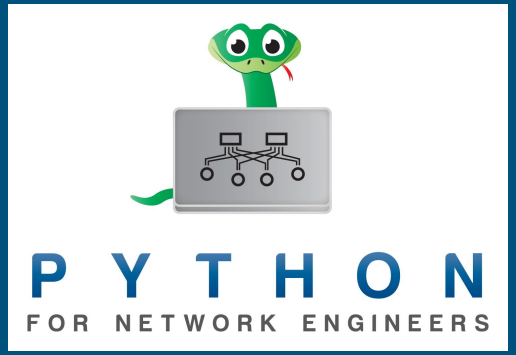
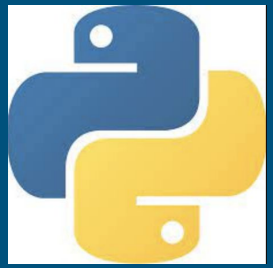
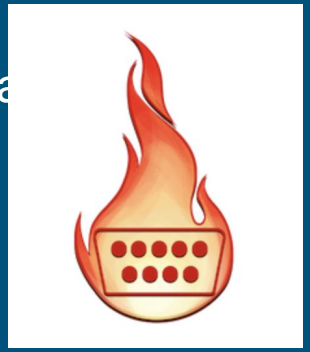




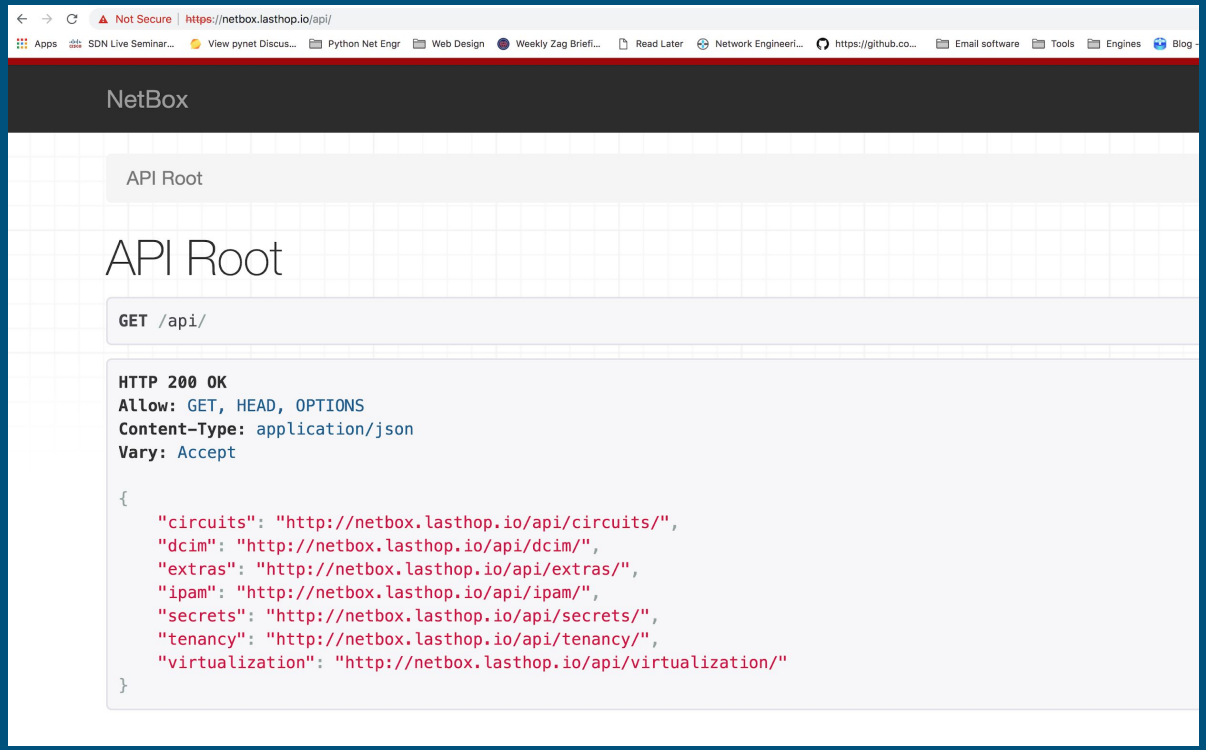
Live Training Session
December 2021

Day4 Schedule

- Requests and using a REST-API
- Jinja2 Templating
- Deploying Jinja2 Generated Configurations
- NAPALM
- Integrating to the operating system with subprocess (optiona
- Intro to Unit Testing and CI/CD



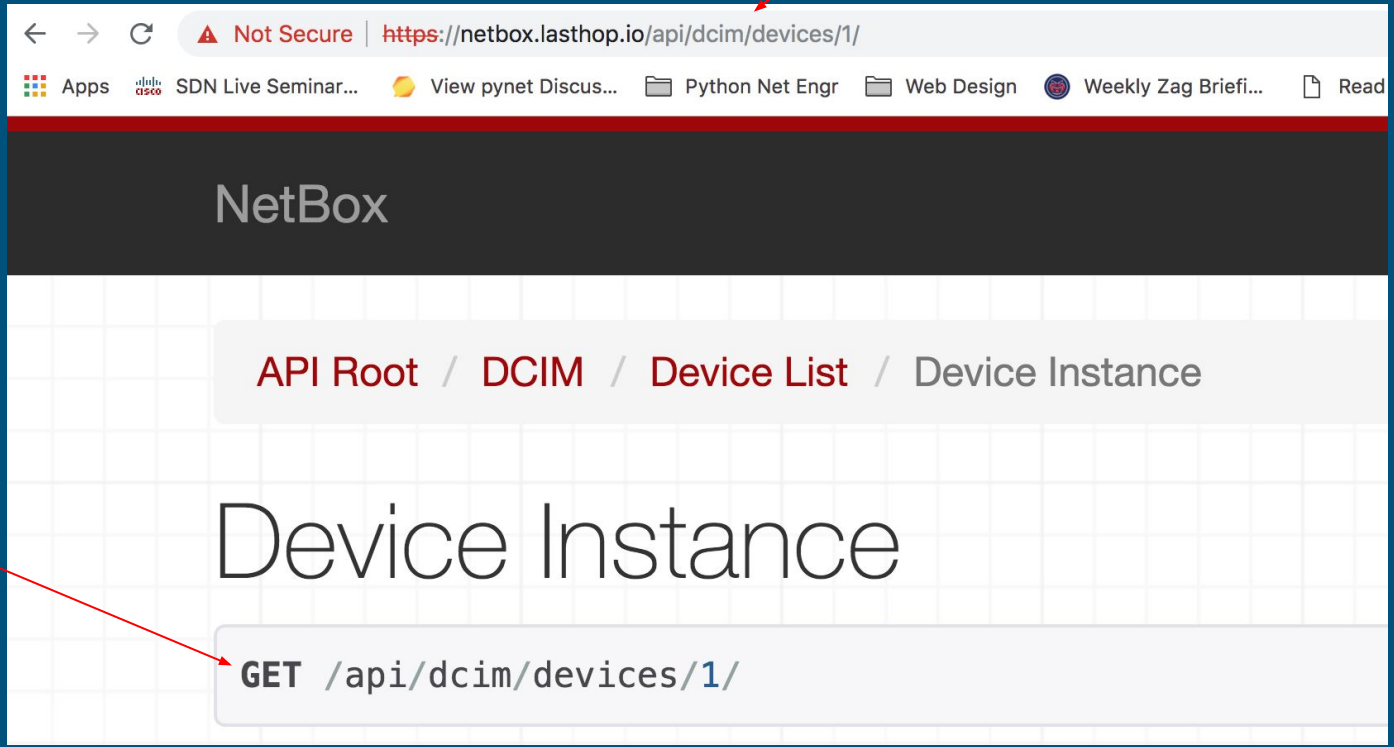
REST API


Not Secure | https://netbox.lasthop.io/api/
Apps
SDN Live Seminar...
View pynet Discus...
Python Net Engr
Web Design
Weekly Zag Briefi...
Read Later
Network Engineeri...
https://github.co...
Email software
Tools
Engines
Blog
NetBox
API Root
API Root
GET /api/
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
{
"circuits": "http://netbox.lasthop.io/api/circuits/",
"dcim": "http://netbox.lasthop.io/api/dcim/",
"extras": "http://netbox.lasthop.io/api/extras/",
"ipam": "http://netbox.lasthop.io/api/ipam/",
"secrets": "http://netbox.lasthop.io/api/secrets/",
"tenancy": "http://netbox.lasthop.io/api/tenancy/",
"virtualization": "http://netbox.lasthop.io/api/virtualization/"
}

REST API - Characteristics

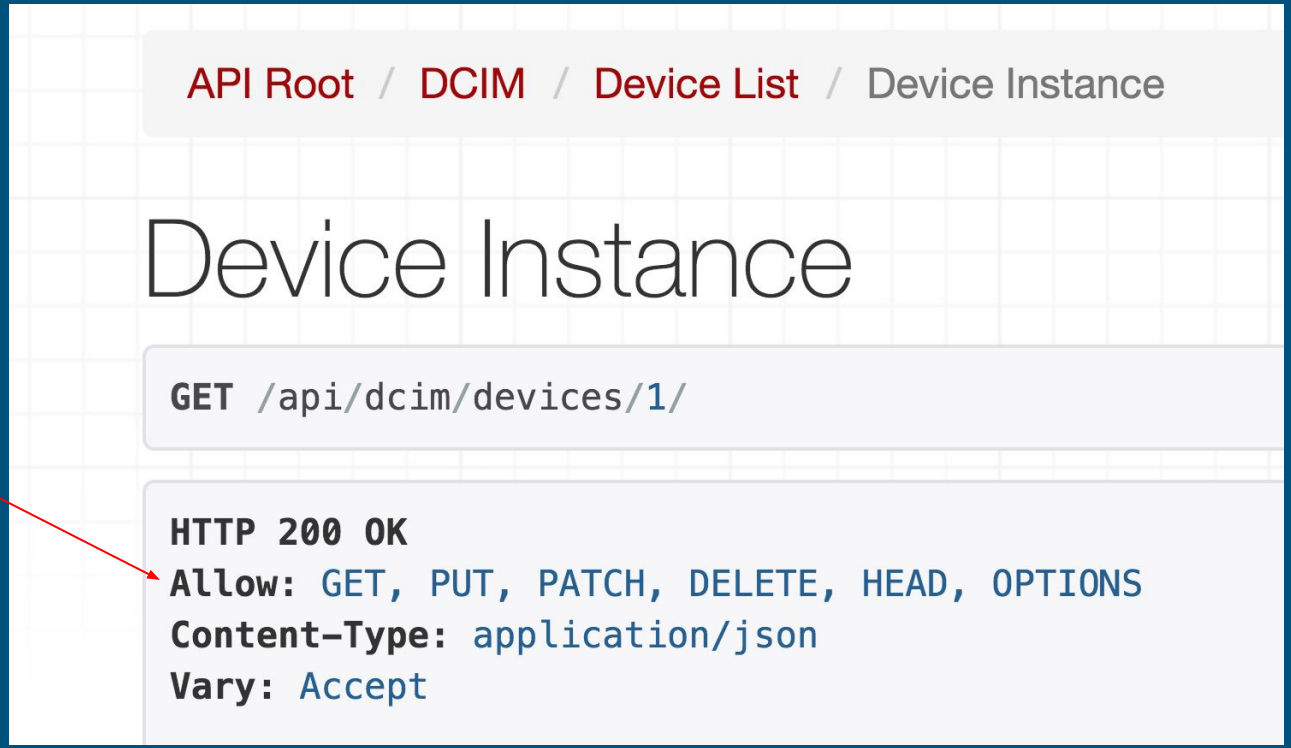
URL - the object I am accessing.

HTTP Method



REST API - Other HTTP Methods

Available HTTP Methods



REST API - CRUD

- Create - HTTP Post
- Read - HTTP Get
- Replace - HTTP Put
- Update - HTTP Patch
- Delete - HTTP Delete

Remember: Not all APIs are the same!

HTTP Methods Ref: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

REST API - Accessing API via Browser + CLI

```
(py3_venv) [kbyers@ip-172-30-0-118 ~]$
(py3_venv) [kbyers@ip-172-30-0-118 ~]$ curl -s https://netbox.lasthop.io/api/ --insecure | jq "."
{
  "circuits": "http://netbox.lasthop.io/api/circuits/",
  "dcim": "http://netbox.lasthop.io/api/dcim/",
  "extras": "http://netbox.lasthop.io/api/extras/",
  "ipam": "http://netbox.lasthop.io/api/ipam/",
  "secrets": "http://netbox.lasthop.io/api/secrets/",
  "tenancy": "http://netbox.lasthop.io/api/tenancy/",
  "virtualization": "http://netbox.lasthop.io/api/virtualization/"
}
(py3_venv) [kbyers@ip-172-30-0-118 ~]$
```

REST API - Basic Requests Get

Reference Material in:
{{ github_repo }}/rest_api

```python
import requests
from rich import print
import pdbr


if __name__ == "__main__":

    url = "https://netbox.lasthop.io/api/dcim/"
    http_headers = {"accept": "application/json; version=2.4;"}
    response = requests.get(url, headers=http_headers, verify=False)
    response = response.json()

    print()
    print(response)
    print()
```

Authentication

- Simple Auth
- Token Based
- OAuth

```python
import requests
from pprint import pprint

from urllib3.exceptions import InsecureRequestWarning

requests.packages.urllib3.disable_warnings(category=InsecureRequestWarning)


if __name__ == "__main__":

    token = "1234123412341234123412341341341134123433"
    url = "https://netbox.lasthop.io/api/dcim/devices/1"
    http_headers = {"accept": "application/json; version=2.4;"}
    if token:
        http_headers["authorization"] = "Token {}".format(token)

    response = requests.get(url, headers=http_headers, verify=False)
    response = response.json()

    print()
    pprint(response)
    print()
```

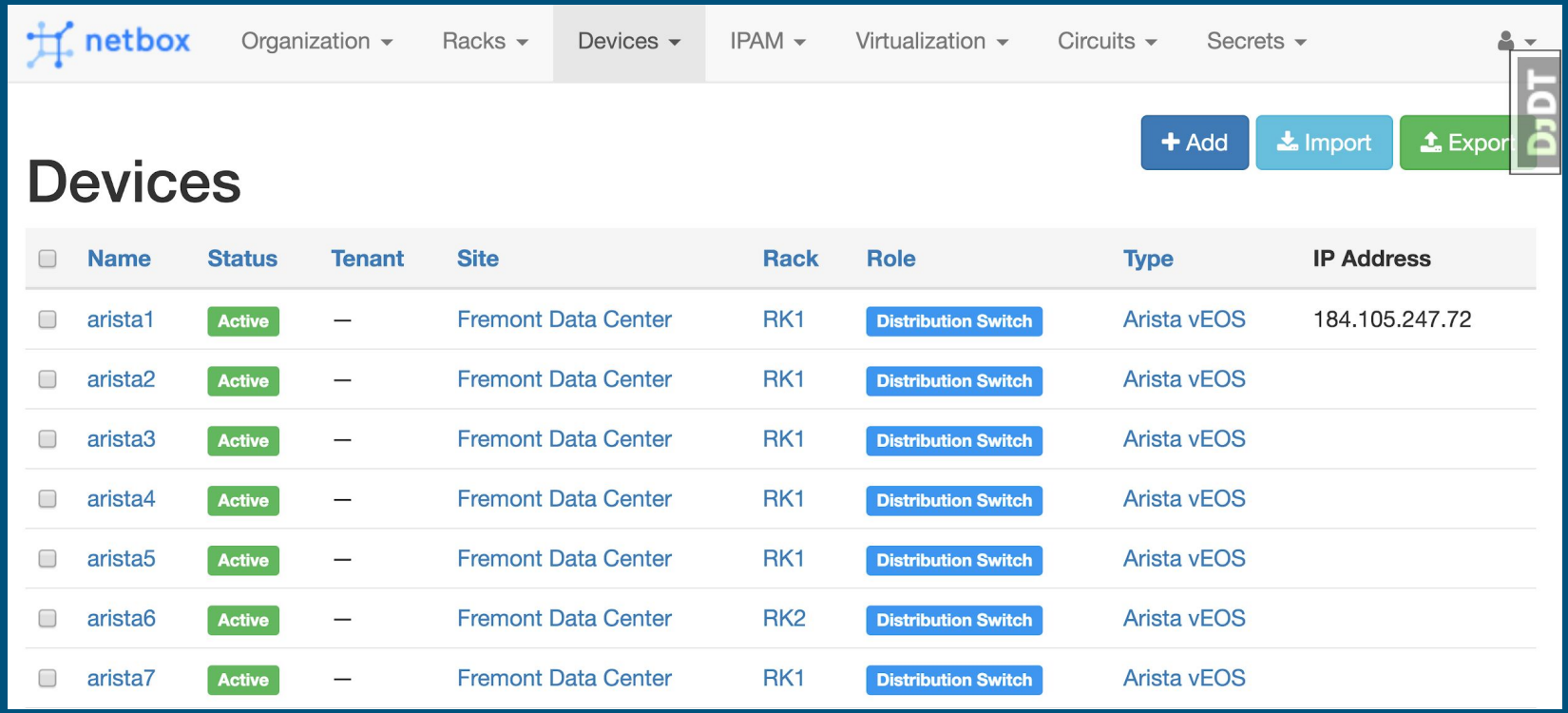
Tokens, Tokens Everywhere! & REST API - Basic Requests POST

- Tokens can be included in multiple locations:
  - Headers
  - Encoded in URL
  - In a payload

- POST - expects a “payload”

```python
def main():
    # Get list of channels; authenticate with token in the header
    headers = {"Authorization": f"Bearer {SLACK_TOKEN}"}
    resp = requests.get(f"{SLACK_BASE_URL}/channels.list", headers=headers)
    pprint(resp.json())

    print()

    # Get list of channels; authenticate with token encoded in url
    resp = requests.get(f"{SLACK_BASE_URL}/channels.list?token={SLACK_TOKEN}")
    pprint(resp.json())

    print()

    # Get list of channels; authenticate with token encoded in url
    data = {"token": SLACK_TOKEN}
    resp = requests.post(f"{SLACK_BASE_URL}/channels.list", data=data)
    pprint(resp.json())

    print()
```

REST API - Adding a device using HTTP POST



REST API - Adding a device using HTTP POST

```
http_headers = {
    "Content-Type": "application/json; version=2.4;",
    "authorization": "Token {}".format(token),
}
post_data = {
    "name": "arista8",
    "device_role": 3,  # Distribution Switch
    "device_type": 2,  # vEOS
    "display_name": "arista8",
    "platform": 4,  # Arista EOS
    "rack": 1,  # RK1
    "site": 1,  # Fremont Data Center
    "status": 1,  # Active
}

response = requests.post(
    url, headers=http_headers, data=json.dumps(post_data), verify=False
)
response = response.json()
```

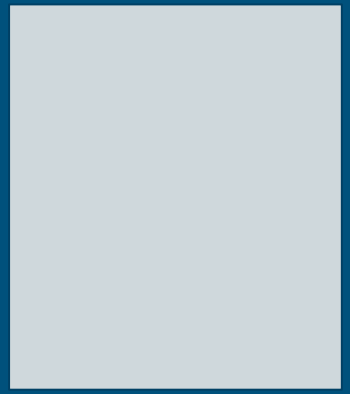
REST API - Modify (put) and Delete


```
response = requests.put(
    url, headers=http_headers, data=json.dumps(arista6), verify=False
)
```

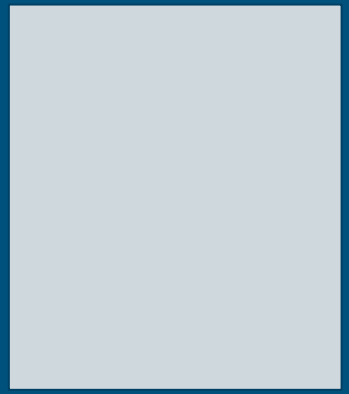
```
response = requests.delete(url, headers=http_headers, verify=False)
if response.ok:
    print("Device deleted successfully")
```

Exercises:
./day4/rest/ex1.txt
./day4/rest/ex2.txt

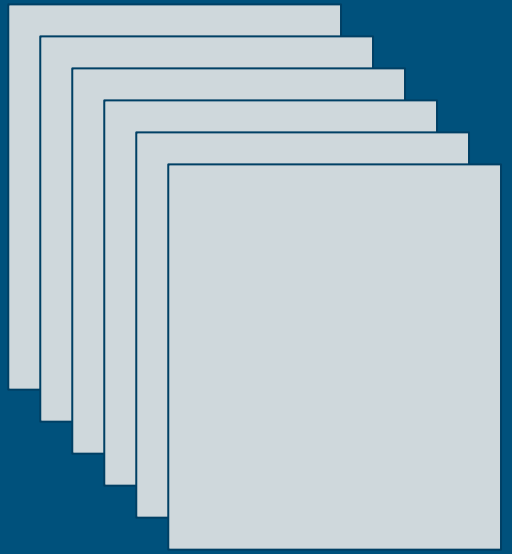
Variables



Templates



Output Files



Jinja2 Templating

```
import jinja2

my_dict = {
    "ip_addr1": "1.1.1.2",
    "ip_addr2": "2.2.2.2"
}

my_template = """
some
text
of
something
{{ ip_addr1 }}
{{ ip_addr2 }}
something
"""

t = jinja2.Template(my_template)
print(t.render(my_dict))
```

Reference Material in:

{{ github_repo }}/jinja2_example/jinja2_simple.py

{{ github_repo }}/jinja2_example/jinja2_bgp.py

```
some
text
of
something
1.1.1.2
2.2.2.2
something
```

Jinja2 Templating
Loading Template from a File


```python
import jinja2

template_file = "juniper_bgp.j2"
with open(template_file) as f:
    bgp_template = f.read()

my_vars = {
    "peer_as": "22",
    "neighbor1": "10.10.10.2",
    "neighbor2": "10.10.10.99",
    "neighbor3": "10.10.10.220",
}

template = jinja2.Template(bgp_template)
print(template.render(my_vars))
```

Reference Material in:

{{ github_repo }}/jinja2_example/jinja2_bgp_file.py

```
protocols {
    bgp {
        group external-peers {
            type external;
            peer-as 22;
            neighbor 10.10.10.2;
            neighbor 10.10.10.99;
            neighbor 10.10.10.220;
        }
    }
}
```

Reference Material in:
{{ github_repo }}/jinja2_example/jinja2_env.py

Jinja2 Template - Environment

```
from jinja2 import FileSystemLoader, StrictUndefined
from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)
env.loader = FileSystemLoader([".", "./templates/"])

my_vars = {"bgp_as": 22, "router_id": "1.1.1.1", "peer1": "10.20.30.1"}

template_file = "bgp_config.j2"
template = env.get_template(template_file)
output = template.render(**my_vars)
print(output)
```

Jinja2 Templating - Conditionals


```
interface GigabitEthernet0/0/0
 {%- if primary_ip %}
 ip address 10.220.88.22 255.255.255.0
 {%- endif %}
 negotiation auto
```

Jinja2 Templating - Loops

```
protocols {
    bgp {
        group external-peers {
            type external;
            {%- for neighbor_ip, neighbor_as in my_list %}
                neighbor {{ neighbor_ip }} {
                    peer-as {{ neighbor_as }};
                }
            {%- endfor %}
        }
    }
}
```

Reference Material in:

{{ github_repo }}/jinja2_example/jinja2_bgp_loop.py

Jinja2 - Other Topics

- Jinja2 Whitespace Stripping
- Jinja2 Create Variables `{% set ip_addr = "10.220.179.10" %}`
- Jinja2 Filters `{{ def_gateway | default("10.220.179.1") }}`
- Jinja2 Macros
- Jinja2 Includes / Hierarchy `{% include 'other_template.j2' %}`

Exercises:
./day4/jinja2/ex1.txt
./day4/jinja2/ex2.txt
./day4/jinja2/ex3.txt

Jinja2 - Deploying Generated Configurations

```
def render_configs(j2_env, template_file, my_vars):
    # Render configs
    template = j2_env.get_template(template_file)
    config_section = template.render(**my_vars)
    return config_section


def load_configs(host, config, junos_format="text"):

    # PyEZ connection
    a_device = Device(host=host, user=USER, password=PASSWORD)
    a_device.open()
    a_device.timeout = 90

    # Load config object
    cfg = Config(a_device)
    cfg.load(config, format=junos_format, merge=True)
    return cfg
```

Reference Material in:
{{ github_repo }}/jinja2_deploy/

PYTHON
FOR NETWORK ENGINEERS

NAPALM

Purpose of NAPALM: create a standard set of operations across a range of platforms (generally Cisco IOS/IOS-XE, NX-OS, IOS-XR, Arista EOS, and Junos)

Operations fall into two general categories: Config Operations + Getter Operations.

Reference Material in:
{{ github_repo }}/napalm_example/

NAPALM Vendors

CORE
Arista EOS
Cisco IOS/IOS-XE
Cisco IOS-XR
Cisco NX-OS
Juniper Junos

NAPALM Community Drivers

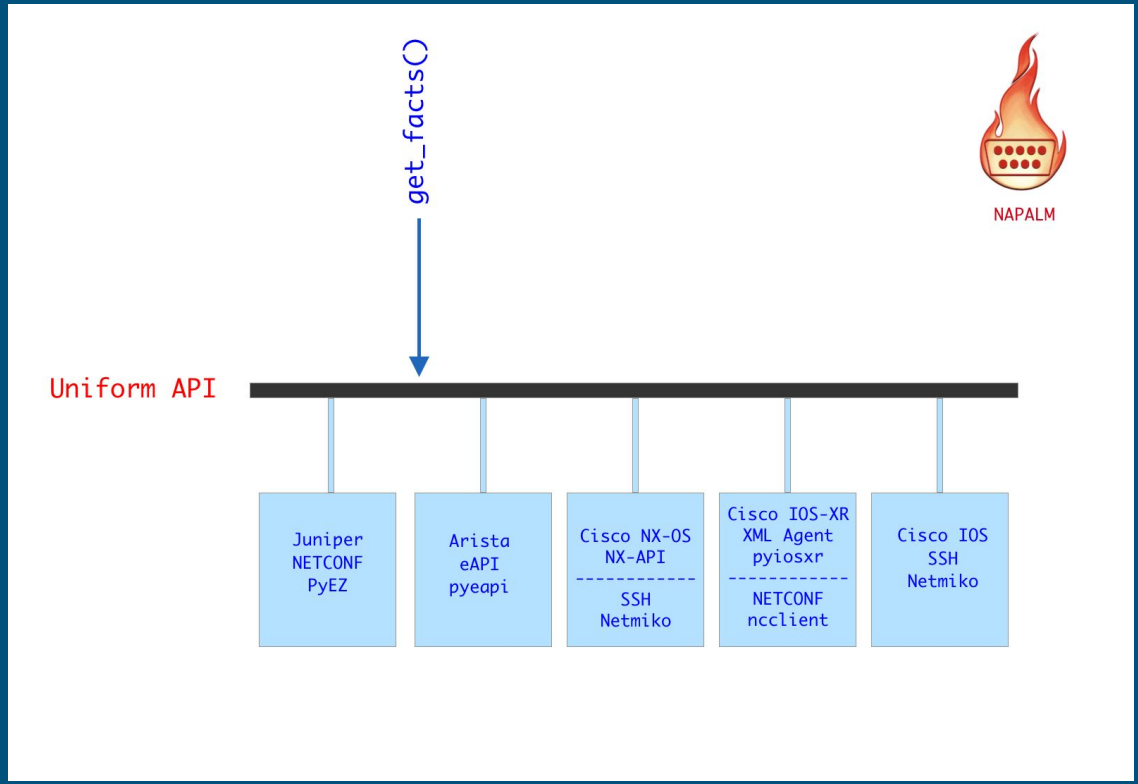
https://github.com/napalm-automation-community

NAPALM-Ansible Integration

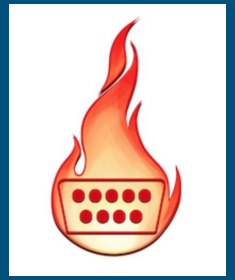
NAPALM-Salt Integration

NAPALM-Nornir Integration

NAPALM and a Uniform API



NAPALM Getters



Getters support matrix

Note

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

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

NAPALM Getters

https://napalm.readthedocs.io/en/latest/support/#getters-support-matrix

get_arp_table
get_bgp_config
get_bgp_neighbors
get_bgp_neighbors_detail
get_config
get_environment
get_facts
get_interfaces
get_interfaces_counters
get_interfaces_ip
get_ipv6_neighbors_table
get_lldp_neighbors
get_lldp_neighbors_detail
get_mac_address_table
get_network_instances
get_ntp_peers
get_ntp_servers
get_ntp_stats
get_optics
get_probes_config
get_probes_results
get_route_to
get_snmp_information
get_users
is_alive
ping
traceroute

Getter Example

```
cisco3 = {
    "hostname": "cisco3.lasthop.io",
    "username": "pyclass",
    "password": getpass(),
    "optional_args": {},
}

driver = get_network_driver("ios")
device = driver(**cisco3)

print()
print("\n\n>>>Test device open")
device.open()
output = device.get_facts()

print()
print(output)
print()
```

NAPALM Config Operations

Config Merge
Config Replace

Compare Config (Diff)
Discard Config (Drop Pending Changes)

Commit Config

Rollback (Rollback to State Before Commit)

NAPALM Config Example

```
driver = get_network_driver("nxos")
device = driver(host, username, password, optional_args=optional_args)

device.open()
device.load_merge_candidate(filename="nxos-merge.conf")
print(device.compare_config())

device.discard_config()
print("--- Diff ---")
print(device.compare_config())
print("--- End Diff ---")

print(">>>Load config change (merge) - commit")
device.load_merge_candidate(filename="nxos-merge.conf")
device.commit_config()

device.rollback()
```

Exercises:
./day4/napalm/ex1.txt
./day4/napalm/ex2.txt

Subprocess - *Integrating to the System Operating System*

```
import os

print()
print("Current working directory")
start_dir = os.getcwd()
print(os.getcwd())

print()
print("Path of module we are executing")
print(os.path.realpath(__file__))

print()
print("Is this a file?")
print(os.path.isfile(__file__))

print()
print("Change directory into /tmp")
os.chdir("/tmp")
print(os.getcwd())
```

PYTHON
FOR NETWORK ENGINEERS

Subprocess - *Integrating to the System Operating System*

```
import subprocess

cmd_list = ["ls", "-a", "-l"]
proc = subprocess.Popen(cmd_list, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
std_out, std_err = proc.communicate()
(std_out, std_err) = [x.decode("utf-8") for x in (std_out, std_err)]

print(std_out)
print(std_err)
```

Exercises:
./day4/subprocess/ex1.txt

Reference Material in:
{{ github_repo }}/subprocess_example

PYTHON
FOR NETWORK ENGINEERS

Unit testing with pytest

```python
# Functions
def increase_by_one(x):
    return x + 1

# Tests
def test_simple_case():
    assert increase_by_one(3) == 4

def test_various_cases():
    assert increase_by_one(-1) == 0
    assert increase_by_one(0) == 1
    assert increase_by_one(10) == 11
    assert increase_by_one(-10) == -9
```

1. Test the components of your code (functions, methods).
2. Raises confidence that your code is working properly.
3. Allows you to change the code with a higher degree of confidence that you didn't introduce new errors.

Reference Material in:

{{ github_repo }}/unittest_example

pytest - How to get started?

```python
# Functions
def increase_by_one(x):
    return x + 1


# Tests
def test_simple_case():
    assert increase_by_one(3) == 4


def test_various_cases():
    assert increase_by_one(-1) == 0
    assert increase_by_one(0) == 1
    assert increase_by_one(10) == 11
    assert increase_by_one(-10) == -9
```

1. Create a file named test_something.py.
2. In this file define functions named test_thing_you_are_testing.
3. Add assert statements inside the tests (if the assert evaluates to True, your test will pass; otherwise, it will fail).

pytest - Run the test.


```
(py3_venv) [student20@onslab1a pytest_dir]$ py.test -s -v test_simple.py
==================================== test session starts ====================================
platform linux -- Python 3.6.10, pytest-5.4.3, py-1.9.0, pluggy-0.13.1 -- /home/student20/VENV
/py3_venv/bin/python3
cachedir: .pytest_cache
rootdir: /home/student20/pynet-ons/unittest_example/pytest_dir
plugins: pylama-7.7.1, f5-sdk-3.0.21
collected 2 items

test_simple.py::test_simple_case PASSED
test_simple.py::test_various_cases PASSED

==================================== 2 passed in 0.01s =====================================
```

pytest - Run the test.

1. Use the -v (verbose) argument and the -s (std-out) arguments.
2. Change directory into the directory with your test files.
3. You can specify: py.test -s -v .
4. This will search for all files named test_ and inside those files for any functions named test_.
5. There is also a way you can test clases.
6. If you specify a directory py.test will recurse down into subdirectories.

pytest - Skip tests


```python
@pytest.mark.skip(reason="Unable to test")
def test_example():
    assert True

@pytest.mark.skipif(sys.version_info < (3, 7), reason="requires python3.7 or later")
def test_skip_os_ver():
    print(sys.version_info)
    assert True
```

pytest - parametrize


```python
# Functions
def increase_by_one(x):
    return x + 1

def test_various_cases():
    assert increase_by_one(-1) == 0
    assert increase_by_one(0) == 1
    assert increase_by_one(10) == 11
    assert increase_by_one(-10) == -9

@pytest.mark.parametrize("number, result", [(-1, 0), (0, 1), (10, 11), (-10, -9)])
def test_various_cases_params(number, result):
    assert increase_by_one(number) == result
```

pytest - *These examples show you the mechanics, but how to approach the problem?*

1. Have one clear purpose to your functions and methods.
2. Test verifies that one clear purpose.
3. Layer unit tests, integration tests, and systems tests.
4. Don’t forget linting tools.
5. Use code-coverage tools to look for how well you are testing.
6. Use CI-CD tools to integrate testing into your workflow.

But my unit tests work...

1. Don’t miss the forest for the trees.
2. How to test things that interface to remote systems (mock, test systems).
3. How to apply this all in a network engineering context.

Exercises:
./day4/testing/ex1.txt
./day4/testing/ex2.txt

Creating a fixture


```python
@pytest.fixture(scope="module")
def netmiko_connect():
    """Establish a netmiko connection."""
    device = {
        "device_type": "juniper_junos",
        "host": "vmx2.lasthop.io",
        "username": "pyclass",
        "password": getpass(),
    }
    return ConnectHandler(**device)
```

Using a fixture

Reference Material in:
{{ github_repo }}/unittest_example/separate_fixture

```
def test_prompt(netmiko_connect):
    assert netmiko_connect.find_prompt() == "pyclass@vmx2>"


def test_show_version(netmiko_connect):
    output = netmiko_connect.send_command("show version")
    assert "Junos: 18.4R1.8" in output


def test_config_mode(netmiko_connect):
    netmiko_connect.config_mode()
    prompt = netmiko_connect.find_prompt()
    assert prompt == "pyclass@vmx2#"
```

If it doesn't happen automatically; it didn't happen.

Continuous Integration using GitHub Actions.

Define a .github/workflows/commit.yaml

Specify Events

Setup Environment

Install Dependencies

Add linting/tests

```yaml
---
name: build
on: [push, pull_request]
jobs:
  std_tests:
    runs-on: ubuntu-latest
    strategy:
      max-parallel: 4
      matrix:
        python-version: [3.6, 3.7, 3.8, 3.9, 3.10.0]

    steps:
      - name: Checkout repository
        uses: actions/checkout@v2

      - name: Setup Python ${{ matrix.python-version }}
        uses: actions/setup-python@v2
        with:
          python-version: ${{ matrix.python-version }}

      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
          pip install -r requirements-dev.txt

      - name: Run black
        run: |
          black --check .
```

The end...

Questions?

ktbyers@twb-tech.com