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

Vagrant box file acts as a template in creating an virtual machines, by importing vagrant boxfiles we create virtual machine disk image per virtual machine.
We might want to create multiple virtual machines we same operating system and software utilities & tools and configurations so, instead of creating machines by following the length process of installation and configurations we can create a boxfile that consists of the software tools and configurations which can be imported in creating virtual machines quickly.

How does vagrant up will brings up the vagrant machines?
There are 3 stages in which a vagrant machine will be brought up
1. download the Vagrant boxfile from Vagrant cloud
based on the config.vm.box = "Hashicorp/precise64" value in Vagrantfile vagrant engine goes to the vagrant cloud and locates the vm.box file and downloads and places in the local machine under directory $VAGRANT_HOME\.vagrant.d\boxes

2. import the boxfile into the project directory.
The vagrantbox file is a compressed file that will be extracted into an virtual machine disk image file under directory ~/Virtualbox VMs/timestamp and the extracted file will acts as an disk image for the virtual machine

3. booting up the virtual machine
in the project directory it creates a folder called .vagrant inside which it stores box metadata information
While creating the virtual machine it generates an unique id through which the machine is identified.
Along with that it creates a metadata file in which it stores information about virtual disk image location to be used for running the virtual machine, machine name and various configurations related to synced folders and networking etc.
In addition it generates public/private key and seeds the public_key on the virtual machine and stores the public/privates keys locally under .vagrant directory so that vagrant ssh command reads these keys in connecting to the virtual machine.

What are the states in which a vagrant machine can exists?
There are 5 status/states in which a vagrant machine can exists
1. Not Created = The vagrant machine has not yet been created and there doesnt exists any .vagrant directory under the project directory which results in not created status

2. Running = The vagrant machine has been booted up and running and .vagrant directory has been created associating with virtual machine

3. Poweroff = The vagrant machine has been halted or shutdown

4. Saved = The current state of the Virtual machine will be written on to the disk of the host computer and machine will be stopped moving it to saved state. So that when we bring up the machine, the previous state of the machine will be brough back.

5. Aborted = if we kill the running vagrant machine by forcebily by pressing ctrl + c then the machine will be moved to aborted state. (Terminated unsuccessfuly)

vagrant up
vagrant status
vagrant suspend
vagrant resume
vagrant halt

What are the different statuses in which a vagrant machine can be there?
There are 5 statuses in which a machine can exists in vagrant.
1. not created = we have Vagrantfile created for our project, but we havent started atleast once the machine using vagrant up, so that vagrant machine has not created, i.e.. it reports status as not created
2. running = if the vagrant machine is running with vagrant up command then the machine is in running status
3. saved = when we suspend a running machine then the machine goes to saved status, so that upon resuming the machine the programs will be restored to the previous state.
In order to resume the programs to the previous state, while resuming vagrant stores the current state of the machine onto the local disk location of the host computer.
4. poweroff = after the bootup of the machine if we halt the machine then it goes to poweroff state
5. aborted = if we forcebily terminate a vagrant machine during startup then it goes to aborted state

cli commands
---
vagrant init
vagrant status
vagrant up
vagrant halt
vagrant suspend
vagrant resume
vagrant reload = if we have modified vagrant configuration in Vagrantfile to have these changes reflected we need to stop/restart the machine instead of running 2 commands we can use reload as handy command that does the job
vagrant ssh = login to the vagrant machine remotely
vagrant destroy = deletes the virtual machine including the disk image

1. What is Vagrant, why do we need it?
2. How does Vagrant helps the developers in quickly setting up the development environments and test engineers to quickly test and certify the applications?
3. The role of devops engineers in producing Vagrant machines to both developers/test engineers
4. vagrant architecture / components of vagrant
5. vagrant features
6. How to create an Vagrant Machine?
7. What is vagrant boxfile how does it acts as an template in creating vagrant machine?
8. advantages of vagrant
- multiple hypervisor providers
- vagrant configuration is portable across various hypervisor providers
- unified workflow for developers/testers/engineers in managing the virtual machines across any hypervisor provider
9. vagrant machine states
10. vagrant cli

There are several things we can configure for a virtual machine while creating. for e.g..
1. network configuration
2. mount locations
3. hardware configuration

can we have these configurations been setup as part of vagrant machine configurations, so that we can bring up the virtual machine with underlying hypervisor provider?
Yes, ofcourse vagrant supports these configurations as part of Vagrantfile which are nothing but features of vagrant.

#1 synced folders
It is used for sharing the directories of the host machine to the guest machine. we can configure one of the folder of the host computer as a synced folder so that it would be mounted onto the guest machine can be accessible from guest.

This is one of the common feature every hypervisor supports and we can make these configuration in vagrant file so that while booting up the machine we can have synced folders accessible in the vagrant machine.

by default vagrant will automatically adds the project directory as a synced folder and is mounted onto the vagrant machine under /vagrant directory. so that the Vagrantfile of the project will be visible to the vagrant machine.

We can additionally configure folders to be synced with vagrant machine by specifying the Vagrantfile as below.

Vagrantfile

```
Vagrant.configure(2) do | config |
config.vm.box = "ubuntu/focal64"
config.vm.synced_folder "source", "mountLocation"
end
```

Networking
A Virtual Machine has to be connected to the host, other guest machines or external network to be accesible, which can be done using virtual network.
In Virtualbox what are the network modes that are supported?
1. Not attached
2. NAT
3. NAT Network
4. Internal Network
5. Host-Only Network
6. Bridge Network
7. Generic Adapter

How can we have a vagrant machine being created with different network modes?
Vagrant supports 3 types of network configurations to be configured as apart of Vagrantfile
1. forwarded port
2. private network
3. public network

synced_folders :-
We can mount host machine directories onto the guest machine using sync folders, so that we can access all the files/folders of the host machine directory inside the guest under mount path.

by default vagrant configures project directory as a synced folded under /vagrant path of guest machine

Vagrantfile
Vagrant.configure(2) do | config |
config.vm.synced_folder "source", "mountLocation"
end

Vagrant Networking:-
A Virtual Machine should be configured/setup with Networking to enable it accessible by host, other guest machines or external network so that it can be used for running multi-layered application deployments.

Hypervisor providers enable us to configure virtual network adapters that can be attached to the virtual machines, so that those can be brought onto the network.

What are the different networking modes supported by Oracle Virtualbox hypervisor provider?
1. Not Attached
2. NAT ------ vm -> host | vm -> external n/w
3. NAT Network ------ vm -> host | vm -> external n/w | vm -> vm [on same nat network]
4. Internal Network ------ vm -> vm [same internal network]
5. Host-Only Network Adapter ------ host -> vm | vm -> host
6. Bridge Network ------ public
7. Generic Driver

Instead of we setting up the virtual network adapters on the virtual machines of the hypervisor provider, we want vagrant to provision the virtual machines with network configurations provided. That is where vagrant supports 3 types of networking configurations.

1. forwarded_port = it is nat with port forwarding option in virtualbox
usage:- during the development time the developer develops the application in virtual machine environment and inorder to test the application he want to access the appplication from host machine. Here Host machine should be able access the application on a specific port in which case we can configure NAT with port forwarding thus making the virtual machine highly secured by allowing only one port exposed to the host.

2. private_network = Internal Network with Network Name configured so that all the Virtual Machines on the same internal network can communicate.
usage:- in a multi-tier application deployments we want database virtual machine to be accessible only to the application virtual machine but not to the external network. In such a case we configure both application/database virtual machine on the same private_network so that those 2 machines can communicate with each other.

3. public_network = Bridge Network which opens up everything to the world. It is less secured and not recommended in production deployments
usage:- used for general purpose development environments or if we are working on experimental projects where we are least bothered about network security rather wanted all the Virtual Machines to be accessible then go for public network.

How to configure these network options?

forwarded_port:-

Vagrantfile

```
Vagrant.configure(2) do | config |
config.vm.box = "Hashicorp/precise64"
config.vm.synced_folder "source", "mountLocation"
```

syntax:-
config.vm.network "networking mode", options

```
config.vm.network "forwarded_port", guest: 80, host: 8081
end
```

The above configuration indicates the traffic to host machine 8081 port will be forwarded to the guest machine 8080
Only the host computer running with ip address: 192.168.23.4 should be allowed to forwarded to guest machine thus making more secure

```
config.vm.network "forwarded_port", guest: 80, host: 8081, host_ip: "192.168.23.1"
```

Now the Virtual Machine can be accessed by the specific host ip computer only.

There are 3 networking configurations support by vagrant

1. forwarded_port = nat network configured with port forwarding from host to vm/guest
Vagrant.configure(2) do | config |
config.vm.box = "hashicorp/precise64"
config.vm.network "forwarded_port", guest: 80, host: 8080
end

any request of any ip address on port 8080 will be forwarded to guest instead we can configure a specific host ip: address request to be forwarded to the guest by using below configuration

config.vm.network "forwarded_port", guest: 80, host:8080, host_ip: "192.168.1.8"
---
2. private_network = it maps to host-only network adapter mode of virtualbox, in private network all the virtual machines connected to the same network can communicate with each other.
usage:- in multi-tier application deployments we use private network

private network configuration:-

#1
Vagrant.configure(2) do | config |
config.vm.network "private_network", type: "dhcp"
end
In the above configuration we havent specified the network name so it acts as host-only network adapter and every guest will be on individual network. "dhcp" is responsible for generating ip address for all the guests.

#2
Machine-1
Vagrant.configure(2) do | config |
config.vm.network "private_network", ip: "192.168.1.2"
end

Machine-2
Vagrant.configure(2) do | config |
config.vm.network "private_network", ip: "192.168.1.3"
end
In the above vagrant network configuration rather than enabling "dhcp" we did static ip binding that allows application to use fixed ip address so that we dont need to modify the code for deployment.

#3
Machine-1
Vagrant.configure(2) do | config |
config.vm.network "private_network", virtualbox_intnet: "network1"
end

Machine-2
Vagrant.configure(2) do | config |
config.vm.network "private_network", virtualbox_intnet: "network2"
end

In the above configuration we can apply static ip also or dhcp also along with that we are

specifying internal network name with which the virtual machine should be started.
all the virtual machines connected to the same internal network can communicate with each other.

3. public_network

What is private network how is it configured with the underlying hypervisor provider?
A private network is configured as an Host-Only Network adapter on the underlying hypervisor, so that host can access the guest and guest can access the host and all the guests on the same private network can access each other.
usage:- used under mutli-tier application deployments

#1 dhcp
config.vm.network "private_network", type: "dhcp"
#2 static ip
config.vm.network "private_network", ip: "192.168.23.1"
#3 private network name
config.vm.network "private_network", ip: "192.168.23.2", virtualbox_intnet = "network1"
---
public network

The public network is configured as brige network with the underlying virtualbox provider so that the virtual machine is open to all for access. In general it is not recommended to use public_network as it opens to everyone and poses a security problem.

How to configure public network?

#1 - default network adapter uses to configure bridge network
Vagrant.configure(2) do | config |
config.vm.box = "hashicorp/precise64"
config.vm.network "public_network"
end

#2 - static ip
config.vm.network "public_network", ip: "192.168.1.23"

#3 we can specify the network adapters to be bridged
config.vm.network "public_network", bridge: [
""
]
---
vagrant ssh

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.

public network configuration
---
config.vm.network "public_network"
config.vm.network "public_network", ip: ""
config.vm.network "public_network", bridge []

what is vagrant ssh?

Vagrant during the time of provisioning the virtual machine will configures the SSH access to the machine enabling it for remote access to the developers / devops engineers to use it.

The base vagrant boxfile we use comes up with default installation of openssh server so that when we create a vagrant machine by importing the boxfile it is created with ssh server pre-configured and enabled.

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?

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

What is ClassPath in Java Application, where do we use it?
The java compiler (javac) or java virtual machine (java) in order to compile the classes or execute them they should locate the .class files of the Java Classes that are under the filesystem location of our machine.

```
class Test {
public static void main(String[] args) {
System.out.println("Test completed");
}
}
```

javac Test.java
Test.class

java Test
CLASSPATH is an environment variable that points to the directory location of the .class files. So that javac and java will search for .class files based on this variable pointed location directories. if we dont set any CLASSPATH environment variable the default location is "." (Current Directory)

What are Jar files in java?

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

classpath = its an environment variable that should be pointed to the directories (or) jar files containing ".class" files of java application. java runtime engine and java compiler looks for ".class" files under CLASSPATH environment variable pointed directory locations.

Jar = Java Application Archive
There are 2 ways we use Jar files in a Java Application

#1 to distribute java application as libraries to others [distributable jars]

In a typical java application, we can reuse the libraries that are build by other developers rather than building the application from scratch this saves lot of cost/time in development of the application.

Here libraries are nothing but bunch of classes that are written by someothers and can be reused in building our application.
While building libraries we need to distribute ".class" files rather than source code. How to distribute ".class" files of our application to others?
1. We cannot give directory containing files to the others, as directories are non-transferable
2. package the directory contents into zip file and distribute to others. The people who want to use your library ".class" files has to unzip your file and place it in a directory use your library this makes it a tedious job.
- here we need a zip utility software for zipping/unzipping the files to be used.

That is where java has introduced a packaging standard "Jar" using which we can compress and package ".class" files of a directory into one single file so that we can distribute it easily to others.
The people dont need to uncompress the jar file to use ".class" files in it, rather they can pass the Jar file directly to javac/java,these tools are capable of reading .class files from a Jar file directly.

How to create a Jar file with our ".class" files?
Java has provided an utility called jar as part of Jdk software distribution. we can use this tool for creating and extracting a Jarfile

create jar file
---
jar -cvf jarname.jar directory
c = create
v = verbose [print the output of the operation you are doing]
f = filename

extract jarfile
---
jar -xvf jarname.jar
x = extract
v = verbose
f = filename

#2 deliver java applications to the end-user

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 does vagrant ssh works?
in the vagrant boxfiles we use are baked with
- operating system
- ssh server software
- and a global in-secured public/private key
using which vagrant provisions the vagrant machine, while creating the vagrant machine on the underlying hypervisor provider it configures networking as nat with port forwarding with host port: 2222 to guest: 22 to facilate ssh into the guest.
it configures the ssh server with configuration as PasswordAuthentication No
For the 1st time while booting up the virtual machine it generates a pair of public/private keys and ssh into the guest with insecured keys and replaces the insecured keys with generated keys then stores the public/private key in .vagrant directory of the project and boots the machine.

when we use vagrant ssh command it works only from relative to project directory it picks the public and private key from .vagrant directory and login into the guest/remote machine using ip as localhost and port as 2222 (since port forwarding is configured)
---
vagrant provider configuration
Vagrant is a virtualization workflow automation tool that works with multiple hypervisor providers like virtualbox, hyper-v, VMWare fusion, VMWare desktop, VMWare workstation, parallels etc.
The default hypervisor provider vagrant uses in provisioning the virtual machines is oracle virtualbox provider and it supports all the versions of virtualbox from 5.1 to 6.x

How can we change the default virtualization provider of the vagrant?
There are 2 ways to do this
#1 Globally change the virtualization provider so that all the machines we create uses the provider we configured through environment variable.
set VAGRANT_DEFAULT_PROVIDER=vmware_desktop

#2 Change provider for a specific virtual machine rather effecting globally
vagrant up --provider=vmware_desktop

A Virtual Machine / Vagrant Machine provisioned with one hypervisor provider cannot work with other provider, we cannot run a virtual machine with hyperv provider when it has been created with virtualbox provider. In order to do this we need to destroy machine and recreate.

Vagrant Providers

To provision a vagrant machine we need vagrant boxfile, the boxfiles are provided by many people around the world and has published them in vagrant cloud.

boxfiles are specific to hypervisor provider and supports creating vagrant machine only on that provider. while publishing a boxfile to vagrant cloud the author of the boxfile will bind the boxfile with metadata saying it works with which hypervisor provider and other details.

The author/creator of a boxfile can comeup with multiple copies of the boxfile to work with different hypervisor providers.
For e.g.. hashicorp/bionic64 supports 3 hypervisor providers vmware, virtualbox and vmware_desktop

Vagrantfile

```
Vagrant.configure(2) do | config |
config.vm.box = "hashicorp/bionic64"
end
```

In the above configuration we said the boxfile name as "hashicrop/bionic64", when we do vagrant up vagrant automatically chooses the appropriate box file based on the hypervisor provider we are using in creating the machine. in our case it uses virtualbox boxfile.

"This is called automatic pickup of the boxfiles based on provider"

Vagrant configurations we write will work with multiple hypervisor providers by default because of vagrant intelligent chooses the right boxfile based on the provider.
We can say vagrant configurations are portable across the providers, due to this we say vagrant is a virtualization workflow automation tool that can work with multiple hypervisor providers, we dont need to know different hypervisor providers, we just use vagrant that takes care of working with multiple providers.

Vagrant supports customization of vagrant machines specific to the provider.
For eg.. we can tell vagrant when you are creating vagrant machine on virtualbox create network configuration as "forwarded_port" and if it is hyperv create network configuration as "private_network"

we can write provider specific configurations that works with a specific provider so that when we switch the hypervisor provider still the configuration file works.

2 Types of provider specific customizations are supported by vagrant

#1 customize the machine based on provider
#2 apply provider specific configurations based on provider

thus eliminating the need of writing multiple vagrant configuration files for each provider.

1. Using Environment variable so that it would effect all the vagrant machines being provisioned on that host
set DEFAULT_VAGRANT_PROVIDER=hyperv

2. We can pass --provider switch while running vagrant up, this would effect only the current machine we are staring
vagrant up --provider=hyperv

Note:- after the vagrant machine was created, we can switch the hypervisor provider in running the machine, if we want to change the provider on which we want the machine to boot we need to destroy and recreate

Vagrant boxfile is specific to hypervisor provider on whom we want to create vagrant machine. The author/publisher of the boxfile creates multiple copies of the boxfile each for an hypervisor provider as well.

Choosing boxfile automatically by vagrant
---
so while choosing a boxfile in vagrant configuration only we need to tell the baseName of the box file without provider, so vagrant automatically picks the relevant boxfile based on the hypervisor provider we are running.
so thus the need of modifying the vagrant configuration file for switching between multiple hypervisor provider has been eliminated.

The Vagrant configurations like synced_folders, network configurations we write in Vagrantfile are portable and works across multiple hypervisor providers so that we dont need to modify the Vagrantfile while switching between provider.

By the above we can say Vagrantfile is portable across the Hypervisor Providers.

Problem #1 = [Customize the Vagrantfile based on Hypervisor provider]
But for example we want to customize the Vagrant Machine based on the hypervisor provider on whom we are running the machine.
Lets say while creating the Vagrant Machine on Virtualbox we want the machine to use hashicorp/precise64 box file with networking mode as forwarded_port and if we run the same Vagrantfile on Hyperv we want the machine to use hashicorp/bionic64 and network configuration as private_network. How to customize the machine based on the Hypervisor provider on whom we are creating the machine?

Problem#2 =
There are some configurations/settings that changes from hypervisor provider to another. For eg.. in virtualbox it allows a machine to be configured with Ram, Cpus, display memory etc and these could be different in Hyperv may it support similar settings or altogether different
Having these provider specific settings being written as part of Vagrantfile makes the file not portable across the hypervisor providers
How to solve this?

#1 customize the vagrant machine based on hypervisor provider
Vagrantfile

```
Vagrant.configure(2) do | config |
#goes to default provider
config.vm.box = "hashicorp/bionic64"
```

config.vm.network "forwarded_port", host: 8081, guest: 80

config.vm.provider "hyperv" do | hv, ovveride |
override.vm.box = "hashicorp/precise64"
override.vm.network "private_network", ip: 192.168.2.13
end
end

#2 How to incorporate provider specific configurations in Vagarntfile and still make it portable across the hypervisor providers

Vagrantfile

Vagrant.configure(2) do | config |
config.vm.box = "hashicorp/ubuntu64"

#directly we are writing provider specific configuration here the it will try to apply for all hypervisors and eventually leads to failure in creating the box because of un supported settings

config.vm.provider "virtualbox" do | vb |
vb.gui = true
vb.name = "java development machine"
vb.memory = 2048
vb.cpus = 2
end
end

jar stands for java application archive
There are 2 reasons for which we use Jar in Java Application
1. distributable jars = these jars are shipped as libraries so that other application can use these jars by adding them to the classpath of the application
2. executable jar = to deliver application to the end-user

jar -cvf filename.jar directory
jar -cvf filename.jar . -C directory
jar -xvf filename.jar
jar -tvf filename.jar

~/workspace/java/
gtm
|-src
|-com.gtm.service
|-TaxAnalyzerService.java
|-TaxManagerService.java
|-bin
|-com.gtm.service
|-TaxAnalyzerService.class
|-TaxManagerService.class

~/workspace/java/gtm/$ jar -cvf gtm.jar . -C bin
How does the Jar file looks like, what is its structure?

jarfile.jar
|-META-INF
|-manifest.mf
|-packages
|-*.class

META-INF = Meta Information means information about the project will be described under META-INF directory inside the manifest.mf file. manifest.mf refers to the information about the project published to the world. in manifest.mf we can write projectName, version, author, license etc.

We can run class files in a jar directly by setting the CLASSPATH environment variable pointing to Jar file.
workspace/java/library/
bin
com
library
shelves
Catalog.class
library.jar
src
com
library
shelves
Catalog.java
Catalog.java
package com.library.shelves;
public class Catalog {

```
public static void main(String args[]) {
System.out.println("books in library");
}
}
```

after compiling while running the above class we need to set CLASSPATH to the library.jar file

```
$export CLASSPATH=~/workspace/java/library/library.jar
$java com.library.shelves.Catalog
```

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.

Vagrant providers
#1 by setting the global env variable
set VAGRANT_DEFAULT_PROVIDER=hyperv

#2 pass the switch while running vagrant up
vagrant up --provider=hyperv

destroy and recreate to switch between the providers after provisioning

can the vagrant configuration is portable or not?
How about the customizations we want to make for an Vagrant machine based on the hypervisor provider?
Yes we can do that by using hypervisor provider block in vagrant file.

```
Vagrant.configure(2) do | config |
config.vm.box = "hashicorp/bionic64"
config.vm.network "forwarded_port", host: 8081, guest: 80, host_ip: "192.168.2.12"
config.vm.provider "hyperv" do | hypv, override |
override.vm.box = "hashicorp/precise64"
override.vm.network = "private_network"
end
end
```

#2 how can we apply provider specific settings for a vagrant machine and make sure those are portable while working with various providers?

```
Vagrant.configure(2) do | config |
config.vm.box = "ubuntu/bionic64"

config.vm.provider "virtuabox" do | vb |
vb.memory = 2048
vb.cpus = 2
vb.gui = true
vb.name = "java machine"
vb.linked_clone = true
vb.customize ["modifyvm", :id, "--cpuexecutioncap" , "50"]
end
end
```

What are vagrant boxfiles?
vagrant boxfiles are the the template files using which we can create a vagrant machine.

Each time while creating a virtual machine with a hypervisor provider we need to go through length process of installing the operating system from an iso and software packages required, make configuration necessary for running the programs. This takes lot of time in creating an virtual machine on any hypervisor provider.

instead of doing this manually vagrant compresses the virtual machine image file and packages the machine configurations into a boxfile which we can directly import in creating any no of vagrant machines quickly.

The hashicorp team has provided basebox files which we can directly use for creating virtual machines. Not only the hashicorp lot of vendors/organizations has contributed in publishing the basebox files which we can use based on our requirement.

We always can create vagrant machines directly from the vagrant boxfiles that are available in vagrant cloud, but these has certain programs.
For eg.. in a project to setup a development environment in a virtual machine we can create vagrant machine from a boxfile, but the boxfiles comes with only operating system and basic set of software utilities, but inorder to run the project we need project specific software packages and configurations, so developer after creating the vagrant machine out of the boxfile he has to manually install and configure software packages on the machine to run the project. and the samething has to be applied across various developers in the team, that kills the productivity of the entire team in setting up the development environments

How to solve this problem?
We can setup our own vagrant box file out of the virtual machine we have created.
(or)
Creating project specific Vagarnt boxfiles.

How to create our own Vagrant boxfile?
There are 2 ways of creating an vagrantbox file
1. Use one of the Vagarnt basebox files provided in creating your own vagrant boxfile
Create an vagrant machine from vagrant basebox file so that all the basic configurations will automatically comes to our machine, then install project specific softwares required and apply configuration settings to the run the project. Then export the virtual machine we setup now into boxfile.
This process is the quickest way of creating our own boxfiles for our project requirement.

2. Building vagrant boxfile from scratch iso.
Now we setup virtual machine manually from an iso image and do necessary installations and configurations for
1. running the machine as vagrant machine
2. softwares and configurations required for running project.

This process is complex and takes little more time in creating vagrant boxfiles.

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

What is vagrant boxfiles?
boxfiles are templates using which we can create vagrant machine quickly. These are pre-packaged virtual machine image files with configurations which we can import in creating the vagrant machine quickly.

While working on the project to produces similar development environments across the developers machines rather than setting up the environment manually from a base box file we can create our own boxfile from the existing environment.

There are 2 ways of creating an vagrant boxfile.
1. creating an vagrant boxfile from base boxfile
- Provision an vagrant machine from vagrant base boxfile, with this all the basic set of things like os, ssh server and other configurations to act this as an vagrant boxfile comes automatically on top of install and configure softwares required for running your project. Then export the virtual machine image we created as vagrant boxfile.

2. creating an vagrant boxfile from the scratch
- Without using any of the base boxfile we create an virtual machine from an scratch using iso, do necessary configurations to make the machine work with vagrant. Then install and configure s/w libraries and packages that are needed to run the project. Then export it as an boxfile.

In the above approaches the #1 has advantages it is quick to setup and easy to create the boxfile. but there are some challenges involved.
1. In an organization it may not be feasible to create an boxfile out of an base boxfile since it might poses a legal/licensing issues.
2. There is nothing as an authority certifying us to use the base boxfile produced by someone to be used as part of our organization.
3. There could be organization policies due to which we might have to setup and use proprietary softwares that are allowed which might stop you in creating a boxfile from basebox files.

Due to the above problems many of the times organizations may have to create boxfiles from scratch.

Steps to be followed in creating our own boxfile from scratch.
1. Create an virtual machine in virtual box
- goto virtual machine setings and make the below changes
disable audio and usb access [For security reason]
increase the processors allocated from 1 to 3/4
configure NAT network with port forwarding as host: 2222 on to the guest: 22
2. Start the Virtual Machine
Now virtual box prompts for an iso file for installation provide ubuntu-20.04-LTS image for creating virtual machine
During the install it asks for username/password, provide the username/password as vagrant/vagrant since vagrant has to ssh into the machine during the first boot of the vagrant machine to replace in in-secured pub/private keys.

3. Install Guest Additions on the Virtual Machine to ensure synced_folders and network configurations works on the machine. then restart the machine

4. update ubuntu repository cache
sudo apt update -y

5. Install open ssh server
sudo apt install -y openssh-server

6. Configure SSH Access to the machine
- copy the vagrant insecured public key from github
- create an .ssh folder under user home
- cat "public key" > ~/.ssh/authorized_keys
- chmod 700 ~/.ssh
- chmod 600 ~/.ssh/authorized_keys

7. Configured password less sudo access for the root user
visudo /etc/sudoers.d/vagrant
vagrant ALL=(ALL) NOPASSWD:ALL

8. enable ssh access by modifying /etc/sshd_config
PasswordAuthentication No
authorizedKeysFile = ~/.ssh/authorized_keys
restart ssh service
systemctl restart ssh

How to build war file from below application?

```
railway-enquiry
|-src
|-*.java
|-WebContent
|-*.html
|-WEB-INF
|-lib
|-servlet-api.jar
|-jsp-api.jar
|-web.xml
|-bin
|-*.class
|-dist

|-railway-enquiry
|-*.html
|-WEB-INF
|-lib
|-servlet-api.jar
|-jsp-api.jar
|-classes
|-*.class
|-web.xml

export CLASSPATH=~/workspace/web/railway-enquiry/WEB-INF/lib/servlet-api.jar:~/
workspace/web/railway-enquiry/WEB-INF/lib/jsp-api.jar
~/workspace/web/railway-enquiry/$ javac -d bin src/*.java
~/workspace/web/railway-enquiry/$ jar -cvf rail-enquiry.war . -C dist/railway-enquiry
```

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.

How to install and configure mysql server?
#1 sudo apt update
update the ubuntu local cache

#2 sudo apt install -y mysql-server
This installs the mysql-server 8 version along with client components as well. during the installation it will create a default user called "root"
mysql server root user has access to the entire database system. he has administrative access in managing the mysql server database.
For eg..
1. can create users and grant privileges to the others
2. can create schemas and drop schemas
3. create databases and drop
but during installation the root user is created with no password so that we can connect/login to the database.

#3 run sudo mysql_secure_installation
this utility comes as part of mysql-server installation, which is used for configuring the database server we installed.
will creates root user with password and allows you to configure default schemas and remote access to the root user by asking questions.

post installation of mysql server, we need to enable remote access to the mysql server by following the below steps.
#1 change bind-address to let mysql server listen for requests that are coming from remote computer as well.
sudo vim /etc/mysql/mysql.d/mysqld.cnf
modify bind-address from 127.0.0.1 to 0.0.0.0/0
then restart mysql server through systemctl

#2 configure remote access to the mysql users
create user 'rail_user'@'%' identified by 'password' with grant option;
grant all privileges on *.* to 'rail_user'@'%';

Now we create a root user for remote access of the database and we grant all privileges to the user like connect, resource, create, drop, alter on all the objects of the database where he can login from remote computer of any ip address '%'

Now prepare your database for application access.
create schema and tables for running your application.The developer has given db-schema.sql which contains appropriate scripts for creating schema and tables we just need to run the sql file.
using mysql client utility asking him to connect to db server and execute the sql file as below.

sudo mysql -hlocalhost -uroot -proot < db-schema.sql

Now check whether you are able to access the mysql database from remote machine by installing mysql client utility and connecting to the server.
sudo apt install -y mysql-client
sudo mysql -hipaddress -uroot -proot

How to compile rail-reservation project?
be in project directory

then set classpath
export classpath=~/workspace/web/railway-reservation/WebContent/WEB-INF/lib/servlet-api.jar:~/workspace/web/railway-reservation/WebContent/WEB-INF/lib/jsp-api.jar:~/workspace/web/railway-reservation/WebContent/WEB-INF/lib/mysql-connector-java-8.0.22.jar:~/workspace/web/railway-reservation/bin

then compile
javac -d bin src/com/railreservation/dto/*.java
javac -d bin src/com/railreservation/dao/*.java
javac -d bin src/com/railreservation/servlet/*.java

build war file structure under dist directory and copy files from project directoy
cd dist/
sriman@sriman:~/workspace/web/railway-reservation/dist$ mkdir railreservation
sriman@sriman:~/workspace/web/railway-reservation/dist$ mkdir -p railreservation/WEB-INF
sriman@sriman:~/workspace/web/railway-reservation/dist$ mkdir -p railreservation/WEB-INF/lib
sriman@sriman:~/workspace/web/railway-reservation/dist$ mkdir -p railreservation/WEB-INF/classes
sriman@sriman:~/workspace/web/railway-reservation/dist$ cd ..
sriman@sriman:~/workspace/web/railway-reservation$ cp -R bin/* dist/railreservation/WEB-INF/classes/
sriman@sriman:~/workspace/web/railway-reservation$ cp -R WebContent/WEB-INF/lib/* dist/railreservation/WEB-INF/lib/
sriman@sriman:~/workspace/web/railway-reservation$ cp WebContent/*.html dist/railreservation/
sriman@sriman:~/workspace/web/railway-reservation$ cp WebContent/WEB-INF/web.xml dist/railreservation/WEB-INF/

vagrant is an virtualization workflow automation tool that can be used by developers, qa assurance engineers and operation/support engineers.

while we are building solutions based on client/server architecture we can manage development and delivery aspects of such applications using vagrant. Typically a Web Application is an example of Client/Server architecture application.