ORACLE + Dyn

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

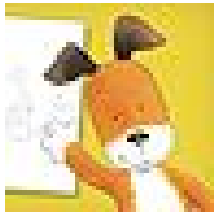
Some key accomplishments

- 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

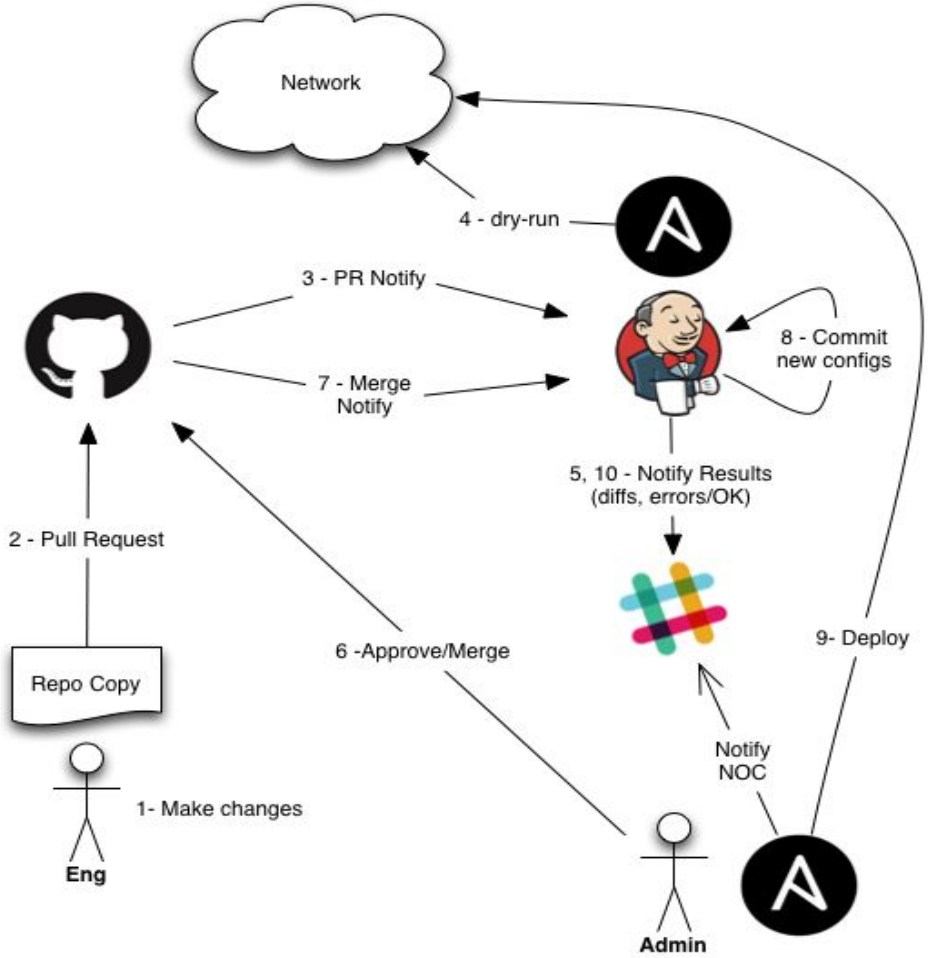


- Continuous integration approach to configuration management
  - Treat configurations as code (build, test, deploy)
- Leverage existing tools

ANSIBLE

netconf

GitHub


Network
4 - dry-run
3 - PR Notify
7 - Merge
Notify
8 - Commit
new configs
5, 10 - Notify Results
(diffs, errors/OK)
2 - Pull Request
6 -Approve/Merge
9- Deploy
Repo Copy
Notify
NOC
1- Make changes
Eng
Admin

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

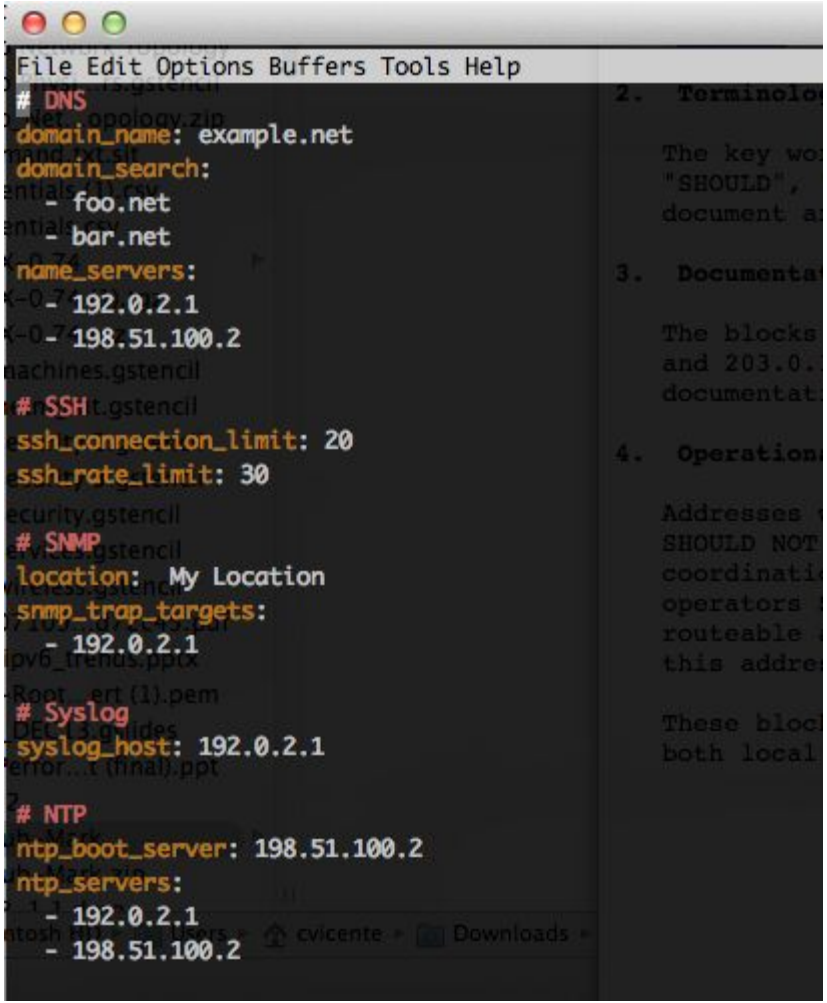
Other Variables (YAML)

```
group_vars/

    all.yml
    ams.yml
    iad.yml
    edge.yml

host_vars/

    edge-01-ams.yml
    vpn-01-iad.yml
```



Tabular Data

- Too much data to put in YML files

Take advantage of dynamic inventory, e.g.

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

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

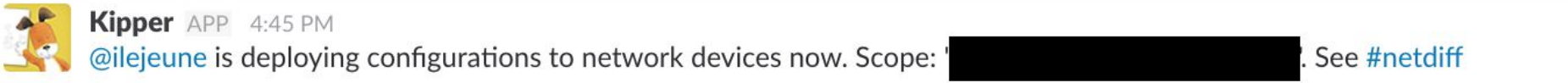
```
  <host-name>{{ host_basename }}</host-name>
  <domain-name>{{ domain_name }}</domain-name>
{% for domain in domain_search %}
  <domain-search>{{ domain }}</domain-search>
{% endfor %}
{% if backup_router is defined %}
  <backup-router>
      <address>{{ backup_router }}</address>
      <destination>0.0.0.0/0</destination>
  </backup-router>
{% endif %}
  <root-authentication>
    <encrypted-password>{{ root_password_hash }}</encrypted-password>
  </root-authentication>
{% for name_server in name_servers %}
  <name-server>
    <name>{{ name_server }}</name>
  </name-server>
{% endfor %}
```

Test playbook

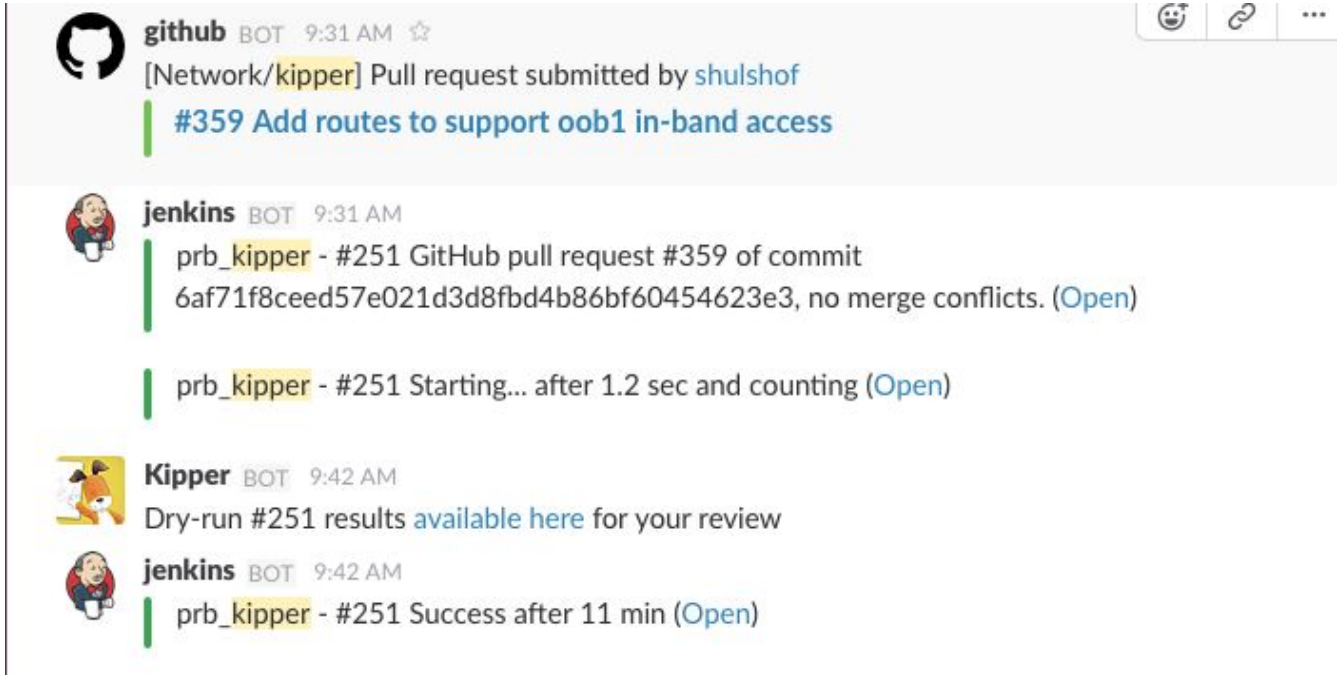
- Take each configuration file and perform a *dry-run* using NETCONF
  - aka `commit-check` in JunOS
  - Gather *diffs* from each device
    - or report syntax errors
  - Combine *diffs* to create a pretty *Gist*
  - Send Gist URL to net admins via Slack

Deploy playbook

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



Someone is making a change



netcfg / **netcfg_dry_run_103.diff** SECRET

Kipper dry-run #103 results

netcfg_dry_run_103.diff Raw

```
1
2  ================================================================
3  [redacted].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 [redacted].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;
```

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-abc1 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-abc1 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

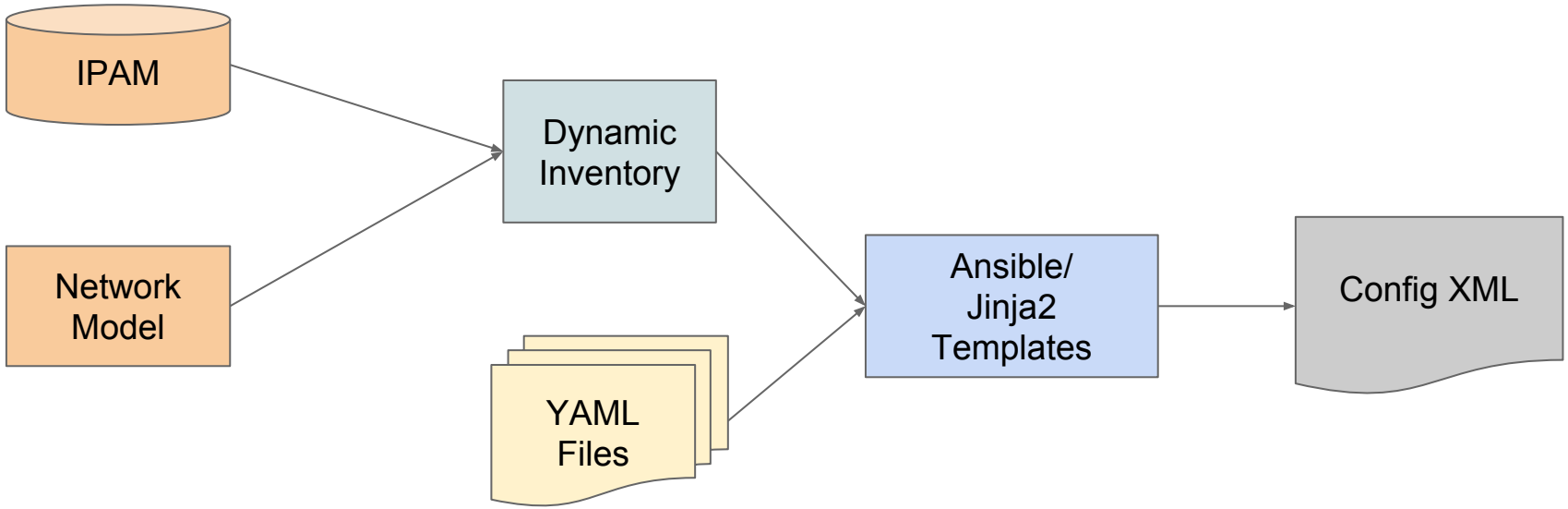
scheduled_dry_run - #447 Back to normal after 12 min (Open)

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`


IPAM
Network Model
Dynamic Inventory
YAML Files
Ansible/ Jinja2 Templates
Config XML

Operational changes

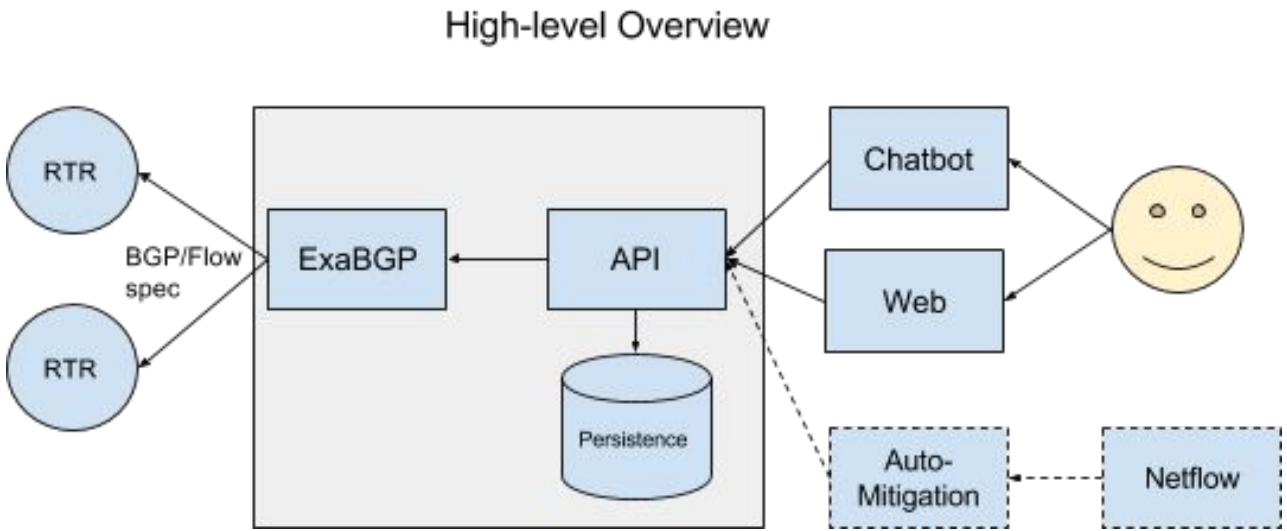
- Some operational changes do not merit the CI/CD process
  - Need to be done very quickly and possibly off-hours
  - Short-lived
  - How to still avoid CLI?
- Identified most common ones:
  - Block or rate-limit abusive traffic
  - Manipulate BGP announcements

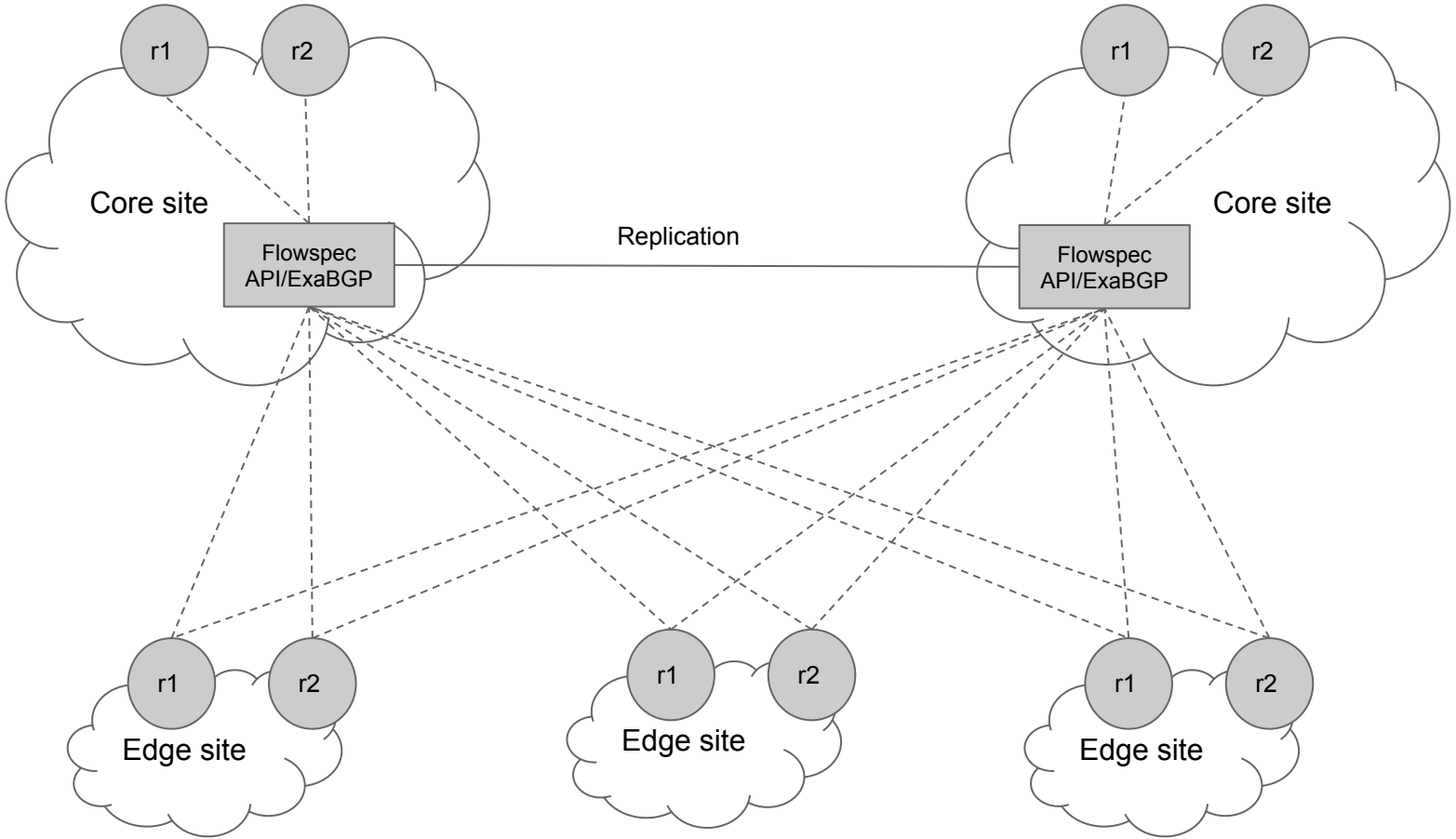
Enter Flowspec

- 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




r1
r2
Core site
Flowspec
API/ExaBGP
Replication
r1
r2
Core site
Flowspec
API/ExaBGP
r1
r2
Edge site
r1
r2
Edge site
r1
r2
Edge site

[redacted] 4:29 PM
ok

ok. i'm thinking we need to block that IP [redacted]

it's skyrocketing, and the box is hitting FBS spikes.

[redacted] 4:33 PM
Sounds good. I don't think its doing anything good

[redacted] 4:33 PM
okay. i'm blocking now.

[redacted] 4:33 PM
next IR will be IR-3449 (edited)

[redacted] 4:34 PM
yes

flowbot addflows -name ir-3449 -sites defra1 -src_ips [redacted] -dst_ips [redacted]

flowbot APP 4:34 PM

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

> **flow name**: ir-3449
> **neighbors**: defra1
> **dst_ips**:
> [redacted]
> [redacted]
>
> [redacted]

[redacted] 4:34 PM
flowbot addflows -name ir-3449 -sites defra1 -src_ips [redacted] -dst_ips [redacted] -commit

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