NYU COURANT INSTITUTE OF MATHEMATICAL SCIENCES

Infrastructure as Code: Vagrant + VirtualBox + Docker

Fall 2020, CSCI-GA 2820, Graduate Division, Computer Science

Instructor:

John J Rofrano

Senior Technical Staff Member | DevOps Champion
IBM T.J. Watson Research Center
rofrano@cs.nyu.edu (@JohnRofrano)

DevOps Principle of the Day:

“**AUTOMATE EVERYTHING!**”

Even the Development Team !!!

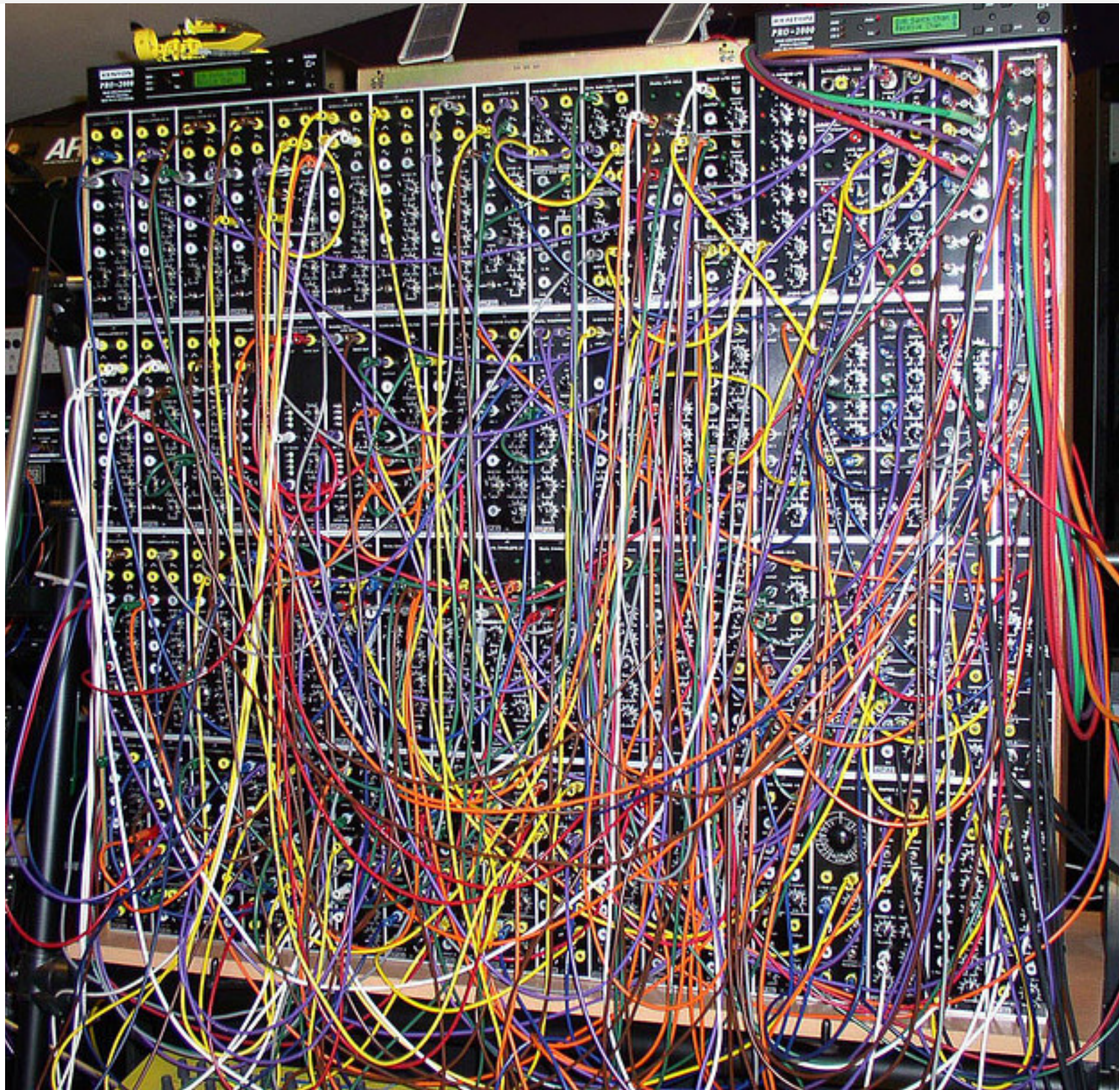
Objectives

The objectives of this class are to:

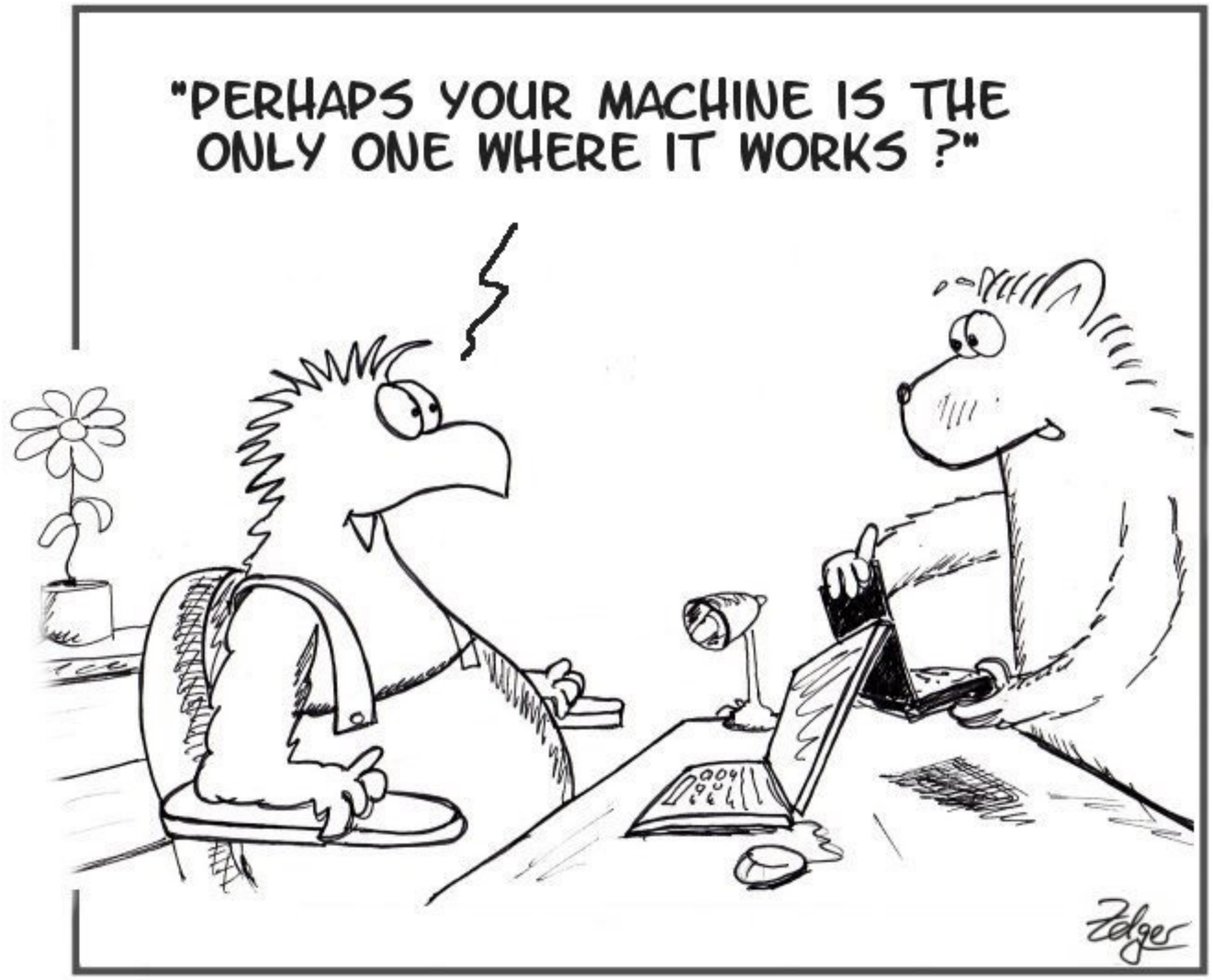
- How to create customized development environments with Vagrant and VirtualBox
- Create and destroyed them and created again in a consistent fashion
- How to leverage Docker containers for middleware deployment inside of your Virtual Machines
- How to make your DevOps team more productive with just a single command:
  > vagrant up

The Problem

- Manually creating local environments for developers to work in is:
  - Time consuming
  - Error prone
  - Inconsistent at best
  - Unreproducible at worst!



The Goal is to Avoid This



It works on my machine

The Wrong Solution

- Carefully and painstakingly create documentation in the form of “runbooks” that give step-by-step instructions on how to recreate the development environment via cut-n-paste
- NOT VERY AGILE !!!
  - We value: Working software over comprehensive documentation

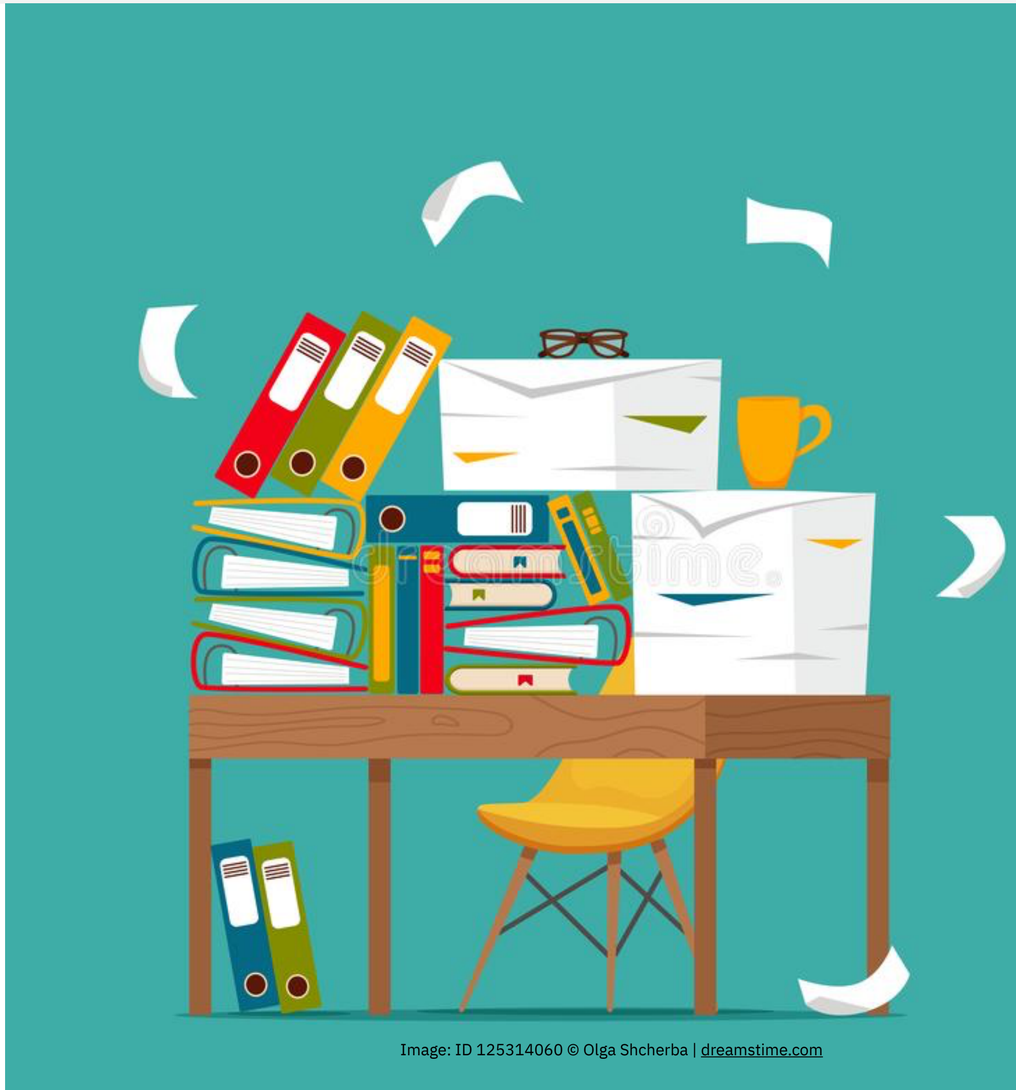


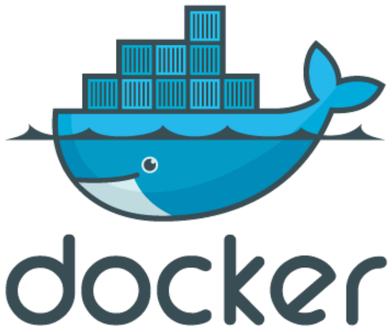
Image: ID 125314060 © Olga Shcherba | dreamstime.com

The Right Solution: Infrastructure as Code

Automate the creation of local development environments right on your laptop or desktop using Virtual Machines

Use **Vagrant** to quickly provision complex configurations consistently with repeatability

Use **Docker** to handle middleware without any installation



Use **VirtualBox** to create Virtual Machines for development

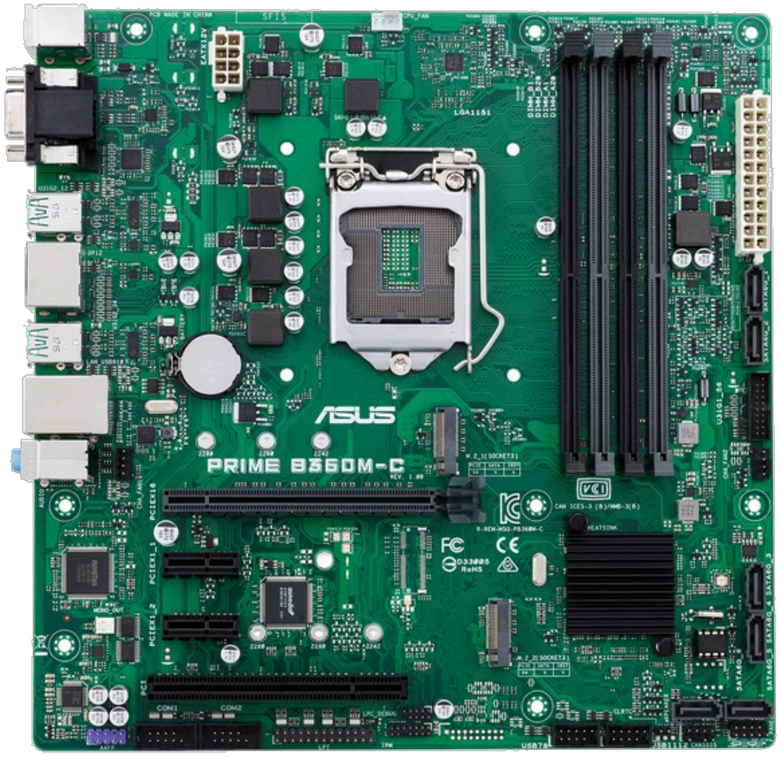
Before we can understand **Virtual Machines** we must first understand **Physical Machines**

Physical Machines

Before you can understand Virtual Machines you must understand Physical Machines

Physical Machines

Before you can understand Virtual Machines you must understand Physical Machines

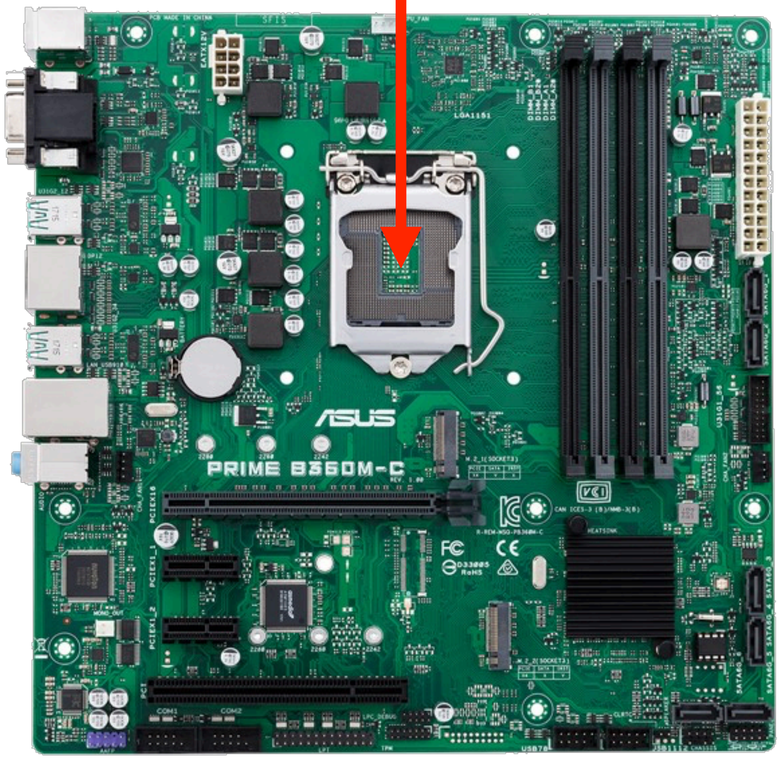


Computer Motherboard

Physical Machines

Before you can understand Virtual Machines you must understand Physical Machines

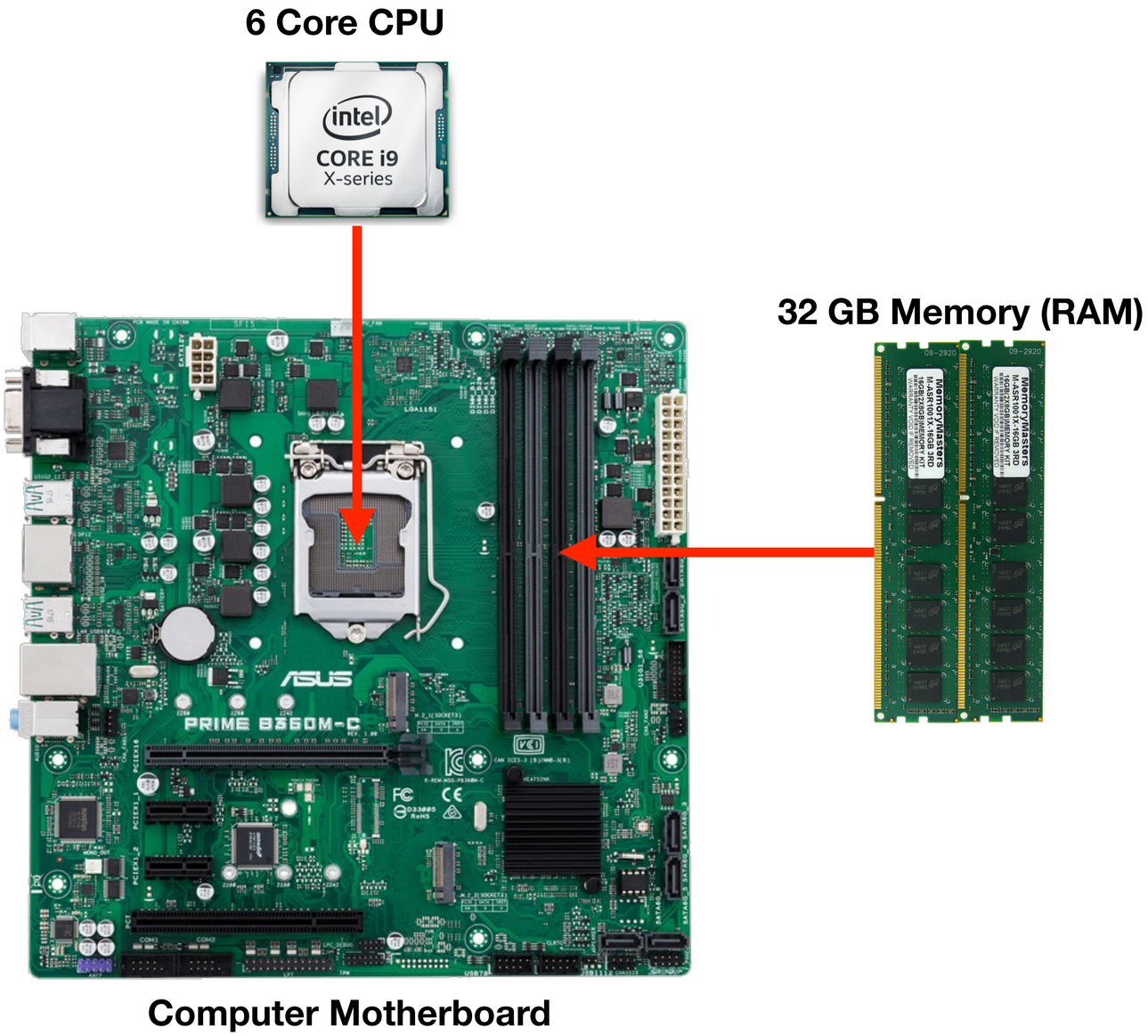
6 Core CPU



Computer Motherboard

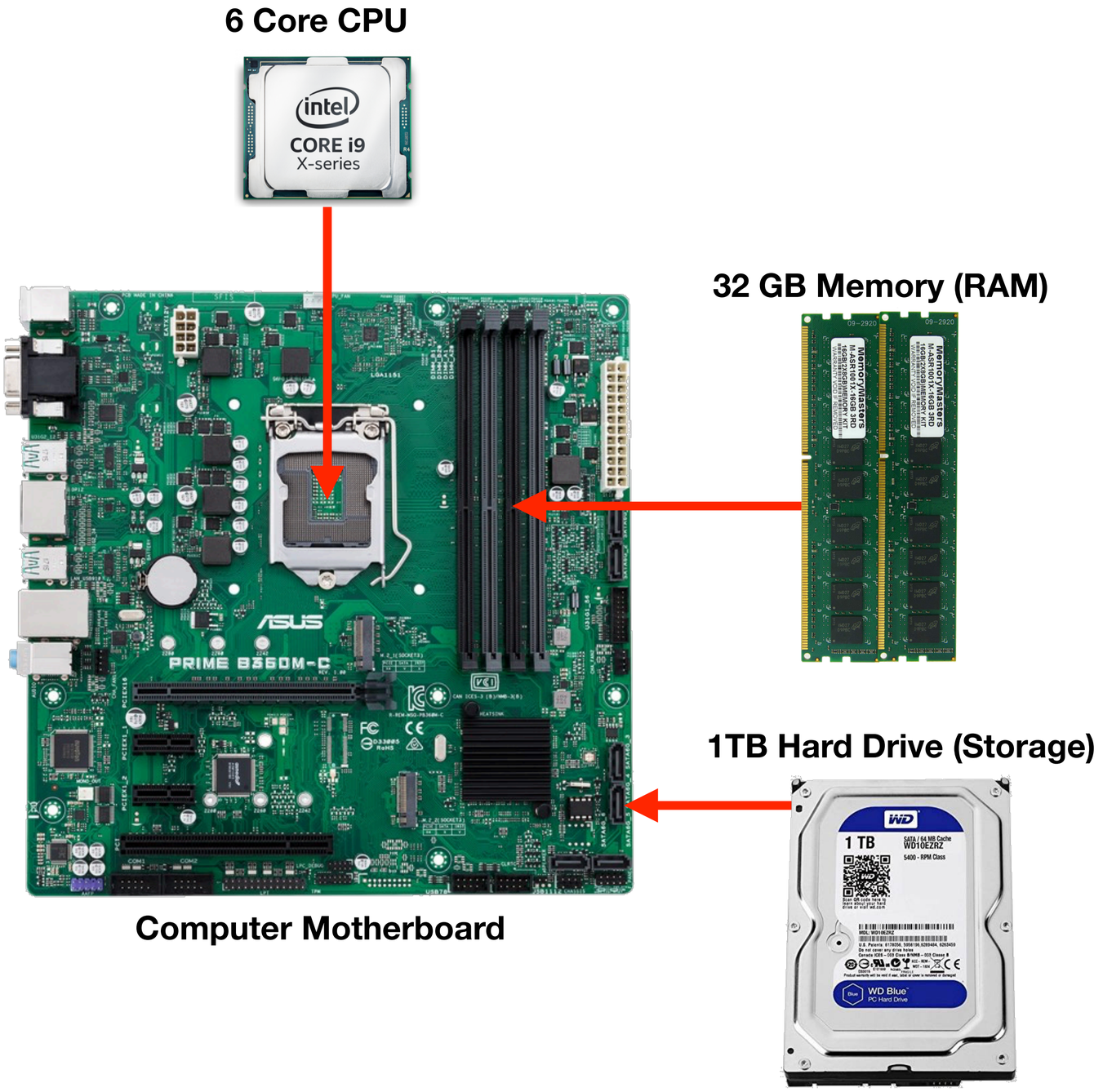
Physical Machines

Before you can understand Virtual Machines you must understand Physical Machines



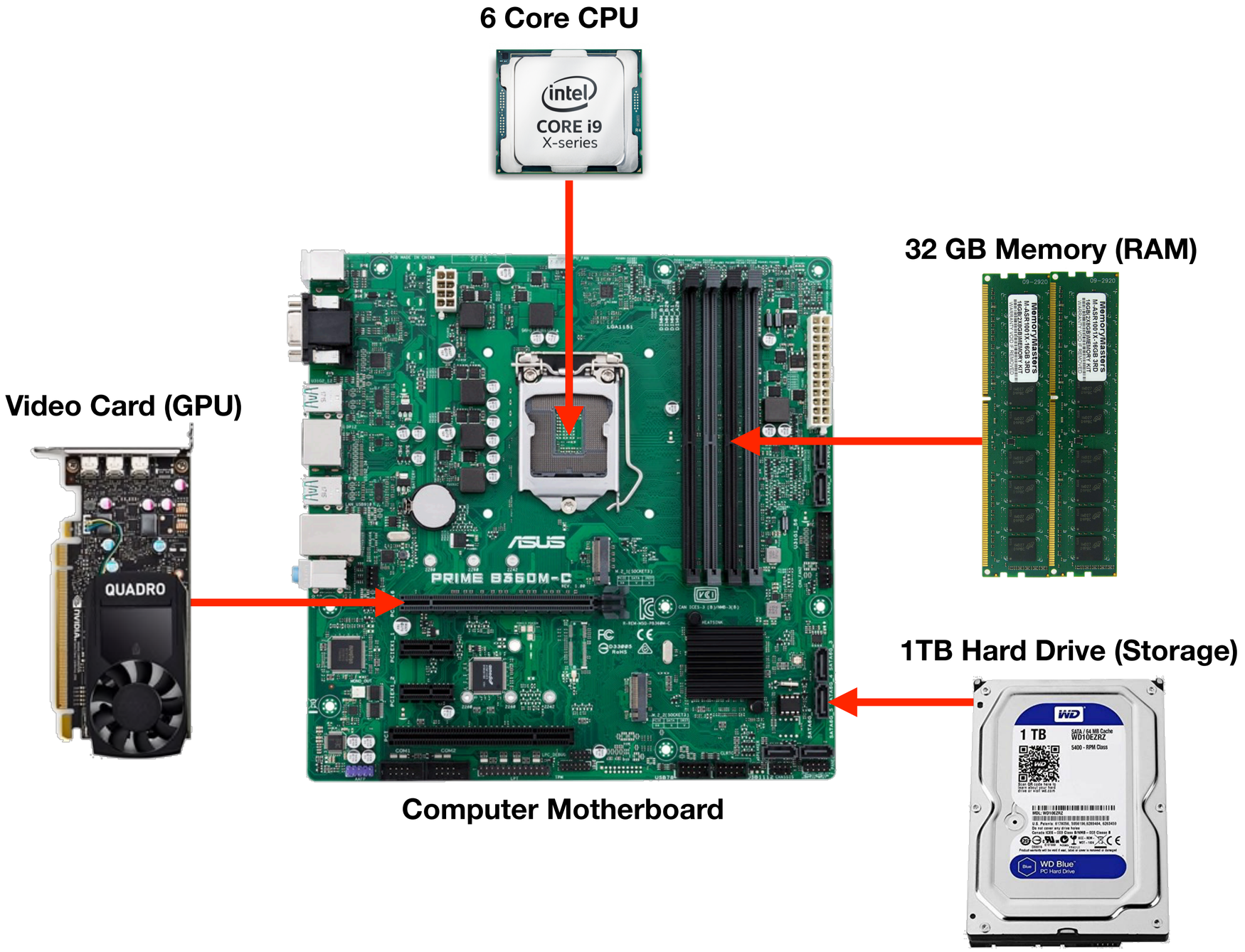
Physical Machines

Before you can understand Virtual Machines you must understand Physical Machines



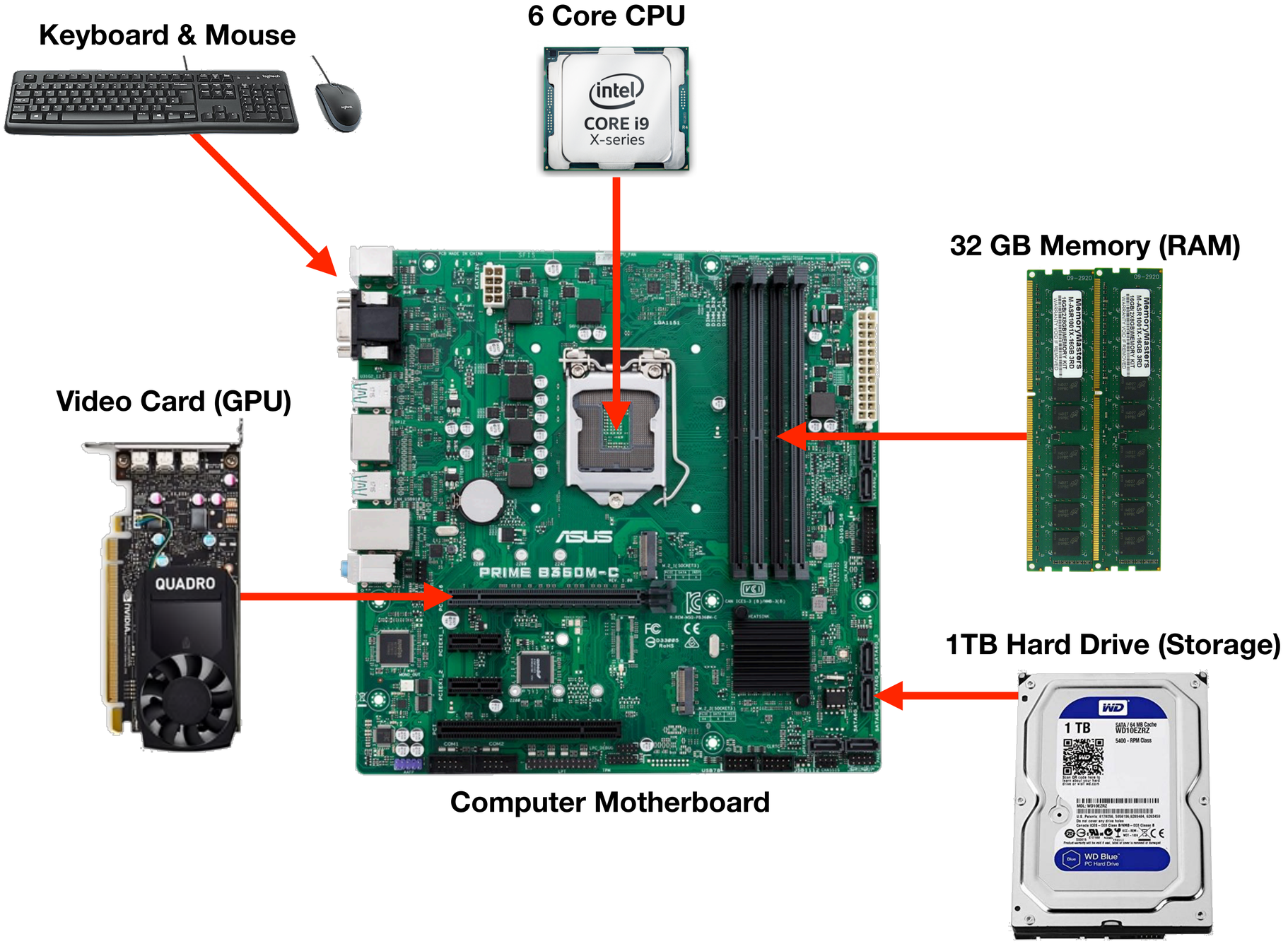
Physical Machines

Before you can understand Virtual Machines you must understand Physical Machines



Physical Machines

Before you can understand Virtual Machines you must understand Physical Machines



The Software Stack

The Software Stack

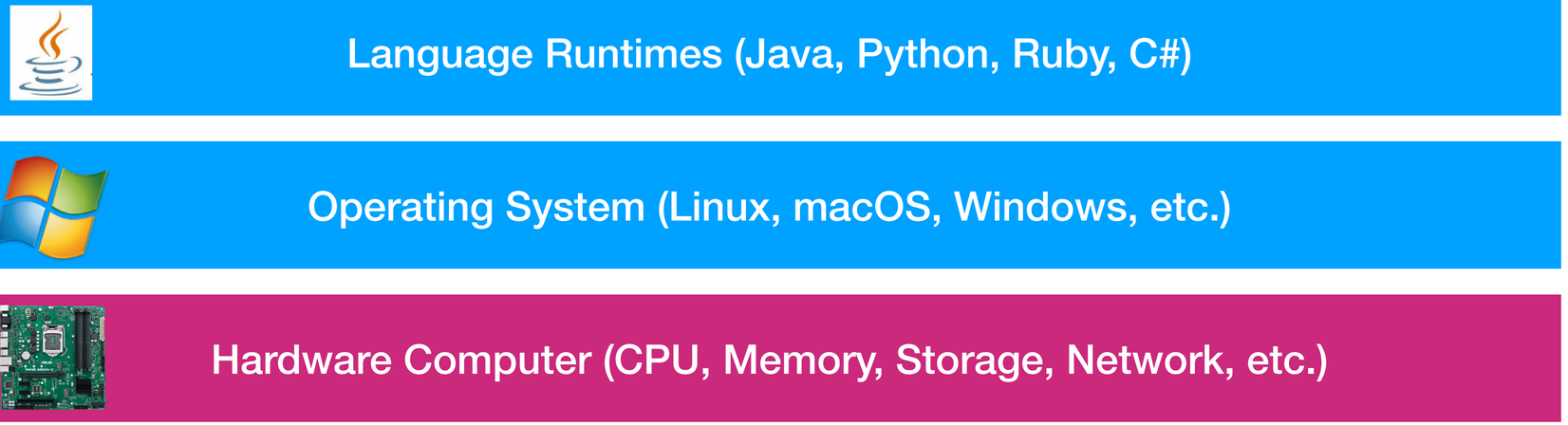
Hardware Computer (CPU, Memory, Storage, Network, etc.)

The Software Stack

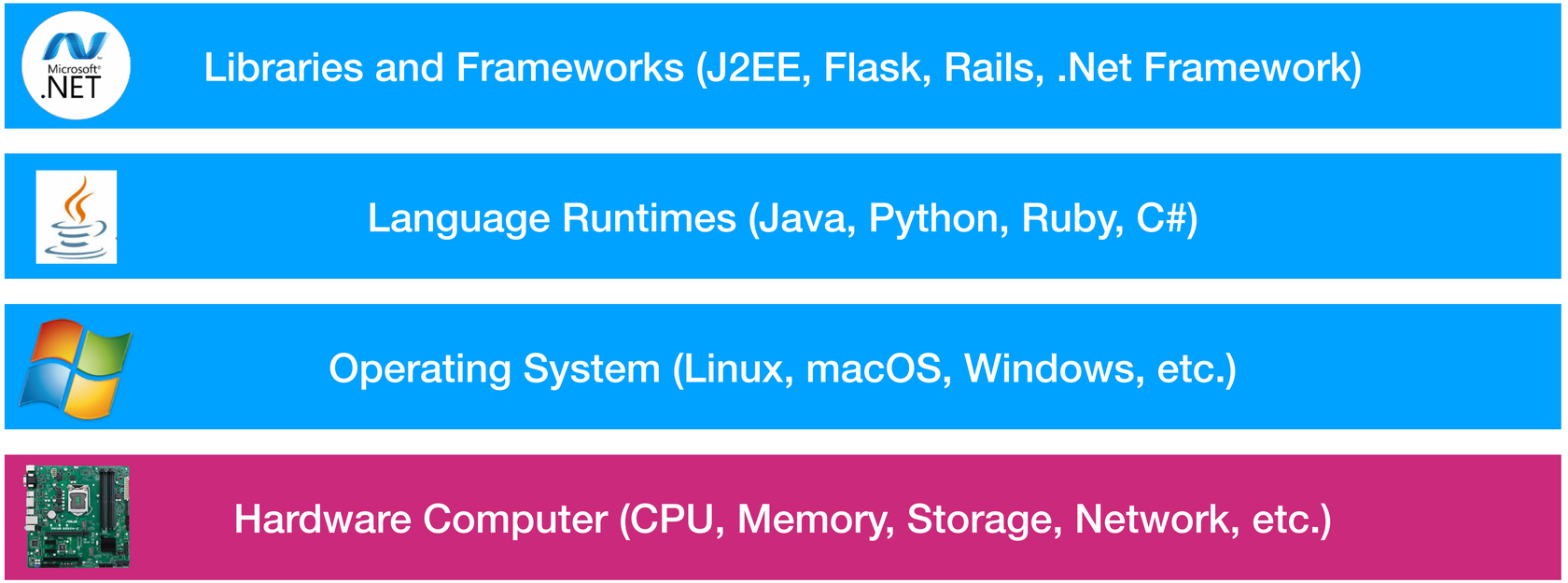
Operating System (Linux, macOS, Windows, etc.)

Hardware Computer (CPU, Memory, Storage, Network, etc.)

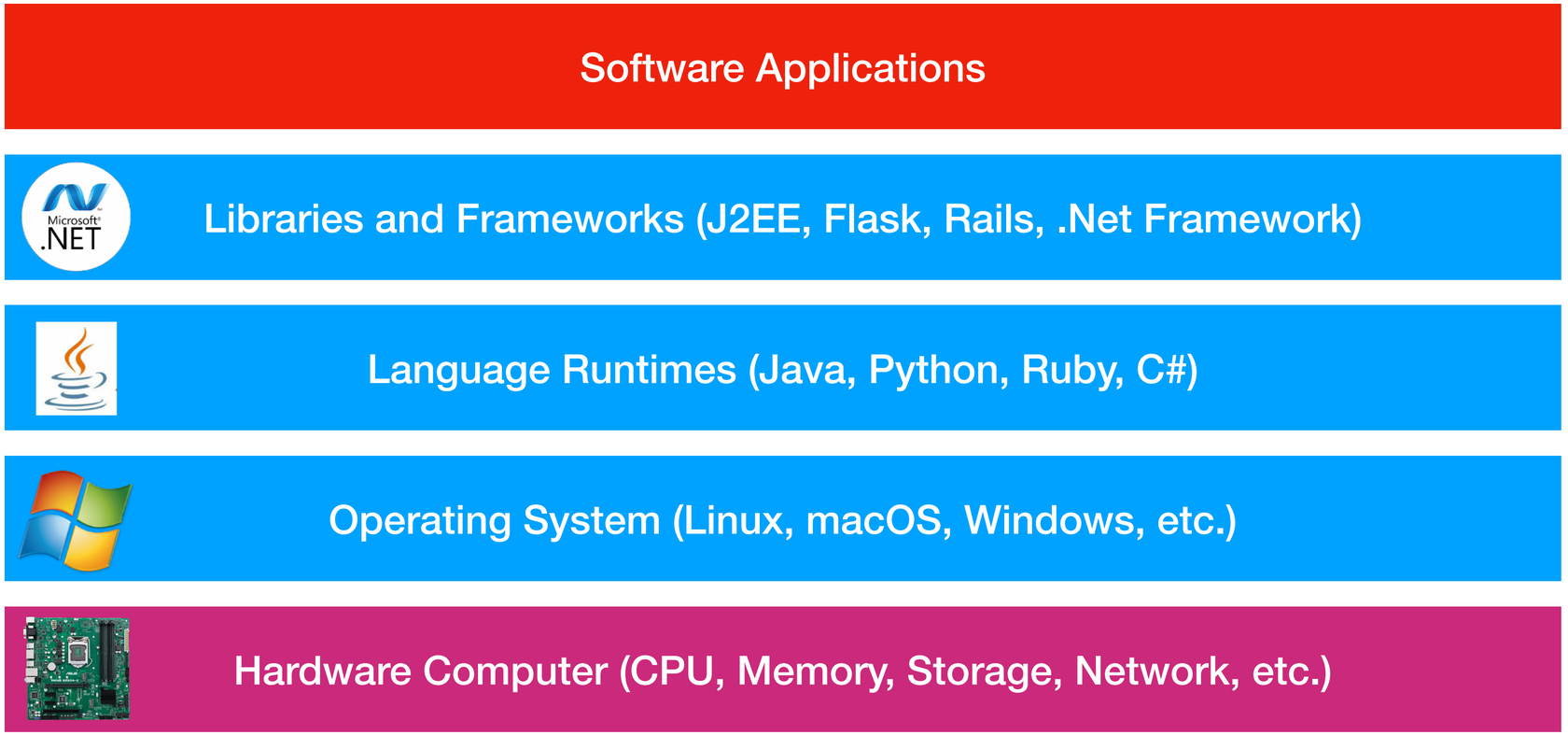
The Software Stack



The Software Stack

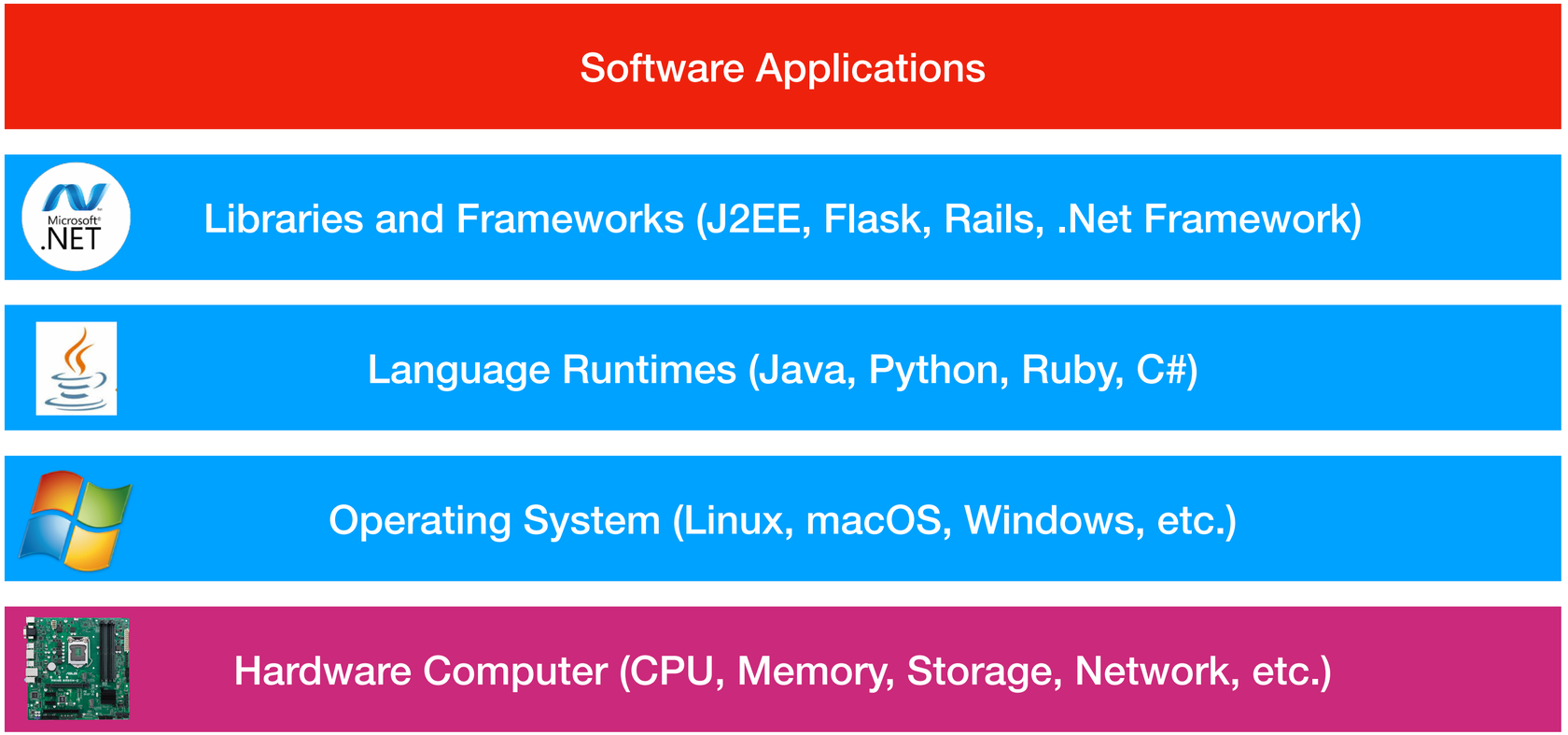


The Software Stack



The Software Stack

How do you know if you have the correct infrastructure to run the application?



The Software Stack

How do you know if you have the correct infrastructure to run the application?

Software Applications

Libraries and Frameworks (J2EE, Flask, Rails, .Net Framework)

Language Runtimes (Java, Python, Ruby, C#)

Operating System (Linux, macOS, Windows, etc.)

Hardware Computer (CPU, Memory, Storage, Network, etc.)

Do have enough CPU, Memory, and Disk space?

The Software Stack

How do you know if you have the correct infrastructure to run the application?

Software Applications

Libraries and Frameworks (J2EE, Flask, Rails, .Net Framework)

Language Runtimes (Java, Python, Ruby, C#)

Operating System (Linux, macOS, Windows, etc.)

Hardware Computer (CPU, Memory, Storage, Network, etc.)

Do have the correct version of Windows?

Do have enough CPU, Memory, and Disk space?

The Software Stack

How do you know if you have the correct infrastructure to run the application?

Software Applications

Libraries and Frameworks (J2EE, Flask, Rails, .Net Framework)

Language Runtimes (Java, Python, Ruby, C#)

Operating System (Linux, macOS, Windows, etc.)

Hardware Computer (CPU, Memory, Storage, Network, etc.)

Do have the correct version of Java?

Do have the correct version of Windows?

Do have enough CPU, Memory, and Disk space?

The Software Stack

How do you know if you have the correct infrastructure to run the application?

Do have the correct version of .Net Framework?

Do have the correct version of Java?

Do have the correct version of Windows?

Do have enough CPU, Memory, and Disk space?

Software Applications

Libraries and Frameworks (J2EE, Flask, Rails, .Net Framework)

Language Runtimes (Java, Python, Ruby, C#)

Operating System (Linux, macOS, Windows, etc.)

Hardware Computer (CPU, Memory, Storage, Network, etc.)

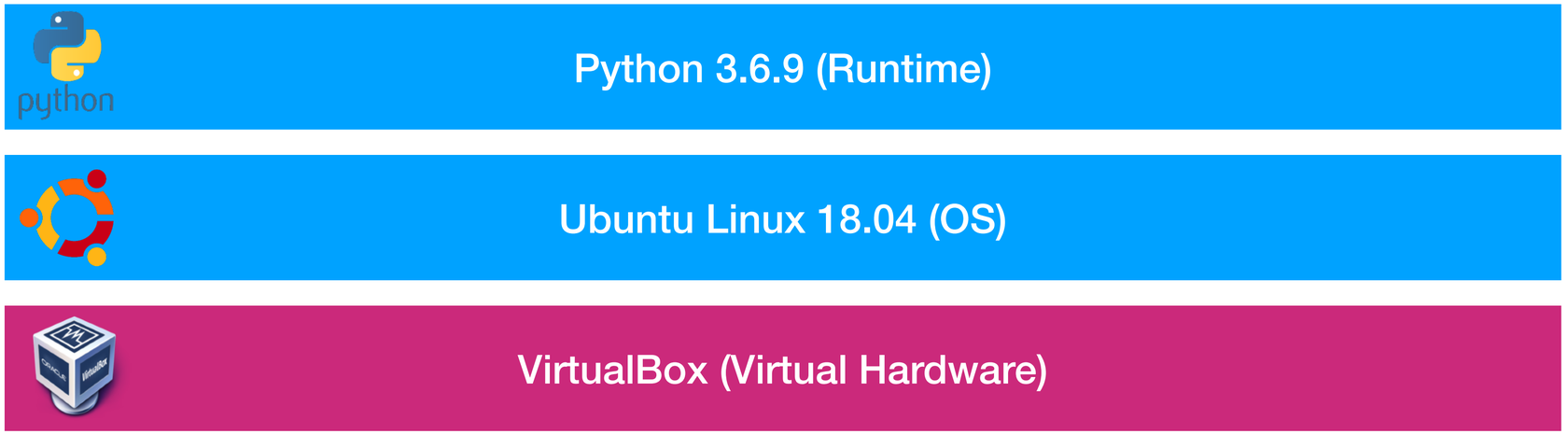
Software Stack for This Class

Software Stack for This Class

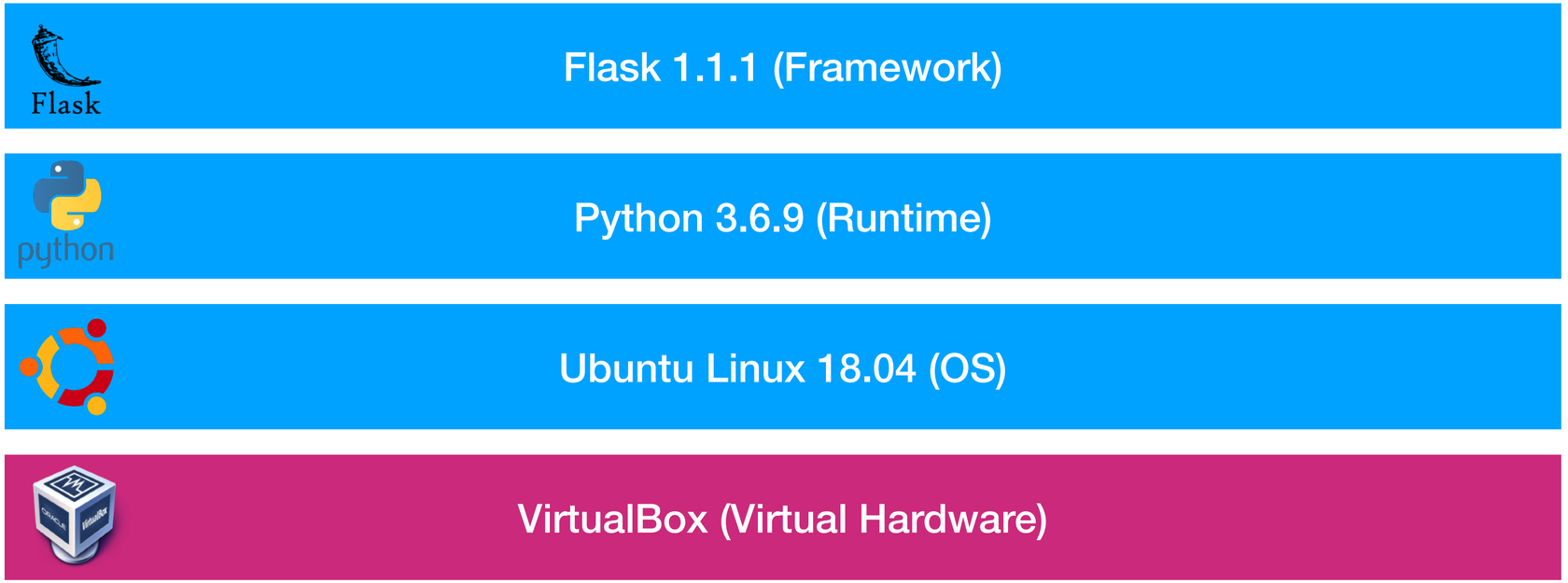
VirtualBox (Virtual Hardware)

Software Stack for This Class

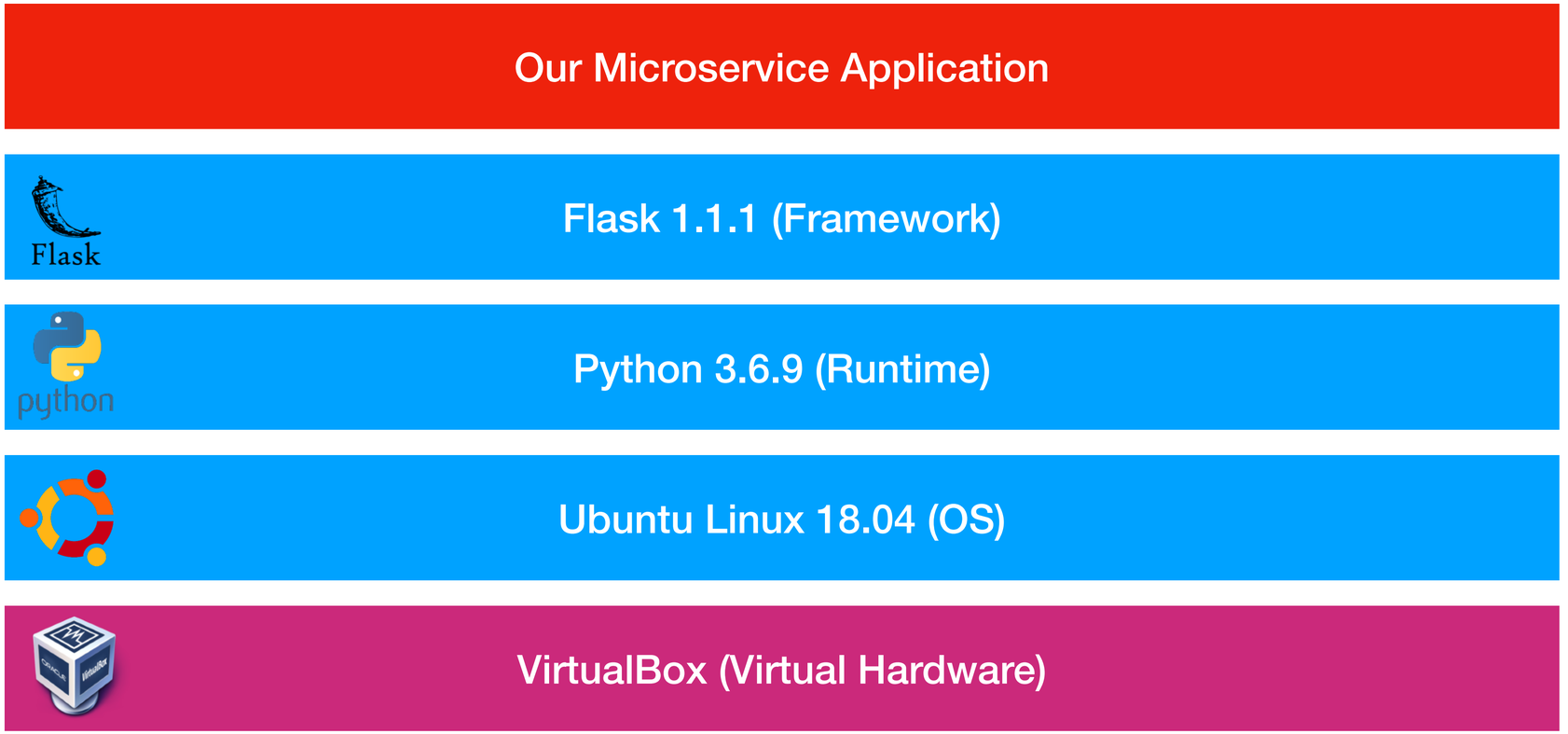
Software Stack for This Class



Software Stack for This Class

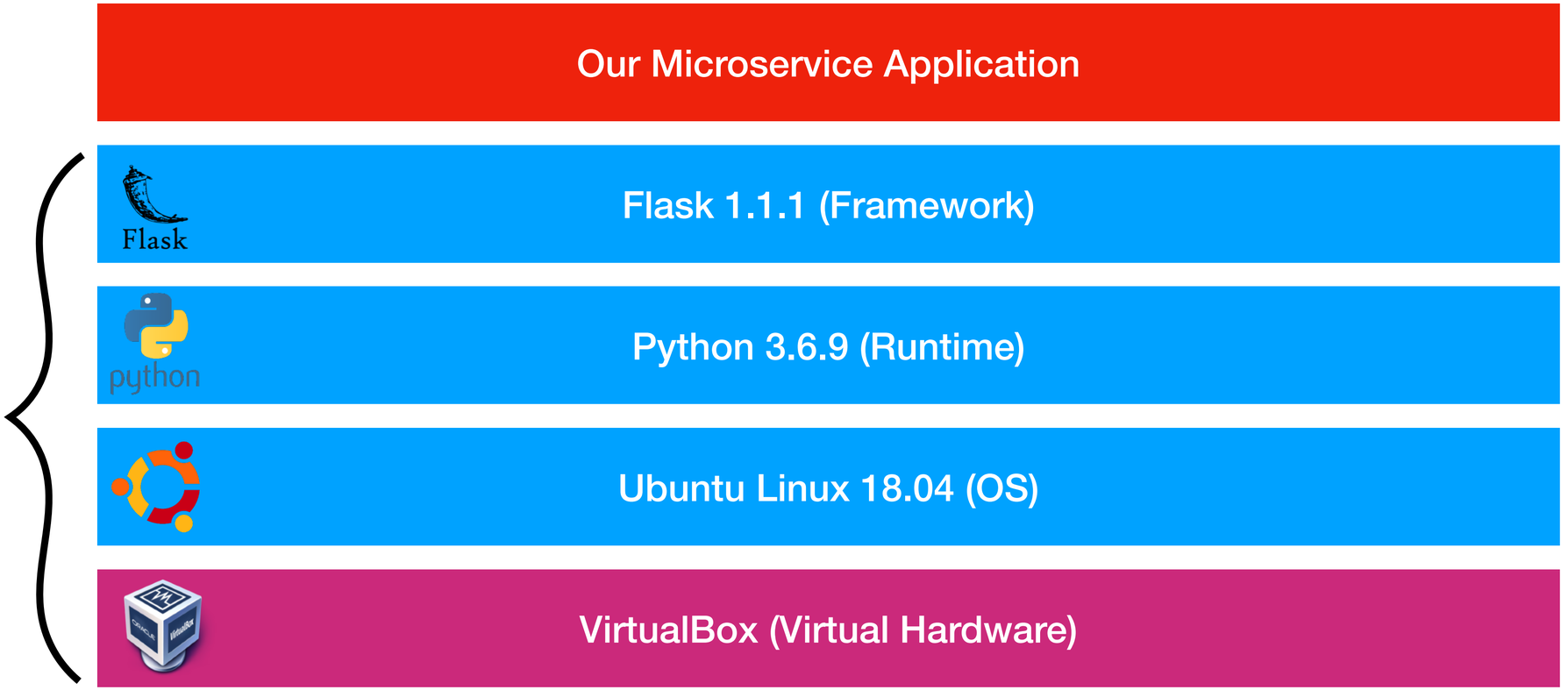


Software Stack for This Class



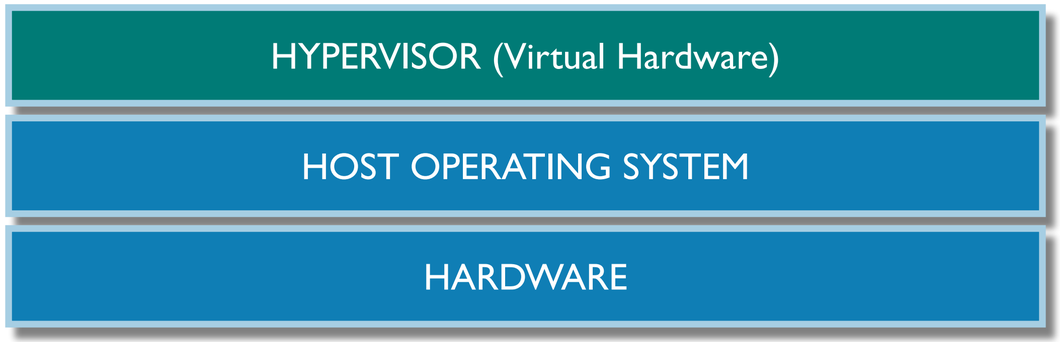
Software Stack for This Class

Vagrant will provision all of this for you with one command



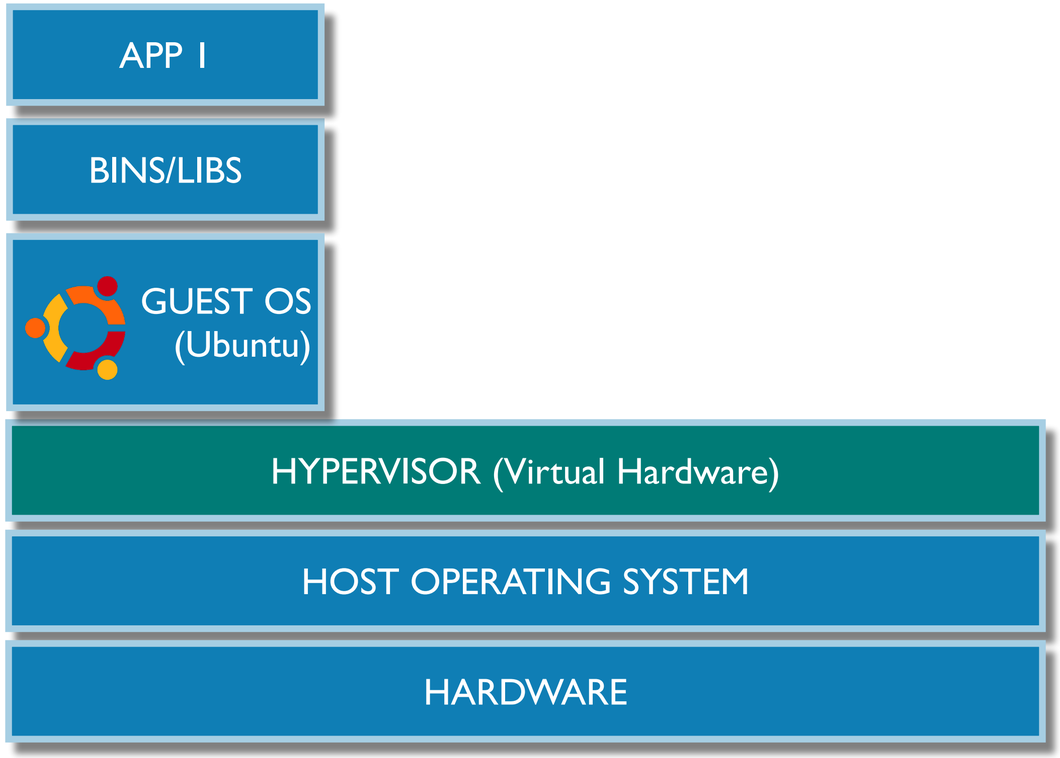
Virtual Machines (a.k.a. Virtual Hardware)

- **Virtual Machines** (VM) are created by emulating computer hardware in software
- The emulation is provided by software called a **Hypervisor**
- Each **Guest OS** thinks it's talking to dedicated computer hardware but it is really talking to the hypervisor that is sharing a much larger system



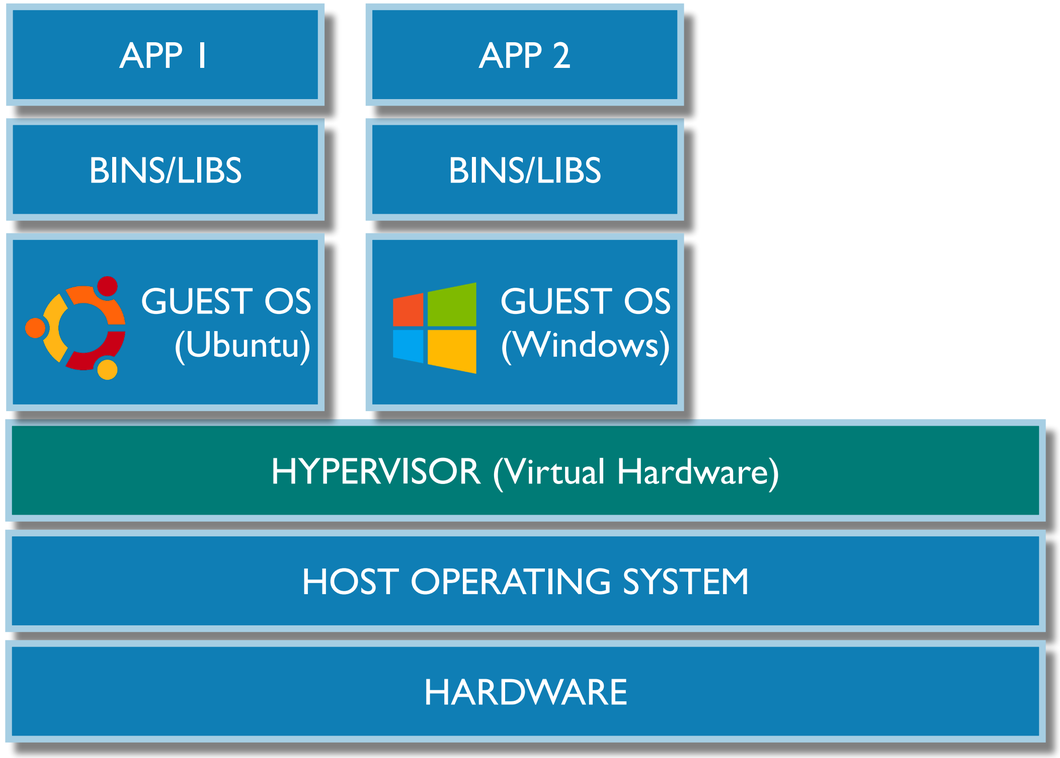
Virtual Machines (a.k.a. Virtual Hardware)

- **Virtual Machines** (VM) are created by emulating computer hardware in software
- The emulation is provided by software called a **Hypervisor**
- Each **Guest OS** thinks it's talking to dedicated computer hardware but it is really talking to the hypervisor that is sharing a much larger system



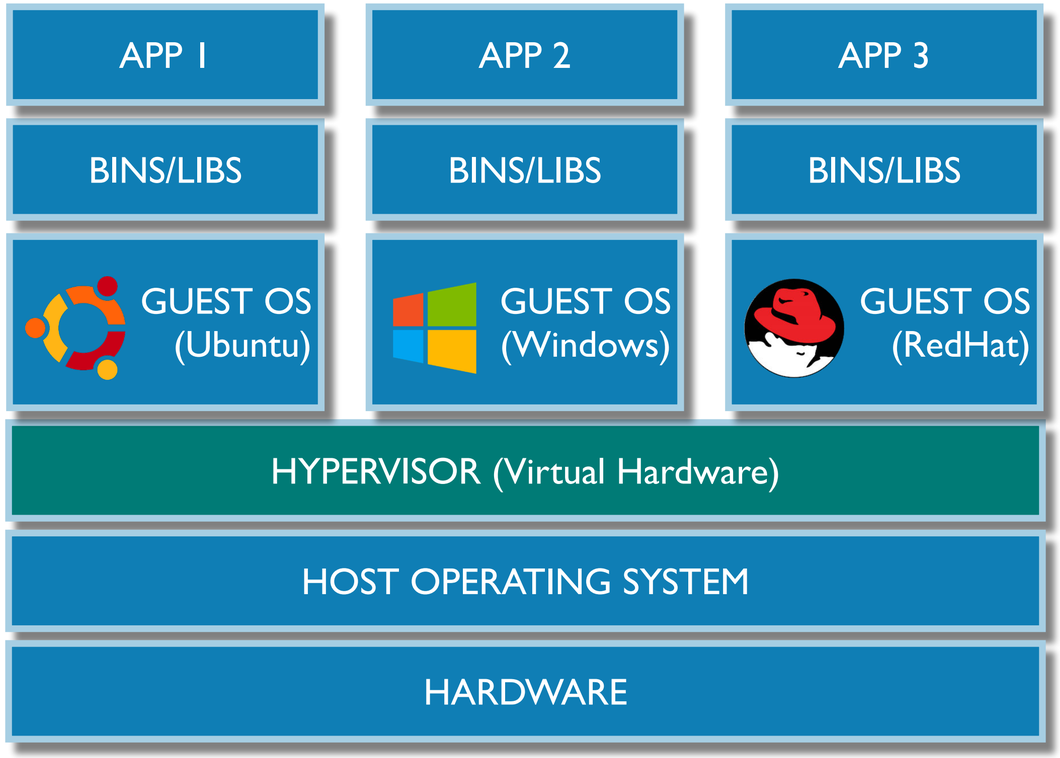
Virtual Machines (a.k.a. Virtual Hardware)

- **Virtual Machines** (VM) are created by emulating computer hardware in software
- The emulation is provided by software called a **Hypervisor**
- Each **Guest OS** thinks it's talking to dedicated computer hardware but it is really talking to the hypervisor that is sharing a much larger system



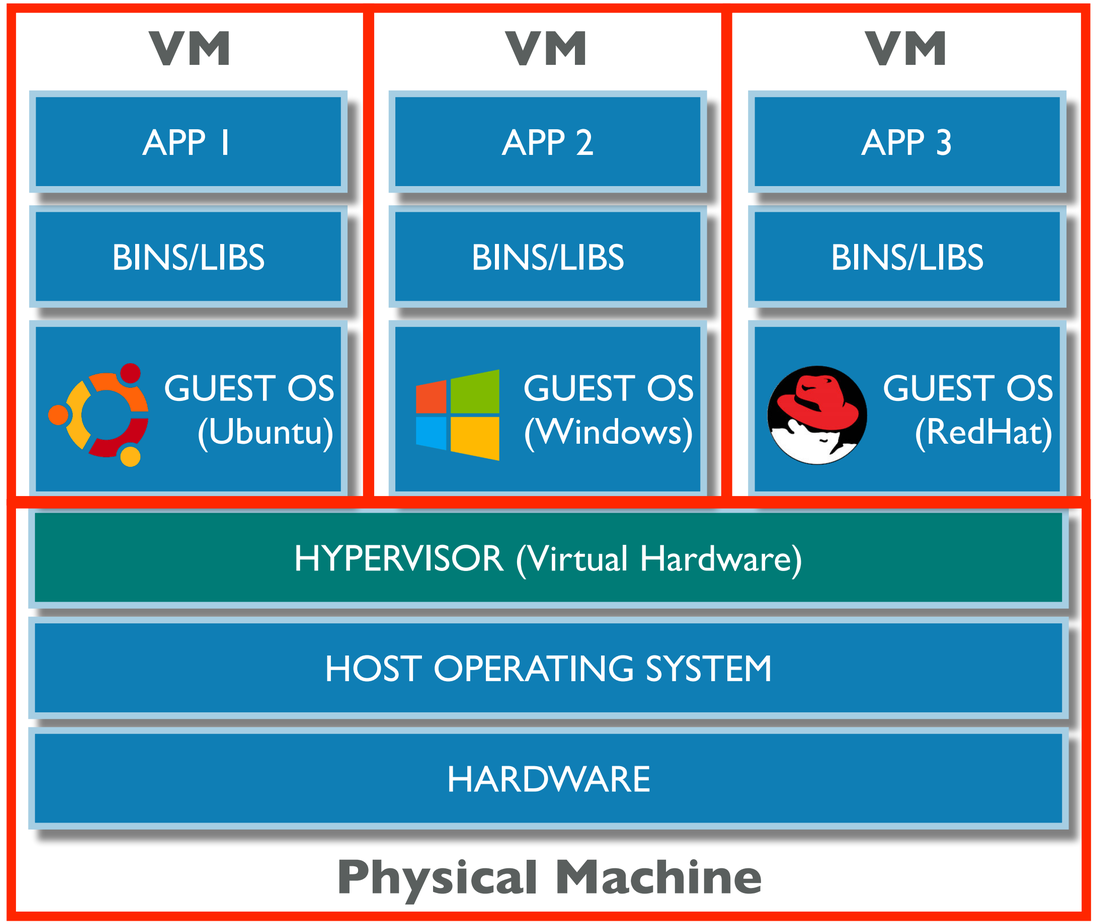
Virtual Machines (a.k.a. Virtual Hardware)

- **Virtual Machines** (VM) are created by emulating computer hardware in software
- The emulation is provided by software called a **Hypervisor**
- Each **Guest OS** thinks it's talking to dedicated computer hardware but it is really talking to the hypervisor that is sharing a much larger system



Virtual Machines (a.k.a. Virtual Hardware)

- **Virtual Machines** (VM) are created by emulating computer hardware in software
- The emulation is provided by software called a **Hypervisor**
- Each **Guest OS** thinks it's talking to dedicated computer hardware but it is really talking to the hypervisor that is sharing a much larger system



What is VirtualBox?

- VirtualBox is a free Hypervisor that runs on macOS, Windows, and Linux
- Emulates CPU, Memory, Hard Disks, CD, Network Adapters, etc.
- Similar to VMware Workstation on a PC, or VMware Fusions and Parallels Desktop on a Mac
- Allows you to run applications in a Virtual Machine

VirtualBox "Live" Demo

Go to: http://ubuntu.com/

Check requirements for Ubuntu

Download Ubuntu 18.04 LTS Image

Create a Virtual Machine

Install Ubuntu on the VM

Let's build an Ubuntu Server on VirtualBox

What is Vagrant?

- Vagrant is a developers tool for automating the creation of lightweight, reproducible and portable virtual environments via command-line
- It supports VirtualBox, VMware, SoftLayer, Amazon AWS and Digital Ocean
- It supports configuration management utilities like Puppet, Chef, etc.
- It supports Docker natively which makes is easy to use Docker containers in VMs

Vagrant is the Puppet Master

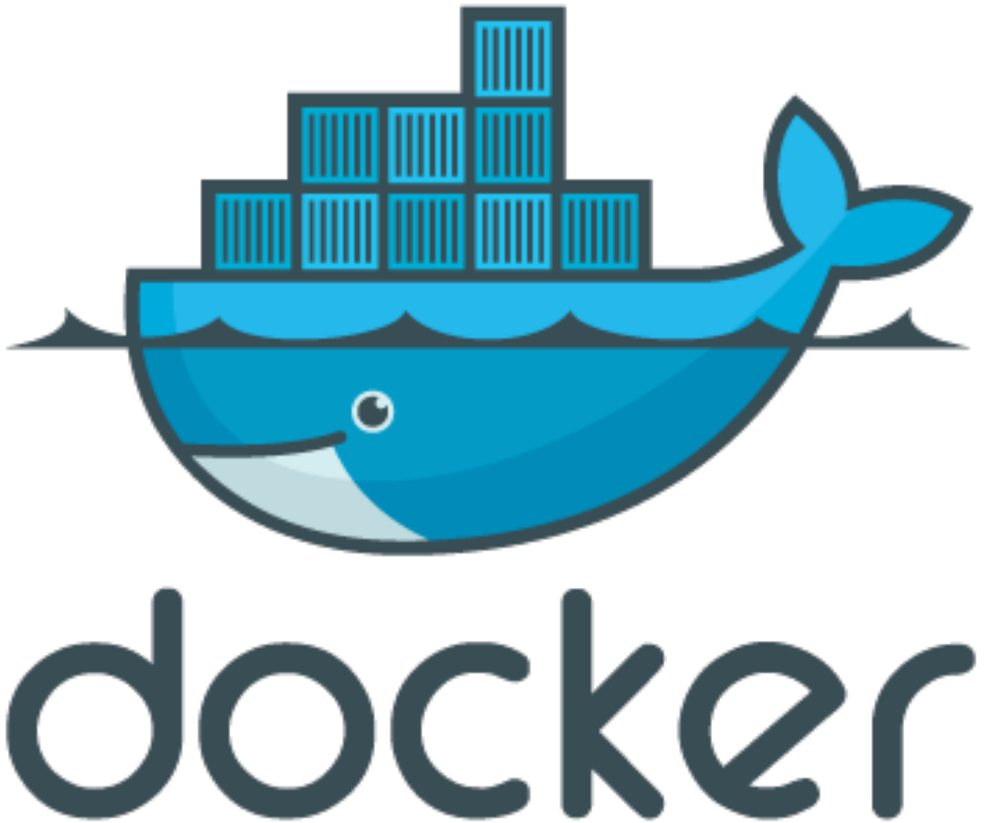
- Vagrant controls VirtualBox so you don't have to
- Vagrant also installs software inside of the Virtual Machine
- All of this is controlled by a single file called: `Vagrantfile`
- You start it with one command: `vagrant up`
- This is known as "*Infrastructure as Code*"

Example Vagrantfile

```
Vagrant.configure(2) do |config|
  config.vm.box = "ubuntu/bionic64"

  config.vm.network "private_network", ip: "192.168.33.10"

  config.vm.provider "virtualbox" do |vb|
    vb.memory = "1024"
    vb.cpus = 2
  end

  config.vm.provision "shell", inline: <<-SHELL
    sudo apt-get update
    sudo apt-get install -y git python-pip python-dev
    sudo apt-get -y autoremove
  SHELL

end
```

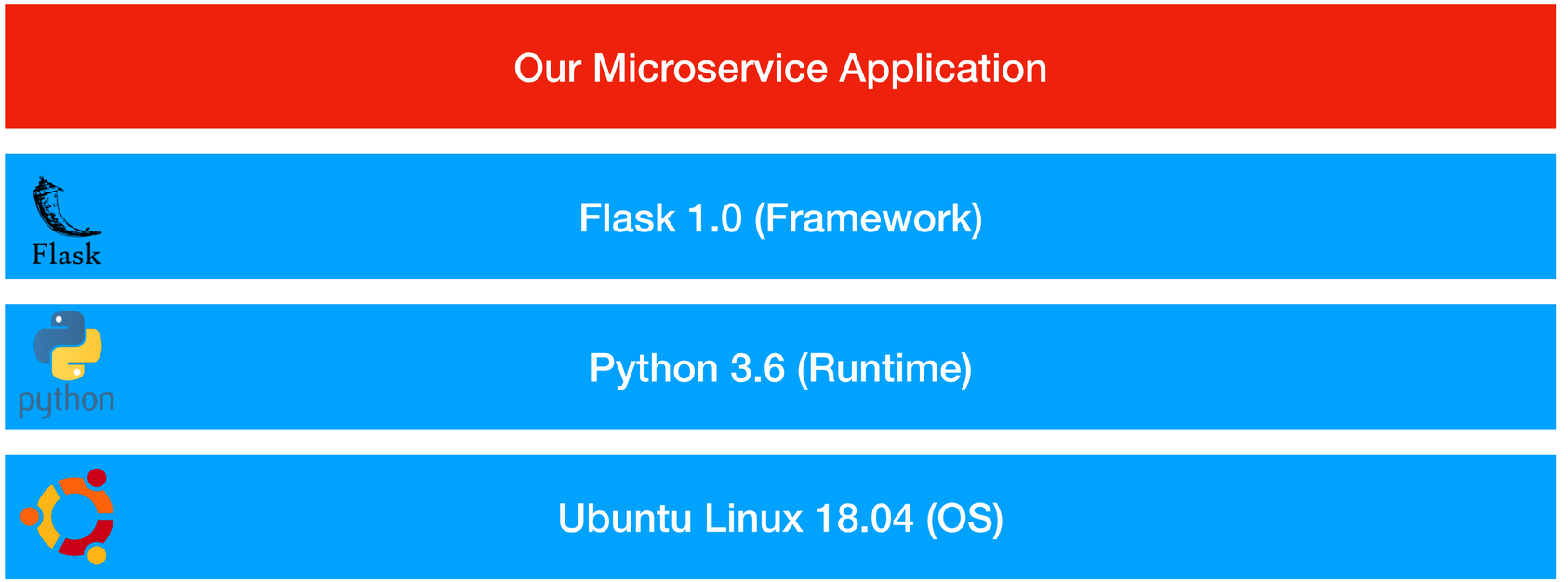
What is Docker?

- Docker packages applications into Containers that include all of their dependencies, but share the kernel with other containers
- Containers run as an isolated process in userspace on the host operating system
- Containers are not tied to any specific infrastructure
  - Docker containers run on any computer, on any infrastructure and in any cloud.
- This guarantees that it will always run the same, regardless of the environment it is running in.



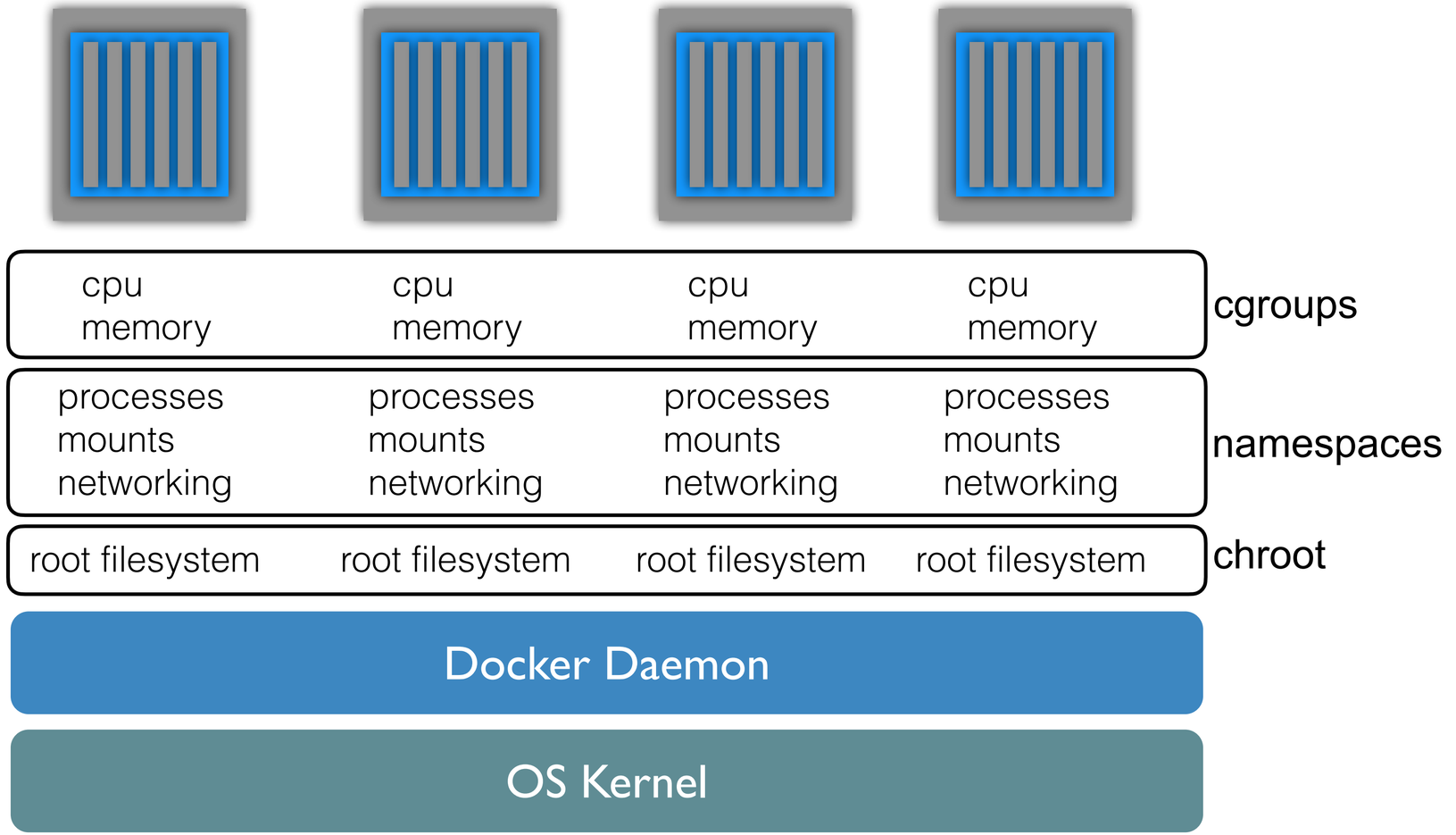
What is a Docker Image

- Docker can take your entire Software Stack and package it into an Image from which you can create Containers (like Tupperware for your code)



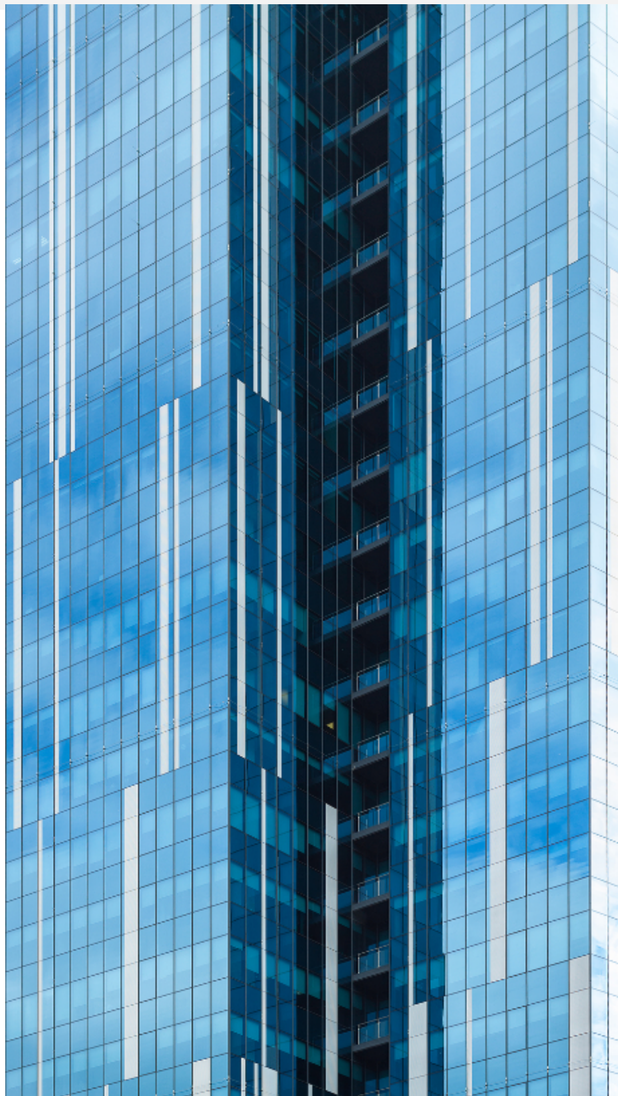
Containers are just Linux Capabilities Under the Covers

- Containers are:
  - Linux® processes
  - with isolation and resource confinement
  - that enable you to run sandboxed applications
  - on a shared host kernel
  - using cgroups, namespaces, and chroot



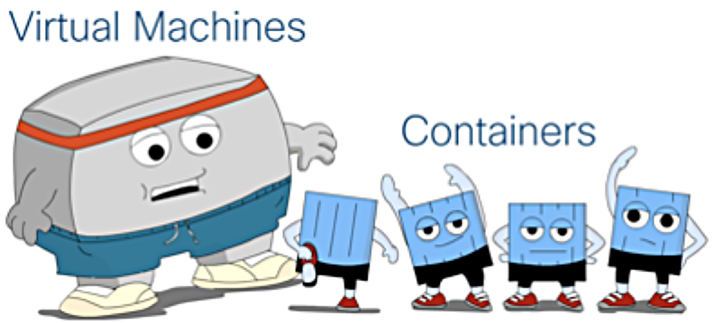
Put another way

- Docker Containers allow you to control
  - What resources a process can **see**
  - What resources a process can **control**
  - What filesystem a process **uses**



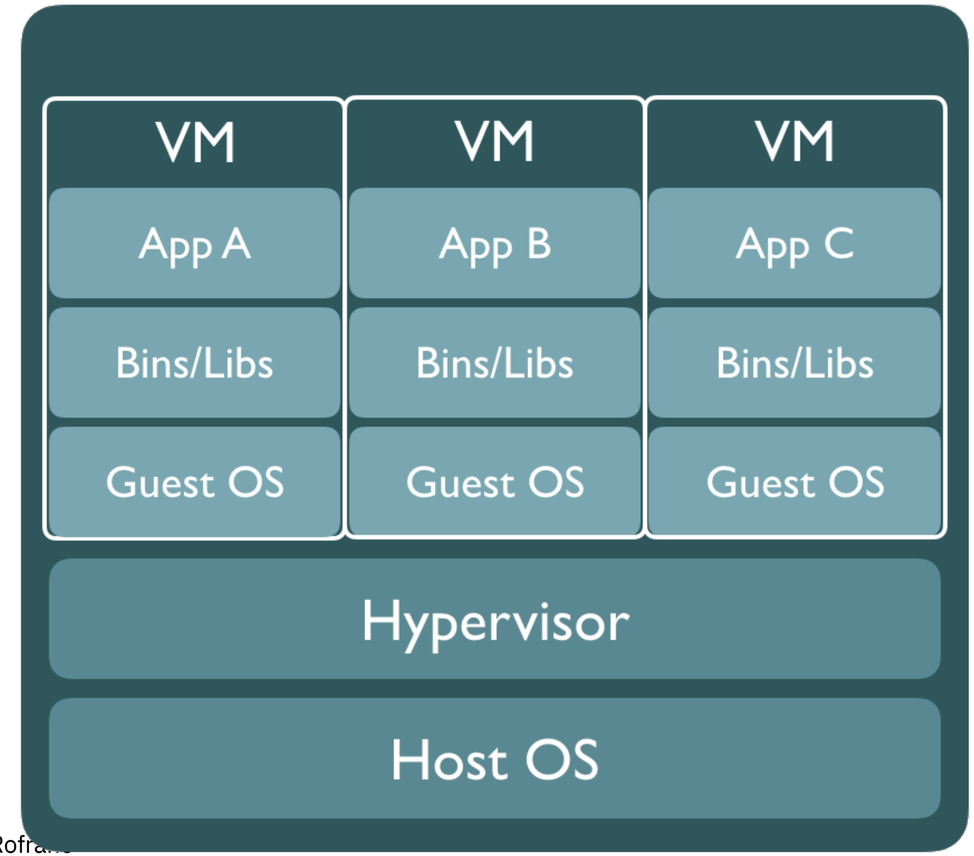
Containers vs Virtual Machines

- A container runs natively on Linux and shares the kernel of the host machine with other containers
  - It runs a discrete process, taking no more memory than any other executable, making it lightweight
- By contrast, a virtual machine (VM) runs a full-blown “guest” operating system with virtual access to host resources through a hypervisor
  - In general, VMs provide an environment with more resources than most applications need



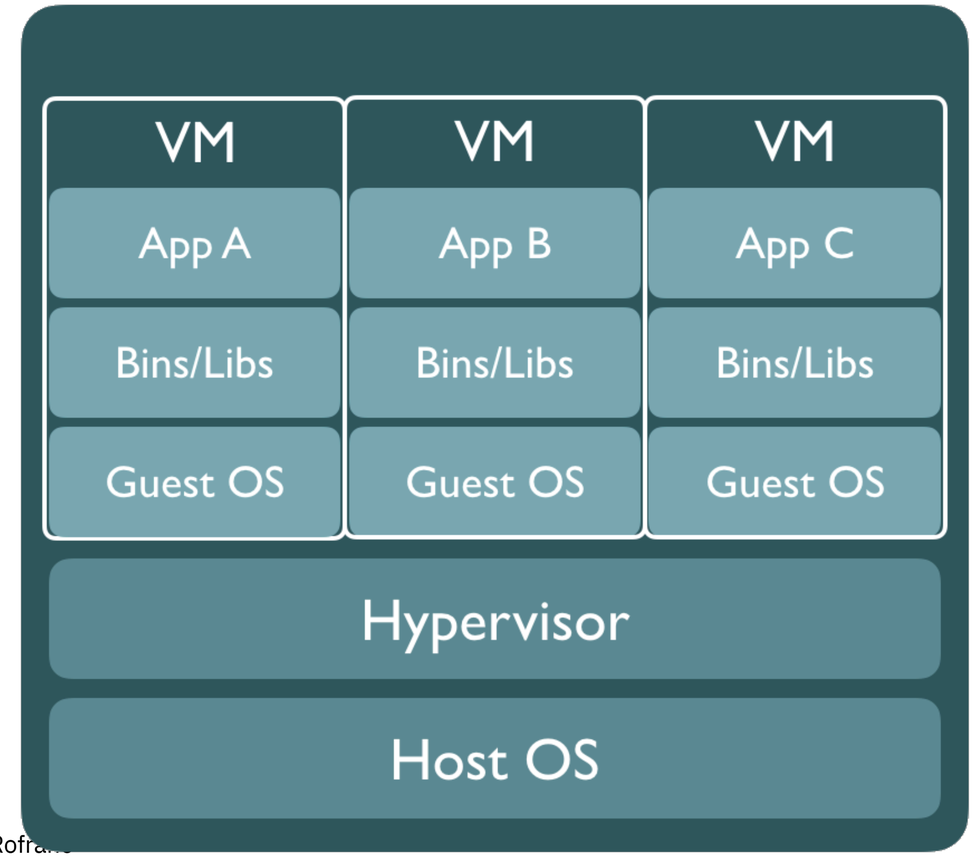
Containers vs Virtual Machines

Virtual Machines are heavy-weight emulations of real hardware

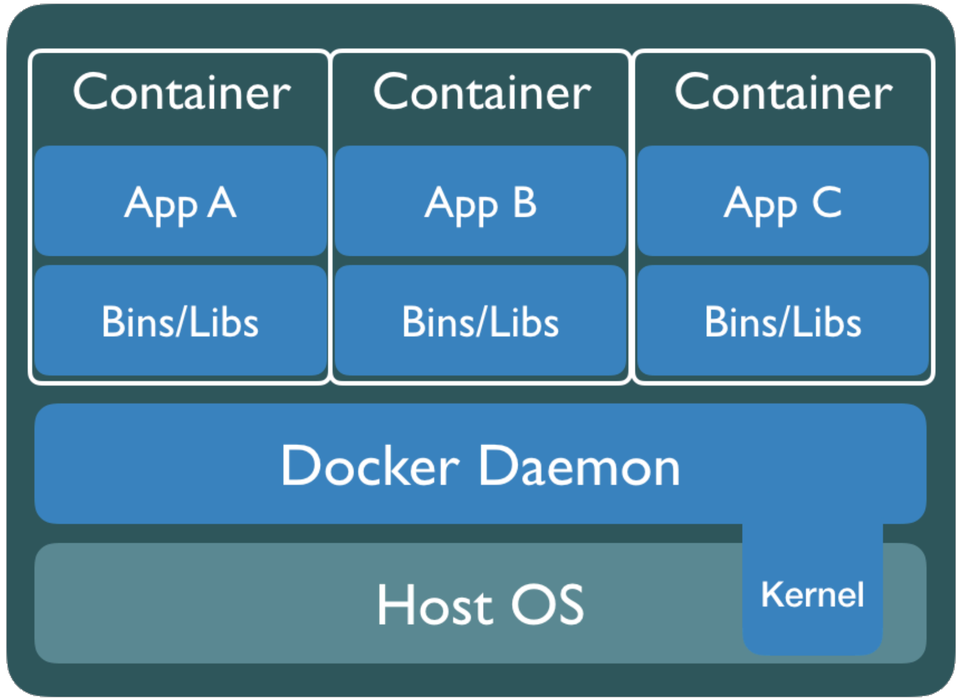


Containers vs Virtual Machines

Virtual Machines are heavy-weight emulations of real hardware

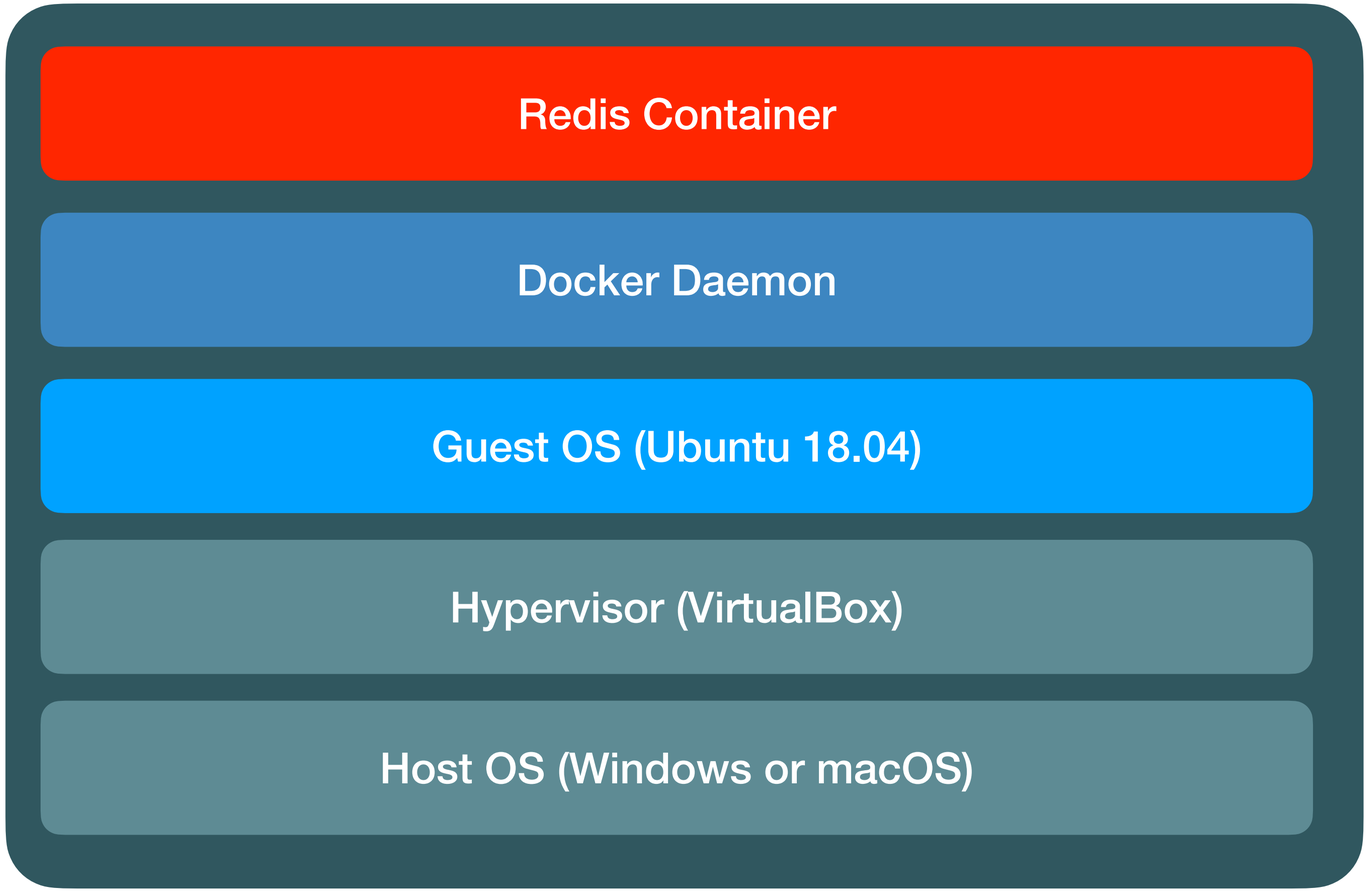


Containers are light-weight process
The app looks like it's running on the Host OS



Containers inside a Virtual Machine

We will be running Docker containers inside of a Virtual Machine



Baked vs. Fried

- Do you Bake software into the image?
  - VM's can be created quicker because most software is pre-installed
- Or do you Fry it up fresh when you need it?
  - More versatile and less maintenance when software needs updating
- I like to Fry!


Summit of Innovation
Sea of Technical Debt

Hands-On

“live session”

Demo from Scratch

- Create a folder to work in:

```
$ mkdir lab-vagrant-demo
$ cd lab-vagrant-demo
```

- Go from Zero to running Ubuntu 18.04 Bionic 64 with these 2 commands:

```
$ vagrant init ubuntu/bionic64
$ vagrant up --provider virtualbox
```

- That’s it! 😀

* `--provider virtualbox` (only needs for Windows)

What Did “init” do?

- A default `Vagrantfile` was created with the `init` command
- The `Vagrantfile` controls how the VM will be provisioned
- By editing the `Vagrantfile` we can change the VM’s behavior
- The `Vagrantfile` will be what you check-in to GitHub

Default Vagrantfile

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

  config.vm.box = "ubuntu/bionic64"

  # config.vm.box_check_update = false

  # config.vm.network "forwarded_port", guest: 80, host: 8080

  # config.vm.network "private_network", ip: "192.168.33.10"

  # config.vm.network "public_network"

  # config.vm.synced_folder "../data", “/vagrant_data"

  # config.vm.provider "virtualbox" do |vb|
  #   vb.gui = true
  #   vb.memory = "1024"
  # end

  # config.push.define "atlas" do |push|
  #   push.app = "YOUR_ATLAS_USERNAME/YOUR_APPLICATION_NAME"
  # end

  # config.vm.provision "shell", inline: <<-SHELL
  #   apt-get update
  #   apt-get install -y apache2
  # SHELL
end
```

Default Vagrantfile

This is a Ruby file that uses Ruby syntax

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

  config.vm.box = "ubuntu/bionic64"

  # config.vm.box_check_update = false

  # config.vm.network "forwarded_port", guest: 80, host: 8080

  # config.vm.network "private_network", ip: "192.168.33.10"

  # config.vm.network "public_network"

  # config.vm.synced_folder "../data", “/vagrant_data"

  # config.vm.provider "virtualbox" do |vb|
  #   vb.gui = true
  #   vb.memory = "1024"
  # end

  # config.push.define "atlas" do |push|
  #   push.app = "YOUR_ATLAS_USERNAME/YOUR_APPLICATION_NAME"
  # end

  # config.vm.provision "shell", inline: <<-SHELL
  #   apt-get update
  #   apt-get install -y apache2
  # SHELL
end
```

Default Vagrantfile

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

  config.vm.box = "ubuntu/bionic64"

  # config.vm.box_check_update = false

  # config.vm.network "forwarded_port", guest: 80, host: 8080

  # config.vm.network "private_network", ip: "192.168.33.10"

  # config.vm.network "public_network"

  # config.vm.synced_folder "../data", “/vagrant_data"

  # config.vm.provider "virtualbox" do |vb|
  #   vb.gui = true
  #   vb.memory = "1024"
  # end

  # config.push.define "atlas" do |push|
  #   push.app = "YOUR_ATLAS_USERNAME/YOUR_APPLICATION_NAME"
  # end

  # config.vm.provision "shell", inline: <<-SHELL
  #   apt-get update
  #   apt-get install -y apache2
  # SHELL
end
```

This is a Ruby file that uses Ruby syntax

Selects the box to use as a base OS

Default Vagrantfile

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

  config.vm.box = "ubuntu/bionic64"

  # config.vm.box_check_update = false

  # config.vm.network "forwarded_port", guest: 80, host: 8080

  # config.vm.network "private_network", ip: "192.168.33.10"

  # config.vm.network "public_network"

  # config.vm.synced_folder "../data", “/vagrant_data"

  # config.vm.provider "virtualbox" do |vb|
  #   vb.gui = true
  #   vb.memory = "1024"
  # end

  # config.push.define "atlas" do |push|
  #   push.app = "YOUR_ATLAS_USERNAME/YOUR_APPLICATION_NAME"
  # end

  # config.vm.provision "shell", inline: <<-SHELL
  #   apt-get update
  #   apt-get install -y apache2
  # SHELL
end
```

This is a Ruby file that uses Ruby syntax

Selects the box to use as a base OS

Everything else is commented out

What Did “up” Do?

- Vagrant downloaded the ubuntu/bionic64 box from atlas.hashicorp.com
- VirtualBox was called to create a VM from that box
- Installed VirtualBox tools and setup the network and shared your current folder as /vagrant
- Ran provisioning scripts to prepare the VM (more on that later)
- Set up SSH keys to logon to the VM

Initial Vagrant Run

```
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Checking if box 'ubuntu/bionic64' is up to date...
==> default: Clearing any previously set forwarded ports...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
    default: Adapter 2: hostonly
==> default: Forwarding ports...
    default: 5000 (guest) => 5000 (host) (adapter 1)
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Running 'pre-boot' VM customizations...
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: ubuntu
    default: SSH auth method: password
    default: Warning: Remote connection disconnect. Retrying...
    default: Warning: Connection reset. Retrying...
==> default: Machine booted and ready!
[default] GuestAdditions 5.1.26 running --- OK.
==> default: Checking for guest additions in VM...
==> default: Configuring and enabling network interfaces...
==> default: Mounting shared folders...
    default: /vagrant => /Users/rofrano/github/lab-vagrant
==> default: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> default: flag to force provisioning. Provisioners marked to run always will still run.
```

Initial Vagrant Run

```
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Checking if box 'ubuntu/bionic64' is up to date...
==> default: Clearing any previously set forwarded ports...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
    default: Adapter 2: hostonly
==> default: Forwarding ports...
    default: 5000 (guest) => 5000 (host) (adapter 1)
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Running 'pre-boot' VM customizations...
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: ubuntu
    default: SSH auth method: password
    default: Warning: Remote connection disconnect. Retrying...
    default: Warning: Connection reset. Retrying...
==> default: Machine booted and ready!
[default] GuestAdditions 5.1.26 running --- OK.
==> default: Checking for guest additions in VM...
==> default: Configuring and enabling network interfaces...
==> default: Mounting shared folders...
    default: /vagrant => /Users/rofrano/github/lab-vagrant
==> default: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> default: flag to force provisioning. Provisioners marked to run always will still run.
```

Creates the VM

Initial Vagrant Run

```
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Checking if box 'ubuntu/bionic64' is up to date...
==> default: Clearing any previously set forwarded ports...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
    default: Adapter 2: hostonly
==> default: Forwarding ports...
    default: 5000 (guest) => 5000 (host) (adapter 1)
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Running 'pre-boot' VM customizations...
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: ubuntu
    default: SSH auth method: password
    default: Warning: Remote connection disconnect. Retrying...
    default: Warning: Connection reset. Retrying...
==> default: Machine booted and ready!
[default] GuestAdditions 5.1.26 running --- OK.
==> default: Checking for guest additions in VM...
==> default: Configuring and enabling network interfaces...
==> default: Mounting shared folders...
    default: /vagrant => /Users/rofrano/github/lab-vagrant
==> default: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> default: flag to force provisioning. Provisioners marked to run always will still run.
```

Creates the VM

Setup Network/Port Forwarding

Initial Vagrant Run

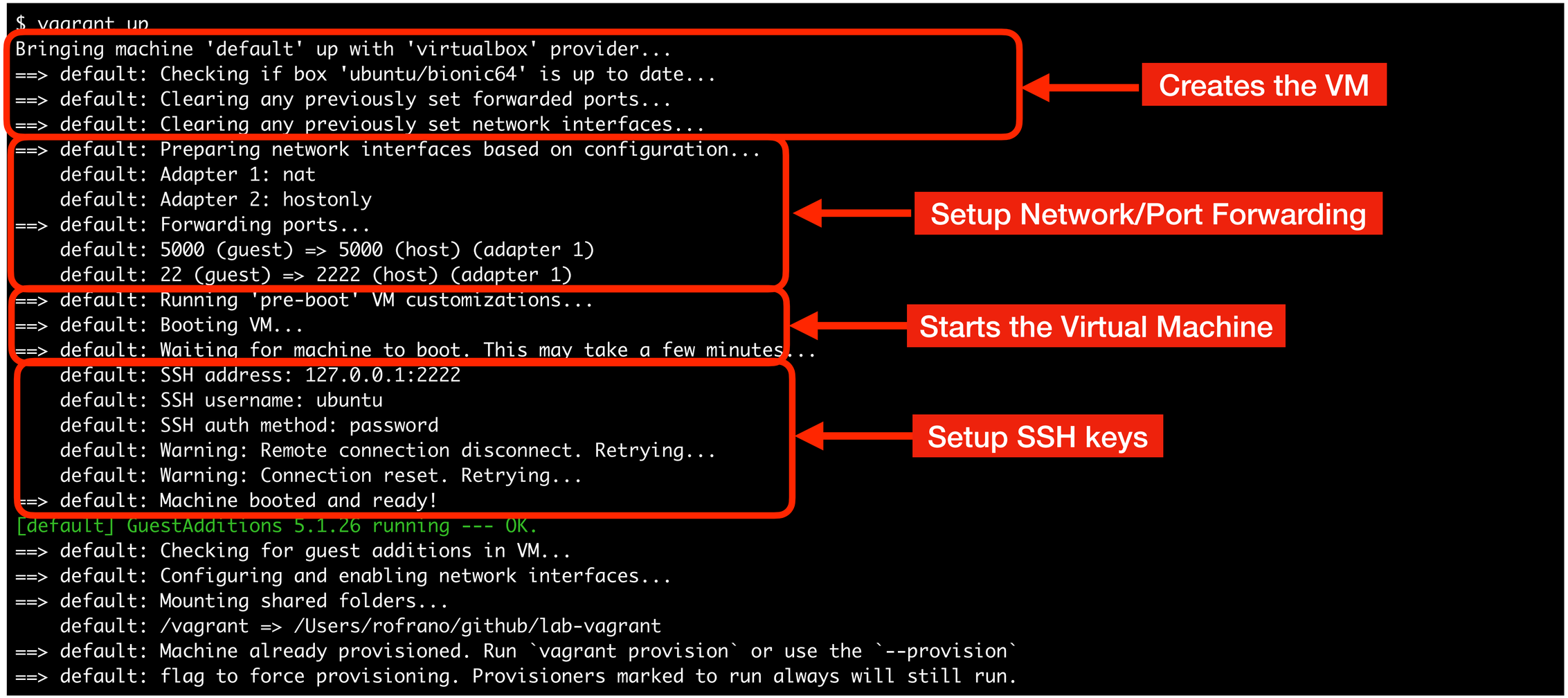
```
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Checking if box 'ubuntu/bionic64' is up to date...
==> default: Clearing any previously set forwarded ports...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
    default: Adapter 2: hostonly
==> default: Forwarding ports...
    default: 5000 (guest) => 5000 (host) (adapter 1)
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Running 'pre-boot' VM customizations...
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: ubuntu
    default: SSH auth method: password
    default: Warning: Remote connection disconnect. Retrying...
    default: Warning: Connection reset. Retrying...
==> default: Machine booted and ready!
[default] GuestAdditions 5.1.26 running --- OK.
==> default: Checking for guest additions in VM...
==> default: Configuring and enabling network interfaces...
==> default: Mounting shared folders...
    default: /vagrant => /Users/rofrano/github/lab-vagrant
==> default: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> default: flag to force provisioning. Provisioners marked to run always will still run.
```

Creates the VM

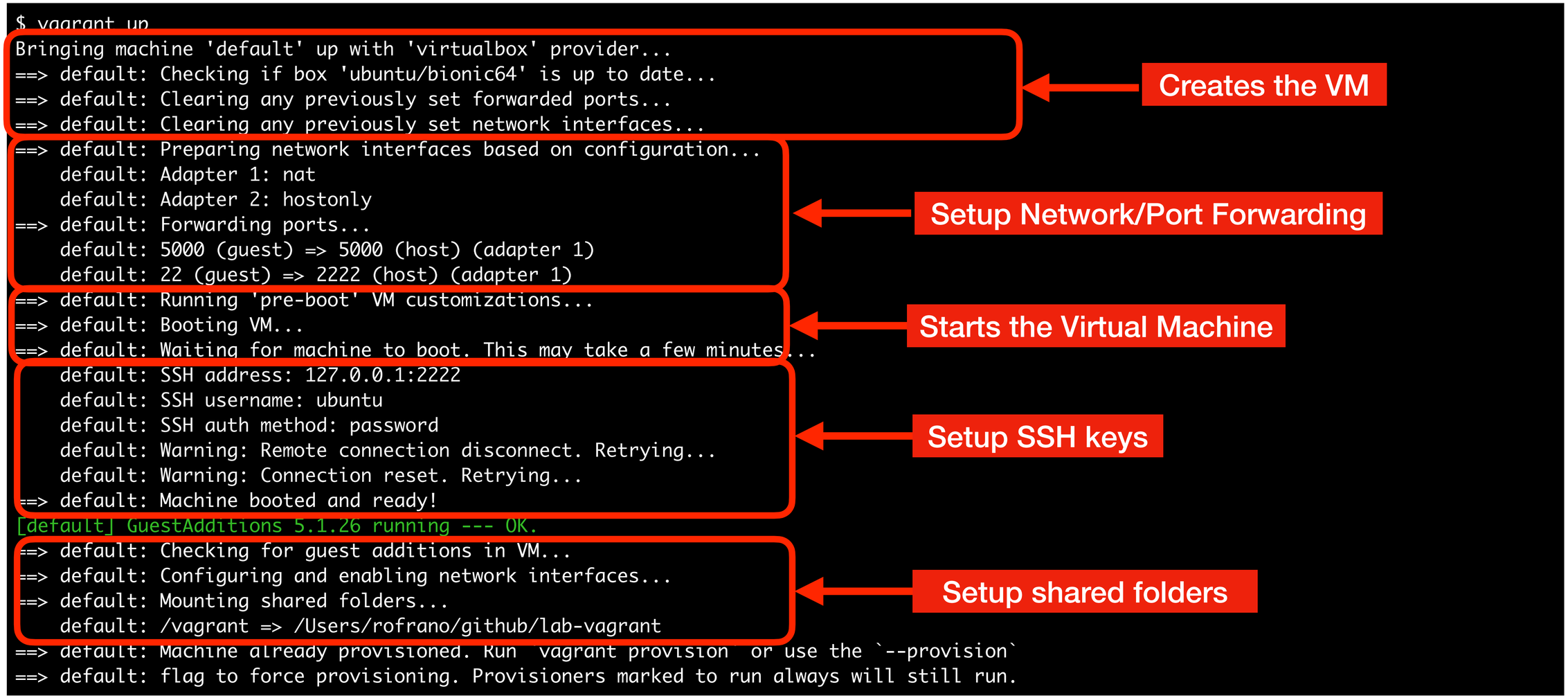
Setup Network/Port Forwarding

Starts the Virtual Machine

Initial Vagrant Run



Initial Vagrant Run



Where did Ubuntu/ Bionic64 Come From?

HashiCorp Vagrant Cloud | Search | Pricing | Vagrant | Help | Create an Account | Sign In

https://app.vagrantup.com/boxes/search

Discover Vagrant Boxes

Search for boxes by operating system, included software, architecture and more

Provider: any | virtualbox | vmware | libvirt | more

Sort by: Downloads | Recently Created | Recently Updated

ubuntu/trusty64 20180212.0.1 Official Ubuntu Server 14.04 LTS (Trusty Tahr) builds	virtualbox	Downloads **30,069,877**	Released **5 days ago**
laravel/homestead 5.1.0 Official Laravel local development box.	hyperv, parallels, virtualbox, vmware_desktop	Downloads **12,754,249**	Released **22 days ago**
hashicorp/precise64 1.1.0 A standard Ubuntu 12.04 LTS 64-bit box.	hyperv, virtualbox, vmware_fusion	Downloads **6,641,256**	Released **almost 4 years ago**
centos/7 1801.02 CentOS Linux 7 x86_64 Vagrant Box	hyperv, libvirt, virtualbox, vmware and 3 more providers	Downloads **4,659,035**	Released **about 1 month ago**
ubuntu/xenial64 20180224.0.0 Official Ubuntu 16.04 LTS (Xenial Xerus) Daily Build	virtualbox	Downloads **2,927,657**	Released **3 days ago**
puphpet/ubuntu1404-x64 20161102	parallels, virtualbox, vmware_desktop	Downloads **2,505,702**	Released **over 1 year ago**

Here are some Boxes from Bento*

HashiCorp Vagrant Cloud Search Pricing Vagrant Help Create an Account Sign In

https://app.vagrantup.com/bento

bento

These boxes are built using templates from the Chef's Bento project.
Should you find any bugs, please open an issue or send a pull request.
Thanks!

Boxes		
	bento/centos-8.1 v202002.04.0 Centos 8.1 Vagrant box created with Bento by Chef	last release 5 days ago 223 downloads
	bento/oracle-8 v202002.04.0 Oracle 8.x Vagrant box created with Bento by Chef. This box will be updated with the latest releases of Oracle 8 as they become available	last release 5 days ago 22 downloads
	bento/oracle-8.1 v202002.04.0 Oracle 8.1 Vagrant box created with Bento by Chef	last release 5 days ago 12 downloads
	bento/debian-10.2 v202002.04.0 Debian 10.2 Vagrant box created with Bento by Chef	last release 5 days ago 220 downloads

* Bento is a project that uses Packer templates for building minimal Vagrant baseboxes for multiple platforms http://chef.github.io/bento

What Other Boxes Are There?

http://www.vagrantbox.es/

Vagrantbox.es

Vagrant is an amazing tool for managing virtual machines via a simple to use command line interface. With a simple `vagrant up` you can be working in a clean environment based on a standard template.

These standard templates are called base boxes, and this website is simply a list of boxes people have been nice enough to make publicly available.

Suggest a Box

Do you know of another base box? Send a pull request and we'll add it to the list below.

Available Boxes

To use the available boxes just replace {title} and {url} with the information in the table below.

```
$ vagrant box add {title} {url}
$ vagrant init {title}
$ vagrant up
```

Search:

Name	Provider	URL	Size (MB)
Virtuozzo 7 x64 (Guest Additions 4.3.26)	VirtualBox	Copy https://atlas.hashicorp.com/OpenVZ/boxes/Virtuozzo-7b2	912
Debian Jessie 8.1.0 Release x64 (Minimal, Guest Additions 4.3.26)	VirtualBox	Copy https://atlas.hashicorp.com/ARTACK/boxes/debian-jessie	692
CentOS 7.0 x64 (Minimal, VirtualBox Guest Additions 4.3.28, Puppet 3.8.1 - see here for more infos)	VirtualBox	Copy https://github.com/tommy-muehle/puppet-vagrant-boxes/releases/download/1.1.0/centos-7.0-x86_64.box	475
CentOS 6.6 x64 (Minimal, VirtualBox Guest Additions, Puppet 3.7.5 - see here for more infos)	VirtualBox	Copy https://github.com/tommy-muehle/puppet-vagrant-boxes/releases/download/1.0.0/centos-6.6-x86_64.box	348
CentOS 7 x64 (Minimal, Shrinked, Guest Additions 4.3.26) (Monthly updates)	VirtualBox	Copy https://github.com/holms/vagrant-centos7-box/releases/download/7.1.1503.001/CentOS-7.1.1503-x86_64-netboot.box	437
CentOS 7.1 x64 (Minimal, Puppet 4.2.3, Guest Additions 4.3.30)[notes]	VirtualBox	Copy https://github.com/CommanderK5/packer-centos-template/releases/download/0.7.1/vagrant-centos-7.1.box	576
CentOS 6.7 x64 (Minimal, Puppet 4.2.3, Guest Additions 4.3.30)[notes]	VirtualBox	Copy https://github.com/CommanderK5/packer-centos-template/releases/download/0.6.7/vagrant-centos-6.7.box	577
Debian Jessie 8.0 Release x64 (Minimal, Shrinked, Guest Additions 4.3.26)	VirtualBox	Copy https://github.com/holms/vagrant-jessie-box/releases/download/Jessie-v0.1/Debian-jessie-amd64-netboot.box	437
Debian 8.1 x64 (LXC, Puppet 3.7.2, Guest Additions 4.3.28)	VirtualBox	Copy https://dl.dropboxusercontent.com/u/3523744/boxes/debian-8.1-amd64-lxc-puppet/debian-8.1-lxc-puppet.box	594
Debian Jessie 8.0 RC2 64-bit (Mininimal, Shrinked + Guest Additions 4.3.24 + Puppet 3.7.2)	VirtualBox	Copy http://static.gender-api.com/debian-8-jessie-rc2-x64-slim.box	328

What Other Boxes Are There?

http://www.vagrantbox.es/

Vagrantbox.es

Vagrant is an amazing tool for managing virtual machines via a simple to use command line interface. With a simple `vagrant up` you can be working in a clean environment based on a standard template.

These standard templates are called base boxes, and this website is simply a list of boxes people have been nice enough to make publicly available.

Suggest a Box

Do you know of another base box? Send a pull request and we'll add it to the list below.

Available Boxes

To use the available boxes just replace {title} and {url} with the information in

```
$ vagrant box add {title} {url}
$ vagrant init {title}
$ vagrant up
```

DO NOT manually download new boxes ! ! !
It is better to specify them in your Vagrantfile
so that all developers have the same boxes

Search:

Name	Provider	URL	Size (MB)
Virtuozzo 7 x64 (Guest Additions 4.3.26)	VirtualBox	Copy https://atlas.hashicorp.com/OpenVZ/boxes/Virtuozzo-7b2	912
Debian Jessie 8.1.0 Release x64 (Minimal, Guest Additions 4.3.26)	VirtualBox	Copy https://atlas.hashicorp.com/ARTACK/boxes/debian-jessie	692
CentOS 7.0 x64 (Minimal, VirtualBox Guest Additions 4.3.28, Puppet 3.8.1 - see here for more infos)	VirtualBox	Copy https://github.com/tommy-muehle/puppet-vagrant-boxes/releases/download/1.1.0/centos-7.0-x86_64.box	475
CentOS 6.6 x64 (Minimal, VirtualBox Guest Additions, Puppet 3.7.5 - see here for more infos)	VirtualBox	Copy https://github.com/tommy-muehle/puppet-vagrant-boxes/releases/download/1.0.0/centos-6.6-x86_64.box	348
CentOS 7 x64 (Minimal, Shrinked, Guest Additions 4.3.26) (Monthly updates)	VirtualBox	Copy https://github.com/holms/vagrant-centos7-box/releases/download/7.1.1503.001/CentOS-7.1.1503-x86_64-netboot.box	437
CentOS 7.1 x64 (Minimal, Puppet 4.2.3, Guest Additions 4.3.30)[notes]	VirtualBox	Copy https://github.com/CommanderK5/packer-centos-template/releases/download/0.7.1/vagrant-centos-7.1.box	576
CentOS 6.7 x64 (Minimal, Puppet 4.2.3, Guest Additions 4.3.30)[notes]	VirtualBox	Copy https://github.com/CommanderK5/packer-centos-template/releases/download/0.6.7/vagrant-centos-6.7.box	577
Debian Jessie 8.0 Release x64 (Minimal, Shrinked, Guest Additions 4.3.26)	VirtualBox	Copy https://github.com/holms/vagrant-jessie-box/releases/download/Jessie-v0.1/Debian-jessie-amd64-netboot.box	437
Debian 8.1 x64 (LXC, Puppet 3.7.2, Guest Additions 4.3.28)	VirtualBox	Copy https://dl.dropboxusercontent.com/u/3523744/boxes/debian-8.1-amd64-lxc-puppet/debian-8.1-lxc-puppet.box	594
Debian Jessie 8.0 RC2 64-bit (Mininimal, Shrinked + Guest Additions 4.3.24 + Puppet 3.7.2)	VirtualBox	Copy http://static.gender-api.com/debian-8-jessie-rc2-x64-slim.box	328

How To Use A Different Box?

- To use a different box, specify a name in `config.vm.box` and the url in `config.vm.box_url`

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

 config.vm.box = "centos64-x86_64"
 config.vm.box_url = "https://github.com/2creatives/vagrant-centos/releases/
download/v6.4.2/centos64-x86_64-20140116.box"
 config.vm.box_download_insecure = true

end
```

Accessing Your VM

- Use SSH client to access your VM just as if it were a remote server

```
$ vagrant ssh
```

- The `/vagrant` folder holds your current directory on the host OS

```
$ cd /vagrant
```

What is SSH?

- Secure Shell (SSH) is a Network protocol for one server to talk to another via a secure tunnel

$ ssh myaccount@172.166.14.20

ssh provides a remote login to a server

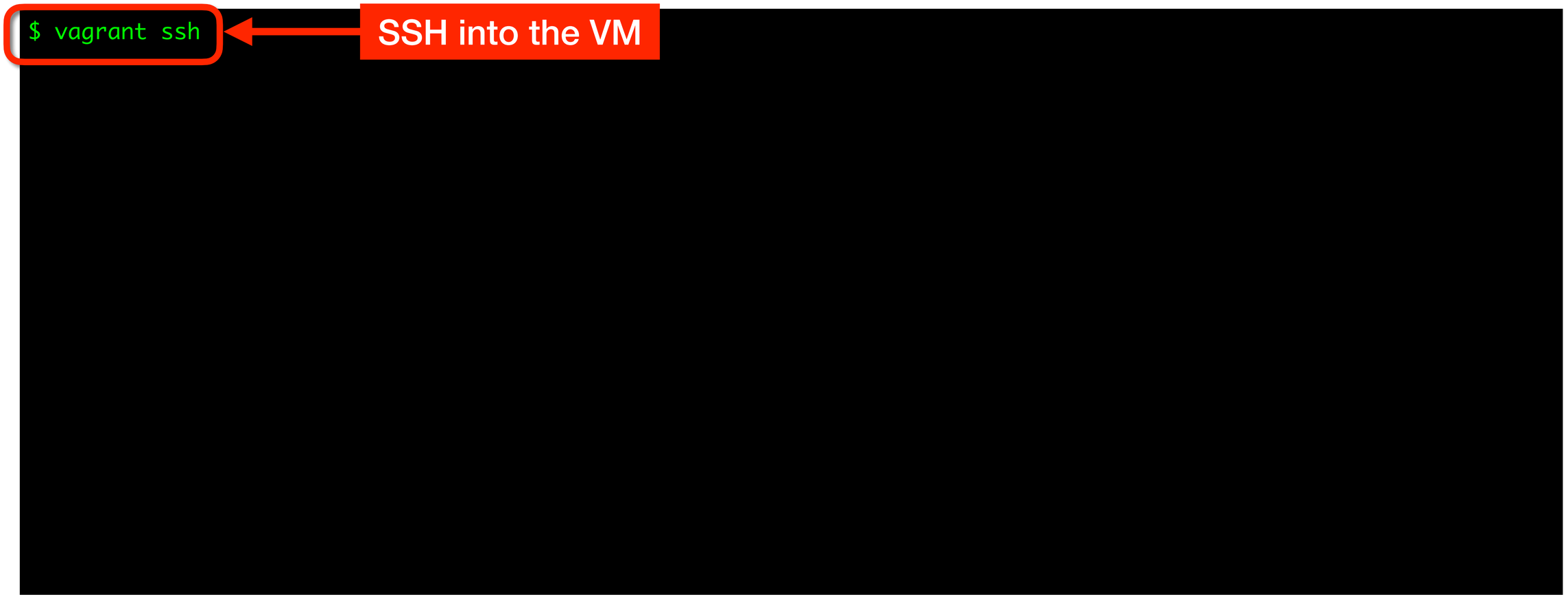
> _

172.166.14.20

Vagrant SSH

```
$ vagrant ssh
```

Vagrant SSH



Vagrant SSH

```
$ vagrant ssh

Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

2 packages can be updated.
0 updates are security updates.


Last login: Wed Jul  3 15:19:31 2019 from 10.0.2.2
ubuntu@ubuntu-xenial:~$
```

Vagrant SSH

```
$ vagrant ssh

Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

2 packages can be updated.
0 updates are security updates.


Last login: Wed Jul  3 15:19:31 2019 from 10.0.2.2
ubuntu@ubuntu-xenial:~$ cd /vagrant
ubuntu@ubuntu-xenial:/vagrant$ ls -l
```

Vagrant SSH

```
$ vagrant ssh

Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

2 packages can be updated.
0 updates are security updates.


Last login: Wed Jul  3 15:19:31 2019 from 10.0.2.2
ubuntu@ubuntu-xenial:~$ cd /vagrant
ubuntu@ubuntu-xenial:/vagrant$ ls -l
```

access to local filesystem via the `/vagrant` folder

Vagrant SSH

```
$ vagrant ssh

Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

2 packages can be updated.
0 updates are security updates.


Last login: Wed Jul  3 15:19:31 2019 from 10.0.2.2
ubuntu@ubuntu-xenial:~$ cd /vagrant
ubuntu@ubuntu-xenial:/vagrant$ ls -l
-rw-r--r-- 1 ubuntu ubuntu  4715 Sep  7 12:52 Vagrantfile
ubuntu@ubuntu-xenial:/vagrant$
```

Vagrant SSH

```
$ vagrant ssh

Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

2 packages can be updated.
0 updates are security updates.


Last login: Wed Jul  3 15:19:31 2019 from 10.0.2.2
ubuntu@ubuntu-xenial:~$ cd /vagrant
ubuntu@ubuntu-xenial:/vagrant$ ls -l
-rw-r--r-- 1 ubuntu ubuntu  4715 Sep  7 12:52 Vagrantfile
ubuntu@ubuntu-xenial:/vagrant$
```

This is your `Vagrantfile` from inside the VM

Vagrant file sharing

- Vagrant is sharing the host folder it was invoked from within the VM as `/vagrant`
- This means that you and edit files with your native Mac or Windows editor
- ...and still be able to run the code on your Linux VM

Edit the Vagrantfile

- We can install software using Chef, Ansible, Puppet, Salt, Shell, etc.
- Let’s use the “shell” provisioner to add apache2

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

  config.vm.box = "ubuntu/bionic64"

  config.vm.provision "shell", inline: <<-SHELL
    apt-get update
    apt-get install -y apache2
  SHELL

end
```

Edit the Vagrantfile

- We can install software using Chef, Ansible, Puppet, Salt, Shell, etc.
- Let’s use the “shell” provisioner to add apache2

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

  config.vm.box = "ubuntu/bionic64"

  config.vm.provision "shell", inline: <<-SHELL
    apt-get update
    apt-get install -y apache2
  SHELL

end
```

Uncomment this code
from the bottom of the Vagrantfile
It uses shell syntax for the
OS (Ubuntu in this case)

Re-provision the VM after Updates

- After making changes to a `Vagrantfile` in any of the provisioning sections, you must re-provision the VM
- Exit and run the new provision block

```
$ exit

$ vagrant provision
```

Testing the Web Server

- Use SSH client to access your VM and CURL to access Apache

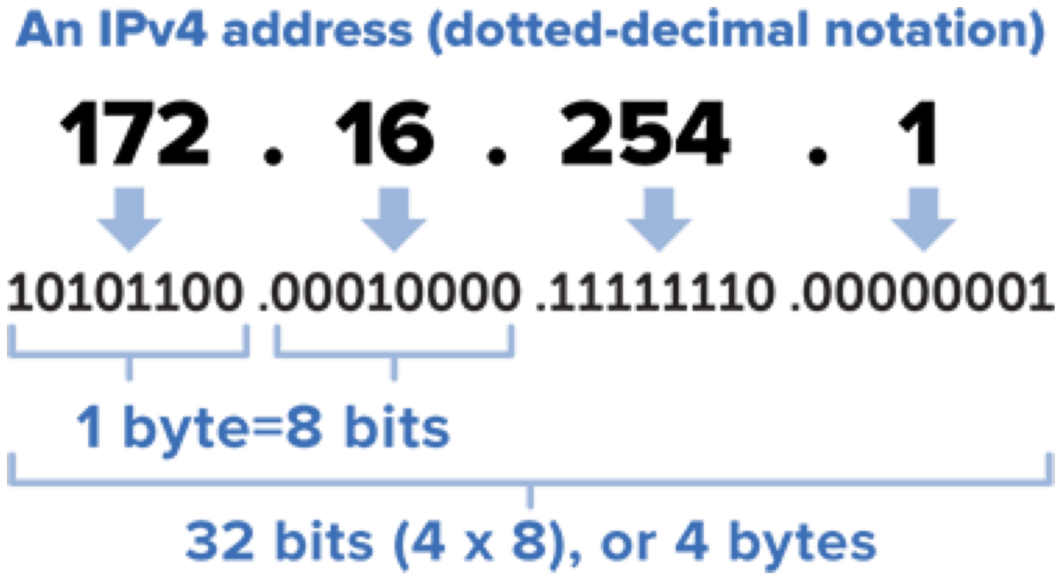
```
$ vagrant ssh
$ curl http://127.0.0.1
$ exit
```

- You should see HTML returned
- But if you access it from your browser as `127.0.0.1` it will fail! Why?

Let's fix this by adding an IP address and Forwarding the Port!

What is an IP Address?

- An IP address is like a street address for a server
- An IP Address has two parts:
  - Network ID (street)
  - Host ID (house number)
- `192.168.33.10` is server `10` on subnet 33 of subdomain 168 of primary domain 192



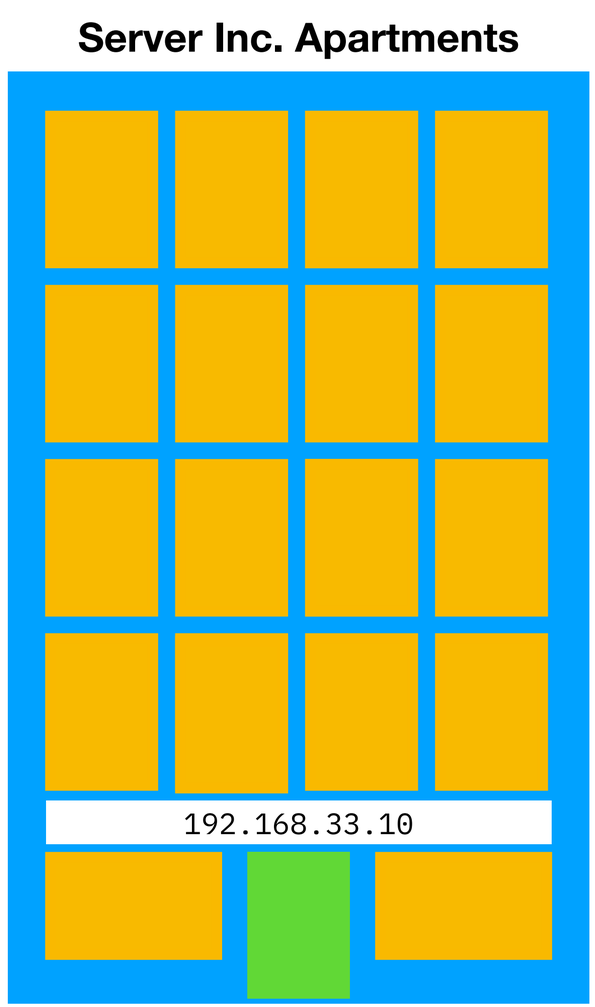
Some special IP Address

- Some IP addresses are reserved with special meaning:
  - `10.0.0.1` to `10.255.255.255`
    is a private non-routable* network with 16,777,215 addresses
  - `192.168.0.1` to `192.168.255.255`
    is another private non-routable* network with 65,535 addresses
  - `127.0.0.1` is a local loopback adapter address otherwise known as `localhost`.

* Non-routable means that network routers will not forward to these addresses

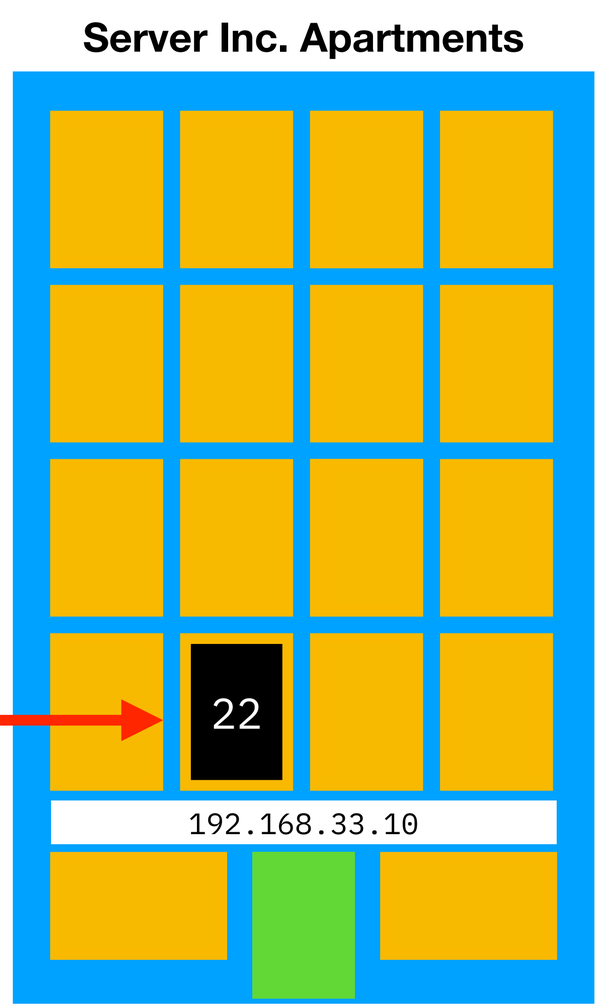
What is a Port?

- If an IP address is like a street address for a server (i.e., it gets you to the building where people live)
- Then a PORT is like an apartment number with a door or window that is either open or closed



What is a Port?

- If an IP address is like a street address for a server (i.e., it gets you to the building where people live)
- Then a PORT is like an apartment number with a door or window that is either open or closed



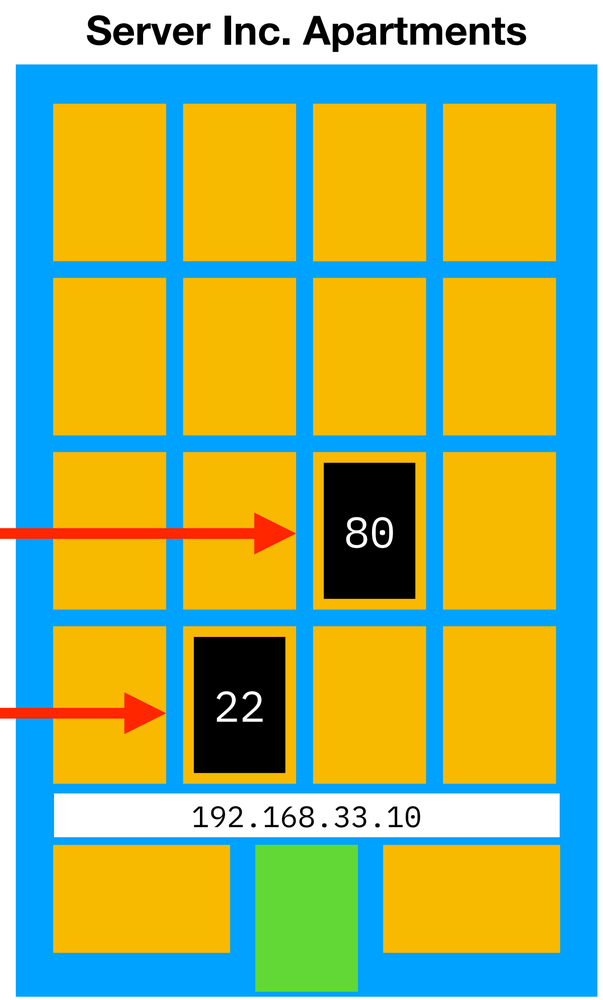
Open port 22 for ssh (`192.168.33.10:22`)

What is a Port?

- If an IP address is like a street address for a server (i.e., it gets you to the building where people live)
- Then a PORT is like an apartment number with a door or window that is either open or closed

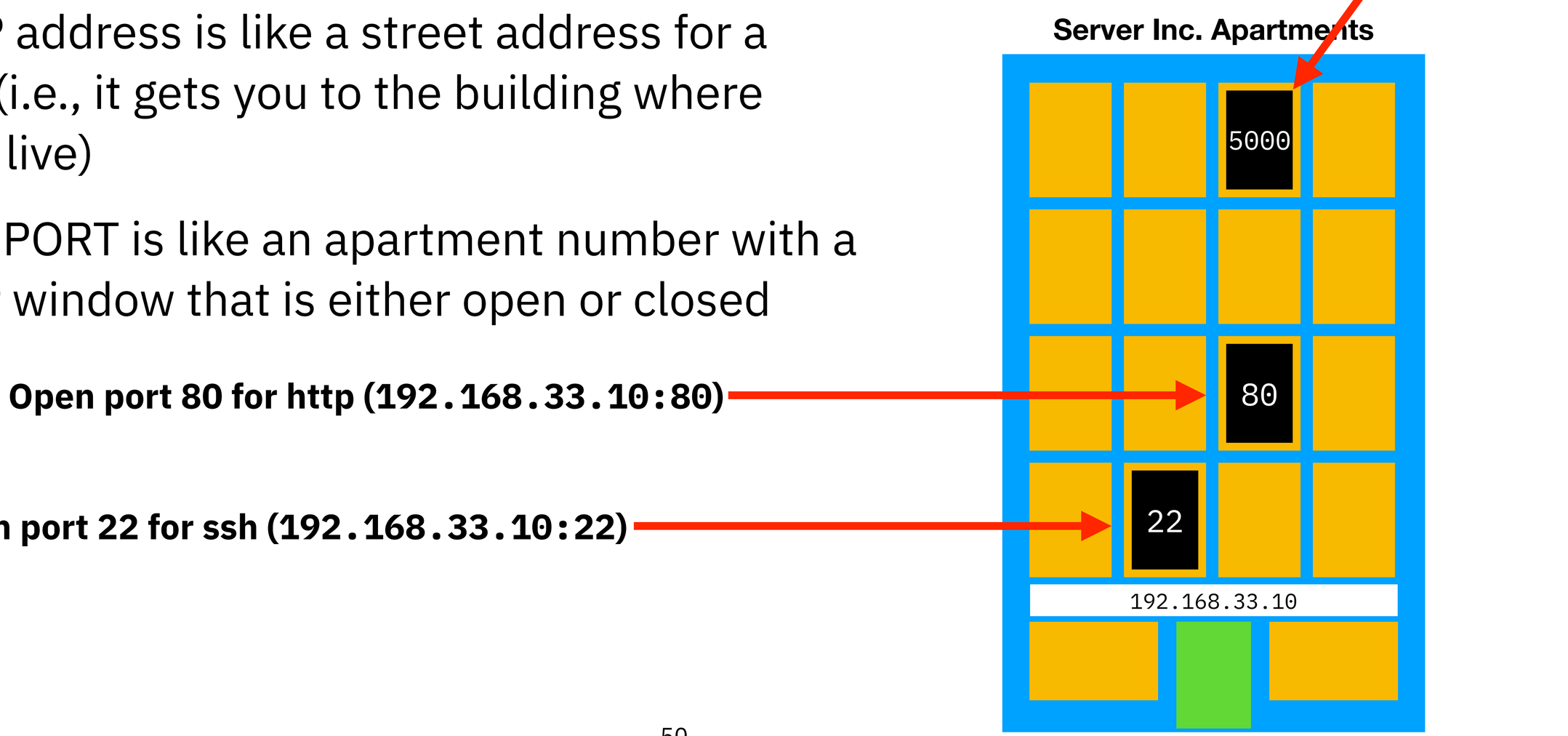
Open port 80 for http (192.168.33.10:80)

Open port 22 for ssh (192.168.33.10:22)



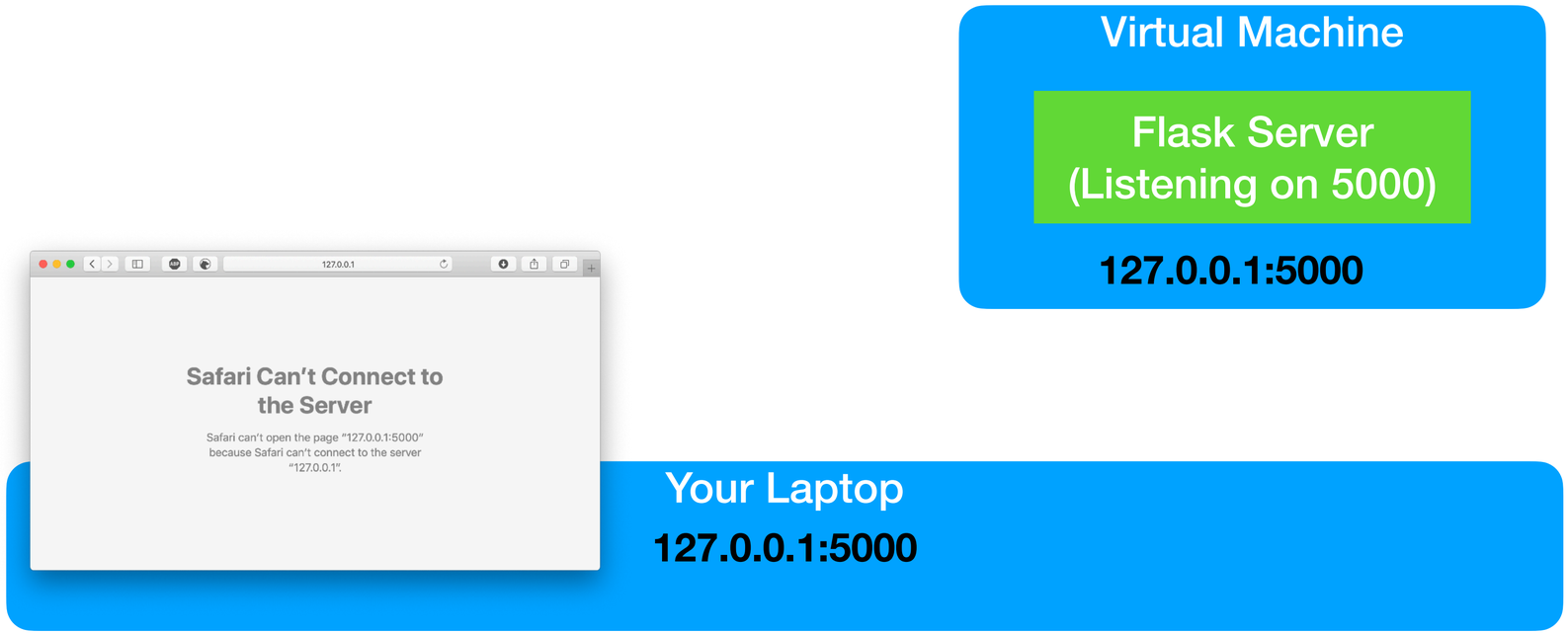
What is a Port?

- If an IP address is like a street address for a server (i.e., it gets you to the building where people live)
- Then a PORT is like an apartment number with a door or window that is either open or closed



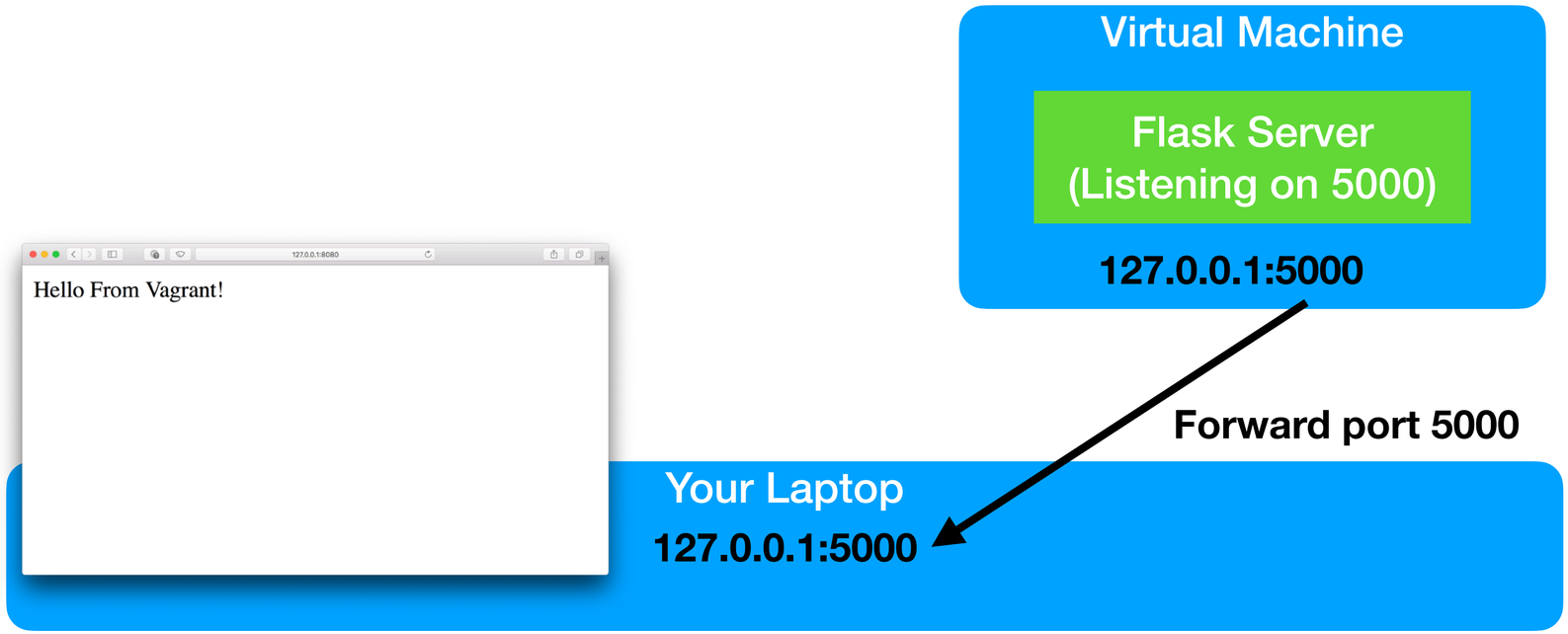
What is Port Forwarding?

- Port Forwarding is just like mail forwarding
- When a message gets sent to a particular port it gets forwarded to another port usually on another server, just like a letter would in the mail



What is Port Forwarding?

- Port Forwarding is just like mail forwarding
- When a message gets sent to a particular port it gets forwarded to another port usually on another server, just like a letter would in the mail



Let’s Add an IP address and Forward The Web Port

- We must forward the ports that we want to access from outside the VM. Let’s add a private IP Address for direct access too:

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

  config.vm.box = “ubuntu/bionic64"

  # accessing "localhost:8080" will access port 80 on the guest machine.
  config.vm.network "forwarded_port", guest: 80, host: 8080

  config.vm.network "private_network", ip: "192.168.33.10"

  config.vm.provision "shell", inline: <<-SHELL
    sudo apt-get update
    sudo apt-get install -y apache2
  SHELL

end
```

Let’s Add an IP address and Forward The Web Port

- We must forward the ports that we want to access from outside the VM. Let’s add a private IP Address for direct access too:

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

  config.vm.box = “ubuntu/bionic64"

  # accessing "localhost:8080" will access port 80 on the guest machine.
  config.vm.network "forwarded_port", guest: 80, host: 8080

  config.vm.network "private_network", ip: "192.168.33.10"

  config.vm.provision "shell", inline: <<-SHELL
    sudo apt-get update
    sudo apt-get install -y apache2
  SHELL

end
```

We should use ports higher than 1024 if we are not root

Reload the VM after Updates

- Since we didn't change any of the provisioning blocks in the `Vagrantfile`, we can just exit and reload the VM and the port forwarding will be picked up

```
$ vagrant reload
```

New Vagrant Run

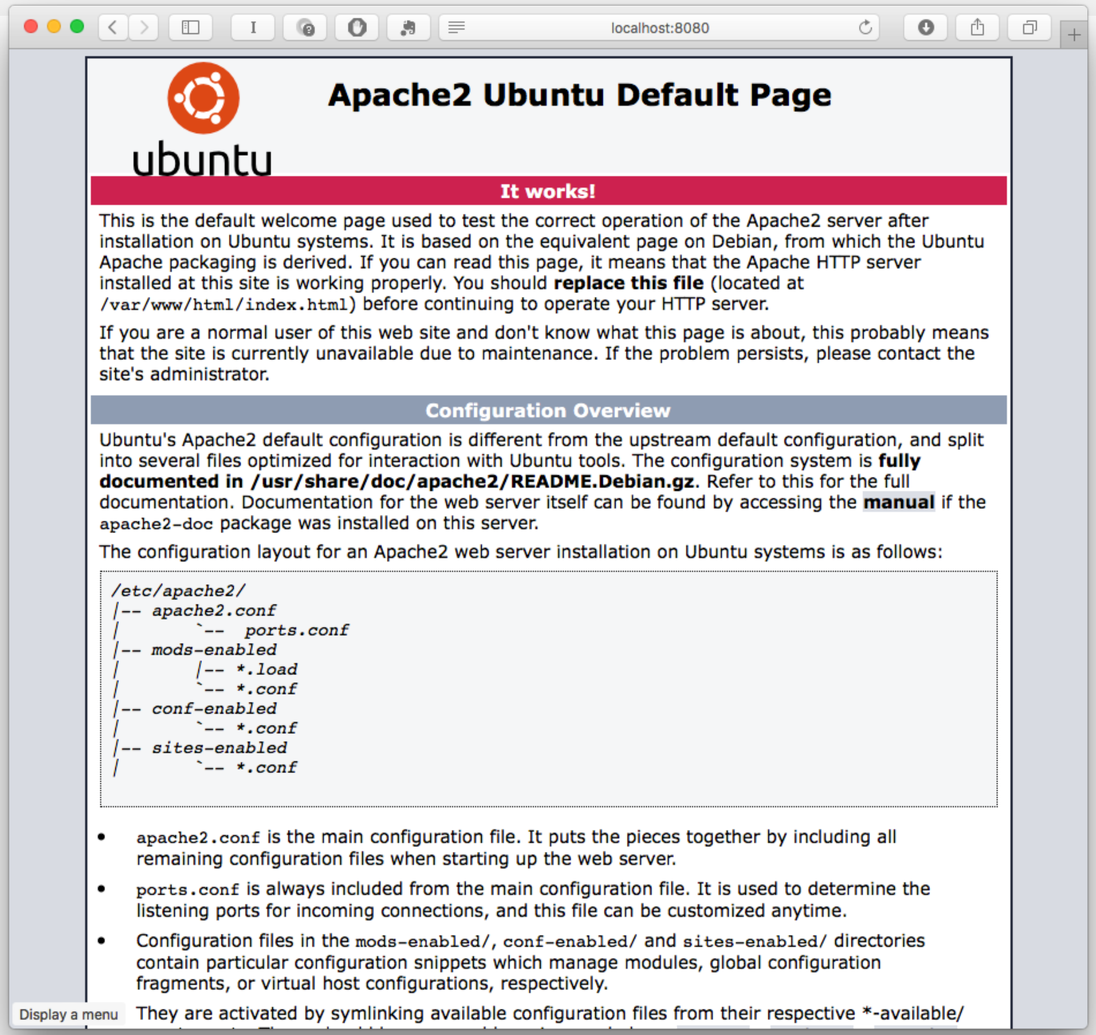
```
iotia:lab-vagrant-demo rofrano$ vagrant reload
==> default: Attempting graceful shutdown of VM...
==> default: Checking if box 'ubuntu/bionic64' is up to date...
==> default: Clearing any previously set forwarded ports...
==> default: Couldn't find Cheffile at ./Cheffile.
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
    default: Adapter 2: hostonly
==> default: Forwarding ports...
    default: 80 (guest) => 8080 (host) (adapter 1)
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: ubuntu
    default: SSH auth method: password
    default: Warning: Remote connection disconnect. Retrying...
    default: Warning: Remote connection disconnect. Retrying...
==> default: Machine booted and ready!
GuestAdditions 5.0.20 running --- OK.
==> default: Checking for guest additions in VM...
==> default: Configuring and enabling network interfaces...
==> default: Mounting shared folders...
    default: /vagrant => /Users/rofrano/github/Demo/lab-vagrant-demo
==> default: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> default: flag to force provisioning. Provisioners marked to run always will still run.
iotia:lab-vagrant-demo rofrano$
```

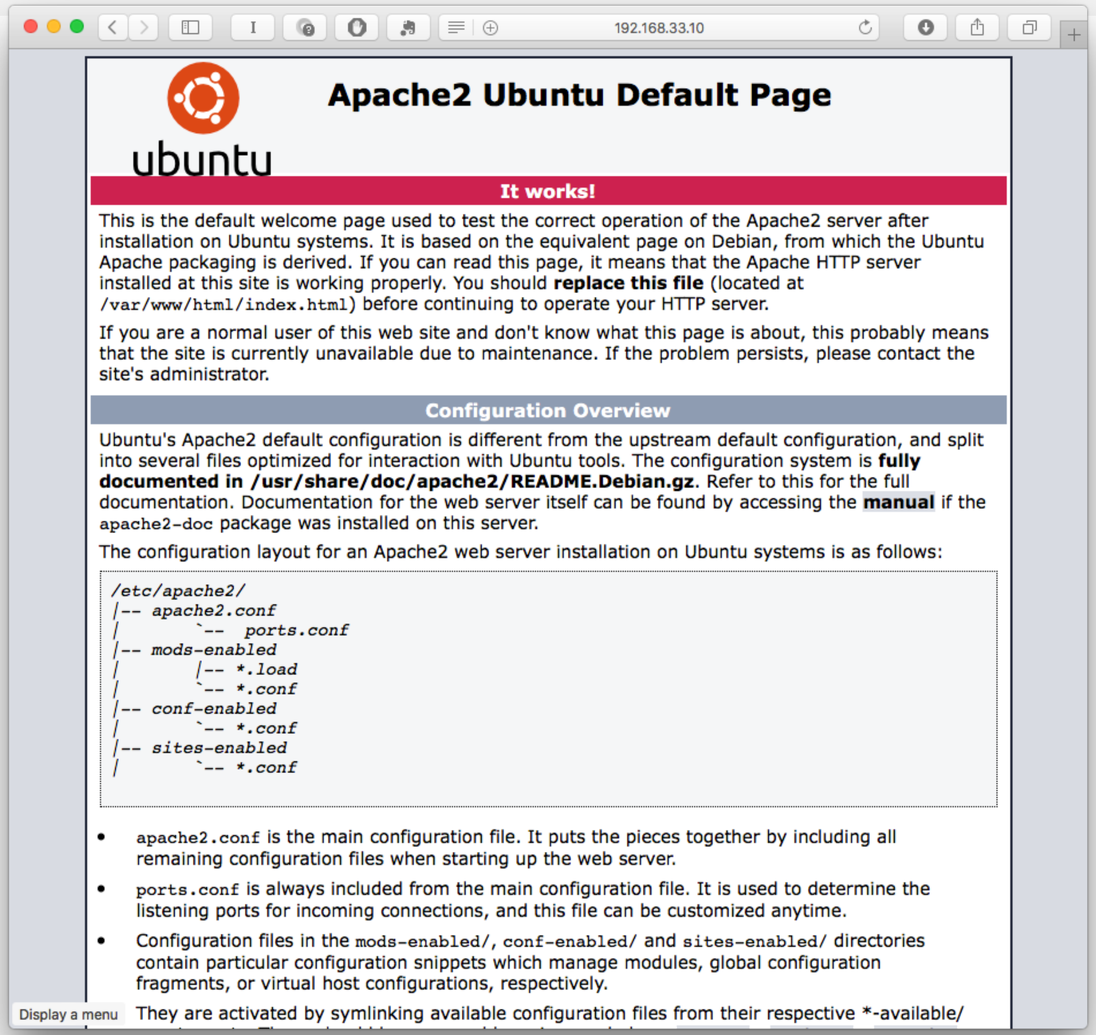
New Vagrant Run

```
iotia:lab-vagrant-demo rofrano$ vagrant reload
==> default: Attempting graceful shutdown of VM...
==> default: Checking if box 'ubuntu/bionic64' is up to date...
==> default: Clearing any previously set forwarded ports...
==> default: Couldn't find Cheffile at ./Cheffile.
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
    default: Adapter 2: hostonly
==> default: Forwarding ports...
    default: 80 (guest) => 8080 (host) (adapter 1)
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: ubuntu
    default: SSH auth method: password
    default: Warning: Remote connection disconnect. Retrying...
    default: Warning: Remote connection disconnect. Retrying...
==> default: Machine booted and ready!
GuestAdditions 5.0.20 running --- OK.
==> default: Checking for guest additions in VM...
==> default: Configuring and enabling network interfaces...
==> default: Mounting shared folders...
    default: /vagrant => /Users/rofrano/github/Demo/lab-vagrant-demo
==> default: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> default: flag to force provisioning. Provisioners marked to run always will still run.
iotia:lab-vagrant-demo rofrano$
```

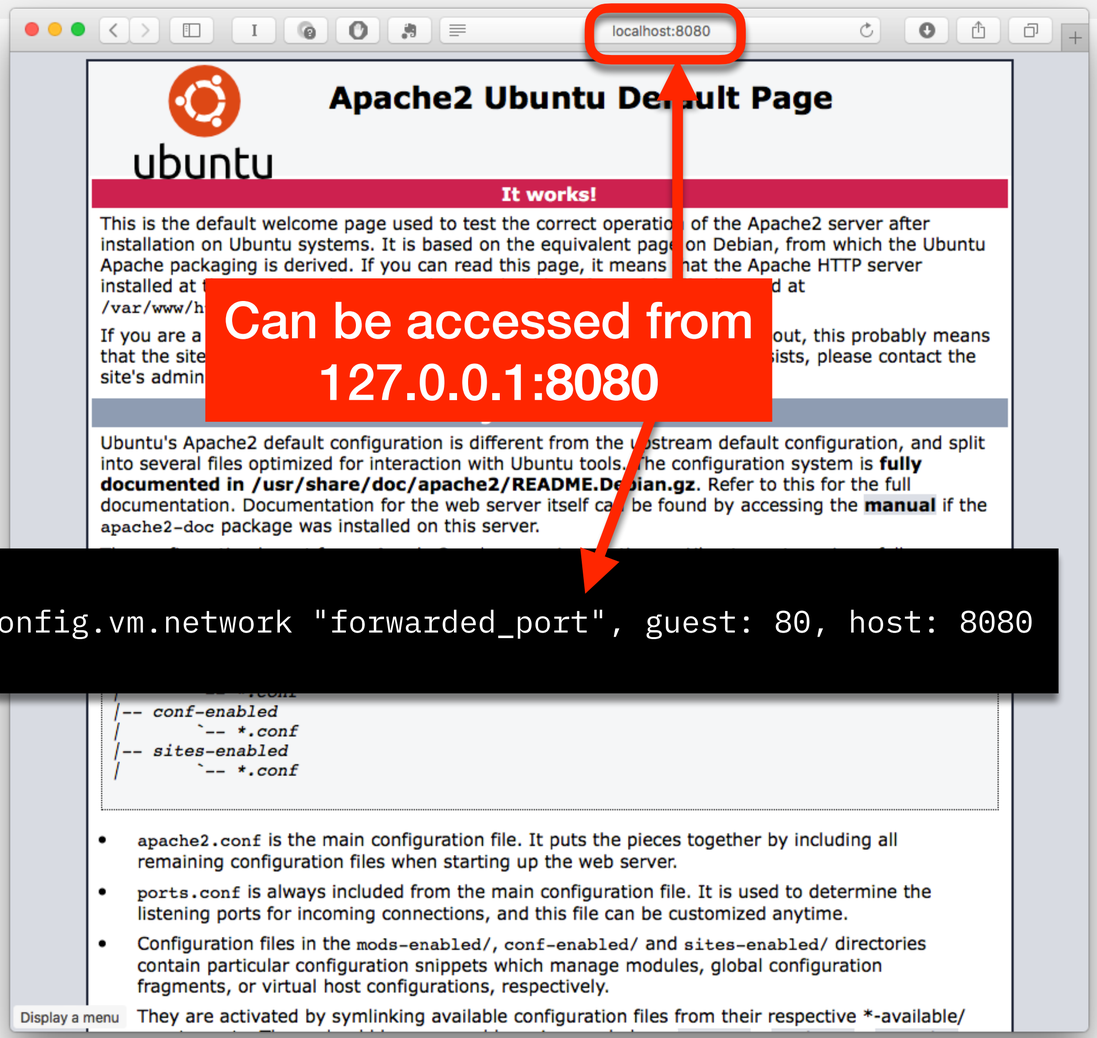
Our port got forwarded from 80 inside the VM to 8080 outside the VM

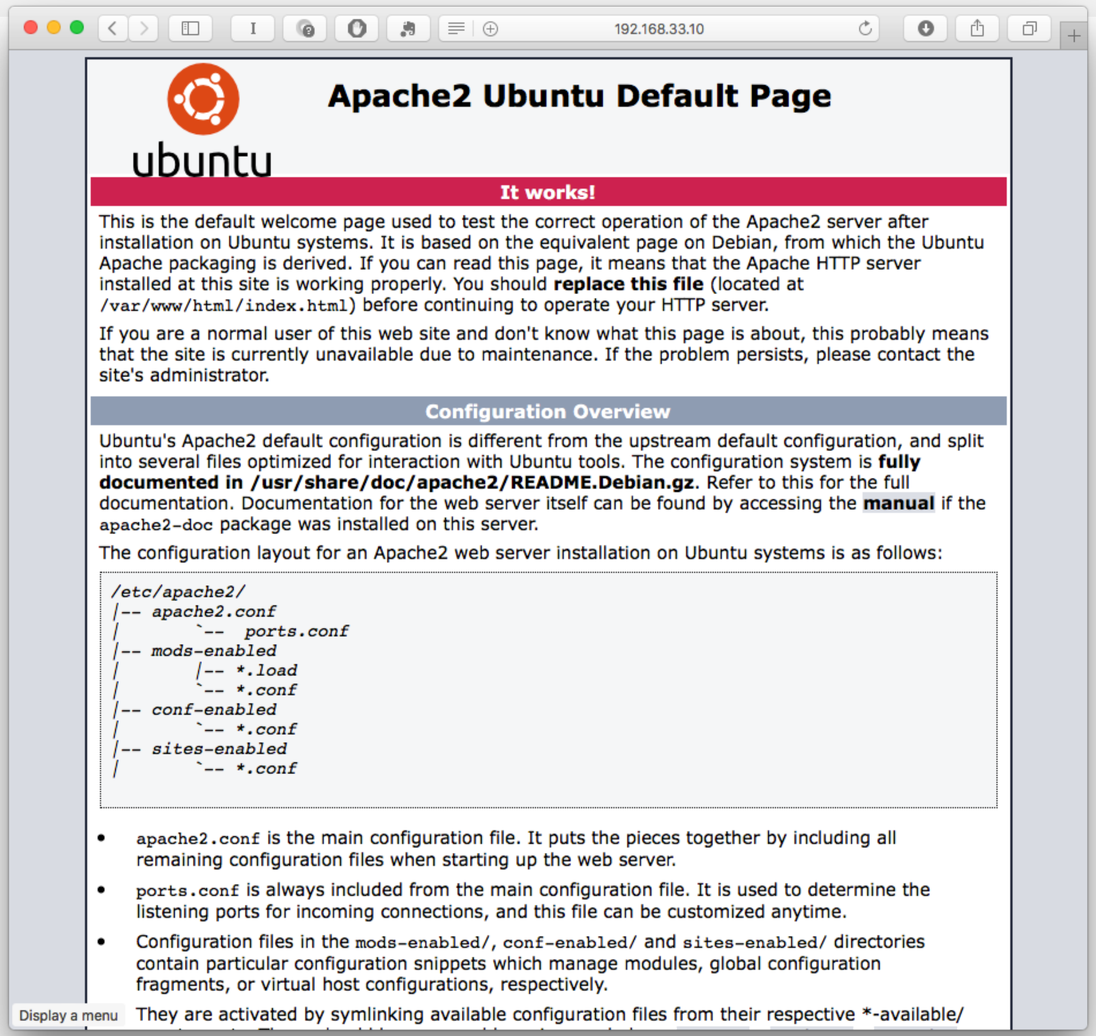
Test It In Your Browser



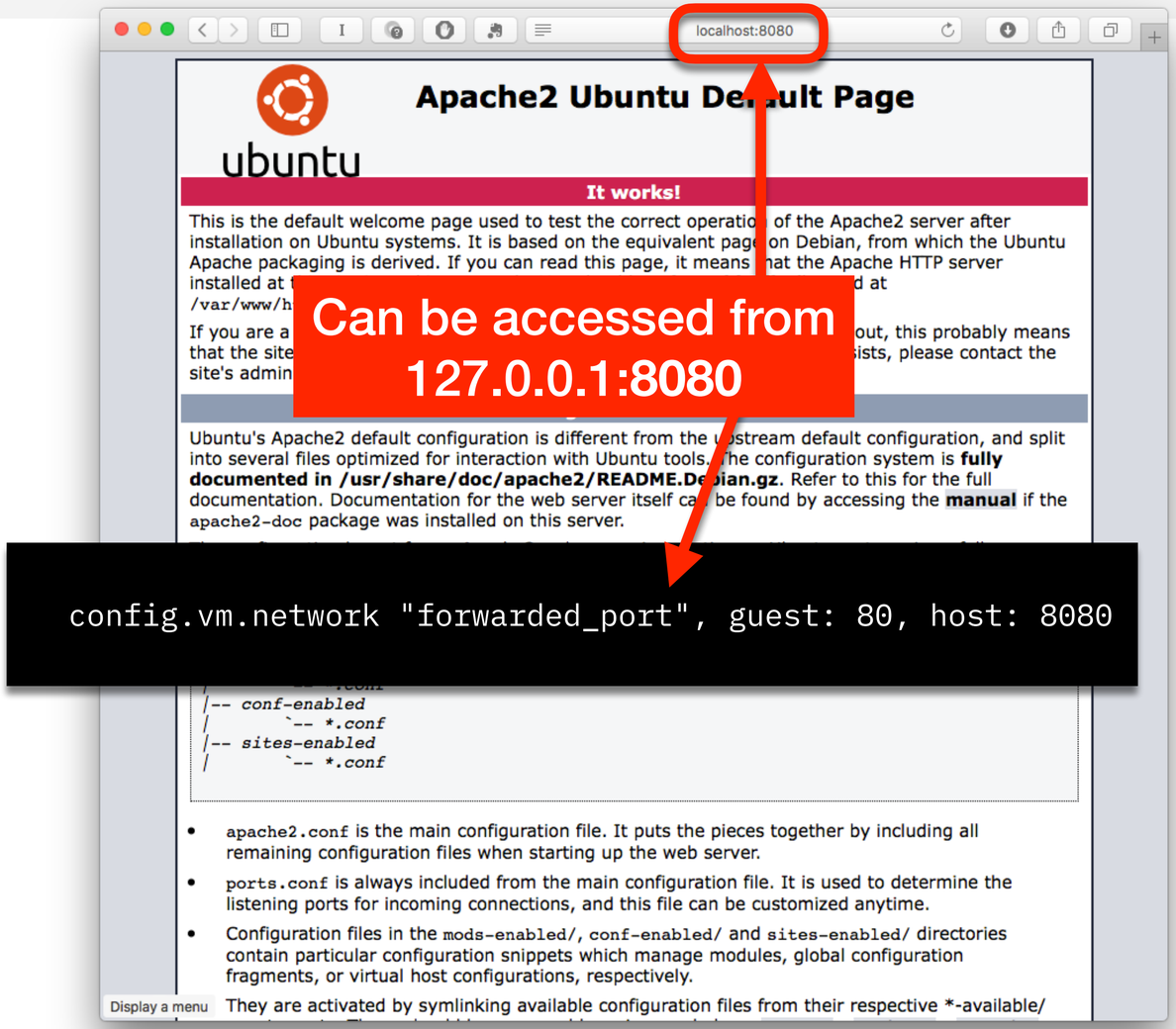


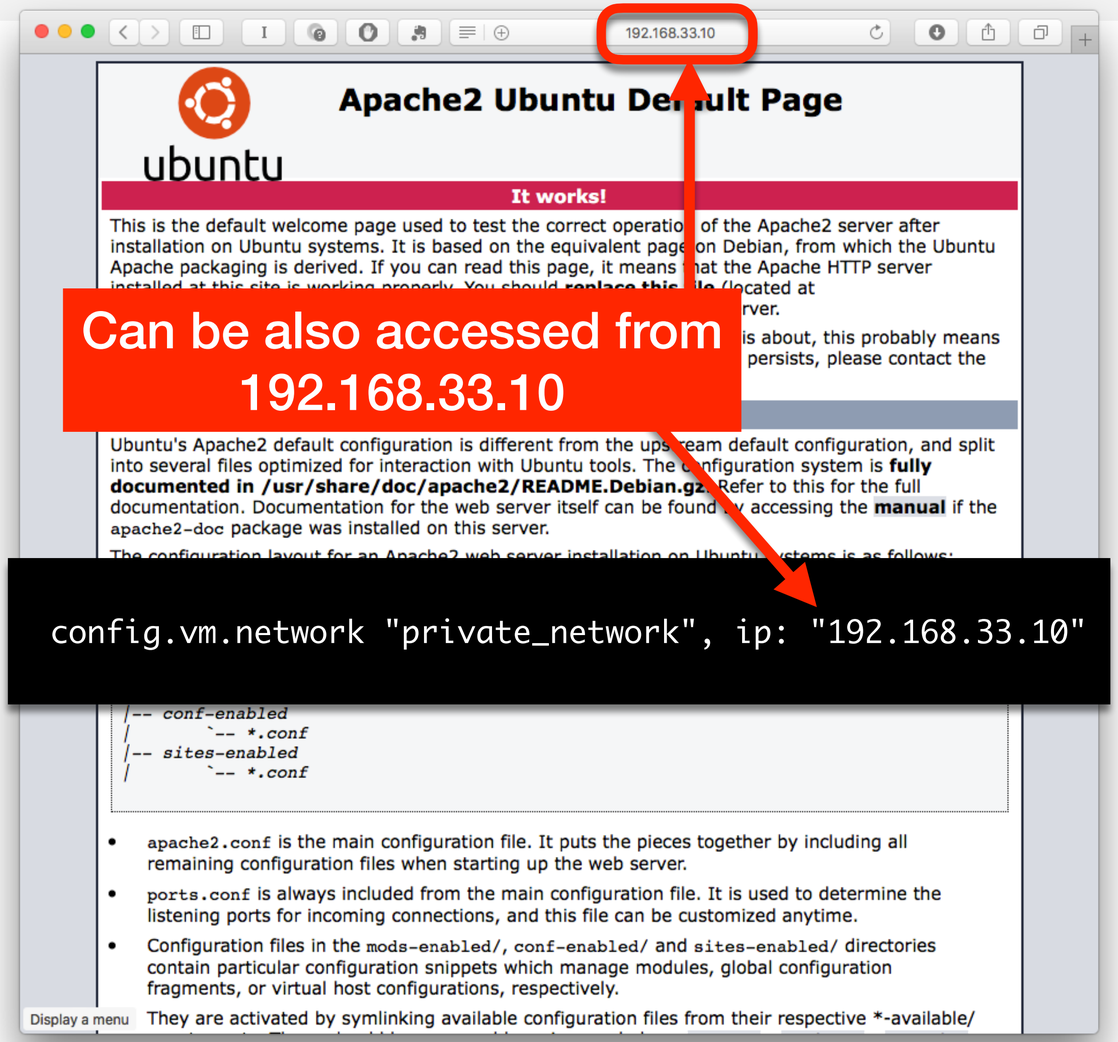
Test It In Your Browser





Test It In Your Browser





More Port Forwarding Examples

```
# Forward RAILS port
config.vm.network :forwarded_port, guest: 3000, host: 3000, auto_correct: true

# Forward Python Flask port
config.vm.network :forwarded_port, guest: 5000, host: 5000, auto_correct: true

# Forward PostgreSQL port
config.vm.network :forwarded_port, guest: 5432, host: 5432, auto_correct: true

# Forward MySQL port
config.vm.network :forwarded_port, guest: 3306, host: 3306, auto_correct: true

# Forward Redis port
config.vm.network :forwarded_port, guest: 6379, host: 6379, auto_correct: true

# Forward RabbitMQ Ports
config.vm.network :forwarded_port, guest: 15672, host: 15672, auto_correct: true
config.vm.network :forwarded_port, guest: 4369,  host: 4369,  auto_correct: true
config.vm.network :forwarded_port, guest: 5672,  host: 5672,  auto_correct: true
```

Adding Your Own Content

- We can link a local folder to the html root to serve up our own content (remember our local folder is shared with the VM!)
- First lets create a new `./html` folder and add `index.html`:

```
$ mkdir html
$ cd html
$ echo 'Hello From Vagrant!' > index.html
```

Link Our HTML Folder

- We can point `/var/www/html` to our host folder `/vagrant/`

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

  config.vm.box = “ubuntu/xenial64"

  # accessing "localhost:8080" will access port 80 on the guest machine.
  config.vm.network "forwarded_port", guest: 80, host: 8080

  config.vm.network "private_network", ip: "192.168.33.10"

  config.vm.provision "shell", inline: <<-SHELL
    apt-get update
    apt-get install -y apache2
    rm -rf /var/www/html
    ln -s /vagrant/html /var/www/html
  SHELL

end
```

Link Our HTML Folder

- We can point `/var/www/html` to our host folder `/vagrant/`

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

  config.vm.box = “ubuntu/xenial64"

  # accessing "localhost:8080" will access port 80 on the guest machine.
  config.vm.network "forwarded_port", guest: 80, host: 8080

  config.vm.network "private_network", ip: "192.168.33.10"

  config.vm.provision "shell", inline: <<-SHELL
    apt-get update
    apt-get install -y apache2
    rm -rf /var/www/html
    ln -s /vagrant/html /var/www/html
  SHELL

end
```

Remove the /var/www/html folder and link to our host’s ./html folder

Add 'rm' and 'ln' Commands

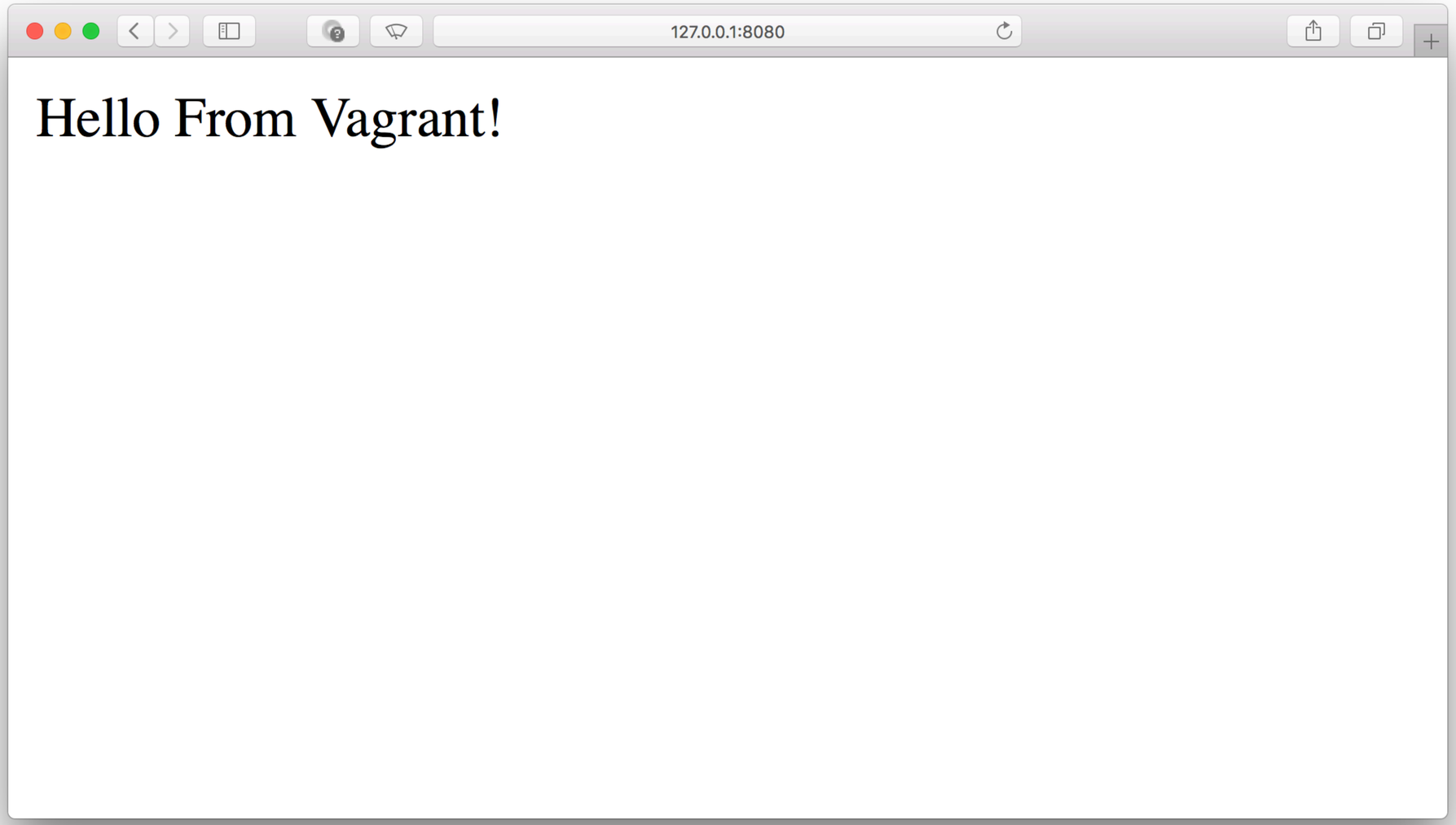
```
config.vm.provision "shell", inline: <<-SHELL
    apt-get update
    apt-get install -y apache2
    rm -rf /var/www/html
    ln -s /vagrant/html /var/www/html
SHELL
```

Re-provision the VM after Updates

- Since we changed a provision block in the `Vagrantfile` we must reload and provision

```
$ vagrant reload --provision
```

Apache is Serving Our Page



What Just Happened?

- We are serving up local files from our laptop inside of the VM
- That means we can develop locally with our OS IDE tools
- And we can run the code in a Linux VM just like a production server!
- Very Cool ! ! ! 😎

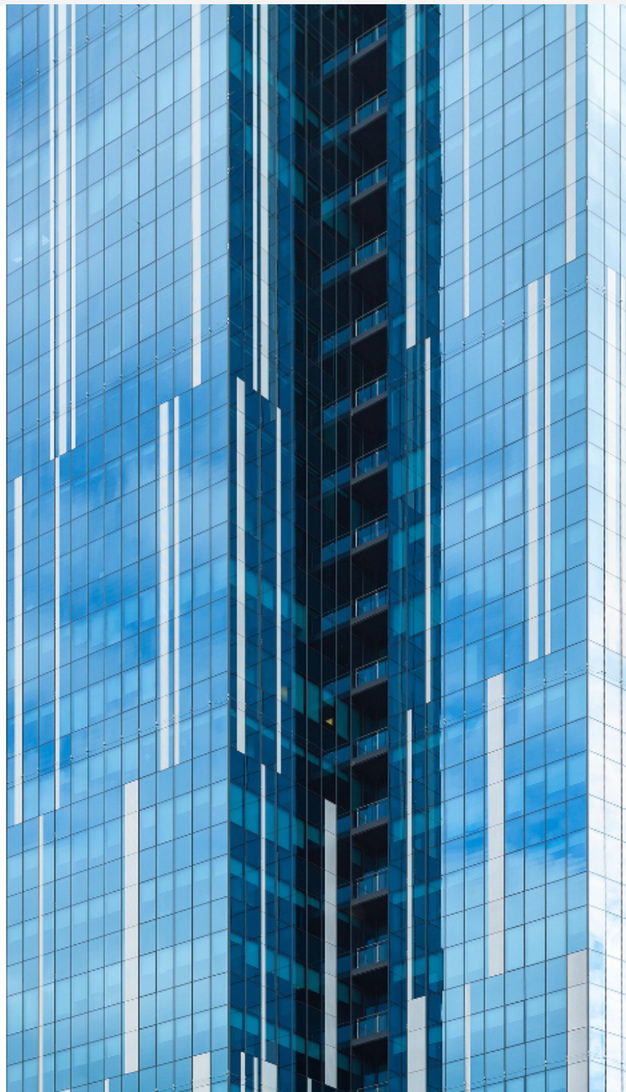
Get Ready for New Demo

- You can now shutdown the vagrant VM and get out of that folder

```
$ vagrant halt
$ cd ..
```

Lets Add Docker To The Mix

- We want to create an environment to run a Python / Flask microservice that uses a Redis database to store a hit counter
- We will use Docker to run Redis without having to install anything
- Vagrant supports a Docker provisioner that we can leverage



Clone the Project from GitHub

- You are a new developer to the project and you want to get up and running quickly so that you can start developing:

`https://github.com/rofrano/lab-vagrant.git`

```
$ git clone git@github.com:rofrano/lab-vagrant.git
$ cd lab-vagrant
$ vagrant up
```

Complete Vagrantfile

```
Vagrant.configure(2) do |config|
  config.vm.box = "ubuntu/bionic64"
  config.vm.network "forwarded_port", guest: 5000, host: 5000
  config.vm.network "private_network", ip: "192.168.33.10"

  config.vm.provider "virtualbox" do |vb|
    vb.memory = "1024"
    vb.cpus = 1
  end

  config.vm.provision "shell", inline: <<-SHELL
    sudo apt-get update
    sudo apt-get install -y git python-pip python-dev build-essential
    sudo apt-get -y autoremove
    # Install app dependencies
    cd /vagrant
    sudo pip install -r requirements.txt
    # Prepare Redis data share
    sudo mkdir -p /var/lib/redis/data
    sudo chown vagrant:vagrant /var/lib/redis/data
    # Make vi look nice
    echo "colorscheme desert" > ~/.vimrc
  SHELL

  # Add Redis docker container
  config.vm.provision "docker" do |d|
    d.pull_images "redis:alpine"
    d.run "redis:alpine",
      args: "--restart=always -d --name redis -h redis -p 6379:6379 -v /var/lib/redis/data:/data"
  end
end
```

Complete Vagrantfile

```
Vagrant.configure(2) do |config|
  config.vm.box = "ubuntu/bionic64"
  config.vm.network "forwarded_port", guest: 5000, host: 5000
  config.vm.network "private_network", ip: "192.168.33.10"

  config.vm.provider "virtualbox" do |vb|
    vb.memory = "1024"
    vb.cpus = 1
  end

  config.vm.provision "shell", inline: <<-SHELL
    sudo apt-get update
    sudo apt-get install -y git python-pip python-dev build-essential
    sudo apt-get -y autoremove
    # Install app dependencies
    cd /vagrant
    sudo pip install -r requirements.txt
    # Prepare Redis data share
    sudo mkdir -p /var/lib/redis/data
    sudo chown vagrant:vagrant /var/lib/redis/data
    # Make vi look nice
    echo "colorscheme desert" > ~/.vimrc
  SHELL

  # Add Redis docker container
  config.vm.provision "docker" do |d|
    d.pull_images "redis:alpine"
    d.run "redis:alpine",
      args: "--restart=always -d --name redis -h redis -p 6379:6379 -v /var/lib/redis/data:/data"
  end
end
```

Create an Ubuntu VM and Forward the ports you need

Complete Vagrantfile

```
Vagrant.configure(2) do |config|
  config.vm.box = "ubuntu/bionic64"
  config.vm.network "forwarded_port", guest: 5000, host: 5000
  config.vm.network "private_network", ip: "192.168.33.10"

  config.vm.provider "virtualbox" do |vb|
    vb.memory = "1024"
    vb.cpus = 1
  end

  config.vm.provision "shell", inline: <<-SHELL
    sudo apt-get update
    sudo apt-get install -y git python-pip python-dev build-essential
    sudo apt-get -y autoremove
    # Install app dependencies
    cd /vagrant
    sudo pip install -r requirements.txt
    # Prepare Redis data share
    sudo mkdir -p /var/lib/redis/data
    sudo chown vagrant:vagrant /var/lib/redis/data
    # Make vi look nice
    echo "colorscheme desert" > ~/.vimrc
  SHELL

  # Add Redis docker container
  config.vm.provision "docker" do |d|
    d.pull_images "redis:alpine"
    d.run "redis:alpine",
      args: "--restart=always -d --name redis -h redis -p 6379:6379 -v /var/lib/redis/data:/data"
  end
end
```

Create an Ubuntu VM and Forward the ports you need

Control the VM resources

Complete Vagrantfile

```
Vagrant.configure(2) do |config|
  config.vm.box = "ubuntu/bionic64"
  config.vm.network "forwarded_port", guest: 5000, host: 5000
  config.vm.network "private_network", ip: "192.168.33.10"

  config.vm.provider "virtualbox" do |vb|
    vb.memory = "1024"
    vb.cpus = 1
  end

  config.vm.provision "shell", inline: <<-SHELL
    sudo apt-get update
    sudo apt-get install -y git python-pip python-dev build-essential
    sudo apt-get -y autoremove
    # Install app dependencies
    cd /vagrant
    sudo pip install -r requirements.txt
    # Prepare Redis data share
    sudo mkdir -p /var/lib/redis/data
    sudo chown vagrant:vagrant /var/lib/redis/data
    # Make vi look nice
    echo "colorscheme desert" > ~/.vimrc
  SHELL

  # Add Redis docker container
  config.vm.provision "docker" do |d|
    d.pull_images "redis:alpine"
    d.run "redis:alpine",
      args: "--restart=always -d --name redis -h redis -p 6379:6379 -v /var/lib/redis/data:/data"
  end
end
```

Create an Ubuntu VM and Forward the ports you need

Control the VM resources

Provision the development environment

Complete Vagrantfile

```
Vagrant.configure(2) do |config|
  config.vm.box = "ubuntu/bionic64"
  config.vm.network "forwarded_port", guest: 5000, host: 5000
  config.vm.network "private_network", ip: "192.168.33.10"

  config.vm.provider "virtualbox" do |vb|
    vb.memory = "1024"
    vb.cpus = 1
  end

  config.vm.provision "shell", inline: <<-SHELL
    sudo apt-get update
    sudo apt-get install -y git python-pip python-dev build-essential
    sudo apt-get -y autoremove
    # Install app dependencies
    cd /vagrant
    sudo pip install -r requirements.txt
    # Prepare Redis data share
    sudo mkdir -p /var/lib/redis/data
    sudo chown vagrant:vagrant /var/lib/redis/data
    # Make vi look nice
    echo "colorscheme desert" > ~/.vimrc
  SHELL

  # Add Redis docker container
  config.vm.provision "docker" do |d|
    d.pull_images "redis:alpine"
    d.run "redis:alpine",
      args: "--restart=always -d --name redis -h redis -p 6379:6379 -v /var/lib/redis/data:/data"
  end
end
```

Create an Ubuntu VM and Forward the ports you need

Control the VM resources

Provision the development environment

Add Docker Containers for middleware

Application Code

app.py

```python
import os
import redis
from flask import Flask, jsonify

port = os.getenv('PORT', '5000')
hostname = os.getenv('HOSTNAME', '127.0.0.1')
redis_port = os.getenv('REDIS_PORT', '6379')

app = Flask(__name__)

# Application Routes
@app.route('/')
def index():
    return jsonify(name='Hit Me Service', version='1.0', url='/hits'), 200

@app.route('/hits')
def hits():
    redis_server.incr('hit_counter')
    count = redis_server.get('hit_counter')
    return jsonify(hits=count), 200


if __name__ == "__main__":

    redis_server = redis.Redis(host=hostname, port=int(redis_port))
    app.run(host='0.0.0.0', port=int(port))

```

Application Code

app.py

```python
import os
import redis
from flask import Flask, jsonify

port = os.getenv('PORT', '5000')
hostname = os.getenv('HOSTNAME', '127.0.0.1')
redis_port = os.getenv('REDIS_PORT', '6379')

app = Flask(__name__)

# Application Routes
@app.route('/')
def index():
    return jsonify(name='Hit Me Service', version='1.0', url='/hits'), 200

@app.route('/hits')
def hits():
    redis_server.incr('hit_counter')
    count = redis_server.get('hit_counter')
    return jsonify(hits=count), 200


if __name__ == "__main__":

    redis_server = redis.Redis(host=hostname, port=int(redis_port))
    app.run(host='0.0.0.0', port=int(port))

```

Pull runtime data from the environment

Application Code

app.py

```python
import os
import redis
from flask import Flask, jsonify

port = os.getenv('PORT', '5000')
hostname = os.getenv('HOSTNAME', '127.0.0.1')
redis_port = os.getenv('REDIS_PORT', '6379')

app = Flask(__name__)

# Application Routes
@app.route('/')
def index():
    return jsonify(name='Hit Me Service', version='1.0', url='/hits'), 200

@app.route('/hits')
def hits():
    redis_server.incr('hit_counter')
    count = redis_server.get('hit_counter')
    return jsonify(hits=count), 200


if __name__ == "__main__":

    redis_server = redis.Redis(host=hostname, port=int(redis_port))
    app.run(host='0.0.0.0', port=int(port))

```

Pull runtime data from the environment

Connect to Redis and run Flask Server

Application Code

app.py

```python
import os
import redis
from flask import Flask, jsonify

port = os.getenv('PORT', '5000')
hostname = os.getenv('HOSTNAME', '127.0.0.1')
redis_port = os.getenv('REDIS_PORT', '6379')

app = Flask(__name__)

# Application Routes
@app.route('/')
def index():
    return jsonify(name='Hit Me Service', version='1.0', url='/hits'), 200

@app.route('/hits')
def hits():
    redis_server.incr('hit_counter')
    count = redis_server.get('hit_counter')
    return jsonify(hits=count), 200


if __name__ == "__main__":

    redis_server = redis.Redis(host=hostname, port=int(redis_port))
    app.run(host='0.0.0.0', port=int(port))

```

Pull runtime data from the environment

/ returns info

Connect to Redis and run Flask Server

Application Code

app.py

```python
import os
import redis
from flask import Flask, jsonify

port = os.getenv('PORT', '5000')
hostname = os.getenv('HOSTNAME', '127.0.0.1')
redis_port = os.getenv('REDIS_PORT', '6379')

app = Flask(__name__)

# Application Routes
@app.route('/')
def index():
    return jsonify(name='Hit Me Service', version='1.0', url='/hits'), 200

@app.route('/hits')
def hits():
    redis_server.incr('hit_counter')
    count = redis_server.get('hit_counter')
    return jsonify(hits=count), 200


if __name__ == "__main__":

    redis_server = redis.Redis(host=hostname, port=int(redis_port))
    app.run(host='0.0.0.0', port=int(port))

```

Pull runtime data from the environment

/ returns info

/hits calls Redis to increment counter and return it

Connect to Redis and run Flask Server

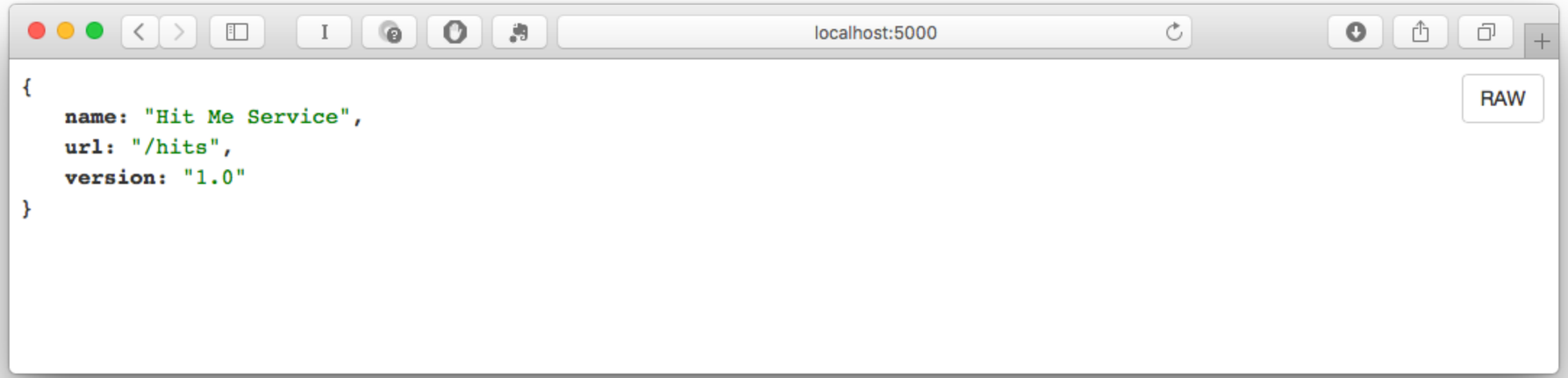
Start the Application

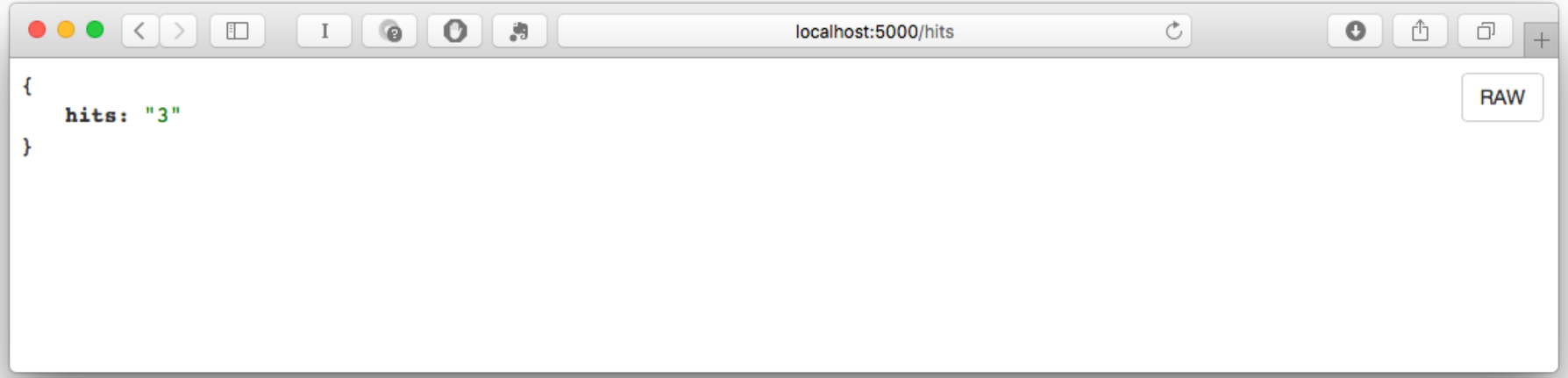
- Once the Vagrant VM has been provisioned we can `ssh` into it and start the Python Flask application

```
$ vagrant ssh
$ cd /vagrant
$ python app.py
```

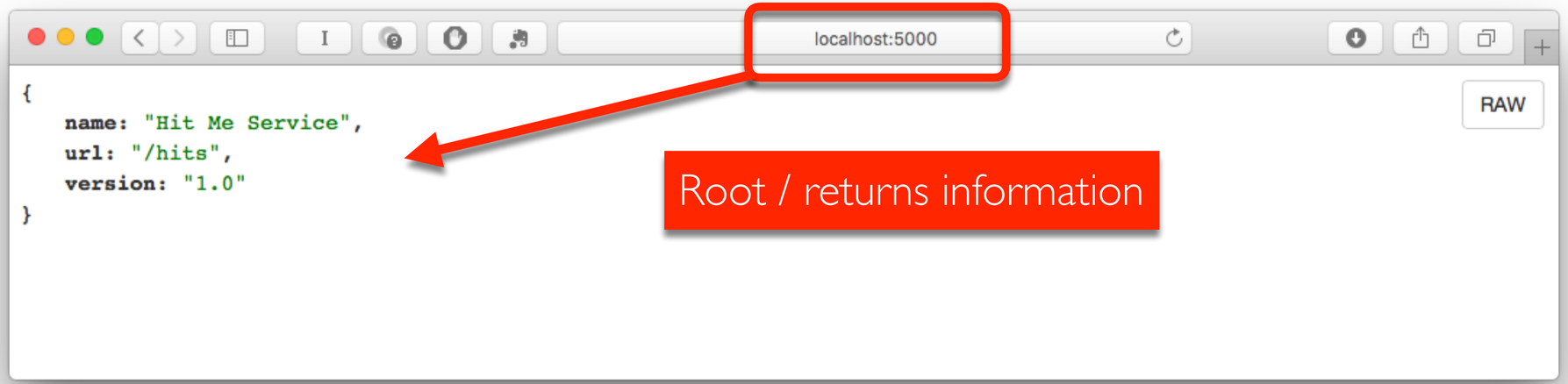
```
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

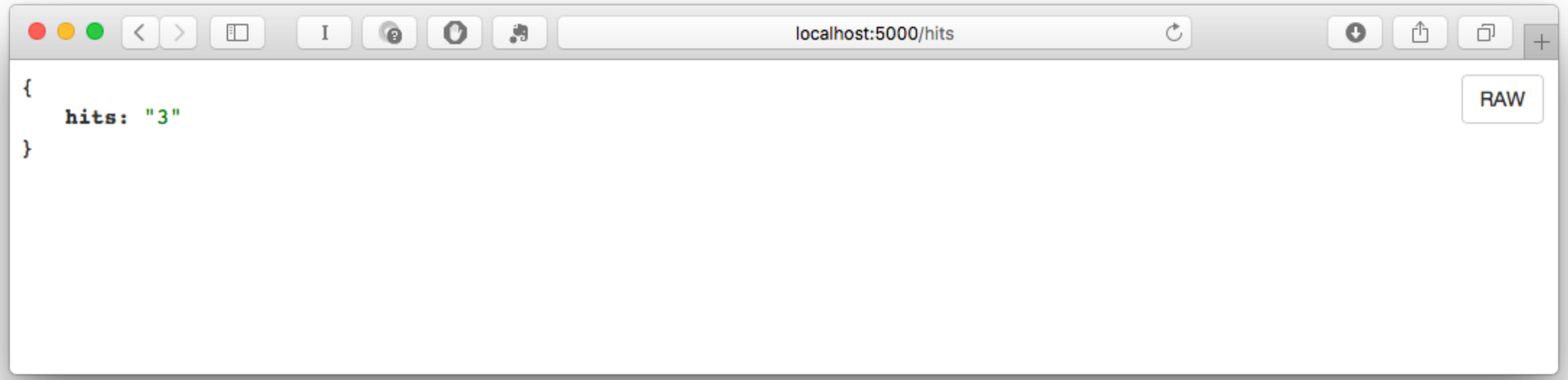
Flask is Serving Our Application



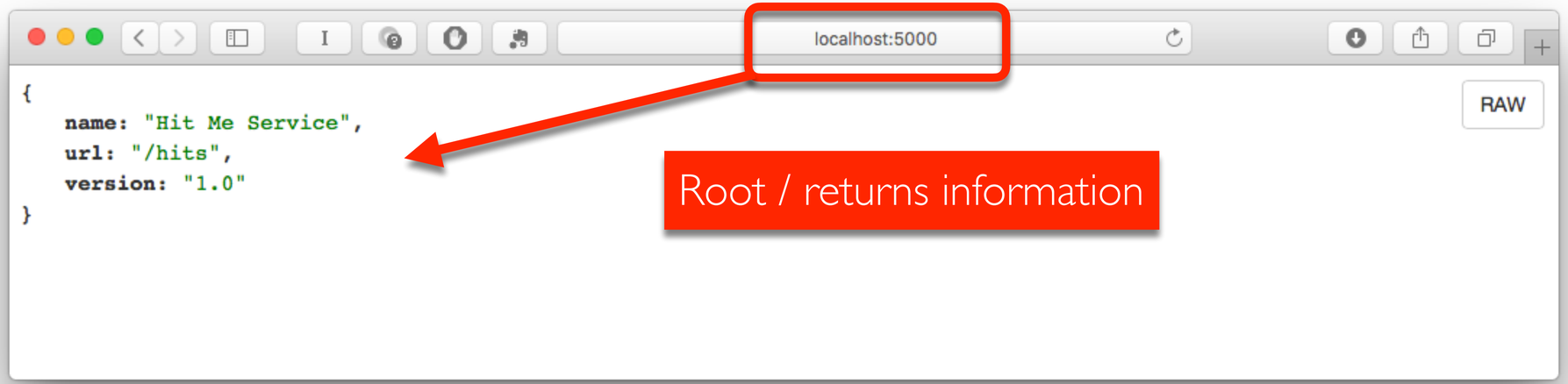


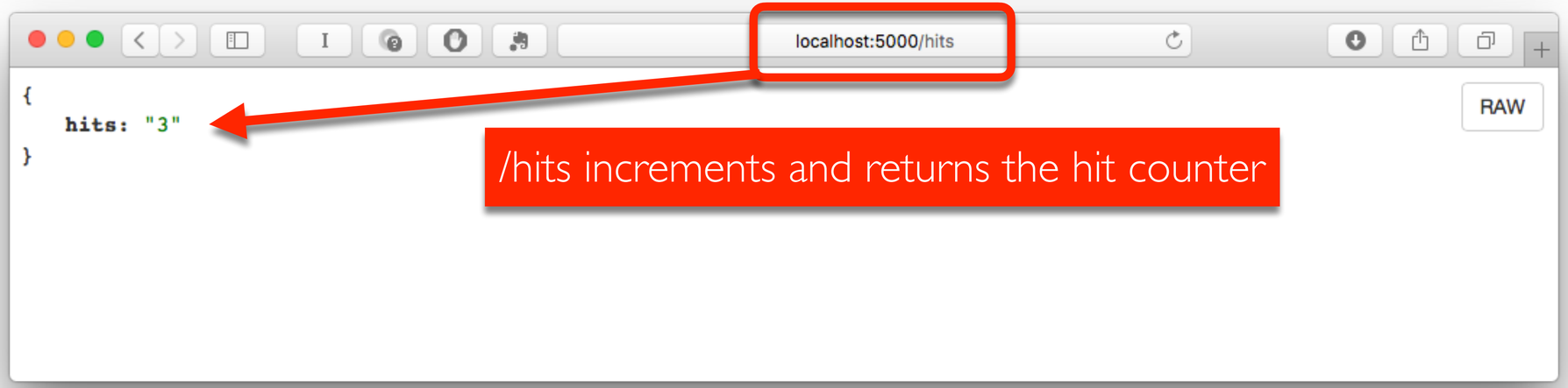
Flask is Serving Our Application





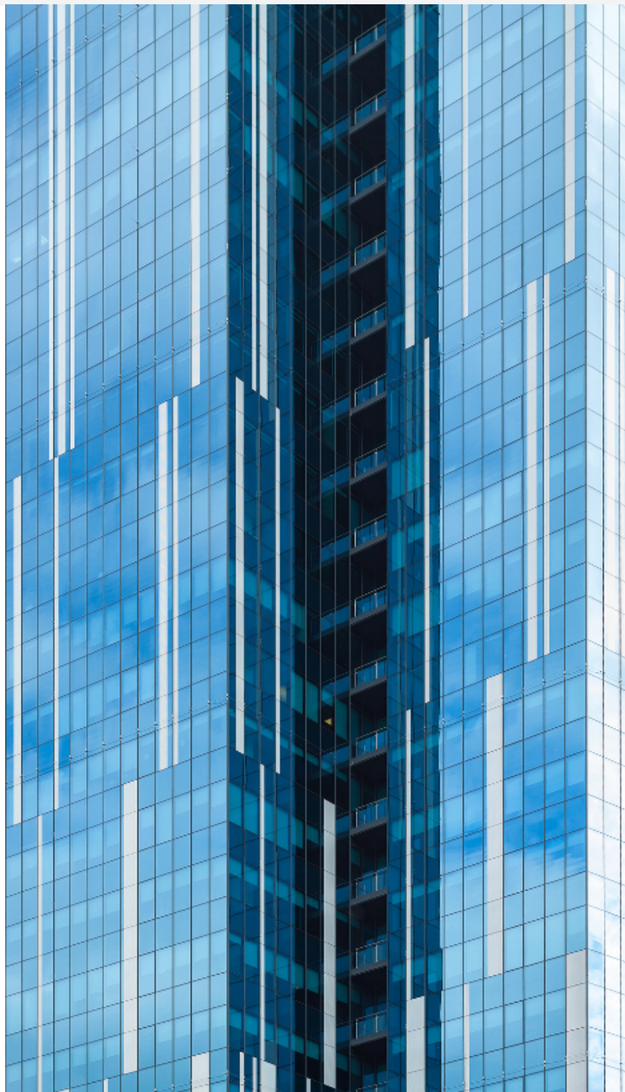
Flask is Serving Our Application





What Just Happened?

- A developer (you) who possibly never worked with Python, Flask, or Redis before
- Just cloned a git repo and provisioned a working development environment
- With one command: `vagrant up`



Python Environment

```
# Forward Python Flask port
config.vm.network :forwarded_port, guest: 5000, host: 5000, auto_correct: true

# Setup a Python development environment
config.vm.provision "shell", inline: <<-SHELL
  sudo apt-get update
  sudo apt-get install -y git python3-dev python3-pip python3-venv
  sudo apt-get -y autoremove
  # Install app dependencies
  cd /vagrant
  sudo pip install -r requirements.txt
SHELL
```

Ruby on Rails Environment

```
# Forward RAILS port
  config.vm.network :forwarded_port, guest: 3000, host: 3000, auto_correct: true

# Setup a Ruby on Rails development environment
  config.vm.provision :shell, privileged: false, inline: <<-SHELL
    sudo apt-get update
    sudo apt-get -y install git libsqlite3-dev libpq-dev nodejs
    sudo apt-get -y autoremove

    # disable docs during gem install
    echo "gem: --no-document" >> ~/.gemrc
    gem update --no-rdoc --no-ri
    gpg --keyserver hkp://keys.gnupg.net --recv-keys 409B6B1796C275462A1703113804BB82D39DC0E3
    # use RVM to install Ruby and Rails
    curl -sSL https://get.rvm.io | bash -s stable --rails

    # Add Heroku toolbelt
    sudo wget -O- https://toolbelt.heroku.com/install-ubuntu.sh | sh
  SHELL
```

Working With Docker In The VM

- We can run any Docker commands inside the VM to get the current status of any Docker containers

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
redis               alpine              8a8bdae38335        7 days ago          29.07 MB
```

Working With Docker In The VM

- We can run any Docker commands inside the VM to get the current status of any Docker containers


```
vagrant@vagrant-ubuntu-trusty-64:/vagrant$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
redis               alpine              8a8bdae38335        7 days ago          29.07 MB
```

Working With Docker In The VM

- We can run any Docker commands inside the VM to get the current status of any Docker containers

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
redis               alpine              8a8bdae38335        7 days ago          29.07 MB
```

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED        STATUS          PORTS                    NAMES
8ef040bad484        redis:alpine        "docker-entrypoint.sh"   25 hours ago   Up 16 minutes   0.0.0.0:6379->6379/tcp   redis
```

Working With Docker In The VM

- We can run any Docker commands inside the VM to get the current status of any Docker containers

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
redis               alpine              8a8bdae38335        7 days ago          29.07 MB
```

List Running Containers

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
8ef040bad484        redis:alpine        "docker-entrypoint.sh"   25 hours ago        Up 16 minutes       0.0.0.0:6379->6379/tcp   redis
```

Vagrantfile to Docker Mapping

- This is the mapping from the Vagrantfile to the Docker containers:

```
config.vm.provision "docker" do |d|
   d.pull_images "redis:alpine"
   d.run "redis:alpine",
     args: "--restart=always -d --name redis -h redis -p 6379:6379 -v /var/lib/redis/data:/data"
 end
```

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED        STATUS          PORTS                    NAMES
8ef040bad484        redis:alpine        "docker-entrypoint.sh"   25 hours ago   Up 16 minutes   0.0.0.0:6379->6379/tcp   redis
```

Vagrantfile to Docker Mapping

- This is the mapping from the Vagrantfile to the Docker containers:

```
config.vm.provision "docker" do |d|
   d.pull_images "redis:alpine"
   d.run "redis:alpine",
     args: "--restart=always -d --name redis -h redis -p 6379:6379 -v /var/lib/redis/data:/data"
 end
```

Image to run

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED        STATUS          PORTS                    NAMES
8ef040bad484        redis:alpine        "docker-entrypoint.sh"   25 hours ago   Up 16 minutes   0.0.0.0:6379->6379/tcp   redis
```

Vagrantfile to Docker Mapping

- This is the mapping from the Vagrantfile to the Docker containers:

```
config.vm.provision "docker" do |d|
   d.pull_images "redis:alpine"
   d.run "redis:alpine",
     args: "--restart=always -d --name redis -h redis -p 6379:6379 -v /var/lib/redis/data:/data"
 end
```

Image to run

Port to listen on

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED        STATUS          PORTS                    NAMES
8ef040bad484        redis:alpine        "docker-entrypoint.sh"   25 hours ago   Up 16 minutes   0.0.0.0:6379->6379/tcp   redis
```

Vagrantfile to Docker Mapping

- This is the mapping from the Vagrantfile to the Docker containers:

```
config.vm.provision "docker" do |d|
   d.pull_images "redis:alpine"
   d.run "redis:alpine",
     args: "--restart=always -d --name redis -h redis -p 6379:6379 -v /var/lib/redis/data:/data"
 end
```


```
vagrant@vagrant-ubuntu-trusty-64:/vagrant$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
8ef040bad484        redis:alpine        "docker-entrypoint.sh"   25 hours ago        Up 16 minutes       0.0.0.0:6379->6379/tcp   redis
```

What about data persistence?

- Map a Volume in the VM to persist data cross Docker containers

```
Vagrant.configure(2) do |config|
 config.vm.provision "shell", inline: <<-SHELL
    # Prepare Redis data share
    sudo mkdir -p /var/lib/redis/data
    sudo chown vagrant:vagrant /var/lib/redis/data
 SHELL

  # Add Redis docker container
  config.vm.provision "docker" do |d|
    d.pull_images "redis:alpine"
    d.run "redis:alpine",
      args: "--restart=always -d --name redis -h redis -p 6379:6379 -v /var/lib/redis/data:/data"
  end
end
```

What about data persistence?

- Map a Volume in the VM to persist data cross Docker containers

```
Vagrant.configure(2) do |config|
 config.vm.provision "shell", inline: <<-SHELL
    # Prepare Redis data share
    sudo mkdir -p /var/lib/redis/data
    sudo chown vagrant:vagrant /var/lib/redis/data
 SHELL

  # Add Redis docker container
  config.vm.provision "docker" do |d|
    d.pull_images "redis:alpine"
    d.run "redis:alpine",
      args: "--restart=always -d --name redis -h redis -p 6379:6379 -v /var/lib/redis/data:/data"
  end
end
```

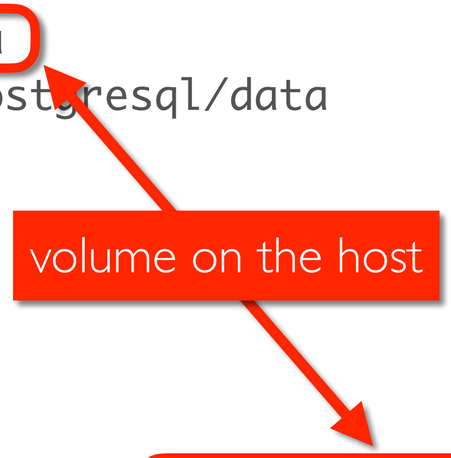
Persist container data in the VM

Data Volumes for PostgreSQL

```
##############################################################
# Add PostgreSQL docker container
##############################################################
config.vm.provision "shell", inline: <<-SHELL
  # Prepare PostgreSQL data share
  sudo mkdir -p /var/lib/postgresql/data
  sudo chown vagrant:vagrant /var/lib/postgresql/data
SHELL

# Add PostgreSQL docker container
config.vm.provision "docker" do |d|
  d.pull_images "postgres"
  d.run "postgres",
    args: "-d --name postgres -p 5432:5432 -v /var/lib/postgresql/data:/var/lib/postgresql/data"
end
```

Data Volumes for PostgreSQL

```
##############################################################
# Add PostgreSQL docker container
##############################################################
config.vm.provision "shell", inline: <<-SHELL
  # Prepare PostgreSQL data share
  sudo mkdir -p /var/lib/postgresql/data
  sudo chown vagrant:vagrant /var/lib/postgresql/data
SHELL

# Add PostgreSQL docker container
config.vm.provision "docker" do |d|
  d.pull_images "postgres"
  d.run "postgres",
    args: "-d --name postgres -p 5432:5432 -v /var/lib/postgresql/data:/var/lib/postgresql/data"
end
```



Data Volumes for PostgreSQL

```
##############################################################
# Add PostgreSQL docker container
##############################################################
config.vm.provision "shell", inline: <<-SHELL
  # Prepare PostgreSQL data share
  sudo mkdir -p /var/lib/postgresql/data
  sudo chown vagrant:vagrant /var/lib/postgresql/data
SHELL

# Add PostgreSQL docker container
config.vm.provision "docker" do |d|
  d.pull_images "postgres"
  d.run "postgres",
    args: "-d --name postgres -p 5432:5432 -v /var/lib/postgresql/data:/var/lib/postgresql/data"
end
```

Data Volumes for MySQL

```
##############################################################
# Add MySQL docker container
##############################################################
config.vm.provision "shell", inline: <<-SHELL
  # Prepare MySQL data share
  sudo mkdir -p /var/lib/mysql
  sudo chown vagrant:vagrant /var/lib/mysql
SHELL

# Add MySQL docker container
config.vm.provision "docker" do |d|
  d.pull_images "mariadb"
  d.run "mariadb",
    args: "--restart=always -d --name mariadb -p 3306:3306 -v /var/lib/mysql:/var/lib/mysql  -e
MYSQL_ROOT_PASSWORD=passw0rd"
end
```

Data Volumes for Redis

```
##############################################################
# Add Redis docker container
##############################################################
config.vm.provision "shell", inline: <<-SHELL
  # Prepare Redis data share
  sudo mkdir -p /var/lib/redis/data
  sudo chown vagrant:vagrant /var/lib/redis/data
SHELL

# Add Redis docker container
config.vm.provision "docker" do |d|
  d.pull_images "redis:alpine"
  d.run "redis:alpine",
    args: "--restart=always -d --name redis -h redis -p 6379:6379 -v /var/lib/redis/data:/data"
end
```

Data Volumes for CouchDB

```
##############################################################
# Add CouchDB docker container
##############################################################
config.vm.provision "shell", inline: <<-SHELL
  # Prepare CouchDB data share
  sudo mkdir -p /usr/local/var/lib/couchdb
  sudo chown vagrant:vagrant /usr/local/var/lib/couchdb
SHELL

# Add CouchDB docker container
config.vm.provision "docker" do |d|
  d.pull_images "couchdb"
  d.run "couchdb",
    args: "--restart=always -d --name couchdb -p 5984:5984 -v /usr/local/var/lib/couchdb:/usr/local/
var/lib/couchdb"
end
```

Using Docker for Middleware

Here is an example of using the Docker container as a client for running commands

Start a container called "redis"

```
$ docker run -d --name redis -p 6379:6379 redis:alpine
```

Call `redis-cli` with parameters*

```
$ docker exec redis redis-cli incr mycounter
1
```

Start an interactive session with `redis-cli`

```
$ docker exec -it redis redis-cli
127.0.0.1:6379>
```

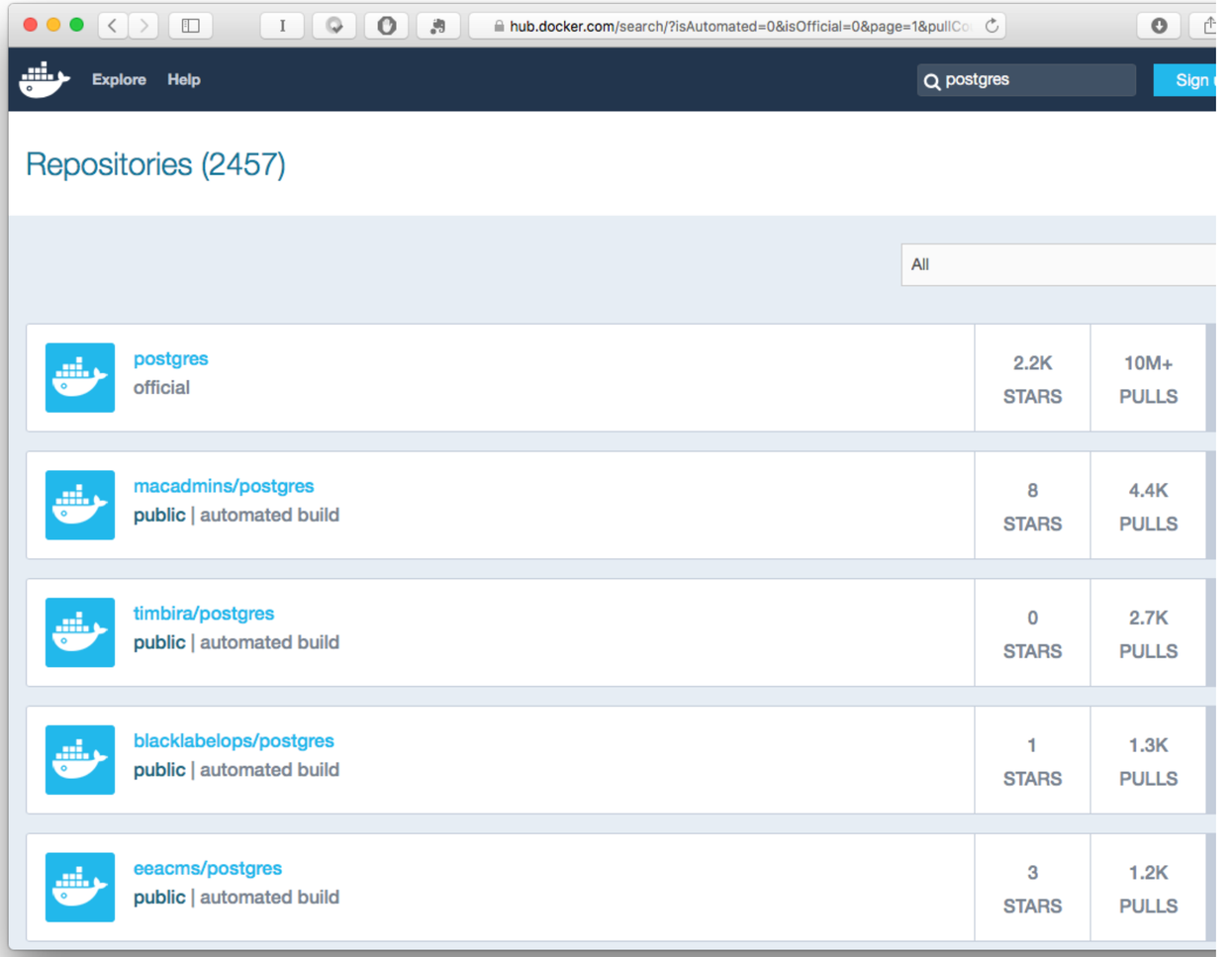
Handy Docker Startup Args

- `--restart=always` <— if it stops, restart it
- `-d` <— run as a daemon process (background)
- `--name <name>` <— name of running container in docker ps
- `-p <host>:<cont>` <— forward host port to container port
- `-v </host/folder>:<container/folder>` <— map volume
- `-e MY_ENV_VAR=<value>` <— inject environment variable

Where To Get More Docker Imges?

hub.docker.com

- Contains 1000's of Docker images with almost every software you can imagine
- Usually "official" images come from the creator of the software
- Most have documentation on how to best use them



Getting In and Out of your VM

- Just like any remote server that you ssh into, you use exit to leave

```
$ exit
iotia:lab-vagrant rofrano$
```

- To get back in you just ssh again

```
iotia:lab-vagrant rofrano$ vagrant ssh
$
```

Ending Your Vagrant Session

- When you are done for the day, it’s a good idea to shutdown your VM.
- This is accomplished with the `halt` command

```
$ exit
iotia:lab-vagrant rofrano$ vagrant halt
==> default: Attempting graceful shutdown of VM...
```

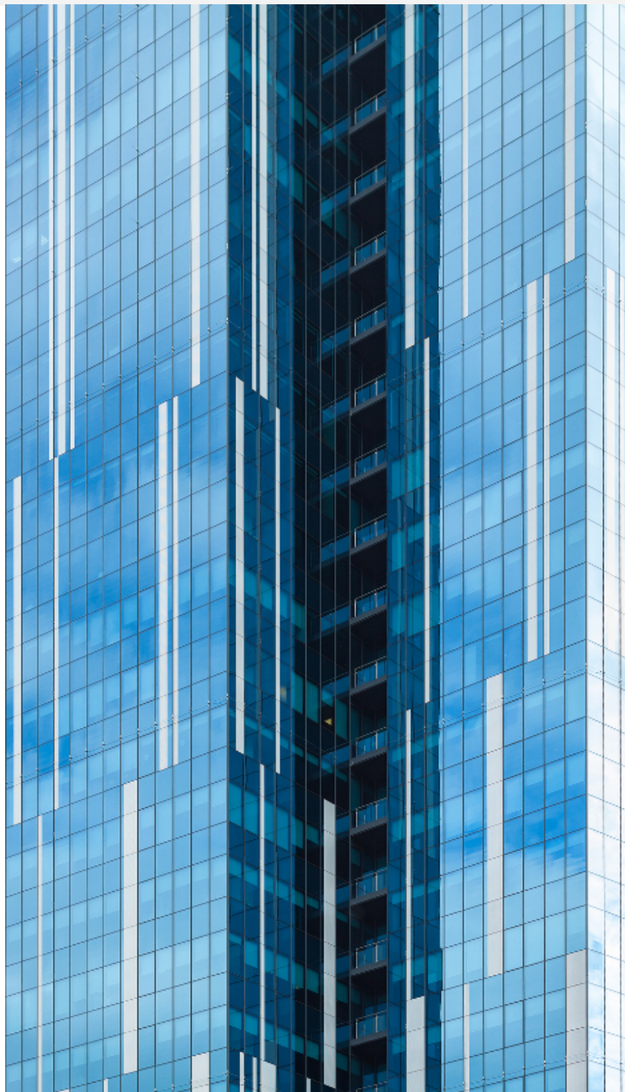
Removing a VM Permanently

- When you no longer need a VM you can delete it from your computer
- This is accomplished with the `destroy` command

```
$ vagrant destroy

    default: Are you sure you want to destroy the 'default' VM? [y/N]
==> default: Destroying VM and associated drives...
```

Vagrant Destroy

- Vagrant Destroy is your friend
- Don’t keep any data inside the VM, always map a folder
- When things go wrong, destroy the VM and create a new one!
  - Don’t waste time debugging VM problems
  - That’s the whole idea of infrastructure as code ! ! !



Handy Vagrant Commands

- **`vagrant up`** — bring up the vm
- **`vagrant ssh`** — open a shell inside the vm
- **`exit`** — exit out of the vm shell back to your host computer
- **`vagrant halt`** — shutdown the vm to return later with vagrant up
- **`vagrant suspend`** — suspend the vm
- **`vagrant resume`** — resume a suspended vm
- **`vagrant reload --provision`** — restarts vagrant machine, loads new Vagrantfile configuration and reprovisions shell scripts
- **`vagrant destroy`** — delete the vm from your hard drive
- **`vagrant status`** — see the status of all the current Vagrantfile's VM
- **`vagrant global-status`** — see the status of all Vagrant VM's

Summary

- You just deployed your first Vagrant environment
- You learned how to automatically install additional software
- You added Docker containers to supply needed middleware
- You can now check your `Vagrantfile` into GitHub so that others can create the perfect development environment when working on your project.