

What are You Going to Learn?

We hope you have gained a good understanding of what is Infrastructure as a code. In this course, you will learn about

- Ansible and its benefits
- Ansible in Infrastructure as code (IaC)
- YAML format
- Few Ad_Hoc commands
- Different parts of Ansible
- Role of Modules in Ansible
- How to write Playbooks
- How to Control flow of execution in Ansible
- Few points about configuration file (ansible.cfg)
- How to setup environment in local machine and online playground

By the end of this course, you will be in a position to write a few **ad-hoc commands** and **Playbook** of your own.

IaC means writing code for infrastructure i.e. your systems and devices, which are used to run Softwares, are to be treated as software and defined using code(which can be done using a high level or descriptive language).

For example: version control, testing, small deployments, use of design patterns etc.

Configuration Management tools are used to accomplish IaC.

Infrastructure as Code can be achieved using **Ansible**, which is one of the popular **Configuration Management tools**.

Configuration Management is a process of establishing, tracking and controlling the current design and build state of the system (software versions).

It also ensures that **past records of system state** is easily and accurately accessible which helps in project management, audit purposes, debugging etc.

Before Configuration Management

Consider you are planning for a **New Year Special Sale** in your e-commerce site. You need to

- **Scale up** your servers
- Then **configure** them(and all other old servers) for special new year sale
- This whole process would take **lot of effort and time**

- What if new configurations did not work as expected? Then you will have to **roll back to previous stable version**, which will add more work and subtract the profits and potential customers while in downtime.

You need some kind of Configuration Management tool that can automate these tasks:

Roll back to stable version with zero downtime

Provide you with constant computing environment throughout SDLC

Automatically scale up or down depending upon traffic

Some of the popular Configuration Management tools are: Ansible, Puppet, Chef and Saltstack.

Ansible is an open source software, first released by Michael DeHaan in 2012 and owned by Red Hat.

It is used to **automate**

- *configuration management*
- *application deployment*
- *software provisioning* and other IT needs

Benefits of Ansible

Simple: Very easy to install, setup and learn. Written in YAML file which is pretty much like reading English.

Agentless: Do not need to install any agents on target nodes.

Powerful: It can model any complex IT workflow as it has 1100+ modules

Efficient: You can customize modules, using any programming language

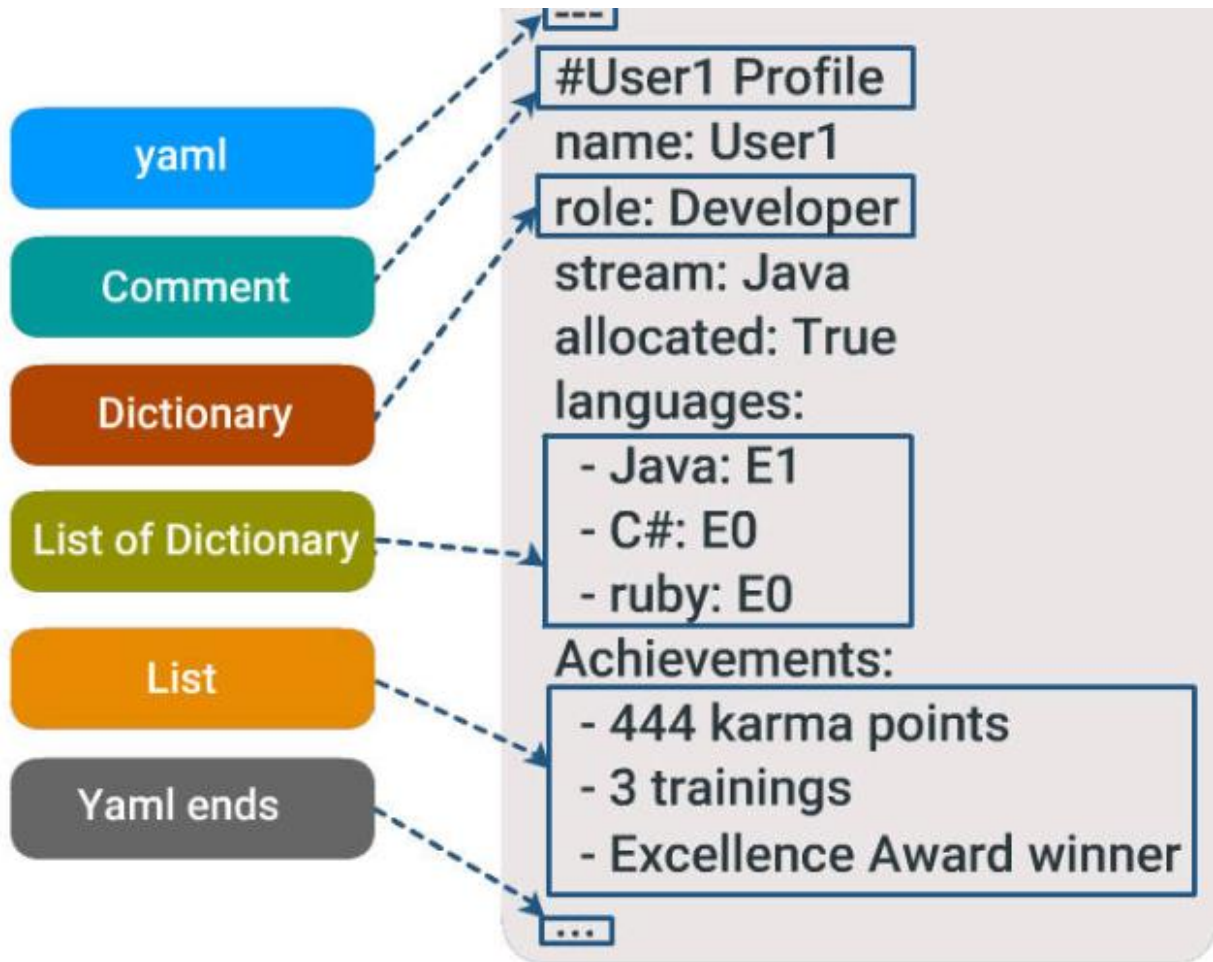
Secure: Uses SSH for connection

What is YAML?

YAML stands for **Yet Another Markup Language**. It is a data serialization language just like JSON.

Why YAML, when we already have JSON or XML?

- YAML files are easy to read and write for humans, similar to English. YAML files should end as .yaml or .yml
- **Begins** with --- and **ends** with ...
- # defines comment



YAML is Case and Indentation Sensitive

- **Members of a list should be at the same indentation level** starting with a dash(-) and space.
- Each item in the list is a **key: value pair (colon must be followed by a space)**, called as **dictionary**.
- At each level, exactly **two spaces are used for indentation**. Using tabs is not recommended here.

Boolean Values

Variables can be defined in YAML files as shown:

```
stream: Java
```

```
allocated: true
```

Variables can be assigned boolean values in different ways as shown:

```
allocated: yes
```

```
allocated: no
```

allocated: True

allocated: TRUE

allocated: false

Data Structures

Complicated data structures are possible in YAML.

You can define **lists having dictionaries**, **dictionaries having lists** or a **mix** of both.

In the following example,

- name and job are dictionaries. Skill is a **list of dictionaries**.
- David and Amy are **lists having dictionaries**

Employee records

- David:

name: David Moore

job: Developer

skills:

- python
- sql
- java

- Amy:

name: Amy Brown

job: Developer

skills:

- angular
- redux
- react

Ansible is a/an _____.

Orchestration Engine

Configuration management

Infrastructure as code

All the options

Design goals of Ansible include _____.

All the options

Highly reliable

Consistent

Low learning curve

Secure

_____ is a valid YAML syntax.

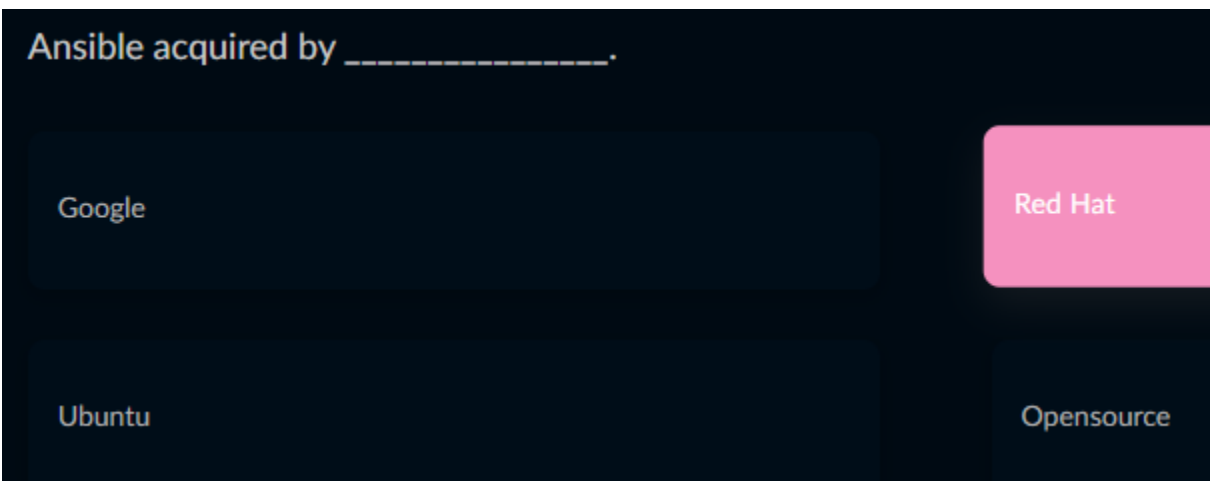
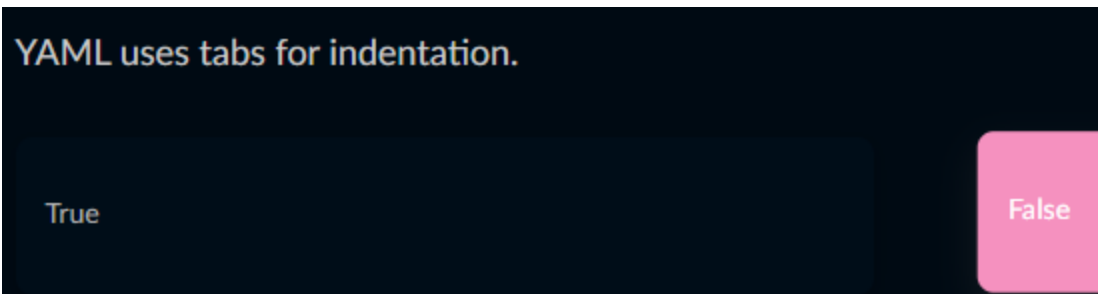
All the options

path: F:\test

path: "F\test"\programs

path: F:

All the options .



*As already discussed, Ansible is a configuration management tool, based on push-based architecture to automate configuration of your hosts to achieve a **desired state**.*

Following are the components of **Ansible architecture**

- **Inventory** - Defines the list of target hosts
- **Playbook (YAML file)** - Defines list of tasks
- **Module** - A python code invoked from tasks and executed on hosts
- **Control Machine** - Takes playbook and executes each task on particular group of hosts

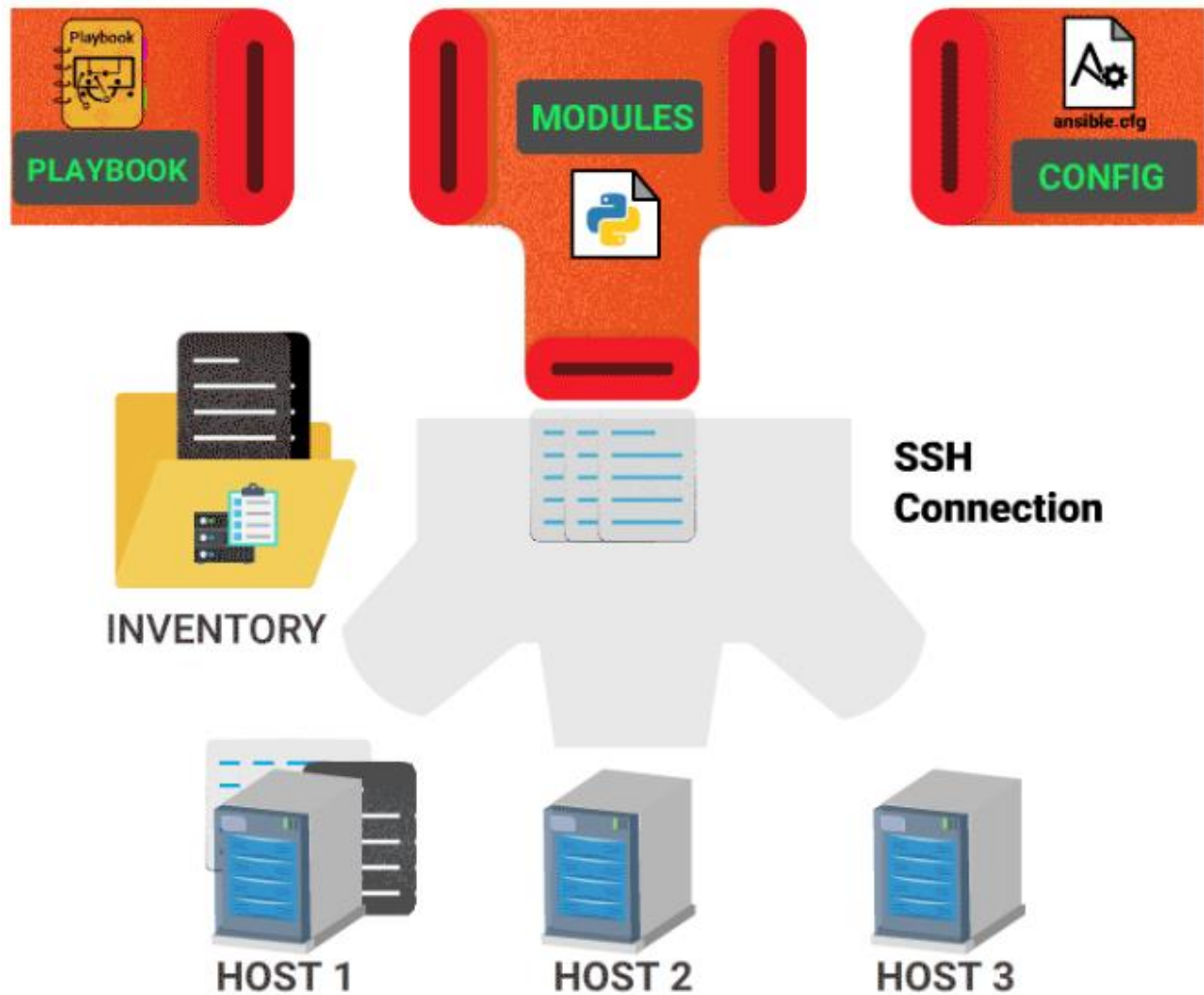
A Playbook is a file that defines the desired state of your system.

It contains plays, which has a list of tasks to run in sequence against a list of hosts.

A play is set of tasks, grouped together to achieve an objective

A task is an instruction you give to Ansible.

They are written in YAML format, a data serialization language, that we discussed in previous cards.



So Ansible captured the desired state through Playbook, but how would ansible know which machines it should configure through Inventory?

The Inventory file in Ansible contains the list of all hosts (target systems/servers) that need to be configured. You can also group hosts under different names as shown:

[group A]

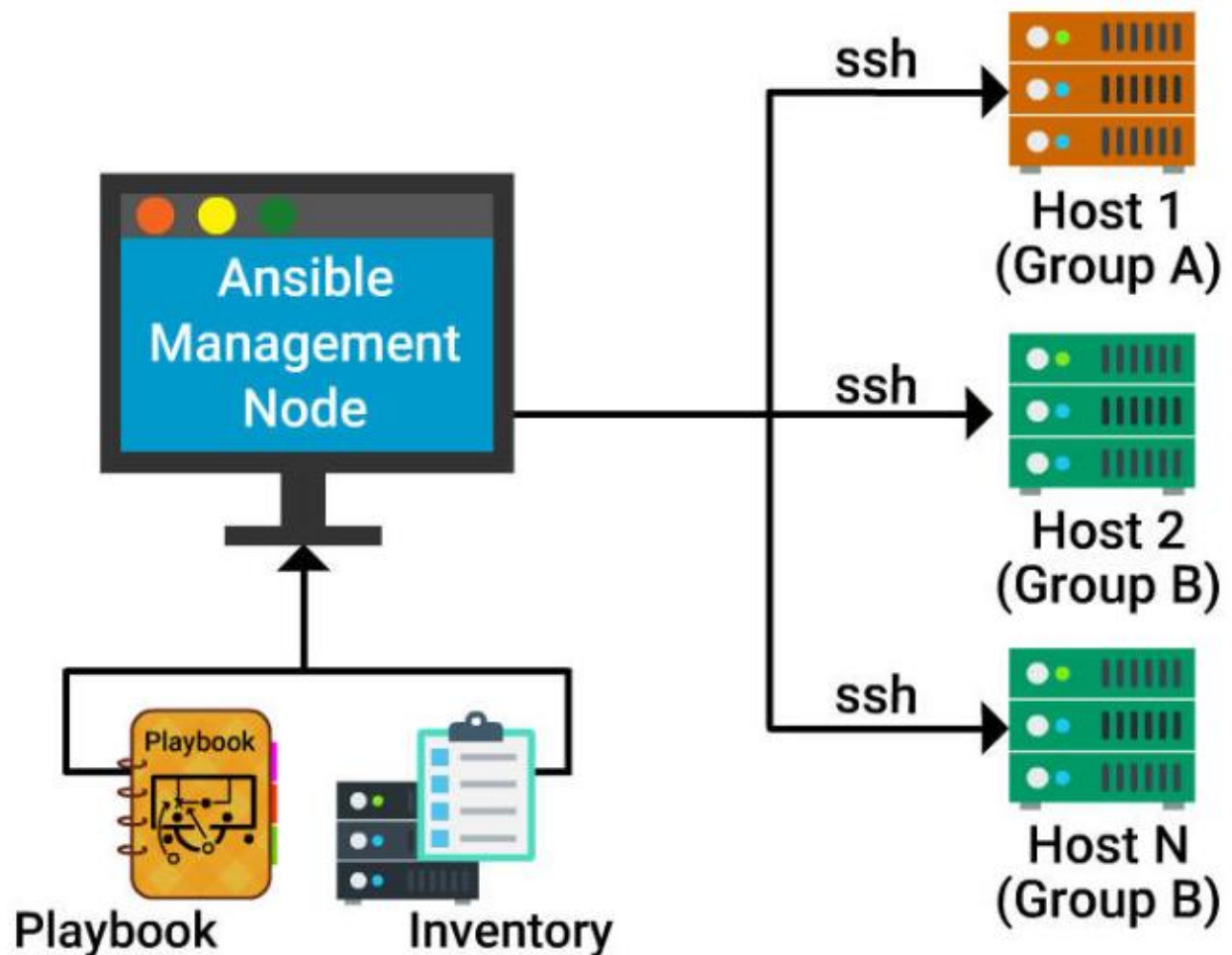
Host 1

[group B]

Host 2

Host N

- Default location of Inventory File: `/etc/ansible/hosts`
- You can define an Inventory file **statically (.ini file)** or **dynamically (.json file)** to Ansible.



Modules are a piece of code that gets executed when you run playbook. You use them to describe the state you want the host to be in.

Each task in play is made of module and arguments.

Ad-Hoc Keywords

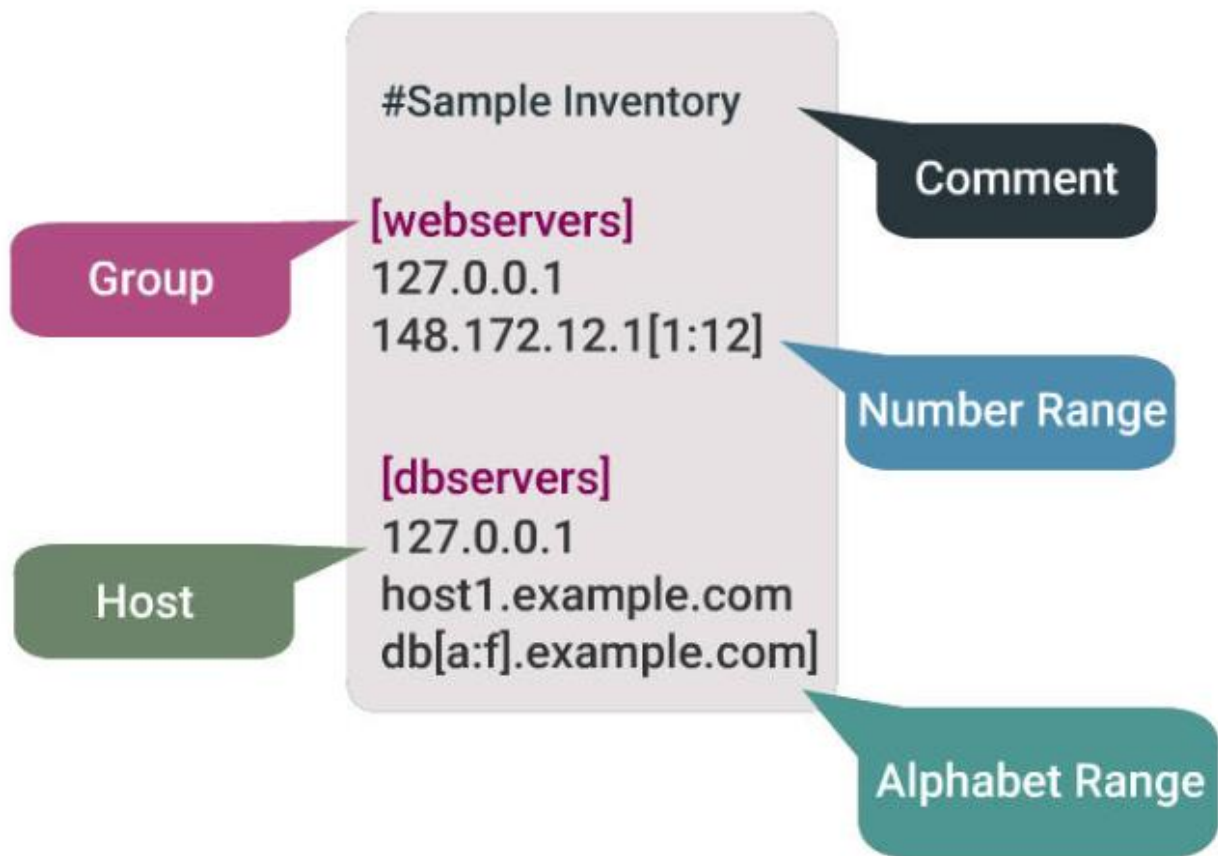
*Before hopping into Ad-Hoc commands, let us first learn **Ansible keywords**:*

- **ansible**: This is a tool that allows you to **run a single task** at a time.

```
$ ansible <host-pattern> [-m module_name] [-a args] [options]
```

- **ansible-playbook**: This is the tool used to **run ansible playbook**

```
$ ansible-playbook <filename.yml> ... [options]
```

Ad-Hoc Keywords

- **ansible-console:** *This is a REPL using which you can run ad-hoc commands on chosen inventories.*

\$ ansible-console <host-pattern> [-m module_name] [-a args] [options]

- **ansible-pull:** *This inverts the default push architecture of Ansible into a pull architecture, which has near-limitless scaling potential.*

ansible-pull -U URL [options] [<filename.yml>]

- **ansible-doc:** *Displays data on modules installed in Ansible libraries.*

\$ ansible-doc [-M module_path] [-l] [-s] [module...]

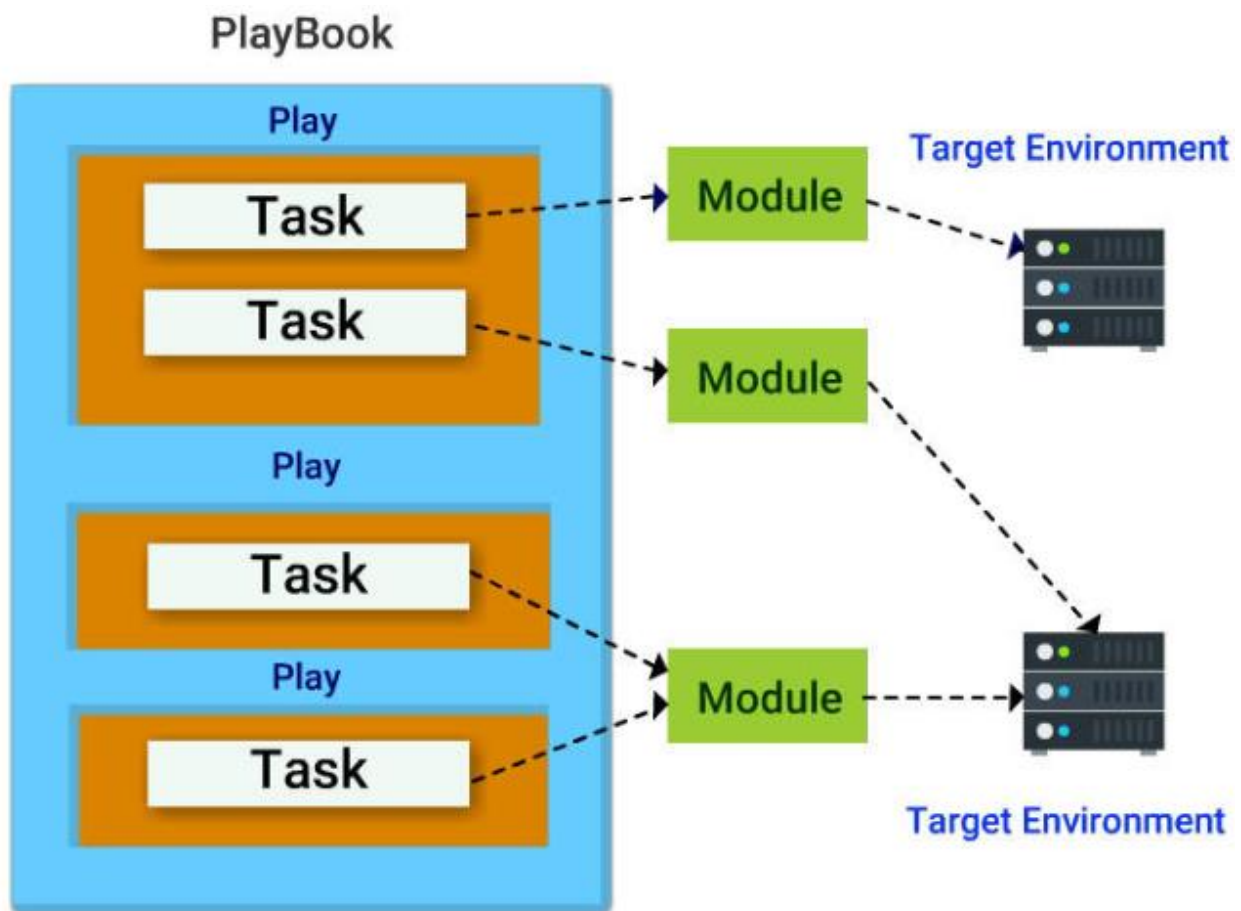
Ad-Hoc Keywords

- **ansible-vault:** *Using this you can **encrypt** any structured data file used by Ansible.*

\$ ansible-vault [create|decrypt|edit|encrypt|rekey] [--help] [options] file_name

- **ansible-galaxy:** *This is a shared **repository for Ansible roles**. This ansible-galaxy command can be utilized to manage these roles, or to create a skeleton framework for the roles to be uploaded to Galaxy.*

```
$ ansible-galaxy [delete|import|info|init|install|list|login|remove|search|setup] [--help] [options]
```



Short Hands in Ansible

- -a: This tells the **arguments** to pass to the module
- -m: Execute the **module**
- -b: Use **privilege escalation (become)**
- -i: The path to the **inventory**, which defaults to /etc/ansible/hosts
- --version: Show program **version number**
- --help: Shows help message

An ad-hoc command is a single statement to complete a particular task. For example: consider you want to check if you could connect to your hosts.

Enter the following command:

```
ansible group1 -i myhosts -m ping
```

The above statement is a single task to ping target host and return pong if the connection is successful.

ansible is a keyword you need to write before running any ad-hoc command

group1 is the group name of the list of hosts

-m means module, this is followed by the module name ping, which will be executed to achieve the task

To know more about each module you can try: `ansible-doc ping`

Copy a File to the Servers

You can use **copy module** to copy a file from your **control machine to host** as shown:

\$ touch test.txt

- This will create a sample file which could be used to copy

\$ ansible host01 -i myhosts -m copy -a "src=test.txt dest=/tmp/"

- copy the file test.txt **from your control machine** (where ansible is installed) **to all the hosts** defined in myhosts inventory group
- -a means **arguments** of that module (*here copy module*)
- src is **attribute of copy module** that defines the **source path** of file or directory **on control machine**

*Similarly, to fetch a file from Host to your Control Machine, you can use **fetch module**. You may use `ansible-doc fetch` to know about it.*

Encrypting Your File

As you just created a test.txt file, let us now encrypt the same using **ansible-vault keyword**.

- \$ ansible-vault encrypt test.txt: encrypts the file.

This asks for a password to be set. Give a password and confirm it.

- ansible-vault edit test.txt: to edit the file and add some content.

This opens vi editor. Type some text, then save it(:wq)

- cat test.txt: to view the content inside.

Observe the output carefully

- ansible-vault decrypt test.txt: to decrypt the file, use the password set during encryption
- cat test.txt: now observe the output

Create Directories and Files

You can use **file module** to create files and directories, manage their permissions and ownership as shown:

```
ansible host01 -i myhosts -m file -a "dest=/tmp/test mode=644 state=directory"
```

- This will create **directory /tmp/test** on all the host01 of *myhosts group*
- mode defines permission of file/directory
- state can take value: file, directory, link, absent, etc

You can set the state to **absent** to delete a file or directory to delete it:

```
$ ansible host01 -i myhosts -m file -a "dest=/tmp/test state=absent"
```

Automating with Ansible

Till now you were executing each task (using Ad-hoc command) to create a folder, copy a file, encrypt or decrypt a file, etc.

What if you need to execute, say, **500 tasks** to configure a server?

Do not worry, as Ansible got you.

You just need to define a Playbook that Ansible can play with, and have a popcorn watching Ansible in action.

Read on to find out what Playbook is.

Modules are blocks of Python code that gets executed on remote hosts when you run each task in Playbook.

Each module is built for a particular task and you can use arguments to change the behavior of module.

Things You Should Know About Modules

- Also referred as **task plugins** or **library plugins**
- **Default location** for Ansible modules is */usr/share/ansible*
- Take arguments in **key=value** format (*state=stopped*)
- **Returns data in JSON format**
- **Modules should be idempotent**, meaning they should not do any changes if current state of system matches with the desired final state
- To access **list of all installed modules** using command line: *ansible-doc -l*
- To see the **documentation of particular module** using command line: *ansible-doc yum* where *yum* is the module name
- You can **run modules from the command line or include them in Playbook**.
- Ansible allows you to **write your own module** (this you will learn later in advanced courses of Ansible).

Let us now go through some standard modules: apt, yum, shell, command, and template.

Which one is not a valid value of state argument of "file" module?

folder

absent

file

link

Where is Inventory file located by default?

/etc/ansible

/etc/inventory

/etc/configurations

/etc/ansible/hosts

Which module can be utilized to copy files from a remote machine to control machine?

ping

move

copy

fetch

Which command instructs Ansible to execute the playbook on all the hosts except host1?

ansible-playbook playbooks/PLAYBOOK_NAME.yml -limit
'all:!host1'

ansible-playbook playbooks/PLAYBOOK_NAME.yml --limit
"!host1"

ansible-playbook playbooks/PLAYBOOK_NAME.yml --limit
'all:!host1'

ansible-playbook playbooks/PLAYBOOK_NAME.yml --limit
"all:!host1"

By using `--limit` argument with **ansible-playbook** command we can exclude a host from playbook execution. If hostname starts with "!" it will be excluded from host execution.

APT Module

APT (Advanced Package Tool) is a command-line tool used to easily manage (install, remove, search, etc.) packages on Ubuntu/Debian based Linux systems.

Debian Based OS: Ubuntu, Kali Linux, SteamOS and much more.

```
$ ansible host01 -i myhosts -m apt -a "name=sudo state=latest"
```

#This is how you write in Playbook

```
- name: Upgrade sudo to the latest version
```

```
  apt:
```

```
    name: sudo
```

```
    state: latest
```

YUM Module

YUM (Yellowdog Updater Modified) is a command-line tool used to easily manage (install, remove, search, etc.) packages on RPM (Red Hat Package Manager) based Linux systems.

Red Hat Based OS: Fedora, Qubes OS, CentOS and much more.

#This is how you write in Playbook

```
- name: upgrade all packages
```

```
  yum:
```

```
    name: sudo
```

```
    state: latest
```

shell Module

In shell module, the command name is followed by arguments, which **runs** on remote hosts through a shell(/bin/sh).

You **can use** various operations(|,<,> etc) and environment variable(#HOME).

```
ansible host01 -i myhosts -m shell -a "echo $TERM": This displays the terminal name of host machine
```

#This is how you write in Playbook

```
- name: Execute the command in remote shell
```

```
  shell: echo $TERM
```

command Module

In command module, the command name is followed by **arguments, which does not run on remote hosts through a shell(/bin/sh).**

You **cannot use** various operations(|,<,> etc) and environment variable(#HOME).

- **Make a directory in remote host:** ansible host01 -i myhosts -m command -a "mkdir folder1"
- **Check files or folders in remote host:** ansible host01 -i myhosts -m command -a "ls"

*Now to check files or folders in **your terminal** use ls and observe the output. As you can see, **using command, you can execute tasks on remote host.***

command Everytime

Most of the Ansible modules are idempotent.

But command module does not exhibit this property as this **runs every time you run playbook.** Hence you will always find the changed status on running the same Playbook again and again.

Consider you wrote a task to copy a file to remote hosts using command module.

Ansible **command module will copy the same file the number of times you run the Playbook.**

Had this been idempotent; Ansible will not copy from the second time, as the file is already present (current state = desired state).

command can be Idempotent

To overcome this, you can use creates or remove parameter, where you define the filename/pattern.

- creates: if filename **exists**, task will not run
- remove: if filename **does not exist**, task will not run

#This is how you write in Playbook

- name: copy a file, but do not copy if the file already exists

command: cp /home/dist/file1.txt /usr/someFolder/ creates=file1.txt

Ansible template is a file having a defined structure with some variables and expressions, which can be replaced with the corresponding values to generate new configuration file.

- Template files are **written in Jinja2 language (.j2)** (*Read on to know more about Jinja2*)
- Generally, in Ansible, these files are copied to remote hosts in **JSON or YAML format**

There are basically two files involved to define templates in Ansible:

- playbook(YAML file): here you substitute variables in template file with values
- template(Jinja2 file): here you define the template file in Jinja2 format

Let us take the example of the human body as a template. Depending on the various elements (footwear, dress, hair) you could name the template as a man or a woman.

1. Ansible Sibelius Module Explained

This scenario is used to explain the usage of module in ansible.

For this you have to stop and start a service named **ssh** .

Tasks to be done:

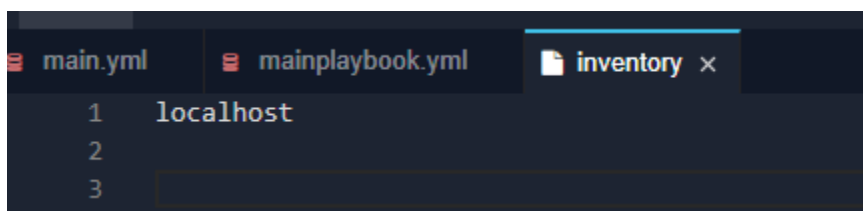
- Write a task in main.yml file present in fresco_module\tasks folder.
- the task is to stop and start the service named "ssh" using service module in ansible.

- Don't use restarted, try to stop and start the service.
- Don't use with_items iterator.

Note:

- Run project install to install ansible.
- mainplaybook.yml file is provided to ansible-playbook.
- Use the localhost for the inventory for ansible-playbook.

NOTE: use the state module to stop and start the service. DO NOT use the command module.



```
main.yml  mainplaybook.yml  inventory x
1  localhost
2
3
```

Specify Localhost in your hosts directive of your playbook.

If you are running a playbook which you want to run on localhost (or) in other words you have a playbook you want to run locally. Within the playbook Just mention `localhost` in the hosts segment where you usually specify the host group. Consider the following playbook.

I have just specified the `localhost` in the hosts directive or key.

When you run this playbook. Despite you have no entry named **localhost** in your ansible inventory file `/etc/ansible/hosts` or custom hosts file specified with `-i` option. It would work

As an additional measure that it should run in `localhost` or locally. You can add a one more parameter named `connection` and set it to `local`

```
---  
  
- name: "Playing with Ansible and Git"  
  
  hosts: localhost  
  
  connection: local
```

Add an entry in your Inventory

This is a second method to run ansible playbook locally.

You can also explicitly define your localhost in your inventory file. your inventory file can be at the default location `/etc/ansible/hosts` or a customized `ansible_hosts` file in your present working directory.

```
→ Ansible-Examples git:(master) ✗ cat ansible_hosts      Local Ansible_hosts file / Inventory  
[app]  
mwivmapp01  
mwivmapp02  
  
localhost ansible_connection=local  
→ Ansible-Examples git:(master) ✗ cat /etc/ansible/hosts Default Inventory  
[app]  
mwivmapp01  
mwivmapp02  
  
localhost ansible_connection=local  
→ Ansible-Examples git:(master) ✗
```

if you would like to add your localhost with someother alias name. like “`controlmachine`” you can do the same.

```

→ Ansible-Examples git:(master) x cat ansible_hosts
[app]
mwivmapp01
mwivmapp02

controlmachine ansible_connection=local
→ Ansible-Examples git:(master) x

```

1. become: yes = sudo
become_user: user_name = sudo -u user_name
2. become: yes
become_user: root is equivalent of become: yes
3. become_method – allows you to activate the privilege escalation,
4. become_user – allows switching to other users (shankym).
5. remote_user is the account that logs into the machine
6. If the remote_user is going to perform tasks as root or another user, set the become: option to true (at either the playbook or the task level) and the become_user: option to the name of the other user or root. If you don't specify become_user, the default is to switch to root.

```

main.yml  mainplaybook.yml x  in
1  ---
2  - hosts: localhost
3    connection: local
4    become: yes
5    become_method: sudo
6    roles:
7      - fresco_module

```

```

main  mainplaybook.yml
---
- hosts: localhost
  remote_user: root
  roles:
    - copy-files

```

```
ml  main.yml .../tasks x  main.
---
- hosts: localhost
  connection: local
  become: true
  tasks:
    - name: stop ssh
      service:
        name: ssh
        state: stopped
    - name: start ssh
      service:
        name: ssh
        state: started
```

You will usually need to set `become: true` when using the service module because `root` or `superuser` permissions are required when managing services on most systems.

```
main.yml
user@workspacekgk4od4itrmpmls0:/projects/challenge/fresco_module/tasks$ ansible-playbook main.yml

PLAY [localhost] *****

TASK [Gathering Facts] *****
[DEPRECATION WARNING]: Distribution Ubuntu 16.04 on host localhost should use
/usr/bin/python3, but is using /usr/bin/python for backward compatibility with prior
Ansible releases. A future Ansible release will default to using the discovered
platform python for this host. See
https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html
for more information. This feature will be removed in version 2.12. Deprecation
warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.
ok: [localhost]

TASK [stop ssh] *****
changed: [localhost]

TASK [start ssh] *****
ok: [localhost]

PLAY RECAP *****
localhost                : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0

user@workspacekgk4od4itrmpmls0:/projects/challenge/fresco_module/tasks$
```

How to start a service

Set the `name` parameter to the service name and the `state` parameter to `started` to start a service. If the service is already running, Ansible will do nothing.

```
- name: ensure nginx service is started
  service:
    name: nginx
    state: started
  become: true
```

How to stop a service

Set the `name` parameter to the service name and the `state` parameter to `stopped` to stop a service. If the service is not running, Ansible will do nothing.

```
- name: ensure nginx service is stopped
  service:
    name: nginx
    state: stopped
  become: true
```

How to restart a service

Set the `name` parameter to the service name and the `state` parameter to `restarted` to restart a service.

```
- name: restart the nginx service
  service:
    name: nginx
    state: restarted
  become: true
```

JINJA2 :

Used to create templates .

How we set variable ?

```
{% set value=100 %}

{% if value == 100 %}
    <p>{{ value }}</p>
{% else %}
    <p> Value is not 100 </p>
{% endif %}

{% set host_list = ['host1','host2','host3'] %}

{%for host in host_list %}
{{ host }}
{% endfor %}

Reference URL: http://jinja.pocoo.org/
```

file: sample-playbook.yml

This is where your template and variables are merged.

Let us consider another example of **Payslip as a template**, where a structure of payslip is pre-defined. Depending on the values being passed new payslip is generated for each name.

#This is how you write in Playbook

- hosts: all

vars:

quarter: false,

salary : 30000,

extra : 10000,

names: ["John","David"]

tasks:

- name: Ansible Template

template:

src: ../templates/sample-template.j2

dest: /home/sample-template

- src: defines path where your template file is kept.
- dest: path where you want to copy your file (mostly JSON or YAML formatted file).

file: sample-template.j2

This is where you define your template.

```
{% for name in names %}
```

```
Hi {{name}}!
```

```
{% if quarter -%}
```

```
    Your pay cheque for this month is {{ salary + extra }}.
```

```
{% else -%}
```

```
    Your pay cheque for this month is {{ salary }}.
```

```
{%- endif %}
```

```
{% endfor %}
```

variable names are given within double curly braces {{ }}

Hands-on Template

You can test your Jinja2 templates [online](#) with the steps as explained.

- Copy sample-template.j2 file and paste in **Template** box
- Copy **variables** from sample-playbook.yml and paste in **Values** box
- Click on **Convert**

Try It Out - Template

- **Try to make a template to display Students mark sheet that displays name of student, marks in Physics, Chemistry and Maths.**
- **Provide relevant values via variables.**

Modules are tentatively stored in the nodes and interact with the control machine via a protocol over the standard output.

JSON

XML

The handler runs only once even if it is called multiple times from a play.

True

False

Ansible modules apply the changes every time one runs Playbook.

False

True

13. Which module copies dynamically generated file from control machine to target hosts?

template