Building scalable tooling

David Barroso @dbarrosop {github,twitter,linkedin}

whoami

- Principal Engineer at Fastly
    - Dealing with large scale distributed control plane orchestation and management systems

- Creator and maintainer of various opensource libraries
    - `napalm`, `nornir`, `gornir`, `yangify`, `ntc-rosetta`...

A story in two parts

- Motivation and design goals
- Nornir and how it helps meeting those goals

Motivation and design goals

Why do we want automation?

- Reliability
- Consistency
- Maintainability

Speed is not a goal but a consequence

At this point there is litttle argument about our motivation for automation, however, why don't we apply the same principles when writing our automation system?

How can you argue your tooling brings those three properties to your network if you can't say the same about your tooling?

Reliability

Does our software do what we claim it does?

Can we change it without breaking anything?

Forget about {unit, integrations, acceptance} tests

Test the interactions with the system from a user perspective

If there is a bug, make sure you add a test that simulates how the user may trigger it and fix to avoid regressions

If you think it's worth it, add unit tests, but always focus first on interactions from the user perspective

Consistency

Avoid cognitive overhead which can lead to bugs, wasted time and bikeshedding

Adopt frameworks and best practices

Choose a framework and stick by it unless strictly necessary

If you need external services standardized and adopt them across the board (i.e. databases, message buses, etc)

Adopt a coding style (or an opinionated linter) to minimize arguments about style (i.e, black)

The goal is to be able to collaborate on multiple projects without having to pay a very expensive context switch or waste time arguing about tabs vs space or mysql vs postgres

Maintainability

- Readability
- Abstractions
- Developer's tooling

Readability

Code is read more often than is written so optimize for reading

One-liners look clever and might save you some typing but you will eventually have to read it and remember how it worked.

In [1]:

```
# filter odd vlans and capitalize name, take 1
hosts = {
    "hostA": {
        "vlans": {
            "prod": 20,
            "dev": 21,
        }
    },
    "hostB": {
        "vlans": {
            "prod": 20,
            "dev": 21,
        }
    },

}

hosts_capitalized = {n: {"vlans": {v.upper(): i}for v, i in h["vlans"].items() if
i % 2 == 0} for n, h in hosts.items()}
print(hosts_capitalized)
```

```
{'hostA': {'vlans': {'PROD': 20}}, 'hostB': {'vlans': {'PROD': 20}}}
```

In [2]:

```
# filter odd vlans and capitalize name, take 2
hosts = {
    "hostA": {
        "vlans": {
            "prod": 20,
            "dev": 21,
        }
    },
    "hostB": {
        "vlans": {
            "prod": 20,
            "dev": 21,
        }
    },

}

hosts_capitalized = {}
for name, host in hosts.items():
    hosts_capitalized[name] = {"vlans": {}}
    for vlan_name, vlan_id in host["vlans"].items():
        if vlan_id % 2 == 0:
            hosts_capitalized[name]["vlans"] = {vlan_name.upper(): vlan_id}
print(hosts_capitalized)
```

```
{'hostA': {'vlans': {'PROD': 20}}, 'hostB': {'vlans': {'PROD': 20}}}
```

First example has a bug, good luck finding it and fixint it :)

Avoid nested code and complex logic:

In [3]:

```
# filter odd vlans and capitalize name, take 3
hosts = {
    "hostA": {
        "vlans": {
            "prod": 20,
            "dev": 21,
        }
    },
    "hostB": {
        "vlans": {
            "prod": 20,
            "dev": 21,
        }
    },

}

def get_even_vlans_with_name_in_caps(vlans):
    return {vlan_name.upper(): vlan_id
            for vlan_name, vlan_id in vlans.items() if vlan_id % 2 == 0}

hosts_capitalized = {}
for name, host in hosts.items():
    hosts_capitalized[name] = {
        "vlans": get_even_vlans_with_name_in_caps(host["vlans"])
    }
print(hosts_capitalized)
```

```
{'hostA': {'vlans': {'PROD': 20}}, 'hostB': {'vlans': {'PROD': 20}}}
```

In [4]:

```
# filter odd vlans and capitalize name, take 4
hosts = {
    "hostA": {
        "vlans": {
            "prod": 20,
            "dev": 21,
        }
    },
    "hostB": {
        "vlans": {
            "prod": 20,
            "dev": 21,
        }
    },

}

def get_even_vlans_with_name_in_caps(vlans):
    return {vlan_name.upper(): vlan_id
            for vlan_name, vlan_id in vlans.items() if vlan_id % 2 == 0}


hosts_capitalized = {
    hostname: {"vlans": get_even_vlans_with_name_in_caps(host["vlans"])}
    for hostname, host in hosts.items()
}
print(hosts_capitalized)
```

```
{'hostA': {'vlans': {'PROD': 20}}, 'hostB': {'vlans': {'PROD': 20}}}
```

Abstractions

Break down your code into different layers of abstraction

Each abstraction should be concerned about solving the problem presented in its layer

Each abstraction should provide a stable contract so other abstractions can consume it

Example, deploying services:

1. Service abstractions: `deploy_vpn_service`, `deploy_peer`, ...
2. Configuration abstractions: `deploy_vlans`, `deploy_bgp_session`, `deploy_policy`...
3. Device abstraction: `send_config`, `get_state`, ...

Abstractions are good for separating of concerns

With good separation of concerns things can be mocked, tested and debugged independently and should allow you to easily ask questions you may have about your software. For instance:

- Given the request of deploying a service, can my software identify which parts need to be configured and which parameters need to be set?
- Given the right input, is my service generating the correct configuration?
- Given some configuration, is my library able to deploy it correctly to the device?

Developer's tooling

A developer should have tooling to:

1. Help writing code; autocompletion, inline documentation, refactoring, etc...
2. Inspect and explore what the program is doing during it's execution
3. Observe how the system behaves in production

Nornir and how it helps meeting those goals

What's Nornir?

Pluggable multi-threaded framework with inventory management to help operate collections of devices

In [5]:

```
from nornir import InitNornir
from nornir.plugins.tasks.commands import command
from nornir.plugins.functions.text import print_result

nr = InitNornir(config_file="1_intro/config.yaml")
result = nr.run(task=command,
                command="echo Hi!")
print_result(result, vars=["stdout"])
```

```
command****************************************************************
**
* leaf00.bma ** changed : False ************************************************
**
vvvv command ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vv INFO
Hi!

^^^^ END command ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^
* leaf01.bma ** changed : False ************************************************
**
vvvv command ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vv INFO
Hi!

^^^^ END command ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^
* spine00.bma ** changed : False ***********************************************
**
vvvv command ** changed : False vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vv INFO
Hi!

^^^^ END command ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^
* spine01.bma ** changed : False ***********************************************
```

Why Nornir

Because it's written in python and meant to be used with python

- Orders of magnitude faster than YAML-based alternatives
- Integrate natively with other python frameworks like flask, django, click, etc...
- Easier to extend
- Cleaner logic
- Leverage linters, debuggers and loggers and IDEs for python

A well-known cloud and hosting provider is using it to gather state from +10.000 devices in less than 5 minutes

Integrations

- with network devices via netmiko, napalm and netconf
- with inventories like yaml, ansible-inventory, nsot and netbox

Extremely easy to add your own if needed

Reliability

Nornir is python code, which means we can use standard python tools for testing and mocking

```
============================= test session starts ==============================
platform linux -- Python 3.6.9, pytest-5.1.2, py-1.8.0, pluggy-0.13.0 -- /usr/local/bin/python
cachedir: .pytest_cache
rootdir: /nornir, inifile: setup.cfg
plugins: nbval-0.9.2, requests-mock-1.7.0, cov-2.7.1, pylama-7.7.1
collected 198 items

tests/core/test_InitNornir.py::Test::test_InitNornir_defaults PASSED
tests/core/test_InitNornir.py::Test::test_InitNornir_file PASSED
tests/core/test_InitNornir.py::Test::test_InitNornir_programmatically PASSED
tests/core/test_InitNornir.py::Test::test_InitNornir_override_partial_section PASSED
tests/core/test_InitNornir.py::Test::test_InitNornir_combined PASSED
tests/core/test_InitNornir.py::Test::test_InitNornir_different_inventory_by_string PASSED
tests/core/test_InitNornir.py::Test::test_InitNornir_different_inventory_imported PASSED
tests/core/test_InitNornir.py::Test::test_InitNornir_different_transform_function_by_string PASSED
tests/core/test_InitNornir.py::Test::test_InitNornir_different_transform_function_imported PASSED
tests/core/test_InitNornir.py::Test::test_InitNornir_different_transform_function_by_string_with_options PASSED
tests/core/test_InitNornir.py::Test::test_InitNornir_different_transform_function_by_string_with_bad_options PASSED
```

A simple task:

```
def configure_description(task, interface, to_device, to_interface):
    return f"interface {interface}\ndescription conntected to {to_device}:{to_int
erface}"
```

Testing the task:

```
class Test:
    def test_configure_interface_description(self, nornir):
        assert configure_description(None, "ten0/1/0", "rtr00", "ten0/1/1") == \
               "interface ten0/1/0\ndescription connected to rtr00:ten0/1/0"
```

Tests allow you to experiment and iterate with confidence

Consistency

Nornir has a system of plugins that allows you to:

1. Perform operations (aka tasks)
2. Read inventory data from various sources
3. Process results and signals from tasks

You can run arbitrary python code where needed but by following the plugin patters it becomes easier to know what to expect

Integrates natively with any python framework:

- django, flask, tornado
- click, argparse
- logging ...

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

nr = InitNornir(config_file="/monit/config.yaml", num_workers=100)

@app.route("/bgp_neighors")
def metrics():
    results = nr.run(
        task=napalm_get,
        getters=["bgp_neighbors"],

    )
    return Response(results.results["bgp_neighbors"])
```

Maintainability

- Readability
- Abstractions
- Developer's tooling

Readability

Being python you can leverage the same techniques as with regular python code to improve readability; functions, classes, decorators, libraries, etc...

Abstractions

- Tasks is the minimum unit of work
- Tasks can embed other tasks

```python
def configure_complex_service(task, parameters):
    bgp_conf = task.run(
        task=template,
        template="templates/{task.host.platform}/bgp.j2",
        bgp_parameters=parameters["bgp"])
    vlan_conf = task.run(
        task=template,
        template="templates/{task.host.platform}/vlan.j2",
        bgp_parameters=parameters["vlan"])
    return bgp_conf.result + vlan_conf.result

def deploy_some_complex_service(task, parameters):
    conf = task.run(
        task=configure_complex_service,
        parameters=paramters)
    task.run(
        task=napalm_configure,
        config=conf.result)


nr.run(
    task=deploy_some_complex_service,
    parameters=paramters,
)
```

Separation of concerns and abstractions:

- `deploy_some_complex_service` is our service-abstraction
- `configure_complex_service` is our configuration abstractions and is reponsible solely of making sure correct configuration is generated
- `napalm`, `netmiko`, `ncclient` tasks represent our device abstractions and are responsible of interacting with our network equipment

Each abstraction is independent and can be tested independently with standard python mocking and testing libraries.

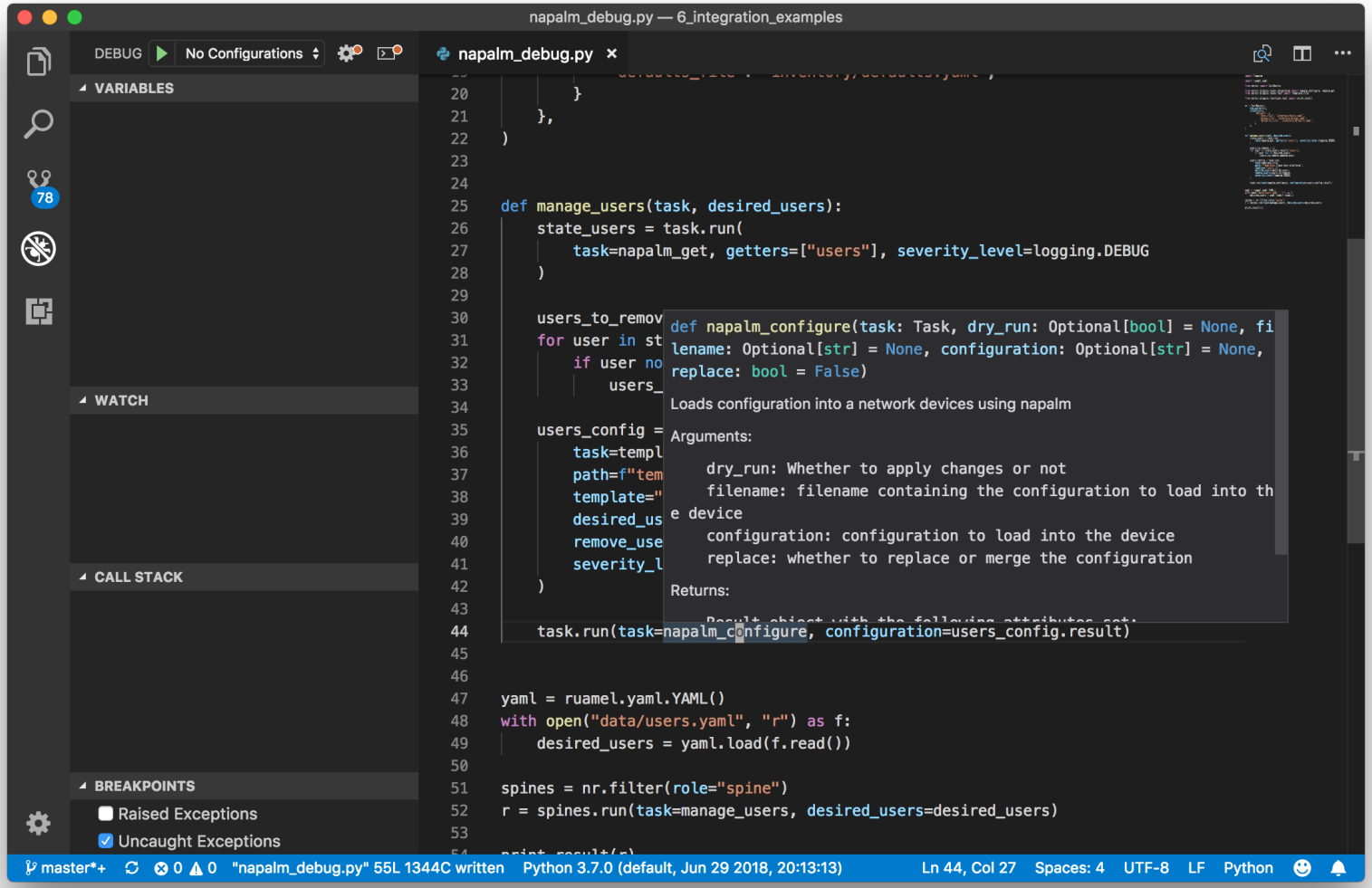
Developer's tooling

Logging

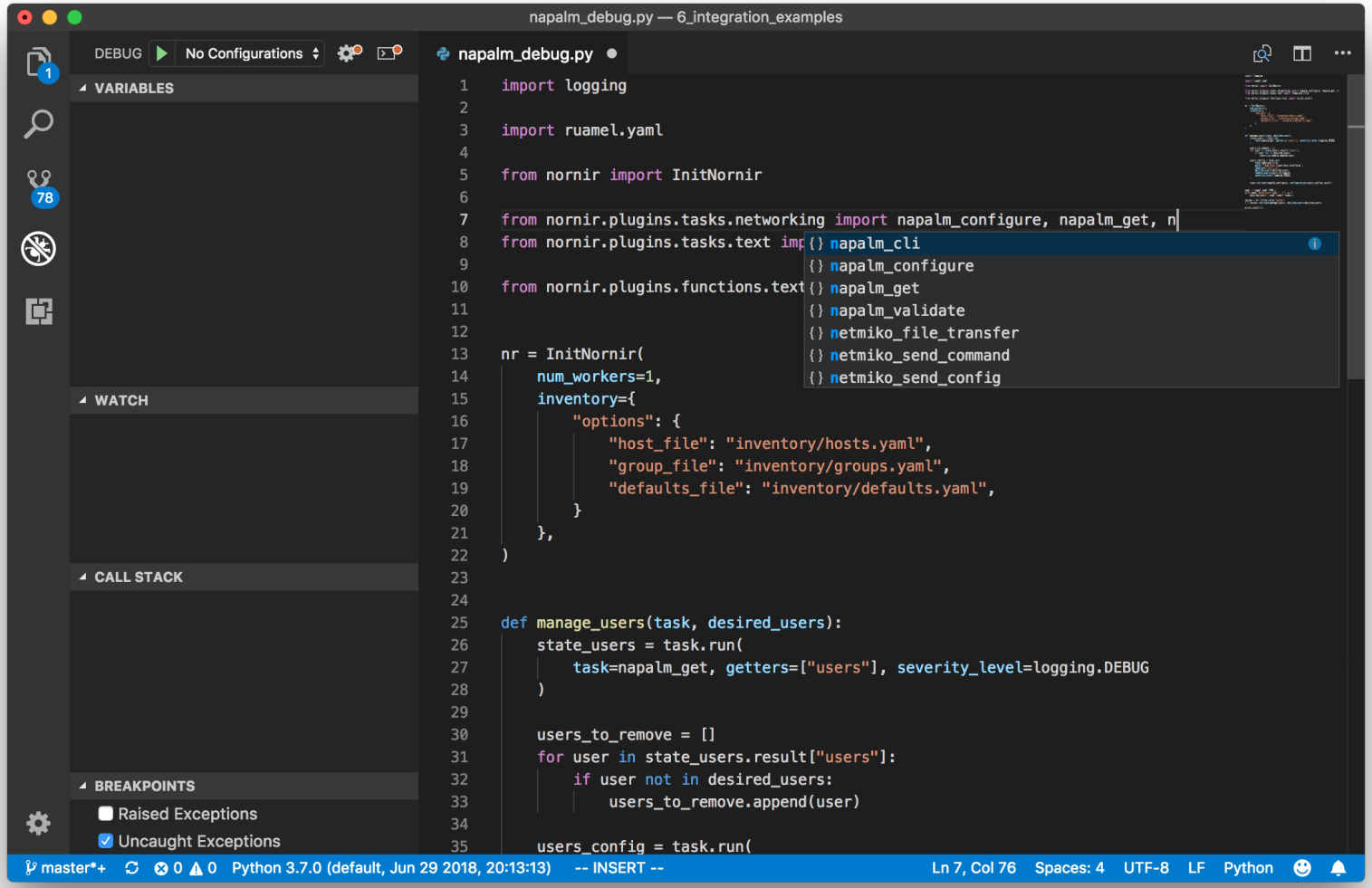
```
import logging

def my_task(task):
    logging.debug(f"doing something on {task.host}")
```

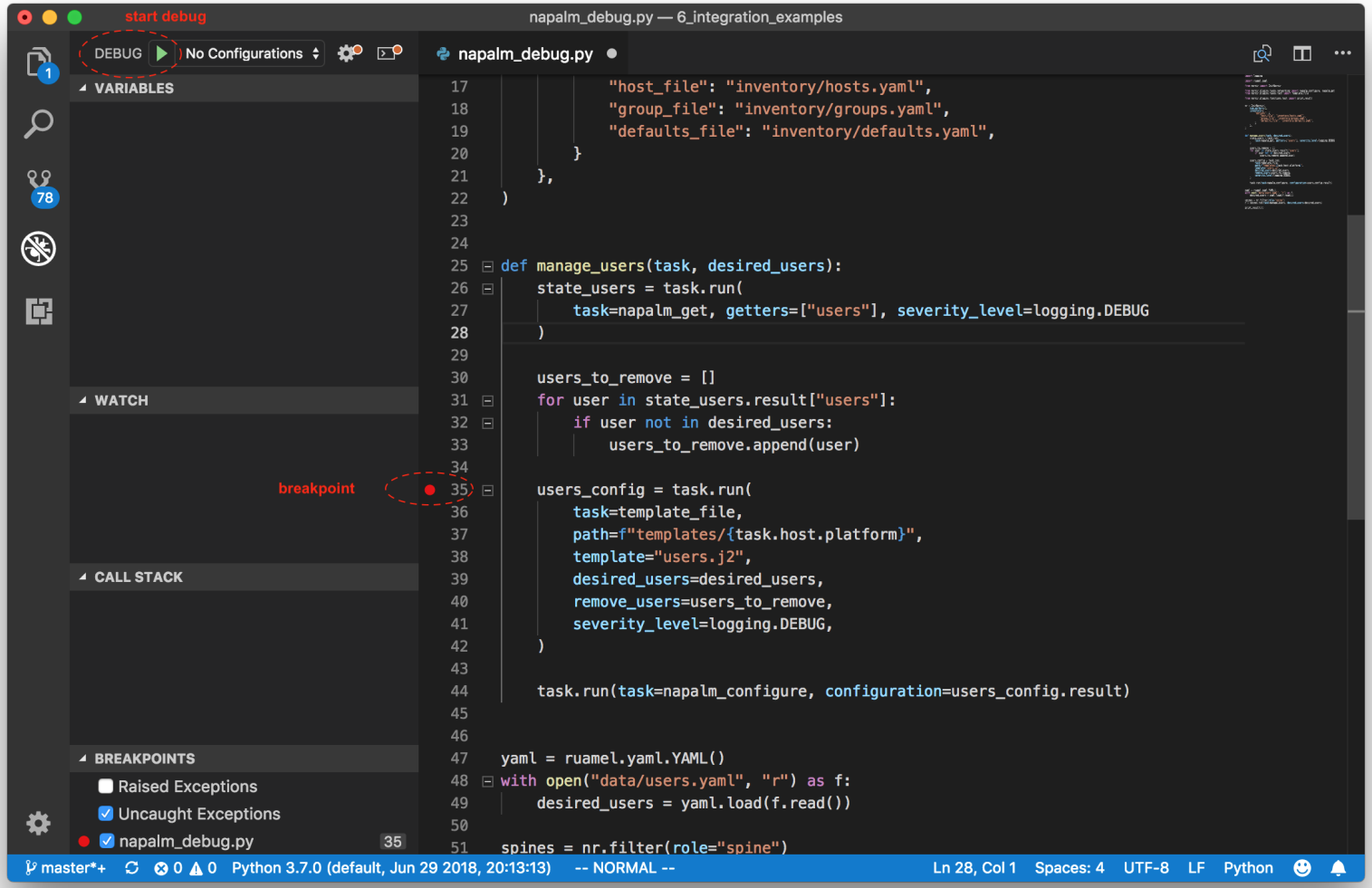
Inline documentation



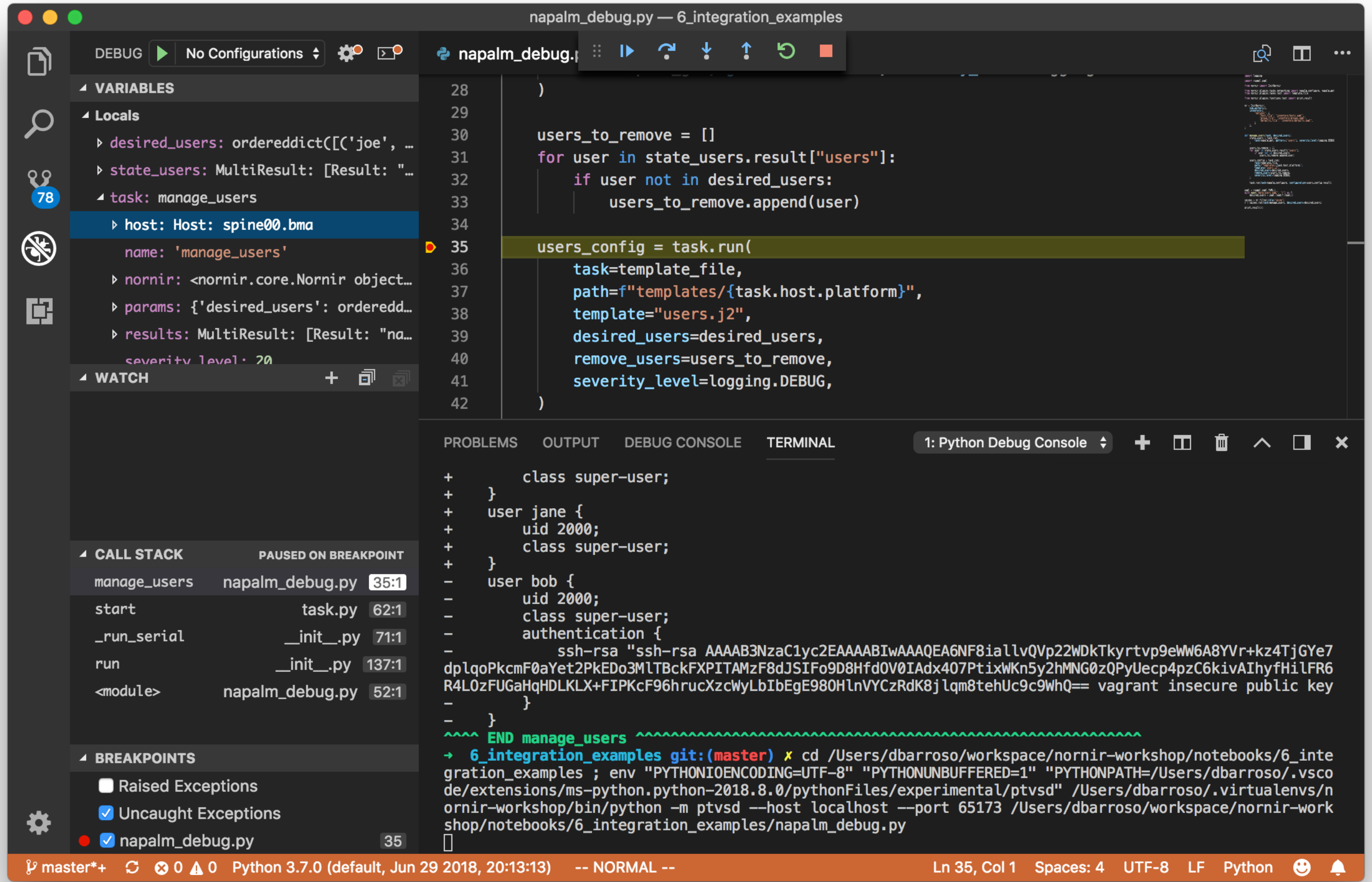
Autocompletion



Debugger

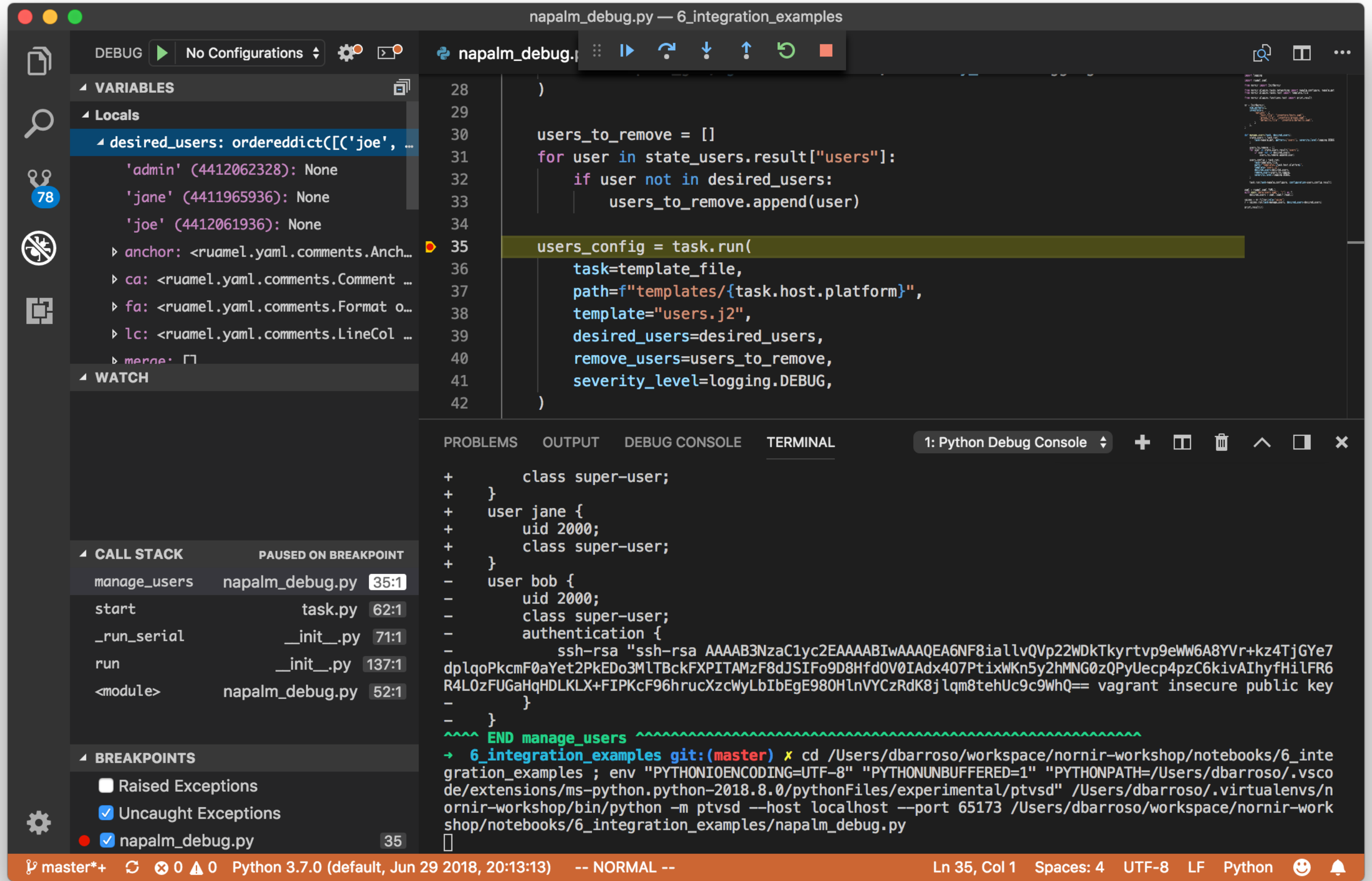

start debug
napalm_debug.py — 6_integration_examples
DEBUG
No Configurations
napalm_debug.py
VARIABLES
WATCH
breakpoint
CALL STACK
BREAKPOINTS
Raised Exceptions
Uncaught Exceptions
napalm_debug.py 35
"host_file": "inventory/hosts.yaml",
"group_file": "inventory/groups.yaml",
"defaults_file": "inventory/defaults.yaml",
}
},
)
def manage_users(task, desired_users):
state_users = task.run(
task=napalm_get, getters=["users"], severity_level=logging.DEBUG
)
users_to_remove = []
for user in state_users.result["users"]:
if user not in desired_users:
users_to_remove.append(user)
users_config = task.run(
task=template_file,
path=f"templates/{task.host.platform}",
template="users.j2",
desired_users=desired_users,
remove_users=users_to_remove,
severity_level=logging.DEBUG,
)
task.run(task=napalm_configure, configuration=users_config.result)
yaml = ruamel.yaml.YAML()
with open("data/users.yaml", "r") as f:
desired_users = yaml.load(f.read())
spines = nr.filter(role="spine")
master*+
0 0
Python 3.7.0 (default, Jun 29 2018, 20:13:13)
-- NORMAL --
Ln 28, Col 1
Spaces: 4
UTF-8
LF
Python

Debugger

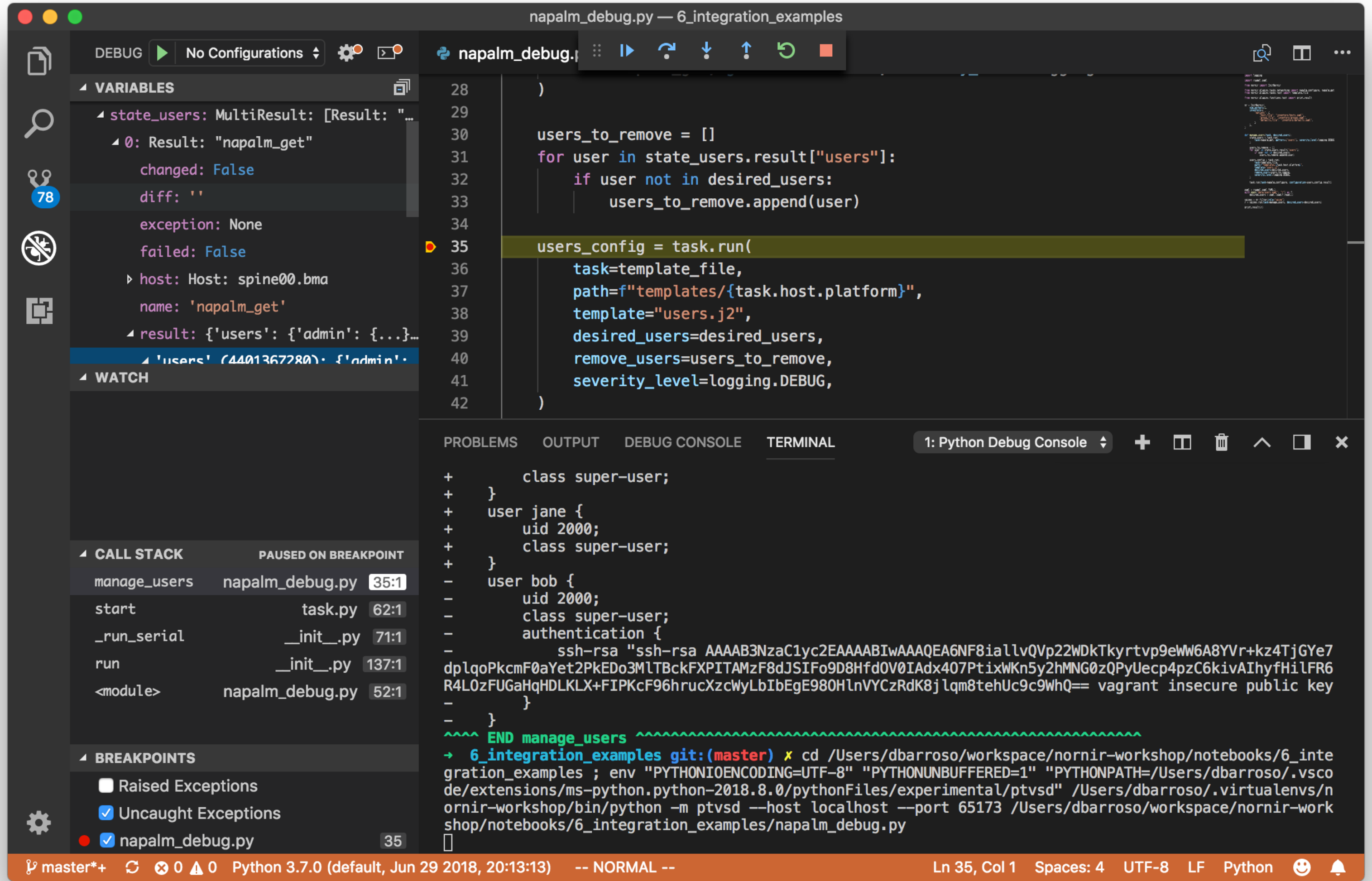


napalm_debug.py — 6_integration_examples
DEBUG
No Configurations
napalm_debug.py
VARIABLES
Locals
desired_users: ordereddict([('joe', …
state_users: MultiResult: [Result: "…
task: manage_users
host: Host: spine00.bma
name: 'manage_users'
nornir: <nornir.core.Nornir object…
params: {'desired_users': ordereddd…
results: MultiResult: [Result: "na…
WATCH
CALL STACK
PAUSED ON BREAKPOINT
manage_users napalm_debug.py 35:1
start task.py 62:1
_run_serial __init__.py 71:1
run __init__.py 137:1
<module> napalm_debug.py 52:1
BREAKPOINTS
Raised Exceptions
Uncaught Exceptions
napalm_debug.py 35
users_to_remove = []
for user in state_users.result["users"]:
if user not in desired_users:
users_to_remove.append(user)
users_config = task.run(
task=template_file,
path=f"templates/{task.host.platform}",
template="users.j2",
desired_users=desired_users,
remove_users=users_to_remove,
severity_level=logging.DEBUG,
)
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: Python Debug Console
class super-user;
user jane {
uid 2000;
class super-user;
user bob {
uid 2000;
class super-user;
authentication {
^^^^ END manage_users
master*+
Python 3.7.0 (default, Jun 29 2018, 20:13:13)
-- NORMAL --
Ln 35, Col 1
Spaces: 4
UTF-8
LF
Python

Debugger

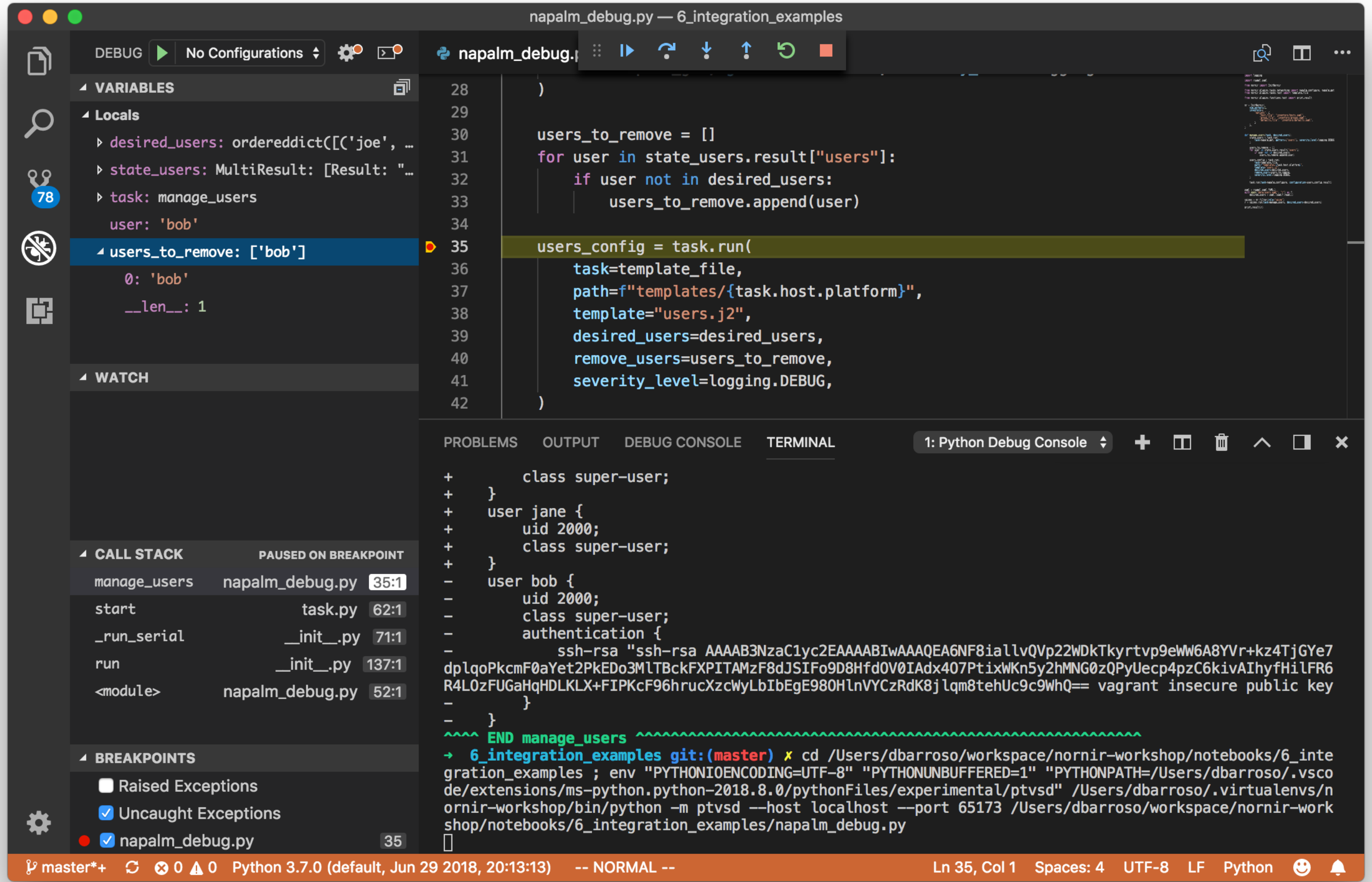

napalm_debug.py — 6_integration_examples
DEBUG
No Configurations
napalm_debug.py
VARIABLES
Locals
desired_users: ordereddict([('joe', …
'admin' (4412062328): None
'jane' (4411965936): None
'joe' (4412061936): None
anchor: <ruamel.yaml.comments.Anch…
ca: <ruamel.yaml.comments.Comment …
fa: <ruamel.yaml.comments.Format o…
lc: <ruamel.yaml.comments.LineCol …
WATCH
CALL STACK
PAUSED ON BREAKPOINT
manage_users napalm_debug.py 35:1
start task.py 62:1
_run_serial __init__.py 71:1
run __init__.py 137:1
<module> napalm_debug.py 52:1
BREAKPOINTS
Raised Exceptions
Uncaught Exceptions
napalm_debug.py 35
users_to_remove = []
for user in state_users.result["users"]:
if user not in desired_users:
users_to_remove.append(user)
users_config = task.run(
task=template_file,
path=f"templates/{task.host.platform}",
template="users.j2",
desired_users=desired_users,
remove_users=users_to_remove,
severity_level=logging.DEBUG,
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: Python Debug Console
class super-user;
user jane {
uid 2000;
class super-user;
user bob {
uid 2000;
class super-user;
authentication {
^^^^ END manage_users
master*+
Python 3.7.0 (default, Jun 29 2018, 20:13:13)
-- NORMAL --
Ln 35, Col 1
Spaces: 4
UTF-8
LF
Python

Debugger

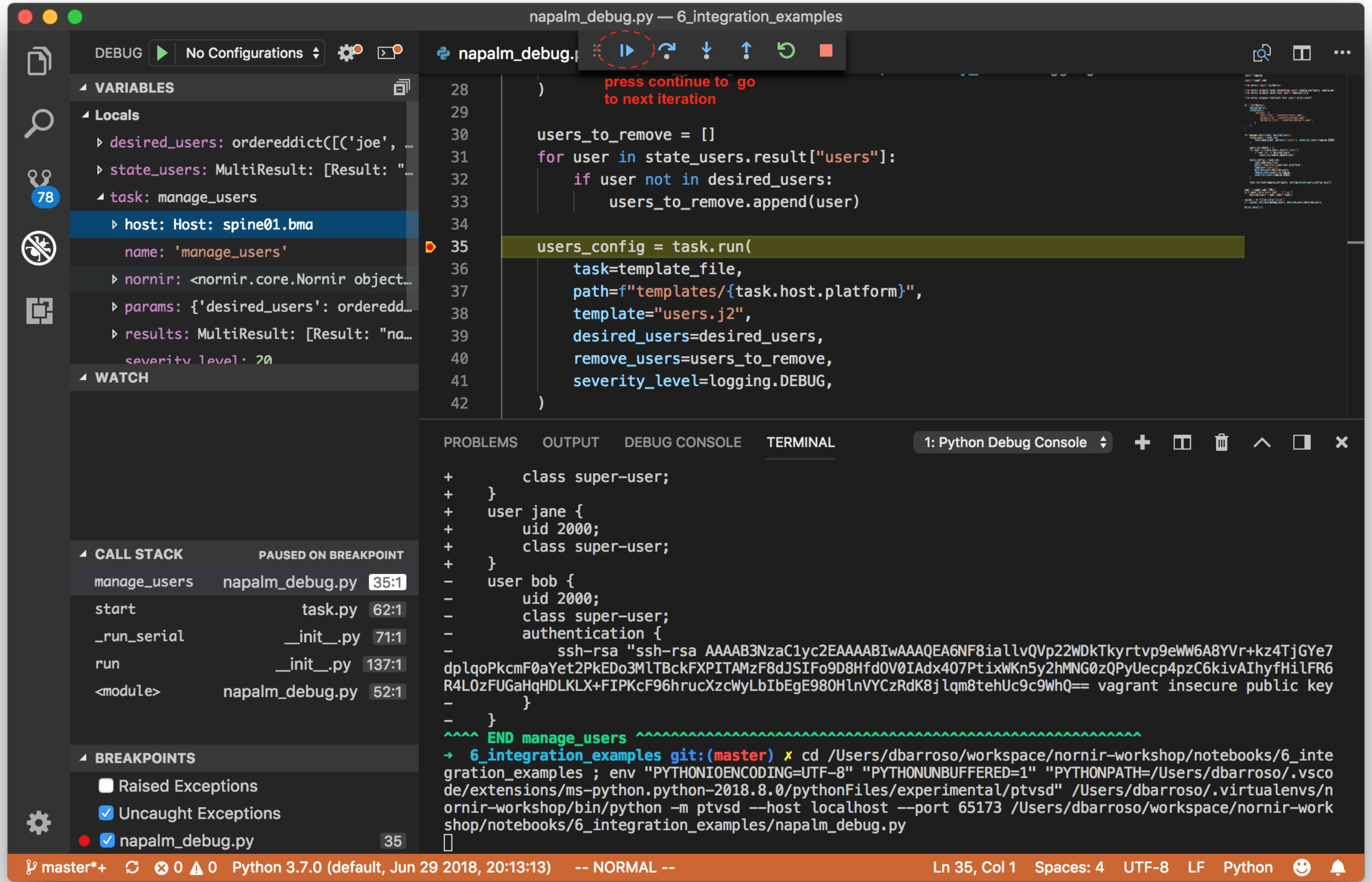


napalm_debug.py — 6_integration_examples
DEBUG
No Configurations
VARIABLES
state_users: MultiResult: [Result: "…
0: Result: "napalm_get"
changed: False
diff: ''
exception: None
failed: False
host: Host: spine00.bma
name: 'napalm_get'
result: {'users': {'admin': {...}…
WATCH
CALL STACK
PAUSED ON BREAKPOINT
manage_users napalm_debug.py 35:1
start task.py 62:1
_run_serial __init__.py 71:1
run __init__.py 137:1
<module> napalm_debug.py 52:1
BREAKPOINTS
Raised Exceptions
Uncaught Exceptions
napalm_debug.py 35
)
users_to_remove = []
for user in state_users.result["users"]:
if user not in desired_users:
users_to_remove.append(user)
users_config = task.run(
task=template_file,
path=f"templates/{task.host.platform}",
template="users.j2",
desired_users=desired_users,
remove_users=users_to_remove,
severity_level=logging.DEBUG,
)
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: Python Debug Console
+ class super-user;
+ }
+ user jane {
+ uid 2000;
+ class super-user;
+ }
- user bob {
- uid 2000;
- class super-user;
- authentication {
- ssh-rsa "ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA6NF8iallvQVp22WDkTkyrtvp9eWW6A8YVr+kz4TjGYe7dplqoPkcmF0aYet2PkEDo3MlTBckFXPITAMzF8dJSIFo9D8HfdOV0IAdx4O7PtixWKn5y2hMNG0zQPyUecp4pzC6kivAIhyfHilFR6R4LOzFUGaHqHDLKLX+FIPKcF96hrucXzcWyLbIbEgE98OHlnVYCzRdK8jlqm8tehUc9c9WhQ== vagrant insecure public key
- }
- }
^^^^ END manage_users ^^^
→ 6_integration_examples git:(master) ✗ cd /Users/dbarroso/workspace/nornir-workshop/notebooks/6_integration_examples ; env "PYTHONIOENCODING=UTF-8" "PYTHONUNBUFFERED=1" "PYTHONPATH=/Users/dbarroso/.vscode/extensions/ms-python.python-2018.8.0/pythonFiles/experimental/ptvsd" /Users/dbarroso/.virtualenvs/nornir-workshop/bin/python -m ptvsd --host localhost --port 65173 /Users/dbarroso/workspace/nornir-workshop/notebooks/6_integration_examples/napalm_debug.py
master*+
Python 3.7.0 (default, Jun 29 2018, 20:13:13)
-- NORMAL --
Ln 35, Col 1
Spaces: 4
UTF-8
LF
Python

Debugger


napalm_debug.py — 6_integration_examples
DEBUG
No Configurations
napalm_debug.py
VARIABLES
Locals
desired_users: ordereddict([('joe', …
state_users: MultiResult: [Result: "…
task: manage_users
user: 'bob'
users_to_remove: ['bob']
0: 'bob'
__len__: 1
WATCH
CALL STACK
PAUSED ON BREAKPOINT
manage_users napalm_debug.py 35:1
start task.py 62:1
_run_serial __init__.py 71:1
run __init__.py 137:1
<module> napalm_debug.py 52:1
BREAKPOINTS
Raised Exceptions
Uncaught Exceptions
napalm_debug.py 35
28)
29
30 users_to_remove = []
31 for user in state_users.result["users"]:
32 if user not in desired_users:
33 users_to_remove.append(user)
34
35 users_config = task.run(
36 task=template_file,
37 path=f"templates/{task.host.platform}",
38 template="users.j2",
39 desired_users=desired_users,
40 remove_users=users_to_remove,
41 severity_level=logging.DEBUG,
42)
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: Python Debug Console
+ class super-user;
+ }
+ user jane {
+ uid 2000;
+ class super-user;
+ }
- user bob {
- uid 2000;
- class super-user;
- authentication {
- }
- }
^^^^ END manage_users ^^^
→ 6_integration_examples git:(master) ✗ cd /Users/dbarroso/workspace/nornir-workshop/notebooks/6_integration_examples ; env "PYTHONIOENCODING=UTF-8" "PYTHONUNBUFFERED=1" "PYTHONPATH=/Users/dbarroso/.vscode/extensions/ms-python.python-2018.8.0/pythonFiles/experimental/ptvsd" /Users/dbarroso/.virtualenvs/nornir-workshop/bin/python -m ptvsd --host localhost --port 65173 /Users/dbarroso/workspace/nornir-workshop/notebooks/6_integration_examples/napalm_debug.py
master*+
0 0
Python 3.7.0 (default, Jun 29 2018, 20:13:13)
-- NORMAL --
Ln 35, Col 1
Spaces: 4
UTF-8
LF
Python

Debugger


napalm_debug.py — 6_integration_examples
DEBUG No Configurations
VARIABLES
Locals
desired_users: ordereddict([('joe', …
state_users: MultiResult: [Result: "…
task: manage_users
host: Host: spine01.bma
name: 'manage_users'
nornir: <nornir.core.Nornir object…
params: {'desired_users': orderedd…
results: MultiResult: [Result: "na…
WATCH
CALL STACK PAUSED ON BREAKPOINT
manage_users napalm_debug.py 35:1
start task.py 62:1
_run_serial __init__.py 71:1
run __init__.py 137:1
<module> napalm_debug.py 52:1
BREAKPOINTS
Raised Exceptions
Uncaught Exceptions
napalm_debug.py 35
press continue to go to next iteration
)
users_to_remove = []
for user in state_users.result["users"]:
if user not in desired_users:
users_to_remove.append(user)
users_config = task.run(
task=template_file,
path=f"templates/{task.host.platform}",
template="users.j2",
desired_users=desired_users,
remove_users=users_to_remove,
severity_level=logging.DEBUG,
)
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: Python Debug Console
+ class super-user;
+ }
+ user jane {
+ uid 2000;
+ class super-user;
+ }
- user bob {
- uid 2000;
- class super-user;
- authentication {
vagrant insecure public key
^^^^ END manage_users ^^^^
6_integration_examples git:(master) ✗ cd /Users/dbarroso/workspace/nornir-workshop/notebooks/6_integration_examples ; env "PYTHONIOENCODING=UTF-8" "PYTHONUNBUFFERED=1" "PYTHONPATH=/Users/dbarroso/.vscode/extensions/ms-python.python-2018.8.0/pythonFiles/experimental/ptvsd" /Users/dbarroso/.virtualenvs/nornir-workshop/bin/python -m ptvsd --host localhost --port 65173 /Users/dbarroso/workspace/nornir-workshop/notebooks/6_integration_examples/napalm_debug.py
master*+ 0 0 Python 3.7.0 (default, Jun 29 2018, 20:13:13) -- NORMAL -- Ln 35, Col 1 Spaces: 4 UTF-8 LF Python

Summary

- Look for reliability, repeatibility and maintainability both in your network and your automation tooling
- If you can't guarantee a property anywhere in your stack you can't guarantee it in the system
- It's not enough to learn to code, you need to learn the tooling and best practices

FIN