Introduction to Ansible by RedHat

What is Ansible by RedHat?

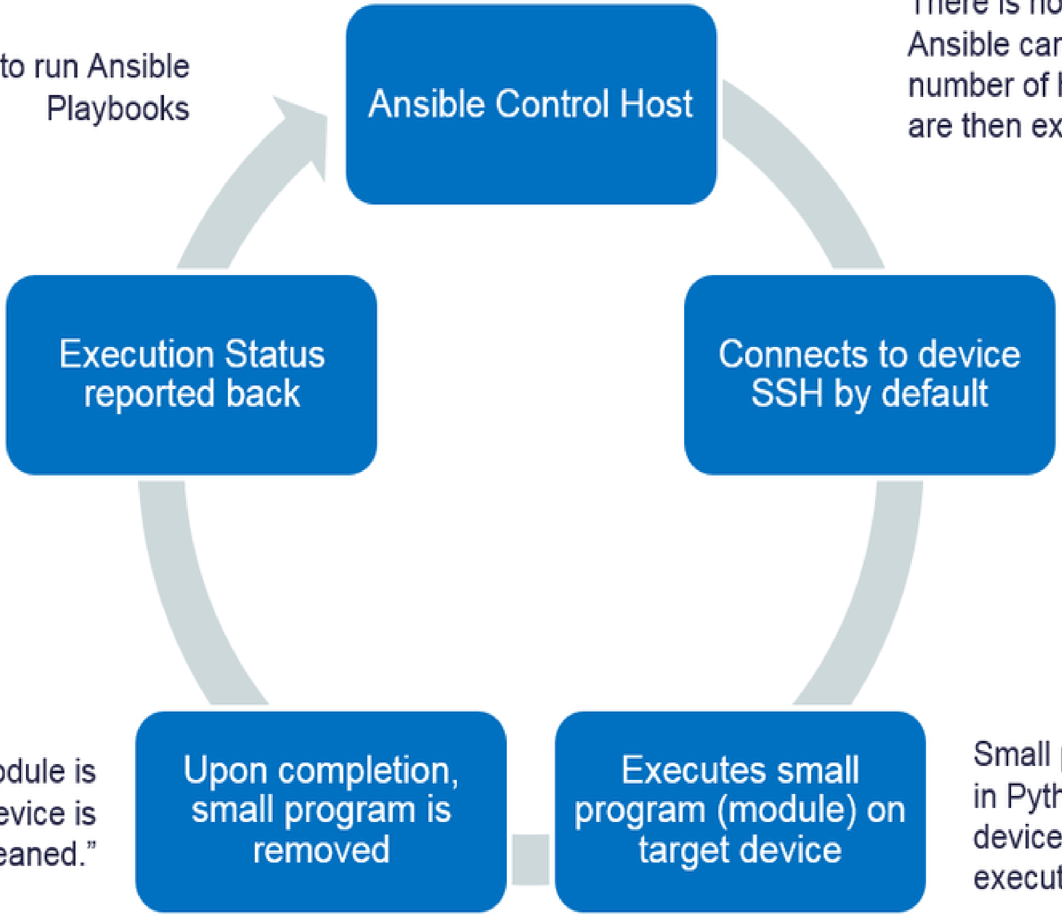
- Founded in 2012
- Acquired by RedHat in 2015
- Open source DevOps Configuration management, automation, and orchestration platform
- Low barrier to entry with no programming skills necessary
- Batteries included
- 900+ modules
- Built its home for application deployments in cloud environments
- Rapidly gaining traction for network automation

Diving into Ansible

- Written in Python
- Extended in any language (not common for open source modules)
- Native integration with Jinja2 templates
- Automation instructions are defined in YAML
- **Agentless**

We cover Jinja2 and YAML in the course.

How Ansible Works



There is no official control host. Ansible can be installed on any number of hosts. Playbooks are then executed.

Ansible uses SSH by default to connect to the device. 3rd party API integrations are possible with custom development, which is required for Cloud, Network, etc.

Small program module written in Python is transferred to the device (by default) and executed.

The program module is removed and device is “cleaned.”

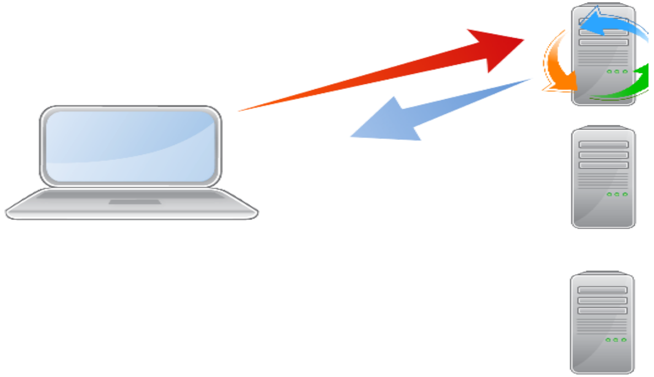
On the control host, there is visual feedback, with verbose options to see the status of the tasks (data returned from module).

Continue to run Ansible Playbooks

How Ansible Works

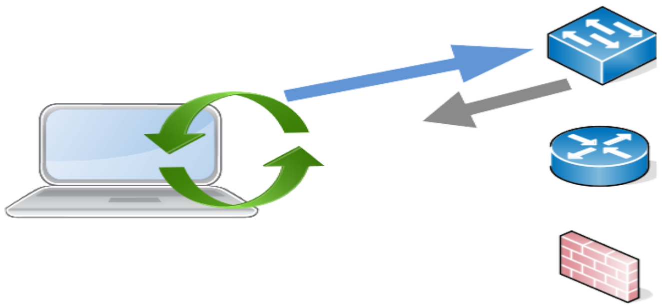
Automating Linux Servers

- Uses SSH to connect to the server
- Server does not have Ansible installed
- Copies Python code to the server (server must have Python execution engine)
- Server executes code and returns status of tasks

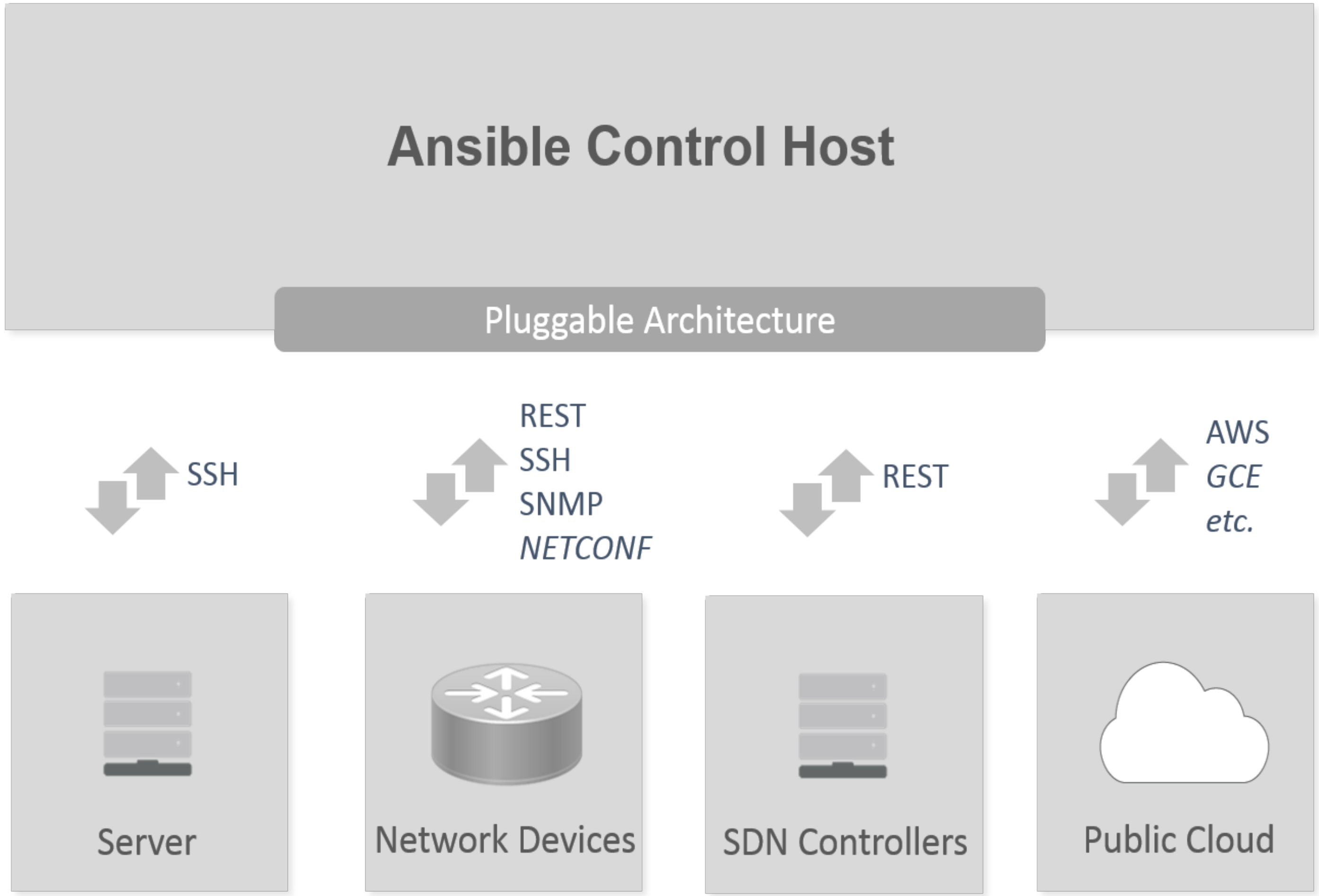


Automating Network Devices

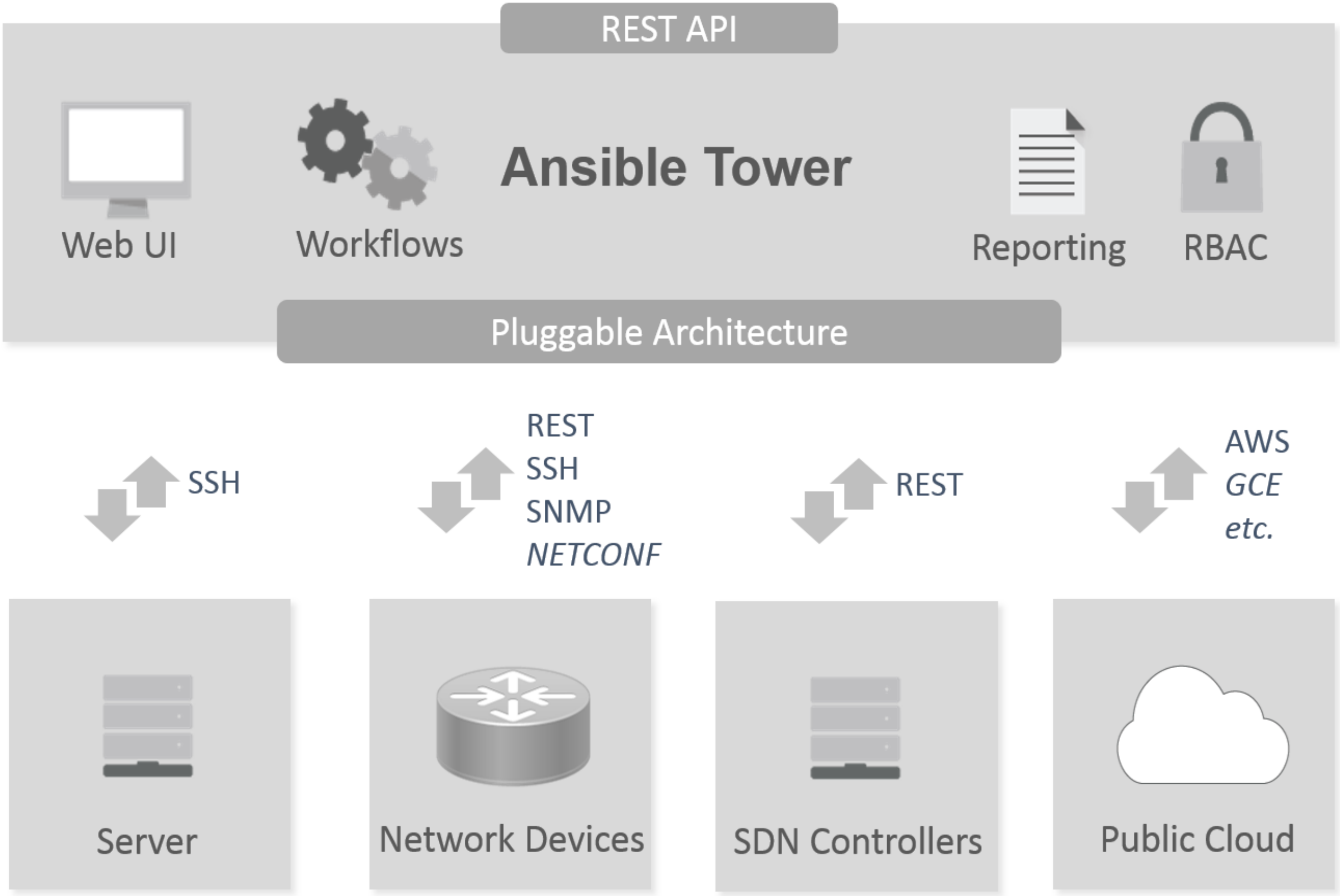
- Python code runs *locally* on the Ansible control host (where Ansible is installed)
- Equivalent of writing Python scripts on a single server
- No code is copied to the device
- Device does not need to support SSH or Python



Ansible Architecture



Ansible Tower



What's Possible with Ansible?

Example Ansible Playbooks

Updating SNMP Community strings

```yaml
---

  - name: DEPLOY SNMP COMMUNITY STRINGS ON IOS DEVICES
    hosts: iosxe
    connection: network_cli
    gather_facts: no

    tasks:

      - name: USE COMMANDS IN THE PLAYBOOK
        ios_config:
          lines:
            - "snmp-server community ntc123 ro"

      - name: DEPLOY FROM CONFIG FILE
        ios_config:
          src: "configs/snmp.cfg"

      - name: DEPLOY USING JINJA2 TEMPLATE
        ios_config:
          src: "snmp.j2"
```

Upgrade Cisco NX-OS Devices

```yaml
---

  - name: UPGRADE NEXUS SWITCHES
    hosts: nxos
    connection: network_cli
    gather_facts: no

    tasks:

      - name: ENSURE SCP SERVER IS ENABLED
        nxos_feature:
          feature: scp-server
          state: enabled

      - name: ENSURE FILE EXISTS ON DEVICE
        nxos_file_copy:
          local_file: "../os-images/cisco/nxos/nxos.7.0.3.I2.2d.bin"

      - name: PERFORM THE UPGRADE
        nxos_install_os:
          system_image_file: nxos.7.0.3.I2.2d.bin
```

Note: a more seamless upgrade requires a few more tasks.

Backup Configuration and Restore

```
---

  - name: BACKUP AND RESTORE
    hosts: "eos,junos"
    connection: local
    gather_facts: no

    tasks:

      - name: BACKUP CONFIGS
        ntc_save_config:
          provider: "{{ connection_details }}"
          local_file: "backups/{{ inventory_hostname }}.cfg"
          platform: "{{ ntc_vendor }}_{{ ansible_network_os }}_{{ ntc_api }}"
        tags: backup

      - name: DEPLOY CONFIGS
        napalm_install_config:
          provider: "{{ connection_details }}"
          config_file: "backups/{{ inventory_hostname }}.cfg"
          diff_file: "diffs/{{ inventory_hostname }}.diffs"
          replace_config: true
          commit_changes: true
          dev_os: "{{ ansible_network_os }}"
        tags: restore
```

Auto-Configure Interface Descriptions

Configure interface descriptions based on active neighbors

```yaml
---

  - name: Auto-configure port descriptions
    hosts: nxos
    connection: local
    gather_facts: no

    tasks:

      - name: GET NEIGHBOR INFORMATION
        ntc_show_command:
          connection: ssh
          provider: "{{ connection_details }}"
          platform: "{{ ntc_vendor }}_{{ ansible_network_os }}"
          command: 'show lldp neighbors'
        register: neighbors

      - name: AUTO-CONFIGURE PORT DESCRIPTIONS BASED ON LLDP DATA
        nxos_interface:
          interface: "{{ item.local_interface  }}"
          description: "Connects to {{ item.neighbor_interface }} on {{ item.neighbor }}"
        loop: "{{ neighbors.response }}"
        when: item.local_interface != 'mgmt0'
```

Takeaways

- All 4 previous playbooks have just 2-3 tasks
- Imagine if you had a dozens (or more) tasks for comprehensive workflows
- Ansible makes it simple to automate:
  - 1 device with N tasks
  - N devices with a 1 task

Ansible for Network Automation

Introduction to Ansible

Ansible for Network Automation

Ansible Terminology

- Inventory
- Playbooks
- Plays
- Tasks
- Modules
- Parameters
- Variables

```yaml
---

- name: BASIC TESTING
  hosts: dc1
  connection: network_cli
  gather_facts: no

  tasks:

    - name: ENSURE VLAN 10 EXISTS # resource/feature module
      nxos_vlan:
        vlan_id: 10
        name: web_vlan

    - name: DEPLOY SNMP CONFIG FROM FILE
      nxos_config:
        src: "configs/snmp.cfg"
```

Automating with Ansible

Two files are required to get started:

- **Inventory file**
- Playbook

Inventory File

Inventory Basics

- ini like file that statically defines which devices are automated
- Uses IP addresses or FQDNs
- The name of the inventory file is arbitrary
- When using "core" modules, use the variable called `ansible_network_os` to define the device OS (helpful to have defined anyway)

```
10.1.1.1          ansible_network_os=eos
switch1.ntc.com   ansible_network_os=eos
r1.ntc.com        ansible_network_os=ios
r2                ansible_network_os=ios
```

Inventory Groups (cont'd)

All devices are in a implicit group called **all**.

```
10.1.1.1         ansible_network_os=eos
switch1.ntc.com  ansible_network_os=eos
r1.ntc.com       ansible_network_os=ios
r2               ansible_network_os=ios
```

Three groups: **all**, **switches**, **routers**.

```
[switches]
10.1.1.1         ansible_network_os=eos
switch1.ntc.com  ansible_network_os=eos

[routers]
r1.ntc.com       ansible_network_os=ios
r2               ansible_network_os=ios
```

Inventory Groups (cont'd)

Three groups: **all**, **switches**, **routers**.

```
[switches]
10.1.1.1          ansible_network_os=eos
switch1.ntc.com   ansible_network_os=eos

[routers]
r1.ntc.com        ansible_network_os=ios
r2                ansible_network_os=ios
```

Four groups: **all**, **switches**, **routers**, and **nyc**.

```
[nyc:children]
switches
routers

[switches]
10.1.1.1          ansible_network_os=eos
switch1.ntc.com   ansible_network_os=eos

[routers]
r1.ntc.com        ansible_network_os=ios
r2                ansible_network_os=ios
```

Inventory Variables

- Group based variables
- Host based variables

Host, or more specific variables, take priority.

Group Variables

Four groups: **all**, **switches**, **routers**, and **nyc**.

Devices can be in more than one group.

```
[nyc:children]
switches
routers

[switches]
10.1.1.1         ansible_network_os=eos
switch1.ntc.com  ansible_network_os=eos

[routers]
r1.ntc.com       ansible_network_os=ios
r2               ansible_network_os=ios
```

Define group variables under `[<group-name>:vars]`

Location of group variables does not matter

```
[all:vars]
ansible_user=ntc
ansible_ssh_pass=ntc123
snmp_ro=networktocode
ansible_network_os=eos

[nyc:children]
switches
routers

[switches]
10.1.1.1
switch1.ntc.com

[routers]
r1.ntc.com
r2

[routers:vars]
snmp_ro=netcode-routers
ansible_network_os=ios
```

Host Variables

Define host variables on the same line as the host.

```
[all:vars]
ansible_user=ntc
ansible_ssh_pass=ntc123
snmp_ro=networktocode
ansible_network_os=eos

[nyc:children]
switches
routers

[switches]
10.1.1.1
switch1.ntc.com   snmp_ro=public123 ansible_ssh_pass=ntc

[routers]
r1.ntc.com
r2                snmp_ro=not-secure

[routers:vars]
snmp_ro=netcode-routers
ansible_network_os=ios
```

Inventory File - Example

```
[all:vars]
location=AMERS
ansible_user=admin
ansible_ssh_pass=admin


[routers]
r1.ntc.com  mgmt_ip=1.1.1.1
r2.ntc.com  mgmt_ip=1.1.1.2

[routers:vars]
ansible_ssh_pass=secret
ansible_network_os=ios

[switches]
s1.ntc.com  mgmt_ip=1.1.2.1  ansible_ssh_pass=supersecret
s2.ntc.com  mgmt_ip=1.1.2.2

[switches:vars]
location=EMEA
ansible_network_os=eos
```

	username	password	mgmt_ip	location	os
r1.ntc.com	admin	secret	1.1.1.1	AMERS	ios
r2.ntc.com	admin	secret	1.1.1.2	AMERS	ios
s1.ntc.com	admin	supersecret	1.1.2.1	EMEA	eos
s2.ntc.com	admin	admin	1.1.2.2	EMEA	eos

Note: `os` is used instead of `ansible_network_os` as a column header only to save space on the slide.

Automating with Ansible

Two files are required to get started:

- Inventory file
- **Playbook**

What is the Playbook?

Ansible uses sports terminology to define the tasks to be automated.

- A playbook contains plays
- Plays contain tasks
- Tasks *do the automation*

Playbook

- Contains instruction set on tasks to be automated
- name of the playbook is arbitrary
- Uses YAML data format
- Playbook contains one or more plays

```yaml
---




















# empty playbook (deploy.yml)
# blank canvas
```

Play(s)

- Begins with a hyphen
  - denotes list of plays (YAML list)
- `name`
  - arbitrary description of the play and is displayed to terminal when executed (optional)
- `hosts`
  - one or more hosts or groups as defined in inventory file or *expression*
- `connection: network_cli`
  - uses persistent SSH connection for network devices
- Play contains one or more tasks

```yaml
---

- name: Play 1 - Deploy router configs
  hosts: routers
  connection: network_cli
  gather_facts: no

  tasks:

    # list of tasks
```

Play(s)

- Begins with a hyphen
  - denotes list of plays (YAML list)
- `name`
  - arbitrary description of the play and is displayed to terminal when executed (optional)
- `hosts`
  - one or more hosts or groups as defined in inventory file or *expression*
- `connection: network_cli`
  - uses persistent SSH connection for network devices
- Play contains one or more tasks

```yaml
---

- name: Play 1 - Deploy router configs
  hosts: routers
  connection: network_cli
  gather_facts: no

  tasks:

    # list of tasks
```

```yaml
- name: Play 2 - Deploy vlans on switches
  hosts: switches
  connection: network_cli
  gather_facts: no

  tasks:
```

Connection Types

- Theres are two different locations we can define our connection type.
  - Inventory file, e.g. `ansible_connection=local`
  - Play definition, e.g. `connection: local`
- `local` is often used on Ansible versions prior to 2.5 and 3rd party modules since 3rd party modules have their own connection mechanisms (APIs, etc.)
- Common "core" connection types for networking:
  - `network_cli` (MOST COMMON)
  - `netconf`
  - `httpapi`

```
[all:vars]
ansible_connection=local
```

```
---
  - name: Connection Types
    hosts: all
    connection: local
    gather_facts: no
```

Connection Types (cont)

- Ansible 2.5 introduces two top-level persistent connection types `network_cli` and `netconf`
- With `network_cli` and `netconf` the playbook passes the connection parameters once.
- We recommend to use `network_cli` and `netconf` whenever possible for your Ansible core modules.
- For more details on available options on each network platform, we can look at the [Ansible docs] - make sure you check your Ansible version beforehand!

```yaml
---
  - name: Connection Types
    hosts: all
    connection: network_cli
    gather_facts: no
```

```yaml
---
  - name: Connection Types
    hosts: junos
    connection: netconf
    gather_facts: no
```

Task(s)

- One or more tasks comprise a play
- Executed on devices defined in inventory file
- Each task:
  - Executes a module using specified parameters (key/value pairs)
  - `name` : optional, arbitrary text displayed when task is executed
- There is more than one supported syntax
  - Native YAML is recommended

```yaml
---

- name: PLAY 1 - Deploy vlans on switches
  hosts: switches
  connection: network_cli
  gather_facts: no

  tasks:

    - name: TASK ONE - YOUR TASK NAME HERE
      MODULE_NAME:
        key1: value1
        key2: value

    - name: TASK TWO - MANAGE SNMP
      ios_config:
        commands:
          - snmp-server contact NET_BOB
        save_when: modified
```

Modules, Parameters, and Variables

- Modules
  - Idempotent
  - Mostly written in Python
  - Parameterized
  - `nxos_vlan` is the module name
- Parameters
  - `vlan_id`, `name`, and `state` are all module parameters

```yaml
---

- name: MANAGE VLANS
  hosts: switches
  connection: network_cli
  gather_facts: no

  tasks:

    - name: ensure VLAN exists
      nxos_vlan:
        vlan_id: 10
        name: web_vlan
        state: present
```

Idempotency

In the context of Ansible...

- Modules that perform a change *should* only make the change once (the first execution)
- You can run the task a 1000 and it'll only occur once
- If you see something different, the module is not idempotent or there is a bug in the module (or the API)

Introducing the CONFIG module

- Module name: `ios_config`, e.g. *_config for main OSs or `cli_config` for multi-vendor environments.
- Basic Parameters: `commands`, and `src`
- Technically `lines` is the parameter and `commands` is an alias since they are just "lines within a config file".
- `src` and `lines/commands` are mutually exclusive for this module
- Each task can use, `name`, an optional task attribute that maps to arbitrary text that is displayed when you run the playbook providing context on where in the playbook execution you are.
- **YOUR FIRST PLAYBOOK CAN BE ONE TASK!!**

```yaml
---

  - name: PLAY 1 - DEPLOYING SNMP CONFIGURATIONS ON IOS
    hosts: routers
    connection: network_cli
    gather_facts: no

    tasks:

      - name: ENSURE SNMP COMMANDS EXIST ON IOS DEVICES TASK 1 i
        ios_config:
          commands:
            - snmp-server community ntc-course RO
            - snmp-server location NYC_HQ
            - snmp-server contact JOHN_SMITH

      - name: ENSURE STATIC ROUTE EXISTS ON IOS DEVICES TASK 2 i
        ios_config:
          lines:
            - ip route 172.16.1.0 255.255.255.0 172.16.2.1

      - name: ENSURE CONFIG EXISTS ON IOS DEVICES TASK 3 in PLAY
        ios_config:
          src: cisco_ios.cfg
```


Executing a Playbook

To execute the Playbook, Explicitly state which inventory file is used, and then the Playbook.

```
$ ansible-playbook -i <inventory-file> <playbook.yml>
```

```
$ ansible-playbook -i inventory deploy-vlans.yml
```

You have other options so you don't have to always use `-i` :

- Default inventory file is `/etc/ansible/hosts`
- Define (export) an environment variable called `ANSIBLE_INVENTORY`
- Over-ride the default in your `ansible.cfg` file (verify with `ansible --version`)

Play Recap

CHANGE - `"changed": true`

```
$ ansible-playbook -i inventory deploy-vlans.yml

PLAY [MANAGE VLANS] ***************************************************************************

TASK [ensure VLAN exists] *********************************************************************
changed: [nxos-spine1] => {"changed": true, "commands": ["vlan 20", "name web_vlan", "exit"]}

PLAY RECAP ************************************************************************************
nxos-spine1                : ok=1    changed=1    unreachable=0    failed=0
```

SUCCESS - `"changed": false`

```
$ ansible-playbook -i inventory deploy-vlans.yml

PLAY [MANAGE VLANS] **************************************************************************

TASK [ensure VLAN exists] ********************************************************************
ok: [nxos-spine1] => {"changed": false, "commands": []}

PLAY RECAP ***********************************************************************************
nxos-spine1                : ok=1    changed=0    unreachable=0    failed=0
```

Play Recap (Cont)

FAIL - `"changed": false` and `failed=1`

```
$ ansible-playbook -i inventory deploy-vlans.yml

PLAY [MANAGE VLANS] *********************************************************************************************************************

TASK [ensure VLAN exists] ***************************************************************************************************************
fatal: [nxos-spine1]: FAILED! => {"changed": false, "msg": "vlan 4098\r\r\n                          ^\r\n% Invalid value/range at '^' marker.\r\n\rnxos-spine1(config)# "}
    to retry, use: --limit @/Users/jump-host/ansible/deploy_vlans.retry

PLAY RECAP ******************************************************************************************************************************
nxos-spine1                : ok=0    changed=0    unreachable=0    failed=1
```

RETRY - Fix the error and try again by running the `.retry` file

to retry, use: --limit @/Users/jump-host/ansible/deploy_vlans.retry

```
$ ansible-playbook -i inventory deploy-vlans.yml --limit @/Users/jump-host/ansible/deploy_vlans.retry

PLAY [MANAGE VLANS] *********************************************************************************************************************

TASK [ensure VLAN exists] ***************************************************************************************************************
changed: [nxos-spine1]

PLAY RECAP ******************************************************************************************************************************
nxos-spine1                : ok=1    changed=1    unreachable=0    failed=0
```

Managing Credentials

- Command Line Flags with Interactive Prompts
- Define as variables in inventory file (or other types of files)
- Interactive Prompts
- Ansible Vault - encrypted (requires passphrase)
- Ansible AWX/Tower - encrypted

Managing Credentials - Command Line Arguments

- Pass in the username from the command with the `-u` or `--user` flag
- Prompt for password with the `-k` or `--ask-pass` flag

Example:

```
$ ansible-playbook -i inventory snmp-config.yml -u ntc -k
SSH password:
```

- Executing Privilege Commands
  - Use `-b` or `--become` for privilege escalation
  - Use `-K` , `--ask-become-pass` to get prompted for "enable" password

Managing Credentials - Defined as Variables

- Define variables `ansible_user` and `ansible_ssh_pass` (use correct group/host variables)
- These are built-in variables that map to the `-u/--user` and `-k/--ask-pass` command line flags

Example:

```
[all:vars]
ansible_user=ntc
ansible_ssh_pass=admin

[routers:vars]
ansible_user=admin

[routers]
r1.ntc.com
r2.ntc.com
```

Before the First Lab

Playbook Task Syntax

Recommended YAML Syntax (key:value)

```yaml
---

- name: MANAGE VLANS
  hosts: switches
  connection: network_cli
  gather_facts: no

  tasks:

    - name: ensure VLAN exists
      nxos_vlan:
        vlan_id: 10

    # no curly braces used with var parameter
    - debug:
        var: inventory_hostname
```

Vertical and/or Horizontal (key=value)

This is more common in older playbooks.

```yaml
---

- name: MANAGE VLANS
  hosts: switches
  connection: network_cli
  gather_facts: no

  tasks:

    - name: ensure VLAN exists
      nxos_vlan:
        vlan_id=10
        host={{ inventory_hostname }}
        username={{ username }}
        password={{ password }}

    - name: ensure VLAN exists
      nxos_vlan: vlan_id=10 host={{ inventory_hostname }} userna

    # no curly braces used with var parameter
    - debug: var=inventory_hostname
```

Lab Time

- Lab 1 - Deploying "Basic" Configurations with Ansible
  - Write Your First Ansible Playbook that will configure SNMP setting on 6 devices!
    - 3 IOS and 3 JUNOS devices
- Lab 2 - Deploying Configs From a File Using *_config
  - Shows how to push configuration using files.
- Lab 3 - Deploying Configs From a File Using cli_config
  - Shows how to push configuration using files.

Understanding Variables

Ansible for Network Automation

Variables

- Group based variables
- Host based variables
- Special variables
- Extra variables

There are more types that we'll cover, but we're still just getting started.

Group Based Variables

- They can be defined in the inventory file or within a directory called `group_vars`
- Variables that are specific to a group.
- Accessible within playbooks and templates

```
# inventory
[eos]
eos-spine1
eos-spine2

[ios]
csr1
csr2
```

```
.
├── inventory
├── pb.yml
├── group_vars
│   ├── all.yml
│   ├── eos.yml
│   └── ios.yml
```

Group Based Variables (cont'd)

- You can alternatively create a directory equal to the group name and have individual files in that directory

```
# inventory
[eos]
eos-spine1
eos-spine2

[ios]
csr1
csr2
```

```
.
├── inventory
├── pb.yml
├── group_vars
│   ├── eos
│   │   ├── aaa.yml
│   │   └── interfaces.yml
│   └── ios
│       ├── aaa.yml
│       └── interfaces.yml
```

Host Based Variables

- They can be defined in the inventory file or within a directory called `host_vars`
- Variables that are specific to a host.
- Accessible within playbooks and templates

```
├── inventory
├── pb.yml
├── group_vars
│   ├── all.yml
│   ├── eos.yml
│   └── ios.yml
├── host_vars
│   ├── csr1.yml
│   ├── csr2.yml
│   ├── eos-spine1.yml
│   └── eos-spine2.yml
```

Host Based Variables (cont'd)

- You can alternatively create a directory equal to the host name and have individual files in that directory

```
.
├── inventory
├── pb.yml
├── host_vars
│   ├── csr1
│   │   ├── aaa.yml
│   │   └── interfaces.yml
│   └── eos-spine1
│       ├── aaa.yml
│       └── interfaces.yml
```

Variable Priority

- You can define host and group variables in the inventory file and respective host vars and group vars files
- The host variable file takes priority over the group variable file

You can **Prove** variable priority: using the **debug** module

Special (Built-in) Variables

Ansible has several built-in, *special*, variables

Variable	Description
inventory_hostname	Name of the current host as defined in the inventory file.
ansible_host	Helpful if inventory hostname is not in DNS or /etc/hosts. Set to IP address of host and use instead of inventory_hostname to access IP/FQDN
hostvars	Dictionary- it's keys are Ansible host names (inventory_hostname) and values is dictionary of every variable that host has (flattend)
play_hosts	A list of inventory hostnames that are in scope for the current play
group_names	List of all groups that the current host is a member of
groups	A dictionary- keys are all group names defined in the inventory file and values are list of host names that are members of the group.
ansible_version	Dictionary representing Ansible major, minor, revision of the release.

Extra Variables

- Known as "extra vars"
- Variables passed into a playbook upon execution.
- Highest priority

```
---

  - name: DEMO PLAYBOOK
    hosts: "{{ devices }}"

    tasks:
      ...
      ...
      ...
```

Pass variables using `-e` or `--extra-vars`

```
$ ansible-playbook -i inventory playbook.yml -e "devices=all"
$ ansible-playbook -i inventory playbook.yml -e "devices=eos"
$ ansible-playbook -i inventory playbook.yml --extra-vars "devices=eos"
```

User Input

You can request user input and capture the user response as a variable using the `var_prompt` module. The `name` under `vars_prompt` is the variable name where the user input will be captured.

```
---

- name: COLLECT USERNAME AND PASSWORD
  hosts: csr1
  connection: local
  gather_facts: no


  vars_prompt:
    - name: un
      prompt: "Please enter the username"
      private: no
```

By default Ansible does not echo user input back to the terminal. To allow user input to be echoed back to to the terminal set the `private` parameter to `no` .

debug module and Playbook Variables

Print and Verify Variable Assignment

Playbook Variables

Ansible uses Jinja2 syntax for variables within a playbook, and uses curly brackets to indicate a variable, like `{{ vlan }}`

Variables within a playbook can be defined under the optional `vars` paramater

```
---
- name: PRINT VLANS
  hosts: all
  connection: local
  gather_facts: no


  vars:
    vlan: 300

  tasks:
    - name: PRINT HOSTNAME
      debug:
        msg: "The VLAN is {{ vlan }}"
```

Since Ansible uses `“{{ var }}”` for variables, if a value after a colon starts with a “{”, YAML will think it is a Python dictionary, so you must put quotation marks around it, if it is not already enclosed in quotes

Playbook Variable Results

As an example, the playbook below uses both a custom Playbook variable, `priority`, and the Ansible built-in `inventory_hostname` variable

```
---
- name: PRINT HOSTS
  hosts: all
  connection: local
  gather_facts: no


  vars:
    priority: "P1"

  tasks:
    - name: PRINT HOSTNAME
      debug:
        msg: "{{ inventory_hostname }} has a priority of {{ pr
```

Note the inventory_hostname iterates through all the hosts, yet the `priority` variable stays the same

```
$ ansible-playbook -i inventory var_test.yml

PLAY [PRINT HOSTS] ******************************

TASK [PRINT HOSTNAME] ***************************

ok: [csr2] => {
    "msg": "csr2 has a priority of P1"
}
ok: [csr1] => {
    "msg": "csr1 has a priority of P1"
}
ok: [csr3] => {
    "msg": "csr3 has a priority of P1"
}
ok: [nxos-spine1] => {
    "msg": "nxos-spine1 has a priority of P1"
}
ok: [nxos-spine2] => {
    "msg": "nxos-spine2 has a priority of P1"
}
# other hosts truncated for brevity
```

debug module

```
# inventory

[all:vars]
ansible_user=ntc
ansible_ssh_pass=ntc123

[datacenter]
leaf1 mgmt_ip=10.1.1.1 ansible_ssh_pass=admin123
leaf2 mgmt_ip=10.1.1.2
```

```
---

  - name: DEBUGGING VARIABLES
    hosts: all
    connection: local
    gather_facts: no

    tasks:

      - name: PRINT INVENTORY HOSTNAME
        debug:
          var: inventory_hostname
```

```
.
├── inventory
├── test-pb.yml
```

debug module

```
# inventory

[all:vars]
ansible_user=ntc
ansible_ssh_pass=ntc123

[datacenter]
leaf1 mgmt_ip=10.1.1.1 ansible_ssh_pass=admin123
leaf2 mgmt_ip=10.1.1.2
```

```
---

  - name: DEBUGGING VARIABLES
    hosts: all
    connection: local
    gather_facts: no

    tasks:

      - name: PRINT INVENTORY HOSTNAME
        debug:
          var: inventory_hostname
```

```
.
├── inventory
├── test-pb.yml
```

```
$ ansible-playbook -i inventory debug.yml

PLAY [DEBUGGING VARIABLES] ********************************************

TASK [PRINT INVENTORY HOSTNAME] ***************************************
ok: [leaf1] => {
    "inventory_hostname": "leaf1"
}
ok: [leaf2] => {
    "inventory_hostname": "leaf2"
}

PLAY RECAP ************************************************************
leaf1                      : ok=1    changed=0    unreachable=0    failed=
leaf2                      : ok=1    changed=0    unreachable=0    failed=
```

debug module

```
# inventory

[all:vars]
ansible_user=ntc
ansible_ssh_pass=ntc123

[datacenter]
leaf1 mgmt_ip=10.1.1.1 ansible_ssh_pass=admin123
leaf2 mgmt_ip=10.1.1.2
```

```
---

  - name: DEBUGGING VARIABLES
    hosts: all
    connection: local
    gather_facts: no

    tasks:

      - name: PRINT PASSWORD
        debug:
          var: ansible_ssh_pass
```

```
.
├── inventory
├── test-pb.yml
```


debug module

```
# inventory

[all:vars]
ansible_user=ntc
ansible_ssh_pass=ntc123

[datacenter]
leaf1 mgmt_ip=10.1.1.1 ansible_ssh_pass=admin123
leaf2 mgmt_ip=10.1.1.2
```

```
---

  - name: DEBUGGING VARIABLES
    hosts: all
    connection: local
    gather_facts: no

    tasks:

      - name: PRINT PASSWORD
        debug:
          var: ansible_ssh_pass
```

```
.
├──── inventory
├──── test-pb.yml
```

```
$ ansible-playbook -i inventory debug.yml

PLAY [DEBUGGING VARIABLES] **********************************************

TASK [PRINT INVENTORY HOSTNAME] *****************************************
ok: [leaf1] => {
    "ansible_ssh_pass": "admin123"
}
ok: [leaf2] => {
    "ansible_ssh_pass": "ntc123"
}

PLAY RECAP **************************************************************
leaf1                      : ok=1    changed=0    unreachable=0    failed=
leaf2                      : ok=1    changed=0    unreachable=0    failed=
```


>>> network.toCode()

debug module

```
# inventory

[all:vars]
ansible_user=ntc
ansible_ssh_pass=ntc123

[datacenter]
leaf1 mgmt_ip=10.1.1.1 ansible_ssh_pass=admin123
leaf2 mgmt_ip=10.1.1.2
```

```
---

  - name: DEBUGGING VARIABLES
    hosts: all
    connection: local
    gather_facts: no

    tasks:

      - name: PRINT MGMT IP
        debug:
          msg: "THE MGMT IP IS {{ mgmt_ip }}"
```

```
.
├── inventory
├── test-pb.yml
```

debug module

```
# inventory

[all:vars]
ansible_user=ntc
ansible_ssh_pass=ntc123

[datacenter]
leaf1 mgmt_ip=10.1.1.1 ansible_ssh_pass=admin123
leaf2 mgmt_ip=10.1.1.2
```

```
---

  - name: DEBUGGING VARIABLES
    hosts: all
    connection: local
    gather_facts: no

    tasks:

      - name: PRINT MGMT IP
        debug:
          msg: "THE MGMT IP IS {{ mgmt_ip }}"
```

```
.
├──── inventory
├──── test-pb.yml
```

```
$ ansible-playbook -i inventory debug.yml

PLAY [DEBUGGING VARIABLES] **********************************************

TASK [PRINT MGMT IP] ****************************************************
ok: [leaf1] => {
    "msg": "THE MGMT IP IS 10.1.1.1"
}
ok: [leaf2] => {
    "msg": "THE MGMT IP IS 10.1.1.2"
}

PLAY RECAP **************************************************************
leaf1                      : ok=1    changed=0    unreachable=0    failed=
leaf2                      : ok=1    changed=0    unreachable=0    failed=
```

Modules return JSON Data

- Every module returns JSON Data
- You can view this data by running a playbook in verbose mode (`-v`)
- For example, data returned includes commands being sent to the network device

```
PLAY [MANAGE VLANS] *************************************************************

TASK [ensure VLAN exists] *******************************************************
changed: [nxos-spine1] => {"changed": true, "end_state": {"admin_state": "up", "mapped_vni": "", "name": "VLAN0010",
"vlan_id": "10", "vlan_state": "active"}, "end_state_vlans_list": ["1", "10"], "existing": {},
"existing_vlans_list": ["1"], "proposed": {}, "proposed_vlans_list": ["10"], "updates": ["vlan 10", "exit"]}

TASK [ensure VLAN exists] *******************************************************
ok: [nxos-spine1] => {"changed": false, "end_state": {"admin_state": "up", "mapped_vni": "", "name": "VLAN0010",
"vlan_id": "10", "vlan_state": "active"}, "end_state_vlans_list": ["1", "10"], "existing": {"admin_state": "up",
"mapped_vni": "", "name": "VLAN0010", "vlan_id": "10", "vlan_state": "active"}, "existing_vlans_list": ["1", "10"],
"proposed": {}, "proposed_vlans_list": ["10"], "updates": []}

TASK [debug] ********************************************************************
ok: [nxos-spine1] => {
    "inventory_hostname": "nxos-spine1"
}

PLAY RECAP **********************************************************************
nxos-spine1                : ok=3    changed=0    unreachable=0    failed=0
```

Understanding the "check" mode

- Does not make configuration changes - dry run
- Verbose mode in combination with the `--check` flag shows the actual commands

```yaml
---

  - name: PLAY 1 - DEPLOYING SNMP CONFIGURATIONS ON IOS
    hosts: iosxe
    connection: network_cli
    gather_facts: no

    tasks:

      - name: TASK 1 in PLAY 1 - ENSURE SNMP COMMANDS EXIST ON IOS DEVI
        ios_config:
          commands:
            - snmp-server community ntc-course RO
            - snmp-server community supersecret RW
            - snmp-server location NYC_HQ_COLO
            - snmp-server contact JOHN_SMITH
```

```
ntc@jump-host:ansible$ ansible-playbook -i lab-inventory snmp-config-02.yml --check -v
Using /etc/ansible/ansible.cfg as config file

PLAY [PLAY 1 - DEPLOYING SNMP CONFIGURATIONS ON IOS] ********************************

TASK [TASK 1 in PLAY 1 - ENSURE SNMP COMMANDS EXIST ON IOS DEVICES] *****************
changed: [csr3] => {"banners": {}, "changed": true, "commands": ["snmp-server location
 NYC_HQ_COLO"], "failed": false, "updates": ["snmp-server location NYC_HQ_COLO"]}
changed: [csr2] => {"banners": {}, "changed": true, "commands": ["snmp-server location
 NYC_HQ_COLO"], "failed": false, "updates": ["snmp-server location NYC_HQ_COLO"]}
changed: [csr1] => {"banners": {}, "changed": true, "commands": ["snmp-server location
 NYC_HQ_COLO"], "failed": false, "updates": ["snmp-server location NYC_HQ_COLO"]}

PLAY RECAP ***************************************************************************
csr1                       : ok=1    changed=1    unreachable=0    failed=0
csr2                       : ok=1    changed=1    unreachable=0    failed=0
csr3                       : ok=1    changed=1    unreachable=0    failed=0

ntc@jump-host:ansible$
```

Module Documentation

- Demo
- Understand the parameters each module supports
  - Choices, defaults, and description
- docs.ansible.com
- `ansible-doc debug`
- `ansible-doc ios_config`
- `ansible-doc $any_module`

Lab Time

- Lab 4 - Using Check Mode and Verbosity
- Lab 5 - Building the course inventory file
- Lab 6 - Using the debug module
- Lab 7 - Prompting the User for Input

Issuing Show (Exec) Commands on Network Devices

Ansible for Network Automation

Core Modules

We cover three types of core modules:

- *_command - Run arbitrary commands on devices
- *_config - Manage configuration sections on devices
- *_facts - Gather facts on devices

The modules are vendor specific and usually support SSH plus a vendor API:

- iosxr_*
- ios_*
- eos_*
- junos_*
- nxos_*
- actively growing

*_command

The _command modules are used to send enable and exec mode commands to the device, e.g. execute show commands and gather data from devices.

- `commands` parameter can accept a single command or a list of commands
- Refer to `ansible-doc` for full list of parameters

```yaml
---

- name: EXECUTE SHOW COMMANDS IOS DEVICES
  hosts: ios
  connection: network_cli
  gather_facts: no

  tasks:

    - name: EXECUTE A SINGLE COMAND
      ios_command:
        commands: "show version"

    - name: EXECUTE LIST OF COMMANDS
      ios_command:
        commands:
          - "show version"
          - "show ip int brief"
```

Demo

- Review `ansible-doc` for `ios_command` and see available parameters
- Take note of the examples at the bottom of the output

*_command (cont'd)

- There are a number of options available to change the format of data returned (for select OS types).

```yaml
- name: SINGLE COMMAND
  nxos_command:
    commands: show version

- name: LIST OF COMMAND STRINGS
  nxos_command:
    commands:
      - show version
      - show hostname

- name: LIST OF DICTIONARIES
  nxos_command:
    commands:
      - command: show version
        output: json
      - command: show version
        output: text
```

```yaml
- name: SHOW COMMANDS TO JUNOS
  hosts: junos
  connection: netconf
  gather_facts: no

  tasks:

    - name: EXECUTE JUNOS COMMANDS
      junos_command:
        commands:
          - show version
          - show interfaces

    - name: EXECUTE JUNOS COMMANDS - TEXT
      junos_command:
        format: text
        commands:
          - show version
          - show interfaces
```

Playbook Execution

Sample playbook execution:

```
$ ansible-playbook -i inventory gather.yml

PLAY [RUN COMMANDS] **********************************************************************

TASK [SINGLE COMMAND] ********************************************************************
ok: [csr1]
ok: [csr2]
ok: [csr3]

PLAY RECAP *******************************************************************************
csr1            : ok=1    changed=0    unreachable=0    failed=0
csr2            : ok=1    changed=0    unreachable=0    failed=0
csr3            : ok=1    changed=0    unreachable=0    failed=0
```

How do you see the data being gathered?

Viewing Response Data

There are two ways to view data returned from a module.

Remember, every module returns JSON data.

There are two ways to see it:

1. Execute playbooks in verbose mode,e.g. `-v`
2. Use the register task attribute with the debug module

```
- name: EXECUTE COMMANDS
  nxos_command:
    commands:
      - show version
  register: output

- debug:
    var: output
```

Using **register** saves the JSON return data as a dictionary that can be consumed as a variable within a playbook or template.

Demo

- Demo playbook using `ios_command`
- Show sending both 1 and 2 commands and see the difference in response data
- Check length of `stdout`

*_command

- Run arbitrary commands on devices.
- Show command data stored in `stdout` (always a list)
- The `stdout` list has a length equal to the number of commands sent to the device

```
- name: SEND SHOW VERSION TO DEVICE
  ios_command:
    commands:
      - 'show version'
  register: output

- name: TEST REGISTERED OUTPUT --> SEE TO RIGHT
  debug:
    var: output


- name: SEE ALL KEYS OF REGISTERED DICTIONARY
  debug:
    var: output.keys()

- name: TEST GETTING SHOW DATA
  debug:
    var: output['stdout'][0]
```

```
TASK [TEST REGISTERED OUTPUT] ********************************************
ok: [csr1] => {
    "output": {
        "changed": false,
        "stdout": ["Cisco IOS XE Software, Version 16.08.01a\nCisco
            IOS Software [Everest], Virtual XE Software (X86_64_LINUX_IOSD
            Version 16.6.2, RELEASE SOFTWARE (fc2)\nTechnical Support:
            http://www.cisco.com/techsupport\nCopyright (c) 1986-2017 by C
            by cisco Systems, Inc.\nAll rights reserved.  Certain
            components of Cisco IOS-XE software are\nlicensed under the GN
            csr1 uptime is 30 - output omitted"],
        "stdout_lines": [], # list closed for slide formatting
        "warnings": []
    }
}
```

Saving show command data to a file

- Using templates to save data to a file (use any logic as necessary)
- Use `copy` module to just *dump* show response to a file

```
---

  - name: BACKUP SHOW VERSION
    hosts: iosxe
    connection: network_cli
    gather_facts: no

    tasks:
      - name: GET SHOW COMMANDS
        ios_command:
          commands:
            - show version
        register: output

      - name: OPTION 1 - SAVE SH COMMAND TO FILE
        template:
          src: basic-copy.j2
          dest: ./commands/{{ inventory_hostname}}-ver.txt


      - name: OPTION 2 - SAVE SH COMMAND TO FILE
        copy:
          content: "{{ output['stdout'][0] }}"
          dest: ./commands/{{ inventory_hostname}}-ver.txt
```

```
# basic-copy.j2

{{ output['stdout'][0] }}
```

Multi Vendor cli_command Module

- You can create a single task to send a single command for two different vendor devices

```
---

  - name: SHOW VERSION FOR IOS
    hosts: csr1,vmx1
    connection: network_cli
    gather_facts: no

    tasks:

      - name: GET SHOW COMMANDS
        cli_command:
          command: show version
         register: output
```

Note: Not all CLI commands are compatible for all vendors, so make sure the command you are sending works for the devices being targeted.

- You can also send a list of commands using a single module for both vendors

```
---

  - name: SHOW VERSION FOR IOS
    hosts: csr1,vmx1
    connection: network_cli
    gather_facts: no

    tasks:

      - name: GET SHOW COMMANDS
        cli_command:
          command: "{{ item }}"
        register: output
        loop:
          - show version
          - show interface
```

cli_command Single Command

- Run arbitrary commands on devices.
- Show command data stored in `stdout` (always a dictionary)

```yaml
- name: GET SHOW COMMANDS
  cli_command:
    command: show version
  register: output

- name: TEST REGISTERED OUTPUT --> SEE TO RIGHT
  debug:
    var: output

- name: SEE ALL KEYS OF REGISTERED DICTIONARY
  debug:
    var: output.keys()

- name: TEST GETTING SHOW DATA
  debug:
    var: output['stdout']
```

```
TASK [TEST REGISTERED OUTPUT ] *******************************************************
ok: [vmx1] => {
    "output": {
        "changed": false,
        "failed": false,
        "stdout": "Hostname: vmx1\nModel: vmx\nJunos: 18.2R1.9\nJUNOS OS Kernel 64-bit
       ....output omitted
        "stdout_lines": [
            "Hostname: vmx1",
            "Model: vmx",
            "Junos: 18.2R1.9",
            ....output omitted
            ]
    }
}
ok: [csr1] => {
    "output": {
        "changed": false,
        "failed": false,
        "stdout": "Cisco IOS XE Software, Version 16.08.01a\nCisco IOS Software [Fuji]
        ....output omitted
        "stdout_lines": [
            "Cisco IOS XE Software, Version 16.08.01a",
            ....output omitted
               ]
    }
}
```

cli_command List of Commands

- When it's a list of commands we get a list of `results`
- Each element inside `results` will be `stdout` per command

```
- name: GET SHOW COMMANDS
  cli_command:
     command: "{{ item }}"
    register: output
  loop:
      - show version
      - show interfac

- name: TEST REGISTERED OUTPUT --> SEE TO RIGHT
  debug:
    var: output

- name: SEE ALL KEYS OF REGISTERED DICTIONARY
  debug:
    var: output.keys()

- name: TEST GETTING SHOW VERSION
  debug:
    var: output['results'][0]['stdout']

- name: TEST GETTING SHOW INTERFACE
  debug:
    var: output['results'][1]['stdout']
```

```
TASK [SEE ALL KEYS OF REGISTERED DICTIONARY] *************************
ok: [csr1] => {
    "output.keys()": "dict_keys(['results', 'msg', 'changed'])"
}
ok: [vmx1] => {
    "output.keys()": "dict_keys(['results', 'msg', 'changed'])"
}

TASK [TEST GETTING SHOW VERSION] *******************************************************
ok: [csr1] => {
    "output['results'][0]['stdout']": "Cisco IOS XE Software, Version 16.08.01a\nCisco IOS
    Version 16.8.1a, RELEASE SOFTWARE (fc1)\nTechnical Support: http://www.cisco.com/techs
    Tue 03-Apr-18 18:43 by mcpre\n\n\nCisco IOS-XE software, Copyright (c) 2005-2018 by ci
    ...output omitted
}
ok: [vmx1] => {
    "output['results'][0]['stdout']": "Hostname: vmx1\nModel: vmx\nJunos: 18.2R1.9\nJUNOS
    \nJUNOS OS runtime [20180614.6c3f819_builder_stable_11]\nJUNOS OS time zone informatio
    ...output omitted

TASK [TEST GETTING SHOW INTERFACE] *****************************************************
ok: [csr1] => {
    "output['results'][1]['stdout']": "GigabitEthernet1 is up, line protocol is up \n  Har
    Description: MANAGEMENT_INTEFACE__DO_NOT_CHANGE\n  Internet address is 100.96.0.18/12\
    reliability 255/255, txload 1/255, rxload 1/255\n  Encapsulation ARPA
    ...output omitted
ok: [vmx1] => {
    "output['results'][1]['stdout']": "show interfaces \n
      Link-level type: Ethernet, MTU: 1514, MRU: 1522, LA
    ...output omitted
```

Performing Compliance Checks

- Using the `assert` module
- Ensure certain conditions exist within the network
- Leverage data that you previously *registered*
- Validate routes exist, changes happen, and configuration is as desired

```
- name: IOS show version
  ios_command:
    commands:
      - show version
  register: output

- name: Ensure OS version is correct
  assert:
    that:
      - "'Version 16.6.2' in output['stdout'][0]"
```

Performing Compliance Checks (cont'd)

- Using the `assert` module
- Ensure certain conditions exist within the network
- Leverage data that you previously *registered*
- Validate routes exist, changes happen, and configuration is as desired

```
- name: IOS show version
  ios_command:
    commands:
      - show version
  register: output

- name: Ensure OS version is correct
  assert:
    that:
      - "'Version 16.6.2' in output['stdout'][0]"
      - "'0x2102' in output['stdout'][0]"
```

Demo

- What happens when an assertion fails?
- When a task fails, the default is that no other task runs for that device in a playbook.
- An assertion failure counts as a task failure
- Introduce `ignore_errors`
- Introduce `fail_msg` and `success_msg`

Auto Create Files

- Manually creating directories can be a tedious task
- You can use the Ansible `file` module to auto create directories, empty files and symlinks.

Check the docs to see more info and available parameters

```
ntc@jump-host:ansible$ ansible-doc file
> FILE (/home/ntc/.local/lib/python3.6/site-packages/ansible/r

Sets attributes of files, symlinks, and directories,
or removes files/symlinks/directories.
Many other modules support the same options as the
`file' module - including [copy], [template], and [assemble].
For Windows targets, use the [win_file] module instead.
```

Manual Way

```
ntc@jump-host:ansible$ mkdir ios junos nxos arista
ntc@jump-host:ansible$
```

Automated Way

```
    - name: CREATE DIRECTORIES BASED ON OS
      file:
        path: ./{{ ansible_network_os }}/
        state: directory
```

Results

```
ntc@jump-host:ansible$ tree
.
├── arista
├── ios
├── junos
└── nxos

4 directories, 0 files
ntc@jump-host:ansible$
```

Lab Time

- Lab 8 - Auto-Create Directories using the file module
- Lab 9 - Getting Started with the Command Module
- Lab 10 - Continuous Compliance with Ansible

Configuration Templating with Jinja2 and YAML

Jinja2 Templates

Ansible for Network Automation

Templating

- How do I automate multiple devices?
  - What if the IP addresses, VLANS or SNMP communities are different on each device?
- Manual method:
  - Enterprises have golden configuration standard for type of device, location of device etc
  - This is updated and deployed, for a new device coming into the network
- Automated templates:
  - Jinja2 provides a way to write these golden configs as skeleton templates with variables
  - Ansible provides the variables to the template and renders configuration using the `template` module.

What is Jinja2?

- Templating language for Python
- Each programming language has one or more templating language
- Used heavily within HTML programming too
- Double curly braces denote a variable `{{ variable }}`
  - Strings, Lists, Dictionaries (Just like we've seen in Ansible / Python)
  - Jinja2 is built for Python
- Supports conditionals, loops, and more functionality (lightweight programming)
  - Syntax changes to `{% %}` for conditionals and logic
- GOAL: Keep templates simple
- More info got to http://jinja.pocoo.org/docs/

Jinja2 Templating

Two components are required:

1. Jinja2 template
2. Variables (or data) to insert into the template

To expand or replace a section of code, use double curly brackets `{{ }}`

De-constructing configs/files into templates:

Example.yml

```
snmp-server community ro PUBLIC
```

Example.j2

```
snmp-server community ro {{ ro_string }}
```

Text file

```
Dear John,

Thanks for attending.
```

Text.j2

```
Dear {{ NAME }},

Thanks for attending.
```

Jinja2 Basics

Accessing variables in Jinja2 templates:

Basic variable:

```
{{ hostame }}
```

Output:

```
NYCR01
```

Loop through a list of strings:

```
{% for item in interfaces_list %}
{{ item }}
{% endfor %}
```

Output:

```
Eth1
Eth2
Eth3
```

Data Variables (inputs):

```
---
hostname: NYCR01

interfaces_list:
  - Eth1
  - Eth2
  - Eth3
```


>>> network.toCode()

Jinja2 Basics

Loop through a list of dictionaries:

```
{% for item in interfaces_list %}
interface {{ item.name }}
  {{ item.state }}
{% endfor %}
```

Output:

```
interface Eth1
  down
interface Eth2
  up
interface Eth3
  up
```

Data Variables (inputs):

```
---
  interfaces_list:
    - name: Eth1
      state: down
    - name: Eth2
      state: up
    - name: Eth3
      state: up
```

Jinja2 Basics

Conditional Logic (if):

```
{% for item in interfaces_list %}
interface {{ item.name }}
{% if item.state == "up" %}
  no shutdown
{% elif item.state == "down" %}
  shutdown
{% endif %}
{% endfor %}
```

Output:

```
interface Eth1
  shutdown
interface Eth2
  no shutdown
interface Eth3
  no shutdown
```

Data Variables (inputs):

```
---
interfaces_list:
  - name: Eth1
    state: down
  - name: Eth2
    state: up
  - name: Eth3
    state: up
```

Use {# . . . #} to enclose comments

```
{# This code not needed, keep for reference
    {% for vlan in vlans %}
        ...
    {% endfor %}
#}
```

Jinja2 include Directive

- Use include statements to pull in other Jinja2 templates
- Included Jinja2 templates have access to the variables of the parent template

```
{% if install == 'new' %}
{% include 'new_install.conf' %}
{% else %}
{% include 'merge_install.conf' %}
{% endif %}
```

- `ignore missing` prevents Jinja2 from throwing an error if there is a missing file

```
{% include 'merge_install.conf' ignore missing %}
```

Jinja2 set Directive

- Use the set directive to assign values to variables
- There are different ways to insert values into a script.
- One of the ways is to add it to our `YAML` files like we have been doing or to use the set directive statement to define the variable within the jinja2 template itself.

```
{% set install = 'new' %}

{% if install == 'new' %}
{% include 'new_install.conf' %}

{% else %}
{% include 'merge_install.conf' %}

{% endif %}
```

The Ansible template module

The basic usage of the template module in Ansible:

```
- name: RENDER CONFIGURATIONS
  template:
    src: device_config.j2
    dest: "./configs/{{ inventory_hostname }}.cfg"
```

Jinja2 and the *_config modules

Within the network core config modules (like ios_config, junos_config etc). You can specify a Jinja2 template as a src file rather than a config file.

```
    - name: ENSURE THAT SNMP IS CONFIGURED ON IOS DEVICES
      ios_config:
        src: 02-ios-snmp.j2
```

```
    - name: ENSURE THAT SNMP IS CONFIGURED ON JUNOS DEVICES
      junos_config:
        src: 02-junos-snmp.j2
```

Jinja Filters

- Filters transform data within a parameter or Jinja Expression
- Are used with the operator `|` like `hostname | upper` will transform the hostname variable using the upper built-in filter to be uppercase
- Custom filters are possible, and Ansible has built-in filters in addition to Jinja2 built-in filters
- For a complete list of [Ansible filters]()
  - Network Filters
  - List filters
  - Dictionary filters
  - RegEx
- Easily create your own filter

```
vars:
  hostname: nycr1
  device_ip: 10.1.1.1
  bad_ip: X.10.Y.2

tasks:
  - name: COVNERT HOSTNAME TO UPPERCASE
    debug:
      var: hostname | upper

  - name: CHECK TO SEE IF A IP ADDR IS VALID
    debug:
      var: device_ip | ipaddr

  - name: CHECK TO SEE IF A IP ADDR IS VALID
    debug:
      var: bad_ip | ipaddr
```

Sample Output:

```
"hostname | upper": "NYCR1"
"device_ip | ipaddr": "10.1.1.1"
"bad_ip | ipaddr": false
```

Demo

- Testing Jinja filters
- You do not need to create a Jinja template to test Jinja2 filters
- Just use the `debug` module!
- Learn about a few other helpful filters

Lab Time

- Lab 11 - Getting Started with Jinja2 Templating in Ansible
- Lab 12 - Using Improved Jinja2 Templates

Diving Deeper into Core Command and Config Modules

Loops and Register

Embedding Loops within a task

- Iterate over a list of strings using the `loop` task attribute
- `item` is built-in variable equal to an element of the list as you're iterating
- In this case, `item` is a string

```
- name: ITERATE OVER LIST OF STRINGS
  hosts: iosxe
  connection: network_cli
  gather_facts: no

  vars:
    commands:
      - show ip int brief
      - show version
      - show ip route

  tasks:
    - name: SEND A SERIES OF SHOW COMMANDS
      ios_command:
        commands: "{{ item }}"
      loop: "{{ commands }}"
```

loop (cont'd)

- Iterate over a list of dictionaries
- `item` is built-in variable equal to an element of the list as you're iterating
- In this case, `item` is a dictionary

```
---

  - name: ITERATE OVER LIST OF DICTIONARIES
    hosts: csr1
    connection: local
    gather_facts: no

    vars:
      vlans:
        - id: 10
          name: web_vlan
        - id: 20
          name: app_vlan
        - id: 30
          name: db_vlan

    tasks:
      - name: PRACTICE DEBUGGING WITH LOOPS
        debug:
          msg: "The VLAN name is {{ item['name'] }} and the ID is {{ item['id'] }}"
        loop: "{{ vlans }}"
```


loop dict2items filter

- Iterate over a dictionary
- Root keys are `item.key`
- Values are `item.value`

```
---

    vars:
      locations:
        amer: sjc-branch
        apac: hk-dc

    tasks:
      - name: PRINT ALL LOCATIONS
        debug:
          msg: "Region is {{ item['key'] }} and Site is {{ item['value'] }}"
        loop: "{{ locations|dict2items }}"
```

- Sample output:

```
"msg": "Regions are apac and Sites is hk-dc"
"msg": "Regions are amer and Sites is sjc-branch"
```

Register (with loop)

- We saw earlier that `register` gives you access to the JSON data returned from a given module

```yaml
- name: SEND SHOW VERSION TO DEVICE
  ios_command:
    commands:
      - 'show version'
  register: output

- name: TEST REGISTERED OUTPUT --> SEE TO RIGHT
  debug:
    var: output

- name: TEST GETTING SHOW DATA
  debug:
    var: output['stdout'][0]
```

```
TASK [TEST REGISTERED OUTPUT] *********************************************
ok: [csr1] => {
    "output": {
        "changed": false,
        "stdout": ["Cisco IOS XE Software, Version 16.08.01a\nCisco
            IOS Software [Everest], Virtual XE Software (X86_64_LINUX_IOSD
            Version 16.6.2, RELEASE SOFTWARE (fc2)\nTechnical Support:
            http://www.cisco.com/techsupport\nCopyright (c) 1986-2017 by C
            by cisco Systems, Inc.\nAll rights reserved.  Certain
            components of Cisco IOS-XE software are\nlicensed under the GN
            csr1 uptime is 30 - output omitted"],
        "stdout_lines": [], # list closed for slide formatting
        "warnings": []
    }
}
```

Remember, when *registering* the response from a `_command` module, there are 4 keys and we primarily focused on `stdout`.

Register (cont'd)

- Using `register` with `loop`

```yaml
- name: SEND PING COMMANDS TO DEVICES
  ios_command:
    commands: "ping {{ item }} repeat 2"
  register: ping_responses
  loop:
    - 8.8.8.8
    - 4.4.4.4
    - 198.6.1.4

- name: TEST REGISTERED OUTPUT
  debug:
    var: ping_responses
```

Now, there is a `results` key that is a list of dictionaries. Each dictionary contains the "standard" JSON data along with an `item` key to access the *item* that is being iterated over.

```
TASK [TEST REGISTERED OUTPUT] *********************************************
ok: [csr1] => {
    "ping_responses": {
        "changed": false,
        "msg": "All items completed",
        "results": [
            {
                "_ansible_ignore_errors": null,
                "_ansible_item_result": true,
                "_ansible_no_log": false,
                "_ansible_parsed": true,
                "changed": false,
                "failed": false,
                "invocation": {
                    "module_args": {
                        "auth_pass": null,
                        "authorize": null,
                        "commands": [
                            "ping 8.8.8.8 repeat 2"
                        ],
                        # omitted for brevity

                    }
                },
                "item": "8.8.8.8",
                "stdout": [
                    "Type escape sequence to a
                ],
                "stdout_lines": []
            }
```

Register (cont'd)

- Since `items` is a key in the last slide and `item` is also a built-in variable when using loops, be cautious of `item.item`
- `item` is the *item* in the list being iterated over

```
- name: TEST LOOPING OVER REGISTERED VARIABLE
  debug:
    msg: "{{ item }}"
  loop: "{{ ping_responses['results'] }}"
```

- The inside key `item` is the IP address for that iteration, e.g. 8.8.8.8

```
- name: TEST LOOPING OVER REGISTERED VARIABLE
  debug:
    msg: "{{ item['item'] }}"
  loop: "{{ ping_responses['results'] }}"
```

Lab Time

- Lab 13 - Challenge Validating Reachability with the Command Module

Parsing Unstructured Data

Parsing Response Data

When running commands use the core command module it is often necessary to parse needed information from the command response data.

The following methods can be used to parse the response data:

- `parse_cli_textfsm`
- `regex_search`
- `regex_findall`

You will notice that these methods are the same as methods used to parse data in Python.

TextFSM Overview

- Python module for parsing semi-formatted text.
- Originally developed to allow programmatic access to information given by the output of CLI driven devices, such as network routers and switches
    - It can however be used for any such textual output.

Using TextFSM

- The engine takes two inputs
    - Template file
    - Text input (such as command responses from the CLI of device)
- Returns a list of records that contains the data parsed from the text.
- Note: A template file is needed for each uniquely structured text input.

TextFSM

Network Examples

Example 1: Text Input

- show vlan (Arista EOS)
- Filename: `arista_eos_show_vlan.raw`

```
VLAN  Name                             Status    Ports
----- -------------------------------- --------- -------------------------------
1     default                          active    Et1
10    Test1                            active    Et1, Et2
20    Test2                            suspended
30    VLAN0030                         suspended
```

Example 1: Template File

- show vlan (Arista EOS)
- Order is important
- Filename: `arista_eos_show_vlan.template`

```
Value VLAN_ID (\d+)
Value NAME (\w+)
Value STATUS (active|suspended)

Start
  ^${VLAN_ID}\s+${NAME}\s+${STATUS} -> Record
```

Example 1: Executing textfsm

```
VLAN  Name                             Status    Ports
----- -------------------------------- --------- -------------------------------
1     default                          active    Et1
```

```
ntc@jump-host$ python textfsm.py arista_eos_show_vlan.template arista_eos_show_vlan.raw
FSM Template:
Value VLAN_ID (\d+)
Value NAME (\w+)
Value STATUS (active|suspended)

Start
  ^${VLAN_ID}\s+${NAME}\s+${STATUS} -> Record


FSM Table:
['VLAN_ID', 'NAME', 'STATUS']
['1', 'default', 'active']
['10', 'Test1', 'active']
['20', 'Test2', 'suspended']
['30', 'VLAN0030', 'suspended']
```

Parsing Data Using parse_cli_textfsm

The `parse_cli_textfsm` Jinja2 filter can use the same `textfsm` templates that are used in Python to parse data in Ansible.

```
---

  - name: TEST PARSE USING PARSE_CLI_TEXTFSM
    hosts: csr1
    connection: network_cli
    gather_facts: no

    vars:
      template_path: "/etc/ntc/ansible/library/ntc-ansible/ntc-templates/templates/"
      show_version_path: "{{ template_path }}cisco_ios_show_version.template"

    tasks:

      - name: GET SHOW COMMANDS
        ios_command:
          commands: show version
        register: config_data

      - set_fact:
          show_version: "{{ config_data.stdout.0 | parse_cli_textfsm(show_version_path) }}"

      - debug:
          var: show_version
```

Parsing Data Using parse_cli_textfsm

You will see that `parse_cli_textfsm` will return structured data.
This is the output from the `debug` module on the previous slide:

```
TASK [debug] ***************************************************
ok: [csr1] => {
    "show_version": [
        {
            "CONFIG_REGISTER": "0x2102",
            "HARDWARE": [
                "CSR1000V"
            ],
            "HOSTNAME": "csr1",
            "ROMMON": "IOS-XE",
            "RUNNING_IMAGE": "packages.conf",
            "SERIAL": [
                "9KIBQAQ3OPE"
            ],
            "UPTIME": "6 hours, 18 minutes",
            "VERSION": "16.6.2"
        }
    ]
}
```

NTC Templates

- Network to Code maintains the largest open source repository of TextFSM templates for network "show command" parsing
- They are broken down based on vendor and OS:
- https://github.com/networktocode/ntc-templates/tree/master/templates

Parsing Data Using regex filters

You can also use the `regex_search` and `regex_findall` Jinja2 filters to parse data.

In this example we issued the ping command from an IOS device and want to parse out the success precentage.

```
---

  - name: PING TEST AND TRACEROUTE
    hosts: csr1
    connection: network_cli
    gather_facts: no

    vars:
      dest: "8.8.8.8"

    tasks:

    - name: ISSUE PING
      ios_command:
        commands: "ping {{ dest }} repeat 2"
      register: output
```

We are registering the response from the ping command to the `output` variable.

Parsing Data Using regex_search

Using `regex_search` we can parse the `stdout` from the `output` variable to find the success percentage.

```
    - name: PARSE PING RESPONSE TO OBTAIN % OF SUCCESS
      set_fact:
        ping_pct: "{{ output.stdout.0 | regex_search('Success rate is (\\d+)\\s+percent') }}"
```

`ping_pct` is equal to **Success rate is 100 percent**

```
    - name: PARSE PING RESPONSE TO OBTAIN % OF SUCCESS
      set_fact:
        ping_pct: "{{ output.stdout.0 | regex_search('Success rate is (\\d+)\\s+percent') | regex_search('(\\d+)') }}"
```

`ping_pct` is equal to **"100"**

- The solution chains together two `regex_search` filters due to how the `regex_search` filter works.
- The first part returns everything matched within parentheses and the second captures the percentage within the first capture and saves that value in the `ping_pct` fact (variable).

Parsing Data Using regex_findall

Using `regex_findall` is another way to parse the `stdout` from the `output` variable to find the success percentage.

```
  - name: PARSE PING RESPONSE TO OBTAIN % OF SUCCESS
    set_fact:
      ping_pct: "{{ output.stdout.0 | regex_findall('Success rate is (\\d+)\\s+percent') | first }}"
```

- The filter works as you'd expect (unlike `regex_search`). It only returns what's inside parentheses (capture group), but it's always a list.
- The `first` filter returns the first element in the list. This result is assigned to the variable `ping_pct`.

Lab Time

- Lab 14 - Performing a Conditional Traceroute with RegEx filters

*_config Module

*_config

By default, it compares the lines against the running configuration

You can pass commands into the module a few different ways:

- Using the `lines` parameter
- Using the `src` parameter and point to a pre-built config file
- Using the `src` parameter and point to a Jinja2 template

```
---

- name: DEPLOY SNMP COMMUNITY STRINGS ON IOS DEVICES
  hosts: ios
  connection: network_cli
  gather_facts: no

  tasks:

    - name: USE COMMANDS IN THE PLAYBOOK
      ios_config:
        lines:
          - "snmp-server community ntc123 ro"

    - name: DEPLOY FROM CONFIG FILE
      ios_config:
        src: "configs/snmp.cfg"
```

*_config

```yaml
# Ensure these lines are present in the configuration
- nxos_config:
    commands:
      - snmp-server community public group network-operator
      - snmp-server community networktocode group network-operator
```

```yaml
# Ensure these lines are present in the configuration
- junos_config:
    src: snmp.conf
```

*_config (cont'd)

- Modules support many parameters
- Use `ansible-doc` to view them all
- `parents` - ordered list of commands that identify the section the commands should be checked against

```
- name: ENSURE GIGE4 IS CONFIGURED PROPERLY
  ios_config:
    parents:
      - interface GigabitEthernet4
    lines:
      - description Configured by Ansible
      - ip address 10.100.100.1 255.255.255.0
```

*_config (cont'd)

- Modules support many parameters
- Use `ansible-doc` to view them all
- `parents` - ordered list of commands that identify the section the commands should be checked against
- `before` - ordered list of commands to be prepended to `lines` if a change needs to be made
- `after` - ordered list of commands to be appended to `lines` if a change needs to be made
- Note: these are just a sub-set of parameters supported

```
# this would remove any other commands previously covered for
# the other ASN
    - name: ENSURE BGP CONFIG IS CORRECT
      ios_config:
        before: ['no router bgp 65512']
        parents:
          - router bgp 65512
        lines:
          - bgp router-id 10.10.10.10
          - bgp log-neighbor-changes
          - network 10.101.1.0 mask 255.255.255.0
          - network 10.101.2.0 mask 255.255.255.0
          - timers bgp 5 15
          - neighbor 10.10.10.2 remote-as 102
          - neighbor 10.10.10.2 description ISP_CARRIER_X
          - neighbor 10.10.10.2 send-community
          - neighbor 10.10.10.2 soft-reconfiguration inbound
        after: ['copy run start']
```

*_config (cont'd)

The `save_when` parameter is needed to commit running config to the NVRAM. (Deprecated command `save` no longer works with Ansible 2.4 and above)
Available options for the save_when parameter:

- always
- modified
- never

```
- name: ENSURE THAT LOOPBACK 222 IS CONFIGURED
  ios_config:
    commands:
      - ip address 10.222.222.222 255.255.255.255
    parents:
      - interface loopback 222
    save_when: modified
```

The diff_against Parameter

The diff_against Parameter

Introduced in Ansible 2.4. Test running configuration against:

- The startup configuration
  - Check if there are ephemeral configurations
- A configuration intent
  - Check whether running configuration deviates from compliance/golden configuration
- Pending configuration lines
  - Check exact configuration impact of config lines being pushed

Invoked with `--diff` flag

diff_against - startup

```
    - name: COMPARE RUNNING CONFIG WITH STARTUP
      ios_config:
        diff_against: startup
```

```
TASK [COMPARE RUNNING CONFIG WITH STARTUP] **************************************
--- before
+++ after
@@ -36,6 +63,8 @@
 redundancy
 lldp run
 cdp run
+interface Loopback222
+ ip address 10.222.222.222 255.255.255.255
 interface GigabitEthernet1
  vrf forwarding MANAGEMENT
  ip address 10.0.0.51 255.255.255.0
```

The lookup plugin

Powerful Ansible plugin that is used access data from outside sources

- Regular text file content
- CSV
- INI
- DNS Lookup
- MongoDB and many more

Can be used to assign values to variables

```
vars:
  config_file: "{{ lookup('file', './backups/{{ inventory_hostname }}.cfg'

tasks:
  - debug:
      msg: "The file name is {{ config_file }}"
```

```
ntc@jump-host:ansible$ ansible-playbook -i inventory
file_lookup_demo.yml

PLAY [DEMO FILE LOOKUPS]
****************************************************

TASK [debug]
****************************************************************
ok: [csr1] => {
    "msg": "The file name is snmp-server community PUBLIC123 RO 5\nsnmp-
server community PRIVATE123 RW 95\nsnmp-server location GLOBAL\nsnmp-
server contact LOCAL_ADMIN\nsnmp-server host 1.1.1.1\n\nvlan 10\n name
web_servers\nvlan 20\nvlan 30\n name db_servers"
}

PLAY RECAP
******************************************************************
csr1                           : ok=1    changed=0    unreachable=0
failed=0
```

diff_against - intended

```yaml
  tasks:
    - name: VALIDATE CONFIGURATION INTENT
      ios_config:
        diff_against: intended
        intended_config: "{{ lookup('file', './backups/{{ inventory_hostname }}.cfg') }}"
```

```
TASK [VALIDATE CONFIGURATION INTENT] **************************************
--- before
+++ after
@@ -63,6 +63,8 @@
 redundancy
 lldp run
 cdp run
+interface Loopback222
+ ip address 10.222.222.222 255.255.255.255
 interface GigabitEthernet1
  vrf forwarding MANAGEMENT
  ip address 10.0.0.51 255.255.255.0
```

diff_against - impending configuration lines

```
    - name: ENSURE THAT LOOPBACK222 IS CONFIGURED
      ios_config:
        commands:
          - ip address 10.222.222.222 255.255.255.255
        parents:
          - interface loopback 222
        diff_against: running
```

```
TASK [ENSURE THAT LOOPBACK 222 IS CONFIGURED]
**************************************
+++ after
@@ -63,6 +63,8 @@
 redundancy
 lldp run
 cdp run
+interface Loopback222
+ ip address 10.222.222.222 255.255.255.255
 interface GigabitEthernet1
  vrf forwarding MANAGEMENT
  ip address 10.0.0.51 255.255.255.0
```

Note: This task will actually make changes to the running config!

Declerative Configuration

Declarative Configuration

- Data model

```
snmp_communities:
  - community: ntc-public
    group: network-operator
  - community: ntc-private
    group: network-admin
```

- Template for Nexus

```
{% for snmp in snmp_communities %}
snmp-server community {{ snmp.community }} group {{ snmp.group }}
{% endfor %}
```

Declarative Configuration (cont'd)

- Generate configuration

```yaml
---
- name: Declarative Configuration
  hosts: nxos
  connection: network_cli
  gather_facts: False

  tasks:
    - name: GENERATE CONFIGURATION
      template:
        src: "./templates/snmp.j2"
        dest: "./snmp-config.cfg"
```

- snmp-config.cfg

```
snmp-server community ntc-public group network-operator
snmp-server community ntc-private group network-admin
```

Declarative Configuration (cont'd)

- Push configuration

```
- name: PUSH SNMP COMMUNITIES
  nxos_config:
    src: "./snmp-config.cfg"
```

Declarative Configuration (cont'd)

- Get existing SNMP communities and set fact

```
- name: GET CONFIG FOR SNMP PARSING
  nxos_command:
    commands:
      - show run section snmp
  register: output

- name: GET EXISTING SNMP COMMUNITIES AND SET FACT
  set_fact:
    existing_snmp_communities: "{{ output.stdout[0] | regex_findall('snmp-server community (\\S+)') }}"

- debug:
    var: existing_snmp_communities
```

```
TASK [GET CONFIG FOR SNMP PARSING] ***********************************************
ok: [nxos]

TASK [GET EXISTING SNMP COMMUNITIES AND SET FACT] ******************************
ok: [nxos]

TASK [debug] *********************************************************************
ok: [nxos] => {
    "existing_snmp_communities": [
        "ntc-public",
        "ntc-private",
        "public",
        "networktocode"
    ]
}
```

Declarative Configuration (cont'd)

- **public** and **networktocode** communities are not part of the data model. Therefore, they may be undesired and need to be removed
- Calculate communities to remove

```
- name: SET FACT FOR PROPOSED (DESIRED) COMMUNITIES
  set_fact:
    proposed_snmp_communities: "{{ snmp_communities|map(attribute='community')|list }}"

- name: CALCULATE AND SET FACT FOR COMMUNITIES TO REMOVE
  set_fact:
    snmp_communities_to_remove: "{{ existing_snmp_communities|difference(proposed_snmp_communities) }}"

- debug:
    var: snmp_communities_to_remove
```

```
TASK [SET FACT FOR PROPOSED (DESIRED) COMMUNITIES] *****************************
ok: [nxos]

TASK [CALCULATE AND SET FACT FOR COMMUNITIES TO REMOVE] ************************
ok: [nxos]

TASK [debug] *******************************************************************
ok: [nxos] => {
    "snmp_communities_to_remove": [
        "public",
        "networktocode"
    ]
}
```

Declarative Configuration (cont'd)

- Remove undesired communities

```
  - name: PURGE SNMP COMMUNITIES
    nxos_config:
      commands:
        - "no snmp-server community {{ item }}"
    with_items: "{{ snmp_communities_to_remove }}"
```

```
TASK [PURGE SNMP COMMUNITIES] ***************************************************
changed: [nxos] => (item=public)
changed: [nxos] => (item=networktocode)
```

Lab Time

- Lab 15 - Using the Config Module

Data Collection & Reporting

Core *_facts Modules

Core facts modules collect a number of useful pieces of information:

- All IP addresses
- Filesystems
- Hostname
- Image
- All interfaces with some Layer 2 and Layer 3 attributes
- Serial Number
- OS Version
- Neighbors broken down by interface

Collecting Facts

Sample playbook gathering IOS facts:

```
- name: GATHER IOS FACTS
  hosts: iosxe
  connection: network_cli
  gather_facts: no

  tasks:
    - name: GET FACTS
      ios_facts:
```

Collecting Facts

Sample playbook gathering IOS facts:

```yaml
- name: GATHER IOS FACTS
  hosts: iosxe
  connection: network_cli
  gather_facts: no

  tasks:
    - name: GET FACTS
      ios_facts:
```

- Default value for `gather_subset` is **!config**

```yaml
- name: GATHER ALL FACTS
  ios_facts:
    gather_subset: all

- name: GATHER A SHOW RUN AND DEFAULT SYSTEM FACTS
  ios_facts:
    gather_subset:
      - config

- name: GATHER ALL FACTS EXCEPT HARDWARE FACTS
 ios_facts:
  gather_subset:
    - "!hardware"
```

Collecting Facts

```yaml
- name: GATHER IOS FACTS
  hosts: iosxe
  connection: network_cli
  gather_facts: no

  tasks:
    - name: GET FACTS
      ios_facts:
      register: ntc_ios_facts

    - debug:
        var: ntc_ios_facts
```

Sample Response (IOS)


```
"ntc_ios_facts": {
    "ansible_facts": {
        "ansible_net_all_ipv4_addresses": [
            "10.0.0.53"
        ],
        "ansible_net_all_ipv6_addresses": [],
        "ansible_net_filesystems": [
            "bootflash:"
        ],
        "ansible_net_hostname": "csr3",
        "ansible_net_image": "bootflash:packages.conf",
        "ansible_net_interfaces": {
            "GigabitEthernet1": {
                "bandwidth": 1000000,
                "description": null,
                "duplex": "Full",
                "ipv4": {
                    "address": "10.0.0.53",
                    "masklen": 24
                },
                "lineprotocol": "up ",
                "macaddress": "2cc2.604c.4e06",
                "mediatype": "RJ45",
                "mtu": 1500,
                "operstatus": "up",
                "type": "CSR vNIC"
            }
        }
```



```
        "ansible_net_memfree_mb": 322777,
        "ansible_net_memtotal_mb": 2047264,
        "ansible_net_model": null,
        "ansible_net_neighbors": {
            "Gi1": [
                {
                    "host": "eos-leaf1.ntc.com",
                    "port": "Management1"
                },
                {
                    "host": "eos-leaf2.ntc.com",
                    "port": "Management1"
                }
            ]
        },
        "ansible_net_serialnum": "9KXI0D7TVFI",
        "ansible_net_version": "16.3.1"
    }
```

Viewing Facts For a Device

OPTION 1

```
- name: GET FACTS
  ios_facts:
  register: ntc_ios_facts

- debug:
    var: ntc_ios_facts

- debug:
    var: ntc_ios_facts['ansible_facts']['ansible_net_hostname'
```

OPTION 2

```
- name: GET FACTS
  ios_facts:

- name: Display variables/facts known for a given host
  debug:
    var: hostvars[inventory_hostname]

- debug:
    var: ansible_net_hostname
```

Note: any key inside `ansible_facts` can be accessed directly

```
"ntc_ios_facts": {
    "ansible_facts": {
        "ansible_net_all_ipv4_addresses": [
            "10.0.0.53"
        ],
        "ansible_net_all_ipv6_addresses": [],
        "ansible_net_filesystems": [
            "bootflash:"
        ],
        "ansible_net_hostname": "csr3",
```

Using the URI Module

The `uri` module can be used to make HTTP-based API calls.

```
- name: GET INTERFACE IP ADDRESS
  uri:
    url: https://{{ inventory_hostname }}/restconf/data/Cisco-IOS-XE-native:native/interface=GigabitEthernet/1/ip/address
    method: GET
    user: "{{ ansible_user }}"
    password: "{{ ansible_ssh_pass }}"
    return_content: yes
    validate_certs: no
    headers:
      Content-Type: application/yang-data+json
      Accept: application/yang-data+json
  register: response
```

This example shows how to make an API call against an IOSXE device to pull the IP address information for the `GigabitEthernet1` interface and assigning the returned value to the `response` variable.

You can test all of these settings using Postman before building your Ansible tasks.

Tips: Using set_facts

- In contrast to using `register` to store the output of a task into a variable, the `set_fact` module allows a task to define a variable
- Sometimes used to simplify the naming of a variable

```yaml
vars:
  locations:
    amer:
      nyc:
        - nyc-dc
        - nyc-campus
      sjc:
        - sjc-branch
    apac:
      hk:
        - hk-dc
        - hk-campus

tasks:
  - name: PRINT ALL LOCATIONS
    debug:
      var: locations

  - name: SJC LOCATIONS
    set_fact:
      sjc_locations: "{{ locations['amer']['sjc'] }}"

  - name: PRINT ALL LOCATIONS
    debug:
      var: sjc_locations
```

Tips: Using from_json Filter

- In a recent example, an API call to the IOSXE device received a JSON response in python, `json.loads()` would be used to convert the JSON object to a dictionary
- In Ansible, instead use the `from_json` filter
- For example, assuming the `response` API variable is earlier in the play:

```
  - set_fact:
      ip_info: "{{ response['content'] | from_json }}"

  - debug:
      var: ip_info['Cisco-IOS-XE-native:address']['primary']['address']
```

Creating Documentation

You choose:

- text
- html
- markdown
- asciidoc
- ...

Then:

- publish
- alert
- chatops
- mail

Know the Available Facts/Variables

- Template

```
# general.j2

Device: {{ inventory_hostname }}

Vendor:           {{ ntc_vendor }}
Platform:         {{ platform }}
Operating System: {{ ansible_network_os }}
Image:            {{ ansible_net_image }}
```

- Playbook

```
---

  - name: DC P1
    hosts: nxos-spine1
    connection: local
    gather_facts: no

    tasks:
      - nxos_facts:

      - template:
          src: general.j2
          dest: "files/general.md"
```

Know the Available Facts/Variables

- Template

```
# general.j2

Device: {{ inventory_hostname }}

Vendor:           {{ ntc_vendor }}
Platform:         {{ platform }}
Operating System: {{ ansible_network_os }}
Image:            {{ ansible_net_image }}
```

- Playbook

```
---

  - name: DC P1
    hosts: nxos-spine1
    connection: local
    gather_facts: no

    tasks:
      - nxos_facts:

      - template:
          src: general.j2
          dest: "files/general.md"
```

- Document

```
Device: nxos-spine1
Vendor:           cisco
Platform:         NX-OSv Chassis
Operating System: nxos
Image:            bootflash:///titanium-d1.7.3.1.D1.0.10.bin
```

Documenting Neighbors

- Template

```
# neighbors.j2

DEVICE: {{ inventory_hostname }}

{%  for local_int, details in ansible_net_neighbors.items() %}
LOCAL INTERFACE:    {{ local_int }}
{% for neigh_data in details %}
NEIGHBOR:           {{ neigh_data.sysname }}
NEIGHBOR INTERFACE: {{ neigh_data.port }}

{% endfor %}
{% endfor %}
```

- Playbook

```
---
  - name: DC P1
    hosts: nxos-spine1
    connection: network_cli
    gather_facts: no
    tasks:
      - name: LLDP NEIGHBORS
        nxos_facts:

      - name: BUILD TABLE
        template:
          src: neighbors.j2
          dest: "files/neighbors.md"
```

Documenting Neighbors

- Template

```
# neighbors.j2

DEVICE: {{ inventory_hostname }}

{%  for local_int, details in ansible_net_neighbors.items() %}
LOCAL INTERFACE:     {{ local_int }}
{% for neigh_data in details %}
NEIGHBOR:            {{ neigh_data.sysname }}
NEIGHBOR INTERFACE: {{ neigh_data.port }}

{% endfor %}
{% endfor %}
```

- Playbook

```
---
  - name: DC P1
    hosts: nxos-spine1
    connection: network_cli
    gather_facts: no
    tasks:
      - name: LLDP NEIGHBORS
        nxos_facts:

      - name: BUILD TABLE
        template:
          src: neighbors.j2
          dest: "files/neighbors.md"
```

- Document

```
DEVICE: nxos-spine1

LOCAL INTERFACE:    Ethernet2/3
NEIGHBOR:           nxos-spine2(TB604B14E3B)
NEIGHBOR INTERFACE: Ethernet2/3

LOCAL INTERFACE:    Ethernet2/4
NEIGHBOR:           nxos-spine2(TB604B14E3B)
NEIGHBOR INTERFACE: Ethernet2/4

LOCAL INTERFACE:    mgmt0
NEIGHBOR:           nxos-spine2(TB604B14E3B)
NEIGHBOR INTERFACE: mgmt0

LOCAL INTERFACE:    Ethernet2/2
NEIGHBOR:           nxos-spine2(TB604B14E3B)
NEIGHBOR INTERFACE: Ethernet2/2

LOCAL INTERFACE:    Ethernet2/1
NEIGHBOR:           nxos-spine2(TB604B14E3B)
NEIGHBOR INTERFACE: Ethernet2/1
```

Using a Table (Neighbors)

- Template

```
# neighbors-table.j2
| Source      | Interface     |            Neighbor            | Inte
| ---------- |--------------|---------------------------|-----
{% for local_int, details in ansible_net_neighbors.items() %}
{% for neigh_data in details %}
| {{ inventory_hostname }}| {{ local_int }}  |  {{ neigh_data.
{% endfor %}
{% endfor %}
```

- Playbook

```
---
  - name: DC P1
    hosts: nxos-spine1
    connection: network_cli
    gather_facts: no
    tasks:
      - name: LLDP NEIGHBORS
        nxos_facts:

      - template:
          src: neighbors.j2
          dest: "files/neighbors-table.md"
```

Using a Table (Neighbors)

- Template

```
# neighbors-table.j2
| Source     | Interface     |          Neighbor           | Inte
| ---------- |--------------|---------------------------|-----
{% for local_int, details in ansible_net_neighbors.items() %}
{% for neigh_data in details %}
| {{ inventory_hostname }}| {{ local_int }}  |  {{ neigh_data.
{% endfor %}
{% endfor %}
```

- Playbook

```
---
  - name: DC P1
    hosts: nxos-spine1
    connection: network_cli
    gather_facts: no
    tasks:
      - name: LLDP NEIGHBORS
        nxos_facts:

      - template:
          src: neighbors.j2
          dest: "files/neighbors-table.md"
```

- Markdown generated table

Source	Interface	Neighbor	Interface
nxos-spine1	Ethernet2/4	nxos-spine2(TB604B14E3B)	Ethernet2/4
nxos-spine1	Ethernet2/3	nxos-spine2(TB604B14E3B)	Ethernet2/3
nxos-spine1	Ethernet2/2	nxos-spine2(TB604B14E3B)	Ethernet2/2
nxos-spine1	Ethernet2/1	nxos-spine2(TB604B14E3B)	Ethernet2/1

lineinfile

- Add/Remove a line to a file
- Use regexes to match for position
- Insert before/after

```
- name: Ensure a line does not exist
  lineinfile:
    line: "Building configuration..."
    dest: "backups/{{ inventory_hostname }}.cfg"
    state: "absent"
```

```
- name: Ensure the line that matches the regex does not exist
  lineinfile:
    dest: "backups/{{ inventory_hostname }}.cfg"
    regexp: "Current configuration .*"
    state: "absent"
```

```
!
Building configuration...

Current configuration 3942 Bytes
!
hostname nycr01
```

Summary

- Understand the various ways to collect data about the network.
- Templating is great for documentation (not just configurations)
- Know your variables
    - Any variable can be used in the playbook **and** template
- Know your document formats
- Understand Jinja2
- Auto publish to github, web server, etc.
- Key modules:
    - template
    - assemble

Lab Time

- Lab 16 - Making REST API Calls from Ansible
- Lab 17 - Data Collection Modules & Reporting
  - Facts Data Collection Modules
  - Inventory Report

Ansible Roles

Ansible for Network Automation

Reusable Abstractions

- include statement
- *includes* tasks from another file

```
# main playbook
---

  - name: PB
    hosts: all
    connection: network_cli
    gather_facts: no


    tasks:

      - include: get-facts.yml
```

```
---
# get-facts.yml

- name: GET FACTS FROM ARISTA DEVICES
  eos_facts:

  ...<more getter tasks>...
```

Parameterized Include

- Pass parameters to *included* tasks

```yaml
# main playbook
---
  - name: CONFIGURE DEVICES
    hosts: all
    connection: network_cli
    gather_facts: no

    tasks:

      - include: get-facts.yml vendor={{ vendor }}
```

```yaml
# get-facts.yml
---

- name: GET FACTS FROM ARISTA DEVICES
  eos_facts:
  when: vendor == "arista"

- name: GET FACTS FROM CISCO NXOS DEVICES
  nxos_facts:
  when: vendor == "cisco"
```

Roles

- re-usable abstraction of code (tasks, variables, and handlers)
- Ansible Galaxy shares roles

```
site.yml
inventory
roles/
   common/
     files/
     templates/
     tasks/
     handlers/
     vars/
     defaults/
   vlans/
     templates/
     tasks/
       main.yml
       eos.yml
       nxos.yml
       junos.yml
     vars/
   snmp/
     templates/
     tasks/
     vars/
```

```
---

- name: SAMPLE PLAYBOOK USING ROLES
  hosts: leaves
  connection: network_cli
  gather_facts: no

  roles:
     - common
     - vlans
```

- directory structure is an example
- no need to have tasks or vars for every role
- files in each sub-directory are called `main.yml`

Parameterized Roles

```yaml
---

- hosts: spine
  connection: network_cli
  gather_facts: no
  roles:
    - common
    - spine
    - role: vlan
      vlan_id: 10
    - role: snmp
      contact: Bob
```

VLAN Role

```yaml
# main playbook
.---

  - name: DC P1
    hosts: datacenter
    connection: network_cli
    gather_facts: no
    roles:
      - vlans
```

```yaml
# roles/vlans/tasks/main.yml
---

- name: ARISTA VLANs
  eos_vlan:
    vlanid: "{{ item['id'] }}"
  loop: "{{ vlans }}"
  when: vendor == "arista"

- name: CISCO VLANs
  nxos_vlan:
    vlan_id: "{{ item['id'] }}"
  loop: "{{ vlans }}"
  when: vendor == "cisco"
```

```ini
[datacenter]
spine1 vendor=arista
n9k1   vendor=cisco
```

```yaml
# group_vars/all.yml
---

vlans:
  - id: 10
    name: web_servers
  - id: 20
  - id: 30
    name: db_servers
```

VLAN Role Improved

```
# main playbook
.---

  - name: DC P1
    hosts: datacenter
    connection: network_cli
    gather_facts: no
    roles:
      - vlans
```

```
# roles/vlans/tasks/main.yml
---
- include: "{{ vendor }}.yml"
```

```
# roles/vlans/tasks/arista.yml
---
- name: ARISTA VLANs
  eos_vlan:
    vlanid: "{{ item['id'] }}"
  loop: "{{ vlans }}"
```

```

---
# roles/vlans/tasks/cisco.yml
- name: CISCO VLANs
  nxos_vlan:
    vlan_id: "{{ item['id'] }}"
```

```
[datacenter]
spine1 vendor=arista
n9k1   vendor=cisco
```

```
# group_vars/all.yml
---

vlans:
  - id: 10
    name: web_servers
  - id: 20
  - id: 30
    name: db_servers
```

Summary

- Think about the functions, features, or applications of the device
- Big Picture vs. Details
- Encapsulate
- For networking- think multi-vendor features

Lab Time

- Lab 18 - Creating an Ansible Role
    - Create a multi-vendor VLAN role that works with Cisco and Arista devices

3rd Party Modules

Exploring NAPALM, NTC and more modules

Introduction to NAPALM

Managing Device Configuration (includes restoring configurations)

NAPALM

NAPALM (Network Automation and Programmability Abstraction Layer with Multivendor support) is a Python library that implements a set of functions to interact with different network device Operating Systems using a unified API.

NAPALM supports several methods to connect to the devices, to manipulate configurations or to retrieve data.

Also has associated Ansible Modules

https://napalm.readthedocs.io/en/latest/

NAPALM

Three core functions:

- Retrieving Data
- Declarative Configuration Management
- Deployment Validation

All three are done in a uniform and vendor-neutral fashion

NAPALM Support Matrix

- Palo Alto PANOS
- Cisco IOS
- Cisco NX-OS
- Cisco IOS-XR
- Arista EOS
- Juniper Junos
- IBM
- Pluribus
- FortiOS
- Cumulus Linux
- Actively growing

Retrieving Data

Uses a uniform and consistent data model across all device types supported by NAPALM

- Facts
- ARP Table
- BGP Configuration
- BGP Neighbors
- BGP Neighbor Detail
- Interface
- Interface Counters
- LLDP neighbors
- LLDP neighbors detail
- NTP Peers
- NTP Stats
- NTP Servers
- ... plus another dozen and actively growing...

NAPALM Facts

Network Device Facts

```
{
    "os_version": "4.15.2F-2663444.4152F",
    "uptime": 5837,
    "interface_list": [
        "Ethernet1",
        "Ethernet2",
        "Ethernet3",
        "Ethernet4",
        "Ethernet5",
        "Ethernet6",
        "Ethernet7",
        "Management1"
    ],
    "vendor": "Arista",
    "serial_number": "",
    "model": "vEOS",
    "hostname": "eos-spine1",
    "fqdn": "eos-spine1.ntc.com"
}
>>>
```

Interfaces


```
{
    "Management1": {
        "is_enabled": true,
        "description": "",
        "last_flapped": 1467419703.0212176,
        "is_up": true,
        "mac_address": "2c:c2:60:0d:52:90",
        "speed": 1000
    },
    "Ethernet2": {
        "is_enabled": true,
        "description": "",
        "last_flapped": 1467419702.7812023,
        "is_up": true,
        "mac_address": "2c:c2:60:12:98:52",
        "speed": 1000
    },
    "Ethernet3": {
        "is_enabled": true,
        "description": "",
        "last_flapped": 1467419702.7812028,
        "is_up": true,
        "mac_address": "2c:c2:60:60:20:9b",
        "speed": 1000
    },
    "Ethernet1": {
        "is_enabled": true,
        "description": "",
        "last_flapped": 1467419702.781203,
        "is_up": true,
        "mac_address": "2c:c2:60:48:80:70",
        "speed": 1000
    },
    "Ethernet5": {
        "is_enabled": true,
        "description": "",
        "last_flapped": 1467419702.8092043,
        "is_up": true,
        "mac_address": "2c:c2:60:40:8d:10",
        "speed": 1000
    },
    "Ethernet4": {
        "is_enabled": true,
        "description": "",
        "last_flapped": 1467419702.7692015,
        "is_up": true,
        "mac_address": "2c:c2:60:2e:c6:f8",
        "speed": 1000
    }
}
```

Layer 3 Interfaces

Get Interfaces IP Addresses


```
{
   "Management1":{
      "ipv4":{
         "10.0.0.11":{
            "prefix_length":24
         }
      },
      "ipv6":{

      }
   }
}
```

Environment

Device Environment Status


```
{
    "fans": {},
    "cpu": {
        "0": {
            "%usage": 5.4
        }
    },
    "temperature": {},
    "power": {},
    "memory": {
        "available_ram": 99060,
        "used_ram": 1798476
    }
}
```

NAPALM Configuration Management

Two main ways to manage device configurations with NAPALM

Configuration Replace

- Declarative configuration always pushing the full configuration
- Only commands required to get the device into its intended state are applied
- No "negation (no)" commands are sent to the device

Configuration Merge

- Send a set of commands or configuration stanza
- Only commands required to get the device into its intended state are applied
- You can use the merge for declarative management on a stanza based on OS

It does vary based on operating system.

NAPALM Configuration Management

Example Workflow

Works slightly different than based on individual drivers and operating systems.

1. Connect to Device
2. Copy desired configuration to device (checkpoint file/rollback, candidate configuration, config session, bootflash as candidate_config.txt)
3. Use a vendor command to view diffs
4. Use a vendor command apply configuration changes
5. Optionally, rollback to a config that exists in the file system.

Note: you dictate if the supplied configuration is a full config file or partial configuration

Configuration Replace

Focus on desired configuration commands.

There are no `no` commands used. The underlying OS generates the diffs (for most NAPALM drivers).

```
$ more diffs/csr1.diffs
+hostname csr1
-hostname csr_old_name
-interface Loopback100
 -ip address 1.1.1.1 255.255.255.255
-interface Loopback200
 -ip address 22.2.1.1 255.255.255.255
-ip route 10.1.1.0 255.255.255.0 192.0.1.1
```

Full configuration is sent to the device, but only diffs are applied. You do not need to worry about going from A to B - you just focus on B.

Configuration Merge

You can use NAPALM for declarative management for a sectional config too.

Current BGP Config

```
router bgp 65512
   neighbor 10.0.0.0 remote-as 65500
   neighbor 10.0.0.0 maximum-routes 12000
   neighbor 10.0.0.1 remote-as 65512
   neighbor 10.0.0.1 maximum-routes 12000
   network 20.20.20.0/24
!
```

Desired BGP Config (file sent to device)

```
router bgp 65512
   neighbor 10.0.0.2 remote-as 65500
   neighbor 10.0.0.2 maximum-routes 12000
   neighbor 10.0.0.1 remote-as 65512
   neighbor 10.0.0.1 maximum-routes 12000
   neighbor 10.0.0.10 remote-as 65512
   network 100.0.100.0/24
!
```

Configuration Merge

You can use NAPALM for declarative management for a sectional config too.

Current BGP Config

```
router bgp 65512
   neighbor 10.0.0.0 remote-as 65500
   neighbor 10.0.0.0 maximum-routes 12000
   neighbor 10.0.0.1 remote-as 65512
   neighbor 10.0.0.1 maximum-routes 12000
   network 20.20.20.0/24
!
```

Desired BGP Config (file sent to device)

```
router bgp 65512
   neighbor 10.0.0.2 remote-as 65500
   neighbor 10.0.0.2 maximum-routes 12000
   neighbor 10.0.0.1 remote-as 65512
   neighbor 10.0.0.1 maximum-routes 12000
   neighbor 10.0.0.10 remote-as 65512
   network 100.0.100.0/24
!
```

Diff Generated by NAPALM

```
    neighbor 10.0.0.0 maximum-routes 12000
    neighbor 10.0.0.1 remote-as 65512
    neighbor 10.0.0.1 maximum-routes 12000
+   neighbor 10.0.0.2 remote-as 65500
+   neighbor 10.0.0.2 maximum-routes 12000
+   neighbor 10.0.0.10 remote-as 65512
+   neighbor 10.0.0.10 maximum-routes 12000
    network 20.20.20.0/24
+   network 100.0.100.0/24
 !
 management api http-commands
    protocol http
```

Configuration Merge (Advanced)

You can use NAPALM for declarative management for a sectional config too.

Current BGP Config

```
router bgp 65512
   neighbor 10.0.0.0 remote-as 65500
   neighbor 10.0.0.0 maximum-routes 12000
   neighbor 10.0.0.1 remote-as 65512
   neighbor 10.0.0.1 maximum-routes 12000
   network 20.20.20.0/24
!
```

Desired BGP Config (file sent to device)

```
no router bgp 65512
router bgp 65512
   neighbor 10.0.0.2 remote-as 65500
   neighbor 10.0.0.2 maximum-routes 12000
   neighbor 10.0.0.1 remote-as 65512
   neighbor 10.0.0.1 maximum-routes 12000
   neighbor 10.0.0.10 remote-as 65512
   network 100.0.100.0/24
!
```

Configuration Merge (Advanced)

You can use NAPALM for declarative management for a sectional config too.

Current BGP Config

```
router bgp 65512
   neighbor 10.0.0.0 remote-as 65500
   neighbor 10.0.0.0 maximum-routes 12000
   neighbor 10.0.0.1 remote-as 65512
   neighbor 10.0.0.1 maximum-routes 12000
   network 20.20.20.0/24
!
```

Desired BGP Config (file sent to device)

```
no router bgp 65512
router bgp 65512
   neighbor 10.0.0.2 remote-as 65500
   neighbor 10.0.0.2 maximum-routes 12000
   neighbor 10.0.0.1 remote-as 65512
   neighbor 10.0.0.1 maximum-routes 12000
   neighbor 10.0.0.10 remote-as 65512
   network 100.0.100.0/24
!
```

Diff Generated by NAPALM

```
 router bgp 65512
-   neighbor 10.0.0.0 remote-as 65500
-   neighbor 10.0.0.0 maximum-routes 12000
    neighbor 10.0.0.1 remote-as 65512
    neighbor 10.0.0.1 maximum-routes 12000
-   network 20.20.20.0/24
+   neighbor 10.0.0.2 remote-as 65500
+   neighbor 10.0.0.2 maximum-routes 12000
+   neighbor 10.0.0.10 remote-as 65512
+   neighbor 10.0.0.10 maximum-routes 12000
+   network 100.0.100.0/24
 !
```

Be cautious of device support. This is based on NAPALM driver implementation which is dictated by vendor OS support. This example is EOS.

NAPALM Ansible Module

Parameter	required	default	choices	comments
username	yes			Username
dev_os	yes			OS running the device
config_file	yes			Where to load the configuration from.
hostname	yes			IP or FQDN of the device you want to connect to
replace_config	no			If set to True the entire configuration on the device will be replaced during the commit. If set to False, we will merge the new config with the existing one. Default is False.
diff_file	no			A file where to store the "diff" between the running configuration and the new configuration. If it's not set the diff between configurations is not saved.
password	yes			Password
commit_changes	yes			If set to True the configuration will be actually replaced. If the set to False, we will not apply the changes, just check the differences.

Backing up and Restoring Configurations

Backing Up Configuration Files

- **ntc_show_command** - built on top of `pyntc` and part of the **ntc-ansible** project
- Supports Junos, IOS, NX-OS, and EOS
- Save the running configuration as a file to the Ansible control host.

```
- name: BACKUP ALL CONFIGURATIONS
  hosts: all
  connection: local
  gather_facts: no

  tasks:
    - name: BACKUP CONFIG
      ntc_show_command:
        platform: cisco_ios
        command: show running
        provider: "{{ connection_details }}"
        local_file: "./backups/{{ inventory_hostname }}.cfg"
        template_dir: "{{ ntc_template_path }}"
```

There are other way to backup config files too, which we'll look at later

lineinfile

- Add/Remove a line to a file
- Use regexes to match for position
- Insert before/after

```
- name: Ensure a line does not exist
  lineinfile:
    line: "Building configuration..."
    dest: "backups/{{ inventory_hostname }}.cfg"
    state: "absent"
```

```
- name: Ensure the line that matches the regex does not exist
  lineinfile:
    dest: "backups/{{ inventory_hostname }}.cfg"
    regexp: "Current configuration .*"
    state: "absent"
```

```
!
Building configuration...

Current configuration 3942 Bytes
!
hostname nycr01
```

Selectively Execute Tasks and/or Plays

- Can assign one or more tags (using a list)
- `tags` can be used for plays and/or tasks

```
- name: VLAN CHANGE
  nxos_vlan:
    vlan_id: "10"
  tags: vlan
```

```
- name: VLAN DEMO
  nxos_vlan:
    vlan_id: 10
  tags:
    - cisco
    - nxos
    - vlan
```

```
ansible-playbook -i inventory playbook.yml --tags=vlan
```

```
- name: DATA CENTER AUTOMATION
  hosts: all
  connection: network_cli
  gather_facts: no
  tags: datacenter
```

```
ansible-playbook -i inventory playbook.yml --tags=datacenter
```

Limit the Devices being Automated

- Use `--limit` command line flag to limit the groups or hosts being automated.
- Must be a sub-set of the devices in the `hosts: group` in the playbook.

```
ansible-playbook -i inventory playbook.yml --limit nxos
```

```
ansible-playbook -i inventory playbook.yml --limit nxos-spine1
```

```
ansible-playbook -i inventory playbook.yml --limit nxos,eos,csr1
```

NAPALM Ansible Module restore and tags

- NAPALM module using a task attribute called `tags`.
- Tags are used to selectively run particular tasks

```
# backup tasks

- name: DEPLOY CONFIGURATION
  napalm_install_config:
    dev_os: "{{ ansible_network_os }}"
    config_file: "./backups/{{ inventory_hostname }}.cfg"   # file with commands to apply
    commit_changes: true                                  # apply changes (or only generate diffs)
    replace_config: true                                  # full config replace or merge (just a few commands)
    diff_file: "./diffs/{{ inventory_hostname }}.diffs"     # file to save diffs.  You can view diffs before committing changes
    provider: "{{ connection_details }}"
  tags: deploy
```

Save and Run the playbook.

```
$ ansible-playbook -i inventory backup.yml --limit nxos --tags=deploy
```

Other Backup Examples

- NTC BACKUP

```
- name: BACKUP CONFIG
  ntc_show_command:
    platform: cisco_ios
    command: show running
    provider: "{{ connection_details }}"
    local_file: "./backups/{{ inventory_hostname }}.cfg
    template_dir: "{{ ntc_template_path }}"
```

- NAPALM BACKUP

```
- name: BACKUP CONFIG NAPALM
  napalm_get_facts:
    dev_os: "{{ ansible_network_os }}"
    provider: "{{ connection_details }}"
    filter:
    - "config"

- name: STORE BACKUP IN FILE
  copy:
    content: "{{ ansible_facts['napalm_config']['runnin
    dest: ./backups/{{ inventory_hostname }}-napalm.cfg
```

- CORE BACKUP

```
- name: BACKUP CONFIG CORE
  ios_config:
    backup: True
```

  - backup parameter

    This argument will cause the module to create a full backup of the current `running-config` from the remote device before any changes are made.

    The backup file is written to the `backup` folder in the playbook root directory or role root directory, if playbook is part of an ansible role. If the directory does not exist, it is created. [Default: no] type: bool version_added: 2.2

Lab Time

- Lab 19 - Backup and Restore Network Configurations Part 1
- Lab 20 - Backup and Restore Network Configurations Part 2

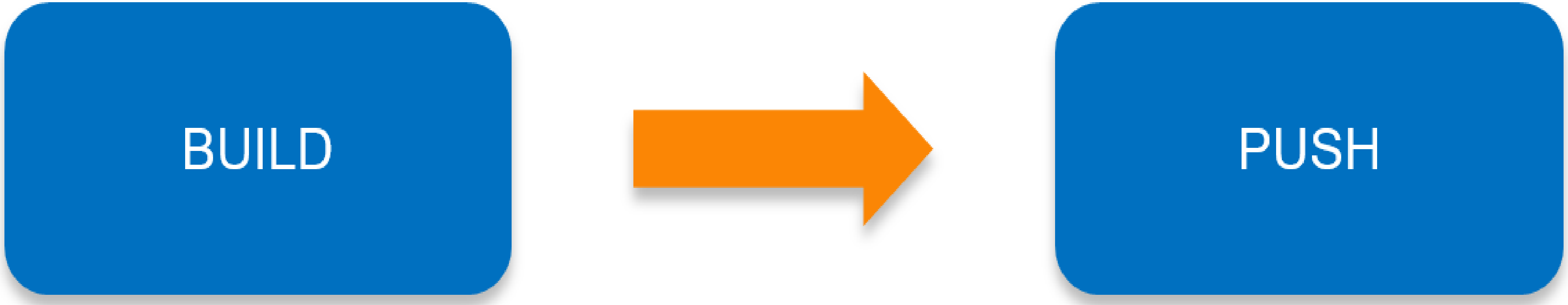
Build / Push Configuration Management

Ansible for Network Automation

BONUS

Build / Push

- One approach to *pushing* configurations is to use the *build / push* method
- Especially useful for initial device provisioning



BUILD

- De-compose existing configuration files
- Create templates
- De-couple variables (inputs) from CLI syntax

PUSH

- Push Configuration to Device
    - ZTP / POAP
    - Ansible

Build

- De-compose existing configuration files (example)
  - VLANs
  - Interfaces
  - SNMP
- Create template(s)
- Create YAML Variable(s) files
- Render templates with variables

Network Configuration Building

Jinja

TEMPLATE

YAML

VARIABLES

- inventory file
- group_vars
- host_vars
- vars files
- other
    - register
    - Dynamic inv

RENDERING ENGINE

`Ansible`

Jinja2 Templates

Standard config file

```
snmp-server community PUBLIC123 RO 5
snmp-server community PRIVATE123 RW 95
snmp-server location GLOBAL
snmp-server contact LOCAL_ADMIN
snmp-server host 1.1.1.1

vlan 10
 name web_servers
vlan 20
vlan 30
 name db_servers

interface Ethernet1
  no shutdown
interface Ethernet2
  no shutdown
interface Ethernet3
  shutdown
interface Ethernet4
  no shutdown
interface Ethernet5
  shutdown
```

Jinja2 Templates

Standard config file

```
snmp-server community PUBLIC123 RO 5
snmp-server community PRIVATE123 RW 95
snmp-server location GLOBAL
snmp-server contact LOCAL_ADMIN
snmp-server host 1.1.1.1

vlan 10
 name web_servers
vlan 20
vlan 30
 name db_servers

interface Ethernet1
  no shutdown
interface Ethernet2
  no shutdown
interface Ethernet3
  shutdown
interface Ethernet4
  no shutdown
interface Ethernet5
  shutdown
```

De-constructed config as a template

```
snmp-server community {{ snmp_ro }} RO 5
snmp-server community {{ snmp_rw }} RW 95
snmp-server location {{ snmp_location }}
snmp-server contact {{ snmp_contact }}
snmp-server host {{ snmp_trap_dest }}

{% for vlan in vlans %}
vlan {{ vlan.id }}
{% if vlan.get('name') %}
 name {{ vlan.name }}
{% endif %}
{% endfor %}

{% for interface in interfaces %}
interface {{ interface.name }}
{% if interface.admin == 'up' %}
  no shutdown
{% elif interface.admin == 'down' %}
  shutdown
{% endif %}
{% endfor %}
```

Templates & Variables

`templates/config.j2`

```
snmp-server community {{ snmp_ro }} RO 5
snmp-server community {{ snmp_rw }} RW 95
snmp-server location {{ snmp_location }}
snmp-server contact {{ snmp_contact }}
snmp-server host {{ snmp_trap_dest }}


{% for vlan in vlans %}
vlan {{ vlan.id }}
{% if vlan.get('name') %}
 name {{ vlan.name }}
{% endif %}
{% endfor %}


{% for interface in interfaces %}
interface {{ interface.name }}
{% if interface.admin == 'up' %}
  no shutdown
{% elif interface.admin == 'down' %}
  shutdown
{% endif %}
{% endfor %}
```

Templates & Variables

templates/config.j2

```
snmp-server community {{ snmp_ro }} RO 5
snmp-server community {{ snmp_rw }} RW 95
snmp-server location {{ snmp_location }}
snmp-server contact {{ snmp_contact }}
snmp-server host {{ snmp_trap_dest }}


{% for vlan in vlans %}
vlan {{ vlan.id }}
{% if vlan.get('name') %}
 name {{ vlan.name }}
{% endif %}
{% endfor %}


{% for interface in interfaces %}
interface {{ interface.name }}
{% if interface.admin == 'up' %}
  no shutdown
{% elif interface.admin == 'down' %}
  shutdown
{% endif %}
{% endfor %}
```

group_vars/all.yml

```
snmp_ro: PUBLIC123
snmp_rw: PRIVATE123
snmp_location: GLOBAL
snmp_contact: LOCAL_ADMIN
snmp_trap_dest: 1.1.1.1


vlans:
  - id: 10
    name: web_servers
  - id: 20
  - id: 30
    name: db_servers

interfaces:
  - name: Ethernet1
    admin: up
  - name: Ethernet2
    admin: up
  - name: Ethernet3
    admin: down
  - name: Ethernet4
    admin: up
  - name: Ethernet5
    admin: down
```

Data Modeling & Variables

Think through data inputs and impact it has on templates

```yaml
snmp_ro: PUBLIC123
snmp_rw: PRIVATE123
snmp_location: GLOBAL
snmp_contact: LOCAL_ADMIN
snmp_trap_dest: 1.1.1.1



vlans:
  - id: 10
    name: web_servers
  - id: 20
  - id: 30
    name: db_servers
```

```yaml
snmp:
  ro:
    - PUBLIC123
  rw:
    - PRIVATE123
  location: GLOBAL
  contact: LOCAL_ADMIN
  trap_dest:
    - 1.1.1.1


vlans:
  '10':
    name: web_servers
  '20': {}
  '30':
    name: db_servers
```

Building Configurations

`build-push.yml`

```yaml
---
  - name: BUILD CONFIGS
    hosts: all
    connection: network_cli
    gather_facts: no

    tasks:

      - name: BUILD NETWORK CONFIGURATIONS
        template:
          src: config.j2
          dest: "configs/{{ inventory_hostname }}.conf"
```

Building Configurations

`build-push.yml`

```yaml
---
  - name: BUILD CONFIGS
    hosts: all
    connection: network_cli
    gather_facts: no

    tasks:

      - name: BUILD NETWORK CONFIGURATIONS
        template:
          src: config.j2
          dest: "configs/{{ inventory_hostname }}.conf"
```

```
.
├── build-push.yml
├── configs
│   ├── leaf1.conf
│   ├── leaf2.conf
│   ├── leaf3.conf
│   ├── leaf4.conf
├── group_vars
│   └── all.yml        # variables in scope by all devices
├── inventory
└── templates
    └── config.j2
```

Optimizing the BUILD Process

- Build a directory per device

```
- name: ENSURE DIRECTORY EXISTS PER DEVICE (AND PARTIALS SUB-DIR)
  file:
    path: "/home/ntc/ansible/configs/{{ inventory_hostname }}/partials"
    state: directory
```

Optimizing the BUILD Process

- Build a directory per device

```
    - name: ENSURE DIRECTORY EXISTS PER DEVICE (AND PARTIALS SUB-DIR)
      file:
        path: "/home/ntc/ansible/configs/{{ inventory_hostname }}/partials"
        state: directory
```

- De-construct single Jinja2 Template into individual templates

```
# templates/01_snmp.j2

snmp-server community {{ snmp_ro }} RO 5
snmp-server community {{ snmp_rw }} RW 95
snmp-server location {{ snmp_location }}
snmp-server contact {{ snmp_contact }}
snmp-server host {{ snmp_trap_dest }}
```

```
# templates/02_vlans.j2

{% for vlan in vlans %}
vlan {{ vlan.id }}
{% if vlan.get('name') %}
 name {{ vlan.name }}
{% endif %}
{% endfor %}
```

```
# templates/03_interfaces.j2

{% for interface in interfaces %}
interface {{ interface.name }}
{% if interface.admin == 'up' %}
  no shutdown
{% elif interface.admin == 'down' %}
  shutdown
{% endif %}
{% endfor %}
```

Optimizing the BUILD Process (cont'd)

- Create configuration snippets

```
- name: BUILD NETWORK CONFIGURATIONS
  template:
    src: templates/01_snmp.j2
    dest: "configs/{{ inventory_hostname }}/partials/01_snmp.conf"

- name: BUILD NETWORK CONFIGURATIONS
  template:
    src: templates/02_vlans.j2
    dest: "configs/{{ inventory_hostname }}/partials/02_vlans.conf"

- name: BUILD NETWORK CONFIGURATIONS
  template:
    src: templates/03_interfaces.j2
    dest: "configs/{{ inventory_hostname }}/partials/03_interfaces.conf"
```

- Assemble configuration snippets

```
- name: ASSEMBLE PARTIAL CONFIGURATIONS PER DEVICE INTO SINGLE CONFIG FILE
  assemble:
    src: "configs/{{ inventory_hostname }}/partials"
    dest: "configs/{{ inventory_hostname }}/{{ inventory_hostname }}.conf"
```

Optimizing the BUILD Process (cont'd)

- Why not optimize the template module from multiple tasks to a single one?

```
- name: BUILD NETWORK CONFIGURATIONS
  template:
    src: templates/01_snmp.j2
    dest: "configs/{{ inventory_hostname }}/partials/01_snmp.conf"

- name: BUILD NETWORK CONFIGURATIONS
  template:
    src: templates/02_vlans.j2
    dest: "configs/{{ inventory_hostname }}/partials/02_vlans.conf"

- name: BUILD NETWORK CONFIGURATIONS
  template:
    src: templates/03_interfaces.j2
    dest: "configs/{{ inventory_hostname }}/partials/03_interfaces.conf"
```

Optimizing the BUILD Process (cont'd)

- Why not optimize the template module from multiple tasks to a single one?

```
- name: BUILD NETWORK CONFIGURATIONS
  template:
    src: templates/01_snmp.j2
    dest: "configs/{{ inventory_hostname }}/partials/01_snmp.conf"

- name: BUILD NETWORK CONFIGURATIONS
  template:
    src: templates/02_vlans.j2
    dest: "configs/{{ inventory_hostname }}/partials/02_vlans.conf"

- name: BUILD NETWORK CONFIGURATIONS
  template:
    src: templates/03_interfaces.j2
    dest: "configs/{{ inventory_hostname }}/partials/03_interfaces.conf"
```

- Use a loop (iterator) instead of doing each one individually

```
  - name: BUILD NETWORK CONFIGURATIONS
    template:
      src: "{{ item }}"
      dest: "configs/{{ inventory_hostname }}/partials/{{ item | basename | replace('j2', 'conf') }}"
    with_fileglob:
      - templates/*
```

Play 1 - *BUILD*

```yaml
---

  - name: BUILD PROCESS
    hosts: all
    connection: local
    gather_facts: no

    tasks:

      - name: ENSURE DIRECTORY EXISTS PER DEVICE (AND PARTIALS SUB-DIR)
        file:
          path: "/home/ansible/configs/{{ inventory_hostname }}/partials"
          state: directory

      - name: BUILD NETWORK CONFIGURATIONS
        template:
          src: "{{ item }}"
          dest: "configs/{{ inventory_hostname }}/partials/{{ item | basename | replace('j2', 'conf') }}"
        with_fileglob:
          - templates/*

      - name: ASSEMBLE PARTIAL CONFIGURATIONS PER DEVICE INTO SINGLE CONFIG FILE
        assemble:
          src: "configs/{{ inventory_hostname }}/partials"
          dest: "configs/{{ inventory_hostname }}/{{ inventory_hostname }}.conf"
```

Play 2 - *PUSH*

```yaml
---

  - name: DEPLOY CONFIGURATIONS USING NAPALM
    hosts: all
    connection: local
    gather_facts: no

    tasks:

    - name: DEPLOY MERGE
      napalm_install_config:
        provider: "{{ connection_details }}"
        dev_os: eos
        config_file: "configs/{{ inventory_hostname }}/{{ inventory_hostname }}.conf"
        commit_changes: true
        replace_config: false
        diff_file: "configs/{{ inventory_hostname }}/{{ inventory_hostname }}.diff"
```

```
# inventory
[leaves]
leaf1
leaf2
leaf3
leaf4
leaf5
leaf6
```

Build /

```yaml
---

  - name: BUILD PROCESS
    hosts: all
    connection: local
    gather_facts: no
    tags: build

    tasks:

      - name: ENSURE DIRECTORY EXISTS PER DEVICE (AND PARTIALS SUB-DIR)
        file:
          path: "/home/ansible/configs/{{ inventory_hostname }}/partials"
          state: directory

      - name: BUILD NETWORK CONFIGURATIONS
        template:
          src: "{{ item }}"
          dest: "configs/{{ item | basename | replace('j2', 'conf') }}"
        with_fileglob:
          - templates/*

      - name: ASSEMBLE PARTIAL CONFIGURATIONS PER DEVICE INTO SINGLE CONFIG FILE
        assemble:
          src: "configs/{{ inventory_hostname }}/partials"
          dest: "configs/{{ inventory_hostname }}/{{ inventory_hostname }}.conf"
        # CONTINUE TO NEXT PAGE FOR THE REST OF THE PLAYBOOK
```

/ Push Playbook (cont'd)

```
- name: DEPLOY CONFIGURATIONS USING NAPALM
  hosts: all
  connection: local
  gather_facts: no
  tags: deploy

  tasks:

  - name: DEPLOY MERGE
    napalm_install_config:
      provider: "{{ connection_details }}"
      config_file: "configs/{{ inventory_hostname }}/{{ inventory_hostname }}.conf"
      commit_changes: true
      replace_config: false
      diff_file: "configs/{{ inventory_hostname }}/{{ inventory_hostname }}.diff"
```

Executing the Playbook

```
$ ansible-playbook build-push.yml --tags=build
```

- Only the push tasks

```
$ ansible-playbook build-push.yml --tags=push
```

- All

```
$ ansible-playbook build-push.yml
```

Executing the Playbook (cont'd)

```
$ ansible-playbook -i inventory build-push.yml

PLAY [BUILD configs] *******************************************************

TASK: [ENSURE DIRs created per device] *************************************
changed: [leaf1]
changed: [leaf2]
changed: [leaf3]
changed: [leaf4]
changed: [leaf5]
changed: [leaf6]

TASK: [BUILD service configs using template modules] ***********************
changed: [leaf1] => (item=/home/ntc/ansible/templates/02_vlans.j2)
changed: [leaf1] => (item=/home/ntc/ansible/templates/01_snmp.j2)
changed: [leaf1] => (item=/home/ntc/ansible/templates/03_interfaces.j2)

TASK: [ASSEMBLE CONFIGS per device] ****************************************
changed: [leaf1]

PLAY [Push configs] ********************************************************

TASK: [deploy configs] *****************************************************
changed: [leaf1]

PLAY RECAP *****************************************************************
leaf1                      : ok=4    changed=4    unreachable=0    failed=0
```

Executing the Playbook (cont'd)

BEFORE:

```
.
├── build-push.yml
├── configs
├── group_vars
│   └── all.yml
├── inventory
└── templates
    ├── 01_snmp.j2
    ├── 02_vlans.j2
    └── 03_interfaces.j2
```

AFTER (for 2 devices):

```
.
├── build-push.yml
├── configs
│   └── leaf1
│       ├── leaf1.conf
│       └── partials
│           ├── 01_snmp.conf
│           ├── 02_vlans.conf
│           └── 03_interfaces.conf
│   └── leaf2
│       ├── leaf2.conf
│       └── partials
│           ├── 01_snmp.conf
│           ├── 02_vlans.conf
│           └── 03_interfaces.conf
├── group_vars
│   └── all.yml
├── inventory
└── templates
    ├── 01_snmp.j2
    ├── 02_vlans.j2
    └── 03_interfaces.j2
```

List Tasks

- View all tasks that will be executed
- Great for architecture discussions and planning playbooks

```
$ ansible-playbook -i inventory build-push.yml --list-tasks

playbook: build-push.yml

  play #1 (BUILD PROCESS):  TAGS: [build]
    ENSURE DIRECTORY EXISTS PER DEVICE (AND PARTIALS SUB-DIR) TAGS: [build]
    BUILD NETWORK CONFIGURATIONS    TAGS: [build]
    ASSEMBLE PARTIAL CONFIGURATIONS PER DEVICE INTO SINGLE CONFIG FILE: [build]

  play #2 (Push configs):   TAGS: [push]
    DEPLOY MERGE  TAGS: [push]
```

Summary

- Build requires building out one or more templates for your environment
  - Template per role
  - Template per service per role
  - Templates could become fragile
  - Good for initial device configuration
  - Possible to use ALL the time (install_config)
- Push
  - NAPALM provides a nice abstraction for pushing full and partial configurations to several device types
  - Always always test

Lab Time

- Lab 24 - Build / Push with NAPALM3
  - You will use the "template" and "napalm_install_config" modules
  - Choose any *1* vendor to complete this lab

NTC and More Modules

ntc_get_facts

- uptime - Uptime of the device
- vendor - vendor of the device
- model - Device model
- hostname - Hostname of the device
- fqdn - FQDN of the device
- os_version - String with the OS version running on the device.
- serial_number - Serial number of the device
- interfaces - List of the interfaces of the device
- vlans - List of the vlans configured on the device

```
- ntc_get_facts:
    provider: "{{ ntc_provider }}"
    platform: "{{ ntc_vendor }}_{{ ansible_network_os }}_{{ ntc_api }}"
```

ntc_show_command

- Multi-vendor Ansible module to streamline converting raw text into JSON key/value pairs
- Leverages TextFSM
- netmiko is used for transport to enable support for *all* devices

```
tasks:

  - name: GET DATA
    ntc_show_command:
      connection: ssh
      platform: cisco_nxos
      command: 'show vlan'
      provider: "{{ ntc_provider }}"
      template_dir: "/etc/ntc/ansible/library/ntc-ansible/ntc-templates/templates"
```

ntc_show_command

- JSON data now available to re-use
- Use as inputs to other modules or in templates (docs)

```
TASK: [GET DATA] ***************************************************************
ok: [n9k1] => {"changed": false, "response": [{"name": "default", "status": "active", "vlan_id": "1"}, {"name": "VLAN0002",
"status": "active", "vlan_id": "2"}, {"name": "VLAN0003", "status": "active", "vlan_id": "3"}, {"name": "VLAN0004", "status":
"active", "vlan_id": "4"}, {"name": "VLAN0005", "status": "active", "vlan_id": "5"}, {"name": "VLAN0006", "status": "active",
"vlan_id": "6"}, {"name": "VLAN0007", "status": "active", "vlan_id": "7"}, {"name": "VLAN0008", "status": "active", "vlan_id":
"8"}, {"name": "VLAN0009", "status": "active", "vlan_id": "9"}, {"name": "VLAN10_WEB", "status": "active", "vlan_id": "10"},
{"name": "VLAN0011", "status": "active", "vlan_id": "11"}, {"name": "VLAN0012", "status": "active", "vlan_id": "12"}, {"name":
"VLAN0013", "status": "active", "vlan_id": "13"}, {"name": "VLAN0014", "status": "active", "vlan_id": "14"}, {"name":
"VLAN0015", "status": "active", "vlan_id": "15"}, {"name": "VLAN0016", "status": "active", "vlan_id": "16"}, {"name":
"VLAN0017", "status": "active", "vlan_id": "17"}, {"name": "VLAN0018", "status": "active", "vlan_id": "18"}, {"name":
"VLAN0019", "status": "active", "vlan_id": "19"}, {"name": "peer_keepalive", "status": "active", "vlan_id": "20"}, {"name":
"VLAN0022", "status": "active", "vlan_id": "22"}, {"name": "VLAN0030", "status": "active", "vlan_id": "30"}, {"name":
"VLAN0040", "status": "active", "vlan_id": "40"}, {"name": "native", "status": "active", "vlan_id": "99"}, {"name": "VLAN0100",
"status": "active", "vlan_id": "100"}, {"name": "VLAN0101", "status": "active", "vlan_id": "101"}, {"name": "VLAN0102",
"status": "active", "vlan_id": "102"}, {"name": "VLAN0103", "status": "active", "vlan_id": "103"}, {"name": "VLAN0104",
"status": "active", "vlan_id": "104"}, {"name": "VLAN0105", "status": "active", "vlan_id": "105"}, {"name": "VLAN0123",
"status": "active", "vlan_id": "123"}, {"name": "VLAN0200", "status": "active", "vlan_id": "200"}]}
```

snmp_facts

- Multi-vendor fact gathering using SNMP
- Returns:
  - All IPv4 addresses
  - interfaces
  - sys contact
  - sys description
  - uptime

```yaml
- snmp_facts:
    host: "{{ inventory_hostname }}"
    version: v2c
    community: networktocode
```

snmp_device_version

The `snmp_device_version` module can be used to discover the device vendor, os, and version. These items are returned as variables `ansible_device_vendor` , `ansible_device_os` , and `ansible_device_version` .

Example Task:

```
    - name: QUERY DEVICE VIA SNMP
      snmp_device_version:
        community: networktocode
        version: 2c
        host: "{{ inventory_hostname }}"
```

You can use these discovered variables for further processing devices in a dynamic fashion.

```
- ntc_show_command:
    platform: "{{ ansible_device_vendor }}_{{ ansible_device_os }}"
    ...
```

Network Engine and NTC Parsers

Network Engine Overview

- Set of consumable functions distributed as Ansible Roles
- The Network Engine Role extracts data about your network devices as Ansible facts in a JSON data structure, ready to be added to your inventory host facts and/or consumed by Ansible tasks and templates
- You define the data elements you want to extract from each network OS command in parser templates, using either YAML or Google TextFSM syntax
- The initial release of the Network Engine role includes two parser modules:
  - `command_parser` accepts YAML input, uses an internally maintained, loosely defined parsing language based on Ansible playbook directives
  - `textfsm_parser` accepts Google TextFSM input, uses Google TextFSM parsing language

Network Engine Examples

Example 1: Text Input

- show interfaces (Cisco IOS)

```
GigabitEthernet1 is up, line protocol is up
  Hardware is CSR vNIC, address is 2cc2.6031.1341 (bia 2cc2.6031.1341)
  Internet address is 10.0.0.51/24
  MTU 1500 bytes, BW 1000000 Kbit/sec, DLY 10 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
  Full Duplex, 1000Mbps, link type is auto, media type is RJ45
  output flow-control is unsupported, input flow-control is unsupported
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:00:15, output 00:00:04, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/375/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/40 (size/max)
  5 minute input rate 2000 bits/sec, 2 packets/sec
  5 minute output rate 2000 bits/sec, 2 packets/sec
     8063 packets input, 732260 bytes, 0 no buffer
     Received 0 broadcasts (0 IP multicasts)
     0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
     0 watchdog, 0 multicast, 0 pause input
     9171 packets output, 723414 bytes, 0 underruns
     0 output errors, 0 collisions, 0 interface resets
     254 unknown protocol drops
     0 babbles, 0 late collision, 0 deferred
     0 lost carrier, 0 no carrier, 0 pause output
     0 output buffer failures, 0 output buffers swapped out
GigabitEthernet2 is up, line protocol is up
  Hardware is CSR vNIC, address is 2cc2.6069.e08c (bia 2cc2.6069.e08c)
  Description: CONNECTS_CSR3
  Internet address is 10.254.13.1/24
  MTU 1500 bytes, BW 1000000 Kbit/sec, DLY 10 usec,
```

```
---
  - name: USING PARSER TEMPLATES
    hosts: csr1
    connection: network_cli
    gather_facts: no
    roles:
      - ansible-network.network-engine

    tasks:

      - name: SHOW INTERFACES
        ios_command:
          commands: show interfaces
        register: interface_data
```

Example 1: Template File

- show interfaces (Cisco IOS)
- Filename: `show_interfaces.yml`

```yaml
---
- name: parser meta data
  parser_metadata:
    version: 1.0
    command: show interfaces
    network_os: ios

- name: match sections
  pattern_match:
    regex: "^(\\S+) is up,"
    match_all: true
    match_greedy: true
  register: section

- name: match interface values
  pattern_group:
    - name: match name
      pattern_match:
        regex: "^(\\S+)"
        content: "{{ item }}"
      register: name

    - name: match hardware
      pattern_match:
        regex: "\\s+Hardware is ([\\w ]+)"
        content: "{{ item }}"
      register: type
```

Continued

```yaml
    - name: match mtu
      pattern_match:
        regex: "MTU (\\d+)"
        content: "{{ item }}"
      register: mtu

    - name: match description
      pattern_match:
        regex: "Description: (.*)"
        content: "{{ item }}"
      register: description
  loop: "{{ section }}"
  register: interfaces

- name: generate json data structure
  json_template:
    template:
      - key: "{{ item.name.matches.0 }}"
        object:
            - key: name
              value: "{{ item.name.matches.0 }}"
            - key: type
              value: "{{ item.type.matches.0 }}"
            - key: mtu
              value: "{{ item.mtu.matches.0 }}"
            - key: description
              value: "{{ item.description.matches.0 }}"
  loop: "{{ interfaces }}"
  export: true
  export_as: "dict"
  register: interface_facts
```

Example 1: Playbook

- The `command_parser` module has two parameters:
  - `file` : Define the path to YAML file
  - `content` : Define the path to a raw file or the variable that contains the raw output

```
---

- name: USING PARSER TEMPLATES
  hosts: csr1
  connection: network_cli
  gather_facts: no
  roles:
    - ansible-network.network-engine

  tasks:

    - name: SHOW INTERFACES
      ios_command:
        commands: show interfaces
      register: interface_data

    - name: COMMAND PARSER
      command_parser:
        file: "./parsers/ios/show_interfaces.yml"
        content: "{{ interface_data['stdout'][0] }}"
      register: interfaces

    - name: PARSED DATA
      debug:
        var: interfaces
```

```
TASK [PARSED DATA] *********************************************************************
ok: [csr1] => {
    "interfaces": {
        "ansible_facts": {
            "interface_facts": {
                "GigabitEthernet1": {
                    "description": null,
                    "mtu": "1500",
                    "name": "GigabitEthernet1",
                    "type": "CSR vNIC"
                },
                "GigabitEthernet2": {
                    "description": "CONNECTS_CSR3",
                    "mtu": "1500",
                    "name": "GigabitEthernet2",
                    "type": "CSR vNIC"
                },
                "GigabitEthernet4": {
                    "description": "CONNECTS_CSR2",
                    "mtu": "1500",
                    "name": "GigabitEthernet4",
                    "type": "CSR vNIC"
                },
                "Loopback0": {
                    "description": null,
                    "mtu": "1514",
                    "name": "Loopback0",
                    "type": "Loopback"
                },
                "Loopback100": {
                    "description": "OSPF ROUTER ID",
                    "mtu": "1514",
                    "name": "Loopback100",
                    "type": "Loopback"
                }, output omitted...
```

Example 2: Text Input

- show interfaces (Cisco IOS)

```
GigabitEthernet1 is up, line protocol is up
  Hardware is CSR vNIC, address is 2cc2.6031.1341 (bia 2cc2.6031.1341)
  Internet address is 10.0.0.51/24
  MTU 1500 bytes, BW 1000000 Kbit/sec, DLY 10 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
  Full Duplex, 1000Mbps, link type is auto, media type is RJ45
  output flow-control is unsupported, input flow-control is unsupported
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:00:15, output 00:00:04, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/375/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/40 (size/max)
  5 minute input rate 2000 bits/sec, 2 packets/sec
  5 minute output rate 2000 bits/sec, 2 packets/sec
     8063 packets input, 732260 bytes, 0 no buffer
     Received 0 broadcasts (0 IP multicasts)
     0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
     0 watchdog, 0 multicast, 0 pause input
     9171 packets output, 723414 bytes, 0 underruns
     0 output errors, 0 collisions, 0 interface resets
     254 unknown protocol drops
     0 babbles, 0 late collision, 0 deferred
     0 lost carrier, 0 no carrier, 0 pause output
     0 output buffer failures, 0 output buffers swapped out
GigabitEthernet2 is up, line protocol is up
  Hardware is CSR vNIC, address is 2cc2.6069.e08c (bia 2cc2.6069.e08c)
  Description: CONNECTS_CSR3
  Internet address is 10.254.13.1/24
  MTU 1500 bytes, BW 1000000 Kbit/sec, DLY 10 usec,
```

```
---
  - name: USING PARSER TEMPLATES
    hosts: csr1
    connection: network_cli
    gather_facts: no
    roles:
      - ansible-network.network-engine

    tasks:

      - name: SHOW INTERFACES
        ios_command:
          commands: show interfaces
        register: interface_data
```

Example 2: Template File

- show interfaces (Cisco IOS)
- Order is important
- Filename: `show_interfaces.template`

```
Value Required name (\S+)
Value type ([\w ]+)
Value description (.*)
Value mtu (\d+)

Start
  ^${name} is up
  ^\s+Hardware is ${type} -> Continue
  ^\s+Description: ${description}
  ^\s+MTU ${mtu} bytes, -> Record
```

Example 2: Playbook

- The `textfsm_parser` module has two parameters:
  - `file` : Define the path to YAML file
  - `content` : Define the path to a raw file or the variable that contains the raw output

```
---
- name: USING PARSER TEMPLATES
  hosts: csr1
  connection: network_cli
  gather_facts: no
  roles:
    - ansible-network.network-engine
  tasks:

    - name: SHOW INTERFACES
      ios_command:
        commands: show interfaces
      register: interface_data

    - name: TEXTFSM PARSER
      textfsm_parser:
        file: "./parsers/{{ ansible_network_os }}/show_interfaces.template"
        content: "{{ interface_data['stdout'][0] }}"
        name: interface_facts
      register: interface

    - name: PARSED DATA
      debug:
        var: interface
```

```
TASK [PARSED DATA] ***********************************************************
ok: [csr1] => {
    "interface": {
        "ansible_facts": {
            "interface_facts": [
                {
                    "description": "",
                    "mtu": "1500",
                    "name": "GigabitEthernet1",
                    "type": "CSR vNIC"
                },
                {
                    "description": "CONNECTS_CSR3",
                    "mtu": "1500",
                    "name": "GigabitEthernet2",
                    "type": "CSR vNIC"
                },
                {
                    "description": "CONNECTS_CSR2",
                    "mtu": "1500",
                    "name": "GigabitEthernet4",
                    "type": "CSR vNIC"
                },
                {
                    "description": "",
                    "mtu": "1514",
                    "name": "Loopback0",
                    "type": "Loopback"
                },
                output omitted...
                ]
```

NTC Parser Example

Example 3: Text Input

- show interfaces (Cisco IOS)

```
GigabitEthernet1 is up, line protocol is up
  Hardware is CSR vNIC, address is 2cc2.6031.1341 (bia 2cc2.6031.1341)
  Internet address is 10.0.0.51/24
  MTU 1500 bytes, BW 1000000 Kbit/sec, DLY 10 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
  Full Duplex, 1000Mbps, link type is auto, media type is RJ45
  output flow-control is unsupported, input flow-control is unsupported
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:00:15, output 00:00:04, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/375/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/40 (size/max)
  5 minute input rate 2000 bits/sec, 2 packets/sec
  5 minute output rate 2000 bits/sec, 2 packets/sec
     8063 packets input, 732260 bytes, 0 no buffer
     Received 0 broadcasts (0 IP multicasts)
     0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
     0 watchdog, 0 multicast, 0 pause input
     9171 packets output, 723414 bytes, 0 underruns
     0 output errors, 0 collisions, 0 interface resets
     254 unknown protocol drops
     0 babbles, 0 late collision, 0 deferred
     0 lost carrier, 0 no carrier, 0 pause output
     0 output buffer failures, 0 output buffers swapped out
GigabitEthernet2 is up, line protocol is up
  Hardware is CSR vNIC, address is 2cc2.6069.e08c (bia 2cc2.6069.e08c)
  Description: CONNECTS_CSR3
  Internet address is 10.254.13.1/24
  MTU 1500 bytes, BW 1000000 Kbit/sec, DLY 10 usec,
```

```yaml
---
  - name: USING PARSER TEMPLATES
    hosts: csr1
    connection: local
    gather_facts: no

    vars:
      template_path: "./parsers/{{ ansible_network_os }}/"
      show_interfaces: "{{ template_path }}show_interfaces.templ

    tasks:

      - name: Using ntc_show_command and txt_fsm
        ntc_show_command:
          platform: "cisco_ios"
          command: 'show interfaces'
          provider: "{{ connection_details }}"
          template_dir: "{{ template_path }}"
```

Example 3: Template File

- show interfaces (Cisco IOS)
- Order is important
- Filename: `show_interfaces.template`

```
Value Required name (\S+)
Value type ([\w ]+)
Value description (.*)
Value mtu (\d+)

Start
  ^${name} is up
  ^\s+Hardware is ${type} -> Continue
  ^\s+Description: ${description}
  ^\s+MTU ${mtu} bytes, -> Record
```

Note: We are using the same TextFSM template used for Network Engine

Example 3: Playbook

- The `ntc_show_command` module sends the operational command and also parses the data.

```yaml
---

  - name: USING PARSER TEMPLATES
    hosts: csr1
    connection: local
    gather_facts: no

    vars:
      template_path: "./parsers/{{ ansible_network_os }}/"
      show_interfaces: "{{ template_path }}show_interfaces.template"


    tasks:

      - name: Using ntc_show_command and txt_fsm
        ntc_show_command:
          platform: "cisco_ios"
          command: 'show interfaces'
          provider: "{{ connection_details }}"
          template_dir: "{{ template_path }}"
        register: interface


      - name: PARSED DATA
        debug:
          var: interface
```

```
TASK [PARSED DATA] ***********************************************************
ok: [csr1] => {
    "interface": {
        "ansible_facts": {
            "interface_facts": [
                {
                    "description": "",
                    "mtu": "1500",
                    "name": "GigabitEthernet1",
                    "type": "CSR vNIC"
                },
                {
                    "description": "CONNECTS_CSR3",
                    "mtu": "1500",
                    "name": "GigabitEthernet2",
                    "type": "CSR vNIC"
                },
                {
                    "description": "CONNECTS_CSR2",
                    "mtu": "1500",
                    "name": "GigabitEthernet4",
                    "type": "CSR vNIC"
                },
                {
                    "description": "",
                    "mtu": "1514",
                    "name": "Loopback0",
                    "type": "Loopback"
                },
                output omitted...
                ]
```

Lab Time

- Lab 25 - Using Parser Templates

Extra Bonus

Device discovery, Dynamic Groups and Dynamic Inventory

Creating Dynamic Groups

You are able to create dynamic groups within Ansible using the `group_by` module. Using the data collected by the `snmp_device_version` module we are able to dynamically group devices by vendor, os, or version.

In this example we are grouping the devices by vendor. This will create a group named `vendor_` followed by the vendor name. This group can be used in subsequent plays within the playbook. Dont forget about `groups` builtin variable that will show a dictionary of keys that are all group names defined in the inventory file and values are list of host names that are members of the group.

```
---

  - name: DISCOVER VENDOR
    hosts: iosxe,nxos,vmx
    connection: local
    gather_facts: no

    tasks:

      - name: QUERY DEVICE VIA SNMP
        snmp_device_version:
          community: networktocode
          version: 2c
          host: "{{ inventory_hostname }}"
        tags: snmp

      - group_by:
          key: vendor_{{ ansible_device_vendor }}

      - debug:
          var: groups
```

Dynamic Inventory

Ansible for Network Automation

Inventory

- Ansible Inventory files are static
- Managing large YAML files doesn't scale
- Manage lots of YAML files doesn't scale
- Users usually already have a CMDB, NMS, etc.
- What if you have a dynamic environment?
  - AWS, Rackspace, VMs
- **Enter Dynamic Inventory**

Executing Playbooks

- You currently use the `-i` flag to specify the inventory file
- You can also specify a script that is used to generate an inventory
  - Needs to return JSON k/v pairs
- `ansible-playbook -i dynamic_script.py site.yml`
- Script needs to be an executable (x)

- Sample Inventory generated from a script
- Groups, group vars, hosts, host vars (not shown)

```json
{
    "hp": {
        "hosts": [
            "hp1.ntc.com",
            "hp2.ntc.com",
            "hp3.ntc.com",
            "hp4.ntc.com"
        ],
        "vars": {
            "platform": "comware7"
        }
    },
    "cisco": {
        "hosts": [
            "n9k1.ntc.com",
            "n9k2.ntc.com"
        ],
        "vars": {
            "platform": "nexus"
        }
    },
    "juniper": [
        "jnprfw.ntc.com"
    ],
    "arista": [
        "arista1.ntc.com",
        "arista2.ntc.com"
    ]
}
```

Multiple Sources

- If the parameter used with the `-i` flag is a directory, multiple sources can be used
  - inventory file with scripts
  - multiple scripts

>>> network.toCode()

Example Script

```
#!/usr/bin/env python

import requests

import json

requests.packages.urllib3.disable_warnings()

def main():

    url = 'https://2z3oa80l2c.execute-api.us-east-1.amazonaws.com/prod/switch'

    inventory = requests.get(url, verify=False)

    return inventory.text

if __name__ == "__main__":

    rsp = main()

    print rsp
```

Executing a Playbook

```
---

  - name: test playbook for dynamic inventory
    hosts: all
    connection: local
    gather_facts: no



    tasks:

      - debug:
          var: inventory_hostname
```

Viewing the Results

- JSON data generated from script

```
{
    "cisco": {
        "hosts": [
            "n9k1.ntc.com",
            "n9k2.ntc.com"
        ],
        "vars": {
            "platform": "nexus"
        }
    },
    "arista": [
        "arista1.ntc.com",
        "arista2.ntc.com"
    ],
    "apic": [
        "aci.ntc.com"
    ]
}
```

- Playbook output

```
ansible-playbook -i dynamo.py site.yml

PLAY [test playbook for dynamic inventory] ************************************

TASK: [debug var=inventory_hostname] ******************************************
ok: [n9k2.ntc.com] => {
    "var": {
        "inventory_hostname": "n9k2.ntc.com"
    }
}
ok: [n9k1.ntc.com] => {
    "var": {
        "inventory_hostname": "n9k1.ntc.com"
    }
}
ok: [arista1.ntc.com] => {
    "var": {
        "inventory_hostname": "arista1.ntc.com"
    }
}
ok: [aci.ntc.com] => {
    "var": {
        "inventory_hostname": "aci.ntc.com"
    }
}
ok: [arista2.ntc.com] => {
    "var": {
        "inventory_hostname": "arista2.ntc.com"
    }
}

PLAY RECAP ************************************
aci.ntc.com                : ok=1    changed=0    unre
arista1.ntc.com            : ok=1    changed=0    unreachable=0    failed=0
```

Summary

- YAML files do the trick *most* of the time, especially for testing
- Custom dev work required for dynamic network inventories, i.e. HPOV, Solar Winds, etc.
- As with modules, can be done in any language
  - Just need to return JSON in the proper format

Lab Time

- Lab 26 - Dynamic Device Discovery with Dynamic Groups
- Lab 27 - Using a Dynamic Inventory Script
    - You will execute a playbook that uses a pre-created inventory script that queries a public REST API

The End

EXTRA - BONUS

Creating Custom Ansible Modules

Ansible for Network Automation

Ansible Modules

- Over 400 Core modules
  - Linux
  - Windows
  - EC2
  - Rackspace
  - VMware

- Extras
- Galaxy Roles...
- You may need to:
  - Write your own
  - Help fix others

Workflow

- Especially for those just getting started, build out your script in native Python first
- Write a main() function
  - This should call other functions needed
  - Conditional (idempotent) logic should exist within main() for this workflow
- if **name** == "**main**" should just call main()
- Use variables to simulate all user inputs, i.e. Ansible parameters
- Finishing by returning a Python dictionary that'll include key/value pairs on what you want to see/use within a playbook, etc.

Existing Python Script

```
#!/usr/bin/env python

import socket
import json

from pycsco.nxos.device import Device
from pycsco.nxos.utils import nxapi_lib


def main():

    username = 'cisco'
    password = 'cisco'
    protocol = 'http'
    node = 'n9k1'
    host = socket.gethostbyname(node)

    device = Device(ip=host, username=username, password=password,
                    protocol=protocol)

    neighbors = nxapi_lib.get_neighbors(device)

    print neighbors   # returns a list

    neighs = dict(neighbors=neighbors)

if __name__ == "__main__":
    main()
```

From Script to Module

```python
#!/usr/bin/env python

import socket
import json

from pycsco.nxos.device import Device
from pycsco.nxos.utils import nxapi_lib


def main():

    username = 'cisco'
    password = 'cisco'
    protocol = 'http'
    node = 'n9k1'
    host = socket.gethostbyname(node)

    device = Device(ip=host, username=username, password=password,
                    protocol=protocol)

    neighbors = nxapi_lib.get_neighbors(device)

    print neighbors    # returns a list

    neighs = dict(neighbors=neighbors)

if __name__ == "__main__":
    main()
```

```python
def main():

    module = AnsibleModule(
        argument_spec=dict(
            type=dict(choices=['cdp', 'lldp'], default='cdp'),
            protocol=dict(choices=['http', 'https'], default='http'),
            host=dict(required=True),
            username=dict(type='str', required=True),
            password=dict(type='str', required=True),
        ),
        required_together=(
            ['host', 'password', 'username'],
        ),
        supports_check_mode=False

    username = module.params['username']
    password = module.params['password']
    protocol = module.params['protocol']
    host = socket.gethostbyname(module.params['host'])

    neigh_type = module.params['type'].lower()

    device = Device(ip=host, username=username, password=password,
                    protocol=protocol)

    neighbors = nxapi_lib.get_neighbors(device, neigh_type)
    hostname = nxapi_lib.get_hostname(device)

    results = {}
    results['resource'] = neighbors
    results['local_hostname'] = hostname

    module.exit_json(**results)
from ansible.module_utils.basic import *
```

AnsibleModule Initalization

- **AnsibleModule**
- **argument_spec**
- **choices**
- **default**
- **required**
- **required_together**
- required_one_of
- mutually_exclusive
- **supports_check_mode**
- module.exit_json(key=value, vlan=vlan_id)
- module.fail_json(msg='foo', key=value)

```
def main():

    module = AnsibleModule(
        argument_spec=dict(
            type=dict(choices=['cdp', 'lldp'], default='cdp'),
            protocol=dict(choices=['http', 'https'], default='ht
            host=dict(required=True),
            username=dict(type='str', required=True),
            password=dict(type='str', required=True),
        ),
        required_together=(
            ['host', 'password', 'username'],
        ),
        supports_check_mode=False

        # extract values from module.params
        username = module.params['username']

        # application code / from your script
        # call you rmethods, use your libs,etc.

        results = dict()
        # return dictionary
        module.exit_json(**results)
```

Using the Module

- Place the module `get_cisco_neighbors.py` in the `library` directory

```
.
├── configs
│   └── leaf1
│       ├── leaf1.conf
│       └── partials
│           ├── 01_snmp.j2.conf
│           ├── 02_vlans.j2.conf
│           └── 03_interfaces.j2.conf
├── group_vars
│   └── all.yml
├── inventory
├── playbook.yml
├── library
│   └── get_cisco_neighbors.py
├── templates
│   ├── 01_snmp.j2
│   ├── 02_vlans.j2
│   └── 03_interfaces.j2
```

```
- name: GET NEIGHBORS
  get_cisco_neighbors: host={{ inventory_hostname }} username=cisco password=cisco
```

Multi-Vendor Modules

- Not many exist, expecially as it pertains to configuration automation
- Build on previous example on gathering neighbor information from Arista

What's needed?

Multi-Vendor Modules

- Not many exist, expecially as it pertains to configuration automation
- Build on previous example on gathering neighbor information from Arista

What's needed?

- Device/Vendor parameter?
- Standardize keys (k/v)
    - neighbor
    - neighbor_interface
    - local_interface

Idempotent Modules

- Check current state first
  - Perform a *get* or *show* operation
  - We'll call this existing state
- Parameters being sent in from from playbook
  - We'll call this proposed state
- Perform a diff / delta on proposed vs. existing
- Idempotent logic:
  - If there isn't a delta, exit `module.exit_json()` - run N times, make one change
  - If there is a delta, make *only* the needed changes
- *Your* own logic
  - Configuring a VLAN that doesn't exist globally, *fail_out* OR something already configured you want to know about if a change is being made, `module.fail_json`

Example Code (VLAN module)

- existing = dictionary
- proposed = dictionary
- Convert a dictionary to a set and use the difference method to calculate a change set, or delta
- Perform logic

```
# snippet
cmds = None
if state == 'present':
    delta = dict(set(proposed.iteritems()).difference(
        existing.iteritems()))
    if delta:
        cmds = vlan.build(**delta)
elif state == 'absent':
    if existing:
        vlan.remove()
end_state = existing
```

```
if module.check_mode:
    module.exit_json(changed=True, commands=cmds)
else:
    device.push_commands(cmds)
    end_state = vlan.get_config()
    changed = True

results = {}
results['proposed'] = proposed
results['existing'] = existing
results['state'] = state
results['commands'] = commands
results['changed'] = changed
results['end_state'] = end_state

module.exit_json(**results)
```

Summary

- When getting started, write raw Python code first
- Use `module.exit_json(k=v, a=b)` for troubleshooting
- Return helpful data for troubleshooting
- Return helpful data that can be used as inputs for other modules
- Keep the module code small - use libs, functions, etc. to minimize change to the actual module

EXTRA - BONUS

Using the Ansible Vault Feature

Ansible for Network Automation

Ansible Vault

The Ansible Vault functionality allows the user:

- To store sensitive data as a one-way hash on the filesystem
- Use unencrypted data on the fly during playbook execution

Typically used to store username and passwords on the control machine.

```
ntc@jump-host:all$ ansible-vault create vaultfile.yml
New Vault password:
Confirm New Vault password:
```

Ansible Vault

The unencrypted file itself, is standard `yaml` that contains structured YAML variables

```yaml
---
user: ntc
pass: ntc123
```

The encrypted version of above data:

```
ntc@jump-host:all$ ls
vaultfile.yml
ntc@jump-host:all$ cat vaultfile.yml
$ANSIBLE_VAULT;1.1;AES256
38353863306139626235623263313439653437646261393562323036356531336432323736646534
3161333737316430396431313931633863646535303432660a353461636464303238353765343162
31346366353766663063303636386265326665643331326632613536363831346364663065316462
6365646337363838650a326563386465383662643733633930323264333065633034363338643735
33323566656238633436623732623062313562386465666664333961386161313034
```

Ansible Vault

- Use the `--ask-vault-pass` flag while invoking the playbook.
- This will prompt you to enter the password used to encrypt the vault file.

```
ntc@jump-host:ansible$ ansible-playbook -i inventory use_vault.yml --ask-vault-pass
Vault password:

PLAY [USE ENCRYPTED LOGIN]
*****************************************************************************************************

TASK [COLLECT THE SERIAL NUMBER]
****************************************************************************************************
ok: [csr1]
ok: [csr2]
ok: [csr3]

....
```

Ansible Vault - Summary

- Use to encrypt sensitive data on disk
- Encrypt using the `ansible-vault` command
- Invoke a playbook using flag `--ask-vault-pass`

EXTRA - BONUS

Cisco Nexus Ansible Modules

Ansible for Network Automation

Cisco Nexus Modules

All of these are in Ansible Core.

```
nxos_aaa_server_host.py      nxos_hsrp.py            nxos_pim.py             nxos_udld_interface.py
nxos_aaa_server.py           nxos_igmp_interface.py  nxos_pim_rp_address.py  nxos_udld.py
nxos_acl_interface.py        nxos_igmp.py            nxos_ping.py            nxos_vlan.py
nxos_acl.py                  nxos_igmp_snooping.py   nxos_portchannel.py     nxos_vpc_interface.py
nxos_bgp_af.py               nxos_install_os.py      nxos_reboot.py          nxos_vpc.py
nxos_bgp_neighbor_af.py      nxos_interface_ospf.py  nxos_rollback.py        nxos_vrf_af.py
nxos_bgp_neighbor.py         nxos_interface.py       nxos_smu.py             nxos_vrf_interface.py
nxos_bgp.py                  nxos_ip_interface.py    nxos_snapshot.py        nxos_vrf.py
nxos_command.py              nxos_mtu.py             nxos_snmp_community.py  nxos_vrrp.py
nxos_config.py               nxos_ntp_auth.py        nxos_snmp_contact.py    nxos_vtp_domain.py
nxos_evpn_global.py          nxos_ntp_options.py     nxos_snmp_host.py       nxos_vtp_password.py
nxos_evpn_vni.py             nxos_ntp.py             nxos_snmp_location.py   nxos_vtp_version.py
nxos_facts.py                nxos_nxapi.py           nxos_snmp_traps.py      nxos_vxlan_vtep.py
nxos_feature.py              nxos_ospf.py            nxos_snmp_user.py       nxos_vxlan_vtep_vni.py
nxos_file_copy.py            nxos_ospf_vrf.py        nxos_static_route.py    nxos_gir_profile_management.py
nxos_overlay_global.py       nxos_switchport.py      nxos_gir.py             nxos_pim_interface.py
```

nxos_facts

- Gathers facts from Nexus devices
  - fan_info
  - hostname
  - uptime
  - interfaces
  - last reboot reason
  - operating system
  - vlan_list

```
tasks:
  - nxos_facts:
```


>>> network.toCode()

nxos_vlan

- Manages VLAN resources on a Nexus switch
- Parameters:
  - admin_state
  - name
  - vlan_id
  - vlan_state

```
# ensure vlan 110 exists with a name configured
- nxos_vlan:
    vlan_id: 110
    name: DB_VLAN
```

nxos_feature

- Manages features on a Nexus switch
- Parameters:
  - feature

```
# ensure eigrp is disabled
- nxos_feature:
    feature: eigrp
    state: disabled
```

```
# ensure lacp is enabled
- nxos_feature:
    feature: lacp
    state: enabled
```

nxos_file_copy

- Copies a local file to bootflash (by default) of NXOS device using SCP
- Parameters
  - dest_file
  - source_file

```
# copy latest NX-OS to NXOS switch
- nxos_file_copy:
    source_file: /home/cisco/Downloads/nxos.7.0.3.I2.1.bin
```

nxos_portchannel

- Manage port-channel interfaces on Nexus devices
- Parameters:
  - group
  - min_links
  - members (list)
  - mode
- Complex args as a parameter must use YAML format

```
# ensure port-channel 100 exists
- nxos_portchannel:
    group: 100
    min_links: 2
    members:
      - Ethernet1/28
      - Ethernet1/29
```

nxos_vpc

- Manages global VPC configuration
- Parameters:
    - domain
    - role_priority
    - system_priority
    - pkl_src
    - pkl_dest
    - pkl_vrf
    - peer_gw
    - auto_recovery
    - delay_restore

```yaml
# ensure port-channel 100 exists
- nxos_vpc:
    domain: 100
    role_priority: 1000
    system_priority: 2000
    pkl_dest: 192.168.100.4
    pkl_src: 10.1.100.20
    peer_gw: true
    auto_recovery: true
```

nxos_vpc_interface

- Manages interface VPC configuration
- Parameters:
  - portchannel
  - vpc
  - peer_link

```
# ensure port-channel 100 exists
- nxos_vpc_interface:
    portchannel: 10
    vpc: 100
```

EXTRA - BONUS

Juniper Modules

Ansible for Network Automation

Juniper Modules

- **junos_commit** – Commit candidate configuration on device.
- **junos_get_config** – Retrieve configuration of device.
- **junos_get_facts** – Retrieve device-specific information from the host.
- **junos_install_config** – Modify the configuration of a device running Junos OS.
- **junos_install_os** – Install a Junos OS software package.
- **junos_rollback** – Rollback configuration of device.
- **junos_shutdown** – Shut down or reboot a device running Junos OS.
- **junos_srx_cluster** – Enable/Disable cluster mode for SRX devices
- **junos_zeroize** – Remove all configuration information and reset all key values on a device.
- **junos_cli** - Execute CLI command on device and save file locally
- **junos_rpc** - Execute Junos RPC on device and save file locally
- **junos_get_table** - Retrieve data from device using Junos Tables/Views

junos_commit

- Commit candidate configuration on device.

```
- name: COMMIT JUNOS CONFIGURATION
  junos_commit:
   host={{ inventory_hostname }}
   logfile=changes.log
   comment="Commit existing candidate"
```

junos_rollback

- Rollback configuration of device.
- Parameters:
  - confirm - confirmation in minutes to the commit of the configuration
  - diffs_file - location where diffs are stored

```
- junos_rollback:
   host: "{{ inventory_hostname }}"
   logfile=rollback.log
   diffs_file=rollback.diff
   rollback=1
   comment="Rolled back by Ansible"
   confirm=5
```

junos_get_config

- Retrieve configuration of device.

```
- name: GET CONFIG
  junos_get_config: user={{ ansible_user }} passwd={{ ansible_ssh_pass }} host={{ inventory_hostname }} filter="interfaces" dest=
```

```
- name: GET CONFIG
  junos_get_config: user={{ ansible_user }} passwd={{ ansible_ssh_pass }} host={{ inventory_hostname }} filter="interfaces/protoc
```

junos_install_config

- Modify the configuration of a device running Junos OS.

```
# load merge a change to the Junos OS configuration using NETCONF
- junos_install_config:
    host={{ inventory_hostname }}
    file=banner.conf

# load overwrite a new Junos OS configuration using the CONSOLE port
- junos_install_config:
    host={{ inventory_hostname }}
    console="--telnet={{TERMSERV}},{{TERMSERV_PORT}}"
    file=default_new_switch.conf
    overwrite=yes

# load replace a change to the Junos OS configuration using NETCONF
- junos_install_config:
    host={{ inventory_hostname }}
    file=snmp.conf
    replace=yes
```

junos_install_os

- Install a Junos OS software package.
- Parameters:
  - reboot - booelan; reboots after the installation completes.
  - reboot_pause - Amount of time in seconds to wait after the reboot is issued. default=10
  - version - Junos OS version string as it would be reported by the show version command
  - package - Absolute path on the local server to the Junos OS software package

```
- junos_install_os:
    host={{ inventory_hostname }}
    version=12.1X46-D10.2
    package=/usr/local/junos/images/junos-vsrx-12.1X46-D10.2-domestic.tgz
```

Summary

- Always test the modules
- Vendors and community are always adding new features
- Understand module idempotency
- Understand if the module supports check mode

EXTRA - BONUS

ntc-ansible Modules

Ansible for Network Automation

ntc-ansible Modules

- These are modules that are primarily built off of `pyntc`.
- NTC Modules (covered here)
  - ntc_show_command - send arbitrary show commands and get back structured data
  - ntc_config_command - send arbitrary configuration commands
  - ntc_facts - gather facts
  - ntc_save_config - Save & backup configs
  - ntc_reboot - Reboot devices
  - ntc_file_copy - Copy files to devices
  - ntc_install_os - Upgrade devices
  - ntc_rollback - creates local backup and restores upon failure in subsequent task

Supported Parameters

Common module parameters between NTC Modules:

- `platform` , `host` , `username` , `password`

IMPORTANT:

- `platform` values for **ntc_show_command** and **ntc_config_command**:
  - `{{ ntc_vendor }}_{{ ansible_network_os }}`
  - matches what Netmiko supports: **cisco_ios**, **cisco_nxos**, **arista_eos** (optionally _ssh can be added too, e.g. **cisco_ios_ssh**)
  - Anything Netmiko supports for `device_type` is supported here (all SSH)

- `platform` values for all other modules must be one of the following:
  - **cisco_ios_ssh**
  - **cisco_nxos_nxapi**
  - **arista_eos_eapi**
  - **juniper_junos_netconf**

ntc_show_command

- Multi-vendor Ansible module to streamline converting raw text into JSON key/value pairs
- Leverages TextFSM
- netmiko is used for transport to enable support for all device

```
tasks:

  - name: GET DATA
    ntc_show_command:
      connection=ssh          # can be offline; if offline uses file parameter for parsing
      platform=cisco_ios
      command='show ip interface brief'
      provider={{ ntc_provider }}
      template_dir: "/etc/ntc/ansible/library/ntc-ansible/ntc-templates/templates"
      use_templates: yes
```


ntc_show_command

- JSON data now available to re-use
- Use as inputs to other modules or in templates (docs)

```
$ ansible-playbook -i hosts test-playbook.yml -v

PLAY [GET STRUCTURED DATA BACK FROM CLI DEVICES] ******************************

TASK: [GET DATA] **************************************************************
ok: [csr1] => {"changed": false, "response": [{"intf": "GigabitEthernet1", "ipaddr": "10.0.0.50", "proto": "up", "status":
"up"}, {"intf": "GigabitEthernet2", "ipaddr": "10.254.13.1", "proto": "up", "status": "up"}, {"intf": "GigabitEthernet3",
"ipaddr": "unassigned", "proto": "down", "status": "administratively down"}, {"intf": "GigabitEthernet4", "ipaddr":
"10.254.12.1", "proto": "up", "status": "up"}, {"intf": "Loopback100", "ipaddr": "1.1.1.1", "proto": "up", "status": "up"}]}

PLAY RECAP ********************************************************************
csr1                       : ok=1    changed=0    unreachable=0    failed=0
```

```
csr1#show ip int brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       10.0.0.50       YES NVRAM  up                    up
GigabitEthernet2       10.254.13.1     YES NVRAM  up                    up
GigabitEthernet3       unassigned      YES NVRAM  administratively down down
GigabitEthernet4       10.254.12.1     YES NVRAM  up                    up
Loopback100            1.1.1.1         YES manual up                    up
```

ntc_config_command

- Send config command from a list or from a file
- This is simply a *wrapper* for Netmiko
- If Netmiko supports it, ntc_config_command does
- Not idempotent (use core _config modules if they exist for your OS)

```
# write from a command list
- ntc_config_command:
    connection: ssh
    platform: cisco_ios
    commands:
      - vlan 10
      - name vlan_10
      - end
    provider: "{{ ntc_provider }}"
```

```
# write config from file
- ntc_config_command:
    connection: ssh
    platform: cisco_ios
    commands_file: "dynamically_created_config.txt"
    provider: "{{ ntc_provider }}"
```

ntc_get_facts

Facts returned include:

- uptime (string)
- uptime (seconds)
- model
- vendor
- os_version
- serial_number
- hostname
- fqdn
- vlans
- interfaces

```
- ntc_get_facts:
    platform: cisco_nxos_nxapi
    provider: "{{ ntc_provider }}"
    transport: http

- ntc_get_facts:
    platform: cisco_ios
    provider: "{{ ntc_provider }}"
```

ntc_save_config

- Save the running configuration as the startup configuration or to a file on the network device.
    - Performs a commit on Juniper devices / copy run start on others
- **Optionally, save the running configuration as a file to the Ansible control host.**

```
# does a copy run start
- ntc_save_config:
    platform: cisco_nxos_nxapi
    provider: "{{ ntc_provider }}"
    local_file: ./backups/{{ inventory_hostname }}.cfg
```

ntc_reboot

Reboot a network device, optionally on a timer.

```yaml
# reboots the device in 5 minutes
- ntc_reboot:
    platform: cisco_ios
    confirm: true
    timer: 5
    provider: "{{ ntc_provider }}"
```

```yaml
# reboot immediately
- ntc_reboot:
    platform: cisco_ios
    confirm: true
    provider: "{{ ntc_provider }}"
```

ntc_file_copy

- Copy local files to remote network devices using SCP
- Supported platforms
  - Cisco Nexus switches with NX-API
  - Arista switches with eAPI.
  - Cisco IOS switches or routers

```
- ntc_file_copy:
    platform: cisco_nxos_nxapi
    local_file: ./images/{{ ansible_network_os }}/{{ os_version }}
    provider: "{{ ntc_provider }}"
    transport: http

- ntc_file_copy:
    platform=cisco_ios
    local_file=./images/c2800nm-adventerprisek9_ivs_li-mz.151-3.T4.bin
    provider: "{{ ntc_provider }}"
```

ntc_install_os

- Set boot commands and/or upgrade devices (not supported on Juniper)

```
- ntc_install_os:
    platform: cisco_nxos_nxapi
    provider: "{{ ntc_provider }}"
    transport: http
    system_image_file: n9000-dk9.6.1.2.I3.1.bin

- ntc_install_os:
    platform: cisco_nxos_nxapi
    provider: "{{ ntc_provider }}"
    system_image_file: n3000-uk9.6.0.2.U6.5.bin
    kickstart_image_file: n3000-uk9-kickstart.6.0.2.U6.5.bin

- ntc_install_os:
    platform: cisco_ios_ssh
    provider: "{{ ntc_provider }}"
    system_image_file: c2800nm-adventerprisek9_ivs_li-mz.151-3.T4.bin
```

Upgrade Workflow

```
# BACKUP CONFIG FILE

- ntc_save_config:
    platform: cisco_ios
    provider: "{{ ntc_provider }}"
    local_file: "{{ inventory_hostname }}.cfg"


# COPY IMAGE TO DEVICE

- ntc_file_copy:
    platform: cisco_ios
    local_file: "{{ image_file }}"
    provider: "{{ ntc_provider }}"



.
```

```
# UPGRADE AND/OR JUST SET BOOT FILE

- ntc_install_os:
    platform: cisco_nxos_nxapi
    provider: "{{ ntc_provider }}"
    transport: http
    system_image_file: n9000-dk9.6.1.2.I3.1.bin

# OPTIONAL REBOOT: not needed for nxos

- ntc_reboot:
    platform: cisco_ios
    confirm: true
    timer: 5
    provider: "{{ ntc_provider }}"
```


>>> network.toCode()

ntc_rollback

- Create checkpoint files
- Rollback to previously created checkpoint file

```
- ntc_rollback:
    checkpoint_file: backup.cfg
    platform: cisco_nxos_nxapi
    provider: "{{ ntc_provider }}"

- ntc_rollback:
    rollback_to: backup.cfg
    platform: cisco_nxos_nxapi
    provider: "{{ ntc_provider }}"
```

ntc_rollback

- Prior to Ansible 2.0, you would use ntc_rollback for a single task (as also done in an upcoming lab)
- Ansible 2.0 introduced blocks and error handling (`block` , `rescue` , `always`)

```
tasks:
  - block:
      - name: CREATE LAST KNOWN GOOD (CHECKPOINT)
        ntc_rollback:
          checkpoint_file=last_known_good.conf
          platform: "{{ ntc_vendor }}_{{ ansible_network_os }}_{{ ntc_api }}"
          provider: "{{ ntc_provider }}"
      - nxos_vlan:
          vlan_id: 500
          provider: "{{ provider }}"
      - nxos_vlan:
          vlan_id: 5000
          provider: "{{ provider }}"
    rescue:
      - name: ROLLBACK TO CHECKPOINT FILE UPON ERROR
        ntc_rollback:
          rollback_to: last_known_good.conf
          platform: "{{ ntc_vendor }}_{{ ansible_network_os }}_{{ ntc_api }}"
          provider: "{{ ntc_provider }}"
```