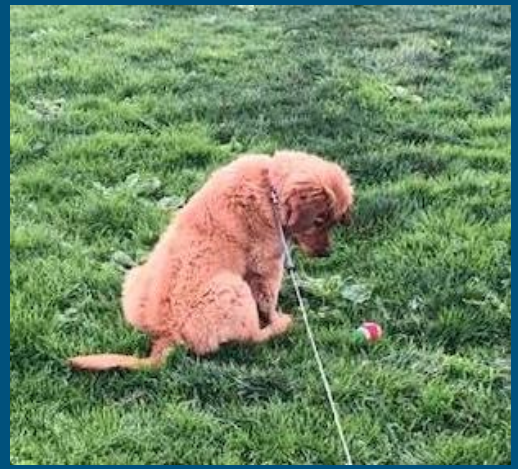
Ansible Network Automation

Live Training Session
December 2021

Day4 Schedule - Roles / Parsers / Debugging / Vault

Roles
Ansible and Network Parsers
Dynamic Inventory - Some Python Required
Ansible Lookups / Filters / Callback Plugins
Ansible Debugging
Using Vault
Creating your own Ansible Filter - Some Python Required
Creating your own Ansible Module - Some Python Required

What exactly do you want me to do with this ball thingamajig?



Roles - A predefined directory structure where Ansible “knows” where to look for things.

```
roles/
    common/
        tasks/
        handlers/
        library/
        files/
        templates/
        vars/
        defaults/
        meta/
```

Reference Material in:

{{ github_repo }}/roles

https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html#role-directory-structure

ANSIBLE

Roles

```
$ tree -C roles/common/
roles/common/
├── defaults
├── files
├── handlers
│   └── main.yml
├── library
├── meta
├── tasks
│   └── main.yml
├── templates
└── vars
    └── main.yml

8 directories, 3 files
```

Role name: common

Location for files you copy (for example, the copy module)

Look for handlers at roles/{{ role_name }}/handlers/main.yml

Custom modules can be distributed as part of the role

Automatically look for tasks at roles/{{ role_name }}/tasks/main.yml

Location for Jinja2 Templates (template module)

Automatically look for variables at roles/{{ role_name }}/vars/main.yml

Executing Roles

```yaml
---
- name: Original way for using roles
  hosts: vmx1
  gather_facts: False
  roles:
    - common
```

Reference Material in:

{{ github_repo }}/roles

Exercises:
./day4/roles/ex1.txt
./day4/roles/ex2.txt

```yaml
---
- name: IOS Example
  hosts: cisco
  gather_facts: True
  tasks:
    - name: Use newer import_role form
      ansible.builtin.import_role:
        name: core_router4
      when: True
      tags: foo1

    - name: Use dynamic instead of static form
      ansible.builtin.include_role:
        name: core_router4
      when: False
      tags: foo2
```

Reference Material in:
{{ github_repo }}/parsers

Ansible and Network Parsers

TextFSM - Regex + Finite State Machine; leverage ntc-templates to use existing parsers.

Genie - Part of Cisco pyATS framework (black-box, send raw text in and get structured data out).

Regex - Parse yourself using regular expressions.

*TextFSM was previously made available as a filter plugin. Ansible users should transition to the cli_parse module.

Reference Material in:
{{ github_repo }}/parsers

Ansible and Network Parsers - TextFSM

```yaml
- name: TextFSM Example
  hosts: cisco5
  gather_facts: False
  vars:
    fsm_template: "cisco_ios_show_ip_interface_brief.template"

  tasks:
    - name: Executing command
      cisco.ios.ios_command:
        commands: show ip int brief
      register: output

    - ansible.builtin.set_fact:
        show_ip: "{{ output.stdout[0] | parse_cli_textfsm(fsm_template) }}"

    - ansible.builtin.debug:
        var: show_ip

    - ansible.builtin.debug:
        msg: "{{ item['INTF'] }}"
      loop: "{{ show_ip }}"
```

Using a filter

Reference Material in:
{{ github_repo }}/parsers

Ansible and Network Parsers - TextFSM

```yaml
- name: TextFSM Example4
  hosts: cisco5
  gather_facts: True
  vars:
    platform: "cisco_ios"
    command: "show ip int brief"

  tasks:
    - name: Executing command
      cisco.ios.ios_command:
        commands: "{{ command }}"
      register: output

    - ansible.builtin.set_fact:
       show_ip: "{{ output.stdout[0] | ntc_parse(command, platform) }}"

    - ansible.builtin.debug:
        var: show_ip
```

Using ntc_parse

Reference Material in:
{{ github_repo }}/parsers

Ansible and Network Parsers - Genie Filter

```yaml
tasks:
  - name: Executing command
    cisco.ios.ios_command:
      commands: "{{ command }}"
    register: output

  - ansible.builtin.debug:
      var: output

  - ansible.builtin.set_fact:
      output_struct: "{{ output.stdout[0] | clay584.genie.parse_genie(command=command, os='iosxe') }}"

  - ansible.builtin.debug:
      msg: "{{ output_struct }}"
```

Exercises:
./day4/parsers/ex1.txt
./day4/parsers/ex2.txt

Dynamic Inventory - Some Python Required (optional)

Requirement: Program must support –list argument

```
$ ./dyn_inv.py --list
```

Output the groups, hosts that belong to each group, group variables, and child groups (all as JSON)

```
$ ./dyn_inv.py --host arista6 | jq "."
{
  "ansible_host": "arista6.lasthop.io"
}
```

For each host in the --list output, you can retrieve the host variables by executing --host {hostname}. Output the host variables in JSON.

Reference Material in:

{{ github_repo }}/dynamic_inventory

Also can support “_meta” keyword to avoid excessive number of host calls.

Using Dynamic Inventory

```
$ ansible-playbook vlans_eos.yml -i ./dyn_inv.py -k
```

Pointing to an Inventory Directory

```
$ ansible-playbook test_pb.yml -i my_inventory/ -k
```

Exercises:
./day4/dynamic_inventory/ex1.txt

ANSIBLE

Ansible Lookups and Filters

Reference Material in:
{{ github_repo }}/filters

Reference Material in:
{{ github_repo }}/lookups

Exercises:
./day4/debug/ex1.txt

```yaml
- name: Test lookups
  hosts: local
  vars:
    number: "22"
    my_dict:
      key1: val1
      key2: val2
      key3: val3
      key4: val4

  tasks:
    - ansible.builtin.debug:
        var: number
      when: number | int == 22

    - ansible.builtin.debug:
        msg: "{{ my_dict | to_nice_yaml(indent=4) }}"

    - ansible.builtin.debug:
        msg: "{{ my_dict | to_nice_json(indent=4) }}"

    - ansible.builtin.set_fact:
        data: "{{ lookup('file', 'ip_addresses.yml') | from_yaml }}"
```

Ansible Callback Plugins

```
stdout_callback = community.general.yaml
# stdout_callback = community.general.unixy
# stdout_callback = selective
# stdout_callback = dense
# stdout_callback = debug
```

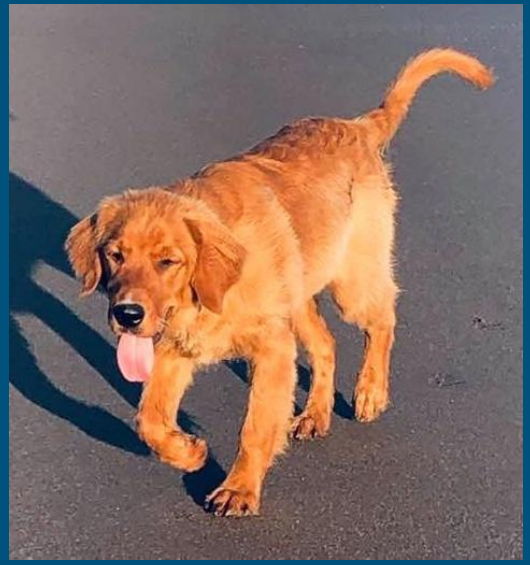
```
[defaults]
inventory = ~/ansible-hosts.ini
#library = ~/ansible-extras/ntc-ansible/library:~/ansible-dec
library = ~/ansible-extras/ntc-ansible/library
filter_plugins = ~/ansible-extras/ntc-ansible/filter_plugins
host_key_checking = False
retry_files_enabled = False
action_warnings = False
deprecation_warnings = False
stdout_callback = debug
```

Change the Ansible output behavior

Reference Material in:
{{ github_repo }}/callbacks

Ansible Debugging

1. Debugging/Troubleshooting Ansible is hard.
2. Set the stdout_callback to one that is easier to read.
3. Read the actual error message. Look closely at the location Ansible is complaining about.
4. --syntax-check might help.
5. Double check your YAML and playbook structure.
6. You can use ansible-inventory utility to check your inventory.
7. Simplify your problem.



Exercises:
./day4/debug/ex1.txt

Reference Material in:
{{ github_repo }}/debug_tshoot

Using Ansible Vault

```
$ cat commands
 ansible-vault create new.yml
 ansible-vault edit new.yml
 ansible-vault encrypt creds.yml
 ansible-vault decrypt creds.yml
 ansible-vault view creds.yml
```

Reference Material in:
{{ github_repo }}/vault

Exercises:
./day4/vault/ex1.txt

Using Ansible Vault

```
$ ansible-vault create new.yml
New Vault password:
Confirm New Vault password:
```

```
$ ansible-vault view new.yml
Vault password:
---
- 1.1.1.1
- 1.1.1.2
- 1.1.1.3
- 1.1.1.4
- 1.1.1.5
```

```yaml
vars:
  ansible_user: pyclass
  ansible_ssh_pass: !vault |
        $ANSIBLE_VAULT;1.1;AES256
        38386462393934316437636362303962636533366161393366363330646432353261653930623865
        33653466626262666639346431333264323635643964616 10a6466613930633665636436623139 39
        35396333616462373839363735613463356133333638343133356462643935666230623165646133
        61353038653737663 10a30383934613063303838643139343862306237353363626464306364656 5
        6430
```

Using Vault

```
deprecation_warnings = False
stdout_callback = community.general.yaml
vault_password_file = ~/vault-pass
```

```
$ ansible-playbook vault1.yml --ask-vault-password
```

```
$ ansible-playbook vault1.yml --vault-password-file ./vault-pass
```

```
$ export ANSIBLE_VAULT_PASSWORD_FILE=/home/ktbyers/ansible-dec21/vault/vault-pass
```

Creating your own Ansible Filter - Some Python Required (optional)

```python
def some_filter(my_string):
    return my_string.upper()


class FilterModule:
    def filters(self):
        return {"some_filter": some_filter}
```

Exercises:
./day4/custom_filters/ex1.txt
./day4/custom_filters/ex2.txt

Reference Material in:
{{ github_repo }}/filters_create

Reference Material in:
{{ github_repo }}/filters_role_create

```python
def another_filter(my_string, arg1, arg2):
    return f"{my_string}...{arg1}...{arg2}"


class FilterModule(object):
    def filters(self):
        return {"another_filter": another_filter}
```

Creating your own Ansible Module - Some Python Required (optional)

```python
def main():

    # Define your modules arguments
    module_args = dict(
        name=dict(type="str", required=True),
        new=dict(type="bool", required=False, default=False),
    )

    # Create an instance of the AnsibleModule class
    module = AnsibleModule(argument_spec=module_args, supports_check_mode=True)

    # Define standard results
    result = {
        "changed": False,
        "original_message": "Something",
        "message": f"name: {module.params['name']}; new: {module.params['new']}...it worked!!!"
    }

    # Return items as JSON
    module.exit_json(**result)
```

Creating your own Ansible Module - Some Python Required (optional)

```python
# Ensure Netmiko is installed; exit using JSON (if not).
if not netmiko_found:
    module.fail_json(msg="The Netmiko library is not installed!")

# Extract the arguments
host = module.params["host"]
device_type = module.params["device_type"]
username = module.params["username"]
password = module.params["password"]
command = module.params["command"]

net_connect = ConnectHandler(
    host=host,
    device_type=device_type,
    username=username,
    password=password
)
output = net_connect.send_command(command)
result["msg"] = output
module.exit_json(**result)
```

Reference Material in:

{{ github_repo }}/module_create

Exercises:
./day4/custom_module/ex1.txt

The end...

Questions?

ktbyers@twb-tech.com