

A Practical Look at Network Automation

Jason Edelman
@jedelman8
jedelman.com
jason@networktocode.com

AGENDA

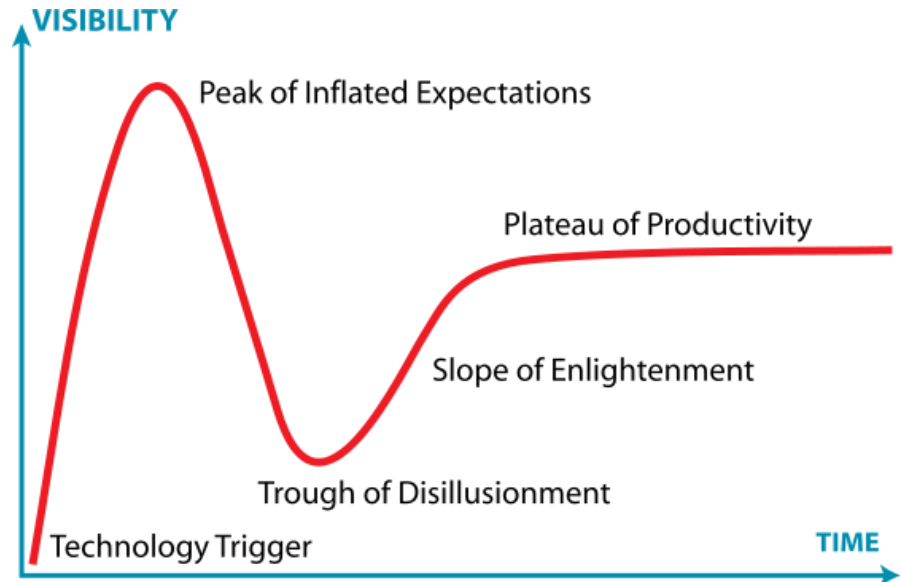
- Why Are We Here?
- SDN Alongside Network Automation
- Use Cases
- Action Plan

AGENDA

- Why Are We Here?
- SDN Alongside Network Automation
- Use Cases
- Action Plan

WHY ARE WE HERE

- OpenFlow
- Software Defined Networking
- APIs
- DevOps
- Network Automation
- Learn how to Program?



source: Wikipedia.com

THE REALITY

It's 2014 on highway 101 from San Francisco to San Jose, some cars are driving themselves. Around the world there are military aircraft flying around with no pilot, being controlled by remotely from another country. In your data center there is an engineer/admin configuring a switch on a CLI. What's wrong with this picture?

Joe Onisick – Principal Engineer Cisco Systems

Case Conference



PROBLEM: NETWORK AGILITY

1994

```
Router> enable
Router# configure terminal
Router(config)# enable secret cisco
Router(config)# ip route 0.0.0.0 0.0.0.0 20.2.2.3
Router(config)# interface ethernet0
Router(config-if)# ip address 10.1.1.1 255.0.0.0
Router(config-if)# no shutdown
Router(config-if)# exit
Router(config)# interface serial0
Router(config-if)# ip address 20.2.2.2 255.0.0.0
Router(config-if)# no shutdown
Router(config-if)# exit
Router(config)# router rip
Router(config-router)# network 10.0.0.0
Router(config-router)# network 20.0.0.0
Router(config-router)# exit
Router(config)# exit
Router# copy run start
```

Terminal Protocol: **Telnet**

2014

```
Router> enable
Router# configure terminal
Router(config)# enable secret cisco
Router(config)# ip route 0.0.0.0 0.0.0.0 20.2.2.3
Router(config)# interface ethernet0
Router(config-if)# ip address 10.1.1.1 255.0.0.0
Router(config-if)# no shutdown
Router(config-if)# exit
Router(config)# interface serial0
Router(config-if)# ip address 20.2.2.2 255.0.0.0
Router(config-if)# no shutdown
Router(config-if)# exit
Router(config)# router rip
Router(config-router)# network 10.0.0.0
Router(config-router)# network 20.0.0.0
Router(config-router)# exit
Router(config)# exit
Router# copy run start
```

Terminal Protocol: **SSH**

LOOKING AHEAD

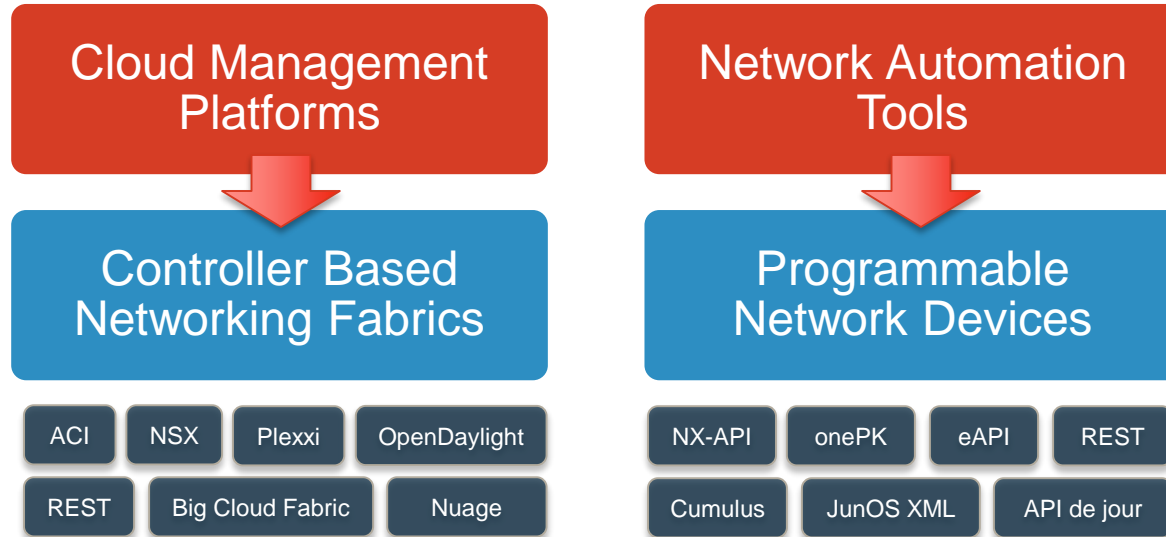
- Network Operations does in fact need to be improved, but there is more...
- Need to embrace the people that embrace the culture, process, and technology that adapt to change
- Re-think: Engineer for Change



AGENDA

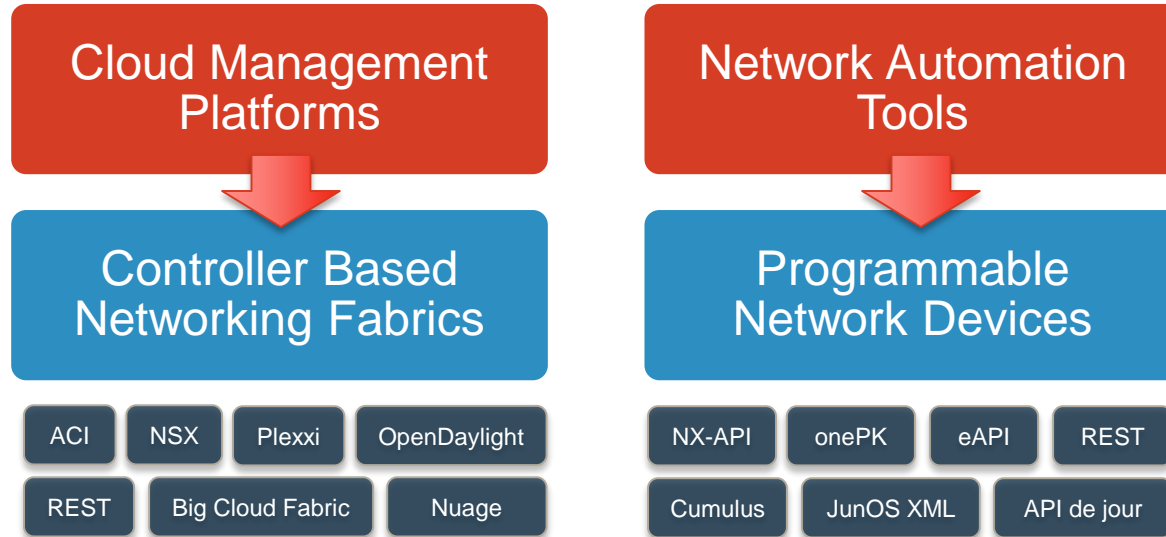
- Why Are We Here?
- **SDN Alongside Network Automation**
- Use Cases
- Action Plan

EVOLVING ECOSYSTEMS



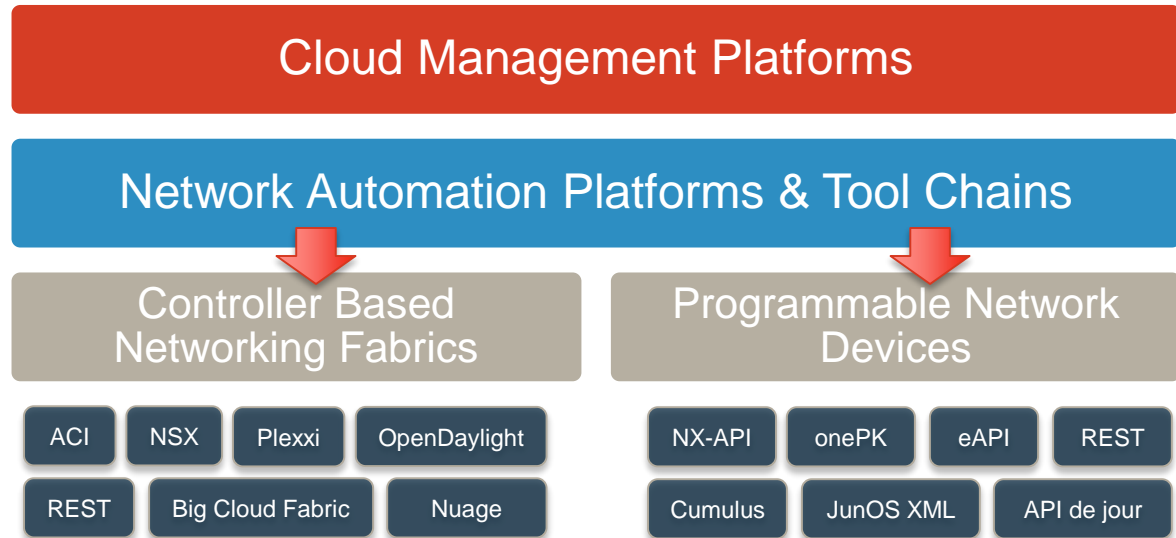
Are These Divergent Paths?

EVOLVING ECOSYSTEMS



Programmability and Platform Extensibility Should be Key Decision Making Criteria

CONSISTENCY



Consistent policy, configuration, tools, and common languages and interfaces
DESIGN FOR CHANGE

AGENDA

- Why Are We Here?
- SDN alongside Network Automation
- **Use Cases**
- Action Plan

Let's Get Practical

Template Building

Device Configurations, Vendor Migrations, IPv4 to IPv6 Migration, Site Rollouts, Office/DC Relocations, BYOD configs for switches

Data Collection

Cabling Check, Neighbors, Serial Numbers (support contracts?), Linecards, Modules, Audit Checks, PSIRT checks

Super Commands

Wireless Client to AP to Switchport, Phone to switchport, BGP Table + Routing Table, Integrate to UC, WLAN, IPAM

Troubleshooting (Ops)

Cabling, L2 neighbors, L3 adjacencies (have it tell you WHY the neighbor relationship failed), Interface Errors, ACLs

Source Control

Configuration, Templates, Dynamic state stored in central repositories. Re-deploy infrastructure → DR/BCP, Relocations

Provisioning

The Scary Part? Configs, config snippets, one-off changes

SO MUCH CAN BE DONE WITHOUT “PUSHING” CONFIGS

TEMPLATE BUILDING



WHERE TO BEGIN?

TEMPLATIZE CONFIGS



```
config-template - Notepad
File Edit Format View Help
snmp-server community ro_string RO 5
snmp-server community rw_string RW 95
snmp-server location INTEROP
snmp-server contact JASON_EDELMAN
snmp-server host 10.10.10.10

ip name-server 10.10.10.11
ntp server 10.10.10.12
```



```
config-template.j2
1
2
3 snmp-server community {{ snmp_ro }} RO 5
4 snmp-server community {{ snmp_rw }} RW 95
5 snmp-server location {{ snmp_location }}
6 snmp-server contact {{ snmp_contact }}
7 snmp-server host {{ snmp_snmp_trap_dest }}
8
9 ip name-server {{ dns_server }}
10 ntp server {{ ntp_server }}
11
```

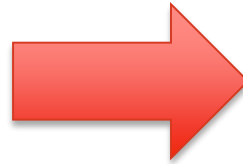
DE-COUPLE THE VARIABLES



```
config-template - Notepad
File Edit Format View Help
snmp-server community ro_string RO 5
snmp-server community rw_string RW 95
snmp-server location INTEROP
snmp-server contact JASON_EDELMAN
snmp-server host 10.10.10.10

ip name-server 10.10.10.11

ntp server 10.10.10.12
```



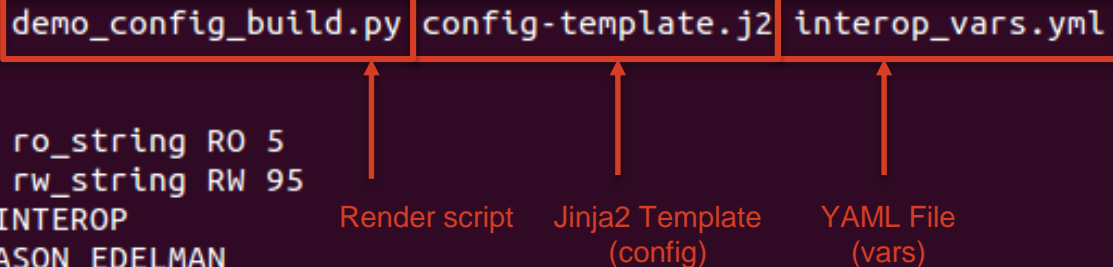
```
config-template.j2
interop_vars.yml
1 ---
2
3
4
5
6   snmp_contact: JASON_EDELMAN
7   snmp_location: INTEROP
8
9   snmp_ro: ro_string
10  snmp_rw: rw_string
11  snmp_trap_dest: 10.10.100.10
12
13  dns_server: 10.10.10.11
14  ntp_server: 10.10.10.12
15
```


RENDER THE TEMPLATE

```
cisco@onepk:~$ python demo_config_build.py config-template.j2 interop_vars.yml

snmp-server community ro_string R0 5
snmp-server community rw_string RW 95
snmp-server location INTEROP
snmp-server contact JASON_EDELMAN
snmp-server host

ip name-server 10.10.10.11
ntp server 10.10.10.12
```



Great way to get started, but for more robust templates, a “real” tool should be used.

Render script available on my GitHub page.

Think config snippets:

- v4 to v6
- BYOD
- one-offs

<https://github.com/jedelman8/interop-nyc-2014>

CABLE VERIFICATION



source: peterskastner.files.wordpress.com

- Is the cabling accurate?
- How do you know if something is mis-cabled?
- Ever work 3rd party contractors that cable based on your patch schedule and somehow it doesn't come out right?

DEFINE THE DESIRED STATE

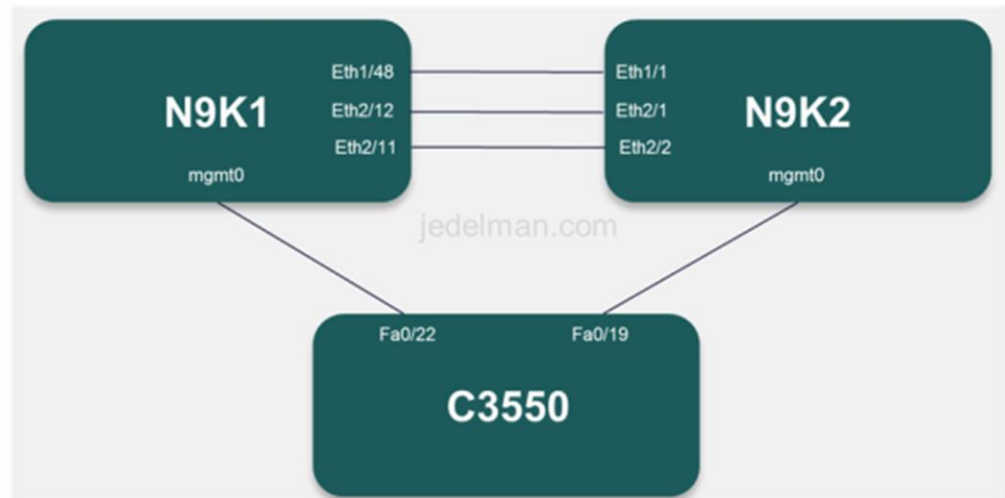
1 Define the Desired Cabling Scheme

```
topo1.yml
1 ---
2
3 ## hostnames are required. If FQDN is set, the domain should not be used
4 ## topo is the top level variable
5 ## make sure all interface keys are equal to their local_intf!!!
6
7 topo:
8   ## 9K1 is a spine / core device
9   N9K1:
10     mgmt0:
11       { remote_hostname: c3550, remote_intf: FastEthernet0/22 }
12     Ethernet1/48:
13       { remote_hostname: N9K2, remote_intf: Ethernet1/1 }
14     Ethernet2/11:
15       { Ethernet2/11, remote_hostname: N9K2, remote_intf: Ethernet2/2 }
```

(only showing portion of YAML file)



YAML



OBTAIN THE ACTUAL STATE

- 1 Define the Desired Cabling Scheme
- 2 Get the actual (run time) topology via CDP/LLDP

```
topo1.yml
1 ---
2
3 ## hostnames are required. If FQDN is set, the domain should not be use
4 ## topo is the top level variable
5 ## make sure all interface keys are equal to their local_intf!!!
6
7 topo:
8   ## 9K1 is a spine / core device
9   N9K1:
10     mgmt0:
11       { remote_hostname: c3550, remote_intf: FastEthernet0/22 }
12     Ethernet1/48:
13       { remote_hostname: N9K2, remote_intf: Ethernet1/1 }
14     Ethernet2/11:
15       { Ethernet2/11, remote_hostname: N9K2, remote_intf: Ethernet2/2 }
```

(only showing portion of YAML file)

Multiple methods available

This example uses a Python script and gets neighbor info using NX-API on the Nexus 9000

DESIRED VS. ACTUAL

- 1 Define the Desired Cabling Scheme
- 2 Get the actual (run time) topology via CDP/LLDP
- 3 Examine Desired vs. Actual

```

topo1.yml
1 ---
2
3 ## hostnames are required. If FQDN is set, the domain should not be use
4 ## topo is the top level variable
5 ## make sure all interface keys are equal to their local_intf!!!
6
7 topo:
8   ## 9K1 is a spine / core device
9   N9K1:
10    mgmt0:
11     { remote_hostname: c3550, remote_intf: FastEthernet0/22 }
12    Ethernet1/48:
13     { remote_hostname: N9K2, remote_intf: Ethernet1/1 }
14    Ethernet2/11:
15     { Ethernet2/11, remote_hostname: N9K2, remote_intf: Ethernet2/2 }

```

(only showing portion of YAML file)

```

cisco@edelman:~/apps/nxapi/library/pyfiles$ python get_cdp.py

SOURCE DEVICE:  N9K1
=====
N9K1:Ethernet1/48 -> N9K2:Ethernet1/1      Status: OK
N9K1:Ethernet2/11 -> N9K2:Ethernet2/2      Status: OK
N9K1:mgmt0 -> c3550:FastEthernet0/22      Status: OK
N9K1:Ethernet2/12 -> N9K2:Ethernet2/1      Status: OK
=====

SOURCE DEVICE:  N9K2
=====
N9K2:Ethernet1/1 -> N9K1:Ethernet1/48      Status: OK
N9K2:mgmt0 -> c3550:FastEthernet0/19      Status: OK
N9K2:Ethernet2/2 -> N9K1:Ethernet2/11      Status: OK
N9K2:Ethernet2/1 -> N9K1:Ethernet2/12      Status: OK
=====

```

PAUSE: SAMPLE NX-API OUTPUT

```
cisco@edelman:~/apps/nxapi/library/pyfiles$ python interop-cdp.py
=====
Neighbor: c3550
Local Interface: mgmt0
Neighbor Interface: FastEthernet0/22
=====
Neighbor: N9K2.cisconxapi.com(SAL1819S6LU)
Local Interface: Ethernet1/48
Neighbor Interface: Ethernet1/1
=====
Neighbor: N9K2.cisconxapi.com(SAL1819S6LU)
Local Interface: Ethernet2/11
Neighbor Interface: Ethernet2/2
=====
Neighbor: N9K2.cisconxapi.com(SAL1819S6LU)
Local Interface: Ethernet2/12
Neighbor Interface: Ethernet2/1
=====
```

NX-API

< 20 Lines of Code

```
interop-cdp.py
1 #!/usr/bin/env python
2
3 import xmltodict
4 from device import Device
5
6 if __name__ == "__main__":
7
8     switch = Device(ip='192.168.200.50')
9
10    switch.open()
11
12    my_data = switch.show('show cdp neighbors')
13
14    result = xmltodict.parse(my_data[1])
15
16    cdp_table = result['ins_api']['outputs']['output']['body'] \
17                ['TABLE_cdp_neighbor_brief_info']['ROW_cdp_neighbor_brief_info']
18
19    for each_neighbor in cdp_table:
20        print '=' * 40
21        for key, value in each_neighbor.iteritems():
22            if key == 'intf_id': print 'Local Interface: ', value
23            if key == 'device_id': print 'Neighbor: ', value
24            if key == 'port_id': print 'Neighbor Interface: ', value
25    print '=' * 40
```

- 1 Connect to Device
- 2 Wrap CLI and get return data
- 3 Convert XML to dict (JSON)
- 4 Extract CDP information
- 5 Print CDP information

TROUBLESHOOTING OSPF

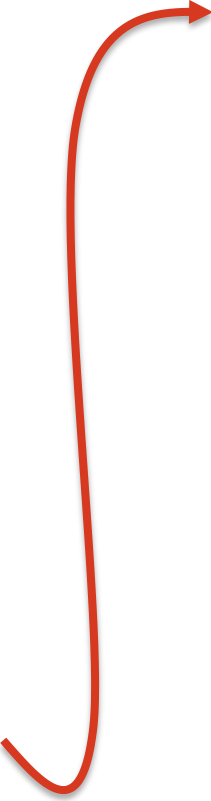
- Remember how neighbors are formed in OSPF?
- Do you remember at 3am on a Saturday?
- Does the junior network engineer remember when you're on vacation?

TROUBLESHOOTING OSPF

- Remember how neighbors are formed in OSPF?
- Do you remember at 3am on a Saturday?
- Does the junior network engineer remember when you're on vacation?
- How about we automate the process of a neighbor check?
- Do we really enjoy bouncing back and forth between routers?
- Let's get to it!

ANSIBLE

```
cisco@onepk:~/apps/a4n$ ansible-playbook ospfops.yml
PLAY [collecting ospf data] *****
TASK: [get ospf facts] *****
ok: [10.1.1.120]
ok: [10.1.1.110]
TASK: [interface ip addresses used for OSPF peering] *****
ok: [10.1.1.110] => {
  "msg": "local router interface IP address- 10.1.1.110/24 on gig0/2"
}
ok: [10.1.1.120] => {
  "msg": "local router interface IP address- 10.1.1.120/24 on gig0/1"
}
TASK: [is ospf active on interface?] *****
ok: [10.1.1.110] => {
  "msg": "ospf active on interface = True"
}
ok: [10.1.1.120] => {
  "msg": "ospf active on interface = True"
}
TASK: [process id check] *****
ok: [10.1.1.110] => {
  "msg": "at least one ospf process configurd on router = True"
}
ok: [10.1.1.120] => {
  "msg": "at least one ospf process configurd on router = True"
}
```



```
TASK: [MTUs of interfaces] *****
ok: [10.1.1.110] => {
  "msg": "MTU = 1500"
}
ok: [10.1.1.120] => {
  "msg": "MTU = 1500"
}
TASK: [ospf network type] *****
ok: [10.1.1.110] => {
  "msg": "network type = BROADCAST"
}
ok: [10.1.1.120] => {
  "msg": "network type = BROADCAST"
}
TASK: [ospf timers on interface] *****
ok: [10.1.1.110] => {
  "msg": "{u'hello': u'10', u'dead': u'40'}"
}
ok: [10.1.1.120] => {
  "msg": "{u'hello': u'10', u'dead': u'40'}"
}
TASK: [interface status] *****
ok: [10.1.1.110] => {
  "msg": "interface status=up and line protocol = up"
}
ok: [10.1.1.120] => {
  "msg": "interface status=up and line protocol = up"
}
TASK: [display neighbors and state] *****
ok: [10.1.1.110] => {
  "msg": "{u'': [u'FULL', u'BDR']}"
}
ok: [10.1.1.120] => {
  "msg": "{u'': [u'FULL', u'DR']}"
}
```

GET FACTS AND ANALYZE

```

cisco@onepk:~/apps/a4n$ ansible-playbook ospfops.yml
PLAY [ospfops] *****
TASK [get ospf facts] *****
ok: [10.1.1.120]
ok: [10.1.1.110]
*****
110/24 on gig0/2"
"msg": "local router interface IP address- 10.1.1.120/24 on gig0/1"
}
}
TASK: [is ospf active on interface?] *****
ok: [10.1.1.110] => {
  "msg": "ospf active on interface = True"
}
ok: [10.1.1.120] => {
  "msg": "ospf active on interface = True"
}
}
TASK: [process id check] *****
ok: [10.1.1.110] => {
  "msg": "at least one ospf process configurd on router = True"
}
ok: [10.1.1.120] => {
  "msg": "at least one ospf process configurd on router = True"
}
}

```

Everything else
is just printing
data from facts

```

TASK: [MTUs of interfaces] *****
ok: [10.1.1.110] => {
  "msg": "MTU = 1500"
}
ok: [10.1.1.120] => {
  "msg": "MTU = 1500"
}
}
TASK: [ospf network type] *****
ok: [10.1.1.110] => {
  "msg": "network type = BROADCAST"
}
ok: [10.1.1.120] => {
  "msg": "network type = BROADCAST"
}
}
TASK: [ospf timers on interface] *****
ok: [10.1.1.110] => {
  "msg": "{u'hello': u'10', u'dead': u'40'}"
}
ok: [10.1.1.120] => {
  "msg": "{u'hello': u'10', u'dead': u'40'}"
}
}
TASK: [interface status] *****
ok: [10.1.1.110] => {
  "msg": "interface status=up and line protocol = up"
}
ok: [10.1.1.120] => {
  "msg": "interface status=up and line protocol = up"
}
}
TASK: [display neighbors and state] *****
ok: [10.1.1.110] => {
  "msg": "{u': [u'FULL', u'BDR']}"
}
ok: [10.1.1.120] => {
  "msg": "{u': [u'FULL', u'DR']}"
}
}

```

THE ANSIBLE PLAYBOOK

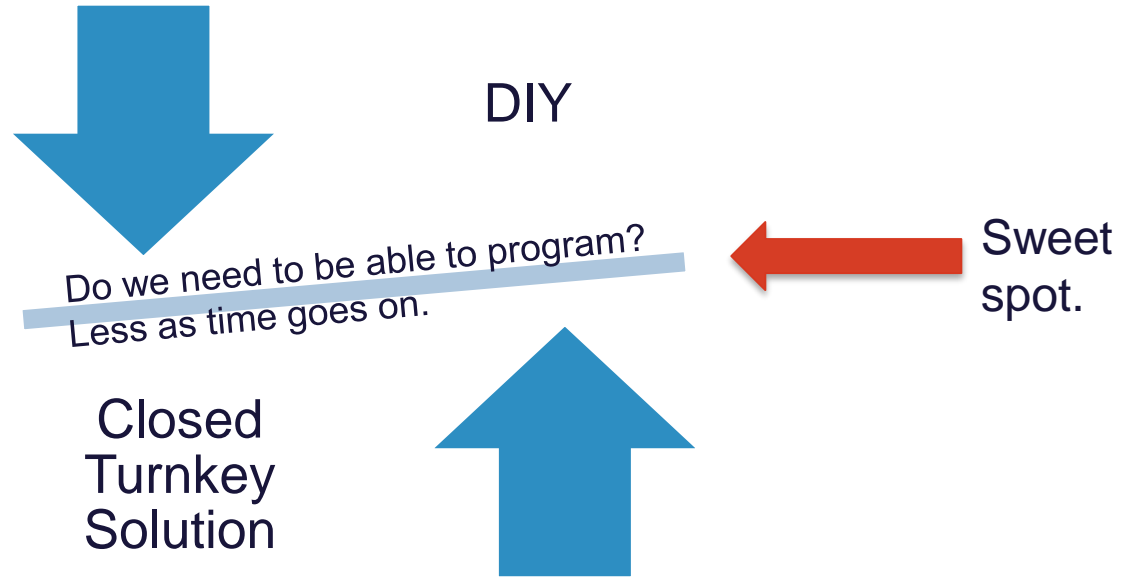
```
ospfops.yml
1  ---
2  #-----PLAY 1 -----#
3
4  - name: collecting ospf data
5    hosts: routers
6    connection: local
7    gather_facts: no
8
9    tasks:
10
11     - name: get ospf facts
12       ospf_facts: device={{inventory_hostname}} interface={{ ospf_interface }}
13       register: ospf_data
14
15     - name: interface ip addresses used for OSPF peering
16       debug: msg="local router interface IP address- {{ ospf_data.ofacts.oif_ip }}"
17
18     ## BOTTOM DEBUG / PRINTS HAVE BEEN REMOVED ##
```

THE ANSIBLE PLAYBOOK

```
ospfops.yml
1 ---
2 #-----PLAY 1 -----#
3
4 - name: collecting ospf data
5   hosts: routers
6   connection: local
7
8   name: get ospf fact:
9     ospf_facts: device=
10
11     ospf_facts: device={{inventory_hostname}} interface={{ ospf_interface }}
12     register: ospf_data
13
14
15 - name: interface ip addresses used for OSPF peering
16   debug: msg="local router interface IP address- {{ ospf_data.ofacts.oif_ip }}"
17
18 ## BOTTOM DEBUG / PRINTS HAVE BEEN REMOVED ##
```

- ospf_facts is an Ansible module
- Ansible modules can be written in Python
- BUT, WHO WRITES THEM?

WHAT OPTIONS DO WE HAVE FOR TOOLS?



AGENDA

- Why Are We Here?
- SDN alongside Network Automation
- Use Cases
- Action Plan

ACTION PLAN

- Dedicate time, maybe lots...
 - Remember how much time it took to get your existing certifications or learn any new skill?
- Document existing workflow and processes
 - Start with small tasks
 - You can't automate what you don't know
- Research DevOps Culture

ACTION PLAN

- Dedicate time, maybe lots...
 - Remember how much time it took to get your existing certifications or learn any new skill?
- Document existing workflow and processes
 - Start with small tasks
 - You can't automate what you don't know
- Research DevOps Culture
- Templating
 - Jinja2/YAML
- Scripting
 - Not building applications!
 - Python
- Try out a Device API
- Explore automation tools
 - Ansible (even if it's to see what can be done with servers)

THANK YOU

