>>> network.toCode()

Network Programming & Automation

Module 1: Introduction to Using Ansible for Network Automation

Course Outline

Introduction

- Course Overview
- What's Possible with Ansible
- Introduction to Ansible Vocabulary

Ansible Variables and Password Security

- Quick Look at YAML & JSON
- Understanding Ansible Variables
- Ansible Vault for Password Security
- Debug Module to Display Ansible Variable Content

Showing Network Device Configurations

- Show Commands on Network Devices
- How do you see the data being gathered?
- Jinja2 Templates
- Loops and Registers
- Parsing Unstructured Data

Course Outline Contd.

Diving Deeper into Command and Config Modules

- *_config Modules
- diff_against Parameter
- Declarative Configuration
- Data Collection and Reporting
- Core *_facts Modules

Reusable and Repeatable Code

- Ansible Collections
- Ansible Roles
- Ansible Dynamic Inventories
- Third Party Modules

Module
Table of Contents

Course Objectives

- Learn Ansible vocabulary and how to use it
- See Ansible used to accomplish challenges
- Apply Ansible yourself to those challenges
- Look for opportunities to apply within your own organization
- Consider how to apply Ansible to accomplish your challenges

Lecture 1: Ansible Architecture and Playbook Examples

Topic 1 - Ansible History and Architecture

Topic 2 - Ansible Terminology

Topic 3 - Creating An Ansible Project From Scratch

Labs 1-2

Topic 1: Ansible History and Architecture

What is Ansible and How it Works

>>> 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 with a great base of 4,500+ modules with which to build automation
- Built its home for application deployments in cloud environments
- Rapidly gaining traction for network automation

>>> Diving into Ansible

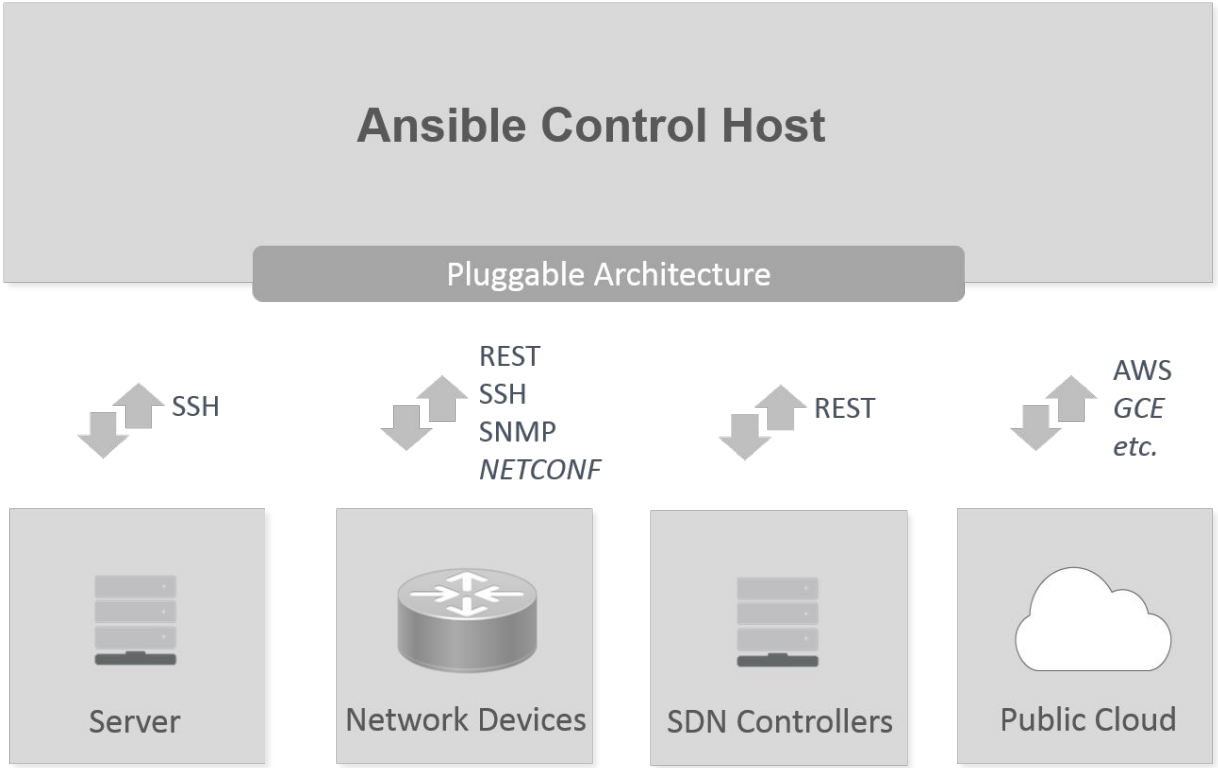
- Written in Python
- Extended in any language (not common for open source modules)
- Native integration with Jinja2 templates
- Automation instructions are defined in YAML
- Agentless

(We cover Jinja2 and YAML in this course.)

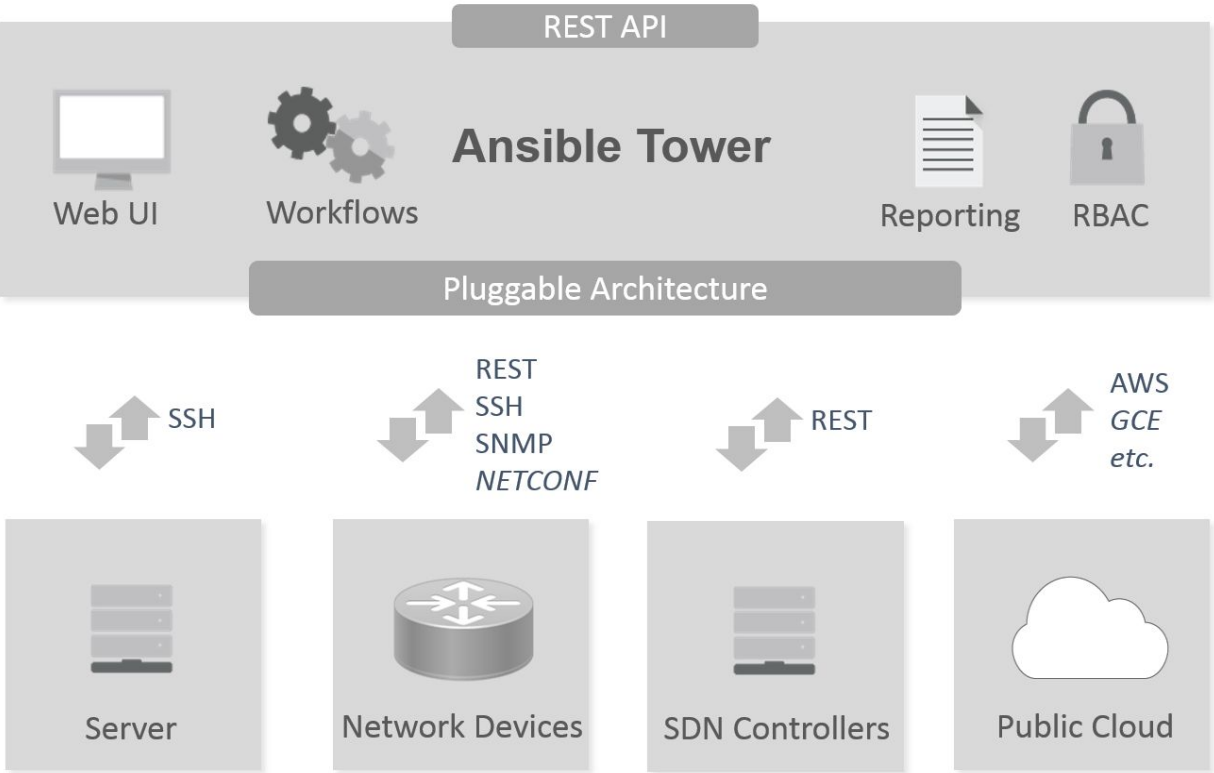
>>> How Ansible Works

- 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



>>> Updating SNMP Community strings

```yaml
---
- name: "DEPLOY SNMP COMMUNITY STRINGS ON IOS DEVICES"
  hosts: "ios"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false

  tasks:

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

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

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

>>> Upgrade Cisco NX-OS Devices

```yaml
---
- name: "UPGRADE NEXUS SWITCHES"
  hosts: "nxos"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false

  tasks:

    - name: "ENSURE SCP SERVER IS ENABLED"
      cisco.nxos.nxos_feature:
        feature: "scp-server"
        state: "enabled"

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

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

>>> Takeaways

- All 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

Topic 2: Ansible Terminology

Ansible Terminology and Creating a Static Inventory

>>> Ansible Terminology

- When learning a new language, including Network Automation and Ansible, you first need to learn some basic terminology.
- Inventory
- Playbooks
- Plays
- Tasks
- Collections
- Modules
- Parameters
- Variables

```yaml
---

- name: "BASIC TESTING"
  hosts: "dc1"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false

  tasks:

    - name: "ENSURE VLAN 10 EXISTS"
      cisco.nxos.nxos_vlan:
        vlan_id: 10
        name: "web_vlan"

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

>>> Ansible Terminology

- Map Ansible terminology to an already known language:
- Inventory - NOUNS
- Playbooks - CHAPTERS
- Plays - PARAGRAPHS
- Tasks - SENTENCES
- Collections
- Modules - VERBS
- Parameters
- Variables

```yaml
---

- name: "BASIC TESTING"
  hosts: "dc1"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false

  tasks:

    - name: "ENSURE VLAN 10 EXISTS"
      cisco.nxos.nxos_vlan:
        vlan_id: 10
        name: "web_vlan"

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

Automating with Ansible

Two files are required to get started:

- **Inventory file**
- Playbook

Topic 3: Creating An Ansible Project From Scratch

Reading and setting up a simple device inventory with host and group variables

Writing an Ansible playbook and understanding its components

Understanding how Ansible connects to network devices

Executing a playbook and managing credentials

>>> 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
```

hostname	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

```
---
# 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: "ansible.netcommon.network_cli"
  gather_facts: false

  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: "ansible.netcommon.network_cli"
  gather_facts: false

  tasks:

    # list of tasks
```

```yaml
- name: "PLAY 2 - DEPLOY VLANS ON SWITCHES"
  hosts: "switches"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false

  tasks:
```

>>> Connection Types

- There's 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 third party modules since they may have their own connection mechanisms (APIs, etc.)
- Run `ansible-doc -t connection -l` to see all available connection types.
- Common “core” connection types for networking:
  - `ansible.netcommon.network_cli` (MOST COMMON)
  - `ansible.netcommon.netconf`
  - `ansible.netcommon.httpapi`

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

```
---
- name: "CONNECTION TYPES"
  hosts: "all"
  connection: "local"
  gather_facts: false
```

>>> 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: "ansible.netcommon.network_cli"
  gather_facts: false
```

```yaml
---
- name: "CONNECTION TYPES"
  hosts: "all"
  connection: "ansible.netcommon.netconf"
  gather_facts: false
```

>>> 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: "ansible.netcommon.network_cli"
  gather_facts: false

  tasks:

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

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

>>> Modules, Parameters, and Variables

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

```yaml
---

- name: "MANAGE VLANS"
  hosts: "switches"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false

  tasks:

    - name: "ENSURE VLAN EXISTS"
      cisco.nxos.nxos_vlan:
        vlan_id: 10
        name: "web_vlan"
        state: "present"
```

>>> Introducing the CONFIG module

- Module name: cisco.ios.ios_config (e.g. *_config for main OSs prefixed by namespace.collection) or ansible.netcommon.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: "ansible.netcommon.network_cli"
  gather_facts: false
  tasks:
    - name: "ENSURE SNMP COMMANDS EXIST ON IOS DEVICES TASK 1 in PLAY 1"
      cisco.ios.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 in PLAY 1"
      cisco.ios.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 1"
      cisco.ios.ios_config:
        src: "cisco_ios.cfg"
```

>>> 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 times and it'll only change once
- If you see something different, the module is not idempotent or there is a bug in the module (or the API)

>>> 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

>>> Collections of Modules and Plugins

- Introduced in Ansible 2.8, usable in 2.9, getting better in 2.10+
- Fully Qualified Collection Name (FQCN)
  - Similar to how Fully Qualified Domain Name (FQDN) is for DNS
  - namespace.collection.content_name
  - e.g., `cisco.ios.ios_config` instead of the former ios_config only
- List of Collections at:

  https://docs.ansible.com/ansible/latest/collections/index.html#list-of-collections
- Many non-product-specific modules are in the `ansible.builtin` collection including assemble, assert, debug, file, copy, include, include_role, include_tasks, lineinfile, template, and uri.
- Note that a namespace and a collection can ONLY contain characters from [a-zA-Z0-9_]

>>> Ansible Ad hoc Commands

- For commands run infrequently or to test what goes into the Ansible playbook
- Define and run a single task 'playbook' (of sorts) against a set of hosts

```
$ ansible -h
usage: ansible [-h] [--version] [-v] [-b] [-K] [-i INVENTORY] [--list-hosts]
  [-k] [-u REMOTE_USER] [-c CONNECTION] [-T TIMEOUT]
  [--syntax-check] [-D] [-e EXTRA_VARS] [-a MODULE_ARGS]
  [-m MODULE_NAME] pattern
$ # some content removed for brevity
```

>>> Ansible Ad hoc Commands (con't)

```
ntc@ntc-training:~$ ansible --version
ansible [core 2.11.6]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/etc/ntc/ansible/library']
  ansible python module location = /usr/local/lib/python3.8/site-packages/ansible
  ansible collection location =
/home/ntc/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/local/bin/ansible
  python version = 3.8.12 (default, Oct 13 2021, 09:22:51) [GCC 8.3.0]
  jinja version = 3.0.2
  libyaml = True
ntc@ntc-training:~$ ansible localhost -m command -a "python --version"
[WARNING]: No inventory was parsed, only implicit localhost is available
localhost | CHANGED | rc=0 >>
Python 3.8.12
```

>>> Ansible Ad hoc Commands (con't)

```
ntc@ntc-training:~$ ansible localhost -m ping
[WARNING]: No inventory was parsed, only implicit localhost is available
localhost | SUCCESS => {
      "changed": false,
      "ping": "pong"
}
```

>>> 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
- Override 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
```

```
$ 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": true

```
$ 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
```

>>> Play Recap (Cont)

```
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=ntc123

[routers:vars]
ansible_user=admin

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

>>> Playbook Task Syntax

Recommended YAML Syntax (key:value)

```yaml
---

- name: "MANAGE VLANS"
  hosts: "switches"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false

  tasks:

    - name: "ENSURE VLAN EXISTS"
      cisco.nxos.nxos_vlan:
        vlan_id: 10

    # no curly braces used with var parameter
    - ansible.builtin.debug:
        var: "inventory_hostname"
```

Vertical and/or Horizontal (key=value)

This is more common in older playbooks.

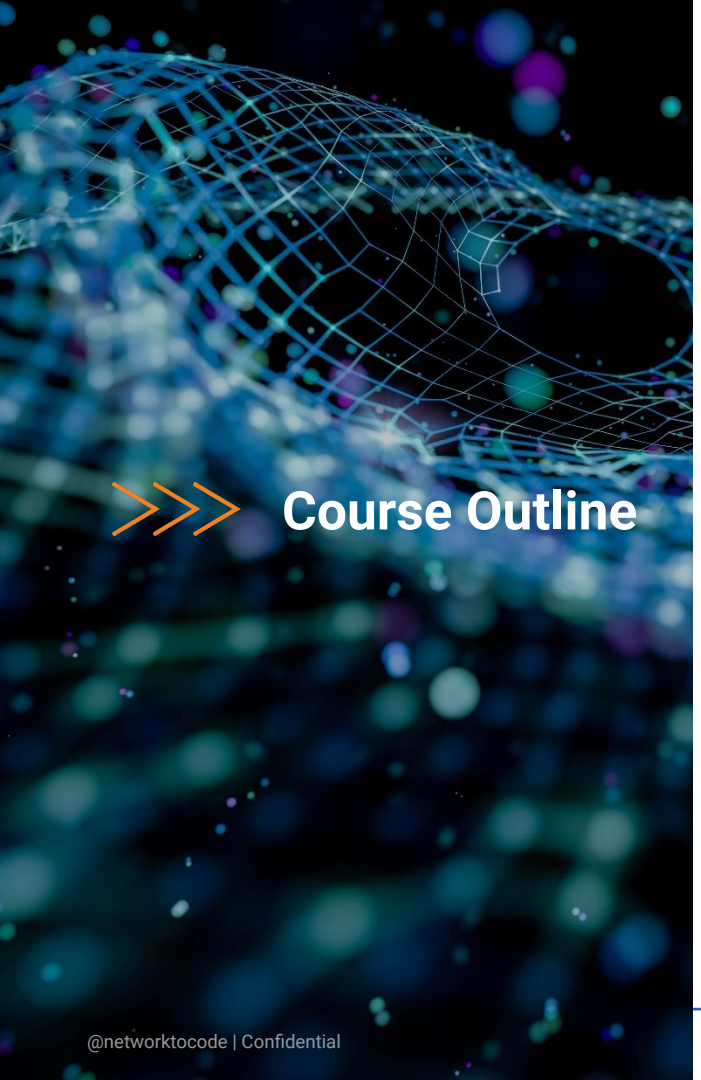
```yaml
---

- name: "MANAGE VLANS"
  hosts: "switches"
  connection: "ansible.netcommon.network_cli"
  gather_facts: false

  tasks:

    - name: "ENSURE VLAN EXISTS"
      cisco.nxos.nxos_vlan:
        "vlan_id: 10
        host: '{{ inventory_hostname }}'
        username: '{{ username }}'
        password: '{{ password }}'"

    - name: "ENSURE VLAN EXISTS"
      cisco.nxos.nxos_vlan: "vlan_id: 10 host: '{{ inventory_hostname
}}' username: '{{ username }}' password: '{{ password }}'"

    # no curly braces used with var parameter
    - ansible.builtin.debug: "var=inventory_hostname"
```

Lecture 1: Ansible Architecture and Playbook Examples

Labs 1-2

>>> Lab Time

- Lab 1 - Deploying "Basic" Configurations Commands with Ansible
- Lab 2 - Deploying Configurations From Files Using *_config



Course Outline

Introduction

- Course Overview
- What's Possible with Ansible
- Introduction to Ansible Vocabulary

Ansible Variables and Password Security

- Quick Look at YAML and JSON
- Understanding Ansible Variables
- Ansible Vault for Password Security
- Debug Module to Display Ansible Variable Content

Lecture 2: Understanding YAML and Ansible Variables

Topic 4 - Quick Look at YAML and JSON

Topic 5 - Understanding Ansible Variables

Labs 3-5

>>> Topic 4: Quick Look at YAML and JSON

YAML and JSON basics

>>> YAML

- Human readable data serialization language
- Heavily used for configuration files
- Relies heavily on indentation
- 2 space indent is common
- Superset of JSON

>>> YAML Basics

YAML documents should start with 3 hyphens (`---`)
Basic Key-Value Pairs

YAML

```yaml
---
hostname: switch1
snmp_ro: public
snmp_rw: private
snmp_location: "nyc"

# integer
vlan_id: 100

# string
voice_vlan_id: "101"
```

JSON

```json
{
  "hostname": "switch1",
  "snmp_ro": "public",
  "snmp_rw": "private",
  "snmp_location": "nyc",
  "vlan_id": 100,
  "voice_vlan_id": "101"
}
```

Note: You can comment YAML but not JSON.

>>> YAML Basics

List of Strings / Numbers

YAML

```yaml
---
snmp_ro_communities:
  - public
  - public123

vlans:
  - 100
  - 101
  - 102
  - 103
  - 104
```

JSON

```json
{
    "snmp_ro_communities": [
        "public",
        "public123"
    ],
    "vlans": [
        100,
        101,
        102,
        103,
        104
    ]
}
```

>>> YAML Basics

List of dictionaries

YAML

```yaml
---

- vlan_name: web
  vlan_id: '10'
  vlan_state: active
- vlan_name: app
  vlan_id: '20'
  vlan_state: active
- vlan_name: DB
  vlan_id: '30'
  vlan_state: active
```

JSON

```json
[
    {
    "vlan_name": "web",
    "vlan_id": "10",
    "vlan_state": "active"
    },
    {
    "vlan_name": "app",
    "vlan_id": "20",
    "vlan_state": "active"
    },
    {
    "vlan_name": "DB",
    "vlan_id": "30",
    "vlan_state": "active"
    }
]
```

>>> YAML Advanced Data Types

Dictionaries

YAML

```yaml
---

snmp:
  ro: public
  rw: private
  info:
    location: nyc
    contact: bob

vlans:
  10:
    name: web
  20:
    name: app
```

JSON

```json
{
  "snmp": {
    "ro": "public",
    "rw": "private",
    "info": {
      "location": "nyc",
      "contact": "bob"
    }
  },
  "vlans": {
    "10": {
      "name": "web"
    },
    "20": {
      "name": "app"
    }
  }
}
```

>>> YAML Advanced Data Types

Dictionaries that are lists of dictionaries

YAML

```yaml
---
vlans:
  - id: 10
    name: web
  - id: 20
    name: app

snmp_community_strings:
  - type: RO
    community: public
  - type: RO
    community: networkcode
  - type: RW
    community: private
```

JSON

```json
{
 "vlans": [
   {
     "id": 10,
     "name": "web"
   },
   {
     "id": 20,
     "name": "app"
   }
 ],
 "snmp_community_strings": [
   {
     "type": "RO",
     "community": "public"
   },
   {
     "type": "RO",
     "community": "networktocode"
   },
   {
     "type": "RW",
     "community": "private"
   }
 ]
}
```

>>> YAML Advanced Data Types

YAML is a superset of JSON

YAML

```yaml
---
ned:Loopback:
  #YAML supports comments
  name: 200
  ip:
    address:
      primary:
        address: 100.200.2.2
        mask: 255.255.255.0
      secondary:
        - address: 100.200.20.20
        - address: 100.200.200.200
```

JSON

```json
{
  "ned:Loopback": {
    "name": 200,
      "ip": {
      "address": {
        "primary": {
          "address": "100.200.2.2",
          "mask": "255.255.255.0"
        },
        "secondary": [
          {
            "address": "100.200.20.20"
          },
          {
            "address": "100.200.200.200"
          }
        ]
      }
      }
  }
}
```

>>> YAML Data Types - Summary

- For most automation tasks, YAML and JSON have 1-1 mapping
- They both tie back to dictionaries and lists
- YAML tends to be more concise than JSON
- YAML allows comments while JSON does not allow comments
- A lot of initial automation tasks revolve around parsing return data, therefore it is important to understand:
  - Lists of lists
  - Lists of dictionaries
  - Dictionaries with lists
  - Complex nested objects
- Always remember to traverse a complex object from left to right

>>> Comparison of YAML, JSON, and XML

Description	YAML	JSON	XML
Outer descriptor	(none)	{ }	<name> </name>
Dictionary delimiter	(none)	{ }	<> </>
Dictionary example	key: value	{ key1: value1, key2: value2 }	<key1> value1 </key1>
List delimeter	-	[ ]	Sequential items
List example	- value	[value]	name1=value1 name2=value2
Comment delimeter	#	No (though non-standard /* ... */)	No
Block comment delimeter	""" or '''	No (though non-standard /* ... */)	No
Block comment example	""" comment here """	/* comment here */	No
Common application	Python	Javascript or Python	Java

>>> YAML Demo

- Validate YAML
- http://yamllint.com/
- YAML to JSON Conversion
- JSON to YAML Conversion
- https://www.json2yaml.com
- Understand how to model network configuration data in YAML (for use with Ansible)
- Compare/Contrast Data Models on different platforms

>>> Topic 5: Understanding Ansible Variables

Group based variables

Host based variables

Special variables

Extra variables

>>> 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"
```

>>> Ansible Variable Precedence -- see Ansible docs “Using variables”

https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#

Documentation

ANSIBLEFEST PRODUCTS COMMUNITY WEBINARS & TRAINING BLOG

EXTENDING ANSIBLE

Developer Guide

COMMON ANSIBLE SCENARIOS

Public Cloud Guides

Network Technology Guides

Virtualization and Containerization Guides

NETWORK AUTOMATION

Network Getting Started

Network Advanced Topics

Network Developer Guide

ANSIBLE GALAXY

Galaxy User Guide

Galaxy Developer Guide

REFERENCE & APPENDICES

Collection Index

Indexes of all modules and plugins

Playbook Keywords

Return Values

Ansible Configuration Settings

Controlling how Ansible behaves:

Understanding variable precedence

Ansible does apply variable precedence, and you might have a use for it. Here is the order of precedence from least to greatest (the last listed variables override all other variables):

1. command line values (for example, `-u my_user`, these are not variables)
2. role defaults (defined in role/defaults/main.yml) [1]
3. inventory file or script group vars [2]
4. inventory group_vars/all [3]
5. playbook group_vars/all [3]
6. inventory group_vars/* [3]
7. playbook group_vars/* [3]
8. inventory file or script host vars [2]
9. inventory host_vars/* [3]
10. playbook host_vars/* [3]
11. host facts / cached set_facts [4]
12. play vars
13. play vars_prompt
14. play vars_files
15. role vars (defined in role/vars/main.yml)
16. block vars (only for tasks in block)
17. task vars (only for the task)
18. include_vars
19. set_facts / registered vars
20. role (and include_role) params
21. include params
22. extra vars (for example, `-e "user=my_user"`)(always win precedence)

Lecture 2: Understanding YAML and Ansible Variables

Labs 3-5

>>> Lab Time

- Lab 3 - Deploying Configuration From Files using cli_config
- Lab 4 - Using Check Mode and Verbosity
- Lab 5 - Creating the Course Inventory

>>> Lecture 3: Interacting with Ansible Data

Topic 6 - Introducing the Debug Module to View Data Stored in Variables

Topic 7 - User Input

Labs 6-7

Topic 6: Introducing the Debug Module to View Data Stored in Variables

Playbook Variables

Debug Mode

>>> Playbook Variable

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 parameter

```yaml
---
- name: "PRINT VLANS"
  hosts: "all"
  connection: "local"
  gather_facts: false
  vars:
    vlan: 300
  tasks:
    - name: "PRINT VLAN"
      ansible.builtin.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

```yaml
---
- name: "PRINT HOSTS"
  hosts: "all"
  connection: "local"
  gather_facts: false

  vars:
    priority: "P1"

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

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
```

```
.
├── inventory
├── debug.yml
```

```yml
---
- name: "DEBUGGING VARIABLES"
  hosts: "all"
  connection: "local"
  gather_facts: false
  tasks:
    - name: "PRINT INVENTORY HOSTNAME"
      ansible.builtin.debug:
        var: "inventory_hostname"
```

>>> 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: false
  tasks:
    - name: "PRINT INVENTORY HOSTNAME"
      ansible.builtin.debug:
        var: "inventory_hostname"
```

```
.
├── inventory
├── debug.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=0
leaf2                      : ok=1    changed=0
unreachable=0    failed=0
```

>>> 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
```

```
.
├── inventory
├── debug.yml
```

```yaml
---
- name: "DEBUGGING VARIABLES"
  hosts: "all"
  connection: "local"
  gather_facts: false
  tasks:
    - name: "PRINT PASSWORD"
      ansible.builtin.debug:
        var: "ansible_ssh_pass"
```

>>> debug module

```
.
├── inventory
├── debug.yml
```

```
# 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: false
  tasks:
    - name: "PRINT PASSWORD"
      ansible.builtin.debug:
        var: "ansible_ssh_pass"
```

```
$ 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=0
leaf2                      : ok=1    changed=0
unreachable=0    failed=0
```

>>> 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
```

```
.
├── inventory
├── debug.yml
```

```yaml
---
- name: "DEBUGGING VARIABLES"
  hosts: "all"
  connection: "local"
  gather_facts: false
  tasks:
    - name: "PRINT MGMT IP"
      ansible.builtin.debug:
        msg: "The MGMT IP is {{mgmt_ip}}"
```

>>> 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: false
  tasks:
    - name: "PRINT MGMT IP"
      ansible.builtin.debug:
        msg: "The MGMT IP is {{mgmt_ip}}"
```

```
.
├── inventory
├── debug.yml
```

```
$ ansible-playbook -i inventory debug.yml
PLAY [DEBUGGING VARIABLES]
*******************************************

TASK [PRINT INVENTORY HOSTNAME]
*******************************************
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=0
leaf2                      : ok=1    changed=0
unreachable=0    failed=0
```

>>> debug module with multiple lines

```
# 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: "leaf1"
  connection: "local"
  gather_facts: false
  tasks:
    - name: "PRINT MGMT IP"
      ansible.builtin.debug:
        msg:
          - "The MGMT IP is {{mgmt_ip}}"
          - "  with more lines in a list."
```

```
.
├── inventory
├── debug.yml
```

```
$ ansible-playbook -i inventory debug.yml
PLAY [DEBUGGING VARIABLES]
*******************************************

TASK [PRINT INVENTORY HOSTNAME]
*******************************************
ok: [leaf1] => {
    "msg": [
        "THE MGMT IP IS 10.1.1.1",
        "  with more lines in a list."
    ]
}

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

Topic 7: User Input

JSON Data

Documentation

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: false
  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.

>>> 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
```

>>> 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`

Lecture 3: Interacting with Ansible Data

Labs 6-7

>>> Lab Time

- Lab 6 - Using the debug module
- Lab 7 - Passing in Interactive User Input

>>> network.toCode()

Ansible - Bonus Topics

Ansible Vault and Password Security

Bonus Topic

>>> Ansible Vault

The Ansible Vault functionality allows the user:

- To store sensitive data as encrypted text on the filesystem.
- Use unencrypted data on the fly during playbook execution.
- Typically used to store username and passwords on the control machine.
- Tell playbook to look for variables in a file with a command such as "`vars_file: vaultfile.yml`" or "`include_vars:`" followed by a list of secret variable definitions.
- Option: Consider using the Linux file structure to "hide" this file by instead naming it something like "`.vaultfile.yml`" where it will not be seen by an "`ls -l`" command (only "`ls -al`")

```
ntc@ntc: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 variables

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

The encrypted version of above data:

```
ntc@jump-host:all$ cat vaultfile.yml
$ANSIBLE_VAULT;1.1;AES256
38353863306139626235623263313439653437646261393562323036356531336432323736646534
31613337373164303964313139316338636465353034326 60a353461636464303238353765343162
31346366353766663063303636386265326665643331326632613536363831346364663065316462
63656463373638386 50a326563386465383662643733633930323264333065633034363338643735
3332356665623863343662373262306231356238646566666433396138616131303 4
```

>>> Ansible Vault - Strategy 1 of 2

- 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@ntc: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 - Strategy 2 of 2

- Use the `--vault-password-file` flag to reference the file containing the vault password.
- Use file permission 600, 640, or 644 to minimize who has access read and write access..
- Avoid execute permission (no odd octal permission digits) so Ansible doesn't try to execute the file as a script
- Reference encrypted files containing variables using "vars_files: *insert_filename_here*"

```
ntc@ntc:ansible$ ansible-playbook -i inventory use_vault.yml --vault-password-file ~/.vaultpw

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` or `--vault-password-file`