NAPALM/ANSIBLE TUTORIAL

dbarrosop@dravetech.com

https://github.com/dravetech/napalm-ansible-tutorial

Twitter | Linkedin | Github
@dbarrosop

- **Network Systems Engineer** at Fastly
- Previously:
  - **Network Engineer** at Spotify
  - **Network Engineer** at NTT
  - **Network & Systems Engineer** at Atlas IT
- Creator of:
  - N.A.P.A.L.M.
  - SDN Internet Router

AGENDA

- Introduction to NAPALM
- Hello World
- NAPALM Modules
- Configuration Management (merge)
- Fully Automated Verification
- From a Brown to a Shiny Green Field
- Distributed `ping`

INTRODUCTION TO NAPALM

NAPALM is an API that abstracts vendors' APIs (ssh screen scraping, NETCONF, REST, RESTCONF...). NAPALM allows you to write code once and reuse it everywhere.

CONFIGURATION MANAGEMENT

- Merge Configuration
- Replace Configuration
- Compare Running/Candidate Configurations
- Commit Configuration
- Discard Configuration
- Rollback Configuration

GETTERS

- get_facts
- get_interfaces
- get_interfaces_ip
- get_lldp_neighbors
- get_bgp_config
- get_bgp_neighbors
- Many, many others

```
>>> from napalm_base import get_network_driver
>>> import pprint
>>> pp = pprint.PrettyPrinter(indent=4)
>>>
>>> junos_driver =  get_network_driver('junos')
>>> eos_driver =  get_network_driver('eos')
>>>
>>> junos_config = {
...     'hostname': '127.0.0.1',
...     'username': 'vagrant',
...     'password': '',
...     'optional_args': {'port': 12203}
... }
>>>
>>> eos_config = {
...     'hostname': '127.0.0.1',
...     'username': 'vagrant',
...     'password': 'vagrant',
...     'optional_args': {'port': 12443}
... }
>>>
```

```
>>> with junos_driver(**junos_config) as junos:
...     pp.pprint(junos.get_facts())
...
{   'fqdn': u'new-hostname',
    'hostname': u'new-hostname',
    'interface_list': ['ge-0/0/0', 'gr-0/0/0',
                       'lt-0/0/0', 'mt-0/0/0', 'vlan'],
    'model': u'FIREFLY-PERIMETER',
    'os_version': u'12.1X47-D20.7',
    'serial_number': u'5b2b599a283b',
    'uptime': 1080,
    'vendor': u'Juniper'}
>>>
>>> with eos_driver(**eos_config) as eos:
...     pp.pprint(eos.get_facts())
...
{   'fqdn': u'a-new-hostname',
            'hostname': u'a-new-hostname',
            'interface_list': [u'Ethernet1', u'Ethernet2',
                               u'Management1'],
            'model': u'vEOS',
            'os_version': u'4.16.6M-3205780.4166M',
            'serial_number': u'',
            'uptime': 1217,
            'vendor': u'Arista'}
```

```
>>> from napalm_base import get_network_driver
>>> junos_driver =  get_network_driver('junos')
>>> eos_driver =  get_network_driver('eos')
>>>
>>> junos_conf = {
...     'hostname': '127.0.0.1',
...     'username': 'vagrant',
...     'password': '',
...     'optional_args': {'port': 12203}
... }
>>>
>>> eos_conf = {
...     'hostname': '127.0.0.1',
...     'username': 'vagrant',
...     'password': 'vagrant',
...     'optional_args': {'port': 12443}
... }
>>>
>>> def change_configuration(device, conf):
...     device.load_merge_candidate(config=conf)
...     print(device.compare_config())
...     device.commit_config()
...
>>>
```

```
>>> with junos_driver(**junos_conf) as junos:
...     change_configuration(
...             junos,
...             "system {host-name rtr00;}"
...     )
...
[edit system]
-  host-name hostname;
+  host-name rtr00;
>>>
>>> with eos_driver(**eos_conf) as eos:
...     change_configuration(
...             eos,
...             'hostname rtr01'
...     )
...
@@ -8,7 +8,7 @@
 !
 transceiver qsfp default-mode 4x10G
 !
-hostname localhost
+hostname rtr01
 !
 spanning-tree mode mstp
 !
>>>
```

USEFUL LINKS

- Github Repo
- Documentation
- Supported Network Operating Systems
- Getters

HELLO WORLD

1. Install necessary components
2. Configure ansible
3. Create and execute our first playbook

INSTALLATION

```
# Create python venv
virtualenv .venv

# Activate
. ./.venv/bin/activate

# Install ansible
pip install ansible

# Install napalm
pip install napalm

# Install ansible plugin
cd ansible
git clone git@github.com:napalm-automation/napalm-ansible.git
```

CONFIGURATION

hosts

```
[all]
rtr00 os=eos   host=127.0.0.1 user=vagrant password=vagrant port=12443
rtr01 os=ios   host=127.0.0.1 user=vagrant password=vagrant port=12202
rtr02 os=junos host=127.0.0.1 user=vagrant password="" port=12203

[all:vars]
ansible_python_interpreter="/usr/bin/env python" # Using python venv
```

ansible.cfg

```
[defaults]
inventory = hosts                  # Location of hosts file
library = napalm-ansible/library   # Location of napalm plugins
stdout_callback = selective        # stdout plugin (optional)
```

OUR FIRST PLAYBOOK

playbook_hello_world.yaml

```
---
- name: Get facts
  hosts: all
  connection: local                          # code is run locally
  gather_facts: no                           # don't gather facts
  tasks:
    - name: get facts from device
      napalm_get_facts:                      # NAPALM plugin
        hostname: "{{ host }}"               # start of connection parameters
        username: "{{ user }}"
        dev_os: "{{ os }}"
        password: "{{ password }}"
        optional_args:
          port: "{{ port }}"                # end of connection parameters
        filter: ['facts']                   # which NAPALM getters to use
      register: napalm_facts                # store information here
    - name: Print gathered facts
      debug:
        msg: "{{ napalm_facts.ansible_facts|to_nice_json }}"
      tags: [print_action]
```

Modules/playbooks are vendor agnostic!!!

RUNNING THE PLAYBOOK

```
ansible-playbook playbook_hello_world.yaml
```

EXERCISES

- Retrieve information related to interfaces as well

SUMMARY

1. We installed and configured napalm and ansible
2. We wrote and ran our first playbook
3. We learnt that playbooks, modules and returned data are vendor agnostic

NAPALM MODULES

`napalm_get_facts`

Retrieves information from devices using napalm getters.

`napalm_install_config`

Can load configuration on a device and replace it (full configuration), merge it (a subset of the configuration) or just return a diff (when available).

`napalm_validate`

Validate config/state. Basically, use a *getter* to gather info from the devices and verify retrieved data using simple rules.

`napalm_ping`

Execute a ping and return results

`napalm_parse_yang` (beta)

Retrieve state/configuration from a device (or backup file) and return a `YANG` model.

napalm_diff_yang (beta)

Compare two objects of the same YANG model.

`napalm_translate_yang` (beta)

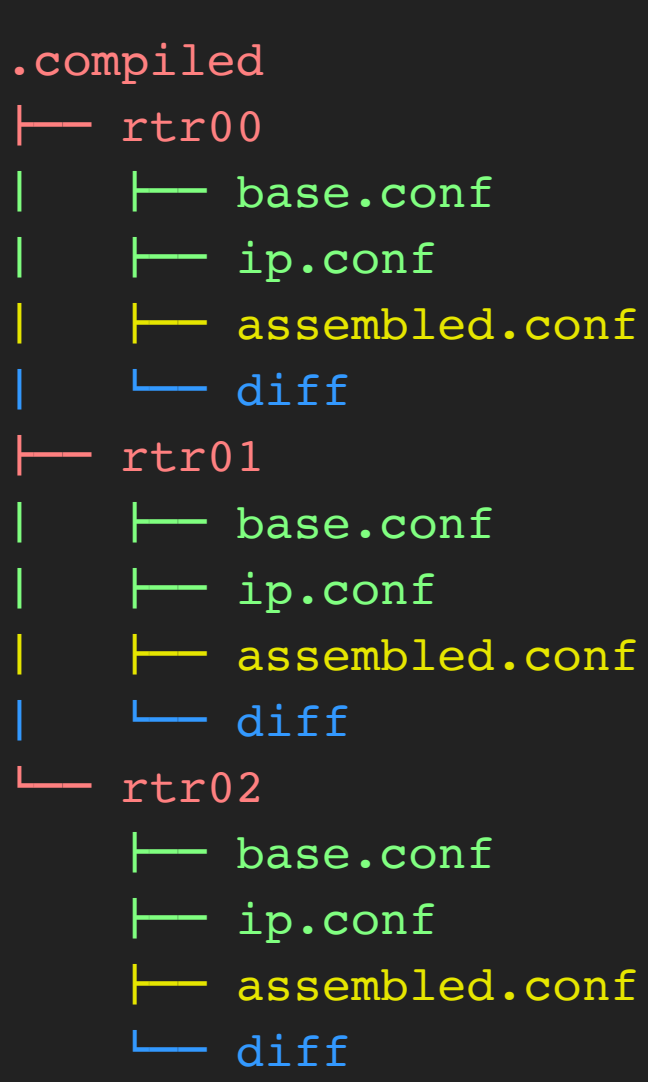
Translates a `YANG` model to native configuration.

CONFIGURATION MANAGEMENT (MERGE)

1. Configure basic system's configuration
2. Configure interfaces and IPs

WORKFLOW

1. We create a temporary directory where we will generate and store snippets of configurations and diffs
2. Different roles will generate snippets of configuration
3. We gather all the snippets and assemble them into a single file.
4. We load the configuration into the device and we commit it and/or download a diff



DATA

host_vars/rtr00

```yaml
---
interfaces:
  - name: "Loopback0"
    description: "Loopback interface"
    ips:
     - "10.0.0.100/32"
    enabled: true
  - name: "Ethernet1"
    description: "Link1"
    ips:
     - "192.168.1.100/24"
    enabled: true
  - name: "Ethernet2"
    description: "Link2"
    ips:
     - "192.168.2.100/24"
    enabled: true
```

host_vars/rtr01

```yaml
---
interfaces:
  - name: "Loopback0"
    description: "Loopback interface"
    ips:
     - "10.0.0.101/32"
    enabled: true
  - name: "GigabitEthernet2"
    description: "Link1"
    ips:
     - "192.168.1.101/24"
    enabled: true
  - name: "GigabitEthernet3"
    description: "Link2"
    ips:
     - "192.168.2.101/24"
    enabled: true
```

host_vars/rtr02

```yaml
---
interfaces:
  - name: "lo0"
    description: "Loopback interface"
    ips:
     - "10.0.0.102/32"
    enabled: true
  - name: "ge-0/0/1"
    description: "Link1"
    ips:
     - "192.168.1.102/24"
    enabled: true
  - name: "ge-0/0/2"
    description: "Link2"
    ips:
     - "192.168.2.102/24"
    enabled: true
```

group_vars/all

```yaml
---
domain: acme.com
```

PLAYBOOK (I)

playbook_config_management.yaml

```yaml
---
- name: "Simple configuration"
  hosts: all
  connection: local
  gather_facts: no
  vars:
      conf_dir: "{{ playbook_dir }}/.compiled/"
  pre_tasks:
    - name: "Assign tmp folder to host"
      set_fact:
         host_tmpdir: "{{ conf_dir}}/{{ inventory_hostname}}"
      changed_when: no   # Don't report changes
      check_mode: no     # Always make changes
    - name: "Make sure there are no remains from a previous run"
      file:
        path: "{{ host_tmpdir }}"
        state: absent
      changed_when: no   # Don't report changes
      check_mode: no     # Always make changes
    - name: "Create folder to store configurations and diffs for/from the devices"
      file:
        path: "{{ host_tmpdir }}"
        state: directory
      changed_when: no   # Don't report changes
      check_mode: no     # Always make changes
```

PLAYBOOK (II)

playbook_config_management.yaml (continuation)

```yaml
- name: "Automated Configuration"
  hosts: all
  connection: local
  roles:
    - base
    - ip
```

PLAYBOOK (III)

playbook_config_management.yaml (continuation)

```yaml
post_tasks:
  - name: "Assemble all the configuration bits"
    assemble:
        src: "{{ host_tmpdir }}/"
        dest: "{{ host_tmpdir }}/assembled.conf"
    changed_when: no   # Don't report changes
    check_mode: no     # Always make changes
  - name: "Load configuration into the device"
    napalm_install_config:
      hostname: "{{ host }}"
      username: "{{ user }}"
      dev_os: "{{ os }}"
      password: "{{ password }}"
      optional_args:
        port: "{{ port }}"
      config_file: "{{ host_tmpdir }}/assembled.conf"
      commit_changes: "{{ not ansible_check_mode }}"
      replace_config: false
      get_diffs: true
      diff_file: "{{ host_tmpdir }}/diff"
    tags: [print_action]
```

ROLES

roles layout (ip)

```
roles
└── ip
    ├── tasks
    │   └── main.yaml
    └── templates
        ├── eos
        │   └── ip.j2
        ├── ios
        │   └── ip.j2
        └── junos
            └── ip.j2
```

roles/ip/tasks/main.yaml

```yaml
---
- name: Interfaces/IP configuration
  template:
    src: "{{ os }}/ip.j2"
    dest: "{{ host_tmpdir }}/ip.conf"
  changed_when: no
  check_mode: no
```

TEMPLATES

roles/ip/templates/{ios,eos}/ip.j2

```
{% for i in interfaces %}

interface {{ i.name }}
  description {{ i.description }};
  {{ 'no switchport' if not i.name.startswith("Lo") else "" }}
  {% for ip in i.ips %}
  ip address {{ ip }} {{ "secondary" if loop.index0 else "" }}
  {% endfor %}
  {{ "no" if i.enabled else "" }} shutdown
{% endfor %}
```

roles/ip/templates/junos/ip.j2

```
{% for i in interfaces %}

interfaces {
  {{ i.name }} {
    description "{{ i.description }}";
    unit 0 {
      family inet {
        {% for ip in i.ips %}
        address {{ ip }};
        {% endfor %}
      }
    }
    {{ "disable;" if not i.enabled else "" }}
  }
}

{% endfor %}
```

DEPLOYING THE CONFIGURATION

```
ansible-playbook playbook_config_management.yaml -C
ansible-playbook playbook_config_management.yaml
```

SEMI-AUTOMATED VERIFICATION (PLAYBOOK)

playbook_semi_automated_verification.yaml

```yaml
---
- name: Get facts
  hosts: all
  connection: local
  gather_facts: no
  tasks:
  - name: get facts from device
      napalm_get_facts:
        hostname: "{{ host }}"
        username: "{{ user }}"
        dev_os: "{{ os }}"
        password: "{{ password }}"
        optional_args:
          port: "{{ port }}"
        filter:
          - 'facts'
          - 'interfaces'
          - 'interfaces_ip'
      register: napalm_facts
    - name: Facts
      debug:
        msg: "{{ napalm_facts.ansible_facts|to_nice_json }}"
      tags: [print_action]
```

SEMI-AUTOMATED VERIFICATION (COMMAND)

```
ansible-playbook playbook_semi_automated_verification.yaml
```

EXERCISES

Wait for the next module ;)

SUMMARY

- We learnt how to use NAPALM to start doing some simple configuration changes
- We also leveraged NAPALM and Ansible to avoid having to connect to the CLI for simple manual verifications

We are yet to understand the risks and challenges of merging configuration!

FULLY AUTOMATED VERIFICATION

Use `napalm_validate` module to automatically validate changes after doing configuration deployment

WORKFLOW

1. Deploy configuration as in the previous module
2. Generate validation rules off the same data that was used to generate configurations
3. Connect to the device, gather data and compare it against the validation rules

VALIDATION PLAYBOOK

includes/validate.yaml

```yaml
---
- name: "Generate validation rules"
  template:
    src: "validate.j2"
    dest: "{{ host_tmpdir }}/validate.yaml"
  changed_when: no
  check_mode: no
- name: "Read validation rules, gather data and verify it against rules"
  napalm_validate:
    hostname: "{{ host }}"
    username: "{{ user }}"
    dev_os: "{{ os }}"
    password: "{{ password }}"
    optional_args:
      port: "{{ port }}"
    validation_file: "{{ host_tmpdir }}/validate.yaml"
  register: validation
- name: "Failed Compliance Report"
  fail:
    msg: "{{ validation.compliance_report|to_nice_json }}"
  when: not validation.compliance_report.complies
- name: "Compliance report"
  debug:
    msg: "Complies"
  when: validation.compliance_report.complies
  tags: [print_action]
```

VALIDATION RULES TEMPLATE

includes/templates/validate.j2

```
---
- get_facts:
    hostname: {{ inventory_hostname }}
    fqdn: {{ inventory_hostname }}.{{ domain }}
- get_interfaces:
{% for i in interfaces %}
    {{ i.name }}:
        description: {{ i.description }}
        is_enabled: {{ i.enabled }}
- get_interfaces_ip:
{% for i in interfaces %}
    {{ i.name }}{{ ".0" if os == "junos" else "" }}:
      ipv4:
        _mode: strict
{% for ip in i.ips %}
        {{ ip.split("/")[0] }}:
          prefix_length: {{ ip.split("/")[1] }}
{% endfor %}
{% endfor %}
{% endfor %}
```

host_vars/rtr00 (for reference)

```
---
interfaces:
  - name: "Loopback0"
    description: "Loopback interface"
    ips:
     - "10.0.0.100/32"
    enabled: true
  - name: "Ethernet1"
    description: "Link1"
    ips:
     - "192.168.1.100/24"
    enabled: true
  - name: "Ethernet2"
    description: "Link2"
    ips:
     - "192.168.2.100/24"
    enabled: true
```

GENERATED VALIDATION RULES

rtr00/validate.yaml

```yaml
---
- get_facts:
    hostname: rtr00
    fqdn: rtr00.acme.com
- get_interfaces:
    Loopback0:
        description: Loopback interface
        is_enabled: True
    Ethernet1:
        description: Link1
        is_enabled: True
    Ethernet2:
        description: Link2
        is_enabled: True
```

rtr00/validate.yaml (continuation)

```yaml
- get_interfaces_ip:
    Loopback0:
      ipv4:
        _mode: strict
        10.0.0.100:
          prefix_length: 32
    Ethernet1:
      ipv4:
        _mode: strict
        192.168.1.100:
          prefix_length: 24
    Ethernet2:
      ipv4:
        _mode: strict
        192.168.2.100:
          prefix_length: 24
```

POST-ACTION HOOK

playbook_fully_automated_verification.yaml (post_tasks)

```
post_tasks:
  - name: Assemble all the configuration bits
    assemble:
      src: "{{ host_tmpdir }}/"
      dest: "{{ host_tmpdir }}/assembled.conf"
    check_mode: no
    changed_when: no
  - name: Load configuration into the device
    napalm_install_config:
      hostname: "{{ host }}"
      username: "{{ user }}"
      dev_os: "{{ os }}"
      password: "{{ password }}"
      optional_args:
        port: "{{ port }}"
      config_file: "{{ host_tmpdir }}/assembled.conf"
      commit_changes: "{{ not ansible_check_mode }}"
      replace_config: false
      get_diffs: true
      diff_file: "{{ host_tmpdir }}/diff"
    tags: [print_action]
  - include: includes/validate.yaml
    when: not ansible_check_mode
```

COMMAND

```
ansible-playbook playbook_fully_automated_verification.yaml
```

EXERCISES

- Disable and enable interfaces
- Add and remove an IP address on any device

SUMMARY

- We built a system to automatically verify deployments
- Thanks to the automatic validation we learnt some things didn't work as expected
- Merging configuration is challenging due to the stateless nature of the system

Potential solutions to overcome our merge issues are to build an adhoc system that runs different playbooks based on events (congratz, you just built a CLI 2.0) or tell the device what you exactly want and get rid of the rest

FROM A BROWN TO A SHINY GREEN FIELD

Set the foundation to fully manage our device with ansible allowing us to do a **replace** rather than a merge.

WORKFLOW

- Each device will have a dedicated "unmanaged" configuration file the will contain "static" configuration
- A brownfield role will be responsible of adding that snippet of configuration to the device's temporary folder
- Existing playbook will add this role and instruct napalm to replace configuration
- Device will ensure only desired configuration is present and remove the rest

```
├── roles
|   ├── base
|   ├── ip
|   └── brownfield
|       └── tasks
|       └── main.yaml
└── unmanaged
    ├── rtr00
    |   └── brown.static
    ├── rtr01
    |   └── brown.static
    └── rtr02
        └── brown.static
```

UNMANAGED CONFIGURATION

unmanaged/rtr00/brown.static

```
event-handler dhclient
 trigger on-boot
 action bash sudo /mnt/flash/initialize_ma1.sh
!
transceiver qsfp default-mode 4x10G
!
spanning-tree mode mstp
!
aaa authorization exec default local
!
aaa root secret sha512 $6$tIAmhTRMpbM5b7hH$MhTVW04xJbdXH871BYyT4WAFnV5q.cxmi2NWK8DhIKt60u45eDDrgeZX
!
username admin privilege 15 role network-admin secret sha512 $6$8qFGjMSK.H2fIrtr$VUQDDIqab.aMssxbXe
username vagrant privilege 15 role network-admin secret sha512 $6$u9Umd2O3S/Egu473$IHIs3nNahQ3elB4P
!
interface Management1
 ip address 10.0.2.15/24
!
no ip routing
!
management api http-commands
 no shutdown
```

Note: Long term objective would be to make sure this file is empty although it can be used for overrides

ROLE

roles/brownfield/tasks/main.yaml

```yaml
---
- name: Configuration not yet automated
  copy:
    src: "unmanaged/{{ inventory_hostname }}/brown.static"
    dest: "{{ host_tmpdir }}/z_brown.conf"
  changed_when: no
  check_mode: no
```

PLAYBOOK

playbook_config_management_replace.yaml

```yaml
- name: Automated Configuration
  hosts: all
  connection: local
  roles:
    - base
    - ip

- name: Unamanged Configuration
  hosts: all
  connection: local
  roles:
    - brownfield
```

Note: Rest of the playbook is identical to `playbook_config_management.yaml`

RUNNING THE PLAYBOOK

```
ansible-playbook playbook_config_management_replace.yaml -C
ansible-playbook playbook_config_management_replace.yaml
```

EXERCISES

- Disable/Enable an interface on each device
- Add/Remove an IP on any device
- Add a manual change to the device via its CLI
- Add a manual change to the device via `brown.static`

SUMMARY

- We built a system that allows us to easily manage the entire configuration
- Using the "brownfield" role/technique has two advantages:
  - Forces us to track manual changes via the same system as the parts that are automated
  - Allows us to keep our system lean and stateless
- By tracking the "brownfield" role we can easily check and work on automating all the bits not yet automated
- Your compliance fellows will love it

DISTRIBUTED PING

Automate/Simplify operations

PLAYBOOK

playbook_ping.html

```
---
- name: Distributed ping
  hosts: all
  connection: local
  gather_facts: no
  tasks:
    - napalm_ping:
        hostname: "{{ host }}"
        username: "{{ user }}"
        dev_os: "{{ os }}"
        password: "{{ password }}"
        optional_args:
          port: "{{ port }}"
        destination: "{{ d }}"
      register: ping
    - name: "Ping results towards {{ d }}"
      debug:
        msg: "{{ ping|to_nice_json }}"
      tags: [print_action]
```

RUNNING THE PLAYBOOK

```
ansible-playbook playbook_ping.yaml -e d=8.8.8.8
```

SUMARY

Automation is more than configuration management!

RECAP

- We learnt how to install ansible, napalm and how to integrate them
- We learnt about napalm modules
- We built a simple configuration management system that merged configuration into an existing device. It wasn't flawless but it was better than nothing
- We also built a simple playbook to allow operators to visually inspect multiple devices without having to connect to them
- We added a post action to our playbook to verify deployments. This allowed us to verify that changes were applied as expected
- We added a "brownfield" role to track "manual" changes that allowed us to tell our devices which exact configuration we wanted instead of telling them "what commands to run"
- We built a distributed ping with ansible and napalm for the sole purpose of showing that `automation != configuration_management`

HAPPY AUTOMATION!

https://github.com/dravetech/napalm-ansible-tutorial

dbarrosop@dravetech.com