redhat | ANSIBLE

AUTOMATION FOR NETWORK INFRASTRUCTURE

Steven Carter

Principal Solutions Architect, Ansible
scarter@redhat.com

http://www.ansible.com/network-automation

MANAGING NETWORKS HASN'T CHANGED IN 30 YEARS.

WHY HASN'T NETWORKING CHANGED?

ANSIBLE

PEOPLE

- Domain specific skillsets
- Vendor oriented experience
- Siloed organizations
- Legacy operational practices

PRODUCTS

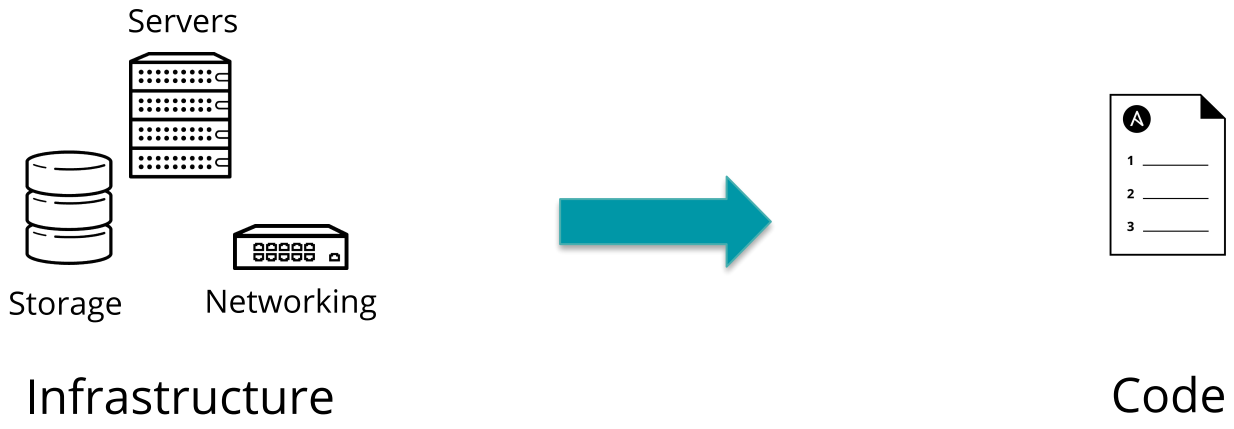
- Infrastructure-focused features
- Baroque, CLI-only methodologies
- Siloed technologies
- Monolithic, proprietary platforms

In the end, it's all about culture!

Hero as Code

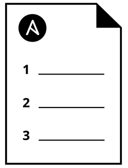
1
2
3
Hero
Code

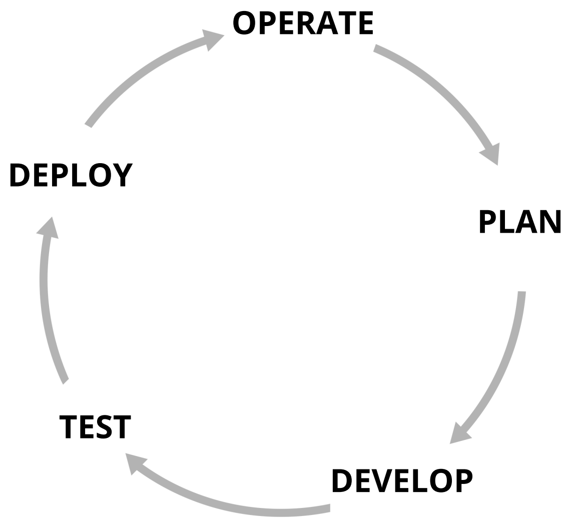
Step 1: Translate Infrastructure into Code



- Define Intent, Policy, Architecture
- Apply across device type, vendor

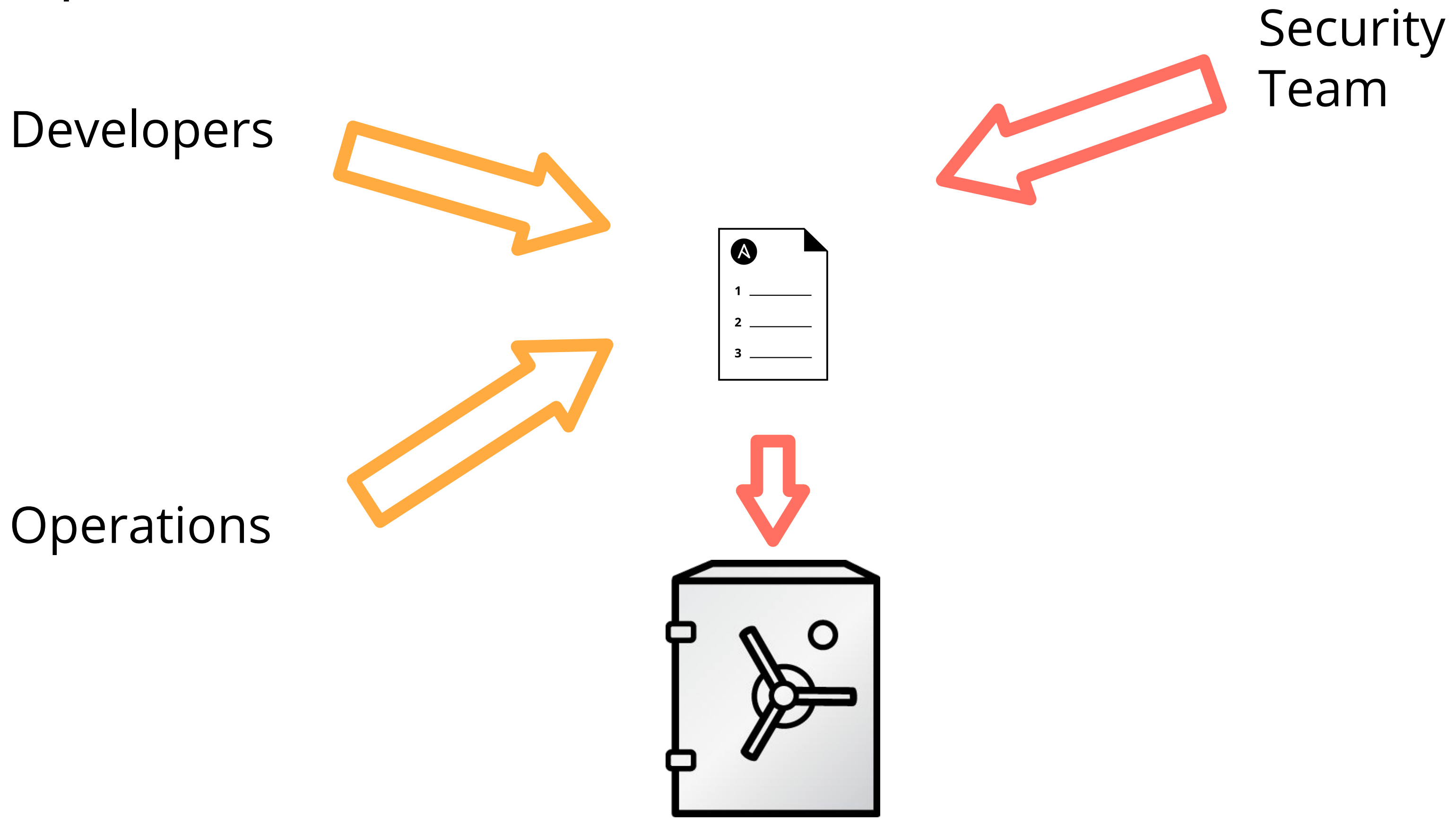
Step 2: Manage Lifecycle with Code + Process





- Revision control, configuration management
- Ensure an ongoing steady-state
- Automated testing, reduce human error

Step 3: Communicate with Code



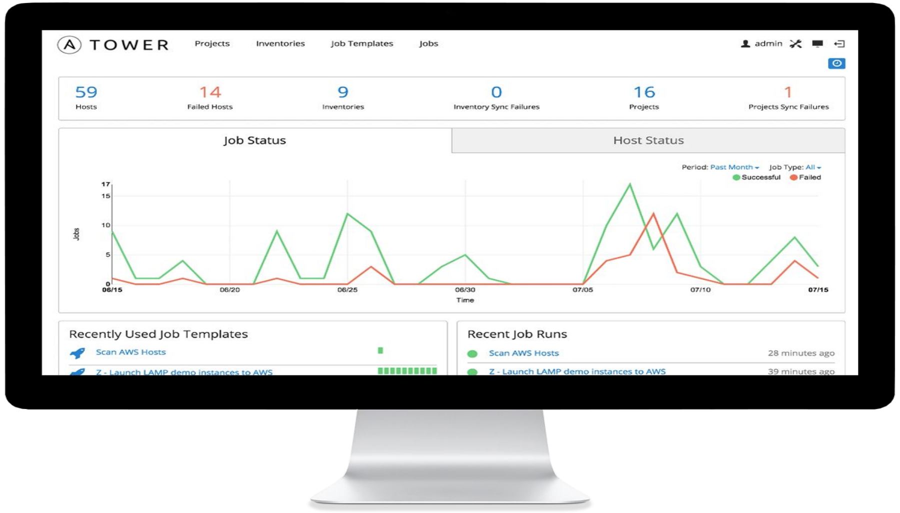
WHAT IS ANSIBLE?

ANSIBLE

Ansible is a simple automation language that can perfectly describe an IT application infrastructure in Ansible Playbooks.

Ansible Engine is an automation engine that runs Ansible Playbooks.

Ansible Tower is an enterprise framework for controlling, securing and managing your Ansible automation with a UI and RESTful API.



ANSIBLE: IT’S A MULTIFUNTION TOOL

WHY ANSIBLE?

ANSIBLE

SIMPLE

Human readable automation

No special coding skills needed

Tasks executed in order

Get productive quickly

POWERFUL

Image updates

Configuration management

Configuration validation

Compliance

Orchestrate the network lifecycle

AGENTLESS

Agentless architecture

Uses OpenSSH & WinRM

No agents to exploit or update

More efficient & more secure

OPERATIONAL CULTURE SHIFT

ANSIBLE

Traditional Network Operations

Legacy Culture

Risk averse

Proprietary Solutions

Siloed talents

Hand-crafted configuration

$\longrightarrow$

Next-Gen Network Operations

Community Culture

Risk aware

Open Solutions

Integrated teams

Automation / DevOps

COMMIT, VERIFY, CHECK

ANSIBLE TOWER FOR NETWORK OPERATIONS

ANSIBLE

ANSIBLE TOWER by Red Hat®

Building, managing dynamic inventory

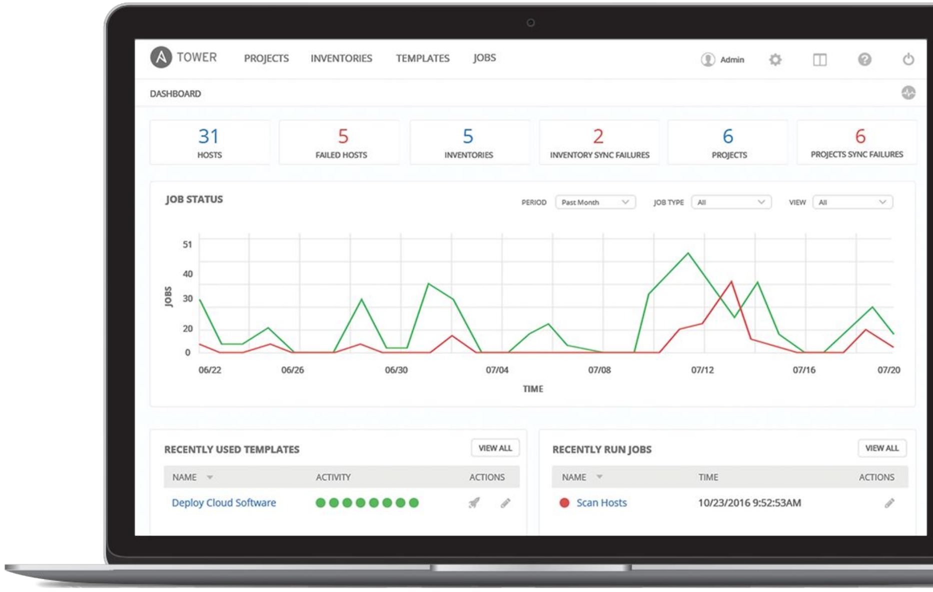
Organizing admin control with users and teams

Leverage Ansible Workflows to break up tasks

Ongoing compliance

- compare running configs to golden masters on schedules

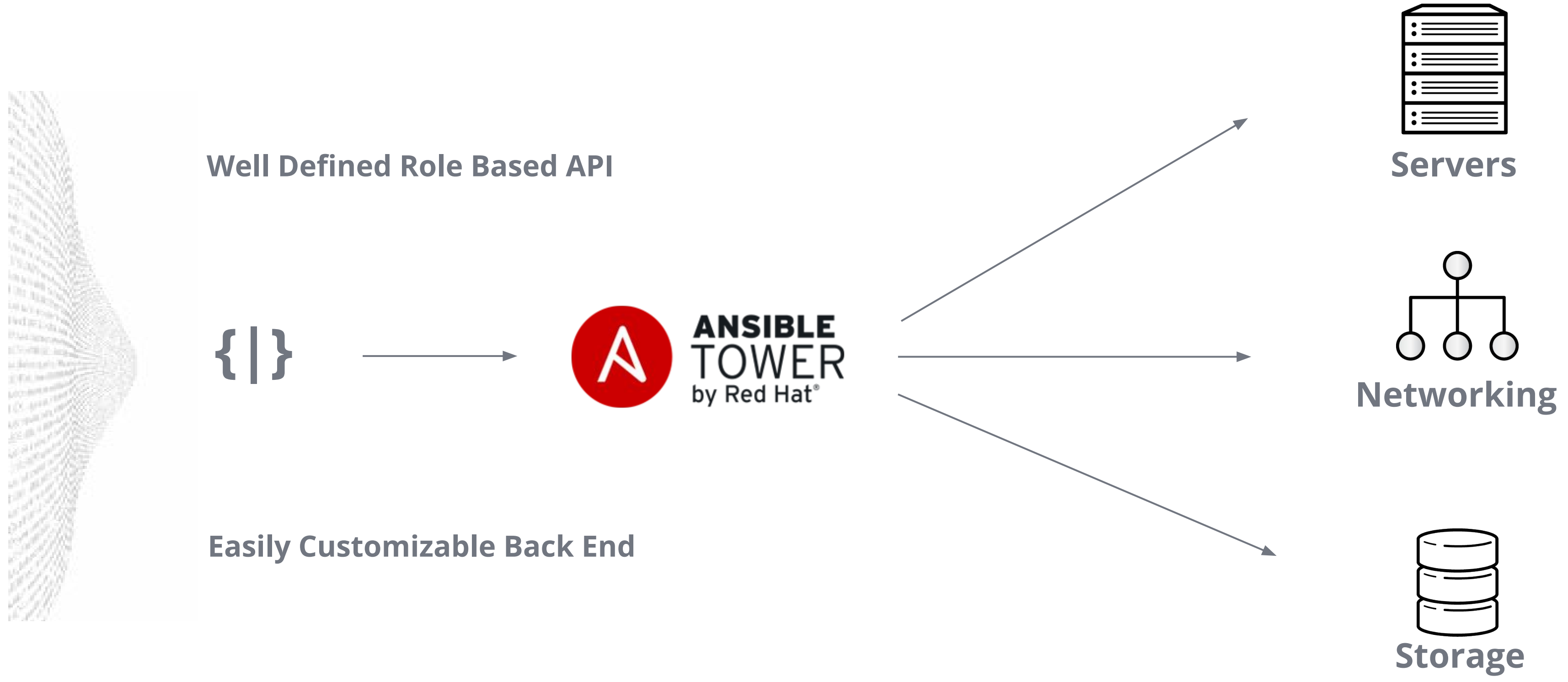
Utilize the RESTful API for anything



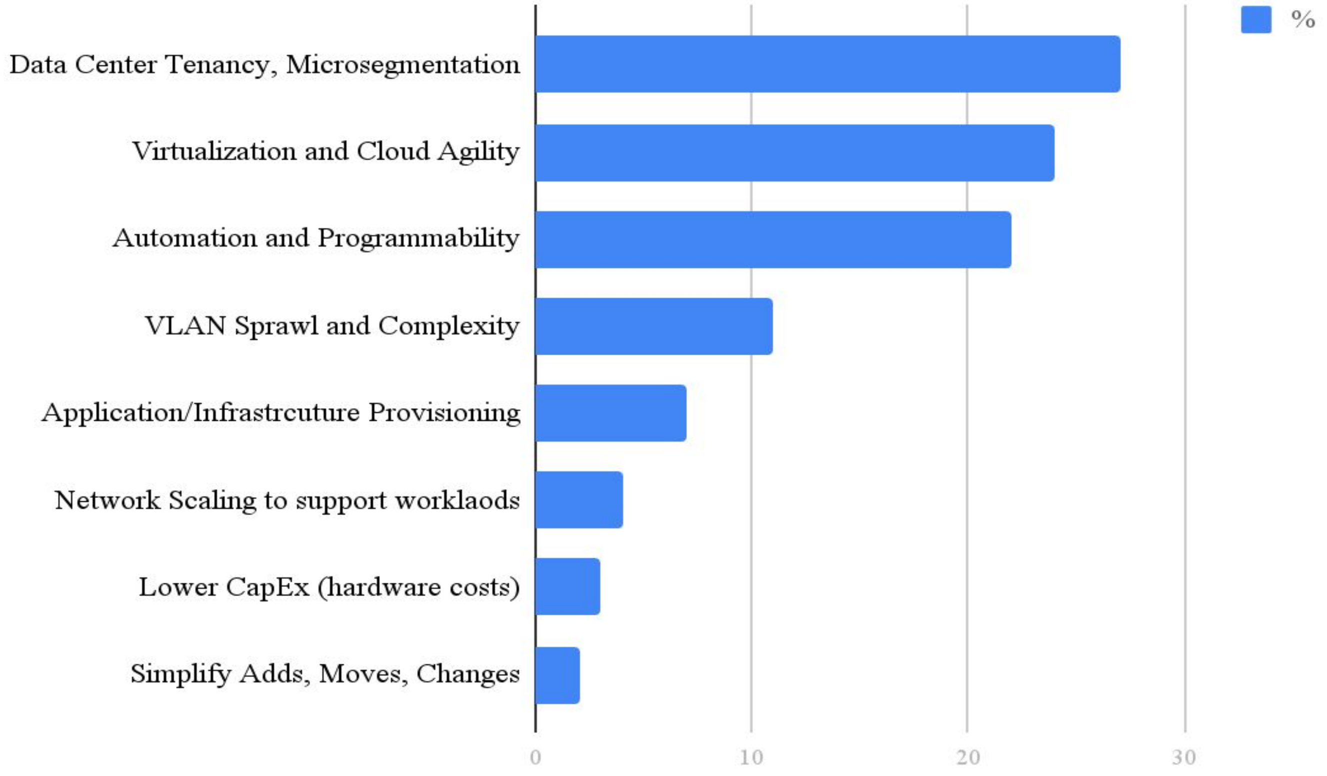
NAME	ACTIVITY	ACTIONS
Deploy Cloud Software	●●●●●●●●	

NAME	TIME	ACTIONS
● Scan Hosts	10/23/2016 9:52:53AM	

API DRIVEN INFRASTRUCTURE



PRIMARY SDN USE CASES



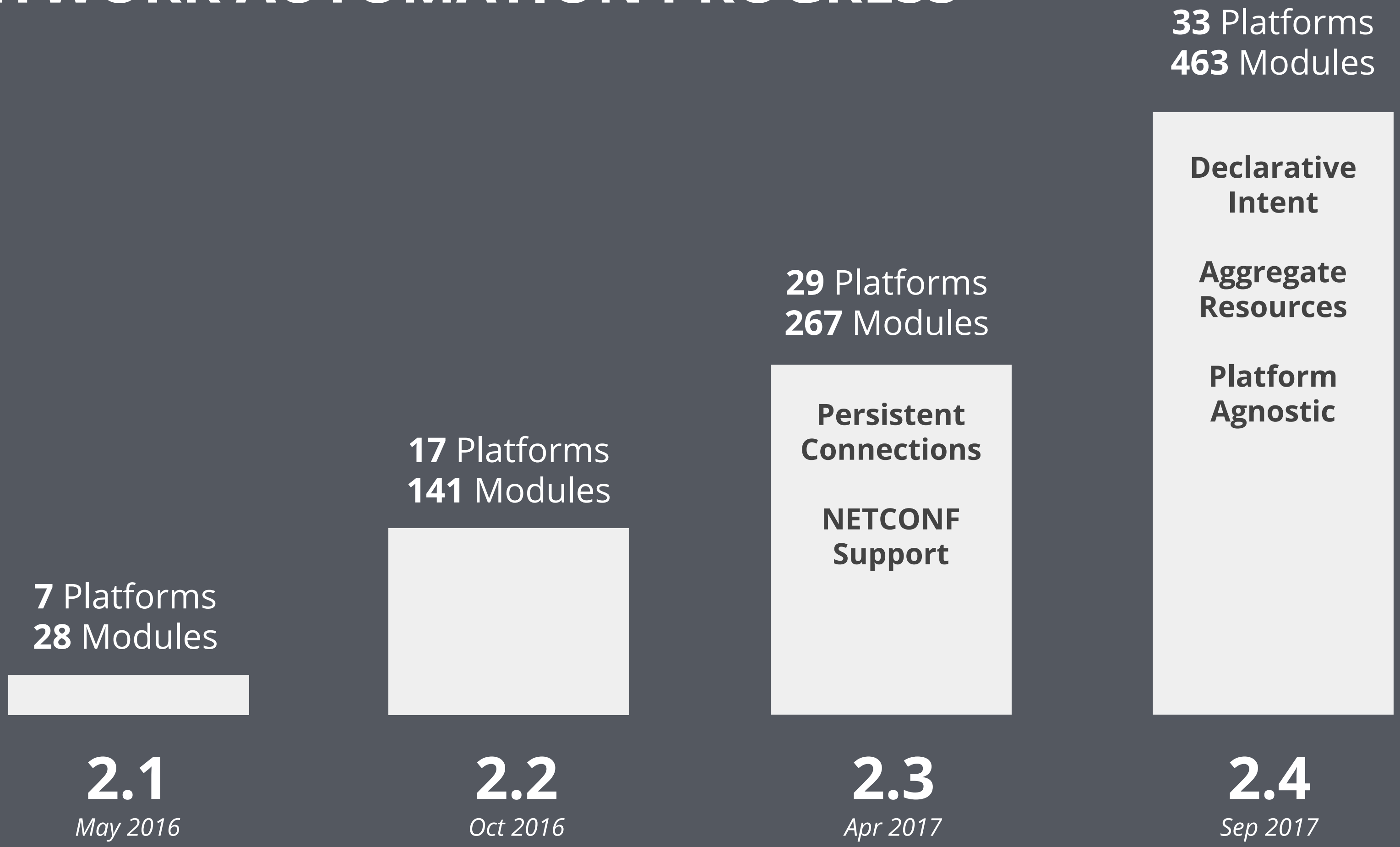
AUTOMATION vs SDN-only

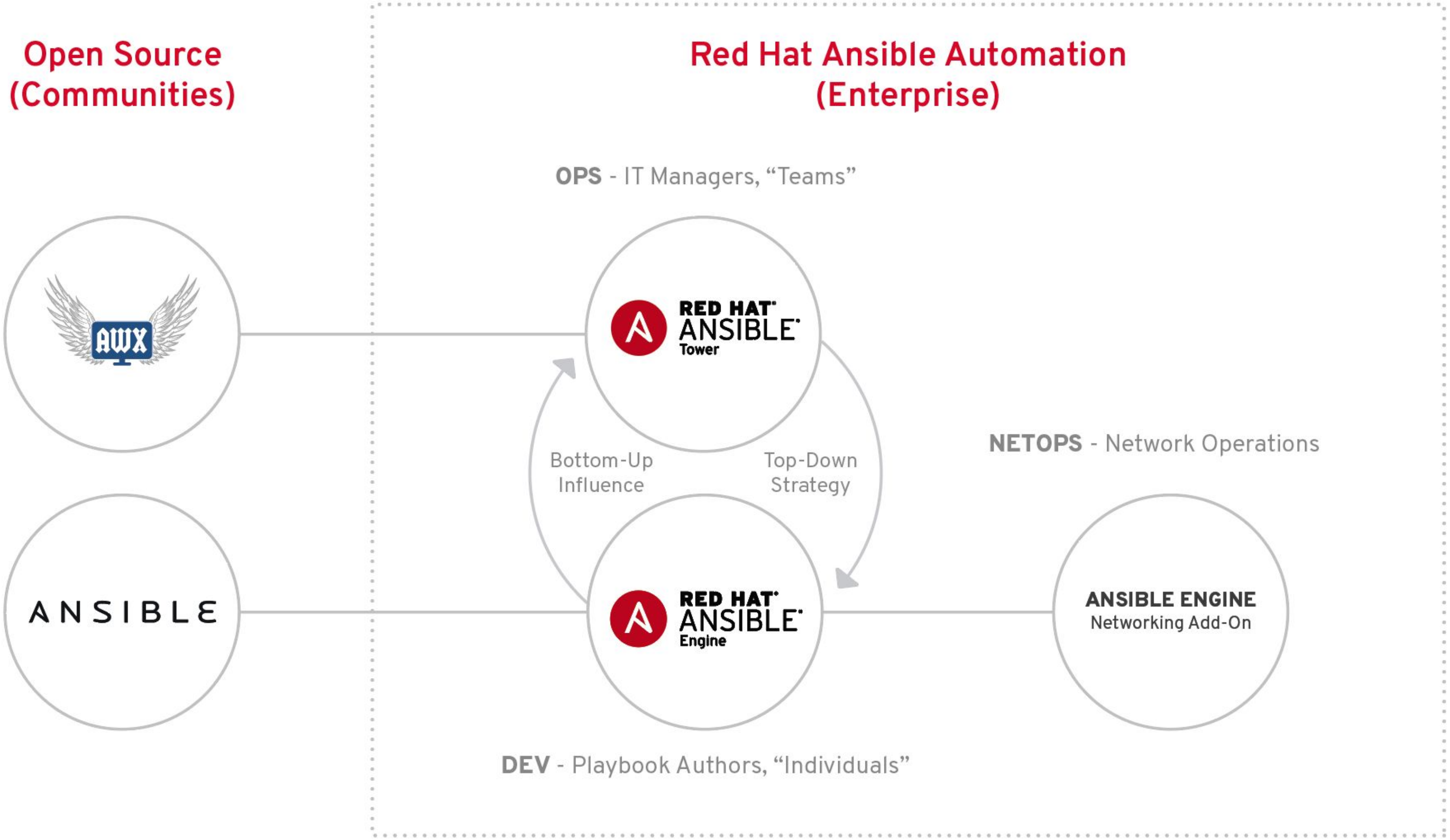
BENEFIT	SDN	AUTOMATION
Reconfigure the network from a central point	✓	✓
Reduced vendor lock in with commodity hardware	?	✓
Leverage existing infrastructure	✖	✓
Programmability	✓	✓
Reduced opex/capex costs	?	✓

NETWORK MODULES: DEVICE ENABLEMENT INCLUDED

- A10
- Apstra
- Arista EOS (cli, eAPI), CVP
- Aruba Networks
- AVI Networks
- Big Switch Networks
- Cisco ACI, AireOS, ASA, IOS, IOS-XR, NX-OS
- Citrix Netscaler
- Cumulus Linux
- Dell OS6, OS9, OS10
- Exoscale
- F5 BIG-IP
- Fortinet FortIOS
- Huawei
- Illumos
- Juniper Junos
- Lenovo
- Ordnance
- NETCONF
- Netvisor
- Openswitch
- Open vSwitch (OVS)
- Palo Alto PAN-OS
- Nokia SR OS
- VyOS

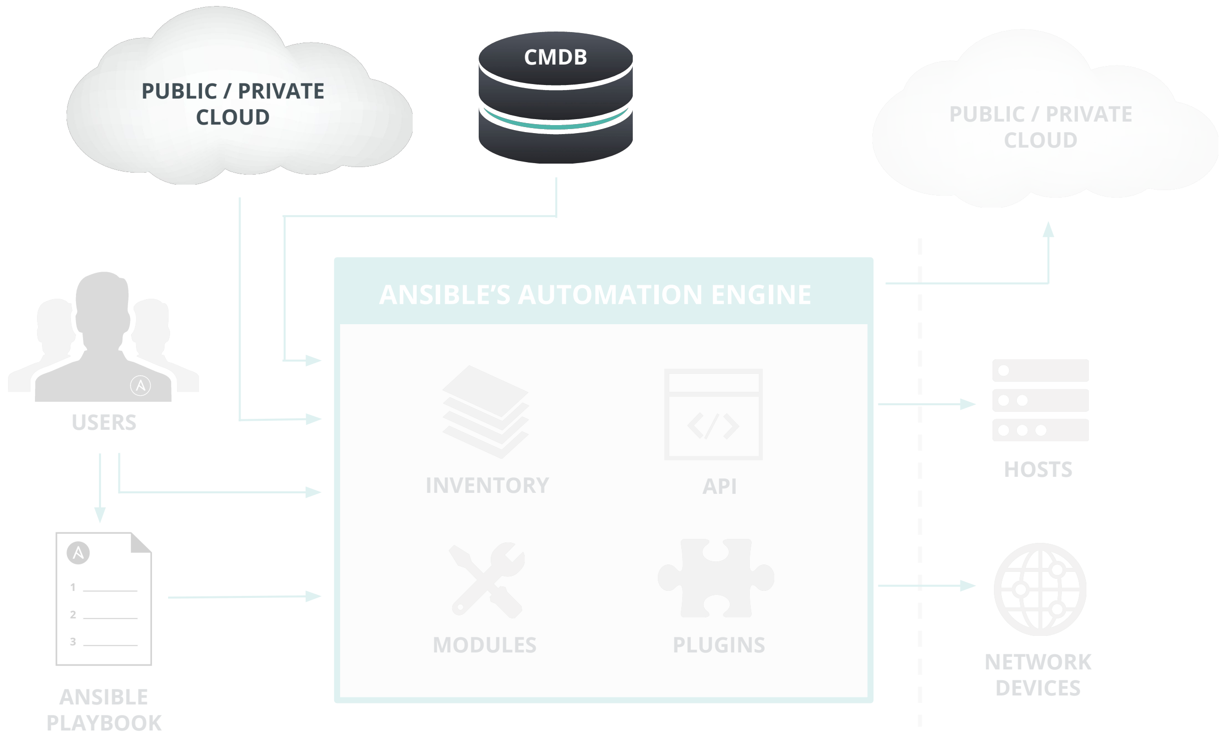
NETWORK AUTOMATION PROGRESS




Open Source
(Communities)
Red Hat Ansible Automation
(Enterprise)
OPS - IT Managers, "Teams"
AWX
RED HAT ANSIBLE Tower
Bottom-Up Influence
Top-Down Strategy
NETOPS - Network Operations
ANSIBLE
RED HAT ANSIBLE Engine
ANSIBLE ENGINE
Networking Add-On
DEV - Playbook Authors, "Individuals"

Playbook Examples

HOW ANSIBLE WORKS

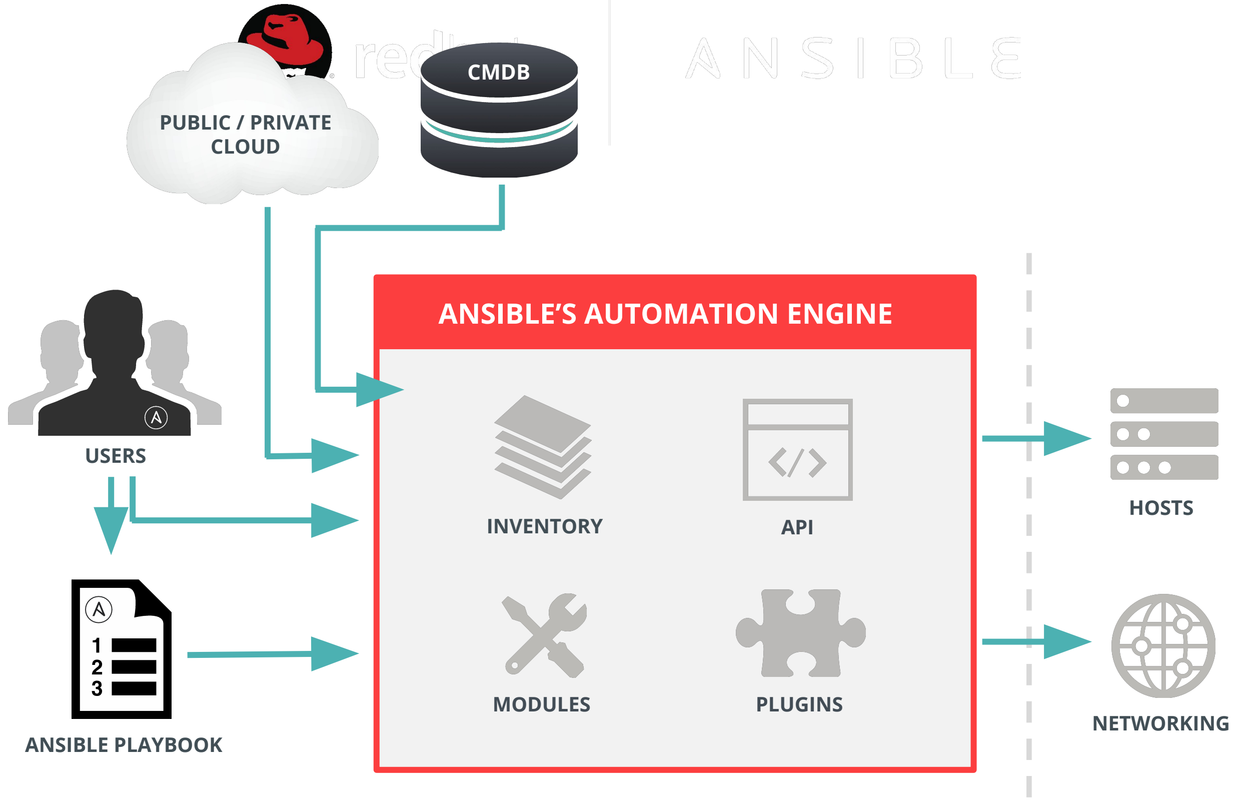


CLOUD

OpenStack, VMware, EC2, Rackspace, GCE, Azure, Spacewalk, Hanlon, Cobbler

CUSTOM CMDB

ANSIBLE UNDER THE HOOD



CONFIG EXAMPLE

```
vars:
   ntp_servers:
     - 10.11.160.238
     - 10.5.27.10
 tasks:
   - name: Set the switch name and domain name
     nxos_config:
       lines:
         - "hostname {{ inventory_hostname }}"
         - ip domain-name lab.eng.rdu.redhat.com
       provider: "{{ cli }}"

   - name: Set the NTP server
     nxos_ntp:
       server: "{{ item }}"
       prefer: enabled
       provider: "{{ cli }}"
     with_items: "{{ ntp_servers }}"
```

CONFIG EXAMPLE

```
vars:
  ntp_servers:
    - 10.11.160.238
    - 10.5.27.10
 tasks:
  - name: Set the switch name and domain name
    nxos_config:
      lines:
        - "hostname {{ inventory_hostname }}"
        - ip domain-name lab.eng.rdu.redhat.com
      provider: "{{ cli }}"

  - name: Set the NTP server
    nxos_ntp:
      server: "{{ item }}"
      prefer: enabled
      provider: "{{ cli }}"
    with_items: "{{ ntp_servers }}"
```

CONFIG EXAMPLE

```
vars:
   ntp_servers:
     - 10.11.160.238
     - 10.5.27.10
 tasks:
   - name: Set the switch name and domain name
     nxos_config:
       lines:
         - "hostname {{ inventory_hostname }}"
         - ip domain-name lab.eng.rdu.redhat.com
       provider: "{{ cli }}"

   - name: Set the NTP server
     nxos_ntp:
       server: "{{ item }}"
       prefer: enabled
       provider: "{{ cli }}"
     with_items: "{{ ntp_servers }}"
```

CONFIG EXAMPLE

```
vars:
   ntp_servers:
     - 10.11.160.238
     - 10.5.27.10
 tasks:
   - name: Set the switch name and domain name
     nxos_config:
       lines:
         - "hostname {{ inventory_hostname }}"
         - ip domain-name lab.eng.rdu.redhat.com

   - name: Set the NTP server
     nxos_ntp:
       server: "{{ item }}"
       prefer: enabled
     with_items: "{{ ntp_servers }}"
```

CONFIG EXAMPLE

```
$ ansible-playbook --ask-vault-pass -i ucso-hosts configure-tor.yml
Vault password:

PLAY [ucso-tor] ****************************************************************

TASK [Set the switch name and domain name] *************************************
ok: [nexus-sw03-mgmt]
ok: [nexus-sw04-mgmt]

TASK [Set the NTP server] ******************************************************
ok: [nexus-sw03-mgmt] => (item=10.11.160.238)
ok: [nexus-sw04-mgmt] => (item=10.11.160.238)
changed: [nexus-sw04-mgmt] => (item=10.5.27.10)
changed: [nexus-sw03-mgmt] => (item=10.5.27.10)

PLAY RECAP *********************************************************************
nexus-sw03-mgmt            : ok=2     changed=1    unreachable=0    failed=0
nexus-sw04-mgmt            : ok=2     changed=1    unreachable=0    failed=0
```

RESOURCE MODULES

```
---
- name: system node properties
  hosts: all

  tasks:
    - name: configure eos system properties
      eos_system:
        domain_name: ansible.com
        vrf: management
      when: network_os == 'eos'

    - name: configure nxos system properties
      nxos_system:
        domain_name: ansible.com
        vrf: management
      when: network_os == 'nxos'

    - name: configure ios system properties
      ios_system:
        domain_name: ansible.com
        lookup_enabled: yes
      when: network_os == 'ios'
```

- Per Platform Implementation

- Declarative by design

- Abstracted over the connection

- Violates DRY principals

- Makes platforms happy

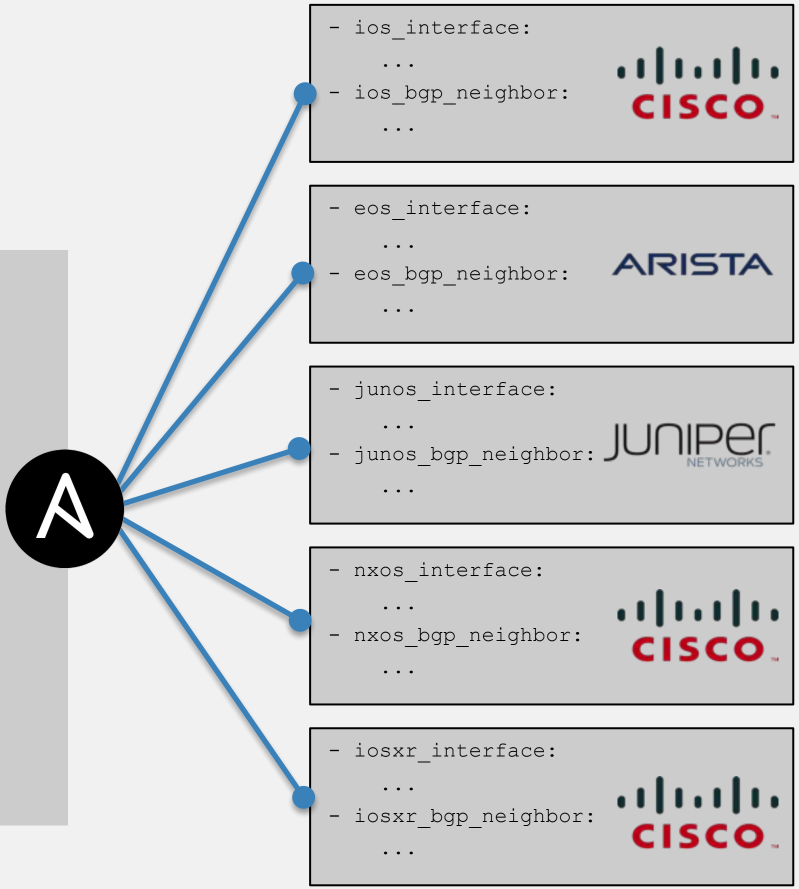
... Not so much for operators

MVPA* MODULES

** Minimum Viable Platform Agnostic*

```
- name: configure network interface
  net_interface
    name: “{{ interface_name }}”
    description: “{{ interface_description }}”
    enabled: yes
    mtu: 9000
    state: up

- name: configure bgp neighbors
  net_bgp_neighbor:
    peers: “{{ item.peer }}”
    remote_as: “{{ item.remote_as }}”
    update_source: Loopback0
    send_community: both
    enabled: yes
    state: present
```



```
- ios_interface:
    ...
- ios_bgp_neighbor:
    ...
```

```
- eos_interface:
    ...
- eos_bgp_neighbor:
    ...
```

```
- junos_interface:
    ...
- junos_bgp_neighbor:
    ...
```

```
- nxos_interface:
    ...
- nxos_bgp_neighbor:
    ...
```

```
- iosxr_interface:
    ...
- iosxr_bgp_neighbor:
    ...
```

DECLARATIVE INTENT

Declared Configuration

Intended State

```
- name: configure interface
  net_interface:
     name: GigabitEthernet0/2
     description: public interface configuration
     enabled: yes
     state: connected
     neighbors:
       - host: core-01
         port: Ethernet5/2/6
```

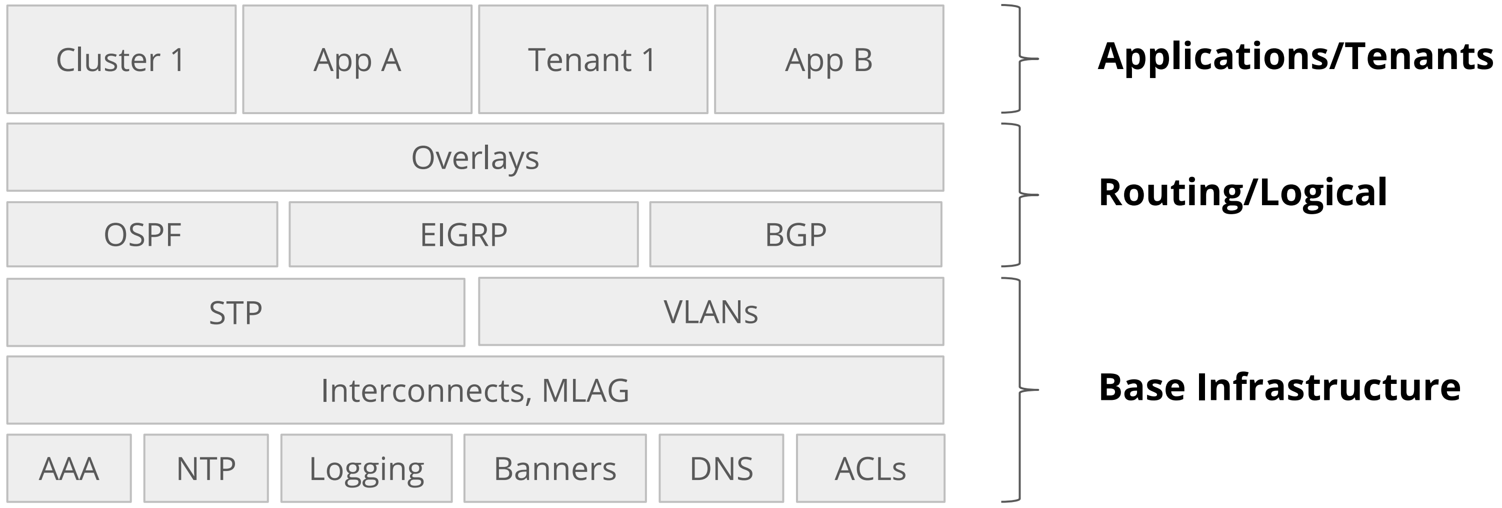
AGGREGATE RESOURCES

```
- name: configure vlans neighbor
  net_vlan:
    vlan_id: “{{ item.vlan_id }}”
    name: “{{ item.name }}”
    state: “{{ item.state | default(‘active’) }}”
  with_items:
    - { vlan_id: 1, name: default }
    - { vlan_id: 2, name: Vl2 }
    - { vlan_id: 3, state: suspend }

- name: configure vlans neighbor
  net_vlan:
    aggregate:
      - { vlan_id: 1, name: default }
      - { vlan_id: 2, name: Vl2 }
      - { vlan_id: 3, state: suspend }
    state: active
    purge: yes
```

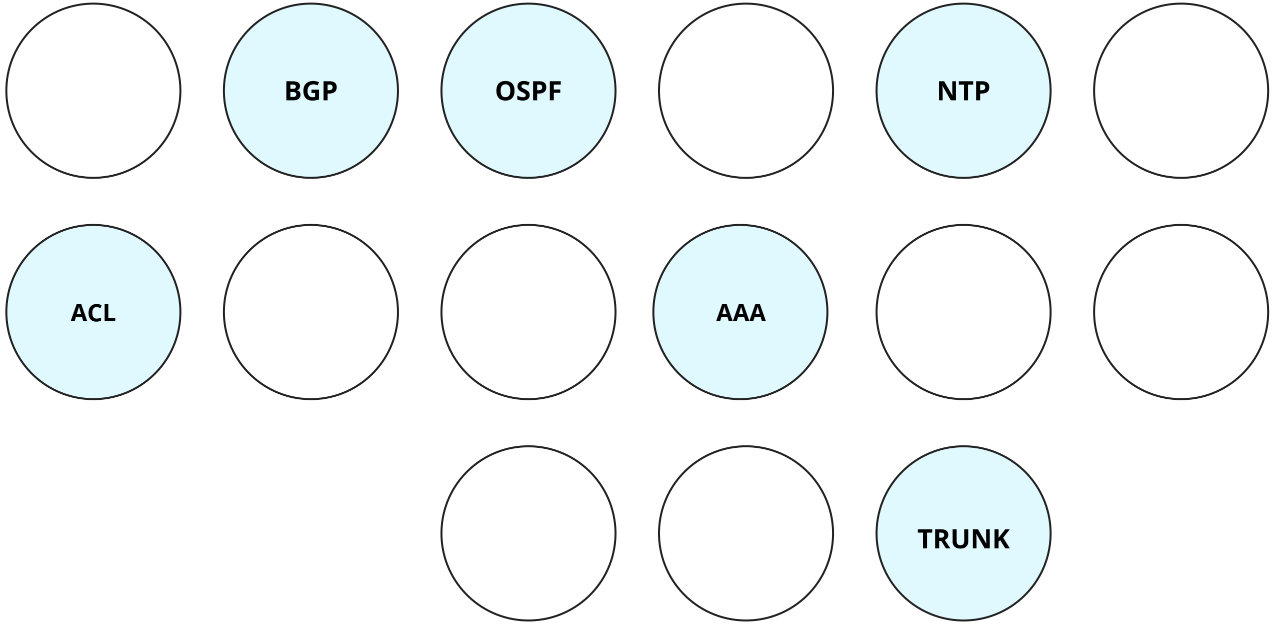
Ansible Best Practices and Concepts

Layered Implementation



Simplifies playbooks, limits blast radius, and facilitates RBAC

Manage Network Applications



Inventory

```
[switches]
spine1
spine2

[switches:vars]
ansible_network_os=nxos

[routers]
juniper1 ansible_network_os=junos
cisco1 ansible_network_os=ios


[network:children]
switches
routers
```

- Enumerates inventory
- Groups hosts by function, location, etc.
- Specify additional attributes

The Anatomy of a Repository

Project Repository

```
├── ansible.cfg
├── inventory/
│   ├── test/
│   │   ├── hosts
│   │   ├── host_vars/
│   │   ├── group_vars/
│   ├── prod/
│   │   ├── hosts
│   │   ├── host_vars/
│   │   ├── group_vars/
├── roles/
│   ├── access_switch/
│   ├── dist_router/
│   ├── tenant_firewall/
├── playbook1.yml
├── playbook2.yml
```

Per-Environment Inventory and Data

Community/Organizational shared code

Repository-Specific Playbooks

Decouple Definition from Implementation

Definition

```
project_tag: foo
tenant_nets:
  - 192.133.157.0/24

fw_outside_ip: 192.133.159.73
fw_inside_ip: 192.133.159.137

vlan_data:
  - { id: 600, name: foo-external }
  - { id: 601, name: foo-provider601 }

svis:
  - { id: 600, cidr: 192.133.157.1/27, vrf: foo, switch: "csn-sjc18
  - { id: 601, cidr: 192.133.157.33/27, vrf: foo, switch: "csn-sjc1

port_data:
  - { desc: "mcp1.titan1", switch: "aa17-n9k-1", interface: "Ethern
  - { desc: "mcp1.titan1", switch: "aa17-n9k-2", interface: "Ethern
```

Define Once

Implementation

```
- name: Creating vlans
    nxos_vlan:
      host: "{{ item[0] }}"
      transport: cli
      vlan_id: "{{ item[1].id }}"
      state:   "{{ item[1].state | default('present') }}"
      admin_state: "{{ item[1].admin | default('up') }}"
      name:    "{{ item[1].name }}"
    with_nested:
      - "{{ vlan_devices | default([]) }}"
      - "{{ vlan_data | default([]) }}"

  - name: Create the SVI interfaces
    nxos_interface:
      host: "{{ item.switch }}"
      transport: cli
      interface: "vlan{{ item.id }}"
      admin_state: up
    with_items: "{{ svi_data | default([]) }}"
```

Apply Many

Source of Truth

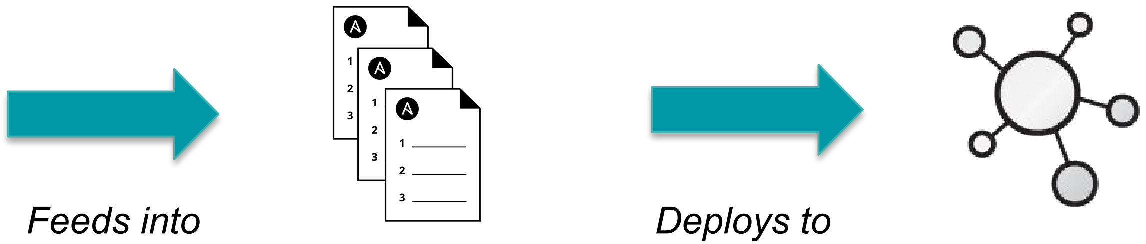
```
system:
 hostname: "{{ inventory_hostname }}"
 domain_name: eng.ansible.com

 source_interface:
   name: Management1
   vrf: default

 domain_lookup: no

 name_servers:
   - 1.1.1.1
   - 2.2.2.2

vlan_data:
  - { id: 600, name: management }
  - { id: 601, name: users }
```



Definition

Implementation

Infrastructure

Source of Truth

Frequent
Configuration
Changes

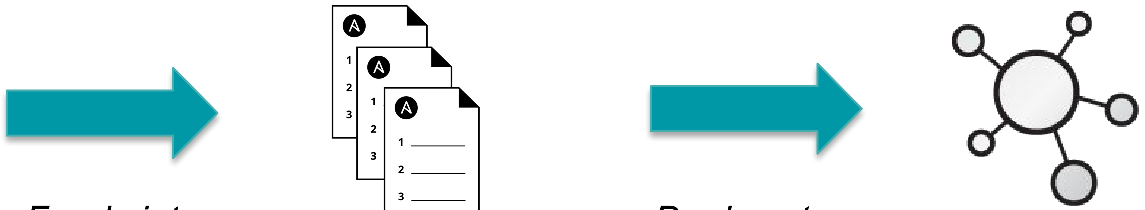
```
system:
 hostname: "{{ inventory_hostname }}"
 domain_name: eng.ansible.com

 source_interface:
   name: Management1
   vrf: default

 domain_lookup: no

 name_servers:
   - 1.1.1.1
   - 2.2.2.2

vlan_data:
  - { id: 600, name: management }
  - { id: 601, name: users }
```



Feeds into

Deploys to

Engineering/
Implementation
Changes

Definition

Implementation

Infrastructure

Facts Cache

```
hostvars[inventory_hostname]:
  interfaces:
    Gi1/0/1:
      description:
"ht3-node1:eth0"
      enabled: True
      mtu: 1500
      mode: trunk
      native_vlan: 99
    Gi1/0/2:
      description:
"ht3-node2:eth0"
      enabled: True
      mtu: 1500
      mode: access
      access_vlan: 10
    Gi1/0/3:
      description:
"ht3-node3:eth0"
      enabled: True
      mtu: 1500
      mode: access
      access_vlan: 10
```

Per-Inventory Item
Facts Cache

Facts Cache

Load SoT from Inventory:

```
host_vars\switch1\interfaces.ym
l
```

or CMDB

Manually load w/Playbook:

```
- include_role:
    name: load_interface_data
```

```
hostvars[inventory_hostname]:
  interfaces:
    Gi1/0/1:
      description:
"ht3-node1:eth0"
      enabled: True
      mtu: 1500
      mode: trunk
      native_vlan: 99
    Gi1/0/2:
      description:
"ht3-node2:eth0"
      enabled: True
      mtu: 1500
      mode: access
      access_vlan: 10
    Gi1/0/3:
      description:
"ht3-node3:eth0"
      enabled: True
      mtu: 1500
      mode: access
      access_vlan: 10
```

Per-Inventory Item
Facts Cache

Facts Cache

Load SoT from Inventory:

```
host_vars\switch1\interfaces.ym
l
```

or CMDB

Manually load w/Playbook:

```
- include_role:
    name: load_interface_data
```

```
hostvars[inventory_hostname]:
  interfaces:
    Gi1/0/1:
      description:
"ht3-node1:eth0"
      enabled: True
      mtu: 1500
      mode: trunk
      native_vlan: 99
    Gi1/0/2:
      description:
"ht3-node2:eth0"
      enabled: True
      mtu: 1500
      mode: access
      access_vlan: 10
    Gi1/0/3:
      description:
"ht3-node3:eth0"
      enabled: True
      mtu: 1500
      mode: access
      access_vlan: 10
```

Per-Inventory Item Facts Cache

Available for Playbooks to reference:

```
- name: Set Interface Attributes
  net_interface
    name: "{{ item }}"
    description: "{{ item.description
}}"
    enabled: "{{ item.enabled }}"
  with_items: "{{ interfaces.keys() }}"
```

Saving Facts

```
hostvars[inventory_hostname]:
  interfaces:
    Gi1/0/1:
      description:
"ht3-node1:eth0"
      enabled: True
      mtu: 1500
      mode: trunk
      native_vlan: 99
    Gi1/0/2:
      description:
"ht3-node2:eth0"
      enabled: True
      mtu: 1500
      mode: access
      access_vlan: 10
    Gi1/0/3:
      description:
"ht3-node3:eth0"
      enabled: True
      mtu: 1500
      mode: access
      access_vlan: 10
```

Per-Inventory Item
Facts Cache

Playbook writes out to inventory:

```
    - name: write out the interfaces vars
      copy:
        dest: "{{ inventory_dir }}/{{ inventory_hostname
}}/interfaces.yml"
        content: "{{ interfaces | to_nice_yaml }}"
```

or write out to CMDB

```
    - include_role:
        name: save_to_cmdb
```

The Role of Roles

```
ios_command
…
ios_vlan
…
ios_interface
```

Set of complex tasks developed by SME

```
include_role:

  name: access_switch
```

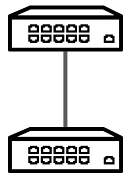
Re-usable, Testable Code by others

Testing Roles

```
-  hosts: access_switches
   roles:
     -  access_switch
```

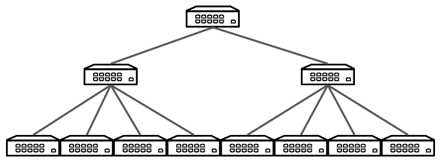
Switch by specifying inventory

```
[access_switches]
```



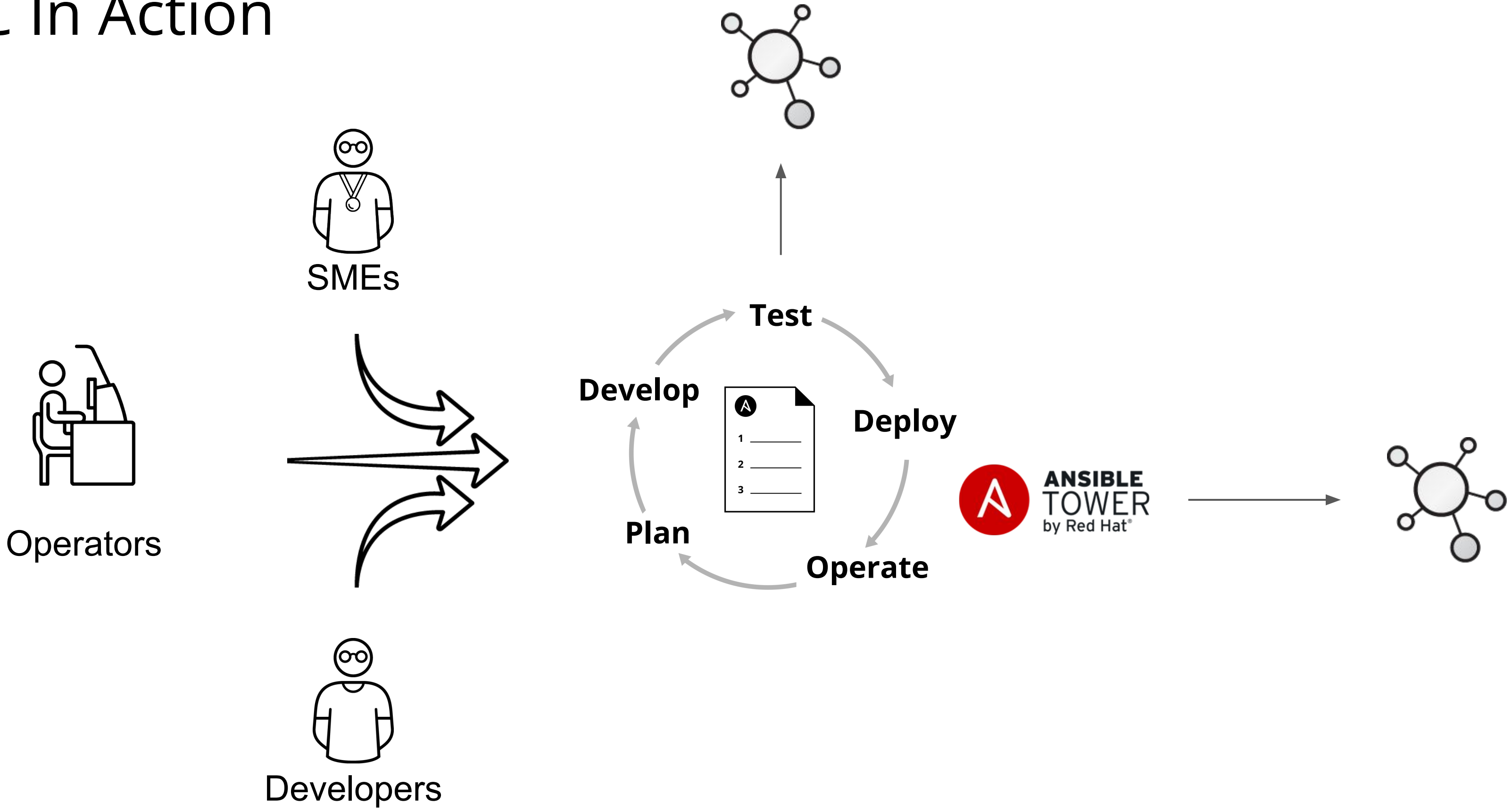
Test

```
[access_switches]
```

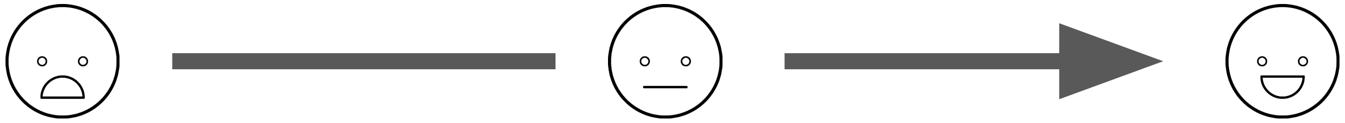


Prod

IaC In Action



From Zero to Hero



Determine:
- Snapshot configs
- Detect unauthorized changes

Standardize:
- Standardize Configs
- Determine Roles
- Manage Applications
- Secure Network

Automate:
- CI for all changes
- Automate testing
- Automate deployment

USE CASES

AUTOMATING MANUAL TASKS

ANSIBLE

Problem:

- Deploying, configuring, and maintaining a network requires many manual tasks by skilled artisans. Configuration issues and unknown changes account for a majority of downtime.

Solution:

- Describe Infrastructure as Code, then use that code to automate and check for deviation.

EXAMPLE: DATA CENTER TENANT NETWORKS

ANSIBLE

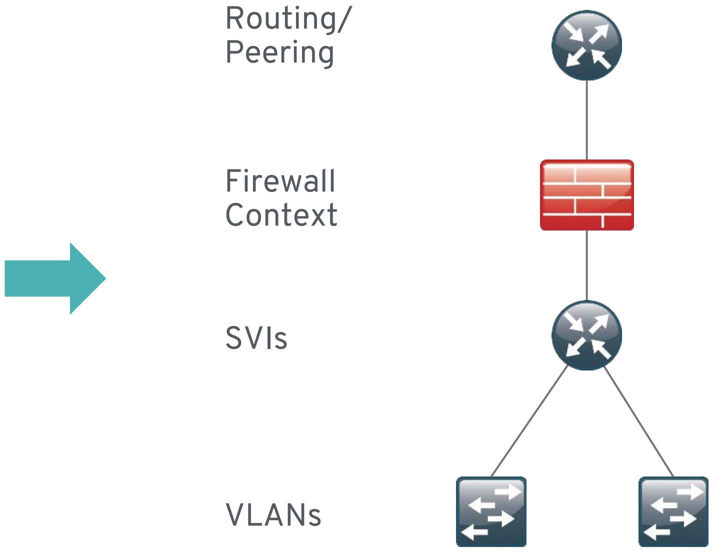
```
project_tag: foo
tenant_nets:
  - 192.133.157.0/24

fw_outside_ip: 192.133.159.73
fw_inside_ip: 192.133.159.137

vlan_data:
  - { id: 600, name: foo-external }
  - { id: 601, name: foo-provider601 }

svis:
  - { id: 600, cidr: 192.133.157.1/27, vrf: foo, switch: "csn-sjc18
  - { id: 601, cidr: 192.133.157.33/27, vrf: foo, switch: "csn-sjc1

port_data:
  - { desc: "mcp1.titan1", switch: "aa17-n9k-1", interface: "Ethern
  - { desc: "mcp1.titan1", switch: "aa17-n9k-2", interface: "Ethern
```



SECURITY

ANSIBLE

Problem:

- Managing policies across different types of hardware and software is difficult and prone to error
- Implementing security requirements (e.g. STIG) across the infrastructure is difficult to implement and maintain

Solution:

- Define the policy once, then apply to multiple infrastructures (e.g. physical, virtual, cloud, network, system, etc.)
- Leverage pre-defined policies and guidelines to implement across the entire infrastructure

EXAMPLE: DEFENSE IN DEPTH

Problem:
different Devices/Vendors require different ACL formats

```
fw_rules:
   - { rule: "public", src_ip: 0.0.0.0/0, dst_ip: 192.133.160.23/32, dst_port: 32400, proto: tcp, action: allow, comment: plex }
   - { rule: "public", src_ip: 0.0.0.0/0, dst_ip: 192.133.160.23/32, dst_port: 1900, proto: udp, action: allow, comment: plex  }
   - { rule: "public", src_ip: 0.0.0.0/0, dst_ip: 192.133.160.23/32, dst_port: 3005, proto: tcp, action: allow, comment: plex  }
   - { rule: "public", src_ip: 0.0.0.0/0, dst_ip: 192.133.160.23/32, dst_port: 5353, proto: udp, action: allow, comment: plex  }
```

Solution:
apply the same abstracted rule set to firewalls, hosts, routers, etc.

```
 - name: Insert ASA ACL
     asa_config:
       lines:
         - "access-list {{ item.rule }} extended {{ item.ac
| ipaddr('network') }}{{ item.dst_ip | ipaddr('network') }}{
       provider: "{{ cli }}"
     with_items: "{{ fw_rules }}"
```

```
- iptables:
    chain: "{{ item.chain | default('INPUT') }}"
    source: "{{ item.src_ip | default(omit) }}"
    destination: "{{ item.src_ip }}"
    destination_port: "{{ item.dst_port }}"
    protocol: "{{ item.proto | default('tcp') }}"
    jump: "{{ 'ACCEPT' if item.action == 'allow' else 'DENY' }}"
    comment: "{{ item.comment | default(omit) }}"
  with_items: "{{ fw_rules }}"
```

PUBLIC/HYBRID CLOUD

ANSIBLE

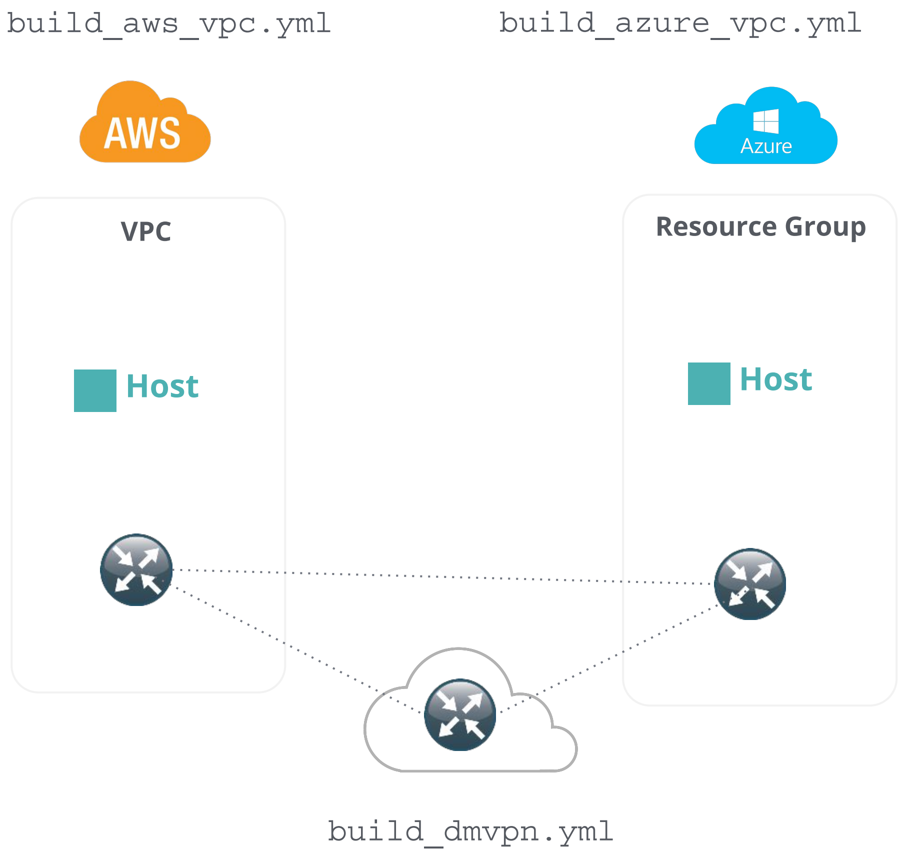
Problem:

- Public/Hybrid cloud increases the number of things to manage
- Cloud things are different than on-prem things and different between clouds increasing complexity

Solution:

- Automate tasks across multiple devices with the same workflow
- Define the policy once, then apply to multiple infrastructures (e.g. physical, virtual, cloud, network, system, etc).

EXAMPLE: ELASTIC CLOUD



1. Create Clouds:

```
ansible-playbook build_aws_vpc.yml
ansible-playbook build_azure_vpc.yml
```

Builds "hosts" file

2. Build DMVPN Overlay:

```
ansible-playbook -i hosts build-dmvpn.yml
```

MONITORING INFRASTRUCTURE

Problem:

- Monitoring Infrastructure gets out of sync with real infrastructure providing little value when problems occur.

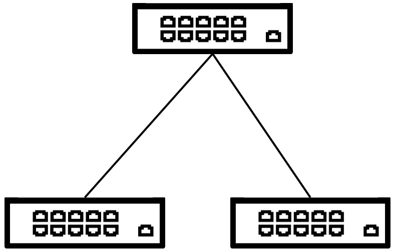
Solution:

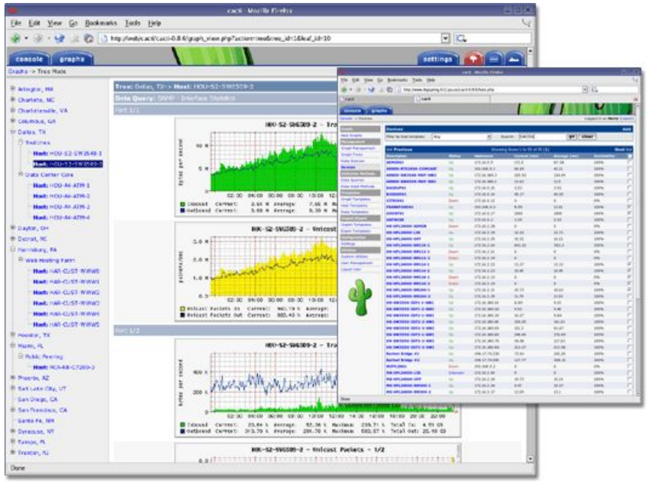
- Automate the configuration of your monitoring infrastructure in parallel with your physical infrastructure to keep them in sync.

EXAMPLE: CONFIGURING MONITORING IN PARALLEL

ANSIBLE

```
port_data:
  - { desc: ″Host_A", switch: ″tor1", interface: "Port-channel17", vpc: 17, port_list: ["Eth1/17"],  port_profile: "ucs-fi" }
  - { desc: ″Host_A", switch: ″tor1", interface: "Port-channel18", vpc: 18, port_list: ["Eth1/18"],  port_profile: "ucs-fi" }
  - { desc: ″Host_B", switch: ″tor2", interface: "Port-channel17", vpc: 17, port_list: ["Eth1/17"],  port_profile: "ucs-fi" }
  - { desc: ″Host_B", switch: ″tor2", interface: "Port-channel18", vpc: 18, port_list: ["Eth1/18"],  port_profile: "ucs-fi" }
```





DEVOPS

ANSIBLE

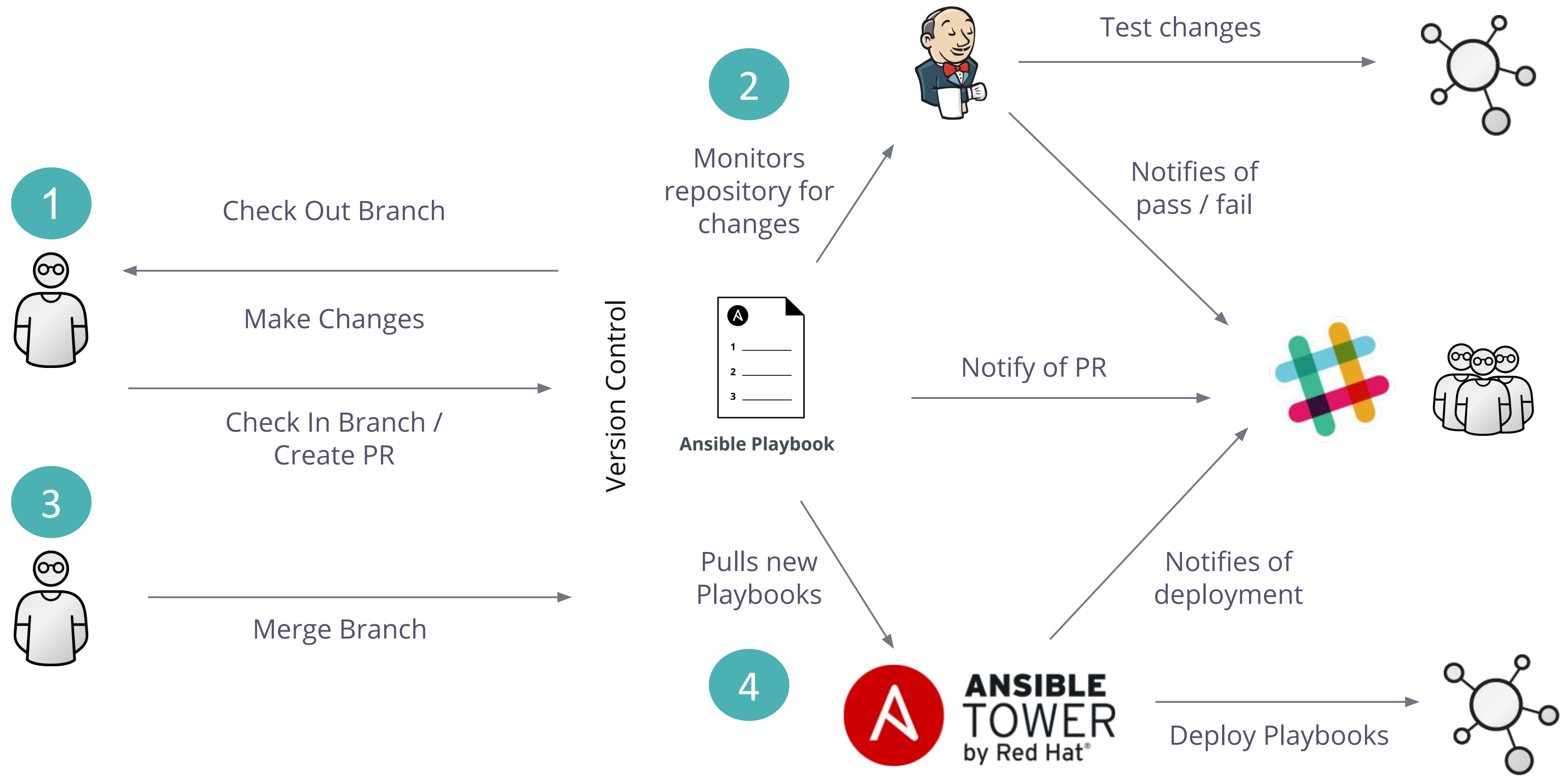
Problem:

- The network, and infrastructure in general is not included in most DevOps workflows causing either a delay in testing or less fidelity

Solution:

- Include the network in the CI workflow with Ansible. Develop and test in the same way as the other elements of the system.
- Increased testing provides greater likelihood that problem will be found/fix sooner

EXAMPLE: NETWORK CI WORKFLOW



WORKFLOW AUTOMATION

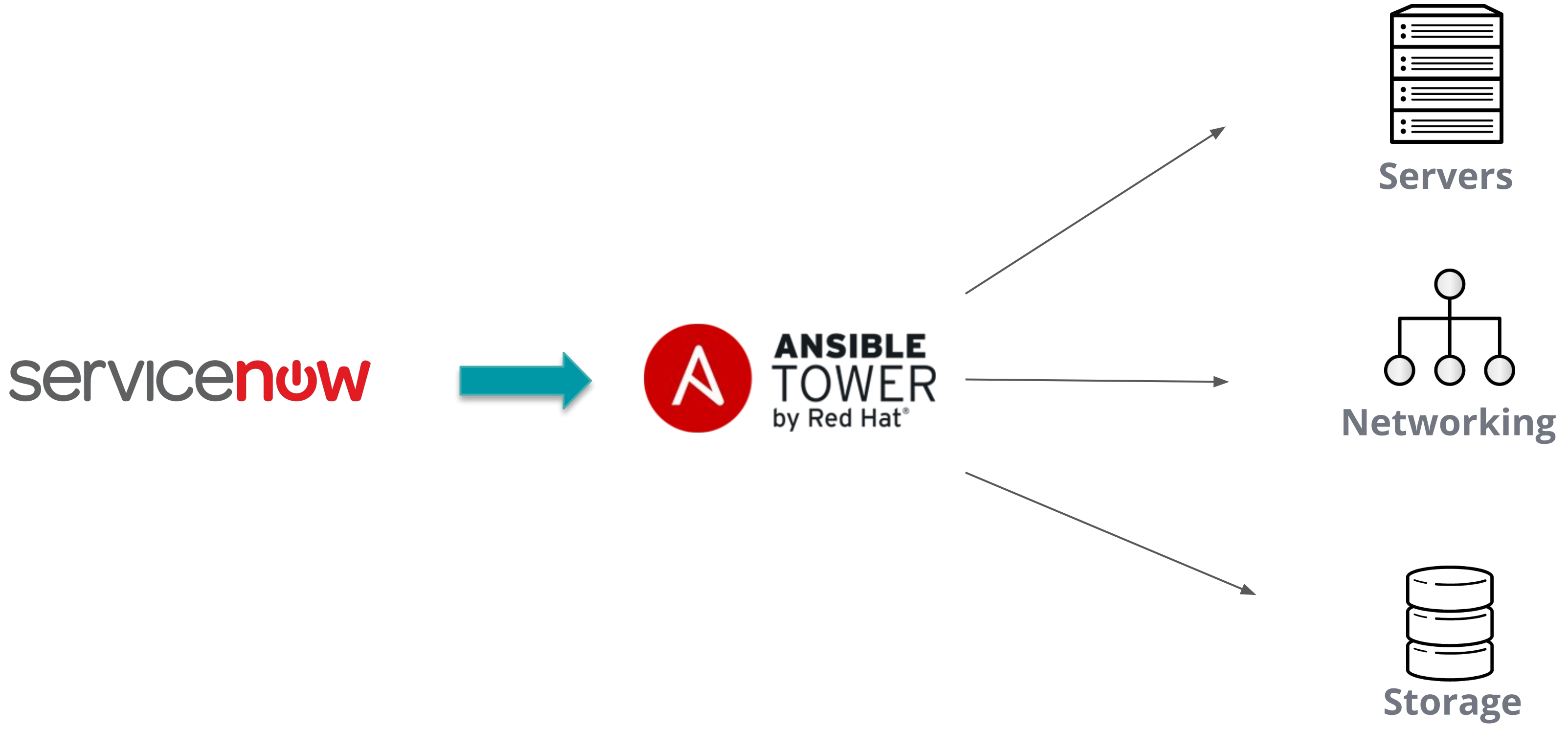
Problem:

- Most enterprises have a ticketing/ approval system for common IT tasks. Once the task goes through the approval process, it ends up in a person's queue for manual action.

Solution:

- Integrate the ticketing system with Ansible Tower's API interface to automatically resolve the issue.

EXAMPLE: API DRIVEN INFRASTRUCTURE



LOG/EVENT ANALYTICS

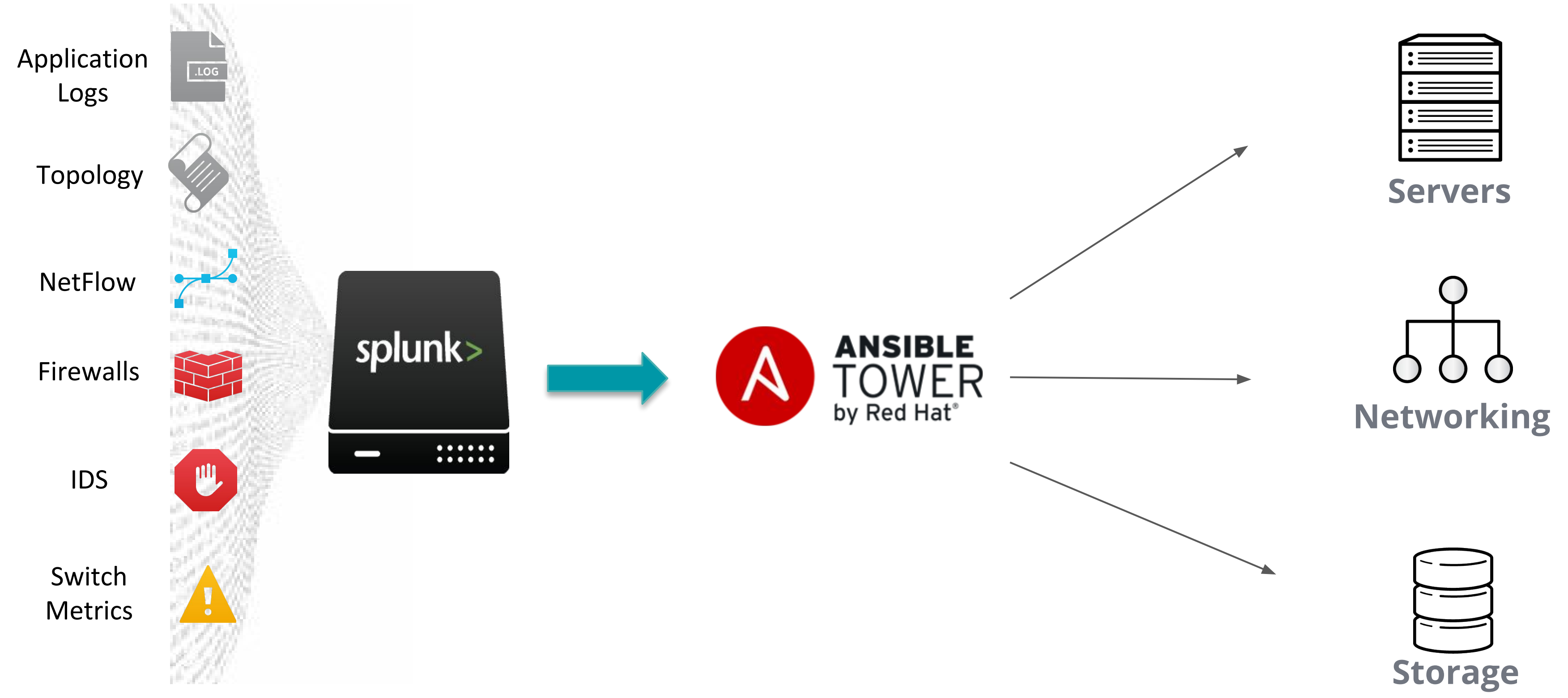
Problem:

- Customers make significant investments into aggregating and analyzing logs/events. The output of these events is relegated to after-the-fact notification and reports.

Solution:

- Use Ansible Tower’s API to allow the event correlation system to respond to events in real time.

EVENT DRIVEN INFRASTRUCTURE



RESOURCES

Ansible Webinars:
https://www.ansible.com/network-automation

Download Ansible 2.4:
http://releases.ansible.com/ansible/

Evaluate Ansible Tower:
http://www.ansible.com/tower-trial/
Email gettingstarted@ansible.com

Join the Community
Users list: ansible-project
Development list: ansible-devel
Announcement list: ansible-announce *(read only)*
irc.freenode.net: #ansible

Todo

- Add slide for Parse CLI
- Add slide for git
- Clean up the playbook examples
- host_vars