NETWORK AUTOMATION

A PRACTICAL APPROACH

MATT GRISWOLD
CTO, UNITED INTERNET EXCHANGE
matt@unitedix.net
WHAT WE’LL DISCUSS

- Quick ideals
- Getting your hands dirty

Questions welcome anytime
Some of the most important things in automation:

- Testing
- Logging
- Versioning
CURRENT STATE

- Lots of homebrew tools
- Networks are so drastically different, hard to do a complete one size fits all package
- Frequently "start from scratch" and just reuse scripts and templates (from a central repo)
The Unix philosophy, originated by Ken Thompson, is a set of cultural norms and philosophical approaches to minimalist, modular software development.*

*token Wikipedia copy / paste
SEPARATION OF CONCERNS

DATA

LOGIC

DEPLOYMENT
IF YOU’RE FETCHING DATA...
Fetch data and write it.

IF YOU’RE BUILDING A CONFIG...
Take existing data and build the config.

IF YOU’RE PUSHING TO A DEVICE...
Take a text file and put it on a device.
Data can be validated to be correct
Data can be reused in many different places—including some you haven't thought of yet

Logic without data in it...

Concise code
Unit test with example data
DO ONE THING AND DO IT WELL

- Easy to understand and modify code
- Easier to unit test

*Please do not integrate your network automation with systemd*
"Write programs to work together. Write programs to handle text streams, because that is a universal interface."
– Doug McIlroy

In this case, don't worry as much about streaming from one to the other, but rather writing data and reading data
EXAMPLE: BGPQ3

- Do: use bgpq3 -j to output json to a file
  - easy to validate that the data you have is correct
  - any other process can read and use that data
Do: use API to query records and write to a file
- Easy to validate that the data you have is correct
- Any other process can read and use that data
- If the API has version changes, it's easy to make changes, test only that, move on

EXAMPLE: PEERINGDB
Do: add only customer data to a data source
  ○ Later processes can all reread this data to do things
Easy way to turn data into config

```
address {{ host.loopback.ip }}/32;
```
Read from a directory—no need for one process to try to build a whole config.

config/$hostname/00-system.conf
config/$hostname/10-bgp.conf
config/$hostname/10-interfaces.conf
Separate push from config building

- Allows use of many tools to build config snippets
- Allows manual overrides if needed
- $push_config is a script that only takes pre-generated text files and puts them on a router
One-offs

- Refrain from `{if == $hostname}`,
- Instead, use `extra_config/$hostname.conf`
  - Separates logic from data
  - Keeps templates clean and simple
Small, sharp tools are easy to unit test

Take input, produce output
- script that fetches data is tiny, check data, write it
- script that uses logic to build configs
- script to push to a device, only job should be taking generated text and putting it on a machine
- easy to write multiple scripts for different devices
DEPLOYMENT

- Test on dev machine; virtual network
- Don't deploy to everything at once
- Version config and log diffs
- Human-controlled deploy—magical "automated" deploys save little time and can be disastrous
- Key auth—it's 2016, stop using passwords!
AUTOMATION ENVIRONMENTS

● Engineer-controlled
  ○ Triggered by engineer
  ○ Stores data in YAML/git
  ○ Deploy via ansible, puppet, chef

● Customer-controlled
  ○ Triggered by customer or any outside input
  ○ Stores data in a database
  ○ Deploy via custom real time software
NOTE: build_acl_config is a small reused component

- push_acl (minus logging, testing, etc)
  - Finds customer
  - Looks up switch information
  - Looks up mac address and blackhole routes
  - build_acl_config > tmpfile
  - Push tmpfile to devices
EXAMPLE: CHIX L2 ACLS

Used by:

- Engineer provisions customer, one of the steps calls the script to provision the ACL
Used by:

- Customer updates MAC address via website
  - Writes to DB
  - Then triggers
    - push_acl --asn=33713

EXAMPLE: CHIX L2 ACLS
Used by:

- Customer adds blackhole route via BGP Community
  - Bird outputs to script that updates DB
    - add_blackhole --asn=33713 127.0.0.1/32
  - Then triggers
    - push_acl --asn=33713
https://github.com/20c/ngage

- Evolved from internal tools
Usage: ngage [OPTIONS] COMMAND [ARGS]...

Commands:

  commit
  diff
  push
  rollback
  save
ngage push 00-system.conf --user=root

Prompts for password
Create a git repo

- Get a copy of your current config
  - RANCID
  - ngage save

Save as config/$hostname/00-starting.conf
bin/diff.sh

#!/bin/bash

hostname=$1
shift
HELPER SCRIPTS

if test -z "$hostname"; then
    echo "usage, $0 <hostname> [OPTIONS]"
    exit 1
fi

ngage push --diff --no-commit $hostname
gen/$hostname/* $@
ngage rollback $hostname
HELPER SCRIPTS

bin/push_edge.sh

#!/bin/bash

hosts="edge0 edge1"

for hostname in $hosts; do
  ngage push --diff $hostname gen/$hostname/* $@
done
How to Start

Play around with config.

```bash
gage push --diff --no-commit config/dev0/00-starting.conf
```

- Import all device config
- Commit
- Use favorite text editor
- Profit?
prod/group_vars/ch2/customer.yml

customer_ports:
  - name: office vlan
cust_id: 11230
ports:
  - vlan_id: 1230
prefixes:
  - 10.243.122.0/29
switch: agg0
intf: ge-0/0/2
{% for cust in customer_ports %}
{% for port in cust.ports %}
{% if inventory_hostname_short == port.switch | default() %}
{% do cust_vlans_made.append(port.vlan_id) %}

interfaces {
replace:
{{port.intf}} {
  description "Cust: {{cust.name}} ID{{cust.cust_id}}";
  unit 0 {
    family ethernet-switching {
      interface-mode access;
      vlan {
        members {{port.vlan_id}};
      }
    }
    storm-control cust_default;
  }
}
}
ACCESS SWITCH

{% for intf in intf_to_core | default() %}
  interfaces {
    {{intf}} {
      unit 0 {
        family ethernet-switching {
          vlan {
            members [ {{cust_vlans_made | join(' ')}} ];
          }
        }
      }
    }
  }
{% endfor %}
{% for cust in customer_ports %}
  {% for port in cust.ports if port.prefixes is defined %}
    cust-{{cust.ncid}}-{{port.vlan_id}} {
      description "{{cust.name}} ID{{cust.ncid}}";
      vlan-id {{port.vlan_id}};
      routing-interface irb.{{port.vlan_id}};
    }
  
{% endfor %}
{% endfor %}
{% for cust in customer_ports %}
  {% for port in cust.ports %}
    policy-options {
      prefix-list cust-{{cust.cust_id}}-{{port.vlan_id}}_allowed {
        {% for ip in port.prefixes | default() %}
          {{ip}};
        {% endfor %}
      }
    }
  {% endfor %}
{% endfor %}
firewall {
  family inet {
    filter cust-{{cust.cust_id}}-{{port.vlan_id}}-in {
      term prefixes {
        from {
          prefix-list {
            Cust-{{cust.cust_id}}-{{port.vlan_id}}_allowed;
          }
        }
        then accept;
      }
    }
  }
}
unit {{port.vlan_id}} {
    description "Cust: {{cust.name}} ID{{cust.cust_id}}";
    family inet {
        {% for ip in port.prefixes %}
        {% if ip | ipaddr('prefix') == 31 %}
            address {{ip | ipaddr('0')}};
        {% else %}
            address {{ip | ipaddr('1')}};
        {% endif %}
        {% endfor %}
    }
}

EDGE ROUTER
address {{ip | ipaddr(vrrp_idx)}} {
    vrrp-group 1 {
        virtual-address {{ip | ipaddr(1)});
        priority {{102 - vrrp_idx}};
        advertise-interval 1;
        authentication-type simple;
        authentication-key "$9$SECRETYO";
    }
}
group customer {
    type external;
    {% for peer in bgp.group.customer.neighbor %}
    replace:

    neighbor {{peer.ipv4}} {
        import as{{peer.asn}}-in;
        family inet {
            any {
                prefix-limit {
                    maximum {{peer.max_prefix}};
                    teardown;
                }
            }
        }
        export as{{peer.asn}}-out;
        peer-as {{peer.asn}};
    }
    {% endfor %}
}
QUESTIONS / COMMENTS?
matt@unitedix.net
https://github.com/inex/IXP-Manager
https://github.com/20c/django-ixpmgr
NETWORK AUTOMATION

A PRACTICAL APPROACH

MATT GRISWOLD
CTO, UNITED INTERNET EXCHANGE
matt@unitedix.net