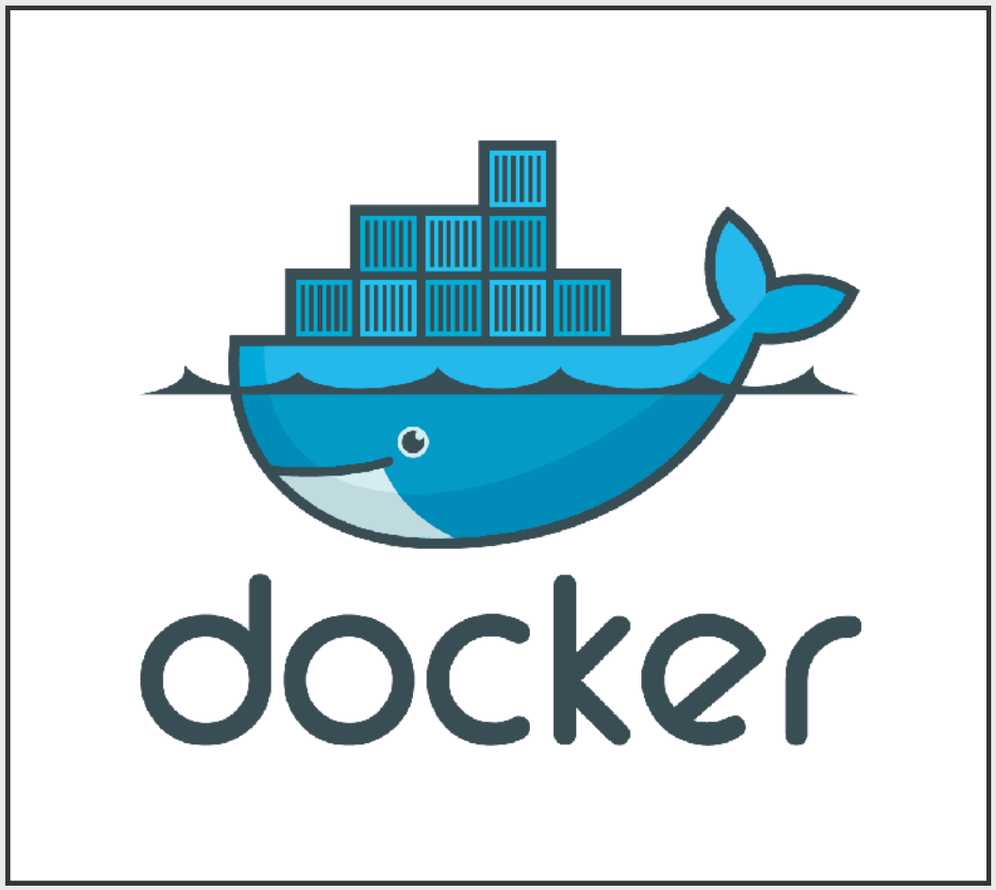
MLH localhost

Workshop

Introduction to Docker

What is Docker?

Docker is a platform that helps developers **build, ship and run any application, in any environment.**



What is Docker, Inc?

Sponsor of the Docker Project

- Primary contributor and maintainer
- 7B+ Image Downloads, 3,000+ Contributors, 700,000 Dockerized Applications

Offers containers as a Service

- End to end platform for IT Team and a Developers using Docker
- Commercial tech support for Docker users
- Over 60% of Docker users are using the platform in production

60%

use Docker, Inc's Containers as a Service in Production

Applications are Changing!

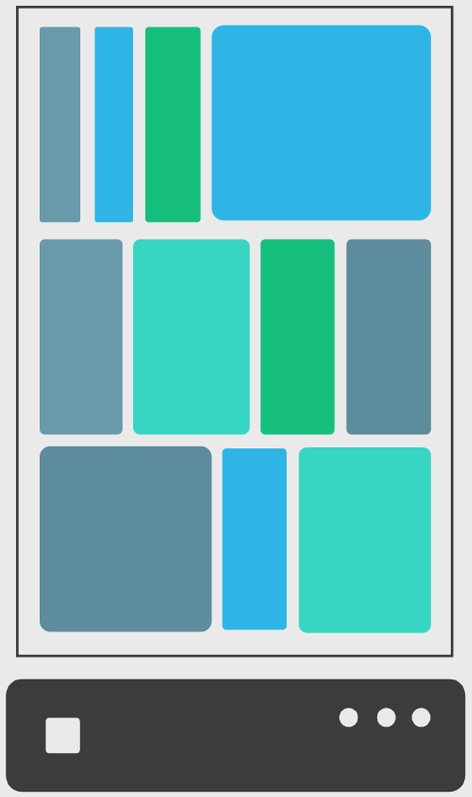
~2000

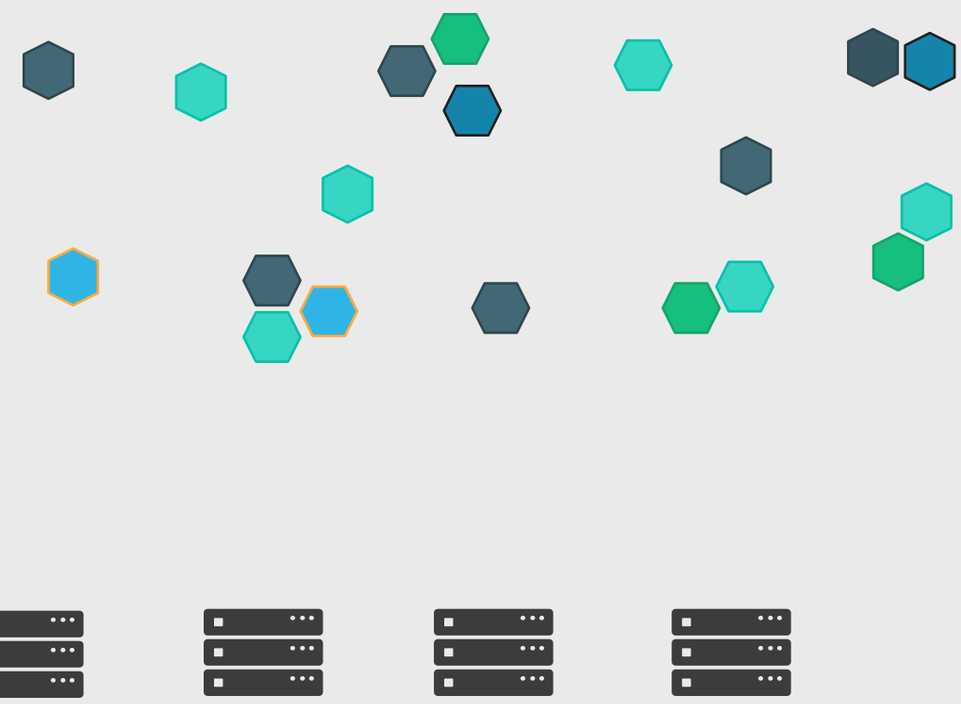
Today

Monolithic

Slow changing

Big Servers





Loosely Coupled Services

Rapidly updated

Many Small Servers

Distributed Applications

Multiplicity of Stacks

Static website

nginx 1.5 + modsecurity + openssl + bootstrap 2

User DB

postgresql + pgv8 + v8

Queue

Redis + redis-sentinel

Analytics DB

hadoop + hive + thrift + OpenJDK

Background workers

Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs

Web frontend

Ruby + Rails + sass + Unicorn

API endpoint

Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client

Do services and apps interact appropriately?

Multiplicity of hardware environments

Development VM

QA server

Customer Data Center

Public Cloud

Production Cluster

Disaster recovery

Contributor's laptop

Production Servers

Can I migrate smoothly and quickly?

docker

Cargo Transport Pre-1960

Multiplicity of Goods



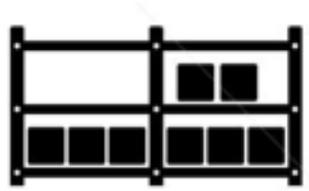




Do I worry about how goods interact (e.g. coffee beans next to spices)

Multipilicity of methods for transporting/storing




Can I transport quickly and smoothly (e.g. from boat to train to truck)

docker

Solution: Intermodal Shipping Container

Docker: A platform for the Software Supply Chain Ecosystem

Static website

User DB

Web frontend

Queue

Analytics DB

Multiplicity of Stacks

Do services and apps interact appropriately?

An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container…

…that can be manipulated using standard operations and run consistently on virtually any hardware platform

Multiplicity of hardware environments

Can I migrate smoothly and quickly

Development VM

QA server

Customer Data Center

Public Cloud

Production Cluster

Contributor's laptop

Docker Containers are NOT VMs!

While they share some characteristics, Virtual Machines and Docker Containers are very different under the hood.

Key Term **Virtual Machine (VM)**

A software computer that, like a physical computer, runs an operating system and applications.

Major similarities:

- Both provide isolated environment for applications to run inside.
- Both are easily movable between different hosts.

Think Houses vs. Apartments

Virtual Machine (VM)

- Each VM has a full copy of the OS with dedicated resources.
- Each new VM creates a full copy of this environment.

Think Houses vs. Apartments

Docker Containers

- Contains exactly what they need to run their application.
- Share underlying resources.

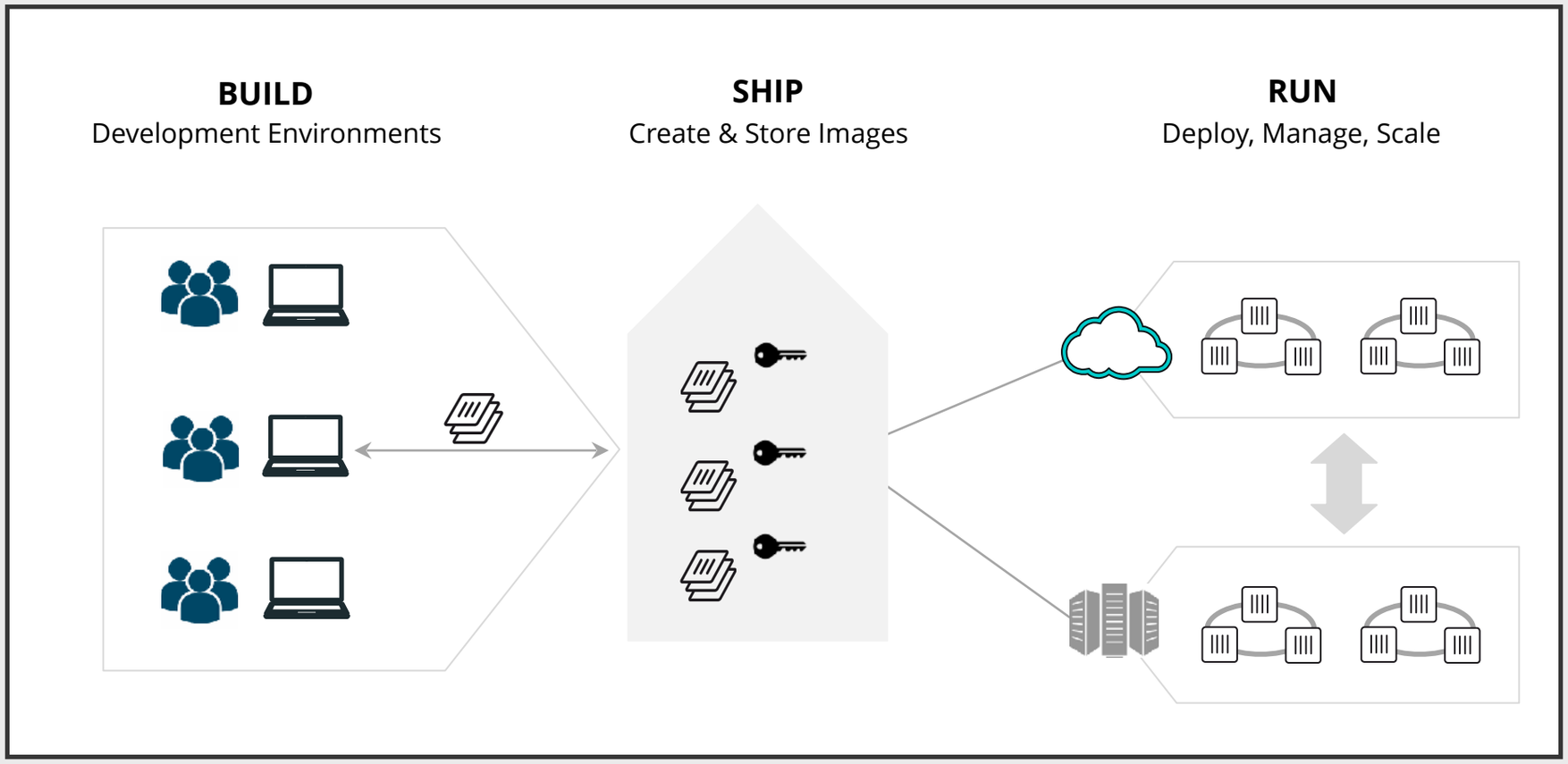
Build, Ship, Run – Any App Anywhere!

An app written on your laptop will **run exactly the same anywhere:**

- Your Friend's Laptop
- Bare Metal Servers
- The Cloud
- A Raspberry Pi

Build, Ship, Run – Any App Anywhere!

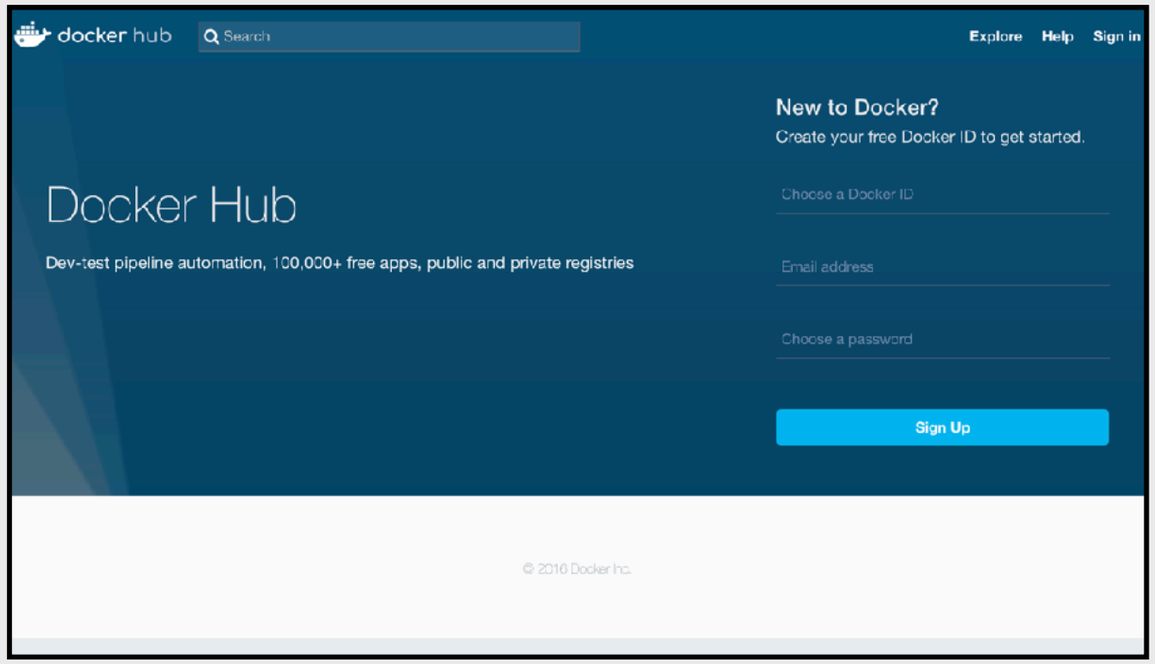
Docker Containers can be easily shared between developers and IT operations. If it runs on your laptop, it will run in production!



Meet Docker Hub!

What is Docker Hub?

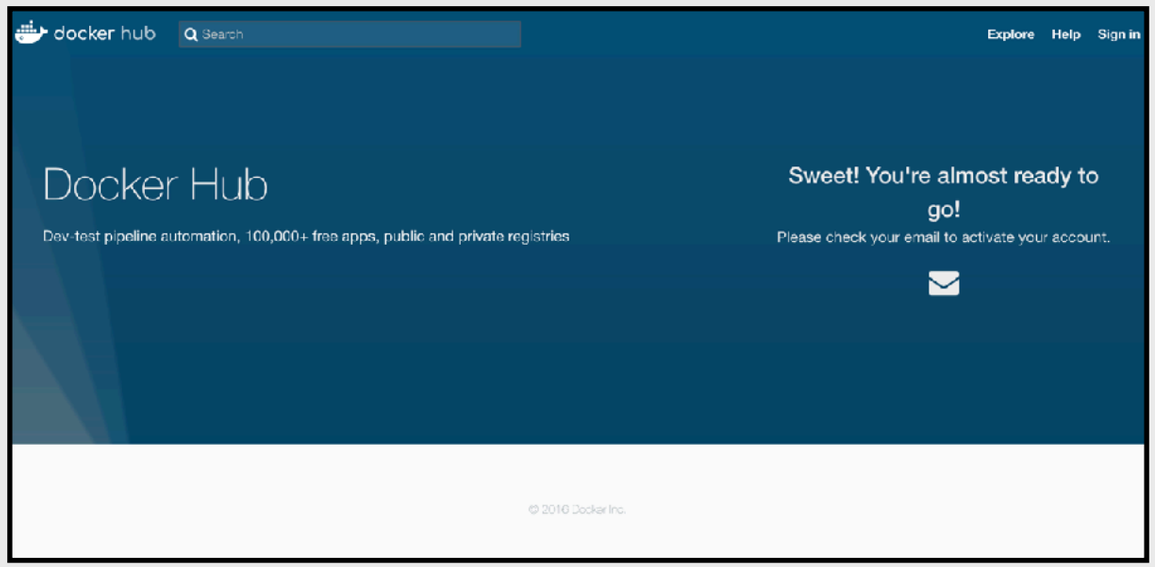
Docker Hub is a registry of Docker images. You can think of the registry as a directory of all available Docker images. You'll be using this later in this tutorial.



Sign up for a Docker ID

Fill out the Form & Verify your Email Address.

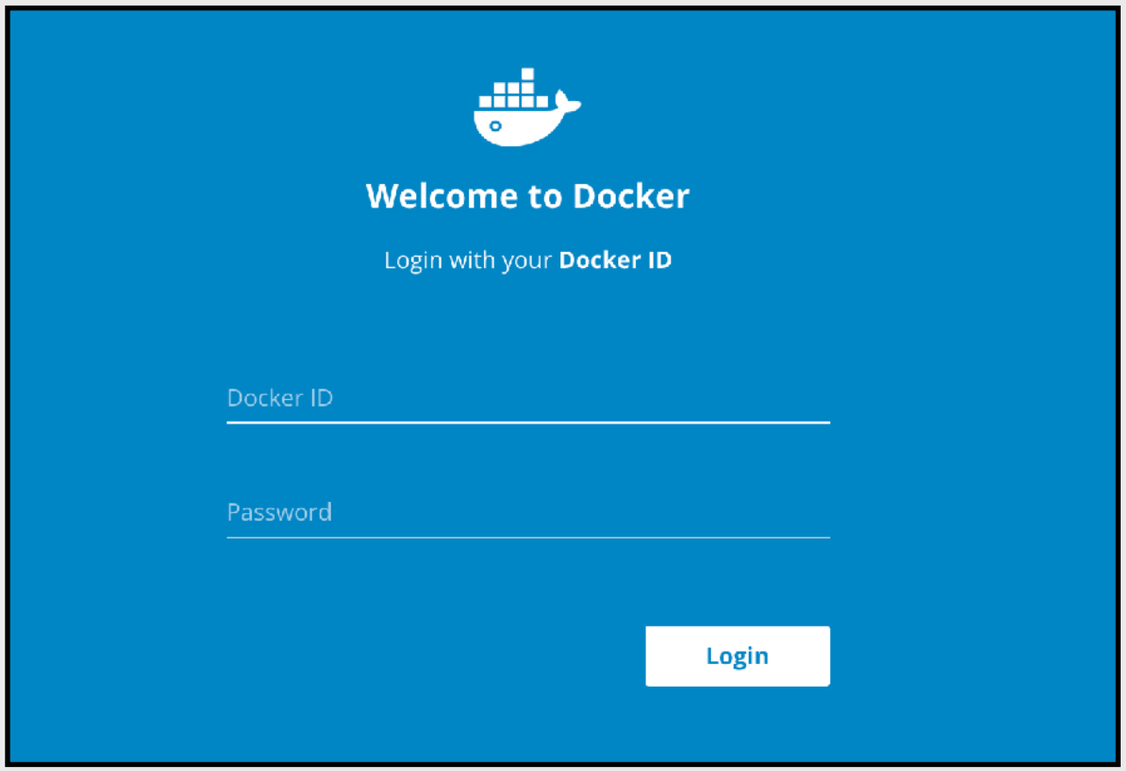
Enter your email and chose a Docker ID and password for your account. You'll be asked to verify your email address before being able to log in though. Do that now!

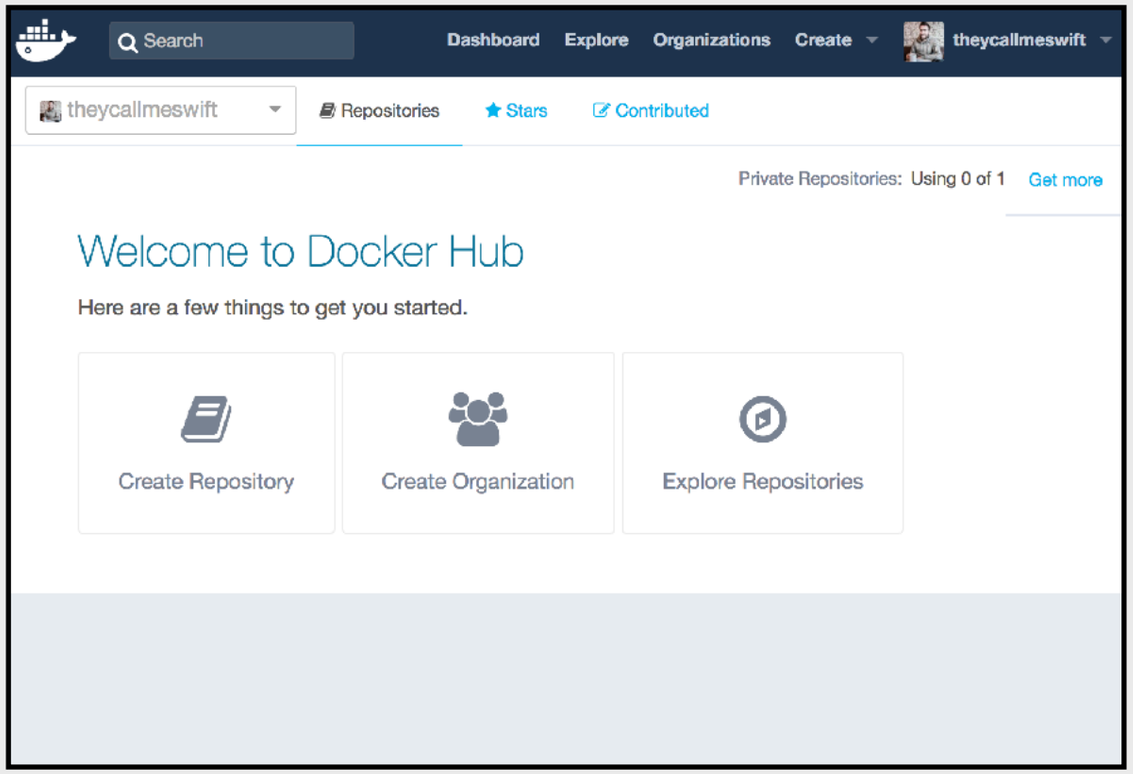


http://mlhlocal.host/docker-hub

Note: *Remember your Docker ID, you'll need it later!*

Login Using your Docker ID





Follow the link you got from Docker Hub's email to verify your account. You can now login using the Docker ID and password you picked!

Test your Installation

Once you are done installing Docker, test your Docker installation by running the following in terminal or PowerShell:

```
$ docker run hello-world

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
…
```

Let’s Run our First Container!

We’re going to run Alpine Linux!

- Alpine Linux is a lightweight Linux distribution.
- We can get a **Docker Image** containing Alpine Linux from **Docker Hub.**

Pull the Alpine Linux Image

The pull command fetches the Alpine Linux image from the Docker registry (**Docker Hub**) and saves it in our system.

```
$ docker pull alpine

Using default tag: latest
latest: Pulling from library/alpine
0a8490d0dfd3: Pull complete
Digest: sha256:dfbd4a3a8ebca874ebd2474f044a0b33600d4523d03b0df76e5c5986cb02d7e8
Status: Downloaded newer image for alpine:latest
```

Review the List of Local Docker Images

You can use the **docker images** command to see a list of all images on your system.

```
$ docker images

REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
alpine              latest              88e169ea8f46        4 weeks ago         3.98 MB
```

Run your First Docker Container!

Let's now run a Docker container based on this image.

```
$ docker run alpine ls -l  #docker container run alpine ls -l

total 52
drwxr-xr-x    2 root     root          4096 Dec 26 21:32 bin
drwxr-xr-x    5 root     root           340 Jan 25 22:26 dev
drwxr-xr-x   14 root     root          4096 Jan 25 22:26 etc
drwxr-xr-x    2 root     root          4096 Dec 26 21:32 home
drwxr-xr-x    5 root     root          4096 Dec 26 21:32 lib
drwxr-xr-x    5 root     root          4096 Dec 26 21:32 media
drwxr-xr-x    2 root     root          4096 Dec 26 21:32 mnt
dr-xr-xr-x  144 root     root             0 Jan 25 22:26 proc
drwx------    2 root     root          4096 Dec 26 21:32 root
drwxr-xr-x    2 root     root          4096 Dec 26 21:32 run
drwxr-xr-x    2 root     root          4096 Dec 26 21:32 sbin
drwxr-xr-x    2 root     root          4096 Dec 26 21:32 srv
dr-xr-xr-x   12 root     root             0 Jan 25 22:26 sys
drwxrwxrwt    2 root     root          4096 Dec 26 21:32 tmp
drwxr-xr-x    7 root     root          4096 Dec 26 21:32 usr
drwxr-xr-x   12 root     root          4096 Dec 26 21:32 var
```

Wow! What just Happened?

```
$ docker run alpine ls -l
```

#docker container run alpine ls -l

1. Locate the requested image.
2. Create a new container using the requested image.
3. Execute the provided command in the container.

Let's Try Some Other Commands!

What other Unix commands can you run? Try any of these or some of your personal favorites!

```
$ docker run alpine cal
```

```
$ docker run alpine echo "hello World"
```

```
$ docker run alpine pwd
```

```
$ docker run alpine id -u -n
```

Containers are FAST!

The commands we run execute almost immediately!

- Each time the Docker client runs the command in our container and then exits.
- Booting up a virtual machine, running a command, and then killing it would take *forever!*

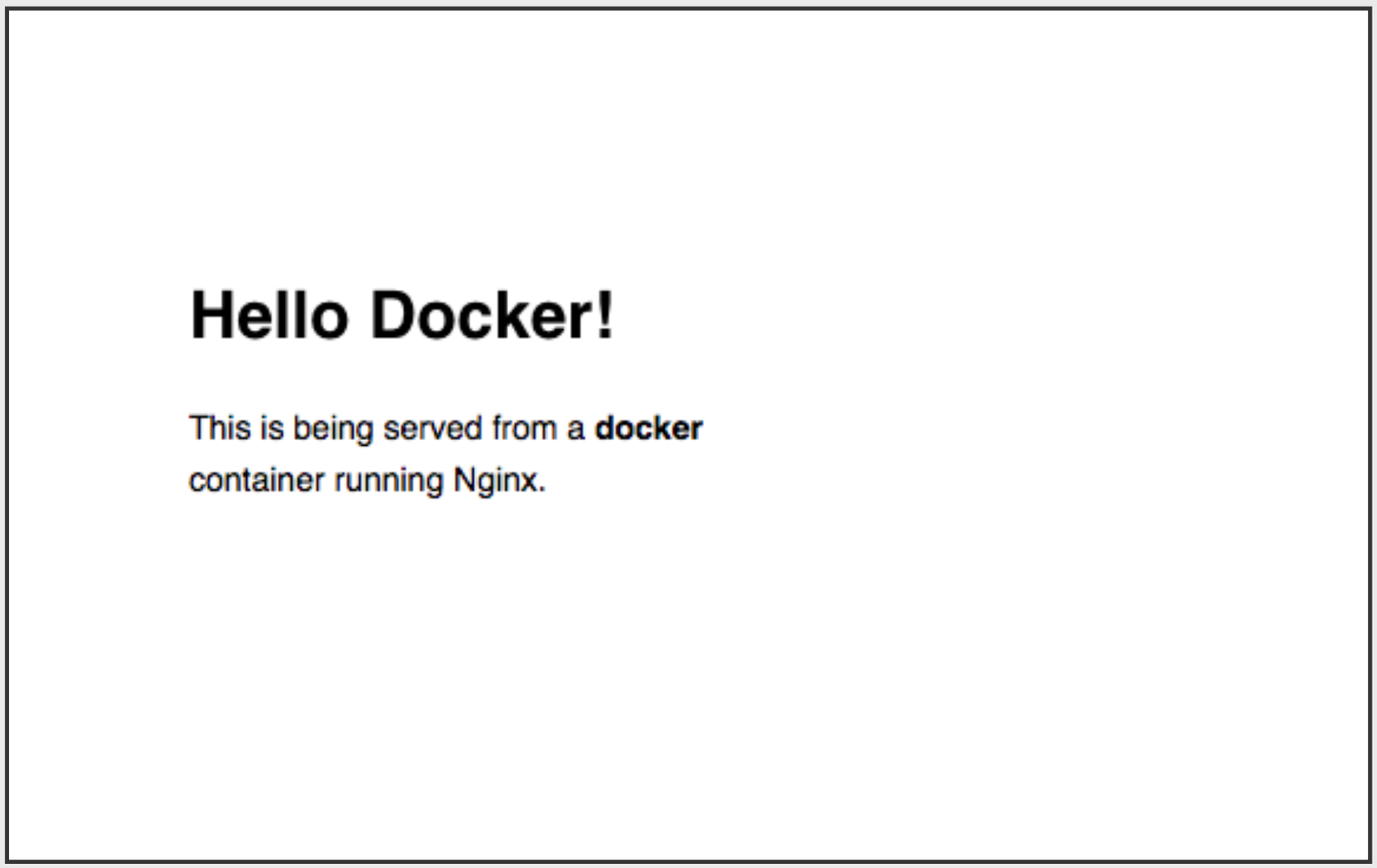
Let's get Interactive!

All of our commands have exited immediately after running them.
How do we run a container interactively?

```
$ docker run -it alpine /bin/sh

/ # uname -a
Linux 1b579b6024b4 4.9.4-moby #1 SMP Wed Jan 18 17:04:43 UTC 2017 x86_64 Linux

/ # ls
bin    dev    etc    home   lib    media  mnt    proc   root   run    sbin   srv
sys    tmp    usr    var

/ # cd usr/

/usr # ls
bin    lib    local  sbin   share

/ # exit
```

Let's Run a Web App with Docker!

Time for the real stuff – deploying web applications with Docker!
We're going to make a website that looks like this:



Let's Run the Image from Docker Hub.

We created an image on Docker Hub that contains the code under `seqvence/static-site`.

```
$ docker run -d seqvence/static-site

Unable to find image 'seqvence/static-site:latest' locally
latest: Pulling from seqvence/static-site
fdd5d7827f33: Pull complete
a3ed95caeb02: Pull complete
716f7a5f3082: Pull complete
7b10f03a0309: Pull complete
aff3ab7e9c39: Pull complete
Digest: sha256:41b286105f913fb7a5fbdce28d48bc80f1c77e3c4ce1b8280f28129ae0e94e9e
Status: Downloaded newer image for seqvence/static-site:latest
f3ade96bc4ffd15956021753783 7e213fea1ccb25aff772186aaec1b301b7060
```

***Note**: The -d flag enables detached mode, which detaches the running container from the terminal.*

So, What just Happened?

Notice that we didn't run `docker pull` before we ran the new image. Here's what happened:

```
$ docker run -d seqvence/static-site
```

1. Locate the requested image *(not found locally in this case)*.
2. Download this missing image from Docker Hub.
3. Create a new container using the requested image.
4. Run it in the background (*detached*) and return to the prompt.

Wait... Where is the Website?

We didn't tell Docker to publish any ports, so the website isn't accessible.

- Use the `-p` flag with `docker run` to tell Docker which ports to make accessible.
- Docker doesn't publish ports by default for security reasons.

How do we Stop a Detached Image?

We can use `docker stop` and `docker rm` to stop and remove a Docker Image that is running in the background.

```
$ docker ps  #docker container ls

CONTAINER ID        IMAGE                    COMMAND                  CREATED
f3ade96bc4ff        seqvence/static-site   "/bin/sh -c 'cd /u..."   2 hours ago

STATUS              PORTS                NAMES
Up 2 hours          80/tcp, 443/tcp      competent_bartik

$ docker stop f3ade96bc4ff  #docker container stop
f3ade96bc4ff

$ docker rm f3ade96bc4ff
f3ade96bc4ff
```

***Note**: The example above provides the CONTAINER ID on our system; you should use the value that you see in your terminal.*

Let's Try This Again...

This time we'll use the `-p` flag to publish the website on port 8888.

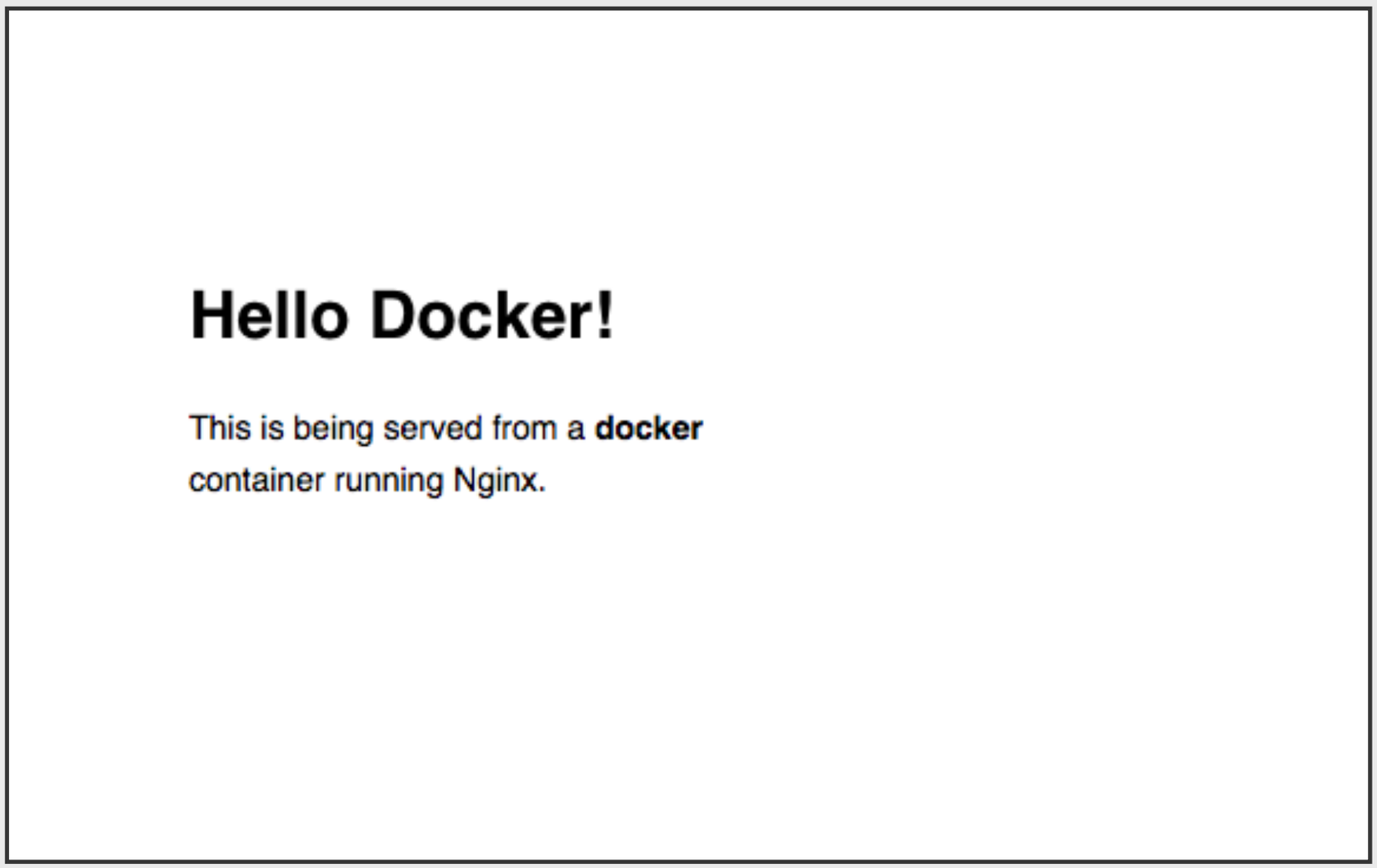
```
$ docker run --name static-site -d -p 8888:80 seqvence/static-site

Unable to find image 'seqvence/static-site:latest' locally
latest: Pulling from seqvence/static-site
fdd5d7827f33: Pull complete
a3ed95caeb02: Pull complete
716f7a5f3082: Pull complete
7b10f03a0309: Pull complete
aff3ab7e9c39: Pull complete
Digest: sha256:41b286105f913fb7a5fbdce28d48bc80f1c77e3c4ce1b8280f28129ae0e94e9e
Status: Downloaded newer image for seqvence/static-site:latest
1fd4d53f64d48534ad7db3f8bfa8b745977aab87161f70da56b4672e9e0868eb
```

***Note**: We also gave our container a name using the --name parameter. This will make it easier to stop when the time comes.*

Try Visiting your Website!

Head over to the following URL and you should see the website below:

http://localhost:8888



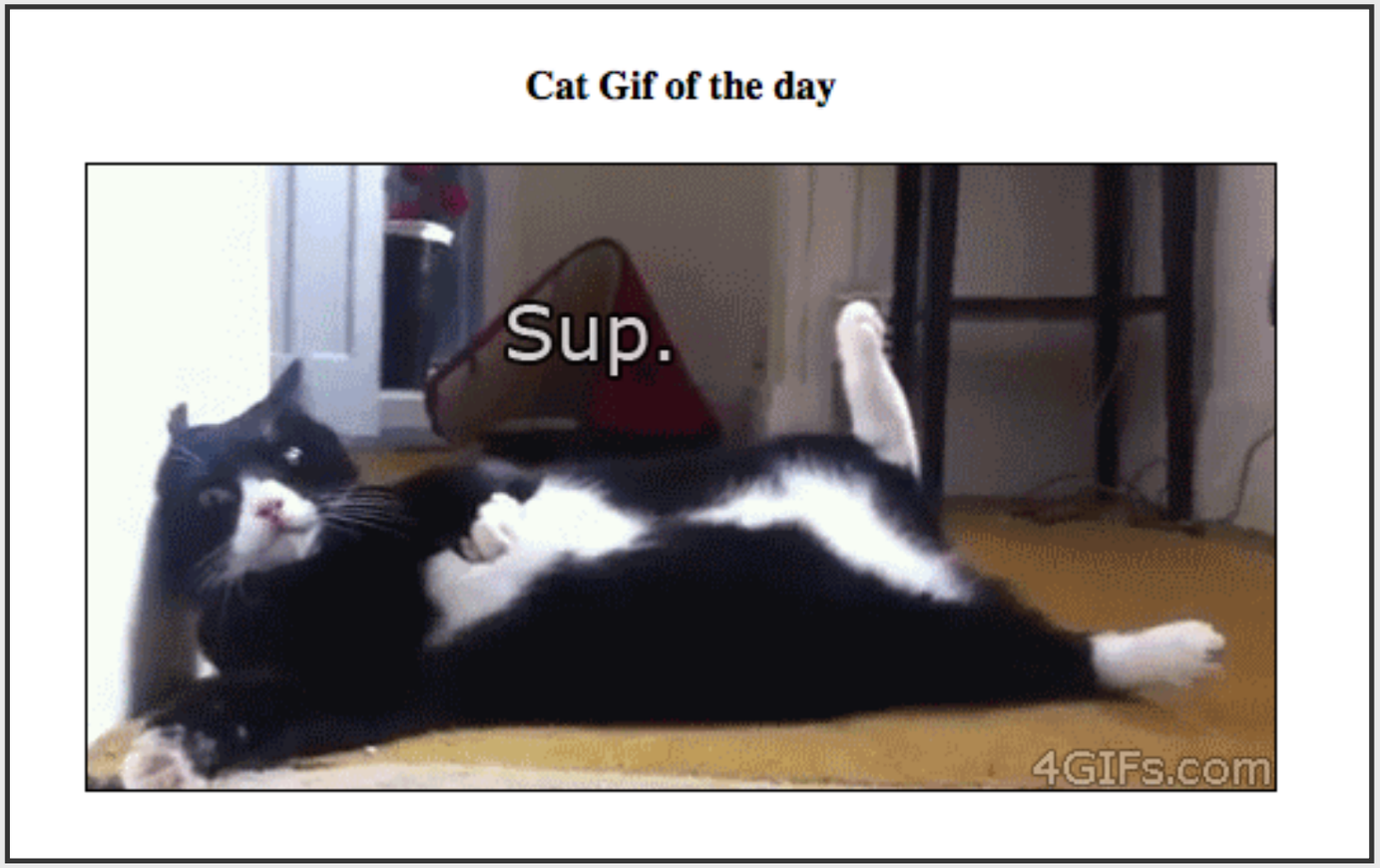
Let's Clean Up!

Since we started this process with a name, we don't need to use `docker ps` to figure out the ID.

```
$ docker stop static-site
static-site

$ docker rm static-site
static-site
```

Let's Make a Docker Image!

We've run other people's images so far, how do we create our own?
Let's find out by creating an app that displays cat photos.



Cat GIFs Courtesy of

BuzzFeed

https://github.com/docker/labs/blob/master/beginner/chapters/webapps.md

We're Going to Use Python + Flask!

Flask is a Framework for Python

- Makes it easy to build web applications by providing a standard set of tools.
- You don't need to know flask to understand this example.

Create a New Directory

Let’s start by creating a new directory on your local machine.

```
$ mkdir flask-app

$ cd flask-app
```

Open your Favorite Code Editor.

Open up the program you most frequently use to write code.
Some popular options are WebStorm, Atom, Vim, Sublime, & Notepad++.

Create your app.py File

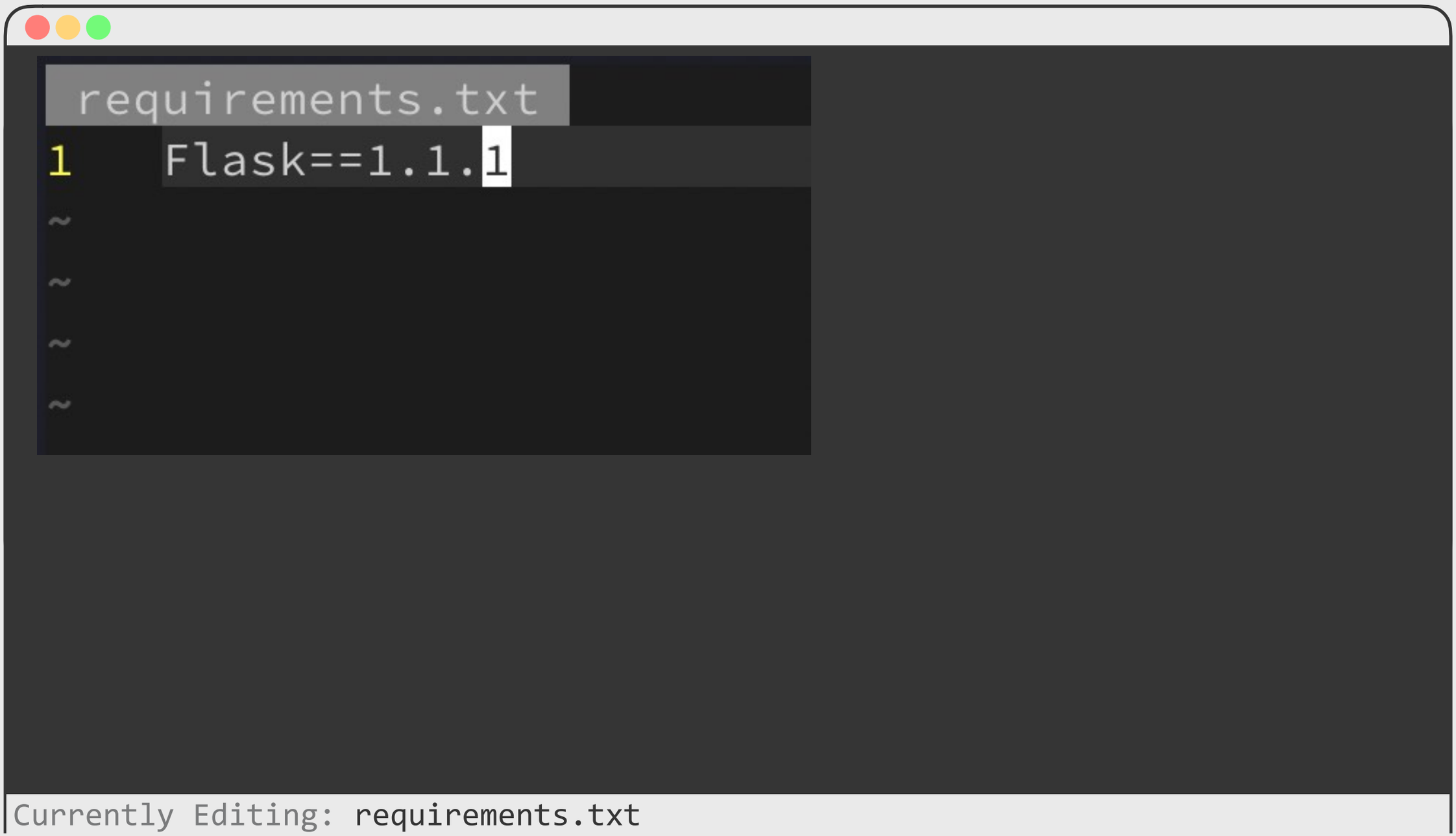
Create a file called `app.py` with the following content:

```
from flask import Flask, render_template
import random

app = Flask(__name__)

images = [
  “http://bit.ly/docker-cat-1",
  "http://bit.ly/docker-cat-2",
  "http://bit.ly/docker-cat-3"
]

@app.route('/')
def index():
  url = random.choice(images)
  return render_template("index.html", url=url)

if __name__ == "__main__":
  app.run(host="0.0.0.0")
```

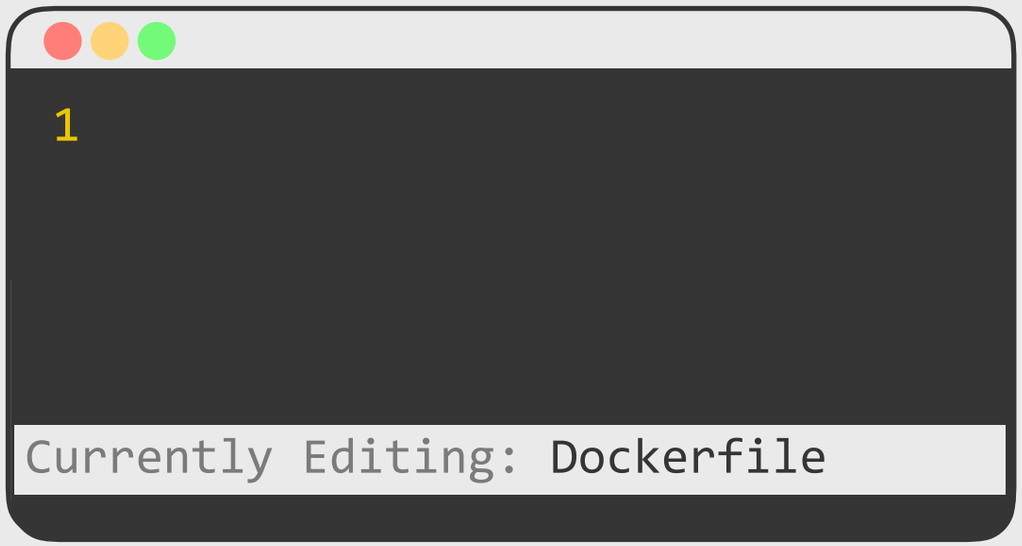
Currently Editing: app.py

Create your requirements.txt File

Create a file called `requirements.txt` with the following content:



Write your First Dockerfile!



A Dockerfile is a text file that contains a list of commands that Docker calls while creating an image.

FROM: Specify a Base Image

Think of this as a starting point for our brand new Docker Image:

```
1 # our base image
2 FROM alpine:3.5
```

Currently Editing: Dockerfile

RUN: Install Python & PIP

Our application requires Python and PIP to run, let's install them:

```
# our base image
FROM alpine:3.5

# Install python and pip
RUN apk add --update py2-pip
```

Currently Editing: Dockerfile

COPY & RUN: Install Flask

Let's copy our requirements.txt and use Pip to install Flask:

```
1 # our base image
2 FROM alpine:3.5
3
4 # Install python and pip
5 RUN apk add --update py2-pip
6
7 # install Python modules needed by the Python app
8 COPY requirements.txt /usr/src/app/
9 RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt
```

Currently Editing: Dockerfile

COPY: Create our Application Files

We can use the COPY command to import our app's files:

```
# our base image
FROM alpine:3.5

# Install python and pip
RUN apk add --update py2-pip

# install Python modules needed by the Python app
COPY requirements.txt /usr/src/app/
RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt

# copy files required for the app to run
COPY app.py /usr/src/app/
COPY templates/index.html /usr/src/app/templates/
```

Currently Editing: Dockerfile

EXPOSE: Tell Docker about Open Ports

Tell Docker to open up port 5000 (the default Flask port)

```
1 # our base image
2 FROM alpine:3.5
3
4 # Install python and pip
5 RUN apk add --update py2-pip
6
7 # install Python modules needed by the Python app
8 COPY requirements.txt /usr/src/app/
9 RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt
10
11 # copy files required for the app to run
12 COPY app.py /usr/src/app/
13 COPY templates/index.html /usr/src/app/templates/
14
15 # tell the port number the container should expose
16 EXPOSE 5000
```

Currently Editing: Dockerfile

CMD: Tell Docker how to Run the Image

The CMD command tells Docker how to run your image by default:

```
# our base image
FROM alpine:3.5

# Install python and pip
RUN apk add --update py2-pip

# install Python modules needed by the Python app
COPY requirements.txt /usr/src/app/
RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt

# copy files required for the app to run
COPY app.py /usr/src/app/
COPY templates/index.html /usr/src/app/templates/

# tell the port number the container should expose
EXPOSE 5000

# run the application
CMD ["python", "/usr/src/app/app.py"]
```

Currently Editing: Dockerfile

Your Finished Dockerfile!

Here's what your Dockerfile should look like all together:

```
# our base image
FROM alpine:3.5

# Install python and pip
RUN apk add --update py2-pip

# install Python modules needed by the Python app
COPY requirements.txt /usr/src/app/
RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt

# copy files required for the app to run
COPY app.py /usr/src/app/
COPY templates/index.html /usr/src/app/templates/

# tell the port number the container should expose
EXPOSE 5000

# run the application
CMD ["python", "/usr/src/app/app.py"]
```

Currently Editing: Dockerfile

Build the Docker Image

Now that you have your Dockerfile, you can build your image.
The `docker build` command does this:

```
$ docker build -t mlhacks/flask-example .

Sending build context to Docker daemon 7.168 kB
Step 1/8 : FROM alpine:3.5
 ---> 88e169ea8f46
Step 2/8 : RUN apk add --update py2-pip
 ---> Using cache
 ---> 092f0d63efa5
Step 3/8 : COPY requirements.txt /usr/src/app/
 ---> 6698c1620af9
Removing intermediate container 1cf339c62124
…
Step 8/8 : CMD python /usr/src/app/app.py
 ---> Running in db811e85de07
 ---> 58358996ea4c
Removing intermediate container db811e85de07
Successfully built 58358996ea4c
```

Run your Docker Image

Use the `docker run` command to run the image we just built.

Replace `mlhacks` with your Docker ID.

```
$ docker run -p 8888:5000 mlhacks/flask-example

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

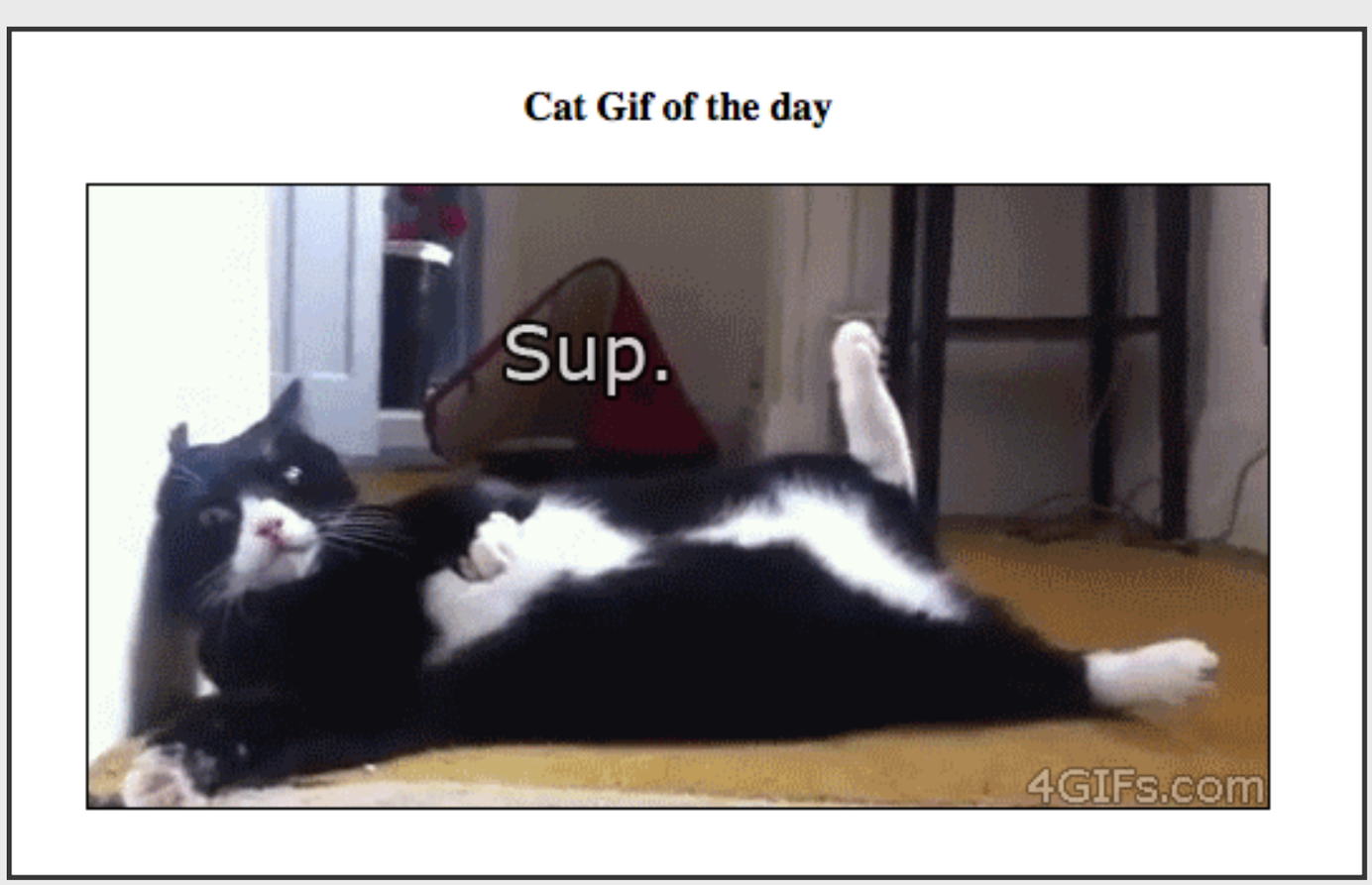
Head over to the following URL to see your handy work:

http://localhost:8888

Try Visiting your Website!

Head over to the following URL and you should see the website below:

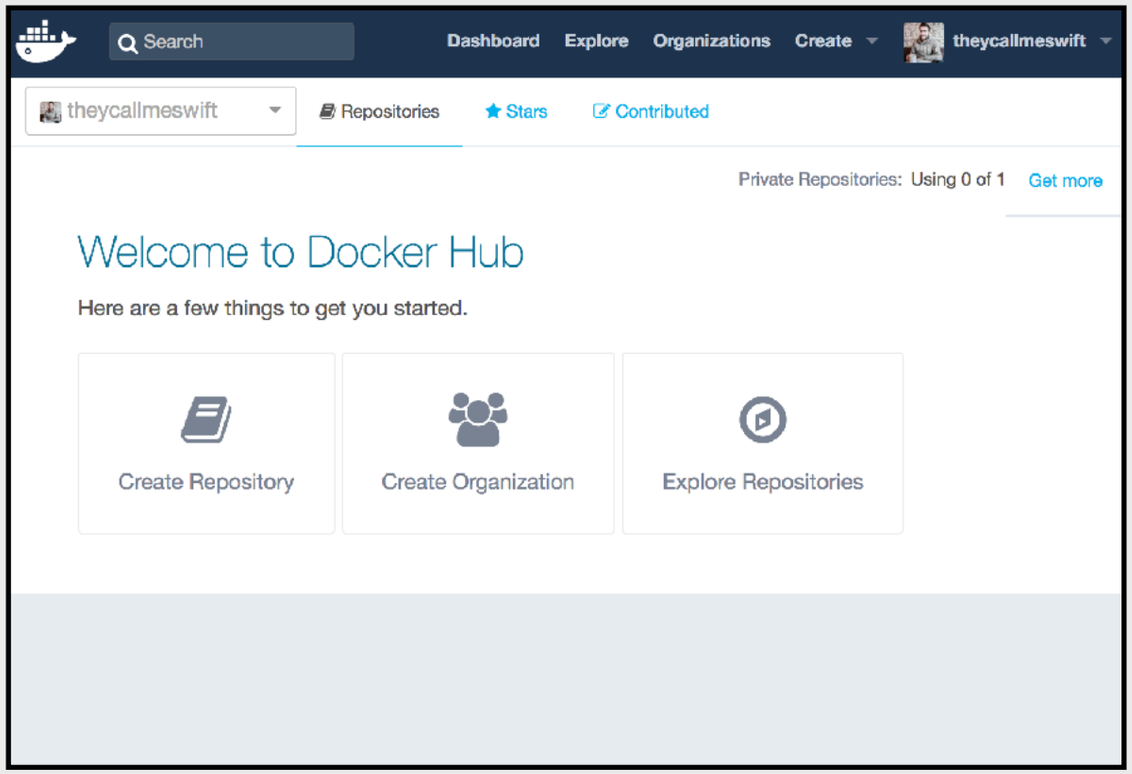
http://localhost:8888



Cat GIFs Courtesy of BuzzFeed

Host your Images on Docker Hub

Now that you've created you first Docker Image, you can publish it on Docker Hub for later use.



Login to your Docker Hub Account

In order to publish your image on Docker Hub, you need to login on the command line.

```
$ docker login

Login with your Docker ID to push and pull images from Docker Hub. If you don't
have a Docker ID, head over to https://hub.docker.com to create one.

Username: mlhacks
Password:

Login Succeeded
```

***Note**: Replace "mlhacks" with the Docker ID you created at the beginning of the workshop. Your password will be hidden when you type it out.*

Push your Image to Docker Hub

We can use the `docker push` command to publish an image we have locally on Docker Hub for later use.

```
$ docker push mlhacks/flask-example

The push refers to a repository [docker.io/mlhacks/flask-example]
efa6c37d0ae8: Pushed
779aa7159987: Pushed
b9fb8f60d4f6: Pushed
a6d6947400ab: Pushed
1f8090c7aa46: Pushed
1f8090c7aa46: Pushed

latest: digest:
sha256:512526fbdd5fc34d7b9f61712f114e7453ab01c5629e3f5e0cc35ed864daf size: 1572
```

***Note**: Replace "mlhacks" with the Docker ID you created at the beginning of the workshop.*

Find your Image on Docker Hub

Log into your Docker Hub Account to see your Image!

You just published your first Docker Image on the Docker Hub Registry. Other developers can download and install your image by using `docker run` now!