



Experiences with Network Automation at Dyn

NANOG 67

Carlos Vicente
Dyn

What is Dyn

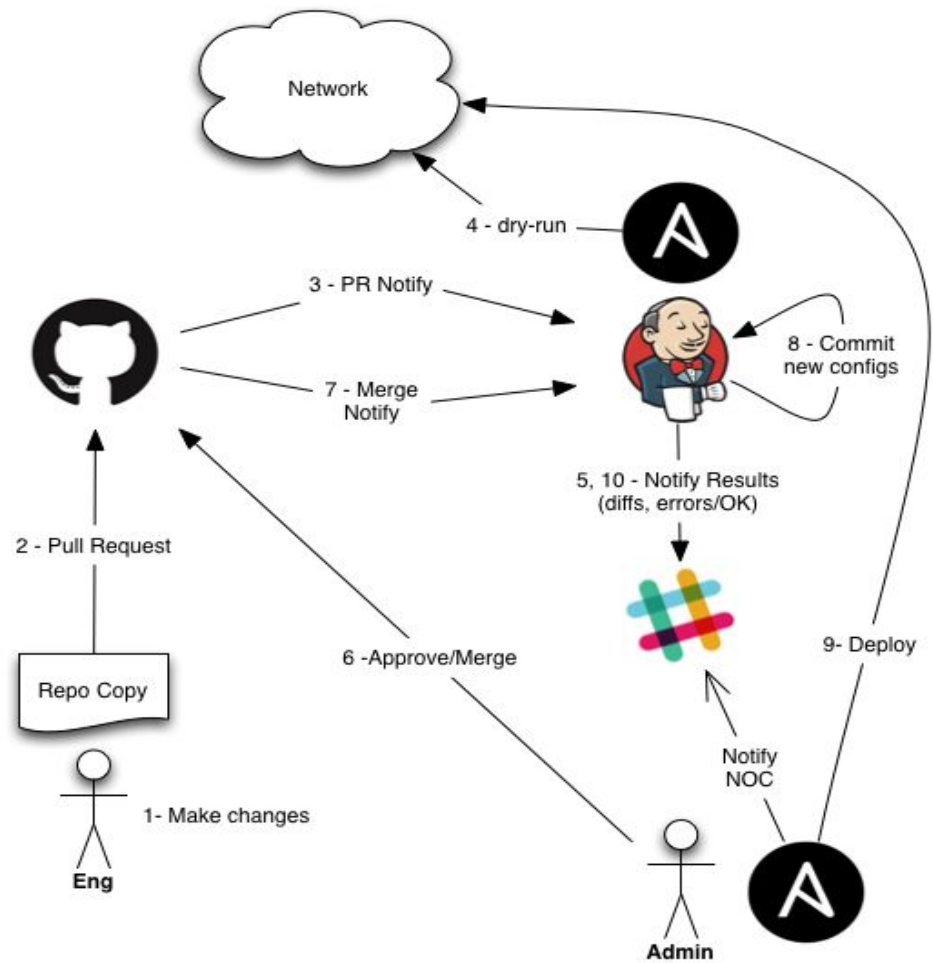
- Internet Performance (DNS, E-mail, Analytics)
- 20+ data centers
- Hundreds of network devices
- Small teams
- Automation is a priority at every level



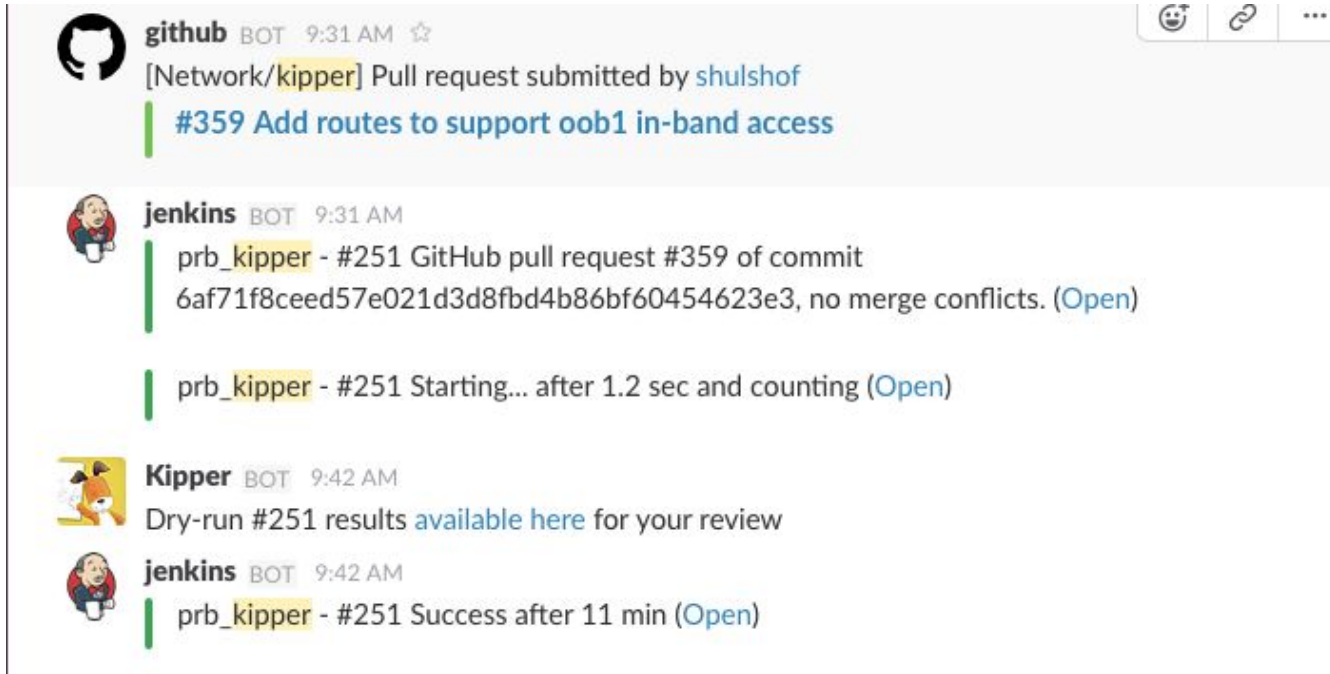
Project “Kipper”

- Continuous integration approach to configuration management
- Dyn code + open source tools
- See presentation at NANOG63
- This talk is about successes and lessons learned since then





Here comes a change



The screenshot shows a Slack channel conversation. At the top, a message from the 'github' bot at 9:31 AM states: '[Network/kipper] Pull request submitted by shulshof #359 Add routes to support oob1 in-band access'. Below this, the 'jenkins' bot at 9:31 AM reports: 'prb_kipper - #251 GitHub pull request #359 of commit 6af71f8ceed57e021d3d8fbd4b86bf60454623e3, no merge conflicts. (Open)'. The next message from 'jenkins' at 9:31 AM says: 'prb_kipper - #251 Starting... after 1.2 sec and counting (Open)'. Then, the 'Kipper' bot at 9:42 AM reports: 'Dry-run #251 results available here for your review'. Finally, the 'jenkins' bot at 9:42 AM reports: 'prb_kipper - #251 Success after 11 min (Open)'. The interface includes icons for emojis, links, and a menu in the top right of the first message.

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

```
netcfg_dry_run_103.diff Raw
1
2 =====
3 ██████████.as33517.net.diff
4
5 [edit groups]
6   TRUNK_INTERNET { ... }
7   ! AE_INTERFACES { ... }
8 [edit]
9 - apply-groups [ ROUTING_INSTANCES RE_PROTECT_V4 RE_PROTECT_V6 AE_INTERFACES ];
10 + apply-groups [ ROUTING_INSTANCES AE_INTERFACES RE_PROTECT_V4 RE_PROTECT_V6 ];
11 [edit interfaces ae0]
12 - mtu 1514;
13
14 =====
15 ██████████.as33517.net.diff
16
17 [edit groups]
18   TRUNK_INTERNET { ... }
19   ! AE_INTERFACES { ... }
20 [edit]
21 - apply-groups [ ROUTING_INSTANCES RE_PROTECT_V4 RE_PROTECT_V6 AE_INTERFACES ];
22 + apply-groups [ ROUTING_INSTANCES AE_INTERFACES RE_PROTECT_V4 RE_PROTECT_V6 ];
23 [edit interfaces ae0]
24 - mtu 1514;
```

So we've been using it and...

- It actually works :-)
- We are using Kipper to rebuild all our edge sites globally
 - HK first...halfway around the world seemed best place to start ;-)
- Everybody in NetEng uses it
- SysEng uses it when provisioning new hardware
 - NetEng is out of the port turnup game YAY!
- We learned some things, and made it better.



Some stats

- ~300 devices in inventory (and growing)
- 88 template files (Jinja2)
- 900K lines of generated configuration (XML)
- 1286 repository commits (average 4.7 commits per active day)
- 17 Authors
- Most commits on Thursdays around 4pm? ㄒ_(ツ)_ㄒ



Too many device groups

- Grouping hosts in manual inventory was not ideal

Fix

1. Use Ansible's Dynamic Inventory
 - a. Python script reads list of devices and creates groups:
 - b. Based on naming convention
 - i. Site (US-NBN1, JP-TYO1, etc.)
 - ii. Function (Edge, Spine, ToR, etc.)
 - iii. Intersections of these
 - c. Based on model (MX, EX, etc)



Too much data

- Too much data to put in YML files

Fix (example)

1. Map all interconnections in a shared spreadsheet, convert to CSV and use that to feed Ansible's inventory
2. Use subnet prefixes and calculate IPs in the script
3. CSV file is version controlled

A	B	C	D	E	F	G	H	I	J
A_Name	A_Port	Z_Name	Z_Port	Type	A_Bundle	Z_Bundle	VLAN	v4	v6
edge01.us-xyz1	ge-1/0/0	edge02.us-xyz1	ge-1/0/0	bundle_member	ae0	ae0	n/a	n/a	n/a
edge01.us-xyz1	ge-1/0/1	edge02.us-xyz1	ge-1/0/1	bundle_member	ae0	ae0	n/a	n/a	n/a
edge01.us-xyz1	ae0	edge02.us-xyz1	ae0	vrf_lite_trunk	n/a	n/a	1100	192.0.2.0/31	2001:db8:40:f008::/127
edge01.us-xyz1	ae0	edge02.us-xyz1	ae0	vrf_lite_trunk	n/a	n/a	1200	192.0.2.128/31	2001:db8:40:f208::128/127

Garbage In, Garbage Out

- Need initial data from someplace, which means a human
- IPs and interfaces come from CSV files
- Humans make mistakes, like duplicate and wrong IPs

Fix

1. Added data sanity checker to inventory script
2. IP blocks for sites are now automatically subnetted (v4/v6)
common network design = codified network design
3. Point to point and other data now generated by a script



Server Provisioning

- We do multi-chassis LAG between ToRs and new servers
- LACP needs to be “forced-up” on one switch at provisioning time
- We were temporarily changing the ports each time
- We would forget to remove the “force-up” parameter later. Bad.

Fix

1. Configure all unused ports ready for provisioning by default
2. Once server is installed, SysEng updates server port inventory and Kipper does the rest



Is everything still working?

- “I’m positive this little change won’t break anything.”
- “Are you sure that was working before my change?”
- “&*(^! OK, I’ll rollback.” :-)

- We didn’t have a good way to test connectivity both before and after changes. We have a lot of flows intra/inter sites/routing-instances.

Fix

1. Wrote Ansible playbook that uses Netconf to test server flows between sites and between routing instances
2. Added a playbook to do rollbacks in bulk



Testing with one command

```
$ make pingtest
```

```
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'})
```



Multipoint tests with Ansible

```
- name: Reachability tests
hosts: ping-nodes
connection: local
tasks:
  - name: "New Network ping tests"
    junos_ping:
      host={{ inventory_hostname }}
      user=admin
      dest_ip={{ item.dst_ip }}
      source_ip={{ item.src_ip }}
      routing_instance={{ item.src_ri }}
      rapid=yes
      count=3
      timeout=5
      acceptable_packet_loss=1
    with_items: "{{ ping_targets }}"
```



Change Awareness

- Sometimes we would forget to notify others when we pushed a change
- NOC in particular did not like this...we couldn't imagine why? :-)

Fix

1. Modify the deploy job to notify NOC and NetEng automatically



Kipper BOT 10:39 AM

@cvicente is deploying configurations to network devices now. Scope: 'hk-tko1-tsw'. See [#netdiff](#)

Unattended deployments

- We wanted to make the “push” happen automatically after a merge
- It wasn't realistic
- Sometimes a change depends on another change
 - Example: Need to change policies in edge routers, then change the spine layer, etc.

Fix

1. A human makes these decisions.
2. After a change is merged, we update our copy of the repo and run Ansible (`$ make deploy something something`)



Legacy configs

- Very hard to adapt configurations crafted by hand over years
 - Inconsistencies hard to fix while in production
 - Is this section still necessary...?

Fix

1. Focus on new sites, new designs
 - a. Fully automate from the beginning
2. Then go back and fix the old as time allows
 - a. Tackle one section at a time
 - b. Or better yet, upgrade to new design



Audits

- Someone makes a manual change. Nobody finds out until we have to push a template

Fix

1. Schedule nightly dry-runs in Jenkins
2. Send diffs to Slack channel
3. Team decides if the change needs to be overwritten, or if the templates need to be updated to reflect it



Gathering Facts

- We used Netconf to gather “facts” from each device before rendering the templates
 - If version > 12.3 include this section
 - If model == ‘SRX’ change this limit
- Did not allow us to generate and verify configs prior to new devices being online
- Added significant delay to the dry-run process

Fix

1. Keep this information in the inventory
 - a. Minor downside: need to update the inventory after a firmware upgrade



Interrupted dry-runs

- Jenkins dry-runs take a while
 - Many CPU-challenged devices take over 2 minutes to do a “commit check” ...the joys of cheap L3 switches
- Jenkins run gets interrupted
 - Device config is locked by another user on the CLI
 - A device is out of storage space, can't commit
 - Random Netconf errors that later go away
- Generally we solve the issue quickly and then tell Jenkins to do the dry-run again
- Sometimes the issue is more involved
 - Comment out the device from Ansible inventory temporarily



Challenges with Git

- Git is hard (at first)
 - “Uh oh! I didn’t create a branch”
 - Can’t interrupt and fix something else
 - Committed a change that was not accepted
 - “I merged from the wrong branch”
 - “Didn’t commit before switching to master”
 - “I messed it up, don’t know what I did...”

Fix

1. Training

Dyn inside joke: all three-letter words are hard.
Git, DNS, BGP...



But it works for me!

- Everybody makes changes on their local repo on their Mac
- Different OS versions, different library versions present challenges
- Never serious issues, but can be annoying

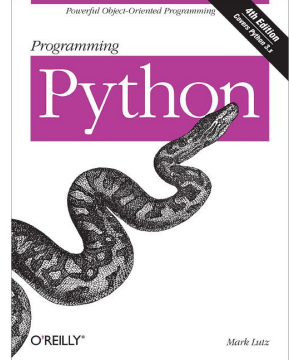
Fix

1. We're going to move to a common VM that everybody accesses remotely.



Coding

- No complex code necessary with these tools, but...
 - Still foreign for network folks without a programming background
 - Important to learn at least the basics:
 - Ansible variables, playbooks...
 - Data structures, loops, regular expressions...



Consistency, what a concept!

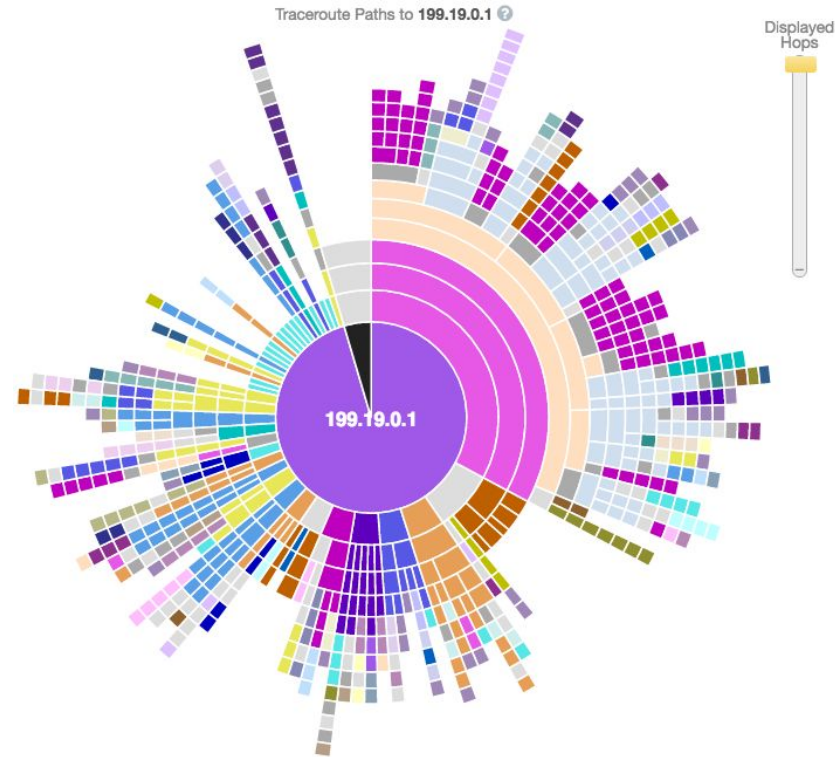
In **almost all** cases, two or more things that were supposed to be the same, were not

- Copy/paste mistakes
- “Forgot the redundant device”
- “Forgot that site”
- Kipper really helped us notice and correct these issues



The Future

- We want to test beyond our own network:
 - BGP announcements and their propagation across Internet
 - Reachability across Internet
- Solution: Use Dyn's Internet Intelligence API and integrate it in our change management



Conclusions

- Network automation is not without its challenges
 - Some technical, some organizational
- But it is an investment that pays off **significantly** in many ways



Questions?





Thank you

cvicente@dyn.com

**For more information on
Dyn's services visit dyn.com**

**INTERNET
PERFORMANCE.
DELIVERED.**

dyn.com [@dyn](https://twitter.com/dyn)