VAGRANT

Vagrant

Vagrant is a tool for building and managing virtual machine environments.
With an easy-to-use workflow and focus on automation, Vagrant lowers development environment setup time, increases production parity, and makes the "works on my machine" excuse a relic of the past.
Vagrant is suitable for development environment.

Why Vagrant ??

Vagrant provides easy to configure, reproducible, and portable work environments built on top of industry-standard technology and controlled by a single consistent workflow to help maximize the productivity and flexibility of you and your team.

To achieve its magic, Vagrant stands on the shoulders of giants. Machines are provisioned on top of VirtualBox, VMware, AWS, or any other provider. Then, industry-standard provisioning tools such as shell scripts, Chef, or Puppet, can automatically install and configure software on the virtual machine.

For Developers

If you are a **developer**, Vagrant will isolate dependencies and their configuration within a single disposable, consistent environment, without sacrificing any of the tools you are used to working with (editors, browsers, debuggers, etc.). Once you or someone else creates a single Vagrantfile, you just need to vagrant up and everything is installed and configured for you to work. Other members of your team create their development environments from the same configuration, so whether you are working on Linux, Mac OS X, or Windows, all your team members are running code in the same environment, against the same dependencies, all configured the same way. Say goodbye to "works on my machine" bugs.

For Operators

If you are an **operations engineer** or **DevOps engineer**, Vagrant gives you a disposable environment and consistent workflow for developing and testing infrastructure management scripts. You can quickly test things like shell scripts, Chef cookbooks, Puppet modules, and more using local virtualization such as VirtualBox or VMware. Then, with the *same configuration*, you can test these scripts on remote clouds such as AWS or RackSpace with the *same workflow*. Ditch your custom scripts to recycle EC2 instances, stop juggling SSH prompts to various machines, and start using Vagrant to bring sanity to your life.

For Everyone

Vagrant is designed for everyone as the easiest and fastest way to create a virtualized environment!

What is the meaning of setting up environment ??

Let’s say you need web server
Let’s say you need db server
Let’s say you need app server

Let's say you need some machine for R & D purpose quickly, configure that machine quickly and able to use that machine very quickly, now using vagrant we can reduce installation time.

Typically when we setup OS, you may take around 30-45 min to go along with that.
But with vagrant you can spin up the development environment very very quickly.

Now how to spin up the environment ??

lets see how it can be done

Goto the official website of vagrant vagrantup.com

We see something like find boxes right, let’s understand some terminology:
So when we are going with Base Installation of Linux, we need couple of things

- You need a CD or ISO image
- You need a physical machine
- You define some CPU & RAM
- Storage
- Network

So whenever we are going with installation we need to go with all these steps always.

So if we are working with cloud we have some images like CentOS, ubuntu etc

- AMI (Amazon Machine Images)
- Virtualization Layer
- CPU {how much CPU i need}
- RAM {how much RAM i need}
- Storage {how much storage i need}
- Network

I need to configure all the above
So in vagrant, we call these pre-installed Images/OS as **BOXES**
Now you can easily download these boxes, and can spin up the virtual machines easily.

As i told to setup the OS it takes around 30-45 min, but using the vagrant it hardly takes 5-10 min depending on internet speed.

Download and Install vagrant
Vagrant supports on following platforms: WIN/MAC/LINUX

But there is another dependency to vagrant, which is the **virtualization layer**.
So in our laptop/desktop, what is the virtualization layer we use:

Oracle Virtual Box

1. Download and install {win - CMD}
# vagrant --version

Now i don't have any box right, let’s download the box:

# goto vagrantup.com
# find BOXES {gives list of all boxes}

So once you go to find boxes, it shows what’s the virtualization layer is and the diff boxes available.

There is provider, lets understand the terminology

Here **Virtualbox →   Provider**
Provider: tool which is giving you the virtual layer
But we have diff providers available which provides virtualization layer.

Getting Started
We will use Vagrant with VirtualBox, since it is free, available on every major platform, and built-in to Vagrant.
But do not forget that Vagrant can work with many other providers.

Providers
While Vagrant ships out of the box with support for VirtualBox, Hyper-V, and Docker, Vagrant has the ability to manage other types of machines as well. This is done by using other *providers* with Vagrant.

Before you can use another provider, you must install it. Installation of other providers is done via the Vagrant plugin system.

Once the provider is installed, usage is straightforward and simple, as you would expect with Vagrant.

Your project was always backed with VirtualBox. But Vagrant can work with a wide variety of backend providers, such as VMware, AWS, and more.

Once you have a provider installed, you do not need to make any modifications to your Vagrantfile, just vagrant up with the proper provider and Vagrant will do the rest:

```
# vagrant up --provider=vmware_fusion
# vagrant up --provider=aws
```

Up and Running

```
# vagrant init centos/7
# vagrant up
```

After running the above two commands, you will have a fully running virtual machine in VirtualBox running.
You can SSH into this machine with **# vagrant ssh**, and when you are done playing around, you can terminate the virtual machine with **# vagrant destroy**.

Project Setup

The first step in configuring any Vagrant project is to create a **Vagrantfile**. The purpose of the Vagrantfile is:

1. Mark the root directory of your project. Many of the configuration options in Vagrant are relative to this root directory.
2. Describe the kind of machine and resources you need to run your project, as well as what software to install and how you want to access it.

Vagrant has a built-in command for initializing a directory for usage with Vagrant:

```
# vagrant init
```

This will place a Vagrantfile in your current directory. You can take a look at the Vagrantfile if you want, it is filled with comments and examples. Do not be afraid if it looks intimidating, we will modify it soon enough.

You can also run vagrant init in a pre-existing directory to setup Vagrant for an existing project.

```
# vagrant init centos/7
```

Vagrantfile
=========
The primary function of the **Vagrantfile** is to describe the type of machine required for a project, and how to configure and provision these machines.
Vagrant is meant to run with one Vagrantfile per project, and the Vagrantfile is supposed to be committed to version control.
The syntax of Vagrantfiles is Ruby, but knowledge of the Ruby programming language is not necessary to make modifications to the Vagrantfile, since it is mostly simple variable assignment.

# **vagrant up**
==========
In less than a minute, this command will finish and you will have a virtual machine running centos 7. You will not actually *see* anything though, since Vagrant runs the virtual machine without a UI.
# vagrant ssh
===========
This command will drop you into a full-fledged SSH session. Go ahead and interact with the machine and do whatever you want.

How to work with vagrant

1. create a project directory
2. create Vagrantfile (configuration file - # vagrant init)
3. define the image name you want to use

Let’s go and do these steps

```
# mkdir project1
# cd project1
```

It's similar to git, whenever you start with git we say # git init
similarly for vagrant we use

```
# vagrant init
```

Open Vagrantfile

→ It's a ruby configuration file
→ I'll remove unwanted things like commented section, just keep

```
Vagrant.configure("2") do |config|
        config.vm.box = "base"
end
```

I want centos-7 box so let's search for it in vagrantup.com
Change the vagrantfile **# vi Vagrantfile**

```
Vagrant.configure("2") do |config|
        config.vm.box = "centos/7"
```

ed

# vagrant up

# vagrant up {this command does the following}

1. It will create a VM with name "default"
2. It downloads the centos image to project dir
3. Start the VM (by defualt it will be 1 CPU, 512MB RAM)
4. Creates NAT n/w
5. Setup SSH port forwarding
6. Installs SSH keys
7. Maps the storage
8. Execute provision script

# vagrant up {execute the command, this brings up the machine}

# vagrant ssh {logs you into the machine}

# vagrant halt {brings down the machine}

# vagrant status {status}

# vagrant port {port forwarding}

Network
=======

In order to access the Vagrant environment created, Vagrant exposes some high-level networking options for things such as forwarded ports, connecting to a public network, or creating a private network.

Port Forwarding
=============
Port forwarding allows you to specify ports on the guest machine to share via a port on the host machine. This allows you to access a port on your own machine, but actually have all the network traffic forwarded to a specific port on the guest machine.

Let us setup a forwarded port so we can access Apache in our guest. Doing so is a simple edit to the Vagrantfile, which now looks like this:

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/precise64"
  config.vm.provision :shell, path: "bootstrap.sh"
  config.vm.network :forwarded_port, guest: 80, host: 4567
end
```

Run a # vagrant reload or # vagrant up. Once the machine is running again, load http://127.0.0.1:4567 in your browser. You should see a web page that is being served from the virtual machine that was automatically setup by Vagrant.

Vagrant also has other forms of networking, allowing you to assign a static IP address to the guest machine, or to bridge the guest machine onto an existing network.

By default vagrant uses **NAT** network provided by your provider aka Virtual Box.
But let say you want to use bridge network to get connected to router, just uncomment the “public_network”, and when you do **# vagrant up**, it asks for the network to connect.

```
	# config.vm.network "public_network"
	# config.vm.network :public_network, bridge: "en0: Wi-Fi (AirPort)"
```

Configure Hostname
=================

```
	# config.vm.hostname = “centos”
```

Changing RAM
=============

```
# Customize the amount of memory on the VM:
config.vm.provider "virtualbox" do |vb|
	vb.memory = "1024"
end
```

Changing CPU
=============

```
# Customize the number of CPU’s on the VM:
config.vm.provider "virtualbox" do |vb|
        vb.cpus = "2"
end
```

Provisioning
===========
Provisioners in Vagrant allow you to automatically install software, alter configurations, and more on the machine as part of the vagrant up process.

Of course, if you want to just use vagrant ssh and install the software by hand, that works. But by using the provisioning systems built-in to Vagrant, it automates the process so that it is repeatable. Most importantly, it requires no human interaction.

Vagrant gives you multiple options for provisioning the machine, from simple shell scripts to more complex, industry-standard configuration management systems.

Provisioning happens at certain points during the lifetime of your Vagrant environment:

- On the first vagrant up that creates the environment, provisioning is run. If the environment was already created and the up is just resuming a machine or booting it up, they will not run unless the --provision flag is explicitly provided.
- When vagrant provision is used on a running environment.
- When vagrant reload --provision is called. The --provision flag must be present to force provisioning.

You can also bring up your environment and explicitly *not* run provisioners by specifying --no-provision.

If we want to install web server in our server**.** We could just SSH in and install a webserver and be on our way, but then every person who used Vagrant would have to do the same thing. Instead, Vagrant has built-in support for *automated provisioning*. Using this feature, Vagrant will automatically install software when you vagrant up so that the guest machine can be repeatedly created and ready-to-use.

We will just setup Apache for our basic project, and we will do so using a shell script. Create the following shell script and save it as bootstrap.sh in the same directory as your Vagrantfile.

bootstrap.sh
==========

```
        yum -y install git
        yum -y install httpd
        systemctl start httpd
        systemctl enable httpd
        git clone https://github.com/devopsguy9/food.git /var/www/html/
        systemctl restart httpd
```

Next, we configure Vagrant to run this shell script when setting up our machine. We do this by editing the Vagrantfile, which should now look like this:

```
Vagrant.configure("2") do |config|
                config.vm.box = "centos/7"
                config.vm.provision :shell, path: "bootstrap.sh"
        end
```

The "provision" line is new, and tells Vagrant to use the shell provisioner to setup the machine, with the bootstrap.sh file. The file path is relative to the location of the project root (where the Vagrantfile is).

After everything is configured, just run vagrant up to create your machine and Vagrant will automatically provision it. You should see the output from the shell script appear in your terminal. If the guest machine is already running from a previous step, run vagrant reload --provision, which will quickly restart your virtual machine.

After Vagrant completes running, the web server will be up and running. You can see the website from your own browser.
This works because in shell script above we installed Apache and setup the website.

Load Balancing
=============
In this project we are going to work with three different machines,
In our previous Load balance example we took one nginx and two apache servers.

But in this example we will be using three nginx servers:

Vagrantfile
==========

```
Vagrant.configure("2") do |config|

config.vm.define "lb1" do |lb1|
        lb1.vm.box = "ubuntu/trusty32"
        lb1.vm.network "private_network", ip: "192.168.45.10"
```

```
  lb1.vm.network :forwarded_port, host: 7777, guest: 80
        lb1.vm.provision "shell", path: "provision-nginx.sh"
        end

config.vm.define "web1" do |web1|
        web1.vm.box = "ubuntu/trusty32"
        web1.vm.network "private_network", ip: "192.168.45.11"
  web1.vm.network :forwarded_port, host: 8888, guest: 80
        web1.vm.provision "shell", path: "provision-web1.sh"
end

config.vm.define "web2" do |web2|
        web2.vm.box = "ubuntu/trusty32"
        web2.vm.network "private_network", ip: "192.168.45.12"
  web2.vm.network :forwarded_port, host: 9999, guest: 80
        web2.vm.provision "shell", path: "provision-web2.sh"
end

end
```

provision-nginx.sh
================

```
#!/bin/bash

echo "Starting Provision: Load balancer"
sudo apt-get -y install nginx
sudo service nginx stop
sudo rm -rf /etc/nginx/sites-enabled/default
sudo touch /etc/nginx/sites-enabled/default

echo "upstream testapp {
        server 192.168.45.11;
        server 192.168.45.12;
}

server {
        listen 80 default_server;
        listen [::]:80 default_server ipv6only=on;
        server_name  localhost;
        root            /usr/share/nginx/html;
        #index index.html index.htm;
        location / {
        proxy_pass http://testapp;
}
        }" >> /etc/nginx/sites-enabled/default
sudo service nginx start
echo "MACHINE: LOAD BALANCER" >> /usr/share/nginx/html/index.html
echo "Provision LB1 complete"
```

provision-web1.sh
==================

```
echo "Starting Provision on A"
sudo apt-get install -y nginx
echo "<h1>MACHINE: A</h1>" >> /usr/share/nginx/html/index.html
echo "Provision A complete"
```

provision-web2.sh
==================

```
echo "Starting Provision on B"
sudo apt-get install -y nginx
echo "<h1>MACHINE: B</h1>" >> /usr/share/nginx/html/index.html
echo "Provision B complete"
```

```
# vagrant status
# vagrant global-status
# vagrant port lb1
# vagrant port web1
```

Running Provisioners
==================

Provisioners are run in three cases:

```
        # vagrant up
        # vagrant provision
        # vagrant reload --provision
```