ANSIBLE

→ Ansible is an open-source automation tool that is used for configuration management, application deployment and task automation. It is designed to simlify the management and orchestration of IT infrastructure.

→ We can configure all the n no. of servers by sitting on one machine.

→ This is called CM (configuration management)

Advantages of CM tools

1) Resource saving
2) Provisioning of server
3) DR (Disaster Recovery)
4) Snowflake Servers effect
5) Idempotent

Scanned with ACE Scanner

1) Resource Saving :

→ We are saving time
→ We are saving human resource
→ We are saving money.

2) Provisioning of servers

→ There can be 'n' no. of softwares to install, upgrade software, etc on all the no. of servers, this can be done very effectively because human efforts are less

3) DR (Disaster Recovery)

→ We create data centers in different places, with replica data center,
→ If one data center is damaged we can have another data center of same replica, It creates setup file

4) Snowflake Servers effect :

→ Every server in a data center it

Scanned with ACE Scanner

behaves like snowflake after some time. which means either it is running on different hardware configuration or different software configura-tions at a particular time

5) Idempotent :-

→ CM checks whether the configuration is present on 'n' no. of servers or not, and then configured

→ ~~Suppose~~

→ For example if I have to install tomcat9 on remoter servers, on them, tomcat9 is present (it is called desired state) if it present it will not disturb that server it is called idempotency.

→

tomcat9	tomcat9	tomcat9
tomcat8	tomcat8	tomcat8
-	-	-

→ it won't disturb
→ it will upgrade
→ it will install tomcat9

it is called idempotency

→ Ansible is idempotent and terraform too

Popular CM tools

1) ansible
2) chef
3) puppet
4) salt stack

→ Ansible is created ~~with~~ using scripting knowledge of python, this ansible is created by python

→ Ansible is installed on one server that is called Controller

→ All the remaining servers that ansible is trying to configure are called managed nodes or managed hosts

→ Ansible (controller) communicate with all other remote servers via SSH i.e passwordless SSH

Scanned with ACE Scanner

→ Controller →

→ Server1 } these are my managed nodes
→ Server2

→ First establish passwordless connections b/w controller to Server1 and Server2

To install ansible on controller

→ `sudo apt-get update`

→ `sudo apt-get install -y software-properties-common`

→ To install latest version of ansible

`sudo apt-add-repository ppa:ansible/ansible`

if you don't do this it will install earlier version of ansible

→ Now `sudo apt-get update`

→ Now install ansible

`sudo apt-get install -y ansible`

Scanned with ACE Scanner

→ Ansible knows the servers or nodes by reading a file called inventory file

→ Go to inventory file and add the ip addresses of all the nodes

```
cd /etc/ansible
```

hosts is called inventory file

→ Now open that file and add ip address

```
sudo vim hosts
```

→ Now if you execute any commands on controller, then those commands also will be executed on those two servers

→ `ls -la` is command shows list of all the files

→ To execute this code on all the machines by my ansible

```
ansible all -a 'ls -la'
```

Scanned with ACE Scanner

* Idempotent means if the ansible makes the changes to the system, if the current state differs from the desired state

Make connections for different flavors of linux

→ There will be different types of flavors of linux managed nodes will be there

→ Then there will be different users in different flavors

→ Identify the users and configure with passwords and make ssh connections with the controller.

→ Establish the passwordless connection

```
sudo vi /etc/ssh/sshd_config
sudo service sshd restart
```

→ In the inventory file write the below text

```
ipaddress, ansible_user = username of different OS servers
```

then only it will connect to the manged node of different flavors of OS.

Scanned with ACE Scanner

→ This is done when there is a mismatch of OS between controller and managed nodes

→ Ansible is agentless because there is no client application on remote servers. whereas in chief we need chief controller and chief client.

→ Ansible is pushing the content into the remote servers via ssh passwordless connection, it is called push (Pushing mechanism)

Modules in Ansible :-

(1) Command :-

If you want to execute any linux commands on remote servers

(2) Shell :-

The redirection operator (>) is used by shell, if the o/p of an file is stored, that is done with shell command.

→ To execute a script/python program on remote

servers, we use shell

③ file :-

It is used to creating files, deleting files, changing permission of owner, group, others and creating directories

④ User :-

The user administration is done by user command, they are, creation of users, deleting users, setting password

⑤ copy :-

This is used to copy files/directories from the controller to managed nodes

⑥ fetch :- (opposite of copy command)

This is used to copy files/directories from managed nodes to controller

⑦ apt :-

This command is used to install any software in managed nodes, uninstallations,

Scanned with ACE Scanner

software upgradation are done with apt command

(8) yum :-

It does the same action which done by apt, but it does on redhat linux centOS, where as apt does on ubuntu linux servers

(9) service :-

To restart on every tomcat server, we use service command, sitting on controller we can restart a service, stop a service

(10) git :-

We use the git command in ansible, we use git command, to clone in 'n' no. of servers we use git command

(11) get_url :-

It downloads files from any remote servers into managed nodes

Scanned with ACE Scanner

(12). uri :-
It is used to check whether the managed nodes are reachable to any websites or not, pingable or not

(13) debug :- (print)
It is used to print any content on the screen.

(14) stat :-
It gives the all the information of files and folders on remote servers,

(15) include :-
This is used for calling child playbooks,
eg :- I have a parent playbook, it is trying to invoke child playbook there we use include command

(16) replace :-
to the configuration file, we have to replace some data we use this replace module

Scanned with ACE Scanner

(17) pause:-

It pauses the activity for sometime. then ansible waits untill that activity finishes

(18) docker_container:-

If you want to create container in 'n' no. of remote servers

(19) docker_image

If you want to create images in 'n' no. of remote servers

→ Ansible perform configuration on remote servers, in 3 ways, they are

1. Adhoc Commands
2. Ansible playbooks
3. Roles

modules remains the same in all three ways.

Scanned with ACE Scanner

① Adhoc commands :-

Syntax :-

```
ansible all/ipaddress/group -i path_of_inventory -m
        module_name -a " "
```

all → all the servers which are present in inventory file

ipaddr :- if you give one ipadd it work on that only

group → if you group some servers and give their name it will work on them only

-i → inventory file path

-m → for module name

-a → argument present in command

eg :- `free -m` is on linux command will capture the memory information

→ Execute this on managed nodes

```
ansible all -i /etc/ansible/hosts -m command -a
                                    'free -m'
```

Scanned with ACE Scanner

ansible all -i /etc/ansible/hosts -m command
-a 'ls -la'

use linux commands
if module is command

Inventory file is a ~~any~~ file where you can simply store the managed nodes ipaddr

→ You can create your own inventory file on any name

→ default inventory file is /etc/ansible/hosts

vim myinventory

vim file-name inventory

→ If you are using default inventory file no need of giving -i in the ansible command

eg:- ansible all -m command -a 'free-m'

If you are using your own inventory file you must specify -i.

Scanned with ACE Scanner

→ Only for <u>command module</u> you may/ maynot give -m, for all other modules you must give -m

eg:- ansible all -a 'free-m'

Redirection commands :-

shell --

free-m > file1

It stores all the memory information is file1

→ To store all the memory information of all remote servers on file1

ansible all -m shell -a 'free-m > file1'

It will create a file1 in all the remote servers and stores the memory information at that time.

Scanned with ACE Scanner

Users :-

→ The users information is stored in a file called /etc/passwd

To add a user

```
ansible all -m user -a 'name = sudarshan
                        password = sudarshan' -b
```

-b → become / giving sudo permissions

→ We can assign user id, root id, group id, comment, shell, ...

```
ansible all -m user -a 'name = sudarshan password=
sudarshan uid = 1001 comment = "a normal user"
home=/tmp/.sudarshan shell = /bin/bash' -b
```

file :-

- → name = destination of file to be stored
- → state = touch → creating files
  - = absent → deleting files
  - = directory → creating directories

Scanned with ACE Scanner

ansible all -m file -a 'name=/tmp/file1 state=touch'

To create a directory

ansible all -m file -a 'name=/tmp/~~file1~~ dir1 state=directory'

To remove a file or directory

ansible all -m file -a 'name=/tmp/~~file1~~ dir1 state=absent'

To change the permissions of a file

ansible all -m file -a 'name=/tmp/file1 state=touch mode=770 owner=xyz group=xyz'

-b

Copy:

- → src
- → destination
- → content = used to replace content

ansible all -m copy -a 'src=filename dest=/tmp'

We can copy directories too, copy into any destination

ansible all -m copy -a 'src=directoryname dest=/tmp'

Scanned with ACE Scanner

To replace the content

```
ansible all -m copy -a 'content = "" dest=/tmp/filename
```

We can also change permissions by copy module

```
ansible all -m copy -a 'src = filename dest = any destination filename mode = 777 owner = root group = sudoers'
```

apt :-

- name
- state
- update_cache = yes

To install any software

state = present → to install
= absent → to remove
= latest → upgrade to higher version

purge = yes → it deletes everything related to application

```
ansible all -m apt -a 'name = git state = present update_cache = yes' -b
```

update_cache = yes → works as sudo apt-get update

uri :- Any website is reachable or not

- → url =
- → status_code =

if the website is reachable it shows two status codes i.e 200, if not reachable -1

```
ansible all -m uri -a 'url of any website'
```

Service

- → name → name of the service
- → state = restarted → to restart
  - = stopped → to stop
  - = started → to restart stopped service

```
ansible all -m service -a 'name = tomcat9
state = restarted'
```

replace :- to change some specific text of file

- → regexp → what value are you searching for
- → replace → that text you want to add, write
- → path → here path of the file.

Scanned with ACE Scanner

```
ansible all -m replace -a 'regexp=8080
replace=9090 path=/etc/tomcat9/server.xml
-b
```

→ We are changing the port number of tomcat
→ port number is present in server.xml

get_url :-

- → url → url of application/software
- → destination → which folder you have to download

```
ansible all -m get_url -a 'url=paste the
url address dest=filename'
```

User :-

- → name
- → password
- → uid
- → shell
- → owner id
- → group id

* To change name of the host

sudo hostnamectl set-hostname name of the host

eg:- sudo hostnamectl set-hostname ansible-controller

Git :-

→ repo
→ dest

ansible all -m git -a 'repo = paste url from of github, dest = anyfilename'

fetch :- reverse operation of copy

ansible all -m fetch -a 'src = file name which we have to copy from managed nodes dest = filename in which it is saved

eg:- copy passwords file from managed nodes into controller

ansible all -m fetch -a 'src = /etc/passwd dest = /tmp

Modules and their arguments

1) fetch (src, dest)

2) copy (src, dest, content, owner, group, mode)

3) apt (name, state, update_cache = yes)

4) Service (name, state)

5) uri (url, status_code)

6) git (repo, dest)

7) get_url (url, dest)

8) file (name, state, owner, group owner, mode

9) user (name, password, uid, shell, comment, groupid, home)

10) replace (regexp, replace, path)

→ purge = yes → it deletes everything related to that particular application

11) stat (path, register)

12) debug (var, msg)

Scanned with ACE Scanner

2) Playbooks

→ With adhoc commands we can do configuration but it can be done with single commands, but in some cases one configuration requires many steps, everytime we have to go with all the commands, it makes complexity and time consuming.

→ So we go with playbooks, it is onetime activity, next time we just execute the playbook it will do all the configurations

→ Adhoc commands are calling only one module at a time

eg:- → If I want to install tomcat9, I need

→ apt module — 2 time → tomcat9, tomcat9-admin

→ copy module → to copy tomcat-users.xml

→ Service module

→ Put all these information in yml file and save and execute the playbook, and it is reusable

Scanned with ACE Scanner

→ The ansible playbooks are called plays

→ The no. of plays combined together is called ansible playbook

→ These playbooks are created by yml,

in yml we have xml, json, yml

<tomcat-users> → nodes
child element

<user username = sudarshan password = sudharshan

roles = manager-script"/>

</tomcat-users>

but in yml

...

→ starting will ---

→ ending with ...

→ Now represent tomcat-user.xml file in yml, is shown below.

Scanned with ACE Scanner

- - -

tomcat-users :

User :

Username : sudarshan

key value

No space space

password : sudarshan

roles : manager-script

. . .

→ after the colon, there should be space, before the colon no space

→ To check whether code is correct or not we can check in online in YamlLint.com

→ Create a folder called playbooks

```
mkdir playbooks
```

→ In playbooks, create a playbook

```
cd playbooks/
```

→
```
vim playbook1.yml
```

→ To check whether the playbook syntax is correct

```
ansible-playbook name of playbook --syntax-check
```

Scanned with ACE Scanner

→ To run playbook

```
ansible-playbook name of playbook -b
```

Playbook structure :

```
---
- name: title of the playbook :
  hosts: all / ipaddress / group
  tasks:
    - name: operation that you are doing
      modulename :
      arguments :
      :
      :
      :
...
```

Scanned with ACE Scanner

Playbook for user:

```
---
- name: Create user on all managed nodes
  hosts: all
  tasks:
    - name: Create user
      user:
        name: sudarshan
        password: sudarshan
        uid: 1234
        gid: 1235
        shell: /bin/bash
        home: /tmp/sudarshan
        comment: "A normal user"
```

Create a file and put some content in that file

→ for this required two modules

file

copy

```
---
- name: create a file and put some content
  hosts: all
  tasks:
    - name: create a file
      file:
        name: /home/ubuntu/file1
        state: touch
    - name: put some content
      copy:
        content: "Welcome"
        dest: /home/ubuntu/file1
...
```

Variables! -

1) Global scope variables → highest priority
2) Play scope variables → least "
3) Host scope variables → second "

→ In this mostly used in organisation host scope variables, but highest priority given to global scope variables

Scanned with ACE Scanner

① Global Scope Variables

→ In global scope variables, the variables are given using "{{ }}" and in the command it self they are assigned which things we have to perform.

→ It is reusable,

→ "{{ }}" → jinja2 format

→ To run global scope variables we use extra-vars

ansible-playbook playbookname.yml --extra-vars "gives values over here" -b

eg:- To install git

ansible-playbook playbookname.yml --extra-vars " a = git b = present c = yes " -b

Scanned with ACE Scanner

(*) To copy a file from managed node1 to managed node2.

→ first copy that file into controller using fetch module and next copy that file into managed node2 by using copy module

---

```
- name : To copy file from managed node1
  hosts : ip address of managed node1
  tasks :
    - name : fetch file
      fetch :
        src : filename/path of that file
        dest : any foldername
- name : copy file into manged node2
  hosts : ip address of managed node2
  task :
    - name : copy file
      copy :
        src : filename/path of the file
        dest : any file name
```

...

→ For every ipaddress we have to write name hosts, tasks, this is performed individually

→ If you give all, no need of writing name hosts and tasks

(*) Global variables program/syntax

```
---
- name: Install software
  hosts: all
  tasks:
    - name: install s/w.
      apt:
        name: "{{a}}"
        state: "{{b}}"
        update_cache: yes
...
```

(2). Play scope variables

→ In this variables are given within the playbook itself

→ They are created under vars section

Syntax :-

```
- name :
  hosts :
  vars :
    - a :
    - b :
    - c :
  tasks :
    - name :
      module name :
        arguments : "{{a}}"
```

→ In vars pass the arguments

eg :-

```
- - - name :
    hosts :
    vars :
      - a : tomcat9
      - b : present
      - c : yes
    tasks :
      - name :
        apt :
          name : "{{a}}"
          state : "{{b}}"
```

Scanned with ACE Scanner

update_cache: "{{c}}"

→ Advantages! —

→ If we have 200 or 300 lines of code in a playbook, those are given by a variables, if we want to pass in the command line for runing becomes difficult to remember those variables

* We have large code, with many lines, if we want to work on tomcat9 instead of tomcat8 (any software application), but the code is written for tomcat8, we have replace tomcat8 with tomcat9 in all the lines, to overcome that difficulty make work easier, just go to vars section and change names that's it.

→ And this reusable,

→ Though this written for certain application

can be used to run another application.

eg:- We written playbook for tomcat9 installation, we can use it for git and other installation.

→ Global variables are given more priority

Scenario:-

→ Imagine that in a organisation we have 1000 servers, in that 700 servers are using tomcat9 and remaing are using other s/w such as apache etc,

→ More servers are using tomcat9, but ones in a while we are using apache.

→ Write a default playbook for tomcat9 using play scope variables, whenever you want to other s/w to be used just go to ~~vars section and change vars section and variable names which hardcoded~~. command line and pass extra-vars

Scanned with ACE Scanner

③ Host scope variables

a) Variables that work on a group of servers

b) Variables that work on a single server

→ To work on a particular group of servers or on a single server we can use this concept.

→ For grouping, open the inventory file and add square brackets [any name]

eg:- [webservers]

172.31.3.36

172.31.3.35

172.31.3.38

This is one group

→ We can use many groups like

[webservers]

172.31.3.36

172.31.3.35

[appserver]

172.31.3.38

Scanned with ACE Scanner

→ and one ipaddress used in one group can also be used in another group also.

→ In the playbook, in the place of all in host section. just give that group name it work on that group of servers only

eg:- hosts : webserver

→ one ipaddress can be present in multiple groups.

→ ansible groupname -a 'argoments'

→ In playbook just change in host section instead of all replace with group name and run the command as usual.

→ Multiple groups in a single group

[servers : children] → predefined name

↳ anyname

beneath this write other group names.

Scanned with ACE Scanner

eg:- [servers : children]
webserver
appserver
⋮
etc

→ write this group name
- /etc/ansible/hosts

a) Variables that work on a group servers.

→ I create a variables that works or targets only on a particular group only. and I don't want to disturb other groups.

→ Go into the playbooks directory, where all your playbooks are present.

→ In that create a directory, its name should be group_vars only

→ Inside group_vars ~~folder~~ and create a file, the filename should be group name of servers, ~~groupname of servers~~

→ ~~be~~ becoming filename here.

→ In that file put your variables a, b, c

vim groupname

---

a : tomcat9

b : present

c : yes

...

→ Now come out of the group_vars folder and create a playbook

Scenerio :-

→ If we have no. of servers in a group we want to change any configurations

→ just go to group folder and change in that file, it will be reflected on all the servers in that group.

→ We created no. of playbooks in a group, we want to change or any modifications, just go to group_vars file and change it will be reflected to all playbooks

Scanned with ACE Scanner

eg:/ Scenario:

→ Suppose we created 100's of playbooks targeting one webserver group, It is not easy to go into each playbook and make changes, In that case go to webservers file created under group_vars, there make changes, it will reflect to all the playbooks

b) Variables that work on one single server

→ Go to the location where all your playbooks are present.

→ In that create a directory/folder its name should be host_vars

→ In that folder, create a file, its name should ipaddress of a server on which you work, here ipaddress becomes folder name.

→ In that declare variables a, b, c

Scanned with ACE Scanner

a : tomcat9
b : present
c : yes

→ Now come out of host_vars folder and create a playbook, in that playbook at host section, give ipaddress on which you created a folder in host_vars folder

→ If you change anything in that file, all the playbooks associated to that ipaddress will get reflected with the changes

Loops :-

(i) with_items
(ii) with_sequence

Scanned with ACE Scanner

i) with_items

→ If you are using same modules in a playbook, in that playbook identify the items which is changing always, then replace that in a loop.

eg:- To install java, maven, git, in that playbook, name is changing but all other code remains same, in that case use loops.

Syntax/eg:-

```
---
- name: Install s/w apps
  hosts: all
  tasks:
    - name: Install s/w's
      apt:
        name: "{{item}}"
        state: present
        update_cache: yes
      with_items:
        - java
        - maven
        - git
```

Scanned with ACE Scanner

eg:- I want to install java, uninstall maven & upgrade git

```
---
- name: Install/uninstal/upgrade s/w's
  hosts: all
  tasks:
    - name: install/uninstall/upgrade
      apt:
        name: "{{item.a}}"
        state: "{{item.b}}"
        update_cache: "{{item.c}}"
      with_items:
        - {a: java, b: present, c: yes}
        - {a: maven, b: absent, c: no}
        - {a: git, b: latest, c: yes}
...
```

→ loop will work for only one module i.e for individual modules and we can't implement one loop for all modules

Scanned with ACE Scanner

Handlers :-

→ Handlers are modules that dependent on some other modules, that some other modules should be successfull and made changes, then handlers will be executed.

eg:-

```
---
- name: Implementing handlers
  hosts : all
  tasks:
    - name: install apache2
      apt:
        name: apache2
        state: present
        update_cache : yes
    - name : Edit the index.html file
      copy :
        content: "welcome"
        dest: /var/www/html/index.html
      notify: Restart apache
  handlers:
    - name: Restart apache
```

Scanned with ACE Scanner

Service :

name : apache2

state : restarted

→ notify :- this will inform the next module i.e handler to execute if the changes are made and got success

→ then only handler will run

① Handler are modules that are executed if the modules in the tasks section with notify keyword is successfull and it has made some changes

② Handlers are executed only after all the modules in the tasks section are executed.

③ Handlers are executed in the order that they are mentioned in the handler

section and not in the order that we call them in the tasks section.

(4) * Even if a handler is called multiple times in the tasks section it will be executed only once

Tags:-

→ Tags are nothing but giving alias to the modules,

→ You give alias, then you get more control on module i.e when it want to execute or stop, we can give by tags

→ which module I want to execute and which module I want to skip with tags

```
ansible-playbook playbookname.yml --tags="tagged"
```

Scanned with ACE Scanner

I want to execute untagged modules

```
ansible-playbook playbookname.yml --tags="untagged"
-b
```

→ I want to execute based on tag name then give that name

```
ansible-playbook playbookname.yml --tags="tag_name"
-b
```

eg:-

```
---
- name: Implementing tags
  hosts: all
  tasks:
  - name: Install tree
    apt:
      name: tree
      state: present
      update_cache: yes
    tags: tree_installation
```

tags are given in allined with module name after the arguments

→ You can give tags to each module

Gathering facts :-

when a playbook is executed, first section it has gathering facts, it means it is gathering hardware / software information of remote servers, sometime it might be time taking

→ we can skip those things

→ Go to playbook.

→ beneath the hosts, write

gather_facts : false

then it will stop executing that section, it will be run in background

-b :-

→ It gives sudo permission, if you give become : yes, beneath the hosts in the playbook, you need not to give in command section.

Scanned with ACE Scanner

(*) ansible.cfg

→ This will be present in /etc/ansible/ folder

→ make changes in that, then our own customised inventory file will work as default inventory file.

→ In the inventory section, remove the comment (#) and the give your inventory file's path

eg: /home/ubuntu/playbook/myinventory

→ You can also change colours of the success o/p, changed o/p

Error handling:-

→ if a particular module fails, it will not go to next module.

→ I want to execute when the module fails, for that we write code in block

Scanned with ACE Scanner

and rescue.

→ If the module fails, it will execute next module.

→ If the code in the block section is success the control will not come into rescue section, else it will.

→ apt will work on ubuntu servers

→ yum will " " redhat servers

Syntax for error handling.

```
- name :
hosts : all
tasks :
    - block :
        - name : Install git
          apt :
            name : git
            state : present
            update_cache :
      rescue :
        - name : Install using yum
          yum :
            name : git
            state : present
            update_cache : yes
```

```
always:
  - name: Display succes msg
    debug:
      msg: " "
```

→ always section is alternative you can use or not, just it show the succes or failure msge

Vault:- (locker or storage)

→ It is used to keep password for the playbook, encrypt, decrypt, view playbook

① To create a vault playbook

```
ansible-vault create playbookname.yml
```

② To see the content of a vault playbook

```
ansible-vault view playbookname.yml
```

③ To edit a vault playbook.

```
ansible-vault edit playbookname.yml
```

Scanned with ACE Scanner

(4) To convert a normal playbook into a vault playbook.

```
ansible-vault encrypt playbookname.yml
```

(5) To convert a vault playbook into a normal playbook.

```
ansible-vault decrypt playbookname.yml
```

(6) To change the password of vault playbook

```
ansible-vault rekey playbookname.yml
```

→ Then encrypted vault uploaded in git repository, there also will be in the form of encrypted

To run vault playbook

scenario:-

→ when block and rescue both sections are failed, then program gets stopped.

→ for this you should gather the OS information first and design a playbook according to it.

Scanned with ACE Scanner

→ In this case we use <u>when</u> condition is nothing but <u>if condition</u>

```
---
- name: create a file on all servers
  hosts: all
  vars:
    - a: 10
  tasks: ~~create a file~~
    - name: create a file
      ~~file~~:
        name: /tmp/file1
        state: touch
        mode: ~~770~~
      when: a == 10
...
```

<u>state</u>:-

→ This module is used to capture the detailed information about files and folders which are present on remote servers.

Scanned with ACE Scanner

eg:-

```
- name: check if f1 folder is present if not
        create a file f1
hosts: all
tasks:
  - name: check for f1 folder
    stat:
      path: /home/ubuntu/f1
    register: a
  - name: Display output of the above
          module
    debug:
      var: a
```

register :- It is like datatype which stores the data of any modules output

```
register: anyname
```

It stores the o/p in anyname (whether it is a file/location etc)

Scanned with ACE Scanner

Scenario:

write a program/playbook, if f1 folder is not present on remote server then create a f1 file on it.

```
---
- name: check f1 folder is present if not create
        a file f1
  hosts: all
  tasks:
    - name: check f1 folder is present
      stat:
        path: /home/ubuntu/f1
      register: a
    - name: Display o/p of the above module
      debug:
        var: a
    - name: create f1 ~~directory~~ file if f1 folder not
            present
      file:
        name: /home/ubuntu/f1
        state: touch
      when: a.stat.exists == false
```

this is obtained from o/p debug module

Scanned with ACE Scanner

Scenario 6!

Create a file on all managed node, see that it is executable or not! If it is executable file then ansible should delete it. If it is not executable file ansible should not delete it.

```
---
- name: Check if a file is executable if so
        delete it
hosts: all
tasks:
  - name: check file1 information
    stat:
      path: /tmp/file1
    register: a
  - name: Display o/p of the above module
    debug:
      var: a
  - name: Delete if the file is executable
    file:
      name: /tmp/file1
      state: absent
    when: a.stat.executable == true
```

Scanned with ACE Scanner

Scenario:

(i) On one server setup jenkins environment and other servers setup QA and Prod environment. In real time there may be 100's of prod servers, and 100's of QA servers where we have to setup, for that we have to design a playbook, where it can be reusable

```
---
- name: Install jenkins and tomcat9
  hosts: jenkinsserver
  tasks:
    - name: Install jq/wget
      apt:
        name: "{{ item.a }}"
        state: "{{ item.b }}"
        update_cache: "{{ item.c }}"
      with_items:
        - {a: openjdk-11-jdk, b: present, c: yes}
        - {a: git, b: present, c: no}
        - {a: maven, b: present, c: no}
```

```
- name : Download jenkins.war file
  get_url :
    url : paste url of jenkins.war
    dest : /home/ubuntu

- name : Install tomcat on QA and Prodservers
  hosts : QA and ProdServers groupname
  tasks :
    - name : Install tomcat9
      apt :
        name : "{{item.a}}"
        state : "{{item.b}}"
        update_cache : "{{item.c}}"
      with_items :
        - {a:tomcat9, b:present, c: yes}
        - {a:tomcat9-admin, b:present, c:yes}
    - name : Copy tomcat-users.xml file
      copy :
        src : tomcat-users.xml
        dest : /etc/tomcat9
      notify : Restart tomcat9
  Handlers :
    - name : Restart tomcat9
      service :
        name : tomcat9
        state : restarted
```

Scanned with ACE Scanner

include :-

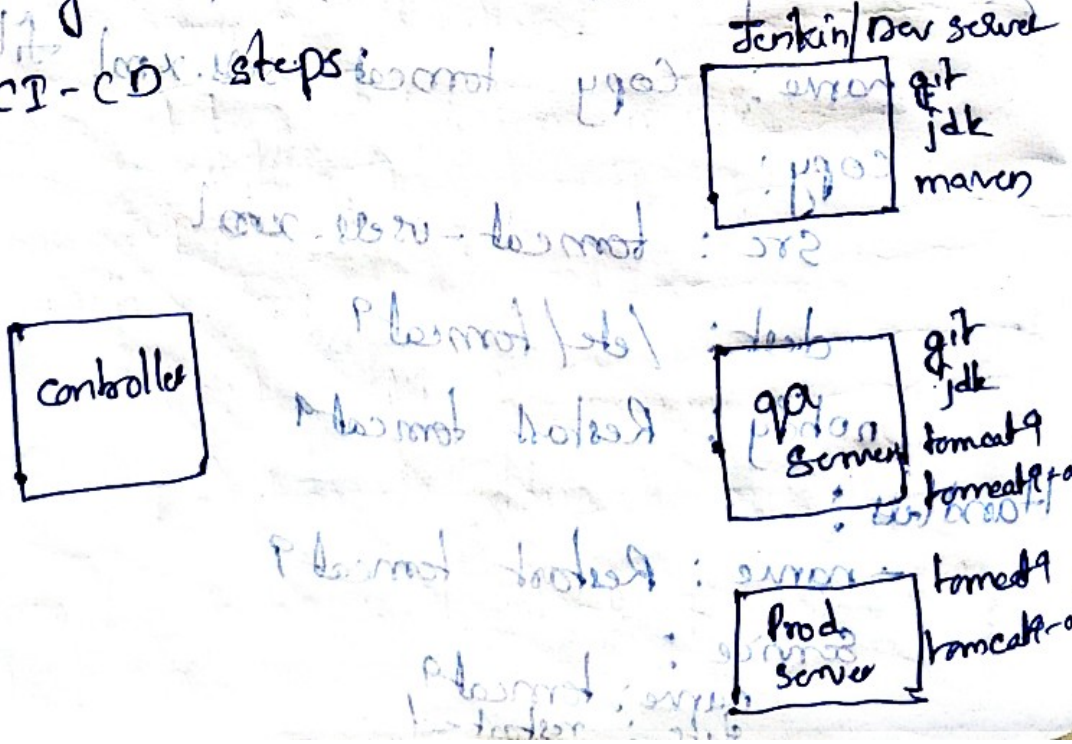
It is used to call the child playbook in a parent playbook.

Syntax:-

```
---
- name : call child playbook using include
        module
  hosts : all
  tasks :
    - name : call child playbook
      include : playbook_name.yml
```

Scenario:-

(*) Using ansible design a playbook for CI-CD steps:



Scanned with ACE Scanner

I. Install all required s/w on respective Servers

II. To download code from git use "git" module.

III. To build use shell module

IV. To deploy copy module.

```
---
- name: Install s/w's on Devserver
  hosts: Devserver
  tasks:
    - name: Install s/w's
      apt:
        name: "{{ item.a }}"
        state: "{{ item.b }}"
        update_cache: "{{ item.c }}"
      with_items:
        - "{a: git, b: present, c: yes}
        - {a: openjdk-11-jdk, b: present, c: no}
        - {a: maven, b: present, c: no}
- name: Install s/w's on qaserver
  hosts: qaserver
  tasks:
    - name: Install s/w's
```

Scanned with ACE Scanner

```
apt:
    name: "{{item.a}}"
    state: "{{item.b}}"
    update_cache: "{{item.c}}"
  with_items:
    - {a: git, b: present, c: yes}
    - {a: openjdk-11-jdk, b: present, c: no}
    - {a: tomcat9, b: present, c: no}
    - {a: tomcat9-admin, b: present, c: no}
- name: Install s/w's on prod server
  hosts: prodserver
  tasks:
    - name: Install s/w's
      apt:
        name: "{{item.a}}"
        state: "{{item.b}}"
        update_cache: "{{item.c}}"
      with_items:
        - {a: tomcat9, b = present, c = yes}
        - {a: tomcat9-admin, b = present, c = no}
- name: Continuous download & cont build
  hosts: Derserver
```

Scanned with ACE Scanner

tasks:

- name: Download code from github

  git:

    repo: Paste url from github

    dest: dev-git

- name: build the code

  shell: cd dev-git; mvn package

- name: copy that artifact into controller

  copy:

    src: path of the ~~present~~ pwd, where it is build

    dest: /tmp.

- name: Cont deployment in qa server

  hosts: qaserver

  tasks:

  - name: copy that artifact into qaserver

    copy:

      src: path of pwd in controller

      dest: ~~test/~~ ~~dev-~~ /var/lib/tomcat9/webapps/testapp.war

  - name: download selenium scripts

    git:

      repo: Paste url from github

dest: test-git

- name: run selenium programs

  shell: java -jar test-git/testing.jar

- name: Cont delivery

  hosts: prodservers

  tasks:

  - name: deploy in prodserver

    copy:

      src: path of pwd from controller

      dest: /var/lib/tomcat9/webapps/prodapp.war

...

To know the OS information

We use debug module

- [msg: "{{ ansible_distribution }}"]

eg:

```
- name : Capture OS info
  hosts : all
  tasks :
    - name: Capture OS name
      debug:
        msg: "{{ ansible_distribution }}"
```

To capture version

```
msg: "{{ ansible_distribution_version }}"
```

eg:- After capturing the OS information then use when conditions in playbook

→ Install git on ubuntu and RHEL

```
- name : Install git using apt and yum
  hosts : all
  tasks :
    - name : Install git using apt module
      apt :
        name : git
        state : present
        update_cache : yes
      when : ansible_distribution == "Ubuntu"
```

Scanned with ACE Scanner

```
- name : Install git using yum module
  yum:
    name : git
    state : present
    update_cache : yes
  when : ansible_distribution == "RedHat"
...
```

(*) Configure apache2/httpd based on OS.

```
---
- name : configure apache2/httpd based on OS
  hosts : all
  tasks :
    - name : configure apache2 on ubuntu
      apt :
        name : apache2
        state : present
        update_cache : yes
      when : ansible_distribution == "ubuntu"
    - name : configure httpd on RHEL
      yum :
        name : httpd
        state : present
        update_cache : yes
      when : ansible_distribution == "RedHat"
```

Scanned with ACE Scanner

```
- name : Edit the index.html file
  copy :
    content : "Welcome"
    dest : /var/www/html/index.html
- name : Restart apache2 on ubuntu
  service :
    name : apache2
    state : restarted
  when : ansible_distribution == "ubuntu"
- name : Restart httpd on RHEL
  service :
    name : httpd
    state : restarted
  when : ansible_distribution == "RedHat"
- name : check server response
  uri :
    url : "{{item}}"
    status_code : 200
  with_items :
    - http : ip-address
    - http : ip-address
```

Scanned with ACE Scanner

(*) Create child playbooks and call them from parent playbook and store those variables in group_vars / install tomcat9 and tomcat9-admin.

```
---
- ~~name: Install tomcat9/tomcat9-admin~~
  ~~tasks: call~~
  tasks:
    - name: Install tomcat9/tomcat9-admin
      apt:
        name: "{{item.a}}"
        state: "{{item.b}}"
        update_cache: "{{item.c}}"
      with_items:
        - {a: "{{a}}", b: "{{b}}", c: "{{c}}"}
        - {a: "{{d}}", b: "{{b}}", c: "{{e}}"}
```

Now go to group_vars folder and create a file in it and declare variables as follows

```
---
a: tomcat9
b: present
c: yes
```

Scanned with ACE Scanner

d : tomcat-admin

e : no

Now create a child playbook for to copy user.xml file

- name : copy tomcat-users.xml

  copy:

    src : "{{f}}"

    dest : "{{g}}"

Now for restart also create a child playbook

- name : Restart

  service:

    name : "{{h}}"

    state : "{{i}}"

declare all variable in a file which is present in group_vars.

Then create a parent playbook, in that use include module and run the play book.

Scanned with ACE Scanner

parent playbook

```
---
- name: Configure tomcat using child playbook
  hosts: all
  tasks:
    - name: call multiple child playbooks
      include: "{{item}}"
      with_items:
        - child playbook name
        -
        -
        -
...
```

Roles :-

we are creating playbooks in a organisation

→ We are declaring variables, handlers, child playbooks,

→ Suppose if the persons who created all these and left the organisation, the next persons who joined and continue the

Scanned with ACE Scanner

works.

→ He feels difficult, what are the variables and where these playbooks are stored and where are child playbooks, to understand all these and start working are time taking and feels difficult.

→ To avoid this problem ansible given a concept called "Roles"

→ In this all the child playbooks, variables, files, handlers etc are organised in a systematic format for every playbook there is the folder structure.

→ Go to /etc/ansible, their you find roles

→ Go inside roles folder and create a role

```
ansible-galaxy init role_name
```

eg:- I want to create a role apache

```
ansible-galaxy init apache
```

Scanned with ACE Scanner

→ In ansible galaxy site, there are in no. roles which are predefined

→ when type that command, it will connect galaxy site and download that role if it is present

→ We can also create our own role in that scenario we use command

```
sudo ansible-galaxy init apache --offline
```

tree structure of role

```
└─ apache
   ├─ README.md
   ├─ defaults
   │   └─ main.yml
   ├─ files
   ├─ handlers
   │   └─ main.yml
   ├─ tasks
   │   └─ main.yml
   ├─ templates
   ├─ tests
   │   ├─ inventory
   │   └─ test.yml
   └─ vars
       └─ main.yml
```

Scanned with ACE Scanner

README.md :-

It is a plane text file, where you can type what you are planning to create.
Its like comment section of entire role.

defaults :-

Here you give entire information about apache application / any application on which you are working on which supports, OS information etc in yml format

files :-

If any files we require place in it
eg :- while installing tomcat9, we require that file in tomcat-users.xml file., place the files folder.

tasks :-

Here you write all the tasks which are required.

Handlers :- place handlers modules in this folder

Scanned with ACE Scanner

meta:-

Data about the data,
ie who created apache role, why it is created, when it is created.

templates:-

→ If you have any data which is dynamically changing, mostly in the case of variables which are passed from extra-vars, that information place in this folder.

tests:-

Certain configurations are done on remote servers, You want to check whether the configurations are done correctly or not. that information keep in this folder.

vars:-

variables are stored in vars folder

*** Role is a predefined folder structure it is helping you to organise your data, it is encapsulating whole information of one software application.

→ After creating role, that role will be executed via playbook.

→ Create a playbook like

```
---
- name: configure apache using roles
  hosts: all
  roles:
    - role name
```

(*) Install tomcat9 and tomcat9-admin

In tasks folder

```
- name: call multiple child playbooks
  include: "{{ items }}"
  with_items:
```

Scanned with ACE Scanner

- install tomcat.yml
  - ~~install tomcat.~~
  - configure.yml
  - restart.yml

Now design a playbook in tasks section

```
---
- name: Install tomcat9 and tomcat9-admin
  apt:
    name: "{{item.a}}"
    state: "{{item.b}}"
    update_cache: "{{item.c}}"
    with_items:
      - {a: "{{pkg1}}", b: "{{state1}}", c: "{{cache1}}"}
      - {a: "{{pkg2}}", b: "{{state1}}", c: "{{cache2}}"}
```

Now in vars folder, store the data..

pkg1 : tomcat9

pkg2 : tomcat9-admin

state1 : present

cache1 : present

cache2 : no

Scanned with ACE Scanner

Create all the child playbooks in tasks section.

At last now create a playbook to be executed

———— x ————

Scanned with ACE Scanner