Ansible Automation:

Ansible is a powerful automation tool that can replace many Bash scripts with more maintainable and scalable solutions. Ansible is an automation tool used for configuration management, application deployment, and task automation.

Key Ansible concepts:

1. Playbooks: YAML files containing a set of tasks to be executed on remote hosts.
2. Tasks: Individual units of work in a playbook.
3. Modules: Pieces of code Ansible executes to perform specific operations.
4. Inventory: A list of managed nodes that Ansible can work with.

Ansible Playbook Structure

```
---
- name: Playbook Name
  hosts: target_hosts
  become: yes/no
  vars:
    variable1: value1
  tasks:
    - name: Task 1 Name
      module_name:
        param1: value1
        param2: value2
```

- `name`: A description of what the playbook or task does.
- `hosts`: Specifies which hosts from the inventory this play applies to.
- `become`: Whether to escalate privileges (like sudo).
- `vars`: Define variables used in the playbook.
- `tasks`: A list of tasks to be executed.

Ansible configuration file:

When we install ansible by default configuration files will get created in the following location:

/etc/ansible/ansible.cfg

This configuration file contains several sections, they are:

1. Defaults
2. Inventory
3. Privilege_escalation
4. SSH_connection
5. Paramiko_connection
6. Persist_connection
7. Colors

Here are some frequently used configuration options:

1. In the [defaults] section:
    - inventory: Specifies the default inventory file
    - remote_user: Default username for SSH connections
    - host_key_checking: Whether to check SSH host keys
    - roles_path: Where to look for roles
    - forks: Number of parallel processes to use
2. In the [privilege_escalation] section:
    - become: Whether to use privilege escalation by default
    - become_method: Default method for privilege escalation (e.g., sudo, su)
    - become_user: Default user to become when using privilege escalation
3. In the [ssh_connection] section:
    - ssh_args: Additional SSH arguments
    - control_path: Location of ControlPath sockets

Environment Variable:

`ANSIBLE_CONFIG` = `/opt/ansible_web.cfg`

Copy of Default Config File in current directory:

`/opt/web_playbooks/ansible.cfg`

Config file in home directory:

`.ansible.cfg`

Default Config File:

`/etc/ansible/ansible.cfg`

If we have all types of configuration files then it follows the priority:

1. Environmental variable: 1st priority is always to the parameters configured in the file specified through an environmental variable

   `ANSIBLE_CONFIG` = `/opt/ansible_web.cfg`

2. Current directory config file: 2nd priority **ansible.cfg** file in the current directory
3. Home directory config file: **.ansible.cfg** file 3rd priority in users home directory
4. Default config file

Example of ansible.cfg:

```
[defaults]
inventory = ./inventory
Log_path = /var/log/ansible.log
library= /usr/share/my_modules
roles_path=atc/ansible/roles
action_plugins=/usr/share/ansible/plugins/action
remote_user = ansible
host_key_checking = False
gathering= implicit
timeout=10
forks = 5

[privilege_escalation]
become = True
become_method = sudo
become_user = root

[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=60s
```

Ansible configuration variables:

There are different ways to pass the environmental variables in

1. For single playbook:

   ANSIBLE_GATHERING= explicit ansible-playbook playbook.yml

2. If we want throughout the shell session, up to we exit from the shell:

   export ANSIBLE_GATHERING= explicit
   Ansible-playbook playbook.yml

3. If we want to change on different shells, on different users on different systems is to create a local copy of configuration file in playbooks directory and update the parameter:

   /opt/web-playbooks/ansible.cfg
   gathering =explicit

To find the different configuration options, what are the corresponding environmental variables are and what they mean

View configuration:

- To find the different configuration options, what are the corresponding environmental variables are and what they mean

  ansible -config list ⇒ list all the configurations

- We have different config files in the system in default /etc/ansible.ansible.cfg , one in present directory , one in home directory, to see which config file is in active, we use

  ansible-config view ⇒shows the current active config file details

- Shows as comprehensive list of current settings picked up, and where it is picked up
  ansible-config dump ⇒ shows the current settings

Eg:
export ANSIBLE_GATHERING=explicit
ansible-config dump | grep GATHERING
DEFAULT_GATHERING(env:ANSIBLE_GATHERING)=explicit

- **Version Control**: Keep your ansible.cfg in version control along with your playbooks.
- **Project-Specific Configurations**: Use project-specific ansible.cfg files in your project directories for settings that should apply only to that project.
- **Comment Your Configurations**: Use comments (lines starting with ';' or '#') to explain non-obvious settings.
- **Security**: Be cautious with settings like host_key_checking = False. While convenient for testing, it can be a security risk in production environments.
- **Use Environment Variables**: For sensitive information, use environment variables instead of hardcoding values in ansible.cfg.
- **Regular Review**: Periodically review and update your configuration to ensure it aligns with current best practices and your project needs.

If we want to change only one parameter in the config file , we dont need to copy the whole default config file , instead of copying the whole config file, we can override the single parameter using environment variables

What the environment variable should be?

Change the parameter in to uppercase and add the ansible word as prefix to it in uppercase

gathering =implicit ANSIBLE_GATHERING=explicit ⇒ this environmental variables have highest precedence

YAML:

- Ansible playbooks or text file or config files are written in YAML
- YAML is used to represent config data
- Key value pair, separated by colon
- Space should be mandatory in between colon and value
- Number of spaces in front of each property should be same

Key value pair:

Fruit: Apple
Vegetable: Carrot
Liquid: Water

Array/list:

Fruits:
- Oranges
- Apple
- Banana

Vegetables:
- Carrot
- tomato

Dictionary/map:

Banana:
    Calories: 104 #here the space up to calories and space up to fat should be same
    Fat: 0.4g
Grapes:
    Calories: 62
    Fat: 0.3g

Dictionary vs list vs list of dictionaries:

Dictionary: if we want to display the all details of the single item/product we use dictionary
List: stores multiple items of same type of object

eg:

- red carer
- blue car
- Black car

List of dictionary:

Stores all info about each car:

- Color: blue
- Model:
    - Name: Corvette
    - Model: 1995
- transmission: manual
- Color: black
- Model:
    - Name: Corvette
    - Model: 1996
- transmission: manual
- Color: grey
- Model:
    - Name: Corvette
    - Model: 1997
- transmission: manual

Dictionary: unordered
List: ordered

Eg:
Dictionary:

1. Banana:
    Calories: 105
    Fat: 0.4g
2. Banana:
    Fat: 0.4g
    Calories: 105

Both 1 and 2 dictionaries are equal, but list:

1. Fruits:
- Oranges
- Grapes
- Banana
- Apple

2. Fruits:
- Banana
- Apples
- Grapes
- oranges

Both 1 and 2 list are not same because of their order

List of directories:

Ansible inventory:

- Ansible can connect to multiple servers by using ssh in linux, powershell in windows
- Agentless: to work with ansible we no need to install any other software on target machines.
- Information of target machines is stored in inventory file, if we don't create that file, the ansible uses the default inventory file to store the information about the target machines(.: etc/Ansible/hosts location)
- Inventory file is in ini format, simply displays n no. of servers one after the other.
- Way 1:
  server1.company.com
  server2.company.com
  server3.company.com

- way2:

  [Mail]

  server1.company.com
  server2.company.com

  [db]

  server3.company.com
  server4.company.com

For giving alias name:

1. Alias name is given in beginning of line, and then address is assigned to ansible_host parameter
2. **Ansible_host** is an **inventory parameter** used to specify the ip address of a server
3. Another inventory parameters:
   a. **ansible_connection-ssh/winrm/localhost # defines how ansible connects to the server, like through windows or linux , etc**
   b. **ansible_port-22/5986 # default it is set to 22**
   c. **ansible_user-root/administrator # defines user who is creating the connection like root or admin**

d. **Ansible_ssh_pass- password #display like text format which is not safe**
e. **Ansible_password for windows password**

Example:

web ansible_host=server1.company.com ansible_connection=ssh ansible_user=root

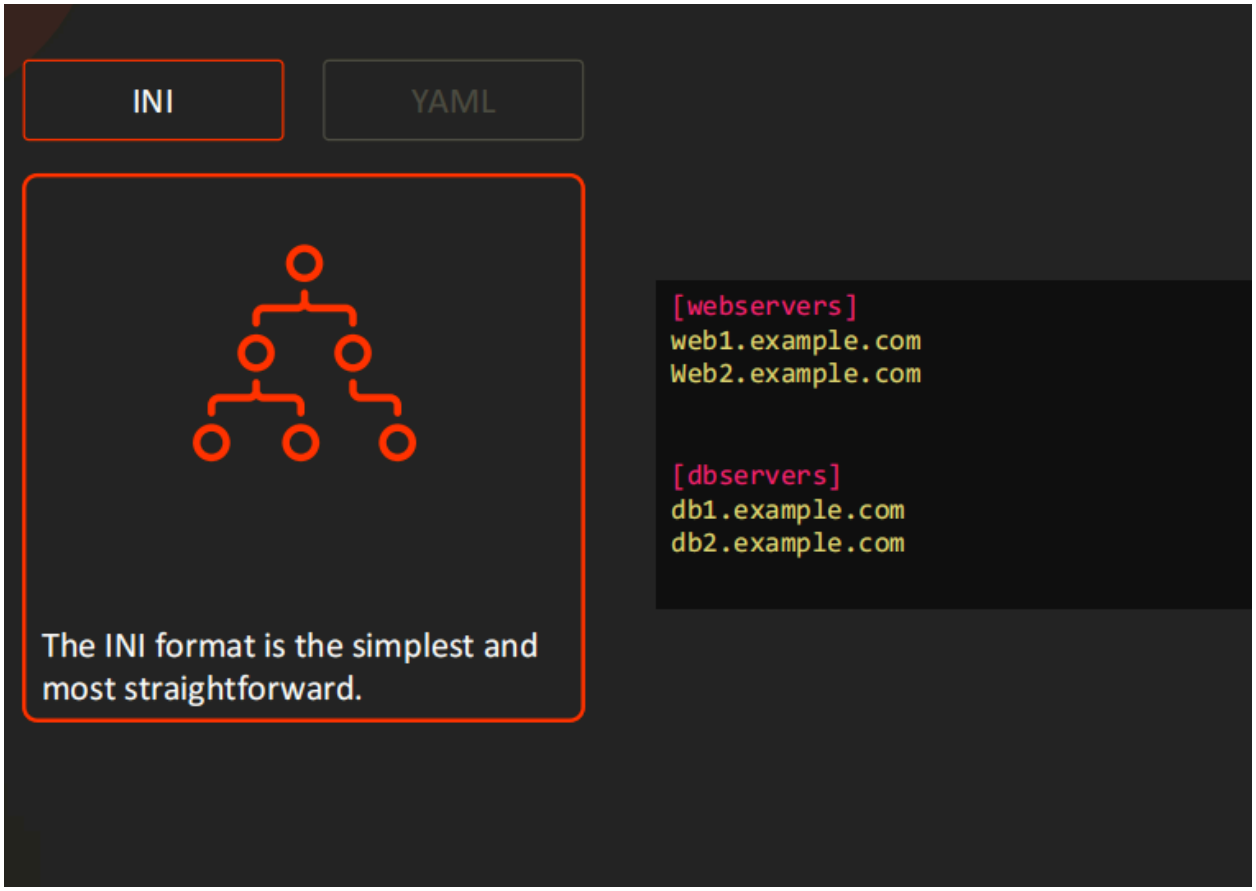
db ansible_host=server2.company.com ansible_connection=winrm ansible_user=admin

mail ansible_host=server3.company.com ansible_connection= ssh ansible_ssh_pass = p!2s#

localhost ansible_connection=localhost

Inventory formats:

1. INI
2. YAML



INI:
Basic that follows in start up, they does only less number of tasks, like managing db, web.

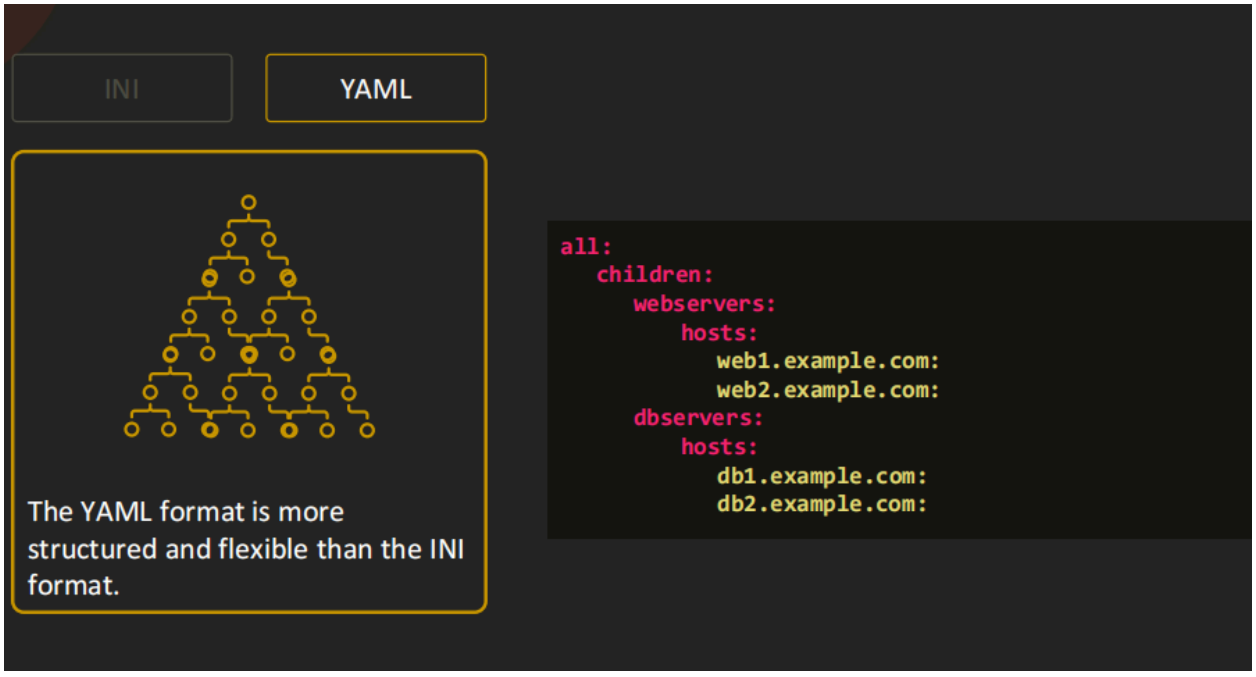
Example:

[Mail]
server1.company.com
server2.company.com

[db]
server3.company.com
server4.company.com

[web]
server5.company.com
server6.company.com

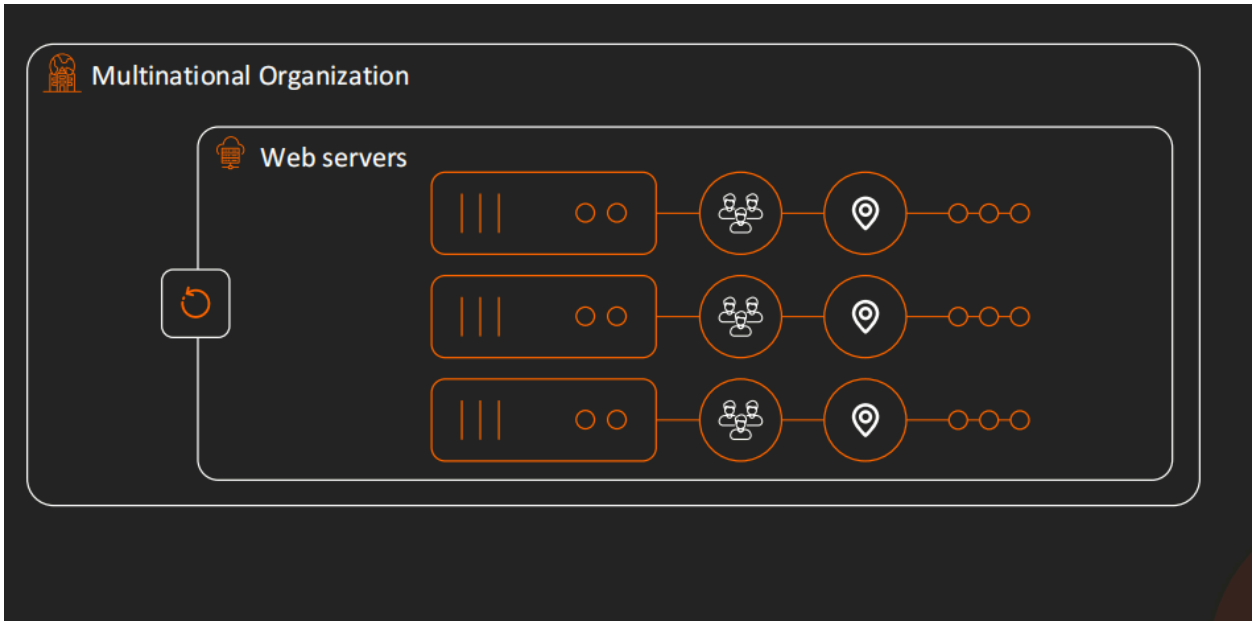
YAML:



Uses in multinational companies like to maintain multiple tasks, supports multiple apps etc
Example:

```
All:
  children:
        webserver:
              hosts:
                    web1.example.com
                    web2.example .com
```

Grouping:

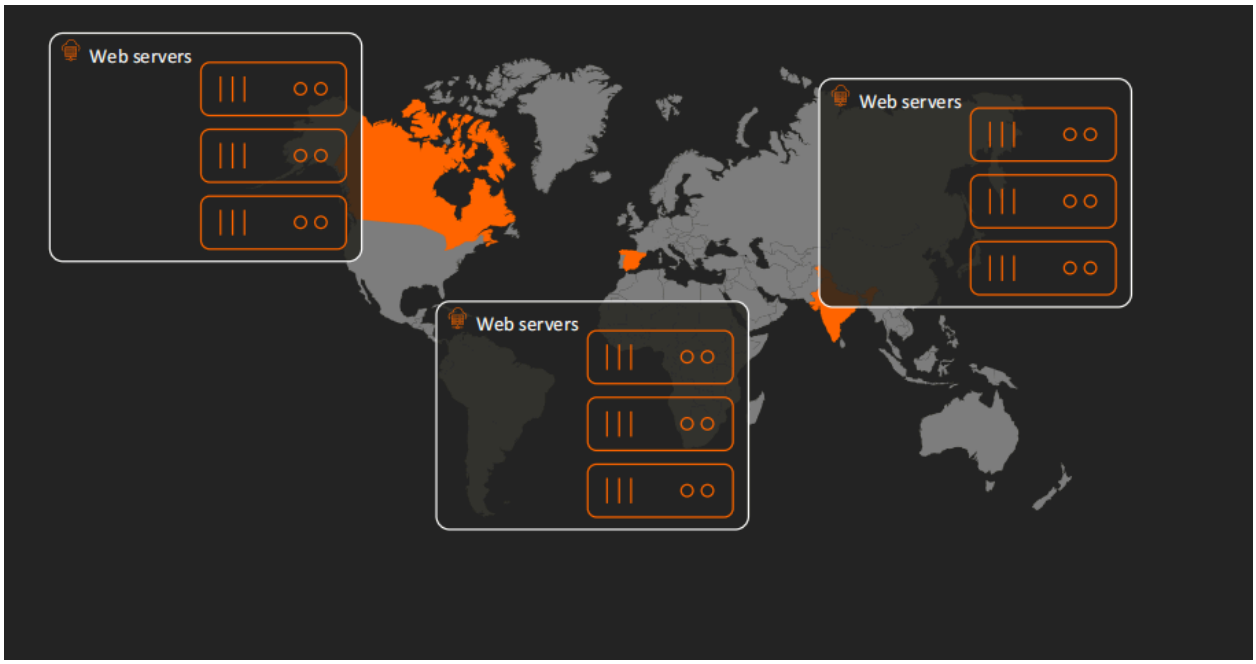


we categorize the servers based roles or locations or any other criteria is called grouping

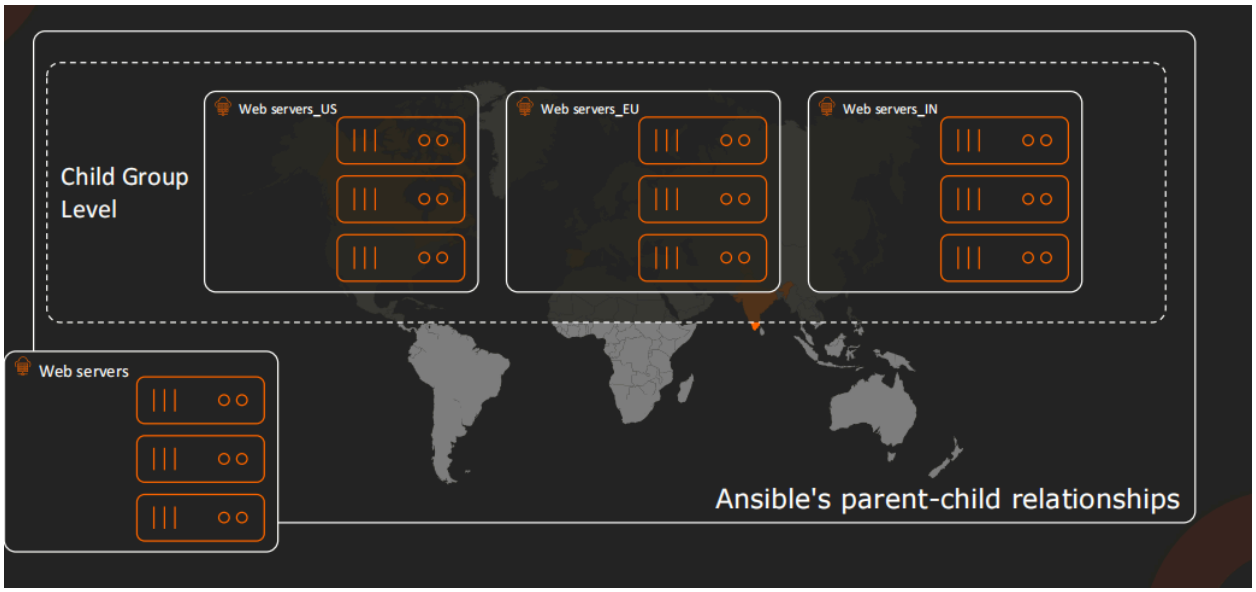
Collectively identify all the web servers under a common label named webservers
If we want to update the webservers , instead of mentioning each server we can mention/target the label name webservers then changes will apply to the all servers in that label this is called grouping this is done by ansible.

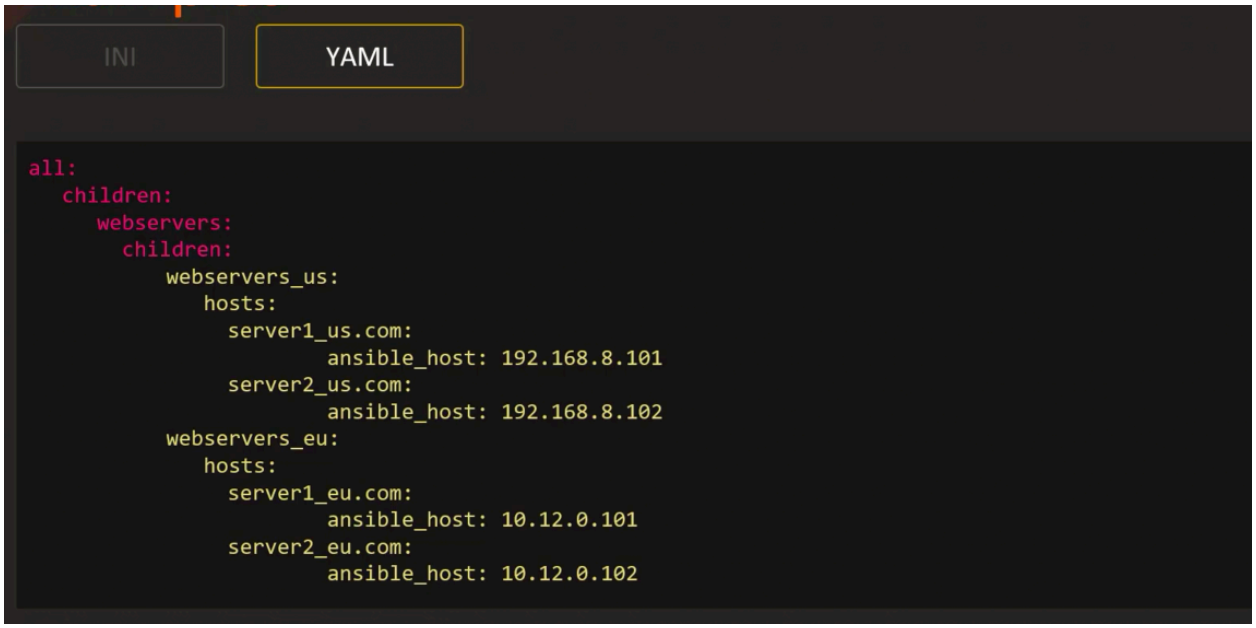
Parent -child relationship:

If we have multiple webservers in different location:



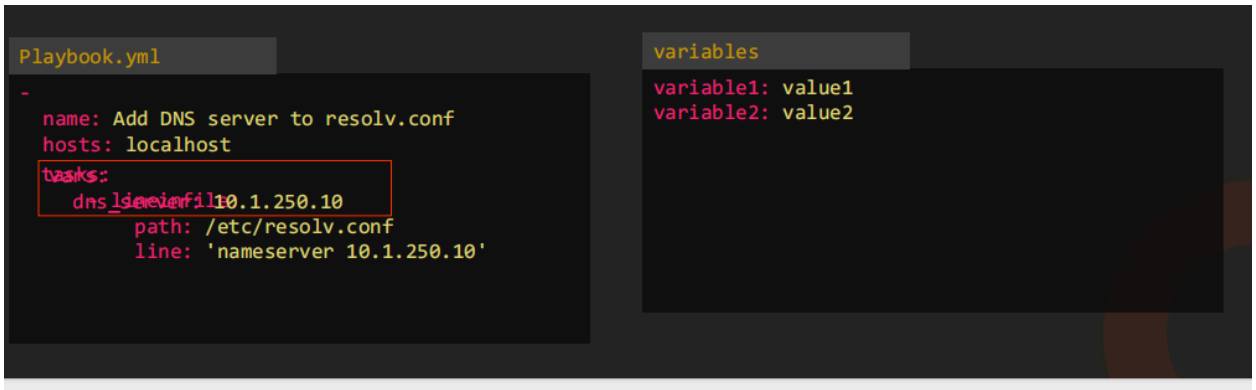
If we are having the webservers in different locations, then we create a webservers label as a parent label and listing webservers according to the location as a childs of that parent label





Ansible variables:

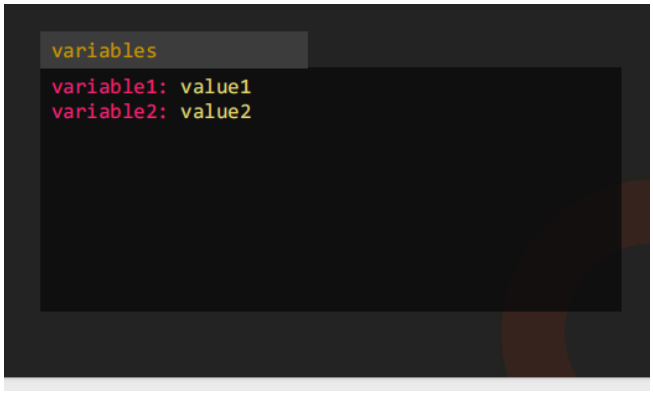
Variable stores, hostnames, username, password info

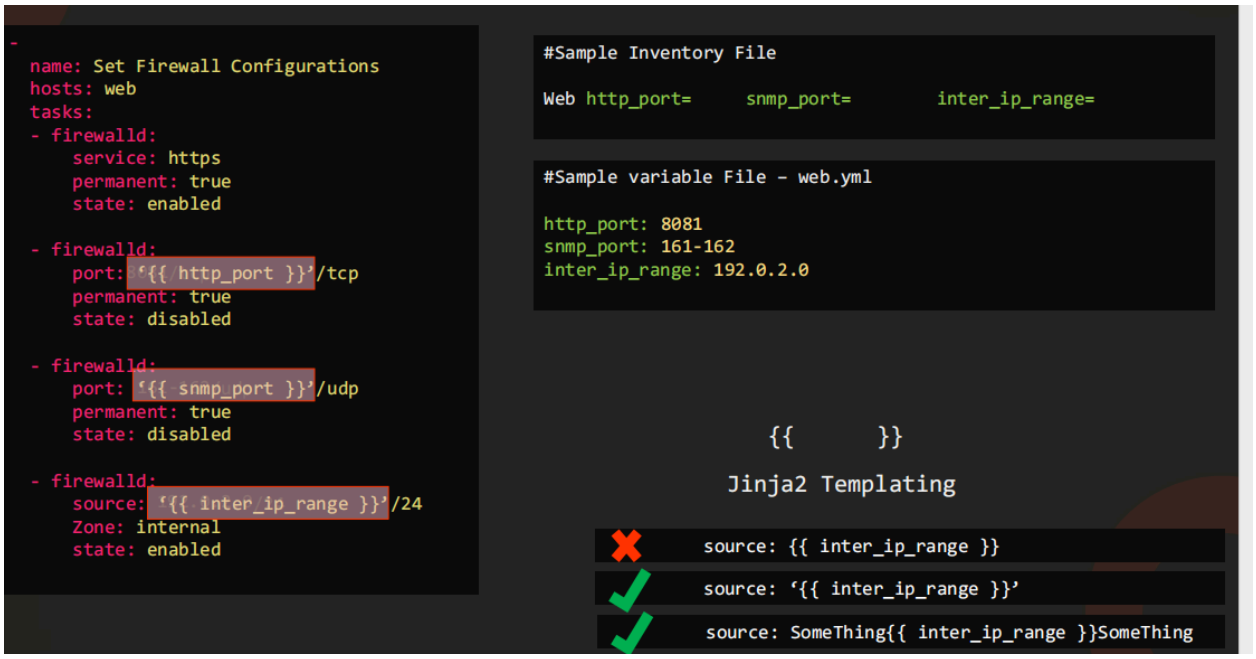


We can add vars in playbook like:

Name: Add DNS server to resolv.config
hosts:.....
Vars :
dns_server=10.1.250.10
tasks:
……..

Or we can add an another variables file separately and add variables into it:





1. We can add the variables in the inventory file and can fetch it to our playbook
2. We can also create a variable file -web.yml and add all the variables and values to that variables into that file as shown in above picture

This format of using variables in play books is called jinja2 templating.
In jinja2 technique we use :

'{{variable_name}}' ⇒ correct
{{variable_nme}} ⇒ wrong

If we mention the variable in between the sentence:

{{variable_name}} ⇒ correct

Variable types:

1. String: sequence of chars
2. Number variables: integer, float
3. Boolean: true or false
4. list
5. Dictionary

Variable presidency;

What if variable defined in two different places like a group variable in inventory file and as host variable

Example:this is the **inventory file**

```
Web1 ansible_host=172.20.1.100
Web1 ansible_host=172.20.1.102   dns_server=11.2.3.2   # declaring the dns server at the host
Web1 ansible_host=172.20.1.103

[web_servers]
web1
web2
web3

[web_servers:vars]    #group variable
dns_server=10.5.5.3
```

Defining Inside the Playbook:

```
—
  - name: configuring dns
    host: all
    var:
        dns_server:10.5.5
    tasks:
    - nsupdate:
          Server: '{{dns_server}}'
```

Precedency:

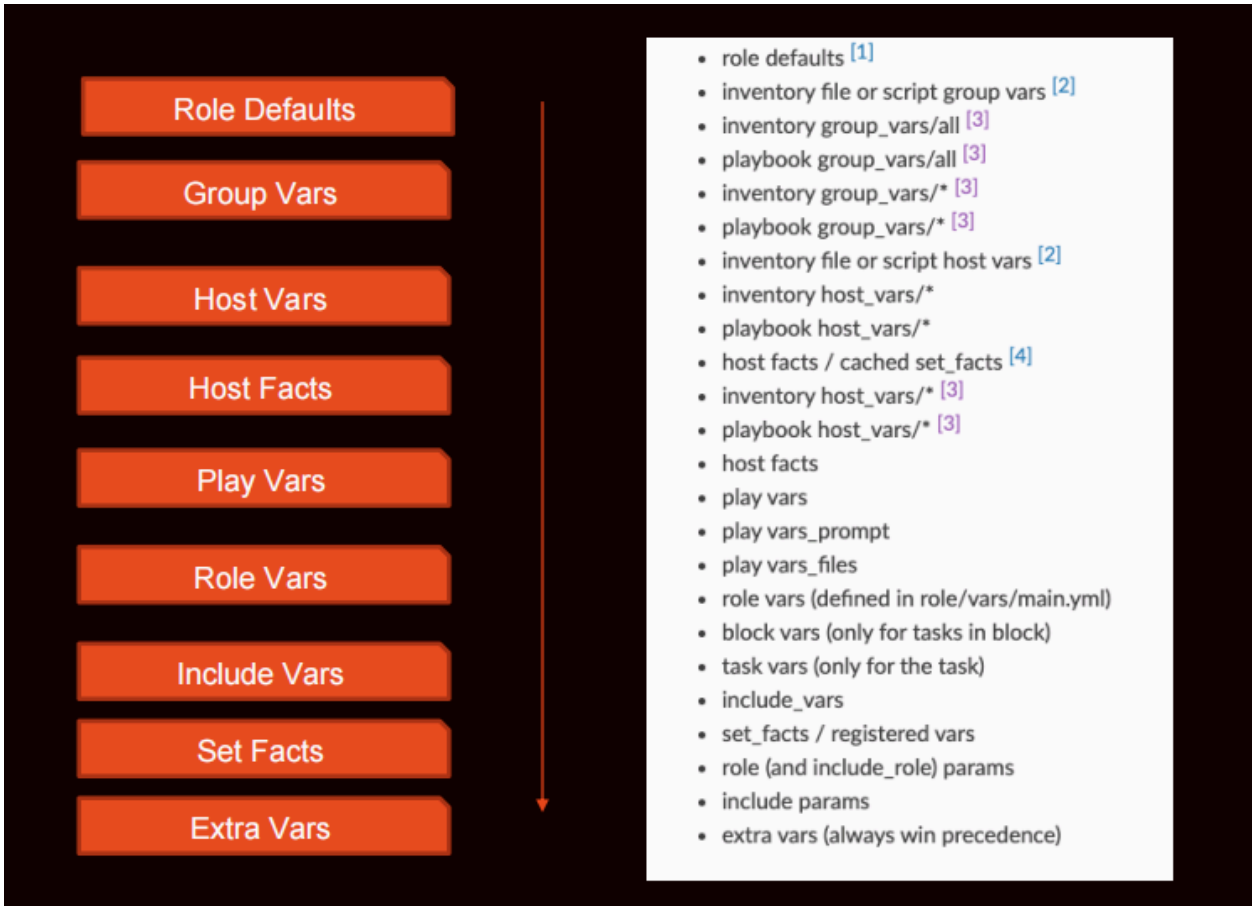
1. Group vars
2. Host vars
3. Play level
4. Extra vars option

Group var has lowest precedence
Extra var has highest precedence

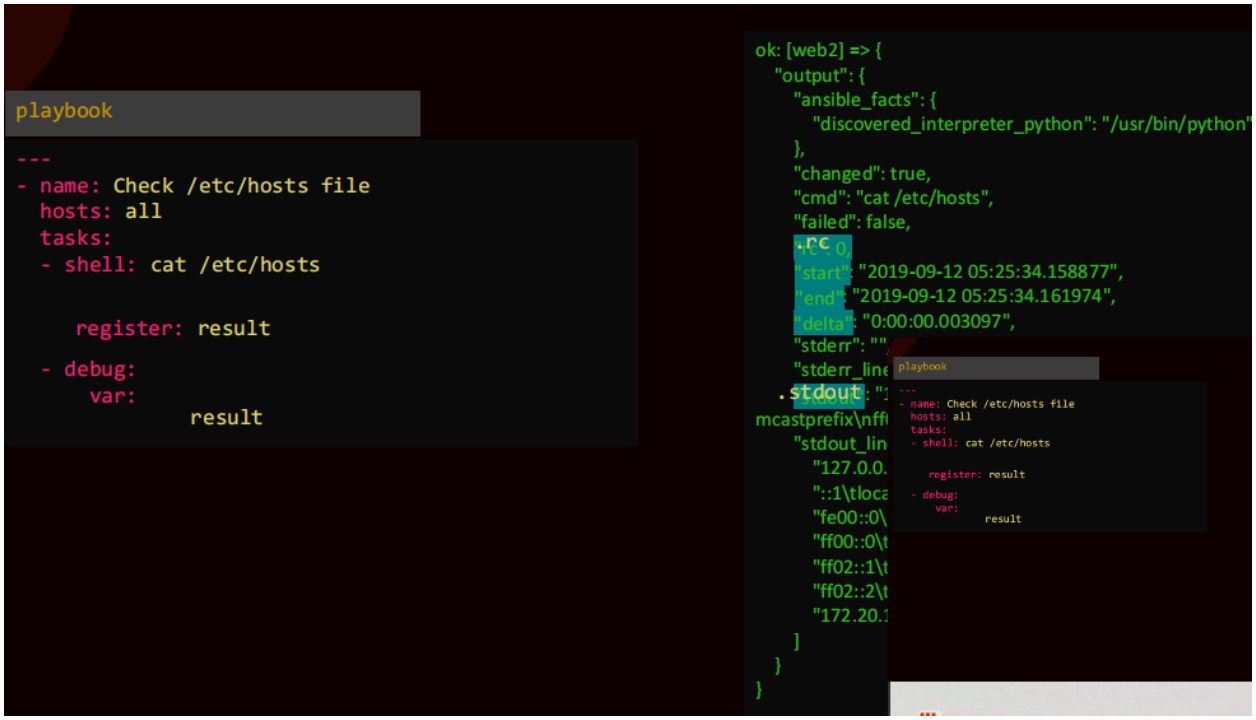
Priority increases from top to bottom, 1st it checks group var and take the value of group var if there is host var then that values is replaced with host var values and so on

Extra vars:

ansible-playbook playbook.yml –extra-vars “dns_server=10.5.5.6”



Register variable:



By using a register variable we can store the output of one task as a result and we can use that result later.

- debug:
  var: result

Result shows:

1. Here return code specified by rc , will be zero if command run successfully, if command does not run successfully then rc value will be other than zero.
2. It also shows start time and end time of command

If we want to view the task in another method without debug module , then we can pass -v option while running playbook:

Ansible-playbook -i inventory playbook.yml -v

<u>Variable scoping:</u>

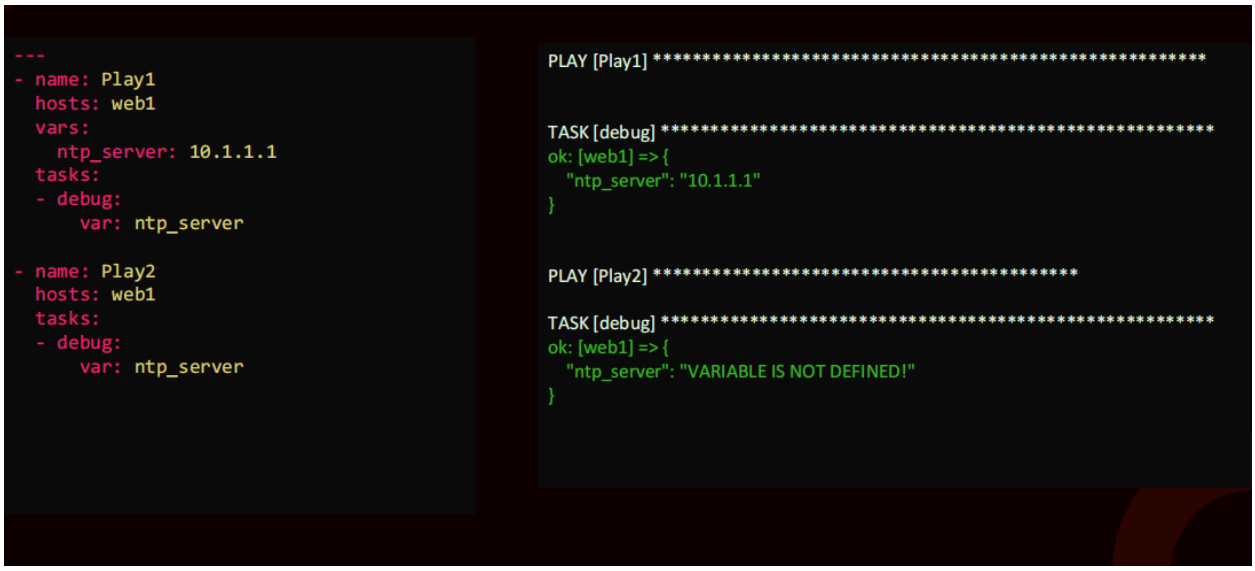
Accessibility or visibility of that variable

1. **Host scope:** if variable is defined at that host line that is accessed by that one host , not other hosts:

Eg:
web1 ansible_host=123.23.2.100
web2 ansible_host=123.23.2.102 dns_server=10.5.5.4

Here dns_server variable is applicable for only web2 host, dns_server here is a host variable.

2. **Play scope : Group variable or group of group variable**
   Here ntp_server is defined in play 1 not in play 2 so that, that ntp_server variable value does not applicable to play2



3. **Global variable**: that can be accessed throughout the playbook execution.

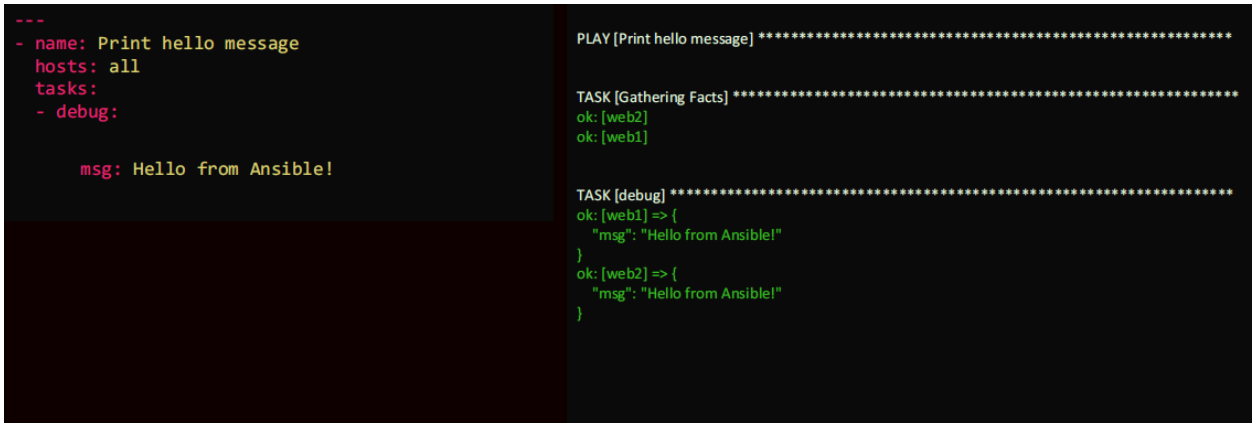
   ansible-playbook playbook.yml -- extra-vars “ntp_server=10.1.1.1.”

Ansible facts:

When ansible connects to the target machine, it first collects the information about the machine about its system information, architecture, os, processor details, host network connectivity, ip address, mac address these all information is called as **facts**.

Ansible gathers all this facts by using setup module, setup module is runned automatically by ansible, it gather facts of that hosts when you run the playbook, even if u not use this module in the playbook

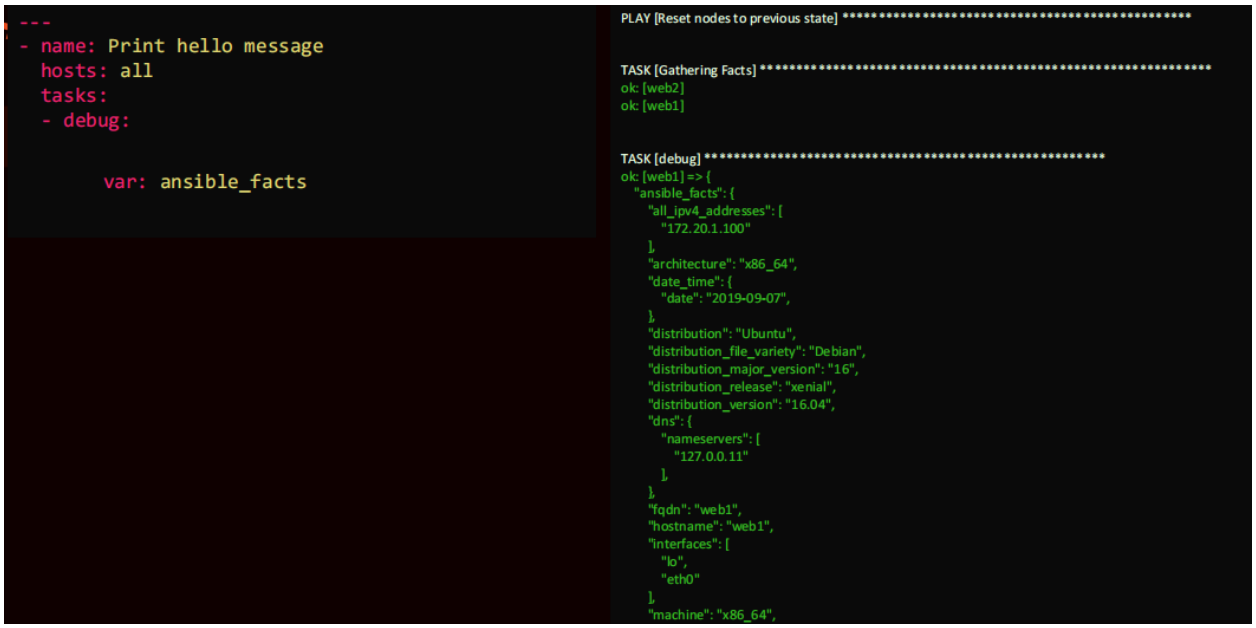
Below is a playbook to print a hello message by using debug. It does 2 tasks , 1st task is to gather the facts this gathering is done by setup module, 2nd task is to print that message



```
---
- name: Print hello message
  hosts: all
  tasks:
  - debug:

      msg: Hello from Ansible!
```

```
PLAY [Print hello message] ***************************************************

TASK [Gathering Facts] *******************************************************
ok: [web2]
ok: [web1]

TASK [debug] *****************************************************************
ok: [web1] => {
    "msg": "Hello from Ansible!"
}
ok: [web2] => {
    "msg": "Hello from Ansible!"
}
```

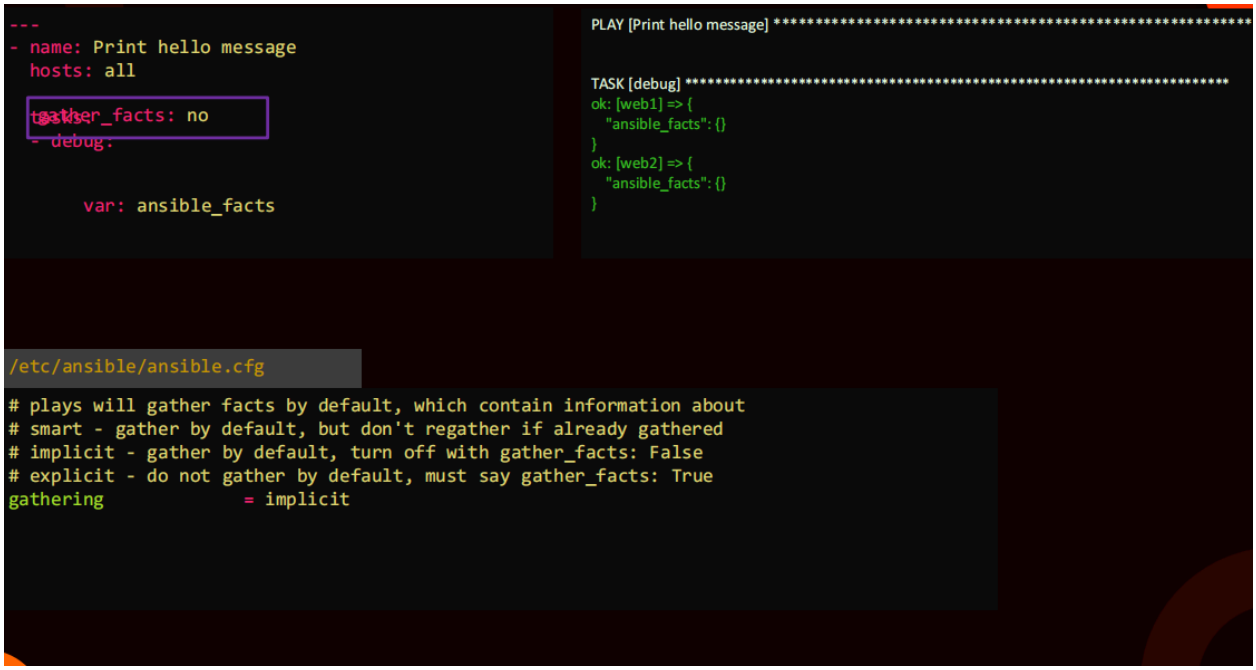
Ansible stores all the facts in **ansible_facts** variable, for displaying what are the facts gathered by ansible_facts we display that ansible_facts by using debug as follows:



```
---
- name: Print hello message
  hosts: all
  tasks:
  - debug:

      var: ansible_facts
```

```
PLAY [Reset nodes to previous state] *****************************************

TASK [Gathering Facts] *******************************************************
ok: [web2]
ok: [web1]

TASK [debug] *****************************************************************
ok: [web1] => {
    "ansible_facts": {
        "all_ipv4_addresses": [
            "172.20.1.100"
        ],
        "architecture": "x86_64",
        "date_time": {
            "date": "2019-09-07",
        },
        "distribution": "Ubuntu",
        "distribution_file_variety": "Debian",
        "distribution_major_version": "16",
        "distribution_release": "xenial",
        "distribution_version": "16.04",
        "dns": {
            "nameservers": [
                "127.0.0.11"
            ],
        },
        "fqdn": "web1",
        "hostname": "web1",
        "interfaces": [
            "lo",
            "eth0"
        ],
        "machine": "x86_64",
```

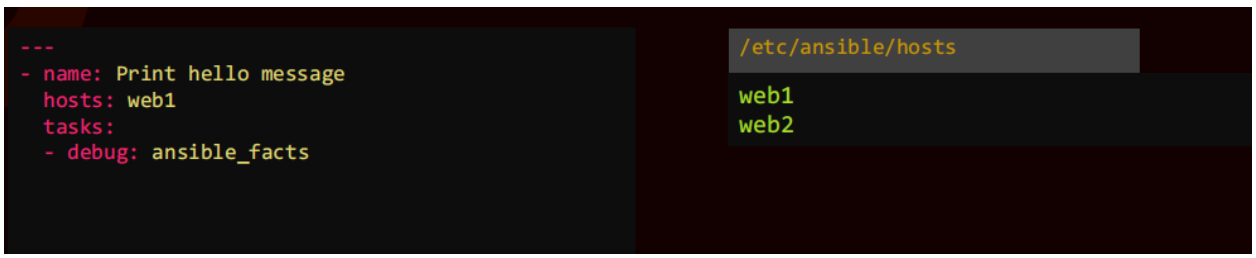
If we dont want to gather the facts by ansible_facts we can turn it off by using **gather_facts : no** in the playbook, by default it is said to implicit , means to gather info automatically, we are turning off now.



Above playbook code:

name:...
hosts:...
gather_facts:no
tasks:
- debug:
var: …

⇒ Ansible only gathers the facts against the hosts that are part of playbook, if we have an inventory file with 2 hosts web1 and web2 and in ansible playbook that targets only on the web1 then the ansible only gathers the facts of only web1 not web2.
⇒ If we don't have facts of host, then in that case that host is not targeted in the playbook so that ansible not gathered that host facts



Precedence:
What if changing the value of variable in two places in playbook and in config file, then playbook take the precedency

==========

Practice sessions:

Q: Adding servers into the list in the inventory file:

Ans:
server1.company.com
server2.company.com
server3.company.com
==========
Q: Adding alias names to db servers and web servers:

Sample Inventory File

web1 ansible_host=server1.company.com
web2 ansible_host=server2.company.com
web3 ansible_host=server3.company.com
db1 ansible_host=server4.company.com
~
Q: adding a server alias name and host details, connection details etc as follows:

```
-------------------------------------------------------------------------------------------
|  Alias |        HOST         | Connection | User          | Password     |
-------------------------------------------------------------------------
|  web1  | server1.company.com |    ssh     | root          | Password123! |
-------------------------------------------------------------------------
|  web2  | server2.company.com |    ssh     | root          | Password123! |
-------------------------------------------------------------------------
|  web3  | server3.company.com |    ssh     | root          | Password123! |
-------------------------------------------------------------------------
|  db1   | server4.company.com |    winrm   | administrator | Dbp@ss123!   |
-------------------------------------------------------------------------------------------
```

Ans:

Sample Inventory File

Web Servers
web1 ansible_host=server1.company.com ansible_connection=ssh ansible_user=root ansible_ssh_pass=Password123!
web2 ansible_host=server2.company.com ansible_connection=ssh ansible_user=root ansible_ssh_pass=Password123!
web3 ansible_host=server3.company.com ansible_connection=ssh ansible_user=root ansible_ssh_pass=Password123!
db1 ansible_host=server4.company.com ansible_connection=winrm ansible_user=administrator ansible_password=Dbp@ss123!

==========

Q: adding a group for web servers as web_servers and similarly add an another group db_servers that contains list of database servers:

Ans: # Sample Inventory File

Web Servers
web1 ansible_host=server1.company.com ansible_connection=ssh ansible_user=root ansible_ssh_pass=Password123!
web2 ansible_host=server2.company.com ansible_connection=ssh ansible_user=root ansible_ssh_pass=Password123!
web3 ansible_host=server3.company.com ansible_connection=ssh ansible_user=root ansible_ssh_pass=Password123!

Database Servers
db1 ansible_host=server4.company.com ansible_connection=winrm ansible_user=administrator ansible_password=Password123!

[web_servers]
web1
web2
web3

[db_servers]
db1

==========
Q: Create a new group called all_servers and add the previously created groups web_servers and db_servers under it.

[parent_group:children]
child_group1
child_group2

Ans: # Sample Inventory File

Web Servers
web1 ansible_host=server1.company.com ansible_connection=ssh ansible_user=root ansible_ssh_pass=Password123!
web2 ansible_host=server2.company.com ansible_connection=ssh ansible_user=root ansible_ssh_pass=Password123!
web3 ansible_host=server3.company.com ansible_connection=ssh ansible_user=root ansible_ssh_pass=Password123!

Database Servers
db1 ansible_host=server4.company.com ansible_connection=winrm ansible_user=administrator ansible_password=Password123!

[web_servers]
web1
web2
web3

[db_servers]
db1

[all_servers:children]
web_servers
db_servers

==========

Q: Update the /home/bob/playbooks/inventory file to represent the data given in the below table in Ansible Inventory format.

Server Alias	Server Name	OS	User	Password
sql_db1	sql01.xyz.com	Linux	root	Lin$Pass
sql_db2	sql02.xyz.com	Linux	root	Lin$Pass
web_node1	web01.xyz.com	Win	administrator	Win$Pass
web_node2	web02.xyz.com	Win	administrator	Win$Pass
web_node3	web03.xyz.com	Win	administrator	Win$Pass

Group the servers together based on this table:

Group	Members
db_nodes	sql_db1, sql_db2
web_nodes	web_node1, web_node2, web_node3
boston_nodes	sql_db1, web_node1
dallas_nodes	sql_db2, web_node2, web_node3
us_nodes	boston_nodes, dallas_nodes

Ans:

Sample Inventory File

Web Servers

```
web_node1 ansible_host=web01.xyz.com ansible_connection=winrm
ansible_user=administrator ansible_password=Win$Pass
web_node2 ansible_host=web02.xyz.com ansible_connection=winrm
ansible_user=administrator ansible_password=Win$Pass
web_node3 ansible_host=web03.xyz.com ansible_connection=winrm
ansible_user=administrator ansible_password=Win$Pass

# DB Servers
sql_db1 ansible_host=sql01.xyz.com ansible_connection=ssh ansible_user=root
ansible_ssh_pass=Lin$Pass
sql_db2 ansible_host=sql02.xyz.com ansible_connection=ssh ansible_user=root
ansible_ssh_pass=Lin$Pass

[db_nodes]
sql_db1
sql_db2

[web_nodes]
web_node1
web_node2
web_node3

[boston_nodes]
sql_db1
web_node1

[dallas_nodes]
sql_db2
web_node2
web_node3

[us_nodes:children]
boston_nodes
dallas_nodes
```

==========

Magic variables:

1. **hostvars**
2. **Groups**

3. **Group_names**
4. **inventory_hostname**

Hostvars - magic variable:

1. When we write the alias to some hosts, ansible 1st creates the sub process to each host
2. Undergoes variable interpolation where it picks the variables from different sources and associates to their respective hosts
3. Here the dns server is defined only for web2 ,not defined for web1 and web3

4. How can web1 and web3 get the dns server ip address specified to web2 host, it is done by using **magic variables**

   '{{hostvars['web2'].dns_server}}'

5. To get the hostname or ip address of other host then use ansible_host:

   '{{hostvars['web2'].ansible_host}}'

6. If we want to access the facts(like architecture,devices,mounts) of other hosts we use the following:

‘{{hostvars[‘web2’].ansible_facts.architecture}}’

‘{{hostvars[‘web2’].ansible_facts.devices}}’

‘{{hostvars[‘web2’].ansible_facts.mounts}}’

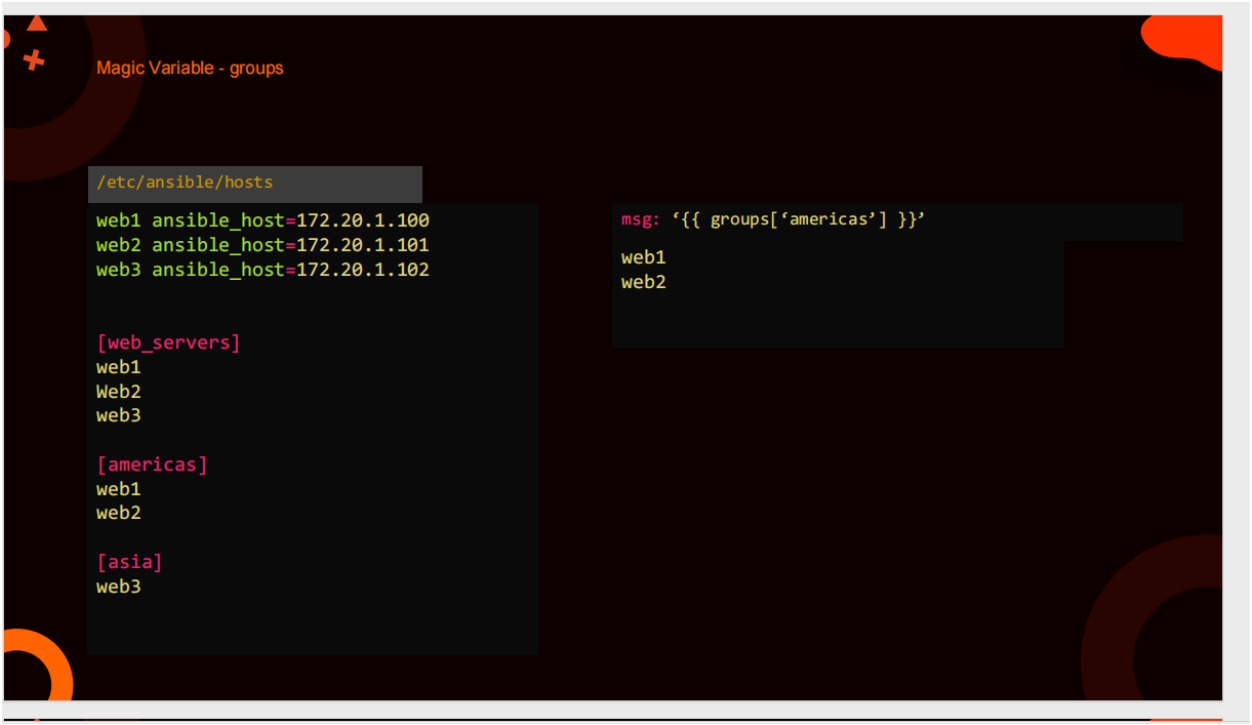
‘{{hostvars[‘web2’].ansible_facts.processor}}’

7. Instead of above line we can also write as follows :

   ‘{{hostvars[‘web2’][‘ansible_facts’][processor’]}}’

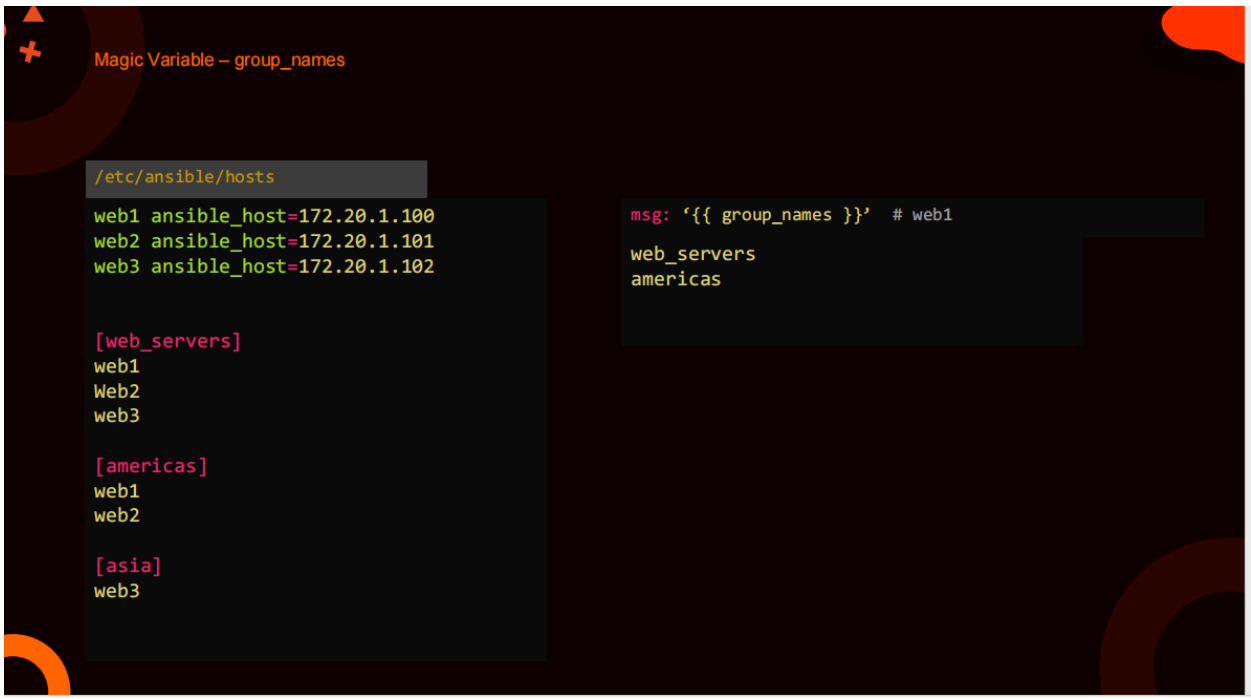
Groups - magic variable:

1. Groups return all the hosts under the given group.



Group_names - magic variable:

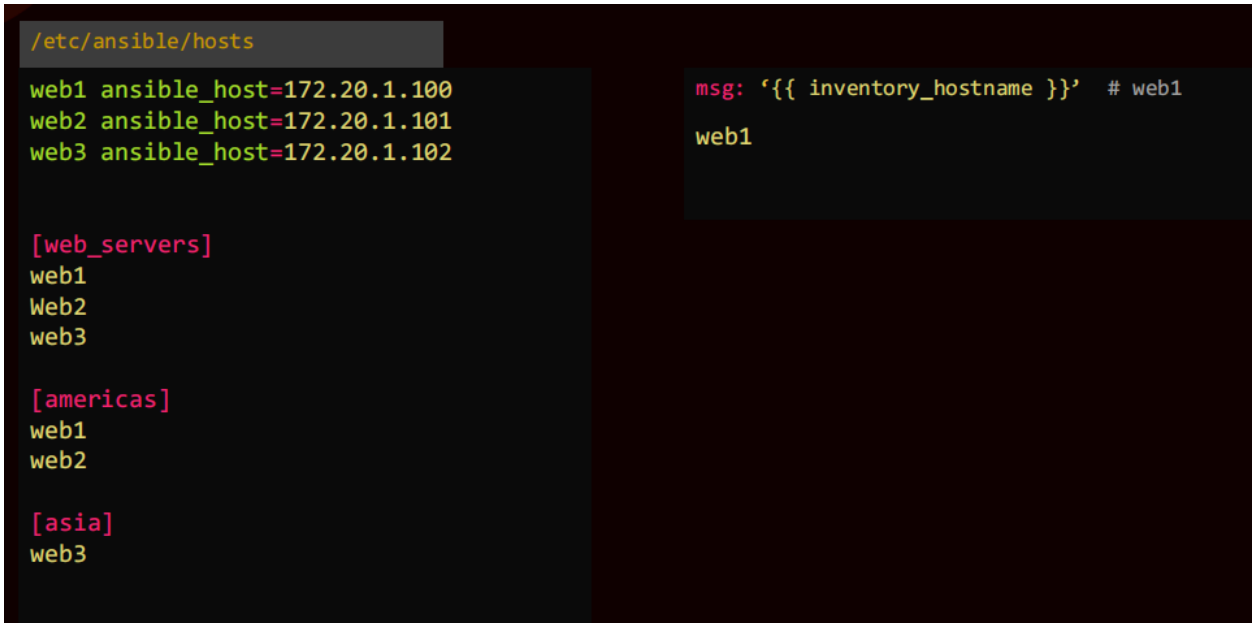
It displays all group names where the particular host is part of(or)
It returns all the group names the current host is part of



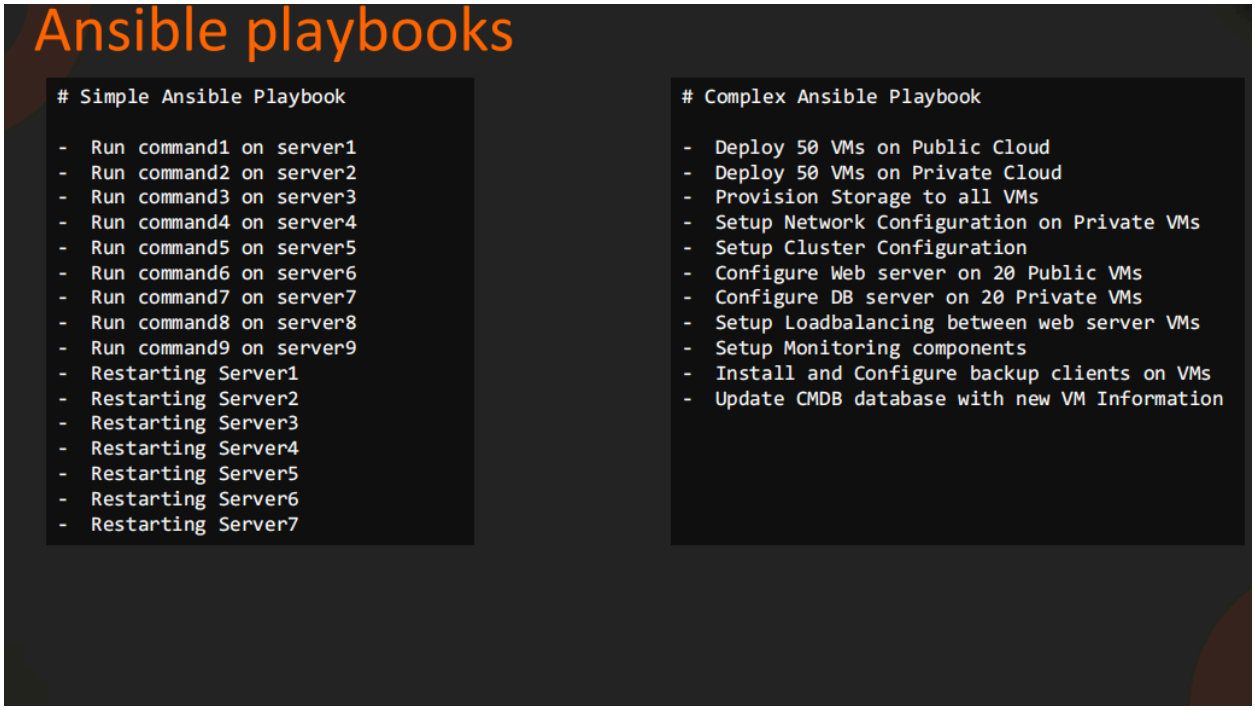
Here the above picture shows that, displaying the group names where the host1 is present

Inventory_hostname - magic variable:

It gives you the name configured for the host in the inventory file and not the hostname or SQDN



Ansible playbooks: It is set of instruction to the ansible what to do



1. Playbooks are written YAML format
2. A playbook is a single YAML file that contains set of plays
3. A play is the set of tasks to be done on the hosts
4. Task is the action like execute the command, run the script, install a package, shutdown/restart

Here are some sample playbooks contains:

Name, host, tasks are properties of dictionary and order of it doesn't matter , we can swap name and host

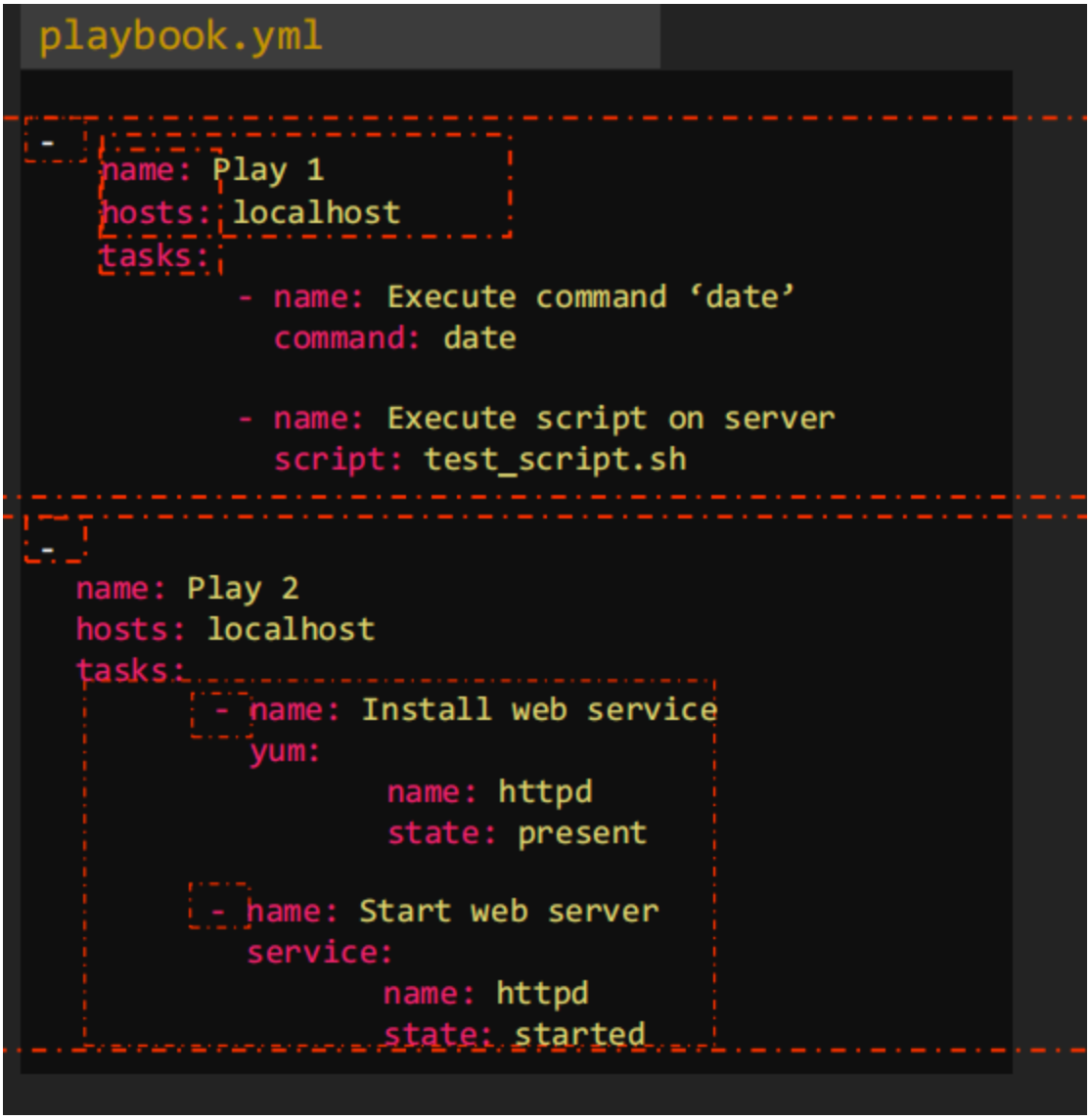
Name: contains play,
hosts: tells where the task should be done like in which server,
Tasks: we should write the actions that to be performed

1. playbook.yml

```
-
  name: Play 1
  hosts: localhost
  tasks:
   - name: Execute command 'date'
     command: date

   - name: Execute script on server
     script: test_script.sh

   - name: Install httpd service
     yum:
      name: httpd
      state: present

   - name: Start web server
     service:
      name: httpd
      state: started
```

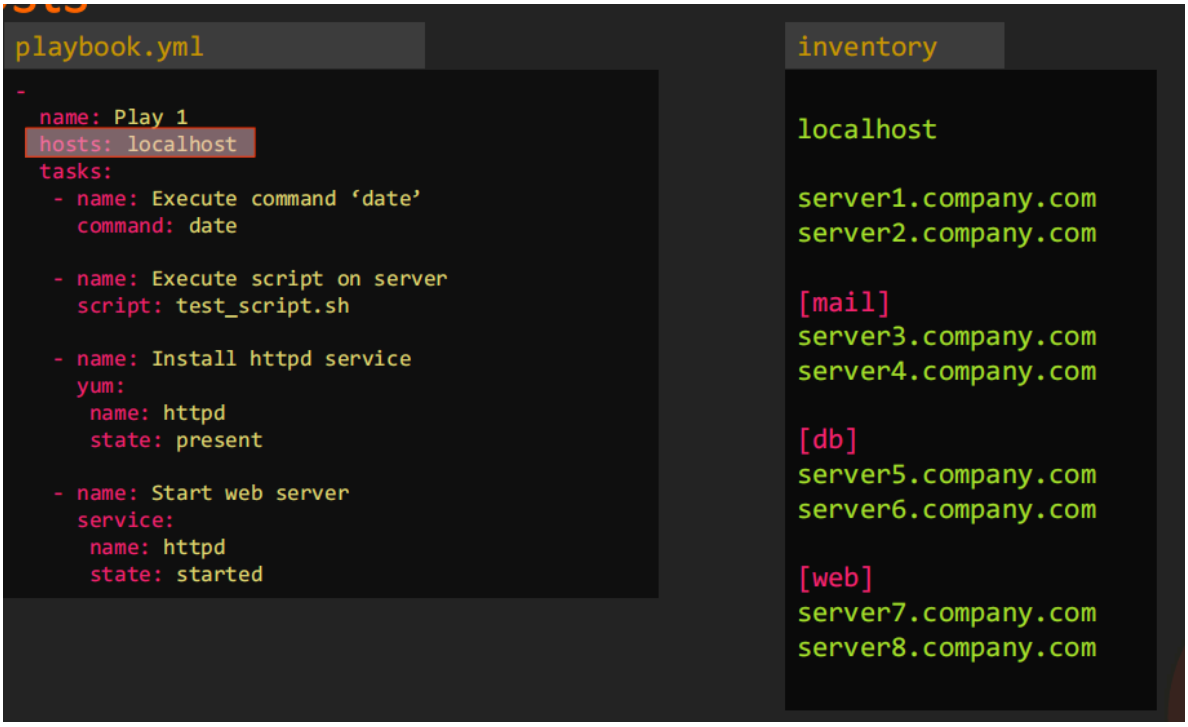
Above playbook contains only one play, the tasks should be done in the local host and tasks are displaying date, executing the test_script.sh and installing the httpd service and restarting the server

2. playbook.yml



In the 2nd YAML file it contains 2 plays, written as a list, so that the playbook is list dictionaries, each play is a dictionary and set of properties like name, host, tasks.

Order of hosts and name can be swapped
But order of the list cant be swapped

3. 

In above playbook and inventory file, it is mandatorily to specify the host where we need to perform the tasks, in inventory file 1st we have to mention the host name where we need to perform tasks in the first line

Before mentioning the host in playbook , we should ensure that the hosts should present in the inventory file we created earlier

hosts defined in the inventory file must match the host used in the playbook and all connection information can be retrieved from the inventory file

Modules:

Different actions run by tasks is called as modules

In the above case like in 3rd playbook, we can say the following as the modules: Command, script, yum, service

There are so many modules: to know about all modules we use following command **ansible-doc-l**

To execute the playbook

Syntax:
ansible-playbook <playbook file name>
Eg: ansible-playbook playbook.yml
ansible-playbook --help

Verifying the playbooks:
Why?

Verifying the playbook is a critical step before deploying updates to production, ensuring that the Ansible automation functions correctly. If it doesn't run properly, it could lead to serious consequences such as system shutdowns and data loss, which may be irreversible.

How:

1. Check mode
2. Diff mode

Check mode:

1. Check mode is a dry run mode where the ansible executes the playbook without making any actual changes on the hosts
2. It allows you to preview of playbook what changes made by playbook without applying them
3. To run the playbook in check mode use **--check option**

install_nginx.yml

```
---

- hosts: webservers
  tasks:
    - name: Ensure nginx is installed
      apt:
        name: nginx
        state: present
      become: yes
```

```
$ ansible-playbook install_nginx.yml --check

- PLAY [webservers] ***********************************************************

  TASK [Gathering Facts] *****************************************************
  ok: [webserver1]

  TASK [Ensure nginx is installed] *******************************************
  changed: [webserver1]

  PLAY RECAP
- *****************************************************************
  webserver1: ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Diff mode:

- This mode shows the differences between the current state and the state after the playbook is run
- Provides before and after comparison useful for understanding what changes happen
- To run in diff mode run the playbook use **--diff option**
-

configure_nginx.yml

```
---

- hosts: webservers
  tasks:
    - name: Ensure the configuration line is present
      lineinfile:
        path: /etc/nginx/nginx.conf
        line: 'client_max_body_size 100M;'
      become: yes
```

```
$ ansible-playbook configure_nginx.yml --check --diff

- PLAY [webservers] ************************************************************
  TASK [Gathering Facts] ********************************************************
  ok: [webserver1]
  TASK [Ensure the configuration line is present] *******************************
  --- before: /etc/nginx/nginx.conf (content)
  +++ after: /etc/nginx/nginx.conf (content)
  @@ -20,3 +20,4 @@
  # some existing configuration lines
  # more existing configuration lines
  #
  +client_max_body_size 100M;
  changed: [webserver1]
  PLAY RECAP
- ********************************************************************
  webserver1: ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

In additional to check mode and diff mode ansible also give syntax check mode

Syntax check mode:

- checks the syntax of playbook for any errors
- To run the playbook in syntax check mode use **--syntax-check option**
- If there is no error in the play book it displays as follows:

```
configure_nginx.yml

---

- hosts: webservers
  tasks:
    - name: Ensure the configuration line is present
      lineinfile:
        path: /etc/nginx/nginx.conf
        line: 'client_max_body_size 100M;'
      become: yes


$ ansible-playbook configure_nginx.yml --syntax-check
playbook: configure_nginx.yml
```

- If there is any error it displays as follows, display at what place exactly the error occurred:

If we remove the colon after lineinfile line, we get the following error:

configure_nginx.yml

```
---
- hosts: webservers
  tasks:
    - name: Ensure the configuration line is present
      lineinfile
        path: /etc/nginx/nginx.conf
        line: 'client_max_body_size 100M;'
      become: yes
```

":" is missing

```
$ ansible-playbook configure_nginx.yml --syntax-check

ERROR! Syntax Error while loading YAML.
  did not find expected key

The error appears to be in '/path/to/configure_nginx.yml': line 5, column 9, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

      lineinfile
        path: /etc/nginx/nginx.conf
        ^ here
```

Ansible-Lint:

If we are running a large Organization we should work on large data , we will have complex tasks in the playbooks then it is very difficult to know the errors where it occurs.

Why Do We Need ansible-lint?

Ansible Lint is a command-line tool that performs linting on Ansible playbooks, roles, and collections.

It checks your code for potential errors, bugs, stylistic errors, and suspicious constructs.

It's akin to having a seasoned Ansible mentor guiding you, providing valuable insights, and catching issues that might have slipped past your notice.

style_example.yml

```yaml
- name: Style Example Playbook
  hosts: localhost
  tasks:
    - name: Ensure nginx is installed and started
      apt:
        name: nginx
        state: latest
        update_cache: yes

    - name: Enable nginx service at boot

service:
        name: nginx
        enabled: yes
        state: started
- name: Copy nginx configuration file
      copy:
        src: /path/to/nginx.conf
        dest: /etc/nginx/nginx.conf
      notify:
        - Restart nginx service

  handlers:
    - name: Restart nginx service
      service:
        name: nginx
        state: restarted
```

```
$ ansible-lint style_example.yml
[WARNING]: incorrect indentation: expected 2 but found 4 (syntax/indentation)
style_example.yml:6

[WARNING]: command should not contain whitespace (blacklisted: ['apt']) (commands)
style_example.yml:6

[WARNING]: Use shell only when shell functionality is required (deprecated in favor of 'cmd') (commands)
style_example.yml:6

[WARNING]: command should not contain whitespace (blacklisted: ['service']) (commands)
style_example.yml:12

[WARNING]: 'name' should be present for all tasks (task-name-missing) (tasks)
style_example.yml:14
```

==========

Server client model: ssh is a server client model
Agent: We need to install an agent on the server that we want to monitor or secure.

- if we want to connect to other server and know all the things happening in that server, all details we need to install agent in that server
- if there is any threats occurs also this agent gives information about it and acts as security
- Agent also act as a backup, we can restore the backup information. If we want all details that happened in a day on that server, we can store all those details as a backup and can retrieve it at the end of the day or later.

==========

Lab practise:

Q. The /home/bob/playbooks/playbook.yaml playbook is adding a name server entry in /tmp/resolv.conf sample file on localhost. The name server information is already added to the /home/bob/playbooks/inventory file as a variable called nameserver_ip.

Replace the hardcoded ip address of the name server in this playbook to use the value from the variable defined in the inventory file.

ans:

```
---
- name: 'Add nameserver in resolv.conf file on localhost'
  hosts: localhost
  become: yes
  tasks:
    - name: 'Add nameserver in resolv.conf file'
      lineinfile:
        path: /tmp/resolv.conf
        line: 'nameserver {{nameserver_ip}}'
```

Q2. We have updated the /home/bob/playbooks/playbook.yaml playbook to add a new task to disable SNMP port on localhost. However, the port is hardcoded in the playbook. Update the playbook to replace the hardcoded value of the SNMP port to use the value from the variable named snmp_port, defined in the inventory file.

Ans:
```
---
- name: 'Add nameserver in resolv.conf file on localhost'
  hosts: localhost
  become: yes
  tasks:
    - name: 'Add nameserver in resolv.conf file'
      lineinfile:
        path: /tmp/resolv.conf
        line: 'nameserver {{  nameserver_ip  }}'
    - name: 'Disable SNMP Port'
      firewalld:
        port: '{{snmp_port}}'
        permanent: true
        state: disabled
```

Q. We have reset the /home/bob/playbooks/playbook.yaml playbook. Its printing some personal information of an employee. We would like to move the car_model, country_name and title values to the respective variables, and these variables should be defined at the play level.

Add three new variables named car_model, country_name and title under the play and move the values over there. Use these variables within the task to remove the hardcoded values.

Ans:

```
---
- hosts: localhost
  vars:
      car_model: 'BMW M3'
      country_name: USA
      title: 'Systems Engineer'
  tasks:
    - command: 'echo "My car is {{car_model}}"'
    - command: 'echo "I live in the {{country_name}}"'
    - command: 'echo "I work as a {{title}}"'
```

==========

Q. The /home/bob/playbooks/app_install.yaml playbook is responsible for installing a list of packages on a remote server(s). The list of packages to be installed is already added to the /home/bob/playbooks/inventory file as a list variable called app_list.

Right now the list of packages to be installed is hardcoded in the playbook. Update the /home/bob/playbooks/app_install.yaml playbook to replace the hardcoded list of packages to use the values from the app_list variable defined in the inventory file. Once updated, please run the playbook once to make sure it works fine.

Ans:

```
---
- hosts: all
  become: yes
  tasks:
    - name: Install applications
      yum:
        name: "{{ item }}"
        state: present
      with_items:
        - "{{ app_list }}"
```

Q. The /home/bob/playbooks/user_setup.yaml playbook is responsible for setting up a new user on a remote server(s). The user details like username, password, and email are

already added to the /home/bob/playbooks/inventory file as a dictionary variable called user_details.

Right now the user details is hardcoded in the playbook. Update the /home/bob/playbooks/user_setup.yaml playbook to replace the hardcoded values to use the values from the user_details variable defined in the inventory file. Once updated, please run the playbook once to make sure it works fine.

Ans:

```
---
- hosts: all
  become: yes
  tasks:
    - name: Set up user
      user:
        name: "{{user_details.username}}"
        password: "{{user_details.password}}"
        comment: "{{user_details.email}}"
        state: present
```

==========

Ansible playbook lab:

Q. Update the playbook /home/bob/playbooks/playbook.yaml to add a task name Task to display hosts file for the existing task.

Ans:

```
---
- name: 'Execute a command to display hosts file on localhost'
  hosts: localhost
  become: yes
  tasks:
    - name: 'Task to display hosts file'
      command: 'cat /etc/hosts'
```

Q. We have reset the playbook /home/bob/playbooks/playbook.yaml, now update it to add another task. The new task must execute the command cat /etc/resolv.conf and set its name to Task to display nameservers.

Ans:

```
---
- name: 'Execute two commands on localhost'
  hosts: localhost
  become: yes
  tasks:
    - name: 'Execute a date command'
      command: date
    - name: 'Task to display nameservers.'
      command: 'cat /etc/resolv.conf'
```

Q. Update the playbook by adding a new playbook for node02 to display cat /etc/hosts

```
---
- name: 'Execute two commands on node01'
  hosts: node01
  become: yes
  tasks:
    - name: 'Execute a date command'
      command: date
    - name: 'Task to display hosts file on node01'
      command: 'cat /etc/hosts'
- name: 'Execute a command on node02'
  hosts: node02
  tasks:
    - name: 'Task to display hosts file on node02'
      command: 'cat /etc/hosts'
```

==========

Conditionals:

⇒ We will use “**when”** conditional statement to specify the condition for each tasks:
Eg:

```
- - -
- name: install NGIN
  hosts: all
```

Tasks:

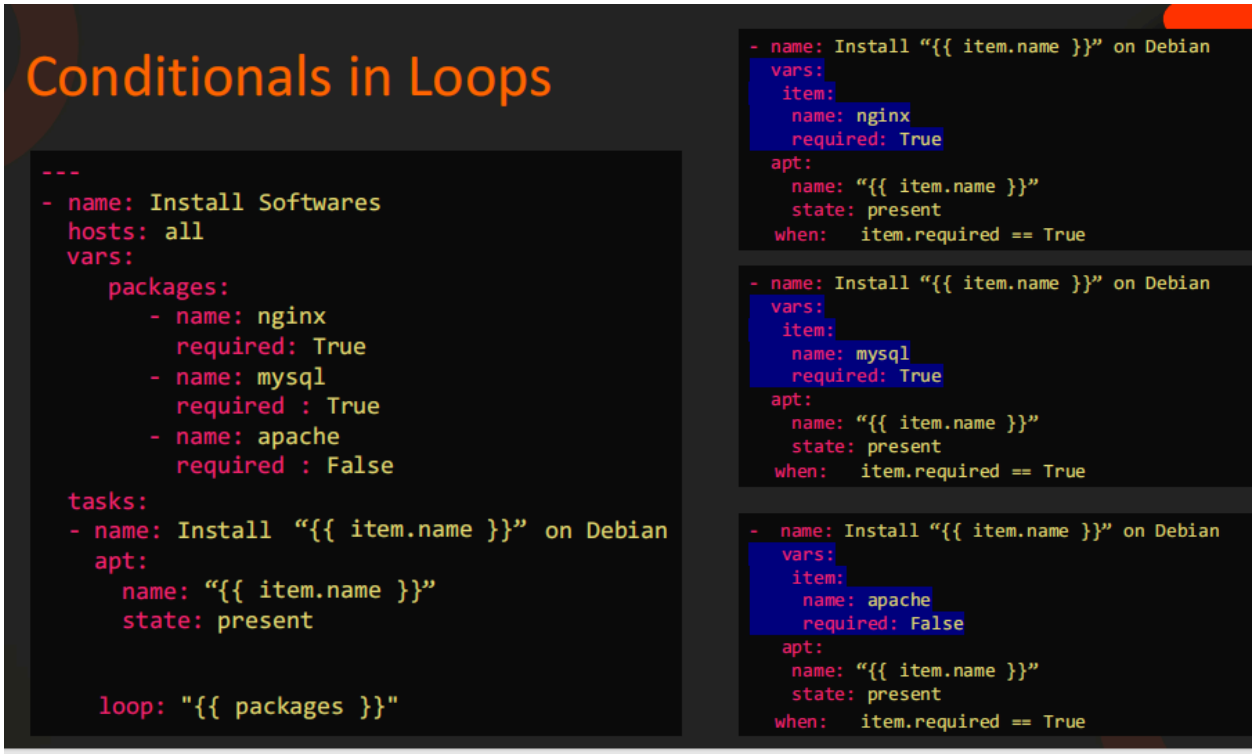
```
- name: install NGINX on debian
  apt:
    name: nginx
    state: present
  when: <<condition>>
```

⇒We will use **or operator** and **and operator** in the condition

Eg2:

```
- name: install NGINX on debian
  apt:
    name: nginx
    state: present
  when: ansible_os_family==”Debian” and ansible_distribution_version==”16.04”
- name: install NGINX on redhat
  yum:
    name: nginx
    state: present
  when: ansible_os_family==’”Redhat” or ansible_os_family== “SUSE”
```

⇒If we want to install array of packages:



Everything is inside the item so that we will write item.name to take name , for every iteration we are using item

- First, we initiate a loop through the directory to install tasks sequentially.
- We are iterating over the packages so that each item is executed one after the other.
- For example, the first loop installs Nginx since the requirement is set to true.
- The next loop installs MySQL, and then the loop encounters Apache, which is marked as false, preventing its installation and stopping the loop.

This loop get executed only **when item.required == true**

```yaml
---
- name: Install Softwares
  hosts: all
  vars:
     packages:
        - name: nginx
          required: True
        - name: mysql
          required : True
        - name: apache
          required : False
  tasks:
  - name: Install “{{ item.name }}” on Debian
    apt:
      name: “{{ item.name }}”
      state: present
    when:   item.required == True
    loop: "{{ packages }}"
```

To use conditional with output of previous task we use register
If we want to check the status of httpd , if httpd is down then we have to send a mail that service is down

```yaml
- name: Check status of a service and email if its down
  hosts: localhost
  tasks:
    - command: service httpd status
      register: result

    -  mail:
        to: admin@company.com
        subject: Service Alert
        body: Httpd Service is down
        when: result.stdout.find('down') != -1
```

The `find` function checks for the presence of "down" in the results. If it finds "down," it returns the index of that occurrence; if not, it returns -1. An email will be sent only if the result is not -1.

Ansible conditionals based on facts variables reuse:

If you want to perform an action in different servers like in windows, centos etc
Scenario1: Conditional based on facts

If we want to install something like NGINX into a particular server

```
- name: Install Nginx on Ubuntu 18.04
  apt:
    name: nginx=1.18.0
    state: present
  when: ansible_facts['os_family'] == 'Debian' and ansible_facts['distribution_major_version'] == '18'
```

Here ansible_facts collects all the details of that server like system information etc and it checks that , is it debian and version is 18, then only it gets installed

Scenario 2: Conditionals based on variables

If your web application have different requirements based on environment, then we can define a variable(app_env) and specify that environment as a value to the variable, then in playbook we can use this variable to deploy the appropriate config file for the specified environment.

```
- name: Deploy configuration files
  template:
    src: "{{ app_env }}_config.j2"
    dest: "/etc/myapp/config.conf"
  vars:
    app_env: production
```

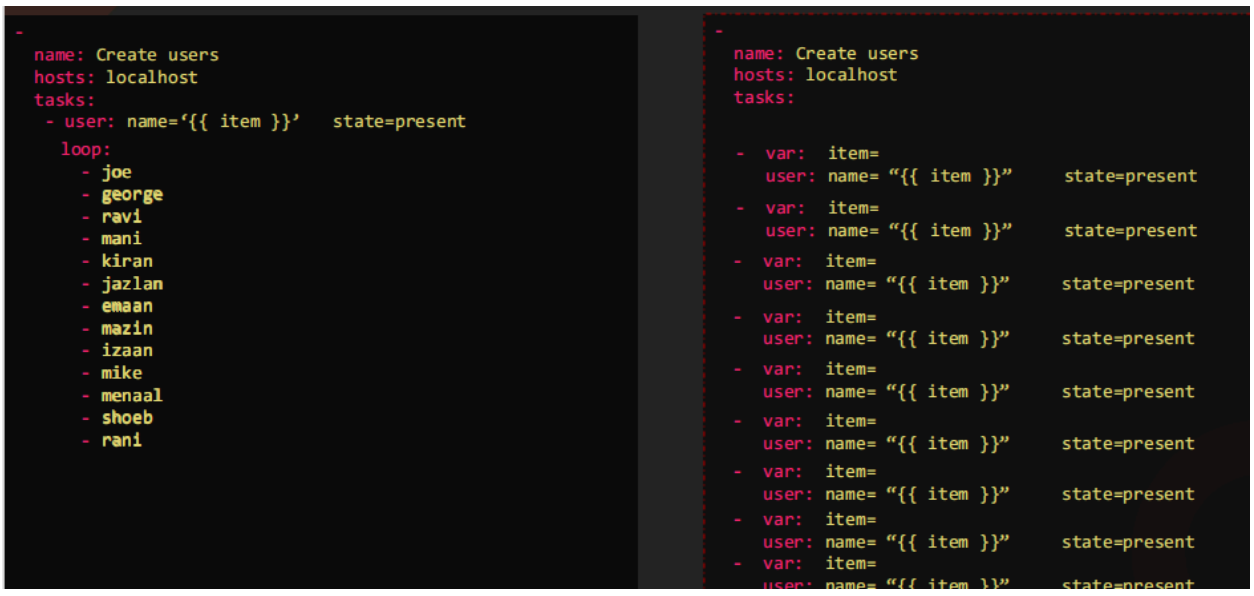
Scenario3:

If we want to perform a common set of tasks to perform on all the servers such as installing the necessary software packages.
If we want to start the service only in the prod environment

```
- name: Install required packages
  apt:
  name:
   - - package1
   - - package2
  state: present

- name: Create necessary directories and set permissions

…

- name: Start web application service
  service:
    name: myapp
    state: started
  when: environment == 'production'
```

Loops:

If we want to add multiple users, we can't write same code for multiple times to create multiple users , instead we use loop to create multiple users



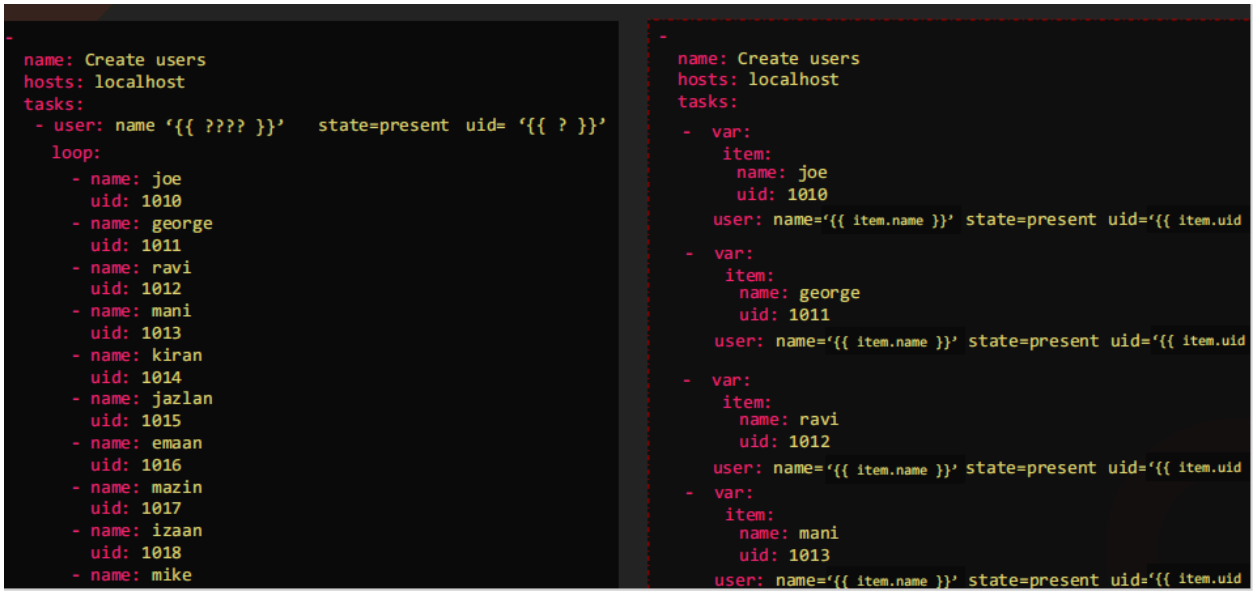
```
-
  name: Create users
  hosts: localhost
  tasks:
   - user: name='{{ item }}'   state=present
     loop:
       - joe
       - george
       - ravi
       - mani
       - kiran
       - jazlan
       - emaan
       - mazin
       - izaan
       - mike
       - menaal
       - shoeb
       - rani
```

```
-
  name: Create users
  hosts: localhost
  tasks:

  - var:  item=
    user: name= "{{ item }}"     state=present
  - var:  item=
    user: name= "{{ item }}"     state=present
  - var:  item=
    user: name= "{{ item }}"     state=present
  - var:  item=
    user: name= "{{ item }}"     state=present
  - var:  item=
    user: name= "{{ item }}"     state=present
  - var:  item=
    user: name= "{{ item }}"     state=present
  - var:  item=
    user: name= "{{ item }}"     state=present
  - var:  item=
    user: name= "{{ item }}"     state=present
  - var:  item=
    user: name= "{{ item }}"     state=present
```

If we want to specify user id also with user name then, each item in the loop has 2 values, username and user id , then we will pass an array of dictionaries into the loop instead of an array of strings.

Each dictionary have 2 key value pairs, name and uid

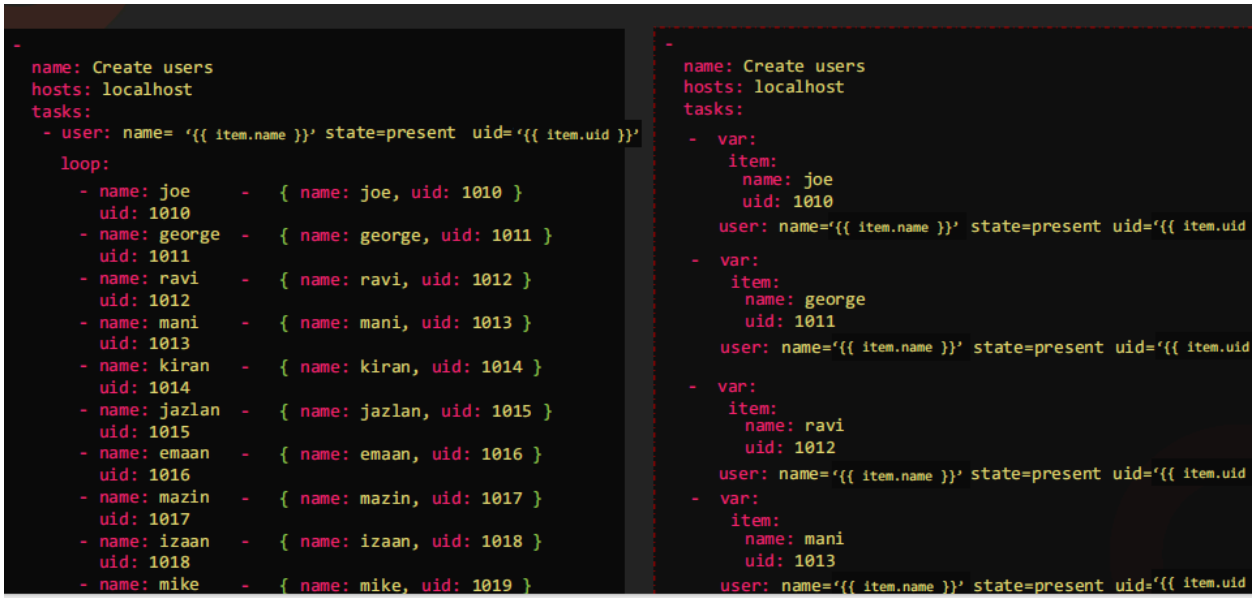
Left one is the yaml code to create multiple users, and the right one is the visualization how loop works



```
-
  name: Create users
  hosts: localhost
  tasks:
   - user: name '{{ ???? }}'   state=present  uid= '{{ ? }}'
     loop:
       - name: joe
         uid: 1010
       - name: george
         uid: 1011
       - name: ravi
         uid: 1012
       - name: mani
         uid: 1013
       - name: kiran
         uid: 1014
       - name: jazlan
         uid: 1015
       - name: emaan
         uid: 1016
       - name: mazin
         uid: 1017
       - name: izaan
         uid: 1018
       - name: mike
```

```
-
  name: Create users
  hosts: localhost
  tasks:
  -  var:
      item:
        name: joe
        uid: 1010
     user: name='{{ item.name }}' state=present uid='{{ item.uid
  -  var:
      item:
        name: george
        uid: 1011
     user: name='{{ item.name }}' state=present uid='{{ item.uid
  -  var:
      item:
        name: ravi
        uid: 1012
     user: name='{{ item.name }}' state=present uid='{{ item.uid
  -  var:
      item:
        name: mani
        uid: 1013
     user: name='{{ item.name }}' state=present uid='{{ item.uid
```

Array of dictionaries can also be represented in json format as follows:



```
-
  name: Create users
  hosts: localhost
  tasks:
   - user: name= '{{ item.name }}' state=present  uid='{{ item.uid }}'
     loop:
       - name: joe      -   { name: joe, uid: 1010 }
         uid: 1010
       - name: george   -   { name: george, uid: 1011 }
         uid: 1011
       - name: ravi     -   { name: ravi, uid: 1012 }
         uid: 1012
       - name: mani     -   { name: mani, uid: 1013 }
         uid: 1013
       - name: kiran    -   { name: kiran, uid: 1014 }
         uid: 1014
       - name: jazlan   -   { name: jazlan, uid: 1015 }
         uid: 1015
       - name: emaan    -   { name: emaan, uid: 1016 }
         uid: 1016
       - name: mazin    -   { name: mazin, uid: 1017 }
         uid: 1017
       - name: izaan    -   { name: izaan, uid: 1018 }
         uid: 1018
       - name: mike     -   { name: mike, uid: 1019 }
```

```
-
  name: Create users
  hosts: localhost
  tasks:
  -  var:
      item:
        name: joe
        uid: 1010
     user: name='{{ item.name }}' state=present uid='{{ item.uid
  -  var:
      item:
        name: george
        uid: 1011
     user: name='{{ item.name }}' state=present uid='{{ item.uid
  -  var:
      item:
        name: ravi
        uid: 1012
     user: name='{{ item.name }}' state=present uid='{{ item.uid
  -  var:
      item:
        name: mani
        uid: 1013
     user: name='{{ item.name }}' state=present uid='{{ item.uid
```

Another way create the loops is with_*:

with_*: with underscore directives

```
-
  name: Create users
  hosts: localhost
  tasks:
   - user: name='{{ item }}'   state=present
     loop:
       - joe
       - george
       - ravi
       - mani
```

```
-
  name: Create users
  hosts: localhost
  tasks:
   - user: name='{{ item }}'   state=present
     with_items:
       - joe
       - george
       - ravi
       - mani
```

For simple loops for which we will work only once then it is recommended to use the loop directive

We have so many with directives like with_items, with_file, with_url that connect to multiple url etc

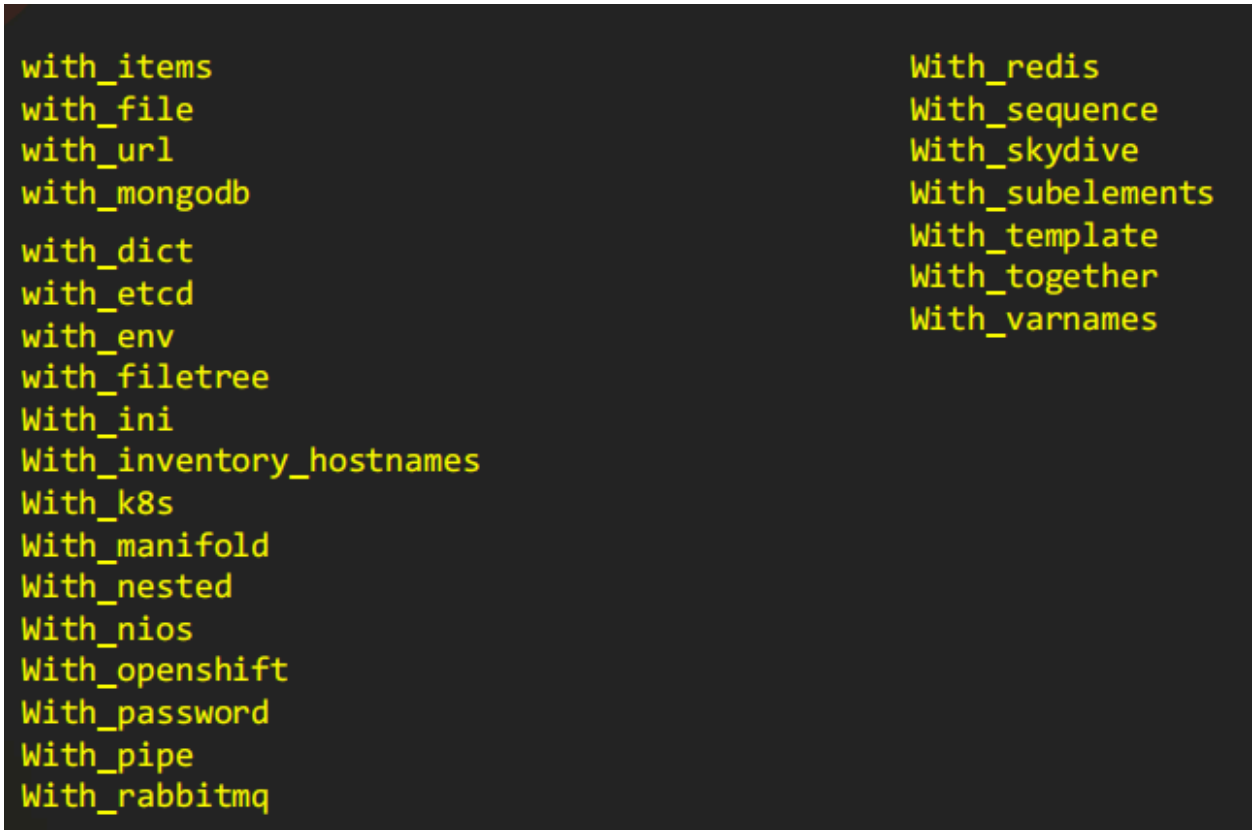
```
-
  name: Create users
  hosts: localhost
  tasks:
   - user: name='{{ item }}'   state=present
     with_items:
       - joe
       - george
       - ravi
       - mani
```

```
-
  name: View Config Files
  hosts: localhost
  tasks:
   - debug: var=item
     with_file:
       - "/etc/hosts"
       - "/etc/resolv.conf"
       - "/etc/ntp.conf"
```

```
-
  name: Get from multiple URLs
  hosts: localhost
  tasks:
   - debug: var=item
     with_url:
       - "https://site1.com/get-servers"
       - "https://site2.com/get-servers"
       - "https://site3.com/get-servers"
```

```
-
  name: Check multiple mongodbs
  hosts: localhost
  tasks:
   - debug: msg="DB={{ item.database }} PID={{ item.pid}}"
     with_mongodb:
       - database: dev
         connection_string: "mongodb://dev.mongo/"
       - database: prod
         connection_string: "mongodb://prod.mongo/"
```

Here are the list of with directives available:



Ansible Modules:

Ansible modules are classified in to various groups based on their functionality

- System
- Command
- Files
- Database
- Cloud
- windows and more

System module:

1. User
2. Group
3. Hostname
4. Iptables
5. Lvg
6. Lvol
7. Make
8. Mount
9. Ping
10. Timezone

11. Systemd
12. service

Command module: it is used to execute scripts or commands on host

1. Command
2. Expect
3. Raw
4. Script
5. Shell

Files module:

1. Acl: To set and retrieve the acl information
2. Archive
3. Copy
4. File
5. Find
6. Lineinfile
7. Replace
8. Stat
9. Template
10. unarchive

Database:

1. Mongodb
2. Mssql
3. Mysql
4. Postgresql
5. Proxysql
6. Vertica

Cloud:

1. Amazon
2. Atomic
3. Azure
4. Centrylink
5. Cloudscale
6. Cloudstack
7. Digital ocean
8. Docker
9. Google

10. Linode
11. Openstack
12. Rackspace
13. Smartos
14. Softlayer
15. VMware

Windows:

1. Win_copy
2. Win_commnad
3. Win_domain
4. Win_file
5. Win_iis_website
6. Win_msg
7. Win_msi
8. Win_package
9. Win_ping
10. win_ path
11. Win_robocopy
12. Win_regedit
13. Win_shell
14. Win_service
15. Win_user

Command module:

There are some parameters in the command module that are listed in documentation. They are:

1. chdir
2. creates
3. executable
4. free_from
5. removes
6. warn

If we use chdir inside the command parameter like

cat resolv.conf chdir

Here chdir=/etc changes the directory first to etc directory and later it executes the cat resolv.conf

Creates command parameter here first check the folder is present or not and if that folder is not found the **mkdir** will create that folder

command: mkdir /folder creates =/folder

Free form parameter:

Here **date** is a **free form** command that directly executes without having key value pairs as parameters and without depending on another parameter or command it will get executed

For example:

command: date
copy: src=/source_file dest=/destination_file

Here **date** is **a free form** command that directly executes, there are no key value pairs in it.

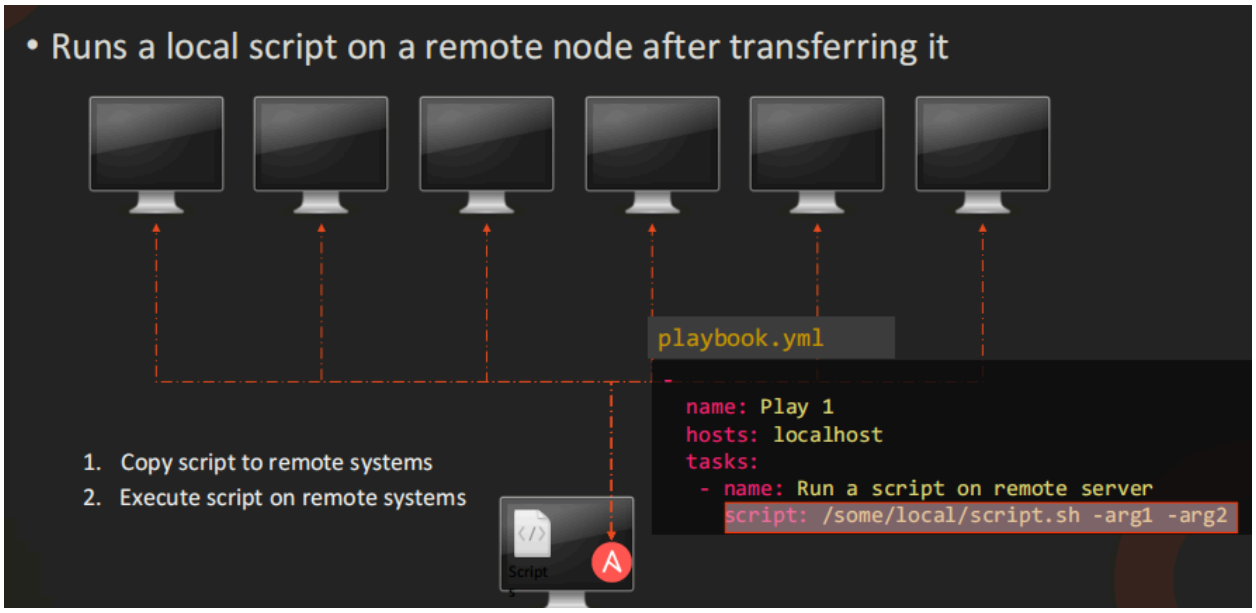
Here command **date , cat /etc/resolv.conf are free form** commands , because it is not like two commands inside it that will depend on one another.

Copy command is **not** a **free form** command because we need both source location and destination location both parameters and we should move source to destination

Some commands only take the parameterized input , like key value pairs , for example **copy** commands takes key value pairs, that commands are **not called free_form commands.**

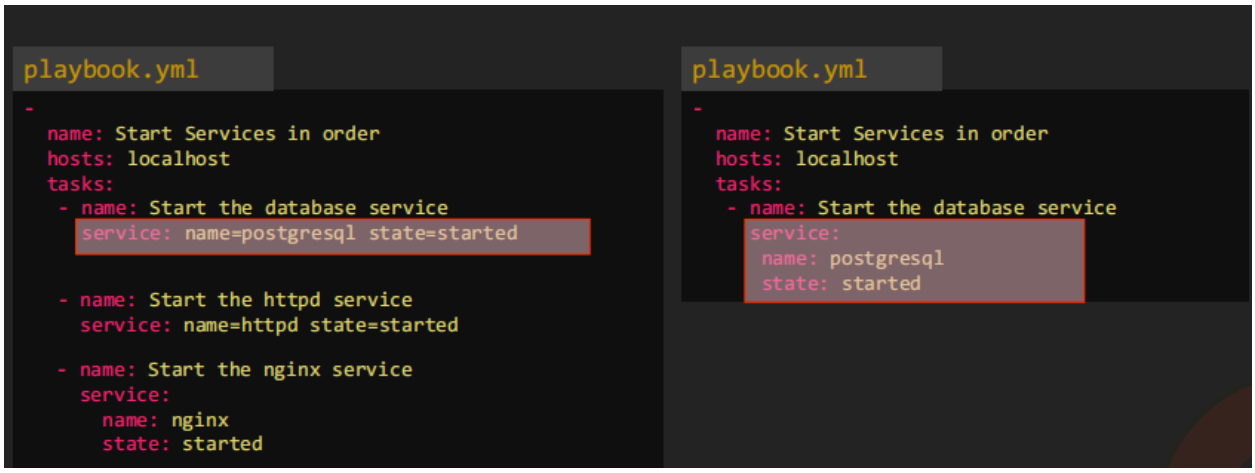
Script module:

1. Runs the local script on remote nodes after transferring it.
2. We don't need to copy the script to all remote nodes, ansible will take care of coping the script to all the remote nodes and execute that script on remote systems.



Service module:

1. Used to maintain the services on the system, like starting, stopping or restarting a service
2. **Service is also a free form parameter** it contains key value pair



We can m=write the service in 2 ways:

1. service: name=httpd state=started
2. service:
   name: httpd
   state: started

Idempotency:

Why started , why not start:

To ensure the httpd is started

If httpd is already started ⇒ do nothing
If httpd is not already started ⇒ start it this is called idempotency

Lineinfile module:

Search for a line in a file and replace it or add it if it doesn't exists

On the right side, we use `echo` to add the DNS server to the configuration file. However, using `echo` means that each time we run the script, the same DNS server can be added multiple times to the file.

In contrast, on the left side, we utilize the `lineinfile` module to add the DNS server to the configuration file. The `lineinfile` module first checks if the DNS server is already present in the file.

If the server is not listed, it will be added to the configuration file. If the server is already present, no action will be taken, and it will leave the file unchanged.

Lab:

1. Writing the playbook in such a way , the service is executed only if the host is node02

```
---
-  name: 'Execute a script on all web server nodes'
   hosts: all
   become: yes
   tasks:
     -  service: 'name=nginx state=started'
        when: 'ansible_host=="node02"'
```

Inventory file:

```
node01 ansible_host=node01 ansible_ssh_pass=caleston123
node02 ansible_host=node02 ansible_ssh_pass=caleston123
[web_nodes]
node01
node02
```

Running the playbook:

```
ansible-playbook -i inventory nginx.yaml
```

Output:

```
PLAY [Execute a script on all web server nodes] *********************************

TASK [Gathering Facts] ****************************************************
ok: [node01]
ok: [node02]

TASK [service] ************************************************************
skipping: [node01]
ok: [node02]

PLAY RECAP ****************************************************************
node01                     : ok=1    changed=0    unreachable=0    failed=0    skipped=1
rescued=0    ignored=0
node02                     : ok=2    changed=0    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```

2. Playbook for displaying the child or adult, it displays child if age is less than 18, and it displays adult if the age is greater than 18.

```
---
- name: 'Am I an Adult or a Child?'
  hosts: localhost
  vars:
    age: 25
  tasks:
    - name: I am a Child
      command: 'echo "I am a Child"'
      when: 'age<18'
    - name: I am an Adult
      command: 'echo "I am an Adult"'
      when: 'age>=18'
```

3.Playbook /home/bob/playbooks/nameserver.yaml attempts to add an entry in /etc/resolv.conf file to add a new nameserver.

The first task in the playbook is using the shell module to display the existing contents of /etc/resolv.conf file and the second one is adding a new line containing the name server details into the file. However, when this playbook is run multiple times, it keeps adding new entries of same line into the resolv.conf file. To resolve this issue, update the playbook as per details mentioned below.

- Add a register directive to store the output of the first task to a variable called command_output
- Then add a conditional to the second task to check if the output already contains the name server (10.0.250.10). Use command_output.stdout.find(<IP>) == -1

Note:
a. A better way to do this would be to use the lineinfile module. This is just for practice.

b.shell and command modules are similar in a way that they are used to execute a

command on the system. However, shell executes the command inside a shell giving us access to environment variables and redirection using >>.

Ans:

```
---
- name: 'Add name server entry if not already entered'
  hosts: localhost
  become: yes
  tasks:
    - shell: 'cat /etc/resolv.conf'
      register: command_output

    - shell: 'echo "nameserver 10.0.250.10" >> /etc/resolv.conf'
      when: 'command_output.stdout.find("10.0.250.10")==-1'
```

4.The playbook /home/bob/playbooks/fruits.yml currently runs an echo command to print a fruit name. Apply a loop directive (with_items) to the task to print all fruits defined under the fruits variable.

```
---
-  name: 'Print list of fruits'
   hosts: localhost
   vars:
     fruits:
       - Apple
       - Banana
       - Grapes
       - Orange
   tasks:
     - command: 'echo "{{ item }}"'
       with_items: '{{fruits}}'
```

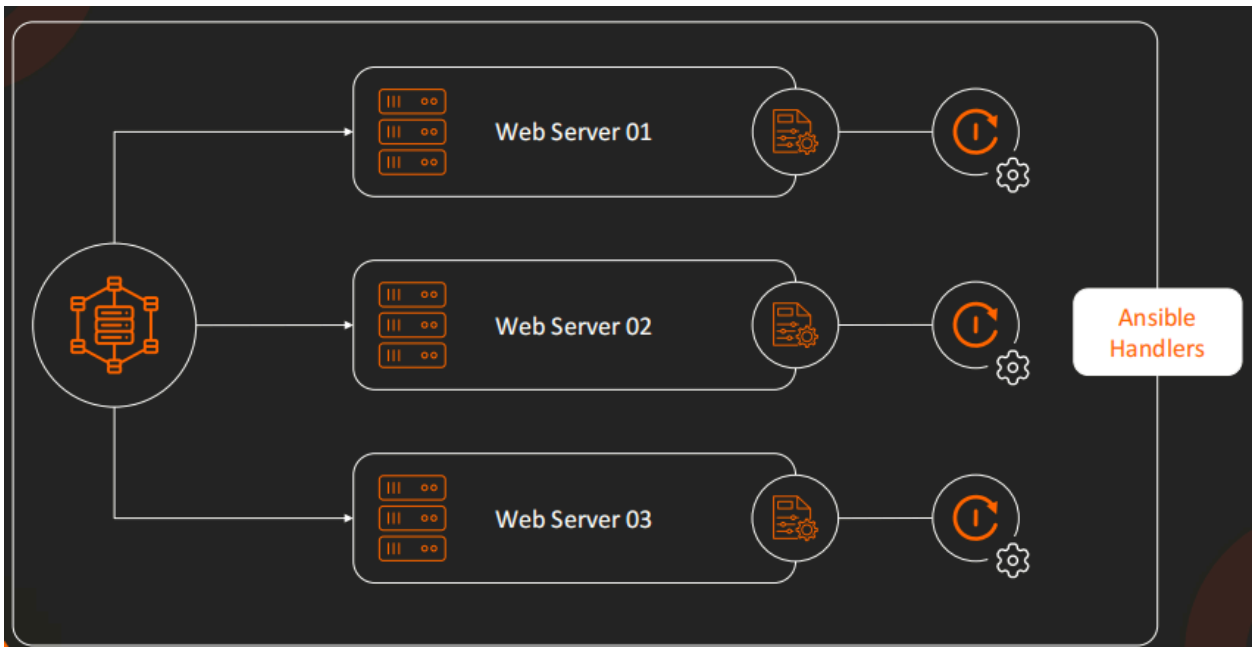
5. Installing the packages one after the other.

```
---
- name: 'Install required packages'
  hosts: localhost
```

```
 become: yes
 vars:
  packages:
    - httpd
    - make
    - vim
 tasks:
  - yum:
     name: '{{item}}'
     state: present
    with_items: '{{packages}}'
```

Handlers:

1. Whenever we change/update the configuration of the server, we need to restart the server.
2. If we are changing the configuration in multiple servers we need to restart all the servers, we might make human errors if there are multiple servers to restart.



3. Here the handler will take care of these servers to restart, the handler will restart the server immediately after any change made in configuration of server.
4. Tasks are triggered by notifications
5. Handlers are defined in playbook and get executed when notified by a task

6. Manage actions based on system configuration/state

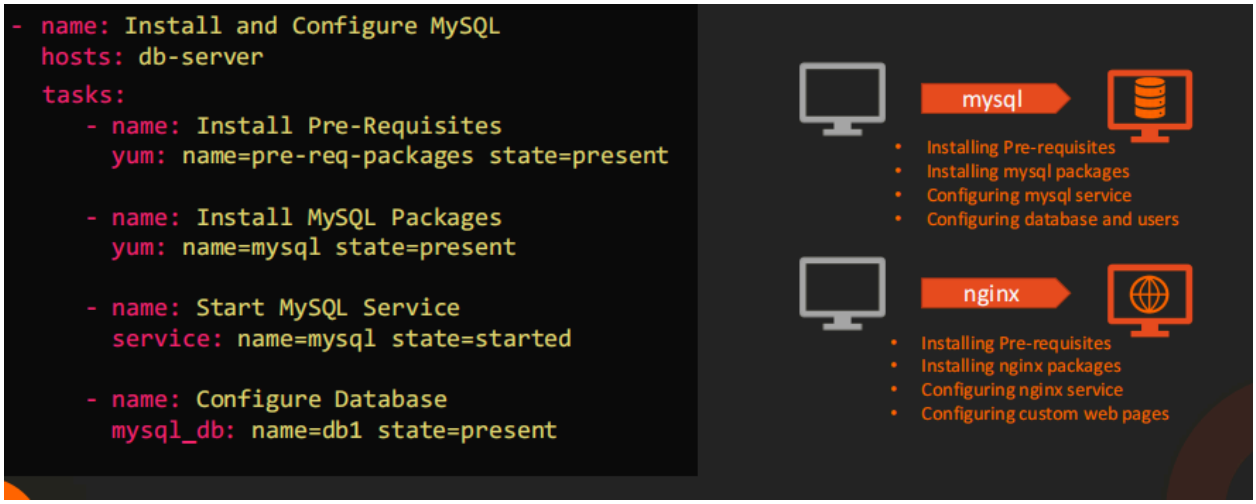
Playbook:

```
---
- name: deploy application
  hosts: application_server
  tasks:
     - name: copy application code
        copy:
          src: location/of/source_code
          dest:  location/of/dest
        notify: Restart the Application service
   handlers:
        - name: Restart the Application service
          service:
                name: application_service
                state: restarted
```

⇒ Here value of notify and name inside the handler should be same

Ansible roles:

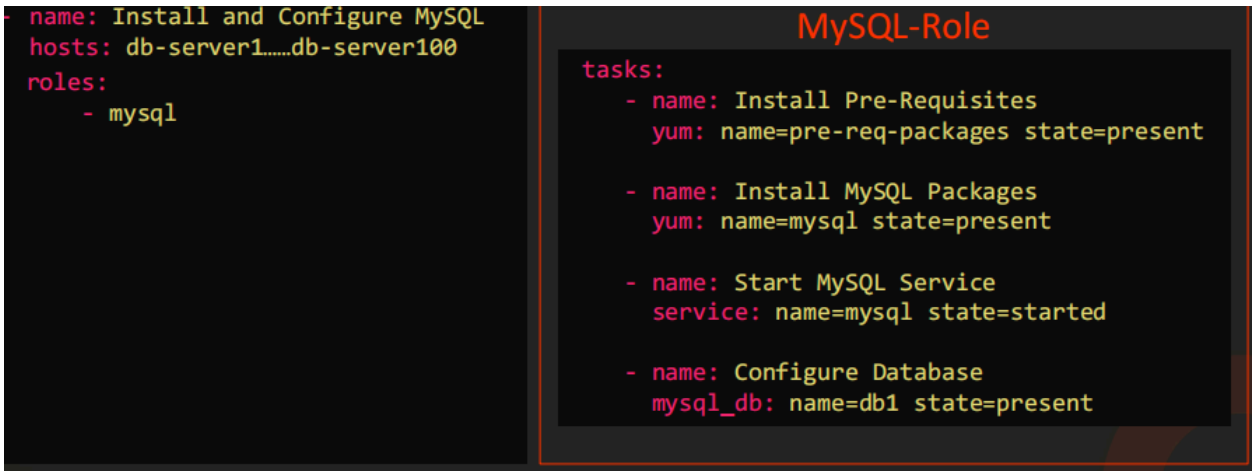
1. Just like assigning roles to the people like doctor, engineer etc, here we assign roles to blank servers to make them as web server, database server, network server etc.
2. Assigning roles means doing everything to make them like database server such as like installing prerequisites required for mysql, installing mysql packages and configuring the mysql services etc
3. We can do by writing playbook like this:



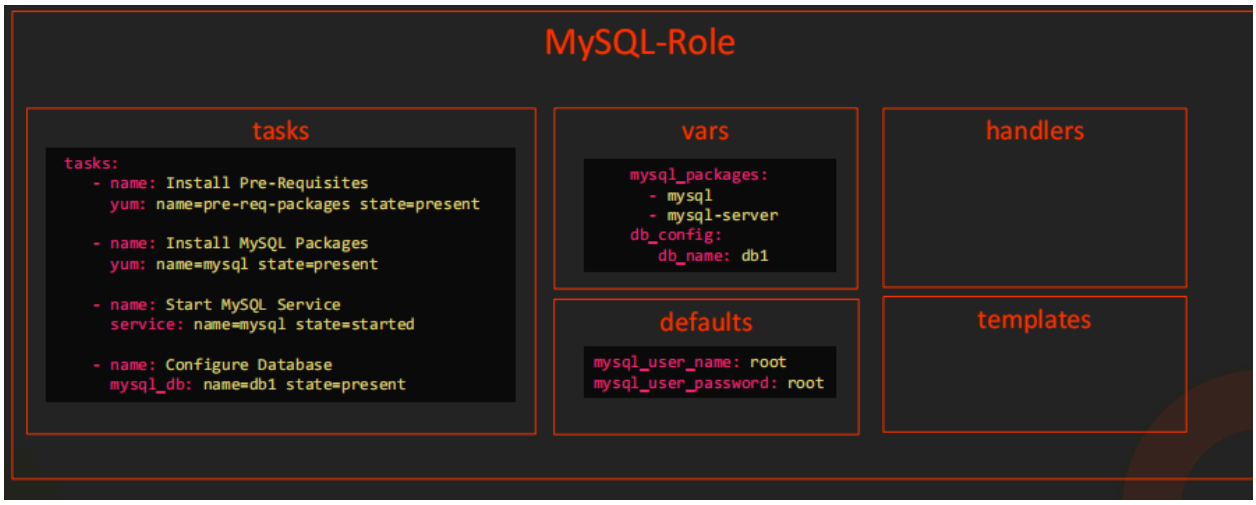
Here in the above playbook we are installing and configuring everything related to db in the db server.

If we want to perform the same tasks, or want to share with others , instead of rewriting the same code again and again we use **Roles.**

Whatever we wrote inside the task we can package it into the role and we can reuse it later as below:



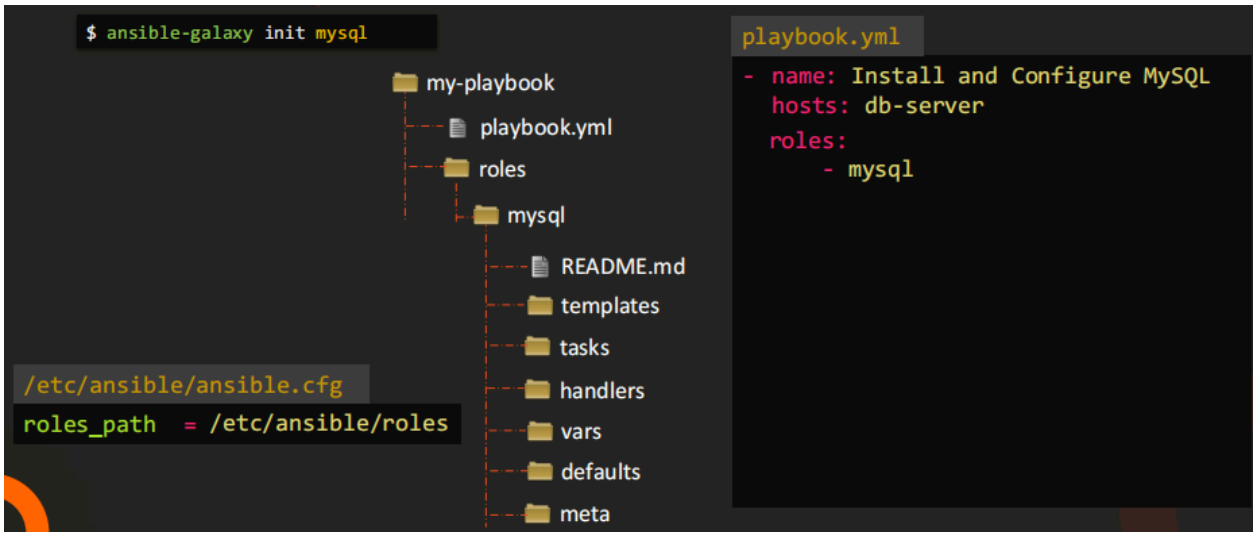
4. Roles also help to organize your code, it organizes all the tasks into task directory like all vars goes into the vars directory, all default values goes into the default directory, all handlers goes into the handlers directory etc.



5. Roles also help us to share the code in ansible community, ansible galaxy is one community where we can found thousands of roles, for installing tools, configuring, etc
6. Ansible galaxy is a tool that can create a skeleton for us for directory structure, by using the following command the galaxy creates directory structure:

   ansible-galaxy init mysql

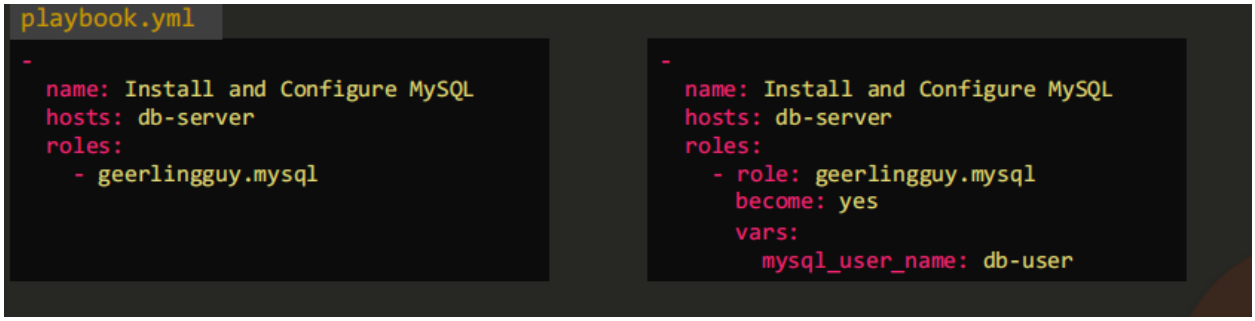
7. If we write the tasks inside the role and if we are writing a playbook and needs to reuse that role in that playbook then how can we access that role in our new playbook, there are different ways to do that:
   a. 1st way: The location in which our new playbook is there, in that directory create a new directory name as role and move everything into that role directory
   b. Another way: we can move that role into ansible roles location /etc/ansible/roles , it is the default location where ansible search for the roles



8. Once you created the roles directory and used it in the playbook, we can share it in the community by uploading it to ansible galaxy by using github repository.
9. If we want to search an existing role in the ansible galaxy , then it is done by following command:

To search the role: ansible-galaxy search mysql
To use that role we should install it first : ansible-galaxy install geerlingguy.mysql ⇒ the highlighted one is name of the role that we need to use
That role will get installed in /etc/ansible/roles

To use that roles in our playbook we can mention in 2 ways as follows:



Left side we mention roles as a list
Right side we mention role as a dictionary

To list the roles:
Ansible-galaxy list

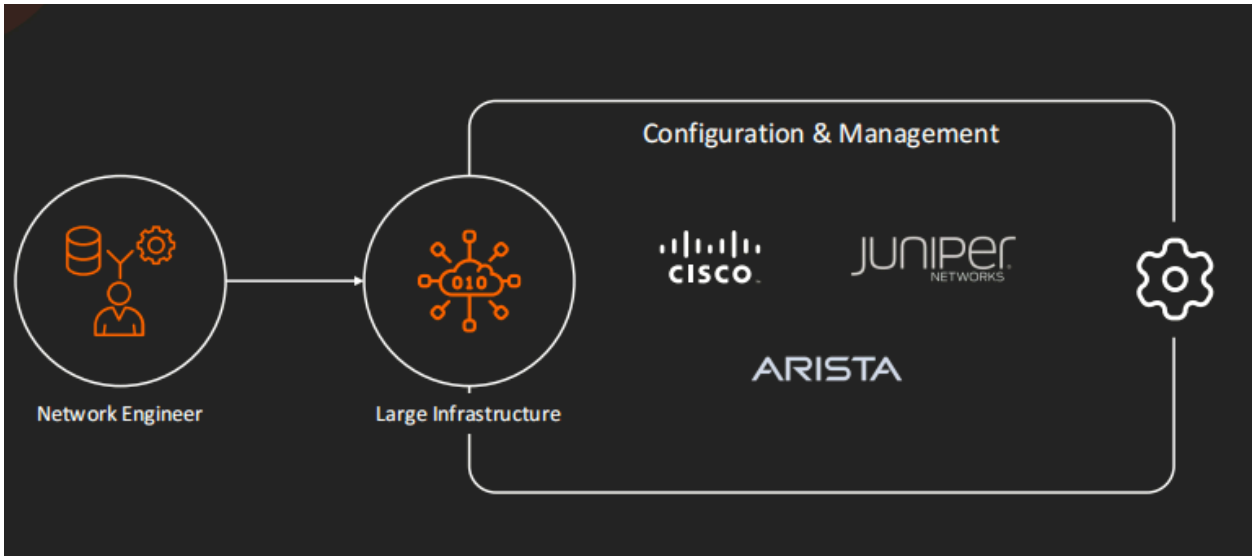
To view the roles where the roles would be installed:
Ansible-config dump | grep ROLE

To install the role in the current directory of playbook under roles directory:
Ansible-galaxy install geerlingguy.mysql -p ./roles

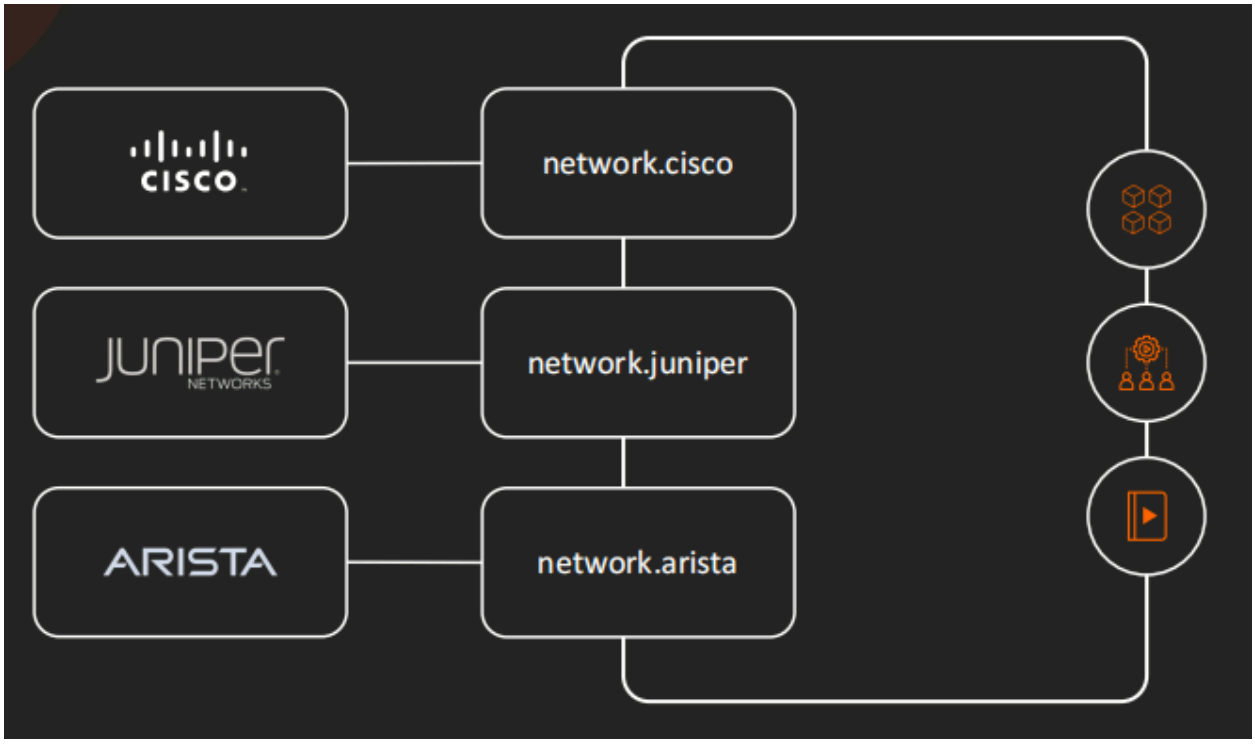
here playbook.yaml is our playbook where we need to use roles. By using above command we can directly install geerlingguy.mysql role in the current directory my-playbook under roles.

Collections:

- If network manager wants to automate the network devices, they need additional vendors for automation



- Here we use collections such as network.cisco, network.juniper, network.arista , there collections offers vendor specific modules, roles for managing network devices



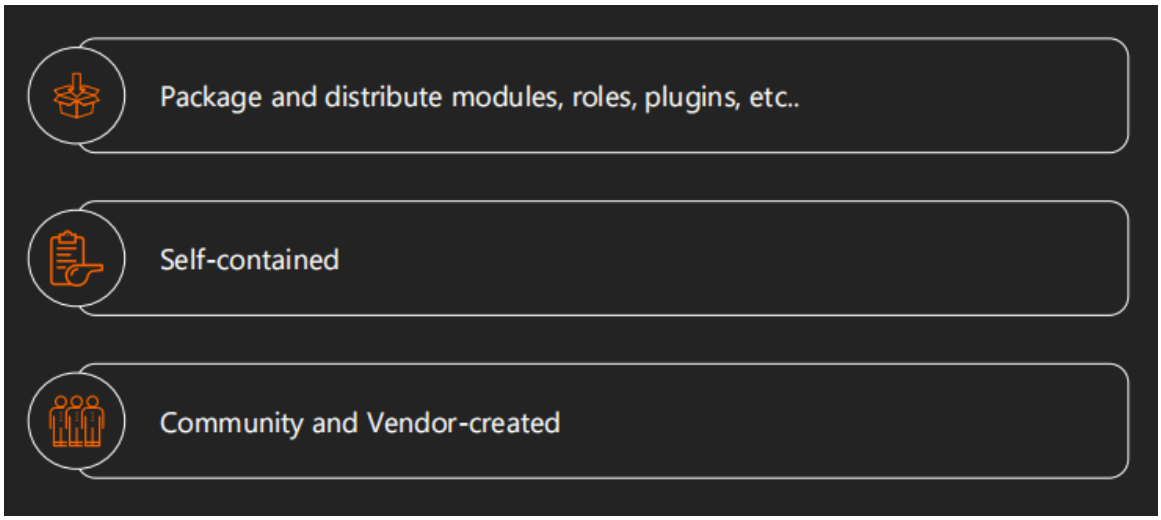
- By installing these collections we can gain access to specialized functionalities required to automate network infrastructure

For installing the collection :
ansible-galaxy collections install collection_name

ansible-galaxy collections install amazon_aws
ansible-galaxy collections install community.general
ansible-galaxy collections install requirements.yml

requirements.yml : it is a yaml file that contains the collections which we need to install
Eg:

```
---
collections:
  - name: community.general
    version: '1.0.0'
  - name: amazon.aws
    version: '1.2.1'
```

- **ansible-galaxy collections install collection_name**
- By installing the network.cisco collections, we can utilize modules, roles, provided by network.cisco collection , and can get specialized functionalities it offers
- 

Benefits of collections:

1. Expanded functionality: first we can install the collection , then we can use its module, functionality etc in the playbook

   Here we are installing amazon.aws collection in that we can also use its extended with aws_s3_bucket

```
$ ansible-galaxy collection install amazon.aws
```

```
---
- hosts: localhost
  collections:
    - amazon.aws

  tasks:
    - name: Create an S3 bucket
      aws_s3_bucket:
        name: my-bucket
        region: us-west-1
```

2. Modularity and reusability:

- We can create a collection that will encapsulate specific roles, modules, and plugins
- This is the directory structure for custom collection: **my_namespace.my_collection**

```
---
- hosts: localhost
  collections:
    - my_namespace.my_collection

  roles:
    - my_custom_role

  tasks:
    - name: Use custom module
      my_custom_module:
        param: value
```

3. Specified distribution and management:

```
requirements.yml
---
collections:
  - name: amazon.aws
    version: "1.5.0"
  - name: community.mysql
    src: https://github.com/ansible-collections/community.mysql
    version: "1.2.1"


$ ansible-galaxy collection install -r requirements.yml
```

We can mention all the requirements in a yml file and execute the following command to install all the requirements:
ansible-galaxy collection install -r requirements.yml

LABS:

Q:
You've been informed about a new Ansible Collection named networking_tools under the namespace company_xyz. This collection contains specialized modules for network automation. You need to write a playbook that installs this collection and then uses a custom module named configure_vlan from it to set up VLAN 10 with the name Admin_VLAN.

Complete the playbook located at /home/bob/playbooks/playbook.yml to correctly install the networking_tools collection and utilize the configure_vlan module. You just need to fill the placeholders in this playbook.

Please note that, you need not to run this playbook for testing, just make the required changes to fill the placeholders correctly.

Ans:

```
---
- hosts: localhost
  tasks:
    - name: Install the networking_tools collection
      ansible.builtin.ansible_galaxy_collection:
        name: company_xyz.networking_tools
        source: https://galaxy.ansible.com

- hosts: switches
  collections:
    -company_xyz.networking_tools
  tasks:
    - name: Configure VLAN 10
      configure_vlan:
        vlan_id: 10
        vlan_name: Admin_VLAN
```

Handler:

Example:
You are given access to a lab environment with a server (i.e `student-node`). On this server, you have a playbook named `/home/bob/ansible_lab/handler_test.yml`. This playbook contains multiple tasks that notify the same handler. Your task is to run the playbook and answer the question below.

How many times will the handler be executed during the playbook run?

a) Once for each notifying task.

b) Only once, at the end of the playbook run.

c) Once for each changed task.

d) The handler will not be executed.

```
---
- name: Test Handler Execution
  hosts: localhost
  tasks:
```

```
    - name: Copy file1.conf
      copy:
        src: files/file1.conf
        dest: /tmp/file1.conf
      notify: Sample Handler

    - name: Copy file2.conf
      copy:
        src: files/file2.conf
        dest: /tmp/file2.conf
      notify: Sample Handler

  handlers:
    - name: Sample Handler
      debug:
        msg: "Handler has been triggered!"
```

Output:

```
PLAY [Test Handler Execution] ***************************************************

TASK [Gathering Facts] **********************************************************
ok: [localhost]

TASK [Copy file1.conf] **********************************************************
changed: [localhost]

TASK [Copy file2.conf] **********************************************************
changed: [localhost]

RUNNING HANDLER [Sample Handler] ************************************************
ok: [localhost] => {
    "msg": "Handler has been triggered!"
}

PLAY RECAP **********************************************************************
localhost                  : ok=4    changed=2    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```

Ans:
b) Only once, at the end of the playbook run.

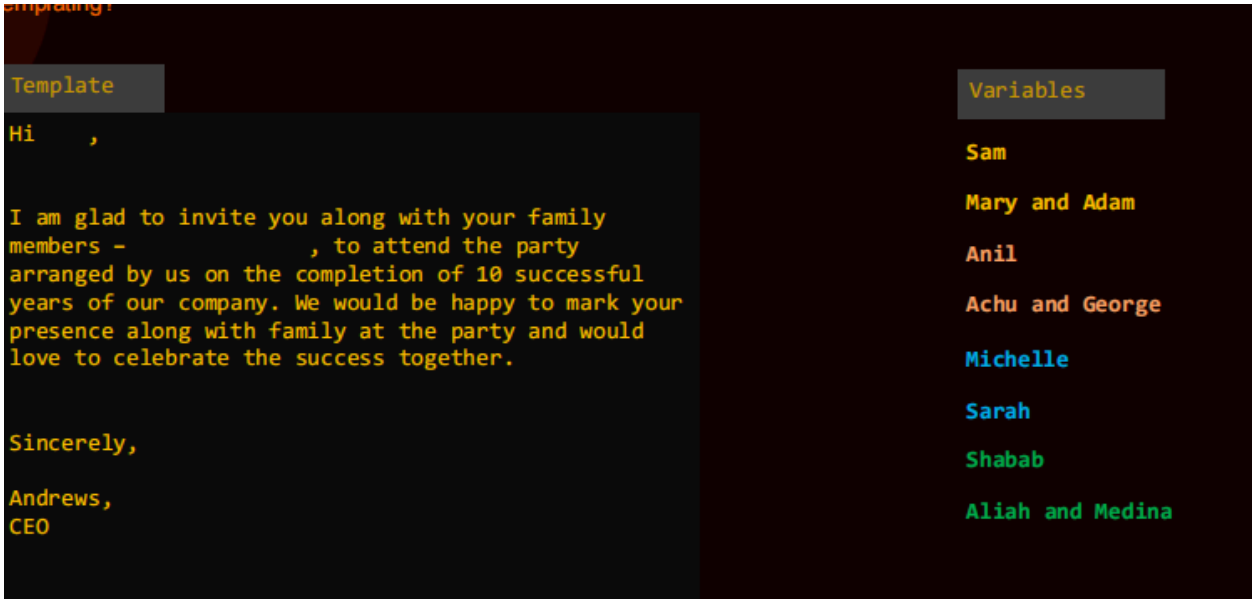
Q:

You have a handler named `Restart Network Service`. Which of the following task implementations correctly notifies this `handler` when the network configuration file is modified?

Ans:

```
- name: Update network configuration
  copy:
    src: /path/to/network.conf
    dest: /etc/network/network.conf
  notify: Restart Network Service
```

Templating:



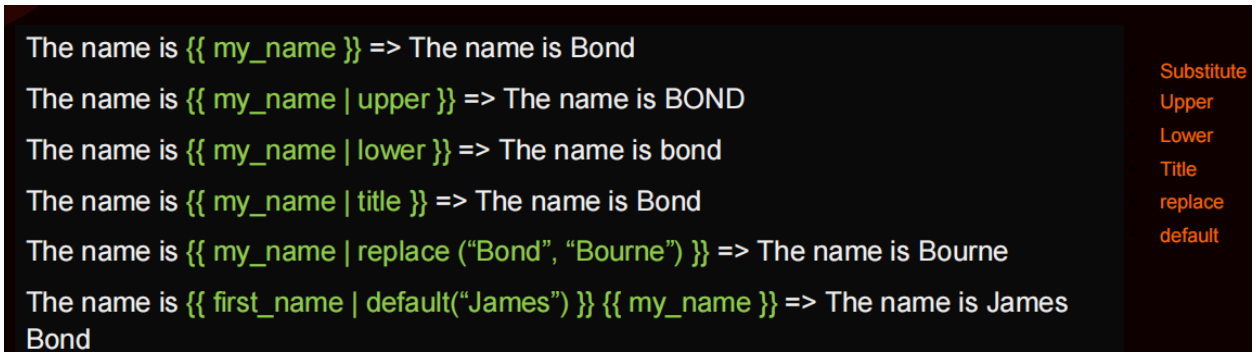
Here we are writing a template and using variables inside that template

Jinja :

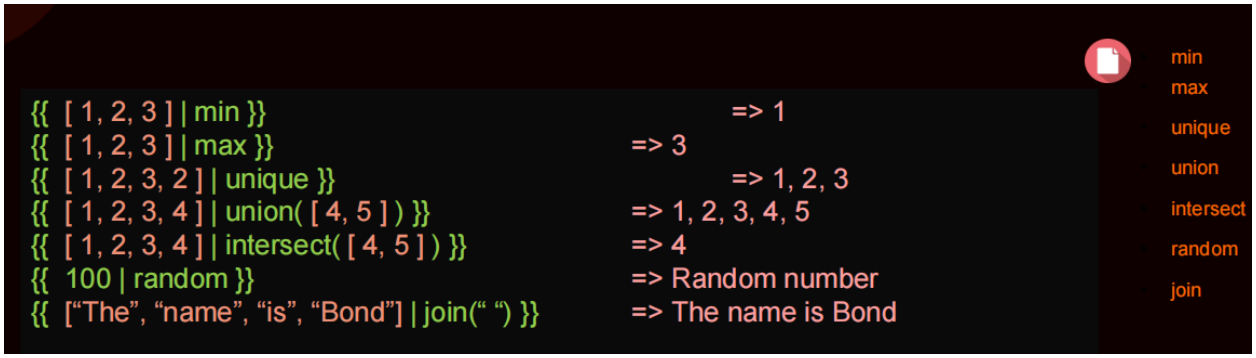
Jinja is friendly templating language for python, modelled after django templates.

```
<title>{% block title %}{% endblock %}</title>
<ul>
{% for user in users %}
  <li><a href="{{ user.url }}">{{ user.username }}</a></li>
{% endfor %}
</ul>
```

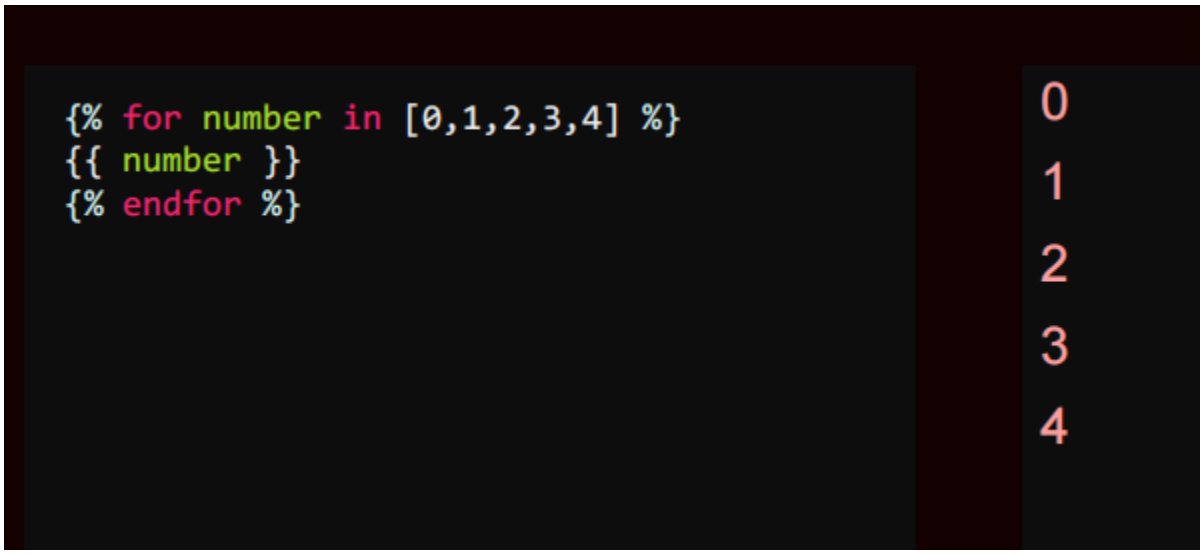
String manipulation:



Lists and set:



Loops:



Here {%.....%} ⇒ indicates that jinja2 code is block not a one liner.

<u>Condition:</u>

```
{% for number in [0,1,2,3,4] %}

{% if number == 2 %}
    {{ number }}
{% endif %}

{% endfor %}
```

2