**Introduction**

* General Concepts
  + Virtual Machines
    - host
    - guest
  + Networking
    - NAT
      * Port Forwarding
    - Private
* What is Vagrant?
  + Manages virtual machine software
    - create/destroy VMs
    - start/stop/restart VMs
    - Access to VMs
    - Networking / VM settings
  + Orchestrates "provisioning"
    - on demand setup
  + sharded folders
  + Graphical user interface, diagram, application

    Description automatically generatedplugins to extend functionality
* Two Architectures
  + Kitchen Sink (All in one)
    - host runs VMs only
    - IDE/Editors in VM
    - GUI in VM
    - All work in VM
    - deploy in VM

Diagram

Description automatically generated

* + Light Server
    - Host has IDE/Editor
    - Host runs VM
    - shared folders
    - VM for local deploy
    - no GUI for VM
* Provisioning
  + build upon base box
  + install / update
    - software
    - OS
  + Configuration
  + re-execution
  + multiple provisioners
  + Provisioners
    - file
    - shell
    - chef, puppet, docker, etc.
* Vagrant Configuration
  + Vagrantfile
  + included in project & versioned
  + ruby programming language
    - prior experience not required
    - execution is top-down, single-pass
  + flexible
  + cascades / overrides
* By default, the vagrant shared folder on the VM is located at /vagrant and the directory contain the Vagrantfile on the host machine (**/vagrantVM** 🡨🡪 **/directoryContainingVagrantfileHOST**)

**Getting Started**

* **vagrant init hashicorp/percise64**
* **vagrant up**
  + launch VM
* **vagrant ssh**
  + ssh into box
* **vagrant suspend**
* **vagrant status**
* **vagrant halt**
  + takes longer than using *suspend*
* **vagrant destroy**
  + completely removes VM

**Managing Boxes**

* To list boxes
  + **vagrant box list**
* to update boxes
  + **vagrant box update**

**Vagrant Plugins**

* install a plugin
  + **vagrant plugin install <plugin name>**
* uninstall plugin
  + **vagrant plugin uninstall <plugin name>**
* list plugins
  + **vagrant plugin list**

**Provisioning**

* Provisioning Approach
  + Big "Baked" Base Box
    - longer to setup
    - harder to update
    - faster "provisioning" step
      * first "vagrant up"
    - not in vagrant spirit
  + Minimal Base Box
    - widely available (no creation)
    - fast creation
    - much slower provisioning cycle
      * not an issue after first "vagrant up"
  + Core Base Box (Hybrid of the two boxes above)
    - requires some setup
    - OS package manager
    - Balanced:
      * common components are "pre" provisioned
      * project dependencies on demand
      * provisioning time is modest
* **Manual "Baked" Provisioning**
  + step necessary to build up a custom server for PHP development (LAMP stack)
    - Linux, apache, MySQL, PHP
  + **vagrant init centos/7**
    - comment out line in *Vagrantfile* to allow port forwarding (for this project it is Line 26)
  + **vagrant up**
  + **vagrant ssh**
  + **sudo yum update -y**
    - make sure that the box is up to date
  + **sudo yum install -y vim git unzip screen**
    - download some useful tools to use later on
  + **sudo yum install -y httpd httpd-devel httpd-tools**
    - install Apache
  + **sudo systemctl start httpd.service**
    - **sudo systemctl --state=active | grep "Apache"**
      * this command will check to see if the Apache service is running
    - add Apache to system initialization scripts
  + by default, Apache, will setup its web documents to be served from "/var/www/html". So, we need to move this "html" directory
    - **cd /var/www/**
    - **sudo rm -rf html**
  + we need to create a symbolic link from out "**/vagrant**" folder (which is our shared folder with our host), to the "**html**" folder that we just deleted
    - **sudo ln -s /vagrant /var/www/html**
  + Now we need to exit our VM and go to our host machine and create a file to serve on our Apache server and call it "index.html"

**<html>**

**<p>Hello from the host/local machine</p>**

**</html>**

* + now ssh back into the VM and start Apache if it is not running
    - check: **sudo systemctl restart httpd**
    - if you get "**403 Forbidden You don't have permission to access / on this server**" we need to disable *SELINUX* and restart our VM
      * **sudo vi /etc/selinux/config (sudo sed -i -e 's/SELINUX=enforcing/SELINUX=disabled/' /etc/selinux/config)**
      * **SELINUX=disabled** (CHANGE SELINUX to disabled)
      * **vagrant reload** (Reboot the vagrant box on the local machine)
    - NOW THE APACHE SERVER SHOULD DISPLAY THE CONTENTS OF OUR file 🡪 index.html
  + log back into VM
    - vagrant ssh
  + now we need to install PHP
    - **sudo yum install -y php php-cli php-common php-devel php-mysql**
  + now restart Apache so it will include the PHP module
    - **sudo systemctl restart httpd.service**
  + no go back to your host/local machine and create a simple .php file called "info.php"

<?php

phpinfo();

?>

* + now reload the vagrant VM by running **vagrant reload**
  + next ssh back into the vagrant box (**vagrant ssh**) and restart the Apache server (**sudo systemctl restart httpd**). Open a web browser and go to 🡪 [*http://localhost:8080/info.php*](http://localhost:8080/info.php)
    - if the page loads, then we know that PHP has successfully been installed and is working with Apache
  + now let's install MySQL. ssh back into the vagrant box (**vagrant ssh**)
    - **sudo yum install -y mysql mysql-server mysql-devel**
  + now let's make sure MySQL is installed and begins to run whenever are VM is turned on
    - **sudo systemctl status mysql**
    - if you got an error "Unit mysql.service could not be found" MySQL did not install. We need to do the following steps in order to successfully install MySQL. Go to <https://dev.mysql.com/downloads/repo/yum/> and select the version of MySQL that you would like to download
      * **wget https://dev.mysql.com/get/mysql57-community-release-el7-11.noarch.rpm**
      * **sudo rpm -ivh mysql57-community-release-el7-11.noarch.rpm**
      * **sudo yum install -y mysql mysql-server mysql-devel**
      * **sudo vi /etc/my.cnf**
        + we need to append"**validate\_password\_policy=LOW**"to the end of the file
      * **sudo grep "password" /var/log/mysqld.log**
        + you will get an output similar to which contains a temporary password that is highlighted

2021-06-09T17:44:12.635082Z 1 [Note] A temporary password is generated for root@localhost: N4/-4!5!pnu:

* + - * **sudo mysql\_secure\_installation** 
        + copy the temporary password and follow instructions to create a new password (I used *password*)
      * **sudo mysql -u root -p**
        + **mysql> show databases;**
      * we have now successfully installed MySQL onto our vagrant box
    - now we can check to see if MySQL is installed by running **sudo systemctl status mysqld.service**
  + now that MySQL is installed run **sudo systemctl enable mysqld.service** and start MySQL by running **sudo systemctl start mysqld.service**
  + Now that our LAMP stack is installed (Linux, apache, MySQL, PHP), if we ran a vagrant destroy, we would lose all the changes that we just made, and we would be at a base installation the next time we would run vagrant up
    - To solve this, we are going to use the vagrant package command. In order to do this, you must ensure that the box is currently running
    - **vagrant package --output centos-lamp.box**
    - Now we need to add the box file to vagrant
      * **vagrant box add centos-lamp centos-lamp.box**
  + create a new directory to test our LAMP stack (**mkdir test-lamp**)
  + uncomment the line in the Vagrantfile to allow port forwarding
  + we need to install a plugin to make sure our guest additions are installed properly
    - **vagrant plugin install vagrant-vbguest**
  + **vagrant up**
  + now we can make an index.html file and visit *http://localhost:8080/* and see our index.html is being displayed. We didn’t even have to ssh into our vagrant box

**File Provisioner**

* create global configurations
  + **git config --global user.name "Nick Brell"**
  + **git config --global user.email "nickbrell@yahoo.com"**
    - **git config --global -l**
  + copy ".gitconfig" into current directory (/Users/nicholausbrell/vagrant/files)
    - **cp ~/.gitconfig git-config**
  + make a new directory (I am using /git-box)
  + **vagrant init centos/7**
  + we need to disable a plugin for automatic updating of guest additions inside the Vagrantfile
    - open up the Vagrantfile and add the following line **config.vbguest.auto\_update = false**
  + **vagrant up**
  + **vagrant ssh**
  + now we need to modify the Vagrantfile to use the file provisioner to copy over the ".gitconfig" file to our user's home directory. Add the following snippet inside of the Vagrant.config block.

**config.vm.provision "file",**

**source: "~/vagrant/files/git-config",**

**destination: "~/.gitconfig"**

* + Next, in order to have Vagrant run any provisioning configuration, we need to run the **vagrant provision** command
  + now if you ssh into your box (vagrant ssh) and list files in the directory, you can see the directory no contains our ".gitignore" file

**Shell Provisioner**

* We are going to edit our Vagrantfile so that when we execute the **vagrant up** command, we will have a shell script run that we define in our Vagrantfile
  + open up the Vagrantfile
  + inside the .config block add the following

**config.vm.provision "shell",**

**inline: "yum install -y git"**

* + - this will install "Git" onto our Vagrant box when we decide to run the vagrant up command
* **Example**: Let's say that we want to install *Git* & *Nano* automatically onto our new vagrant boxes whenever we run the vagrant up command
  + navigate to your home directory, then to your vagrant directory
  + if you **ls** you should see the "files" directory we created in the File Provisioner example. Create a new directory named "scripts" then navigate into that folder (~/vagrant/scripts)
  + **vi provisioner.sh**
    - add the following to the file

**#!/bin/bash**

**# Install Git & Nano**

**yum install -y git**

**yum install -y nano**

* + now navigate back into your vagrant box directory (mine is located my /git-box directory) and add the following to you Vagrantfile inside of the Vagrant.configure block

**#shell provisioner**

**config.vm.provision "shell",**

**path: "~/vagrant/scripts/provision.sh"**

* + **vagrant up**
  + **vagrant ssh**
    - if we run which git and which nano, we can see that they are both installed!

**Custom Boxes**

* Disadvantages
  + Duplicate Work (entire process might be redundant)
    - check online to see if a custom box meets your needs
    - customize with vagrant package
  + Modestly difficult
    - permissions
    - VB guest additions
  + Time consuming / Debugging
* Advantages
  + new / updated / older operating systems
  + corporate requirements
    - comply with security or compliance regulations
  + custom configurations
  + variations
    - GUI, Server, minimalist
* What are we building?
  + CentOS 7.0
  + minimal / server
    - no GUI
    - networking
    - OpenSSH server
    - smaller box size
  + current patches
* Acquire OS Image (<https://www.centos.org/>)
* Launch VirtualBox > New
  + name: vagrant-custom-centos7
  + machine folder: leave as default or change
  + type: Linux
  + version: red hat (64-bit)
  + memory size: 512
  + hard disk: create a virtual hard disk now
  + hard disk file type: VMDK (virtual machine disk)
  + storage on physical hard disk: dynamically allocated
  + file location and size: 20GB
* Settings
  + audio: disable audio
  + ports > USB: disable USB controller
  + network > adapter 1: enable network adapter (attached to: NAT)
    - advanced > port forwarding > add rule: host port = 2222, guest port = 22, Name = Vagrant SSH
  + Start > select ISO we downloaded from centos.org and click start

**Using the Chef Solo Provisioner**

* What is Chef?
  + Chef is a provisioning and deployment automation tool
  + Ruby programming language
  + Chef Domain Specific Language (DSL)
  + Infrastructure as Code (anything you do in Chef can be version controlled)
* Chef Terminology
  + recipe:
    - Ruby + Chef DSL scripts
    - task actions
    - can use supporting scripts
    - can call other recipes
    - entry point of a cookbook
    - default recipe
  + template:
    - embedded Ruby templates
    - uses ".erb" file extension
    - can contain Ruby and dynamic values
    - excellent for configuration files
  + attribute:
    - settings used in cookbooks
    - overriding / precedence
    - customize cookbooks
  + cookbook:
    - collection of recipes, templates, and supporting files / scripts
    - metadata (name, description, version dependencies)
  + role:
    - group related recipes
    - outside, unrelated to cookbooks
    - not versioned in Chef
  + node:
    - system where recipes are executing on
    - chef solo only has 1 node
  + go to your vagrant directory and create a new directory names "chef"
  + in the chef directory (**/vagrant/chef**) create 3 new directories "supermarket", "cookbooks" and "roles"
  + navigate to your supermarket directory (**/vagrant/chef/supermarket**)
  + go to the chef supermarket and search for a "tomcat" recipe and download the cookbook (make sure that it is compatible with centOS)
  + go to the "dependencies" tab and download the dependencies cookbook
  + **tar -xvzf ~/Download/tomcat.tgz**
  + install vagrant Chef plugin
    - **vagrant install plugin vagrant-omnibus**
  + **vagrant init bento/centos-6.8**
  + **git add .**
  + **git commit -m "initial commit"**
  + s