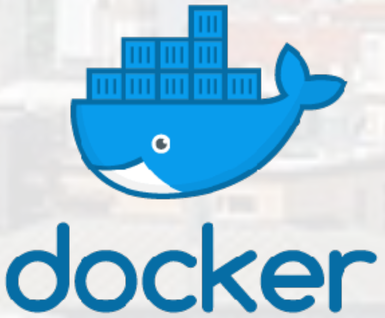
Introduction to Docker



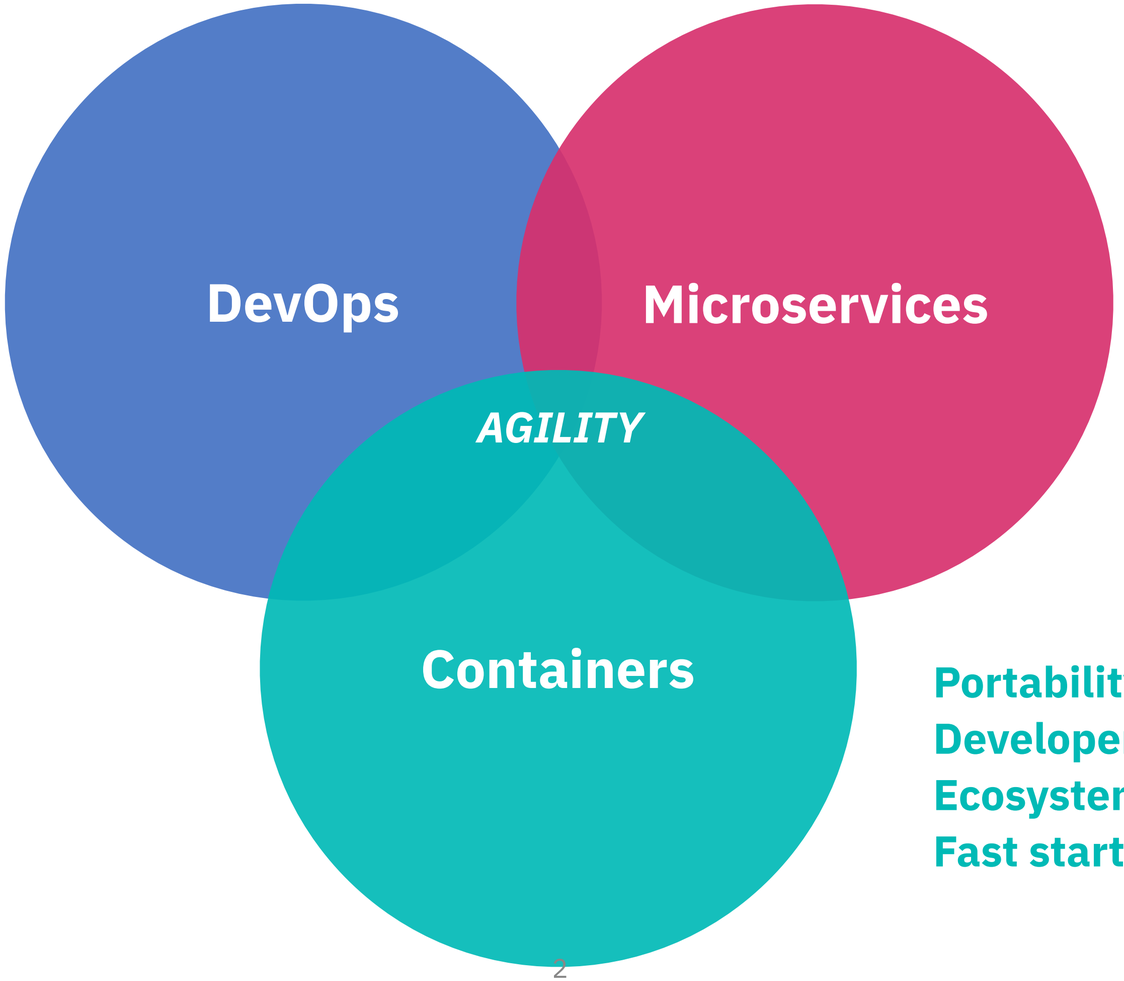
Instructor:

John J Rofrano

Senior Technical Staff Member, DevOps Champion
IBM T.J. Watson Research Center
rofrano@us.ibm.com (@JohnRofrano)

THE PERFECT STORM

Cultural Change
Automated Pipeline
Everything as Code
Immutable Infrastructure

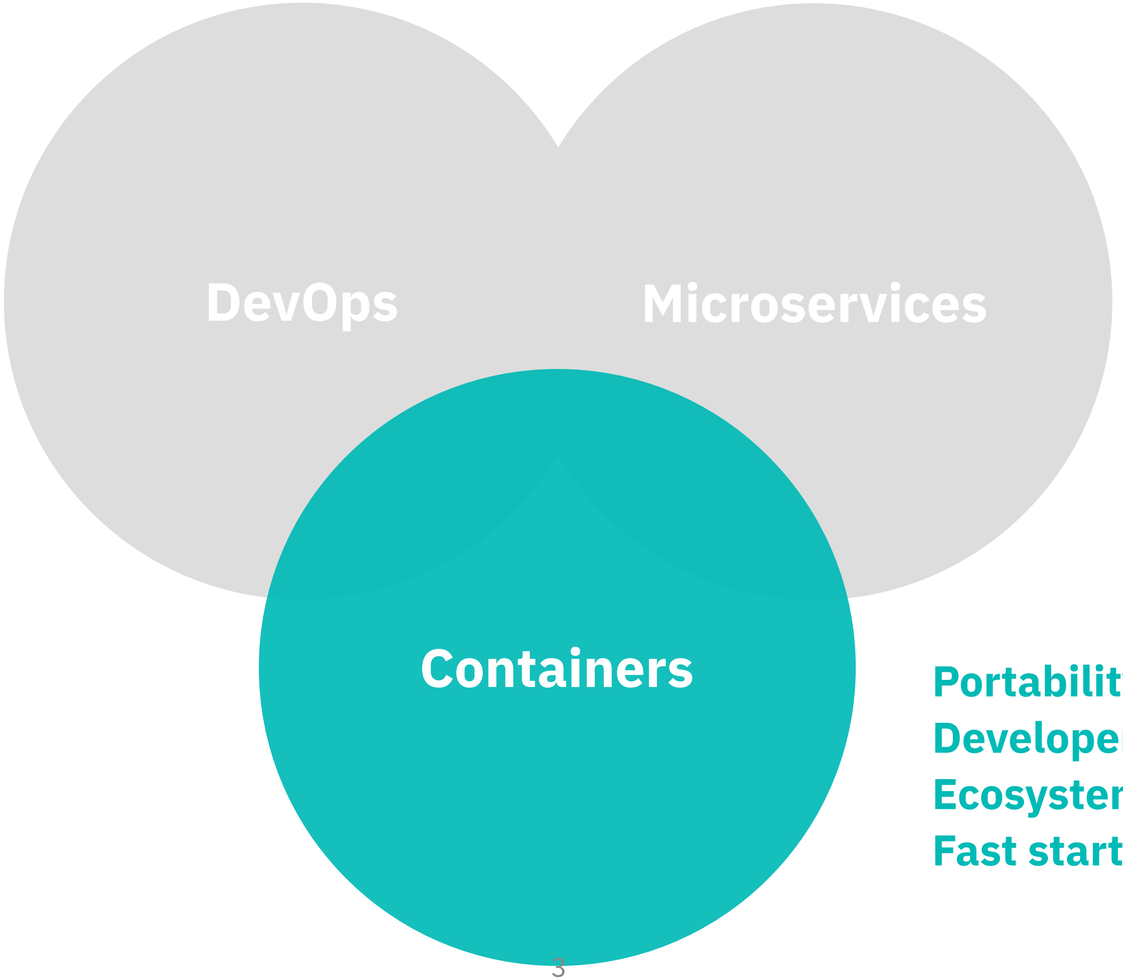


Loose Coupling/Binding
RESTful APIs
Designed to resist failures
Test by break / fail fast

Portability
Developer Centric
Ecosystem enabler
Fast startup

THE PERFECT STORM: Containers

Cultural Change
Automated Pipeline
Everything as Code
Immutable Infrastructure

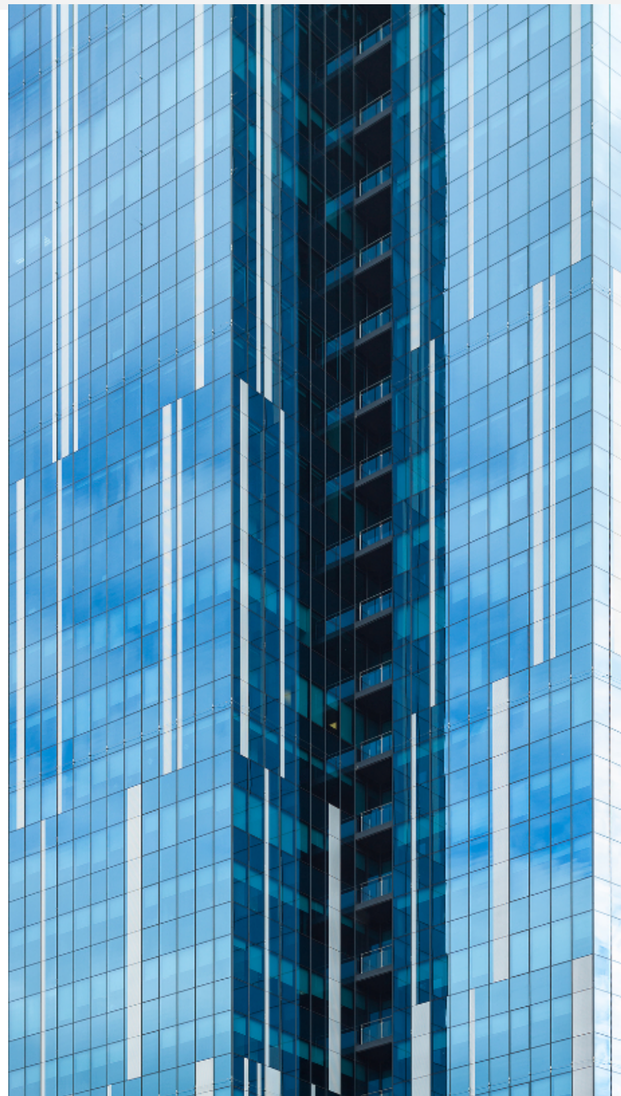


Loose Coupling/Binding
RESTful APIs
Designed to resist failures
Test by break / fail fast

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

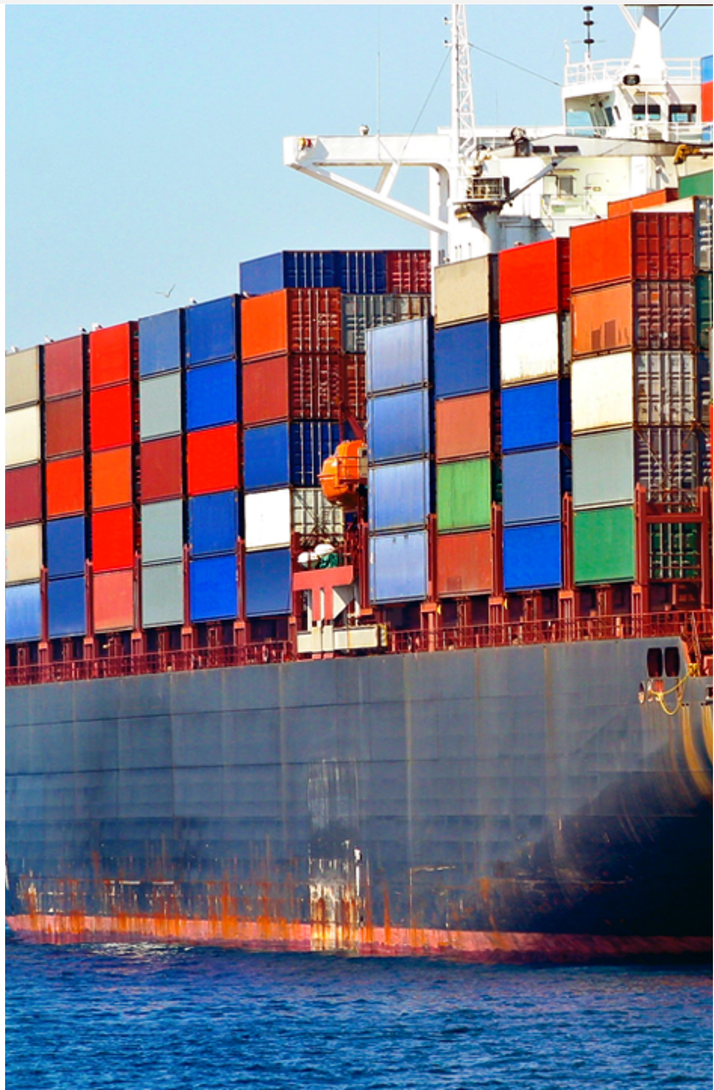


The Lure of Containers

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

How do you ship cargo of variable sizes efficiently?

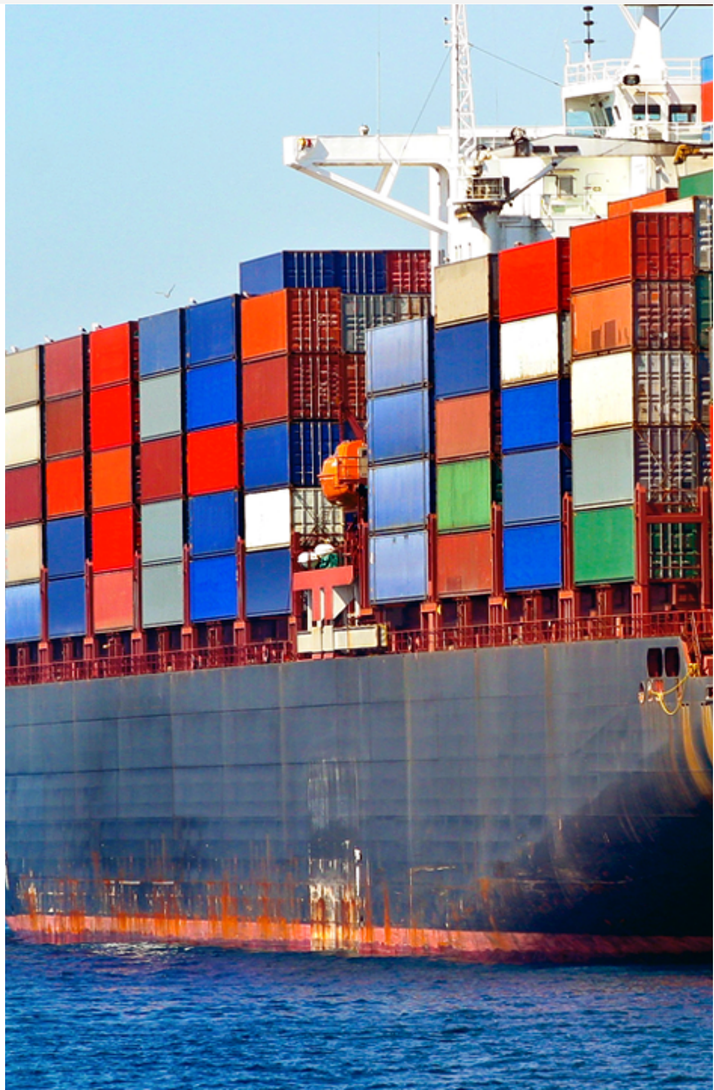
- Give the customer a standard sized container
- They can put anything they want in it, but it's their responsibility for what is inside of it
- Build transport ships that can efficiently hold these containers at scale
- Cargo size no longer matters because you only deal with the standard container



The IT Industry's Solution to Deploying Applications

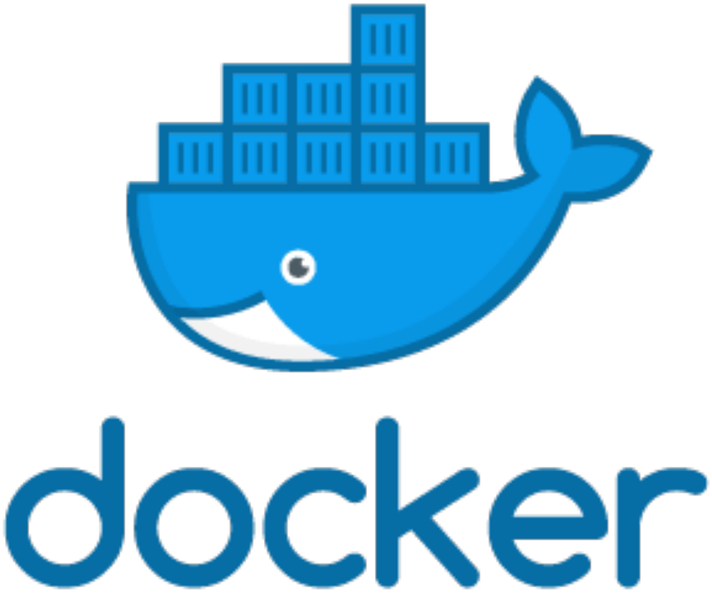
How do you deploy applications with variable dependencies?

- 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



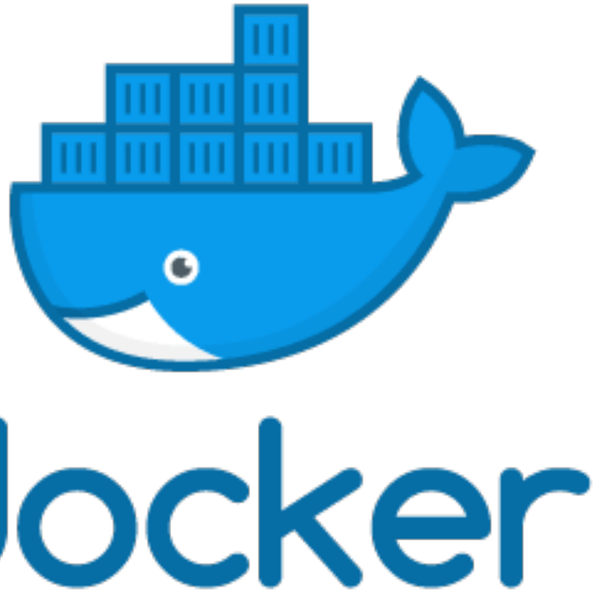
Docker Containers

- 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)



Docker Containers

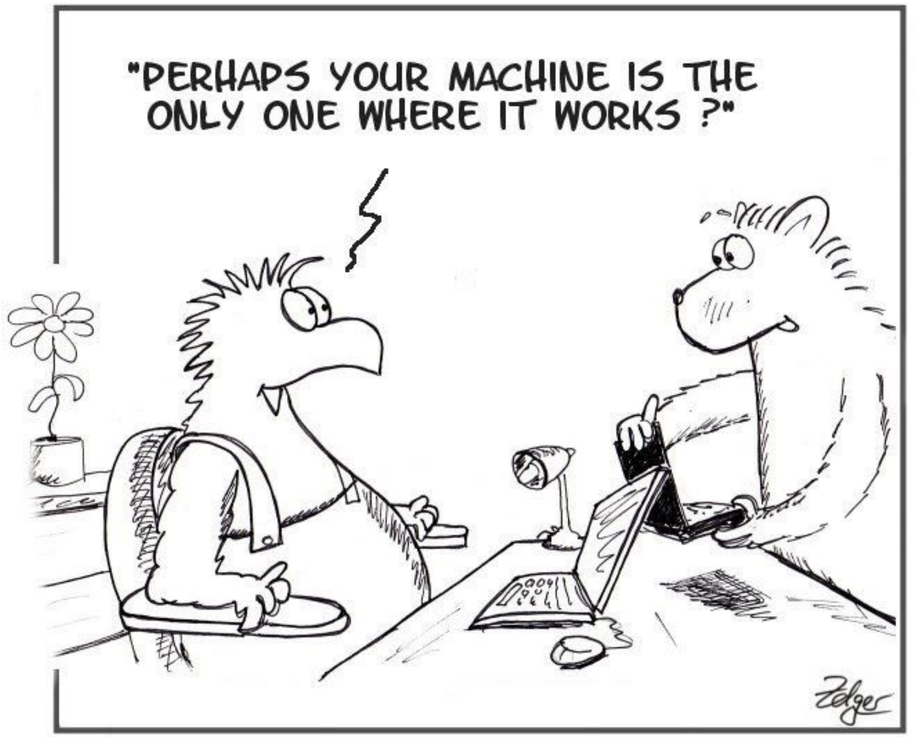
- Containers encapsulate a run-time environment
  - They can contain code, libraries, package manager, data, etc.
- Almost no overhead
  - Containers spin up in seconds not minutes like VMs
  - Native performance because there is no emulation
  - Package only what you need



Containers Enable Immutable Delivery

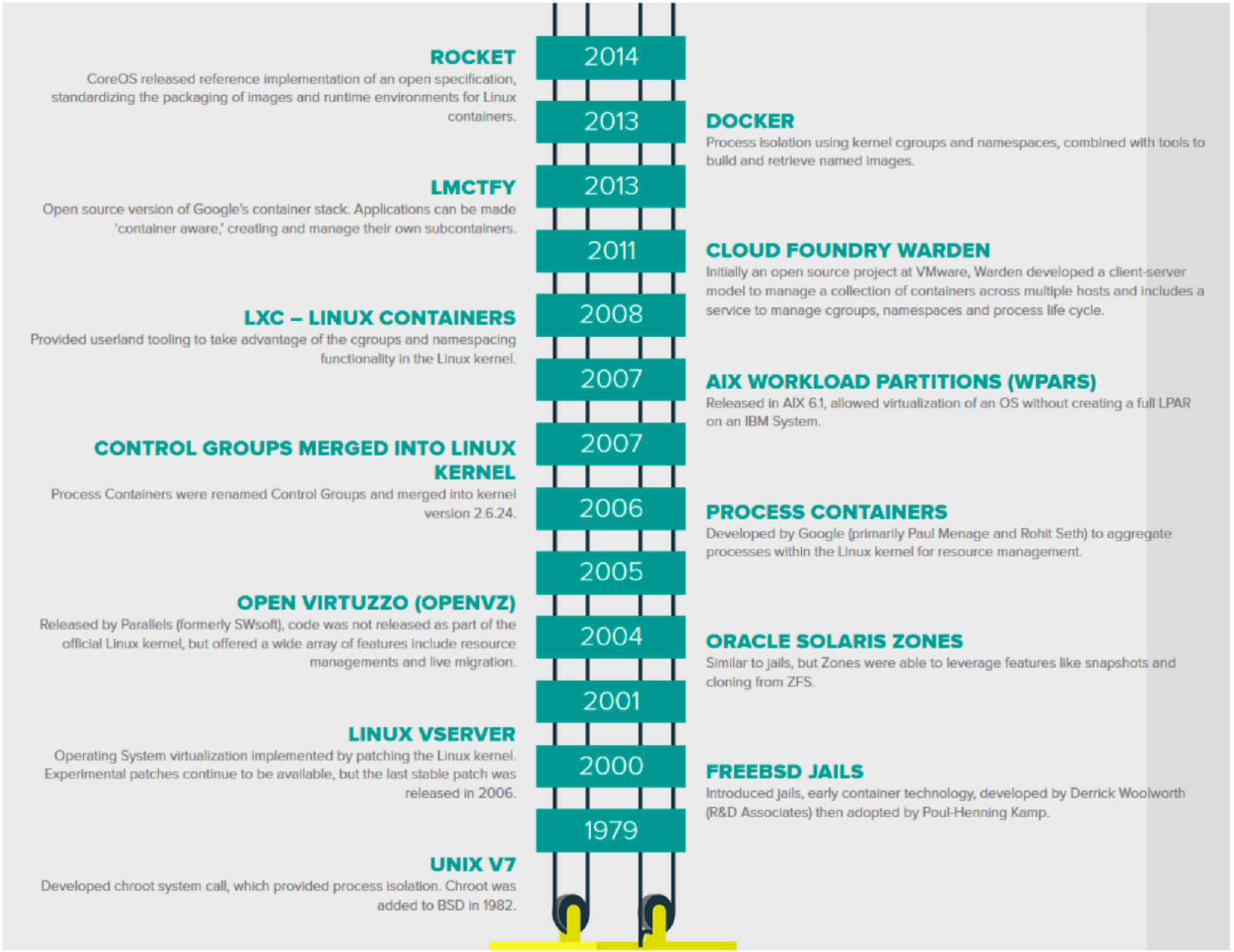
Build once... run anywhere

- 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

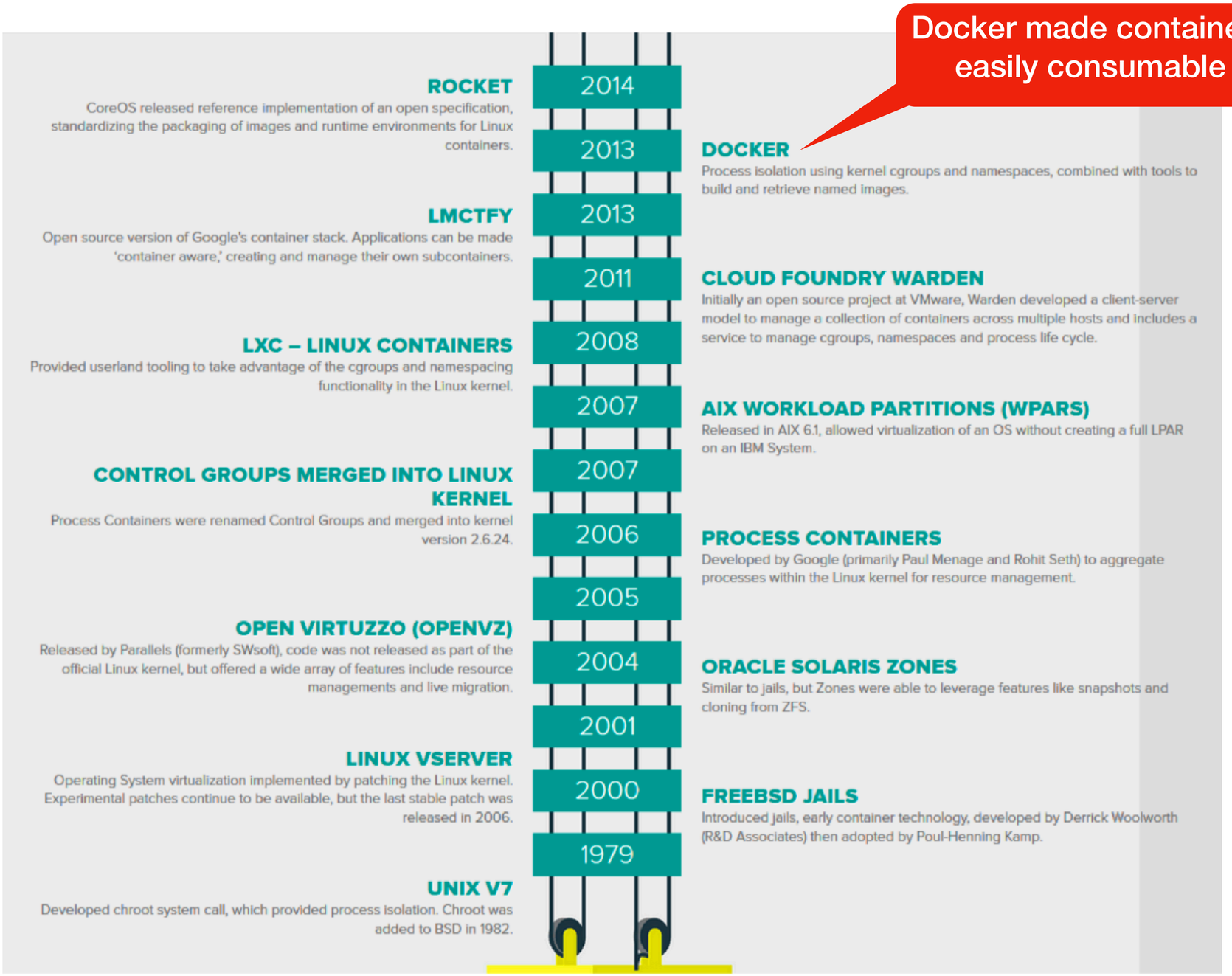


It works on my machine

Containers are not new

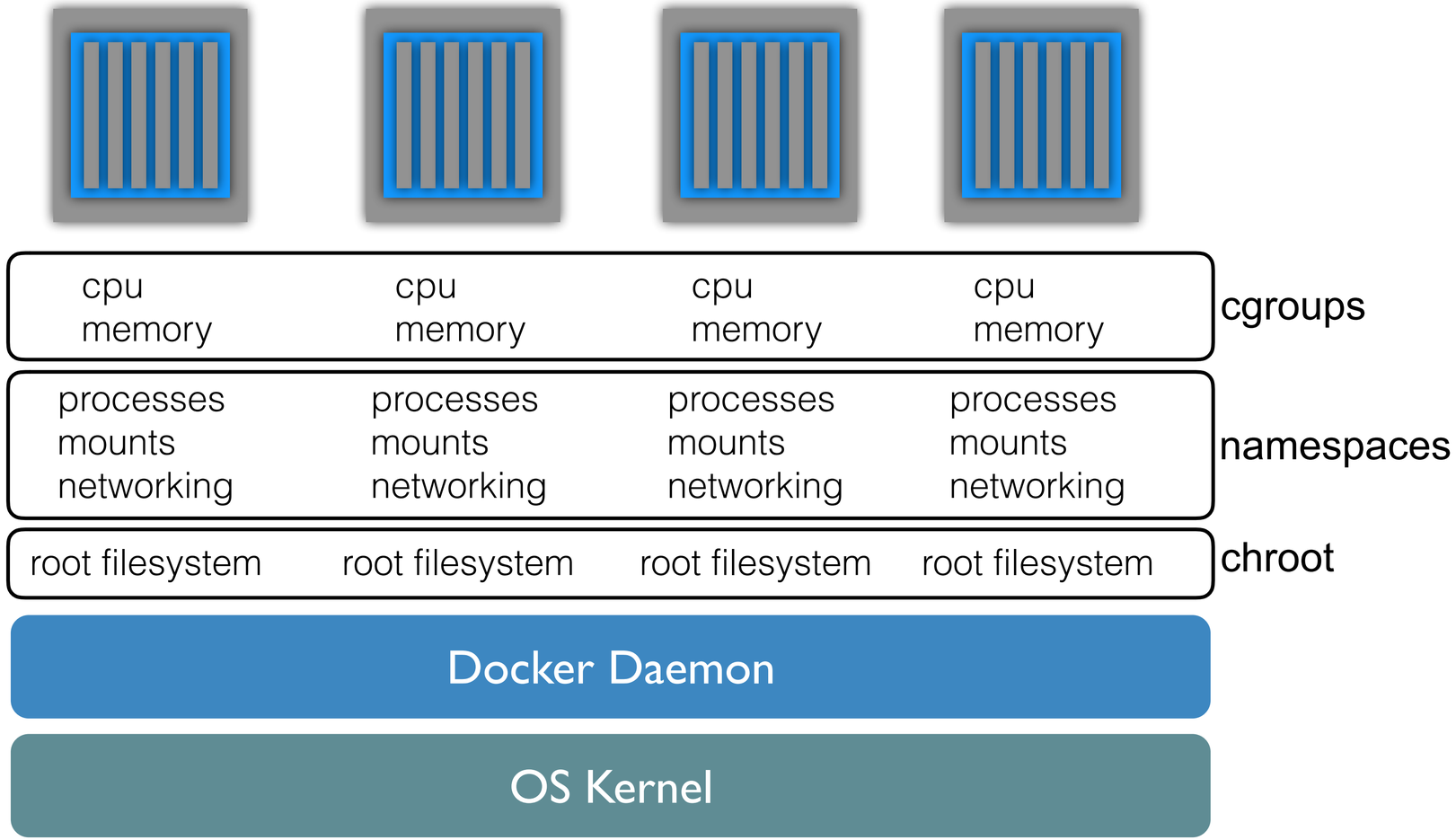


Containers are not new



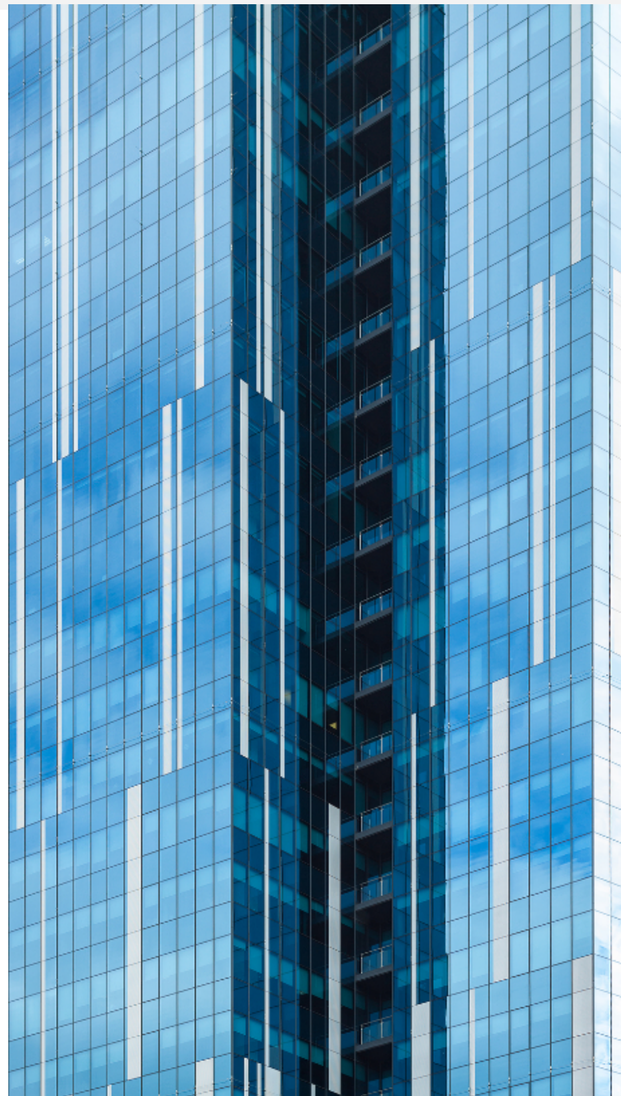
Containers are just Linux Capabilities Under the Covers

- 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



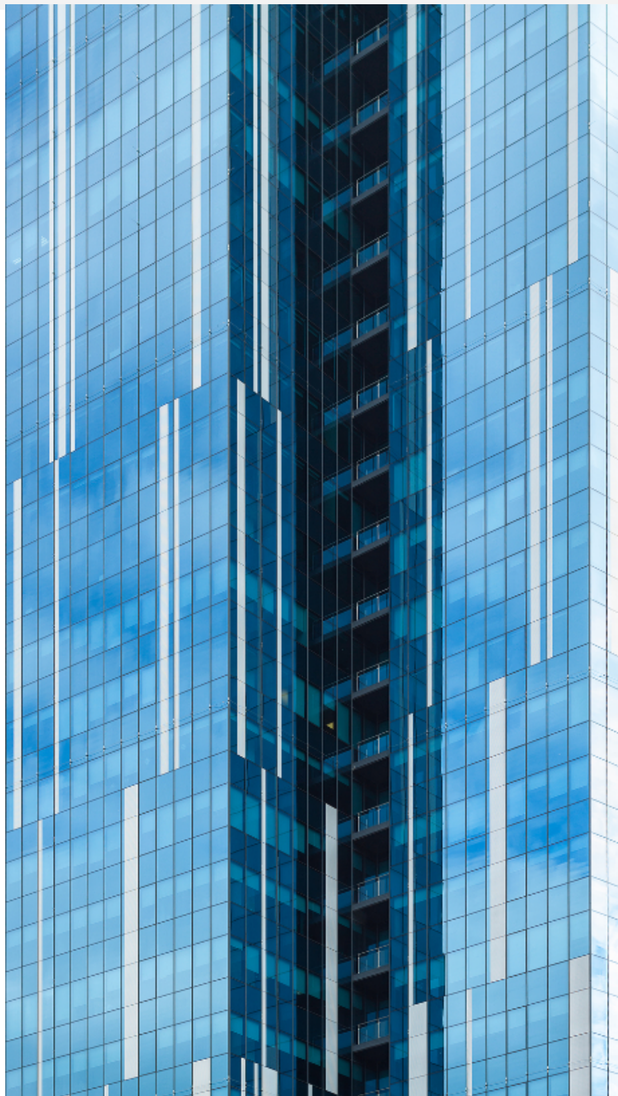
Docker Containers are just Linux...

- **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)



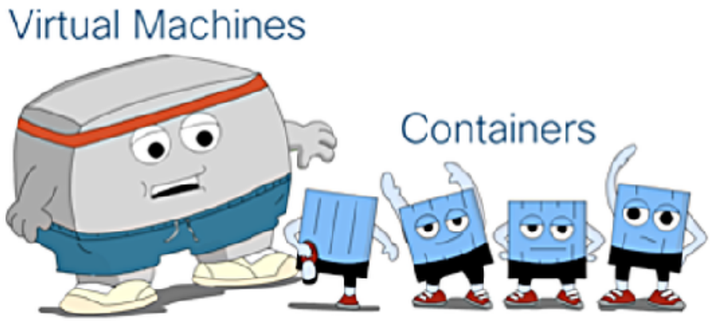
Put another way

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



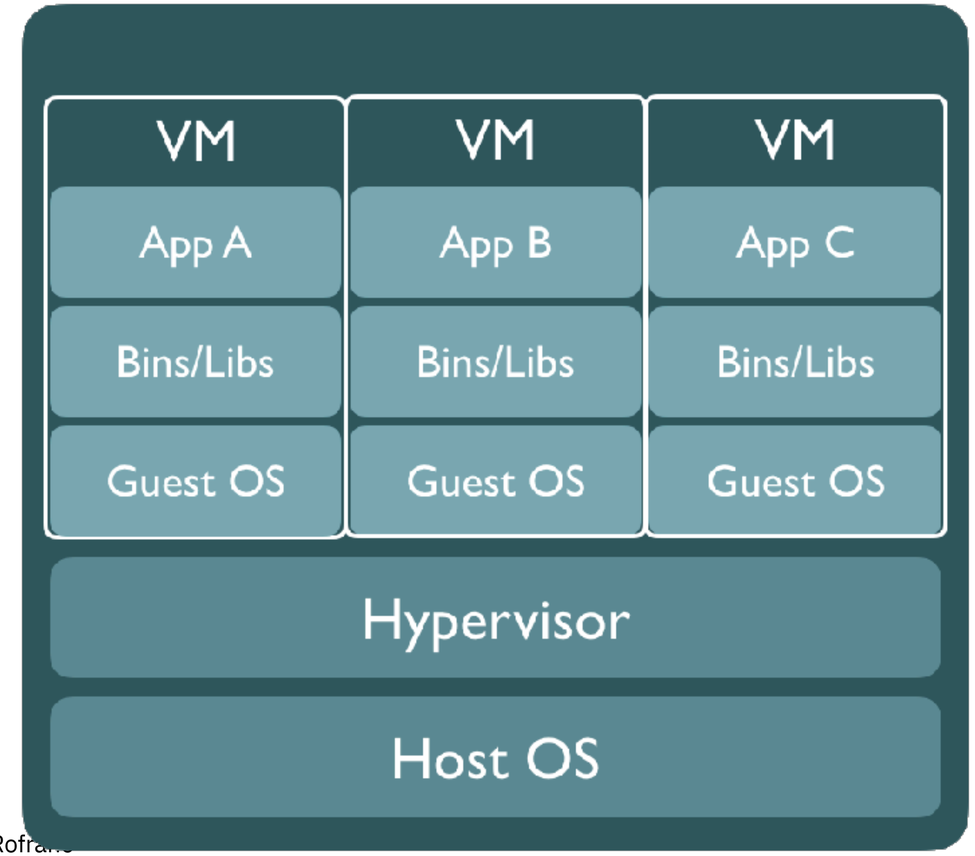
Containers vs Virtual Machines

- 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



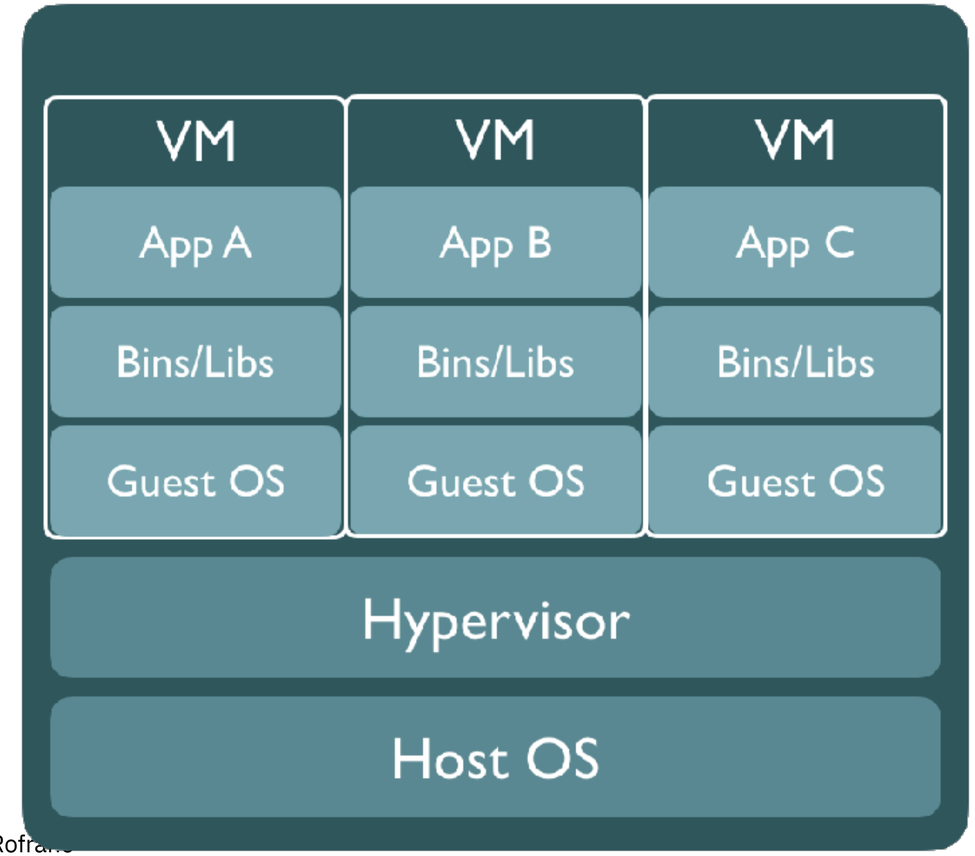
Containers vs Virtual Machines

Virtual Machines are heavy-weight emulations of real hardware

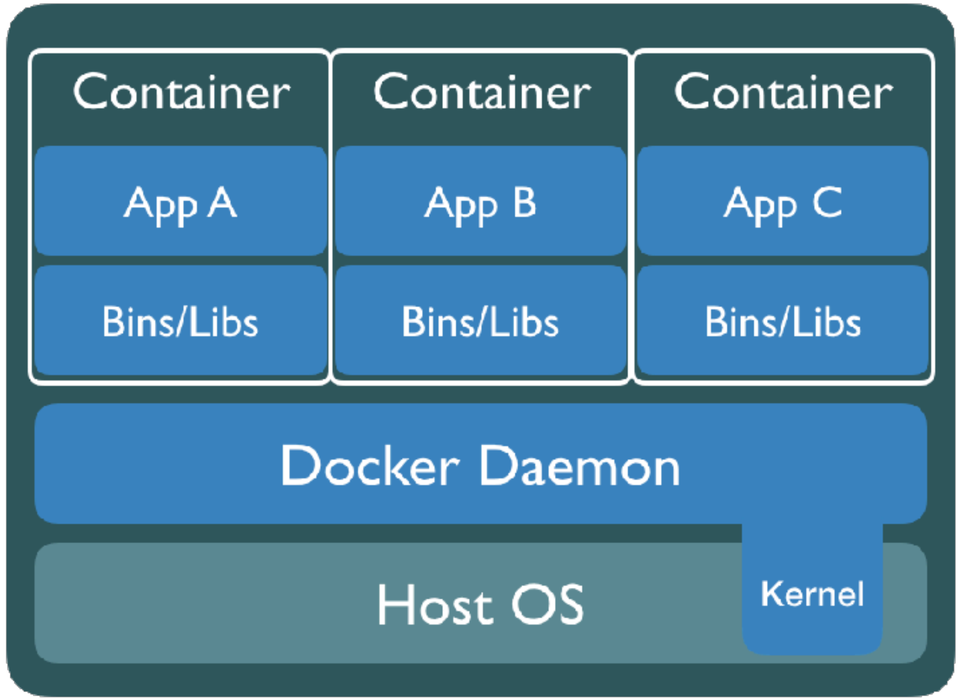


Containers vs Virtual Machines

Virtual Machines are heavy-weight emulations of real hardware

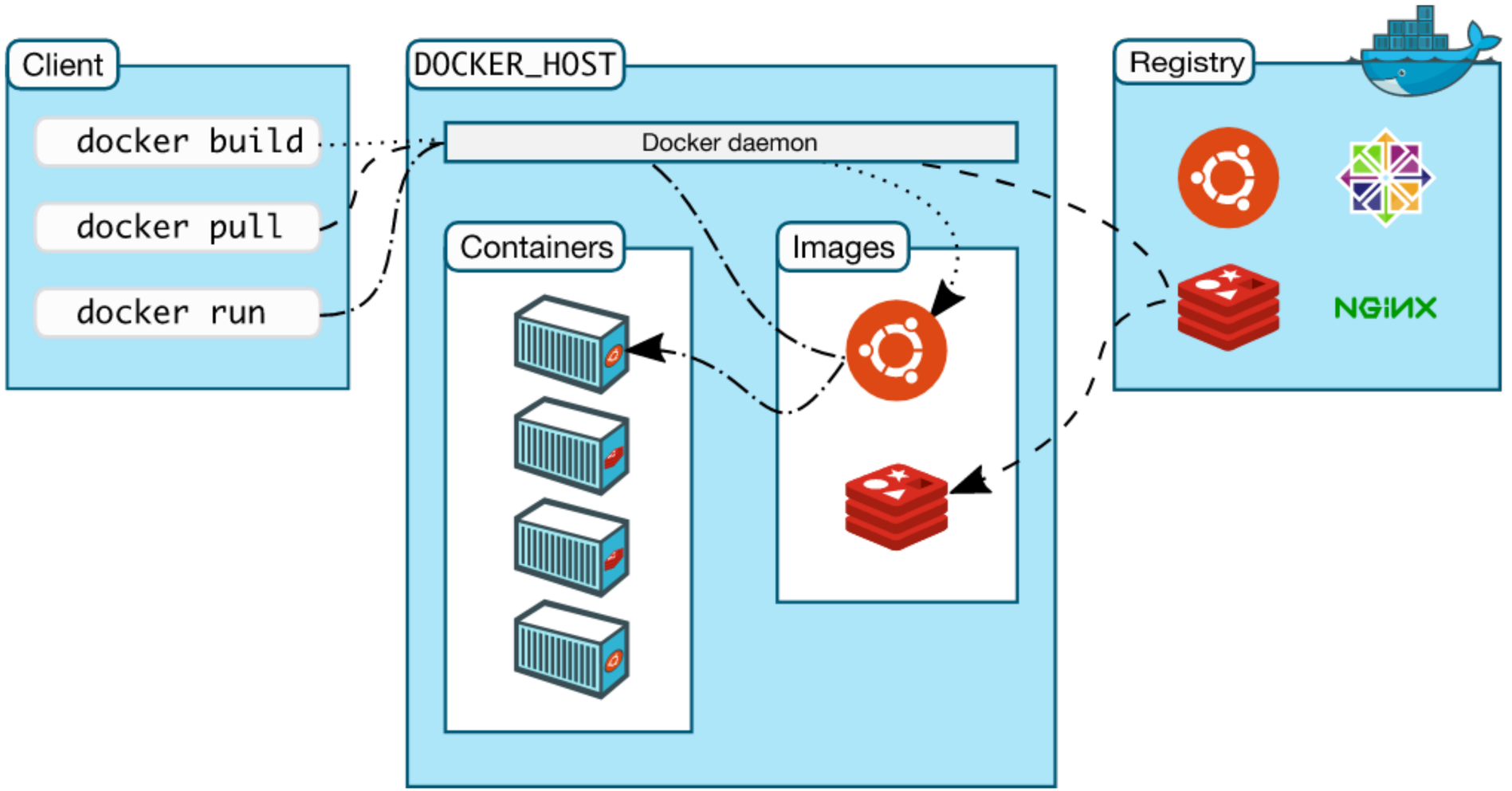


Containers are light-weight process
The app looks like it's running on the Host OS



Containers in Practice

Where do Containers Come From?



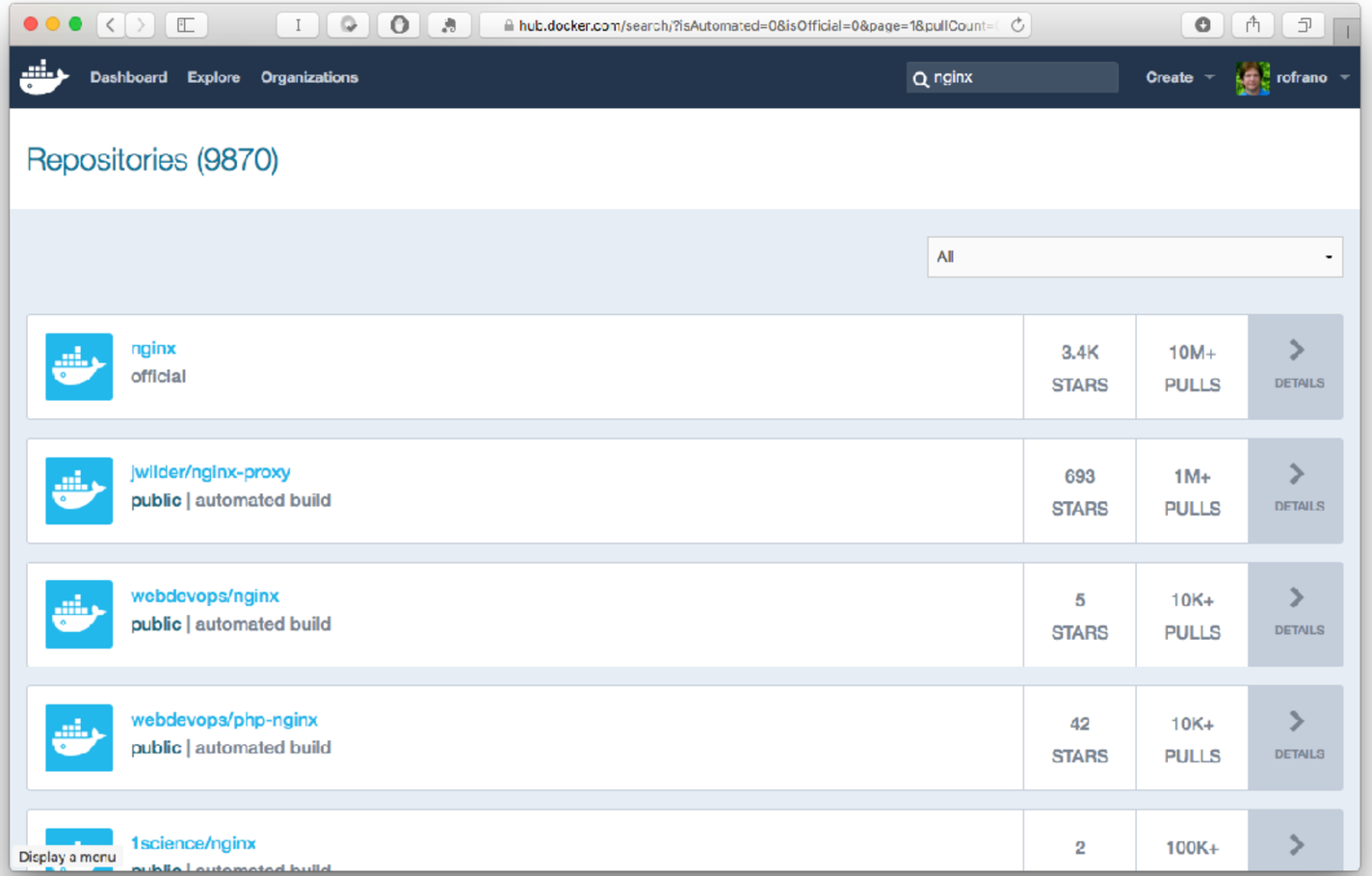
Images and containers

- 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

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.

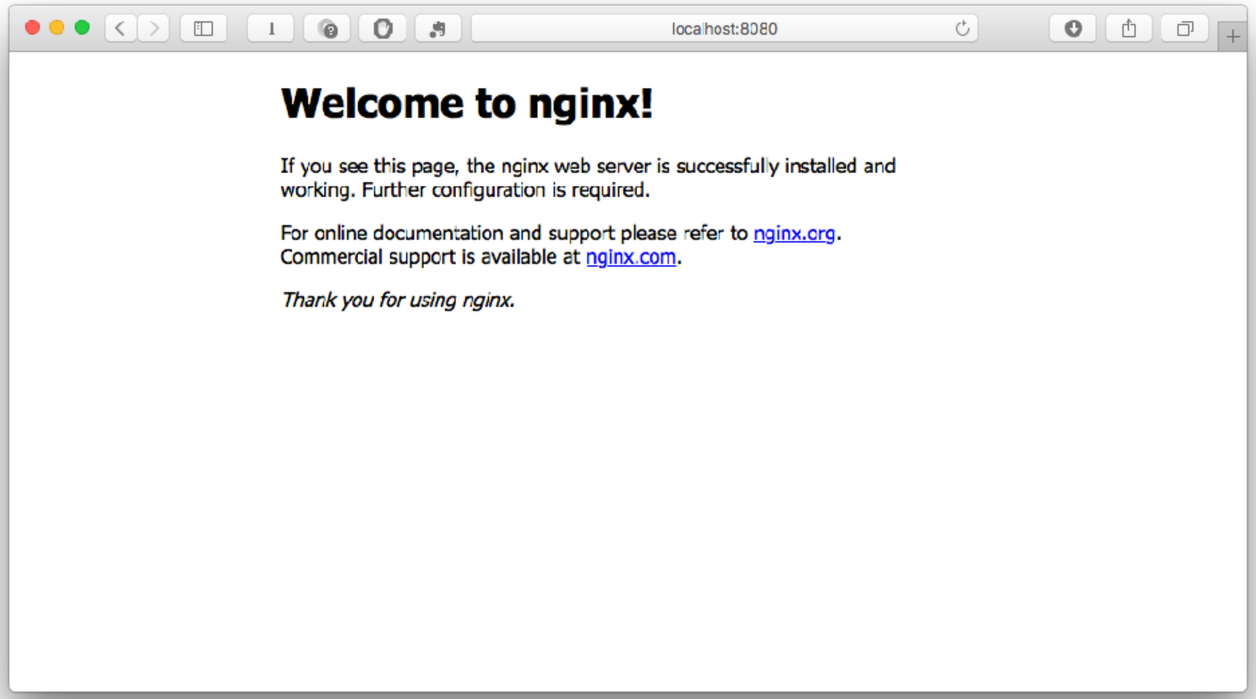
Nginx installed and running with one command

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



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
```

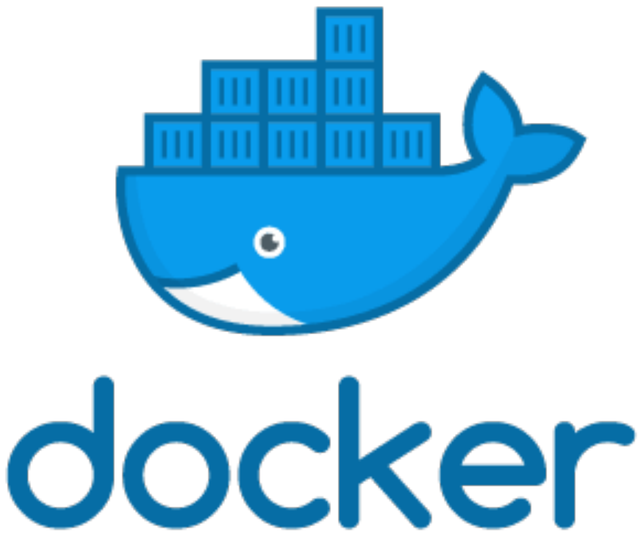
Stop the MySQL server with the following command:

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

To restart the MySQL server, use the following command:

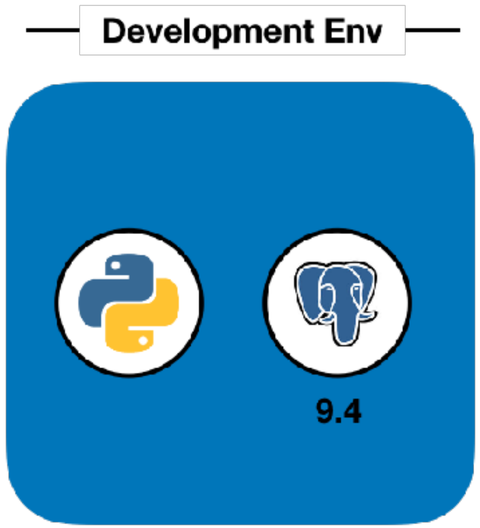
```
shell> sudo service mysql start
```

Wordpress and MySQL with Docker

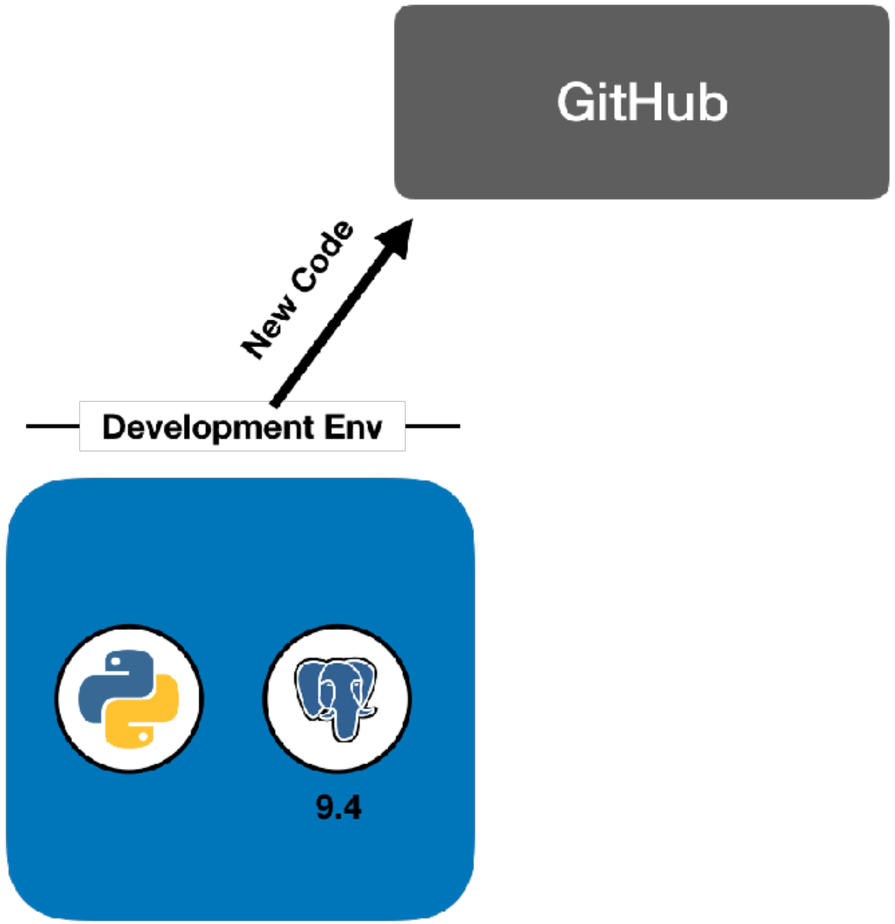



```
docker run -d --name db mysql

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

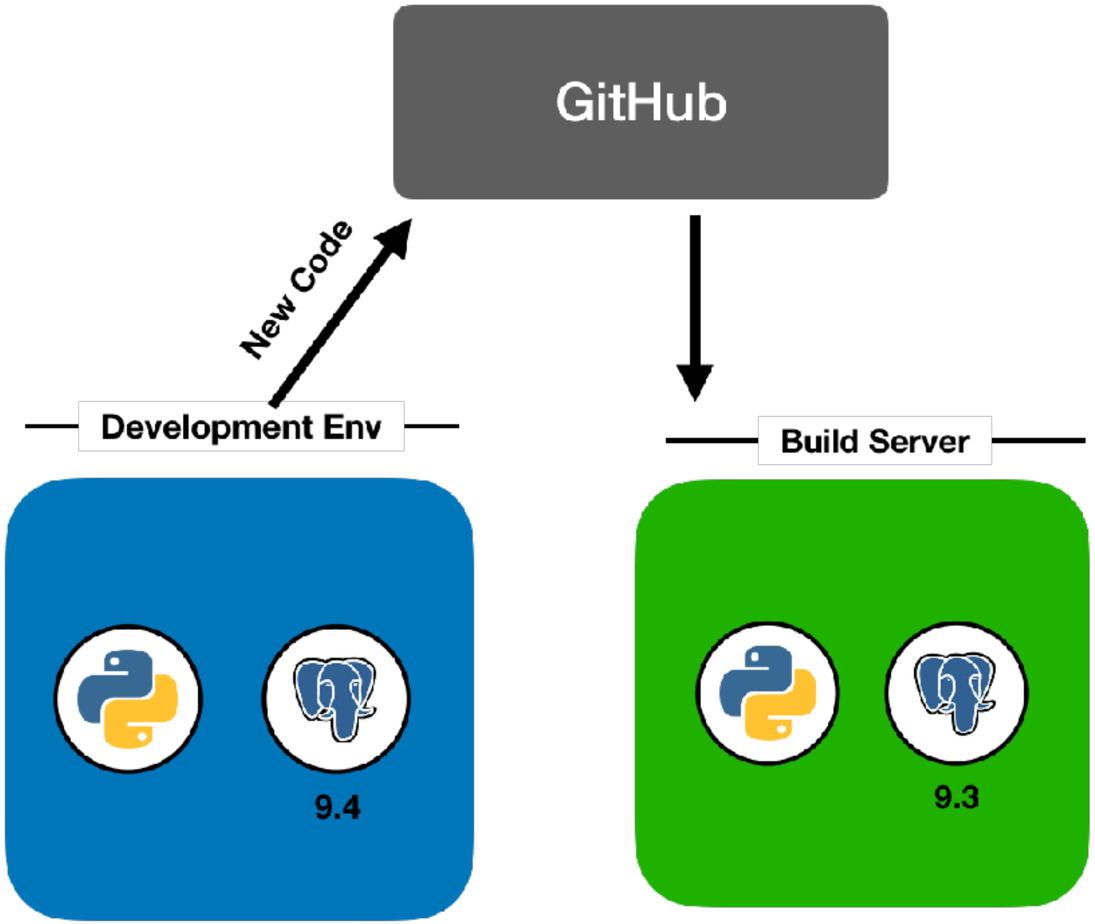
Traditional Development Workflow



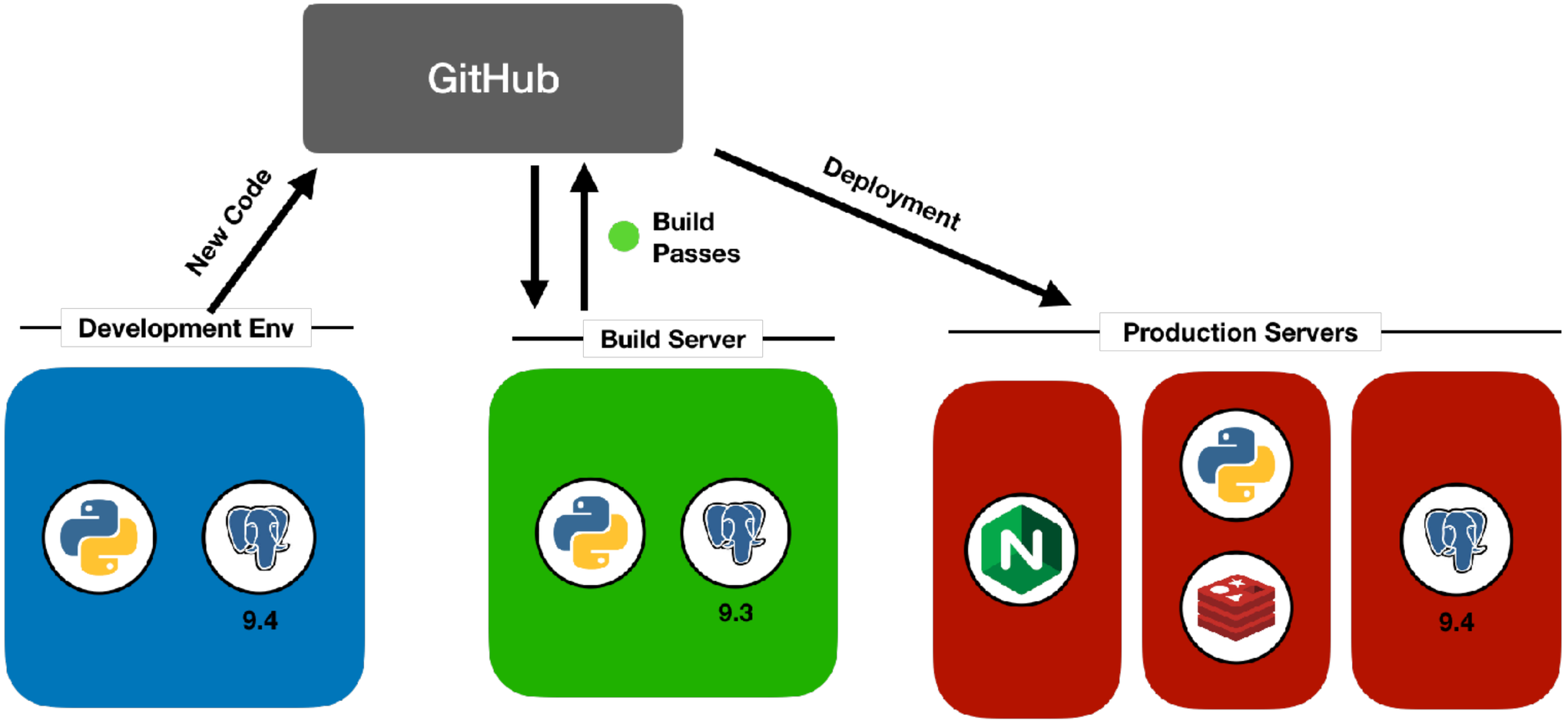
Traditional Development Workflow



Traditional Development Workflow

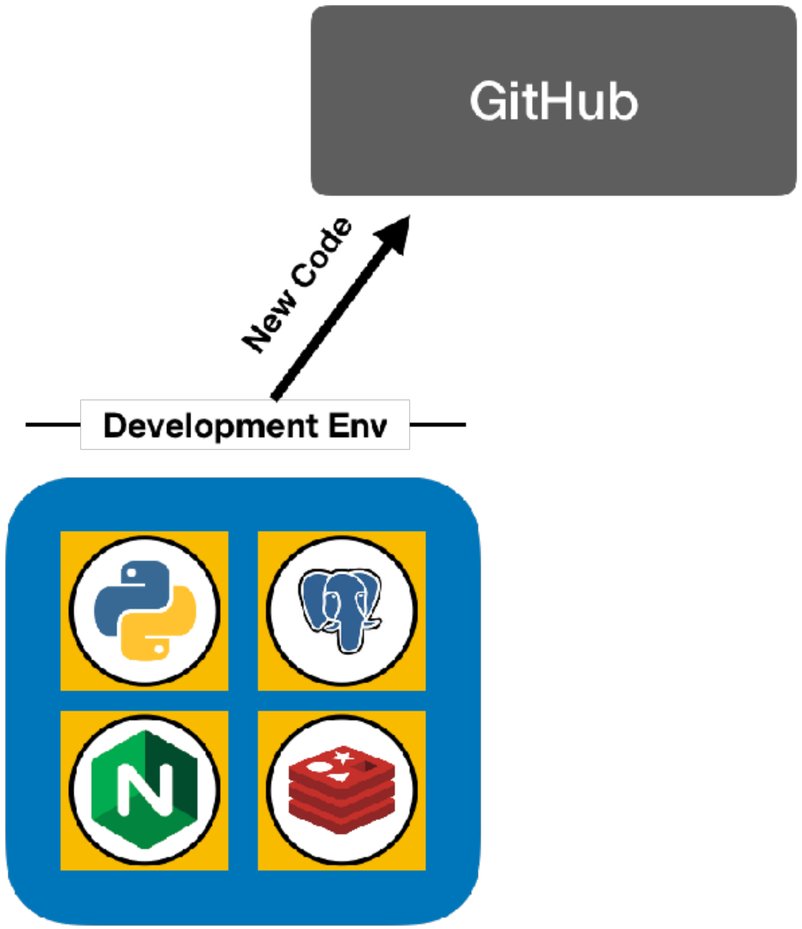


Traditional Development Workflow

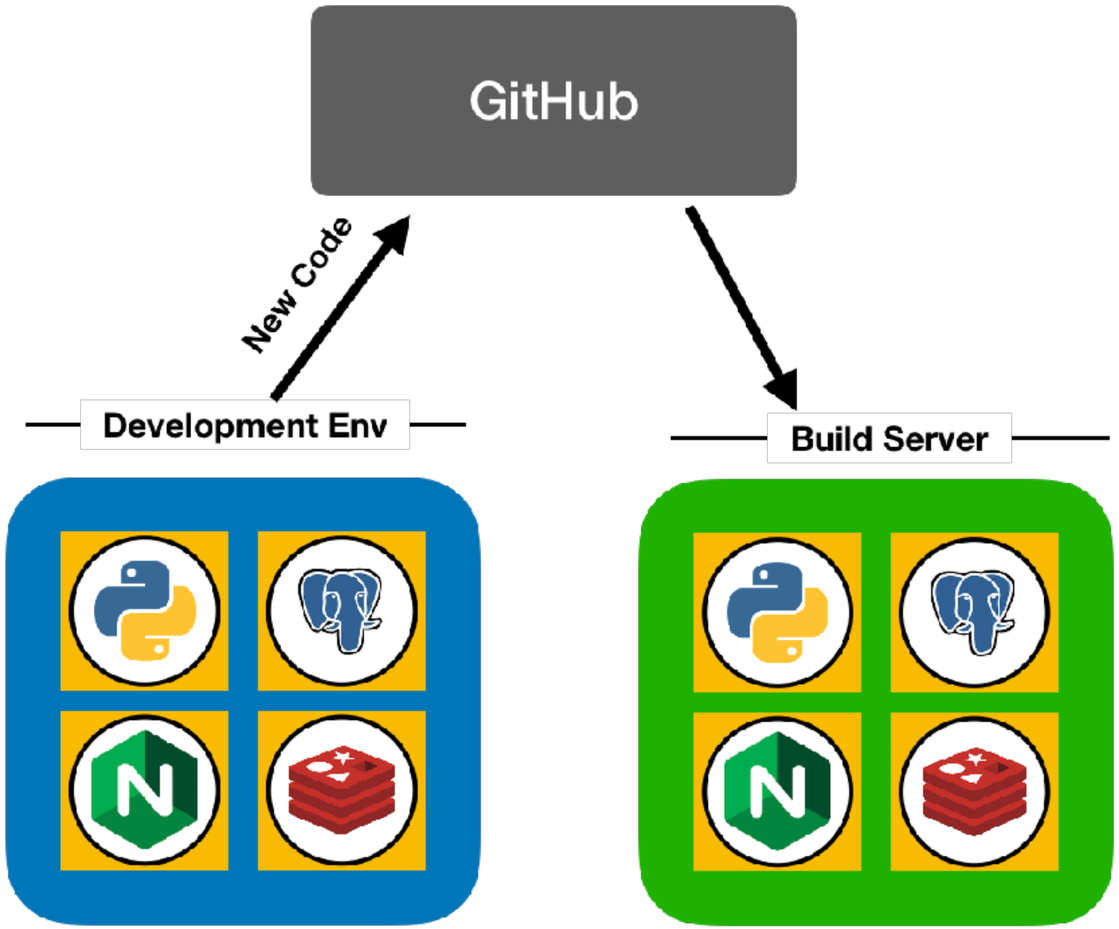


Containerized Development Workflow

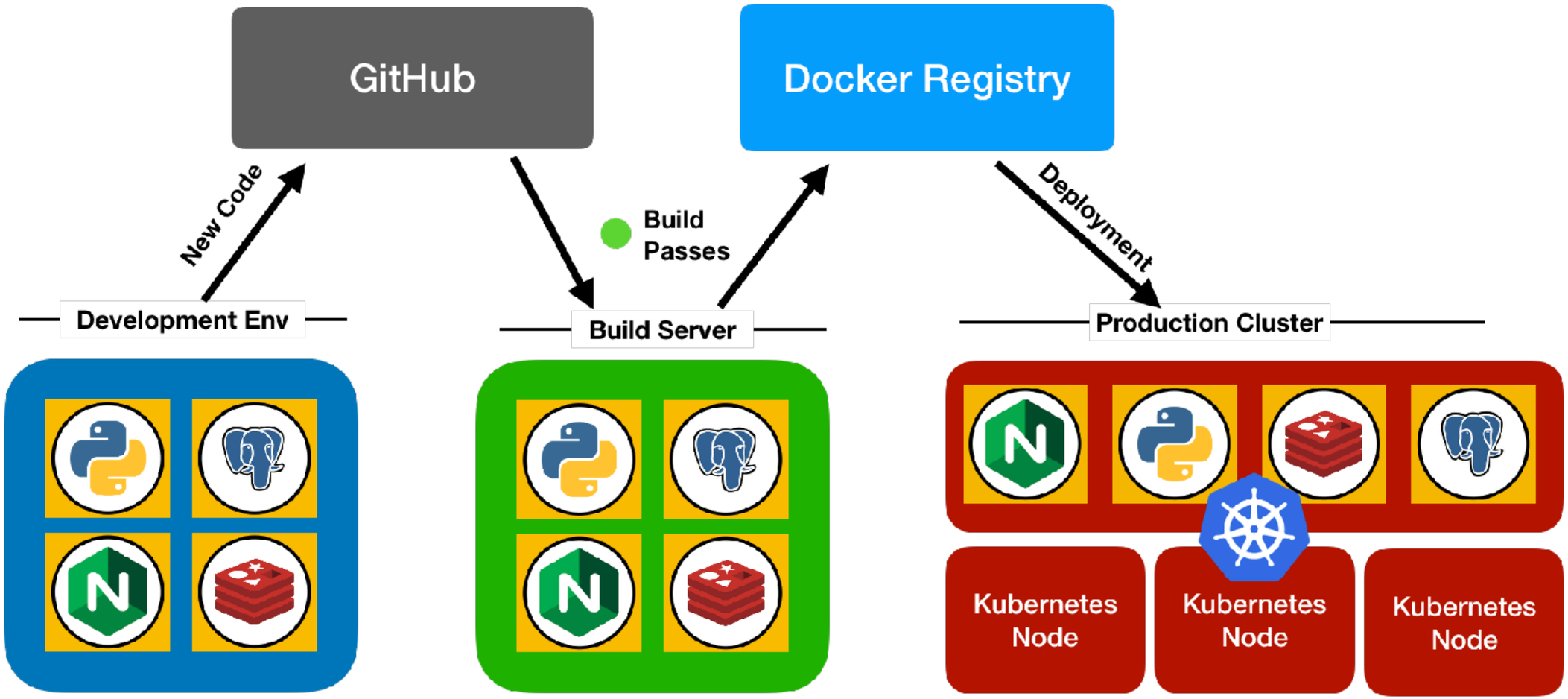
Containerized Development Workflow



Containerized Development Workflow

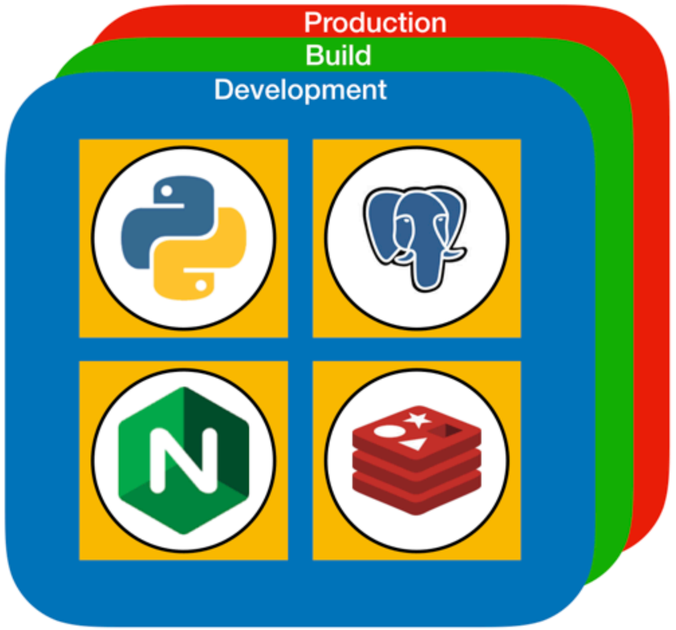


Containerized Development Workflow



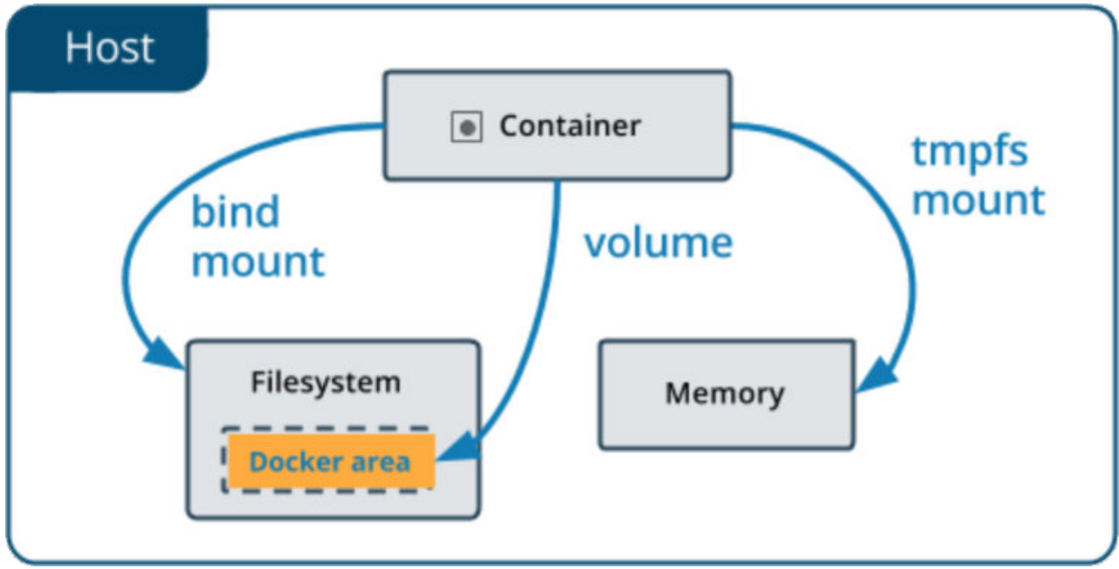
This is a Very Powerful Concept

- 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



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

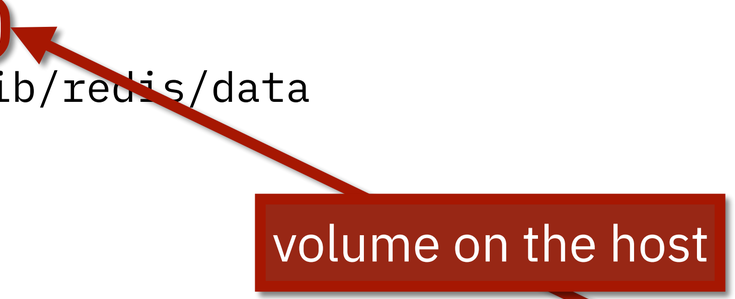


Bind Mount Volumes for Redis

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

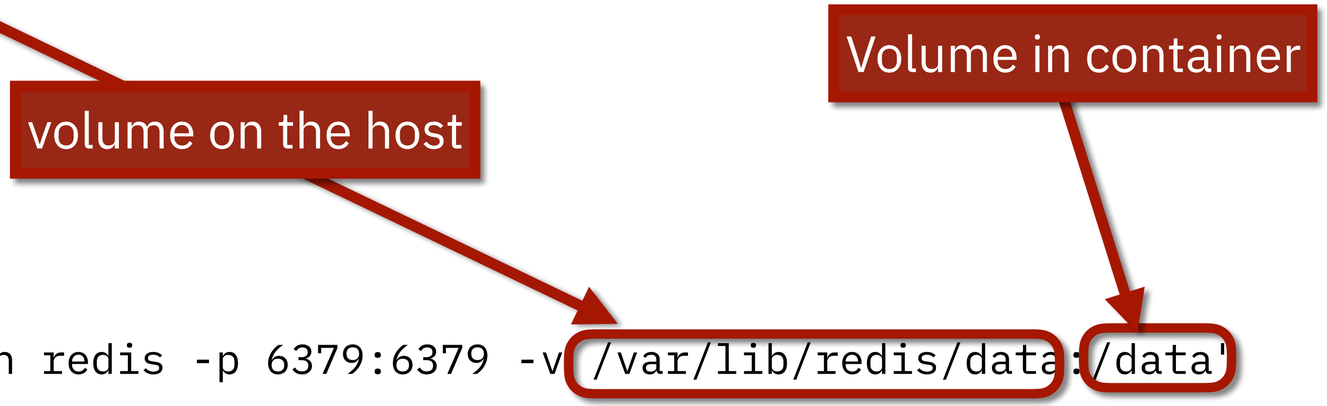
Bind Mount Volumes for Redis

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



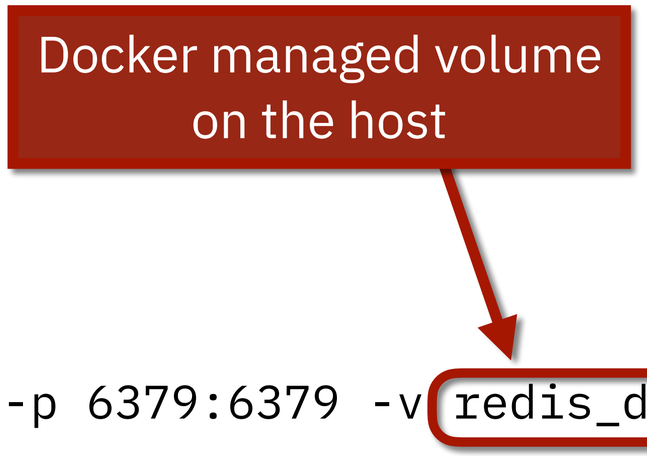
Using a Docker Volume for Redis

- 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

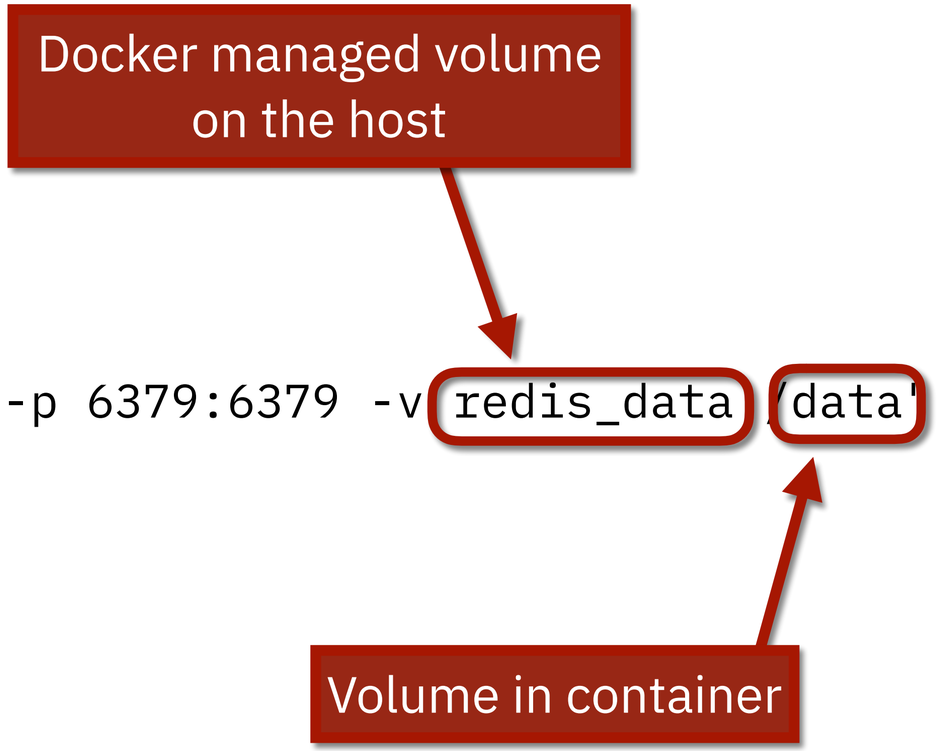



```
# Add Redis docker container
config.vm.provision "docker" do |d|
  d.pull_images "redis:alpine"
  d.run "redis:alpine",
    args: "--restart=always -d --name redis -h redis -p 6379:6379 -v redis_data /data"
end
```

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 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!)

Hands-on Session 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

"live session"

Test the Installation

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

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world

c04b14da8d14: Pull complete
Digest: sha256:0256e8a36e2070f7bf2d0b0763dbabdd67798512411de4cdcf9431a1feb60fd9
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.
 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 Hub account:
 https://hub.docker.com

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

Test the Installation

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

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world

c04b14da8d14: Pull complete
Digest: sha256:0256e8a36e2070f7bf2d0b0763dbabdd67798512411de4cdcf9431a1feb60fd9
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.
 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 Hub account:
 https://hub.docker.com

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

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

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
postgres            alpine              69aad31ddc1c        7 days ago          71.9MB
alpine              latest              4d90542f0623        13 days ago         5.58MB
python              3.7-slim            338ae06dfca5        3 weeks ago         143MB
hello-world         latest              fce289e99eb9        6 months ago        1.84kB
```

Docker Images

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

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
postgres            alpine              69aad31ddc1c        7 days ago          71.9MB
alpine              latest              4d90542f0623        13 days ago         5.58MB
python              3.7-slim            338ae06dfca5        3 weeks ago         143MB
hello-world         latest              fce289e99eb9        6 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

These are from the Vagrantfile

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
postgres            alpine              69aad31ddc1c        7 days ago          71.9MB
alpine              latest              4d90542f0623        13 days ago         5.58MB
python              3.7-slim            338ae06dfca5        3 weeks ago         143MB
hello-world         latest              fce289e99eb9        6 months ago        1.84kB
```

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
e689db2d3f5f   postgres:alpine   "docker-entrypoint.s…"   5 hours ago    Up 5 hours    0.0.0.0:5432->5432/tcp   postgres
```

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
e689db2d3f5f   postgres:alpine   "docker-entrypoint.s…"   5 hours ago    Up 5 hours    0.0.0.0:5432->5432/tcp   postgres
```

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

```
$ docker ps -a
CONTAINER ID   IMAGE             COMMAND                  CREATED         STATUS       PORTS                    NAMES
2f0167b4268d   hello-world       "/hello"                 42 seconds ago  Exited (0) 41 seconds ago             dazzling_ellis
e689db2d3f5f   postgres:alpine   "docker-entrypoint.s…"   5 hours ago     Up 5 hours   0.0.0.0:5432->5432/tcp   postgres
```

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
20e8a1a8bbf4  hello-world      "/hello"                 7 seconds ago  Exited (0) 5 seconds ago                           angry_gates
e689db2d3f5f  postgres:alpine  "docker-entrypoint.s…"   5 hours ago    Up 5 hours                0.0.0.0:5432->5432/tcp   postgres

$ docker rm angry_gates
angry_gates

$ docker ps -a
CONTAINER ID  IMAGE            COMMAND                  CREATED        STATUS            PORTS                    NAMES
e689db2d3f5f  postgres:alpine  "docker-entrypoint.s…"   5 hours ago    Up 5 hours        0.0.0.0:5432->5432/tcp   postgres
(venv) vagrant@devops:/vagrant$
```

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
20e8a1a8bbf4  hello-world      "/hello"                 7 seconds ago   Exited (0) 5 seconds ago                            angry_gates
e689db2d3f5f  postgres:alpine  "docker-entrypoint.s…"   5 hours ago     Up 5 hours                 0.0.0.0:5432->5432/tcp   postgres

$ docker rm angry_gates
angry_gates

$ docker ps -a
CONTAINER ID  IMAGE            COMMAND                  CREATED         STATUS             PORTS                    NAMES
e689db2d3f5f  postgres:alpine  "docker-entrypoint.s…"   5 hours ago     Up 5 hours         0.0.0.0:5432->5432/tcp   postgres
(venv) vagrant@devops:/vagrant$
```

Remove the container named “angry_gates”

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
postgres            alpine              69aad31ddc1c        7 days ago          71.9MB
alpine              latest              4d90542f0623        13 days ago         5.58MB
hello-world         latest              fce289e99eb9        6 months ago        1.84kB

$ docker rmi fce289e99eb9
Untagged: hello-world:latest
Untagged: hello-world@sha256:41a65640635299bab090f783209c1e3a3f11934cf7756b09cb2f1e02147c6ed8
Deleted: sha256:fce289e99eb9bca977dae136fbe2a82b6b7d4c372474c9235adc1741675f587e
Deleted: sha256:af0b15c8625bb1938f1d7b17081031f649fd14e6b233688eea3c5483994a66a3

$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
postgres            alpine              69aad31ddc1c        7 days ago          71.9MB
alpine              latest              4d90542f0623        13 days ago         5.58MB
```

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
postgres            alpine              69aad31ddc1c        7 days ago          71.9MB
alpine              latest              4d90542f0623        13 days ago         5.58MB
hello-world         latest              fce289e99eb9        6 months ago        1.84kB

$ docker rmi fce289e99eb9
Untagged: hello-world:latest
Untagged: hello-world@sha256:41a6564063529[illegible]
Deleted: sha256:fce289e99eb9bca977dae136fbe2a82b6b7d4c372474c9235adc1741675f587e
Deleted: sha256:af0b15c8625bb1938f1d7b17081031f649fd14e6b233688eea3c5483994a66a3

$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
postgres            alpine              69aad31ddc1c        7 days ago          71.9MB
alpine              latest              4d90542f0623        13 days ago         5.58MB
```

Remove the image with id "fce289e99eb9"

Creating Containers from Images

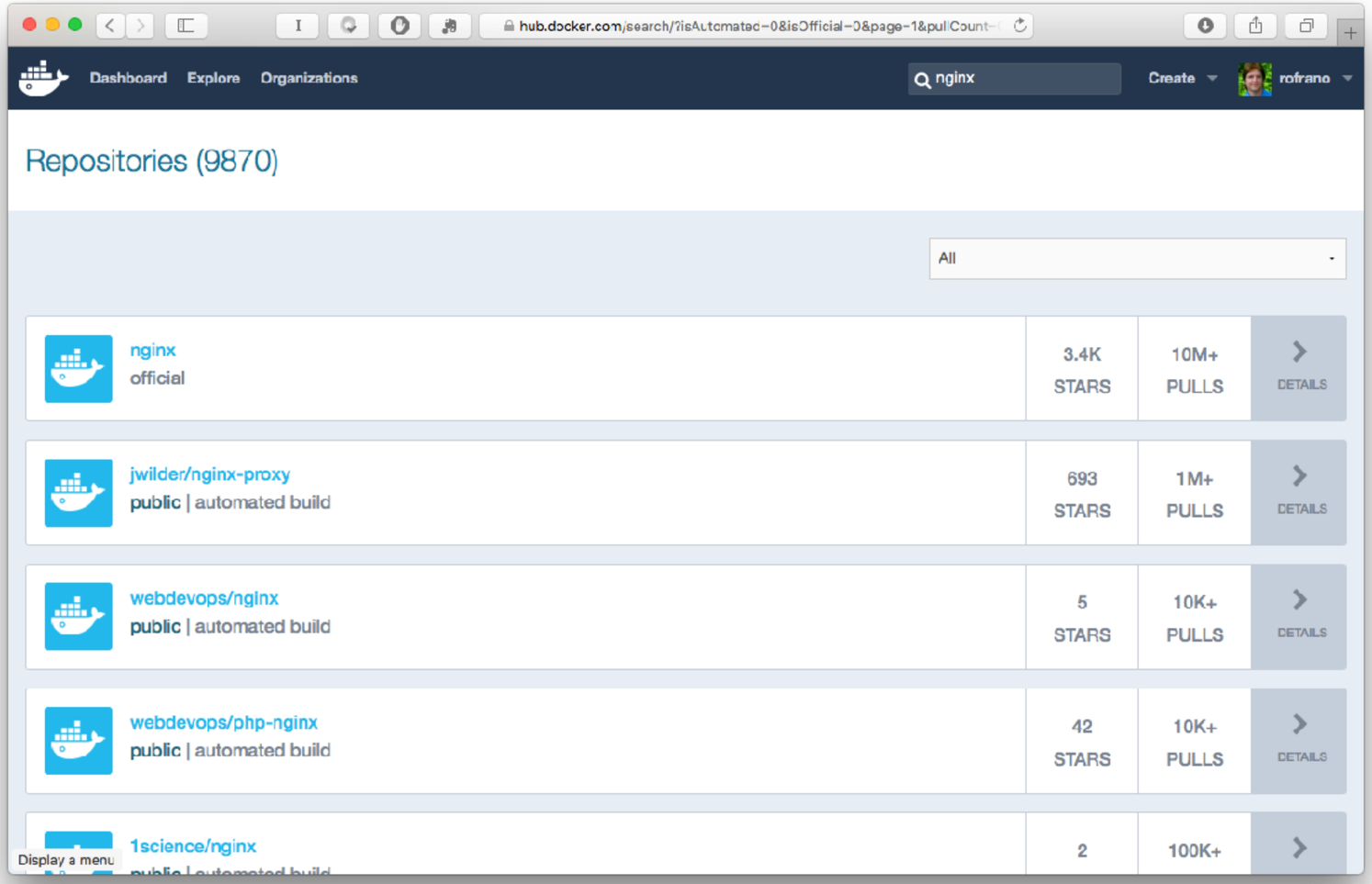
Lets Use an Existing Docker Image

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

Where To Get Images?

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

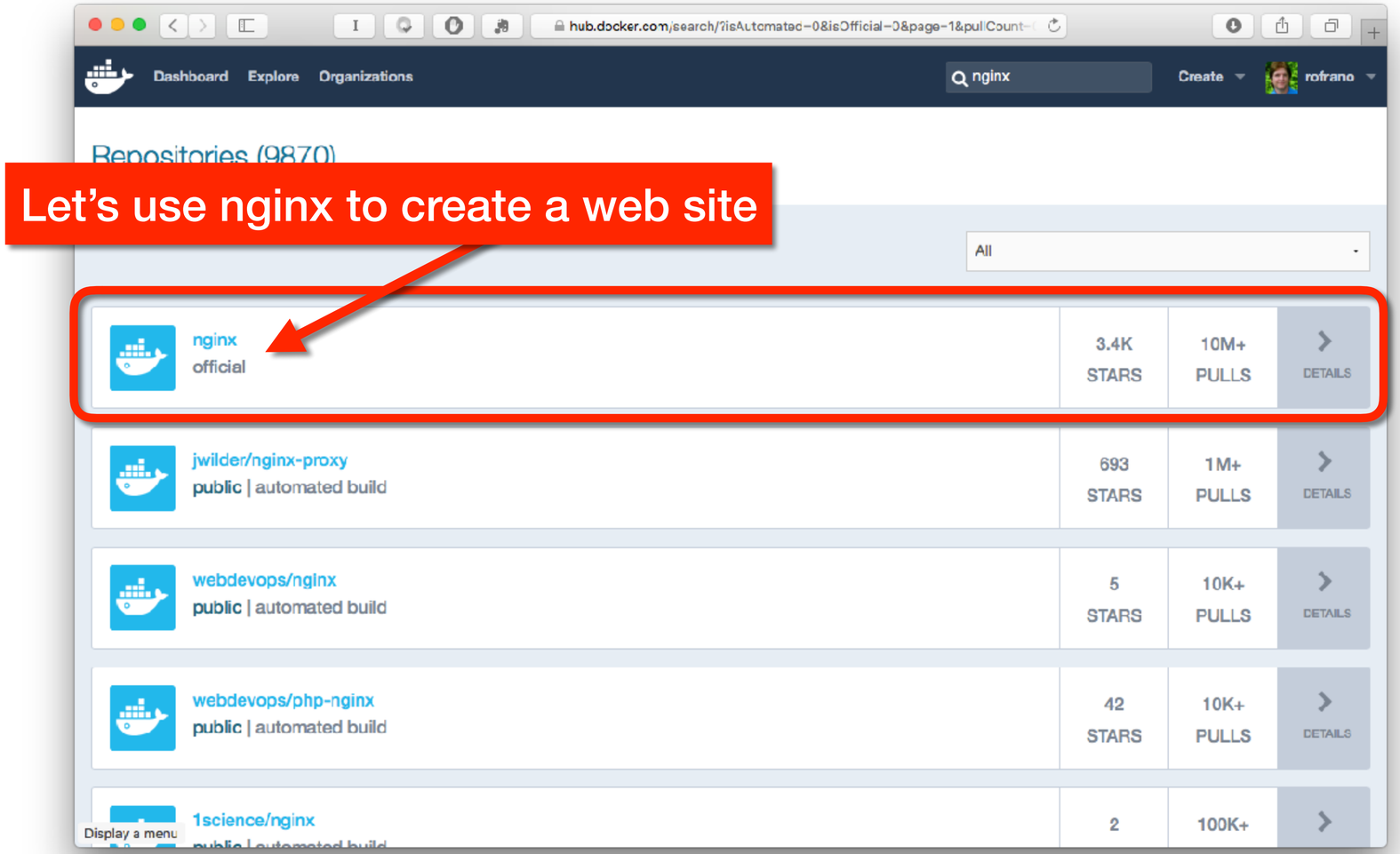
- **Docker Hub**
- Contains 1000's of Docker images with almost every software you can imagine
- Usually "official" images come from the creator of the software
- Most have documentation on how to best use them



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:

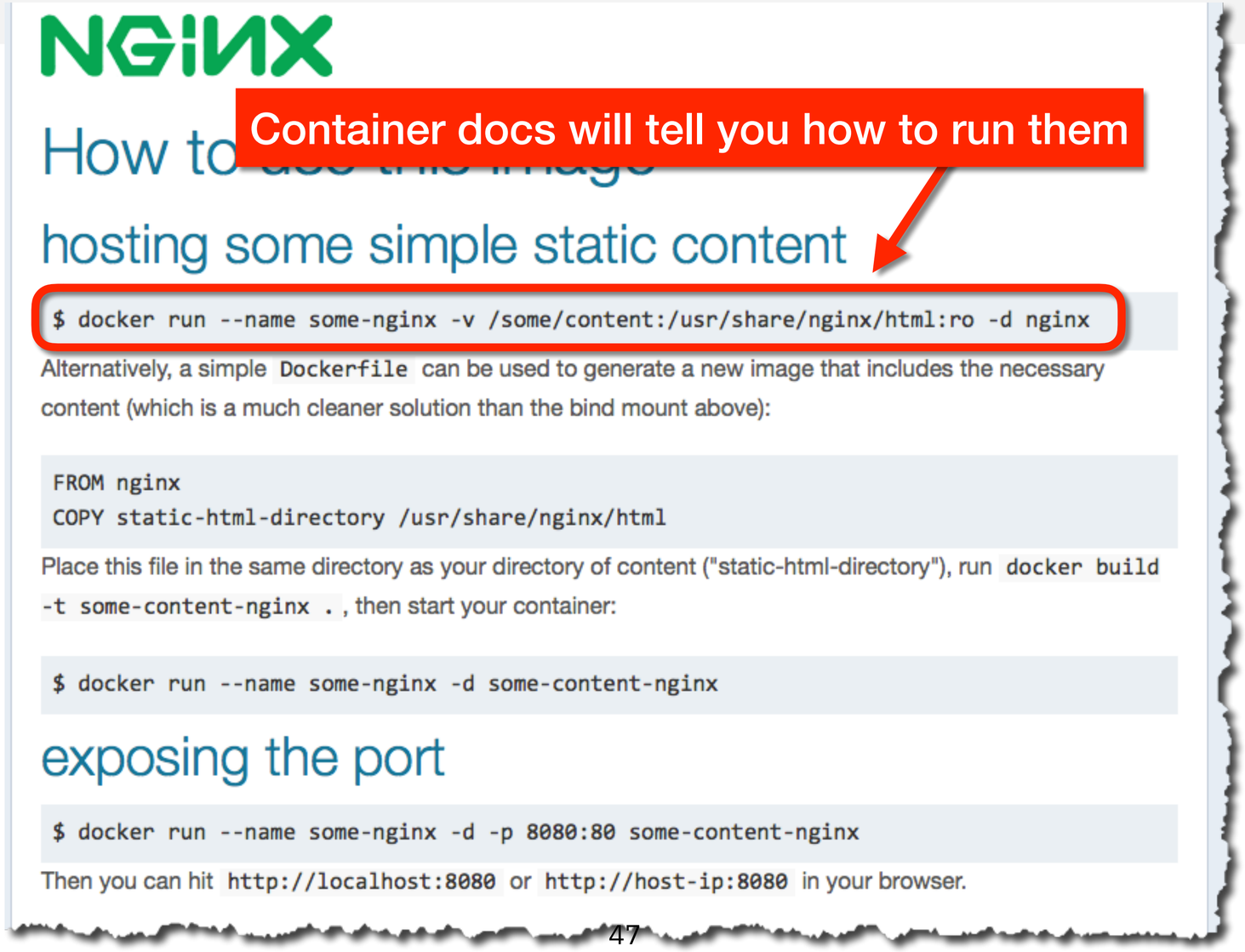
```
$ docker run --name some-nginx -d some-content-nginx
```

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 80:80 nginx:alpine
```

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

Let’s Run Nginx

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

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

Run as a Daemon

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

Let’s Run Nginx

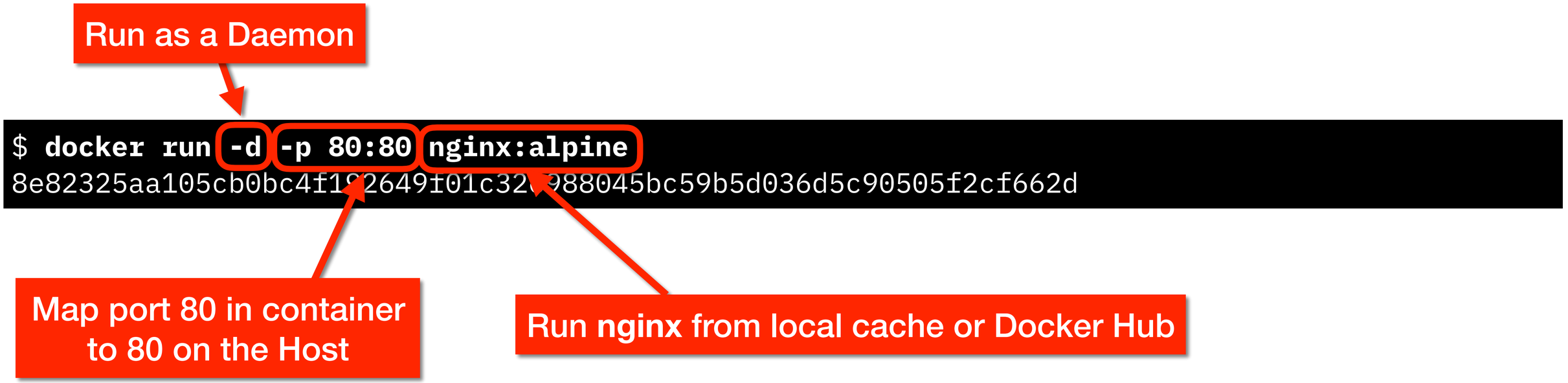
- We can run nginx as a web server with the command:

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

Let’s Run Nginx

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

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

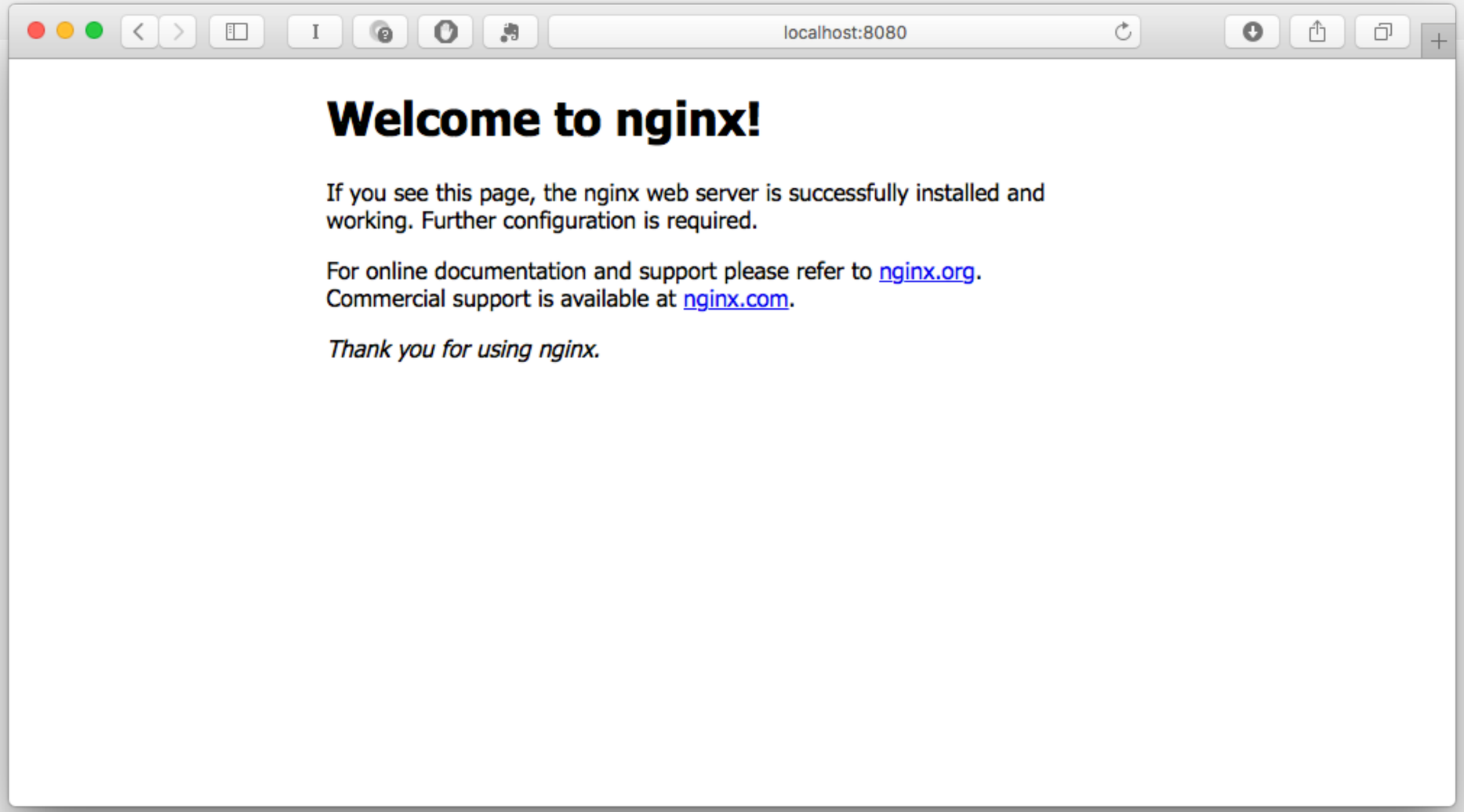


NGINX will be Pulled

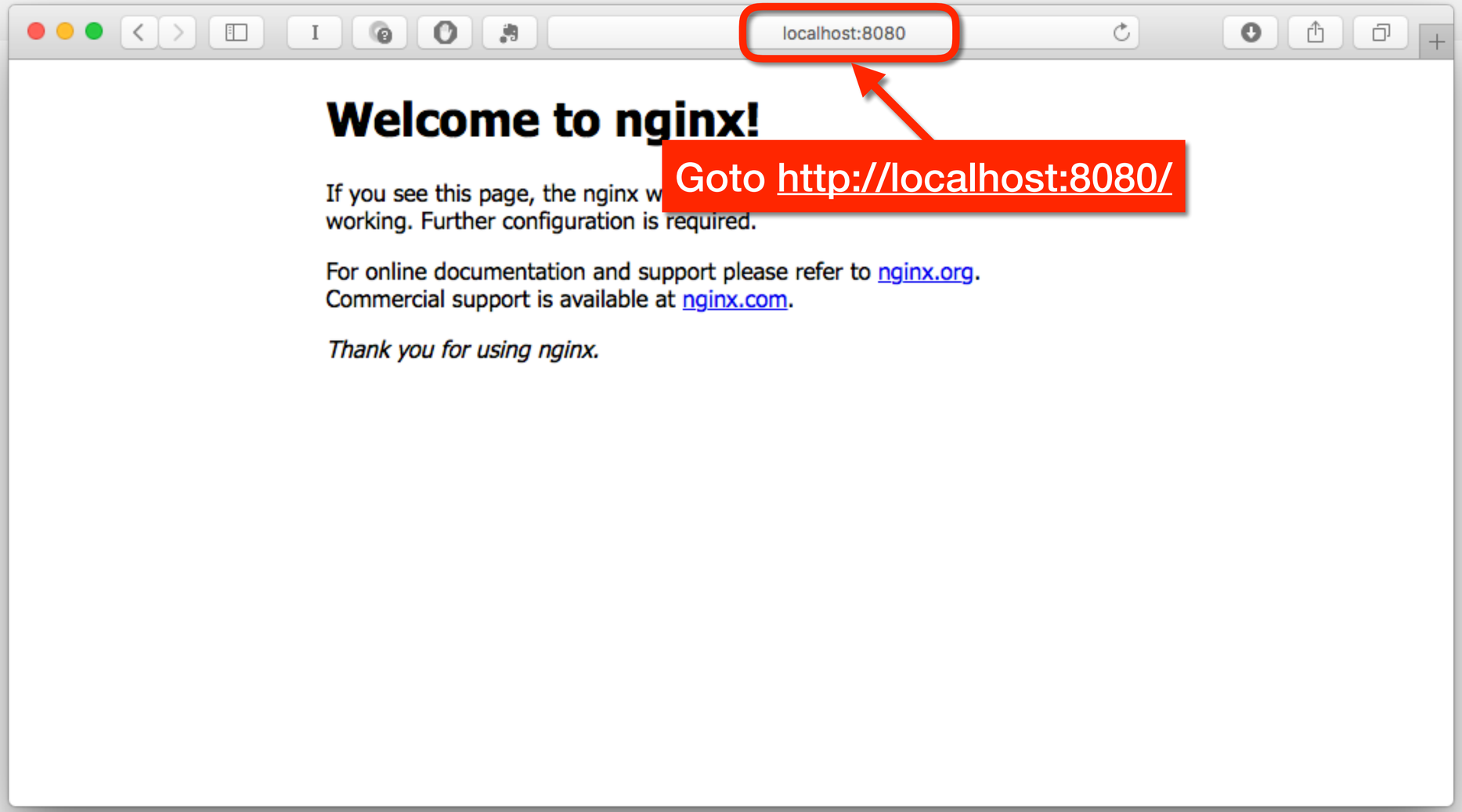
- Since we don’t have a local nginx:alpine image it will be pulled from Docker hub the first time

```
$ docker run -d -p 80:80 nginx:alpine
Unable to find image 'nginx:alpine' locally
alpine: Pulling from library/nginx
e7c96db7181b: Pull complete
3fb6217217ef: Pull complete
Digest: sha256:17bd1698318e9c0f9ba2c5ed49f53d690684dab7fe3e8019b855c352528d57be
Status: Downloaded newer image for nginx:alpine
895a21071f6b08633ad14fcfd0f450ae4a2d0107c436796fba6273f004fd34ae
```

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         "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         "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”

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          "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

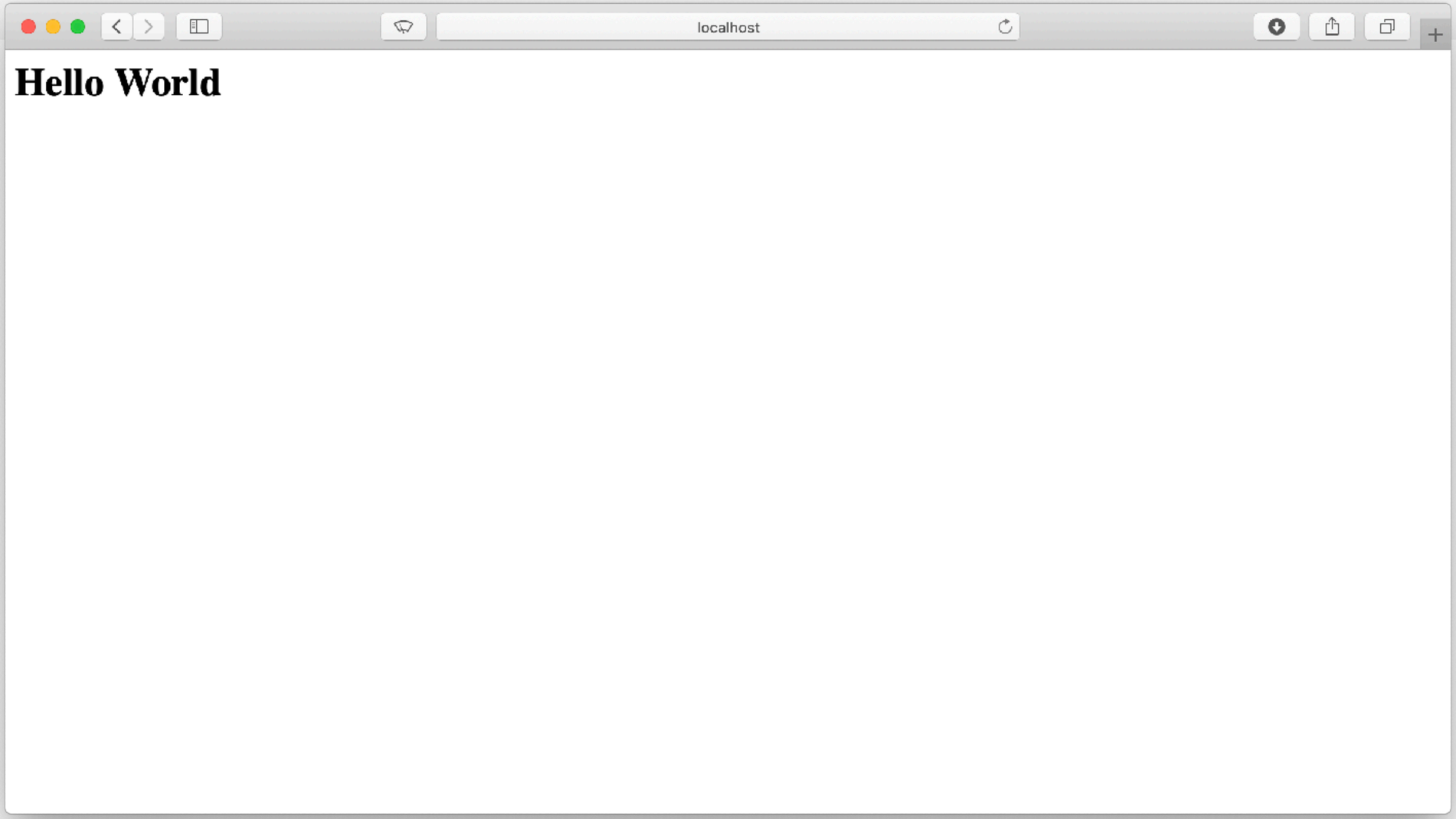
Step 1: Add some content to `/home/vagrant/html`

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

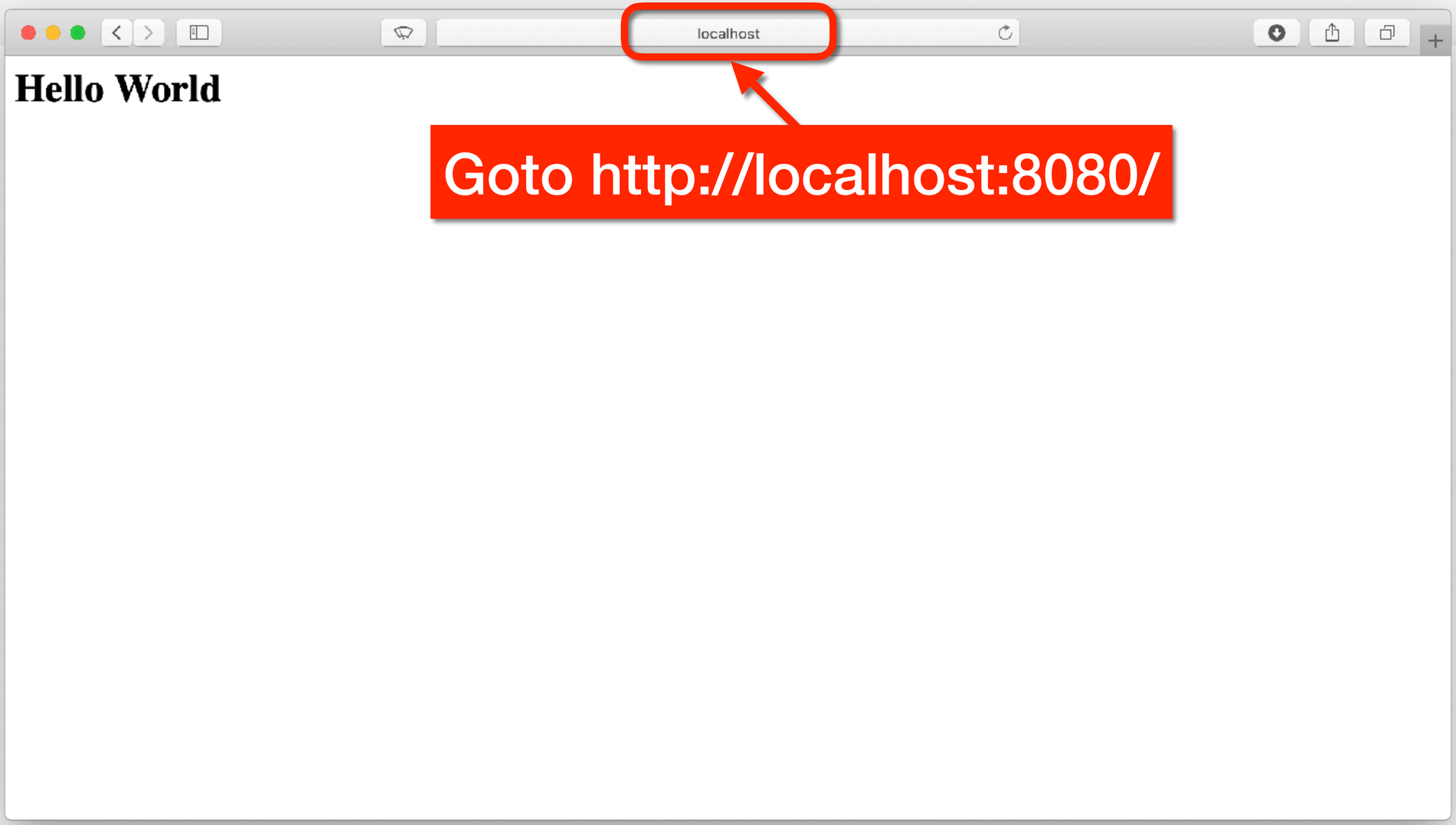
Step 2: Map the html folder with the `-v` (volume) option

```
docker run -d -p 80:80 \
-v /home/vagrant/html:/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...

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 html /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 html /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:

DevOps Workshop Dockerfile

```
FROM python:3.7-slim

# Expose any ports the app requires
ENV PORT 5000
EXPOSE $PORT

# Create working folder
WORKDIR /app
COPY requirements.txt /app
RUN pip install -r requirements.txt

COPY . /app/

ENV GUNICORN_BIND 0.0.0.0:$PORT
CMD ["gunicorn", "service:app"]
```

Build the Dockerfile

```
$ cd /vagrant/
$ docker build -t pet-demo:v1 .
Sending build context to Docker daemon  103.4kB
Step 1/9 : FROM python:3.7-slim
 ---> 338ae06dfca5
Step 2/9 : ENV PORT 5000
 ---> Running in 437e5b5f4993
Removing intermediate container 437e5b5f4993
 ---> fddafd2255b9
Step 3/9 : EXPOSE $PORT
 ---> Running in 257289b3ea35
Removing intermediate container 257289b3ea35
 ---> 77e411be3885
Step 4/9 : WORKDIR /app
 ---> Running in bbf58071e684
Removing intermediate container bbf58071e684
 ---> be13716a1487
Step 5/9 : COPY requirements.txt /app
 ---> aabbaacc1abf
Step 6/9 : RUN pip install -r requirements.txt
 ---> Running in 217722d6f2b7
Collecting Flask==1.0.3 (from -r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/9a/
74/670ae9737d14114753b8c8fdf2e8bd212a05d3b361ab15b44937dfd40985/Flask-1.0.3-py2.py3-none-any.whl (92kB)

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

Removing intermediate container 217722d6f2b7
 ---> 5060311efe21
Step 7/9 : COPY . /app/
 ---> abf15fc79394
Step 8/9 : ENV GUNICORN_BIND 0.0.0.0:$PORT
 ---> Running in c8b6572041ec
Removing intermediate container c8b6572041ec
 ---> ae6ff14f7dc0
Step 9/9 : CMD ["gunicorn", "service:app"]
 ---> Running in ec3285dd6980
Removing intermediate container ec3285dd6980
 ---> c98491c1923b
Successfully built c98491c1923b
Successfully tagged pet-demo:v1
```

Check Your Images

- `docker images`

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
pet-demo            v1                  c98491c1923b        3 minutes ago       213MB
nginx               alpine              ea1193fd3dde        2 days ago          20.6MB
postgres            alpine              69aad31ddc1c        8 days ago          71.9MB
alpine              latest              4d90542f0623        2 weeks ago         5.58MB
python              3.7-slim            338ae06dfca5        3 weeks ago         143MB
$
```

Check Your Images

- `docker images`

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
pet-demo            v1                  c98491c1923b        3 minutes ago       213MB
nginx               alpine              ea1193id3dde        2 days ago          20.6MB
postgres            alpine              69aad31ddc1c        8 days ago          71.9MB
alpine              latest              4d90542f0623        2 weeks ago         5.58MB
python              3.7-slim            338ae06dfca5        3 weeks ago         143MB
$
```

Run the Container

```
docker run --rm -p 5000:5000 pet-demo:v1
```

```
$ docker run --rm -p 5000:5000 pet-demo:v1
[2019-07-04 15:29:42 +0000] [1] [INFO] Starting gunicorn 19.9.0
[2019-07-04 15:29:42 +0000] [1] [INFO] Listening at: http://0.0.0.0:5000 (1)
[2019-07-04 15:29:42 +0000] [1] [INFO] Using worker: sync
[2019-07-04 15:29:42 +0000] [8] [INFO] Booting worker with pid: 8
Setting up logging...
[2019-07-04 15:29:42,227] INFO in routes: Logging handler established
[2019-07-04 15:29:42,227] INFO in __init__: ************************************************************
[2019-07-04 15:29:42,227] INFO in __init__: **************  P E T   S E R V I C E   R U N N I N G  **************
[2019-07-04 15:29:42,227] INFO in __init__: ************************************************************
[2019-07-04 15:29:42,227] INFO in models: Initializing database
[2019-07-04 15:29:42,253] CRITICAL in __init__: (psycopg2.OperationalError) could not connect to server: Connection refused
    Is the server running on host "localhost" (127.0.0.1) and accepting
    TCP/IP connections on port 5432?
could not connect to server: Cannot assign requested address
    Is the server running on host "localhost" (::1) and accepting
    TCP/IP connections on port 5432?

(Background on this error at: http://sqlalche.me/e/e3q8): Cannot continue
[2019-07-04 15:29:42 +0000] [8] [INFO] Worker exiting (pid: 8)
[2019-07-04 15:29:42 +0000] [1] [INFO] Shutting down: Master
[2019-07-04 15:29:42 +0000] [1] [INFO] Reason: App failed to load.
```

Run the Container

`docker run --rm -p 5000:5000 pet-demo:v1`

```
$ docker run --rm -p 5000:5000 pet-demo:v1
[2019-07-04 15:29:42 +0000] [1] [INFO] Starting gunicorn 19.9.0
[2019-07-04 15:29:42 +0000] [1] [INFO] Listening at: http://0.0.0.0:5000 (1)
[2019-07-04 15:29:42 +0000] [1] [INFO] Using worker: sync
[2019-07-04 15:29:42 +0000] [8] [INFO] Booting worker with pid: 8
Setting up logging...
[2019-07-04 15:29:42,227] INFO in routes: Logging handler established
[2019-07-04 15:29:42,227] INFO in __init__: ************************************************************
[2019-07-04 15:29:42,227] INFO in __init__: **************  P E T   S E R V I C E   R U N N I N G  **************
[2019-07-04 15:29:42,227] INFO in __init__: ************************************************************
[2019-07-04 15:29:42,227] INFO in models: Initializing database
[2019-07-04 15:29:42,253] CRITICAL in __init__: (psycopg2.OperationalError) could not connect to server: Connection refused
    Is the server running on host "localhost" (127.0.0.1) and accepting
    TCP/IP connections on port 5432?
could not connect to server: Cannot assign requested address
    Is the server running on host "localhost" (::1) and accepting
    TCP/IP connections on port 5432?

(Background on this error at: http://sqlalche.me/e/e3q8): Cannot continue
[2019-07-04 15:29:42 +0000] [8] [INFO] Worker exiting (pid: 8)
[2019-07-04 15:29:42 +0000] [1] [INFO] Shutting down: Master
[2019-07-04 15:29:42 +0000] [1] [INFO] Reason: App failed to load.
```

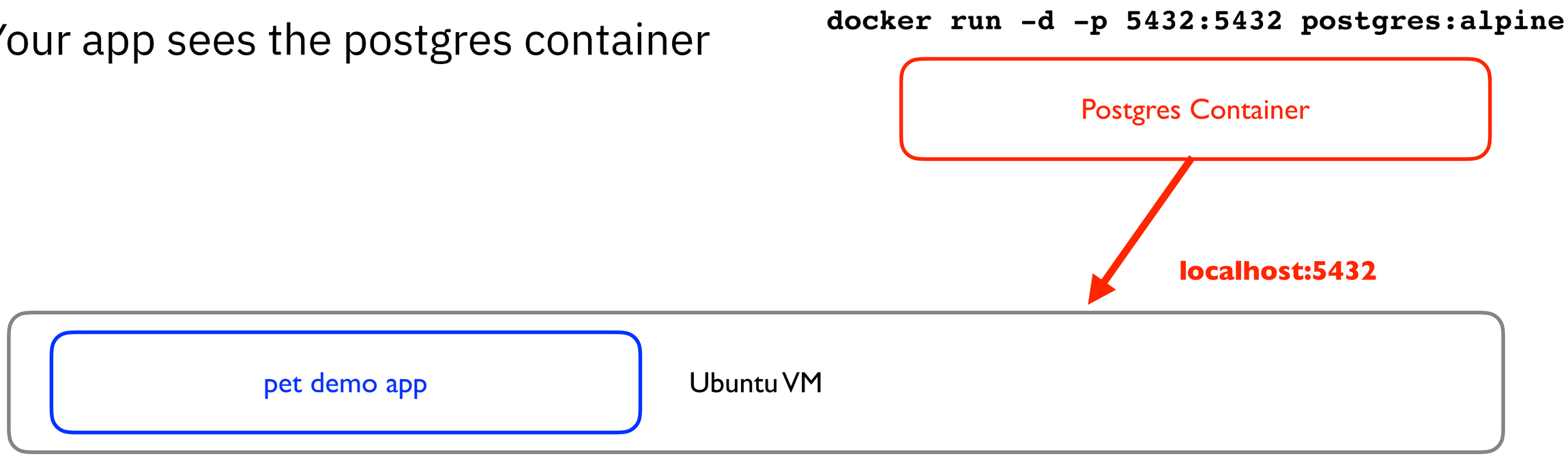
Critical Failure !!!

Where is Postgres?

- Postgres is running in a container which responds to 127.0.0.1:5432 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 Postgres
- We must link the containers together

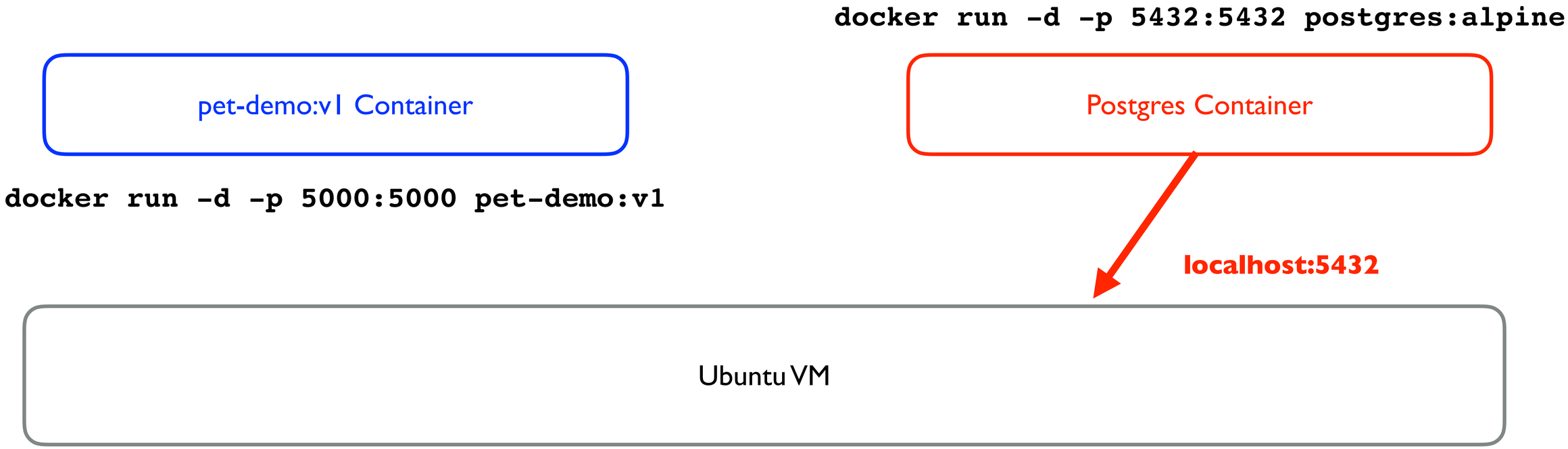
Running in a VM

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



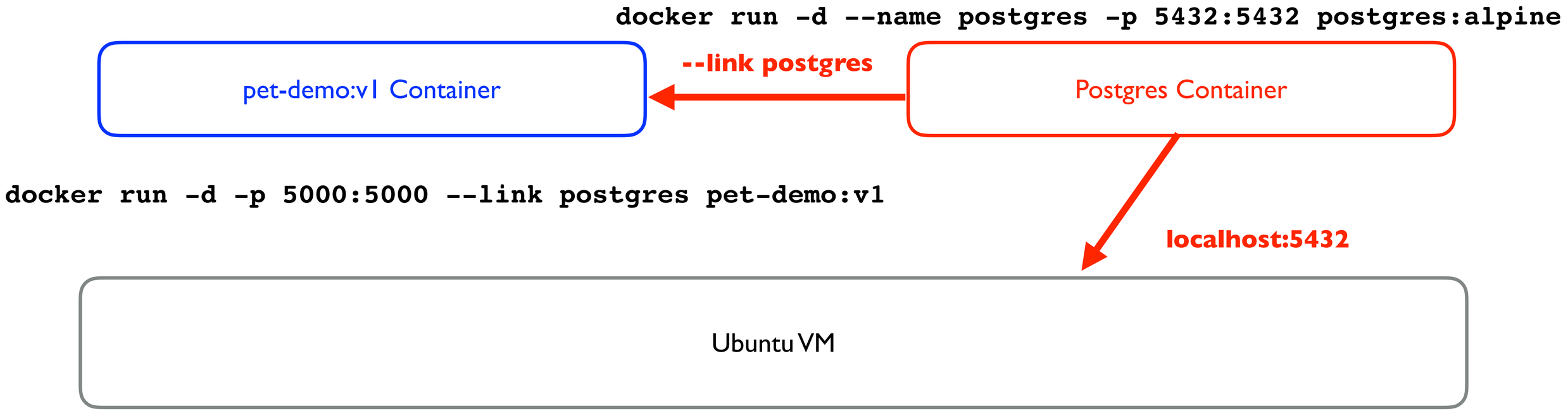
Running in a Container

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



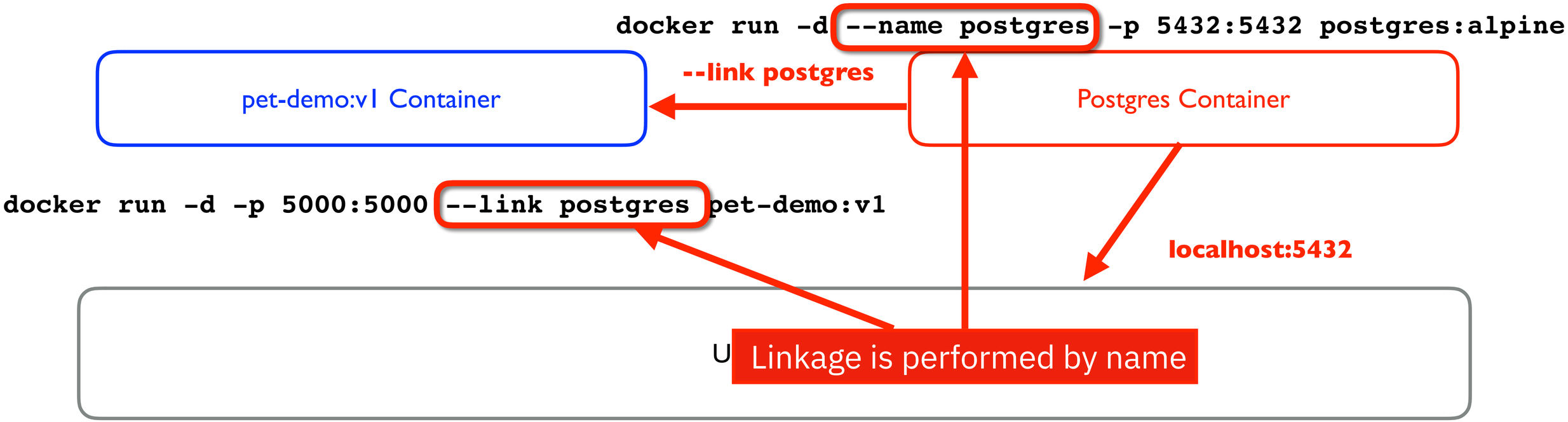
--link is the Solution

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



--link is the Solution

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



Linking Containers

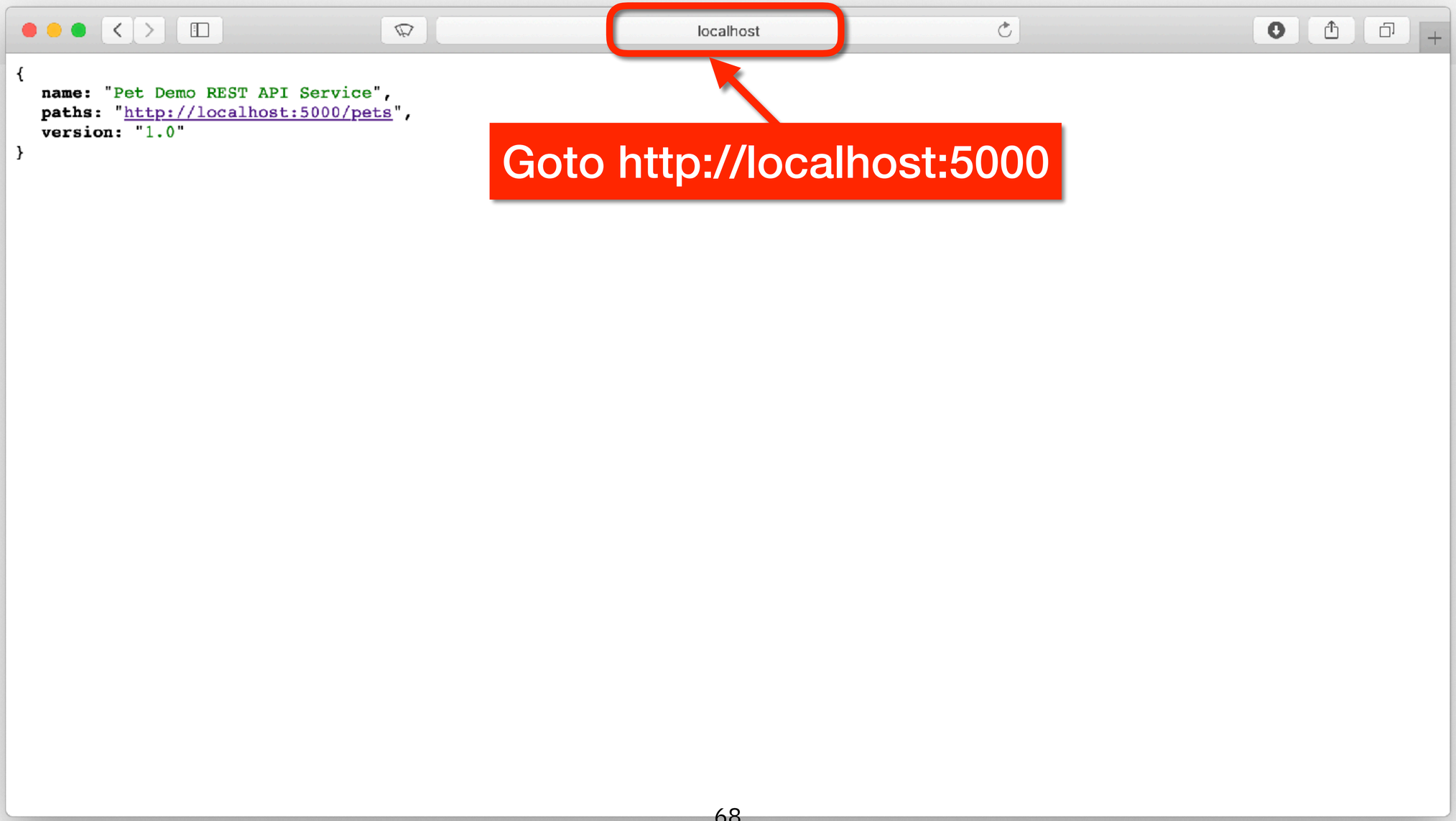
- We can use the `--link` command to tell pet-demo where postgres is but it will still look at localhost until we change the DATABASE_URI environment variable
- We can also use `-e` to pass in a new DATABASE_URI environment variable

```
$ docker run --rm --link postgres -p 5000:5000 \

-e DATABASE_URI="postgres://postgres:postgres@postgres:5432/postgres" \

pet-demo:v1
```

Success !!! 😀

```
$ docker run --rm --link postgres \
> -p 5000:5000 \
> -e DATABASE_URI="postgres://postgres:postgres@postgres:5432/postgres" \
> pet-demo:v1

[2019-07-04 15:50:40 +0000] [1] [INFO] Starting gunicorn 19.9.0
[2019-07-04 15:50:40 +0000] [1] [INFO] Listening at: http://0.0.0.0:5000 (1)
[2019-07-04 15:50:40 +0000] [1] [INFO] Using worker: sync
[2019-07-04 15:50:40 +0000] [8] [INFO] Booting worker with pid: 8
Setting up logging...
[2019-07-04 15:50:41,108] INFO in routes: Logging handler established
[2019-07-04 15:50:41,108] INFO in __init__: ************************************************************
[2019-07-04 15:50:41,109] INFO in __init__: **********  P E T   S E R V I C E   R U N N I N G  **********
[2019-07-04 15:50:41,109] INFO in __init__: ************************************************************
[2019-07-04 15:50:41,109] INFO in models: Initializing database
[2019-07-04 15:50:41,141] INFO in __init__: Service inititalized!
```

Try Again


localhost
{
name: "Pet Demo REST API Service",
paths: "http://localhost:5000/pets",
version: "1.0"
}
Goto http://localhost:5000

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:
  pet_demo:
    build: .
    image: pet-demo:v1
    container_name: pet-demo
    ports:
      - 5000:5000
    environment:
      DATABASE_URI: "postgres://admin:s3cr3t@postgres:5432/postgres"
    restart: on-failure
    depends_on:
      - postgres
    networks:
      - overlay
```

Compose for our PostgreSQL

```
 postgres:
    image: postgres:alpine
    hostname: postgres
    container_name: postgres
    restart: always
    ports:
      - 5432:5432
    volumes:
      - pg_data:/var/lib/postgresql/data
    environment:
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: s3cr3t
    networks:
      - overlay

volumes:
  pg_data:

networks:
  overlay:
```

Docker Compose

```
$ docker-compose up -d
Creating network "vagrant_overlay" with the default driver
Creating postgres
Creating pet-demo

$ docker-compose ps
  Name                 Command               State           Ports
-------------------------------------------------------------------------
pet-demo   gunicorn service:app             Up      0.0.0.0:5000->5000/tcp
postgres   docker-entrypoint.sh postgres    Up      0.0.0.0:5432->5432/tcp

$ docker-compose stop
Stopping pet-demo ... done
Stopping postgres ... done

$ docker-compose down
Removing pet-demo ... done
Removing postgres ... done
Removing network vagrant_overlay
```

Docker Compose management

- docker-compose up
- docker-compose build
- docker-compose start | stop
- docker-compose ps
- docker-compose down
- ...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 pet-demo:v1
IMAGE               CREATED             CREATED BY                                      SIZE                COMMENT
fc9eae04c870        About an hour ago   /bin/sh -c #(nop)  CMD ["gunicorn" "service:…   0B
548cf4cc8750        About an hour ago   /bin/sh -c #(nop)  ENV GUNICORN_BIND=0.0.0.0…   0B
e4d4c782b67e        About an hour ago   /bin/sh -c #(nop) COPY dir:1af2e5b0e88c2d33a…   78.3kB
5060311efe21        About an hour ago   /bin/sh -c pip install -r requirements.txt      69.3MB
aabbaacc1abf        About an hour ago   /bin/sh -c #(nop) COPY file:f5c0656b859b8cf7…   282B
be13716a1487        About an hour ago   /bin/sh -c #(nop) WORKDIR /app                  0B
77e411be3885        About an hour ago   /bin/sh -c #(nop)  EXPOSE 5000                  0B
fddafd2255b9        About an hour ago   /bin/sh -c #(nop)  ENV PORT=5000                0B
338ae06dfca5        3 weeks ago         /bin/sh -c #(nop)  CMD ["python3"]              0B
<missing>           3 weeks ago         /bin/sh -c set -ex;   savedAptMark="$(apt-ma…   7.26MB
<missing>           3 weeks ago         /bin/sh -c #(nop)  ENV PYTHON_PIP_VERSION=19…   0B
<missing>           3 weeks ago         /bin/sh -c cd /usr/local/bin  && ln -s idle3…   32B
<missing>           3 weeks ago         /bin/sh -c set -ex   && savedAptMark="$(apt-…   74.3MB
<missing>           3 weeks ago         /bin/sh -c #(nop)  ENV PYTHON_VERSION=3.7.3     0B
<missing>           3 weeks ago         /bin/sh -c #(nop)  ENV GPG_KEY=0D96DF4D4110E…   0B
<missing>           3 weeks ago         /bin/sh -c apt-get update && apt-get install…   6.48MB
<missing>           3 weeks ago         /bin/sh -c #(nop)  ENV LANG=C.UTF-8             0B
<missing>           3 weeks ago         /bin/sh -c #(nop)  ENV PATH=/usr/local/bin:/…   0B
<missing>           3 weeks ago         /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>           3 weeks ago         /bin/sh -c #(nop) ADD file:5ffb798d64089418e…   55.3MB
```

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

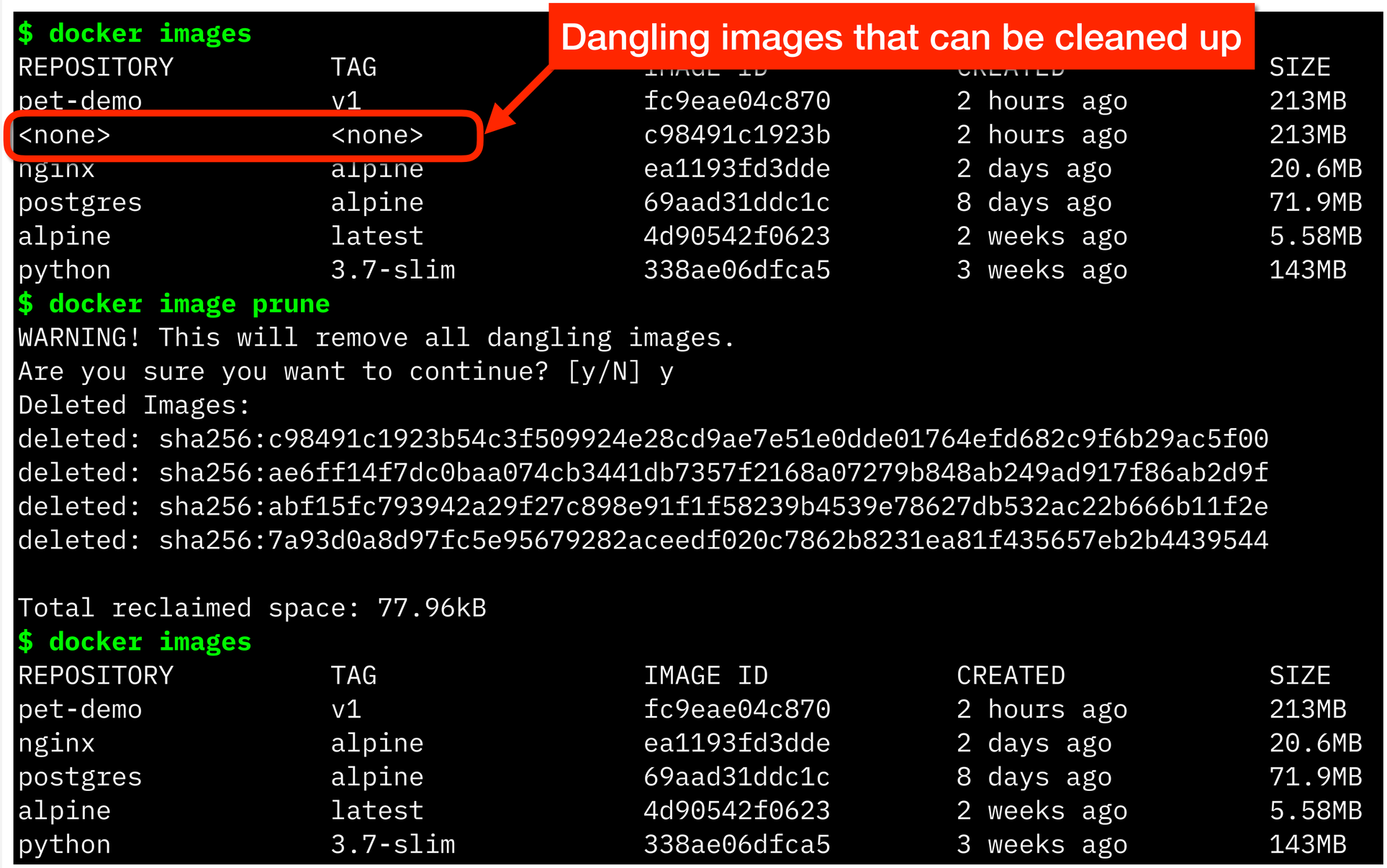
Remove All Dangling Images

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

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
pet-demo            v1                  fc9eae04c870        2 hours ago         213MB
<none>              <none>              c98491c1923b        2 hours ago         213MB
nginx               alpine              ea1193fd3dde        2 days ago          20.6MB
postgres            alpine              69aad31ddc1c        8 days ago          71.9MB
alpine              latest              4d90542f0623        2 weeks ago         5.58MB
python              3.7-slim            338ae06dfca5        3 weeks ago         143MB
$ docker image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N] y
Deleted Images:
deleted: sha256:c98491c1923b54c3f509924e28cd9ae7e51e0dde01764efd682c9f6b29ac5f00
deleted: sha256:ae6ff14f7dc0baa074cb3441db7357f2168a07279b848ab249ad917f86ab2d9f
deleted: sha256:abf15fc793942a29f27c898e91f1f58239b4539e78627db532ac22b666b11f2e
deleted: sha256:7a93d0a8d97fc5e95679282aceedf020c7862b8231ea81f435657eb2b4439544

Total reclaimed space: 77.96kB
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
pet-demo            v1                  fc9eae04c870        2 hours ago         213MB
nginx               alpine              ea1193fd3dde        2 days ago          20.6MB
postgres            alpine              69aad31ddc1c        8 days ago          71.9MB
alpine              latest              4d90542f0623        2 weeks ago         5.58MB
python              3.7-slim            338ae06dfca5        3 weeks ago         143MB
```

Remove All Dangling Images

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



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.