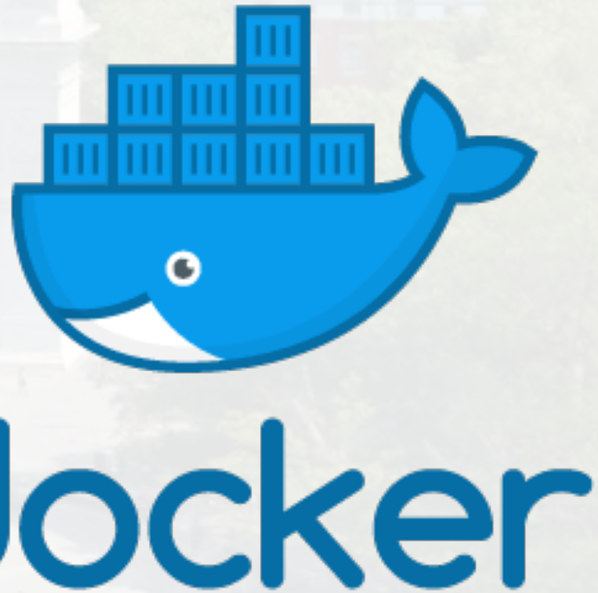
NYU COURANT INSTITUTE OF MATHEMATICAL SCIENCES

# Introduction to Docker




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

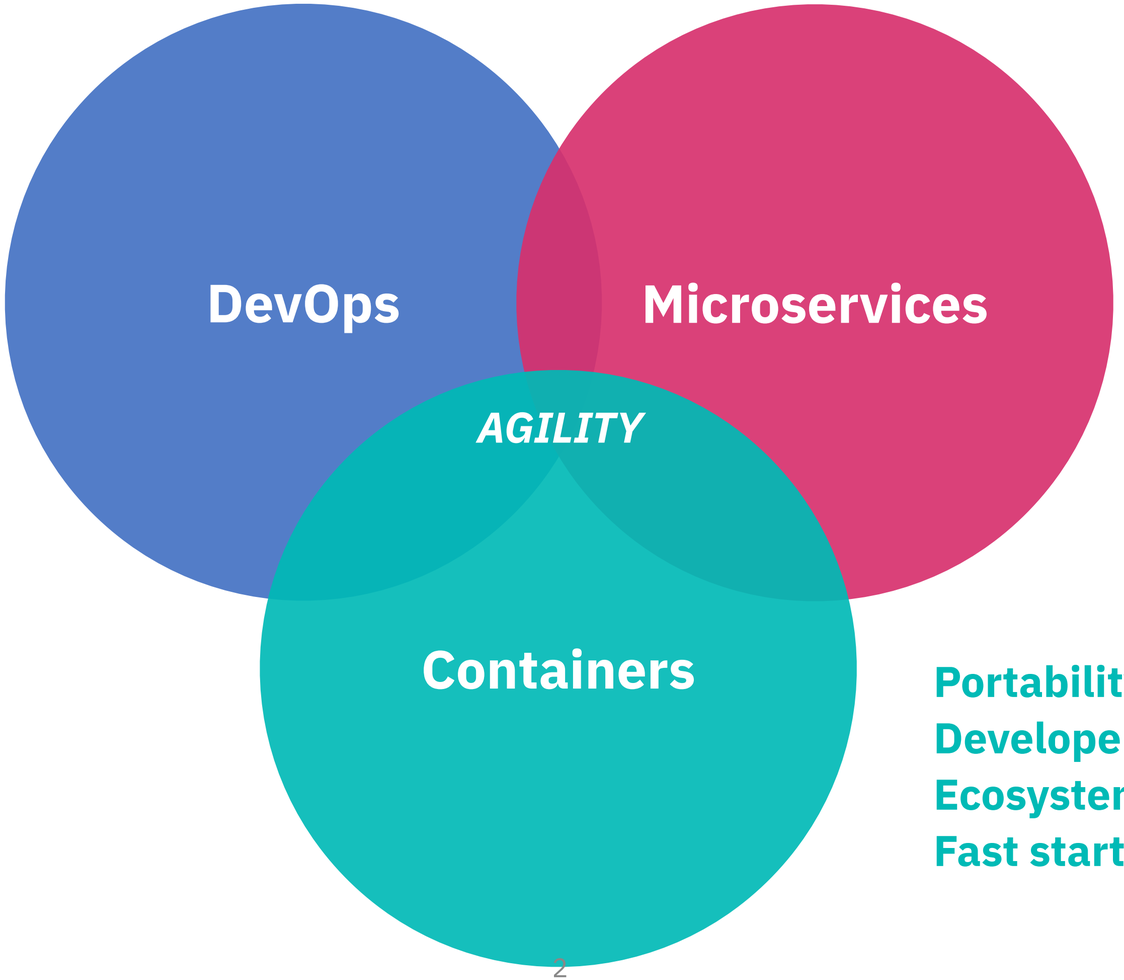
Instructor:

**John J Rofrano**

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

# THE PERFECT STORM

**Cultural Change**
**Automated Pipeline**
**Everything as Code**
**Immutable Infrastructure**

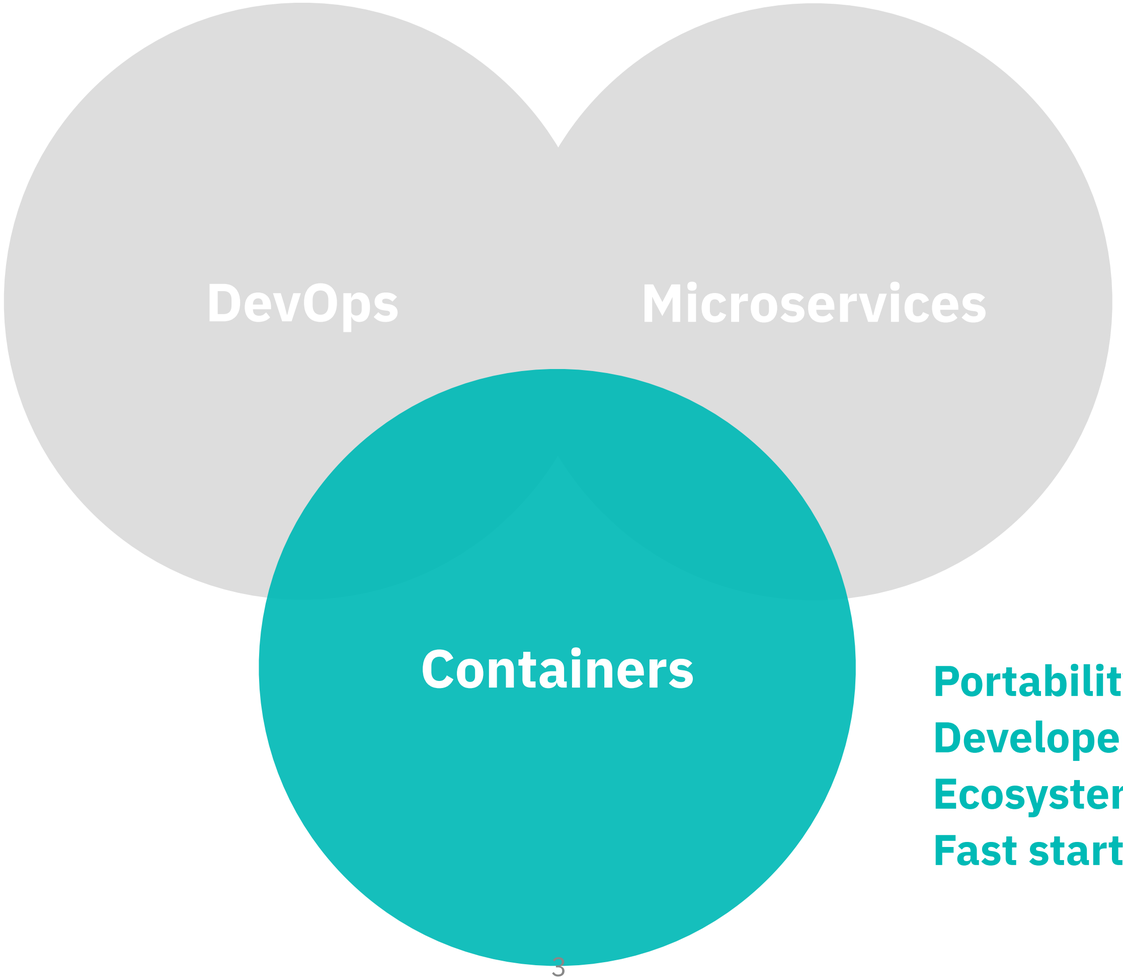



**Loose Coupling/Binding**
**RESTful APIs**
**Designed to resist failures**
**Test by break / fail fast**

**Portability**
**Developer Centric**
**Ecosystem enabler**
**Fast startup**

# THE PERFECT STORM: Containers

Cultural Change
Automated Pipeline
Everything as Code
Immutable Infrastructure

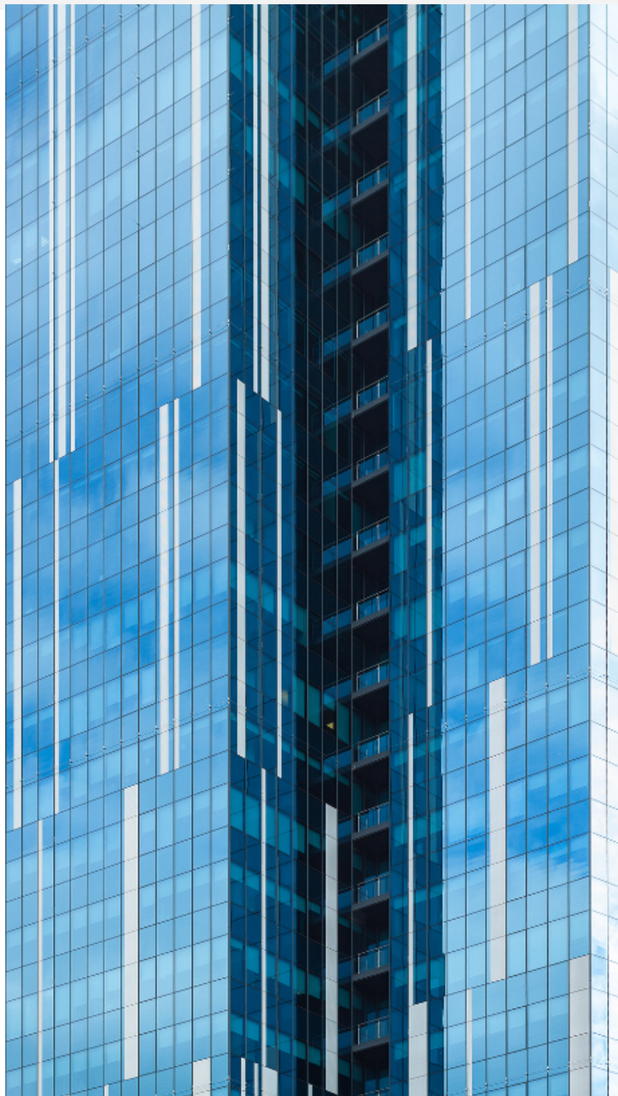



Loose Coupling/Binding
RESTful APIs
Designed to resist failures
Test by break / fail fast

**Portability**
**Developer Centric**
**Ecosystem enabler**
**Fast startup**

# What Will You Learn?

- What Docker is and isn’t
- Why would you use Docker
- How to use Existing Docker Images
- How to Create your own Docker Images from Dockerfiles



# Source for This Lab

- The source for this lab can be found on GitHub at:

```
git clone https://github.com/nyu-devops/lab-kubernetes.git
cd lab-kubernetes
vagrant up
```

# Vagrant Plug-ins


- The `Vagrantfile` can check for required plug-ins and instal, them for you ...but you need to run `vagrant up` again.

```
$ vagrant up
Plugin missing.
Installing the 'vagrant-docker-compose' plugin. This can take a few minutes...
Fetching: vagrant-docker-compose-1.3.0.gem (100%)
Installed the plugin 'vagrant-docker-compose (1.3.0)'!
Dependencies installed, please try the command again.
$
```

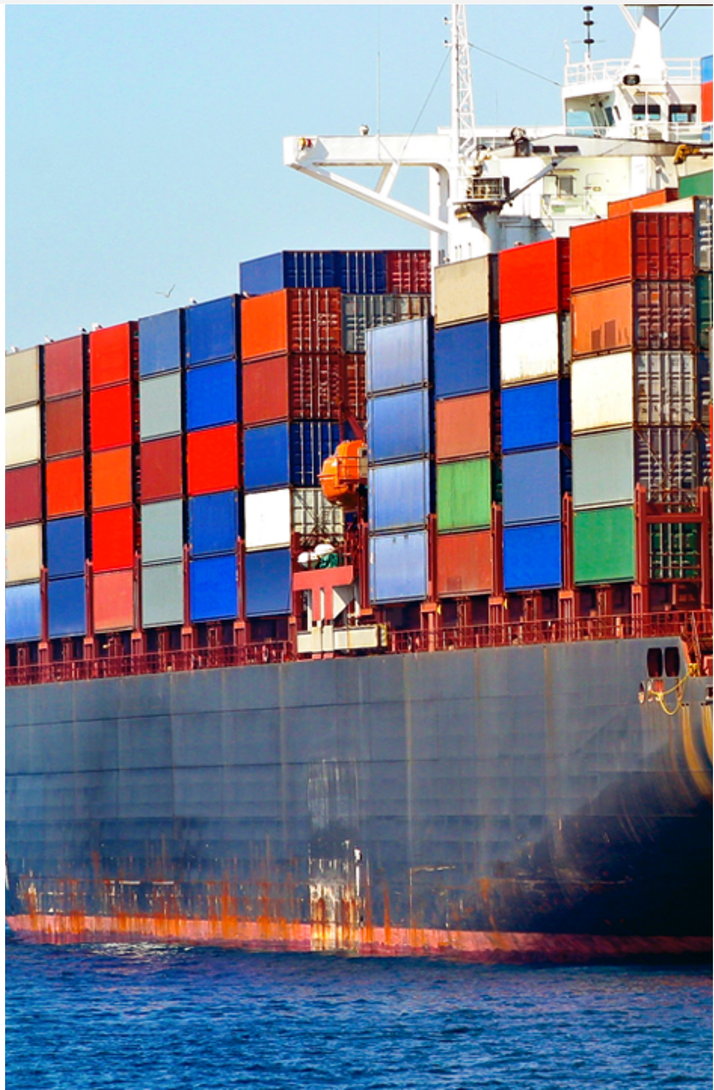

BUSINESS SOLUTION
GLOBAL NETWORK

# The Lure of Containers

# The Shipping Industry's Solution to Transporting Cargo (post-1960)

How do you ship cargo of variable sizes efficiently?

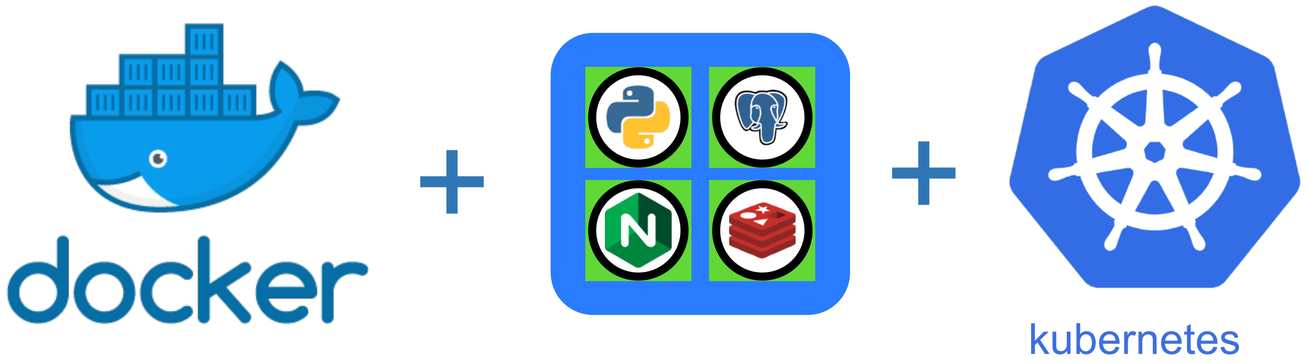
- Give the customer a standard sized container
- They can put anything they want in it, but it's their responsibility for what is inside of it
- Build transport ships that can efficiently hold these containers at scale
- Cargo size no longer matters because you only deal with the standard container

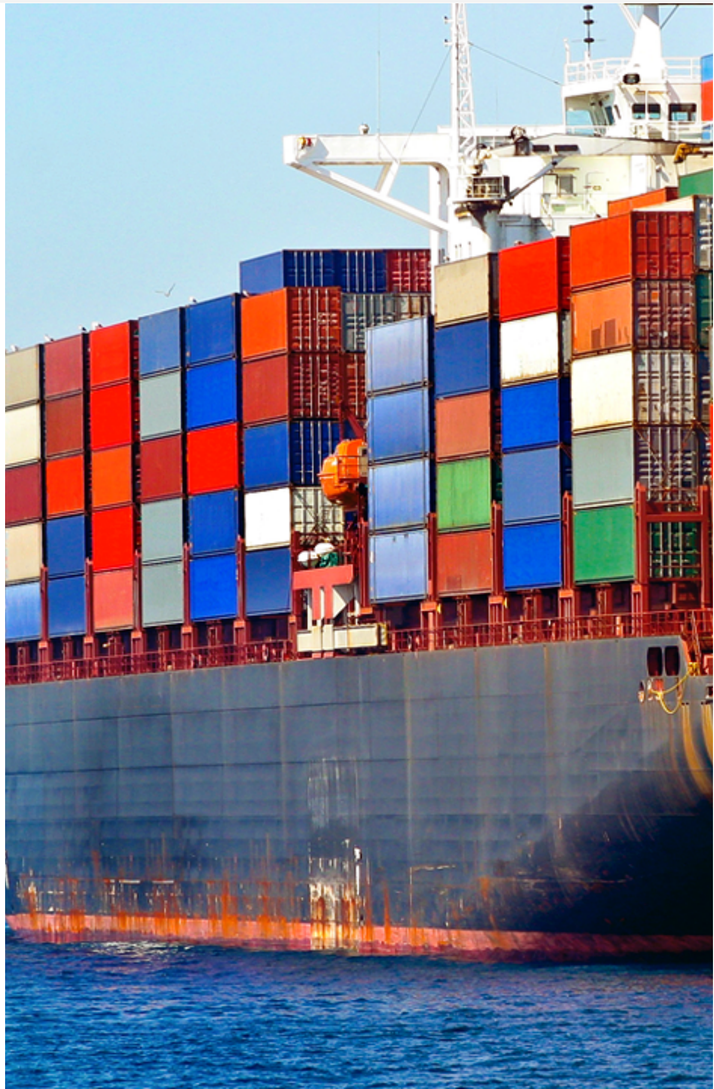



# The IT Industry's Solution to Deploying Applications

How do you deploy applications with variable dependencies?

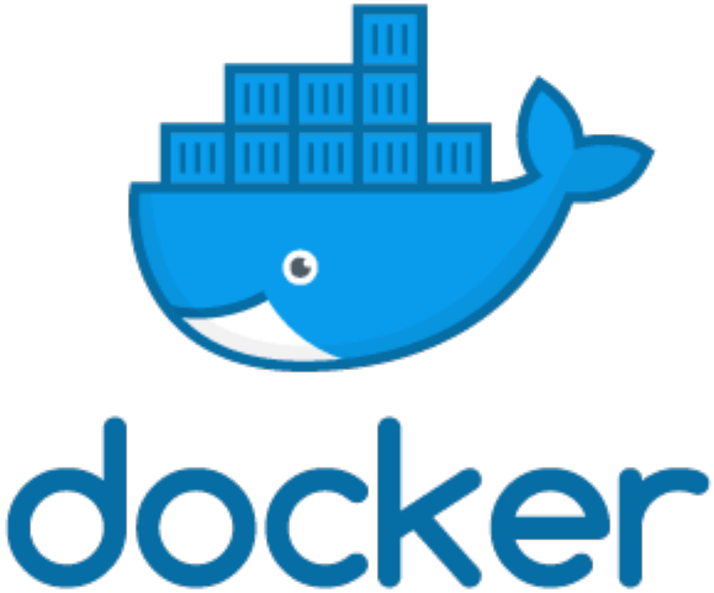
- Give the developer a standard container
- They can put anything they want in it, but it's their responsibility for what is inside of it
- Build a programmable infrastructure that can efficiently run these containers at scale
- Application dependencies no longer matters because you only deal with the container



# Docker Containers

- Docker is a light-weight container service that runs on Linux
  - File system overlay
  - One Process Space
  - One Network Interface
- Shares the Linux kernel with other containers and processes
  - Originally based on LXC (LinuX Containers)
- Containers encapsulate a run-time environment
  - They can contain code, libraries, package manager, data, etc.
- Almost no overhead
  - Containers spin up in seconds not minutes like VMs
  - Native performance because there is no emulation
  - Package only what you need



# Docker Concepts

- Docker is a platform for developers and sysadmins to develop, deploy, and run applications with containers
- The use of Linux containers to deploy applications is called *containerization*
- Containers are not new, but Docker make their use for easily deploying applications simple

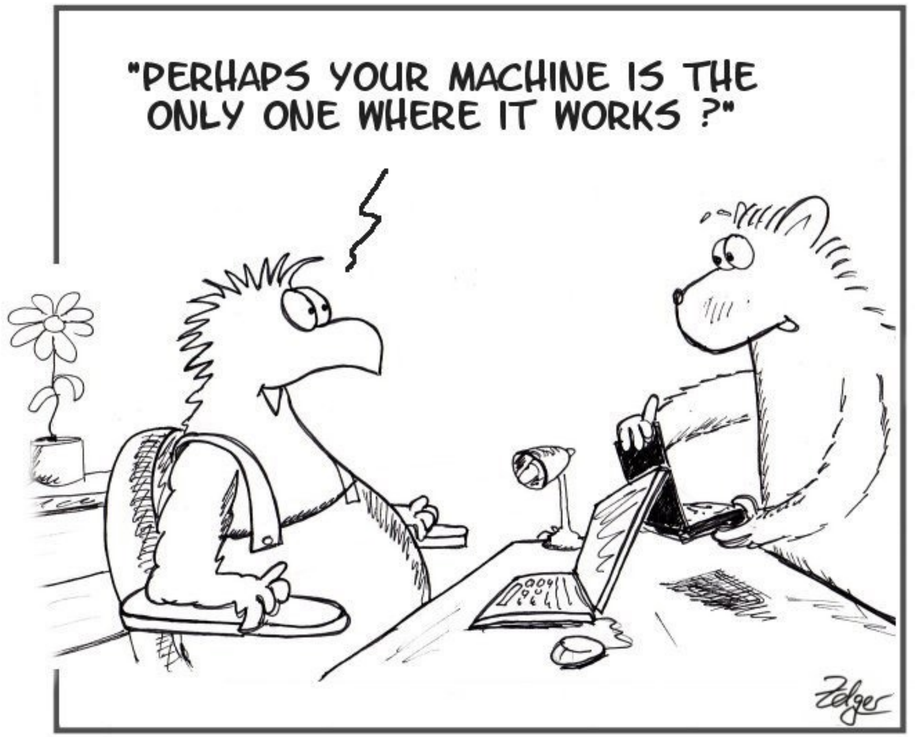
# Benefits of Containers

- Great isolation
- Great manageability
- Container encapsulates implementation technology
- Efficient resource utilization
- Fast deployment

# Containers Enable Immutable Delivery

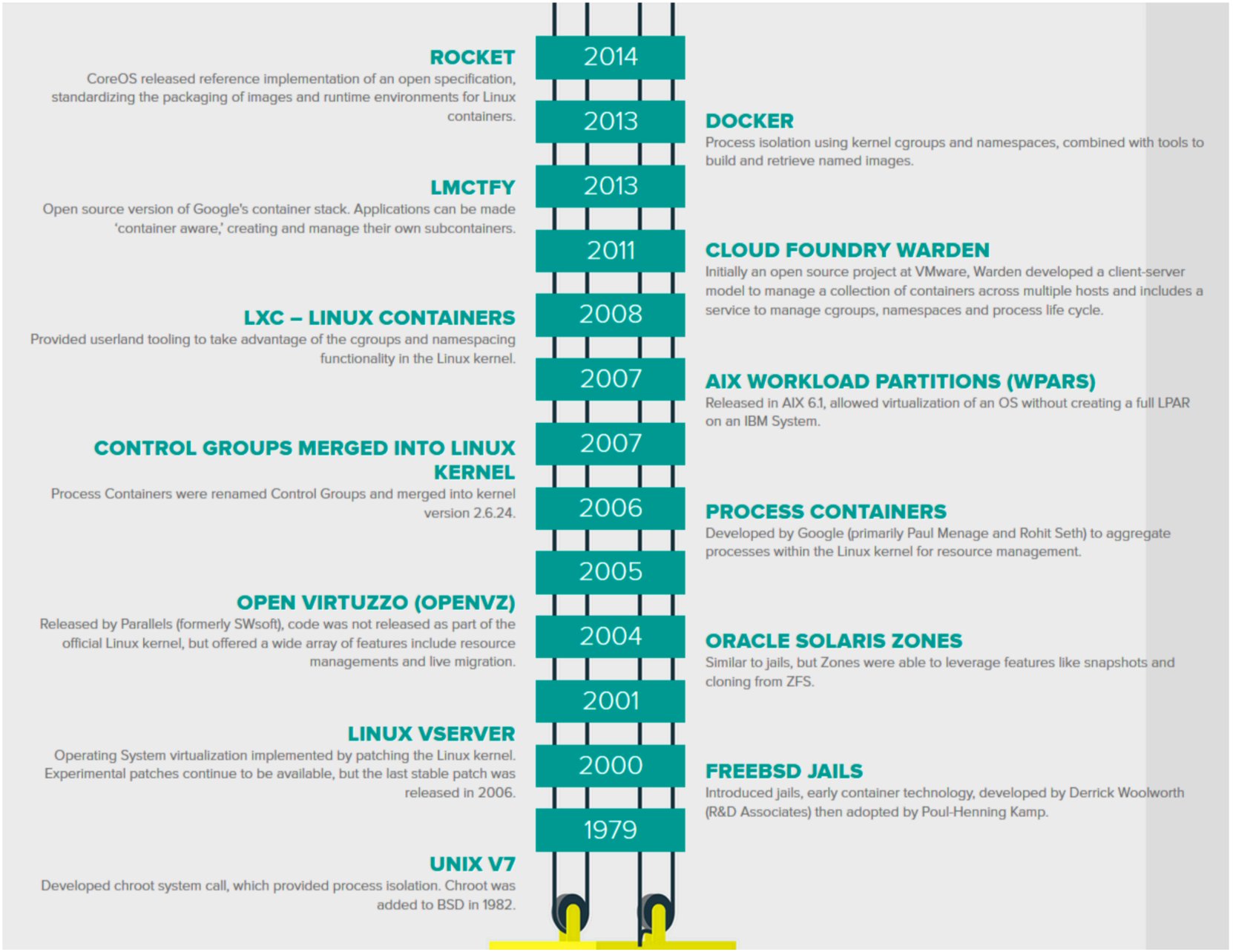
## Build once... run anywhere

- The same binary that a developer runs on their laptop, runs in production
- All dependencies are package in the container
- Facilitates rolling updates with immediate roll-back
- Consistency limits side-effects

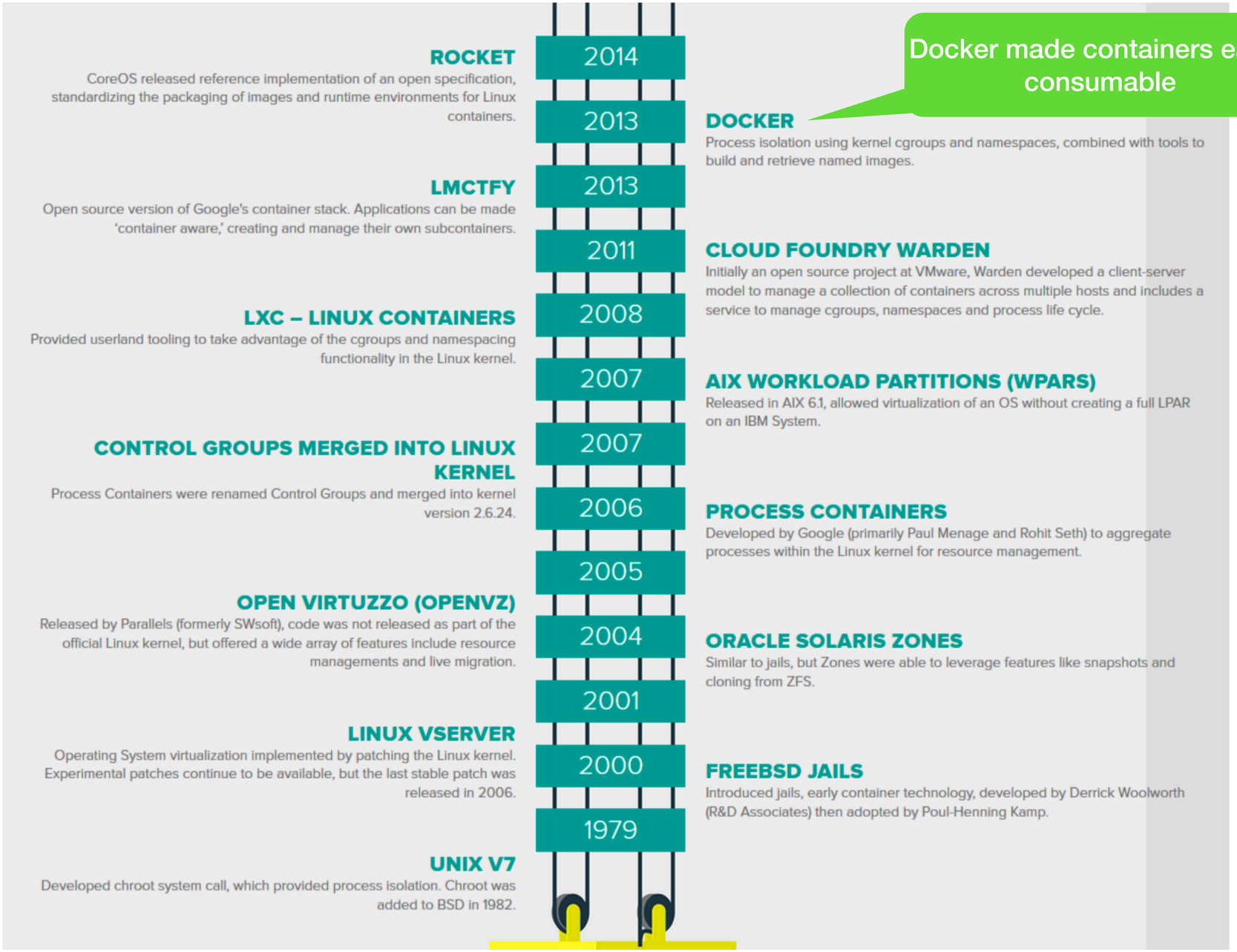



It works on my machine

# Containers are not new

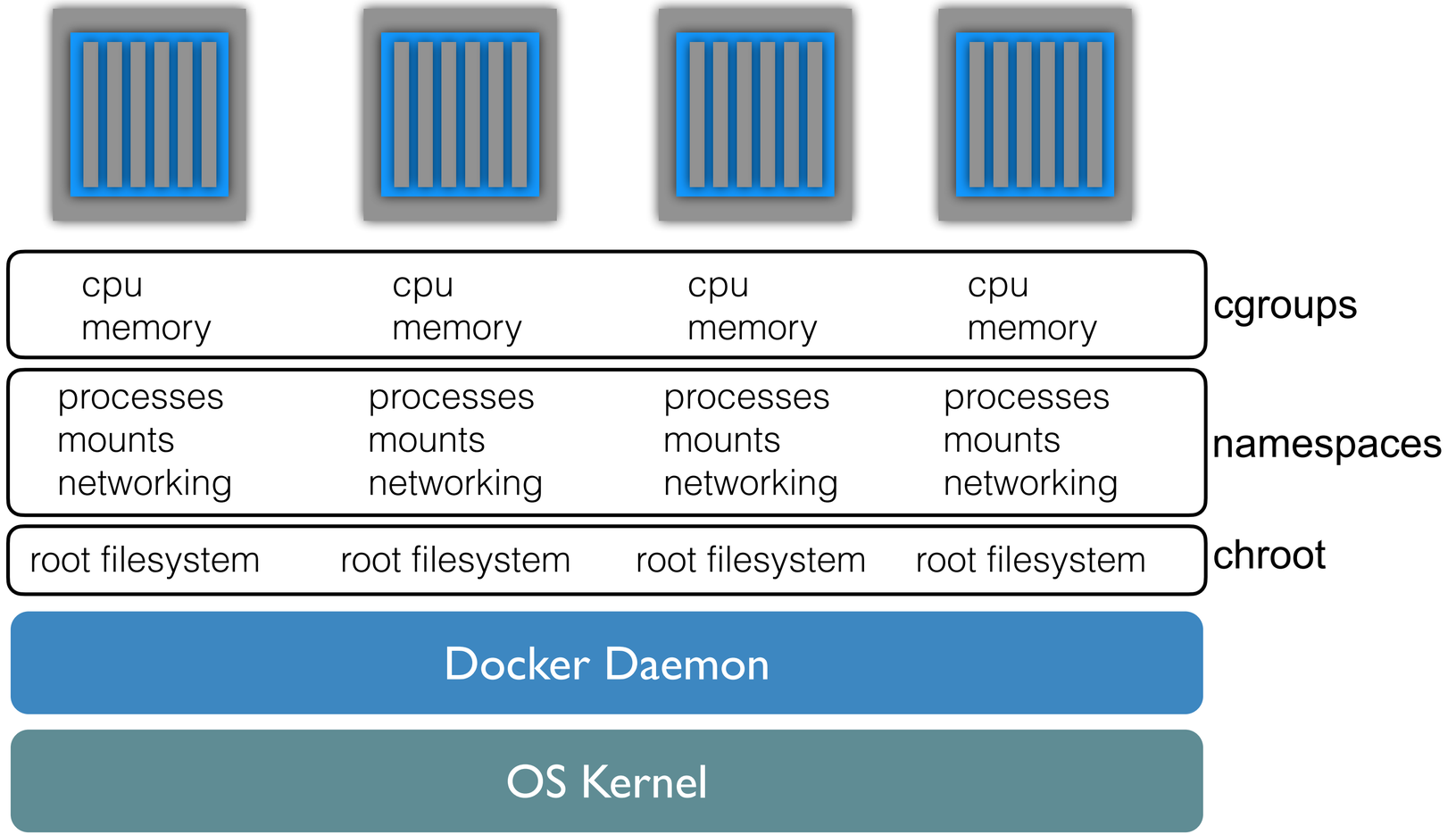


# Containers are not new



# 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

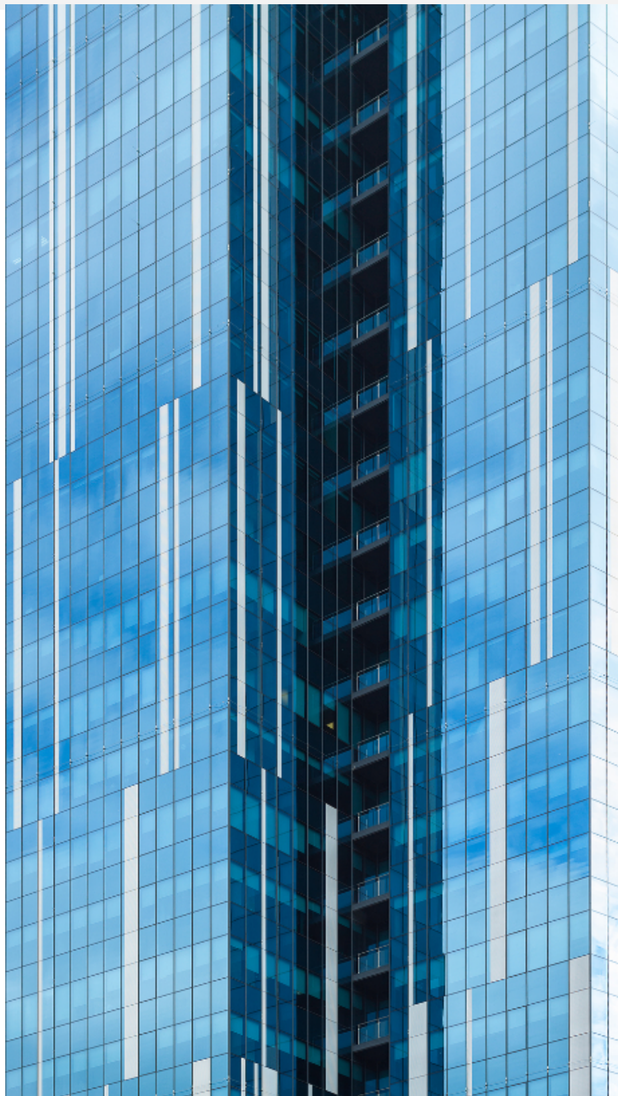


# Docker Containers are just Linux...

- **cgroups** - Control Groups allow you to control how much resources are allocated to a process (e.g., memory, cpu. etc.)
- **namespaces** - control access to what you can see (e.g., processes, mounts, networking, etc.). What you can't see, you can't access!
- **chroot** - Allows you to change the root filesystem. This allows the apparent root to be any linux filesystem whether it be ubuntu, opensuse, redhat or otherwise (i.e., overlay filesystem)

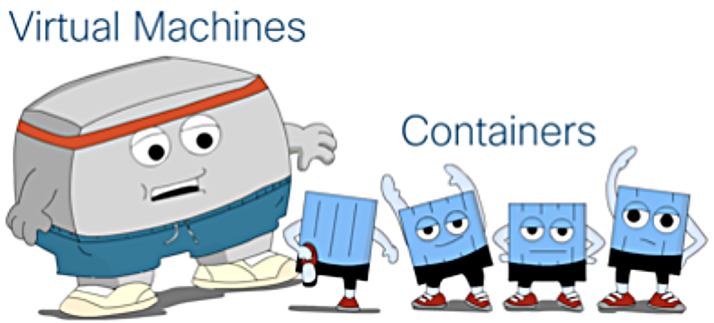
# Put another way

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



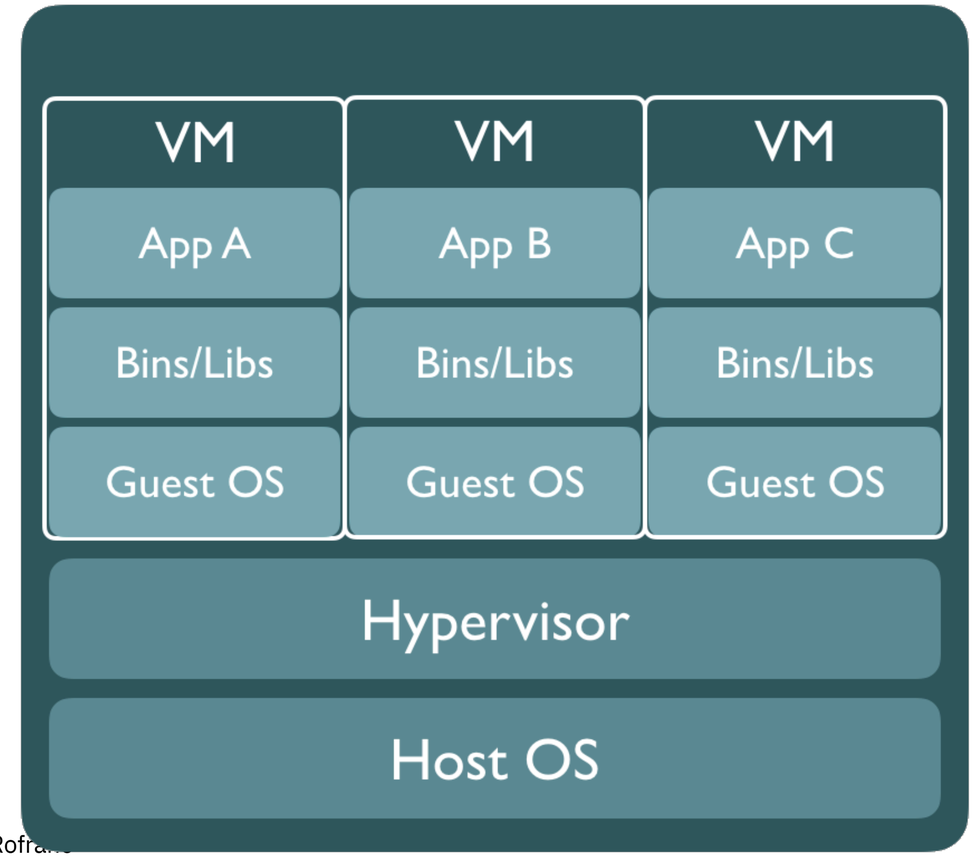
# Containers and Virtual Machines

- 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



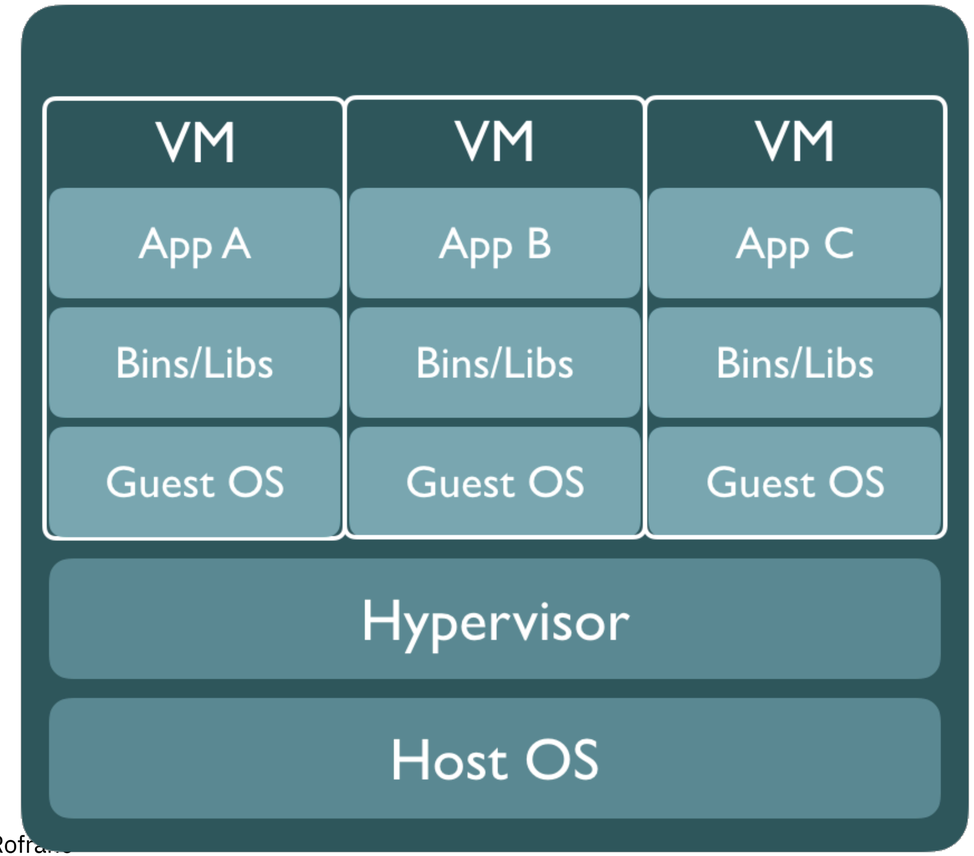
# Containers vs Virtual Machines

Virtual Machines are heavy-weight emulations of real hardware

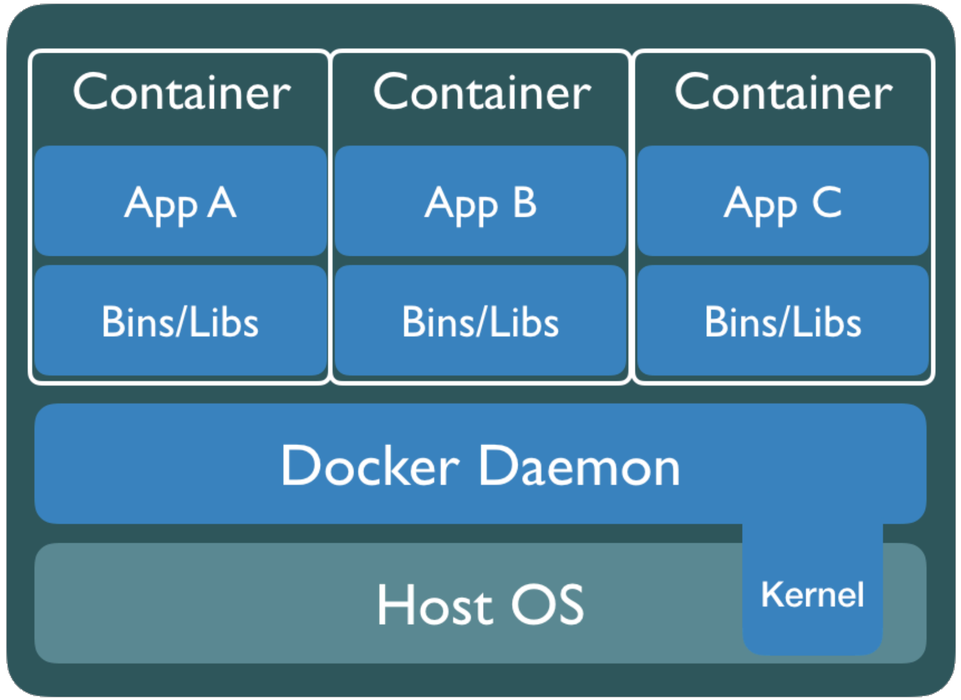


# Containers vs Virtual Machines

Virtual Machines are heavy-weight emulations of real hardware



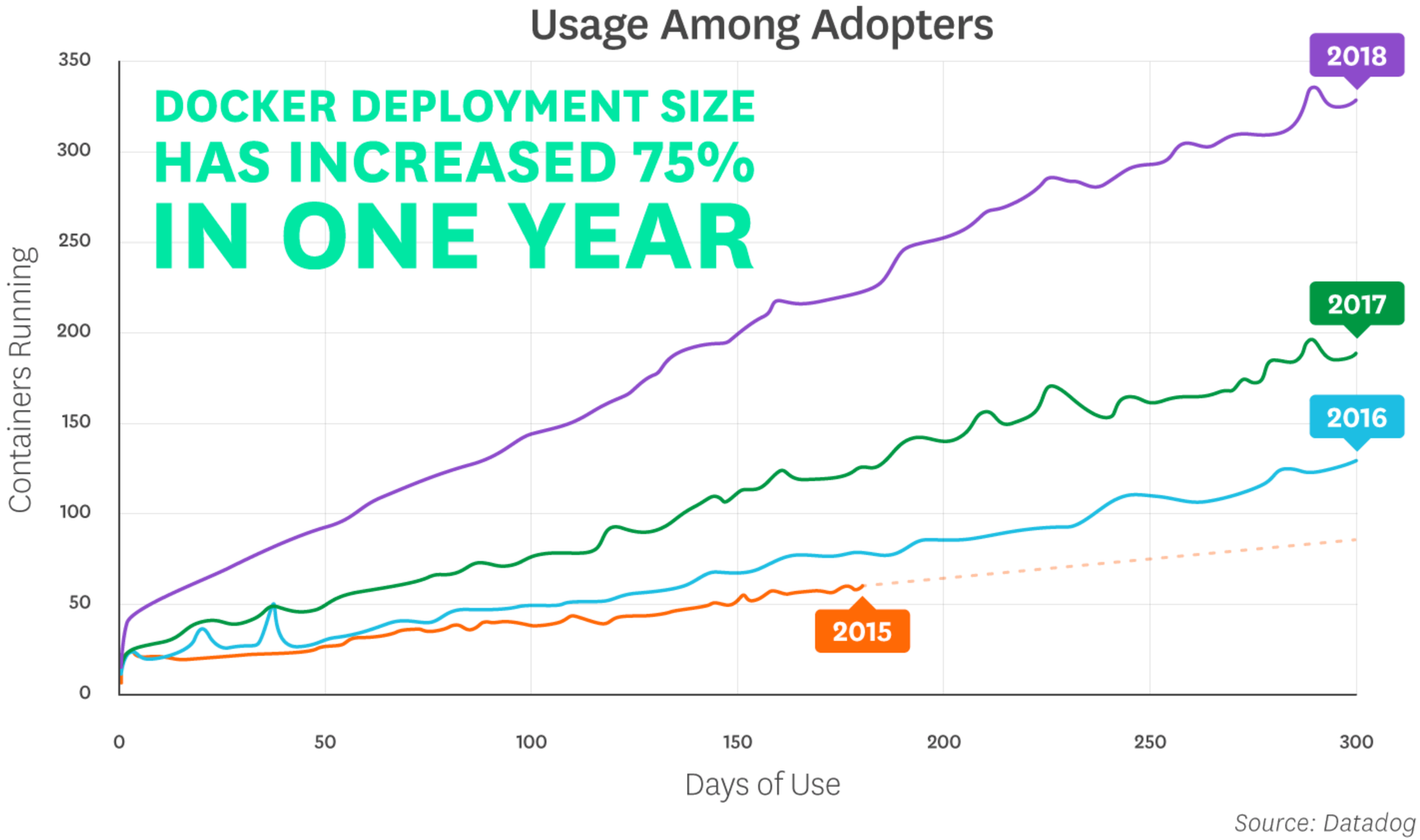

Containers are light-weight process
The app looks like it's running on the Host OS




B2B SALES STRATEGY & BRAND COMMUNICATION

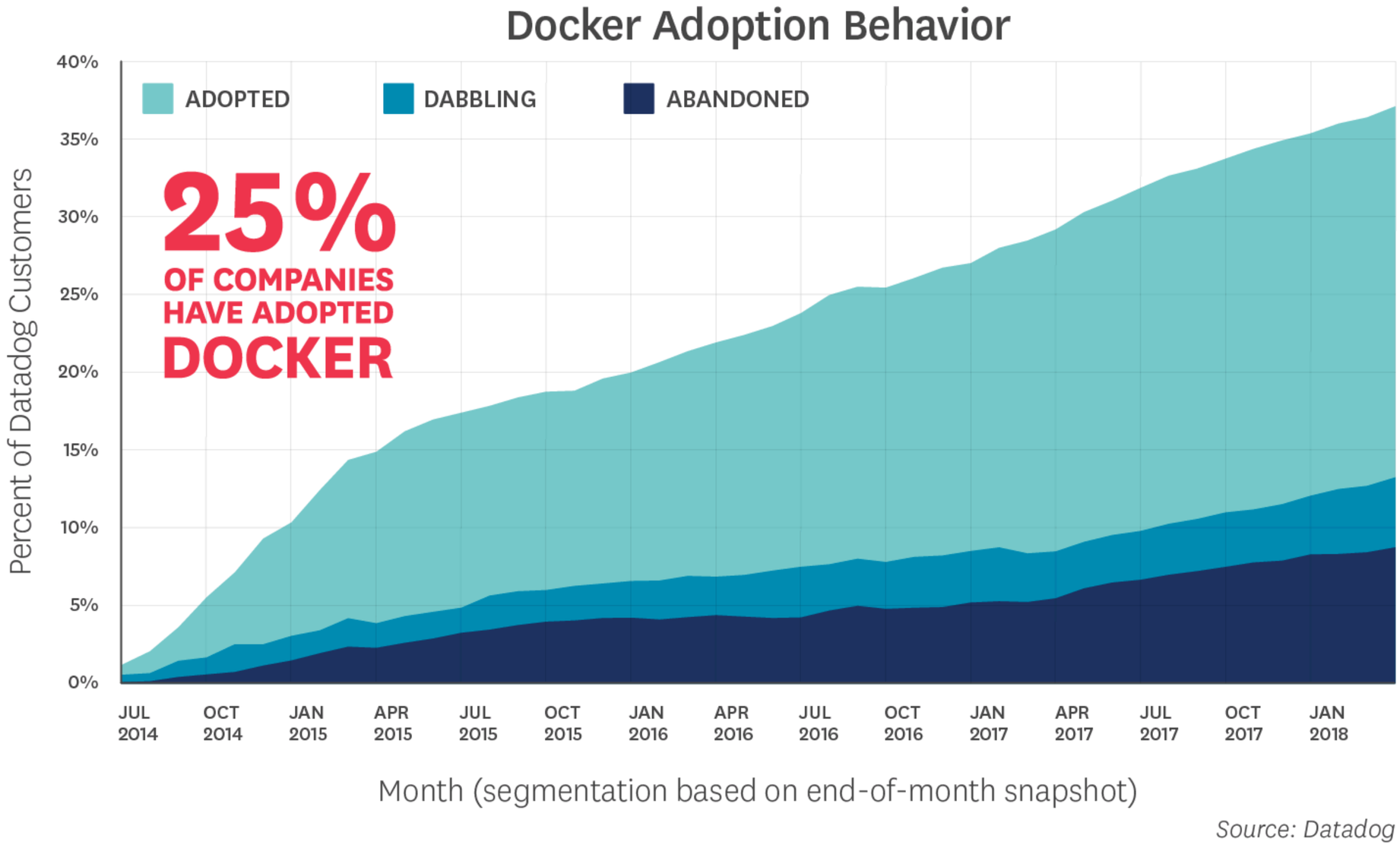
# Container Adoption

# Container Adoption has Grown 75% in 2018



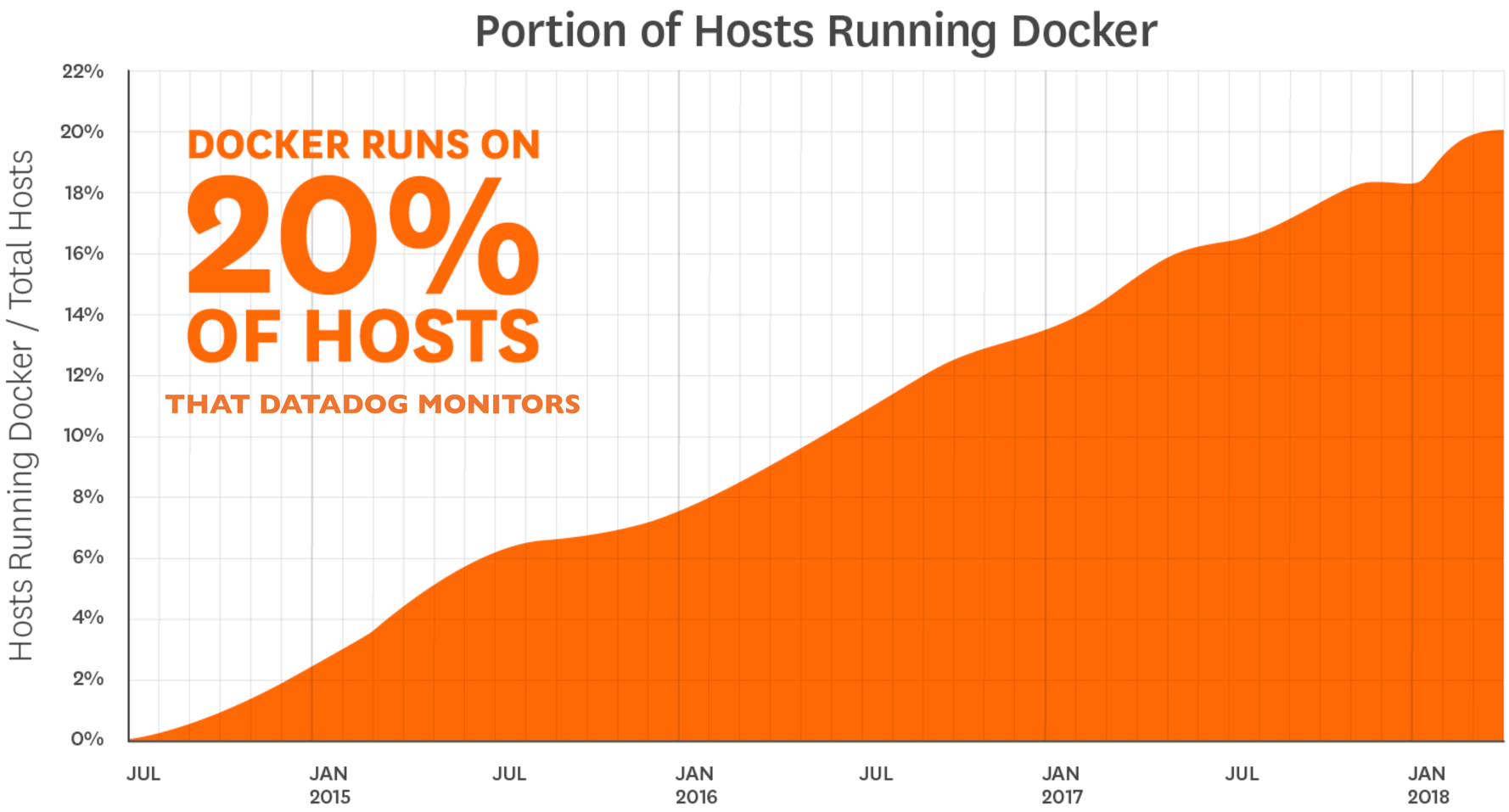

*Source: Datadog*

# Nearly One Quarter of Companies Have Adopted Docker



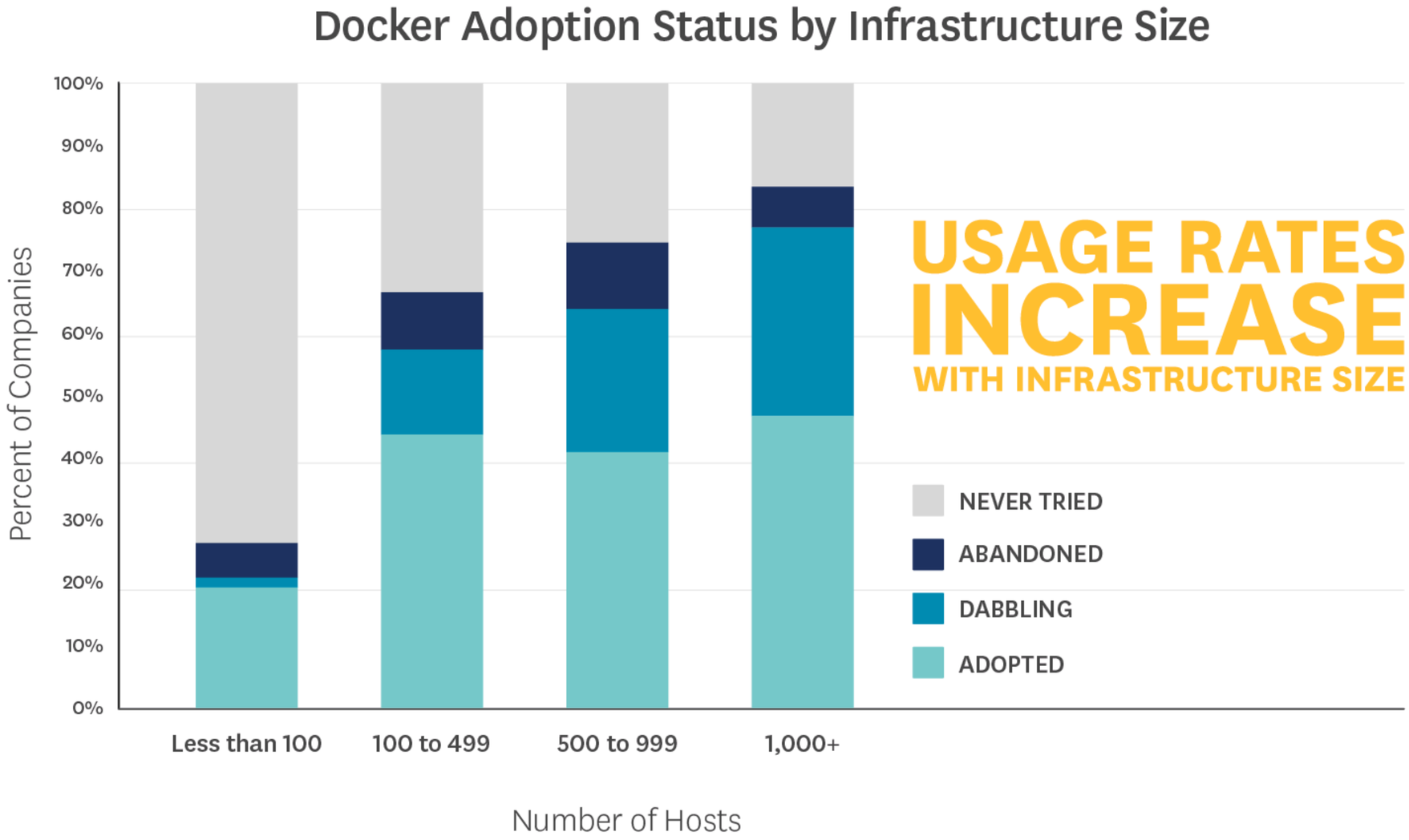

*Source: Datadog*

# Docker Now Runs on 20% of the Hosts Datadog Monitors



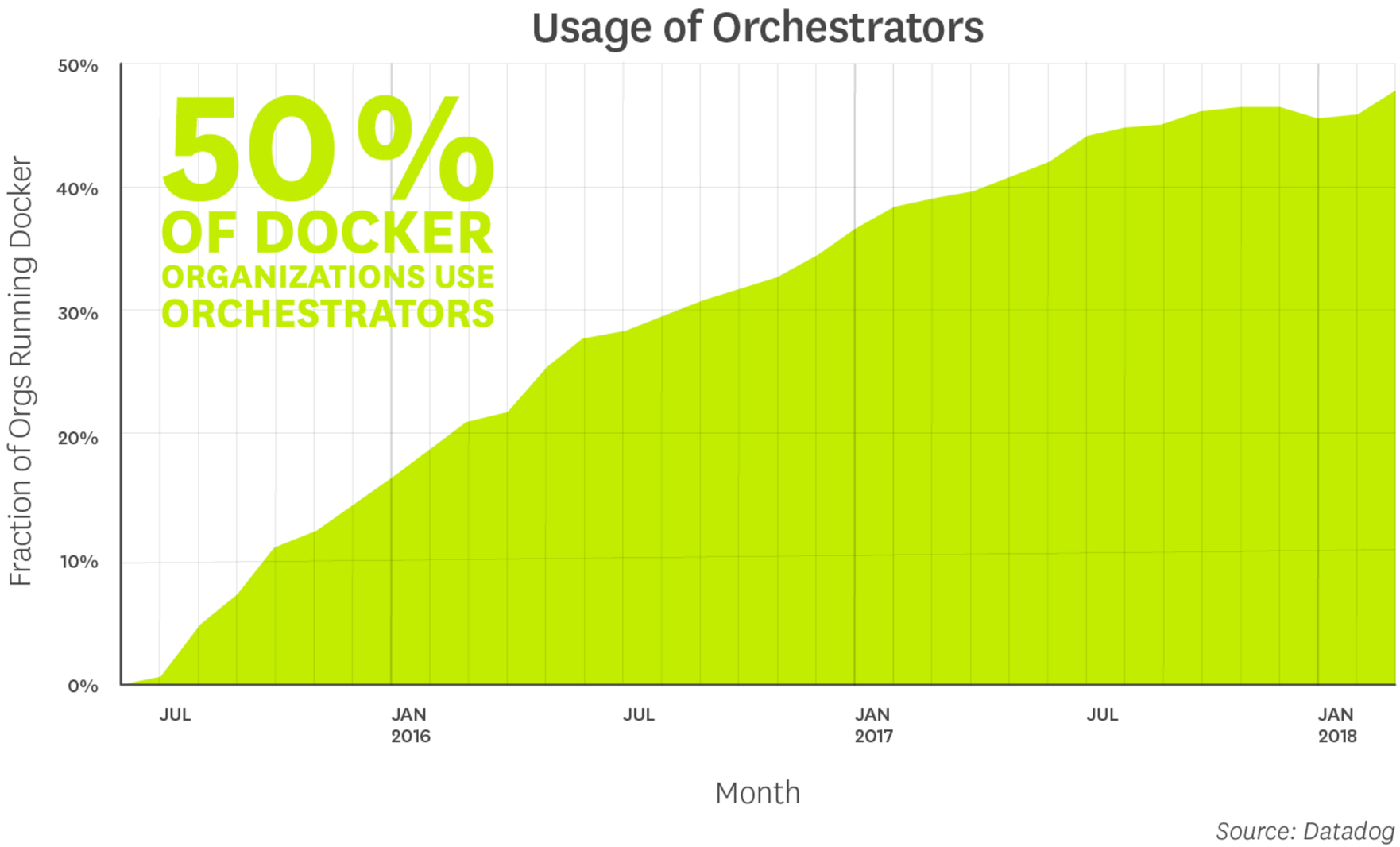

*Source: Datadog*

# Large Enterprises are Leading the Way




*Source: Datadog*

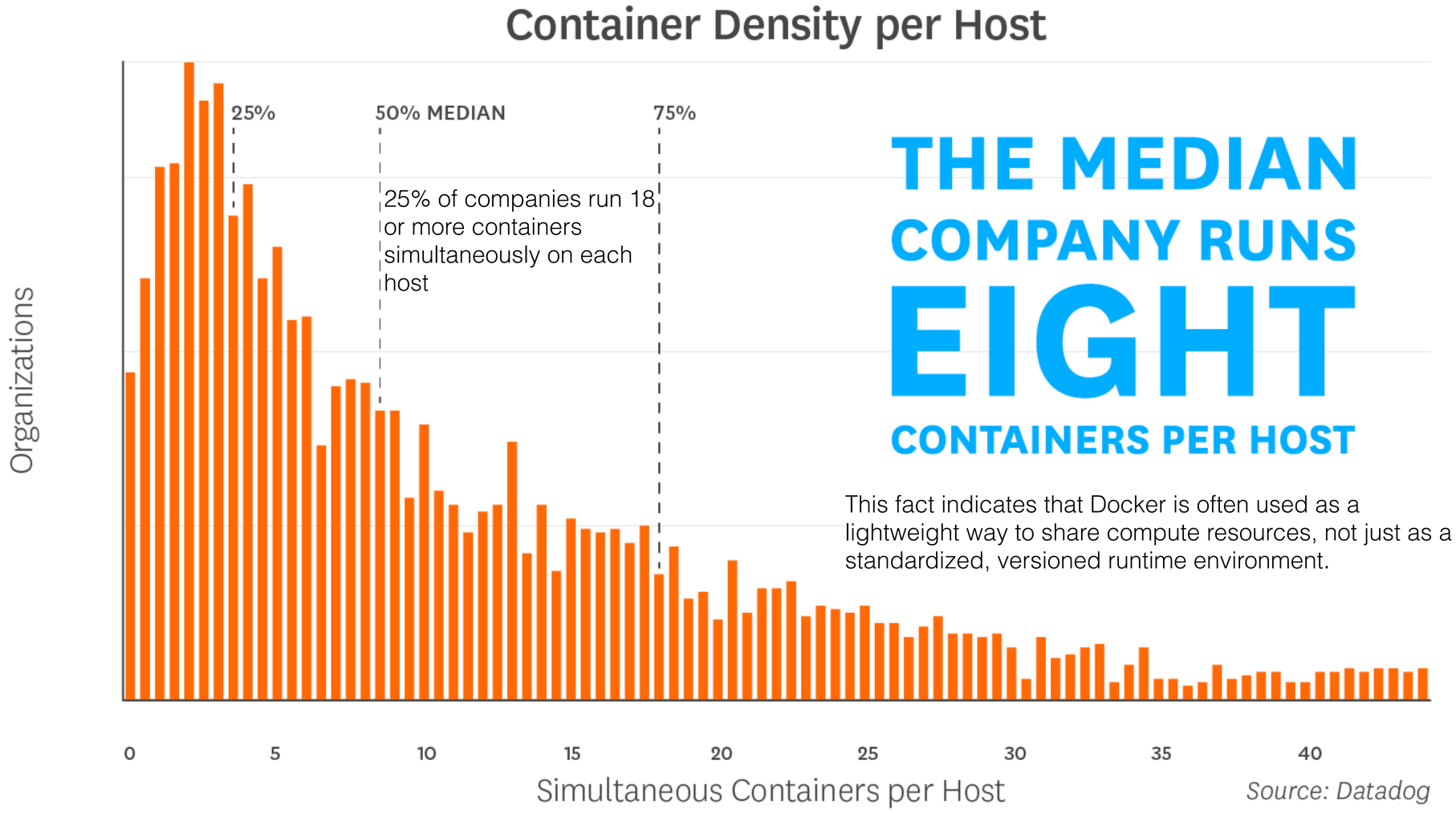
# 50% of Docker Organizations use Orchestration



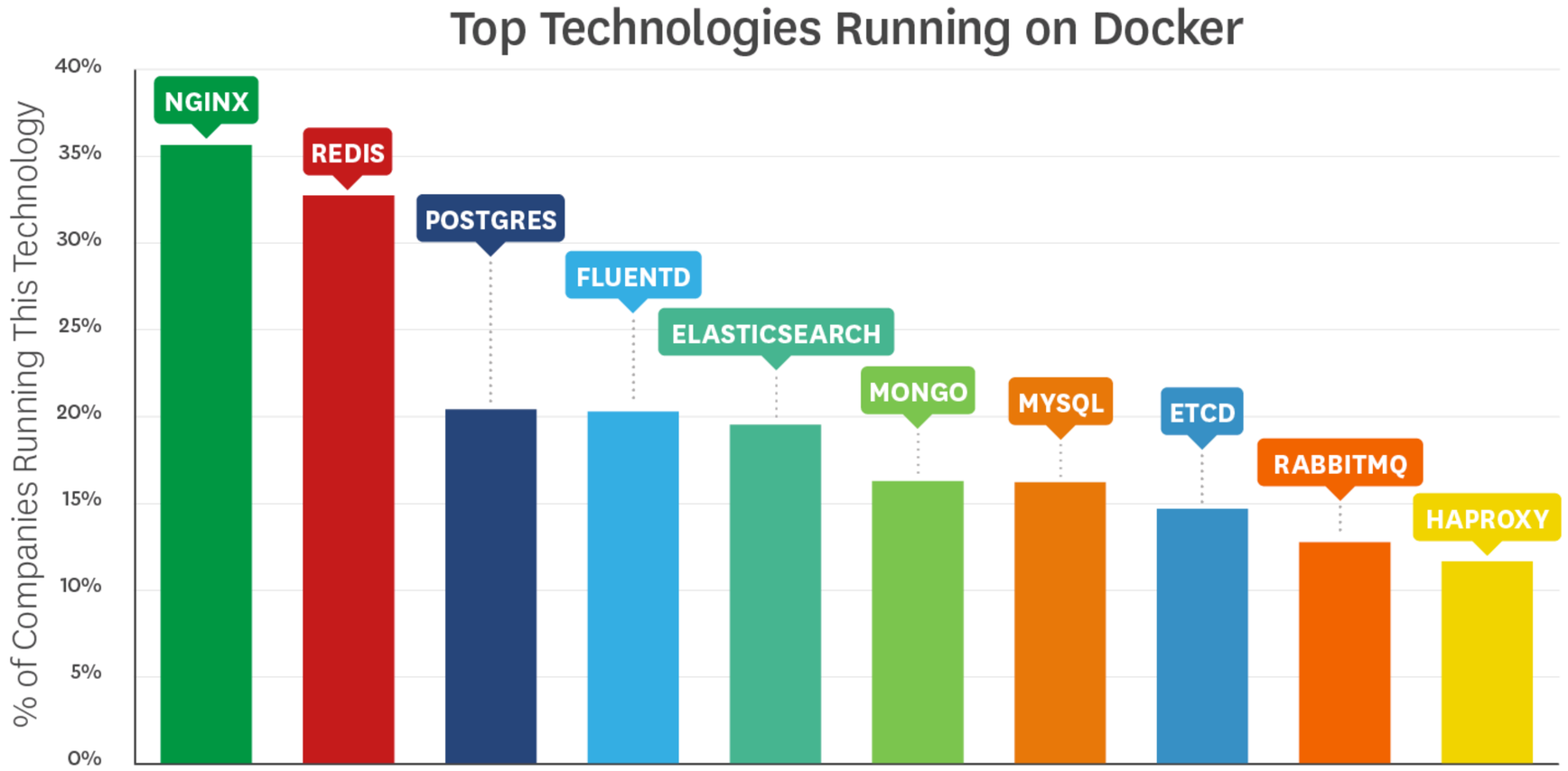

*Source: Datadog*

**Kubernetes is now the fastest-growing orchestration technology**

# The Median Docker Organization Runs Eight Containers per Host

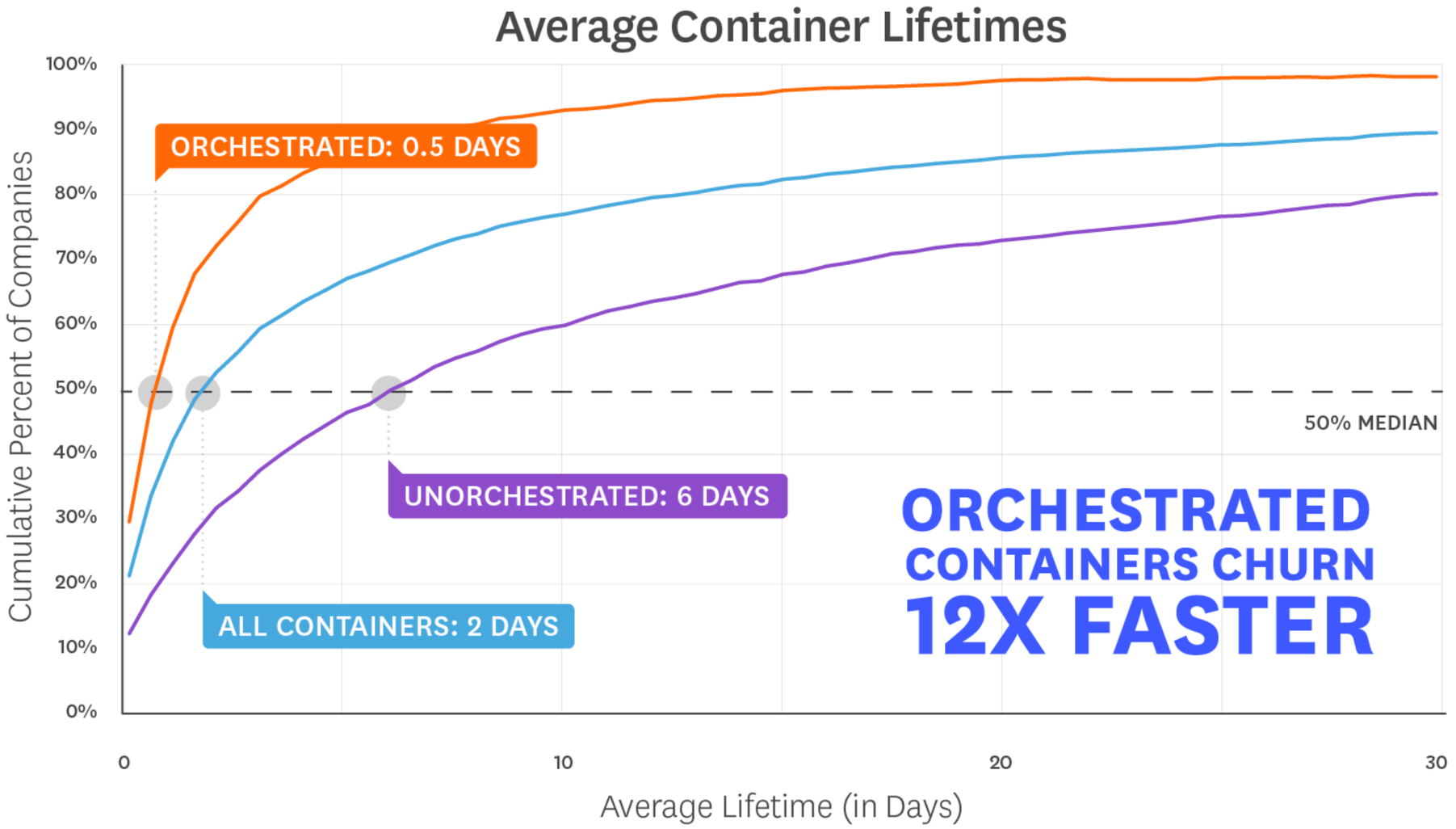


# Top 10 Image Usage




*Source: Datadog*

# The average Container lives for 2 days!




*Source: Datadog*

In organizations running an orchestrator, the typical lifetime of a container is about **12 hours**. At organizations without orchestration, the average container lives for **6 days**.

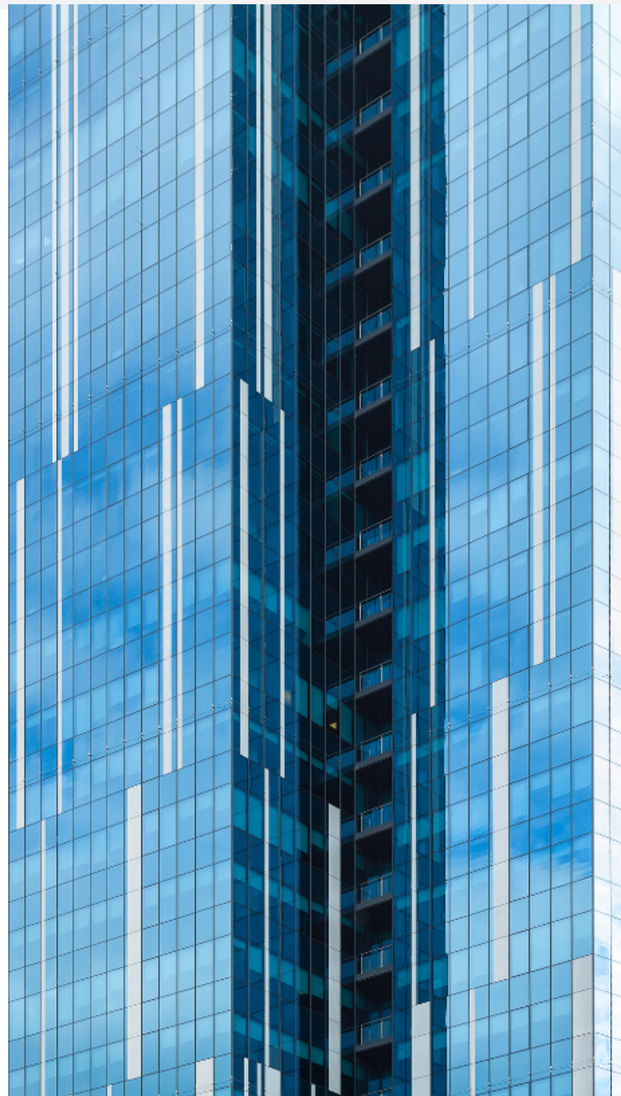
# Remember: Containers are Cattle and not Pets

- Pets are given names like pussinboots.cern.sh
- They are unique, lovingly hand raised and cared for
- When they get ill, you nurse them back to health

- Cattle are given numbers like vm0042.cern.ch
- They are almost identical to other cattle
- When they get ill, you get another one

# Containers are:

- **Flexible**: Even the most complex applications can be containerized
- **Lightweight**: Containers leverage and share the host kernel
- **Interchangeable**: You can deploy updates and upgrades on-the-fly
- **Portable**: You can build locally, deploy to the cloud, and run anywhere
- **Scalable**: You can increase and automatically distribute container replicas

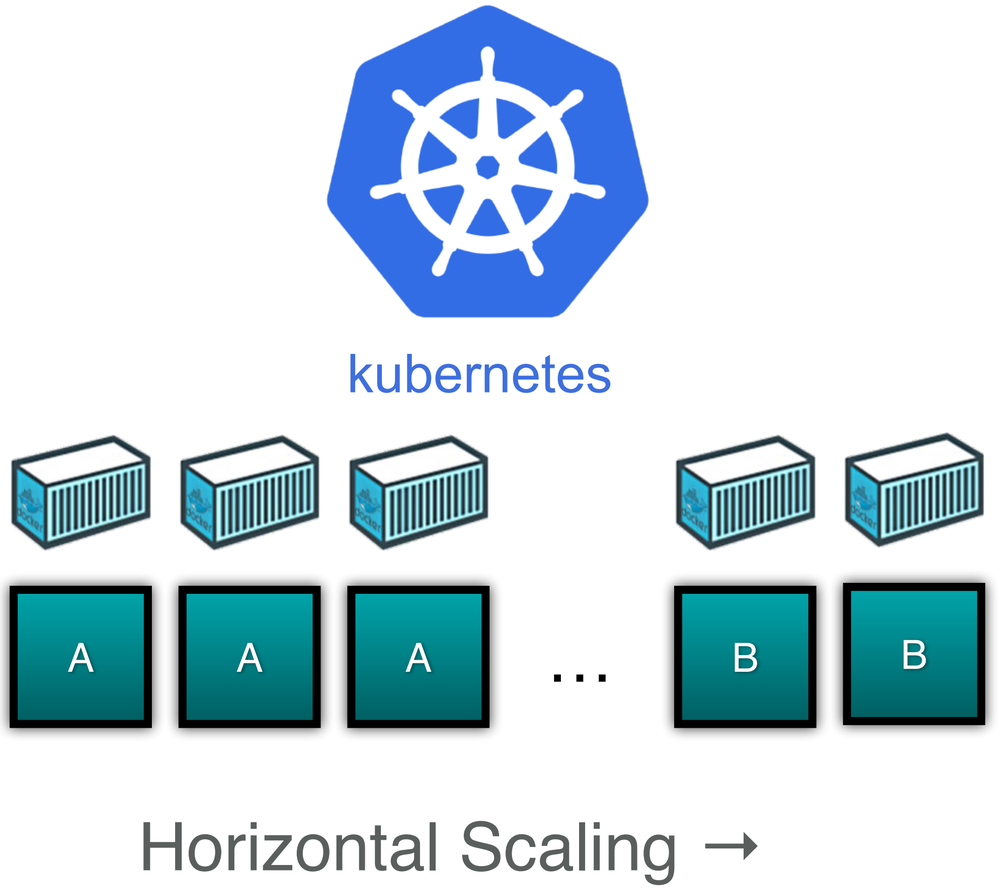




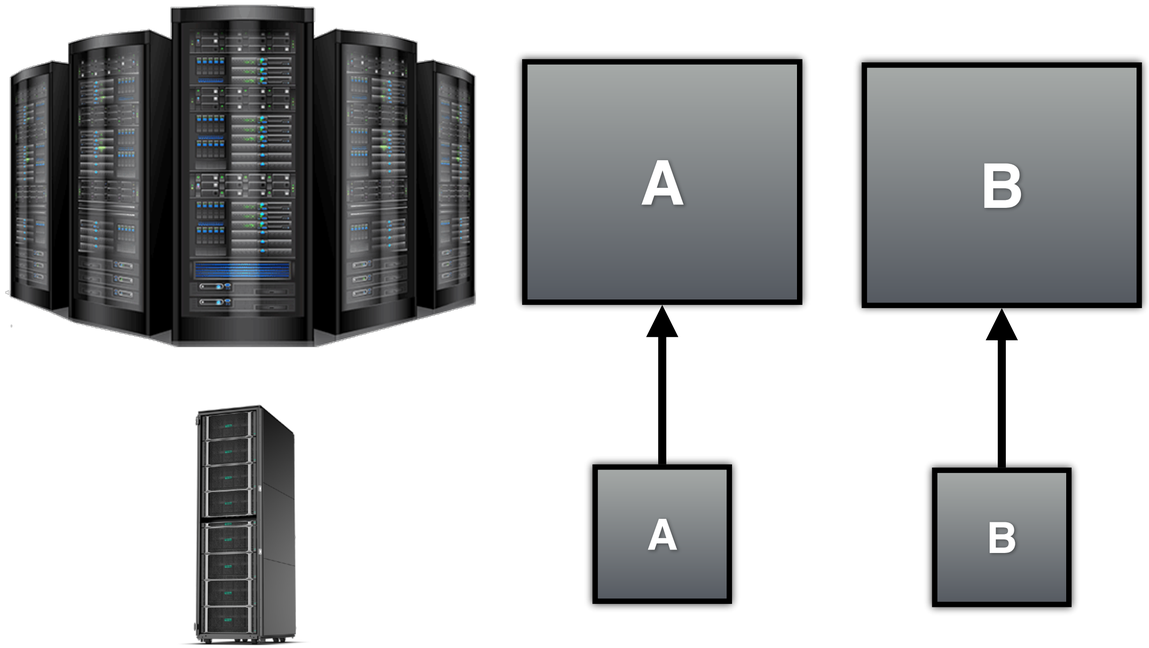
https://docs.docker.com/get-started/#images-and-containers

# Containers are Scaled on Demand

- Cloud Native Microservices use redundancy for resiliency
- Containers are spun up as needed and destroyed when no longer needed
- This why containers should be immutable and stateless

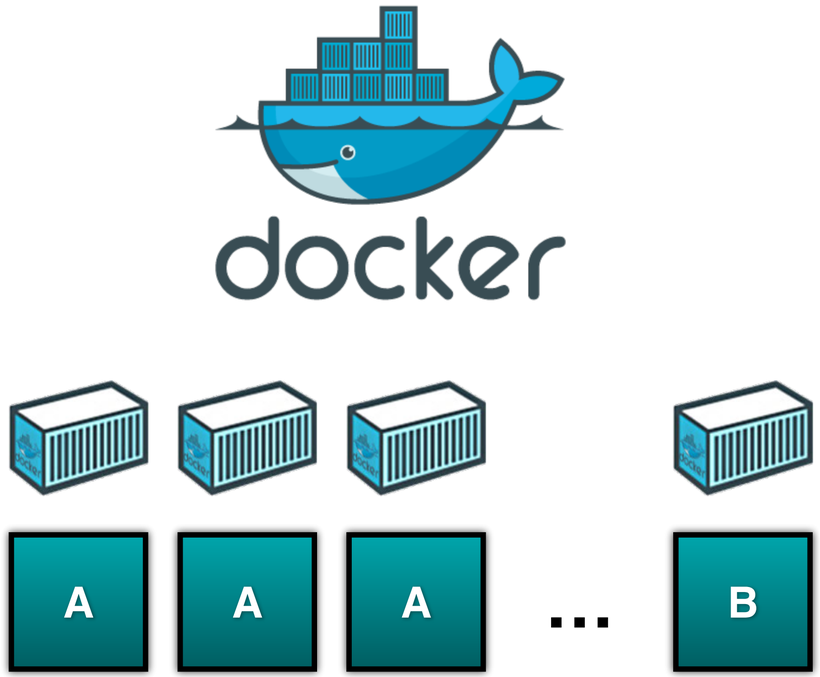


# Vertical vs Horizontal Scalling




**Vertical Scaling ↑**

Make servers bigger

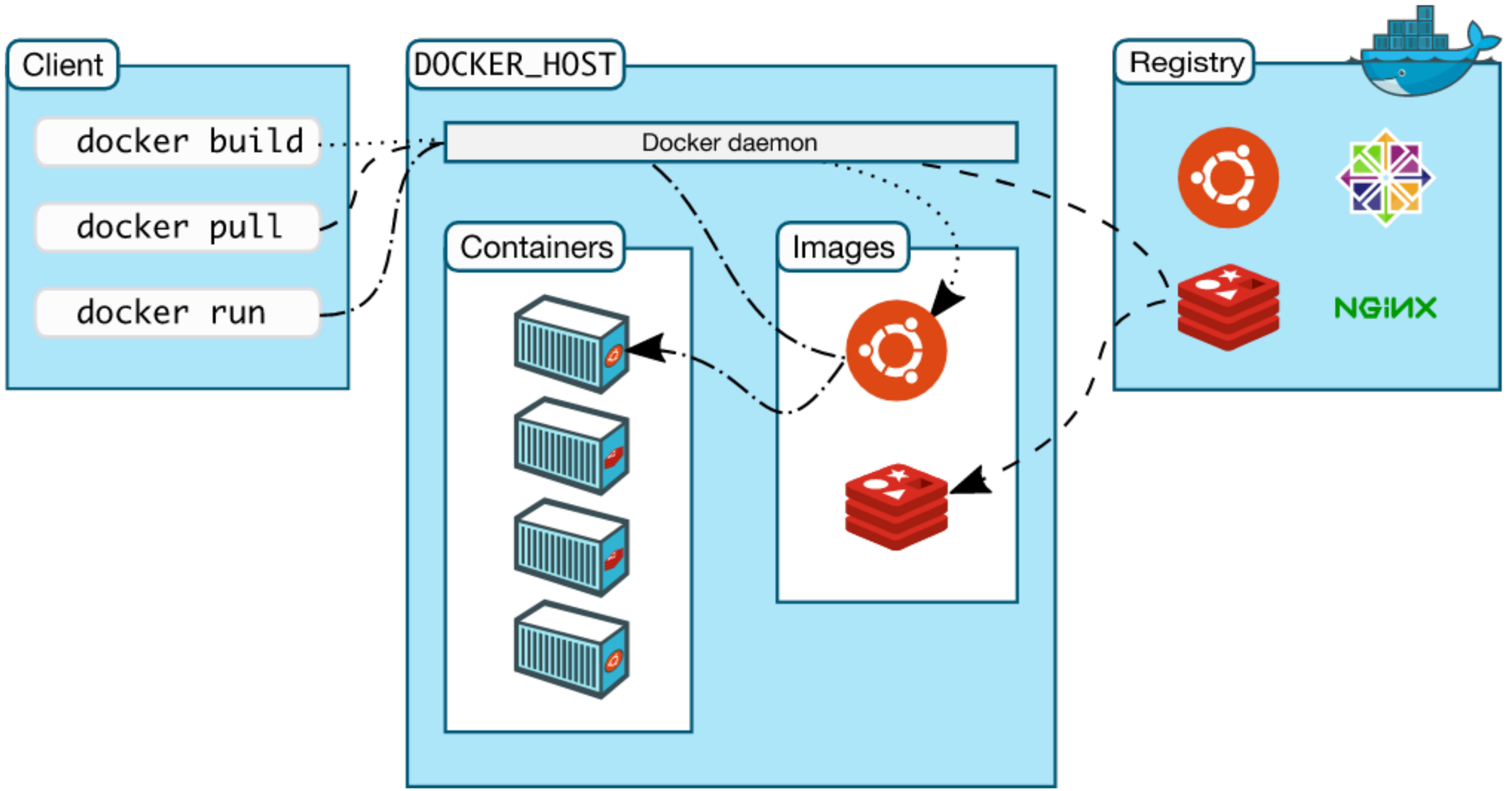



**Horizontal Scaling →**

Make more servers

# Containers in Practice

# Where do Containers Come From?



# Images and containers

- A container is launched by running an image
- An **image** is an executable package that includes everything needed to run an application--the code, a runtime, libraries, environment variables, and configuration files
- A **container** is a *runtime instance* of an image i.e., what the image becomes in memory when executed (that is, an image with state, or a user process)
- You can see a list of your running containers with the command, `docker ps`, just as you would in Linux

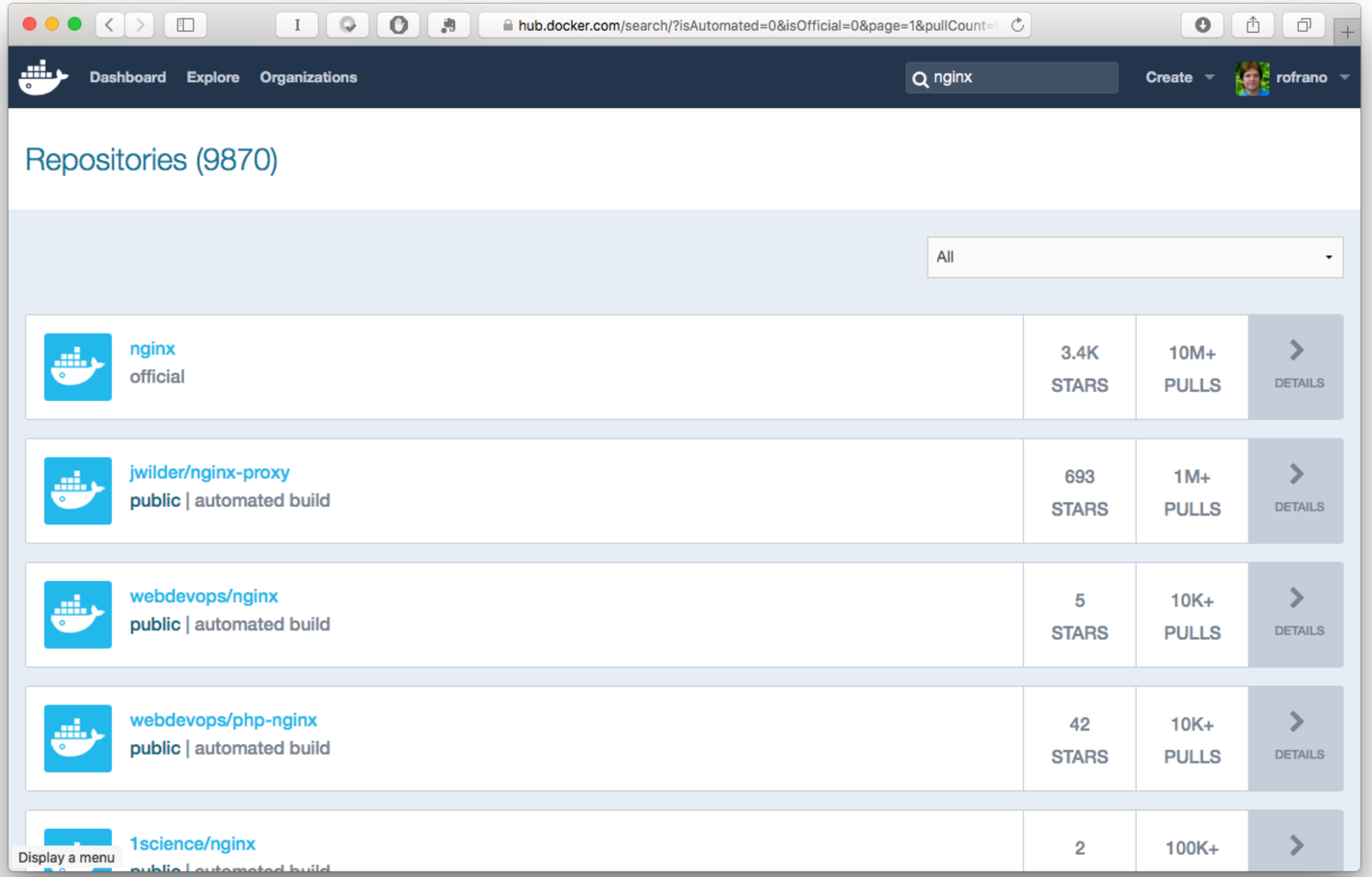


https://docs.docker.com/get-started/#images-and-containers

# Docker Hub Hosts Official and Community Images

http://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



# Images Include Documentation

## Documentation

- Each image has documentation on how to use it
- From simply running the container
- To forwarding ports, mapping storage, etc.

NGINX

How to use this image

hosting some simple static content

```
$ docker run --name some-nginx -v /some/content:/usr/share/nginx/html:ro -d nginx
```

Alternatively, a simple `Dockerfile` can be used to generate a new image that includes the necessary content (which is a much cleaner solution than the bind mount above):

```
FROM nginx
COPY static-html-directory /usr/share/nginx/html
```

Place this file in the same directory as your directory of content ("static-html-directory"), run `docker build -t some-content-nginx .`, then start your container:

```
$ docker run --name some-nginx -d some-content-nginx
```

exposing the port

```
$ docker run --name some-nginx -d -p 8080:80 some-content-nginx
```

Then you can hit `http://localhost:8080` or `http://host-ip:8080` in your browser.

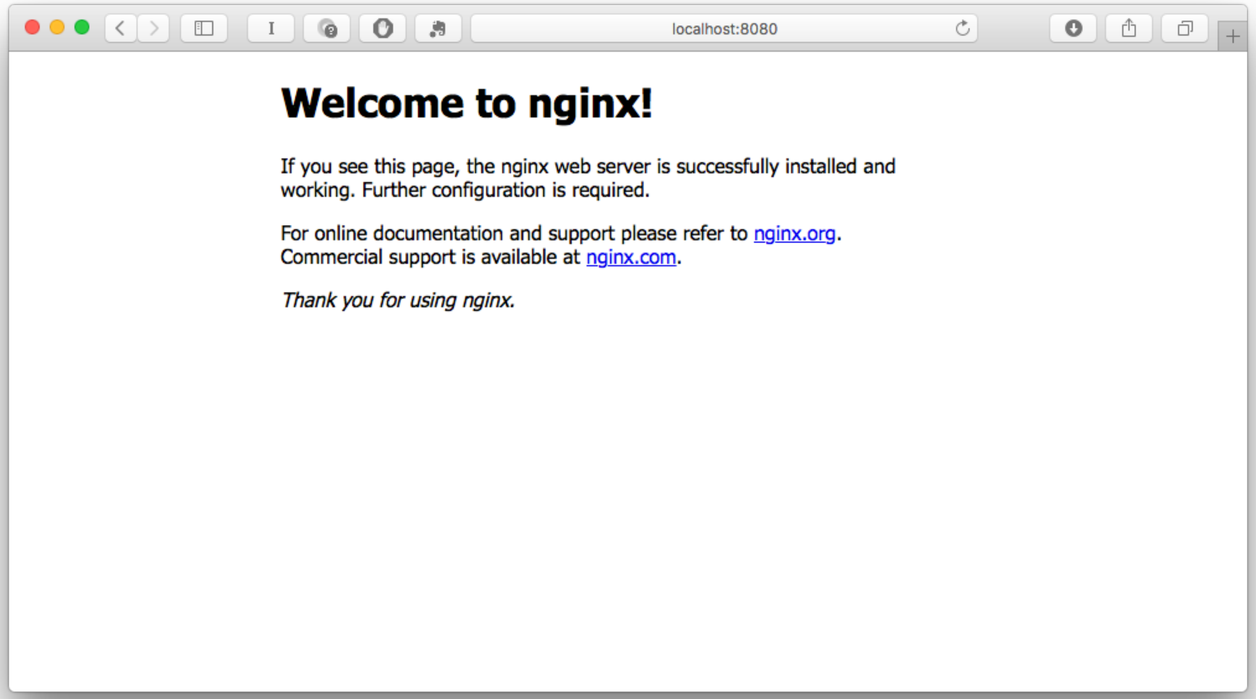
# Nginx installed and running with one command

We can run nginx as a web server with the command:

```
$ docker run -d -p 8080:80 nginx:alpine
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
6a5a5368e0c2: Pull complete
20a0fbbae148: Pull complete
2fbd37c8684b: Pull complete
Digest:
sha256:e40499ca855c9edfb212e1c3ee1a6ba8b2d873a294d897b4840d49f94d20487c
Status: Downloaded newer image for nginx:latest
0d48962ddc380c421851fb808b9b3007c0c9c614bd08ae7e732955ddaa4c7b4a
```

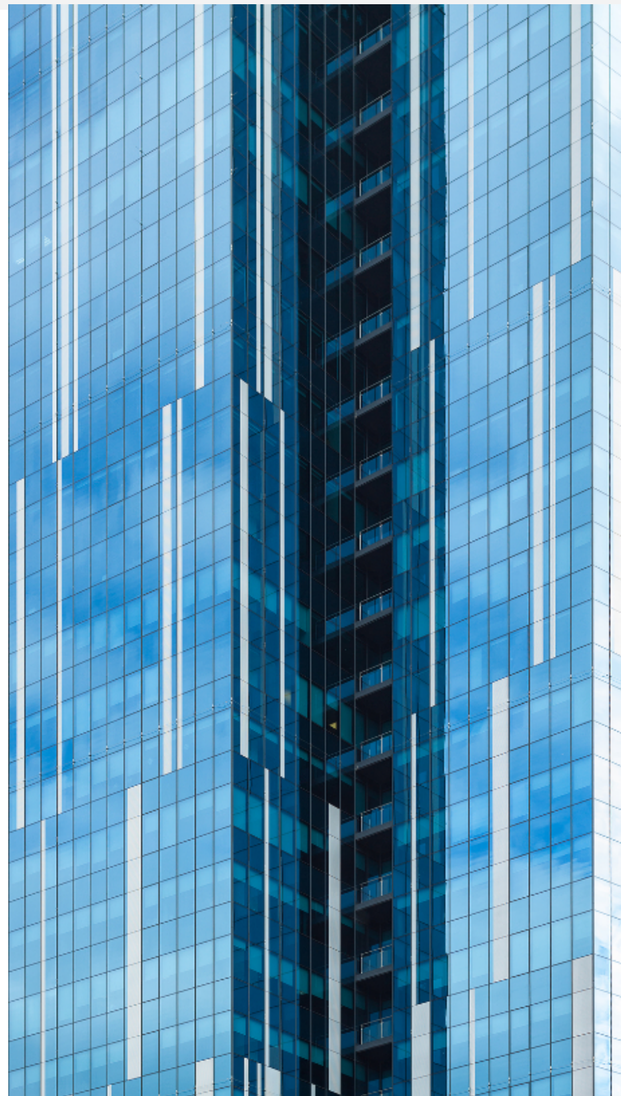
Since we don’t have a local nginx image it will be pulled from Docker hub the first time

Nginx running in a container



# Docker Provides Portable Isolated Environments

- Docker gives you portable isolated environments
- You could have an application that requires Python 3.6 running in one container
- With an application that requires Python 2.7 running in another
- Completely isolated from each other will no chance of library collisions



# Wordpress and MySQL Traditional Installation

## Famous 5-Minute Installation

Here's the quick version of the instructions for those who are already comfortable with performing such installations. More detailed instructions follow.

If you are not comfortable with renaming files, step 3 is optional and you can skip it as the install program will create the `wp-config.php` file for you.

1. Download and unzip the WordPress package if you haven't already.
2. Create a database for WordPress on your web server, as well as a MySQL (or MariaDB) user who has all privileges for accessing and modifying it.
3. (Optional) Find and rename `wp-config-sample.php` to `wp-config.php`, then edit the file (see Editing wp-config.php) and add your database information.
4. Upload the WordPress files to the desired location on your web server:
   - If you want to integrate WordPress into the root of your domain (e.g. `http://example.com/`), move or upload all contents of the unzipped WordPress directory (excluding the WordPress directory itself) into the root directory of your web server.
   - If you want to have your WordPress installation in its own subdirectory on your website (e.g. `http://example.com/blog/`), create the `blog` directory on your server and upload the contents of the unzipped WordPress package to the directory via FTP.
   - ***Note:*** *If your FTP client has an option to convert file names to lower case, make sure it's disabled.*
5. Run the WordPress installation script by accessing the URL in a web browser. This should be the URL where you uploaded the WordPress files.
   - If you installed WordPress in the root directory, you should visit: `http://example.com/`
   - If you installed WordPress in its own subdirectory called `blog`, for example, you should visit: `http://example.com/blog/`

That's it! WordPress should now be installed.

# Wordpress and MySQL Traditional Installation

## Famous 5-Minute Installation

Here's the quick version of the instructions for those who are already comfortable with performing such installations. More detailed instructions follow.

If you are not comfortable with renaming files, step 3 is optional and you can skip it as the install program will create the `wp-config.php` file for you.

1. Download and unzip the WordPress package if you haven't already.
2. Create a database for WordPress on your web server, as well as a MySQL (or MariaDB) user who has all privileges for accessing and modifying it.
3. (Optional) Find and rename `wp-config-sample.php` to `wp-config.php`, then edit the file (see Editing wp-config.php) and add your database information.
4. Upload the WordPress files to the desired location on your web server:
   - If you want to integrate WordPress into the root of your domain (e.g. `http://example.com/`), move or upload all contents of the unzipped WordPress directory (excluding the WordPress directory itself) into the root directory of your web server.
   - If you want to have your WordPress installation in its own subdirectory on your website (e.g. `http://example.com/blog/`), create the `blog` directory on your server and upload the contents of the unzipped WordPress package to the directory via FTP.
   - ***Note:*** *If your FTP client has an option to convert file names to lower case, make sure it's disabled.*
5. Run the WordPress installation script by accessing the URL in a web browser. This should be the URL where you uploaded the WordPress files.
   - If you installed WordPress in the root directory, you should visit: `http://example.com/`
   - If you installed WordPress in its own subdirectory called `blog`, for example, you should visit: `http://example.com/blog/`

That's it! WordPress should now be installed.

## Steps for a Fresh Installation of MySQL

### 1. Adding the MySQL APT Repository

First, add the MySQL APT repository to your system's software repository list. Follow these steps:

- Go to the download page for the MySQL APT repository at https://dev.mysql.com/downloads/repo/apt/.
- Select and download the release package for your Linux distribution.
- Install the downloaded release package with the following command, replacing ***`version-specific-package-name`*** with the name of the downloaded package (preceded by its path, if you are not running the command inside the folder where the package is):

```
shell> sudo dpkg -i /PATH/version-specific-package-name.deb
```

For example, for version ***`w.x.y-z`*** of the package, the command is:

```
shell> sudo dpkg -i mysql-apt-config_w.x.y-z_all.deb
```

Note that the same package works on all supported Debian and Ubuntu platforms.

- During the installation of the package, you will be asked to choose the versions of the MySQL server and other components (for example, the MySQL Workbench) that you want to install. If you are not sure which version to choose, do not change the default options selected for you. You can also choose **none** if you do not want a particular component to be installed. After making the choices for all components, choose **Ok** to finish the configuration and installation of the release package.
  You can always change your choices for the versions later; see Selecting a Major Release Version for instructions.
- Update package information from the MySQL APT repository with the following command (*this step is mandatory*):

```
shell> sudo apt-get update
```

### 2. Installing MySQL with APT

Install MySQL by the following command:

```
shell> sudo apt-get install mysql-server
```

This installs the package for the MySQL server, as well as the packages for the client and for the database common files. During the installation, you are asked to supply a password for the root user for your MySQL installation.

### 3. Starting and Stopping the MySQL Server

The MySQL server is started automatically after installation. You can check the status of the MySQL server with the following command:

```
shell> sudo service mysql status
```

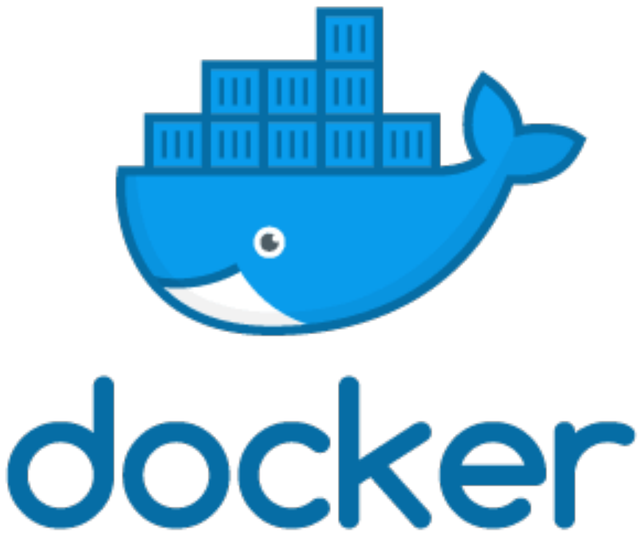
Stop the MySQL server with the following command:

```
shell> sudo service mysql stop
```

To restart the MySQL server, use the following command:

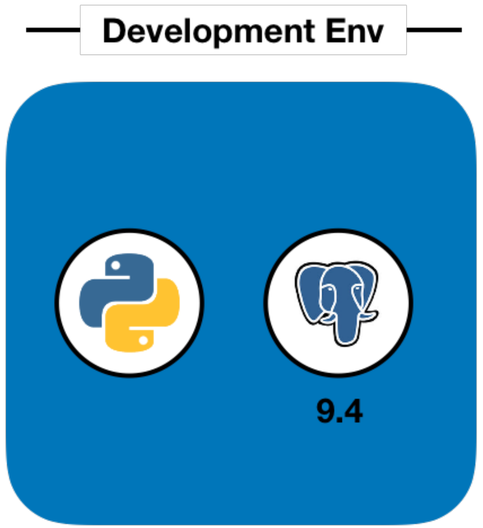
```
shell> sudo service mysql start
```

# Wordpress and MySQL with Docker

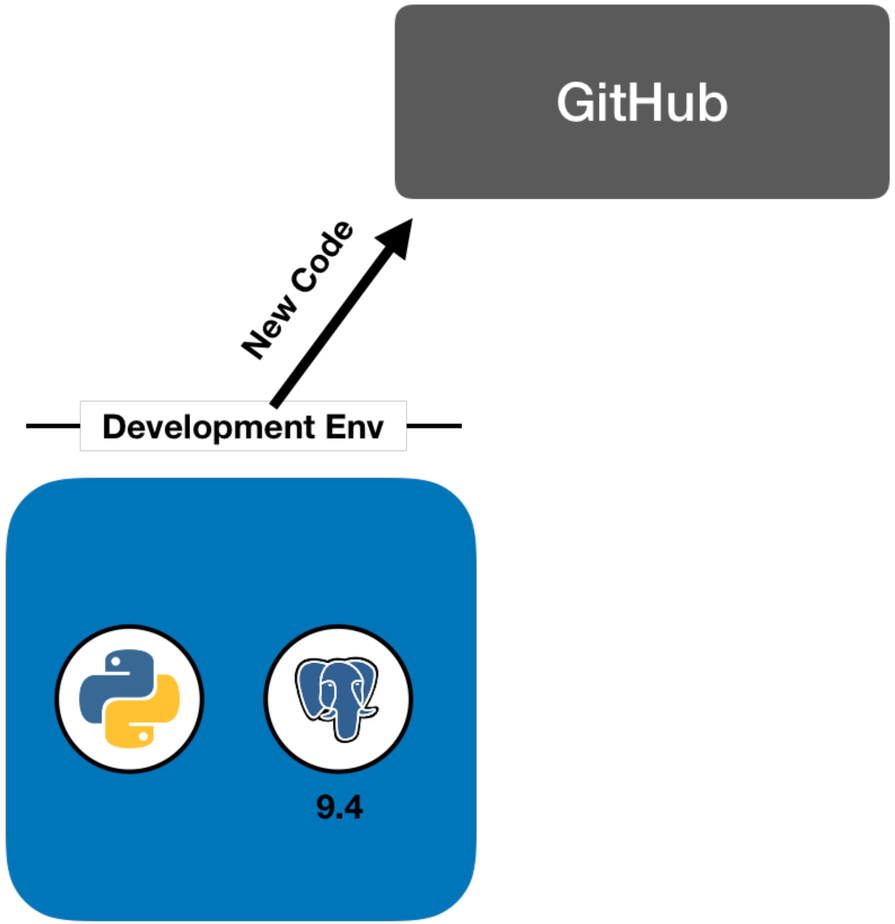



```
docker run -d --name db mysql

docker run -d --link db -p 80:80 wordpress
```

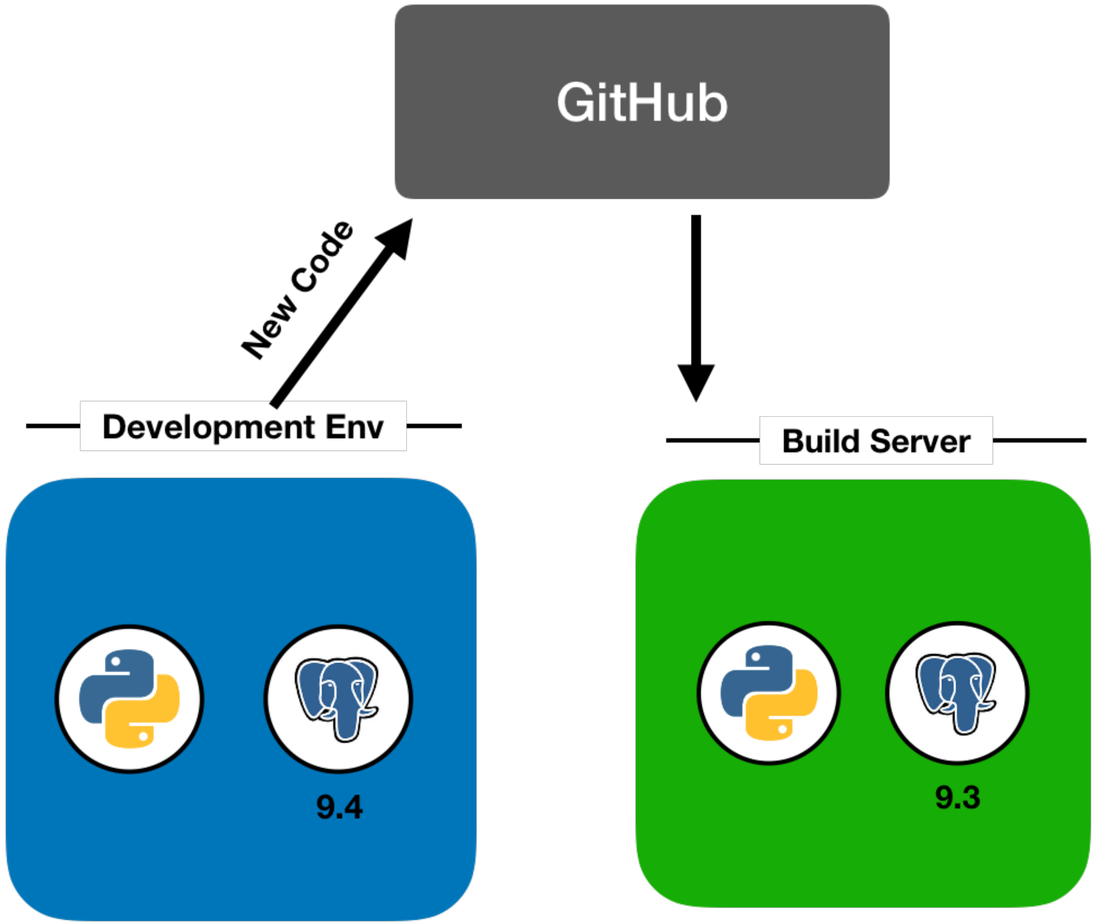
# Traditional Development Workflow



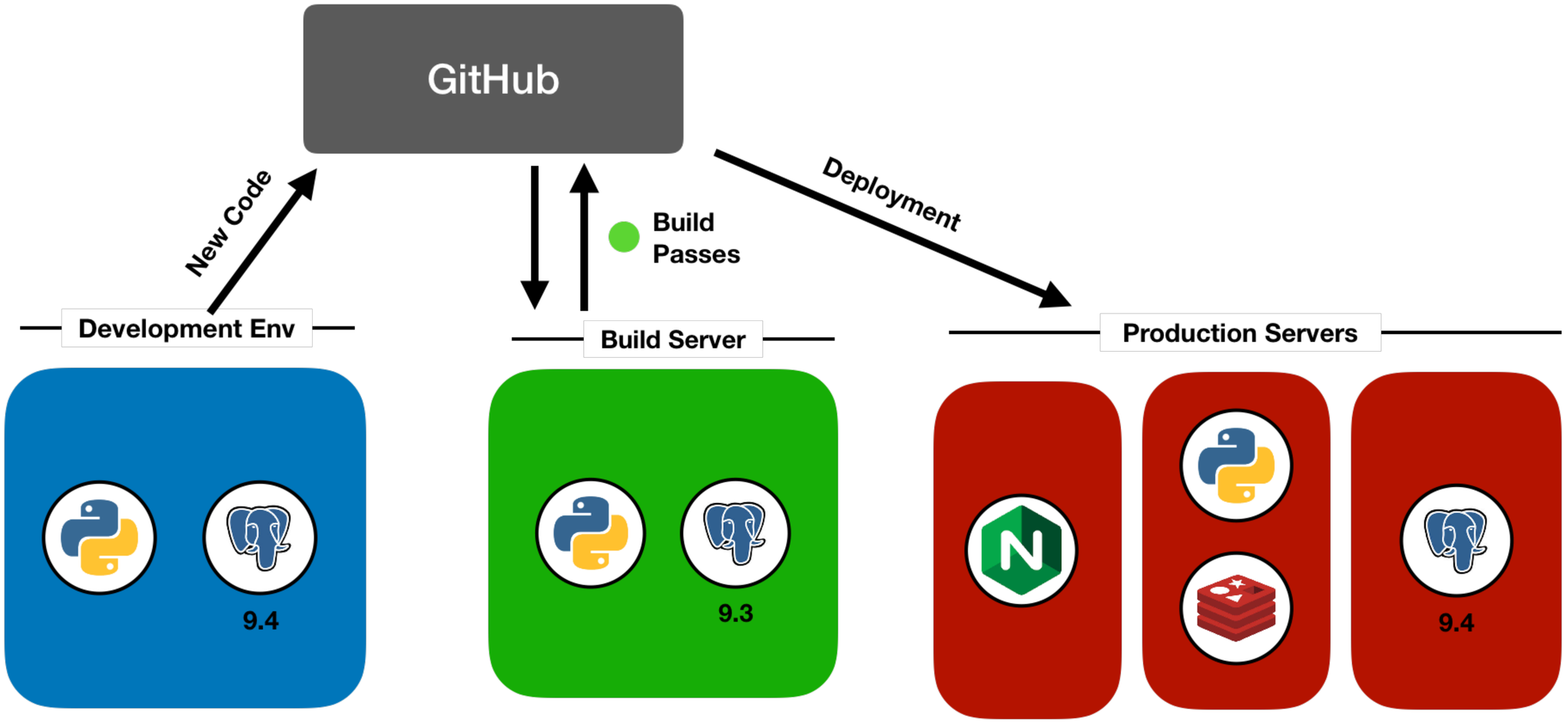
# Traditional Development Workflow



# Traditional Development Workflow

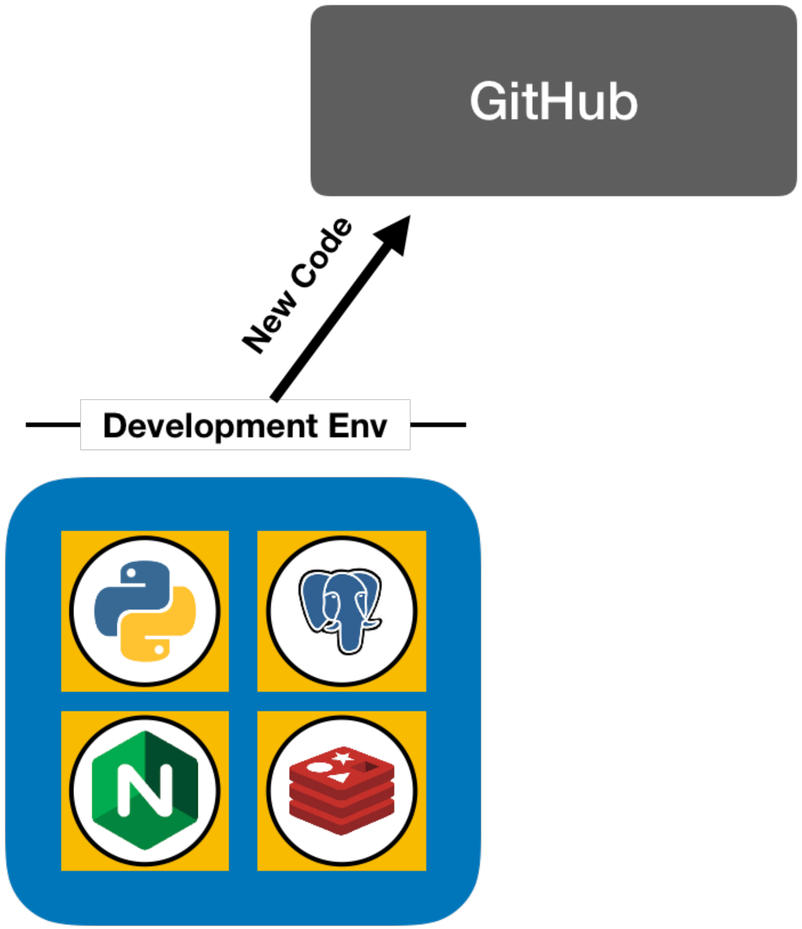


# Traditional Development Workflow

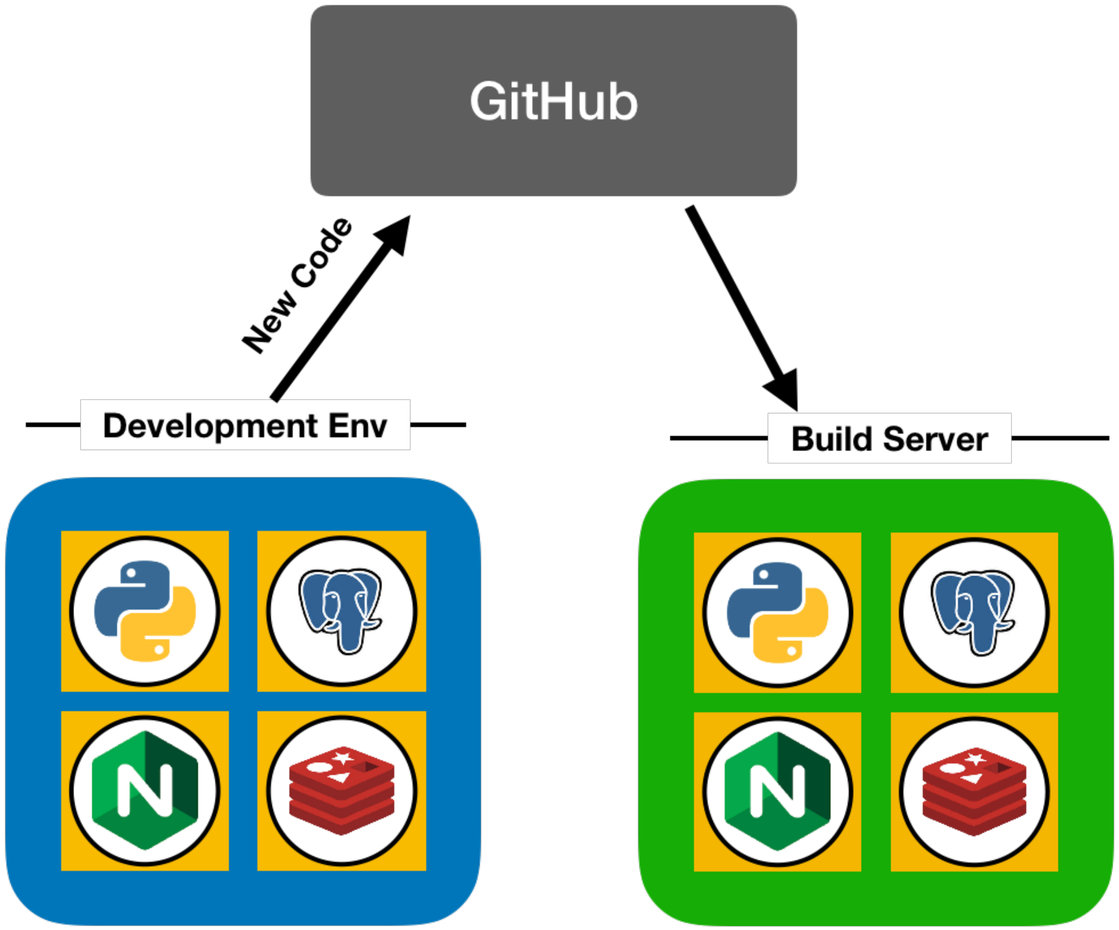


# Containerized Development Workflow

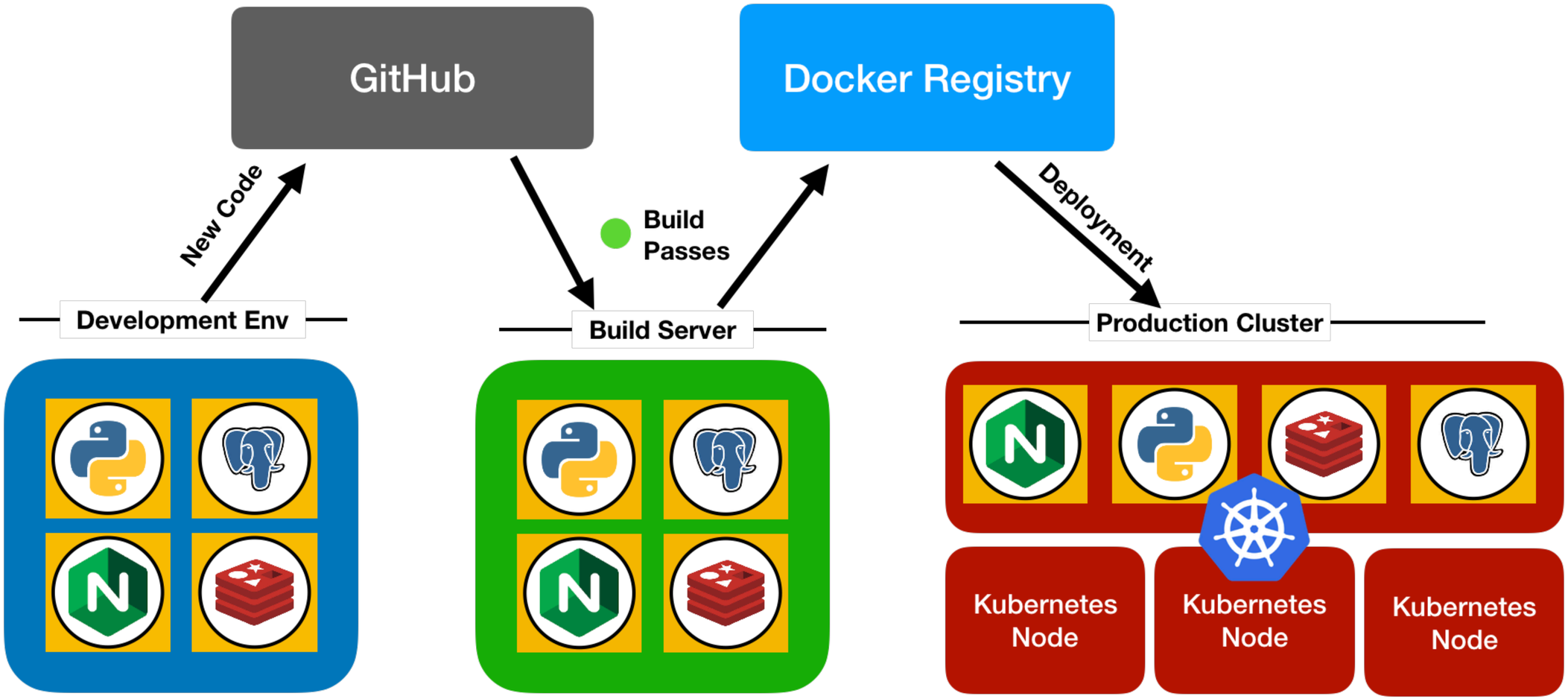
# Containerized Development Workflow



# Containerized Development Workflow

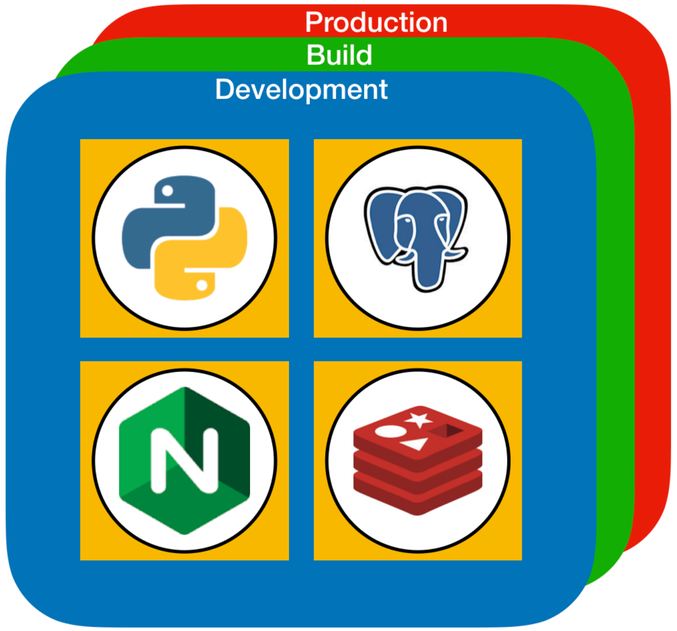


# Containerized Development Workflow



# This is a Very Powerful Concept

- Significant time is spent trying to make development, test, and production environments the same
- Docker allow the same container that is built and tested in development to run unchanged in production
- You can literally deploy an entire environment with a single command



# Creating Your Own Docker Images

# Images from Dockerfiles

- Whatever is needed to build the image is specified in the Dockerfile and checked into GitHub ("infrastructure as code")
- Anyone who uses this image is guaranteed to get the same service instance
- You can start from an operating system image and install all of the prerequisites just like on a VM
- Then copy your application code and resolve and library dependencies
- Finally add the command to run your application

```
FROM alpine:3.7
MAINTAINER John Rofrano "rofrano@us.ibm.com"

# Install just the Python runtime (no dev)
RUN apk add --no-cache \
    python \
    py-pip \
    ca-certificates && \
    pip install --upgrade pip && \
    rm -r /root/.cache

# Expose any ports the app is expecting
ENV PORT 5000
EXPOSE $PORT

# Set up a working folder and install the pre-reqs
WORKDIR /app
ADD requirements.txt /app
RUN pip install -r requirements.txt
ADD . /app

# Run the service
ENV GUNICORN_BIND 0.0.0.0:$PORT
CMD ["gunicorn", "--log-file=-", "app:app"]
```

# Docker build command

- The docker build command is used to build an image from a Dockerfile
- Docker uses a layered filesystem and will reuse layers that have previously been built and not modified
- It will even reuse layers across images
- In the example on the right, all layers except the application layer are cached resulting in a very quick build

```
$ docker build -t my-service .
Sending build context to Docker daemon  36.16MB
Step 1/11 : FROM alpine:3.7
 ---> 34ea7509dcad
Step 2/11 : MAINTAINER John Rofrano "rofrano@us.ibm.com"
 ---> Using cache
 ---> 663be28f442f
Step 3/11 : RUN apk add --no-cache     python     py-pip     ca-
certificates &&     pip install --upgrade pip &&     rm -r /root/.cache
 ---> Using cache
 ---> f0e8446bde3b
Step 4/11 : ENV PORT 5000
 ---> Using cache
 ---> e7a1719d988a
Step 5/11 : EXPOSE $PORT
 ---> Using cache
 ---> 8ec0b9957940
Step 6/11 : WORKDIR /app
 ---> Using cache
 ---> 8f77c8e95bb7
Step 7/11 : ADD requirements.txt /app
 ---> Using cache
 ---> 46873628e505
Step 8/11 : RUN pip install -r requirements.txt
 ---> Using cache
 ---> ced8d4272148
Step 9/11 : ADD . /app
 ---> b8df76d195b6
Step 10/11 : ENV GUNICORN_BIND 0.0.0.0:$PORT
 ---> Running in 20c6c979b175
Removing intermediate container 20c6c979b175
 ---> 9ae6c0c452bf
Step 11/11 : CMD ["gunicorn", "--log-file=-", "app:app"]
 ---> Running in ed5c32e4efa1
Removing intermediate container ed5c32e4efa1
 ---> e4366d80570a
Successfully built e4366d80570a
Successfully tagged my-service:latest
```

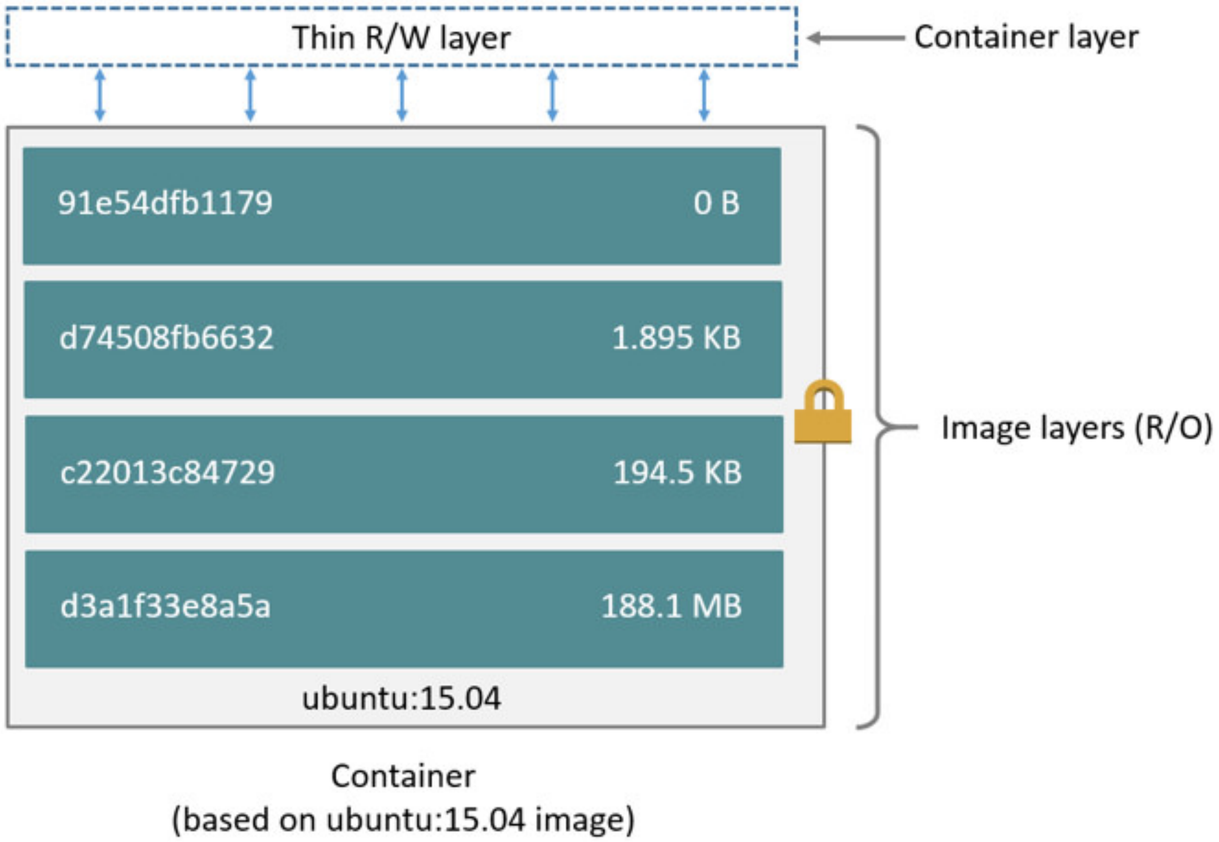
# Docker Layered Filesystem

- Docker uses a Copy-On-Write layered filesystem
  - Only changes from the read-only layers are copied
- You can see the layers when you pull or push an image

```
$ docker pull ubuntu:15.04

15.04: Pulling from library/ubuntu
1ba8ac955b97: Pull complete
f157c4e5ede7: Pull complete
0b7e98f84c4c: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:5e279a9df07990286cce22e1b0f5b0490629ca6d187698746ae5e28e604a640e
Status: Downloaded newer image for ubuntu:15.04
```

# Images and Layers

- Each Docker image references a list of read-only layers that represent filesystem differences
- Layers are stacked on top of each other to form a base for a container's root filesystem
- When you create a new container, you add a new, thin, writable layer on top of the underlying stack
- All changes made to the running container - such as writing new files, modifying existing files, and deleting files - are written to this thin writable container layer



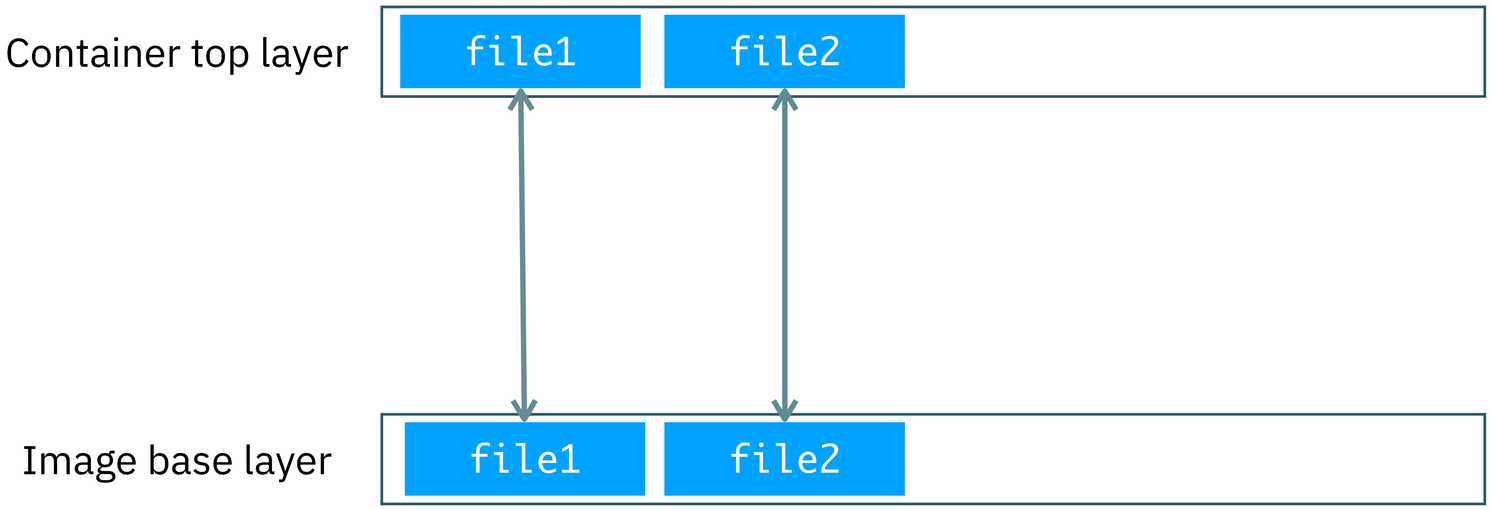
# Base Image

- Start with a base image


Docker container
(AUFS storage-driver demonstrating whiteout file)

# Container Layers are exposed to Top

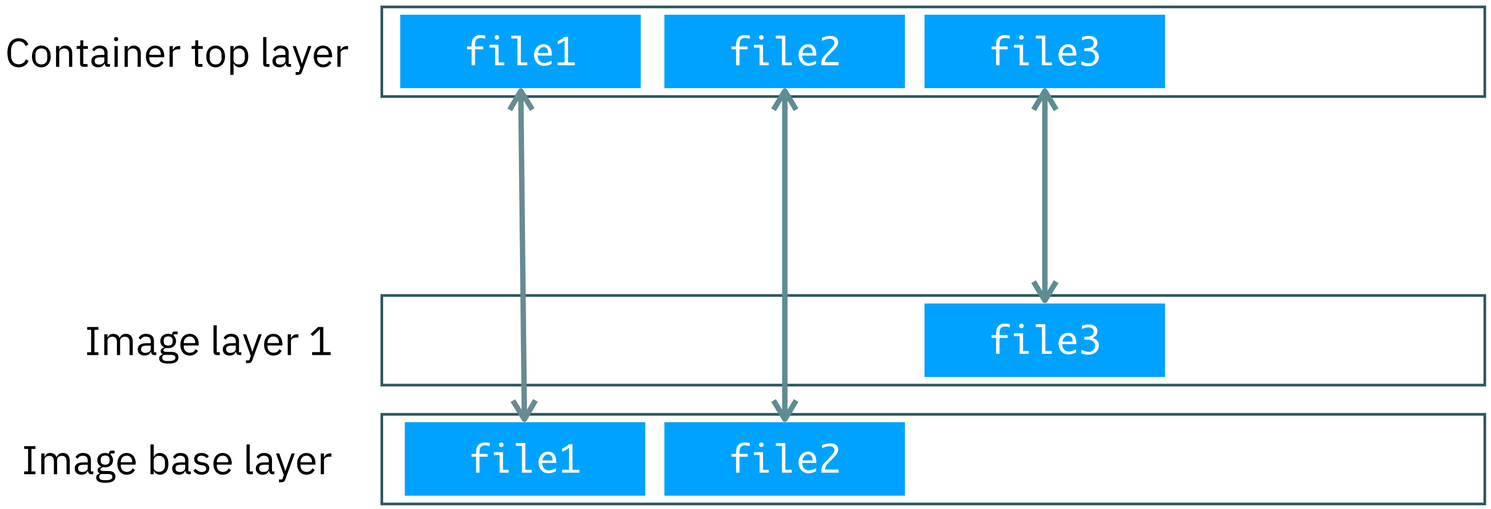
- Files from the read-only layers below are visible to the top layer




Docker container
(AUFS storage-driver demonstrating whiteout file)

# Add Layers with more Files

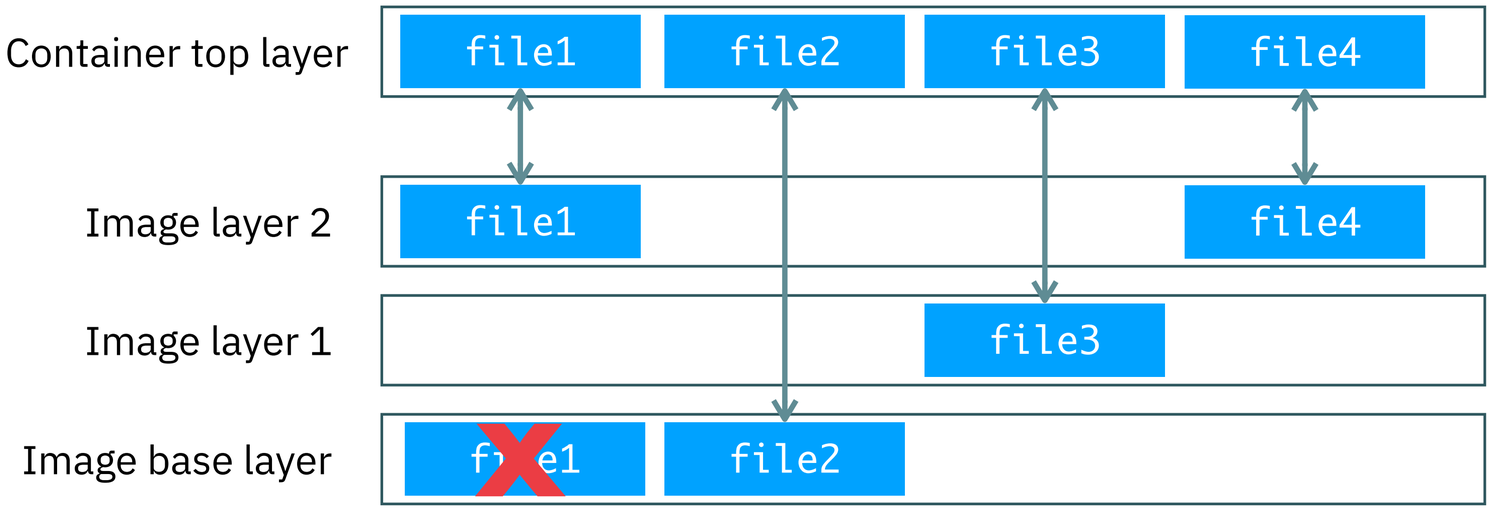
- As you add layers more files become visible at the top layer




Docker container
(AUFS storage-driver demonstrating whiteout file)

# New Versions

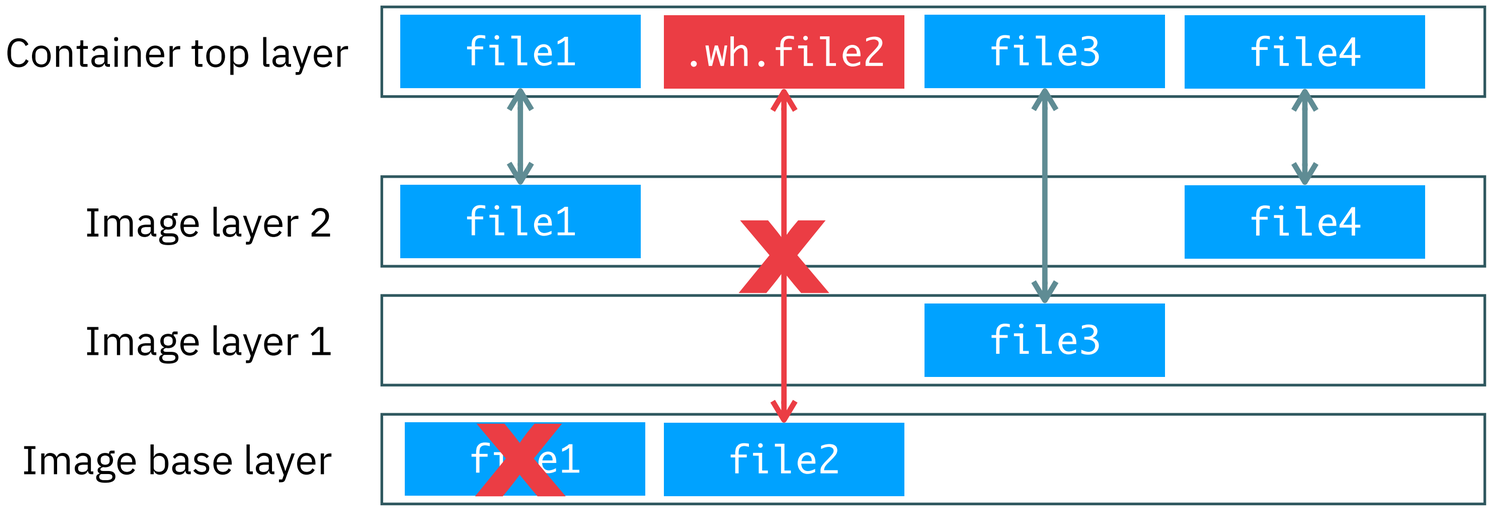
- A new version of `file1` is added and it hides the old `file1` version




Docker container
(AUFS storage-driver demonstrating whiteout file)

# White-out Files

- Special files called "white-out" files are used to make files appear to be deleted




Docker container
(AUFS storage-driver demonstrating whiteout file)

# How Layers are Reused

- Note when you provision with Vagrant how Redis:Alpine reuses Alpine layer:

```
==> default: Running provisioner: docker...
    default: Installing Docker onto machine...
==> default: Pulling Docker images...
==> default: -- Image: alpine:latest
==> default: stdin: is not a tty
==> default: latest: Pulling from library/alpine
==> default: 3690ec4760f9: Pulling fs layer
==> default: 3690ec4760f9: Verifying Checksum
==> default: 3690ec4760f9: Download complete
==> default: 3690ec4760f9: Pull complete
==> default: Digest: sha256:1354db23ff5478120c980eca1611a51c9f2b88b61f24283ee8200bf9a54f2e5c
==> default: Status: Downloaded newer image for alpine:latest
==> default: -- Image: redis:alpine
==> default: stdin: is not a tty
==> default: alpine: Pulling from library/redis
==> default: 3690ec4760f9: Already exists
==> default: 5e231f7bdf9d: Pulling fs layer
==> default: 5a74fb2950f8: Pulling fs layer
. . .
```

# How Layers are Reused

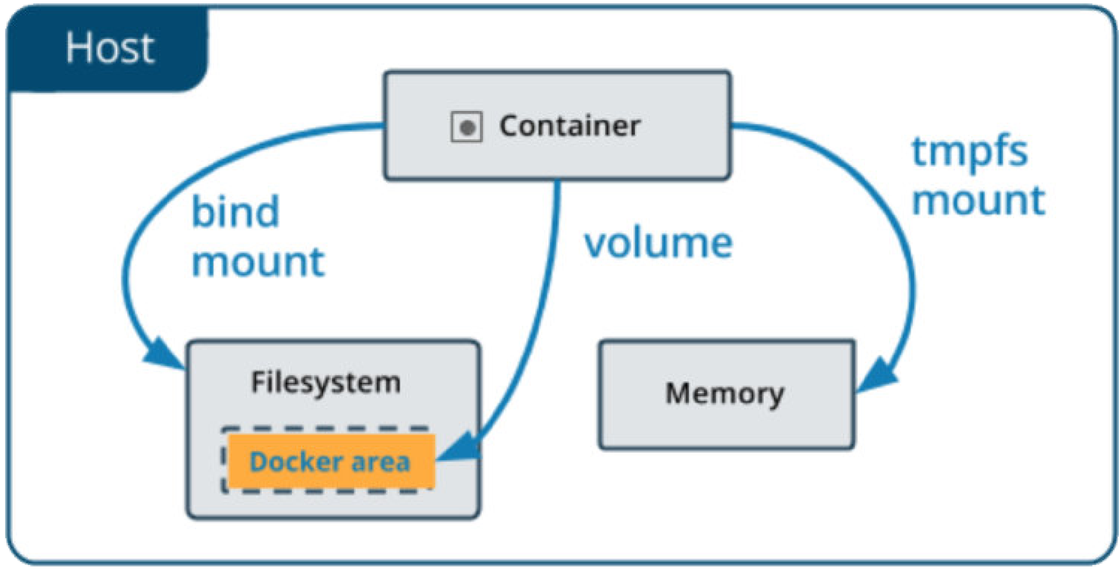
- Note when you provision with Vagrant how Redis:Alpine reuses Alpine layer:

```
==> default: Running provisioner: docker...
    default: Installing Docker onto machine...
==> default: Pulling Docker images...
==> default: -- Image: alpine:latest
==> default: stdin: is not a tty
==> default: latest: Pulling from library/alpine
==> default: 3690ec4760f9: Pulling fs layer
==> default: 3690ec4760f9: Verifying Checksum
==> default: 3690ec4760f9: Download complete
==> default: 3690ec4760f9: Pull complete
==> default: Digest: sha256:1354ab23ff5478120c980eca1611a51c9f2b88b61f24283ee8200bf9a54f2e5c
==> default: Status: Downloaded newer image for alpine:latest
==> default: -- Image: redis:alpine
==> default: stdin: is not a tty
==> default: alpine: Pulling from library/redis
==> default: 3690ec4760f9: Already exists
==> default: 5e231f7baf9a: Pulling fs layer
==> default: 5a74fb2950f8: Pulling fs layer
. . .
```

Alpine layer is being reused by Redis:Alpine

# Docker Volumes

- Volumes are easier to back up or migrate than bind mounts.
- You can manage volumes using Docker CLI commands or the Docker API.
- Volumes work on both Linux and Windows containers.
- Volumes can be more safely shared among multiple containers.
- Volume drivers let you store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.
- New volumes can have their content pre-populated by a container

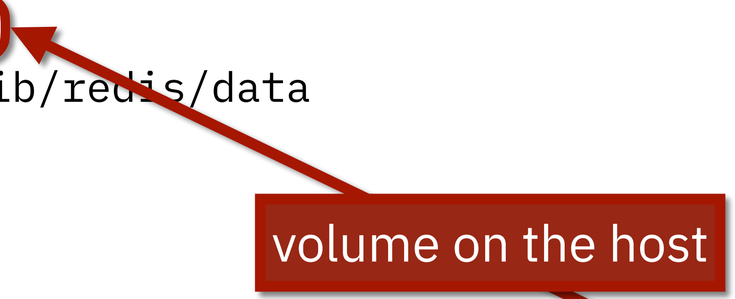


# Bind Mount Volumes for Redis

```
##############################################################################
# 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
```

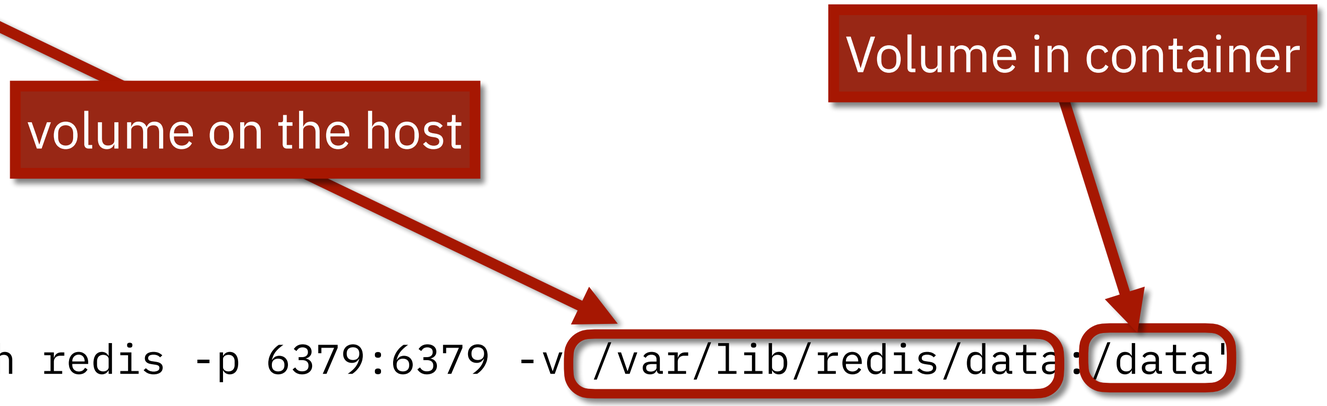
# Bind Mount Volumes for Redis

```
##############################################################################
# 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
```



# Bind Mount 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
```



# Using a Docker Volume for Redis

- We simple give docker a name for the volume like: `redis_data`
- Docker manages this volume for us

```
# 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 redis_data:/data"
end
```

# Using a Docker Volume for Redis

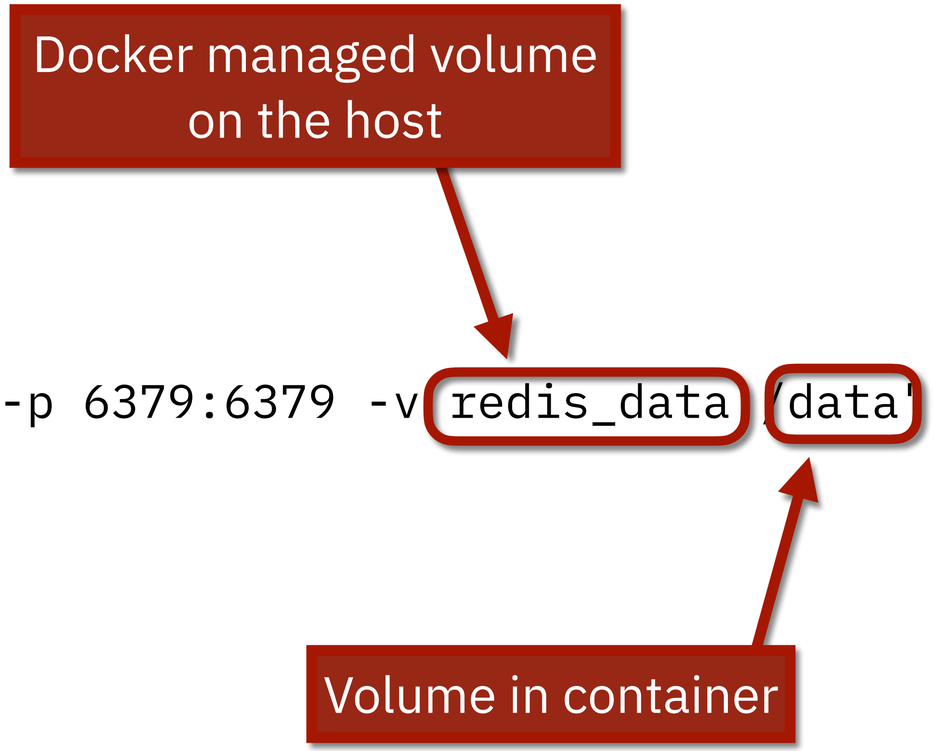
- We simple give docker a name for the volume like: `redis_data`
- Docker manages this volume for us


```
# 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 redis_data /data"
end
```

# Using a Docker Volume for Redis

- We simple give docker a name for the volume like: `redis_data`
- Docker manages this volume for us

```
# 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 redis_data:/data"
end
```



# Docker volume ls

- We can use `docker volume ls` to list the volumes:

```
$ docker volume ls
DRIVER              VOLUME NAME
local               redis_volume
```

An example with named and un-named volumes

```
$ docker volume ls
DRIVER              VOLUME NAME
local               4e469efb9599682f89ff417174256b752afd870c54546021a36d2955d379c4f5
local               87df74bab30c1a4e9f78d4e4a7771b69a2cc5b347d55184d3791204997eaa109
local               368efa07e9f64be73ace7d75c788a363433f3f2007b3de8b74972632d8fcab29
local               cbd7d35d1695b927b59613ee7c9f2cb9b65ed6a9aecbf7745aa233552a9e24de
local               mysql-data
local               postgres-data
```

# Use Minimal Images Whenever Possible

- Minimal images are more secure because they have a smaller attack surface
  - Don't load anything into an image that you don't need
- Minimal images load faster
  - Alpine is only 4.4MB compared to Ubuntu 16.04 at 117MB!
  - PostgreSQL Alpine is 71MB vs 228MB

Docker Images

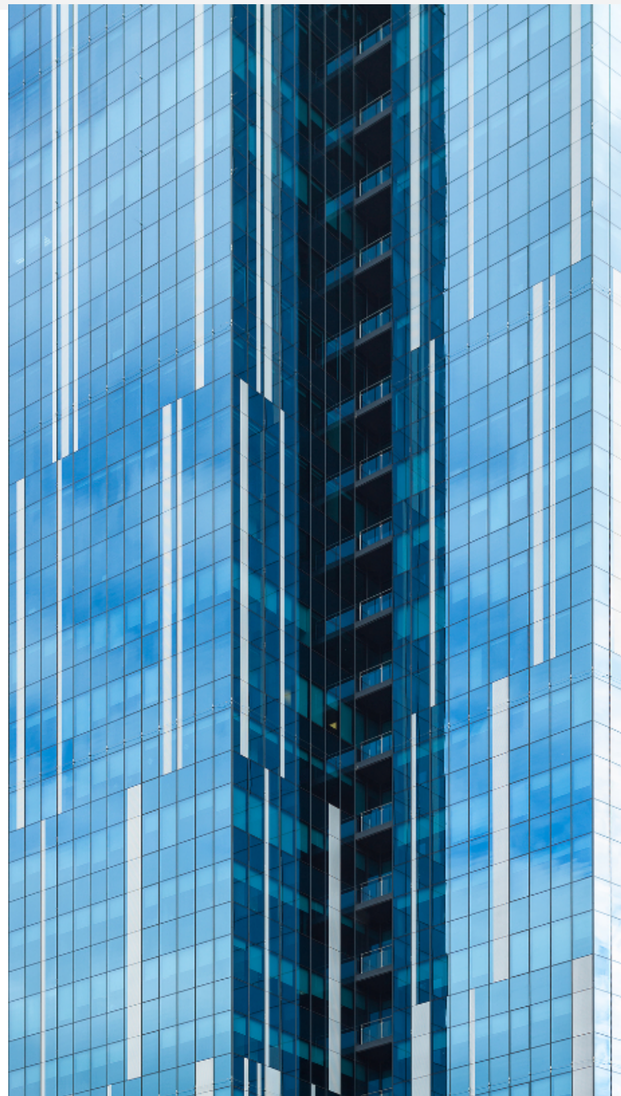
| REPOSITORY | TAG | SIZE |
|---|---|---|
| centos | latest | 202MB |
| debian | stretch-slim | 55.3MB |
| debian | latest | 101MB |
| ubuntu | 16.04 | 117MB |
| ubuntu | latest | 86.7MB |
| python | 2-alpine | 58.3MB |
| python | 2.7-slim | 120MB |
| python | 3.7-alpine | 78.2MB |
| python | 3.7-slim | 143MB |
| bitnami/minideb | latest | 53.7MB |
| alpine | latest | 4.41MB |
| alpine | 3.7 | 4.21MB |
| postgres | alpine | 71.6MB |
| postgres | latest | 228MB |

# Containers should be...

- Stateless
  - All state should be maintained in a Database, Object Store, or Persistent Volume
- Light Weight
  - Only one process per container i.e., Container dies when process dies
- Immutable
  - Do not install an ssh daemon or any other means of entering the container!
- Run from Docker Registry Images or Built from Dockerfiles
  - Treated like code, versioned, and reconstituted when needed... not built by hand!

# What do we mean by Stateless?

- When we say STATE what do we mean?
  - Any pieces of data about the client or transaction
  - Could be the state of a session with the end-user
  - These should be persisted somewhere (database, session cache, etc.) but NOT in memory!



# What Docker Is NOT?

- Docker is NOT a Virtual Machine!
  - Resist the temptation of putting a monolith in a container
  - Resist the urge to run more than one process per container
  - It’s a bad idea to store state in a container (just don’t do it!)

# What Can you Do with Docker?

- You can run **Containers** from the **Images** in the Docker Registry (e.g., Docker Hub)
- You can build Docker **Images** that hold your applications and their dependancies
- You can create Docker **Containers** from those Docker images to run your applications
- You can share those Docker images via **Docker Hub** or your own Docker registry
- You can pull those images from the Docker registry to **deploy** them as Containers on a server running Docker Engine
- You can even deploy those containers in the **Bluemix** Container Cloud!

# Today's Focus

- Install Docker and Cloud Foundry Container Extensions
- Run a Container from an existing Docker Image
- Create a simple Dockefile that spins up a Web Server
- Create a more complex Dockerfile and deploy it

# Hands-On

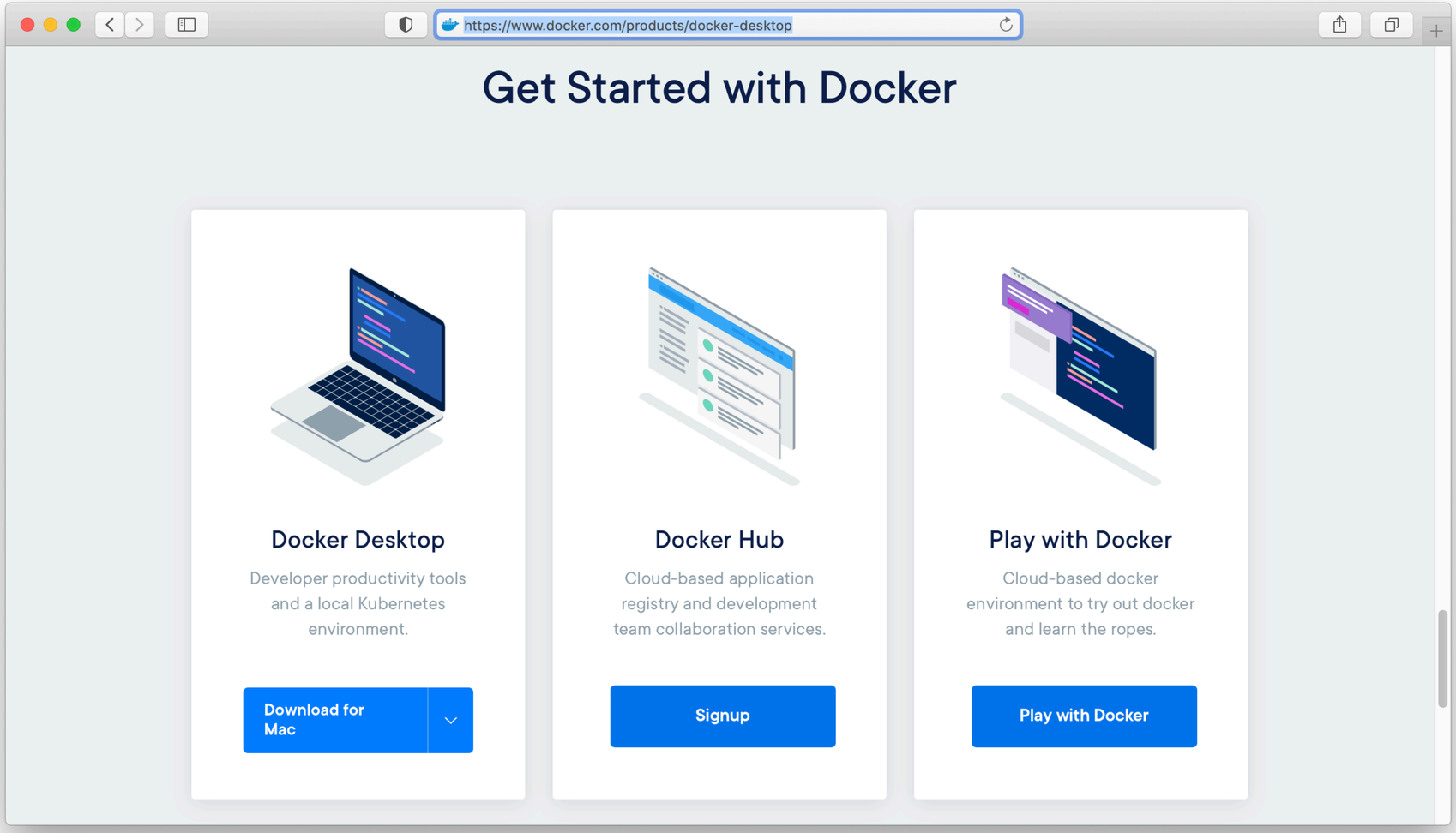
# Some Assembly Required

- Tools you will need to complete this lab:
  - Computer running macOS, Linux, or Windows*
  - Internet Access to download Docker Images
  - Text Editor (e.g., Visual Studio Code)
  - GitHub Account

# Docker Desktop

In this class we are going to use Vagrant and VirtualBox but I want you to be aware that you could also use Docker Desktop

https://www.docker.com/products/docker-desktop



# Create Vagrant / Docker VM

- Using Vagrant is the easiest way to prepare for this lab
- Just issue the following commands:

```
$ git clone https://github.com/nyu-devops/lab-kubernetes.git
$ cd lab-kubernetes
$ vagrant up
$ vagrant ssh
```

# Vagrant Plug-ins


- The `Vagrantfile` can check for required plug-ins and instal, them for you ...but you need to run `vagrant up` again.

```
$ vagrant up
Plugin missing.
Installing the 'vagrant-docker-compose' plugin. This can take a few minutes...
Fetching: vagrant-docker-compose-1.3.0.gem (100%)
Installed the plugin 'vagrant-docker-compose (1.3.0)'!
Dependencies installed, please try the command again.
$
```

# What is in the Vagrantfile?

- Creates an Ubuntu 20.04.3 LTS VM
- Installs a Python 3 Development Environment
- Installs Docker Engine and pulls a Redis image
- Installs Docker Compose
- Installs MicroK8s (single node Kubernetes for developers)
- Installs the IBM Cloud CLI for Kubernetes

# Test the Installation

- Let’s run a Docker “Hello World” container to test our installation:

```
(venv) vagrant@kubernetes:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:c3b4ada4687bbaa170745b3e4dd8ac3f194ca95b2d0518b417fb47e5879d9b5f
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

# Test the Installation

- Let’s run a Docker “Hello World” container to test our installation:

```
(venv) vagrant@kubernetes:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:c3b4ada4687bbaa170745b3e4dd8ac3f194ca95b2d0518b417fb47e5879d9b5f
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

docker pulls hello-world from docker hub

# What Did Docker Run Do?

- It checked the local Docker Image cache to see if `hello-world` was there
- If it was not, it downloaded the `hello-world` image from `hub.docker.com`
- Once downloaded it ran a `hello-world` container from the image

# Docker Images

- We can run `docker images` to see what Images we have locally

```
(venv) vagrant@kubernetes:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
python              3.7-slim            dbb175e1c337        4 days ago          174MB
redis               alpine              6f63d037b592        4 weeks ago         29.3MB
alpine              latest              965ea09ff2eb        4 weeks ago         5.55MB
hello-world         latest              fce289e99eb9        10 months ago       1.84kB
```

# Docker Images

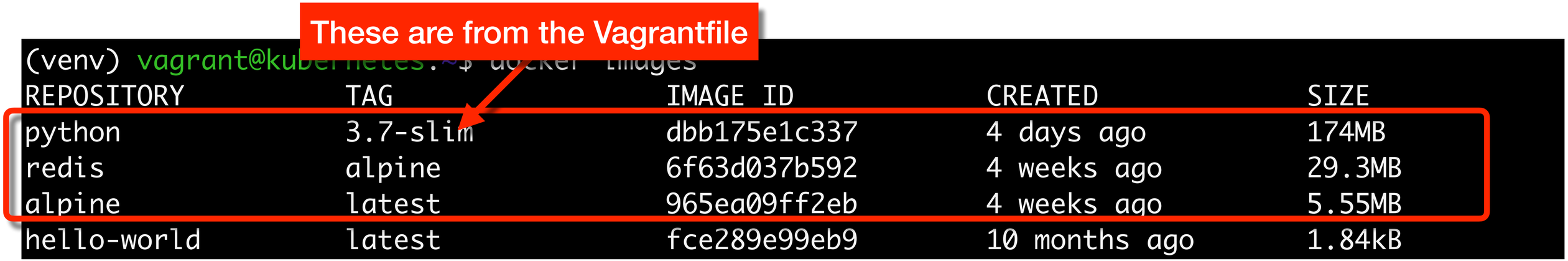
- We can run `docker images` to see what Images we have locally

```
(venv) vagrant@kubernetes:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
python              3.7-slim            dbb175e1c337        4 days ago          174MB
redis               alpine              6f63d037b592        4 weeks ago         29.3MB
alpine              latest              965ea09ff2eb        4 weeks ago         5.55MB
hello-world         latest              fce289e99eb9        10 months ago       1.84kB
```

Here is the hello-world image we just pulled

# Docker Images

- We can run `docker images` to see what Images we have locally



# Working With Docker

- We can run `docker ps` to see what containers we have running

```
$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                    NAMES
ab46390b31b7   redis:alpine   "docker-entrypoint.sh"   26 minutes ago   Up 26 minutes   0.0.0.0:6379->6379/tcp   redis
```

# Working With Docker

- We can run `docker ps` to see what containers we have running

```
$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                    NAMES
ab46390b31b7   redis:alpine   "docker-entrypoint.sh"   26 minutes ago   Up 26 minutes   0.0.0.0:6379->6379/tcp   redis
```

- We can use the `-a` switch to see all containers, even stopped ones

```
$ docker ps -a
CONTAINER ID  IMAGE         COMMAND                  CREATED         STATUS                     PORTS                    NAMES
9a25d7302cf0  hello-world   "/hello"                 17 minutes ago  Exited (0) 17 minutes ago                           silly_boyd
ab46390b31b7  redis:alpine  "docker-entrypoint.sh"   29 minutes ago  Up 29 minutes              0.0.0.0:6379->6379/tcp   redis
```

# Remove Exited Containers

- We can remove a container that has exited with the command:

  `docker rm <container id> | <name>`

```
$ docker ps -a
CONTAINER ID  IMAGE         COMMAND                  CREATED         STATUS                     PORTS                    NAMES
9a25d7302cf0  hello-world   "/hello"                 17 minutes ago  Exited (0) 17 minutes ago                           silly_boyd
ab46390b31b7  redis:alpine  "docker-entrypoint.sh"   29 minutes ago  Up 29 minutes              0.0.0.0:6379->6379/tcp   redis

$ docker rm silly_boyd
silly_boyd

$ docker ps -a
CONTAINER ID  IMAGE         COMMAND                  CREATED         STATUS          PORTS                    NAMES
ab46390b31b7  redis:alpine  "docker-entrypoint.sh"   29 minutes ago  Up 29 minutes   0.0.0.0:6379->6379/tcp   redis
```

# Remove Exited Containers

- We can remove a container that has exited with the command:
  `docker rm <container id> | <name>`

```
$ docker ps -a
CONTAINER ID  IMAGE          COMMAND                  CREATED         STATUS                     PORTS                    NAMES
9a25d7302cf0  hello-world    "/hello"                 17 minutes ago  Exited (0) 17 minutes ago                           silly_boyd
ab46390b31b7  redis:alpine   "docker-entrypoint.sh"   29 minutes ago  Up 29 minutes              0.0.0.0:6379->6379/tcp   redis

$ docker rm silly_boyd
silly_boyd

$ docker ps -a
CONTAINER ID  IMAGE          COMMAND                  CREATED         STATUS          PORTS                    NAMES
ab46390b31b7  redis:alpine   "docker-entrypoint.sh"   29 minutes ago  Up 29 minutes   0.0.0.0:6379->6379/tcp   redis
```

Remove the container named “silly_boyd”

# Remove Unwanted Images

- We can remove an image that we no longer with the:
  `docker rmi <image id> | <repository:tag>`

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
redis               alpine              906fffc0e2f4        35 hours ago        20.38 MB
alpine              latest              baa5d63471ea        39 hours ago        4.803 MB
hello-world         latest              c54a2cc56cbb        3 months ago        1.848 kB
$ docker rmi c54a2cc56cbb
Untagged: hello-world:latest
Untagged: hello-world@sha256:0256e8a36e2070f7bf2d0b0763dbabdd67798512411de4cdcf9431a1feb60fd9
Deleted: sha256:c54a2cc56cbb2f04003c1cd4507e118af7c0d340fe7e2720f70976c4b75237dc
Deleted: sha256:a02596fdd012f22b03af6ad7d11fa590c57507558357b079c3e8cebceb4262d7

$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
redis               alpine              906fffc0e2f4        35 hours ago        20.38 MB
alpine              latest              baa5d63471ea        39 hours ago        4.803 MB
```

# Remove Unwanted Images

- We can remove an image that we no longer with the:
  `docker rmi <image id> | <repository:tag>`

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
redis               alpine              906fffc0e2f4        35 hours ago        20.38 MB
alpine              latest              baa5d63471ea        39 hours ago        4.803 MB
hello-world         latest              c54a2cc56cbb        3 months ago        1.848 kB
$ docker rmi c54a2cc56cbb
Untagged: hello-world:latest
Untagged: hello-world@sha256:0256e8a36e2070f7bf2d0b0763dbabdd67798512411de4cdcf9431a1feb60fd9
Deleted: sha256:c54a2cc56cbb2f04003c1cd4507e118af7c0d340fe7e2720f70976c4b75237dc
Deleted: sha256:a02596fdd012f22b03af6ad7d11fa590c57507558357b079c3e8cebceb4262d7

$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
redis               alpine              906fffc0e2f4        35 hours ago        20.38 MB
alpine              latest              baa5d63471ea        39 hours ago        4.803 MB
```

Remove the image with id "c54a2cc56cbb"

# Creating Containers from Images

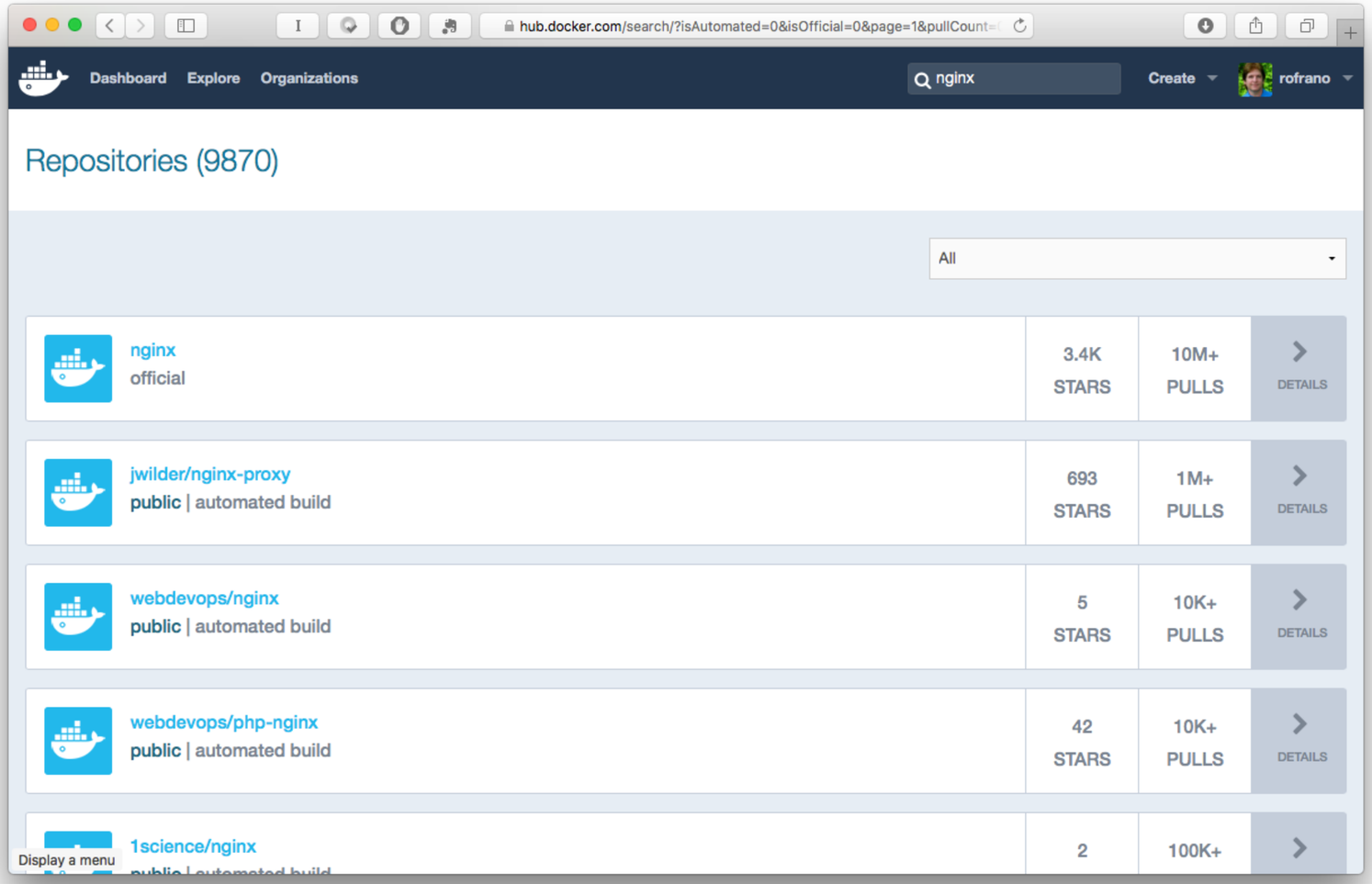
# Lets Use an Existing Docker Image

- Docker Images can be used as-is as a quick way of providing middleware
- Let’s run NGINX without installing anything!

# Where To Get Images?

https://hub.docker.com/_/nginx/

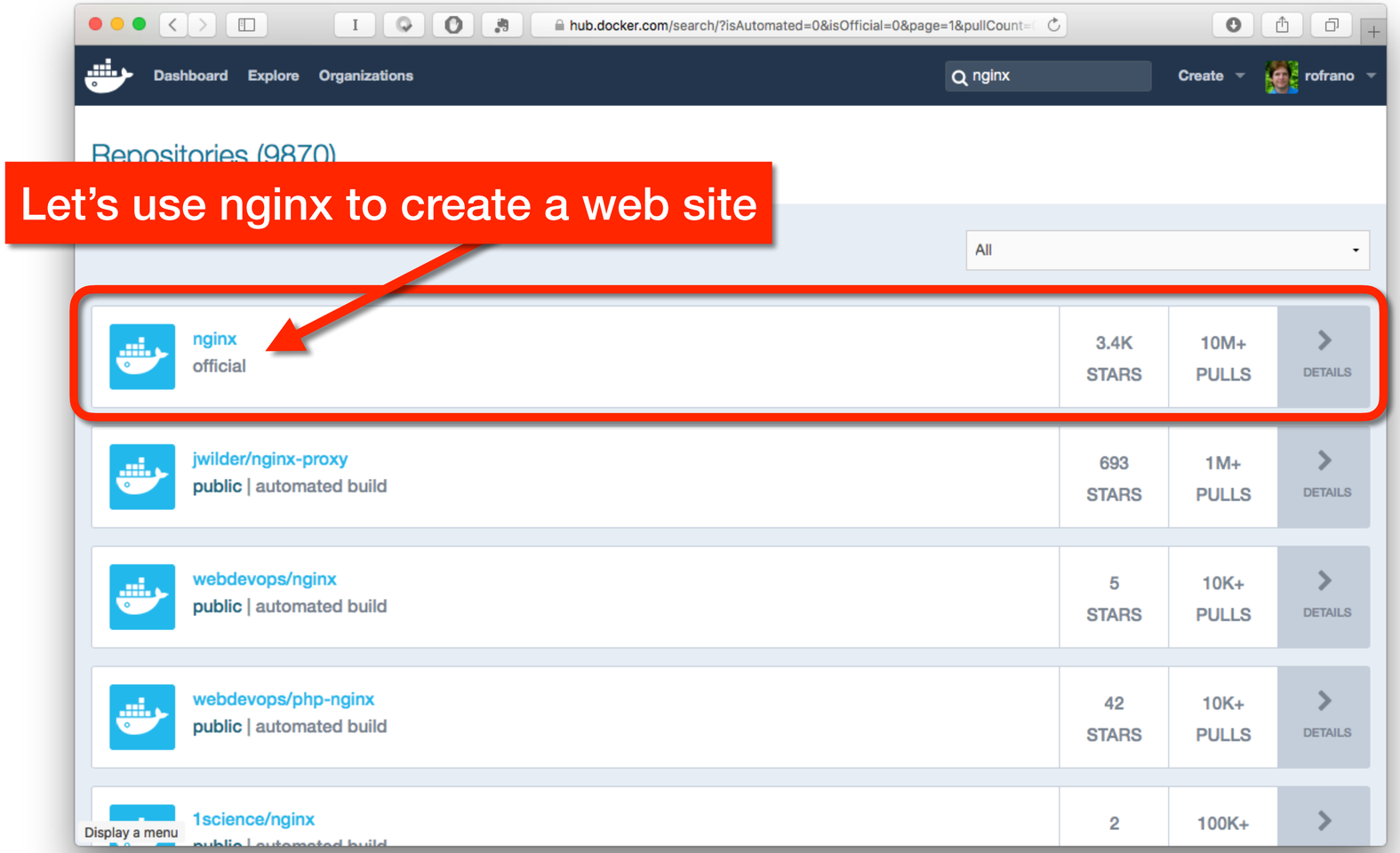
- **Docker Hub**
- Contains 1000's of Docker images with almost every software you can imagine
- Usually "official" images come from the creator of the software
- Most have documentation on how to best use them



# Where To Get Images?

https://hub.docker.com/_/nginx/

- **Docker Hub**
- 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



# Most Images Have Documentation

NGINX

## How to use this image

### hosting some simple static content

```
$ docker run --name some-nginx -v /some/content:/usr/share/nginx/html:ro -d nginx
```

Alternatively, a simple `Dockerfile` can be used to generate a new image that includes the necessary content (which is a much cleaner solution than the bind mount above):

```
FROM nginx
COPY static-html-directory /usr/share/nginx/html
```

Place this file in the same directory as your directory of content ("static-html-directory"), run `docker build -t some-content-nginx .`, then start your container:

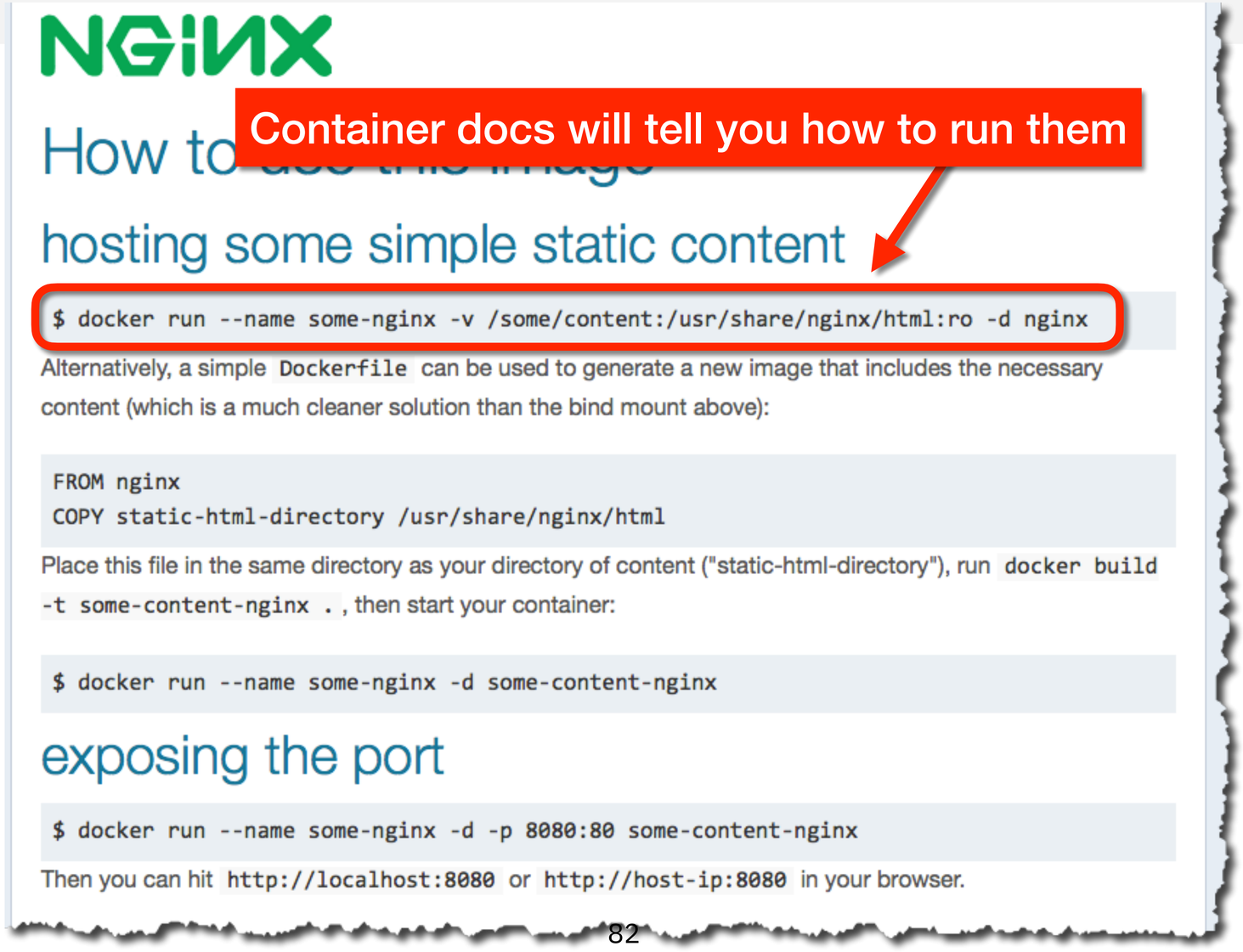
```
$ docker run --name some-nginx -d some-content-nginx
```

### exposing the port

```
$ docker run --name some-nginx -d -p 8080:80 some-content-nginx
```

Then you can hit `http://localhost:8080` or `http://host-ip:8080` in your browser.

# Most Images Have Documentation



# Let’s Run Nginx

- We can run nginx as a web server with the command:

```
docker run -d -p 8080:80 nginx:alpine
```

```
$ docker run -d -p 8080:80 nginx:alpine
8e82325aa105cb0bc4f192649f01c32e988045bc59b5d036d5c90505f2cf662d
```

# Let’s Run Nginx

- We can run nginx as a web server with the command:

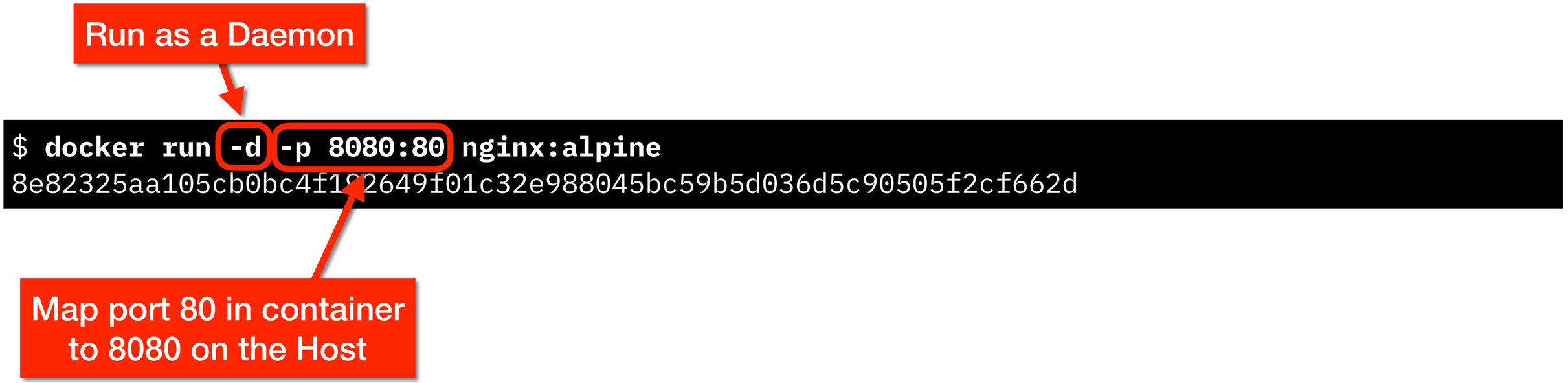
```
docker run -d -p 8080:80 nginx:alpine
```


```
$ docker run -d -p 8080:80 nginx:alpine
8e82325aa105cb0bc4f192649f01c32e988045bc59b5d036d5c90505f2cf662d
```

# Let’s Run Nginx

- We can run nginx as a web server with the command:

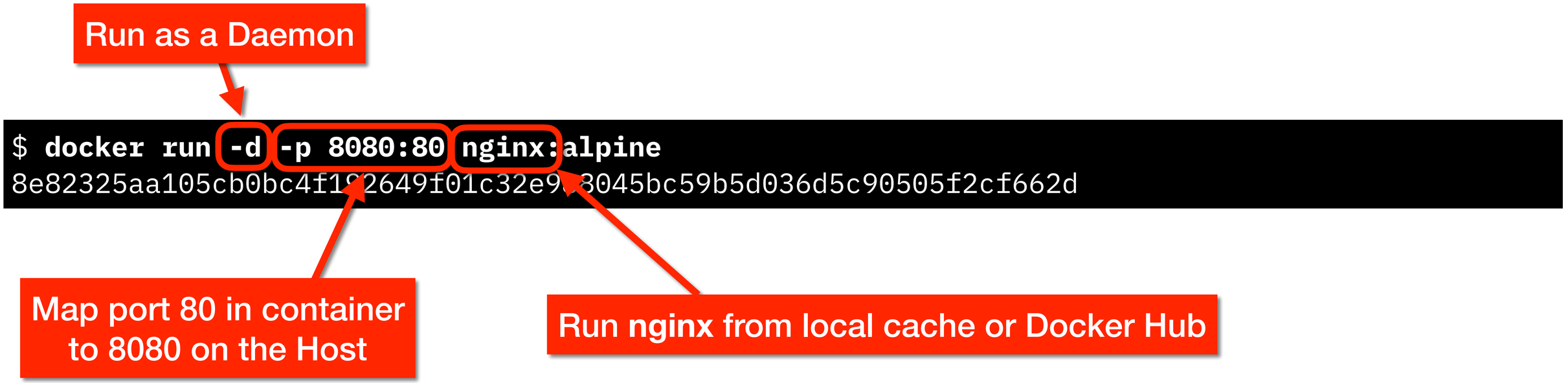
```
docker run -d -p 8080:80 nginx:alpine
```



# Let’s Run Nginx

- We can run nginx as a web server with the command:

```
docker run -d -p 8080:80 nginx:alpine
```

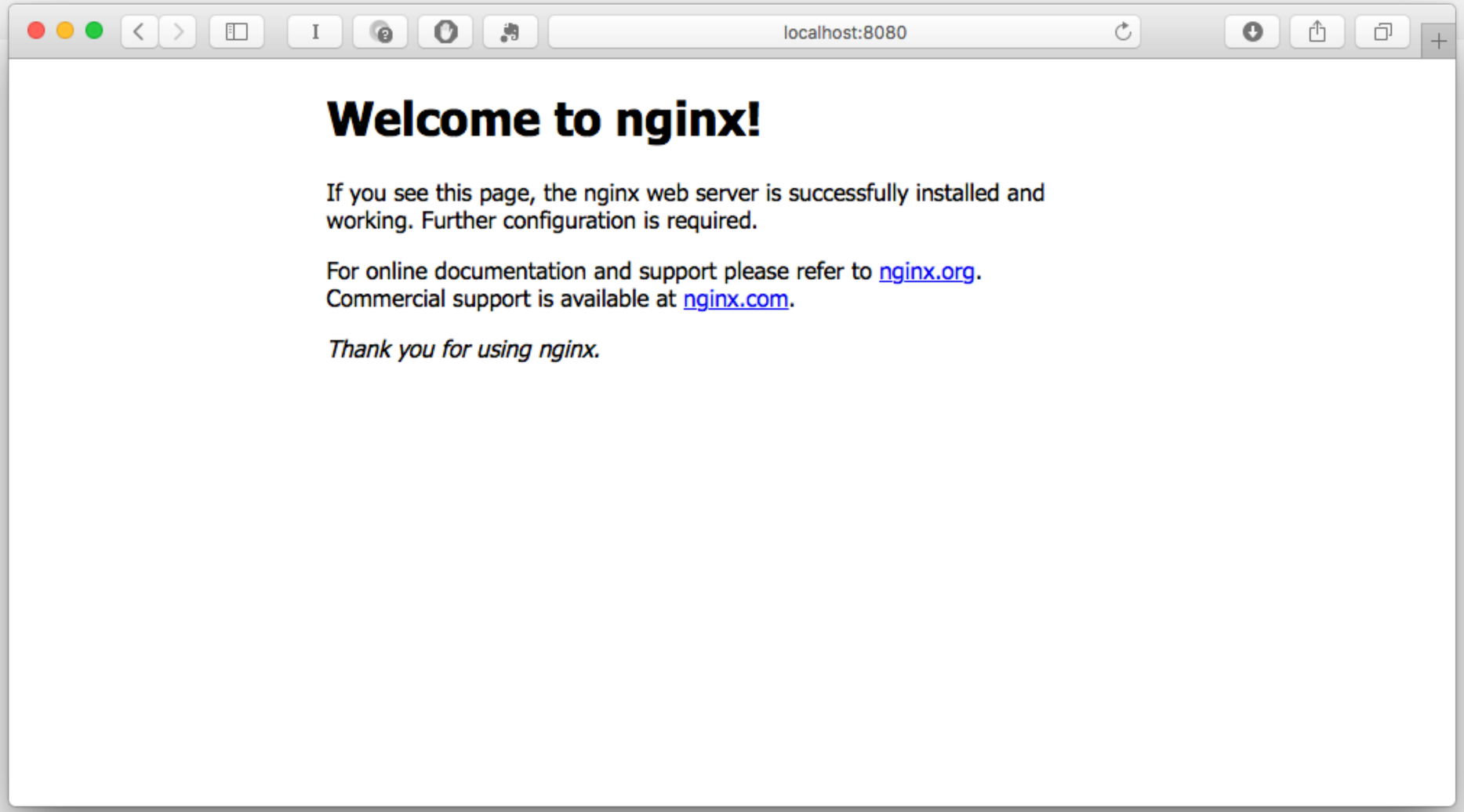


# NGINX will be Pulled

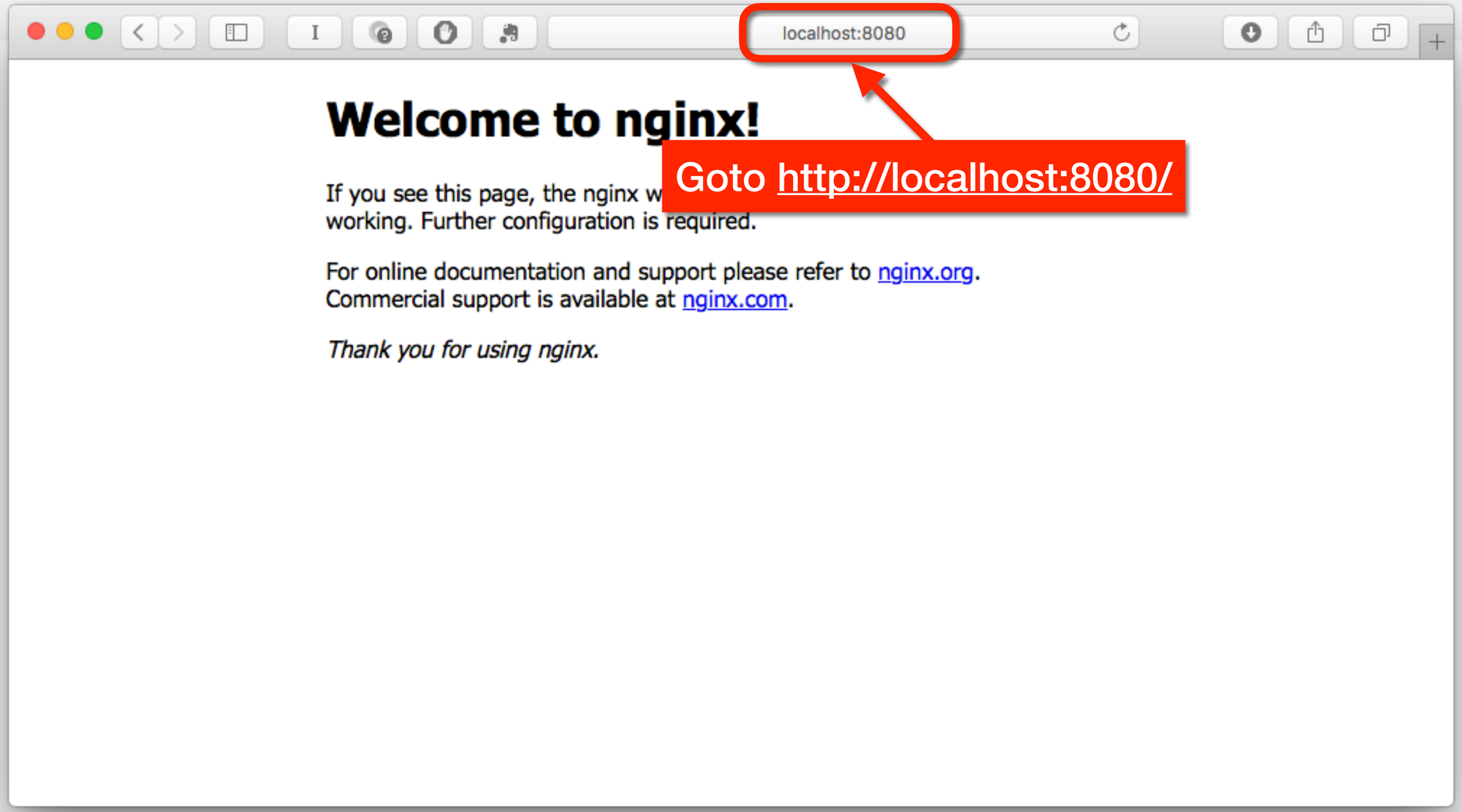
- Since we don’t have a local nginx image it will be pulled from Docker hub the first time

```
$ docker run -d -p 8080:80 nginx:alpine

Unable to find image 'nginx:alpine' locally
alpine: Pulling from library/nginx
89d9c30c1d48: Already exists
110ad692b782: Pull complete
Digest: sha256:559cde0e812dba817c131173217712bf34c2c80a0b6892409cbcf7d760fca8be
Status: Downloaded newer image for nginx:alpine
53cee8d8cf12ca91e7b2fd9173d03772757c36851cf70a855210005e9fa9b26e
```

# In A Web browser



# In A Web browser



# Stop a Container

- We can stop the running container with the command:
  `docker stop <container name> | <container id>`

```
$ docker ps
CONTAINER ID  IMAGE         COMMAND                  CREATED         STATUS         PORTS                              NAMES
0d48962ddc38  nginx:alpine  "nginx -g 'daemon off"   4 minutes ago   Up 4 minutes   443/tcp, 0.0.0.0:8080->80/tcp      stoic_shaw
ab46390b31b7  redis:alpine  "docker-entrypoint.sh"   51 minutes ago  Up 51 minutes  0.0.0.0:6379->6379/tcp             redis

$ docker stop stoic_shaw
stoic_shaw

$ docker rm stoic_shaw
stoic_shaw

$ docker ps -a
CONTAINER ID  IMAGE         COMMAND                  CREATED         STATUS         PORTS                      NAMES
ab46390b31b7  redis:alpine  "docker-entrypoint.sh"   52 minutes ago  Up 52 minutes  0.0.0.0:6379->6379/tcp     redis
```

# Stop a Container

- We can stop the running container with the command:
  `docker stop <container name> | <container id>`

```
$ docker ps
CONTAINER ID  IMAGE         COMMAND                  CREATED         STATUS         PORTS                              NAMES
0d48962ddc38  nginx:alpine  "nginx -g 'daemon off"   4 minutes ago   Up 4 minutes   443/tcp, 0.0.0.0:8080->80/tcp      stoic_shaw
ab46390b31b7  redis:alpine  "docker-entrypoint.sh"   51 minutes ago  Up 51 minutes  0.0.0.0:6379->6379/tcp             redis

$ docker stop stoic_shaw
stoic_shaw

$ docker rm stoic_shaw
stoic_shaw

$ docker ps -a
CONTAINER ID  IMAGE         COMMAND                  CREATED         STATUS         PORTS                      NAMES
ab46390b31b7  redis:alpine  "docker-entrypoint.sh"   52 minutes ago  Up 52 minutes  0.0.0.0:6379->6379/tcp     redis
```

# Stop a Container

- We can stop the running container with the command:
  `docker stop <container name> | <container id>`

```
$ docker ps
CONTAINER ID  IMAGE          COMMAND                  CREATED         STATUS         PORTS                              NAMES
0d48962ddc38  nginx:alpine   "nginx -g 'daemon off"   4 minutes ago   Up 4 minutes   443/tcp, 0.0.0.0:8080->80/tcp      stoic_shaw
ab46390b31b7  redis:alpine   "docker-entrypoint.sh"   51 minutes ago  Up 51 minutes  0.0.0.0:6379->6379/tcp             redis

$ docker stop stoic_shaw
stoic_shaw

$ docker rm stoic_shaw
stoic_shaw

$ docker ps -a
CONTAINER ID  IMAGE          COMMAND                  CREATED         STATUS         PORTS                      NAMES
ab46390b31b7  redis:alpine   "docker-entrypoint.sh"   52 minutes ago  Up 52 minutes  0.0.0.0:6379->6379/tcp     redis
```

Stop container “stoic_shaw”

Remove container “stoic_shaw”

# Add Your Content To Nginx

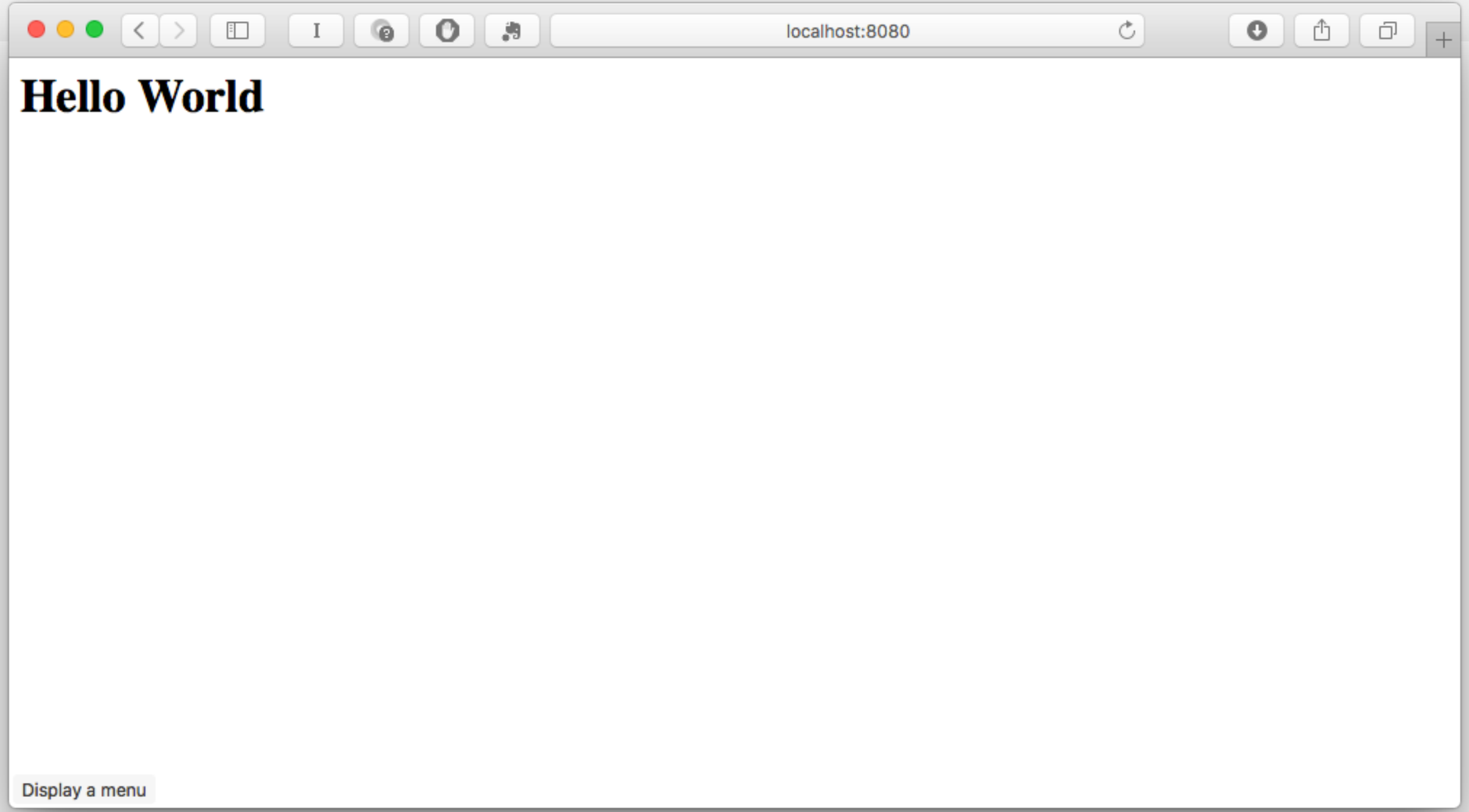
- Add some content to `/home/vagrant`

```
cd /home/vagrant
echo '<html><body><h1>Hello World</h1></body></html>' > index.html
```

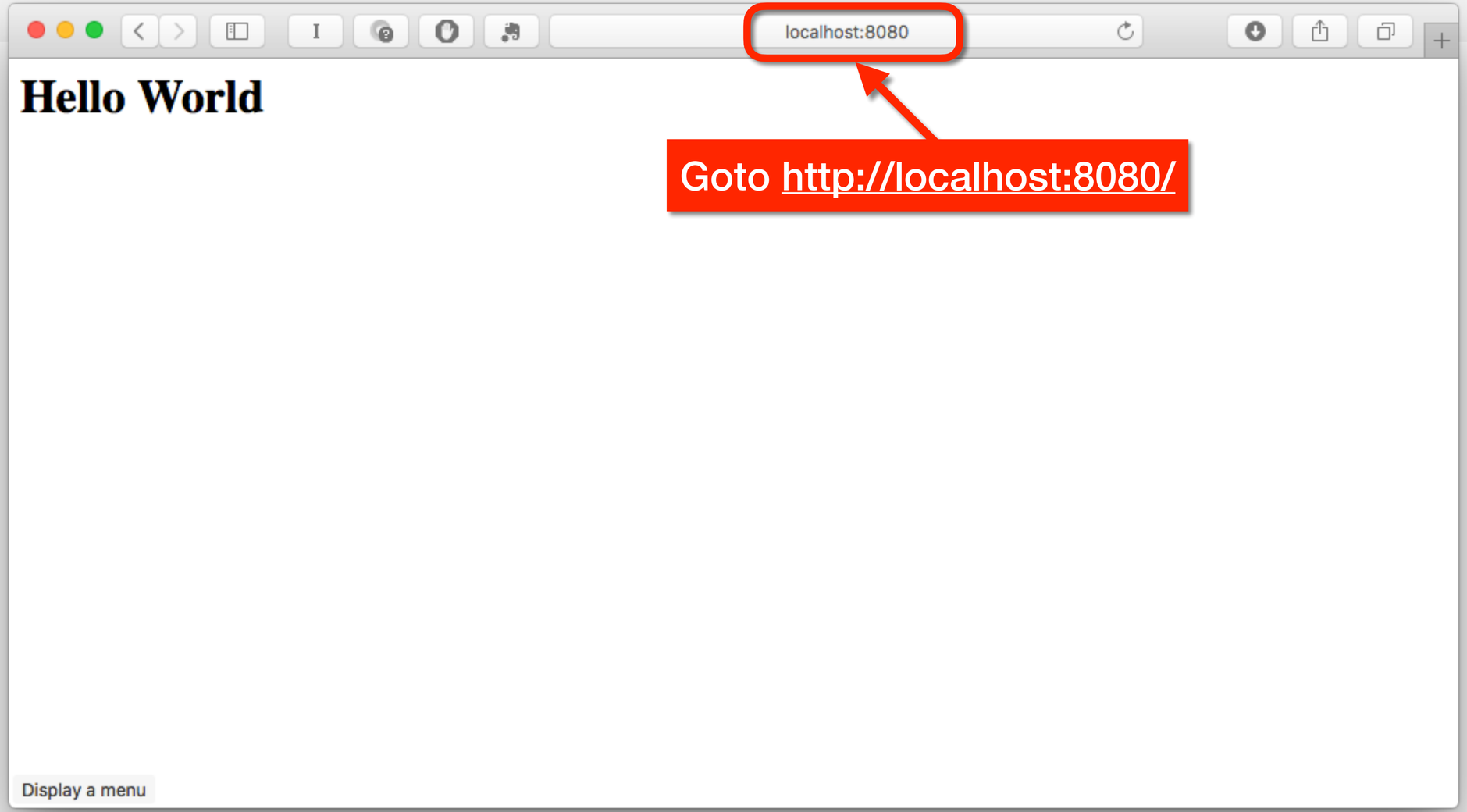
- We can map a local filesystem with the `-v` (volume) option:

```
docker run -d --name Nginx -p 8080:80 -v /home/vagrant:/usr/share/nginx/html:ro
nginx:alpine
```

# Check In A Web browser



# Check In A Web browser



# Dynamic Development

- You can now change the files in that folder and the changes will be reflected in the web browser
- Try it and see...
- Then stop the container with:

```
docker stop nginx
docker rm nginx
```

# Creating Containers from Dockerfiles

# Using A Dockerfile

- We can create a new Image from a `Dockerfile`
- The Dockerfile is just a text file that contains the commands to build the image
- This allows you to treat Images like code

# Create nginx from Dockerfile

- We can create a Dockefile to add our own content to the image
- Create a file called `Dockerfile` and add the following two lines:

```
FROM nginx:alpine
COPY content /usr/share/nginx/html
```

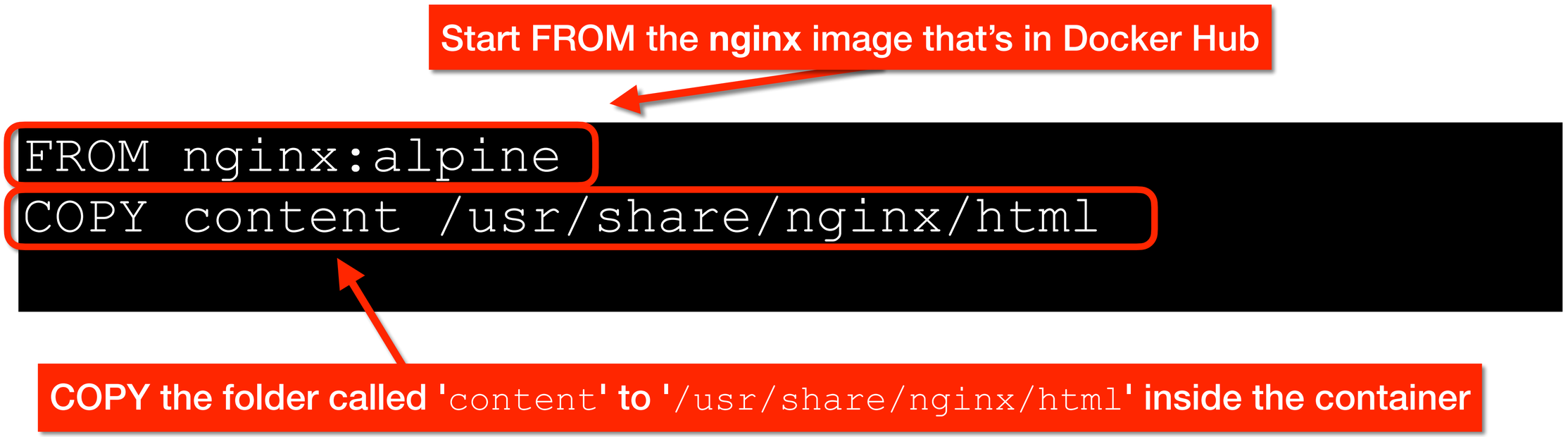
# Create nginx from Dockerfile

- We can create a Dockefile to add our own content to the image
- Create a file called `Dockerfile` and add the following two lines:


```
FROM nginx:alpine
COPY content /usr/share/nginx/html
```

# Create nginx from Dockerfile

- We can create a Dockefile to add our own content to the image
- Create a file called `Dockerfile` and add the following two lines:



# What is `/usr/share/nginx/html`?

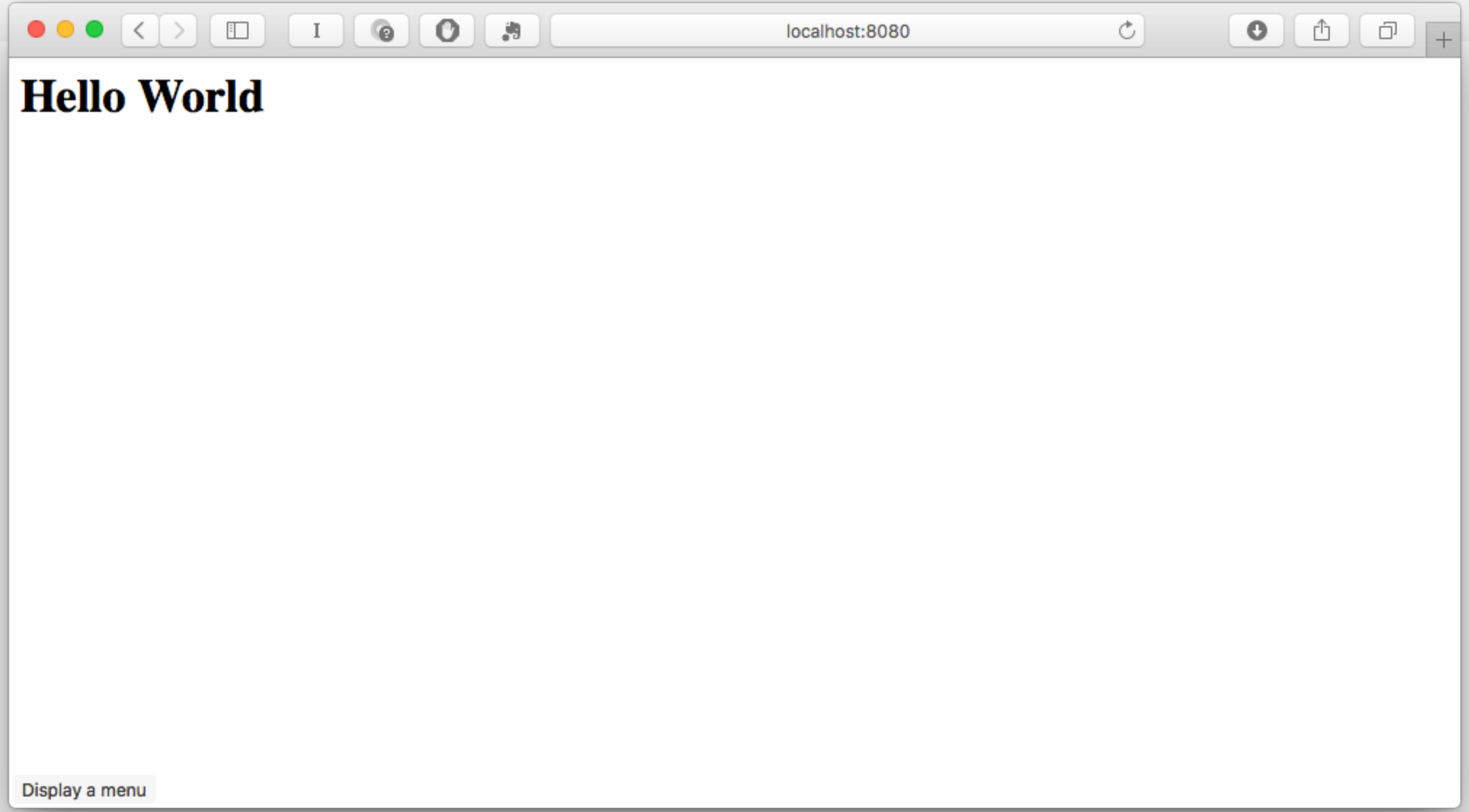
- `/usr/share/nginx/html` is where nginx is looking for a web site
- We want to copy the contents of our local folder into this
- It will create a container that includes the web site

# Create the Content

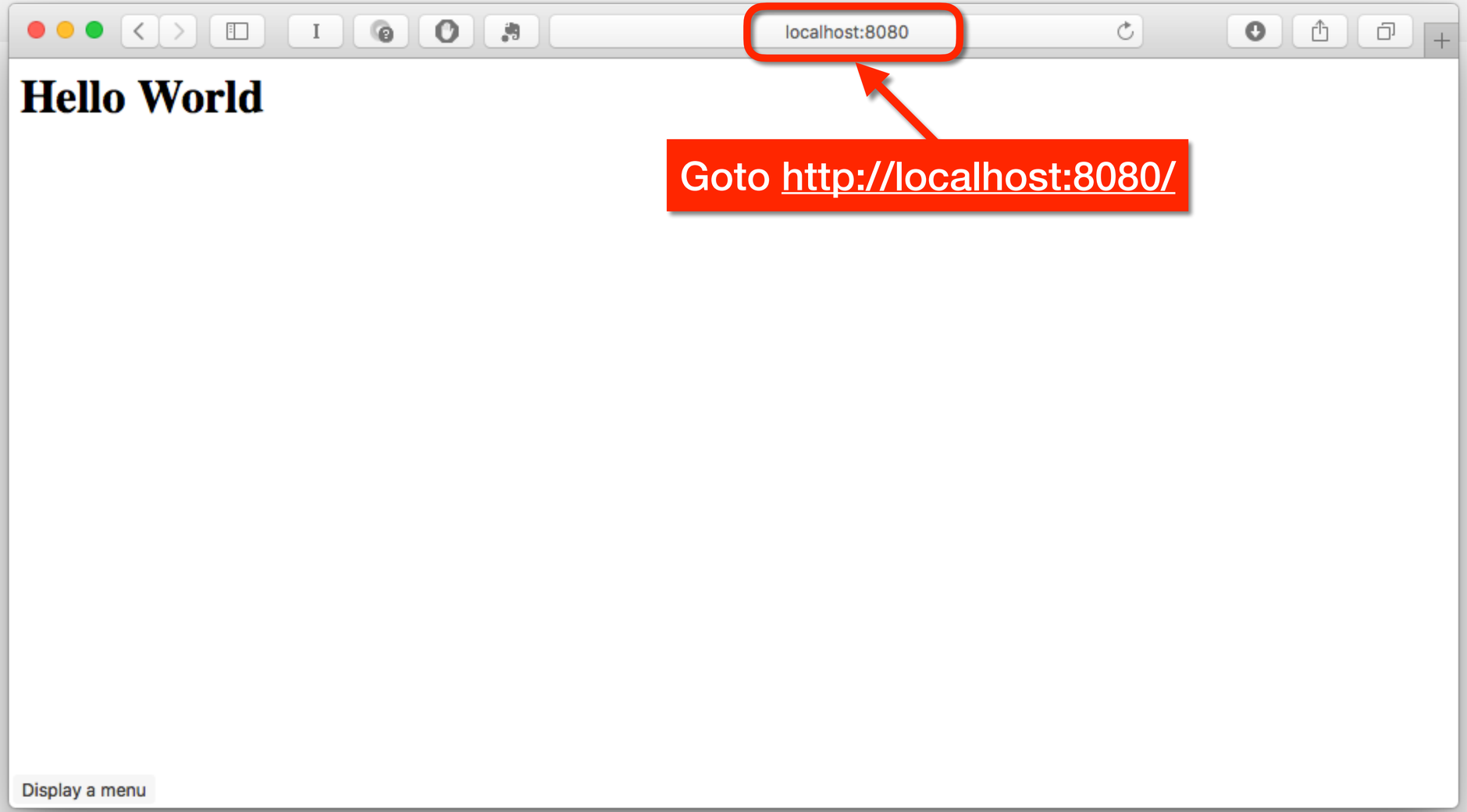
- Next create a `content` folder and move the `index.html` file into it
- Finally build the container

```
$ mkdir content
$ mv index.html content/
$ vi Dockerfile
$ docker build -t my-nginx .
Sending build context to Docker daemon 22.02 kB
Step 1 : FROM nginx
 ---> 0d409d33b27e
Step 2 : COPY content /usr/share/nginx/html
 ---> 16d63ce8beff
Removing intermediate container e286a6fd3eca
Successfully built 16d63ce8beff
$ docker run -d --name nginx -p 8080:80 my-nginx
765743b8ec5671761c3dd479026fd131ac11c9911b89046c87134a1cb2178218
```

# Check In A Web browser

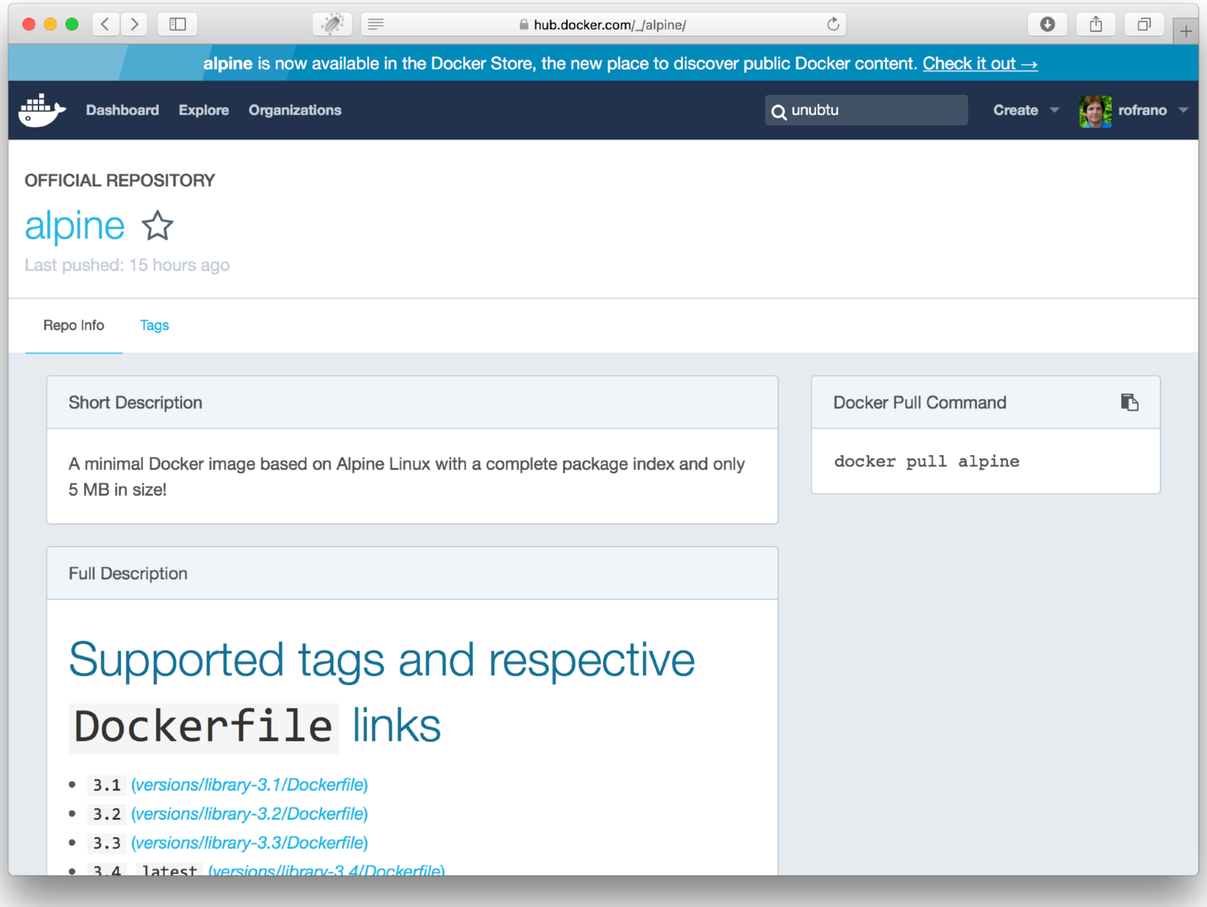


# Check In A Web browser



# Containers should be small

- Its a good practice to use a tiny Linux distribution for building containers
- Docker images should only contain the app and it’s required libraries (not a whole OS!)
- Small distributions have a small attack surface and are more secure
- Alpine is perfect for this base OS



# DevOps Workshop Dockerfile

```
FROM python:3.7-slim

# Create working folder and install dependencies
WORKDIR /app
COPY requirements.txt /app
RUN pip install --no-cache-dir -r requirements.txt

# Copy the application contents
COPY service/ /app/service/

# Expose any ports in the environment
ENV PORT 8080
EXPOSE $PORT

ENV GUNICORN_BIND 0.0.0.0:$PORT
CMD ["gunicorn", "--log-level=info", "service:app"]
```

# Build the Dockerfile

```
$ cd /vagrant/
$ docker build -t hitcounter:1.0 .

Sending build context to Docker daemon  443.4kB
Step 1/9 : FROM python:3.7-slim
 ---> 07ee12a5eb2a
Step 2/9 : WORKDIR /app
 ---> Running in c1e559050e81
Removing intermediate container c1e559050e81
 ---> 3038f6baa90d
Step 3/9 : COPY requirements.txt /app
 ---> f1f0fa408d34
Step 4/9 : RUN pip install --no-cache-dir -r requirements.txt
 ---> Running in 282e12ad02ff
Collecting Flask==1.1.1
  Downloading https://files.pythonhosted.org/packages/9b/
93/628509b8d5dc749656a9641f4caf13540e2cdec85276964ff8f43bbb1d3b/Flask-1.1.1-py2.py3-none-any.whl (94kB)

... lots of python packages get installed here ...

Step 5/9 : COPY service/ /app/service/
 ---> 434eb6e1e911
Step 6/9 : ENV PORT 8080
 ---> Running in ae8c0dbce3b7
Removing intermediate container ae8c0dbce3b7
 ---> 38b81826fcec
Step 7/9 : EXPOSE $PORT
 ---> Running in 9a359968e7b3
Removing intermediate container 9a359968e7b3
 ---> a8e7d7e360f6
Step 8/9 : ENV GUNICORN_BIND 0.0.0.0:$PORT
 ---> Running in f4b72400abec
Removing intermediate container f4b72400abec
 ---> 813303bb1801
Step 9/9 : CMD ["gunicorn", "--log-level=info", "service:app"]
 ---> Running in 3b71a2ccbe1e
Removing intermediate container 3b71a2ccbe1e
 ---> 8ac3c707ab08
Successfully built 8ac3c707ab08
Successfully tagged hitcounter:1.0
```

# Check Your Images

- `docker images`

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hitcounter          1.0                 8ac3c707ab08        2 minutes ago       200MB
python              3.7-slim            07ee12a5eb2a        4 days ago          179MB
nginx               alpine              b6753551581f        4 weeks ago         21.4MB
redis               alpine              6f63d037b592        4 weeks ago         29.3MB
alpine              latest              965ea09ff2eb        4 weeks ago         5.55MB
$
```

# Check Your Images

- `docker images`

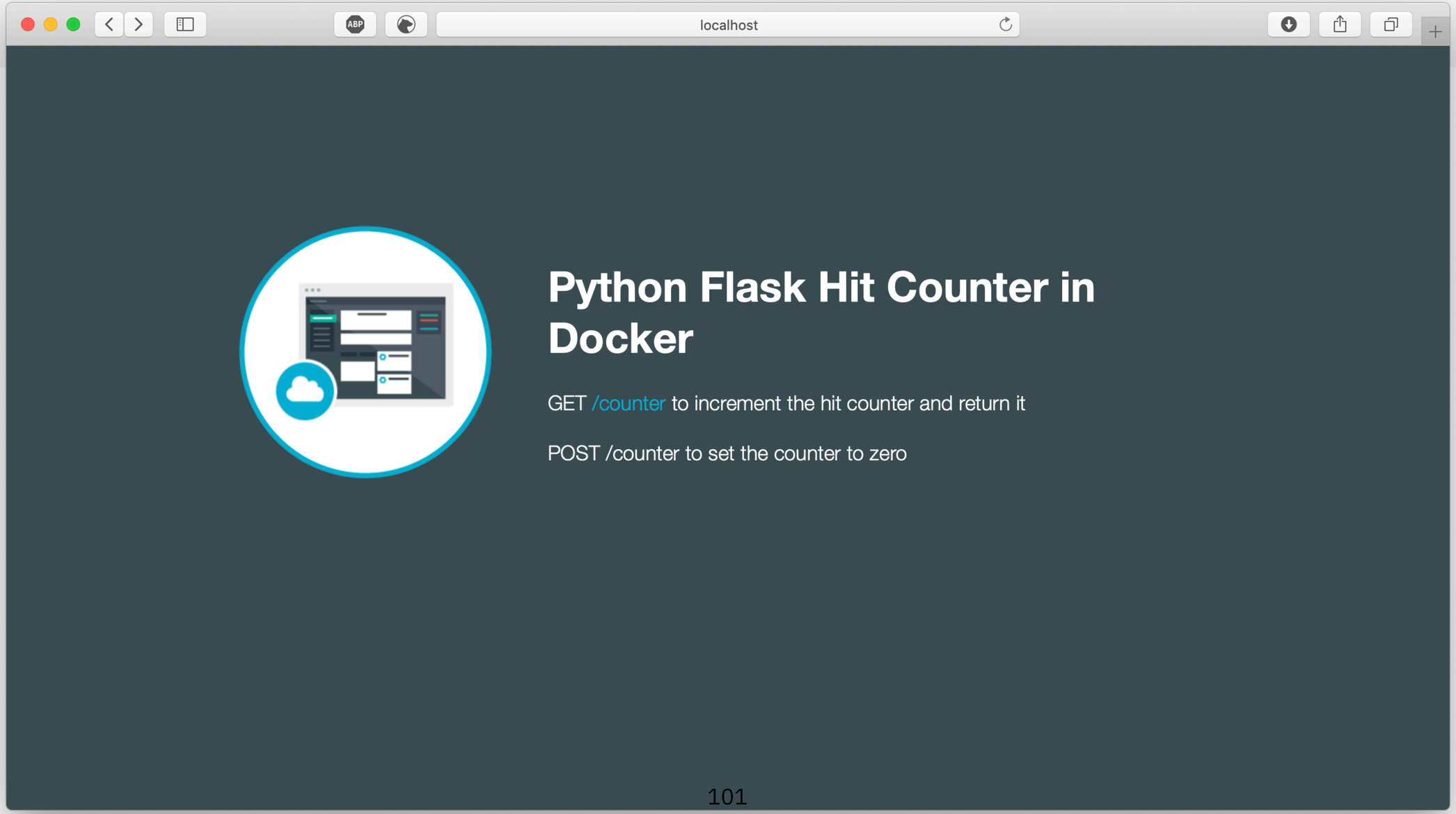
```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hitcounter          1.0                 8ac3c707ab08        2 minutes ago       200MB
python              3.7-slim            07ee12a5eb2a        4 days ago          179MB
nginx               alpine              b6753551581f        4 weeks ago         21.4MB
redis               alpine              6f63d037b592        4 weeks ago         29.3MB
alpine              latest              965ea09ff2eb        4 weeks ago         5.55MB
$
```

# Run the Container

- `docker run --rm -p 8080:8080 hitcounter:1.0`

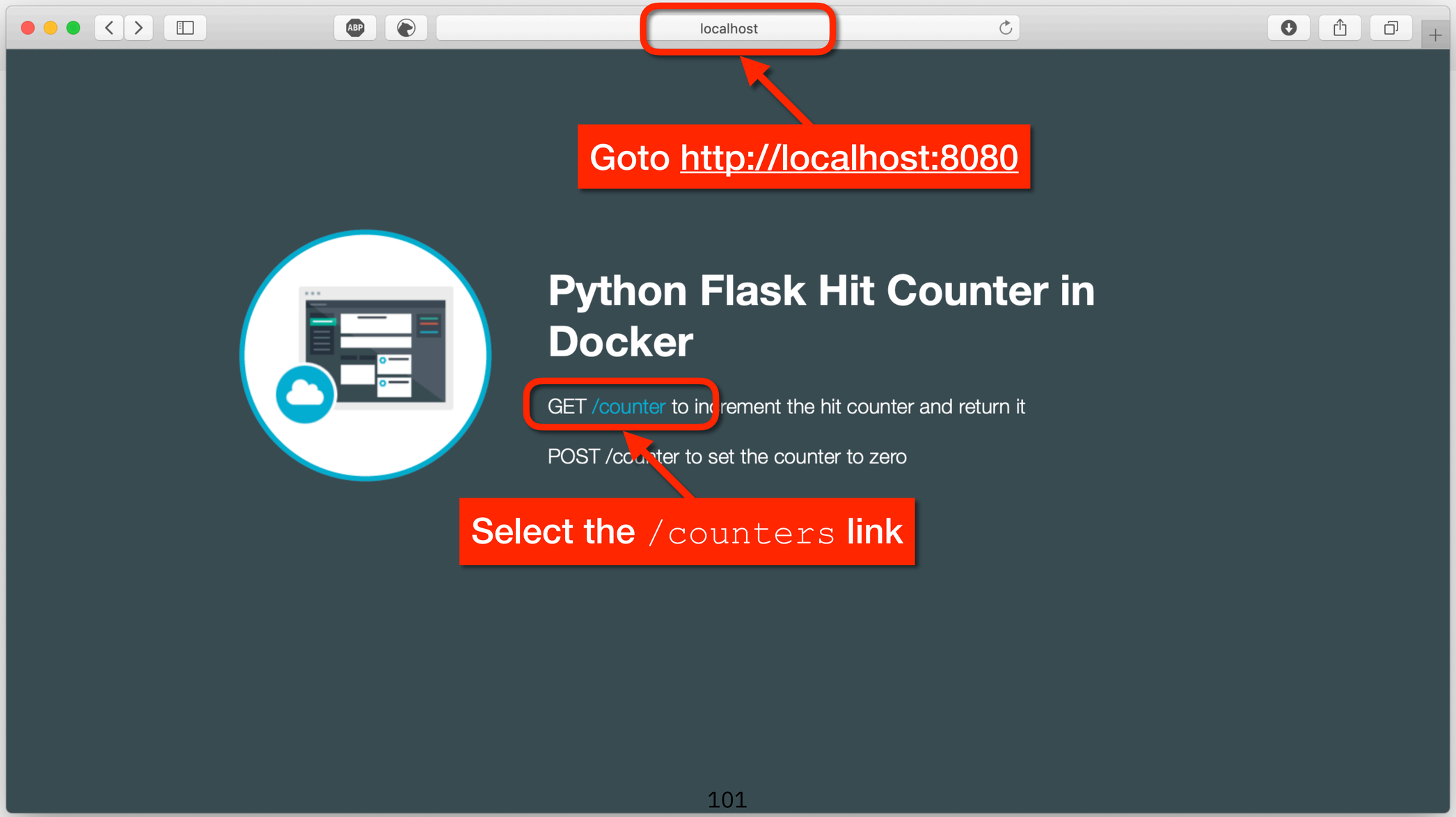
```
$ docker run --rm -p 8080:8080 hitcounter:1.0

[2019-11-20 00:54:33 +0000] [1] [INFO] Starting gunicorn 20.0.0
[2019-11-20 00:54:33 +0000] [1] [INFO] Listening at: http://0.0.0.0:8080 (1)
[2019-11-20 00:54:33 +0000] [1] [INFO] Using worker: sync
[2019-11-20 00:54:33 +0000] [9] [INFO] Booting worker with pid: 9
[2019-11-20 00:54:33 +0000] [9] [INFO] ********************************************************************
[2019-11-20 00:54:33 +0000] [9] [INFO] **************  H I T   C O U N T E R   S E R V I C E  ***************
[2019-11-20 00:54:33 +0000] [9] [INFO] ********************************************************************
[2019-11-20 00:54:33 +0000] [9] [INFO] Service inititalized!
```

# Check the Running Service @ localhost:8080



# Check the Running Service @ localhost:8080

# Check the Running Service @ localhost:8080



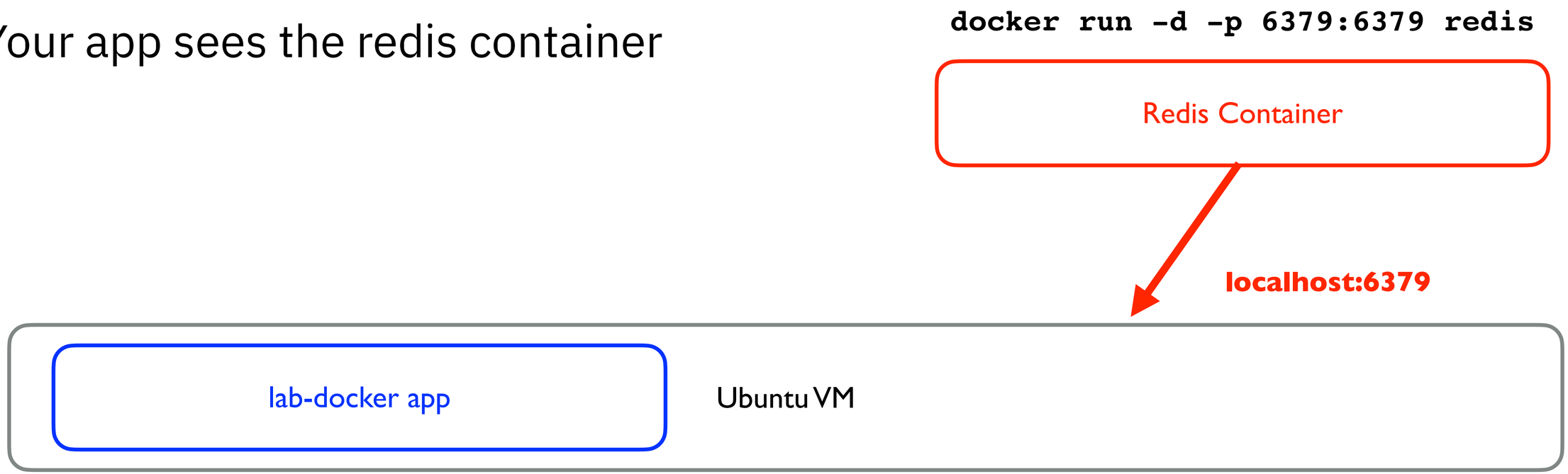
# What Happened?

# Where is Redis?

- Redis is running in a container which responds to 127.0.0.1:6379 because we forwarded the port to the VM
- You can talk to from the VM through it’s forwarded port
- Python running in a Container cannot talk to the VM and so it cannot talk to Redis
- We must link the containers together

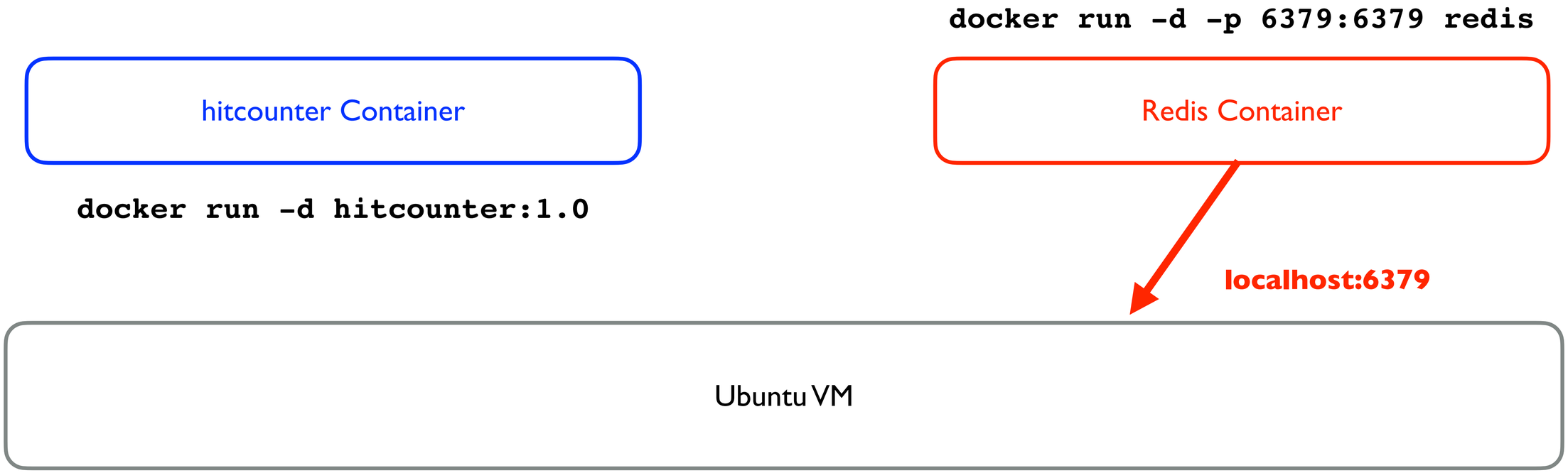
# Running in a VM

- Your code is running on localhost
- The container is exposed on localhost
- Your app sees the redis container



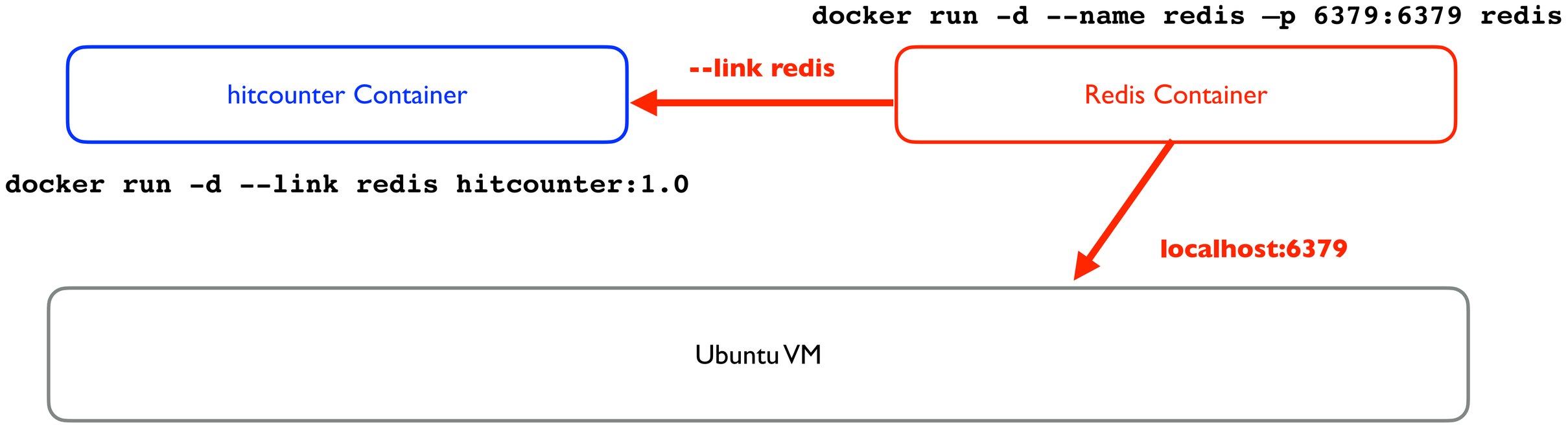
# Running in a Container

- Your code is running in a container not on localhost
- Your app can no longer see redis



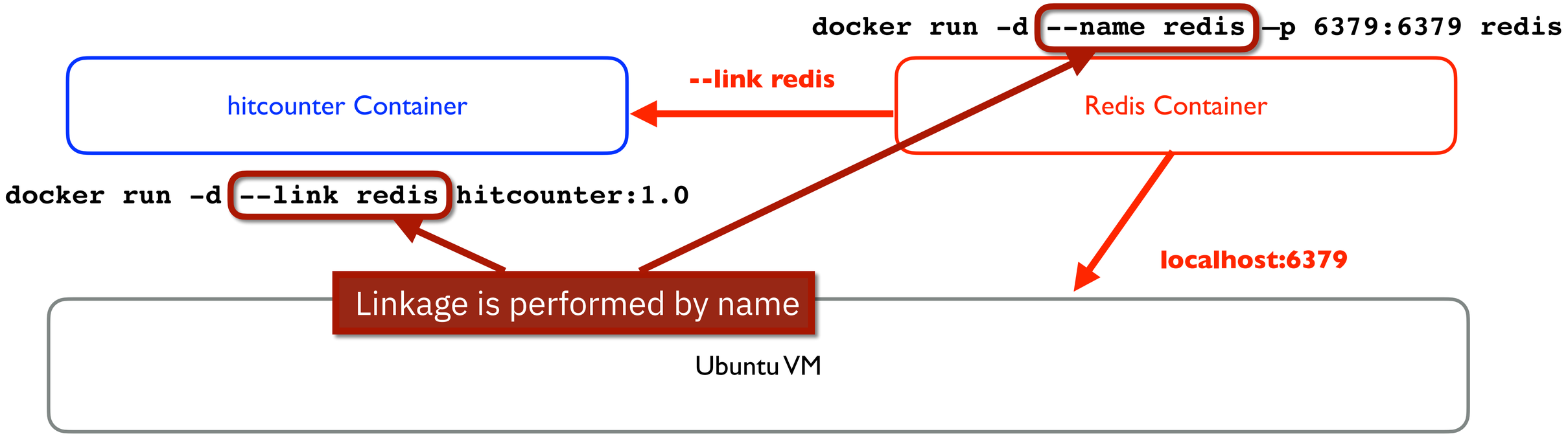
# Solution

- Use the `--link` command to link to redis container
- Your app can no now see redis



# Solution

- Use the `--link` command to link to redis container
- Your app can no now see redis



# Add environment variable to match link

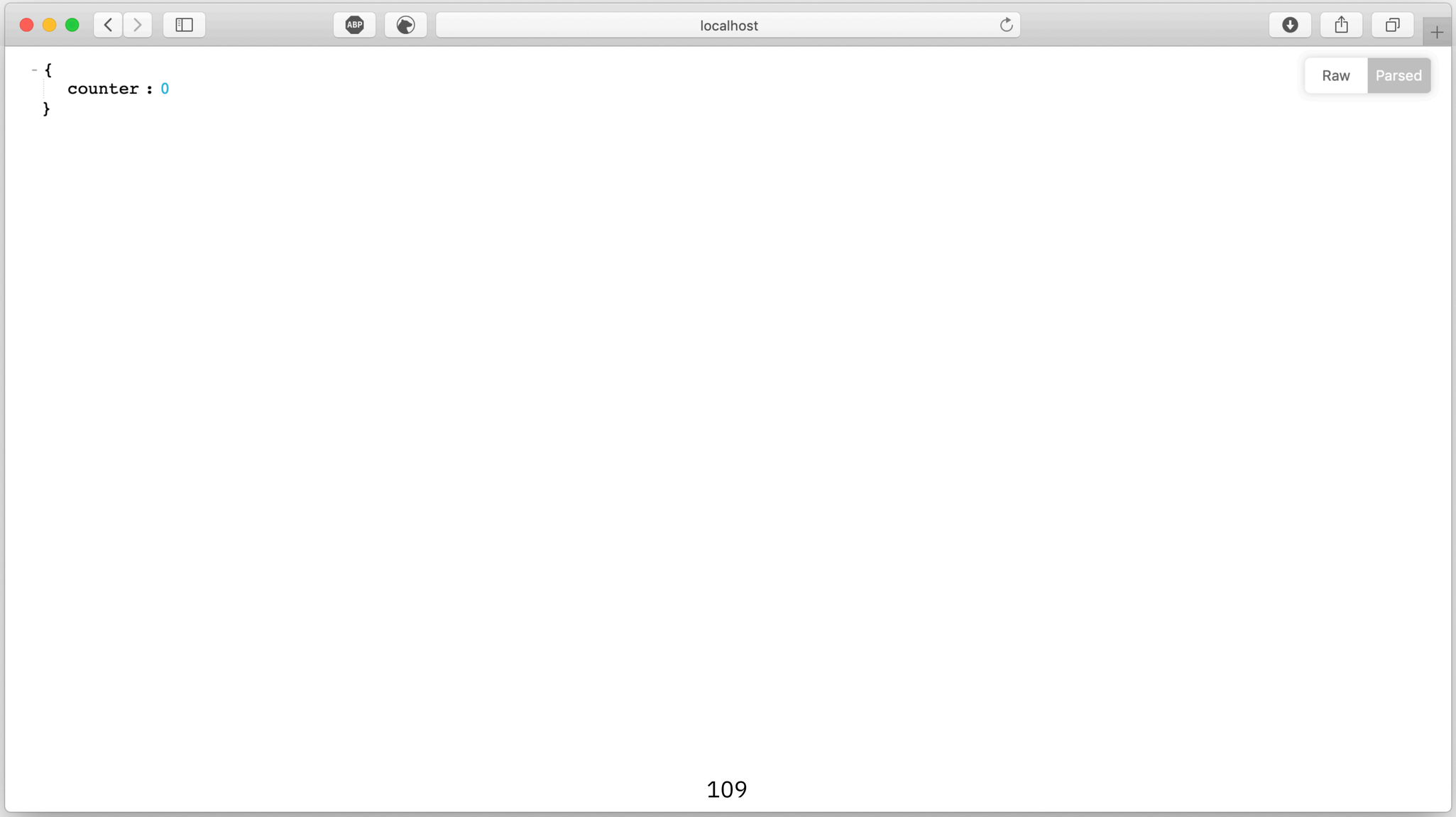
- We can use the `--link` command to tell lab-docker where redis is:
- We use an environment variable to give our app the new DATABASE_URI

```
– docker run --rm -p 8080:8080
  --link redis
  -e DATABASE_URI="redis://redis:6379/0"
  hitcounter:1.0
```

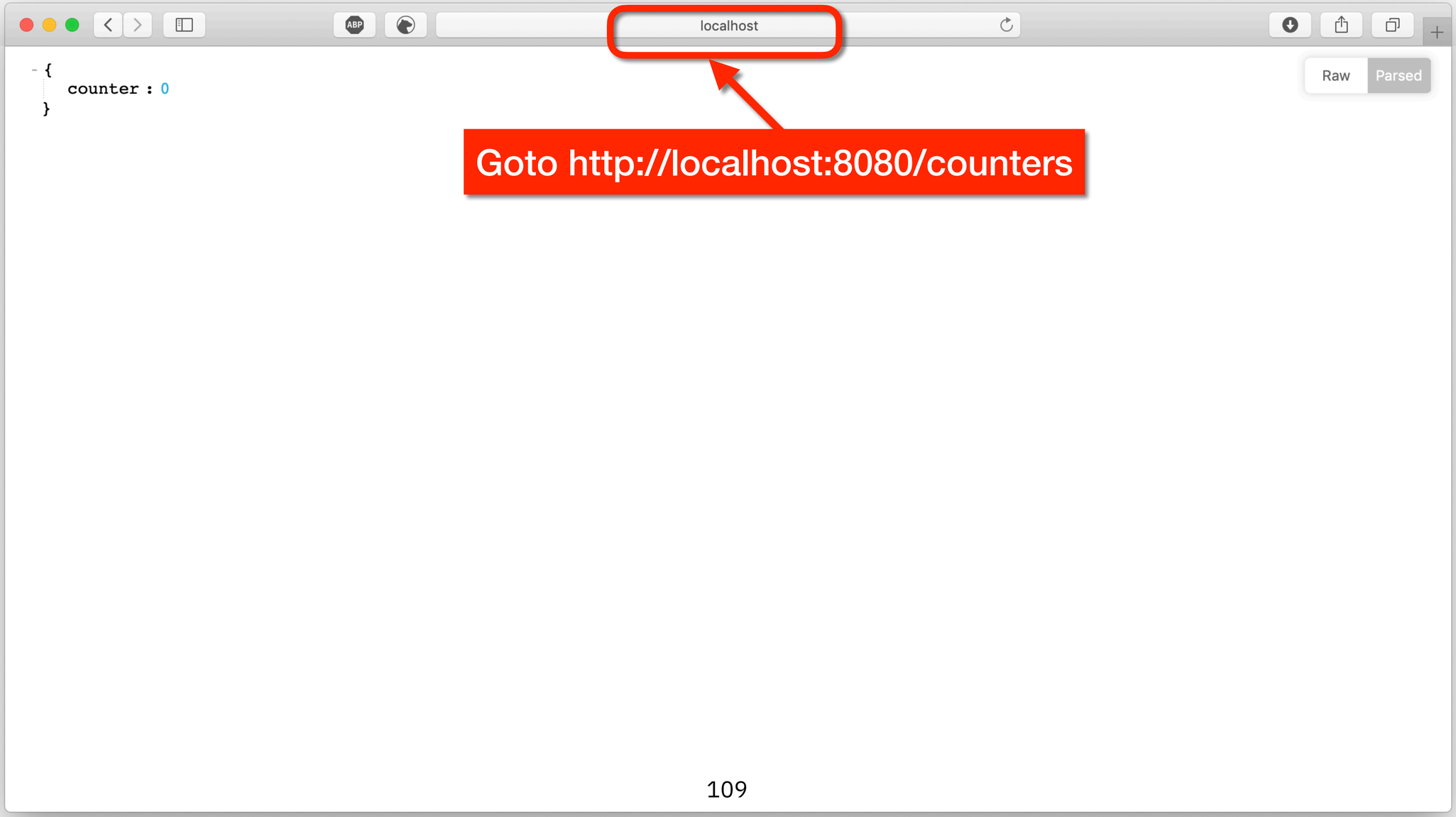
# Linking Containers

```
$ docker run --rm -p 8080:8080 --link redis -e DATABASE_URI="redis://:@redis:6379/0" hitcounter:1.0

[2019-11-20 01:18:02 +0000] [1] [INFO] Starting gunicorn 20.0.0
[2019-11-20 01:18:02 +0000] [1] [INFO] Listening at: http://0.0.0.0:8080 (1)
[2019-11-20 01:18:02 +0000] [1] [INFO] Using worker: sync
[2019-11-20 01:18:02 +0000] [9] [INFO] Booting worker with pid: 9
[2019-11-20 01:18:02 +0000] [9] [INFO] ************************************************************
[2019-11-20 01:18:02 +0000] [9] [INFO] **********  H I T   C O U N T E R   S E R V I C E  **********
[2019-11-20 01:18:02 +0000] [9] [INFO] ************************************************************
[2019-11-20 01:18:02 +0000] [9] [INFO] Service inititalized!
```

# Try Again



# Try Again


localhost
Raw
Parsed
{
counter : 0
}
Goto http://localhost:8080/counters

# Running Commands

- Start a container called "redis"
  - `docker run -d --name redis -p 6379:6379 redis:alpine`
- Call `redis-cli` with parameters
  - `docker exec -it redis redis-cli incr mycounter`

# Multi-stage Builds

```
FROM python:3.7-alpine as base

FROM base as builder
WORKDIR /install
COPY requirements.txt /requirements.txt
RUN pip install --install-option="--prefix=/install" -r /
requirements.txt

FROM base
COPY --from=builder /install /usr/local

WORKDIR /app
COPY src /app

CMD ["gunicorn", "-w 4", "main:app"]
```

# Docker Compose

- Allows you to deploy multiple containers at the same time
- Allows linking of containers together so that they can talk to each other
- Containers can be managed easily using tag names

# docker-compose.yml

From the web app, postgres is available at:  postgres://db:5432

```
version: "3"
services:
  web:
    build: .
    ports:
      - "8080:8080"
  db:
    image: postgres
    ports:
      - "5432:5432"
```

# Wordpress Example

```
version: '3'

services:
  wordpress:
    image: wordpress
    ports:
      - "8080:80"
    networks:
      - overlay
    restart: always

  mysql:
    image: mysql
    volumes:
       - wordpress-data:/var/lib/mysql/data
    networks:
       - overlay
    restart: always

volumes:
  wordpress-data:

networks:
  overlay:
```

# Compose for our App

```
version: "3"
services:
  app:
    build: .
    image: hitcounter:1.0
    container_name: hitcounter
   ports:
      - "8080:8080"
    environment:
      DATABASE_URI: "redis://:@redis:6379/0"
    depends_on:
      - redis
    networks:
      - web

  redis:
    image: redis:alpine
    restart: always
   ports:
      - "6379:6379"
    volumes:
      - redis-data:/data
    networks:
      - web

volumes:
  redis-data:

networks:
  web:
```

# Stop existing Redis container

- We can bring down the existing Redis containers with:

```
$ docker stop redis
redis
```

```
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS   PORTS   NAMES
```

# Docker-compose

- We can bring up both containers together:

```
$ docker-compose up -d

Creating network "vagrant_web" with the default driver
Creating volume "vagrant_redis-data" with default driver
Creating vagrant_redis_1 ... done
Creating hitcounter      ... done

$ docker-compose ps

     Name                    Command               State           Ports
-------------------------------------------------------------------------------
hitcounter        gunicorn --log-level=info  ...   Up      0.0.0.0:8080->8080/tcp
vagrant_redis_1   docker-entrypoint.sh redis ...   Up      0.0.0.0:6379->6379/tcp

$ docker-compose down

Stopping hitcounter      ... done
Stopping vagrant_redis_1 ... done
Removing hitcounter      ... done
Removing vagrant_redis_1 ... done
Removing network vagrant_web
```

# Docker Compose management

- docker-compose up
- docker-compose build
- docker-compose start | stop
- docker-compose ps
- ...and more

# docker history

- If you want to see the commands that create the layers in your docker image you can use the `history` option

```
$ docker history hitcounter:1.0
IMAGE               CREATED             CREATED BY                                      SIZE
d6d0aa01e77d        15 seconds ago      /bin/sh -c #(nop)  CMD ["gunicorn" "--log-le…   0B
0ab3d6deb2b7        15 seconds ago      /bin/sh -c #(nop)  ENV GUNICORN_BIND=0.0.0.0…   0B
0cfc9509778e        15 seconds ago      /bin/sh -c #(nop)  EXPOSE 8080                  0B
cf14f70d51a5        15 seconds ago      /bin/sh -c #(nop)  ENV PORT=8080                0B
de8ae3be436f        16 seconds ago      /bin/sh -c #(nop) COPY dir:72b9695da55ddcd17…   46.3kB
55336121980d        16 seconds ago      /bin/sh -c pip install --no-cache-dir -r req…   21.1MB
eb61244b5956        22 seconds ago      /bin/sh -c #(nop) COPY file:6afa89cc99483f80…   154B
b3b67de03a26        23 seconds ago      /bin/sh -c #(nop) WORKDIR /app                  0B
07ee12a5eb2a        5 days ago          /bin/sh -c #(nop)  CMD ["python3"]              0B
<missing>           5 days ago          /bin/sh -c set -ex;   savedAptMark="$(apt-ma…   7.42MB
<missing>           5 days ago          /bin/sh -c #(nop)  ENV PYTHON_GET_PIP_SHA256…   0B
<missing>           5 days ago          /bin/sh -c #(nop)  ENV PYTHON_GET_PIP_URL=ht…   0B
<missing>           5 days ago          /bin/sh -c #(nop)  ENV PYTHON_PIP_VERSION=19…   0B
<missing>           5 days ago          /bin/sh -c cd /usr/local/bin  && ln -s idle3…   32B
<missing>           5 days ago          /bin/sh -c set -ex   && savedAptMark="$(apt-…   94.9MB
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENV PYTHON_VERSION=3.7.5     0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENV GPG_KEY=0D96DF4D4110E…   0B
<missing>           4 weeks ago         /bin/sh -c apt-get update && apt-get install…   7.03MB
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENV LANG=C.UTF-8             0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  ENV PATH=/usr/local/bin:/…   0B
<missing>           4 weeks ago         /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>           4 weeks ago         /bin/sh -c #(nop) ADD file:74b2987cacab5a6b0…   69.2MB
```

# Useful Docker Commands

- **Create a Dockerfile for your image**
  - `vi Dockerfile`
- **Build an image with:**
  - `docker build –t <your_image_name> .`
- **Run a container with:**
  - `docker run -d --name <your_container_name> <your_image_name>`
- **Show the containers logs with:**
  - `docker logs <your_container_name>`
- **Stop the container with:**
  - `docker stop <your_container_name>`
- **Remove the container with:**
  - `docker rm <your_container_name>`
- **Remove the image with:**
  - `docker rmi <your_image_name>`

# Useful Docker Flags

- **Linking** (`--link <other>`)
  - You can link containers so that they don't need to know each others ip addresses
- **Volumes** (`-v <host>:<cont>`)
  - You can mount volumes so that data can remain outside of the container and keep the containers stateless
- **Ports** (`-P or -p <host_port>:<cont_port>`)
  - You can map ports to expose them outside of the container <host>:<container>
- **Hostname** (`-h <name>`)
  - Assigns a hostname to the container
- **Daemon** (`-d`)
  - Run the container as a daemon process
- **Fault Tolerance** (`--always-restart`)
  - Tells Docker to restart the container if the main process ever fails
- **Environment Variables** (`-e <ENV_VAR>=xxx`)
  - Passes environment variables into a container at startup

# Cleaning up Dangling Images

- Dangling images occur when a new images it build that supersedes it
- These images show `<none>` for their name and tag
- You can list them with: `docker images -f dangling=true`

```
$ docker images -f dangling=true
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              3f7a53d3c3a3        5 seconds ago       63.3 MB
```

# Cleaning up Dangling Images

- Dangling images occur when a new images it build that supersedes it
- These images show `<none>` for their name and tag
- You can list them with: `docker images -f dangling=true`

```
$ docker images -f dangling=true
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              3f7a53d3c3a3        5 seconds ago       63.3 MB
```

Dangling images that can be cleaned up

# Remove All Dangling Images

- We can automatically remove all dangling images with the following command:
  - `docker image prune`

```
$ docker image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N] y

Deleted: sha256:3f7a53d3c3a3a64ec79fea01747b97556c276bc851b65b8e24cfb1c2082d5cff
Deleted: sha256:d76a50fd288ec67ec7699bbb4f5b65b9f40b68790d1fb6c96ac39f60dddc146a
Deleted: sha256:56ee01767abea97505cef0646e4172cb18a36f06c4eff070ef09caa47be63564
```

# Cleanup Docker Volumes

- Whenever you share a folder, Docker creates a volume
- You can list dangling volumes with:
  - `docker volume ls -f dangling=true`
- You can remove all dangling volumes with:
  - `docker volume prune`

# Easy Way to Kill All Containers

- Here are two commands to quickly kill and remove all containers *(remember: "with great power comes great responsibility!)*

```
docker kill $(docker ps -aq) &&
docker rm $(docker ps -aq)
```

# Summary

- You just created your first Docker containers
- You learned how to create our own Images
- You deployed your containers to Bluemix
- You can now check your Dockerfile into github so that others can create the same containers when working on your project.