SECTION 02

Vagrant for instant provisioning

WHAT IS VIRTUALIZATION?

- Virtualization is not a new technology. It is as old as the early 1970s, when the first IBM solution for "Time Sharing" was introduced.
- Virtualization simply refers to abstracting the hardware components of a machine so that it can operate multiple "virtual machines" sharing its own resources.
- For example, the following diagram represents a machine (called the host) running a virtualization software called VirtualBox. It is able to spawn other machines (called guests) all running together.

ORACLE
VirtualBox
ubuntu
redhat
SUSE
We adapt. You succeed.

WHAT IS VAGRANT?

- Vagrant is one of the DevOps tools that helps automate infrastructure provisioning. It works on top of one of the virtualization platforms like VirtualBox or VMWare.
- To understand what Vagrant does, consider the following example. You want to deploy the LAMP (Linux Apache MySQL PHP) stack on a virtual machine running Ubuntu. The typical workflow for such a task would be as follows:
  - Download the ISO file for Ubuntu
  - Configure a new virtual machine on VirtualBox, setup the disk and mount the ISO file as a virtual CD-ROM.
  - Start the installation by following the wizard.
  - Perform the post installation tasks like creating the necessary users and groups.
  - Download and configure Apache.
  - Download and deploy PHP and also configure the necessary Apache modules.
  - Download and install MySQL and perform the necessary security procedures.

HOW CAN VAGRANT HELP?

- The previous procedure will take no less than three hours at best to complete. A clever administrator may write automation scripts that will speed things up.
- You'll take even more time if you want to create a clone environment.
- Vagrant fills in the space. It provides an already-built virtual machine. You don't have to install or configure the operating system and that is already done for you.
- Vagrant uses boxes to provide the image for you. A box is like a template containing everything needed to spawn an image.
- Using a box, you can create as many machines as you need, all with the same configuration.
- This means multiple LAMP stack machines created within seconds instead of hours of even days.

LAB: VAGRANT INSTALLATION

- As mentioned before, Vagrant cannot be installed except if a virtualization platform is already present on target system. Vagrant supports VirtualBox, VMware and other well-known products.
- In this LAB we are going to demonstrate installing VirtualBox, then Vagrant on an Ubuntu machine.
- Please note that we are using Ubuntu for demonstration purposes. VirtualBox and Vagrant can be installed on Windows, Linux, MAC, and Solaris.
- The rest of this class will Vagrant installed on a MAC machine. However, the installation procedures are exactly the same on Windows and on Linux.
- ** Note for Windows users: you may need to install Cygwin (www.cygwin.com), which is a free software that installs several Linux tools on Windows including the bash shell and the SSH tools. This might be necessary to be able to access the Vagrant machine using SSH.

LAB: BUILDING YOUR FIRST VAGRANT MACHINE

- Vagrant uses the command line and a text file called Vagrantfile to operate.
- It works on directory-based containers. This means that each machine will have its own directory where the Vagrant command will apply only to the machine(s) in that directory. More on this later.
- Create a directory in your home path (if you are on Windows you can create it in your Documents folder). Call it ubuntu-16.10 and type `cd ubuntu-16.10`
- The first command you'd use is `vagrant init`. This is used to inform Vagrant about the box you want to use: `vagrant init bento/ubuntu-16.10`
- If you examine the contents of the current directory you will find that you have a new file called Vagrantfile. This is a Ruby file that contains several instructions for the machine to work.
- Now issue `vagrant up`. The box gets downloaded from Vagrant repository. This is done only if the box was not already downloaded before.
- After the download is complete, the machine boots up. In a few seconds it is up and running. Issue `vagrant ssh` command to log in to the machine.

LAB: ACCESSING DATA FROM THE HOST

- A very common scenario when using virtual machines is the need to access external data from the host machine.
- If you are using VirtualBox you must configure the shared folder settings, then create a special user for that on the guest OS.
- Using Vagrant, the shared folder feature is built-in. The directory where the machine is started is automatically shared with the guest machine in a filesystem called `/vagrant`.
- You can add more shared folders by adding `config.vm.synced_folder "/path/to/share", "/share/on/guest"`.
- If you want to disable the shared folders altogether you can add `disabled: true` after the default share.
- Remember to to run `vagrant reload` after running any change to the Vagrantfile.

LAB: NETWORK COMMUNICATION

- You'll definitely need to access network services running on the guest machine, for example, a web server.
- Vagrant supports three modes of network access to the guest machine. Choose the one the best suits your needs:
  - Port forwarding: this means that you'll access the guest network services through the host's IP address. Traffic arriving at the host's IP address and port will be forwarded to the guest network. For example, you can run a web server on the guest machine on port 80, and access it through the host's IP on port 8080.
  - Private network: the guest will share an internal network with the host. It will have its own IP address but only the host and other guests running on the same host will access it.
  - Public network: the gust will be fully accessible from the internal network as well externally. It will bear an IP address that can be reached the same way the host is reached on the network.
- The above settings can be configured in the Vagrantfile.

STOPPING THE GUEST MACHINE

- Once you need to shut down the machine for whatever reason, you have the following options:
    - Suspend: the state of the machine is reserved, RAM is saved to disk. When you resume the machine, all services continue to run automatically. This can be done using `vagrant suspend` command. To resume working, issue `vagrant resume`.
    - Halt: this attempts a graceful machine shutdown. If the machine did not respond to the shutdown attempt, it will be forced to "power off". You shutdown a vagrant machine by running `vagrant halt`. If you want to force the machine to shutdown manually you run `vagrant halt --force`.
    - Destroy: use this subcommand only if you want to reset the machine to the way it was right after it was created. This will remove any changes and delete any files that were added to the machine during its lifecycle. However, any files placed in /vagrant (the shared folder) will not be affected. To issue this command type `vagrant destroy`.

USING VAGRANT FOR INSTANT PROVISIONING

- Vagrant machines comes in two flavors: either a clean OS install, that is, no other software or applications where deployed just the OS, or a prebuilt environment, where applications and services have been deployed to the machine before being packaged as a box.
- Both of them can be used for instant provisioning of your required environment. For example, if you want to deploy the LAMP stack you have the following options:
  - Search https://atlas.hashicorp.com/search (or any other source containing Vagrant boxes) for a box that serves the LAMP stack. Once you deploy the box you can create as many machines all having LAMP deployed.
  - Download a box containing just the OS of your choice and use one of the provisioning options of Vagrant to deploy LAMP.

VAGRANT PROVISIONING OPTIONS

- Vagrant supports many ways to provision your machine. For example:
  - Shell scripting
  - Ansible playbooks
  - Chef recipes
  - Puppet
  - Salt
  - Docker
- Choosing your provisioning method depends greatly on the type of environment you want to create. If it is a simple one like deploying a web server, you may opt for a shell script. However, if you want to install a complete web application that uses multiple languages and an Nginx reverse proxy on top of it, you may be better off using one of the more complex configuration management software mentioned above.
- Actually Vagrant is flexible enough to let you use more than one provider at the same time (with an order of execution).

LAB: USING SCRIPT PROVISIONING TO PROVISION A LAMP STACK

- In this lab, we are going to write a script that will perform the following tasks:
  - Install Apache web server
  - Install the necessary modules for PHP to work
  - Install PHP
  - Install MySQL server
  - Configure MySQL root credentials using a SQL file
- Configure the Vagrantfile to forward port 8080 on the host to port 80 on the guest. Notice that you cannot use any ports that are less than 1024 on the host as they are privileged.
- At the end of the Vagrantfile, add the following:

  ```
  config.vm.provision "shell", path: "deployLamp.sh", privileged: false
  ```

MULTI-MACHINE DEPLOYMENT

- In real-world scenarios, environments are dispersed among different nodes. For example, the LAMP stack could have the application server on one node (running Apache and PHP), and the database server on another one.
- One option is to create a different directory for each machine. However, Vagrant offers the capability of working with more than one machine in the same directory.
- Having one directory to host multiple machines, all serving the same environment makes that environment portable; as they'll be using the same Vagrantfile.
- You can work with each machine by its name. For example, `vagrant up web`, `vagrant reload db`…etc.
- In the following lab, we are going to split the LAMP stack to a database and a web server.

LAB: SPLITTING THE LAMP STACK TO A WEB AND AN APPLICATION SERVER

- Create a new directory called lamp. Copy the vagrant file from the Ubuntu machine and place it in that directory.
- Change the Vagrantfile to accommodate more than one machine by adding the following to the end:

```
config.vm.define "app" do |app|
app.vm.provision "shell", path: "deployWeb.sh", privileged: "false"
end
config.vm.define "db" do |db|
db.vm.provision "shell", path: "deployDb.sh", privileged: "false"
end
```

- Now we have two machines, app and db. Whenever you want to work with one, you must follow the command with the machine name. For example, `vagrant up app`.
- Make sure that the machines are configured the way it was intended using the shell script.