Network Automation at Oracle+Dyn

NANOG on the Road
Boston, 14 Sept 2017

Carlos Vicente
We’ve come a long way

- January 2014: 18 sites and a few hundred devices with configurations manually crafted for years
- Copy/paste errors
- Consistently inconsistent :) 
- Too many people typing CLI commands
Some key accomplishments

- 6 sites redone in 2016 with 100% of configs generated, tested and deployed using automation
- Legacy sites partially maintained using the same system
- CLI interaction now occasional and only by NetEng team
History

- We knew Juniper had good support for NETCONF
- We wanted to use templates
- Chef used for servers, but we wanted “push” instead of “pull” model
- Considered writing our own code
- Ansible attractive due to its simplicity
  - Includes support for templates (Jinja)
  - Juniper had just written NETCONF modules for Ansible
Project “Kipper” is born

- Continuous integration approach to configuration management
  - Treat configurations as code (build, test, deploy)
- Leverage existing tools
Organization

- Inventory
  - All devices grouped by function, location, etc.
- Variables
  - Applied to groups or individual nodes
- Roles
  - Tie groups to templates and variables
  - Common or by function (edge routers, firewalls, etc.)
Dynamic Inventory

- Python script passed to Ansible that loads a list of devices and creates groups:
  - Based on naming convention
    - Site (US-NBN1, JP-TYO1, etc.)
    - Function (Edge, Spine, ToR, etc.)
    - Intersections of these
  - Based on model (MX, EX, etc)
- Also able to assign variables to hosts and groups
Other Variables (YAML)

**group_vars/**
- all.yml
- ams.yml
- iad.yml
- edge.yml

**host_vars/**
- edge-01-ams.yml
- vpn-01-iad.yml
To put tabular data,
Templates

● Ansible uses Jinja2
  ○ Configuration text with embedded code (Python)
    ■ Conditionals, loops, etc.

● XML format
  ○ Better support across versions of JunOS
  ○ But also allows for advanced checks
    ■ Easy to parse and run checks on it
Template example
Test playbook

- Take each configuration file and perform a *dry-run* using NETCONF
  - aka *commit-check* in JunOS
  - Gather *diffs* from each device
    - or report syntax errors
  - Combine *diffs* to create a pretty *Gist*
  - Send Gist URL to net admins via Slack
Deploy playbook

- Sends configs to all devices
  - If there are changes, commits those
  - If there are no changes, device is unaffected
- Notifies NOC

Kipper 4:45 PM
@ilejeune is deploying configurations to network devices now. Scope: [masked]. See #netdiff
Someone is making a change

github BOT 9:31 AM ✨
[Network/kipper] Pull request submitted by shulshof
#359 Add routes to support oob1 in-band access

jenkins BOT 9:31 AM
prb_kipper - #251 GitHub pull request #359 of commit
6af71f8ceed57e021d3d8fbd4b86bf60454623e3, no merge conflicts. (Open)

prb_kipper - #251 Starting... after 1.2 sec and counting (Open)

Kipper BOT 9:42 AM
Dry-run #251 results available here for your review

jenkins BOT 9:42 AM
prb_kipper - #251 Success after 11 min (Open)
Kipper dry-run #103 results

```plaintext
[edit groups]
  TRUNK_INTERNET { ... }
  ! AE_INTERFACES { ... }
[edit]
  - apply-groups [ ROUTING_INSTANCES RE_PROTECT_V4 RE_PROTECT_V6 AE_INTERFACES ];
  + apply-groups [ ROUTING_INSTANCES AE_INTERFACES RE_PROTECT_V4 RE_PROTECT_V6 ];
[edit interfaces ae8]
  - mtu 1514;

[edit]
  - apply-groups [ ROUTING_INSTANCES RE_PROTECT_V4 RE_PROTECT_V6 AE_INTERFACES ];
  + apply-groups [ ROUTING_INSTANCES AE_INTERFACES RE_PROTECT_V4 RE_PROTECT_V6 ];
[edit interfaces ae8]
  - mtu 1514;
```
Did we break anything?

PLAY [Reachability tests] ******************************************************

TASK [Ping test] ******************************************************
ok: [tor104a.us-xyz1] => (item={u'src_ip': u'198.168.145.195', u'dst_ip': u'10.20.112.130', u'src_ri': u'PUBLIC', u'descr': u'From tor108b.us-xyz1 RI 1200 to tor102a.us-zzz1 RI 1300'})
ok: [tor104b.us-xyz1] => (item={u'src_ip': u'10.20.49.131', u'dst_ip': u'10.20.112.130', u'src_ri': u'PRIVATE', u'descr': u'From tor104b.us-xyz1 RI 1300 to tor102a.us-zzz1 RI 1300'})
ok: [tor108a.us-xyz1] => (item={u'src_ip': u'198.168.145.194', u'dst_ip': u'10.20.128.2', u'src_ri': u'PUBLIC', u'descr': u'From tor108a.us-xyz1 RI 1200 to tor102a.hk-abcl RI 1300'})
ok: [tor104a.us-xyz1] => (item={u'src_ip': u'10.20.49.130', u'dst_ip': u'10.20.128.2', u'src_ri': u'PRIVATE', u'descr': u'From tor104a.us-xyz1 RI 1300 to tor102a.hk-abcl RI 1300'})
Nightly dry-runs

jenkins  APP  8:00 PM  ⭐
scheduled_dry_run - #447 Started by timer (Open)

scheduled_dry_run - #447 Starting... after 0.71 sec and counting (Open)

Kipper  APP  8:13 PM
Dry-run #447 results available here for your review

jenkins  APP  8:13 PM
scheduled_dry_run - #447 Back to normal after 12 min (Open)
Implementation on Legacy Sites

● Can’t always reconfigure from scratch
  ○ Fixing engine while car is running

● Started simple
  ○ Covered the most common parts first:
    ■ e.g. Authentication, NTP, DNS, SNMP, common prefix lists, etc.
  ○ Worked towards 100% coverage incrementally
    ■ Slow process until everything is standardized
Implementation on new sites

- Built a model site in the lab
- Wrote templates to match working config
- Modeled the addressing plan
- Wrote code to generate the inputs
  - CSV + YAML files
- All configs generated and tested by migration date
- Then: `make deploy`
Operational changes

- Some operational changes do not merit the CI/CD process
  - Need to be done very quickly and possibly off-hours
  - Short-lived
  - How to still avoid CLI?
- Identified most common ones:
  - Block or rate-limit abusive traffic
  - Manipulate BGP announcements
Enter Flowspec

- IETF standard that allows using BGP to transmit ACL specifications
- Actions include:
  - Discarding packets
  - Rate-limiting
  - Redirecting traffic
- Also good support on JunOS
Project Flux

- Flowspec REST API
  - Python/Flask/Redis + ExaBGP
ok
ok. i'm thinking we need to block that IP it's skyrocketing, and the box is hitting FBS spikes.

Sounds good. I don't think its doing anything good

okay. i'm blocking now.

next IR will be IR-3449 (edited)

yes

flowbot addflows -name ir-3449 -sites defra1 -src_ips -dst_ips

flowbot APP 4:34 PM

DRY RUN: Run with -commit to deploy flow into production

flow name: ir-3449
neighbors: defra1
dst_ips:

flowbot APP 4:34 PM

flowbot addflows -name ir-3449 -sites defra1 -src_ips -dst_ips

flowbot APP 4:34 PM

Flowspec API Response: 201
Flow ir-3449 created
NETCONF API

- Another small Python-based REST API
- Uses Juniper’s pyEZ library for SSH-based NETCONF operations
- Only for discreet, safe operations
  - Change BGP announcements
    - Stop announcing anycast, drop provider, etc.
  - Easy to add more functions
What about upgrades?

● We recently extended the Ansible/Netconf approach to upgrades

● Playbook for ToRs:
  ○ Pre-upgrade config changes
  ○ Uploads image, waits for reboot
  ○ Reverts temp changes

● `make upgrade` - Can do many at once
Ancillary configurations

- When your tool has 100% of the config data, you can also generate:
  - Monitoring configurations (availability, metrics)
  - DNS, DHCP
  - Etc.

- Alternative is to use a separate “discovery” mechanism as inventory
Future plans

- Create virtual labs on demand to test new designs, or changes to existing designs
  - make build test clean
- More functional testing using operational state
  - We are experimenting with jsnapy tool
Future plans

- Work on a bootstrap/ZTP solution
  - When deploying a site, tech needs to install minimal JuNOS config prior to running “make install”:
    - Mgmt IP
    - User authentication
    - Enable SSH/Netconf
Questions?

Thank you