nornir Documentation

Release 0.0.1

David Barroso

Oct 27, 2018

Contents

1 How the documentation is structured **3**

2 A first glance **5**

3 Contents **7**

Python Module Index **57**

Nornir is an automation framework written in python to be used with python. Most automation frameworks hide the language they are written in by using some cumbersome pseudo-language which usually is almost Turing complete but lacks tooling to debug and troubleshoot. Integrating with other systems is also usually quite hard as they usually have complex APIs if any at all. Some of the other common problems of those pseudo-languages is that are usually quite bad at dealing with data and re-usability is limited.

Nornir aims to solve those problems by providing a pure python framework. Just imagine Nornir as the Flask of automation. Nornir will take care of dealing with the inventory where you have your host information, it will take care of dispatching the tasks to your devices and will provide a common framework to write "plugins".

CHAPTER 1

How the documentation is structured

- The *Tutorial* is a great place to start for new users.
- *How-to guides* aim to solve a specific use case or answer key problems. These guides can be more advanced than the tutorial and can assume some knowledge about how Nornir and related technologies work.
- *Reference guides* contains the API reference for Nornir and describe the core functions.
- *Configuration* describe the configuration parameters of Nornir and their default settings.
- *Plugins* shows which tasks and functions are available out of the box with Nornir and describe how they work.

Is something missing from the documentation? Please open an issue and tell us what you are missing or open a pull request and suggest an improvement.

CHAPTER 2

A first glance

Here is an example on how to quickly build a runbook leveraging Nornir to retrieve information from the network:

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

nr = InitNornir(
    config_file="nornir.yaml", dry_run=True, num_workers=20
)

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

You can find this and other examples here.

CHAPTER 3

Contents

3.1 Learning Nornir

We're glad you made it here! This is a great place to learn the basics of Nornir. Good luck on your journey.

3.1.1 What is Nornir?

Nornir is an automation framework written in Python. These days there exists several automation frameworks. What makes Nornir different is that you write Python code in order to use Nornir. This is to be compared to other frameworks which typically use their own configuration language.

Why does this matter?

Typically, a specific configuration language is easy to use in a basic way. Though after a while you need to use more advanced features and might have to extend that configuration language with another programming language. While this works it can be very hard to troubleshoot once it's started to grow.

As Nornir allows you to use pure Python code you can troubleshoot and debug it in the same way as you would do with any other Python code.

What does it compare to?

In some ways, you could compare Nornir to Flask, which is a web framework that allows you to create web applications. Flask provides an easy to use interface which lets you build powerful websites without forcing you to work in a particular way.

Nornir lets you automate your environment by providing you an interface which does a lot of the heavy lifting.

How much Python do you need do know?

As you write Python code to control Nornir it's assumed that you are somewhat familiar with Python. But how good do you have to be with Python in order to make use of Nornir? That's actually the topic for the next section *spoiler alert* Not a lot!

3.1.2 The need to know Python

In order to use Nornir you have to know some Python. This might come as wonderful news to you, or you might find it a bit scary. If you're already a comfortable Python user, just go a head and hit *next*.

If you haven't written any code before you might be heading somewhere else now. Before you go however, answer me this:

Do you know Excel?

Chances are that you know how to use Excel. It's simple right. You just open a sheet and enter some data. It's used by a lot of finance people and unfortunately it's one of the most used IPAM solutions. Though aside from a simple tool to enter data in a sheet Excel has an insane amount of features. Most people will only use 5% of all the features. How good are you at Excel? Does it matter?

It's the same way with Python, it can take very long time to fully master it. The good part is that you don't have to become a master. As long as you know the very basics you will be able to use Nornir.

Python skills required

If you've never seen Python before, and don't have any experience in other programming languages it will probably be a good idea to pick up the basics and come back here later.

In order to follow this tutorial you should be able to:

- Setup Python on your system (Linux or Mac)
- Install Virtualenv and Python packages
- Understand basic Python concepts such as:
  - Variables
  - Functions
  - Imports

3.1.3 Installing Nornir

Before you go ahead and install Nornir it's recommended to create your own Python virtualenv. That way you have complete control of your environment and you don't risk overwriting your systems Python environment.

Note: This tutorial doesn't cover the creation of a Python virtual environment. The Python documentation offers a guide where you can learn more about virtualenvs. We also won't cover the installation of pip, but chances are that you already have pip on your system.

Nornir is published to PyPI and can be installed like most other Python packages using the pip tool. You can verify that you have pip installed by typing:

```
pip --version

pip 9.0.3 from /Users/patrick/nornir/lib/python3.6/site-packages (python 3.6)
```

It could be that you need to use the pip3 binary instead of pip as pip3 is for Python 3 on some systems. Speaking of Python 3, this tutorial is written with Python 3.6 in mind. This has mostly to do with the use of f-strings, you should however be able to follow along even if you are still at Python 2.7. However, if you are starting something new don't use Python 2.7. You have to make sure that your pip is up to date as you might have trouble installing some of the Nornir dependencies if you have a very old pip. If you are on version 9.0 or later you should be fine.

As you would assume, the installation is then very easy.

```
pip install nornir

Collecting nornir
Collecting colorama (from nornir)
[...]
Successfully installed MarkupSafe-1.0 asn1crypto-0.24.0 bcrypt-3.1.4 nornir-0.0.6
```

Please note that the above output has been abbreviated for readability. Your output will be quite a bit longer. You should see that *nornir* is successfully installed.

Now we can verify that Nornir is installed and that you are able to import the package from Python.

```
python
>>>import nornir.core
>>>
```

Great, now you're ready to create an inventory.

3.1.4 Initializing Nornir

Easiest way of initializing nornir is with the function *InitNornir*.

With `InitNornir` you can initialize nornir with a configuration file, with code or with a combination of both.

Let's start with *a configuration file*:

```
In [2]: %highlight_file config.yaml
```

```
Out[2]: <IPython.core.display.HTML object>
```

Note: To pass options to the inventory plugin add a top-level dictionary named after the inventory class name; `SimpleInventory` in this example.

Now to create the *nornir* object:

```
In [3]: from nornir.core import InitNornir
        nr = InitNornir(config_file="config.yaml")
```

You can also initialize nornir programmatically without a configuration file:

```
In [4]: from nornir.core import InitNornir
        nr = InitNornir(num_workers=100,
                        inventory="nornir.plugins.inventory.simple.SimpleInventory",
                        SimpleInventory={"host_file": "inventory/hosts.yaml",
                                         "group_file": "inventory/groups.yaml"})
```

Or with a combination of both methods:

```
In [5]: from nornir.core import InitNornir
        nr = InitNornir(num_workers=50, config_file="config.yaml")
```

```
In [6]: nr.config.num_workers
```

```
Out[6]: 50
```

3.1.5 Inventory

The Inventory is arguably the most important piece of nornir. Let's see how it works. To begin with the *inventory* is comprised of *hosts* and *groups*.

In this tutorial we are using the *SimpleInventory* plugin. This inventory plugin stores all the relevant data in two files. Let's start by checking them:

```
In [2]: # hosts file
        %highlight_file inventory/hosts.yaml
```

```
Out[2]: <IPython.core.display.HTML object>
```

The hosts file is basically a map where the outermost key is the hostname and then any arbitrary `<key, value>` pair you want inside. Usually `nornir_*` keys have special meaning, you can investigate the *hosts* class for details on those. In addition, the `groups` key is a list of groups you can inherite data from. We will inspect soon how the inheritance model works.

Now, let's look at the groups file:

```
In [3]: # groups file
        %highlight_file inventory/groups.yaml
```

```
Out[3]: <IPython.core.display.HTML object>
```

Pretty similar to the hosts file.

Accessing the inventory

You can access the *inventory* with the `inventory` attribute:

```
In [4]: from nornir.core import InitNornir
        nr = InitNornir(config_file="config.yaml")

        nr.inventory
```

```
Out[4]: <nornir.plugins.inventory.simple.SimpleInventory at 0x10fb17cf8>
```

The inventory has two dict-like attributes `hosts` and `groups` that you can use to access the hosts and groups respectively:

```
In [5]: nr.inventory.hosts
```

```
Out[5]: {'host1.bma': Host: host1.bma,
         'host1.cmh': Host: host1.cmh,
         'host2.bma': Host: host2.bma,
         'host2.cmh': Host: host2.cmh,
         'leaf00.bma': Host: leaf00.bma,
         'leaf00.cmh': Host: leaf00.cmh,
         'leaf01.bma': Host: leaf01.bma,
         'leaf01.cmh': Host: leaf01.cmh,
         'spine00.bma': Host: spine00.bma,
         'spine00.cmh': Host: spine00.cmh,
         'spine01.bma': Host: spine01.bma,
         'spine01.cmh': Host: spine01.cmh}
```

```
In [6]: nr.inventory.groups
```

```
Out[6]: {'bma': Group: bma,
         'cmh': Group: cmh,
         'eu': Group: eu,
         'global': Group: global}
```

```
In [7]: nr.inventory.hosts["leaf01.bma"]
```

```
Out[7]: Host: leaf01.bma
```

Hosts and groups are also dict-like objects:

```
In [8]: host = nr.inventory.hosts["leaf01.bma"]
        host.keys()
```

```
Out[8]: dict_keys(['name', 'groups', 'nornir_host', 'nornir_username', 'nornir_password', 'nornir_net
```

```
In [9]: host["site"]
```

```
Out[9]: 'bma'
```

Inheritance model

Let's see how the inheritance models works by example. Let's start by looking again at the groups file:

```
In [10]: # groups file
         %highlight_file inventory/groups.yaml
```

```
Out[10]: <IPython.core.display.HTML object>
```

The host `leaf01.bma` belongs to the group `bma` which in turn belongs to the groups `eu` and `global`. The host `spine00.cmh` belongs to the group `cmh` which doesn't belong to any other group.

Data resolution works by iterating recursively over all the parent groups and trying to see if that parent group (or any of it's parents) contains the data. For instance:

```
In [11]: leaf01_bma = nr.inventory.hosts["leaf01.bma"]
         leaf01_bma["domain"]  # comes from the group `global`
```

```
Out[11]: 'global.local'
```

```
In [12]: leaf01_bma["asn"]  # comes from group `eu`
```

```
Out[12]: 65100
```

The group `defaults` is special. This group contains data that will be returned if neither the host nor the parents have a specific value for it.

```
In [13]: leaf01_cmh = nr.inventory.hosts["leaf01.cmh"]
         leaf01_cmh["domain"]  # comes from defaults
```

```
Out[13]: 'acme.local'
```

If nornir can't resolve the data you should get a KeyError as usual:

```
In [14]: try:
             leaf01_cmh["non_existent"]
         except KeyError as e:
             print(f"Couldn't find key: {e}")
```

```
Couldn't find key: 'non_existent'
```

You can also try to access data without recursive resolution by using the `data` attribute. For example, if we try to access `leaf01_cmh.data["domain"]` we should get an error as the host itself doesn't have that data:

```
In [15]: try:
             leaf01_cmh.data["domain"]
         except KeyError as e:
             print(f"Couldn't find key: {e}")

Couldn't find key: 'domain'
```

Filtering the inventory

So far we have seen that `nr.inventory.hosts` and `nr.inventory.groups` are dict-like objects that we can use to iterate over all the hosts and groups or to access any particular one directly. Now we are going to see how we can do some fancy filtering that will enable us to operate on groups of hosts based on their properties.

The simpler way of filtering hosts is by `<key, value>` pairs. For instance:

```
In [16]: nr.filter(site="cmh").inventory.hosts.keys()

Out[16]: dict_keys(['host1.cmh', 'host2.cmh', 'spine00.cmh', 'spine01.cmh', 'leaf00.cmh', 'leaf01.cmh
```

You can also filter using multiple `<key, value>` pairs:

```
In [17]: nr.filter(site="cmh", role="spine").inventory.hosts.keys()

Out[17]: dict_keys(['spine00.cmh', 'spine01.cmh'])
```

Filter is cumulative:

```
In [18]: nr.filter(site="cmh").filter(role="spine").inventory.hosts.keys()

Out[18]: dict_keys(['spine00.cmh', 'spine01.cmh'])
```

Or:

```
In [19]: cmh = nr.filter(site="cmh")
         cmh.filter(role="spine").inventory.hosts.keys()

Out[19]: dict_keys(['spine00.cmh', 'spine01.cmh'])

In [20]: cmh.filter(role="leaf").inventory.hosts.keys()

Out[20]: dict_keys(['leaf00.cmh', 'leaf01.cmh'])
```

You can also grab the children of a group:

```
In [21]: nr.inventory.groups["eu"].children()

Out[21]: {'host1.bma': Host: host1.bma,
          'host2.bma': Host: host2.bma,
          'leaf00.bma': Host: leaf00.bma,
          'leaf01.bma': Host: leaf01.bma,
          'spine00.bma': Host: spine00.bma,
          'spine01.bma': Host: spine01.bma}
```

Advanced filtering

Sometimes you need more fancy filtering. For those cases you can use a filter function:

```
In [22]: def has_long_name(host):
             return len(host.name) == 11

         nr.filter(filter_func=has_long_name).inventory.hosts.keys()

Out[22]: dict_keys(['spine00.cmh', 'spine01.cmh', 'spine00.bma', 'spine01.bma'])
```

```
In [23]: # Or a lambda function
          nr.filter(filter_func=lambda h: len(h.name) == 9).inventory.hosts.keys()

Out[23]: dict_keys(['host1.cmh', 'host2.cmh', 'host1.bma', 'host2.bma'])

In [1]: from nornir.core import InitNornir
        nr = InitNornir(config_file="config.yaml")
```

3.1.6 Executing tasks

Now that you know how to initialize nornir and work with the inventory let's see how we can leverage it to run tasks on groups of hosts.

Nornir ships a bunch of tasks you can use directly without having to code them yourself. You can check them out *here*.

Let's start by executing the `ls -la /tmp` command on all the device in `cmh` of type `host`:

```
In [2]: from nornir.plugins.tasks import commands
        from nornir.plugins.functions.text import print_result

        cmh_hosts = nr.filter(site="cmh", role="host")

        result = cmh_hosts.run(task=commands.remote_command,
                               command="ls -la /tmp")

        print_result(result, vars=["stdout"])

remote_command*******************************************************************
* host1.cmh ** changed : False ***************************************************
---- remote_command ** changed : False  ----------------------------------------
total 8
drwxrwxrwt  2 root root 4096 Mar 25 17:26 .
drwxr-xr-x 24 root root 4096 Mar 25 17:26 ..


* host2.cmh ** changed : False ***************************************************
---- remote_command ** changed : False  ----------------------------------------
total 8
drwxrwxrwt  2 root root 4096 Mar 25 17:27 .
drwxr-xr-x 24 root root 4096 Mar 25 17:27 ..
```

So what have we done here? First we have imported the `commands` and `text` modules. Then we have narrowed down nornir to the hosts we want to operate on. Once we have selected the devices we wanted to operate on we have run two tasks:

1. The task `commands.remote_command` which runs the specified `command` in the remote device.
2. The function `print_result` which just prints on screen the result of an executed task or group of tasks.

Let's try with another example:

```
In [3]: from nornir.plugins.tasks import networking

        cmh_spines = nr.filter(site="bma", role="spine")
        result = cmh_spines.run(task=networking.napalm_get,
                                getters=["facts"])
        print_result(result)
```

```
napalm_get**********************************************************************
* spine00.bma ** changed : False ***********************************************
---- napalm_get ** changed : False  --------------------------------------------
{ 'facts': { 'fqdn': 'localhost',
             'hostname': 'localhost',
             'interface_list': ['Ethernet1', 'Ethernet2', 'Management1'],
             'model': 'vEOS',
             'os_version': '4.17.5M-4414219.4175M',
             'serial_number': '',
             'uptime': 441,
             'vendor': 'Arista'}}

* spine01.bma ** changed : False ***********************************************
---- napalm_get ** changed : False  --------------------------------------------
{ 'facts': { 'fqdn': 'vsrx',
             'hostname': 'vsrx',
             'interface_list': [ 'ge-0/0/0',
                                 'gr-0/0/0',
                                 'ip-0/0/0',
                                 'lsq-0/0/0',
                                 'lt-0/0/0',
                                 'mt-0/0/0',
                                 'sp-0/0/0',
                                 'ge-0/0/1',
                                 'ge-0/0/2',
                                 '.local.',
                                 'dsc',
                                 'gre',
                                 'ipip',
                                 'irb',
                                 'lo0',
                                 'lsi',
                                 'mtun',
                                 'pimd',
                                 'pime',
                                 'pp0',
                                 'ppd0',
                                 'ppe0',
                                 'st0',
                                 'tap',
                                 'vlan'],
             'model': 'FIREFLY-PERIMETER',
             'os_version': '12.1X47-D20.7',
             'serial_number': 'b4321e51218e',
             'uptime': 334,
             'vendor': 'Juniper'}}
```

Pretty much the same pattern, just different task on different devices.

What is a task

Let's take a look at what a task is. In it's simplest form a task is a function that takes at least a *Task* object as argument. For instance:

```
In [4]: def hi(task):
            print(f"hi! My name is {task.host.name} and I live in {task.host['site']}")
```

```
        nr.run(task=hi, num_workers=1)
```

```
hi! My name is host1.cmh and I live in cmh
hi! My name is host2.cmh and I live in cmh
hi! My name is spine00.cmh and I live in cmh
hi! My name is spine01.cmh and I live in cmh
hi! My name is leaf00.cmh and I live in cmh
hi! My name is leaf01.cmh and I live in cmh
hi! My name is host1.bma and I live in bma
hi! My name is host2.bma and I live in bma
hi! My name is spine00.bma and I live in bma
hi! My name is spine01.bma and I live in bma
hi! My name is leaf00.bma and I live in bma
hi! My name is leaf01.bma and I live in bma
```

```
Out[4]: AggregatedResult (hi): {'host1.cmh': MultiResult: [Result: "hi"], 'host2.cmh': MultiResult:
```

The task object has access to `nornir`, `host` and `dry_run` attributes.

You can call other tasks from within a task:

```
In [5]: def available_resources(task):
            task.run(task=commands.remote_command,
                     name="Available disk",
                     command="df -h")
            task.run(task=commands.remote_command,
                     name="Available memory",
                     command="free -m")

        result = cmh_hosts.run(task=available_resources)

        print_result(result, vars=["stdout"])
```

```
available_resources*************************************************************
* host1.cmh ** changed : False **************************************************
---- available_resources ** changed : False -------------------------------------
---- Available disk ** changed : False ------------------------------------------
Filesystem                    Size  Used Avail Use% Mounted on
/dev/mapper/precise64-root     79G  2.2G   73G   3% /
udev                          174M  4.0K  174M   1% /dev
tmpfs                          74M  284K   73M   1% /run
none                          5.0M     0  5.0M   0% /run/lock
none                          183M     0  183M   0% /run/shm
/dev/sda1                     228M   25M  192M  12% /boot
vagrant                       373G  215G  159G  58% /vagrant

---- Available memory ** changed : False ----------------------------------------
             total       used       free     shared    buffers     cached
Mem:           365         88        277          0          8         36
-/+ buffers/cache:         43        322
Swap:          767          0        767


* host2.cmh ** changed : False **************************************************
---- available_resources ** changed : False -------------------------------------
---- Available disk ** changed : False ------------------------------------------
Filesystem                    Size  Used Avail Use% Mounted on
/dev/mapper/precise64-root     79G  2.2G   73G   3% /
udev                          174M  4.0K  174M   1% /dev
```

```
tmpfs                        74M  284K   73M   1% /run
none                        5.0M     0  5.0M   0% /run/lock
none                        183M     0  183M   0% /run/shm
/dev/sda1                   228M   25M  192M  12% /boot
vagrant                     373G  215G  159G  58% /vagrant

---- Available memory ** changed : False  --------------------------------------
             total       used       free     shared    buffers     cached
Mem:           365         93        271          0          9         36
-/+ buffers/cache:         48        317
Swap:          767          0        767
```

You probably noticed in your previous example that you can name your tasks.

Your task can also accept any extra arguments you may need:

```
In [6]: def count(task, to):
            print(f"{task.host.name}: {list(range(0, to))}")

        cmh_hosts.run(task=count,
                      num_workers=1,
                      to=10)
        cmh_hosts.run(task=count,
                      num_workers=1,
                      to=20)
```

```
host1.cmh: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
host2.cmh: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
host1.cmh: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
host2.cmh: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

```
Out[6]: AggregatedResult (count): {'host1.cmh': MultiResult: [Result: "count"], 'host2.cmh': MultiRes
```

Tasks vs Functions

You probably noticed we introduced the concept of a `function` when we talked about `print_result`. The difference between tasks and functions is that tasks are meant to be run per host while functions are helper functions meant to be run globally.

3.1.7 Grouping tasks

In this section we are going to see how we can group tasks. Grouping tasks might be useful for various reasons, for instance, for reusability purposes (as seen in a previous section) or even just for readability purposes.

We will also see very briefly how to use the functions *functions.text.print_title* and *functions.text.print_result* to make things look pretty.

As an objective in this tutorial we are going to be configuring the hostname and domain name of our network devices.

Let's start with the basic imports and objects we will need:

```
In [1]: from nornir.core import InitNornir
        from nornir.plugins.tasks import networking, text
        from nornir.plugins.functions.text import print_title, print_result
```

```
nr = InitNornir(config_file="config.yaml", dry_run=True)
cmh = nr.filter(site="cmh", type="network_device")
```

You may have noticed that `InitNornir` got a new argument `dry_run=True`. That argument (which defaults to `False`) controls whether to apply the changes to the device or just simulate them. You can control that argument via the configuration as well. Some tasks might even allow you to override this behavior completely at the task level.

Now, let's create task that is going to group what we want to do:

```
In [2]: def basic_configuration(task):
            # Transform inventory data to configuration via a template file
            r = task.run(task=text.template_file,
                         name="Base Configuration",
                         template="base.j2",
                         path=f"templates/{task.host.nos}")

            # Save the compiled configuration into a host variable
            task.host["config"] = r.result

            # Deploy that configuration to the device using NAPALM
            task.run(task=networking.napalm_configure,
                     name="Loading Configuration on the device",
                     replace=False,
                     configuration=task.host["config"])
```

As you can see the task is basically doing two things:

1. Render configuration from a jinja2 template and storing it into a host variable. Note we are using the host network operating system as part of the path so we can load the right template for the given device platform.
2. Deploying that configuration to the device with NAPALM.

Quite straightforward. For reference let's look at the templates:

```
In [3]: %cat templates/eos/base.j2

hostname {{ host }}
ip domain-name {{ site }}.{{ domain }}


In [4]: %cat templates/junos/base.j2

system {
  host-name {{ host }};
  domain-name {{ site }}.{{ domain }};
}
```

Now we can proceed with just a few lines of code:

```
In [5]: print_title("Playbook to configure the network")
        result = cmh.run(task=basic_configuration)
        print_result(result)

**** Playbook to configure the network *****************************************
basic_configuration*************************************************************
* spine00.cmh ** changed : True ************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Base Configuration ** changed : False -------------------------------------- INFO
hostname spine00.cmh
ip domain-name cmh.acme.local
---- Loading Configuration on the device ** changed : True --------------------- INFO
@@ -7,6 +7,9 @@
    action bash sudo /mnt/flash/initialize_ma1.sh
```

```
 !
 transceiver qsfp default-mode 4x10G
+!
+hostname spine00.cmh
+ip domain-name cmh.acme.local
 !
 spanning-tree mode mstp
 !
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* spine01.cmh ** changed : True ************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Base Configuration ** changed : False ------------------------------------- INFO
system {
  host-name spine01.cmh;
  domain-name cmh.acme.local;
}
---- Loading Configuration on the device ** changed : True --------------------- INFO
[edit system]
-  host-name vsrx;
+  host-name spine01.cmh;
+  domain-name cmh.acme.local;
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* leaf00.cmh ** changed : True *************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Base Configuration ** changed : False ------------------------------------- INFO
hostname leaf00.cmh
ip domain-name cmh.acme.local
---- Loading Configuration on the device ** changed : True --------------------- INFO
@@ -7,6 +7,9 @@
    action bash sudo /mnt/flash/initialize_ma1.sh
 !
 transceiver qsfp default-mode 4x10G
+!
+hostname leaf00.cmh
+ip domain-name cmh.acme.local
 !
 spanning-tree mode mstp
 !
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* leaf01.cmh ** changed : True *************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Base Configuration ** changed : False ------------------------------------- INFO
system {
  host-name leaf01.cmh;
  domain-name cmh.acme.local;
}
---- Loading Configuration on the device ** changed : True --------------------- INFO
[edit system]
-  host-name vsrx;
+  host-name leaf01.cmh;
+  domain-name cmh.acme.local;
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

As you can see, pretty straightforward. We just call the function we defined earlier grouping the tasks we wanted to execute and then we leverage on `functions.text.print_title` and `tasks.text.print_result` to make things look pretty.

However, this was a `dry_run`. Let's set the `dry_run` variable to `False` so changes are commited and then run the

code again:

Note: The `dry_run` value is shared between the main nornir objects and its childs so in the snippet below `nr.dry_run = False` and `cmh.dry_run = False` are equivalent.

```
In [6]: nr.dry_run = False
        print_title("Playbook to configure the network")
        result = cmh.run(task=basic_configuration)
        print_result(result)
```

```
**** Playbook to configure the network *****************************************
basic_configuration*************************************************************
* spine00.cmh ** changed : True ************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Base Configuration ** changed : False ------------------------------------- INFO
hostname spine00.cmh
ip domain-name cmh.acme.local
---- Loading Configuration on the device ** changed : True --------------------- INFO
@@ -7,6 +7,9 @@
    action bash sudo /mnt/flash/initialize_ma1.sh
 !
 transceiver qsfp default-mode 4x10G
+!
+hostname spine00.cmh
+ip domain-name cmh.acme.local
 !
 spanning-tree mode mstp
 !
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* spine01.cmh ** changed : True ************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Base Configuration ** changed : False ------------------------------------- INFO
system {
  host-name spine01.cmh;
  domain-name cmh.acme.local;
}
---- Loading Configuration on the device ** changed : True --------------------- INFO
[edit system]
-  host-name vsrx;
+  host-name spine01.cmh;
+  domain-name cmh.acme.local;
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* leaf00.cmh ** changed : True *************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Base Configuration ** changed : False ------------------------------------- INFO
hostname leaf00.cmh
ip domain-name cmh.acme.local
---- Loading Configuration on the device ** changed : True --------------------- INFO
@@ -7,6 +7,9 @@
    action bash sudo /mnt/flash/initialize_ma1.sh
 !
 transceiver qsfp default-mode 4x10G
+!
+hostname leaf00.cmh
+ip domain-name cmh.acme.local
 !
 spanning-tree mode mstp
 !
```

```
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* leaf01.cmh ** changed : True *************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Base Configuration ** changed : False ------------------------------------- INFO
system {
  host-name leaf01.cmh;
  domain-name cmh.acme.local;
}
---- Loading Configuration on the device ** changed : True --------------------- INFO
[edit system]
-  host-name vsrx;
+  host-name leaf01.cmh;
+  domain-name cmh.acme.local;
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

As the configuration should've been commited already if we run it again the task "Loading Configuration on the device" should tell us that `changed : False` and should return an empty diff. Let's see if that's true:

```
In [7]: print_title("Playbook to configure the network")
        result = cmh.run(task=basic_configuration)
        print_result(result)
```

```
**** Playbook to configure the network *****************************************
basic_configuration*************************************************************
* spine00.cmh ** changed : True ************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Base Configuration ** changed : False ------------------------------------- INFO
hostname spine00.cmh
ip domain-name cmh.acme.local
---- Loading Configuration on the device ** changed : True --------------------- INFO
@@ -7,6 +7,9 @@
    action bash sudo /mnt/flash/initialize_ma1.sh
 !
 transceiver qsfp default-mode 4x10G
+!
+hostname spine00.cmh
+ip domain-name cmh.acme.local
 !
 spanning-tree mode mstp
 !
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* spine01.cmh ** changed : True ************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Base Configuration ** changed : False ------------------------------------- INFO
system {
  host-name spine01.cmh;
  domain-name cmh.acme.local;
}
---- Loading Configuration on the device ** changed : True --------------------- INFO
[edit system]
-  host-name vsrx;
+  host-name spine01.cmh;
+  domain-name cmh.acme.local;
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* leaf00.cmh ** changed : True *************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Base Configuration ** changed : False ------------------------------------- INFO
hostname leaf00.cmh
ip domain-name cmh.acme.local
```

```
---- Loading Configuration on the device ** changed : True --------------------- INFO
@@ -7,6 +7,9 @@
    action bash sudo /mnt/flash/initialize_ma1.sh
 !
 transceiver qsfp default-mode 4x10G
+!
+hostname leaf00.cmh
+ip domain-name cmh.acme.local
 !
 spanning-tree mode mstp
 !
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* leaf01.cmh ** changed : True *************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Base Configuration ** changed : False ------------------------------------- INFO
system {
  host-name leaf01.cmh;
  domain-name cmh.acme.local;
}
---- Loading Configuration on the device ** changed : True --------------------- INFO
[edit system]
-  host-name vsrx;
+  host-name leaf01.cmh;
+  domain-name cmh.acme.local;
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

3.1.8 Processing results

In this tutorial we are going to see how we can process the results of running tasks.

Let's start with some code we have seen already:

```
In [1]: from nornir.core import InitNornir
        from nornir.plugins.tasks import networking, text
        from nornir.plugins.functions.text import print_title
        import logging

        nr = InitNornir(config_file="config.yaml", dry_run=True)
        cmh = nr.filter(site="cmh", type="network_device")

        def basic_configuration(task):
            # Transform inventory data to configuration via a template file
            r = task.run(task=text.template_file,
                         name="Base Configuration",
                         template="base.j2",
                         path=f"templates/{task.host.nos}",
                         severity_level=logging.DEBUG)

            # Save the compiled configuration into a host variable
            task.host["config"] = r.result

            # Deploy that configuration to the device using NAPALM
            task.run(task=networking.napalm_configure,
                     name="Loading Configuration on the device",
                     replace=False,
                     configuration=task.host["config"],
                     severity_level=logging.INFO)
```

Now, let's call the task group so we can start inspecting the result object:

```
In [2]: result = cmh.run(task=basic_configuration)
```

The easy way

Most of the time you will just want to provide some feedback on what's going on. For that you can use *print_result* function:

```
In [3]: from nornir.plugins.functions.text import print_result

        print_result(result)
```

```
basic_configuration************************************************************
* spine00.cmh ** changed : True ***********************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Loading Configuration on the device ** changed : True --------------------- INFO
@@ -7,6 +7,9 @@
    action bash sudo /mnt/flash/initialize_ma1.sh
 !
 transceiver qsfp default-mode 4x10G
+!
+hostname spine00.cmh
+ip domain-name cmh.acme.local
 !
 spanning-tree mode mstp
 !
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* spine01.cmh ** changed : True ***********************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Loading Configuration on the device ** changed : True --------------------- INFO
[edit system]
-  host-name vsrx;
+  host-name spine01.cmh;
+  domain-name cmh.acme.local;
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* leaf00.cmh ** changed : True ************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Loading Configuration on the device ** changed : True --------------------- INFO
@@ -7,6 +7,9 @@
    action bash sudo /mnt/flash/initialize_ma1.sh
 !
 transceiver qsfp default-mode 4x10G
+!
+hostname leaf00.cmh
+ip domain-name cmh.acme.local
 !
 spanning-tree mode mstp
 !
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* leaf01.cmh ** changed : True ************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Loading Configuration on the device ** changed : True --------------------- INFO
[edit system]
-  host-name vsrx;
+  host-name leaf01.cmh;
+  domain-name cmh.acme.local;
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

You should also be able to print a single host:

```
In [4]: print_result(result["spine00.cmh"])
```

```
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Loading Configuration on the device ** changed : True --------------------- INFO
@@ -7,6 +7,9 @@
    action bash sudo /mnt/flash/initialize_ma1.sh
 !
 transceiver qsfp default-mode 4x10G
+!
+hostname spine00.cmh
+ip domain-name cmh.acme.local
 !
 spanning-tree mode mstp
 !
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Or even a single task:

```
In [5]: print_result(result["spine00.cmh"][2])
```

```
---- Loading Configuration on the device ** changed : True --------------------- INFO
@@ -7,6 +7,9 @@
    action bash sudo /mnt/flash/initialize_ma1.sh
 !
 transceiver qsfp default-mode 4x10G
+!
+hostname spine00.cmh
+ip domain-name cmh.acme.local
 !
 spanning-tree mode mstp
 !
```

As you probably noticed, not all the tasks where printed. If you check the tests they all got a new argument `severity_level`. This let's us flag tasks with any of the logging levels. Then `print_result` is able to following logging rules to print the results. By default only tasks marked as `INFO` will be printed (this is also the default for the tasks if none is specified). Note that a failed task will have its severity level changed to `ERROR` regardless of the one specified by the user.

Now let's tell `print_result` to print tasks marked as `DEBUG`.

```
In [6]: print_result(result, severity_level=logging.DEBUG)
```

```
basic_configuration************************************************************
* spine00.cmh ** changed : True ***********************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Base Configuration ** changed : False ------------------------------------- DEBUG
hostname spine00.cmh
ip domain-name cmh.acme.local
---- Loading Configuration on the device ** changed : True --------------------- INFO
@@ -7,6 +7,9 @@
    action bash sudo /mnt/flash/initialize_ma1.sh
 !
 transceiver qsfp default-mode 4x10G
+!
+hostname spine00.cmh
+ip domain-name cmh.acme.local
 !
 spanning-tree mode mstp
```

```
 !
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* spine01.cmh ** changed : True ***********************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Base Configuration ** changed : False ------------------------------------- DEBUG
system {
  host-name spine01.cmh;
  domain-name cmh.acme.local;
}
---- Loading Configuration on the device ** changed : True --------------------- INFO
[edit system]
-  host-name vsrx;
+  host-name spine01.cmh;
+  domain-name cmh.acme.local;
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* leaf00.cmh ** changed : True ************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Base Configuration ** changed : False ------------------------------------- DEBUG
hostname leaf00.cmh
ip domain-name cmh.acme.local
---- Loading Configuration on the device ** changed : True --------------------- INFO
@@ -7,6 +7,9 @@
    action bash sudo /mnt/flash/initialize_ma1.sh
 !
 transceiver qsfp default-mode 4x10G
+!
+hostname leaf00.cmh
+ip domain-name cmh.acme.local
 !
 spanning-tree mode mstp
 !
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* leaf01.cmh ** changed : True ************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Base Configuration ** changed : False ------------------------------------- DEBUG
system {
  host-name leaf01.cmh;
  domain-name cmh.acme.local;
}
---- Loading Configuration on the device ** changed : True --------------------- INFO
[edit system]
-  host-name vsrx;
+  host-name leaf01.cmh;
+  domain-name cmh.acme.local;
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

The programmatic way

We have hinted already how to deal with result objects already but let's elaborate on that. To begin with, task groups will return an *AggregatedResult*. This object is a dict-like object you can use to iterate over or access directly hosts:

```
In [7]: result

Out[7]: AggregatedResult (basic_configuration): {'spine00.cmh': MultiResult: [Result: "basic_configu

In [8]: result.keys()

Out[8]: dict_keys(['spine00.cmh', 'spine01.cmh', 'leaf00.cmh', 'leaf01.cmh'])
```

```
In [9]: result["spine00.cmh"]
```

```
Out[9]: MultiResult: [Result: "basic_configuration", Result: "Base Configuration", Result: "Loading C
```

You probably noticed that inside each key in *AggregatedResult* there is a *MultiResult* object. This object is a list-like object you can use to iterate over or access any *Result* you want:

```
In [10]: result["spine00.cmh"][0]
```

```
Out[10]: Result: "basic_configuration"
```

Both `MultiResult` and `Result` should clearly indicate if there was some error or change in the system:

```
In [11]: print("changed: ", result["spine00.cmh"].changed)
         print("failed: ", result["spine00.cmh"].failed)
```

```
changed:  True
failed:  False
```

```
In [12]: print("changed: ", result["spine00.cmh"][0].changed)
         print("failed: ", result["spine00.cmh"][0].failed)
```

```
changed:  False
failed:  False
```

You should be able to access any extra data a particular task might have set:

```
In [13]: print(result["spine00.cmh"][2].diff)
```

```
@@ -7,6 +7,9 @@
    action bash sudo /mnt/flash/initialize_ma1.sh
 !
 transceiver qsfp default-mode 4x10G
+!
+hostname spine00.cmh
+ip domain-name cmh.acme.local
 !
 spanning-tree mode mstp
 !
```

This latter will depend on the task executed so you will have to refer to the documentation of the task to see what might have been populated by it.

3.1.9 Failed Tasks

Sometimes tasks can fail. Let's see how to deal with failed tasks in nornir.

Let's start as usual with the needed boilerplate:

```
In [1]: from nornir.core import InitNornir
        from nornir.plugins.tasks import networking, text
        from nornir.plugins.functions.text import print_result

        nr = InitNornir(config_file="config.yaml")
        cmh = nr.filter(site="cmh", type="network_device")
```

Now, as an example we are going to use a similar task group like the one we used in the previous tutorial:

```
In [2]: def basic_configuration(task):
            # Transform inventory data to configuration via a template file
            r = task.run(task=text.template_file,
```

```
                                name="Base Configuration",
                                template="base.j2",
                                path=f"templates/junos")

                # Save the compiled configuration into a host variable
                task.host["config"] = r.result

                # Deploy that configuration to the device using NAPALM
                task.run(task=networking.napalm_configure,
                         name="Loading Configuration on the device",
                         replace=False,
                         configuration=task.host["config"])
```

Note that the path is hardcoded to templates/junos, this should cause an error when trying to apply the configuration to the EOS devices. Let's see what happens:

```
In [3]: result = cmh.run(task=basic_configuration)
```

Let's inspect the object:

```
In [4]: result.failed
```

```
Out[4]: True
```

```
In [5]: result.failed_hosts
```

```
Out[5]: {'leaf00.cmh': MultiResult: [Result: "basic_configuration", Result: "Base Configuration", Res
         'spine00.cmh': MultiResult: [Result: "basic_configuration", Result: "Base Configuration", Re
```

```
In [6]: result['spine00.cmh'][1].exception
```

As you can see, the result object is aware something went wrong and you can inspect the errors if you so desire.

You can also using the `print_result` function on it:

```
In [7]: print_result(result)
```

```
basic_configuration*************************************************************
* leaf00.cmh ** changed : False ************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv ERROR
Subtask: Loading Configuration on the device (failed)

---- Base Configuration ** changed : False ------------------------------------- INFO
system {
  host-name leaf00.cmh;
  domain-name cmh.acme.local;
}
---- Loading Configuration on the device ** changed : False --------------------- ERROR
Traceback (most recent call last):
  File "/Users/dbarroso/.virtualenvs/nornir/lib/python3.6/site-packages/napalm/eos/eos.py", line 231,
    self.device.run_commands(commands)
  File "/Users/dbarroso/.virtualenvs/nornir/lib/python3.6/site-packages/pyeapi/client.py", line 730,
    response = self._connection.execute(commands, encoding, **kwargs)
  File "/Users/dbarroso/.virtualenvs/nornir/lib/python3.6/site-packages/pyeapi/eapilib.py", line 499,
    response = self.send(request)
  File "/Users/dbarroso/.virtualenvs/nornir/lib/python3.6/site-packages/pyeapi/eapilib.py", line 418,
    raise CommandError(code, msg, command_error=err, output=out)
pyeapi.eapilib.CommandError: Error [1002]: CLI command 3 of 6 'system {' failed: invalid command [In

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/Users/dbarroso/workspace/nornir/nornir/core/task.py", line 62, in start
    r = self.task(self, **self.params)
```

```
  File "/Users/dbarroso/workspace/nornir/nornir/plugins/tasks/networking/napalm_configure.py", line 2
    device.load_merge_candidate(filename=filename, config=configuration)
  File "/Users/dbarroso/.virtualenvs/nornir/lib/python3.6/site-packages/napalm/eos/eos.py", line 245,
    self._load_config(filename, config, False)
  File "/Users/dbarroso/.virtualenvs/nornir/lib/python3.6/site-packages/napalm/eos/eos.py", line 237,
    raise MergeConfigException(e.message)
napalm.base.exceptions.MergeConfigException: Error [1002]: CLI command 3 of 6 'system {' failed: inva

^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* leaf01.cmh ** changed : True **************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
---- Base Configuration ** changed : False -------------------------------------- INFO
system {
  host-name leaf01.cmh;
  domain-name cmh.acme.local;
}
---- Loading Configuration on the device ** changed : True --------------------- INFO
[edit system]
-  host-name vsrx;
+  host-name leaf01.cmh;
+  domain-name cmh.acme.local;
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* spine00.cmh ** changed : False ************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv ERROR
Subtask: Loading Configuration on the device (failed)

---- Base Configuration ** changed : False -------------------------------------- INFO
system {
  host-name spine00.cmh;
  domain-name cmh.acme.local;
}
---- Loading Configuration on the device ** changed : False -------------------- ERROR
Traceback (most recent call last):
  File "/Users/dbarroso/.virtualenvs/nornir/lib/python3.6/site-packages/napalm/eos/eos.py", line 231,
    self.device.run_commands(commands)
  File "/Users/dbarroso/.virtualenvs/nornir/lib/python3.6/site-packages/pyeapi/client.py", line 730,
    response = self._connection.execute(commands, encoding, **kwargs)
  File "/Users/dbarroso/.virtualenvs/nornir/lib/python3.6/site-packages/pyeapi/eapilib.py", line 499,
    response = self.send(request)
  File "/Users/dbarroso/.virtualenvs/nornir/lib/python3.6/site-packages/pyeapi/eapilib.py", line 418,
    raise CommandError(code, msg, command_error=err, output=out)
pyeapi.eapilib.CommandError: Error [1002]: CLI command 3 of 6 'system {' failed: invalid command [Inv

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/Users/dbarroso/workspace/nornir/nornir/core/task.py", line 62, in start
    r = self.task(self, **self.params)
  File "/Users/dbarroso/workspace/nornir/nornir/plugins/tasks/networking/napalm_configure.py", line 2
    device.load_merge_candidate(filename=filename, config=configuration)
  File "/Users/dbarroso/.virtualenvs/nornir/lib/python3.6/site-packages/napalm/eos/eos.py", line 245,
    self._load_config(filename, config, False)
  File "/Users/dbarroso/.virtualenvs/nornir/lib/python3.6/site-packages/napalm/eos/eos.py", line 237,
    raise MergeConfigException(e.message)
napalm.base.exceptions.MergeConfigException: Error [1002]: CLI command 3 of 6 'system {' failed: inva

^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* spine01.cmh ** changed : True *************************************************
vvvv basic_configuration ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
```

```
---- Base Configuration ** changed : False ------------------------------------- INFO
system {
  host-name spine01.cmh;
  domain-name cmh.acme.local;
}
---- Loading Configuration on the device ** changed : True --------------------- INFO
[edit system]
-  host-name vsrx;
+  host-name spine01.cmh;
+  domain-name cmh.acme.local;
^^^^ END basic_configuration ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

There is also a method that will raise an exception if the task had an error:

```
In [8]: from nornir.core.exceptions import NornirExecutionError
        try:
            result.raise_on_error()
        except NornirExecutionError:
            print("ERROR!!!")
```

```
ERROR!!!
```

Skipped hosts

Nornir will keep track of hosts that failed and won't run future tasks on them:

```
In [9]: from nornir.core.task import Result

        def hi(task):
            return Result(host=task.host, result=f"{task.host.name}: Hi, I am still here!")

        result = cmh.run(task=hi)
```

```
In [10]: print_result(result)
```

```
hi******************************************************************************
* leaf01.cmh ** changed : False ************************************************
vvvv hi ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
leaf01.cmh: Hi, I am still here!
^^^^ END hi ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* spine01.cmh ** changed : False ***********************************************
vvvv hi ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
spine01.cmh: Hi, I am still here!
^^^^ END hi ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

You can force the execution of tasks on failed hosts by passing the arguments `run_on_failed=True`:

```
In [11]: result = cmh.run(task=hi, on_failed=True)
         print_result(result)
```

```
hi******************************************************************************
* leaf00.cmh ** changed : False ************************************************
vvvv hi ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
leaf00.cmh: Hi, I am still here!
^^^^ END hi ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* leaf01.cmh ** changed : False ************************************************
vvvv hi ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
leaf01.cmh: Hi, I am still here!
^^^^ END hi ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
* spine00.cmh ** changed : False ***********************************************
vvvv hi ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
spine00.cmh: Hi, I am still here!
^^^^ END hi ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* spine01.cmh ** changed : False ***********************************************
vvvv hi ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
spine01.cmh: Hi, I am still here!
^^^^ END hi ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

You can also exclude the hosts that are "good" if you want to with the `run_on_good` flag:

```
In [12]: result = cmh.run(task=hi, on_failed=True, on_good=False)
         print_result(result)
```

```
hi******************************************************************************
* leaf00.cmh ** changed : False ************************************************
vvvv hi ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
leaf00.cmh: Hi, I am still here!
^^^^ END hi ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
* spine00.cmh ** changed : False ***********************************************
vvvv hi ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv INFO
spine00.cmh: Hi, I am still here!
^^^^ END hi ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

To achieve this `nornir` keeps a list failed hosts in it's shared *data* object:

```
In [13]: nr.data.failed_hosts
```

```
Out[13]: {'leaf00.cmh', 'spine00.cmh'}
```

If you want to mark some hosts as succeeded and make them back eligible for future tasks you can do it individually per host with the function *recover_host* or reset the list completely with *reset_failed_hosts*:

```
In [14]: nr.data.reset_failed_hosts()
         nr.data.failed_hosts
```

```
Out[14]: set()
```

Raise on error automatically

Alternatively, you can configure nornir to raise the exception automatically in case of error with the `raise_on_error` configuration option:

```
In [15]: nr = InitNornir(config_file="config.yaml", raise_on_error=True)
         cmh = nr.filter(site="cmh", type="network_device")
         try:
             cmh.run(task=basic_configuration)
         except NornirExecutionError:
             print("ERROR!!!")
```

```
ERROR!!!
```

Workflows

The default workflow should work for most use cases as hosts with errors are skipped and the `print_result` should give enough information to understand what's going on. For more complex workflows this framework should give you enough room to easily implement them regardless of the complexity.

3.2 How to use Nornir

3.2.1 Transforming Inventory Data

Imagine your data looks like:

```
host1:
    username: my_user
    password: my_password
host2:
    username: my_user
    password: my_password
```

It turns out nornir is going to look for `nornir_username` and `nornir_password` to use as credentials. You may not want to change the data in your backend and you may not want to write a custom inventory plugin just to accommodate this difference. Fortunately, `nornir` has you covered. You can write a function to do all the data manipulations you want and pass it to any inventory plugin. For instance:

```
def adapt_host_data(host):
    host.data["nornir_username"] = host.data["username"]
    host.data["nornir_password"] = host.data["password"]


inv = NSOTInventory(transform_function=adapt_host_data)
nornir = Nornir(inventory=inv)
```

What's going to happen is that the inventory is going to create the *`nornir.core.inventory.Host`* and *`nornir.core.inventory.Group`* objects as usual and then finally the `transform_function` is going to be called for each individual host one by one.

Note: This was a very simple example but the `transform_function` can basically do anything you want/need.

3.2.2 Writing a connection task

Connection tasks are tasks that establish a connection with a device to provide some sort of reusable mechanism to interact with it. You can find some examples of connections tasks in the *Connections* section.

Writing a connection task is no different from writing a regular task. The only difference is that the task will have to establish the connection and assign it to the device.

A continuation you can see a simplified version of the `paramiko_connection` connection task as an example:

```
def paramiko_connection(task=None):
    host = task.host

    client = paramiko.SSHClient()

    parameters = {
        "hostname": host.host,
        "username": host.username,
        "password": host.password,
        "port": host.ssh_port,
    }
```

```
    client.connect(**parameters)
    host.connections["paramiko"] = client
```

Note the last line where the connection is assigned to the host. Subsequent tasks will be able to retrieve this connection by host calling `host.get_connection("paramiko")`

3.2.3 Writing a custom inventory

If you have your own backend with host information or you don't like the provided ones you can write your own custom inventory. Doing so is quite easy. A continuation you can find a very simple one with static data.

```
from builtins import super

from nornir.core.inventory import Inventory


class MyInventory(Inventory):

    def __init__(self, **kwargs):
        # code to get the data
        hosts = {
            "host1": {
                "data1": "value1",
                "data2": "value2",
                "groups": ["my_group1"],
            },
            "host2": {
                "data1": "value1",
                "data2": "value2",
                "groups": ["my_group1"],
            }
        }
        groups = {
            "my_group1": {
                "more_data1": "more_value1",
                "more_data2": "more_value2",
            }
        }
        defaults = {
            "location": "internet",
            "language": "Python",
        }

        # passing the data to the parent class so the data is
        # transformed into actual Host/Group objects
        # and set default data for all hosts
        super().__init__(hosts, groups, defaults, **kwargs)
```

So if you want to make it dynamic everything you have to do is get the data yourself and organize it in a similar format to the one described in the example above.

Note: it is not mandatory to use groups or defaults. Feel free to skip the attribute `groups` and just pass and empty dict or `None` to `super()`.

Finally, to have nornir use it, you can:

```
from nornir.core import InitNornir
nr = InitNornir(inventory=MyInventory)
```

And that's it, you now have your own inventory plugin :)

3.3 Configuration

Each configuration parameter are applied in the following order:

1. Environment variable
2. Parameter in configuration file / object
3. Default value

The configuration parameters will be set by the *Nornir.core.configuration.Config* class.

3.3.1 inventory

Path to inventory modules.

3.3.2 jinja_filters

Path to callable returning jinja filters to be used.

3.3.3 logging_dictConfig

Configuration dictionary schema supported by the logging subsystem. Overrides rest of logging_* parameters.

3.3.4 logging_file

Logging file. Empty string disables logging to file.

3.3.5 logging_format

Logging format

3.3.6 logging_level

Logging level. Can be any supported level by the logging subsystem

---

3.3.7 logging_loggers

List of loggers to configure. This allows you to enable logging for multiple loggers. For instance, you could enable logging for both nornir and paramiko or just for paramiko. An empty list will enable logging for all loggers.

3.3.8 logging_to_console

Whether to log to stdout or not.

3.3.9 num_workers

Number of Nornir worker processes that are run at the same time, configuration can be overridden on individual tasks by using the *num_workers* argument to (`nornir.core.Nornir.run`)

3.3.10 raise_on_error

If set to `True`, (`nornir.core.Nornir.run`) method of will raise an exception if at least a host failed.

3.3.11 ssh_config_file

User ssh_config_file

3.3.12 transform_function

Path to transform function. The transform_function you provide will run against each host in the inventory. This is useful to manipulate host data and make it more consumable. For instance, if your inventory has a 'user' attribute you could use this function to map it to 'nornir_user'

3.4 Plugins

3.4.1 Tasks

APIs

nornir.plugins.tasks.apis.**http_method**(*task=None*, *method='get'*, *url=''*, *raise_for_status=True*, ***kwargs*)

This is a convenience task that uses requests to interact with an HTTP server.

Parameters

- **method** (*string*) – HTTP method to call
- **url** (*string*) – URL to connect to
- **raise_for_status** (*bool*) – Whether to call raise_for_status method automatically or not. For quick reference, raise_for_status will consider an error if the return code is any of 4xx or 5xx
- ****kwargs** – Keyword arguments will be passed to the request method

Returns

- result (string/dict): Body of the response. Either text or a dict if the response was a json object
- reponse (object): Original Response

Return type *nornir.core.task.Result*

Commands

nornir.plugins.tasks.commands.**command**(*task*, *command*)

Executes a command locally

Parameters **command** (str) – command to execute

Returns

- result (str): stderr or stdout
- stdout (str): stdout
- stderr (srr): stderr

Return type *nornir.core.task.Result*

Raises *nornir.core.exceptions.CommandError* – when there is a command error

nornir.plugins.tasks.commands.**remote_command**(*task*, *command*)

Executes a command locally

Parameters **command** (str) – command to execute

Returns

- result (str): stderr or stdout
- stdout (str): stdout
- stderr (srr): stderr

Return type *nornir.core.task.Result*

Raises *nornir.core.exceptions.CommandError* – when there is a command error

Connections

`nornir.plugins.tasks.connections.`**`napalm_connection`**(*task=None*, *timeout=60*, *optional_args=None*)

This tasks connects to the device using the NAPALM driver and sets the relevant connection.

Parameters

- **timeout** (*int, optional*) – defaults to 60
- **optional_args** (*dict, optional*) – defaults to *{"port": task.host["nornir_network_api_port"]}*

Inventory: napalm_options: maps directly to `optional_args` when establishing the connection network_api_port: maps to `optional_args["port"]`

`nornir.plugins.tasks.connections.`**`netmiko_connection`**(*task*, ***netmiko_args*)

Connect to the host using Netmiko and set the relevant connection in the connection map.

Precedence: `**netmiko_args` > discrete inventory attributes > inventory netmiko_options

Parameters **`**netmiko_args`** – All supported Netmiko ConnectHandler arguments

`nornir.plugins.tasks.connections.`**`paramiko_connection`**(*task=None*)

This tasks connects to the device with paramiko to the device and sets the relevant connection.

Data

`nornir.plugins.tasks.data.`**`load_json`**(*task*, *file*)

Loads a json file.

Parameters **file** (*str*) – path to the file containing the json file to load

Returns

- result (`dict`): dictionary with the contents of the file

Return type *nornir.core.task.Result*

`nornir.plugins.tasks.data.`**`load_yaml`**(*task*, *file*)

Loads a yaml file.

Parameters **file** (*str*) – path to the file containing the yaml file to load

Returns

- result (`dict`): dictionary with the contents of the file

Return type *nornir.core.task.Result*

Files

`nornir.plugins.tasks.files.`**`sftp`**(*task*, *src*, *dst*, *action*, *dry_run=None*)

Transfer files from/to the device using sftp protocol

Example:

```
nornir.run(files.sftp,
           action="put",
           src="README.md",
           dst="/tmp/README.md")
```

Parameters

- **dry_run** (*bool*) – Whether to apply changes or not
- **src** (str) – source file
- **dst** (str) – destination
- **action** (str) – put, get.

Returns

- changed (bool):
- files_changed (list): list of files that changed

Return type *nornir.core.task.Result*

nornir.plugins.tasks.files.**write_file**(*task*, *filename*, *content*, *append=False*, *dry_run=None*)

Write contents to a file (locally)

Parameters

- **dry_run** (*bool*) – Whether to apply changes or not
- **filename** (str) – file you want to write into
- **conteint** (str) – content you want to write
- **append** (bool) – whether you want to replace the contents or append to it

Returns

- diff (str): unified diff

Return type

- changed (bool)

Networking

nornir.plugins.tasks.networking.**napalm_cli**(*task*, *commands*)

Run commands on remote devices using napalm

Parameters **commands** (list) – List of commands to execute

Returns

- result (dict): dictionary with the result of the commands

Return type *nornir.core.task.Result*

nornir.plugins.tasks.networking.**napalm_configure**(*task*, *dry_run=None*, *filename=None*, *configuration=None*, *replace=False*)

Loads configuration into a network devices using napalm

Parameters

- **dry_run** (*bool*) – Whether to apply changes or not

- **configuration** (*str*) – configuration to load into the device
- **filename** (*str*) – filename containing the configuration to load into the device
- **replace** (*bool*) – whether to replace or merge the configuration

Returns

- changed (`bool`): whether if the task is changing the system or not
- diff (`string`): change in the system

Return type *nornir.core.task.Result*

`nornir.plugins.tasks.networking.`**`napalm_get`**(*task*, *getters*, *getters_options=None*, ***kwargs*)

Gather information from network devices using napalm

Parameters

- **getters** (*list of str*) – getters to use
- **getters_options** (*dict of dicts*) – When passing multiple getters you pass a dictionary where the outer key is the getter name and the included dictionary represents the options to pass to the getter
- ****kwargs** – will be passed as they are to the getters

Examples

Simple example:

```
> nr.run(task=napalm_get,
>        getters=["interfaces", "facts"])
```

Passing options using `**kwargs`:

```
> nr.run(task=napalm_get,
>        getters=["config"],
>        retrieve="all")
```

Passing options using `getters_options`:

```
> nr.run(task=napalm_get,
>        getters=["config", "interfaces"],
>        getters_options={"config": {"retrieve": "all"}})
```

Returns

- result (`dict`): dictionary with the result of the getter

Return type *nornir.core.task.Result*

`nornir.plugins.tasks.networking.`**`napalm_validate`**(*task*, *src=None*, *validation_source=None*)

Gather information with napalm and validate it:

http://napalm.readthedocs.io/en/develop/validate/index.html

Parameters

- **src** – file to use as validation source

- **validation_source** (*list*) – instead of a file data needed to validate device's state

Returns

- result (dict): dictionary with the result of the validation
- complies (bool): Whether the device complies or not

Return type *nornir.core.task.Result*

nornir.plugins.tasks.networking.**netmiko_file_transfer**(*task*, *source_file*, *dest_file*, ***kwargs*)

Execute Netmiko file_transfer method

Parameters

- **source_file** (*str*) – Source file.
- **dest_file** (*str*) – Destination file.
- **kwargs** (*dict, optional*) – Additional arguments to pass to file_transfer

Returns

- result (bool): file exists and MD5 is valid
- changed (bool): the destination file was changed

Return type *nornir.core.task.Result*

nornir.plugins.tasks.networking.**netmiko_send_command**(*task*, *command_string*, *use_timing=False*, ***kwargs*)

Execute Netmiko send_command method (or send_command_timing)

Parameters

- **command_string** (*str*) – Command to execute on the remote network device.
- **use_timing** (*bool, optional*) – Set to True to switch to send_command_timing method.
- **kwargs** (*dict, optional*) – Additional arguments to pass to send_command method.

Returns

- result (dict): dictionary with the result of the show command.

Return type *nornir.core.task.Result*

nornir.plugins.tasks.networking.**netmiko_send_config**(*task*, *config_commands=None*, *config_file=None*, ***kwargs*)

Execute Netmiko send_config_set method (or send_config_from_file) :param config_commands: Commands to configure on the remote network device. :type config_commands: list, optional :param config_file: File to read configuration commands from. :type config_file: str, optional :param kwargs: Additional arguments to pass to method. :type kwargs: dict, optional

Returns

- result (dict): dictionary showing the CLI from the configuration changes.

Return type *nornir.core.task.Result*

nornir.plugins.tasks.networking.**tcp_ping**(*task*, *ports*, *timeout=2*, *host=None*)

Tests connection to a tcp port and tries to establish a three way handshake. To be used for network discovery or testing.

Parameters

- **ports** (*list of int*) – tcp port to ping
- **timeout** (*int, optional*) – defaults to 0.5
- **host** (*string, optional*) – defaults to `nornir_ip`

Returns

- result (`dict`): Contains port numbers as keys with True/False as values

Return type *nornir.core.task.Result*

Text

`nornir.plugins.tasks.text.`**`template_file`**(*task*, *template*, *path*, *jinja_filters=None*, ***kwargs*)

Renders contants of a file with jinja2. All the host data is available in the tempalte

Parameters

- **template** (*string*) – filename
- **path** (*string*) – path to dir with templates
- **jinja_filters** (*dict*) – jinja filters to enable. Defaults to nornir.config.jinja_filters
- ****kwargs** – additional data to pass to the template

Returns

- result (`string`): rendered string

Return type *nornir.core.task.Result*

`nornir.plugins.tasks.text.`**`template_string`**(*task*, *template*, *jinja_filters=None*, ***kwargs*)

Renders a string with jinja2. All the host data is available in the tempalte

Parameters

- **template** (*string*) – template string
- **jinja_filters** (*dict*) – jinja filters to enable. Defaults to nornir.config.jinja_filters
- ****kwargs** – additional data to pass to the template

Returns

- result (`string`): rendered string

Return type *nornir.core.task.Result*

3.4.2 Functions

Text

`nornir.plugins.functions.text.`**`print_result`**(*result*, *host=None*, *vars=None*, *failed=None*, *severity_level=20*)

Prints the *nornir.core.task.Result* from a previous task to screen

Parameters

- **result** (*nornir.core.task.Result*) – from a previous task
- **vars** (*list of str*) – Which attributes you want to print

- **failed** (bool) – if True assume the task failed
- **severity_level** (*int*) – Print only errors with this severity level or higher

Returns

Return type *nornir.core.task.Result*

nornir.plugins.functions.text.**print_title**(*title*)
Helper function to print a title.

3.4.3 Inventory

Simple

class nornir.plugins.inventory.simple.**SimpleInventory**(*host_file='hosts.yaml'*, *group_file='groups.yaml'*, ***kwargs*)

This is a very simple file based inventory. Basically you need two yaml files. One for your host information and another one for your group information.

- host file:

```
---
host1.cmh:
    site: cmh
    role: host
    groups:
        - cmh-host
    nos: linux

host2.cmh:
    site: cmh
    role: host
    groups:
        - cmh-host
    nos: linux

switch00.cmh:
    nornir_ip: 127.0.0.1
    nornir_username: vagrant
    nornir_password: vagrant
    napalm_port: 12443
    site: cmh
    role: leaf
    groups:
        - cmh-leaf
    nos: eos

switch01.cmh:
    nornir_ip: 127.0.0.1
    nornir_username: vagrant
    nornir_password: ""
    napalm_port: 12203
    site: cmh
    role: leaf
    groups:
        - cmh-leaf
    nos: junos
```

```
host1.bma:
    site: bma
    role: host
    groups:
        - bma-host
    nos: linux

host2.bma:
    site: bma
    role: host
    groups:
        - bma-host
    nos: linux

switch00.bma:
    nornir_ip: 127.0.0.1
    nornir_username: vagrant
    nornir_password: vagrant
    napalm_port: 12443
    site: bma
    role: leaf
    groups:
        - bma-leaf
    nos: eos

switch01.bma:
    nornir_ip: 127.0.0.1
    nornir_username: vagrant
    nornir_password: ""
    napalm_port: 12203
    site: bma
    role: leaf
    groups:
        - bma-leaf
    nos: junos
```

- group file:

```
---
defaults:
    domain: acme.com

bma-leaf:
    groups:
        - bma

bma-host:
    groups:
        - bma

bma:
    domain: bma.acme.com

cmh-leaf:
    groups:
        - cmh
```

```
cmh-host:
    groups:
        - cmh

cmh:
    domain: cmh.acme.com
```

Ansible

class nornir.plugins.inventory.ansible.**AnsibleInventory**(*hostsfile='hosts'*, ***kwargs*)
Inventory plugin that is capable of reading an ansible inventory.

Parameters **hostsfile** (*string*) – Ansible inventory file to load

class nornir.plugins.inventory.ansible.**AnsibleParser**(*hostsfile*)

add(*element*, *element_dict*)

map_nornir_vars(*obj*)

parse()

parse_group(*group*, *data*, *parent=None*)

parse_hosts(*hosts*, *parent=None*)

read_vars_file(*element*, *path*, *is_host=True*)

sort_groups()

class nornir.plugins.inventory.ansible.**INIParser**(*hostsfile*)

load_hosts_file()

normalize(*data*)

normalize_content(*content*)

class nornir.plugins.inventory.ansible.**YAMLParser**(*hostsfile*)

load_hosts_file()

nornir.plugins.inventory.ansible.**parse**(*hostsfile*)

NSOT

class nornir.plugins.inventory.nsot.**NSOTInventory**(*nsot_url='‘*, *nsot_email='‘*, *nsot_auth_method='‘*, *nsot_secret_key='‘*, *nsot_auth_header='‘*, *flatten_attributes=True*, ***kwargs*)
Inventory plugin that uses nsot as backend.

Note: An extra attribute site will be assigned to the host. The value will be the name of the site the host belongs to.

Environment Variables:

- `NSOT_URL`: Corresponds to nsot_url argument
- `NSOT_EMAIL`: Corresponds to nsot_email argument
- `NSOT_AUTH_HEADER`: Corresponds to nsot_auth_header argument
- `NSOT_SECRET_KEY`: Corresponds to nsot_secret_key argument

Parameters

- **flatten_attributes** (*bool*) – Assign host attributes to the root object. Useful for filtering hosts.
- **nsot_url** (*string*) – URL to nsot's API (defaults to `http://localhost:8990/api`)
- **nsot_email** (*string*) – email for authtication (defaults to admin@acme.com)
- **nsot_auth_header** (*string*) – String for auth_header authentication (defaults to X-NSoT-Email)
- **nsot_secret_key** (*string*) – Secret Key for auth_token method. If given auth_token will be used as auth_method.

3.5 Reference Guides

3.5.1 Nornir's Internals

Execution Model

One of the many advantages of using nornir is that it will be parallelize the execution of tasks for you. The way it works is as follows:

1. You trigger the parallelization by running a task via *nornir.core.Nornir.run* with `num_workers > 1` (defaults to `20`).
2. If `num_workers == 1` we run the task over all hosts one after the other in a simple loop. This is useful for troubleshooting/debugging, for writing to disk/database or just for printing on screen.
3. When parallelizing tasks nornir will use a different thread for each host.

Below you can see a simple diagram illustrating how this works:

1. my_first_task
host1 host2 ... hostN
hostN+1 hostN+2 ... hostN+N
lastHost

RELATIVE EXECUTION TIME
1.1
1.2
1.$(NUM_HOSTS/N)

```
1 brigade.run(my_first_task)
2 brigade.run(my_second_task)
3 brigade.run(my_third_task,
              num_workers=1)
```

2. my_second_task
host1 host2 ... hostN
hostN+1 hostN+2 ... hostN+N
lastHost

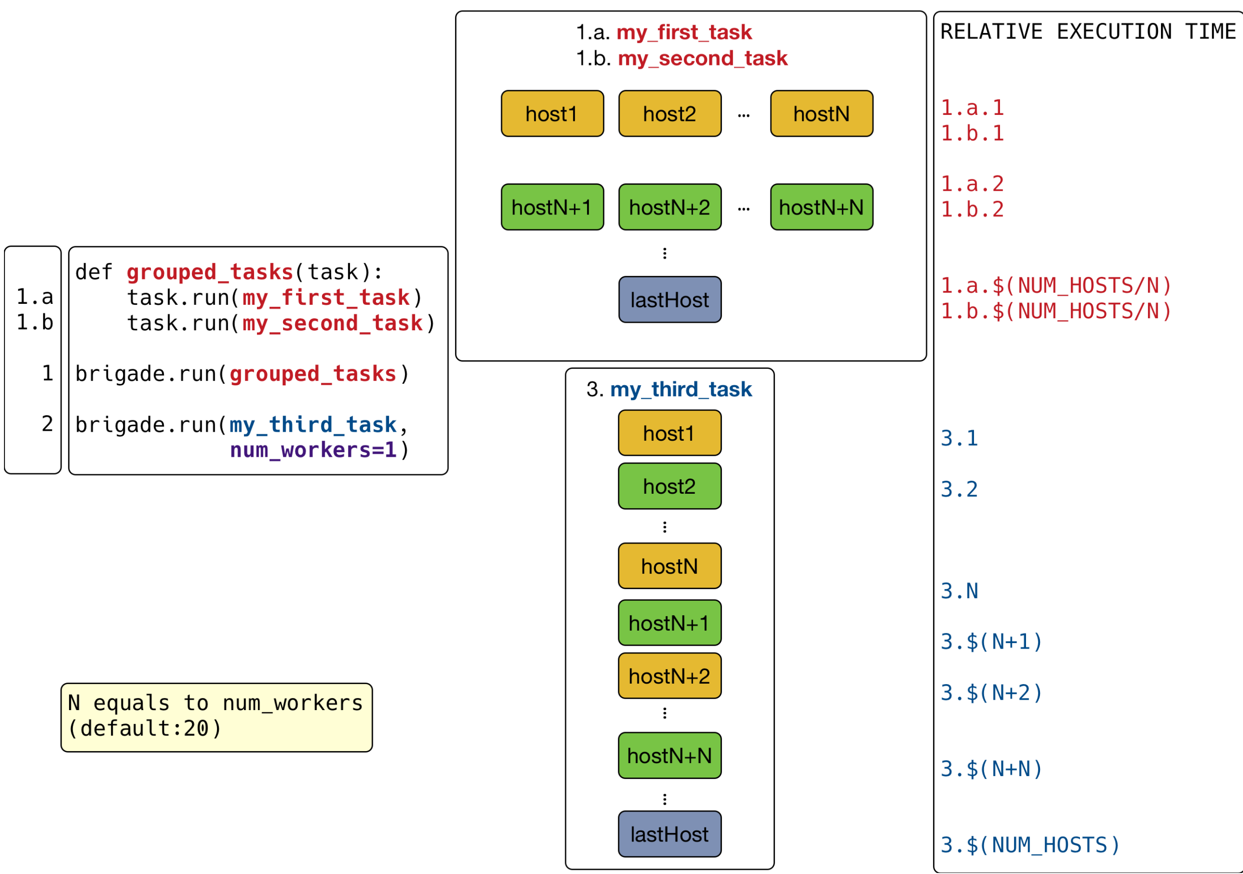
2.1
2.2
2.$(NUM_HOSTS/N)

3. my_third_task
host1
host2
hostN
hostN+1
hostN+2
hostN+N
lastHost

3.1
3.2
3.N
3.$(N+1)
3.$(N+2)
3.$(N+N)
3.$(NUM_HOSTS)

N equals to num_workers (default:20)

Note that you can create tasks with other tasks inside. When tasks are nested the inner tasks will run serially for that host in parallel to other hosts. This is useful as it let's you control the flow of the execution at your own will. For instance, you could compose a different workflow to the previous one as follows:



Why would you do this? Most of the time you will want to group as many tasks as possible. That will ensure your script runs as fast as possible. However, some tasks might require to be run after ensuring the some others are done. For instance, you could do something like:

1. Configure everything in parallel
2. Run some verification tests
3. Enable services

3.5.2 Nornir API Reference

Data

class `nornir.core.`**`Data`**

This class is just a placeholder to share data amongst different versions of Nornir after running `filter` multiple times.

`failed_hosts`

list – Hosts that have failed to run a task properly

`recover_host` (*host*)

Remove `host` from list of failed hosts.

`reset_failed_hosts` ()

Reset failed hosts and make all hosts available for future tasks.

`to_dict` ()

Return a dictionary representing the object.

Nornir

class nornir.core.**Nornir**(*inventory*, *dry_run*, *config=None*, *config_file=None*, *available_connections=None*, *logger=None*, *data=None*)

This is the main object to work with. It contains the inventory and it serves as task dispatcher.

Parameters

- **inventory** (*nornir.core.inventory.Inventory*) – Inventory to work with
- **data** (*nornir.core.Data*) – shared data amongst different iterations of nornir
- **dry_run** (bool) – Whether if we are testing the changes or not
- **config** (*nornir.core.configuration.Config*) – Configuration object
- **config_file** (str) – Path to Yaml configuration file
- **available_connections** (dict) – dict of connection types that will be made available. Defaults to nornir.plugins.tasks.connections.available_connections

inventory

nornir.core.inventory.Inventory – Inventory to work with

data

nornir.core.Data – shared data amongst different iterations of nornir

dry_run

bool – Whether if we are testing the changes or not

config

nornir.core.configuration.Config – Configuration parameters

available_connections

dict – dict of connection types are available

configure_logging()

dry_run

filter(***kwargs*)

See *nornir.core.inventory.Inventory.filter()*

Returns A new object with same configuration as self but filtered inventory.

Return type Nornir

run(*task*, *num_workers=None*, *raise_on_error=None*, *on_good=True*, *on_failed=False*, ***kwargs*)

Run task over all the hosts in the inventory.

Parameters

- **task** (callable) – function or callable that will be run against each device in the inventory
- **num_workers** (int) – Override for how many hosts to run in paralell for this task
- **raise_on_error** (bool) – Override raise_on_error behavior
- **on_good** (bool) – Whether to run or not this task on hosts marked as good
- **on_failed** (bool) – Whether to run or not this task on hosts marked as failed
- ****kwargs** – additional argument to pass to task when calling it

Raises *nornir.core.exceptions.NornirExecutionError* – if at least a task fails and self.config.raise_on_error is set to `True`

Returns results of each execution

Return type *nornir.core.task.AggregatedResult*

to_dict()
Return a dictionary representing the object.

InitNornir

core.**InitNornir**(*config_file=''*, *dry_run=False*, ***kwargs*)

Parameters

- **config_file** (*str*) – Path to the configuration file (optional)
- **dry_run** (*bool*) – Whether to simulate changes or not
- ****kwargs** – Extra information to pass to the *nornir.core.configuration.Config* object

Returns fully instantiated and configured

Return type *nornir.core.Nornir*

Configuration

class nornir.core.configuration.**Config**(*config_file=None*, ***kwargs*)
This object handles the configuration of Nornir.

Parameters **config_file** (str) – Yaml configuration file.

get(*parameter*, *env=None*, *default=None*, *parameter_type='str'*, *root=''*)
Retrieve a custom parameter from the configuration.

Parameters

- **parameter** (*str*) – Name of the parameter to retrieve
- **env** (*str*) – Environment variable name to retrieve the object from
- **default** – default value in case no parameter is found
- **parameter_type** (*str*) – if a value is found cast the variable to this type
- **root** (*str*) – parent key in the configuration file where to look for the parameter

string_to_bool(*v*)

The attributes for the Config object will be the Nornir configuration parameters. For a list of available parameters see *Nornir configuration parameters*

Inventory

Inventory

class nornir.core.inventory.**Inventory**(*hosts*, *groups=None*, *defaults=None*, *transform_function=None*, *nornir=None*)
An inventory contains information about hosts and groups.

Parameters

- **hosts** (*dict*) – keys are hostnames and values are either `Host` or a dict representing the host data.
- **groups** (*dict*) – keys are group names and values are either `Group` or a dict representing the group data.
- **transform_function** (*callable*) – we will call this function for each host. This is useful to manipulate host data and make it more consumable. For instance, if your inventory has a "user" attribute you could use this function to map it to "nornir_user"

hosts
dict – keys are hostnames and values are `Host`.

groups
dict – keys are group names and the values are `Group`.

filter (*filter_func=None*, ***kwargs*)
Returns a new inventory after filtering the hosts by matching the data passed to the function. For instance, assume an inventory with:

```
---
host1:
    site: bma
    role: http
host2:
    site: cmh
    role: http
host3:
    site: bma
    role: db
```

- `my_inventory.filter(site="bma")` will result in `host1` and `host3`
- `my_inventory.filter(site="bma", role="db")` will result in `host3` only

Parameters filter_func (*callable*) – if filter_func is passed it will be called against each device. If the call returns `True` the device will be kept in the inventory

nornir
Reference to the parent `nornir.core.Nornir` object

to_dict()
Return a dictionary representing the object.

Host

class `nornir.core.inventory.`**Host**(*name*, *groups=None*, *nornir=None*, *defaults=None*, ***kwargs*)
Represents a host.

Parameters

- **name** (*str*) – Name of the host
- **group** (`Group`, optional) – Group the host belongs to
- **nornir** (`nornir.core.Nornir`) – Reference to the parent nornir object
- ****kwargs** – Host data

name
str – Name of the host

groups
list of `Group` – Groups the host belongs to

defaults
`dict` – Default values for host/group data

data
dict – data about the device

connections
`dict` – Already established connections

Note: You can access the host data in two ways:

1. Via the `data` attribute - In this case you will get access **only** to the data that belongs to the host. 2. Via the host itself as a dict - `Host` behaves like a dict. The difference between accessing data via the `data` attribute and directly via the host itself is that the latter will also return the data if it's available via a parent `Group`.

For instance:

```
---
# hosts
my_host:
    ip: 1.2.3.4
    groups: [bma]

---
# groups
bma:
    site: bma

defaults:
    domain: acme.com
```

- `my_host.data["ip"]` will return `1.2.3.4`
- `my_host["ip"]` will return `1.2.3.4`
- `my_host.data["site"]` will `fail`
- `my_host["site"]` will return `bma`
- `my_host.data["domain"]` will `fail`
- `my_host.group.data["domain"]` will `fail`
- `my_host["domain"]` will return `acme.com`
- `my_host.group["domain"]` will return `acme.com`
- `my_host.group.group.data["domain"]` will return `acme.com`

---

get (*item*, *default=None*)
Returns the value `item` from the host or hosts group variables.

Parameters

- **item** (str) – The variable to get
- **default** (any) – Return value if item not found

get_connection (*connection*)
The function of this method is twofold:

1. If an existing connection is already established for the given type return it
2. **If non exists, establish a new connection of that type with default parameters** and return it

Raises AttributeError – if it's unknown how to establish a connection for the given type

Parameters **connection_name** (*str*) – Name of the connection, for instance, netmiko, paramiko, napalm...

Returns An already established connection of type connection

has_parent_group (*group*)
Retuns whether the object is a child of the Group group

host
String used to connect to the device. Either nornir_host or self.name

items ()
Returns all the data accessible from a device, including the one inherited from parent groups

keys ()
Returns the keys of the attribute data and of the parent(s) groups.

network_api_port
For network equipment this is the port where the device's API is listening to. Either nornir_network_api_port or None.

nornir
Reference to the parent *nornir.core.Nornir* object

nos
Network OS the device is running. Defaults to nornir_nos.

os
OS the device is running. Defaults to nornir_os.

password
Either nornir_password or empty string.

ssh_port
Either nornir_ssh_port or None.

to_dict ()
Return a dictionary representing the object.

username
Either nornir_username or user running the script.

values ()
Returns the values of the attribute data and of the parent(s) groups.

Group

class nornir.core.inventory.**Group** (*name*, *groups=None*, *nornir=None*, *defaults=None*, ***kwargs*)
Same as Host

Task

class `nornir.core.task.`**`Task`**(*task*, *name=None*, *severity_level=20*, ***kwargs*)

A task is basically a wrapper around a function that has to be run against multiple devices. You won't probably have to deal with this class yourself as `nornir.core.Nornir.run()` will create it automatically.

Parameters

- **`task`** (*`callable`*) – function or callable we will be calling
- **`name`** (`string`) – name of task, defaults to `task.__name__`
- **`severity_level`** (*`logging.LEVEL`*) – Severity level associated to the task
- **`**kwargs`** – Parameters that will be passed to the `task`

`task`

callable – function or callable we will be calling

`name`

`string` – name of task, defaults to `task.__name__`

`params`

Parameters that will be passed to the `task`.

`self.`**`results`**

`nornir.core.task.MultiResult` – Intermediate results

`host`

`nornir.core.inventory.Host` – Host we are operating with. Populated right before calling the `task`

`nornir`

`nornir.core.Nornir` – Populated right before calling the `task`

`severity_level`

logging.LEVEL – Severity level associated to the task

`is_dry_run`(*override=None*)

Returns whether current task is a dry_run or not.

Parameters **`override`** (*`bool`*) – Override for current task

`run`(*task*, ***kwargs*)

This is a utility method to call a task from within a task. For instance:

def grouped_tasks(task): task.run(my_first_task) task.run(my_second_task)

nornir.run(grouped_tasks)

This method will ensure the subtask is run only for the host in the current thread.

`start`(*host*, *nornir*)

Run the task for the given host.

Parameters

- **`host`** (*`nornir.core.inventory.Host`*) – Host we are operating with. Populated right before calling the `task`
- **`nornir`** (*`nornir.core.Nornir`*) – Populated right before calling the `task`

Returns Results of the task and its subtasks

Return type host (*`nornir.core.task.MultiResult`*)

Result

class `nornir.core.task.`**`Result`**(*host*, *result=None*, *changed=False*, *diff=''*, *failed=False*, *exception=None*, *severity_level=20*, ***kwargs*)

Result of running individual tasks.

Parameters

- **`changed`** (*`bool`*) – `True` if the task is changing the system
- **`diff`** (*`obj`*) – Diff between state of the system before/after running this task
- **`result`** (*`obj`*) – Result of the task execution, see task's documentation for details
- **`host`** (*`nornir.core.inventory.Host`*) – Reference to the host that lead ot this result
- **`failed`** (*`bool`*) – Whether the execution failed or not
- **`severity_level`** (*`logging.LEVEL`*) – Severity level associated to the result of the excecution
- **`exception`** (*`Exception`*) – uncaught exception thrown during the exection of the task (if any)

`changed`
bool – `True` if the task is changing the system

`diff`
obj – Diff between state of the system before/after running this task

`result`
obj – Result of the task execution, see task's documentation for details

`host`
`nornir.core.inventory.Host` – Reference to the host that lead ot this result

`failed`
bool – Whether the execution failed or not

`severity_level`
logging.LEVEL – Severity level associated to the result of the excecution

`exception`
Exception – uncaught exception thrown during the exection of the task (if any)

AggregatedResult

class `nornir.core.task.`**`AggregatedResult`**(*name*, ***kwargs*)

It basically is a dict-like object that aggregates the results for all devices. You can access each individual result by doing `my_aggr_result["hostname_of_device"]`.

`failed`
If `True` at least a host failed.

`failed_hosts`
Hosts that failed during the execution of the task.

`raise_on_error`()

Raises *`nornir.core.exceptions.NornirExecutionError`* – When at least a task failed

MultiResult

class `nornir.core.task.`**`MultiResult`**(*name*)

It is basically is a list-like object that gives you access to the results of all subtasks for a particular device/task.

`changed`

If `True` at least a task changed the system.

`failed`

If `True` at least a task failed.

`raise_on_error`()

Raises *`nornir.core.exceptions.NornirExecutionError`* – When at least a task failed

Exceptions

exception `nornir.core.exceptions.`**`CommandError`**(*command*, *status_code*, *stdout*, *stderr*)

Raised when there is a command error.

`status_code`

int – status code returned by the command

`stdout`

str – stdout

`stderr`

str – stderr

`command`

str – command that triggered the error

exception `nornir.core.exceptions.`**`NornirExecutionError`**(*result*)

Raised by nornir when any of the tasks managed by *`nornir.core.Nornir.run()`* when any of the tasks fail.

Parameters **`result`** (*`nornir.core.task.AggregatedResult`*) –

`result`

`nornir.core.task.AggregatedResult`

`failed_hosts`

exception `nornir.core.exceptions.`**`NornirSubTaskError`**(*task*, *result*)

Raised by nornir when a sub task managed by `nornir.core.Task.run()` has failed

Parameters

- **`task`** (*`nornir.core.task.Task`*) – The subtask that failed
- **`result`** (*`nornir.core.task.Result`*) – The result of the failed task

`task`

`nornir.core.task.Task` – The subtask that failed

`result`

`nornir.core.task.Result` – The result of the failed task

3.6 How to contribute to Nornir

First of all, thank you for conidering to contribute to this project!

3.6.1 Several ways to contribute

There are several things you can do to help the project.

- Spread the word about Nornir
- Suggest great features
- Report bugs
- Fix typos
- Write documentation
- Contribute your plugins
- Improve the Nornir core

3.6.2 Spread the word about Nornir

Even if you aren't in the position that you can contribute your time to this project, it still helps us if you spread the word about the project. It could be just a short notice in social media or a discussion you have with your friends. As more people become aware of the project there's a better chance that we reach people who are able to contribute. So, even if you can't directly contribute yourself, someone you refer to us might.

3.6.3 Suggesting new features

It could be that you are aware of something that would be great to have in Nornir and we are always welcoming feature requests. Make sure you explain in what scenario your suggested feature would be useful.

3.6.4 Reporting bugs

When you are reporting bugs, make sure that you give a explaination about the outcome that you expect and what you are seeing. The bugs which are hardest to fix are the ones which we are unable to reproduce. For this reason it's important that you describe what you did and show us how we can reproduce the bug in another environment.

3.6.5 Fix typos

While we try to take care, getting all the works correct can be.. differcult. Typos are the easiest things to fix and if you find any you can help us from looking silly. You can find more typos to fix by looking in the Nornir source code or by visiting the Nornir documentation.

3.6.6 Writing documentation

Documentation is another great way to help if you don't want to contribute actual code. The documentation of Nornir is divided into different sections.

- Tutorials: Aims to help people learn Nornir with a lot of handholding, the user might not end up with something useful after following the tutorial. The goal is for people to learn how to use Nornir.
- How-to guides: This sections goal is to help people solve a specific task with Nornir
- Reference guides: This section describe the Nornir API and plugins. Most of the content in this area is generated from the source code itself.

Contributions to the documentation can be small fixes such as changing scentences to make the text more clear, or it could be new guides.

3.6.7 Contributing plugins

If you have written your custom plugin for Nornir there's a good chance that it can be useful for others as well. General guidelines when writing plugins are:

- Make them as generic as possible, it doesn't help others if they only work in your environment
- Make sure that it's possible to have unit tests which automatically test that the plugins are working

3.6.8 Contributing to the Nornir core

When you are contributing code to the core of Nornir make sure that the existing tests are passing, and add tests to the code you have added. Having your tests in place ensures that other won't accidentally brake it in the future.

Before you make any significant code changes to the core it's recommended that you open an issue to discuss your ideas before writing the code.

3.6.9 Setting up your environment

In order to run tests locally you need to have Docker and Pandoc installed. Docker is used to test the Nornir plugins and Pandoc is required for building the documentation provided by Sphinx. After those are installed you can go ahead and install the needed Python dependencies.

```
pip -r requirements-dev.txt
```

3.6.10 Running tests

While the automated tests will be triggered when you submit a new pull request it can still save you time to run the tests locally first.

```
make tests
```

The test above will run the tests against the Nornir code and documentation.

3.6.11 Coding style

Nornir uses Black, the the uncompromising Python code formatter. Black makes it easy for you to format your code as you can do so automatically after installing it. Note that Python 3.6 is required to run Black.

```
make format
```

The Black GitHub repo has information about how you can integrate Black in your editor.

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

nornir.core.exceptions, 53
nornir.plugins.functions.text, 39
nornir.plugins.inventory.ansible, 42
nornir.plugins.inventory.nsot, 42
nornir.plugins.inventory.simple, 40
nornir.plugins.tasks.apis, 34
nornir.plugins.tasks.commands, 34
nornir.plugins.tasks.connections, 35
nornir.plugins.tasks.data, 35
nornir.plugins.tasks.files, 35
nornir.plugins.tasks.networking, 36
nornir.plugins.tasks.text, 39

Index

A

add() (nornir.plugins.inventory.ansible.AnsibleParser method), 42
AggregatedResult (class in nornir.core.task), 52
AnsibleInventory (class in nornir.plugins.inventory.ansible), 42
AnsibleParser (class in nornir.plugins.inventory.ansible), 42
available_connections (Nornir attribute), 46

C

changed (nornir.core.task.MultiResult attribute), 53
changed (Result attribute), 52
command (nornir.core.exceptions.CommandError attribute), 53
command() (in module nornir.plugins.tasks.commands), 34
CommandError, 53
Config (class in nornir.core.configuration), 47
config (Nornir attribute), 46
configure_logging() (nornir.core.Nornir method), 46
connections (Host attribute), 49

D

Data (class in nornir.core), 45
data (Host attribute), 49
data (Nornir attribute), 46
defaults (Host attribute), 49
diff (Result attribute), 52
dry_run (Nornir attribute), 46
dry_run (nornir.core.Nornir attribute), 46

E

exception (Result attribute), 52

F

failed (nornir.core.task.AggregatedResult attribute), 52
failed (nornir.core.task.MultiResult attribute), 53
failed (Result attribute), 52
failed_hosts (Data attribute), 45
failed_hosts (nornir.core.exceptions.NornirExecutionError attribute), 53
failed_hosts (nornir.core.task.AggregatedResult attribute), 52
filter() (nornir.core.inventory.Inventory method), 48
filter() (nornir.core.Nornir method), 46

G

get() (nornir.core.configuration.Config method), 47
get() (nornir.core.inventory.Host method), 49
get_connection() (nornir.core.inventory.Host method), 50
Group (class in nornir.core.inventory), 50
groups (Host attribute), 49
groups (Inventory attribute), 48

H

has_parent_group() (nornir.core.inventory.Host method), 50
Host (class in nornir.core.inventory), 48
host (nornir.core.inventory.Host attribute), 50
host (Result attribute), 52
host (Task attribute), 51
hosts (Inventory attribute), 48
http_method() (in module nornir.plugins.tasks.apis), 34

I

INIParser (class in nornir.plugins.inventory.ansible), 42
InitNornir() (nornir.core method), 47
Inventory (class in nornir.core.inventory), 47
inventory (Nornir attribute), 46
is_dry_run() (nornir.core.task.Task method), 51
items() (nornir.core.inventory.Host method), 50

K

keys() (nornir.core.inventory.Host method), 50

L

load_hosts_file() (nornir.plugins.inventory.ansible.INIParser method), 42

load_hosts_file() (nornir.plugins.inventory.ansible.YAMLParser method), 42
load_json() (in module nornir.plugins.tasks.data), 35
load_yaml() (in module nornir.plugins.tasks.data), 35

M

map_nornir_vars() (nornir.plugins.inventory.ansible.AnsibleParser method), 42
MultiResult (class in nornir.core.task), 53

N

name (Host attribute), 48
name (Task attribute), 51
napalm_cli() (in module nornir.plugins.tasks.networking), 36
napalm_configure() (in module nornir.plugins.tasks.networking), 36
napalm_connection() (in module nornir.plugins.tasks.connections), 35
napalm_get() (in module nornir.plugins.tasks.networking), 37
napalm_validate() (in module nornir.plugins.tasks.networking), 37
netmiko_connection() (in module nornir.plugins.tasks.connections), 35
netmiko_file_transfer() (in module nornir.plugins.tasks.networking), 38
netmiko_send_command() (in module nornir.plugins.tasks.networking), 38
netmiko_send_config() (in module nornir.plugins.tasks.networking), 38
network_api_port (nornir.core.inventory.Host attribute), 50
normalize() (nornir.plugins.inventory.ansible.INIParser method), 42
normalize_content() (nornir.plugins.inventory.ansible.INIParser method), 42
Nornir (class in nornir.core), 46
nornir (nornir.core.inventory.Host attribute), 50
nornir (nornir.core.inventory.Inventory attribute), 48
nornir (Task attribute), 51
nornir.core.exceptions (module), 53
nornir.plugins.functions.text (module), 39
nornir.plugins.inventory.ansible (module), 42
nornir.plugins.inventory.nsot (module), 42
nornir.plugins.inventory.simple (module), 40
nornir.plugins.tasks.apis (module), 34
nornir.plugins.tasks.commands (module), 34
nornir.plugins.tasks.connections (module), 35
nornir.plugins.tasks.data (module), 35
nornir.plugins.tasks.files (module), 35
nornir.plugins.tasks.networking (module), 36
nornir.plugins.tasks.text (module), 39
NornirExecutionError, 53
NornirSubTaskError, 53
nos (nornir.core.inventory.Host attribute), 50
NSOTInventory (class in nornir.plugins.inventory.nsot), 42

O

os (nornir.core.inventory.Host attribute), 50

P

paramiko_connection() (in module nornir.plugins.tasks.connections), 35
params (Task attribute), 51
parse() (in module nornir.plugins.inventory.ansible), 42
parse() (nornir.plugins.inventory.ansible.AnsibleParser method), 42
parse_group() (nornir.plugins.inventory.ansible.AnsibleParser method), 42
parse_hosts() (nornir.plugins.inventory.ansible.AnsibleParser method), 42
password (nornir.core.inventory.Host attribute), 50
print_result() (in module nornir.plugins.functions.text), 39
print_title() (in module nornir.plugins.functions.text), 40

R

raise_on_error() (nornir.core.task.AggregatedResult method), 52
raise_on_error() (nornir.core.task.MultiResult method), 53
read_vars_file() (nornir.plugins.inventory.ansible.AnsibleParser method), 42
recover_host() (nornir.core.Data method), 45
remote_command() (in module nornir.plugins.tasks.commands), 34
reset_failed_hosts() (nornir.core.Data method), 45
Result (class in nornir.core.task), 52
result (nornir.core.exceptions.NornirExecutionError attribute), 53
result (nornir.core.exceptions.NornirSubTaskError attribute), 53
result (Result attribute), 52
results (Task.self attribute), 51
run() (nornir.core.Nornir method), 46
run() (nornir.core.task.Task method), 51

S

severity_level (Result attribute), 52
severity_level (Task attribute), 51
sftp() (in module nornir.plugins.tasks.files), 35
SimpleInventory (class in nornir.plugins.inventory.simple), 40
sort_groups() (nornir.plugins.inventory.ansible.AnsibleParser method), 42
ssh_port (nornir.core.inventory.Host attribute), 50

start() (nornir.core.task.Task method), 51
status_code (nornir.core.exceptions.CommandError attribute), 53
stderr (nornir.core.exceptions.CommandError attribute), 53
stdout (nornir.core.exceptions.CommandError attribute), 53
string_to_bool() (nornir.core.configuration.Config method), 47

T

Task (class in nornir.core.task), 51
task (nornir.core.exceptions.NornirSubTaskError attribute), 53
task (Task attribute), 51
tcp_ping() (in module nornir.plugins.tasks.networking), 38
template_file() (in module nornir.plugins.tasks.text), 39
template_string() (in module nornir.plugins.tasks.text), 39
to_dict() (nornir.core.Data method), 45
to_dict() (nornir.core.inventory.Host method), 50
to_dict() (nornir.core.inventory.Inventory method), 48
to_dict() (nornir.core.Nornir method), 47

U

username (nornir.core.inventory.Host attribute), 50

V

values() (nornir.core.inventory.Host method), 50

W

write_file() (in module nornir.plugins.tasks.files), 36

Y

YAMLParser (class in nornir.plugins.inventory.ansible), 42