CI/CD For Networks

Myth or Reality?

Ajay Chenampara: @termien0
Gerald Dykeman: @geralddykeman

Red Hat
The idea of DevOps

Wikipedia
DevOps (a portmanteau of "development" and "operations") is a software development method that stresses communication, collaboration and integration between software developers and Information Technology (IT) professionals. DevOps is a response to the interdependence of software development and IT operations.
The idea of DevOps

Wikipedia
DevOps (a portmanteau of "development" and "operations") is a software development method that stresses communication, collaboration and integration between software developers and Information Technology (IT) professionals. DevOps is a response to the interdependence of software development and IT operations.
“In software engineering, CI/CD or CICD may refer to the combined practices of continuous integration and continuous delivery and/or continuous deployment.”

--Wikipedia
“In software engineering, CI/CD or CICD may refer to the combined practices of continuous integration and continuous delivery and/or continuous deployment.”

--Wikipedia

“In software engineering, CI/CD or CICD may refer to the combined practices of continuous integration and continuous delivery and/or continuous deployment.”

--Wikipedia

"In software engineering, CI/CD or CICD may refer to the combined practices of continuous integration and continuous delivery and/or continuous deployment."

---Wikipedia
A CI/CD Walkthrough for a network change

1. When a new change to network Infrastructure as Code(IaC) repo is available, merge it to the main branch of the IaC repo
2. Generate device/vendor specific configuration out of this code
3. Deploy this configuration (entire network config and not just the partial config) “somewhere”
4. Deploy the application infrastructure on top of this infrastructure
Test....

......that the new configurations

★ Did not break the app
★ Did not impact existing High Availability(HA)
★ Did not impact performance
★ Did in fact achieve the end goal
Testing
Testing on staged production (or a representative subset)

Pros:
→ Great for validating HA, convergence, failures
→ Ensures hardware/software compatibility with production
→ Allows for testing one-off, ‘significant’ changes with confidence

Cons:
→ Cost
→ Configuration overhead
→ Potential manual overhead
Testing on virtualized devices

Pros:
- Great for configuration linting
- Cost

Cons:
- Impossible to properly test throughput & convergence in a virtual environment
- Hardware/software differences with virtual devices
- Manual gate for many virtual devices (bootstrapping, initial setup, etc...)
- Many vendors do not offer 'true' virtual devices
Network hardware is built on this premise: does hardware support feature X and does software take advantage of it? Therefore, software based devices/virtual machines are:

➔ Greate for management plane simulation
➔ Good for control plane
➔ Ok for very little data plane purposes.

Some examples:

- Not always able to test production versions on VMs
- No way to test in-line security appliances (bump in the wire)
- No way to map physical modules/ports to match production.
- No way to test hardware specific features:
  - 802.1x, Traffic Engineering, QinQ, QoS
- No way to test anything that requires buffer optimizations
- No way to test impact to application performance
- No way to test for link failures
- No way to replicate live production flows (Policy based routing)
Testing on production devices?

Pros:
➔ Realistic

Cons:
➔ Production!

However....

- Engineers have been doing it for ever
- Smarter tests
- Smaller changes more often over large changes every X weeks/months
- Controlled, periodic failure tests
Testing on production devices?

... Don’t do it*

* Understand the limitations. Limit the scope
Smarter testing: Linting the automation artifacts

Yamllint

```bash
(myvenv) ~/P/m/3/playbooks >>> ansible-lint f5_init.yml
[206] Variables should have spaces before and after: {{ var_name }}
f5_init.yml:24
   provider: "{{rest}}"
```

Yanglint

```bash
(myvenv) ~/P/m/3/playbooks >>> ansible-lint f5_init.yml
[206] Variables should have spaces before and after: {{ var_name }}
f5_init.yml:32
   provider: "{{rest}}"
```

Ansible-lint

```bash
(myvenv) ~/P/m/3/playbooks >>> ansible-lint f5_init.yml
[206] Variables should have spaces before and after: {{ var_name }}
f5_init.yml:38
   provider: "{{rest}}"
```

Molecule

```bash
(myvenv) ~/P/m/3/playbooks >>> ansible-lint f5_init.yml
[206] Variables should have spaces before and after: {{ var_name }}
f5_init.yml:24
   provider: "{{rest}}"
```

Black

Flake8

Etc....
Smarter testing: Linting configuration

[ec2-user@ansible tmp]$ ansible-playbook network_system.yml -v --check --limit rtr1
Using /home/ec2-user/.ansible.cfg as config file
/home/ec2-user/hosts did not meet host_list requirements, check plugin documentation if this is unexpected
/home/ec2-user/hosts did not meet script requirements, check plugin documentation if this is unexpected

PLAY [CONFIGURE SYSTEM CONFIGURATIONS] ********************************************************************

TASK [CONFIGURE THE HOSTNAME AND DOMAIN NAME] ********************************************************************

changed: [rtr1] => {"ansible_facts": {"network_os": "ios"}, "changed": true, "commands": ["ip domain name example.net", "ip domain list example.net", "ip name-server 8.8.8.8", "ip name-server 8.8.4.4"]}

TASK [CONFIGURE HOST LOGGING] ************************************************************************************

skipping: [rtr1] => {"changed": false, "skip_reason": "Conditional result was False"}

TASK [CONFIGURE SNMP] **********************************************************************************************

changed: [rtr1] => {"banners": {}, "changed": true, "commands": ["snmp-server community ansible-public RO", "snmp-server private-community ansible-private RW"], "updates": ["snmp-server community ansible-public RO", "snmp-server private-community ansible-private RW"]}

PLAY RECAP ********************************************************************************************************

rtr1 : ok=2 changed=2 unreachable=0 failed=0

[ec2-user@ansible tmp]$
Smarter testing: Listing configuration

```
[ec2-user@ansible tmp]$ ansible-playbook network_system.yml -v --limit rtr1

Using /home/ec2-user/.ansible.cfg as config file
/home/ec2-user/hosts did not meet host_list requirements, check plugin documentation if this is unexpected
/home/ec2-user/hosts did not meet script requirements, check plugin documentation if this is unexpected

PLAY [CONFIGURE SYSTEM CONFIGURATIONS] **********************************************
TASK [CONFIGURE THE HOSTNAME AND DOMAIN NAME] **************************************
changed: [rtr1] => ({"ansible_facts": {"network_os": "ios"}, "changed": true, "commands": [{"ip domain name example.net", "ip domain list example.net", "ip name-server 8.8.8.8", "ip name-server 8.8.4.4"}]

TASK [CONFIGURE HOST LOGGING] ******************************************************
skipping: [rtr1] => ({"changed": false, "skip_reason": "Conditional result was False"}

TASK [CONFIGURE SNMP] *************************************************************
An exception occurred during task execution. To see the full traceback, use --vvv. The error was: rtr1(config)
fatal: [rtr1]: FAILED! => ({"changed": false, "module_stdout": "Traceback (most recent call last):
File "/home/ec2-user/.ansible/tmp/ansible-local-29655110vXk/ansible-tmp-1557251316.62-34108056203227/Ansible_zios_config.py", line 173, in <module>\n   _ansibleallz_main()\n File "/home/ec2-user/.ansible/tmp/ansible-local-29655110vXk/ansible-tmp-1557251316.62-34108056203227/Ansible_zios_config.py", line 105, in _ansibleallz_main\n   invoke_module(zipped_mod, temp_path, ANSIBLEALLZ_PARAMS)\n File "/home/ec2-user/.ansible/tmp/ansible-local-29655110vXk/ansible-tmp-1557251316.62-34108056203227/Ansible_zios_config.py", line 48, in invoke_module\n   imp.load_module('_main__', mod, module, MOD_DESC)\n File "/tmp/ansible_zios_config_payload_PvDHYL/__main__.py", line 541, in <module>\n   File "/tmp/ansible_zios_config_payload_PvDHYL/__main__.py", line 472, in main\n   File "/tmp/ansible_zios_config_payload_PvDHYL/__main__.py", line 333, in edit_config_or_macro\n File "/tmp/ansible_zios_config_payload_PvDHYL/ansible_zios_config_payload.py", line 173, in __rpc__\n   ^\n   Invalid input detected at '^' marker.\n   "module_stdout": "", "msg": "MODULE FAILURE\nSee stderr for the exact error", "re"})

to retry, use: --limit @/tmp/network_system.retry

PLAY RECAP **********************************************************************

rtr1                        : ok=1  changed=1  unreachable=0  failed=1
```

Smarter testing: Testing policy artifacts

- Engineers have been doing it for ever
- Smarter tests
- Smaller changes more often over large changes every X weeks/months
- Controlled, periodic failure tests

Tools: iperf/jperf, icmp, command-line tools like curl/wget, Ansible assertions

```yml
---
name: "POLICYXYZ.123: Validates that the same community is not defined for both read-only and read-write."
- delegate_to: localhost
- no_log: yes
- assert:
  - that:
    - snmp_ro_community not in snmp rw
  loop: "{{ snmp.ro }}"
  loop_control:
    loop_var: snmp rw community
---
name: "POLICYXYZ.123MW: Validates strength of read-only communities"
- delegate_to: localhost
- no_log: yes
- assert:
  - that:
    # At least 10 characters long
    - snmp ro community is match('(?=.{10,}).*')
    # At least one lower-case
    - snmp ro community is match('(?=.*[a-z]).*')
    # At least one digit
    - snmp ro community is match('(?=.*[0-9]).*')
    # At least one upper-case
    - snmp ro community is match('(?=.*[A-Z]).*')
  loop: "{{ snmp.ro }}"
  loop_control:
    loop_var: snmp ro community
```
Addressing Process
GitOps as a process alternative

Requirements:
- Understanding version control
- Unlearn
(If large open source projects can be successful...)
Adopt branching strategies

master: Latest stable.
full: Entire topology.
site-X: Partial topology.
issue-X: Standard fix.
hotfix-X: Urgent fix.

Tags are immutable

PROTECTED BRANCHES
YOU HAVE NO POWER HERE!
Infrastructure as Code
SPECIFICATIONS + IMPLEMENTATION = DESIRED CONFIG
Addressing Communications
Smarter communications: ChatOps

- ChatOps
  - Git hooks
  - Automation tests
  - Chat enabled troubleshooting
Smarter communications: GitOps
Smarter communications: GitOps
Looking forward
Opinion

- Some vendors have embraced the open source model and are emulating the success of Linux
- Proprietary vendors systems are trending towards being more open (Shell access)
- Mathematical models / Formal methods of testing
- Status of decoupling the control and data plane
A note on formal methods for testing network infrastructure

Promising/exciting development in our industry space. At a high level these tools take in configuration (and some even take in operational state) of the network and builds a mathematical representation of the network. In theory this allows operators to simulate network changes in a more realistic way.

Reasons to be excited about this technology:
- More realistic than virtual machines to simulate complex topologies
- More cost effective than replicating physical devices
- Ability to represent different vendor OS versions

Potential roadblocks(while the technology evolves):
- Learning curve
- Keeping up with hardware vendor features/bugs
- Potential limitations to simulate hardware features/bugs
- Potential Service Level Agreement(SLA) ambiguity
In conclusion...
Takeaways

- Continuous Integration as defined is not a practical reality today for network operators
- Address Infrastructure as Code; leverage version control
- Adapt nimble processes and communications
- Take advantage of available tools and resources to move towards smarter testing
- Stay open to emerging and alternative testing methods
Thank you