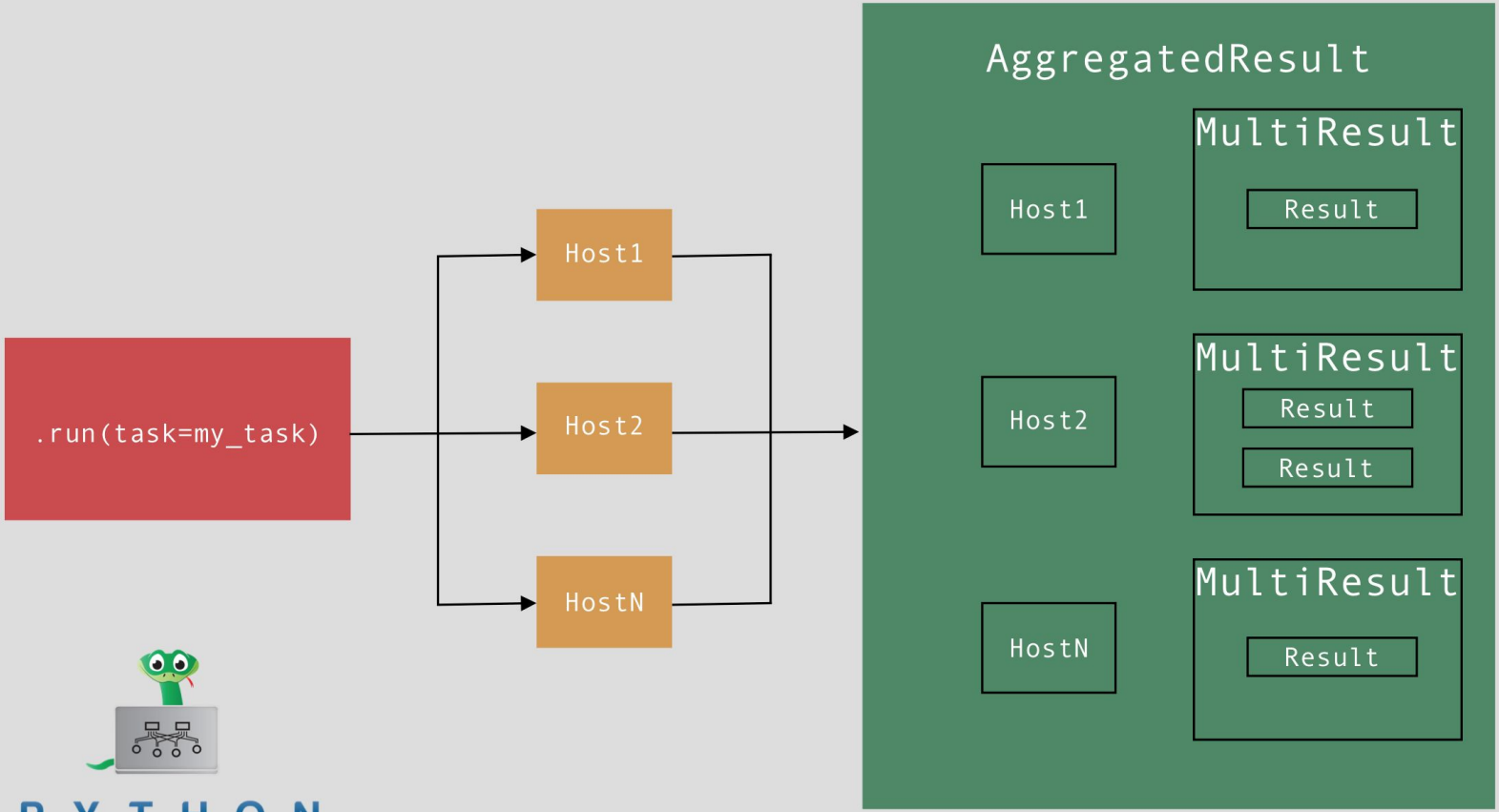
First Task Exercise

Exercises:

./day5/nornir_simple_task/ex1.txt

PYTHON

FOR NETWORK ENGINEERS



PYTHON
FOR NETWORK ENGINEERS

Nornir Results - Aggregated Result Object

```
ipdb> type(agg_result)
nornir.core.task.AggregatedResult
ipdb> agg_result.keys()
dict_keys(['sros1', 'sros2', 'sros3', 'sros4'])
ipdb> agg_result['sros1']
MultiResult: [Result: "netmiko_send_config"]
ipdb>
```

Nornir Results - Multi-Result Object

```
ipdb> multi_result = agg_result['sros1']
ipdb> type(multi_result)
nornir.core.task.MultiResult
ipdb> multi_result
MultiResult: [Result: "netmiko_send_config"]
ipdb> len(multi_result)
1
ipdb> multi_result[0]
Result: "netmiko_send_config"
```

Nornir Results - Result Object

```
ipdb> result_obj = multi_result[0]
ipdb> type(result_obj)
<class 'nornir.core.task.Result'>
ipdb> result_obj
Result: "netmiko_send_config"
ipdb> print(result_obj.result)
/configure system time ntp peer 130.126.24.24
*A:sros1# /configure system time ntp peer 152.2.21.1
*A:sros1# exit all
*A:sros1#
```

Nornir Results Exercise

Exercises:

./day5/nornir_results/ex1.txt

PYTHON
FOR NETWORK ENGINEERS

Nornir and Networking Plugins

```
from nornir import InitNornir
from nornir.plugins.tasks.networking import netmiko_send_command
from nornir.plugins.functions.text import print_result


if __name__ == "__main__":

    nr = InitNornir(config_file="config.yaml")
    results = nr.run(task=netmiko_send_command, command_string="show version")
    print_result(results)
```

Netmiko task plugin - will automatically create Netmiko SSH connection

NETMIKO

Netmiko Task Plugins

```
$ tree -C plugins/tasks/networking/
plugins/tasks/networking/
├── __init__.py
├── napalm_cli.py
├── napalm_configure.py
├── napalm_get.py
├── napalm_ping.py
├── napalm_validate.py
├── netconf_capabilities.py
├── netconf_edit_config.py
├── netconf_get_config.py
├── netconf_get.py
├── netmiko_commit.py
├── netmiko_file_transfer.py
├── netmiko_save_config.py
├── netmiko_send_command.py
├── netmiko_send_config.py
└── tcp_ping.py

0 directories, 16 files
```

NETMIKO

Netmiko Config Changes


```
config_commands = ['/configure system location "San Francisco"']

# Initialize Nornir
nr = InitNornir(config_file="config.yaml")
nr = nr.filter(F(groups__contains="sros"))

# Send configuration
result = nr.run(task=netmiko_send_config, config_commands=config_commands)
print_result(result)

# Save running config to startup
nr.run(task=netmiko_save_config)
```

Nornir + Netmiko Task Exercises

Exercises:

./day5/nornir_netmiko_task/ex1.txt

./day5/nornir_netmiko_task/ex2.txt

NETMIKO

Nornir and NAPALM

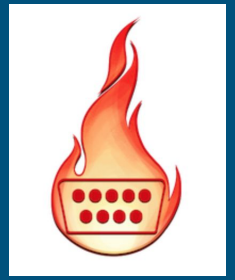
```
from nornir import InitNornir
from nornir.plugins.tasks.networking import napalm_get


if __name__ == "__main__":

    nr = InitNornir(config_file="config.yaml")
    nr = nr.filter(platform="junos")
    agg_result = nr.run(task=napalm_get, getters=["lldp_neighbors"])
    #agg_result = nr.run(task=napalm_get, getters=["facts"])
```

NAPALM task plugin - will automatically create underlying connection

NAPALM Getters



Getters support matrix

Note

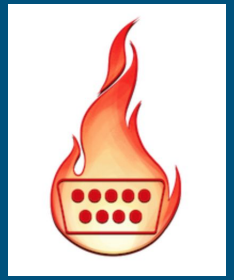
The following table is built automatically. Every time there is a release of a supported driver a built is triggered. The result of the tests are aggregated on the following table.

	EOS	IOS	IOSXR	JUNOS	NXOS	NXOS_SSH
get_arp_table	✅	✅	❌	❌	❌	✅
get_bgp_config	✅	✅	✅	✅	❌	❌
get_bgp_neighbors	✅	✅	✅	✅	✅	✅
get_bgp_neighbors_detail	✅	✅	✅	✅	❌	❌
get_config	✅	✅	✅	✅	✅	✅
get_environment	✅	✅	✅	✅	✅	✅
get_facts	✅	✅	✅	✅	✅	✅

Nornir and NAPALM

```
$ tree -C plugins/tasks/networking/
plugins/tasks/networking/
├── __init__.py
├── napalm_cli.py
├── napalm_configure.py
├── napalm_get.py
├── napalm_ping.py
├── napalm_validate.py
├── netconf_capabilities.py
├── netconf_edit_config.py
├── netconf_get_config.py
├── netconf_get.py
├── netmiko_commit.py
├── netmiko_file_transfer.py
├── netmiko_save_config.py
├── netmiko_send_command.py
├── netmiko_send_config.py
└── tcp_ping.py

0 directories, 16 files
```



NAPALM Configure

```
from nornir.plugins.tasks.networking import napalm_configure


def main():

    nr = InitNornir(config_file="config.yaml")
    nr = nr.filter(F(groups__contains="junos"))

    config = """
set system ntp server 130.126.24.24
set system ntp server 152.2.21.1
"""

    agg_result = nr.run(task=napalm_configure, configuration=config, dry_run=True)
    print_result(agg_result)
```

Nornir + NAPALM Exercises

Exercises:

./day5/nornir_napalm_task/ex1.txt

./day5/nornir_napalm_task/ex2.txt

./day5/nornir_napalm_task/ex3.txt

./day5/nornir_napalm_task/ex4.txt

Custom Tasks

```python
def my_task(task):
    print()
    print('-' * 40)
    print(f"Task: {task}")
    print(f"Host: {task.host.name}")
    print('-' * 40)
    print()


if __name__ == "__main__":

    nr = InitNornir(config_file="config.yaml")
    agg_result = nr.run(task=my_task)
```

Grouped Tasks

```
def my_task(task):
    config_commands = ['/configure system location "San Francisco"']

    # Send configuration
    task.run(task=netmiko_send_config, config_commands=config_commands)

    # Save running config to startup
    task.run(task=netmiko_save_config)

    return "whatever"
```

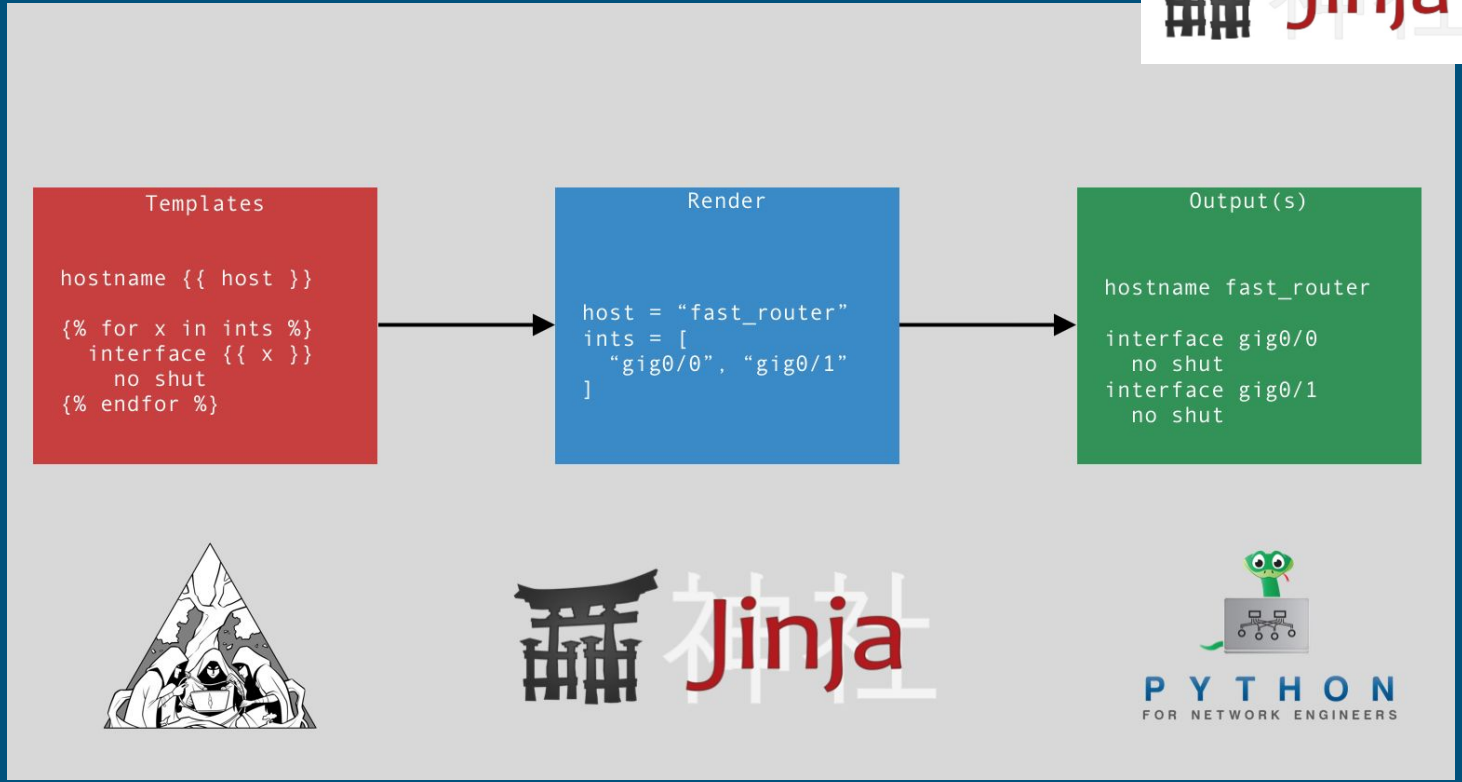
Grouped Tasks and effect on MultiResult

```
ipdb> sros1 = aggr_result['sros1']
ipdb> sros1
MultiResult: [Result: "my_task", Result: "netmiko_send_config", Result: "netmiko_save_config"]
ipdb> len(sros1)
3
ipdb> sros1[0].result
'whatever'
ipdb> sros1[1].result
'/configure system location "San Francisco"\nA:sros1# exit all \nA:sros1# '
ipdb> sros1[2].result
'Writing configuration to cf3:\\config.cfg\nSaving configuration ...   ... OK\nCompleted.'
```

The magic of MultiResult[0]

```
ipdb> sros1 = aggr_result['sros1']
ipdb> sros1
MultiResult: [Result: "my_task", Result: "netmiko_send_config", Result: "netmiko_save_config"]
ipdb> sros1.result
'whatever'
ipdb> sros1[0].result
'whatever'
```

Nornir + Jinja2



Nornir + Jinja2


```
    TEMPLATE_STR = """
interface loopback{{ int_num }}
  description {{ descr | lower }}
  no shut

"""

    nr = InitNornir(config_file="config.yaml", logging={"enabled": False})
    nr = nr.filter(name="srx2")

    my_vars = {
        "int_num": "99",
        "descr": "My Description",
    }

    agg_result = nr.run(task=template_string, template=TEMPLATE_STR, **my_vars)
```

Nornir + Jinja2


```
from nornir.plugins.tasks.text import template_file


def my_task(task):
    result = task.run(
        task=template_file, template="interfaces.j2", path=".", **task.host
    )
    print()
    print("-" * 40)
    print(result[0].result)
    print("-" * 40)
    print()
```

Nornir + Jinja2 Exercise

Exercises:

./day5/nornir_jinja2/ex1.txt

load_yaml / load_json

```python
from nornir.plugins.tasks.data import load_yaml # noqa
from nornir.plugins.tasks.data import load_json # noqa


def custom_task(task):
    import ipdb; ipdb.set_trace()
    # my_data = task.run(task=load_yaml, file=f"sros/{task.host.name}.yaml")
    my_data = task.run(task=load_json, file=f"sros/{task.host.name}.json")
    print(my_data.result)
```

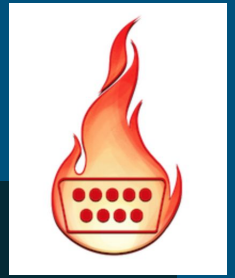
Direct Netmiko Connections


```
def netmiko_direct(task):

    # Manually create Netmiko connection
    net_connect = task.host.get_connection("netmiko", task.nornir.config)

    # Use the connection
    print()
    print("#" * 80)
    print(net_connect.find_prompt())
    output = net_connect.send_command("show system ntp")
    print(output)
    print("#" * 80)
    print()
```

Direct NAPALM Connections



```
def direct(task):

    # Manually create NAPALM connection
    napalm = task.host.get_connection("napalm", task.nornir.config)

    # PyEZ connection
    jnpr_conn = napalm.device

    # PyEZ RPC
    xml_output = jnpr_conn.rpc.get_software_information()
    print()
    print('-' * 40)
    print(etree.tostring(xml_output, encoding="unicode", pretty_print=True))
    print('-' * 40)
    print()
```

Direct Connection Exercises

Exercises:

./day5/nornir_direct/ex1.txt

./day5/nornir_direct/ex2.txt

NETMIKO

Failed Tasks

```
print(aggr_result.failed)
print(aggr_result.failed_hosts.keys())

vmx1 = aggr_result.failed_hosts["vmx1"]
print(vmx1.exception)

try:
    aggr_result.raise_on_error()
except NornirExecutionError:
    print("We can cause this exception to be raised")
```

Run task on failed hosts

```
# Run a task on the failed hosts
aggr_result = nr.run(
    task=failed_task,
    on_failed=True,
    on_good=False
)
```

Recover failed hosts

```
# Recover specific host
print(f"Failed Hosts: {nr.data.failed_hosts}")
nr.data.recover_host("vmx2")

# Reset failed hosts
print(f"Failed Hosts: {nr.data.failed_hosts}")
print("Reset failed hosts")
nr.data.reset_failed_hosts()
print(f"Failed Hosts: {nr.data.failed_hosts}")
```

Failed Task Exercises

Exercises:

./day5/nornir_failure/ex1.txt

./day5/nornir_failure/ex2.txt

NETMIKO

Debugging and Troubleshooting

1. Pdb is your friend.
2. Simplify the problem by setting num_workers=1
3. Look at your log file.
4. Netmiko debugging enable your session_log (or standard logging).
5. Try to isolate the problem to the part of the system that is causing the issue.

Nornir Version3 is Coming

Some Key Changes

- Moving to plugins hosted on external repositories (independent maintainers). Also affects your import references.
- “Runner” is now specified in config.yaml to indicate “threaded” or “serial” versus just controlling with num_workers.
- Change config.yaml references specifically for inventory plugins.
- num_workers eliminated as used inside of nr.run()
- Group behavior is slightly changed with respect to Group “refs” and group filtering.

Nornir Version3 is Coming

Some Changes

- Transform function requires an entry point.
- “log_file” replaces “file” reference in config.yaml.

Nornir Resources

Nornir Discourse Group:
https://nornir.discourse.group/

Nornir Tutorial:
https://nornir.readthedocs.io/en/latest/tutorials/intro/

Nornir Repository on GitHub:
https://github.com/nornir-automation/nornir

The end...

Questions?

ktbyers@twb-tech.com