**ANSIBLE**

**Installation**:

**UBUNTU Installations**:

Apt install ansible

Ansible –version

Using PIP:

Apt-get install python-minimal virtualenv python-dev build-essential

Mkdir ansible

Cd ansible

Virtualenv venv27

Source venv27/bin/activate

Pip install ansible

Another method using git:

Pip install git+https://github.com/ansible/ansible

**RHEL and Centos installations:**

Creating own rpm:

Sudo yum install asciidoc python-jinja2 pyYAML git python-setuptools rpm-build python2-devel

Cd /tmp

Git clone [git://github.com/ansible/ansible.git](https://github.com/ansible/ansible.git)

Cd ansible

Make rpm

Find rpm file inside rpm-build directory inside /tmp and install it using rpm -ivh <name>.rpm

Using EPEL:

Yum install epel-release

Yum install ansible

Virtualenv and pip:

Yum groupinstall ‘Development Tools’

Yum install python-virtualenv libffi-devel openssl-devel python-cffi pyYAML

Mkdir ansible

Cd ansible

Virtualenv venv27

Source venv27/bin/activate

Pip install --upgrade setup-tools

Pip install ansible

Another method using git:

Pip install git+https://github.com/ansible/ansible

**ANSIBLE Configuration**:

Config file precedence:

1 – ANSIBLE\_CONFIG

2 - ./ansible.cfg

3 - ~/.ansible.cfg

4 - /etc/ansible/ansible.cfg

One more default file to be considered in ansible config is hosts file which is in /etc/ansible/hosts

Its better not to use /etc/ansible/hosts and to create our own inventory file which will have host details.

We have to create hosts file in /home/ansible

We have to create ansible.cgf file in /home/ansible

ansible all -m ping 🡪 to pings hosts under group all.

ssh-keygen -H -F 192.168.10.51 🡪 to get known\_hosts file content format of added host

ANSIBLE\_HOST\_KEY\_CHECKING=False 🡪 set this env variable to tell ansible not to check for host key while doing ssh.

If we don’t define a group for host in ansible hosts file then it will be taken by default as part of all group.

ansible all -i 192.168.10.51, -m ping 🡪 command to run if we don’t define inventory of host in ansible configuration.

ansible all -m debug 🡪 this pings host and displays default msg

ansible all -m debug --args='msg="This is a custom message"' 🡪 pings host and prints custom message

ansible ubuntu -m ping -o 🡪 use -o to suppress output to one line

**ANSIBLE ARCHITECTURE & DESIGN:**

**ANSIBLE INVENTORIES**:

To connect to host as root user use ansible\_user=root after hostname in host file.

To sudo in connected host use ansible\_become=true and for sudo password use ansible\_become\_pass=password.

If we are using different ssh ports then use hostname:<portnum> in host file.

jumper ansible\_port=5555 ansible\_host=192.0.2.50 🡪 if we are using static IPs instead of hostnames then we can define an alias name to ip and different ssh port.

Use ansible\_connection to set connection type we are using to connect host.

We can use ranges to supply host variables to multiple hosts using single line in host file 🡪 ubu[1:3] sets value for 3 hosts like ubu1,ubu2 and ubu3

Instead of defining variables for each host we can define group variables which is applicable to all group members 🡪 [ubuntu:vars] now all members of ubuntu group will have this variables defined.

We can use group of groups to categorize the hosts 🡪 [linux:children]

To define a host variable for all hosts irrespective of groups 🡪 [all:vars]

There is precedence order for variables where hostwise variables are considered as top precedence if defined.

We can use yaml,json or standard format for hosts file in ansible.

We can use -e option with ansible command to override variables 🡪 ansible linux -m ping -e ‘ansible\_port=22’

**ANSIBLE MODULES**:

Setup Module:

This is automatically called by playbooks to gather useful information/variables about remote host that can be used in playbooks.

Ansible cent1 -m setup 🡪 to get details/variables of remote host

File Module:

Sets attributes of files,symlinks and directories or remove them.

Even copy,template and assemble modules have the same options as file.

Ansible all -m file -a ‘path=/tmp/test state=touch’ 🡪 to create file in remote host,here module is file and arguments are path and state.

Ansible all -m file -a ‘path=/tmp/test state=file mode=600’ 🡪it will change the file permissions to 600,state=file means it will not create a file if it doesn’t exist,it can change only existing file.

Ansible has idempotent behavior where it will not try to change the file if it is already in desired state of current performing step.

Copy Module:

This is used to copy file from local to remote or remote to remote,use fetch module to copy files from remote to local

Ansible all -m copy -a ‘src=/tmp/x dest=/tmp/x’ 🡪 to copy file from local to remote,by default it is success folocalhost because of idempotent behavior.

Ansible all -m copy -a ‘remote\_src=yes src=/tmp/x dest=/tmp/y’ 🡪 to copy files from src to dest on remote host.

We can use content instead of src to set content of dest with content value.

Command Module:

It takes command name followed by space delimited arguments to execute on remote hosts,It doent execute in shell so use shell module to execute shell commands.

Ansible all -m command -a ‘hostname’ -o 🡪 to execute hostname on all remote hosts.

Command module is the default module of ansible.

Chdir is used to change directory for the command execution.

ansible all -m command -a 'touch /tmp/test-copy-module creates=/tmp/test-copy-module' 🡪 creates a file and creates option will not create a file if it is already exists.

ansible all -m command -a 'rm /tmp/test-copy-module removes=/tmp/test-copy-module' 🡪 removes a file and removes will not remove is file doesn’t exists.

ansible cent1 -m fetch -a ‘src=/tmp/test\_module dest=/tmp/test\_module’

**YAML**:

Every YAML file starts with --- and ends with … .

We can use double,single and no quotes in yaml.

If we are using special characters like \n then use double quotes.

Use | symbol to define multiline strings.

Use > to display multiple line string in yaml to single line .

Use >- for same fuctionality like > but here it doesn’t append \n to the end of string.

Use – at front to represent a list.

We can directly define a dictionary in yaml using {} and grouping all elements.

We can directly define a dictionary in yaml using [] and grouping all elements.

**ANSIBLE PLAYBOOKS,SECTIONS**:

Regions of playbook 🡪 Targets,Variables,Tasks,Handlers,Roles.

Playbook will have list of plays where play is a dictionary.

Blocks are group of tasks in a playbook.

By default setup module will run as a task in every play.

Example target parameters 🡪 sudo,user,sudo-user,connection,gather\_facts.

{{ motd }} 🡪 use curly braces to use value of variables inside playbook.

ansible-playbook motd-cent.yml -e 'motd="This is example"' 🡪 we can use -e directly with ansible-playbook command to define variables.

Use when to compare system variable with our condition.

**ANSIBLE PLAYBOOKS,VARIABLES**:

Variables can be defined as dictionaries inside a dictionary and can be access using . notation.

Vars:

Dict:

Dict\_key: “Dict value”

This can be used as dict.dict\_key.

Variables can be defined as a list as well.

vars:

n\_list:

- item1

- item2

- item3

- item4

N\_list.0 can be used for item1 and similarly for others.

We can define variables in external yaml file and can refer that file inside playbook for variable values.

Vars\_files:

* Ext\_v.yml

We can use vars\_prompt to prompt for variable inputs.

Vars\_prompt:

* Name: example

Private: false

By default value is not displayed while typing so we need to set private to false for variable.

We can use hostvars inside playbook to get the facts of remote host.

Hostvars[ansible\_hostname].ansible\_port 🡪 gives ssh port of host.

By default above command don’t work for default port 22 because it will be commented in sshd file so we need to use default(‘’) so that if some value is not available then default value will get displayed.

| in jinja 2 adds a filter and we can pass any value using default().

We can call group variables directly to get values of group hosts in playbook 🡪 ansible\_user

Groupvars also can be accessed via hostvars. 🡪 hostvar[ansible\_hostname].ansible\_user

We can separate groupvars and hostvars from hosts file by creating separate yaml files under 2 directories named group\_vars and host\_vars for each host and group.

Best practice is to use host\_vars and group\_vars for storing variables.

We can define extra vars with the use of -e and can be used them inside playbook.

We can use -e @extar\_var\_file.yml to get variables from external file.

**ANSIBLE PLAYBOOKS, FACTS**:

Ansible cent1 -m setup -a ‘gather\_subset=network’ 🡪 will display the details of network config of cent1 host.

ansible cent1 -m setup -a 'gather\_subset=network,!all,!min' | more 🡪 to get minimum network information.

We can use filter to get particular info 🡪 ansible cent1 -m setup -a 'filter=ansible\_mem\*'

If we run setup module then we will get ansible fact dictionary key whose values are configurations of host. So by default all values of ansible\_facts for all hosts will be saved by ansible for there usage in playbook.

Ansible\_default\_ipv4.address 🡪 to get ip address of a host.

We can create custom facts for a host which will be gathered during gather\_facts task and they are stored in /etc/ansible/facts.d

We can create custom facts using json and ini format. I prefer ini.

Create /etc/ansible/facts.d and copy created getdate1.fact custom facts.

Our custom facts will be under ansible\_local in output of setup module.

ansible-playbook fact-playbook.yml -l ubuc 🡪 to limit playbook to certain host.

By default custom facts goes to hostvars 🡪 hostvars[ansible\_hostname].ansible\_local.getdate1.date.date

Use state: directory and recurse: yes with file module which is equivalent to mkdir -p .

Always call setup module again in tasks if you are dealing with custom facts.

We can use fact\_path option with setup module to redefine custom fact location if we don’t have root access on remote hosts.

**TEMPLATING WITH JINJA2**:

Everything is inside {} in Jinja 2.

For comments use # with brackets.

For code use % with brackets.

If we start if statement then end it with endif.

msg: >

--== Ansible Jinja Assignment==--

{# If the hostanem is ubuc include a message -#}

{% if ansible\_hostname == "ubuc" -%}

This is ubuntu-c

{% elif ansible\_hostname == "cent1" -%}

This is centos 1

{% else -%}

This is good old {{ ansible\_hostname }}

{% endif %}

To check whether variable is defined.

msg: >

--== Ansible Variable check using JINJA ==--

{% if example\_variable is defined -%}

example\_variable is defined

{% else -%}

example\_variable is not defined

{% endif %}

Use set to set variables.

msg: >

--== Ansible Variable check using JINJA ==--

{% set example\_variable = 'defined' -%}

{% if example\_variable is defined -%}

example\_variable is defined

{% else -%}

example\_variable is not defined

{% endif %}

Using for to get IPs.

msg: >

--== Ansible JINJA For loop ==--

{% for entry in ansible\_all\_ipv4\_addresses -%}

IP Address Entry {{ loop.index }} = {{ entry }}

{% endfor %}

Range:

msg: >

--== Ansible JINJA For loop ==--

{% for entry in range(1,11) -%}

{{ entry }}

{% endfor %}

Reverse Range:

We cannot use break and continue by default in ansible so we need to add below config in ansible.cfg.

jinja2\_extensions = jinja2.ext.loopcontrols

msg: >

--== Ansible JINJA For loop ==--

{% for r in range(10, 0, -1) -%}

{% if r == 5 -%}

{% break %}

{% endif -%}

{{ r }}

{% endfor %}

Entry is ODD:

msg: >

--== Ansible JINJA For loop ==--

{% for r in range(10, 0, -1) -%}

{% if r is odd -%}

{% continue %}

{% endif -%}

{{ r }}

{% endfor %}

We can use filters with JINJA2

We can use | after variable to apply filters such as min,max,unique,random.urlsplit,difference

{{ var }} | <filter>

<https://docs.ansible.com/ansible/2.5/user_guide/playbooks_filters.html>

Template Module:

We can use template module to use jinja2 template.

Template is used to separate jinaj2 code from playbook, Instead of typing jinja code in playbook we can place code in template and we can use template in playbook.

Use template to place jinja code and refer to template in playbook so that it is ran over all hosts and we can save output using dest in all hosts.

If you use ansible variable as a value to some other variable then use “” to define the variable like below.

dest: "/tmp/{{ ansible\_hostname }}\_template.out"

**ANSIBLE PLAYBOOKS CREATING AND EXECUTING**:

We can use yum and apt modules to install some package with update\_cahe,state and name.

We can also use package module to install packages on ubu and cent.

Use patch module to patch a file like patch command.

We can create symlinks using state=link with file module.

Use service module with state=restarted to restart some service on remote host.

Use uri module with url and status\_code to test.

By default firewall is not opened in some distributions in that case run firewalld module with service,permanent,state and run handler to reload service with httpd and state as reloaded,

Use ansible\_managed variable in templates of ansible to tell that the template is managed by ansible and we should not edit it manually because ansible might overwrite it and define ansible\_managed variable in ansible.cfg file.

We can define image variables in group vars and can use them in templates as image sources with jinja 2 code if needed.

**ANSIBLE PLAYBOOKS,ADVANCED TOPICS:**

**ANSIBLE PLAYBOOK MODULES**:

Set fact module is used to dynamically add or change facts of host in playbook.

Use set\_fact module with our\_fact to set a fact.

We can change existing system facts with set\_fact like capitalizing existing fact value {{ ansible\_distribution | upper }}

Pause module will pause playbook execution for some time.

Use pause module with seconds option.

We can use pause module if there are any tasks which neede to be checked manually before proceeding with others.

Use prompt with pause to prompt a message and hit enter to proceed.

We can use wait\_for module if we are waiting for something before executing some tasks.

Use wait\_for and port option if we are looking for some service ona port.

We can use assemble module with src and dest to assemble multiple files to a single file.

Add\_host module allows us to dynamically add host to our module.

Use add\_host with name and groups.

Group\_by is similar to add\_host but here it adds hosts based on key.

Use group\_by with key.

We can use fetch to get files from remote host.

**DYNAMIC INVENTORIES**:

Instead of using host inventories we can dynamic inventory file.

It can be generated using an executable file written in any language.

I prefer creating a python script for generating dynamic inventories.

If we print using inventory script then sjon structure will break but ansible excepts json input from output of script.

If we enable logger module in script then it will store content in a file.

If we have large number of hosts then use dynamic inventory script with include host metadata as true.

Usually scriptis called for each and every host for getting vars by ansible which is not good.

So if we enble to store metadata as list output then it works efficiently.

**REGISTER AND WHEN**:

It is used to register output and use it later.

Registering is nothing but saving output to a variable and using it later.

- name: Registerting output

command: "hostname -s"

register: hostname\_output

- name: Print registered output

debug:

var: hostname\_output

Use var with debug to display registered outputs.

We can use when to filter hosts based on multiple facts using and between them.

when: ansible\_distribution == "CentOS" and ansible\_distribution\_major\_version == "7"

We can use or between two conditions to make more flexible like below.

( ansible\_distribution == "CentOS" and ansible\_distribution\_major\_version | int >= 7 ) or ( ansible\_distribution == "Ubuntu" and ansible\_distribution\_major\_version | int >= 16 )

We can define conditions as list instead of giving them in brackets.

We can run when condition based on changed value of registers.

when: command\_register is changed

when: command\_register is skipped

**LOOPING**:

We can use item loop in our playbooks like below.

- name: Changing the motd

copy:

content: "Welcome to {{ item }} Linux , Ansible Rocks"

dest: /etc/motd

notify: MOTD Changed

with\_items: [ "CentOS" , "Ubuntu" ]

when: ansible\_distribution == item

It will loop through with\_items and gives respective result.

We can pass items as list as well in yaml approach like below.

with\_items:

- "CentOS"

- "Ubuntu"

We will frequently use item looping while creating users in remote hosts.

- name: Creting user

user:

name: "{{ item }}"

with\_items:

- mallrev

- revyuv

- yuvsin

We can state : absent with user module to remove them.

We can create user using with\_dict like below.

- name: Creating user

user:

name: "{{ item.key }}"

comment: "{{ item.value.fullname }}"

with\_dict:

mallrev:

fullname: Mallarapu Revanth

revyuv:

fullname: Mallarapu Yuv

yuvsin:

fullname: Yuvraj

We can use subitems to create users like below.

Where first item is outer loop and second one is inner one like below.

- name: Creating user

user:

name: "{{ item.1 }}"

comment: "{{ item.1 | title }} {{ item.0.surname }}"

state: absent

with\_subelements:

- families:

surname: mallarapu

members:

- Revanth

- Rohit

- members

With\_nested loops will nest between 2 lists. For every item of first list we have all second list items.

- name: Creating user directories

file:

path: "/home/{{ item.0 }}/{{ item.1 }}"

owner: "{{ item.0 }}"

group: "{{ item.0 }}"

state: directory

with\_nested:

- [ Revanth , Rohit ]

- [ photos , videos , documents ]

With\_together will map one item to one in two lists based on same index.

With\_file will just copy the contents of file.

We can use with\_sequence to create sequence of something like below example.

- name: Creating directories with sequence

file:

path: "{{ item }}"

state: directory

with\_sequence: start=0 end=100 stride=10 format="/home/mallrev/sequence\_%d"

Use start as 0 and end as 16 and use %x instead of %d to get hexadecimal output.

We can use count=5 instead of start and stop to do it for 5 times.

We can use with\_random\_choice to slect random item.

- name: Creating directories with sequence

file:

path: "/home/mallrev/{{ item }}"

state: directory

with\_random\_choice:

- "google"

- "apple"

- "nokia"

We can use until with scripts until we get desired output like before.

- name: executing scripts

script: random.sh

register: result

retries: 100

until: result.stdout.find("10") != -1

**ASYNCHRONOUS AND PARALLEL**:

Usually each task will get executed on all nodes one by one. So ssh connections to all nodes will be open until the task is completed on all nodes. If we use async with poll then task will get executed on all nodes at a time.

If we run tasks asynchronously we will not come to know whether it is successful or not so always register the output and check output file and use async\_status module like below.

- name: Capture job ids

set\_fact:

jobids: >

{% if item.ansible\_job\_id is defined -%}

{% jobids + item.ansible\_job\_id -%}

{% else -%}

{% jobids -%}

{% end if %}

with\_items: "{{ [ result1, result2, result3, result4, result5, result6 ] }}"

- name: Show Jobids

debug:

var: jobids

- name: Wait for JIDs

async\_status:

jid: "{{ item }}"

with\_items: "{{ jobids }}"

register: jobs\_result

until: jobs\_result.finished

retries: 30

By default ansible will have 5 forks that is it can do things on 5 hosts parallelly we can increase this forks value in ansible.cfg.

We can use serial in playbook that means it will execute task at a time on that many hosts.

We can use serial as a list that means first time it will run on first value number of hosts and next time next value number of hosts like that.

We can use serial as percentages as well.

We can use strategy: free to run playbooks asap.

**TASK DELEGATION**:

Add all server ips and hostname in /etc/hosts of dnsmasq.

Install hostman in dnamasq server.

Add dnsmasq to host inventory on ubuc.

We can use delegate\_to to run that task on particular hist on behalf of this host.

- name: Add dynamic hostrule

command: "/usr/local/bin/hostman -i dynamic\_{{ ansible\_hostname }}:{{ ansible\_default\_ipv4.address }}"

delegate\_to: dnsmasq

- name: reload dnsmasq

service:

name: dnsmasq

state: reloaded

delegate\_to: dnsmasq

run\_once: true

We can use run\_once to run only once on delegated host if necessary.

We can use play\_hosts variable which will have all targeted hosts of playbook.

**MAGIC\_VARIABLES**:

We can fetch all magic variables to host like below.

- name: copy template to dest

template:

src: templates/dump\_variables

dest: /tmp/ansible\_variables

root@ubuc:~/ansible/virtual# cat templates/dump\_variables

PLAYBOOK VARS (Ansible Vars):

{{ vars | to\_nice\_yaml }}

**BLOCKS**:

Block is nothing but grouping of multiple tasks in to a block.

tasks:

- name: Attempt and graceful roll back demo

block:

- debug:

msg: 'I execute normally'

- command: /bin/false

- debug:

msg: 'I never execute, due to the above task failing'

rescue:

- debug:

msg: 'I caught an error'

- command: /bin/false

- debug:

msg: 'I also never execute :-('

always:

- debug:

msg: "This always executes"

If a task in block fails for a host then execute will run for that host.

**ANSIBLE VAULT**:

It is used to encrypt variables and files.

We can use this to encrypt sensitive information.

ansible-vault encrypt\_string --ask-vault-pass --name "ansible\_become\_pass" "password"

If we use vault then we need to pass –ask-vault-pass to make it work.

Encrypt files using 🡪 ansible-vault --ask-vault-pass encrypt external\_vault\_vars.yml , use decrypt instead of encrypt to decrypt it.

Use rekey to change password for a file or content. 🡪 ansible-vault --ask-vault-pass rekey external\_vault\_vars.yml.

We will have multiple passwords for multiple content and files so we need to pass all to make playbook successful.

Echo “<encrypted string>” | ansible-vault decrypt - 🡪 to decrypt a secret.

Use ansible-vault --vault-id @prompt external\_vault\_vars.yml to use password which is saved in a password\_file.

ansible-vault --vault-id @password\_file decrypt external\_vault\_vars.yml 🡪 to decrypt

We can use different vaults for different purpose like below using –vault-id.

ansible-vault --vault-id vars@prompt encrypt external\_vault\_vars.yml

ansible-vault --vault-id ssh@prompt encrypt\_string --name 'ansible\_become\_pass' 'password'

Now our playbook requires 2 vault passwords for vars and ssh.

We can use below command to give multiple passwords.

ansible-playbook --vault-id vars@prompt --vault-id ssh@prompt vault-playbook.yml.

We can also encrypt entire playbook.

ansible-vault --vault-id playbook@prompt encrypt vault-playbook.yml

Run it using below command.

ansible-playbook --vault-id vars@prompt --vault-id ssh@prompt --vault-id playbook@prompt vault-playbook.yml

**CREATING CUSTOM MODULES**:

We can create our own modules to use with ansible.

First we need to get ansible source code to local.

Clone the ansible repo.

Shell Script Tip - ${target:-127.0.0.1} here if target value is not defined then 127.0.0.1 is taken as default value.

Write a shell script and place it under library directory from where we execute ansible commands.

Icmp.sh

#!/bin/bash

source $1 &>/dev/null

TARGET=${target:-127.0.0.1}

ping -c 1 ${TARGET} &>/dev/null

if [ $? -eq 0 ]

then

echo "{\"changed\": true, \"rc\": 0}"

else

echo "{\"failed\": true, \"msg\": \"failed to ping\", \"rc\": 1}"

then test that icmp using test module from ansible source code.

~/src/ansible/hacking/test-module -m ./icmp.sh -a 'target=128.0.0.1'

Then use icmp as module directly in playbook if testing is successful.

- name: Test ICMP model

icmp:

target: 127.0.0.1

**CREATING PLUGINS**:

Create a directory lookup\_plugins in the path and download particular existing plugin python script from github ansible source code and modify according to our requirement and use it in the playbook as a plugin like with\_items etc..

**STRUCTURING ANSIBLE PLAYBOOKS**:

**USING INCLUDES AND IMPORTS**:

Includes is a module which can be used to combine tasks which are written in a diff yaml file and it can be used to combine 2 playbooks as well.

- name: Play 1 - Task 1

debug:

msg: Play 1 - Task 1

- include: play1\_task2.yml

(venv27) root@ubuc:~/ansible/virtual# cat play1\_task2.yml

---

- name: Play 1 - Task 2

debug:

msg: Play 1 - Task 2

...

Using single – include is deprecated in newer version so intead use include\_tasks and import\_playbooks like below.

- name: Play 1 - Task 1

debug:

msg: Play 1 - Task 1

- include\_tasks: play1\_task2.yml

- import\_tasks: play1\_task3.yml

- import\_playbook: play2.yml

Import\_tasks is static in nature and include\_tasks is dynamic in nature.

For all modules for first run it will negate when condition.

For include\_tasks when is executed only once and if it meets all the tasks will run but for import all tasks run independently with when(same way for playbook but include\_playbook is not there for playbooks)

Include\_tasks or import\_playbook is best practice.

We can loop through include\_tasks using with\_items list like below.

---

-

hosts: cent3

tasks:

- include\_tasks: include\_tasks.yml

with\_items: [1, 2, 3]

...

---

- name: Subtasks

debug:

msg: "Sub Task: {{ item }}"

...

**USING TAGS**:

We can use tags only if we want to run particular tasks from a playbook using –tags.

We can use –skip-tags to skip particular tasks from playbook.

We can use tags with tasks and we can also define tags under variable section.

All the sub-tags will have main tag.

Special tags like always will be executed always even we define list specific tasks to run using tags.

We can skip always tag using –skip-tags.

Tagged tag will run all tasks which are tagged.

Untagged tag will run all tasks which are not tagged.

By default ansible will run as all –tags: all that is running all tasks in playbook.

Tags are inherited from other playbooks or tasks if we are using include or import.

**USING ROLES**:

Roles are used for easier manageability in larger projects.

Role are used for grouping tasks based on the aim.

We can have different roles like webserver-role ,dns-role etc.

Roles can have dependency on other roles.

ansible-galaxy init nginx 🡪 to create an ansible role.

Move everything from full playbook to respective sections of role.

Then place all config like hosts,group\_vars,host\_vars,main playbook and refer role inside main playbook like below.

We can use roles for reusability.

We can separate nginx and webpage thing using two roles.

We can define any file directories as variables in default section of a role and can use that variable in tasks directly.

Source directories can be defined in main playbooks as below.

roles:

- nginx

- { role: awesomeweb, target\_dir: /tmp }

We can define dependency roles in meta/main.yml dependency section and we can use only dependent role in main playbook.

**AWS with ANSIBLE:**

Below is the module to create security group in aws.

- name: Create a security group for ansible

ec2\_group:

name: ansible

description: Ansible Security Group

region: ap-south-1

rules:

- proto: tcp

from\_port: 80

to\_port: 80

cidr\_ip: 0.0.0.0/0

- proto: ssh

from\_port: 22

to\_port: 22

cidr\_ip: 0.0.0.0/0

Install boto and boto3 because it is required if we are dealing with ansible and aws 🡪 pip install boto boto3

Below is the module to create ec2 instances and creating a group in ansible.

- name: Provision Instances

ec2:

key\_name: myec2keypair

group: ansible

instance\_type: t2.micro

image: ami-5b673c34

region: ap-south-1

wait: true

exact\_count: 3

count\_tag:

Name: AnsibleNginxWebservers

instance\_tags:

Name: Ansible

register: ec2

- name: Add all instances public ips to host group

add\_host:

hostname: "{{ item.public\_ip }}"

groups: ansiblehosts

with\_items: "{{ ec2.instances }}"

- name: Show group

debug:

var: groups.ansiblehosts

We can use dynamic inventories with AWS.

wget <https://raw.githubusercontent.com/ansible/ansible/devel/contrib/inventory/ec2.py>

wget <https://raw.githubusercontent.com/ansible/ansible/devel/contrib/inventory/ec2.ini>

export EC2\_INI\_PATH=inventory/ec2.ini

We should update inventory in ansible.cfg to dynamic inventory script.

We need to create set of config under group vars like below.

tag\_Name\_Ansible

---

ansible\_ssh\_private\_key\_file: ~/.ssh/ansible\_aws.pem

ansible\_user: ec2-user

ansile\_become: true

...

**DOCKER SUPPORT WITH ANSIBLE**:

Find below playbook for installing docker.

tasks:

- name: Install yum utils

yum:

name: yum-utils

state: latest

- name: Install device-mapper-persistent-data

yum:

name: device-mapper-persistent-data

state: latest

- name: Install lvm2

yum:

name: lvm2

state: latest

- name: Add Docker repo

get\_url:

url: https://download.docker.com/linux/centos/docker-ce.repo

dest: /etc/yum.repos.d/docer-ce.repo

become: yes

- name: Enable Docker Test repo

ini\_file:

dest: /etc/yum.repos.d/docer-ce.repo

section: 'docker-ce-test'

option: enabled

value: 0

become: yes

- name: Install Docker

package:

name: docker-ce

state: latest

become: yes

- name: Start Docker

service:

name: docker

state: started

Below is playbook for docker practice.

---

- name: Docker Containers

hosts: cent3

tasks:

- name: Pull rastasheep's ubuntu-sshd image

docker\_image:

name: rastasheep/ubuntu-sshd

- name: Running container

docker\_container:

name: sshdserver

image: rastasheep/ubuntu-sshd

ports:

- "2222:22"

command: /usr/sbin/sshd -D

- name: Adding a Host

add\_host:

hostname: docker-container

ansible\_host: 192.168.10.45

ansible\_port: 2222

ansible\_user: root

ansible\_password: root

-

hosts: docker-container

roles:

awesomeweb

-

hosts: cent3

tasks:

- name: Create an image from existing container

command: "docker commit -c 'CMD[\"nginx\",\"-g\", \"daemon off;\"]' sshdserver ansible:awesomeweb:latest"

...

**OTHER ANSIBLE RESOURCES**:

**TROUBLESHOOTING ANSIBLE**:

We can use ssh -v to see what is happening during ssh.

Use the below command to check syntax.

ansible-playbook docker-playbook.yml --syntax-check.

Use below command to run playbook step by step based on tasks.

ansible-playbook docker-playbook.yml –step.

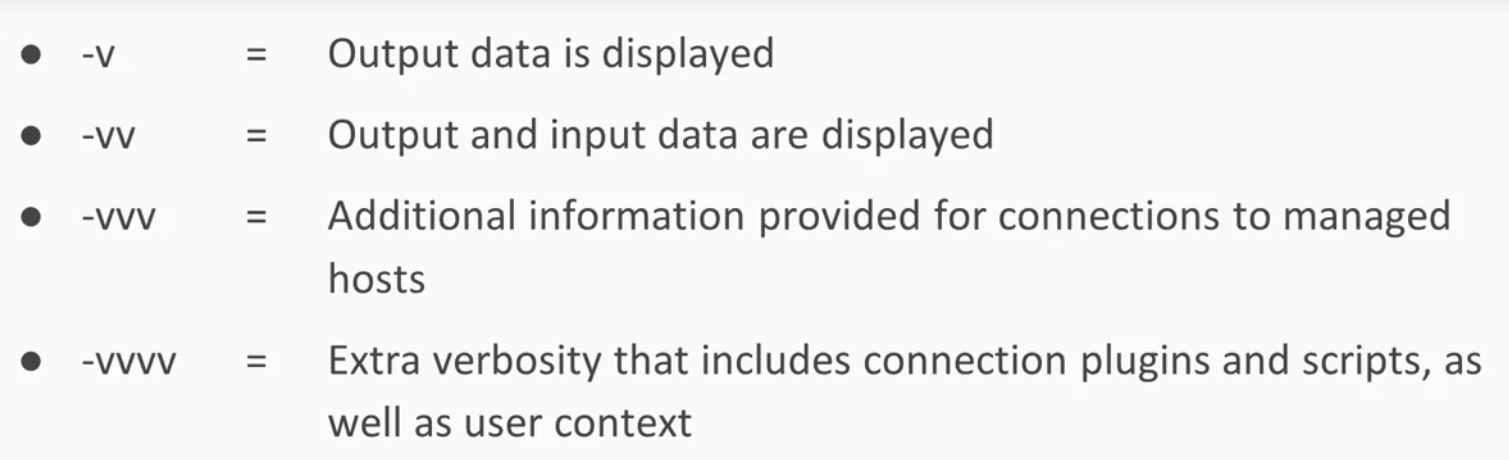
Use below command to start at particular task and continue.

ansible-playbook docker-playbook.yml --start-at-task="Start Docker"

Set the log\_path in ansible.cfg to enable ansible logging.

log\_path=/tmp/ansible.log

We can use below verbose modes with ansible.



ansible-playbook -vvvv docker-playbook.yml --start-at-task="Start Docker"

**VALIDATING TESTING WITH ANSIBLE**:

Go to source code directory of ansible and setup the variables using below command.

source ansible/hacking/env-setup

ansible-test units --tox --python 2.7

install tox if required

**ANSIBLE BEST PRACTICES**:

Refer to best-practices ansible docs.

**UPGRADING ANSIBLE**:

Use below command for upgrading.

Pip install ansible=<version number>

Pip install –upgrade ansible .

We can use pip install with url of github to install ansible latest version.

Pip install git+https:<url>

Please use below URL as reference for playbooks written during course.

https://github.com/revgit12/myAnsible.git