$ docker pull ubuntu:18.04

To get a new Docker image you can either get it from a registry (such as the Docker Hub) or create your own. There are tens of thousands of images available on [Docker Hub](https://hub.docker.com/explore/). You can also search for images directly from the command line using docker search.

An important distinction to be aware of when it comes to images is the difference between base and child images.

* **Base images** are images that have no parent image, usually images with an OS like ubuntu, busybox or debian.
* **Child images** are images that build on base images and add additional functionality.

Then there are official and user images, which can be both base and child images.

* **Official images** are images that are officially maintained and supported by the folks at Docker. These are typically one word long. In the list of images above, the python, ubuntu, busybox and hello-world images are official images.
* **User images** are images created and shared by users like you and me. They build on base images and add additional functionality. Typically, these are formatted as user/image-name.

### Our First Image

Now that we have a better understanding of images, it's time to create our own. Our goal in this section will be to create an image that sandboxes a simple [Flask](http://flask.pocoo.org/) application. For the purposes of this workshop, I've already created a fun little [Flask app](https://github.com/prakhar1989/docker-curriculum/tree/master/flask-app) that displays a random cat .gif every time it is loaded - because you know, who doesn't like cats? If you haven't already, please go ahead and clone the repository locally like so -

$ git clone https://github.com/prakhar1989/docker-curriculum.git

$ cd docker-curriculum/flask-app

*This should be cloned on the machine where you are running the docker commands and not inside a docker container.*

The next step now is to create an image with this web app. As mentioned above, all user images are based on a base image. Since our application is written in Python, the base image we're going to use will be [Python 3](https://hub.docker.com/_/python/).

### Dockerfile

A [Dockerfile](https://docs.docker.com/engine/reference/builder/) is a simple text file that contains a list of commands that the Docker client calls while creating an image. It's a simple way to automate the image creation process. The best part is that the [commands](https://docs.docker.com/engine/reference/builder/#from) you write in a Dockerfile are *almost* identical to their equivalent Linux commands. This means you don't really have to learn new syntax to create your own dockerfiles.

The application directory does contain a Dockerfile but since we're doing this for the first time, we'll create one from scratch. To start, create a new blank file in our favorite text-editor and save it in the **same** folder as the flask app by the name of Dockerfile.

We start with specifying our base image. Use the FROM keyword to do that -

FROM python:3

The next step usually is to write the commands of copying the files and installing the dependencies. First, we set a working directory and then copy all the files for our app.

# set a directory for the app

WORKDIR /usr/src/app

# copy all the files to the container

COPY . .

Now, that we have the files, we can install the dependencies.

# install dependencies

RUN pip install --no-cache-dir -r requirements.txt

The next thing we need to specify is the port number that needs to be exposed. Since our flask app is running on port 5000, that's what we'll indicate.

EXPOSE 5000

The last step is to write the command for running the application, which is simply - python ./app.py. We use the [CMD](https://docs.docker.com/engine/reference/builder/#cmd) command to do that -

CMD ["python", "./app.py"]

The primary purpose of CMD is to tell the container which command it should run when it is started. With that, our Dockerfile is now ready. This is how it looks -

FROM python:3

# set a directory for the app

WORKDIR /usr/src/app

# copy all the files to the container

COPY . .

# install dependencies

RUN pip install --no-cache-dir -r requirements.txt

# define the port number the container should expose

EXPOSE 5000

# run the command

CMD ["python", "./app.py"]

Now that we have our Dockerfile, we can build our image. The docker build command does the heavy-lifting of creating a Docker image from a Dockerfile.

The section below shows you the output of running the same. Before you run the command yourself (don't forget the period), make sure to replace my username with yours. This username should be the same one you created when you registered on [Docker hub](https://hub.docker.com/). If you haven't done that yet, please go ahead and create an account. The docker build command is quite simple - it takes an optional tag name with -t and a location of the directory containing the Dockerfile.

$ docker build -t yourusername/catnip .

Sending build context to Docker daemon 8.704 kB

Step 1 : FROM python:3

# Executing 3 build triggers...

Step 1 : COPY requirements.txt /usr/src/app/

---> Using cache

Step 1 : RUN pip install --no-cache-dir -r requirements.txt

---> Using cache

Step 1 : COPY . /usr/src/app

---> 1d61f639ef9e

Removing intermediate container 4de6ddf5528c

Step 2 : EXPOSE 5000

---> Running in 12cfcf6d67ee

---> f423c2f179d1

Removing intermediate container 12cfcf6d67ee

Step 3 : CMD python ./app.py

---> Running in f01401a5ace9

---> 13e87ed1fbc2

Removing intermediate container f01401a5ace9

Successfully built 13e87ed1fbc2

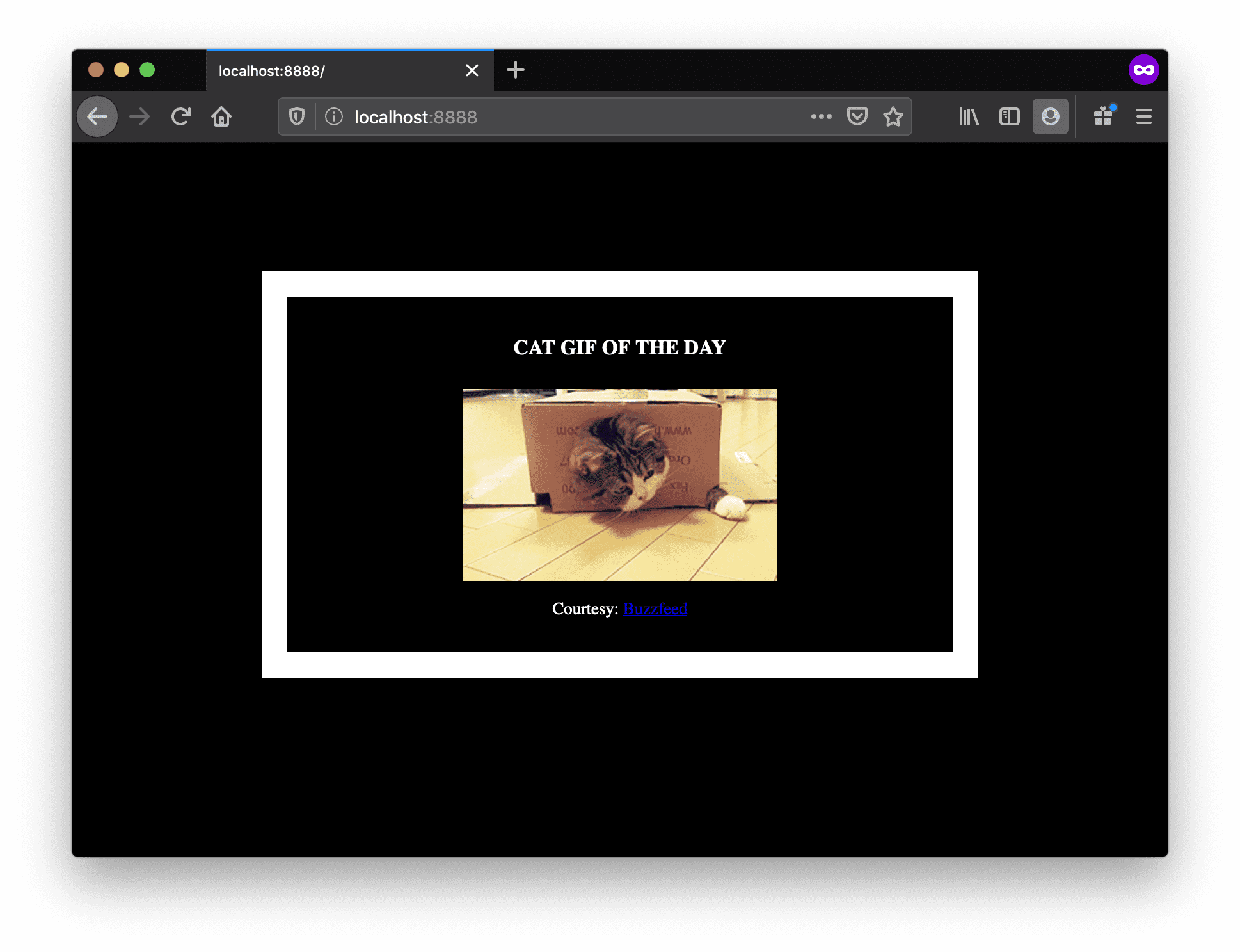
If you don't have the python:3 image, the client will first pull the image and then create your image. Hence, your output from running the command will look different from mine. If everything went well, your image should be ready! Run docker images and see if your image shows.

The last step in this section is to run the image and see if it actually works (replacing my username with yours).

$ docker run -p 8888:5000 yourusername/catnip

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

The command we just ran used port 5000 for the server inside the container and exposed this externally on port 8888. Head over to the URL with port 8888, where your app should be live.



Congratulations! You have successfully created your first docker image.