# Day 1:

***Vagrant***

* Vagrant is a tool for building complete development environments, sandbox and virtual machines.
* Vagrant reduces the time spent on setting up and providing time on making development and production efficient.
* Will provision vm’s vis shell script or configuration management using tools like Chef and Puppet.
* Vagrant will work around the known issues and make the necessary modifications on OS, this kind of fix would be done on host and guest setups.
* Setting up could be done in a flash.
* This tool is capable of handling entire lifecycle of development machines
  + Suspend, Halt, and Resume virtual machines
  + Destroy VM’s and delete data,
  + SSH into your machine
  + Package up your machines entire state and then re-distribute it to other development team members.
* This tool is must have for development environments in which we can automate an environment to resemble as production.

***Why Vagrant***

* Vagrant would ease up the work of setting up the environment for different levels.
* Vagrant would automate our setup of all services need to develop out applications, this would eliminate the human error and environmental changes for the installations.
* Each projects gets there out VM’s and developers can share the virtual machine images by using simple command which would build an exact replica of that environment in minutes.

***Vagrant Work-Flow***

* Developers get code from repositories and run ‘vagrant up’, and developers can work on their own machines and have a stable environment.
* Scripts of configuration in using vagrant to deploy dev would be same as production environment, which would help in limiting issues later on.
* If we get any issues and want to start from scratch, then we would destroy (Vagrant destroy) the machine and remove all traces and build (vagrant up) it again.
* No worry on if developer forgets to shut system on cloud which would cost money.

***Our First Vagrant Machine***

* We could start with ‘vagrant init’ and ‘vagrant up’ commands.
* For Ubuntu : vagrant init precise64 <http://files.vagrantup.com/precise64.box> [This tells vagrant that we would use init (intiate) and box name as ‘precise64’ and get the box from the url]
* Then 🡺 vagrant up
* Vagrant ssh (to enter into the machine created)
* Vagrant destroy to kill the machine and all resources.

***Vagrant Files***

* Vagrant is configured on a per project basis, so each project has their own ‘vagrantfile’.
* ‘Vagrantfile’ would be a text file which vagrant reads and setsup the environment.
* This file talks on which OS, what software to be installed etc.,
* By version controlling this file the team could be in sync with any changes happening.
* Vagrant searches for ‘vagrantfile’ in an bottom-up approach.
* Config versions for Vagrant files
  + As of date the current config version of vagrant is “2”
  + VAGRANTFILE\_API\_VERSION = “2”

Vagrant.configure(VAGRANTFILE\_API\_VERSION) do | configure |

Config.vm.box = “precise64”

end

* + Version “1” config files for vagrant version 1.0.x
  + Version “2” config files for vagrant version 1.1+ leading up to 2.0.x
  + We cannot use the newer version of vagrant for old config versions.
  + We can enforce the users to use a specific version of vagrant [Vagrant.require\_version = “>=1.3.5”] to be loaded.

***Boxes***

* Vagrant Boxes are “templates” which contains the base operating systems already setup. If we need to create a new instance it would use the existing OS from vagrant box rather than downloading it again.
* We can set the box in vagrant file, [config.vm.box = “precise64”]
* Boxes can be managed using following commands
  + Vagrant box list [This command will listout the boxes present]
  + Vagrant box add [Boxes can be added to a project using this command]
  + Vagrant box remove [We can delete the box by specifying box name after remove]
* If any point of time we want to remove the box from the project but we have something running on that then it would not cause any problems as when we start the box it clones image and files into a new machine.
* If we need it run it again then we would need the box image which would be pulled from URL and re-download the image.

***Running Vagrant Machines (Vagrant Up)***

* Once we have got the box downloaded and Vagrant file created, we need to get the OS running so for that 🡺 ‘vagrant up’ is used.
* We will not be seeing an GUI for machine, but we could see **VBoxHeadless.exe** process running.
* We can share the environment with other developers by sharing Vagrantfile and ‘Vagrant up’ command.
* When vagrant up command is executed, then it imports base box [boxname(‘precise64’)]
* Virtual box would randomly generate a separate MAC address when we create a new machine for NAT networking. Vagrant has to set the MAC address for networking to processed correctly.
* Name of the VM is set by Vagrant as [Project Folder] \_ default\_ [timestamp (which would be useful for identifying the different machines)].
* Vagrant will clear out the existing network interface details before setting up details for the current machine.
* Vagrant creates metadata for the forwarded ports which virtual box would need, may be help full while creating SSH.
* Check status of Vagrant machine 🡺 ‘vagrant status’
  + Can use ‘vagrant halt’ (shut it down forcefully)
  + Use ‘vagrant suspend’ (simply suspend the machine)
  + Either cases we can use ‘vagrant up’ to restart.

***SSH to Vagrant Machines***

* To access the machines created using Vagrant, we use SSH to connect to vagrant machine.
* If we mess up something, then we can destroy the machine and create it up using the ‘vagrant up’ command which will use the existing box present which is already downloaded.
* Using ‘vagrant ssh’ command from the project folder we can get into that project terminal.

***Synched Folders***

* Vagrant allows us to share folders between virtual machines that are running and the between the host machines which actually have the virtual machines.
* This privilege to developers on using their personal editors for the file which they can access, also gives a place to store files which could be destroyed using vagrant destroy.
* While ‘vagrant destroy’ files stored on shared folders are not destroyed, as long as shared file mapping system is present we can access the shared folders even after the ‘vagrant up’.
* By default, the project directory is shared to a folder on guest machine in /vagrant.
* We can set the custom paths for having sync folders:
  + Config.vm.synced\_folder “src/”, “/server/website”

End

* Src/ == Host folder
* /server/website == location on guest machine (Would be created if needed).
* Enabling synced folder is done by default using ‘vagrant up’ or ‘vagrant reload’
* Disabling synced folder, we need to set disabled: true under configure
  + Config.vm.synched\_folder “src/”, “/server/website”, disabled: true
* By default, the ssh owners to be set as owners for the syched folder, by sometimes we might need to change it to other users.
  + Config.vm.synched\_folder “src/”, “/server/website”, owner: “root”, group: “root”

***Basic Networking***

* To start, Vagrant sets up the network for developers so that they can start work from the word go.
* Network setup is done by Forwarding ports, forwarding port exposes a port on guest machine as a port on host machine.
* For example, if we use port 80 for a webserver as an example:
  + Config.vm.network “forwarded\_port”, guest: 80, host:8080
  + After Vagrantfile is edited then use ‘vagrant reload’, if its first time then ‘vagrant up’.
  + Start the webserver on guest machine using the following command
    - Sudo python -m SimpleHTTPServer 80
  + Then open localhost:8080 in host machine, we could be able to see the forwarding.

***Environment Management***

* Using Vagrant, if we want to keep the machine clean then we can use Suspend, Halt & Destroy.
* Suspend [‘vagrant suspend’]
  + This would a point time snapshot of the virtual machine.
  + Can be thought as freezing time.
  + During Suspend operation, our virtual machine will not access any CPU or RAM resources on host machine from Vagrant environment.
  + Doing this would have memory issues, as the more disk space would be utilized to store the image and to make the resume operation smooth.
* Halt [‘vagrant halt’ or ‘vagrant halt --force’]
  + Halt would be normal shut down, can start with ‘vagrant up’
  + Halt –force would abruptly stop without clean shut down.
* Destroy
  + This completely remove the machine, state files and database associated.
  + Every information, modification or changes would be lost on that machine other than shared folder.
  + If we do ‘vagrant up’ after destroy, it would build from scratch.

***Provisioning and Provisioners***

* Base Vagrant boxes are bare and only contain SSH is installed. But we can install software on our base box via shell script, configuration management systems or good ole manual command-line entry installations.
* Vagrant allows us to do automatic provisioning, could be done while “vagrant up”
* Provisioners can be used such as: Shell scripts, Chef, Puppet, and several plugins too.

***Installing Apache Manually***

* By default in Ubuntu apache is started in startup.
* We do it on the Guest Ubuntu machine
* 🡺 sudo apt-get install apache2
* Now we have to make apache point to vagrant synched folder rather than to the generic /var/www location.
* 🡺 sudo rm -rf /var/www *[-r = recursive, -f = forcefully, rm = remove] (Remove files in that folder recursively)*
* 🡺 sudo ln -fs /vargrant /var/www *[ln = create link, -s = symbolic link, 1st param = destination, 2nd param = where it has to be linked]*
* We can create an index.html file on host machine and put it in synched folder.
* 🡺 echo “<strong>index.html</strong>” > index.html
* Now we can see that apache server is picking data from the synched folder.

***Installing Apache/Shell Script***

* We use a shell script to install and set apache on the new machine.
* We create 🡺 provision.sh file in host machine
  + #!/usr/bin/env bash [will specific which lang to use, in this case bash and execute the rest of the file]
  + echo “[Message]”
  + apt-get update >/dev/null 2>&1 [>/dev/null = Output(1 – no need to specify) into null ; 2>&1 – STD Error(2) redirected to STDOUT. We can say that ***“All output from this command should be shoved into a black hole”***]
  + apt-get install -y apache2 >/dev/null 2>&1[-y = is the confirmation provided to install]
  + rm -rf /var/www
  + ln -fs /vagrant /var/www
* Now we get back to host console and modify the vagrantfile to add provision details.
  + Config.vm.provision “shell”, path : “provision.sh”
* Now just do ‘vagrant up’ and we have machine ready with apache installed.

***Installing Apache/Chef***

* Vagrant has in built provision to work with chef.
* Can use Chef Solo or Chef Client
  + Chef Solo: This uses local cookbooks to provision our machine.
  + Chef Client: Connects to Chef Server to download the cookbooks.
* During this session we will use Chef Solo.
* We need to change the Vagrant file to set provision for using chef
  + Config.vm.provision “chef\_solo” do|chef|

Chef.add\_recipe “vagrant\_la”

End

* We need to take note that chef would look for this cookbook in default cookbook folder.
* Structure should be /cookbooks/vagrant\_la/recipes and there have ‘default.rb’ file. [create using mkdir and cd to go in.]
* ‘Default.rb’ has instruction set to install apache and point to shared vagrant folder as done above.Inside file we write following commands
  + Execute “apt-get update”
  + Package “apache2”
  + Execute “rm -rf /var/www”
  + Link “/var/www” do

to “/vagrant”

* + End
* Get back to main project folder and execute “vagrant up”.
* Now use the apache server on port 8080.

***Networking***

* *Private Networking*
  + Private address not access from public internet.
  + The virtual machines in same physical machine can communicate with each other over this private networks.
  + Private networks need to configure network adapters on each machine.
  + Vagrant comes with knowledge and automatic know how to configure on variety of guest OS.
  + DHCP can be used to assign IP address within these private networks automatically.
  + Vagrant.configure(“2”) do|config|

Config.vm.network “private\_network”, type: dhcp

End

* + We can also set static IP address

Config.vm.network “private\_network”, ip: “192.186.70.4”

* + We need to make sure that IP address should be private ip addresss with in the private ip address space, so no need of routing or access type later on.
  + If we want to set it manually that’s after system is running, then we need to disable auto-configuration.

Vagrant.configure(“2”) do|config|

Config.vm.network “private\_network”, ip: “192.186.70.4”

Auto\_config: false

end

* + We would do ssh later and then set the ip address.
* *Public Networking* 
  + DHCP

Vagrant.configure(“2”) do| config|

Config.vm.network “public\_network”

end

* + Static IP

Config.vm.network “public\_network”, ip: “192.186.70.17”

* + Default Interface

Config.vm.network “private\_network”, bridge: ‘en1:wifi(AirPort)’

The name specified in bridge has to match with the name configured on the network card. If not found during ‘vagrant up’ process, then vagrant will ask us to pick from a list of connections.

# Day 2:

***Running Multiple Machines***

* With one single vagrant file we set up as many as vagrant machines.
* Previously we have everything flattened into one system and then in long run it would cost issue
* We make use of the capability of vagrant by setting up multiple machines have their specific applications.
* Basic Vagrant File for having multiple instances setup

Vagrant.configure(“2”) do | config|

Config.vm.box = “precise64”

Config.vm.box\_url = <http://files.vagrantup.com/precise64.box>

#Setup the webserver [Define first machine]

Config.vm.define “web” do| web|

Web.vm.hostname = “web”

Web.vm.box = “apache”

Web.vm.network “private\_network”, type: “dhcp”

Web.vm.network “forwarded\_port”, guest: 80, host:8080

Web.vm.provision “chef\_solo” do| chef|

Chef.add\_recipe “vagrant\_la”

End

End

#setup the db [Define second machine]

Config.vm.define “db” do| db|

Db.vm.hostname = “db”

Db.vm.box = “mysql”

Db.vm.network “private\_network”, type: “dhcp”

End

End

* Then do vagrant up
* After that use ‘vagrant ssh [specific\_hostname]’ to get into that.
* Check if all connections are made correctly.

***Vagrant Boxes***

* Till now we have been using Precise64 box which is Ubuntu 12.04 but we can create custom boxes.
* We can create boxes from existing vagrant environment or manually from non-vagrant environment.
* Custom Boxes
  + Create your linux distro of choice
  + Pre bake your applications on these boxes.
* Why Vagrant boxes
  + Optimization
  + Our system need not to install OS every time we do Vagrant up but use the snapshot base image. Base images doesn’t change that often.
  + Gives us disposable infrastructures
  + We get portability with our boxes, as developers like to work on their choice of workstation.
* Box Format
  + $ tree
    - Vagrant file
    - Box-disk1.vmdk [Compressed hard drive]
    - Box.ovf [description of the virtual hardware running in that machine]
    - Metadata.json [Tells what system and what box work with it]
  + Vmdk and ovf files are result of exporting a virtual box machine.

***Basic Box Management***

* Managed globally per user not per project.
* Vagrant puts all global content in “/vagrant.d” folder including these boxes.
* We can move this global directory by setting a variable “vagrant\_home” to another directory.
* Boxes are mapped in vagrant to a logical name in which you name, This name mappings maps to our config.vm.box settings in our vagrantfile to the actual box we are building out machine from.
* Boxes are just files.
* Commands to access boxes
  + $ vagrant box

Usage: vagrant box <command> [<args>]

* + Available subcommands
    - Add
      * Vagrant box add precise64 <http://files.vagrantup.com/precise64.box>
      * We can add the box to any project once we had downloaded that.
    - List
      * Vagrant box list
    - Remove
      * Vagrant box remove precise64 [virtualbox(provider-optional)]

[remove box ‘precise64’ with provider ‘virtualbox’]

* + - * This will free up some disk space.
      * We can take it off once we get it cloned and system is up and running.
    - Repackage
      * If we remove box, then we can repackage it and share that box to others.

***Creating Boxes from an Existing Environment***

* Easiest way to create would be to use existing Vagrant environment as starting point.
* First get into the existing machine using ssh and install pre baking software.
* Once we are in the system, then install tool midnight commander.
* Exit from virtual machine, get back to host machine.
* Use ‘Vagrant package [web(hostname)]’
* Then add the box using ‘vagrant box add [web(hostname)] [package.box (box name)]’
* Use ‘Vagrant box list’ to check all boxes present.
* Package will take the existing environment and creates a reusable box.
* If we modify any thing and want to add those changes into the system then we create other box

***Creating own boxes***

* Create a brand new Virtual machine inside Virtual box. (In GUI application)
* Use dynamically sized drive and set appropriate levels on RAM. (Can modify RAM size later, use minimal amount necessary and can be increased as needed, as starting boing we can set 350-512 MB, be careful on auxillary features [disable audio, USB, and other non-instantiable])
* Vagrant requires that the first device to be absolutely a NAT device, use this device to set port forwards for necessary SSH. (this has to be configured properly so that we can have perfect setup)
* Once above steps are done, then install OS from scratch, can configure it so that vagrant can use it later on.
  + Vagrant user must exist; this is the default SSH user. We can use other SSH user by specifying that in config.ssh.username variable.
  + SSH server must be installed and configured to run on system boot
  + SSH user must be configured to authenticate using a public key authentication, as ssh doesn’t support password.
  + If we want to create a private box then set config.ssh.private\_key\_path to point to our own private key if we wish.
  + Mkdir /home/vagrant/.ssh

Chmod 700 /home/vagrant/.ssh

Cd /home/vagrant/.ssh

Wget –no-check-certificate ‘https://raw.github.com/mitchellh/vagrant/master/keys/vagrant.pub’ -O authorized\_keys

Chmod 600 /home/vagrant/.ssh/authorized\_keys

Chown -R vagrant /home/vagrant/.ssh

* + User should have sudo access.
  + In ssh config be sure to also disable requiretty.
* At this point we need to install virtual box guest additions, that are also the additional kernel drivers and configurations so that virtual machine can take advantage of the virtual box features such as shared folders and improve network performance.
* Steps to install guest additions will differ based on OS.
* For linux OS will need kernel headers and build tools installed
* Additional software can be installed on the box and pre setup our configuration management tools.
* Avoid the well known linux virtual machine network device issues with
  + Rm /etc/udev/rules.d/70-persistent-net.rules
  + Mkdir /etc/udev/rules.d/70-persistent-net.rules
  + Rm -rf /dev/.udev/
  + Rm /lib/udev/rules.d/75-persistent-net-generator.rules
* Be sure to minimize your final box size
  + Uninstall unnecessary packages
  + Remove build tools and linux-headers
* Fill the hard drive with zeros and delete the zero-filled file, this fixes fragmentation issues with underlying disk on linux virtual machines. This would compress much more effectively later on.
* Packing it all up
  + Vagrant package –base(flag argument) la\_new\_box<name of the VM which we created in virtual box>
* We will now have package.box
* We can set the custom ssh user in vagrant file
  + Vagrant:: config.run do | config|

Config.ssh.username = “linuxacademy”

End

* If want to add the changes then use command
  + Vagrant package –base my\_new\_box –vagrantfile Vagrantfile

Exercise

1. Install Virtual Box - ***COMPLETED***
2. Install Ubuntu 13.10 – - ***COMPLETED***
   1. Vagrant box add Ubuntu <http://opscode-vm-bento.s3.amazonaws.com/vagrant/virtualbox/opscode_ubuntu-13.10_chef-provisionerless.box>
   2. Vagrant init Ubuntu
   3. Vagrant up
   4. Vagrant ssh
3. Install CentOS (latest edition) - ***COMPLETED***
   1. Vagrant box add CentOS https://github.com/CommanderK5/packer-centos-template/releases/download/0.7.2/vagrant-centos-7.2.box

[https://www.dropbox.com/s/7u194d6reyr1loe/vagrant-centos-7.2.box]

* 1. Vagrant init centos
  2. Vagrant up
  3. Vagrant ssh