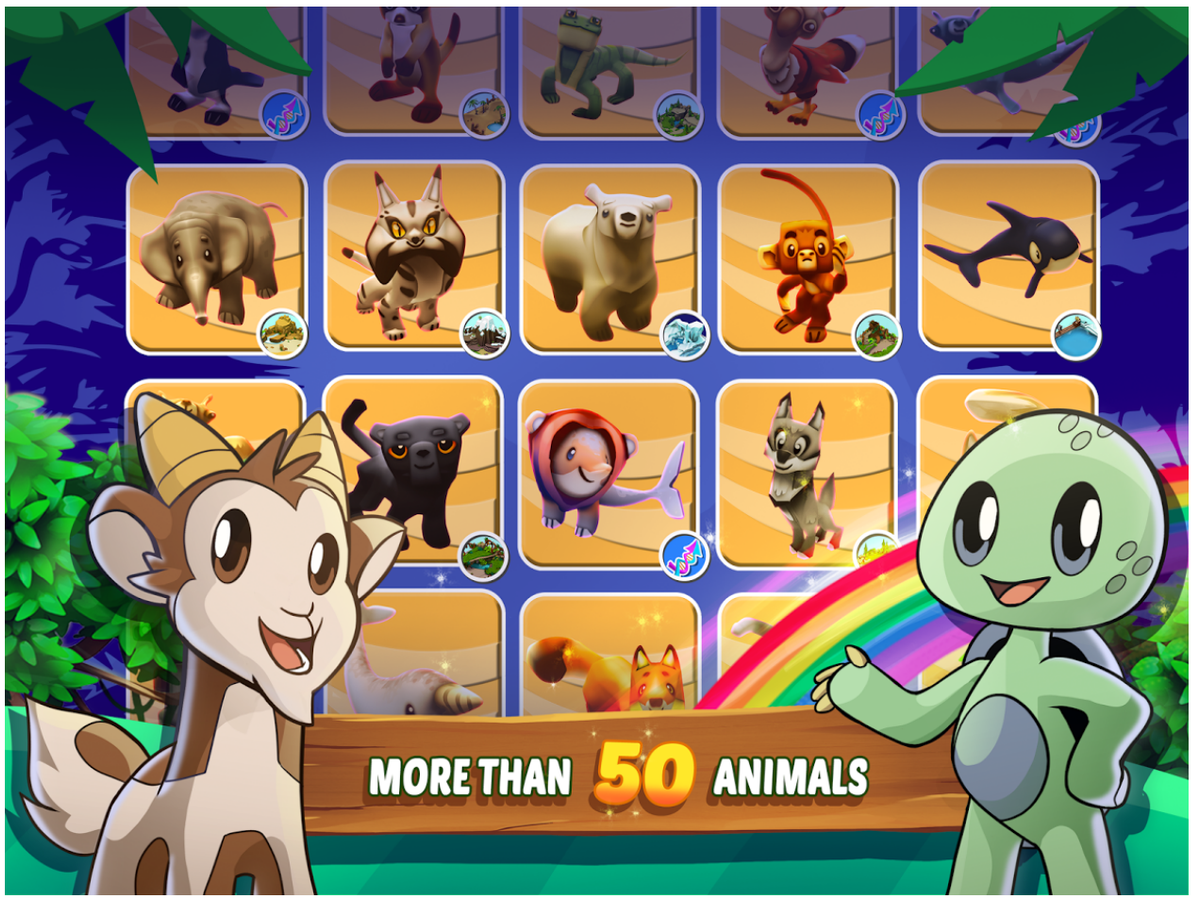
Case study: using Ansible to automate Network Operations

(by OpenTable NetOps team, 2017)

https://github.com/opentable/ansible-examples

Part 1: Ansible project overview



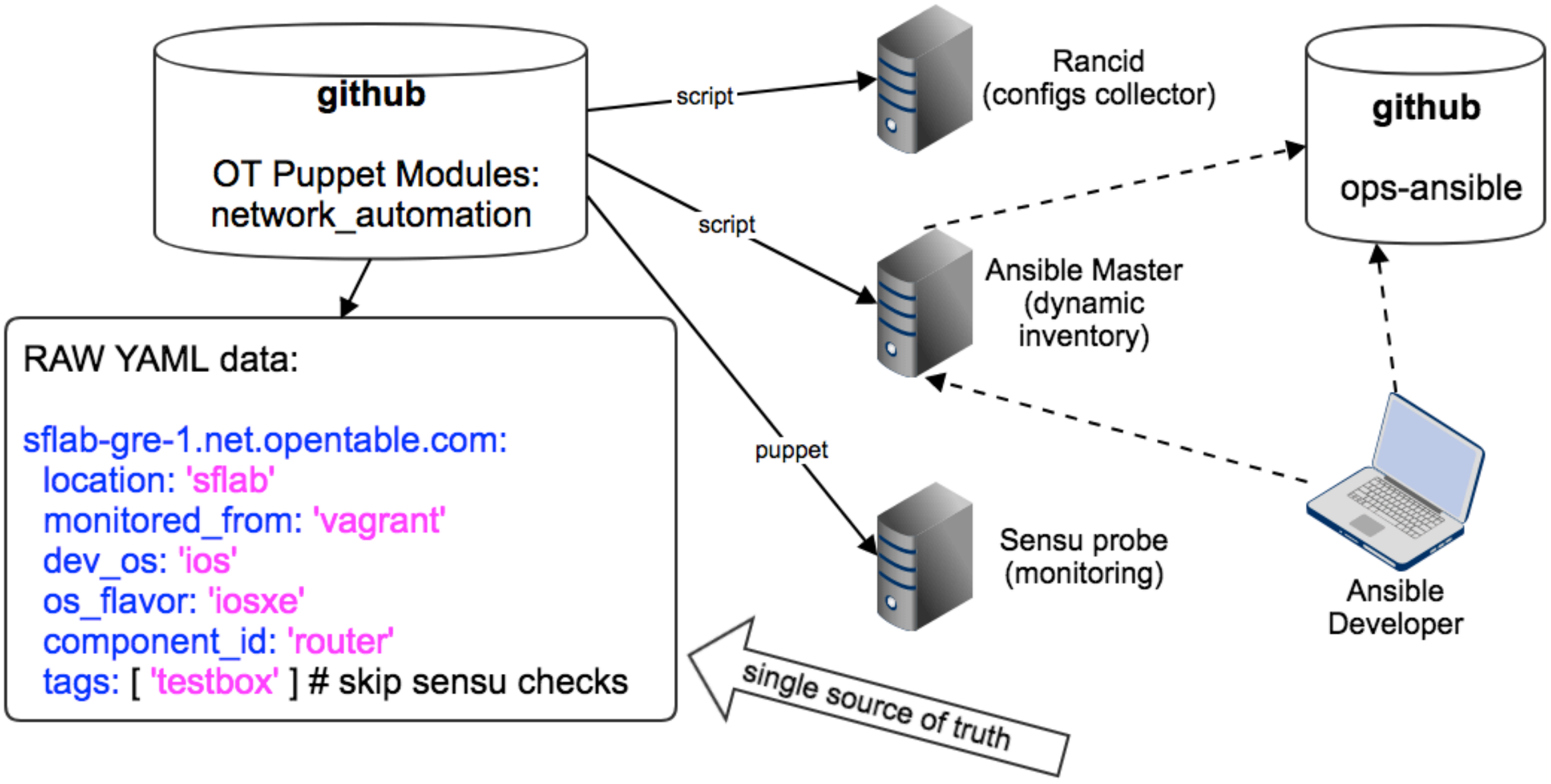
Our network environment by hardware models

- Juniper switches (EX-2200/3200/4200/4500, QFX-5100)
- Juniper firewalls (SRX-220/320)
- Cisco switches (NXOS-9k, C2960X, C2960, C3750)
- Cisco routers (ASR-1001x, ISR-4451, C3900, C2900, C2600)
- Cisco firewalls (ASA-5555X, ASA-5515, etc)
- HP switches (HP 5900AF, J8693A)
- PaloAlto firewalls (PA-5000, PA-3000, etc)
- and some more (but not managed by Ansible yet)

Our Ansible project: stats

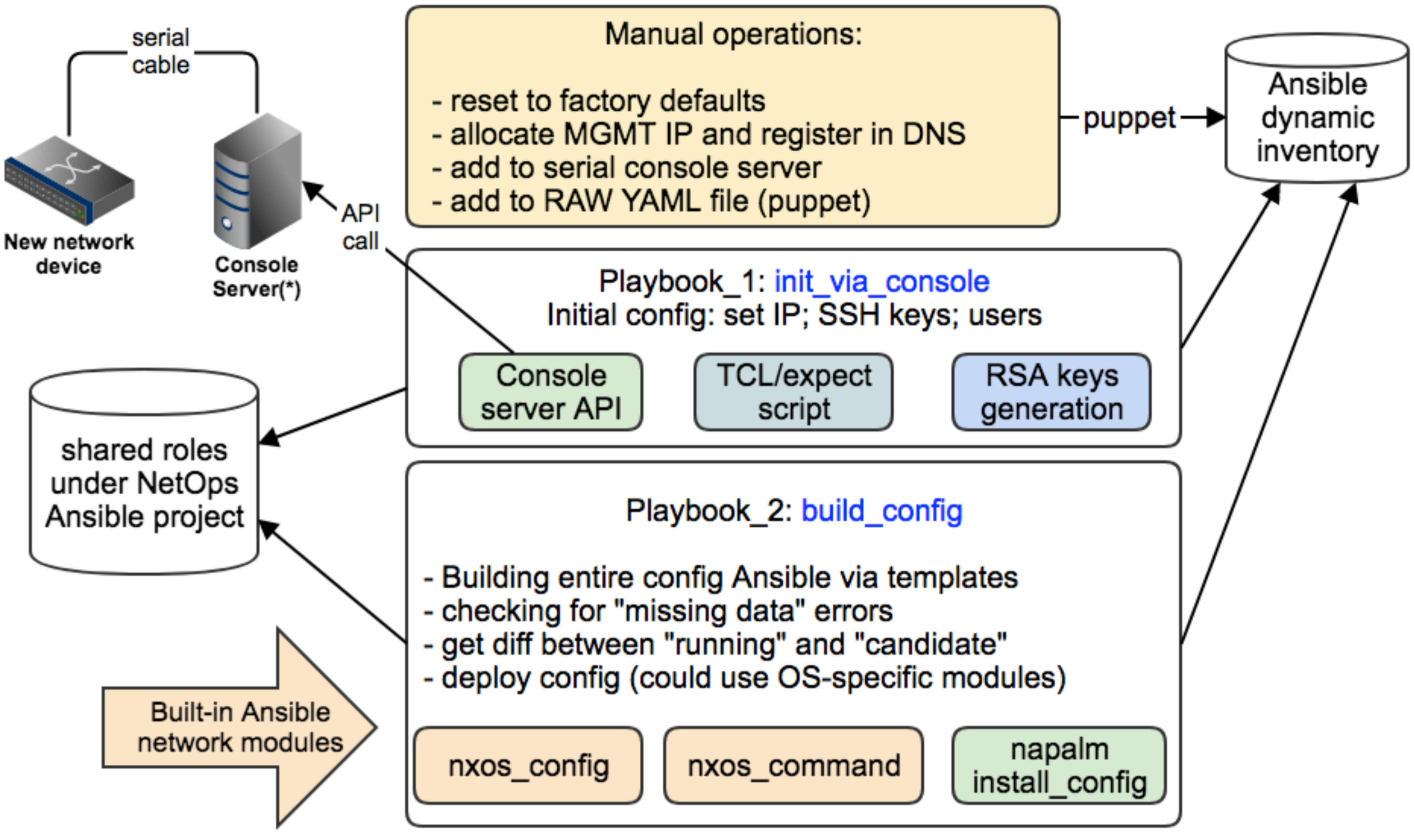
- Active developers: 2 (2 more about to start contributing to the code base)
- length of the project: 1 year (2 developers); 2 years total
- number of hosts in our inventory: 193
- code size:
    - plays: 1,300 lines of YAML
    - Jinja2 templates: 4,594 lines
    - different code fragments: 40 * <number_of_different_OS>
    - configs (global/group/host vars): ~ 11,333 lines of YAML
- GitHub commits: ~1,880 (at the time of creating presentation)

How Ansible inventory is generated, or “the integration with other infrastructure tools at Opentable"



Part 2: playbooks and roles

Steps to provision new network device in diverse network environment



1. Ansible network modules we use for deployments

- napalm_install_config (“diff” for runtime vs candidate)
- junos_install_config (“commit confirmed” option)
- nxos_config

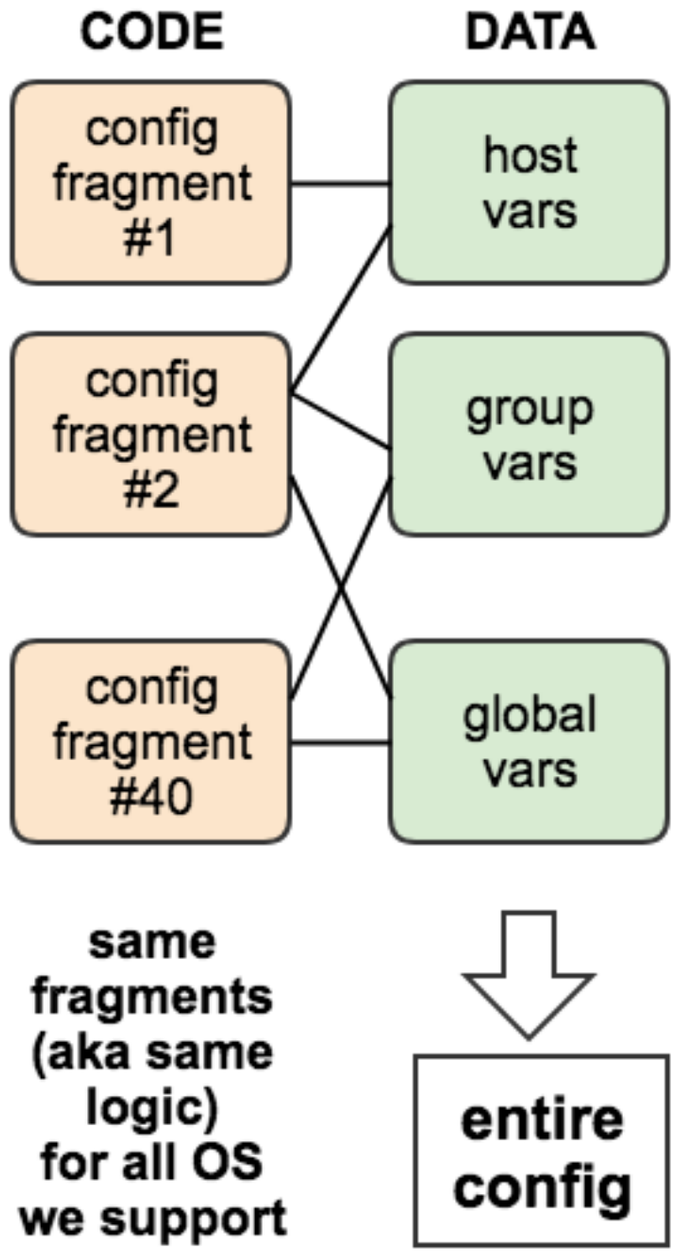
2. Other Ansible network modules used for collecting data

- ntc_save_config (3rd party - fetch runtime configs - simply because “ios_command” not working…)
- nxos_command (get interface details in json)

Why such small number of standard Ansible modules used?

Nxos

- nxos_aaa_server - Manages AAA server global configuration.
- nxos_aaa_server_host - Manages AAA server host-specific configuration.
- nxos_acl - Manages access list entries for ACLs.
- nxos_acl_interface - Manages applying ACLs to interfaces.
- nxos_bgp - Manages BGP configuration.
- nxos_bgp_af - Manages BGP Address-family configuration.
- nxos_bgp_neighbor - Manages BGP neighbors configurations.
- nxos_bgp_neighbor_af - Manages BGP address-family's neighbors configuration.
- nxos_command - Run arbitrary command on Cisco NXOS devices
- nxos_config - Manage Cisco NXOS configuration sections
- nxos_evpn_global - Handles the EVPN control plane for VXLAN.
- nxos_evpn_vni - Manages Cisco EVPN VXLAN Network Identifier (VNI).
- nxos_facts - Gets facts about NX-OS switches
- nxos_feature - Manage features in NX-OS switches.
- nxos_file_copy - Copy a file to a remote NXOS device over SCP.
- nxos_gir - Trigger a graceful removal or insertion (GIR) of the switch.
- nxos_gir_profile_management - Create a maintenance-mode or normal-mode profile for GIR.
- nxos_hsrp - Manages HSRP configuration on NX-OS switches.
- nxos_igmp - Manages IGMP global configuration.
- nxos_igmp_interface - Manages IGMP interface configuration.
- nxos_igmp_snooping - Manages IGMP snooping global configuration.
- nxos_install_os - Set boot options like boot image and kickstart image.
- nxos_interface - Manages physical attributes of interfaces.
- nxos_interface_ospf - Manages configuration of an OSPF interface instance.
- nxos_ip_interface - Manages L3 attributes for IPv4 and IPv6 interfaces.
- nxos_mtu (D) - Manages MTU settings on Nexus switch.
- nxos_ntp - Manages core NTP configuration.
- nxos_ntp_auth - Manages NTP authentication.
- nxos_ntp_options - Manages NTP options.
- nxos_nxapi - Manage NXAPI configuration on an NXOS device.
- nxos_ospf - Manages configuration of an ospf instance.
- nxos_ospf_vrf - Manages a VRF for an OSPF router.
- nxos_overlay_global - Configures anycast gateway MAC of the switch.
- nxos_pim - Manages configuration of a PIM instance.
- nxos_pim_interface - Manages PIM interface configuration.
- nxos_pim_rp_address - Manages configuration of an PIM static RP address instance.
- nxos_ping - Tests reachability using ping from Nexus switch.
- nxos_portchannel - Manages port-channel interfaces.
- nxos_reboot - Reboot a network device.
- nxos_rollback - Set a checkpoint or rollback to a checkpoint.
- nxos_smu - Perform SMUs on Cisco NX-OS devices.
- nxos_snapshot - Manage snapshots of the running states of selected features.
- nxos_snmp_community - Manages SNMP community configs.
- nxos_snmp_contact - Manages SNMP contact info.
- nxos_snmp_host - Manages SNMP host configuration.
- nxos_snmp_location - Manages SNMP location information.
- nxos_snmp_traps - Manages SNMP traps.
- nxos_snmp_user - Manages SNMP users for monitoring.
- nxos_static_route - Manages static route configuration
- nxos_switchport - Manages Layer 2 switchport interfaces.
- nxos_system - Manage the system attributes on Cisco NXOS devices
- nxos_template (D) - Manage Cisco NXOS device configurations
- nxos_udld - Manages UDLD global configuration params.
- nxos_udld_interface - Manages UDLD interface configuration params.



Ios

- ios_banner - Manage multiline banners on Cisco IOS devices
- ios_command - Run commands on remote devices running Cisco IOS
- ios_config - Manage Cisco IOS configuration sections
- ios_facts - Collect facts from remote devices running Cisco IOS
- ios_system - Manage the system attributes on Cisco IOS devices
- ios_template (D) - Manage Cisco IOS device configurations over SSH
- ios_vrf - Manage the collection of VRF definitions on Cisco IOS devices

Part 3: inventory dive in

How our Ansible project is structured

~/ops-ansible/inventory

```
inventory
├── dynamic_inventory.rb
├── group_vars
│   ├── all
│   │   ├── acls.yaml
│   │   ├── address.yaml
│   │   ├── bgp.yaml
│   │   ├── global.yaml
│   │   ├── global_flex.yaml
│   │   ├── loopbacks.yaml
│   │   ├── mgmt_hosts.yaml
│   │   ├── ospf.yaml
│   │   └── tunnels.yaml
│   ├── asa-old_asa.yaml
│   ├── asa.yaml
│   ├── boom.yaml
│   ├── cab.yaml
│   ├── eu.yaml
│   ├── h3c.yaml
│   ├── hp.yaml
│   ├── ios-iosxe.yaml
│   ├── ios-rtr.yaml
│   ├── ios-sw.yaml
│   ├── ios.yaml
│   ├── junos-ex.yaml
│   ├── junos-qfx.yaml
│   ├── junos-srx.yaml
│   ├── junos.yaml
│   ├── la.yaml
│   ├── ln.yaml
│   ├── lnhq.yaml
│   ├── mum.yaml
│   ├── na.yaml
│   ├── nxos.yaml
│   ├── pac.yaml
│   ├── panos-lnhq.yaml
│   ├── panos-sf.yaml
│   ├── panos.yaml
│   ├── sc-junos.yaml
│   ├── sc-nxos.yaml
│   ├── sc.yaml
```

~/ops-ansible/inventory/static_inventory.yaml

```
##
## Manual definitions (test devices: one of each kind)
##
[all-os]
sf3s3-gw-1  # ios/router
cab-imm-1   # ios/switch
sf3s3-fw-1  # asa
la-fw-1     # junos/firewall
sc-sw-07    # junos/switch/ex
sc-sw-21    # junos/switch/qfx
lnhq-l3-02  # hp
ln-l3-core  # h3c
sc-leaf-2   # nxos
lnhq-fw-5   # PAN OS
```

~/ops-ansible/playbooks/netops

```
├── build_config.yaml
├── combine_facts.yaml
├── get_facts.yaml
├── init_via_console.yaml
├── junos_command.yaml
├── roles
└── vars
```

```
roles
├── assemble_config
├── build_fragments
├── common
├── deploy_config
├── deploy_nxos_config
├── deploy_per_fragment
├── file_diff
├── gen_new_ssh_keys
├── get_config
├── get_runtime_status
├── init_via_console
├── register_fragments
└── show_diff
```

Example: Ansible inventory - call web server

curl http://localhost:8081/inventory | jq '.'

```
{
  "all": [
    "cab-test-1",
    "cab-test-2",
    "sc-test-1"
  ],
  "cab": [
    "cab-test-1",
    "cab-test-2"
  ],
  "sc": [
    "sc-test-1"
  ],
  "junos": [
    "cab-test-1",
    "cab-test-2",
    "sc-test-1"
  ],
  "_meta": {
    "hostvars": {
      "cab-test-1": {
        "serial_number": "SN_123"
      }
    }
  }
}
```

~/ops-ansible/inventory/test.rb

```
#!/usr/bin/env ruby¬
¬
require 'open-uri'¬
¬
def get_list¬
  begin¬
    file = open("http://localhost:8081/inventory")
  rescue¬
    '{}'¬
  else¬
    file.read¬
  end¬
end¬
¬
if ARGV[0] == '--list'¬
  puts get_list¬
elsif ARGV[0] == '--host'¬
  puts '{}'¬
end¬
```

Example: web server to support dynamic inventory

webserver/source_of_truth.yaml

```
---
# YAML inventory file (source of truth)

cab-test-1:
  loc: 'cab'
  os:  'junos'

cab-test-2:
  loc: 'cab'
  os:  'junos'

sc-test-1:
  loc: 'sc'
  os:  'junos'
```

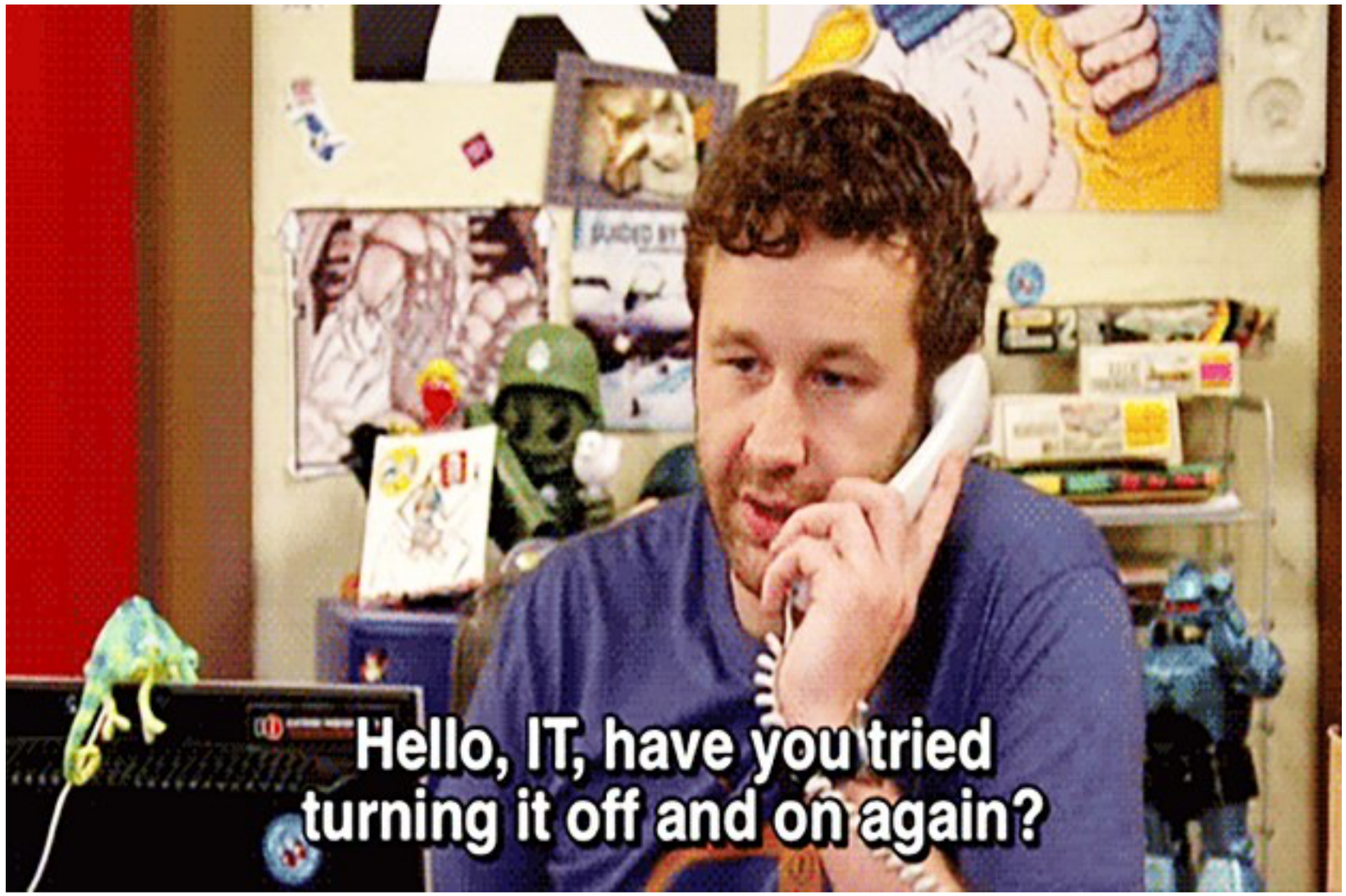
webserver/cab-test-1.facts.yaml

```
---
# Get facts file generated by periodical
# task, such as ansible playbook 'get_facts'

serial_number: 'SN_123'
```

webserver/webserver.rb

```
# Create webserver object
server = WEBrick.new(Port: 8081, DocumentRoot: '/empty')

# read 'source of truth' and 'facts' files
data  = YAML(read('source_of_truth.yaml'))
facts = YAML(read('cab-test-1.facts.yaml'))

# make 'groups' (per locations, per OS)
groups = YOUR_SCRIPT_HERE(data)

# Combine host groups with host facts
inventory = groups.merge('_meta' => {
  'hostvars' => { 'cab-test-1' => facts }})

# Serve entire structure as JSON via URL
server.mount_proc '/inventory' do |_request, response|
  response.body = inventory.to_json
end

server.start  # start web server
```

Part 4: how Ansible helps the network operations



What we use Ansible for

- to provision new devices from scratch (2-stage process)
- operational changes (best works for JunOS)
- consistent management/monitoring/security settings
- troubleshooting/simulation in a Lab
- to build entire new offices from scratch
- to allow other teams to maintain switch configs
- change control (review PR; push at specific time)

Example: Juniper QFX switch config file (hostvars)

inventory/host_vars/sc-sw-NN.yaml

```
host:¬
  sshkey_filebase: "{{ std_sshkey_filebase }}"¬
  lags:¬
    "ae0": "{{ default_qfx_uplinks }}"¬
    "ae1":¬
      description: "_lldp: LACP downlink to sc-imm-204"¬
      settings: "{{ sw_uplink_settings }}"¬
      members:¬
        - "xe-0/0/47"¬
        - "xe-1/0/47"¬
    "ae2":¬
      description: "sc-vmhead-75"¬
      settings: "{{ vmhead_uplink_settings }}"¬
      members:¬
        - "xe-0/0/11"¬
        - "xe-1/0/11"¬
    "ae3":¬
      description: "sc-vmhead-76"¬
      settings: "{{ vmhead_uplink_settings }}"¬
      members:¬
        - "xe-0/0/0"¬
        - "xe-1/0/0"¬
```

NetOps

TechOps

inventory/group_vars/sc-junos.yaml

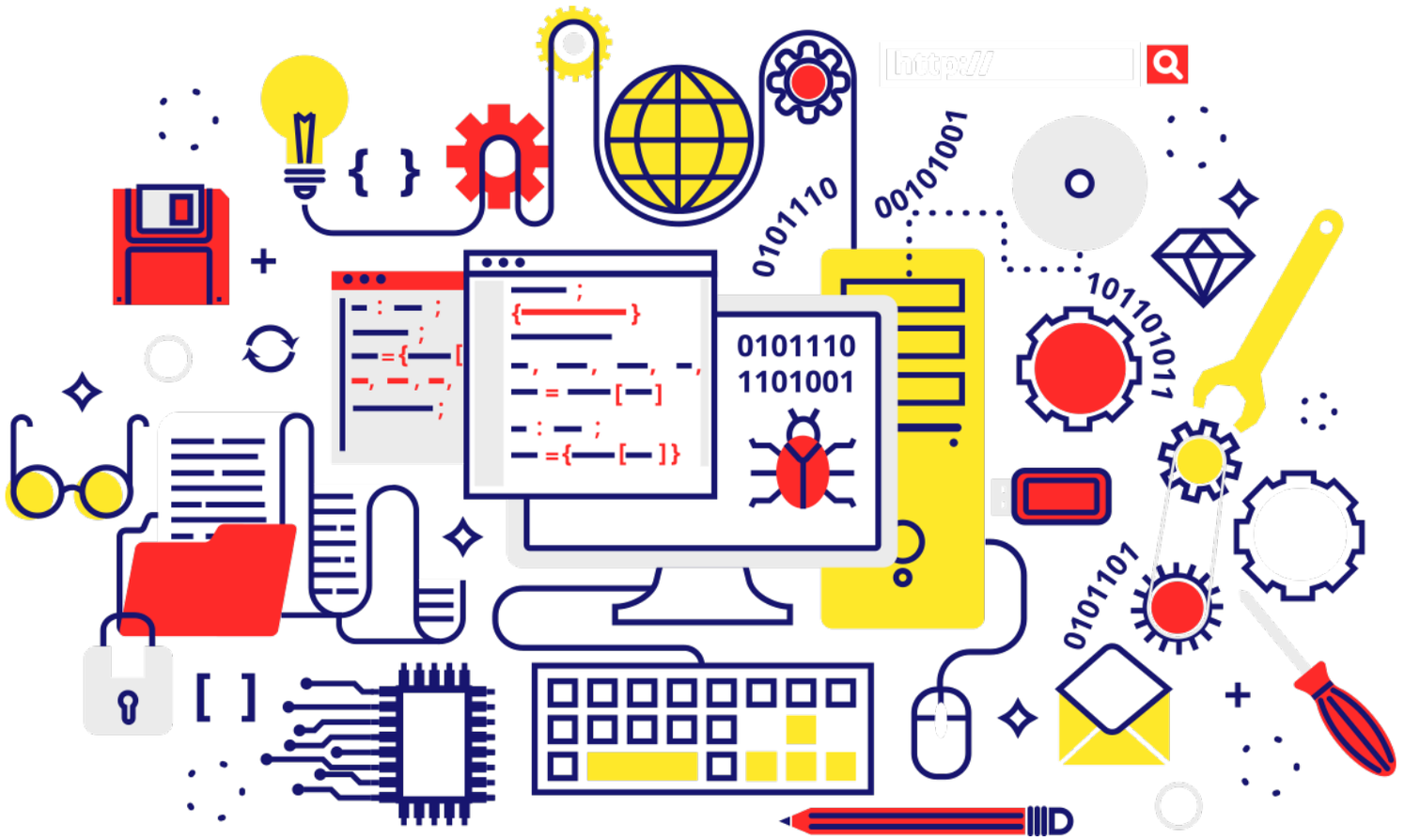
```
default_qfx_uplinks:¬
  description: "_lldp: uplink SC-SPINE 40g vPC"¬
  settings: "{{ sw_uplink_cisco_junos }}"¬
  members:¬
    - "et-0/0/48"¬
    - "et-1/0/48"¬
```

inventory/group_vars/all/global.yaml

```
sw_uplink_settings:¬
  mtu: "{{ global.jumbo_mtu }}"¬
  aggregated-ether-options:¬
    hard-coded:  # text to be copied line by line
      - 'minimum-links 1' # to minimize diff betw
    lacp: active¬
  mode: "trunk"¬
  vlan: "all"¬
  stp_role: "stp"¬
  members_ether-options:¬
    hard-coded: # just a text config options to b
      - 'auto-negotiation'¬
```

Part 5: Ansible is essentially a software development project

(and should be treated like one)



What are s/w challenges

- Python virtual env (libraries, versions, clones, compatibility, etc)
- integration with other TechOps infrastructure
- dynamic inventory with get_facts
- design the groups (location, OS, loc+OS, OS flavor, etc)
- design reusable coding patterns (very important)
- move quickly, but refactor later!
- design data structures to reflect YOUR environment
- design custom Python filters to support your data structures

Example: how router's config file looks like

```
host:¬
  vrf:¬
    - "{{ std_vrf_LAN }}"¬
¬
  acls: "{{ (common_ACLs | slice_dict(['acl_ALL_VPCs', 'SC_GRE_in', 'acl_aws_tunnels']))-
    | combine(SC_common_ACLs) }}"¬
¬
  aspath: "{{ common_BGP_aspath | slice_dict(['10']) }}"¬
¬
  ospf:¬
    '20': "{{ std_OSPF_LAN_with_aws_bgp }}"¬
    '40': "{{ std_OSPF_WAN }}"¬
¬
  route-maps: "{{ BGP_to_OSPF_routemap }}"¬
¬
  tunnels: "{{ std_tunnels_slice | combine(aws_tunnels) }}"¬
¬
  bgp:¬
    as: '65000'¬
    router-id: "{{ vlans['100'][inventory_hostname] }}"¬
    common: "{{ common_router_BGP }}"¬
    family:¬
      - name: ipv4¬
        common: "{{ common_BGP_family }}"¬
        vrf: "{{ std_vrf_LAN.vrf }}"¬
        network:   "{{ aws_vpc_private_networks }}"¬
        peer_groups:¬
          - name: US-homed-VPCs¬
            localpref: 120¬
            prepend_qty: 1¬
```

Reusable code patterns

Example: how an Ansible-generated IOS config looks like before deployment

```
"",
"route-map BGP_localpref_90 permit 10",
" description Manage egress traffic: above or below default value 100",
" set local-preference 90",
"",
"route-map BGP_prepend_3 permit 10",
" description Manage ingress traffic: add prepend len=3 to ASPATH",
" set as-path prepend 65000 65000 65000 ",
"",
"",
"router bgp 65000",
" address-family ipv4 vrf OT-IntraNet",
"  neighbor US-homed-QA-VPCs peer-group",
"  neighbor US-homed-QA-VPCs timers 10 30 30",
"  neighbor US-homed-QA-VPCs version 4",
"  neighbor US-homed-QA-VPCs soft-reconfiguration inbound",
"  neighbor US-homed-QA-VPCs filter-list 10 out",
"  neighbor US-homed-QA-VPCs route-map BGP_localpref_90 in",
"  neighbor US-homed-QA-VPCs route-map BGP_prepend_3 out",
" exit-address-family",
"exit",
"",
"",
"router bgp 65000",
" address-family ipv4 vrf OT-IntraNet",
"  neighbor 10.253.253.44 remote-as 65000",
"  neighbor 10.253.253.44 update-source Loopback253",
"  neighbor 10.253.253.44 description iBGP to LAN sc-gre-2",
"  neighbor 10.253.253.44 timers 10 30 30",
"  neighbor 10.253.253.44 version 4",
"  neighbor 10.253.253.44 soft-reconfiguration inbound",
"  neighbor 10.253.253.44 next-hop-self",
"  neighbor 10.253.253.44 fall-over",
"  neighbor 10.253.253.44 activate",
"  neighbor 10.253.253.44 password 7 [redacted]",
" exit-address-family",
"exit",
"",
```

Part 6: operational challenges in NetOps

(not an Ansible fault)

Operational challenge 1: in-place change (replace)

```
"   system { ... }", ¬
"[edit interfaces me0]", ¬
"-    unit 0 {", ¬
"-        family inet;", ¬
"-    }", ¬
"[edit interfaces vlan unit 222]", ¬
"-     proxy-arp restricted;", ¬
"[edit snmp community XXXXX]", ¬
"-   authorization read-only;", ¬
"-   clients {", ¬
"-       192.168.217.80/32;", ¬
"-       10.20.29.60/32;", ¬
"-       10.10.10.60/32;", ¬
"-   }", ¬
"+   authorization read-only;", ¬
"+   clients {", ¬
"+       192.168.217.80/32;", ¬
"+       10.20.29.60/32;", ¬
"+   }", ¬
"[edit snmp trap-group Airwave targets]",
"-    10.10.10.60;"
```

```
‹-Pro-331:/var/tmp/build/netops/sflab-edge-1 $ ls -1l

 ykretov  wheel     68 Jul 25 15:51 diffs
 ykretov  wheel   1156 Jul 25 15:51 frags
 ykretov  wheel   3492 Jul 25 15:52 generated_config.diff
 ykretov  513    32881 Jul 25 15:51 generated_config.txt
```

number of chunks in no particular order (aka hash merge)

```
remote_offices/configs$ ls -la sflab-edge-1*
 28809 Jul 16 21:16 sflab-edge-1.net.opentable.com
```

Actual config from rancid

Operational challenge 2: disruptive changes (need for a rollback)

- this is the area when some OSes shine; others - bleak...
- some ISPs are moving away from OSes that do not support true rollback and true config diff
- Ansible is only a tool for config generation and config deployment. OS is ultimately responsible for ability to compare config versions and ability to travers between commits back and force
- imagine if git would only able to roll back to previous commit as "best effort" and would occasionally fail...

Conclusion

(adopting Ansible by medium size company with diverse network infrastructure)

Organizational impact - all positive

- Tightly integrated with other CM tools (puppet)
- reproducible Ansible box(es) "by click of a button"
- new network devices would only take an hour to be fully provisioned with entire config (you need to create config file first, but you can copy it from existing device - they all rely on reusable patterns...)
- configs are captured in Rancid automatically as device is registered
- other teams may contribute via git (review/propose changes)
- all snowflake changes can be made via Ansible (as it is very flexible) so they will be fully exposed in code till removed
- dynamic inventory/WEB API allows other teams to benefit from "network infrastructure as a code" concept

Investment - relatively small

- Ansible system by itself does not require much of resources
- Human hours spent on software development - medium. Ansible is flexible and simple - allows fast entry.
- Integration with other systems (CM, monitoring): depends on how easy/ complex they are already; how easy for a network engineer it is to learn all bells and whistles
- Ability to code [well] is really at the heart of Ansible project
- Ansible development cost is much smaller compare to classical WEB team developers (less complex, more freedom, less people)
- well worth investment

Other things to remember

- people need to sharpen s/w development skills (a must)
- Ansible is just [effective and very useful] tool, but it does not solve all problems, especially of operational nature
- Ansible is still developing - some [standard] modules might still be faulty - just FYI
- When your development project moves really fast, you might find old devices configs need to be refactored prior usage - as your code is way ahead of your old configs...
- periodical code review and refactoring is important for the overall health of the project - do not ignore
- Ansible does not automatically reduce complexity of existing networks, but it helps to standardize/unify things, and transforms plain text network configs into structured mix of "code" (templates) and "data" (YAML configs). Existing s/w development tools could successfully be applied to both, like TTD, linters, code versioning, automation and so on.

Thanks for watching!

OpenTable, 2017
Yuri Kretov, Sr. Network Engineer
ykretov@opentable.com

OpenTable®

https://github.com/opentable/ansible-examples