Development Environments

with Vagrant and Ansible

Jason DeShazer

Introduction

Jason DeShazer

Software Engineer (Oregon Shakespeare Festival)

Independent Consultant and Web Developer

Bachelors of Science in Computer Science and Mathematics (Southern Oregon University)

Masters in Business and Administration (in progress)

Outside of working with computers

- I practice Aikido
- I play the clarinet

Overview

- Look at a few client server architectures
  - Strategies for reproducing them
  - Look at the benefits of each
  - Look at the downsides of each
- Introduce you to Vagrant
- Introduce you to Ansible
- Build a LAMP stack (interactive)
- Challenges (self study)

Ground Rules

- This presentation will be relatively short
  - These tools are very easy to learn how to use
  - I want to leave plenty of time for you to play with these tools
  - I want you to be engaged and ask questions
- If you have questions, please interrupt me
- Here’s a link to this presentation if you’d like to follow along
  - http://tinyurl.com/jaxsvap
- Before we begin, I’m going to go over some terminology

What are Servers

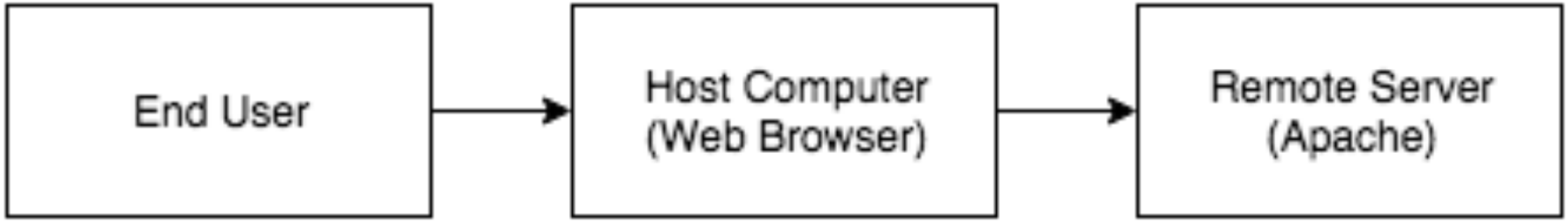
- Servers as software
  - Computer program that manages resources and services and provides access to its resources and service to other computers over a network
- Servers as hardware
  - The physical computer that software runs on
  - When most people say “server,” this is what they’re referring to
- For clarity, we’ll use the word Server to mean hardware stack and Server Software to mean software stack

What are Stacks

A Software Stack is the collection of software that makes up the operational infrastructure on a given server

LAMP = Linux + Apache + MySQL + PHP(Perl)

Typical Hardware Architecture



Downside to Physical Servers

- It costs money
  - Physical servers can be expensive to maintain
- It costs time
  - Physical servers can take longer to provision
- Flexibility is sometimes not an option
  - Imagine wanting to test a new configuration or web service, but not having the necessary physical resources to do so.
- It makes maintaining multiple environments difficult
  - Dev, Staging and Production servers all need access to their own physical resources
  - This is exacerbated by each environment requiring access to multiple Servers

Alternative to Physical Servers

Installing the server software locally

Because server software is just a computer program, it can be installed locally and run on the same environment your development workstation is on.

Virtualizing the remote server

Replace the need for physical servers with virtual ones

WampServer and XAMPP

WampServer

- Web development platform for windows
- Installs Apache, PHP and MySQL on your local windows computer
- Allows you to turn the server software on an off
- Just turn it on and tell it where to serve your PHP application from and WampServer does the rest

XAMPP

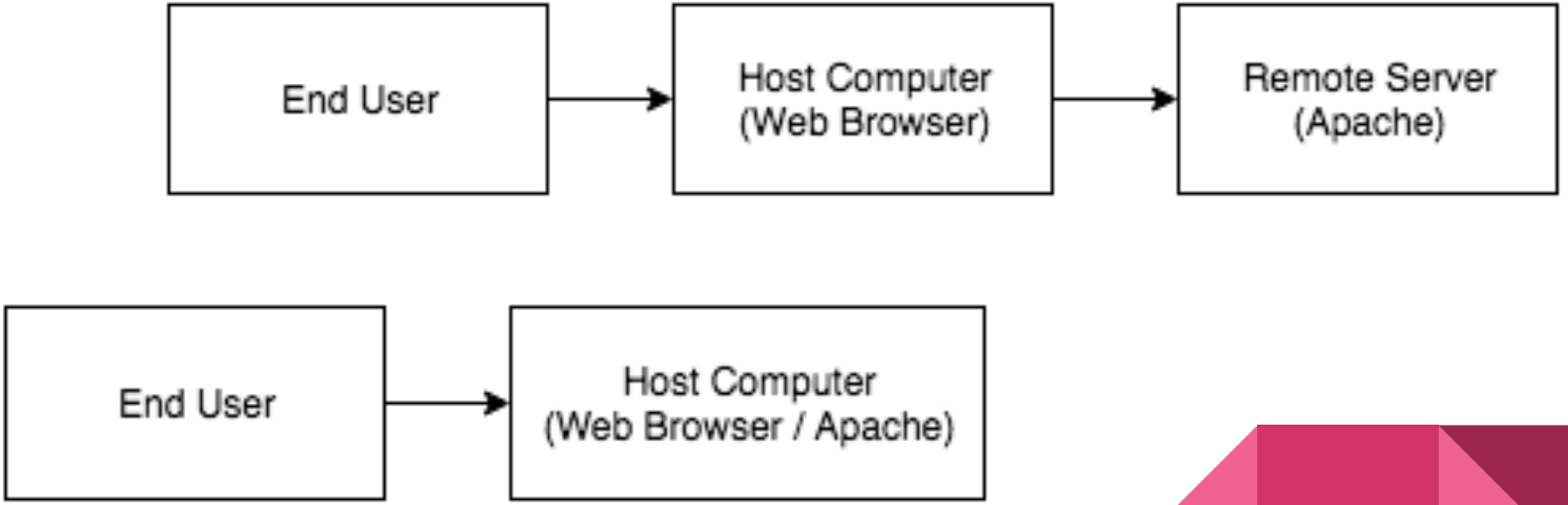
- The same as WampServer, but cross platform
- Capable of running on Windows, OSx or Linux
- Also installs Perl

Benefits of Using WampServer or XAMPP

- It saves resources
  - There’s no need for a physical server to host your application
  - Everything exists locally
- It saves money
  - There’s no overhead costs in maintaining a physical server
- It saves time
  - It’s fast and easy to install
- It allows your development to be more flexible
  - It makes managing multiple applications a simple configuration change
- It’s easy to setup and use
  - Just configure a web root directory and press go

WampServer or XAMMP Architecture

Compared Architectures



Downsides to WampServer or XAMPP

- Development environment doesn’t accurately depict production environment
  - Operating systems might not match
  - Versions of software might not match
  - You application might not function the same way in production because of this
- Makes reproducing development environments difficult
  - These tools only manage Apache, MySQL, PHP and Perl
  - On top of that, they only manage one version
  - Any additional dependencies aren’t managed by these tools
  - Multiple developers might need their own local environment
  - Without tracking dependencies, it can be hard to ensure that both developers have the same environment
- You might find that your code works on your machine but not someone else's

What is Virtualization

Virtualization uses computer software, called a hypervisor, to make a computer act like many computers

Each of these individual computers are called virtual machines (VM)

An example of a hypervisor is Oracle's Virtualbox

The computer running the virtualization software is called the Host

All VM's running under the host are called Guests

Guests use up a pre-allocated portion of the host's resources

- RAM, CPU and memory

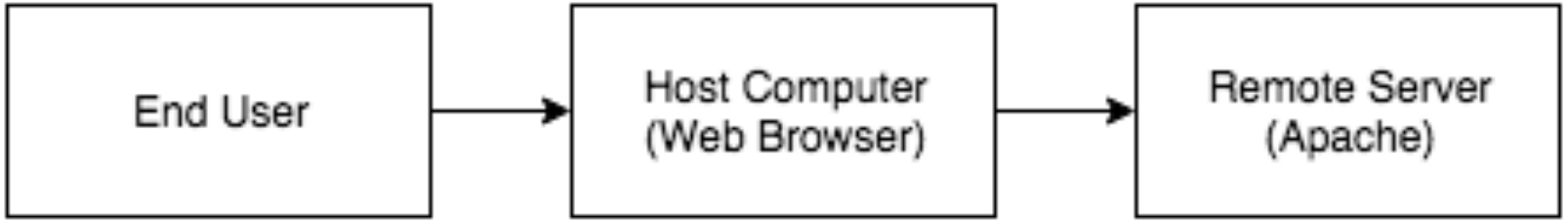
Guests are not limited to the same operating system as their host

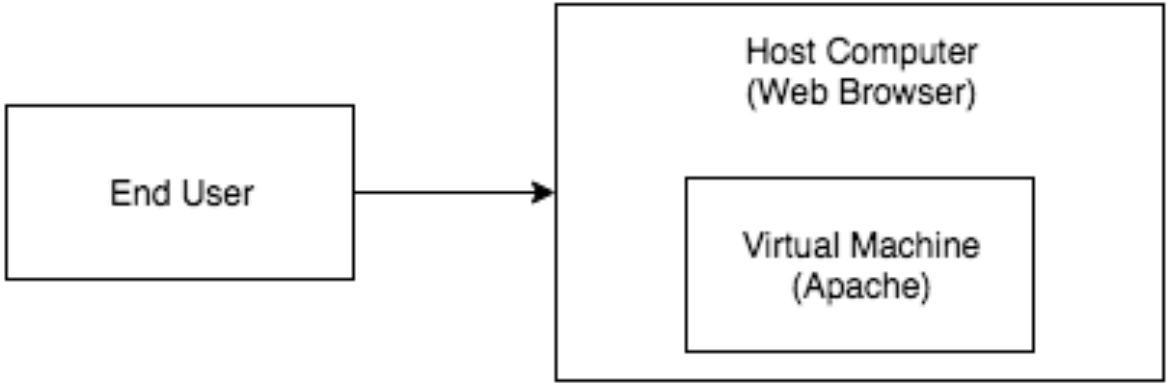
- A windows host can virtualize a guest with Linux installed

Benefits of Virtualization

- Virtual machines can replace the need for physical servers
  - There's no overhead costs to maintain VMs
  - They can be created much faster than physical servers
  - You can spin up as many as you need
- You can use open source hypervisors for development
  - You don't have to worry about the performance implications when it's just for development
- You can simulate any software or hardware stack, despite what operating system is on your host
- You have more control over what is installed to your VM
  - As opposed to WampServer or XAMMP
- You can better emulate your production environment

Compared Architectures





Downside to Virtualization

- Virtual machines can be complex to build
  - Compared to a simple WampServer
- Virtual machines are still difficult to reproduce
- The software stack has to be installed every time a virtual machine is provisioned

Luckily, we can mitigate all of this by using Vagrant and Ansible

Key Takeaways

Sometimes it’s not always sufficient enough to simulate the software stack

You need to simulate the hardware environment as well

You can do that with the use of virtual machines

Vagrant and Ansible make this easy

Vagrant

What is Vagrant

- Written by Mitchell Hashimoto
- Command line tool (written in Ruby)
- Automates creation of virtual machines
  - VirtualBox
  - VMWare
  - Hyper-V
- Integrates well with configuration management tools
  - Ansible
  - Puppet
  - Chef
  - Shell

Why use Vagrant

- Makes creating virtual machines easy
  - Just run one command
- Makes managing multiple VMs easier
- Makes reproducing VMs easier
- Flexible enough to simulate production environments
- VMs are very portable
  - Don’t take up very much space
- Configurations for VMs are written in plain text (Ruby syntax)
  - Easy to check into source control alongside the project your box was built for

How does it work

1. Scaffold a configuration file (Vagrantfile) for your VM
2. Adjust the VM's settings via it's Vagrantfile
3. Run vagrant up

That's it!

Share your Vagrantfile with another developer

All they have to do is run vagrant up

They now have an exact replica of your environment

Creating a configuration file

1. Open a terminal
2. Change directories to where you'd like your Vagrantfile to be generated
3. Run vagrant init

Example basic Vagrantfile

```ruby
# -*- mode: ruby -*-
# vi: set ft=ruby :

VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
    # name of the base box to install
    config.vm.box = "centos/7"
    # defines a single VM
    config.vm.define "lamp-stack" do |node|
        # hostname
        node.vm.hostname = "lamp-stack.example.com"
        # networking settings
        node.vm.network "forwarded_port", host: "8080", guest: "80" # webserver
        node.vm.network "forwarded_port", host: "443", guest: "443" # ssl
        node.vm.network "forwarded_port", host: "3306", guest: "3306" # database
        node.vm.network :private_network, ip: "10.0.0.10"
        # virtualbox synced directories
        node.vm.synced_folder "html", "/var/www/html", type: "virtualbox"
        node.vm.synced_folder ".", "/vagrant", type: "virtualbox"
        # virtualbox settigns
        node.vm.provider "virtualbox" do |vb|
            vb.gui = false
            vb.memory = "1024"
            vb.cpus = "1"
        end
        # ansible settings
        node.vm.provision "ansible_local" do |ansible|
            ansible.playbook = "ansible/playbook.yml"
            ansible.sudo = true
        end
    end
end
```

Example advanced Vagrantfile

```ruby
# -*- mode: ruby -*-
# vi: set ft=ruby :

VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|

  # name of the base box to install
  config.vm.box = "centos/7"

  # defines a VM for the webserver
  config.vm.define "webserver" do |webserver|
    webserver.vm.hostname = "webserver.example.com"
    webserver.vm.network "forwarded_port", host: "8080", guest: "80" # webserver
    webserver.vm.network "forwarded_port", host: "443", guest: "443" # ssl
    webserver.vm.network :private_network, ip: "10.0.0.10"
    webserver.vm.synced_folder "html", "/var/www/html", type: "virtualbox"
    webserver.vm.synced_folder ".", "/vagrant", type: "virtualbox"
  end

  # defines a VM for the database server
  config.vm.define "database" do |database|
    database.vm.hostname = "database.example.com"
    database.vm.network "forwarded_port", host: "3306", guest: "3306" # database
    database.vm.network :private_network, ip: "10.0.0.11"
    database.vm.synced_folder "html", "/var/www/html", type: "virtualbox"
    database.vm.synced_folder ".", "/vagrant", type: "virtualbox"
  end

end
```

Vagrant = Ruby

```ruby
# -*- mode: ruby -*-
# vi: set ft=ruby :

VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|

  # name of the base box to install
  config.vm.box = "centos/7"

  # defines a range of VMs
  (1..3).each do |i|
    config.vm.define "webserver#{i}" do |webserver|
      webserver.vm.hostname = "webserver.example.com"
      webserver.vm.network "forwarded_port", host: "8080", guest: "80" # webserver
      webserver.vm.network "forwarded_port", host: "443", guest: "443" # ssl
      webserver.vm.network :private_network, ip: "10.0.0.1#{i}"
      webserver.vm.synced_folder "html", "/var/www/html", type: "virtualbox"
      webserver.vm.synced_folder ".", "/vagrant", type: "virtualbox"
    end
  end

end
```

Basic Vagrant commands

vagrant up

First run creates the virtual machine and starts it

Subsequent runs start the virtual machine

Can make use of the --provision flag

vagrant halt

Stops the virtual machine

vagrant reload

Restarts the virtual machine

It’s the equivalent of running vagrant halt && vagrant up

vagrant ssh

Opens a secure shell connection to the virtual machine

vagrant destroy

Destroys the virtual machine

More Information at VagrantUp.com

Documentation for building Vagrantfiles

https://www.vagrantup.com/docs/getting-started/

Ansible

What is Ansible

Configuration management tool (written in python)

Helps orchestrate the state of servers

It’s idempotent

- Change only occurs if need be
- Focuses more on the state the machine should be in
- Not necessarily how to put it in that state

It’s agentless

Will integrate well with vagrant for provisioning our development environments

Ansible Terminology

Inventory - Groups of servers, defined by IP address or DNS and denoted by a unique group name

Playbook - Defines the state of the servers in each group from your inventory

Roles - Logical groupings of variables, tasks and handlers

Vars - variables

Tasks - An action with a name

Handlers - A special type of task that only occurs if notified that the state of another task changed

Modules - Tasks and Handlers are made up of these
This is Ansible's smallest building block

Basic Example of Ansible

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at boot)
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

More Information at Ansible.com

Documentation for building ansible playbooks

http://docs.ansible.com/

Let’s build a LAMP stack

Instructions

1. Open a terminal
2. Clone https://deshazerj@bitbucket.org/deshazerj/vagrant-ansible-presentation.git
3. Change directory to that repository
4. Run vagrant up

That’s it!

Challenges

Refactor our LAMP stack so that it spins up two virtual machines

- One with a database (database server)
- One with Apache and PHP installed (web server)

Pull down your own personal project and build a development environment for it

Use a different provisioner, like shell, to provision your virtual machine

- Hint: https://www.vagrantup.com/docs/provisioning/shell.html

Other