Ansible SVCUG Workshop

"I find your lack of faith disturbing."
— Darth Vader

$ whoami

Kirk Byers
Network Engineer:
CCIE #6243 (emeritus)

Programmer:
Netmiko
NAPALM

Teach Python and Ansible
SF Network Automation Meetup

Why should we care about Automation?

Manual work is:
- Tedious
- Error prone
- Relies too much on individual knowledge (i.e. the intelligence is not baked into the system)
- Slow (beyond n-devices)
- Results in lots of variations

"Hokey religions and ancient weapons are no match for a good blaster at your side, kid."

— Han Solo

What is Ansible good at?

The Good:

- Config Management
- Modular
- Systematic

The Bad:

- Complex logic
- complex data structures.

“She may not look like much, but she’s got it where it counts, kid.”

— Han Solo

Can I avoid knowing about programming?

Ansible has two programming languages embedded inside of it.

1. Its own
2. Jinja2

“An elegant weapon for a more civilized age.”

— Obi-Wan Kenobi

Why use a Platform?

- Systematic
- Easier to reuse work of others
- Create automation that endures
- Simplifies concurrency

"I suggest a new strategy, Artoo: Let the Wookiee win."
— C-3PO

General:

1:00 - 6:00PM

Focused
Minimize Distractions
Short Sessions
Exercises and Examples

“Do or do not. There is no try.”

— Yoda

Schedule

1. Introduction
2. Ansible Overview (Historical Roots)
3. YAML
4. Jinja2
5. Inventory / Variables / Facts
6. Ansible modules
7. Show Operations - Cisco IOS/NX-OS
8. Loops
9. Conditionals
10. Config Operations Using Ansible Core
11. NAPALM + Ansible

Collateral Material

https://github.com/ktbyers/ansible-svcug

https://github.com/ktbyers/ansible-svcug/svcug-presentation.pdf

Free Python Course
https://pynet.twb-tech.com/email-signup.html

Ansible Network Automation Course (Paid)
https://pynet.twb-tech.com/class-ansible.html

Ansible Overview (Historical Roots)

- *NIX Server management
- SSH transport
- Assumes Python on box

Implication: connection=local

"I am altering the deal. Pray I don't alter it any further."
— Vader

YAML

- Why do we care about serialization?
- Ansible playbooks are written in YAML
- Reading YAML
- Writing YAML
- Indentation matters

“First, you must unlearn what you've learned.”
— Yoda

Exercises - Section 3

- Get logged into lab environment (if not already done).
- Create a YAML list consisting of four elements (long format)
- Create a YAML list consisting of four elements (condense format)
- Create a YAML dictionary with at least three key-value pairs (long format)
- Create a YAML dictionary named 'routers' that contains another dictionary with three key-value pairs (where the inner key is router_name; value is IP)

Jinja2

Ansible is closely coupled to Jinja2.

What is Jinja2?

Its implications to us?

Why do I have to?   “{{ my_var }}”

“We’re doomed.”

— C-3PO

Inventory / Variables / Facts

Ansible has a large inventory system.

Inventory: ansible-hosts

group_vars and host_vars

Adding other variables into a playbook.

Ansible facts.

“When 900 year old you reach, look as good you will not.”
— Yoda

Exercises - Section 5

- Build a simple inventory file consisting of group ‘local’ and host ‘localhost’
  - Set to ansible_connection=local
  - Set the Python interpreter
- Test your playbook using ‘ansible -m ping local -i ./inventory’
- Expand your inventory to include a ‘cisco’ group with two routers (‘pynet_rtr1’ and ‘pynet_rtr2’). Set the ansible_host of these two devices to cisco1.twb-tech.com and cisco2.twb-tech.com.
- For the Cisco group, set to connection: local and set the Python interpreter.
- Expand your ‘ansible -m ping’ to all devices

Ansible Terms and Modules

- Playbooks
- Plays
- Tasks
- Modules

Ansible Fundamentals: Putting it all together

Playbook (YAML)

Inventory System

Jinja2 (Variable System + Templating)

Plays / Tasks / Modules

Executing Ansible

Our First Script (Section 6)

```
$ cat script1.yml
---
- name: Our first script
  hosts: local
  tasks:
    - ping:
```

```
$ ansible-playbook script1.yml
```

Add -vvv for more verbose

Introducing Debug and Set Fact

```
---
- name: Introducing debug
  hosts: local
  tasks:
    - name: Print out something
      debug:
        msg: Hello world
```

Introducing Debug and Set Fact

```
---
- name: Introducing set_fact
  hosts: local
  tasks:
    - name: Set a variable
      set_fact:
        router1: 1.1.1.74
    - name: Print out new variable
      debug:
        msg: "{{ router1 }}"
```

More Variables

```
---
- name: More variables
  hosts: local
  vars:
    ntp_server1: 1.1.1.1
    ntp_server2: 2.2.2.2
  tasks:
    - name: Print out variables
      debug:
        msg: "{{ ntp_server1 }} {{ ntp_server2 }}"
```

Exercises - Section 6

- In a playbook define two NTP servers, two DNS servers, and a default domain. Run the playbook against the ‘local’ group.
- Use debug to print out the two DNS servers.
- Use set_fact to define a third DNS server.
- Use debug and ‘var’ argument to print out this third DNS server.

More Inventory (Section 6)

Problem: Inventory file does not scale well as it gets larger in size.

Solution1:

- host_vars
- group_vars

Solution2:

- Dynamic Inventory

Exercises - Section 6

- Create a new directory. In that directory, create both ‘group_vars’ and ‘host_vars’.
- For group_vars define an all.yml file that contains two DNS servers and a default domain.
- For the ‘cisco’ group define a group_vars variable named ‘common_vlans’ that specifies a list of five VLAN IDs.
- For pynet-rtr1 and pynet-rtr2 define a unique_vlans variable that contains a list of VLANS containing three unique VLAN IDs.
- Create a playbook that prints out all of these variables using debug

Ansible Modules

http://docs.ansible.com/ansible/latest/list_of_network_modules.html

Common Modules used in Networking:

Ansible Core Modules: platform_facts, platform_command, platform_config

NAPALM-Ansible

NTC-Ansible

Network Show Operations (Section 7)

```
- hosts: cisco
  vars:
    ssh_provider:
        host: "{{ ansible_host }}"
        username: "{{ username }}"
        password: "{{ password }}"
        timeout: 30
  tasks:
    - ios_facts:
        provider: "{{ ssh_provider }}"
```

Fact Gathering NX-API

```
nxapi_provider:
    host: "{{ ansible_host }}"
    username: "{{ username }}"
    password: "{{ password }}"
    transport: nxapi
    use_ssl: yes
    validate_certs: no
    port: 8443
    timeout: 30
```

Exercises - Section 7

- Gather facts on one of the Cisco IOS / IOS-XE devices
- Gather facts on one of the NX-OS devices using NX-API*
- Use debug to print out the ‘ansible_net_model’ for each device
- Use group_vars to store the providers

ios_command

```
---
- name: Execute show commands
  hosts: cisco
  tasks:
    - ios_command:
        provider: "{{ ssh_provider }}"
        commands: show ip int brief
      register: output
```

nxos_command

```
---
- name: Execute show commands
  hosts: nxos
  tasks:
    - nxos_command:
        provider: "{{ nxapi_provider }}"
        commands: show ip arp vrf management
      register: output_api
```

Exercises - Section 7 (_command)

- Execute ‘show ip interface’ on one of the Cisco IOS/IOS-XE devices. Save the output of this command to a variable.

- Process the ‘stdout_lines’ key in the output variable and use the debug module to print this to the screen.

with_items (for loops) [Section 8]

Ansible Structure

```
with_items:
    - router1
    - router2

with_items: "{{ my_list }}"
```

Python Equivalent

```
for item in my_list:
    print(item)
```

"Don't you call me a mindless philosopher, you overweight glob of grease!"
— C-3PO

with_items (for loops)

```
- name: Loops
  hosts: local
  tasks:
    - debug:
        msg: "{{ item }}"
      with_items:
        - router1
        - router2
        - router3
```

Exercises - Section 8

- Construct a “vars” data structure that is a list of three routers. Each list element should be a dictionary with a router name, device type, and IP address.

- Use a with_items for loop to loop over this data structure and print out the router name and IP address.

when (conditionals) [Section 9]

Conditionally execute tasks:

```
- name: Substring in larger string
  debug:
    msg: This is Cisco IOS
  when: "'Cisco IOS' in version"
```

Config Operations using Ansible Core

```
- hosts: cisco
  vars:
    dns1: 8.8.8.8
    dns2: 8.8.4.4
  tasks:
    - ios_config:
        provider: "{{ ssh_provider }}"
        lines:
          - "ip name-server {{ dns1 }}"
          - "ip name-server {{ dns2 }}"
```

Exercises - Section 10

- On one of the Cisco devices configure two DNS servers, two NTP servers, and a default domain-name.

- All of your configuration variables should be in group_vars/all.yml

Configuration Templating (barely)

```
---
- name: Configure General Items
  hosts: pynet-rtr1
  tasks:
    - ios_config:
        provider: "{{ ssh_provider }}"
        src: "{{ inventory_hostname }}.txt"
```

“That’s no {{ moon }}. It’s a space station.”
— Obi-Wan Kenobi

https://pynet.twb-tech.com/blog/ansible/ansible-cfg-template.html

Config with Hierarchy

```
- ios_config:
    provider: "{{ ssh_provider }}"
    parents: ["ip access-list extended TEST-ACL"]
    lines:
      - permit ip host 1.1.1.1 any log
      - permit ip host 2.2.2.2 any log
      - permit ip host 3.3.3.3 any log
    before: ["no ip access-list extended TEST-ACL"]
    replace: block
    match: line
```

NAPALM + Ansible (Section 11)

Purpose of NAPALM: create a standard set of operations across a range of platforms.

Operations fall into two general categories: Config Operations + Getter Operations.

"Somebody has to save our skins. Into the garbage chute, flyboy!"
— Leia Organa

NAPALM Vendors

CORE

Arista EOS

Cisco IOS

Cisco IOS-XR

Cisco NX-OS

Juniper Junps

COMMUNITY

Fortinet Fortios

Mikrotik RouterOS

Palo Alto NOS

Pluribus

VyOS

NAPALM Ansible Modules

Current

napalm_validate.py
napalm_get_facts.py
napalm_ping.py
napalm_install_config.py

Future YANG (experimentals)

napalm_diff_yang.py
napalm_parse_yang.py
napalm_translate_yang.py

NAPALM Getters

get_facts
get_environment
get_snmp_information
get_ntp_peers
get_ntp_stats
get_mac_address_table
get_arp_table
get_interfaces
get_interfaces_ip
get_lldp_neighbors
get_lldp_neighbors_detail
get_bgp_neighbors
get_bgp_neighbors_detail
get_bgp_config
get_route_to
get_probes_config
get_probes_results
get_users
get_optics

NAPALM Getters

```
- name: NAPALM on IOS
  hosts: pynet-rtr1:csr1
  tasks:
   - name: NAPALM facts
     napalm_get_facts:
       hostname: "{{ ansible_host }}"
       username: "{{ username }}"
       password: "{{ password }}"
       dev_os: "ios"
```

"These aren't the droids you're looking for."

-- Obi-Wan Kenobi

NAPALM Config Operations

```
device.load_merge_candidate()
device.load_replace_candidate()

device.compare_config()
device.discard_config()

device.commit_config()

device.rollback()
```

NAPALM Config Operations

```
tasks:
  - napalm_install_config:
      provider: "{{ creds }}"
      config_file: "CFGS/{{ inventory_hostname }}.txt"
      commit_changes: False
      replace_config: True
      get_diffs: True
      diff_file: "DIFFS/{{ inventory_hostname }}.diff"
```

Exercises - Section 11

Configure an IP interface on two of the CSR routers using a merge operation (don’t change GigabitEthernet1). You should be able to ping between the two routers when done.

Generate a diff before committing the change.

Exercises - Section 11

Configure eBGP between two of the CSR routers. The AS number should match the router number so “csr1” should be AS1.

Use get_bgp_neighbors and napalm_get_facts to verify BGP neighbor relationship.

"When I left you I was but the learner. Now I am the master."
— Darth Vader

Questions?

ktbyers@twb-tech.com
Twitter: @kirkbyers