Ansible Zero to Hero – Network Automation (ver. 2.0)

LTRPRG-1125

Speakers:

Vishal Kakkar – Solutions Architect

Sanjeewa Alahakone- Solutions Architect

Table of Contents

About

Purpose

This lab guide documents step by step procedure for running ansible use cases in the provided lab.

Prerequisites

- Linux operating system commands execution experience
- Basic router configuration experience using CLIs
- Understanding networking concepts will be helpful but not mandatory.

Outcome

Upon completion of this lab, you will be equipped with:

- Strong understanding of Ansible, its architecture, building blocks and application use cases.
- Write up your own playbooks for network automation
- Aware of the ansible best practices, dos and don'ts
- Confidently decide when & when not to use Ansible.

Disclaimer

This lab guide is built for the learning purpose. Audience should use this lab guide to have hands-on experience on ansible. The use cases captured in this guide are designed to cover verity of ansible capabilities and may or may not be used as is for real network environments.

Also, an ansible controller (a Linux host) is setup with pre-installed ansible to save on time.

Ansible

Introduction

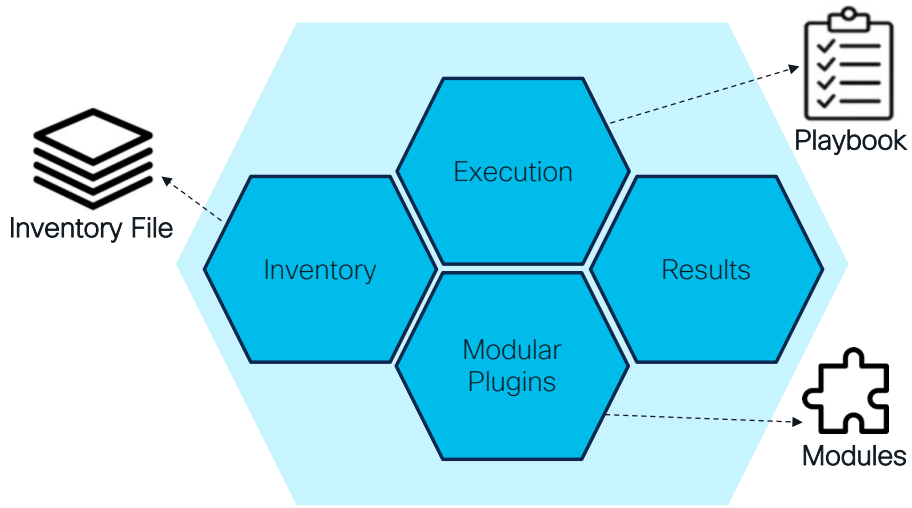
Traditional, manual approaches to network configuration and updates are too slow and error-prone to effectively support today's rapidly shifting application and data transfer requirements. Programmable, software-based automation technologies can help your team better support your organization's digital initiatives.

With network automation, network operations (NetOps) teams can quickly respond to dynamic needs for capacity, application security, load balancing, and multi-cloud integrations. They can also implement self-service and on-demand network activities.

As a result, NetOps teams can become as agile and flexible as applications and infra- structure teams to support modern business demands.

Architecture

Ansible is open-source automation framework with following building blocks.

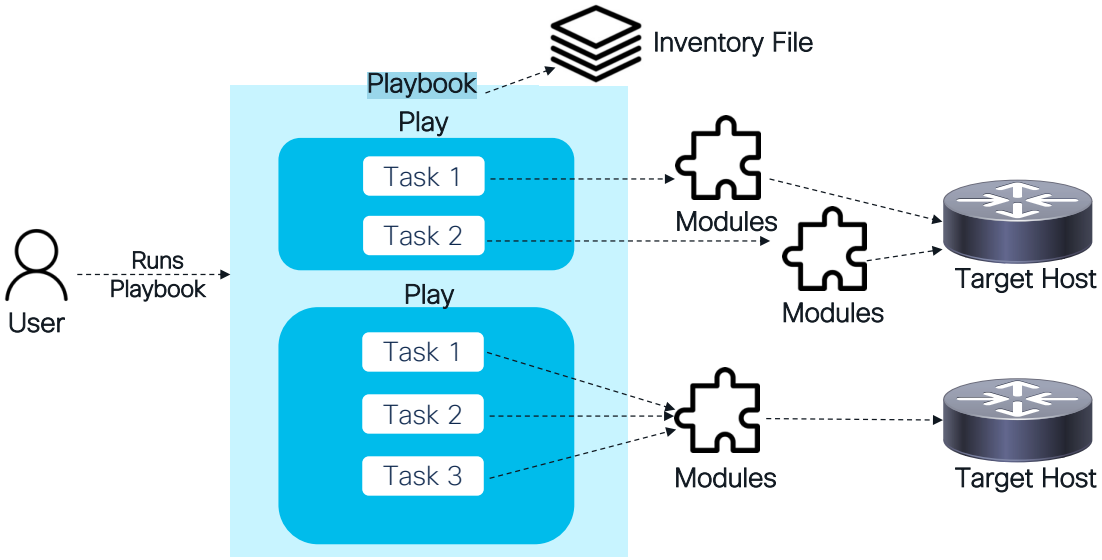


Inventory or host file contains details of all the target hosts/devices to be controlled by ansible.
Playbook contains the list of tasks to be executed.
Modules are reusable scripts to abstract the underlying complexity of running tasks on target device.

Workflow

When User runs an ansible playbook, playbook refers to inventory file to get target host's details like IP, credentials, protocol, any specific variables defined for it.

Playbook then starts executing plays, each play contains one or more tasks. Tasks then uses modules to communicate with the target host.



Advantages

Apart from being open source, following features make ansible stand out.

Simple

- Uses simple syntax written in YAML (Yet Another Markup Language)
- No Code Low Code philosophy, YAML is just enough

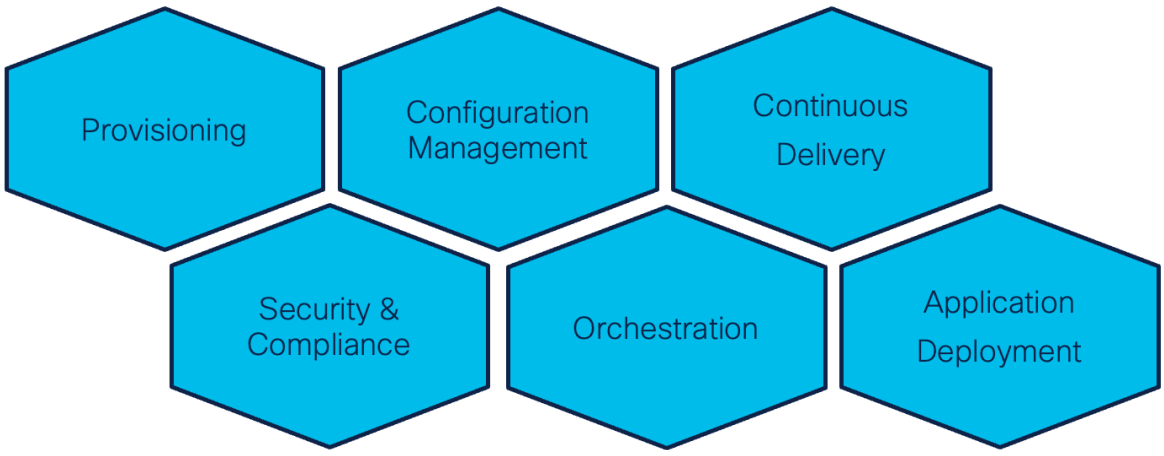
Agentless

- No Agents or software required to be installed on target hosts.
- No special firewall ports need to be opened as ansible uses SSH.

Powerful

- Features that enable to model even complex workflows

Use Cases

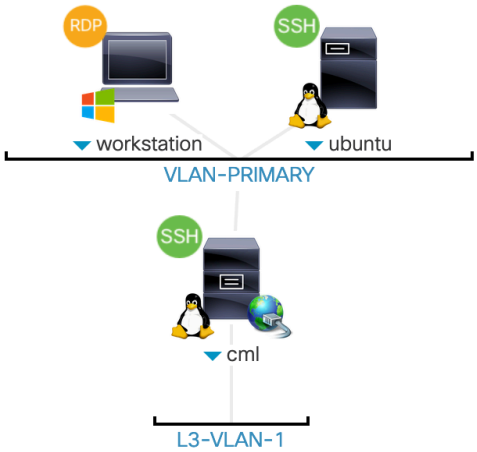


Lab Details

Topology

This content includes preconfigured users and components to illustrate the scripted scenarios and features of the solution. Most components are fully configurable with predefined administrative user accounts. You can see the IP address and user account credentials to use to access a component by clicking the component icon in the **Topology** menu of your active session and in the scenario steps that require their use.

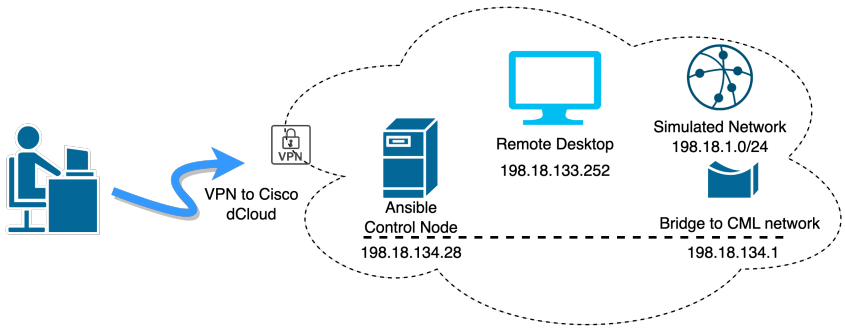
dCloud Topology



This Lab hosted on dcloud has three main components:

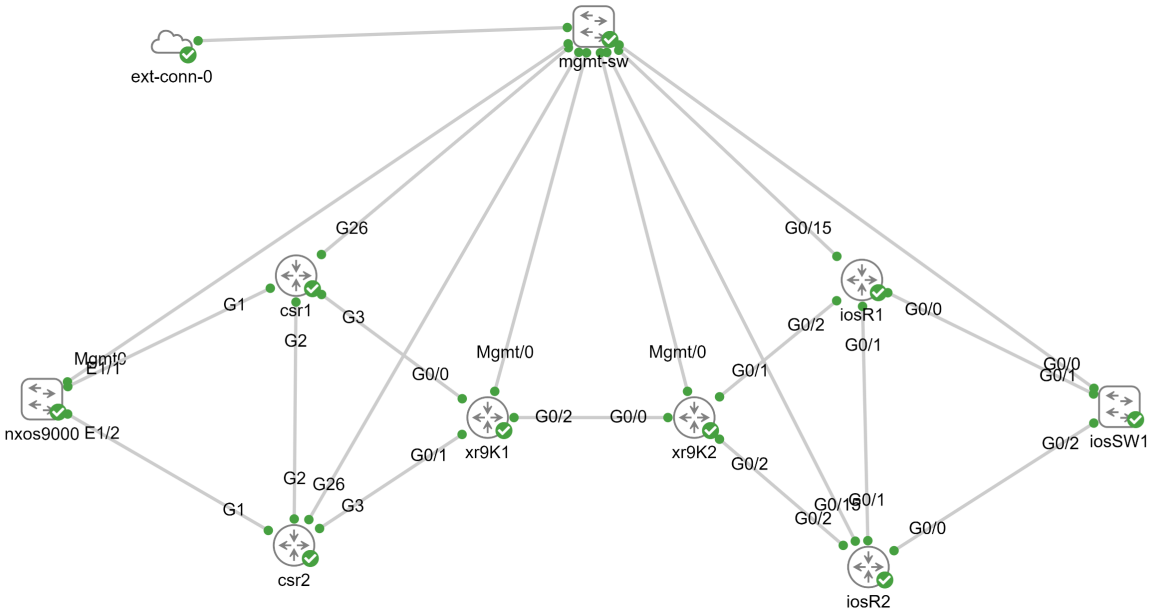
- Windows jump host
- Ubuntu VM with pre-installed ansible
- CML instance to simulate network topology

Connectivity Details



- Connect to the windows jump host using remote desktop.
- Jump host has pre-installed putty with saved sessions to Ansible control node (ubuntu VM) and simulated network devices.
- Jump host also has Visual Studio Code editor for working on ansible without logging into ansible controller

Network Topology



Above network Topology is designed to cover most of the Cisco routing platforms like IOS, IOS-XR, CSR, NXOS.

Equipment IPs & Credentials

Name	Description	IP Address	Username	Password
NXOS9000	Cisco NXOS	198.18.1.11	cisco	cisco
CSR1	Cisco IOS-XE	198.18.1.12	cisco	cisco
CSR2	Cisco IOS-XE	198.18.1.13	cisco	cisco
XR9K1	Cisco IOS-XR	198.18.1.14	cisco	cisco
XR9K2	Cisco IOS-XR	198.18.1.24	cisco	cisco
iosR1	Cisco IOS	198.18.1.22	cisco	cisco
iosR2	Cisco IOS	198.18.1.23	cisco	cisco
iosSW	Cisco IOS	198.18.1.21	cisco	cisco
Ansible controller	Ubuntu VM	198.18.134.28	cisco	C1sco12345
Jump host	Windows VM	198.18.133.252	Administrator	C1sco12345

Lab Guide Notes

Symbol represents a procedure that **you must execute** to understand the topic better.

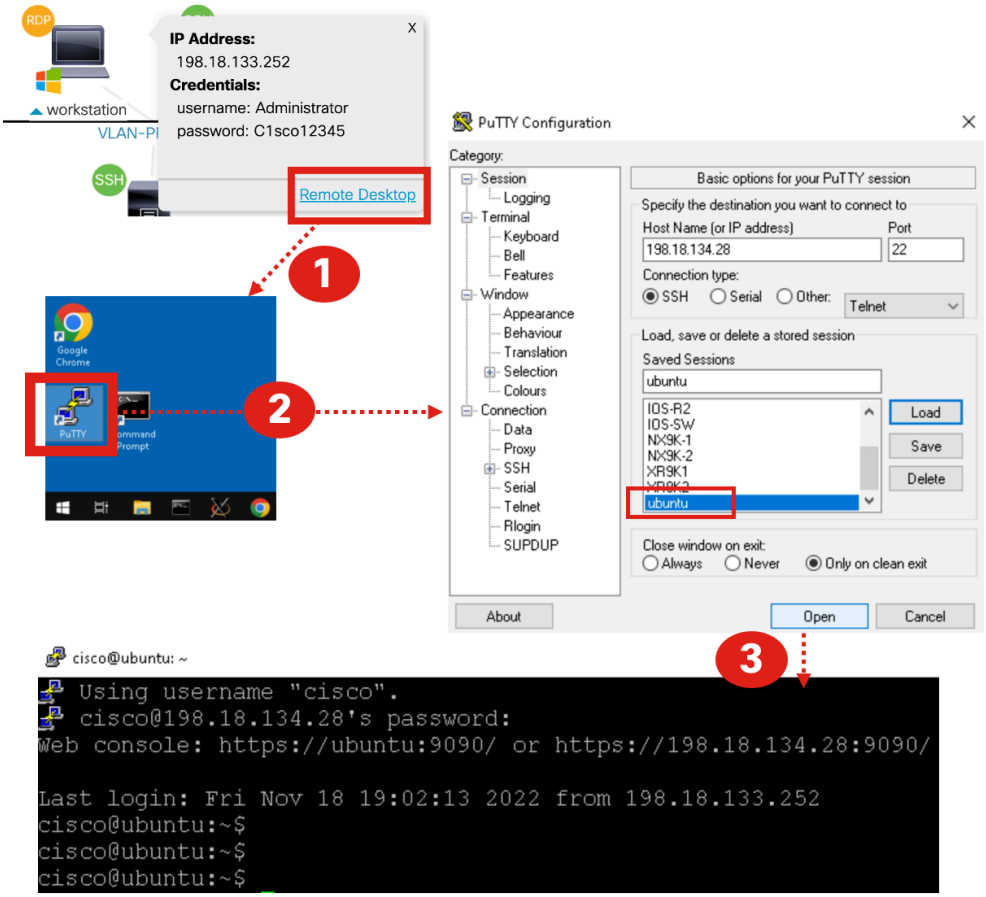
Throughout the lab guide, you should be able to use **copy & paste** for **commands** than typing them manually to save on time and accuracy.

Lab-1: Familiarize with Ansible Environment

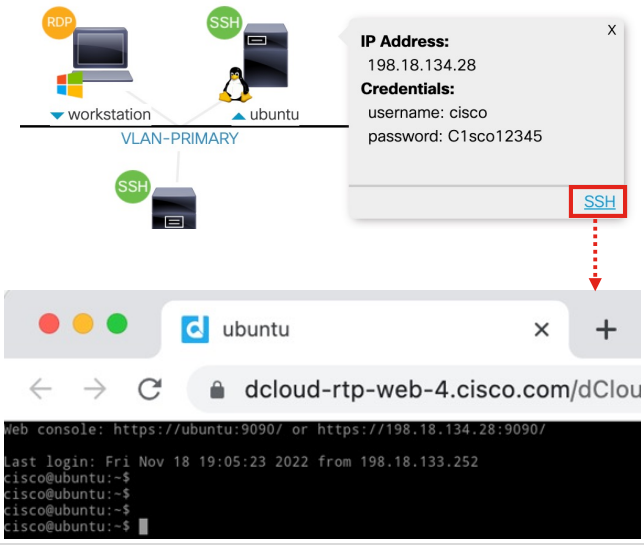
This lab has an **ubuntu** virtual machine with **pre-installed ansible**. This section will cover the ansible environment available on that host.

The host where ansible is installed to act as centralized controller is known as **ansible control node**. Two ways to connect to the control node are:

- **Remote desktop** to Windows Jump host and then use **putty**, credentials **cisco/C1sco12345**



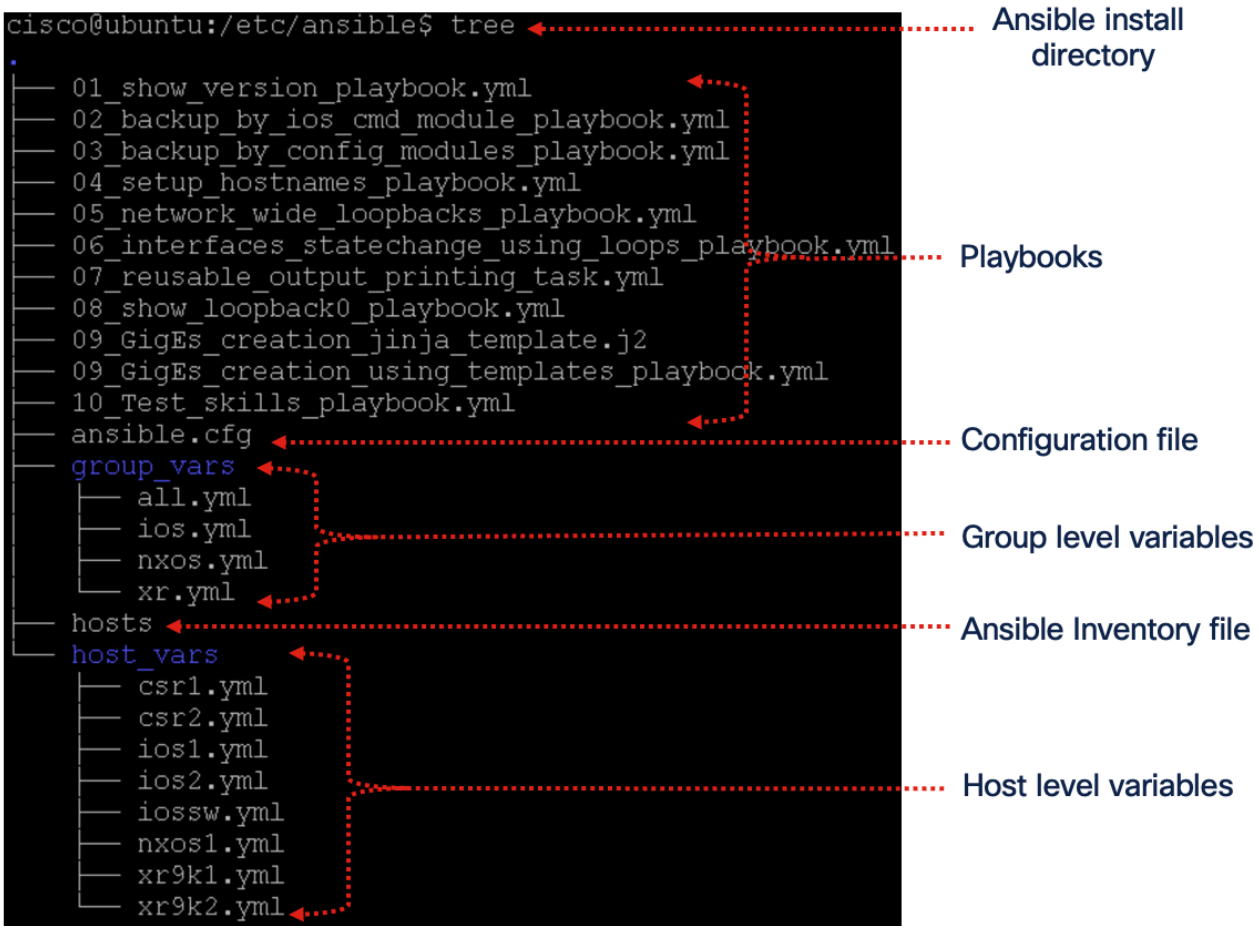
- Use **dcloud browser** to SSH in control node directly.



Task 1: Control Node directory structure

Ansible is installed in **/etc/ansible** directory on the control node.

Login to control node and check this directory location and its contents.



Default files that come with ansible installation are **ansible.cfg** & **hosts**, rest all the playbooks are created for this session.

Task 2: Ansible Configuration file

Configuration file (**/etc/ansible/ansible.cfg)** contains base configurations of ansible environment on control node. As shown below it controls where to look for inventory file, modules, and batch size of the number of target devices to run tasks in parallel.

```
cisco@ubuntu:/etc/ansible$ more /etc/ansible/ansible.cfg
```

```
[defaults]

# some basic default values...

#inventory      = /etc/ansible/hosts
#library        = /usr/share/my_modules/
#module_utils   = /usr/share/my_module_utils/
#remote_tmp     = ~/.ansible/tmp
#local_tmp      = ~/.ansible/tmp
#plugin_filters_cfg = /etc/ansible/plugin_filters.yml

forks          = 15
```

Inventory file location

Module's location

Batch size (number of target devices)

Gathering parameter is used by play to **collect information** about **the target** device. For most of environments hosts information doesn't change frequently so better to use **smart**, for environments when playbooks are not at all dependent on target device information this should be set to **explicit**.

```
# smart - gather by default, but don't regather if already gathered
# implicit - gather by default, turn off with gather_facts: False
# explicit - do not gather by default, must say gather_facts: True

gathering = smart
```

host_key_checking should be False to allow ansible connect to target devices without checking SSH keys.

```
# uncomment this to disable SSH key host checking
host_key_checking = False
```

Verify ansible.cfg these parameters are set correctly.

Task 3: Ansible Inventory file

Inventory file contains the list of target devices. Target devices can be added by IP or hostname (nslookup will be required) and they can be grouped based on their function/types (or as you like).

 Have a look at the inventory file (written in **INI** format) and validate the target devices.

```
cisco@ubuntu:/etc/ansible$ more /etc/ansible/hosts

[nxos]
nxos1

[ios]
csr1
csr2
ios1
ios2
iossw

[xr]
xr9k1
xr9k2
```

Here devices are grouped based on their types (nxos, ios, and xr). Grouping helps to execute a **playbook on multiple devices** by targeting it to a group name than individual device.

A device can be part of **multiple groups.**

Also, variables can be defined in inventory file, host_vars or group_vars folder.

Two default groups are implicitly available for each inventory file.

- **All** – This group contains all the target devices defined in inventory, regardless of whether they are part of existing group/s or not. In our lab **"all"** will represent all seven devices.
- **Ungrouped** – This group contains only those devices that are **not part of any group**. In our lab we won't have any device that's ungrouped so will have this group empty.

Groups can also be nested using "**:children"** suffix.

 Try creating a group named **routers** that contains all the devices of **ios** and **iosxr** groups.

```
[routers: children]
ios
xr
```

Task 4: host_vars & group_vars

A directory named **host_vars** becomes of special significance when created at in same parent directory containing inventory file (/etc/ansible)

Any host/target device scoped variables can be defined in this directory.

Check **csr1.yaml** content inside host_vars.

```
cisco@ubuntu:/etc/ansible$ more host_vars/csr1.yaml

---
ansible_host: 198.18.1.12
ID: 2

new_hostname:
  name: csr1-new
```

This YAML file contains host **IP address** specific to **csr1** device, and **ID value "2"** is also specific to this csr1 device. These variables will be referenced in playbooks upcoming in this lab guide.

Any number of variables, dictionaries, lists can be added in YAML file for a particular host.

Similarly, A directory named **group_vars** becomes of special significance when created at in same parent directory containing inventory file (/etc/ansible)

Any group scoped variables can be defined in this directory.

In our inventory we have grouped devices in three groups: ios, xr, nxos

Check **xr.yaml** content inside group_vars.

```
cisco@ubuntu:/etc/ansible$ more group_vars/xr.yaml

---
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: cisco.iosxr.iosxr

interfaces_to_enable:        [GigabitEthernet0/0/0/0,        GigabitEthernet0/0/0/1,
GigabitEthernet0/0/0/2]
```

This YAML file contains host **network_os** that's supported by devices in "xr" group i.e., xr9k1 & xr9k2. These group variables will also be referenced in playbooks described in this lab guide.

To know what all variables are associated with hosts/target devices, because of host_vars or group_vars. Run the following command

```
cisco@ubuntu:/etc/ansible$ ansible-inventory --list -y
```

Task 5: Ansible Playbooks

Playbooks are Ansible's configuration, deployment, and orchestration language. They consist of sets of **human-readable instructions** called **plays** that define automation across an inventory of hosts.

Each **play includes** one or more **tasks** that target one, many, or all hosts in an inventory. Each task calls an Ansible module that performs a specific function like collecting useful information, backing up network files, managing network configurations, or validating connectivity.

Playbooks are written in **YAML** format. Plays and playbook are helpful to group tasks to have a meaningful and reusable workflow.

 Check the playbooks available on control node for your reference.

```
#cd /etc/ansible/
cisco@ubuntu:/etc/ansible$ ls *.yaml
01_show_version_playbook.yaml
02_backup_by_ios_cmd_module_playbook.yaml
03_backup_by_config_modules_playbook.yaml
04_setup_hostnames_playbook.yaml
05_network_wide_loopbacks_playbook.yaml
06_interfaces_statechange_using_loops_playbook.yaml
07_reusable_output_printing_task.yaml
08_show_loopback0_playbook.yaml
09_GigEs_creation_using_templates_playbook.yaml
10_Test_skills_playbook.yaml
```

Task 6: Ansible Modules

Modules are reusable scripts that can be used by the Ansible API, or by the ansible or **ansible-playbook** programs. Modules encapsulates communication with specific target device type.

Modules expose simple functions that takes inputs and perform desired action on the target device, abstracting all the underlying complexity.

Modules are very powerful as they let ansible users focus on the outcome than indulging into writing low level code for specific tasks implementation.

Refer to following URL to find updated module information.
https://docs.ansible.com/ansible/2.5/modules/list_of_all_modules.html#all-modules

 Check what all modules are available on control node and filter few Cisco modules

```
# ansible-doc -l | grep cisco                  Filter Cisco related modules
# ansible-doc -l | grep cisco.nxos.nxos_lldp   filter specific module from above
# ansible-doc cisco.nxos.nxos_lldp_global.     Reference documentation
```

Lab-2: Basic Ansible Commands

Task 1: Running Ad-hoc commands

An Ansible ad-hoc command uses the **/usr/bin/ansible** or just **ansible** command-line tool to automate a single task on one or more managed nodes.

Ad-hoc commands are quick and easy, but they are not reusable.

Here is your first ansible command to know the version of all devices in the inventory, without ansible one would need to login to the device and use CLI "show version".

```
cisco@ubuntu:/etc/ansible# ansible csr1 -e "ansible_connection=ssh" -m raw -a "show version | include Version"

198.18.1.22 | CHANGED | rc=0 >>

Cisco IOS Software, IOSv Software (VIOS-ADVENTERPRISEK9-M), Version 15.9(3)M4, RELEASE SOFTWARE (fc3)

**************************************************************************
* IOSv is strictly limited to use for evaluation, demonstration and IOS  *
* education. IOSv is provided as-is and is not supported by Cisco's      *
* Technical Advisory Center. Any use or disclosure, in whole or in part, *
* of the IOSv Software or Documentation to any third party for any       *
* purposes is expressly prohibited except as otherwise authorized by     *
* Cisco in writing.                                                      *
**************************************************************************Shared connection to 198.18.1.22 closed.
```

Simple but very useful, Now let's get versions of all your network device by just one CLI.

```
cisco@ubuntu:/etc/ansible# ansible all -e "ansible_connection=ssh" -m raw -a "show version"
```

Note: **-e** (or --extra-vars) is used to overwrite any ansible variable defined at device or group level.

Another good use of **ansible** command is to see the list of devices defined in inventory.

List all the target devices that are part of inventory and observe all 8 devices are listed.

```
cisco@ubuntu:/etc/ansible# ansible all --list-hosts

  hosts (8):
    nxos1
    csr1
    csr2
    ios1
    ios2
    iossw
    xr9k1
    xr9k2
```

List all the **ios-xr** target devices and observe only two devices are listed.

```
cisco@ubuntu:/etc/ansible# ansible xr --list-hosts
 hosts (2):
   xr9k1
   xr9k2
```

Task 2: Running First ansible playbook

Running a **single CLI or action** is **not always enough**. Usually, we would need to run multiple commands on target device/s to achieve the desired outcome.

Playbooks are run using **ansible-playbook <playbook-name>** command.

Using ad-hoc command to get versions of all devices returned so much information in above exercise and you had to scroll through multiple lines to get to version details. Now let's try to get just the precise output i.e., version but nothing else using playbook.

Run your first playbook to get version information of all the devices

```
cisco@ubuntu:/etc/ansible$ ansible-playbook 01_show_version_playbook.yaml
```

```
ok: [198.18.1.11] => {
    "show_ver_output.stdout_lines[0][0]": "Cisco Nexus Operating System (NX-OS) Software"
}
ok: [198.18.1.13] => {
    "show_ver_output.stdout_lines[0][0]": "Cisco IOS XE Software, Version 17.03.04a"
}
ok: [198.18.1.12] => {
    "show_ver_output.stdout_lines[0][0]": "Cisco IOS XE Software, Version 17.03.04a"
}
ok: [198.18.1.24] => {
    "show_ver_output.stdout_lines[0][0]": "Cisco IOS XR Software, Version 6.6.3[Default]"
}
ok: [198.18.1.22] => {
    "show_ver_output.stdout_lines[0][0]": "Cisco IOS Software, IOSv Software (VIOS-ADVENTERPRISEK9-M), Version 15.9(3)M4, RELEASE SOFTWARE (fc3)"
}
ok: [198.18.1.23] => {
    "show_ver_output.stdout_lines[0][0]": "Cisco IOS Software, IOSv Software (VIOS-ADVENTERPRISEK9-M), Version 15.9(3)M4, RELEASE SOFTWARE (fc3)"
}
ok: [198.18.1.14] => {
    "show_ver_output.stdout_lines[0][0]": "Cisco IOS XR Software, Version 6.6.3[Default]"
}
ok: [198.18.1.21] => {
    "show_ver_output.stdout_lines[0][0]": "Cisco IOS Software, vios_l2 Software (vios_l2-ADVENTERPRISEK9-M), Experimental Version 15.2(20200924:215240)
kge-sep24-2020-l2iol-release 135]"
}
```

Here only version information is shown in single line for each device.

Playbook with **multiple tasks** can be run step by step, so that user can intervene to confirm whether to execute or skip the very next task.

```
cisco@ubuntu:/etc/ansible$ ansible-playbook 01_show_version_playbook.yaml --step
```

To observe what's happening in background during playbook execution, use **"-vvv"** to run in verbose mode.

```
cisco@ubuntu:/etc/ansible$ ansible-playbook 01_show_version_playbook.yaml -vvv
```

Lab-3: Playbooks Deep Dive

Task 1: Understand Playbook Content

Objective: Understand the playbook **contents**.

Open **01_show_version_playbook.yaml** playbook and understand the **YAML** structure from below **in-line comments** & the **notes on the right**.

```
1  ---
2
3  # Play book to show version of all IOS & XRs
4  - name: Get Device versions
5    hosts: all,!nxos
6    tasks:
7      # First Task to get the versions from devices
8      - name: Running 'show version' command
9        # Using ios_command module
10       ios_command:
11         commands:
12           - show version
13       # Saving the output of above CLI to a variable named show_ver_output
14       register: show_ver_output
15
16     # Second Task to parse returned JSON and extract only relevant information for user
17     - name: Extracting only relevant information from 'show version' response
18       debug:
19         # Extracting Only the first line for show version CLI output
20         var: show_ver_output.stdout_lines[0][0]
```

Starts with three "-"
"#" used for comments
Name of Play
Target device/groups
Task to run CLI on target
Network Module being used
Function of network module
Parameters to the function
Saving task output to variable
Task to parse CLI output
Module to print to screen
Parameter to print

Apart from above details, do observe this playbook has **one play** (Get Device Versions) and **two tasks** (Running command & extracting relevant info)**.**

In real-world, Playbooks do have **multiple plays** with each play having **multiple tasks**.

Playbook can be checked for YAML syntax errors, make a change in above playbook to remove **"-"** ahead of play's name and validate playbook.

```
cisco@ubuntu:/etc/ansible$ ansible-playbook 01_show_version_playbook.yaml --syntax-check
```

ATTENTION **Didn't work? CLI should have "-syntax-check" and not "--syntax<space>-check".**

```
Syntax Error while loading YAML.
  did not find expected key

The error appears to be in '/etc/ansible/01_show_version_play.yml': line 10, column 7, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

      # Using ios_command module
      ios_command:
      ^ here
```

Hosts parameter in a playbook can be a **group name** (ios, nxos, xr, all) or **combination of group names** (comma or colon separated), even with **negation**.

Make changes to **01_show_version_playbook.yaml** playbook to run on all but not XR devices.

```
hosts: all,!xr
```

Task 2: Router Backup using playbook

Objective: ios_command network module and **COPY** module.

A very common use case is to download running configuration from the network devices and save in local repository for future usage.

Here is a simple playbook that first runs a task to fetch "show running-config" and then second task uses a **copy module** to save the configuration on local filesystem.

Open **02_backup_by_ios_cmd_module_playbook.yaml** and understand what parameters copy module takes (content & destination).

```yaml
---
# Play book to download router configuration
- name: Get Device running configs
  hosts: all,!nxos
  tasks:
    # First Task to get the running configuration from devices
    - name: Running 'show running-config' command
      # Using ios_command module
      ios_command:
        commands:
          - show running-config
      # Saving the output of above CLI to a variable named show_run_output
      register: show_run_output

    # Secong Task to parse returned JSON and save to backup directory on control node
    - name: Saving running config output to local directory with timestamp
      copy:
        # Copy module to copy show_run_output content to device specific file with timestamp
        content: "{{ show_run_output.stdout[0] }}"
        dest: "/etc/ansible/backups_by_ios_cmd/{{ inventory_hostname}}_{{ lookup('pipe','date +%Y-%m-%d_%H-%M-%S') }}.txt"
```

COPY module to save fetched configs

Destination location with timestamp

COPY module is part of ansible core, there are numerous such modules available for end user consumption. This makes ansible great tool for quick wins.

Run **02_backup_by_ios_cmd_module_playbook.yaml** playbook **multiple times** and observe the configurations downloaded in backups folder. Please note you may need to create backups folder if it doesn't exist.

```
cisco@ubuntu:/etc/ansible$ ansible-playbook 02_backup_by_ios_cmd_module_playbook.yaml
```

ATTENTION **COPY** module expects that destination directory must pre-exist. So you would need to create **backups_by_ios_cmd** directory in /etc/ansible if prompted.

```
cisco@ubuntu:/etc/ansible$ mkdir -p backups_by_ios_cmd
```

Observe the backed-up configurations.

```
backups_by_ios_cmd/
├── csr1_2022-11-21_09-39-03.txt
├── csr1_2022-11-21_09-54-13.txt
├── csr2_2022-11-21_09-39-03.txt
├── csr2_2022-11-21_09-54-13.txt
├── ios1_2022-11-21_09-39-03.txt
├── ios1_2022-11-21_09-54-13.txt
├── ios2_2022-11-21_09-39-03.txt
├── ios2_2022-11-21_09-54-13.txt
├── iossw_2022-11-21_09-39-03.txt
├── iossw_2022-11-21_09-54-13.txt
├── xr9k1_2022-11-21_09-39-03.txt
├── xr9k1_2022-11-21_09-54-13.txt
├── xr9k2_2022-11-21_09-39-03.txt
└── xr9k2_2022-11-21_09-54-13.txt
```

Task 3: Router Backup using playbook (Another way)

Objective: group_vars, network modules **ios_config, nxos_config, iosxr_config** & **when condition**.

Here is another playbook to backup device configuration, by using **ios_config** than **ios_command** network module.

Open **03_backup_by_config_modules_playbook.yaml** and understand the new module **ios_config** along with **backup** & **backup_options** parameters.

```
1   ---
2   # Another Play book to download router configuration
3   - name: network wide backup
4     connection: network_cli
5     gather_facts: false
6     hosts: all
7     tasks:
8       - name: Backuping up current IOS config (ios)
9         # Using ios_config Module & backup parameter
10        cisco.ios.ios_config:
11          backup: yes
12          backup_options:
13            dir_path: "{{ backup_location }}"
14        # When clause to run this task only for IOS devices
15        when: ansible_network_os == 'cisco.ios.ios'
```

Config module

Backup running-config

Backup location

Only for IOS devices

Observe the use of **when** condition, having this condition will run this task only for **ios devices.**

In this playbook, there are **three such tasks** one for each type of devices ios, xr, nxos and they all are using different modules **cisco.ios.ios_config, cisco.iosxr.iosxr_config** and **cisco.nxos.nxos_config**

Benefits of using this approach is that destination directory doesn't need manual creation, these network modules take care of creating the directory during execution.

Another new thing here is **{{ backup_location }},** this is way to access variables in ansible. This variable is defined in **group_vars/all.yaml.** Check **backup_location** variable in group_vars/all.yaml.

Run **03_backup_by_config_modules_playbook.yaml** playbook **multiple times** and observe the configurations downloaded in backups folder. Please note the folder is auto created.

```
cisco@ubuntu:/etc/ansible$ ansible-playbook
03_backup_by_config_modules_playbook.yaml
```

Observe the backed-up configurations

```
backups_by_config_module/
├── csr1_config.2022-11-21@10:34:20
├── csr2_config.2022-11-21@10:34:21
├── ios1_config.2022-11-21@10:34:22
├── ios2_config.2022-11-21@10:34:22
├── iossw_config.2022-11-21@10:34:21
├── nxos1_config.2022-11-21@10:34:25
├── xr9k1_config.2022-11-21@10:34:29
└── xr9k2_config.2022-11-21@10:34:29
```

Task 4: Update hostnames using host_vars

Objective: host_vars variables and **when** condition.

All above examples have been only reading the device configurations, now let's try to make configuration changes to devices.

In this task, we will try to assign the new **hostname** to IOS and NXOS devices.

Open **04_setup_hostnames_playbook.yaml** playbook and observe followings:

- Usage of **when** condition to control whether a task should be run or skipped.
- Where Variable **new_hostname.name** is coming from.

```
1   ---
2   # Play book to uddate router hostname, defined in host_vars
3   - name: Update device hostname defined in host_vars
4     hosts: nxos,xr
5     tasks:
6
7       - name: Updating NXOS hostnames
8         # Using nxos_config module
9         nxos_config:
10          lines:
11            # Usage of variable defined in host_vars/{hostname}.yml
12            - hostname {{ new_hostname.name }}
13          save_when: modified
14        when: ansible_network_os == "cisco.nxos.nxos"
15
16      - name: Updating XR hostnames
17        # Using iosxr_config
18        cisco.iosxr.iosxr_config:
19          lines:
20            # Usage of variable defined in host_vars/{hostname}.yml
21            - hostname {{ new_hostname.name }}
22        when: ansible_network_os == "cisco.iosxr.iosxr"
```

Config module

Variable from host_vars

When to move running config to startup config

Here new_hostname variable is **not defined** in the inventory file (**hosts**) but it is **defined in host_vars** subfolder.

```
cisco@ubuntu:/etc/ansible$ tree host_vars/
host_vars/
├── csr1.yml
├── csr2.yml
├── ios1.yml
├── ios2.yml
├── iossw.yml
├── nxos1.yml
├── xr9k1.yml
└── xr9k2.yml
```

Each of these YAML files defined in host_vars contains variables that are **scoped to that specific device** only, so when playbook runs it picks the new_hostname value of the device, task is being performed on.

Again, NXOS and IOS uses different modules (**nxos_config & ios_config**) for updating device configs, **Conditional when** is used to ensure right module is used for the device being acted upon.

Checkout **new_hostname.name** defined in the host_vars for xr9k1 & xr9k2.

```
cisco@ubuntu:/etc/ansible$ vi host_vars/xr9k1.yaml
```

```
cisco@ubuntu:/etc/ansible$ vi host_vars/xr9k2.yaml
```

Run **04_setup_hostnames_playbook.yaml** playbook.

```
cisco@ubuntu:/etc/ansible$ ansible-playbook 04_setup_hostnames_playbook.yaml
```

SSH to impacted devices and check the hostname (**password**: cisco)

```
cisco@ubuntu:/etc/ansible$ ssh cisco@198.18.1.14
RP/0/0/CPU0:xr9k1-new#
```

```
cisco@ubuntu:/etc/ansible$ ssh cisco@198.18.1.24
RP/0/0/CPU0:xr9k2-new#
```

```
cisco@ubuntu:/etc/ansible$ ssh cisco@198.18.1.14
nxos1-new#
```

Task 5: Configuring Loopback0 network wide

Objective: set_facts to define run-time variables, **host_vars**, **lines & parents** params of config module.

This playbook will configure loopback0 on all the devices across network. For each loopback0 to have unique IP address, a **variable** named **"ID"** is defined in each host_vars file.

Validate "ID" variable for few of devices in **host_vars**.

```
cisco@ubuntu:/etc/ansible$ vi host_vars/csr1.yaml
---
ansible_host: 198.18.1.12
ID: 2
```

Open **05_network_wide_loopbacks_playbook.yaml** and observe followings:

- Usage of built-in **set_fact** task to set runtime variables.
- **"lines"** parameter containing device configurations to be pushed and "**parents"** parameter to make ansible push the lines **under specific section**.

```
2   # Playbook to configure Loopback0 network wide
3   # with loopback ip address computed on the fly using "ID" variable
4   # defined in the host_vars/{hostname}.yml
5   - name: Configuring interface loopback0
6     gather_facts: false
7     connection: network_cli
8     hosts: all
9
10    tasks:
11      - set_fact:
12          ip_addr: "{{ '172.16.0.0/24' | ansible.utils.nthhost(ID) }}"
13    #  - debug: var=ip_addr
14      - name: ip config for loopback (ios)
15        cisco.ios.ios_config:
16          lines:
17          - description anchor interface
18          - ip address {{ ip_addr }} 255.255.255.255
19          parents: interface Loopback0
20        when: ansible_network_os == "cisco.ios.ios"
21
22      - name: ip config for loopback (XR)
23        cisco.iosxr.iosxr_config:
24          lines:
25          - description anchor interface
26          - ip address {{ ip_addr }} 255.255.255.255
27          parents: interface Loopback0
28        when: ansible_network_os == "cisco.iosxr.iosxr"
29
30      - name: ip config for loopback (NXOS)
31        nxos_config:
32          lines:
33          - description anchor interface
34          - ip address {{ ip_addr }} 255.255.255.255
35          parents: interface Loopback0
36        when: ansible_network_os == "cisco.nxos.nxos"
```

Task to set runtime variables

Using ID variable to generate IP

Device configurations

CLI section where to apply above configs

Run **05_network_wide_loopbacks_playbook.yaml** playbook.

```
cisco@ubuntu:/etc/ansible$ ansible-playbook 05_network_wide_loopbacks_playbook.yaml
```

Validate that loopback0 are configured on all the devices

```
cisco@ubuntu:/etc/ansible$ ansible all -m cli_command -a "command='show running-config interface Loopback0'"
```

ATTENTION **Didn't work? CLI should have "running-config" and not "running<space>-config".**

Task 6: Update Multiple interfaces single CLI

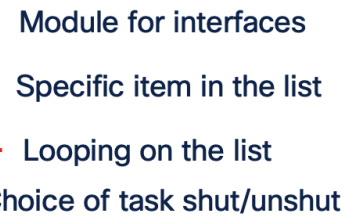
Objective: loops on the list defined in **group_vars, defining/overwriting variables** during playbook execution via ansible-playbook CLI.

This playbook depicts use case of changing state of multiple interfaces by a single playbook.

Open **06_interfaces_statechange_using_loops_playbook.yaml** and observe followings:

- Interfaces being impacted are defined in variable **interfaces_to_change** in **group_vars/xr.yaml**
- **with_items** construct is used to loop through the list of interfaces.
- Conditional **when** is used with **state** variable to check if request is to shut or unshut.

```
1   ---
2   # Play book to enable/disable GigabitEthernet interfaces on XR
3   # interfaces_to_enable list is defined in group_vars/xr.yml
4   - name: Updating GigabitEthernet operational state
5     hosts: xr
6     gather_facts: false
7
8     tasks:
9       - name: UnShuting interfaces on XRs
10        cisco.iosxr.iosxr_interfaces:
11          config:
12          - name: "{{ item }}"
13            enabled: true
14        with_items: "{{ interfaces_to_change }}"
15        when: state == "enable"
16
17      - name: Shuting interfaces on XRs
18        cisco.iosxr.iosxr_interfaces:
19          config:
20          - name: "{{ item }}"
21            enabled: false
22        with_items: "{{ interfaces_to_change }}"
23        when: state == "disable"
```



Checkout **interfaces_to_change** defined in the group_vars for xr9k1 & xr9k2.

```
cisco@ubuntu:/etc/ansible$ vi group_vars/xr.yaml
```

Run **06_interfaces_statechange_using_loops_playbook.yaml** playbook.

```
cisco@ubuntu$ ansible-playbook 06_interfaces_statechange_using_loops_playbook.yaml
```

ATTENTION It will complain for **"state"** variable **not defined**, as its not in any host_vars/group_vars.

Run **06_interfaces_statechange_using_loops_playbook.yaml** again with **state=disable** via CLI

```
cisco@ubuntu$  ansible-playbook  06_interfaces_statechange_using_loops_playbook.yaml
-e "state=disable"
```

Note: **-e** (or --extra-vars) is used to define the state variable that's not defined earlier.

Validate that the **interfaces_to_change** are now **shutdown** state using ad-hoc command.

```
cisco@ubuntu:/etc/ansible$ ansible xr -m cli_command -a "command='show ipv4 interface brief'"
```

Run **06_interfaces_statechange_using_loops_playbook.yaml** again with **state=enable** via CLI

```
cisco@ubuntu$  ansible-playbook  06_interfaces_statechange_using_loops_playbook.yaml -e "state=enable"
```

Re-validate that the **interfaces_to_change** are now back to **up state** using ad-hoc command.

```
cisco@ubuntu:/etc/ansible$ ansible xr -m cli_command -a "command='show ipv4 interface brief'"
```

Lab-4: Advanced Topics

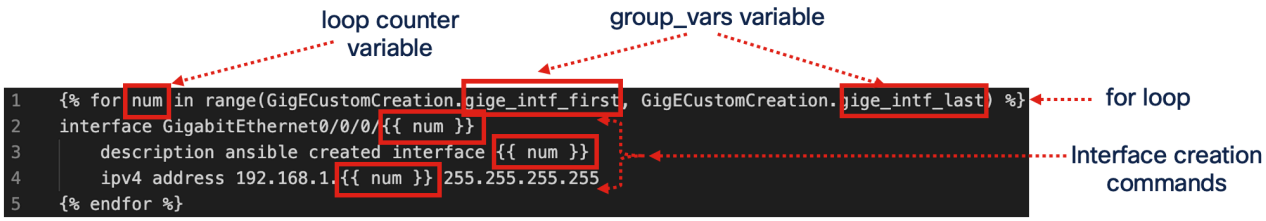
Task 1: Ansible Templates

Objective: Templates to make more dynamic playbooks.

Ansible uses **Jinja template** to create dynamic playbooks. Here is a playbook that uses a template to provision **multiple** GigabitEthernet interfaces.

Open **09_GigEs_creation_jinja_template.j2** and observe followings:

- **for loop** to create GigabitEthernet interfaces, with **num** as **auto incremented** counter variable.
- **gige_intf_first** variable defined in **group_vars** to indicate the starting interface index
- **gige_intf_last** variable defined in **group_vars** to indicate the last interface index



Checkout **gige_intf_first** & **gige_intf_last** defined in the group_vars for **all** device group.

```
cisco@ubuntu:/etc/ansible$ vi group_vars/all.yaml
```

Open **09_GigEs_creation_using_templates_playbook.yaml** and observe how its leveraging above jinja template using "**src**" parameter of **ios_config** network module.

```
1  ---
2  # Playbook to configure GigabitEthernets on all devices
3  # Jinja template is being used to prepare the configuraiton.
4  # Number of interfaces to create is coming from group_vars/all.yml
5  - name: configure Mutiple GigabitEthernet interfaces on target
6    gather_facts: false
7    connection: network_cli
8    hosts: xr
9
10   tasks:
11     - name: Configuring GigabitEthernets on XR
12       cisco.iosxr.iosxr_config:
13         src: 09_GigEs_creation_jinja_template.j2
14         match: none
15       when: ansible_network_os == "cisco.iosxr.iosxr"
```

Configuration source pointing to Jinja template

Run **09_GigEs_creation_using_templates_playbook.yaml**

```
cisco@ubuntu$ ansible-playbook 09_GigEs_creation_using_templates_playbook.yaml
```

Validate that the **GigabitEthernet interfaces** are pre-configured, using ad-hoc command.

```
cisco@ubuntu:/etc/ansible$ ansible xr -m cli_command -a "command='show running-config interface'"
```

Task 2: Component Reuse (Define once, use many times)

Objective: Reusability aspects.

There are instances when same tasks are required to be performed in multiple playbooks. Writing same task repeatedly is not the desired design. Ansible helps us to create once and re-use.

Open **07_reusable_output_printing_task.yaml** and observe it's a small but common task of printing **output_var** variable to console for operator.

```
---
# Reusable task to print the output_var
# Any playbook can import this task than
# writting this mutiple places in playbooks
- name: Play to print output to screen
  debug:
    var: output_var.stdout_lines
```

Open **08_show_loopback0_playbook.yaml** and observe how its leveraging existing task than writing it again in playbook.

```
---
# Play book to show Loopback0 configs from devices
# by importing re-usable task 07_reusable_output_printing_play.yml
- name: Get Loopback0 configuration
  hosts: all
  tasks:
    # First Task to get the versions from devices
    - name: Getting Loopback0 configs
      # Using ios_command module
      ios_command:
        commands:
          - show running-config interface Loopback0
      # Saving the output of above CLI to a variable named show_output
      register: output_var

    # Importing task defined externaly (another YAML)
    - import_tasks: 07_reusable_output_printing_play.yml
```

Re-use existing task

Run **08_show_loopback0_playbook.yaml** and observe outcome.

```
cisco@ubuntu$ ansible-playbook 08_show_loopback0_playbook.yaml
```

Note: Like **import_tasks**, there is another directive **include_tasks**.

- import_tasks are imported to playbook **during playbook parsing** itself
- include_tasks are evaluated **during execution**, so good to be used for decision making workflows.

```
- include_tasks: {{runtime_var}}_task.yaml
```

Depending on what **runtime_var** resolves to, different task will be included, hence changing the workflow.

Task 3: Managing Sensitive data (Credentials)

Objective: Ansible Vault to save secrets.

So far, we have seen variable been defined in various places like host_vars, group_vars, hosts etc, but all those have been stored as clear text human readable format.

To save the sensitive information, ansible offers **ansible-vault** to encrypt and store those secrets.

Open **group_vars/all.yaml** and observe how **ansible_password** is stored.

```
cisco@ubuntu$ vi group_vars/all.yaml

ansible_user: cisco
ansible_password: cisco
```

Use **ansible-vault** command to encrypt the string **cisco ,** as it's the password we want to use.

```
cisco@ubuntu:/etc/ansible$ ansible-vault encrypt_string cisco -n ansible_password

New Vault password:         <Enter any password and remember>
Confirm New Vault password: <Enter any password and remember>
Encryption successful
ansible_password: !vault |
          $ANSIBLE_VAULT;1.1;AES256
  61346362623539393731326336356563373665383765633864663664306261363339613434616263
  37363232386465353633393064363063316664656439356 30a32626134663433353332303062626 6
  35383334663030363038326335353163333339323835636163356661663564613336383261613030
  64623632366132316 60a39353433623665386135343336303534616333313363613739613666306 4
  6432
```

Now copy the output to **group_vars/all.yaml** to start using encrypted password.

```
cisco@ubuntu$ vi group_vars/all.yaml

ansible_user: cisco
ansible_password: !vault |
          $ANSIBLE_VAULT;1.1;AES256
  61346362623539393731326336356563373665383765633864663664306261363339613434616263
  37363232386465353633393064363063316664656439356 30a32626134663433353332303062626 6
  35383334663030363038326335353163333339323835636163356661663564613336383261613030
  64623632366132316 60a39353433623665386135343336303534616333313363613739613666306 4
  6432
```

ATTENTION Don't use above output but use the one you got in your environment from last step.

Run the first playbook **01_show_version_playbook.yaml**

```
cisco@ubuntu:/etc/ansible$ ansible-playbook 01_show_version_playbook.yaml
```

It will fail with following error as ansible expects us to provide the vault password needed to decrypt.
fatal: [csr1]: FAILED! => {"msg": "Attempting to decrypt but no vault secrets found"}

Run the first playbook **01_show_version_playbook.yaml** with **--ask-vault-password** option.

```
cisco@ubuntu:/etc/ansible$  ansible-playbook  01_show_version_playbook.yaml  --ask-vault-password

Vault password: < Enter the password set earlier >
```

Didn't work? CLI should have "--ask-vault-password" and not "--ask-<space>vault-password".

Note: Other option is to save vault password in a file (for example vault_pass.txt) and run

```
cisco@ubuntu:/etc/ansible$ ansible-playbook 01_show_version_playbook.yaml --vault-password-file vault_pass.txt
```

(**Optional**) Complete files can also be encrypted, like the variables as well.

```
cisco@ubuntu:/etc/ansible$ ansible-vault encrypt host_vars/csr1.yaml

New Vault password:         <Enter any password and remember>
Confirm New Vault password: <Enter any password and remember>
Encryption successful
```

Check encrypted file:

```
cisco@ubuntu$ vi host_vars/csr1.yaml
```

Decrypting encrypted file:

```
cisco@ubuntu:/etc/ansible$ ansible-vault decrypt host_vars/csr1.yaml
Vault password: <Enter the password set earlier>
Decryption successful
```

Conclusion

Congratulations!! You have successfully completed this ansible lab. Just to recap on learnings of this session, you had gone through:

- Ansible Architecture (Inventory, Modules, Playbooks)
- Running Ad-hoc commands
- Running Playbooks
- Building Playbooks to read device configurations.
- Building Playbooks to make device configuration changes.
- Using variables, loops, conditions in the playbooks
- Using Jinja templates to have more dynamic playbooks.
- Re-using common tasks in multiple playbooks.
- Special handling of security sensitive data with ansible Vault.

Hope you had great learning session. Please **do share** your **valuable feedback** about **your experience** of this session.