NANOG

Is the Network Ready for Use?

An automated testing approach using pytest

Jeremy Schulman
Senior Network Automation Engineer @MLB
@nwkautomaniac
2019 - October - NANOG 77

About Me

- Jeremy Schulman
  - Software engineer, sales & systems engineer, ...
  - 20 yrs @ Vendors
  - Now @ Major League Baseball
- @nwkautomaniac - Est. 2012
- 100% focused on network automation
- Open-source & community contributor

About this Tutorial

- Motivation for “Network Ready for Use”
- General testing concepts
- pytest to automate network tests
- Live coding tutorials
- Q & A

About You

- You have a working knowledge of Python
- You could be a Network Engineer starting with automation
- You could be a Developer / DevOps working with a Networking team

About pytest

http://pytest.org

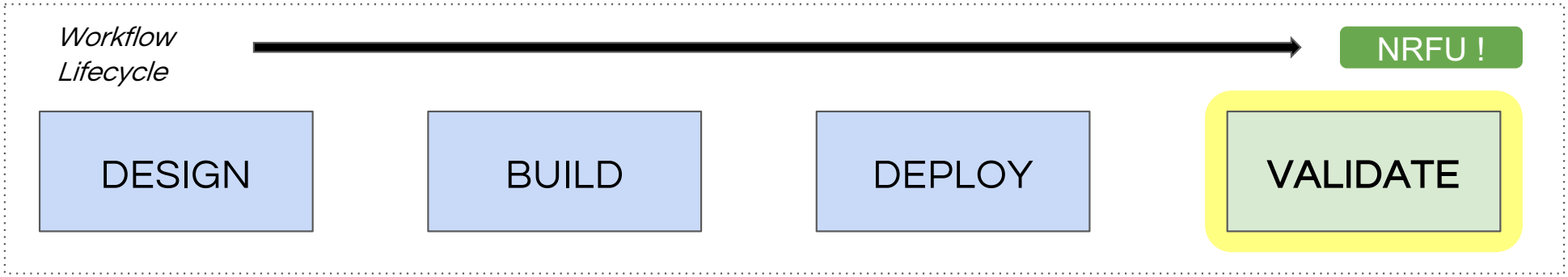
pytest is an open-source software testing framework used by software developers to test their code.

- We are going to treat the network as our “code”
- We are going to use pytest to validate operational states of the network

Motivation

Network Ready For Use

Does the actual operating states of the network match the expected outcomes based on the design?



Motivating Use-Case

New campus network

- Consist of hundreds of network devices
- Multiple floors, multiple rooms per floor
- 3rd-party installing equipment
- 3rd-party (different) installing structured cabling
- Staggered installation schedule over multiple weeks

Use automated testing to run audits during the initial deployment and direct corrective actions

Network Ready For Use - Phase 1

NRFU testing to validate physical installation

- Devices on-boarded into management system
- Optics installed properly
- Cabling installed properly

"Done" means network is ready for use for NETENG to deploy network services

Network Ready for Use - Phase 2

NRFU audit to validate network services are operating correctly

- LACP, MLAG, BGP, VXLAN, …

"Done" means the network is ready for use for other teams to deploy their equipment

- VoIP, IPTV, InfoSec, Building Management, etc.

Phase-1

The Installation & Cabling Plan

We provide spreadsheets to the installation teams, for example:

1	device-A	model-A	location-A	interface-A	optic-type-A	device-Z	model-Z	location-Z	interface-Z	optic-type-Z
112	lx010701	DCS-720XP-48ZC2	MDF-A R01.C07	E51	SFP-25G-SR	lf010701	DCS-7050X3-48YC12	MDF-A R01.C07	E47	SFP-25G-SR
113	lx010701	DCS-720XP-48ZC2	MDF-A R01.C07	E52	SFP-25G-SR	lf010702	DCS-7050X3-48YC12	MDF-A R01.C07	E47	SFP-25G-SR
114	lx010702	DCS-720XP-48ZC2	MDF-A R01.C07	E51	SFP-25G-SR	lf010701	DCS-7050X3-48YC12	MDF-A R01.C07	E48	SFP-25G-SR
115	lx010702	DCS-720XP-48ZC2	MDF-A R01.C07	E52	SFP-25G-SR	lf010702	DCS-7050X3-48YC12	MDF-A R01.C07	E48	SFP-25G-SR
116	lx020101	DCS-720XP-48ZC2	MDF-A R02.C01	E51	SFP-25G-SR	lf020701	DCS-7050X3-48YC12	MDF-A R02.C07	E35	SFP-25G-SR
117	lx020101	DCS-720XP-48ZC2	MDF-A R02.C01	E52	SFP-25G-SR	lf020702	DCS-7050X3-48YC12	MDF-A R02.C07	E35	SFP-25G-SR
118	lx020102	DCS-720XP-48ZC2	MDF-A R02.C01	E51	SFP-25G-SR	lf020701	DCS-7050X3-48YC12	MDF-A R02.C07	E36	SFP-25G-SR
119	lx020102	DCS-720XP-48ZC2	MDF-A R02.C01	E52	SFP-25G-SR	lf020702	DCS-7050X3-48YC12	MDF-A R02.C07	E36	SFP-25G-SR
120	lx020201	DCS-720XP-48ZC2	MDF-A R02.C02	E51	SFP-25G-SR	lf020701	DCS-7050X3-48YC12	MDF-A R02.C07	E37	SFP-25G-SR
121	lx020201	DCS-720XP-48ZC2	MDF-A R02.C02	E52	SFP-25G-SR	lf020702	DCS-7050X3-48YC12	MDF-A R02.C07	E37	SFP-25G-SR
122	lx020202	DCS-720XP-48ZC2	MDF-A R02.C02	E51	SFP-25G-SR	lf020701	DCS-7050X3-48YC12	MDF-A R02.C07	E38	SFP-25G-SR
123	lx020202	DCS-720XP-48ZC2	MDF-A R02.C02	E52	SFP-25G-SR	lf020702	DCS-7050X3-48YC12	MDF-A R02.C07	E38	SFP-25G-SR
124	lx020301	DCS-720XP-48ZC2	MDF-A R02.C03	E51	SFP-25G-SR	lf020701	DCS-7050X3-48YC12	MDF-A R02.C07	E39	SFP-25G-SR
125	lx020301	DCS-720XP-48ZC2	MDF-A R02.C03	E52	SFP-25G-SR	lf020702	DCS-7050X3-48YC12	MDF-A R02.C07	E39	SFP-25G-SR
126	lx020302	DCS-720XP-48ZC2	MDF-A R02.C03	E51	SFP-25G-SR	lf020701	DCS-7050X3-48YC12	MDF-A R02.C07	E40	SFP-25G-SR
127	lx020302	DCS-720XP-48ZC2	MDF-A R02.C03	E52	SFP-25G-SR	lf020702	DCS-7050X3-48YC12	MDF-A R02.C07	E40	SFP-25G-SR
128	lx020401	DCS-720XP-48ZC2	MDF-A R02.C04	E51	SFP-25G-SR	lf020701	DCS-7050X3-48YC12	MDF-A R02.C07	E41	SFP-25G-SR

Phase-1

Audit Criteria

Transceivers

- For each interface is the correct transceiver installed?
- Are transceivers only present where expected?

Interface Status

- For each interface is the operational state as expected?
- Are unused interfaces placed in a *shutdown* state?

Cabling

- Is LLDP reporting the remote system/interface as expected?
- Are any unexpected connections present?

Phase-1

NRFU Audit Reporting

We produce an audit report so we can pinpoint corrective actions for the installation team, for example:

	A	B	C	D	E
1	**date**	**level**	**device**	**interface**	**action**
42	2019-Aug-31	TO-DO	sp050311.nyc1	Ethernet17/1	patch cable to lf8e1b01.nyc1:Ethernet56/1
43	2019-Aug-31	TO-DO	sp050311.nyc1	Ethernet18/1	remote device in ZTP state, check lf8e1b02.nyc1 M1 cabling to management switch
44	2019-Aug-31	TO-DO	sp050311.nyc1	Ethernet19/1	remote device in ZTP state, check lf9w1b01.nyc1 M1 cabling to management switch
45	2019-Aug-31	TO-DO	sp050311.nyc1	Ethernet20/1	remote device in ZTP state, check lf9w1b02.nyc1 M1 cabling to management switch
46	2019-Aug-31	TO-DO	sp050311.nyc1	Ethernet21/1	remote device in ZTP state, check lf9e1a01.nyc1 M1 cabling to management switch
47	2019-Aug-31	TO-DO	sp050311.nyc1	Ethernet22/1	patch cable to lf9e1b02.nyc1:Ethernet56/1
48	2019-Aug-31	ERROR	sp050311.nyc1	Ethernet28/1	wrong patch tg050502.nyc1:Ethernet24/1 -- re-patch cable to tr050502.nyc1:Ethernet24/1
49	2019-Aug-31	TO-DO	sp050311.nyc1	Ethernet29/1	patch cable to tr000103.nyc1:Ethernet24/1
50					
51	2019-Aug-31	WARNING	sp050502.nyc1	Ethernet10/1	remove unexpected optic QSFP-100G-CWDM4
52	2019-Aug-31	WARNING	sp050502.nyc1	Ethernet11/1	remove unexpected optic QSFP-100G-CWDM4
53	2019-Aug-31	WARNING	sp050502.nyc1	Ethernet23/1	remove unexpected optic AOC-Q-Q-100G-5M
54	2019-Aug-31	TO-DO	sp050502.nyc1	Ethernet1/1	patch cable to lf020701.nyc1:Ethernet57/1
55	2019-Aug-31	TO-DO	sp050502.nyc1	Ethernet2/1	patch cable to lf020702.nyc1:Ethernet57/1

Phase-1

Automated Approach Required

- Total devices > 300
- Building with multiple floors / rooms
- Mixture of different hardware devices
- Mixture of transceiver types; SMF, MMF, and CAT6 cabling

Audit Scope and Scale

Based on the campus design, almost 23K test-cases covered by NRFU:

Test Case	Number of Tests
Transceiver check	4,248
Interface status check	16,688
Cabling check	2,046
Total	**22,982**

NRFU Dashboard Report

Not Done!

- 11 devices not tested
- Many failures (275)

			ERRORS	WARNINGS	TO-DO	OFFLINE
MDF-A			26	18	109	6
MDF-B			5	8	87	0
IDF-5E			0	0	4	1
IDF-5W			2	0	10	2
IDF-6E			0	0	6	0
IDF-6W			9	0	2	0
IDF-7E			0	0	4	1
IDF-7W			0	0	1	0
IDF-8E			3	0	8	0
IDF-8W			1	0	3	0
IDF-9E			2	1	6	0
IDF-9W			1	6	2	1
IDF-C1			5	0	4	0
Grand Totals			33	26	216	11

Screenshots using pytest

Showcasing both command-line and WebUI reports

http://pytest.org

pytest run for a given device

PASS

```
test_00_optic_inventory.py::test_optic_inventory[Ethernet50/1:QSFP28-LR4-100G] PASSED                                [  3%]
test_00_optic_inventory.py::test_optic_inventory[Ethernet49/1:QSFP28-LR4-100G] PASSED                                [  7%]
test_00_optic_inventory.py::test_optic_inventory[Ethernet12:none] PASSED                                             [ 11%]
test_01_interface_status.py::test_interface_status[Ethernet1:up] PASSED                                              [ 15%]
test_01_interface_status.py::test_interface_status[Ethernet15:up] PASSED                                             [ 19%]
test_01_interface_status.py::test_interface_status[Ethernet2:up] PASSED                                              [ 23%]
test_01_interface_status.py::test_interface_status[Ethernet29:down] PASSED                                           [ 26%]
test_01_interface_status.py::test_interface_status[Ethernet30:down] PASSED                                           [ 30%]
test_01_interface_status.py::test_interface_status[Ethernet31:down] PASSED                                           [ 34%]
test_01_interface_status.py::test_interface_status[Ethernet32:down] PASSED                                           [ 38%]
test_01_interface_status.py::test_interface_status[Ethernet33/1:down] PASSED                                         [ 42%]
test_01_interface_status.py::test_interface_status[Ethernet34/1:down] PASSED                                         [ 46%]
test_01_interface_status.py::test_interface_status[Ethernet4:up] PASSED                                              [ 50%]
test_01_interface_status.py::test_interface_status[Ethernet6:down] PASSED                                            [ 53%]
test_01_interface_status.py::test_interface_status[Ethernet7:down] PASSED                                            [ 57%]
test_01_interface_status.py::test_interface_status[Ethernet8:down] PASSED                                            [ 61%]
test_01_interface_status.py::test_interface_status[Ethernet9:down] PASSED                                            [ 65%]
test_01_interface_status.py::test_interface_status[Loopback0:up] PASSED                                              [ 69%]
test_01_interface_status.py::test_interface_status[Management1:up] PASSED                                            [ 73%]
test_01_interface_status.py::test_interface_status[Port-Channel10:up] PASSED                                         [ 76%]
test_01_interface_status.py::test_interface_status[Vlan4094:up] PASSED                                               [ 80%]
test_01_interface_status.py::test_interface_status[Port-Channel2000:up] PASSED                                       [ 84%]
test_02_cabling.py::test_cabling[Ethernet49/1<[spine-leaf]->switch-21.bld1:Ethernet3/1] PASSED                       [ 88%]
test_02_cabling.py::test_cabling[Ethernet50/1<[spine-leaf]->switch-22.bld1:Ethernet3/1] PASSED                       [ 92%]
test_02_cabling.py::test_cabling[Ethernet59/1<[peer-pair]->switch-2106.bld1:Ethernet59/1] PASSED                     [ 96%]
test_02_cabling.py::test_cabling[Ethernet60/1<[peer-pair]->switch-2106.bld1:Ethernet60/1] PASSED                     [100%]
```

```
= 26 passed in 0.14s =
```

pytest run for a given device

FAIL

```
test_00_optic_inventory.py::test_optic_inventory[Ethernet50/1:QSFP28-LR4-100G] PASSED                                [  3%]
test_00_optic_inventory.py::test_optic_inventory[Ethernet49/1:QSFP28-LR4-100G] FAILED                                [  7%]
test_00_optic_inventory.py::test_optic_inventory[Ethernet12:none] FAILED                                             [ 11%]
test_01_interface_status.py::test_interface_status[Ethernet1:up] PASSED                                              [ 15%]
test_01_interface_status.py::test_interface_status[Ethernet15:up] PASSED                                             [ 19%]
test_01_interface_status.py::test_interface_status[Ethernet2:up] PASSED                                              [ 23%]
test_01_interface_status.py::test_interface_status[Ethernet29:down] PASSED                                           [ 26%]
test_01_interface_status.py::test_interface_status[Ethernet30:down] PASSED                                           [ 30%]
test_01_interface_status.py::test_interface_status[Ethernet31:down] PASSED                                           [ 34%]
test_01_interface_status.py::test_interface_status[Ethernet32:down] PASSED                                           [ 38%]
test_01_interface_status.py::test_interface_status[Ethernet33/1:down] PASSED                                         [ 42%]
test_01_interface_status.py::test_interface_status[Ethernet34/1:down] PASSED                                         [ 46%]
test_01_interface_status.py::test_interface_status[Ethernet4:up] PASSED                                              [ 50%]
test_01_interface_status.py::test_interface_status[Ethernet6:down] PASSED                                            [ 53%]
test_01_interface_status.py::test_interface_status[Ethernet7:down] PASSED                                            [ 57%]
test_01_interface_status.py::test_interface_status[Ethernet8:down] PASSED                                            [ 61%]
test_01_interface_status.py::test_interface_status[Ethernet9:down] PASSED                                            [ 65%]
test_01_interface_status.py::test_interface_status[Loopback0:up] PASSED                                              [ 69%]
test_01_interface_status.py::test_interface_status[Management1:up] PASSED                                            [ 73%]
test_01_interface_status.py::test_interface_status[Port-Channel10:up] PASSED                                         [ 76%]
test_01_interface_status.py::test_interface_status[Vlan4094:up] PASSED                                               [ 80%]
test_01_interface_status.py::test_interface_status[Port-Channel2000:up] PASSED                                       [ 84%]
test_02_cabling.py::test_cabling[Ethernet49/1<[spine-leaf]->switch-21.bld1:Ethernet3/1] PASSED                       [ 88%]
test_02_cabling.py::test_cabling[Ethernet50/1<[spine-leaf]->switch-22.bld1:Ethernet3/1] PASSED                       [ 92%]
test_02_cabling.py::test_cabling[Ethernet59/1<[peer-pair]->switch-2106.bld1:Ethernet59/1] PASSED                     [ 96%]
test_02_cabling.py::test_cabling[Ethernet60/1<[peer-pair]->switch-2106.bld1:Ethernet60/1] FAILED                     [100%]

= 3 failed, 23 passed in 0.17s ==========================================================================================
```

pytest run for a given device

FAIL

Summary

22 tests ran in 0.16 seconds.

(Un)check the boxes to filter the results.

19 passed, 0 skipped, 3 failed, 0 errors, 0 expected failures, 0 unexpected passes

Results

Show all details / Hide all details

Result	Test
Failed *(hide details)*	tests/test_optic_inventory.py::test_optic_inventory[Ethernet49/1:QSFP28-LR4-100G]
E nrfupytesteos.nrfu_exc.MismatchError: No optic found on interface Ethernet49/1 MISMATCH:EXPECTED data: QSFP28-LR4-100G MISMATCH:ACTUAL data: None	
Failed *(hide details)*	tests/test_optic_inventory.py::test_optic_inventory[Ethernet12:none]
E nrfupytesteos.nrfu_exc.MismatchError: No optic expected, but found on interface Ethernet12 MISMATCH:EXPECTED data: None MISMATCH:ACTUAL data: QSFP28-LR4-100G	
Failed *(hide details)*	tests/test_cabling.py::test_cabling[Ethernet60/1<->switch-2106.bld1:Ethernet60/1]
E nrfupytesteos.nrfu_exc.MismatchError: Wrong remote-device: switch-3106.bld1 MISMATCH:EXPECTED data: switch-2106.bld1:Ethernet60/1 MISMATCH:ACTUAL data: switch-3106.bld1:Ethernet60/1	
Passed *(show details)*	tests/test_optic_inventory.py::test_optic_inventory[Ethernet50/1:QSFP28-LR4-100G]
Passed *(show details)*	tests/test_interface_status.py::test_interface_status[Ethernet1:up]

Demo

Recap

- pytest can be used to execute specific test cases against specific devices
- Test cases are parameterized
- Test cases uniquely named can be filtered based on these names
- Use CLI to show brief PASS / FAIL output
- Use HTML to show more actionable failure information

General Testing Concepts

Testing

Testing is a means to provide a metric of quality and completeness.

We are using testing to answer the question:

Is the network ready for use?

Test Coverage

Test coverage is the metric that measures the amount of testing performed by a set of tests over a collection of Devices Under Test (DUTs)

Done ?

Total number of DUTs that are online and can be measured vs. The total number of DUTs	For each DUT, the total number of test cases PASSING vs. The total number of DUT test cases

Test Coverage

For each Device Under Test (DUT) we need to identify the specific test cases for each of the test functions.

For example, “switch-2105” in our network requires the following number of test cases per test function.

Test Function	Number of Test-cases
test_optic_inventory	30
test_interface_status	36
test_cabling	34
Total	100

Crafting Test-cases

A test case is generally represented as a record of structured data.

Each device, e.g. “switch-2105” will have a JSON file per test function, e.g. “cabling”, containing a list of test cases.

Each unique test case is a specific dictionary in this list.

```
[
    {
        "test-case": "test-cabling",
        "dut": "switch-2105.bld1",
        "params": {
            "interface": "Ethernet49/1",
            "role": "spine-leaf"
        },
        "expected": {
            "remote-hostname": "switch-21.bld1",
            "remote-interface": "Ethernet3/1"
        }
    },
    {
        "test-case": "test-cabling",
        "dut": "switch-2105.bld1",
        "params": {
            "interface": "Ethernet50/1",
            "role": "spine-leaf"
        },
        "expected": {
            "remote-hostname": "switch-22.bld1",
            "remote-interface": "Ethernet3/1"
        }
    },
```

DUT Actual Data

The test function will need to examine the DUT actual data and apply the logic to compare it with the test-case data.

The test function for this tutorial will be tightly coupled to the DUT actual data, i.e. specific to this device/NOS. An example JSON body is shown.

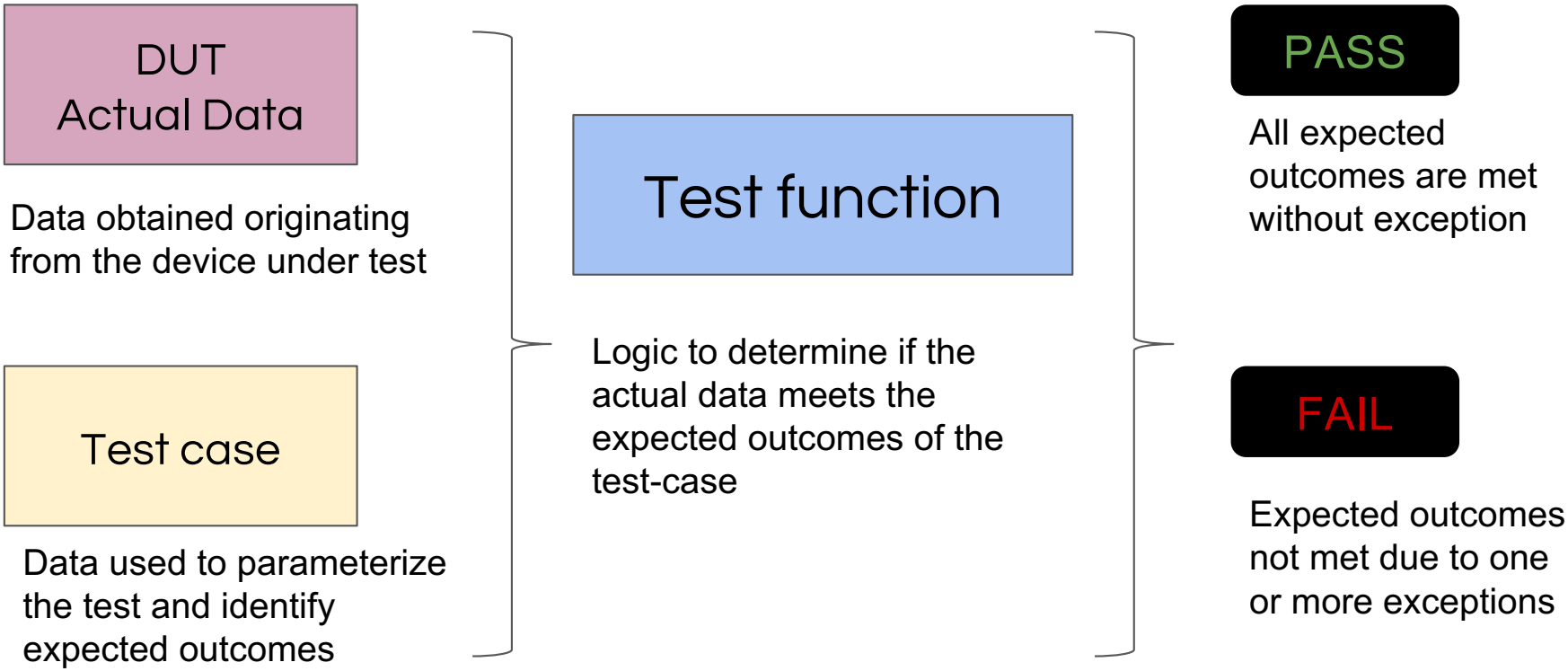
The test function will extract the specific fields and perform the validation logic.

`show lldp neighbors`


```
{
    "lldpNeighbors": [
        {
            "ttl": 120,
            "neighborDevice": "switch-21.bld1",
            "neighborPort": "Ethernet3/1",
            "port": "Ethernet49/1"
        },
        {
            "ttl": 120,
            "neighborDevice": "switch-22.bld1",
            "neighborPort": "Ethernet3/1",
            "port": "Ethernet50/1"
        },
        {
            "ttl": 120,
            "neighborDevice": "switch-2106.bld1",
            "neighborPort": "Ethernet59/1",
            "port": "Ethernet59/1"
        },
        {
            "ttl": 120,
            "neighborDevice": "switch-2106.bld1",
            "neighborPort": "Ethernet60/1",
            "port": "Ethernet60/1"
        }
    ]
}
```


Example Arista EOS via eAPI

Putting it Together



Testing Considerations

Separate the “What” from the “How”

Actual data

- How does the test obtain the DUT data?

Test-cases

- How are these test-cases created?
- How are these test-cases stored?
- How does test get these test-cases?

Obtaining DUT Actual Data

- Open a session directly to the device, e.g. using Netmiko, pyEAPI, etc.
- Use a controller-server that aggregates many devices
- Use a database that is populated by yet another software system
- Normalizing data from multiple vendors

… always based on your specific constraints

Obtaining Test-Cases

- Write a program to generate test-case data based on some other Source-of-Truth system(s), which could be
  - One or more spreadsheets
  - From an internal database system
  - From other SoT systems like NetBox

… always based on your specific constraints

Test Functions

Writing tightly-coupled test functions … is OK

The test functions uses the DUT actual data

If the test function is written to use a specific
DUT output format, then it is tightly-coupled to that DUT

Same is true for the structure and/or content of the test-case

Decisions around abstractions, normalizing data, etc.,
are specific to your specific constraints

Recap

- Test coverage is a collection of test functions
- Test functions are run with multiple test-cases
- Test functions OK to be tightly coupled to DUT
  - Test-case structure not tightly coupled
  - Test-case data tightly coupled to DUT

There are no “right answers” only “right now answers”

Introduction to pytest

pytest

http://pytest.org

pytest Tutorial

- Basic project setup
- Writing test functions
- Writing test function fixtures
- Writing parameterized test-cases
- Running pytest to filter test cases
- Creating HTML reports

Installing pytest

Recommend installing into virtualenv

```
(venv)$ pip install pytest
(venv)$ pip install pytest-html
```

Almost 700 pytest plugins listed:
http://plugincompat.herokuapp.com/

Live Tutorials

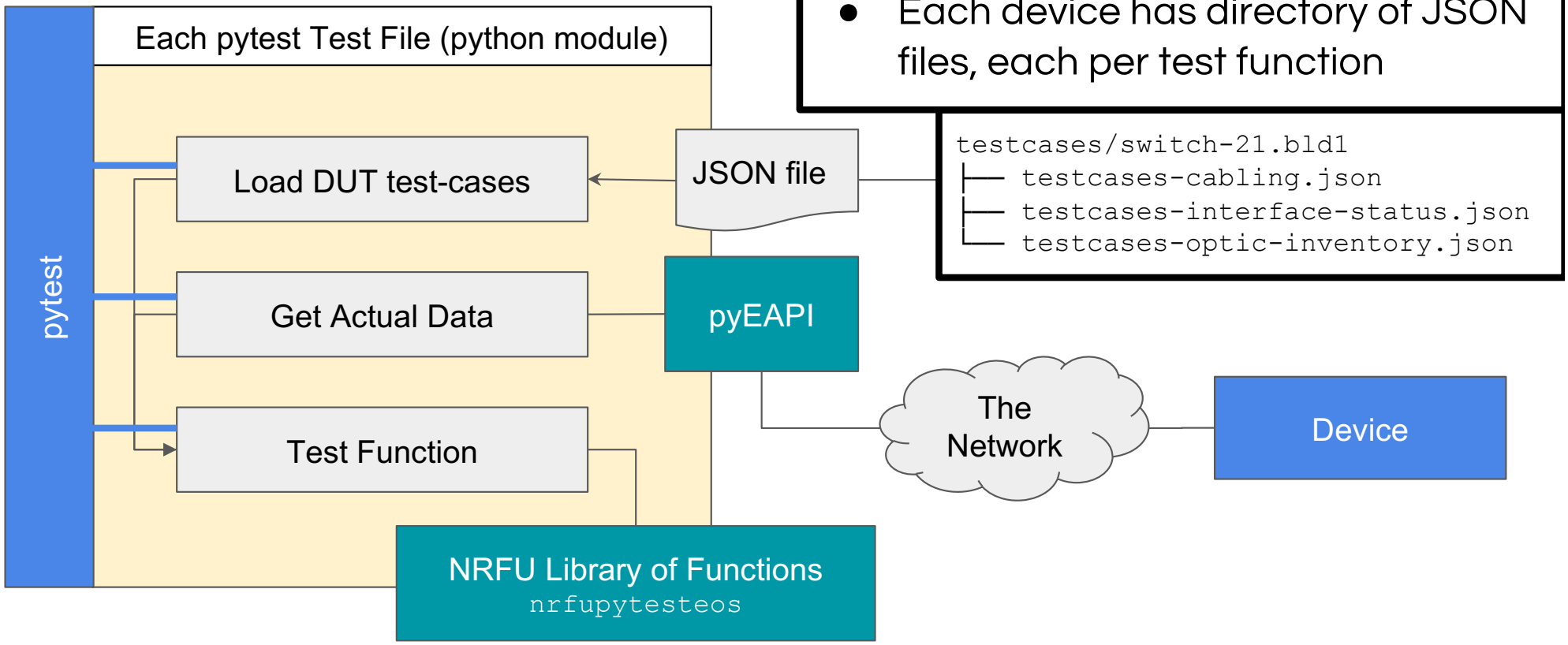
“NRFU” pytest Coding Tutorials

NRFU Tutorials

1. Create custom pytest options
2. Initially verify the device is online
3. Running test-functions
    a. Provide device instance to test-functions
    b. Load device specific test-cases
    c. Using “bulk-data” type device commands

Testing Strategy

- Test one device per pytest run
- Write one test file per test function
- Each device has directory of JSON files, each per test function



what

Create Custom pytest Options

I want to tell pytest the device I want to run tests against. I need to provide the device hostname and also the directory that has the test-case files.

```
$ pytest \
    --nrfu-device switch-21.bld1 \
    --nrfu-testcases /opt/nrfu/testcases/switch-21.bld1 \
    <other pytest options>
```

how

Create Custom pytest Options

Write a `pytest_addoption` hook function to add your custom options:

conftest.py

```
1
2   def pytest_addoption(parser):
3       parser.addoption("--nrfu-device",
4                        required=True,
5                        help='device name or IP address')
6
7       parser.addoption("--nrfu-testcasedir",
8                        required=True,
9                        help='directory storing device test-case files')
10
```

what

Initially Verify Device Online

Once I give pytest the device information, I want pytest to first ensure the device is online before I start running any tests.

```
$ pytest --nrfu-device switch-111.bld2 <...>
Exit: Unable to connect to device switch-111.bld2
!!!!!!!!! _pytest.outcomes.Exit: Unable to connect to device switch-111.bld2 !!!!!!!!!
```

how

Initially Verify Device Online

Write a `pytest_sessionstart` hook function to run code before any tests are executed:

conftest.py

```
1   import os
2   import pytest
3   from nrfupytesteos import Device
4
5
6   def pytest_sessionstart(session):
7       device_name = session.config.getoption("--nrfu-device")
8
9       dev = Device(device_name, username=os.getenv('EOS_USER') or os.environ['USER'],
10                   password=os.environ['EOS_PASSWORD'])
11
12      if not dev.probe():
13          pytest.exit(f"Unable to access device {device_name}: IP unreachable")
14
```

what

Provide the device to test functions

I will need to provide the device object to my test function so that I can retrieve the actual data from the device.

test_cabling.py

```
15
16  import nrfupytesteos.nrfu_cabling as nrfu
17
18
19  def test_cabling(device, testcase):
20
21      if_name = testcase['params']['interface']
22      if_lldp = device.execute(f"show lldp neighbors {if_name}")
23
24      nrfu.test_cabling(
25          device=device,
26          actual=if_lldp, testcase=testcase)
27
```

The `device` object provides access to the DUT

how

Provide the device to test-cases

Create a function called `device()` in `conftest.py` and as a session level fixture. This function will be run only once.

conftest.py

```
29  @pytest.fixture(scope='session')
30  def device(pytestconfig):
31
32      return Device(hostname=pytestconfig.option.nrfu_device,
33                    username=os.getenv('EOS_USER') or os.environ['USER'],
34                    password=os.environ['EOS_PASSWORD'])
```

what

Load device specific test-cases

I need to use test-cases that are specific to the device under test. Using the CLI option I provided, load the test-case file to parameterize the test cases to the test functions

```
$ pytest \
      --nrfu-testcases /opt/nrfu/testcases/switch-21.bld1 \
      <...>
```

what

Load device specific test-cases

I want to see each test case instance reported. For example, cabling test shows the expected remote device and interface:

```
test_02_cabling.py::test_cabling[Ethernet49/1<[spine-leaf]->switch-21.bld1:Ethernet3/1] PASSED          [ 88%]
test_02_cabling.py::test_cabling[Ethernet50/1<[spine-leaf]->switch-22.bld1:Ethernet3/1] PASSED          [ 92%]
test_02_cabling.py::test_cabling[Ethernet59/1<[peer-pair]->switch-2106.bld1:Ethernet59/1] PASSED        [ 96%]
test_02_cabling.py::test_cabling[Ethernet60/1<[peer-pair]->switch-2106.bld1:Ethernet60/1] PASSED        [100%]
```

how

Load device specific test-cases

Write the test function to uses a fixture, in this example called `testcase`. This fixture needs to be parameterized using a JSON file.

test_cabling.py

```
17
18
19  def test_cabling(device, testcase):
20
21      if_name = testcase['params']['interface']
22      if_lldp = device.execute(f"show lldp neighbors {if_name}")
23
24      nrfu.test_cabling(
25          device=device,
26          actual=if_lldp, testcase=testcase)
27
```

The `testcase` fixture is a dictionary for one specific test case that will come from the device JSON file

how

Load device specific test-cases

write the `pytest_generate_tests` hook function, in the same python file containing the test case function:

test_cabling.py

```
37  import json
38  from pathlib import Path
39
40  def pytest_generate_tests(metafunc):
41      tc_file = Path(metafunc.config.option.nrfu_testcasedir) / 'testcases-cabling.json'
42
43      metafunc.parametrize('testcase',
44                           json.load(tc_file.open()),
45                           ids=nrfu.name_test)
46
```

how

Load device specific test-cases

Use `parametrize` to dynamically load the `testcase` fixture values from any source, a JSON file in this example:

test_cabling.py

```
39
40  def pytest_generate_tests(metafunc):
41      tc_file = Path(metafunc.config.option.nrfu_testcasedir) / 'testcases-cabling.json'
42
43      metafunc.parametrize('testcase',
44                           json.load(tc_file.open()),
45                           ids=nrfu.name_test)
46
```

Load from the JSON file the list of test case dictionary items into the fixture called “testcase”

how

Load device specific test-cases

`parametrize` takes the fixture name as a string. This value must match the same argument name used by the test function.

test_cabling.py

```
39
40  def pytest_generate_tests(metafunc):
41      tc_file = Path(metafunc.config.option.nrfu_testcasedir) / 'testcases-cabling.json'
42
43      metafunc.parametrize('testcase',
44                           json.load(tc_file.open()),
45                           ids=nrfu.name_test)
46
```

The fixture name “testcase” MUST MATCH the fixture name used in the test function

how

Generating test-cases names

test_cabling.py

```
40  def name_test(testcase):
41      """ used for pytest verbose output """
42      rmt_host = testcase['expected']['remote-hostname']
43      rmt_ifn = testcase['expected']['remote-interface']
44      return f"{testcase['params']['interface']}<[{testcase['params']['role']}]->{rmt_host}:{rmt_ifn}"
45
46
47  def pytest_generate_tests(metafunc):
48      tc_file = Path(metafunc.config.option.nrfu_testcasedir
49
50      metafunc.parametrize('testcase',
51                           json.load(tc_file.open()),
52                           ids=name_test)
53
```

The `name_test` function is called for each testcase dictionary item

what

Recap: device specific test-cases

The coding shown in the previous slides will result in pytest running the test function multiple times, each with the specific test name

```
test_02_cabling.py::test_cabling[Ethernet49/1<[spine-leaf]->switch-21.bld1:Ethernet3/1] PASSED          [ 88%]
test_02_cabling.py::test_cabling[Ethernet50/1<[spine-leaf]->switch-22.bld1:Ethernet3/1] PASSED          [ 92%]
test_02_cabling.py::test_cabling[Ethernet59/1<[peer-pair]->switch-2106.bld1:Ethernet59/1] PASSED        [ 96%]
test_02_cabling.py::test_cabling[Ethernet60/1<[peer-pair]->switch-2106.bld1:Ethernet60/1] PASSED        [100%]
```

what

Use “bulk-data” device results

In some cases I need to run a device command that will contain the results for many test-cases.

For example I must use the “`show inventory`” command to obtain the transceiver data. That command does not allow me to get data for one interface at a time.

how

Use “bulk-data” device results

Define a module level fixture to return the bulk-data. A module fixture will be called only once per python test-function module.

how

Use “bulk-data” device results

test_optics.py

```python
import json
from pathlib import Path
import pytest

import nrfupytesteos.nrfu_optic_inventory as nrfu

def pytest_generate_tests(metafunc):
    tc_file = Path(metafunc.config.option.nrfu_testcasedir) / 'testcases-optics.json'

    metafunc.parametrize('testcase',
                         json.load(tc_file.open()),
                         ids=nrfu.name_test)


@pytest.fixture(scope='module')
def device_inventory(device):
    return device.execute('show inventory')


def test_optic_inventory(device, device_inventory, testcase):
    nrfu.test_optic_inventory(
        device=device,
        actual=device_inventory, testcase=testcase)
```

Called once

Recapping Tutorial

Summary

- pytest can be used to automate network testing
- pytest has a large community & catalog of plugins
- Useful for both “greenfield” and “brownfield” scenarios
- You still need to write your own test functions
- You still need to determine your testing strategy

Opportunity for Networking community

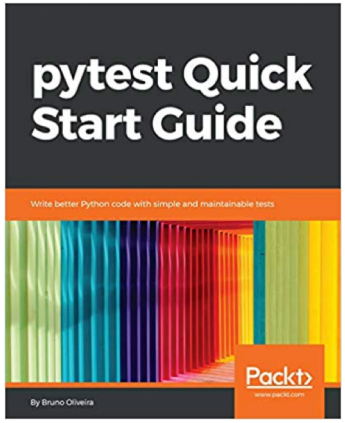
Reference Links

https://twitter.com/pytestdotorg

- http://www.pytest.org
- http://www.pytest.org/en/latest/#documentation
- https://docs.pytest.org/en/latest/talks.html
- https://docs.pytest.org/en/latest/parametrize.html
- https://docs.pytest.org/en/latest/fixture.html#fixture-parametrize

Podcasts, Blogs, and much more at:

https://pythontesting.net/



Q & A

Thank you!

Jeremy Schulman
@nwkautomaniac