debconf = debian database configuration = its a debian database to hold software installation configuration.
export DEBIAN_FRONTEND=non-interactive

install the software manually and retreive the keys using debconf-get-selections
sudo debconf-get-selections | grep mysql-server

echo "KEY VALUE" | sudo debconf-set-selections
---
mysql_secure_installation (shell script)
by using expect package

shell1.sh
#!/bin/bash
echo "Enter name"
read NAME

expectshell1.sh
#!/usr/bin/expect -f
set timeout -1
spawn ./shell1.sh
expect "Enter name"
send -- "Ramu\r"
expect eof
---
mysql server remote access configuration

change bind address in /etc/mysql/mysql.conf.d/mysqld.cnf
bind-address = 127.0.0.1
sudo sed -i 's/^bind-address=127.0.0.1/bind-address=0.0.0.0/g' /etc/mysql
/mysql.conf.d/mysqld.cnf

configure remote access to the mysql server by seeding the user into database
create user 'rail_user'@'%' identified by 'password' with grant option;
grant all privileges on *.* to 'rail_user'@'%';

restart mysql server
systemctl restart mysql
---

install openjdk 11
sudo apt install openjdk-11-jdk

download and extract tomcat server

https://mirrors.estointernet.in/apache/tomcat/tomcat-9/v9.0.41/bin/apache-tomcat-9.0.41.tar.gz
/u01/data
|-apache-tomcat-9

How to add an user into the ansible managed node using ansible.

```
---
- name: add user playbook
hosts: all
tasks:
- name: add user on to ansible managed node
user:
name: ansible
state: present
- name: copy ssh keys on to managed node user
authorized_key:
user: ansible
key: ~/.ssh/id_rsa.pub
state: present
- name: configure ssh access
lineinfile:
path: /etc/sudoers
regexp: '^ansible ALL\='
line: 'ansible ALL=(ALL) NOPASSWD:ALL
validate: 'visudo -cf %'
...
```

Modules we explored in ansbile
debug
ping
command
shell
package
apt
file
copy
user
lineinfile
authorized_key

Working with variables in ansible.
Variables are place holders in which we can store the values, so that we can use values from variables rather than hardcoding the program.

In ansible while writing playbooks, we pass values to the arguments of the ansible modules, for eg.. to an apt module we are passing name of the package we want to install. Similary to a copy module we are passing src and destination values as arguments in copying a source file to destination.
Instead of hardcoding the values we pass to these arguments, if we can define variables and can pass values to these variables we defined, then playbook can be reused for performing multiple times the same activity with different set of inputs.

If I want to install 10 software packages on a managed node, dont write 10 playbooks, rather write a playbook variable as a package to install and run the playbook for 10 times passing different variable values as package names.

How to work with variables in ansible?
There are multiple levels of variables and multiple ways defining and passing values to the

variables are available.

1. Ansible Variables
2. Inventory level
2.1 host level variables
2.2 group level variables
3. playbook variables
4. var_files in playbook
5. vars from command-line
6. register variables
7. special variables (facts)

---

1. Ansible variables
There are pre-defined variables of ansible based on which ansible control node will connects to the managed nodes. We can call them as connection methods in ansible.

By default ansible control node connects to the managed nodes as follows.
1. it uses default ssh port as 22 to connect
2. it uses the same user of the control node to connect to the managed node
3. connection method being used by default is ssh
4. the default key file location it looks for on control node is ~/.ssh/id_rsa.pub|id_rsa
Now if we want to change any of these configurations while connecting to the managed node we can specify these variables with values in ansible inventory file.

There are many other ansible configuration settings are there which we can find them in /etc/ansible/ansible.cfg file.

we want to run the playbook on the local machine where my control node exists, by default ansible connects to the host we specified using ssh protocol, we should say dont connect just run the playbook locally using ansible_connection=local

In the below we are confuring ansible values at host level for each host. so control node uses the values of these variables while running the playbook on each host.

hosts
127.0.0.1 ansible_connection=local

hosts
192.168.164.4 ansible_user=ansible ansible_ssh_port=24 ansible_connection=ssh ansible_private_key_file=~/.ssh/ansible_rsa become=yes

what are variables ansible?
variables are placeholders in which we can store the data using which we can perform operations.

What is the purpose of variables in ansible?
If we want to perform an automation by using different sets of input values, rather than hardcoding those values as part of the playbook we can define them as variables and pass values at the time of running the playbooks, so that we can reuse the playbooks in ansible.

There are 2 types of variables are there in ansible.
1. Ansible variables
2. User-Defined variables

What are Ansible variables?
These are pre-defined variables provided by the Ansible, which are interpreted by the ansible system upon configuring these with values. So, we can use them for configuring the Ansible System. Few of such Ansible variables are.
- ansible_host
- ansible_port
- ansible_user
- ansible_connection
- ansible_private_ssh_key_file
These variables can be configured at different levels in ansible system.
1. in /etc/ansible/ansible.cfg, this is an global ansible configuration file in which ansible system configuration is defined as variables with default values. If we modify a Ansible System variable value, it reflects for the entire System.
For eg.. if we modify ansible_port=23 in ansible.cfg file, then while connecting to any of fleet nodes, ansible control node uses 23 as ssh port.

2. Instead of modifying the Ansible System Variables in global scope we can apply the changes at inventory level as well by defining them as host/group level variables.
1. Host level variables = These variables are binded to the host we defined with and are being passed to the playbook while the playbook is executing on the host.
The Host level Ansible System variables we configured will override the global variables of ansible.cfg.

hosts [inventoryfile]
192.168.164.4 ansible_connection=ssh ansible_port=23
192.168.164.5 ansible_user=ansible ansible_port=22

2. We can define Ansible System Variables at group level as well in inventoryfile, so that for all the group of machines the playbook is executing on these variable values will be passed as an input.

hosts [inventoryfile]
[dbservers]
dbserver1 192.168.164.4
dbserver2 192.168.164.5

[dbservers:vars]
ansible_connection=ssh
ansible_user=ansible

User-Defined variables

We can define user-defined variables in ansible that can be passed as an input during the playbook execution. We can define user defined variables at different scopes in ansible.
1. inventoryfile
2. playbook

When to define variables in InventoryFile?
We define variables in inventoryfile which are of static in nature that holds system configuration information that is used as an input during the playbook execution.
for e.g..
A host requires periodical backup, what is the interval in hours the machine disk should be backed-up.
hosts [inventoryfile]
dbserver1 192.168.164.5 backup_interval=4
backup_disk_location=\net\slc123pwq\instances\2039

Again there are 2 places in inventoryfile we can define user defined Variables
1. host levels
2. group levels

If a variable value is common to all the group of machines then we define variable at group level otherwise we need to bind variable at host level only.

syntax of host-level variable declaration
---
hosts [inventoryfile]
dbserver1 192.168.164.4 backup_interval=10 data_center=us-east-1

syntax of group-level Variables
---
hosts [inventoryfile]
[dbservers]
dbserver1 192.168.164.1
dbserver2 192.168.164.2

[dbservers:vars]
data_center=us_east_1

We can access these variables during the playbook executing using the below syntax.
"{{variable_name}}"

What are variables in ansible?
Variables are placeholders in which we can store the data and can be uses as inputs in performing the operations.
There are 2 types of variables are there
1. Ansible variables
2. User-Defined Variables

What are Ansible Variables?
These are the variables having pre-defined meaning, using which we can configure ansible system/control node. We can configure these variables at 2 places.
1. in global configuration file (/etc/ansible/ansible.cfg)
2. inventory file
- ansible_connection
- ansible_host
- ansible_port
- ansible_user
- ansible_ssh_private_key_file

2. What are User-Defined Variables?
The ansible developers can also define variables so that those can be passed as an input to the playbooks while executing.

We can declare variables in ansible at different places, based on the place at which we declare them we treat them as scopes of variables.
1. We can define variables at inventory file level
2. Playbook level

#1 How many places we can define the variables in an Inventory File.
There are 2 places or levels at which we can define variables in Inventory File.
1. Host level Variables = These variables are written/defined for each host in the inventory file and will be passed as an input while executing the playbook on that host.

hosts [inventory file]
server1 ansible_host=192.168.164.4 ansible_port=23 mysql_port=3307

2. Group level variables = If there are some common variables that carries the same values across a group of machines rather than declaring these variables with same values for each of the host, we can define variables at group level as well. So that we can avoid duplication of the variables with values.

hosts [inventory file]
[javaservers]
javaserver1 ansible_host=192.168.164.4
javaserver2 ansible_host=192.168.164.5

[javaservers:vars]
tomcat.port=8081
tomcat.admin.user=manager

What type of information we generally bind to an Host or Group in inventory file?
Generally we represent the variables at host/group level of an inventory file to store static configuration information about a machine or group of machines, which we can use it as an input for automation.
For e.g.. the mysql server is configured on which port for a machine.

How to pass dynamic inputs to a playbook while executing?
Using Inventory File we attach machine configuration/static information pertaining the machine that would be common for across all the playbook executions as variables. But there are some inputs that has to be passed with different values for each playbook execution. Such inputs that we pass differently for each execution of a playbook are called "Dynamic Inputs".
There are 3 ways are there in passing input data while running a playbook.
1. extra-vars
2. variables declared local to the playbook
3. vars_files

1. extra-vars
While running the playbook we can pass extra variables as an input through command-line, so that these variables we passed can be used as part of playbook execution.

ansible-playbook -i hosts --extra-vars var1=val1 --extra-vars var2=val2 playbook.yml
Here we are passing 2 variables with names var1 and var2 and these can be accessed inside the playbook using {{variableName}} notation.

For e.g.
ansible-playbook -i hosts --extra-vars sourcefile=/home/sriman/schema.sql --extra-vars destfile=/home/ansible/db-schema.sql cpplaybook.yml
---
- name: copy playbook
hosts: all
tasks:
- name: copy tasks
copy:
src: "{{sourcefile}}"
dest: "{{destfile}}"
...

---
2. define variables local to the playbook
instead of passing variables through command-line execution, we can define variables local to playbook itself under vars section.

- name: playbook
hosts: all
vars:
sourcefile:/home/sriman/data.sql
destfile: /home/ansible/db-schema.sql
tasks:
- name: copy
copy:
src: "{{sourcefile}}"
dest: "{{destfile}}"
...

In this case we defined the variables at the playbook, because each time when we run the playbook we want it to execute with same set of values for all of the hosts and it is specific to the playbook only.
---
3. using vars_files

Each time when we run a playbook we want some dynamic values to be passed as an input, we can use --extra-vars for this, but everytime passing values through command-line seems to be an tedious job.
Instead define all the variables with their corresponding values in a external yaml file. Define variables with values as dictionaries / associative arrays or lists. Now we can pass them as variables into playbook using vars_files.

servers.yaml
tomcat:
server_port: 8080
jdk_version: 1.8

apache2:
port: 8080
directory: /vars/www/app1

playbook.yml

- name: playbook
hosts: all
vars_files:
- server.yml
tasks:
- name: debug vars
debug:
msg: "server port {{tomcat.server_port}}"
...

How to pass the data as an input to the playbooks during their execution?
(or)
How to declare variables at playbook level?
There are 3 ways we can define variables or pass the data as an input for a playbook during its execution.
1. extra-vars through command-line
2. define variables inside the playbook
3. vars_file

Declare dynamic variables as part of the playbook tasks?
We can create variables as part of the task execution dynamically within a playbook using set_fact module. Here we are not going to configure static variables at the playbook level, rather during the task execution these variables will be created.

os = ubuntu 64 bit
user = sriman
host = jdk8

[java_servers]
java_server1 ansible_host=192.168.1.1 java_version=jdk8
java_server2 ansible_host=192.168.1.2 java_version=jdk9
java_server3 ansible_host=192.168.1.3 java_version=jdk8

- name: dynamic playbook variables
hosts: java_servers
tasks:
- name: create variable
set_fact:
env: pass
when: []

- name: task2
when: [env = 'pass']
- name: task3
when: [env = 'pass']
- name: task4
when: [env = 'pass']
...

How to register variables to hold the module outcome?
When we run an module in ansible, it produces the outcome of its execution (usually the module outcome is a json). we want to capture the module output into a variable, so that it can be used for either displaying as part of playbook execution for debugging purpose, or want to pass the outcome of the module as an input to another module. This can be done using register variables in ansible.

- name: register variables in tasks
hosts: all
tasks:
- name: find the operating system information

command: uname -a
register: osversion
- name: print os details
debug:
msg: "{{osversion}}"
...

Ansible Facts (or) Magic Variables

What are ansible facts or magic variables of ansible?
When we ask Ansible control node to execute a playbook on the managed nodes/fleet servers, ansible in order to optimize the execution of the modules on the managed nodes, it has to gather the information about the managed nodes like hardware, platform and others.

For eg.. when we define a package module for installing the software, ansible has to identify the operating system and its distribution to find what is the appropriate package managed is available for the platform to install software package unless it can execute the module. So ansible gathers facts on each managed node before executing any of the modules/tasks of the playbook.

Note: For each playbook execution, ansible control node executes gathering_facts as a default task to access the information of the Managed node on which it is executing the playbook.

Gathering Facts : will be done by the ansible control node on managed nodes by executing a set of python scripts/modules, so unless we have python installation available on managed nodes we can gather the facts.

Ansible Control node upon gathering facts of the control node it stores the facts information in a variable called "ansible_facts", so that we can access all of the facts information out of json data kept in ansible_facts variable in a playbook execution. So, from this we can easily customize the playbook executions specific to various conditions written based on facts.

There is an ansible adhoc module called setup which we can run on managed nodes to access the facts information.
In a playbook we can access facts information using ansible_facts['variable_name']

By default ansible control node gathers the facts of the managed node as a first task of a playbook execution, this will kills the performance of a playbook execution, sometimes we dont want to gather facts for simple tasks/modules executions, in such a case we can disable facts collection by using
gather_facts: no in a playbook

How to gather facts in a playbook?

- name: gather facts
hosts: all
tasks:
- name: show linux distribution
debug:
msg: "managed node {{ansible_facts['nodename']}} is of os {{ansible_facts['distribution']}} version {{ansible_facts.distribution_version}}"
...

How to control the execution of playbooks on the fleet servers?

By default when we run a playbook in ansible, the control node runs the tasks of the playbook with default behaviour as below.
1. at any time the playbook task will be executed on 1 managed node only
2. each task of the playbook will be executed on all the nodes one after the other and then pick the next task.

We can change this default behaviours using various configuration options in playbook.
strategy = plugin = using which we can change the order of task executions
serial = we can control a task to be executed on how many machines as a group
run_once = we can control how many machines in a group we should apply the playbook
throttle = we can specify amount of cpu to be allocated while executing a module
forks = how many machines at once paralley the playbook should be applied.
---
1. forks
By default a playbook will be executed only on one machine at a time.

[dbservers]
30 hosts defined in this group

here the ansible control node connects to machine by machine and runs a task on each of the machines. This process of execution takes more amount of time when we have large group of machines.

we can modify a settings in /etc/ansible/ansible.cfg
forks=3 = means run the playbook parallely on 3 machines at a time, now the control node connects to 3 machines and run the tasks parallely

in otherway we can achieve it is by passing -f flag from command-line
ansible-playbook -i inventoryFile groupName -f 3 playbook.yml
---
throttle = While running a task of a playbook ansible allocates all the cpus for that task, this makes the other process running on the managed node slow when our module takes more time to execute. We can limit the no of cpu to be allocated to our task by ansible by using throttle settings.

- name: task1
script: backup_machine.sh
throttle: 1
in this task the backup_machine.sh is a script that runs for long time and if all available cpus are given to this task the overall machine performance go down, so to limit the cpus allocated for this module to execute we can throttle: cpuCount

---
run_once: true/false
is a configuration option that is available at a task level, which will applies a task only on one machine within a group.
For eg.. in below inventory we have 2 groups of 5 machines each, now when we run a task with run_once: true then it executes only on 1 machine of group dbservers-south and 1 machine on group dbservers-north.

This option is quite useful when we want apply patches or software upgrades and want to put of testing for while.

[dbservers-south]
5 machines

[dbservers-north]
5 machines

ansible facts = these are magic variables created by ansible control node for each of the managed nodes of the inventory while executing the playbook. ansible retrieves the managed node information and stores in a variable "ansible_facts", so we can access the information using this variable in playbooks.

how to change the control flow execution of ansible playbooks
By default ansible control node executes the playbook in below process.
1. one playbook task on 1 machine at a time
2. one task at a time on all the machines then goes to the next task of the playbook.

We can alter the above execution flow by using quite a number of plugins/settings in ansible as below.
1. strategy = we can control the task executions of the playbook on fleet servers using strategy. a strategy plugin can take 2 values either free or linear
linear = this is the default execution strategy, one task at a time on all the fleet servers, then the next task in playbook
free = any order

2. serial = while patching and rolling updates we use serial for achieving zero-downtime deployments/patches
serial: 2/10% = serial is a plugin we can configure in playbook, which can take a number or percentage indicates apply the playbook for 2 machines of a group at once, the move to another 2 machines. or either percentage of machines available

3. run_once = per each group of machines, run the playbook on only one machine. Ideally it is useful while testing patches / upgrades of the software packages. So, that we can put the environment under observation for few days and can apply the patches/upgrades to all machines of the group later.

4. throttle = to limit no of cpus allocated to a task while execution

5. fork = how many machines on which a playbook/task of it should be executed parallely. we can either pass it as part of /etc/ansible/ansible.cfg or through command-prompt ansible-playbook -i inventoryFile group -f 2 playbook.yml. -f = stands for fork
---
Working with Ansible Handlers.
By default all the tasks in a playbook are executed in linear strategy from top to bottom. and each of the modules of ansible follows idempotency principle.
idempotancy principle = running a module several times on a managed node, will have a post effect of running it once.

if we write 2 tasks in a playbook as shown below, once the task1 has completed execution, irrespective of the outcome of task1, flow of execution goes to task2. But I want task2 to be an dependent task of task1, which means if task1 has effected/changed the final system state of the managed node then only I want to execute task2. This can be accomplished using handlers in ansible.

- name: task1
...
- name: task2
..

- name: task3
..

What are handlers?
By default all the tasks that are written in playbook will be executed from top to bottom, but we want few tasks to be executed only based on outcome of others.
If we observer all are tasks only, we have 2 groups of them, one should be execute by default and others only when some other task is has changed the system state. So to let ansible identify these we need to classify tasks into 2 groups.

tasks:
- name: task1
...
notify: task3
- name: task2
...

handlers:
- name: task3
...

apache2-playbook.xml

- name: handlers playbook
hosts: all
tasks:
- name: apache2 install
apt:
name: apache2
state: present
update_cache: yes
become: yes
notify:
- restartapache2
handlers:
- name: restartapache2
service:
name: apache2
state: restart
...

Ansible handlers are same as the tasks in ansible, but these tasks only gets executed based on the outcome of executing another task rather than directly.
There are certain tasks we want to perform only based on the outcome of executing a task for eg.. we want to restart an apache2 server post configuration of a site task or after adding a module into apache2 server.
Similar we want to restart ssh server after modifying the /etc/ssh/sshd_config file settings, such tasks those has to be executed based on the change of the other tasks can be configured as handlers in ansible.

Instead of using handlers if we directly write them as part of ansible task declarations those tasks will be exected always irrespective of outcome the previous one.

```
---
- name: ansible handlers playbook
hosts: all
tasks:
- name: install apache2
apt:
name: apache2
state: present
update_cache: yes
become: yes
notify: apache2_restart
handlers:
- name: apache2_restart
service:
name: apache2
state: restart
become: yes
...
```

Instead of using the Handler name to notify an handler for execution we can even use listen address to a handler to notify.

```
---
- name: ansible handlers playbook
hosts: all
tasks:
- name: install apache2
apt:
name: apache2
state: present
update_cache: yes
become: yes
notify:
- restart_servers
handlers:
- name: apache2_restart
service:
name: apache2
state: restart
listen: restart_servers
become: yes
- name: restart_tomcat
```

service:
name: tomcat
state: restart
listen: restart_servers
become: yes
...

In the above we have register the handlers to an listen_address so that when a task after completion notifies all the handlers with that listen_address for execution.
---

working with change_when conditions
---
idempotancy:- the post effect of performing an operation for multiple times will yeilds the same outcome of executing it once.
(or)
The end system state would not be effected uniquely even we perform the operation repeatedly

All the modules of ansible are designed to work based on idempotancy principle. so even we execute an ansible playbook for several times on the fleet/servers or managed nodes the final system state would not be affected.

That is where ansible before executing any module performs a pre-check, to verify whether the system has already been in desired state or not. if not then the module will be executed on the managed node and reports the module has changed the system state with changed(1). If not the module would not be considered as changed the target system and would count as well.

ansible shell/command modules doesnt follow idempotancy principle, at the end of the execution of these modules irrespective of the outcome of their execution they always report changed(1). So instead of always these modules marking the final system state as changed, we want to report them as changed only the real system has been affected this can be achieved using changed_when conditions.

There are 2 types of conditions that can be attached to a task that marks the task execution as changed or failed.
1. changed_when = condition = if the condition written here evaluates to true then only the task will reported as changed.
2. failed_when = condition = if the condition we wrote here evalues to true then task will be reported as failed.

For example we want to install mysql server, but the pre-requisite for installing mysql server is diskspace must be more than 8gb. In such a case we can write shell module to execute an shell command and get the diskspace, if the return value of the execution is more than 8 gb then we would to mark the shell command as success otherwise mark it as failed, so that playbook execution will be stopped on the host. This can be achieved using failed_when condition.

- name: failed_when condition
hosts: all
tasks:
- name: check diskspace
shell: "df -h /dev/sda5 | grep -V Filesystem | awk '{print $4}' | cut -d 'G' -f1"
register: dskspace

```
failed_when: "dskspace.stdout | float < 8"
- name: install mysqlserver
apt:
name: mysql-server-8.2
state: present
update_cache: yes
become: yes
...
```

---

vagrant = virtualization workflow automation tool. It helps us in provisioning the infrastructure and enables us to use the infrastructure/hardware of the computer as a code "its an iac tool" infrastructure as a code

We can quickly setup development environments for the developers of the project and we can implement easily the ci/cd pipelines in testing and release of the software.

Vagrant just helps us in building the infrastructure, but on top of infrastructure we need software packages to be installed and configured to run the software applications. These software installations and configurations cannot be done using Vagrant.
1. Manually
post provisioning of the infrastructure developer/devops engineer has to login to the machine and has to manually install software and configure it for use
1.1 Manual installation of softwares and configuring them takes lot of time and looses the fundamentals of using iac tools
1.2 There are bunch of packages and configurations need to be done, which is very difficult to memorize and perform repeatedly
1.3 high chances of conducting these operations wrongly which eventual leads to re-provisioning everything from scractch which kills lot of time

1.4 In general we have large network of computers are there in an organization to manage, now keeping track of all those systems and identifying which machines runs with which software and their configurations is very difficult.
1.5. Upgradation of the softwares are going to very complex and time taking, why because we need to identify all the machines of the network to find who is running with older versions of the software to upgrade.

So to overcome the above challenges in managing the software configurations on the machine manually we need automation in place

Shell scripting:-

One way to accomplish automation is through shell scripting = programmers / devops engineers can comeup with shell scripting to automate the efforts of software installations and configurations. we get every other benefit of automation like
1. reduced time in installations and configurations
2. no need to memorize the steps
3. there is no chance of something goes wrong as the human is not involved
4. if there is a changed required in upgradation of the softwares, we can reprogram the shell script to achieve it easily.

Looks like the shell scripting is the quickest way of achieving the automation of software configurations but it has plenty of problems.
1. The people has to have programming language background to write shell script for automation, seems like most of the ops engineers doesnt have exposure to programming languages and often find it very complex to achieve automation through shell scripting.
2. idempotancy = the post effect of performing an operation for N times should be equally same as the 1st time.
If we write a shell script for software installation and configuration and if you run it for multiple times the effect of running the script for multiple times should not leave the system in in-consistent state.
so to achieve idemopotancy through shell scripting is very complex
3. shell scripting is poor in logging debugging the shell programs are very difficult
4. there is no error handling mechanism is built as part of the language

5. shell scripting is platform dependent and will not works on other platforms, even on different flavours of the linux also we cannot run the shell script if it is build for one platform.

Then where does people use shell scripting?
If there are simple automations that are seems to be one-time and once done we throw away the code we have written, in such cases we go for shell scripting.

Working with control statements in ansible playbooks

Ansible supports 2 types of control statements as part of the playbook
1. when condition
if we want to execute an ansible task based on the conditional evaluation then we can use when statement. most of the time we build when condition based on ansible facts or other modules outcome.

2. loops
Loops are meant for executing an ansible task repeatitively over the list of values or given range of times.

#1 ansible when condition
If we want to execute an ansible task based on the conditional expression being evaluated to true then we need to use ansible when condition.

We can build when condition expression based on three things
1. ansible facts
2. registered variables (outcome of another task)
3. variables

what is the difference between failed_when|changed_when and when condition in ansible?
failed_when and changed_when are used for reflecting the task status whether it is changed or failed based on an command execution

where as when condition is being evaluated before execution of the task, based on the outcome the task will be either executed or skipped.

syntax:-
- name: taskName
module:
arg:
when: expression

- name: conditional playbook
hosts: all
tasks:
- name: install curl on ubuntu distribution
apt:
name: curl
state: present
update_cache: yes
become: yes
become_method: sudo
when: ansible_facts['distribution'] == "Ubuntu"
...

#2 ansible loop statements
In general loops are used for executing a block of code repeatitively based on some condition instead of duplicating the block code for several times.

Here in ansible we want to execute a task for multiple times by passing different set of inputs instead of duplicating the task definition for several times, then use ansible loop statements.

I want to create 3 users on fleet servers

- name: create users playbook
hosts: all
tasks:
- name: add administrator
user:
name: administrator
state: present
- name: add csr
user:
name: csr
state: present
- name: add support engineer
user:
name: support
state: present
...
Another example is we want to install 4 packages tree, curl, vim, net-tools, now to do that we need to write 4 times the install tasks if we dont have loop statements supported in ansible

In ansible we can build loop statements to iterator over
- simple list of values
- list of dictionaries
- indexed list
- loops with conditions

#1 simple list of values

- name: simple list of values demonstration
hosts: all
tasks:
- name: add users
user:
name: "{{item}}"
state: present
with_items|loop:
- administrator
- csr
- support
...
ansible 2.5 onwards the loop keyword is added, so instead of with_items we can write using loop also here.

we can build loops over dictionary of values as well.

- name: administrator
uid: 1004
password: welcome1
shell: /bin/bash
- name: csr
uid: 1005
password: welcome1

shell: /bin/bash
- name: support
uid: 1006
password: welcome1
shell: /bin/bash

The above is called list of dictionaries

- name: add user
user:
name: "{{item.name}}"
uid: "{{item.uid}}"
password: "{{item.password}}"
shell: "{{item.shell}}"
become:yes
become_method: sudo
loop:
- name: administrator
uid: 1004
password: welcome1
shell: /bin/bash
- name: csr
uid: 1005
password: welcome1
shell: /bin/bash
- name: support
uid: 1006
password: welcome1
shell: /bin/bash

Vagrant = Infrastructure automation, provisioning the virtual machines on the local machine environment

Terraform
Cloud = Infrastructure Service (infrastructure as a service) [Compute Node]
[ec2 instance, compute instance, virtual machine]

By using these iac tools we grab infrastructure, but still we need software packages and configurations to be applied on the environment to use it.
How to get these software packages and configurations to be applied on that infrastructure to have the machined used for running applications?

#1 Manually install and configure which has many dis-advantages obviously to ahchieve large network of computers to be setup and configured the manual process is not idle. so, required automation capabilities

#2 Automation of Software configurations

2.1 Shell scripting
We can write shell programs that takes of performing the installations and configurations of softwares on the machines, so that those can be used for running our software programs.
dis-advantages:-
1. Lack of programming skills makes the ops engineers complex in working them.
2. Acheiving idempotancy in shell script automation is very complex
3. poor logging capabilities
4. debugging is a tedious process, we need to rely on bunch of echo statements to debug the scripts
5. In case of event of failure of the execution of the script, there is no way to apply corrective action, because there is no error hanlding support
6. not platform independent and even we cannot achieve platform portabiblity between different distro's linux also

Conclusion:- Shellscript should not be used for achieving complex automations. We can use it only to perform one-time operations where we just to upgrade a software on the array of computers and there after we can dis-card.

2.2 Python
Python is a platform independent scripting language, and it was initially meant for build programs in acheiving server-side automations and some administrative related activities as an alternate to shell scripting.
advantages:-
1. its a complete programming language can take the advantage of the programming features in achieving automation
2. there are lot of opensource python modules are the using which we can quickly develop script automations.
3. good error handling support
4. good logging support
5. it is easy to achive platform independency with python with little more efforts of programming
dis-advantage:-
1. very complex to program and definitely you should be a programmer to use it.
2. acheiving idemopotancy is very difficult.
3. orchestrating the several code automations are highly complex, because programmer has to manage the several aspects of automation to work together.

Conclusion:- Phython has its dedicated usage, where complex server-side automations are being managed using python but not of software configuration management.

To overcome the above dis-advantages we have software configuration management tools brought into the market.
1. Ansible (popular)
2. Chef
3. Puppet

How does a typical software configuration management tools looks and how do they work?

How many types of Ansible Control Statements are there?
There 2 types of control statements
1. Conditional Control Statements
2. Loop Control Statements

1. Conditional Control Statements = based on the conditional evaluation, we want to execute an ansible task.
syntax:-

- name: task
moduleName:
args:
when: expression
2. Loop Control Statements = We want to execute a task repeatitively over a set of inputs instead of duplicating the task with different inputs
We can build in ansible with 4 different types of inputs
- simple list
- list of dictionaries
- indexed list
- loop with conditions

1. Loop with simple list
- name: install software packages
apt:
name: "{{item}}"
state: present
update_cache: yes
become: yes
become_method: sudo
with_items|loop:
- curl
- tree
- net-tools
- vim
2. loop over list of dictionary

- name: create users
user:
name: {{item.user}}
uid: {{item.uid}}
password: "{{item.password}}"
shell: "{{item.shell}}"
become: yes
become_method: sudo
loop:
- user: bo
uid: 1004
password: bo1
shell: /bin/bash
- user: jo
uid: 1005
password: jo1
shell: /bin/bash

Loop with indexes

- name: indexed list
debug:
msg: "vegetable : {{item.1}} at location {{item.0}}"
with_indexed_items:
- brinjal
- tomato
- carrot
- cabbage
- potato

item = is a map object above, where key and value
item.0 = index position of the value and item.1 gives actual data at that position

---
loop with conditions

net-tools
debconf-utils
expect

curl
tree

- package: curl
distribution: redhat
- package: tree
distribution: redhat
- package: debconf-utils
distribution: Ubuntu
- package: expect
distribution: Ubuntu

- name: ubuntu packages
package:
name: "{{item.package}}"
state: present
update_cache: yes
become: yes
become_method: sudo
when: ansible_facts['distribution'] == "{{item.distribution}}"
loop:

We can automate the installation software packages and configurations using python language. Python is a scripting and programming language, mostly python is being used for build administrative and complex server-side automations.
advantages:-
1. its an full fleged programming language, so we can take full advantage in developing automation scripts
2. platform indepenent
3. huge pre-built code modules, through which we can achieve automation scripts easily
4. error handling support
5. logging and debugging is supported

dis-advantages:-
1. requires programming background, non-programmers often find it difficult to build automation scripts
2. achieving the idemoptancy is very difficult
3. programmers has to build the script logic to achieve platform independence
4. orchestration is not easy

To make the software package installations and configuration automation easy, there are quite a number of tools introduced into market.
1. Chef
2. Ansible
3. Puppet

How does these tools work, what is their general architecture?
These Software Configuration Management tools are of 2 types
1. Pull based architecture / Agent
2. Push based architecture / Agentless

playbook
--syntax-check = check whether the playbook we wrote is syntactically correct or not
--list-hosts = before running the playbook we can get the matching hosts on whom this playbook gets executed.

Ansible Vault

In ansible we write playbooks/inventoryfiles and vars_files describing the machine state configuration information of the fleetservers / managed nodes that we would want to achieve. As part of these configurations we might write configuration information related to how the software packages has to be installed and configured as well. For eg.. in a mysql server installation playbook we might hold the information about the port on which mysql should run and remote user information like username/password with which we can connect the mysql server remotely might be kept.
All this information is confidential and should not be made accessible to others, if anyone has access to the playbooks or vars_file can see the details the machines can gain access to the fleet servers. This would leads to the chance of compromising the network or a machine.

How to keep such confidential information pertaining to our fleet servers written in playbook/ vars_file to kept secure?
That is where ansible-vault comes into picture. ansible-vault is a tool provided by the Ansible, using which we can encrypt/decrypt the playbooks and vars_files using AES256 algorithm based encryption. any playbook encrypted with ansible-vault will be stored on filesystem in encrypted format only. So people if they try to access them directly the can only see encrypted text only. We can run the playbooks or apply vars_file only after decryption.

How to create a playbook in encrypted format using ansible-vault?
ansible-vault create playbook.yml|dbservers_vars = This opens an editor and allows you to type the contents into the playbook or vars_file once done save and close it, so that ansible-vault prompts for an password and will encrypts the contents and store it on your filesystem.

ansible-vault view playbook.yml = it asks for password, upon providing it, it decrypts and shows the contents of the file.

How to encrypt an existing playbook with us?
ansible-vault encrypt playbook.yml = this will prompts for a password and encrypts the contents of the playbook.yml and saves it.

How to decrypt an encrypted playbook or vars_file?
ansible-vault decrypt playbook.yml = prompt for password and decrypts and writes the plain content into the file.

How to run an encrypted playbook?
There are 2 ways
1. first decrypt the playbook and execute it. With this approach we have problems.
- each time to execute a playbook we need to decrypt and execute
- post completion of the usage if we forgot to encrypt back we endup in security problem
2. while running the playbook itself pass the password to ansible asking to decrypt and execute using the below command
ansible-playbook -i inventoryFile group/pattern --ask-vault-pass playbook.yml
when we run the above command it prompts for password and decrypts during execution and runs the playbook, the underlying file contents will not be decrypted. Here the problem is interactive-inputs by which we cannot achieve touchless automation.
To overcome this problem ansible as provided alternate

ansible-playbook -i inventoryFile hostPattern --vault-password-file=locationFile playbook.yml

In this case we need to create an vault-password-file with password to decrypt the contents and store on the filesystem. The file permissions must be only to user (600). Preferrable create this file as hidden file.

How to edit the playbook that is already encrypted?
ansible-vault edit playbook.yml

How to change the password of already encrypted file?
ansible-vault rekey playbook.yml

We can categorize SCM Tools into 2 categories based on their architecture.
1. pull based architectures
2. push based architectures.

1. pull based architectures:-
The examples of pull-based architectured scm tools are Chef, Puppet. In these pull-based architecture scm tools we have
1. scm workstation = devops engineer uses it for applying the automations on the infrastructure
2. scm server/master node/repository = we write scm scripts and places them in scm server/ repository to apply them on all the nodes of the fleet. The server distributes the scripts to all the nodes of the fleet based on criteria and records the outcome of execution of the automations.
3. Fleet Nodes = Here the SCM Agent will be installed and configured to talk to SCM Server. they periodically poll the scm server and checks any scripts to be applied on the node, downloads the scripts executes them locally and records the outcome of execution back on scm server

advantages:-
1. scalability
2. parallel execution
3. we can scheduled the scripts to be executed

dis-advantages:-
1. very complex architecture
2. Setting up the infrastructure takes lot of time and may require additional tools for automating the process of setup
3. less secured, if the master or server is compromised then the whole n/w is compromised
4. On-Deman exection is not supported
5. Not easy to monitor
6. SCM Server changes again we need to reconfigure all the nodes of the fleet

Push-based architecuture:-
The Control Node in order to apply the automation script on the Fleet servers, it will ssh on to the Machines and pushes the script and executes them locally on each Fleet server. As the Control Node itself is pushing the script onto the Nodes to be executed it is called "Push based Architecture".

advantages:-
1. Agentless, which requires minimal setup or configuration to manage the fleets
2. On demand execution of the automations is possible
3. very secured compared with agent architectures
4. Easy to Monitor
5. the change on control node has zero impact on the Fleet
6. rolling updates are possible and zero-down time patching can be achieved.

dis-advantages:-
1. scalability, as the Control Node connects to each Fleet node in applying the automation, more nodes on the network takes more time in applying the automation.
2. scheduled automations are not possible
3. suitable for small/moderate network group of computers and cannot be used for larger networks

There are multiple scm tools like Chef, Ansible, Puppet and Salt across these tools there is not standard language in which we need to write automation scripts. Each of these tools has comeup with their own way of writing the automation scripts by introducing their own languages

and conventions. So we see a lot of difference in one scm tool to another in-terms of working and learning as well.

Chef:- The developers of the Chef scm tool has designed the language based on Food domain, so the naming conventions of the Chef tool resembles the Food relation terms.

Chef = cooks something, here Chef cooks automation on the Fleet servers
Cookbooks = what should be cooked on the servers (automation scripts)
Reciepes = actions we want to perform on the servers
Knife = using which we apply cookbook recepies on the Fleet servers

Cookbook -> automation script file -> we write reciepes -> Chef Engine -> through knife -> asking to apply on Fleet.
Recepies = are blocks of code written in ruby language those perform specific task as part of automation.

receipes -> shell commands (an action)
cookbook
Here cookbooks has programmming language syntaxes, so the Ops engineer has to have programming background in learing and writing cookbooks
Receipes - are ruby language programs, now we might have to write our own receipes in enhancing the Chef system which requires Ruby language knowledge.

advantage:-
platform independent

Puppet:-

Resources declaration = (an action)
Puppet manifest [Puppet language]

Ansible:- is written in Python language.

Modules = are the snippet of code written Python language using which we can perform an operation. These modules are expressed as Tasks in Ansible Playbooks
Task = that has to be executed/performed
Playbook = List tasks to be accomplished

yaml language it is simple to work with ansible and doesnt requires any programming language background

ansible-vault create playbook.yml
ansible-vault view playbook.yml

interactive:- ansible-playbook -i inventoryFile pattern --ask-vault-pass playbook.yml
non-interactive:- ansible-playbook -i inventoryFile pattern --vault-password-file location playbook.yml

ansible-vault encrypt playbook.yml
ansible-vault decrypt playbook.yml
ansible-vault edit playbook.yml
ansible-vault rekey playbook.yml
---
jinja2 templates

While working on software configuration management automation, we may have to configure lot of configuration files pertaining to the machine / server on which we are installing or configuring the software.
These configurations can be created with static values so that can installed/configured on the managed nodes directly. but different servers or softwares while installing requires different values pertaining to the environment on which we are installing and configuring, so having static values in these configuration files makes us difficult in achieving automation, why?

Because for each machine/env or software we are installing/configuring we need to maintain a copy of the configuration, which is a tedious job and endup in having lot of duplicate copies of the same configuration with few values being changed.

To save us in managing such configurations files ansible has provided jinja2 templates. Its an module written in python in managing configurations.

Instead of writing configuration files with static values we tokenize these configuration files using jinja2 template engine. during the playbook execution we pass values to those tokens asking to replace the configuration files with the input values we supplied, in this way we can avoid keeping multiple copies of the same configuration file with few different values.

To work with jinja2 templates first we need to write configuration file in a template format using jinja2 tokens. Every template file in jinja2 will be created with the filename ending with ".j2" to identify it as jinja2 template file not the actual configuration file as shown below.

hosts.conf.j2 (jinja2 template file)
<VirtualHost *:80>
DocumentRoot "/var/www/{{hostingDirectory}}"
ServerName {{serverName}}

Other directives here
</VirtualHost>

The possible tokens we can use are
{{}} :- an input value is supplied matching with key that has to be replaced while copying the configuration file
{% %} :- we can write loops and conditional statements also based on which the template file will be replaced
{# #} :- comments

To replace hostingDirectory and serverName in above configuration lets write a playbook

jinja2-playbook.yml
- name: jinja2 playbook
hosts: all
vars:
hostingDirectory: localnews
serverName: www.localnews.com
tasks:
- name: jinja2 template task
template:
src: hosts.conf.j2
dest: /home/sriman/hosts.conf

In the above playbook we are using template module which takes src as template file path and looks for tokens {{tokenName}} and sees the input values supplied for the playbook execution if found replace these tokens with actual values by copying into the destination location provided as dest:

We can use loops and conditions in replacing the tokens as below.

system.conf.j2
packages:
{% for item in packages %}
{{item}}
{% endfor %}

users:
{% for item in users %}
{{item}}
{% endfor %}

systeminfo.yml
packages:
- curl
- vim
- tree
- net-tools
users:
- joe
- bob

j2loops-playbook.yml
- name: j2loops playbook
hosts: all
vars_file:
- systeminfo.yml
tasks:
- name: j2 template file with loops
template:
src: system.conf.j2
dest: ~/system.conf

We can classify the scm tools into 2 types based on thier architecture.
1. pull-based scm tools
eg.. Chef, Puppet

2. push-based scm tools
eg.. Ansible

There are lot of advantages with push-based scm tools.
1. no need of installing and configuring agents on the Fleet Nodes.
2. secured ssh protocol to connect to fleet servers, so the infrastructure is very secured.
3. on demand execution and zero downtime patching is supported
4. Easy to Monitor the infrastructure
dis-advantages:-
1. cannot scaleup to large network of computers

What the conventions/terminologies of various different scm tools?
Every scm tool has comeup with their conventions/terminologies in designing their own language for writing automation scripts. From this we can understand each of these scm tools has their own language in which we need to write automations.

Chef:- has been greatly designed and motivated from the conventions and terminologies of food domain, that is the reason the name has been given as Chef.

Cookbooks = The contain script automations expressed in terms of recepies
Receipe = a unit of code that is designed to perform an operation or a task
Knife = is a software used to communicate with Chef Server

Puppet:-
Resources = defines an operation/task to be performed
Puppet manifest = puppet manifest contains resource declarations (Puppet language)

Ansible:-

Modules = are written in mostly python language which performs a unit of work / operation
Task = To accomplish a Task we need to execute multiple actions expresed in terms of Modules
Playbook = Playbook contains list of tasks to be performed to accomplish an automation

---

How to host a static website on apache2 server?
1. apache2 server
2. copy the site directory into apache2 hosting location like /var/www/hostingDirectory
3. write the apache2 virtualhost conf file for enabling the site
4. sudo a2ensite siteName -> if site is enabled

5. restart the apache2 service (handler)

apache2-site (project directory)
|-apache2-site-playbook.yml
|-site.yml
|-apache2.conf.j2
|-www
|-index.html
|-store.html

apache2.conf.j2

<VirtualHost *:80>
DocumentRoot "/var/www/{{site.hosting_directory}}"
ServerName {{site.site_name}}

Other directives here
</VirtualHost>

site.yml
---
site:
hosting_directory=toystore
site_name=toystoreonline

apache2-site-playbook.yml

- name: apache2 site playbook
hosts: all
vars_file:
- site.yml
tasks:
- name: install apache2 server
apt:
name: apache2
state: present
update_cache: yes
become:yes
become_method:sudo
when: ansible_facts['distribution'] == 'Ubuntu'
- name: create hosting directory
file:
path: /var/www/"{{site.hosting_directory}}"
state: directory
chmod: 755
owner: sriman
group: sriman

```
become: yes
become_method: sudo
- name: copy website into hosting directory
copy:
src: www/*.html
dest: /var/www/"{{site.hosting_directory}}"
chmod: 655
owner: sriman
group: sriman
- name: configure apache2 virtual host configuration
template:
src: apache2.conf.j2
dest: /etc/apache2/sites-available
become: yes
become_method: sudo
- name: apache2 enable site
shell: a2ensite "{{site.site_name}}"
become: yes
become_method: sudo
register: a2enstatus
changed_when: "a2enstatus.stdout == 'Enabling site {{site.site_name}}'
notify:
- restart apache2 server
- name: print a2ensite
debug:
msg: {{a2enstatus}}
- handlers:
- name: restart apache2 server
service:
name: apache2
state: restarted
become: yes
become_method: sudo
...
```

Ansible roles

ansible roles is a way we can group all the tasks of a playbook into a container or an organized directory structure through which we can apply and achieve automation. It is a more organized manner of writing a playbook with tasks, handlers, files, templates.

How to create an ansible role?
1. create an empty directory representing the role you want to create
/ansible/apache2 (directory name itself acts as a rolename)
2. go into ansible/apache2 directory and execute the command
ansible-galaxy init apache2

ansible-galaxy is a cli tool provided by ansible to create roles with necessary directory structure with default files init.

/ansible/
|-apache2
|-README
|-defaults
|-main.yml = declare variables with default values to be used as apart of tasks/handler execution
|-tasks
|-main.yml = all the playbook tasks
|-handlers
|-main.yml = all the playbook handlers
|-files = files to be used to copy on to the remote managed server
|-templates = j2 templates
|-vars
|-main.yml = these variables will overwrite the values we defined under default/main.yml
|-meta
|-main.yml = describe information about the role (owner, license, version)

What are ansible roles?
While working on complex automations we might have several tasks to be executed as part of the playbook. if we write all the tasks/handlers as part of single playbook, it becomes complex to understand and we cannot reuse few of the tasks across the other playbook automations.

So we can break down the playbook tasks and handlers into smaller collection/group of tasks/ handlers which are nothing but ansible role. In this way we can group all the related tasks to achieve an automation into several roles and we can reuse them across multiple playbooks.

For example to deploy a java application on a ansible managed node we may have to carry several tasks like
- mysql server installation and configuration tasks
- install jdk
- download and configure tomcat as a service
- copy the application war into deployment directory of the server
- start/restart tomcat service

If we write all of the above activities as tasks in a playbook, the playbook becomes quite lengthy and difficult understand. Here few of the tasks like mysql server installation and configurations, install jdk might be common for other application automations as well, but still we cannot reuse if those are part of the playbook.

If we can write these tasks as part of ansible roles like mysql role, java role, we can reuse these roles accross all the playbooks.

ansible roles provides 2 things
1. we can achieve reusability of group/collection of ansible tasks/handlers to achieve automation across multiple playbooks.
2. cleaner directory structure in which we can organize tasks, handlers and other files to achieve automation

How to create an ansible role?
by default ansible roles are created under the directory /etc/ansible/roles

ansible-galaxy init /etc/ansible/roles/apache2
ansible-galaxy is a cli tool provided as apart of ansible install and init is used for initializing the role with directory structure with default files.

apache2
|-default
|-main.yml
|-tasks
|-main.yml
|-import_tasks apache2-install.yml
|-import_tasks copy-site.yml
|-import_tasks configure-and-enable-site.yml

|-apache2-install.yml
|-copy-site.yml
|-configure-and-enable-site.yml

|-handlers
|-main.yml
|-files

|-templates
|-meta
|-main.yml
|-vars
|-main.yml

Now break down our apache2 automation into the above roles directory structure

privelige escalations:
become: initiate privelige escalation
become_method: sudo = tells execute the ansible modules on managed node using sudo

become_user: joe = while executing the module sudo su joe and execute the module
remote_user: we are telling ansible control node to connect to managed node through ssh using remote_user

Basic Terminology related to Ansible.
1. Control Node
The machine on which ansible has been installed using which we apply software configuration automation.
2. Managed Node
The machine on whom we apply scm automation scripts from Control Node though ssh.
3. Inventory
Its a file in which we capture the infrastructure information to pass it as an input to the Control Node.
4. Module
Module is a code block written in mostly python language, that perform an scm operation/action.
5. Task
Grouping of Multiple Modules to be executed together to accomplish a work.
6. Playbook
is written in Yaml format and holds the list tasks containing modules to be applied on target machine.

There are 2 types of scm tools are there
1. programmatic tools
In programmtic tools the ops engineers has to write programming logic to accomplish automation, here we write the code to achieve the final desired state of the system

2. declarative scm management tools
We dont write instructions to perform automation, rather what we want we declare what we want on the end system for e.g.. we say apache2, status: available, now the scm tools itself takes care of ensuring that the system is installed/updated with apach2 server version you have specified.

How to install Ansible?
Ansible cannot be installed on Windows Machines, even though a support of running ansible on Windows is there having it installed and running on Windows is quite complext.

Ansible
Windows 10 cant install Ansible, there is a support for running Ansible directly on Windows but very difficult to achieve.

Windows Subsystem Linux
We should turn on this Feature on Windows 10 operating systems. Using WSL we can run a compact version of Linux operating in Windows, so that most of the basic features works.

In Windows 10 update Microsoft has provided WSL2 version.
There are improvements when compared with WSL (previous version)
1. WSL2 uses virtualization and runs Linux machine on Hyper-v
2. Full Kernal execution support
3. Separate File system

If we enable WSL2 then Virtualbox will work. But we are looking for running Ansible Control Node on WSL and Managed nodes on Virtual Machines using Virtualbox

Ansible Installation

Enable Windows Sub-system Linux WSL2 on Windows 10 Home/Professional

1. Upgrade Windows 10 to 19041.450 build by downloading the Windows Update Tool
https://www.microsoft.com/en-in/software-download/windows10
Windows 10 May 2020 Update
The Update Assistant can help you update to the latest version of Windows 10. To get started, click Update now.
OS Build (19041.450)

---
2. Enable Windows Subsystem Linux 2 by running the below commands.

open powershell [administrator mode]
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart

3. Enable Virtual Machine Platform
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart

4. Updating the WSL 2 Linux kernel
https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi

5. Set WSL2 Default Version as 2
wsl --set-default-version 2

Install Ubuntu 20.04 by going to Microsoft Store and Launch machine.
7. Convert Ubuntu 20.04 from WSL2 to WSL1 by running below command on commandprompt with administrative rights.

verifying the current WSL Linux distribution wsl -l -v you should see current version as 2
NAME STATE VERSION
* Ubuntu-20.04 Stopped 2

then run the below command to convert.
wsl --set-version Ubuntu-20.04 1

after that again run wsl -l -v to verify the change in version.

8. Disable Virtual Hypervisor platform by running the below command on cmd prompt.
[administrative rights]
bcdedit /set hypervisorlaunchtype off

9. restart windows 10

Steps for installing the Ansbile WSL Ubuntu 20.0
1. sudo apt update
2. sudo apt-get install software-properties-common
sudo apt install -y ansible

Cloud Computing = is an on-demand availability of computing resources to the end-users. They day to day infrastructure requirements of running your software applications on the computing resources can be made available through cloud computing.
There are 3 services offered by the cloud computing vendors.
iaas = infrastructure as a service
paas = platform as a a service
saas = software as a service

depends on the cloud computing vendor they provide all of the above services are could be few of them as well.

iaas = infrastructure as a service provides the infrastructure aspects of running the software applications. The cloud vendors provides virtual resources using virtualization technology rather than providing physical machines to us so that they can serve the machines based on the requirements of the users.

in iaas we can create a machine of a customized shape based on the requirement of your business.We never endup in paying more for un-used resources of the machine. We can always can increase or decrease the hardware resources of our machine on demand within couple of minutes.

To manage and provide virtual machines on demand the cloud providers uses some softwares called hypervisors and special softwares for managing these hypervisors as well.
We can signup for using a machine for few hours as well and even for months/years there is no upfront comittment of usage of machines.

All the cloud services are offered in 2 billing models
1. pay-per usage model
When we take a machine from cloud vendor, if we dont use it and turn it off we never need to pay. pay for only the hours of usage.
Pay for the amount of data transfer you do the network rather than paying for the communication line
Pay for the storage allocated to you.

2. fixed rate
You upfront reserver a computer of fixed shape and endup paying the same cost for the whole year irrespective of the usage.

Enable Windows Subsystem Linux on Windows 10 to install Ansible.
WSL2 is the current version, and it uses Hyper-V virtualization engine for running Linux Kernal as an Virtual machine, so we cannot run oracle virtualbox in parallel when we enable Hyper-V.

For our Ansible we want to run Ansible Control Node on WSL and Managed Nodes on Virtualbox, so to do this we need to switch back to WSL instead of WSL2 post installation of the Ubuntu on WSL2.

Once we setup the Control Node and Managed Nodes we need to apply software automation.

What is cloud computing?
Cloud Computing is an on-demand supply of computing resources to the end-users. The cloud computer providers offers 3 types of services.
1. iaas = infrastructure as a service
2. paas = platform as a service
3. saas = software as a service

The cloud providers offers the computing resources whether it could be a machine|network| software on below principles
1. on-demand = whenever we request a machine or a network or a software application they provide immediately with little or no latency.

2. elastic = the resource they offer are flexible where we can scaleup these resources or scaledown these resource with minimal effect or no effect on resource availability.

3. billing
There are 2 billing models the cloud providers are going to offer
1.pay per usage = each of the resources we are using will be billed based on the usage of them, we dont need to pay for the whole month even we use the resources per a day or 2
The pay per usage model differs from resource type to resource

fixed shapes
Debain Ubuntu 20.04 | 2.4 ghz | 4 gb | 10 gb = 0.001$/per sec = 24/7 - 30day
Windows 10 | 2.4 ghz | 4 gb | 10 gb = 5$ / per hour

2.fixed rate
If we are using the resources with a sustained usage then the cummulative cost in pay-per-usage model will be little high, so cloud providers offers fixed pricing model where irrespective of hours of usage of the machine they fix price for each of the resource on monthly bases.
So that we are being charged on fixed amount rather than usage.

The above services are offered by the cloud providers using different technologies like virtualization and bare metal models which indicates they dont serve physical systems , rather they provide virtual resources to the end user.
---

1. iaas = infrastructure as a service (almost every cloud providers serves iaas)
The cloud providers offers the physical resources like machine, network, routers, firewalls etc on-demand for usage. Like how do we buy a computer from a computer market and use it. here also we are going to get a compute machine baked with an operating system with network resources attached.

We can install our own software packages and we can deploy or run our own applications on these compute machines. They provide a dedicated ssh access to these machines using which we can access the compute instances locally from our computer.
---

2. platform as a service
To run software applications that are built by the end-user we not only need a compute machine, we need middleware software to be installed on that computer.

Middleware = The software that we use underlying to make the end-user application work on a computer is called "Middleware" software.

For e.g..
1. jdk software
2. .net sdk
3. python
4. nodejs
5. npm
6. database server like mysql server, mssql server, oracle database server
7. mongo db
9. tomcat, weblogic server, websphere server, glassfish server, jboss wildfly server
10. Message queues (kafka) | (rabitmq) | (mqserver)

Now if we take a iaas compute machine, we cannot directly run end-user software on the machine, because we need middleware software to be installed. To have them install on compute instance, the end-user has to manually setup the software packages as the same way he would install locally on his machine.

Manually installing, configuring and managing the middleware softwares has lot of problems.
1. To setup the middleware software on the compute instances takes lot of time, we may to do several activities to have these middleware software running on the compute instance like, configuring firewalls, security rules, writing configuration files etc, this delays the deployment and delivery of the application

2. periodically software updates and patches will be released by middleware vendors, it is highly critical to ensure these patches are applied on the compute instances, where we are using the middleware software. uptaking these patches are not so easy
2.1 we need to ensure the patch that is released by the middleware is safe to use and will not break the system we need to check. So we have to setup test env equal to our prod and apply the patch on test env and perform through testing then apply the patches on the prod env by taking the prod env backup.

2.2 periodically apply these patches is going to be quite difficult job, certifying and applying the job requires a separate team to be maintained which is going to be costly

3 when we want to scale the compute instances, we not only need to take another compute instance from the cloud provider, again we need to re-setup the middleware software and do necessary configurations to make both machines work as a cluster.

Instead of we taking care of installing/configuring/managing the middleware software cloud providers offers paas services to us. For each of the middleware software there is a paas offering. For eg..

1. java as a cloud service = a compute instance with jdk software being installed will be provided to us directly
2. weblogic server as a cloud service = the provision a compute instance with weblogic server installed and configured for ready to usage.
3. mysql server as a cloud service = they provision a machine with msql server database installed configured with firewall, security rules so that the database is not comprised by the attacks
3.1 periodical backup of mysql server database will be taken care
3.2 crash recovery
3.3 backup and restoration in event of failure
3.4 on heavy loads scale the mysql server on mutiple machines

3.5 including licensing fees
---
saas = software as a service

To serve an organization requirement we need to usually build software applications.
Financial system, of a company (automobiles) like audits, accounts, balance sheet.
- We need build a software system that can take care of handling financial aspects of the company

being an automobile business vendor, he himself cannot build an software application for their organization, they need outsource the software requirements to another software services company. To have the software application being build by the outsourced company we need to do the following.
1. our business team regularly should be in touch with outsourced company in providing our requirements.
2. allocating budget and planned release of financial resources to the other company based on software progress
3. monitoring the development process
4. verification has to be done regularly to ensure the people build the software as per the expectition

On top of the above we need to invest big way money for building software application and wait for a long-term period to get it build and delivered. where the investment that made becomes dead util the software application build comes under use.

Financial management system software is an Enterprise need that every requires to maintain their financial records. There are few technofunctional experts who has both financial domain knowledge and software development knowledge, using their enourmous amount of experience knowledge they build common software system we can be categorized as product that can be installed and used directly by enterprises in the market which are called "ERP" Systems.
- salesforce
- oracle fusion apps
- ebusiness suite
- peoplesoft
- jd adwards
- sap
- tally

Now our company dont need to build software from scratch, rather we can procure license for any of the ERP Softwares and can adopt and use it in our organization.

To adopt these ERP there is a lot of difficulties or challenges are involved.
1. To run these ERP softwares on the organization infrastructure, we need high-grade server machines with huge computing capacity required. The cost of setting up such an infrastructure is very high
2. The licensing cost of these ERP very high where a moderate or small company may not afford it.
3. Requires consultants in adopting these ERP Systems in our organization, because they need special installation and configurations to run on a machine
4. ERP systems has to be customized based on the organization requirement, which is called implementation/adoption of the ERP requires special manpower trained on this job

5. 24X7 monotoring | backup | recovery and regular maintainance activities has to be conducted to ensure these systems works. This requires a crew in doing this job
From the above we cannot understand clearly these can be used only Enterprises only. To rescue from this problem saas (software as a service) comes into picture.

Software as a Service (SaaS) = provides these enterprise software systems pre-installed and pre-configured on the iaas layer.

SaaS works in 2 ways
- multi-tenancy model = one erp system will be shared to multiple users on a multi-tenancy model, so that each enterprise which is using the erp system believes they have their own dedicated erp system for their use even though the underlying infrastructure and erp system is shared across multiple enterprises.

The advantage of multi-tenant model here is
1. keeping the cost of using the ERP is very low
2. Small and Moderate enterprises can make use of ERP in running their business

- private cloud
- a dedicated machine is alloted with saas implementation on it so that your organization can use it without sharing with others.

In generate with SaaS model we have plenty of advantages
3. No installation/configuration is required since cloud provider takes of them
4. backup / recovery and restoration in the event of crash will be taken care by cloud provider
5. autoscaling based on traffic and hanlding performance issues at peak load.

By the above we can understand a cloud provider offers various services and manages their lifecycle operations.
1. provisioning
2. scaleup
3. scaledown
4. scaleout
5. scalein
6. backup
7. restore
8. deprovisioning