Prerequisites

https://docker-curriculum.com/

https://github.com/ → charleshoanduong1111 (UserID)

Windows OS → Docker Desktop installed.

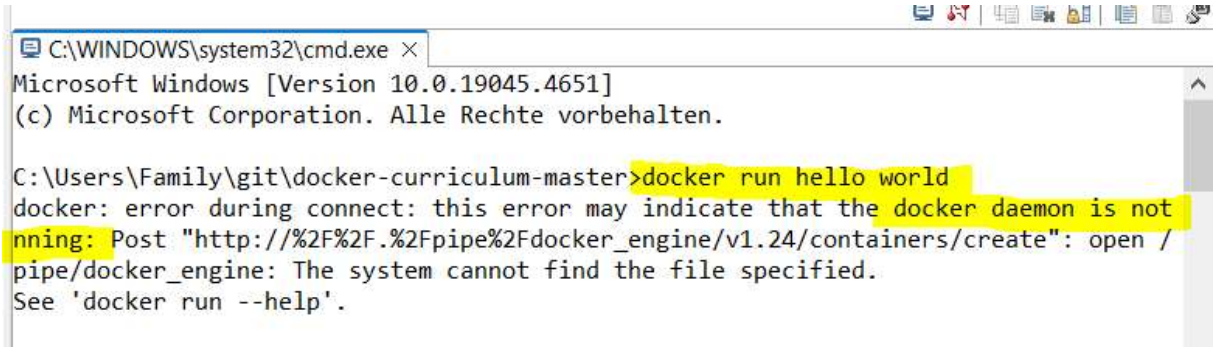
https://hub.docker.com/ → charleshoanduong1111 (UserID) | cduong1111 (UserID)

https://aws.amazon.com/ → charleshoanduong1111 (UserID)

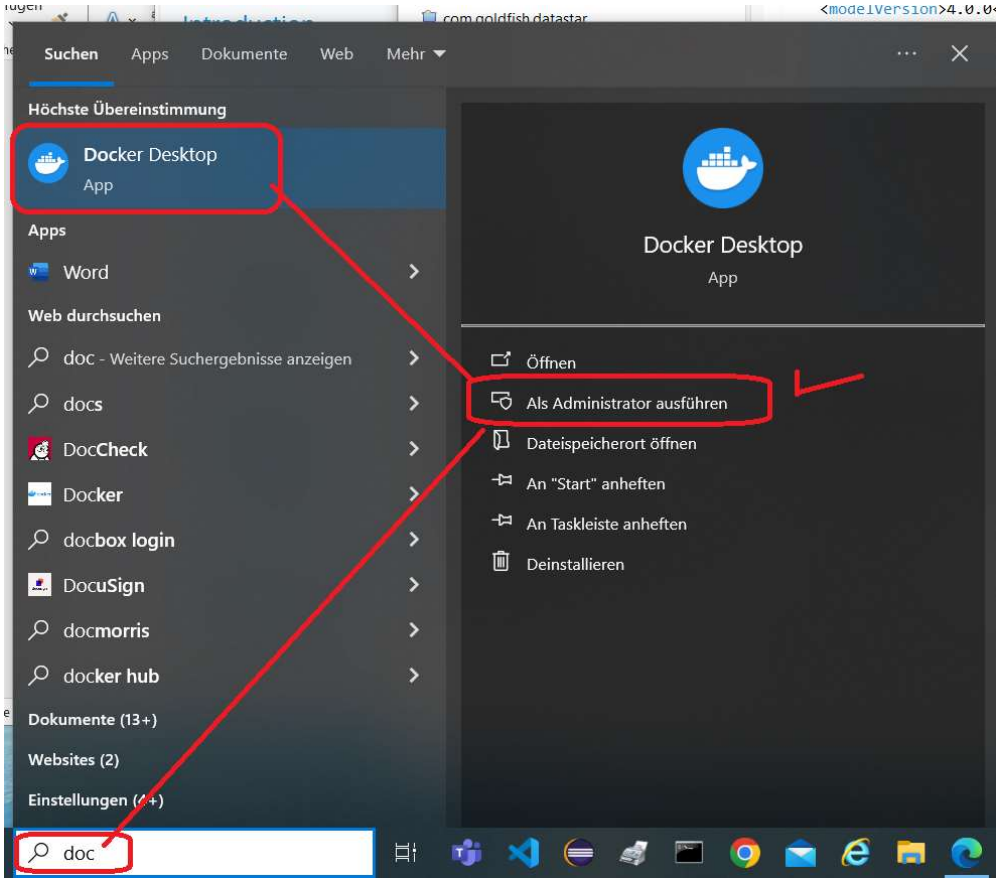
Getting Started

This document contains a series of several sections, each of which explains a particular aspect of Docker. In each section, we will be typing commands (or writing code). All the code used in the tutorial is available in the Github repo. → https://github.com/prakhar1989/docker-curriculum

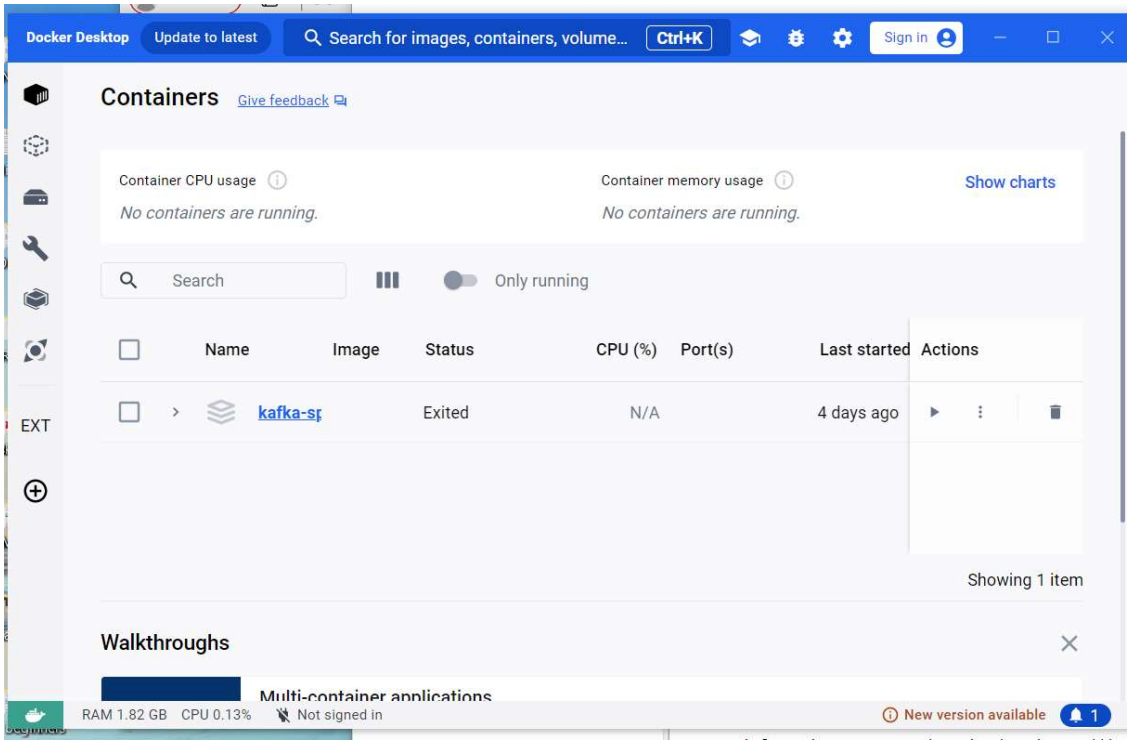
C:\Users\Family\git\docker-curriculum-master>docker run hello-world



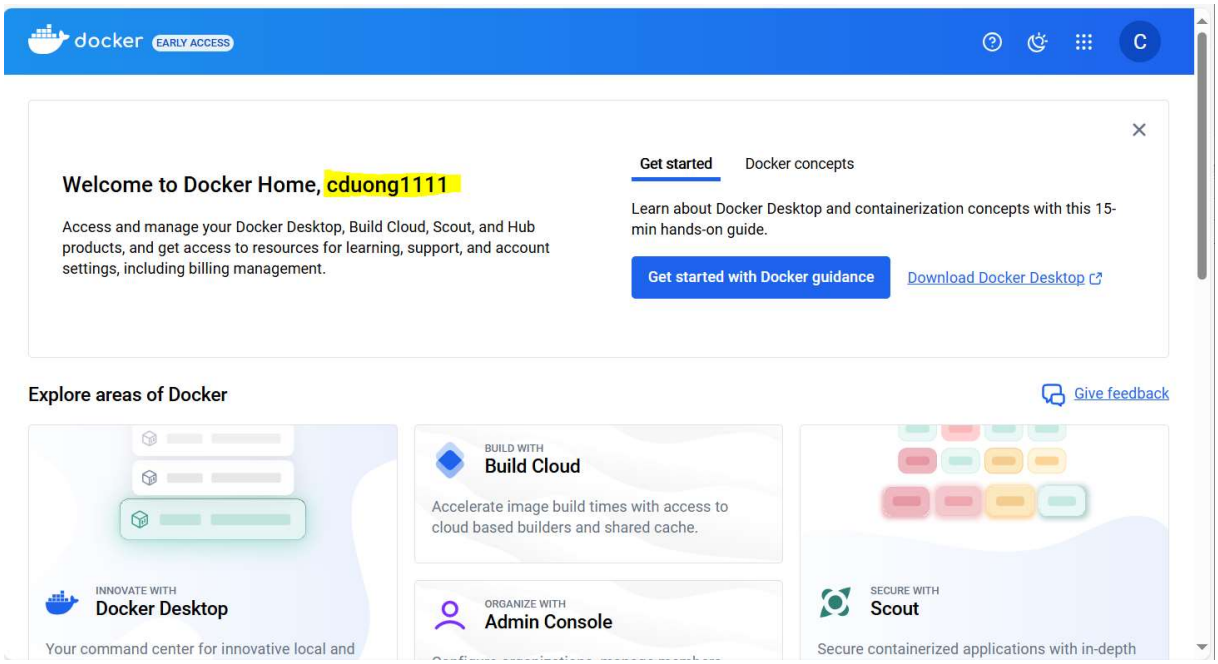
Open → Docker Desktop



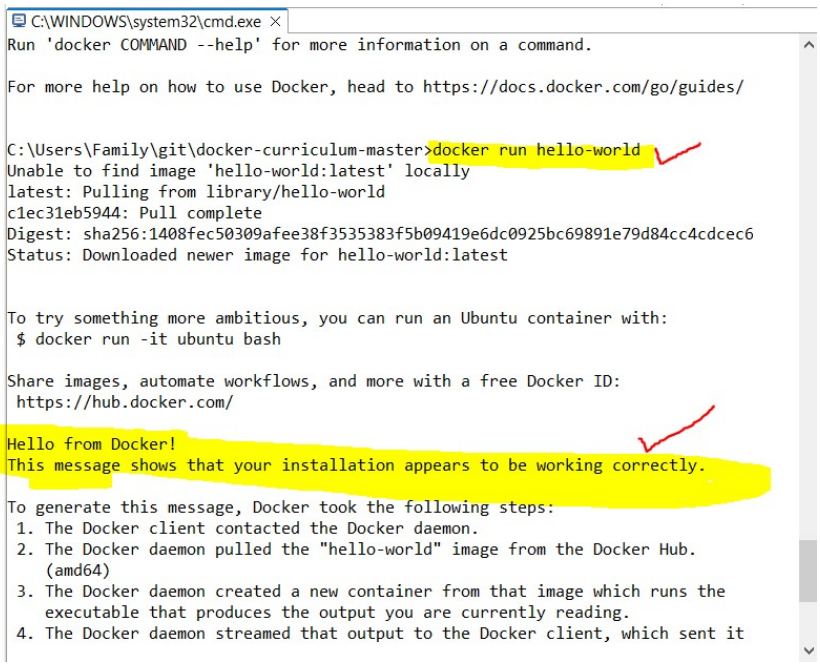
Docker Desktop opened



Login with userid cduong1111 instead of email.



C:\Users\Family\git\docker-curriculum-master>docker run hello-world



To try something more ambitious, you can run an Ubuntu container with:

$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:

https://hub.docker.com/

For more examples and ideas, visit:

https://docs.docker.com/get-started/

C:\Users\Family\git\docker-curriculum-master>docker run -it ubuntu bash

```
C:\Users\Family\git\docker-curriculum-master>docker run -it ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
9c704ecd0c69: Pull complete
Digest: sha256:2e863c44b718727c860746568e1d54afd13b2fa71b160f5cd9058fc436217b30
Status: Downloaded newer image for ubuntu:latest
root@5e3b46a55f92:/#
```

Hello World

Playing with Busybox

C:\Users\Family\git\docker-curriculum-master>docker pull busybox

```
C:\Users\Family\git\docker-curriculum-master>docker pull busybox
Using default tag: latest
latest: Pulling from library/busybox
ec562eabd705: Pull complete
Digest: sha256:9ae97d36d26566ff84e8893c64a6dc4fe8ca6d1144bf5b87b2b85a32def253c7
Status: Downloaded newer image for busybox:latest
docker.io/library/busybox:latest


  View a summary of image vulnerabilities and recommendations → docker scout quickview busybox
```

The pull command fetches the busybox **image** from the **Docker registry** and saves it to our system. You can use the docker images command to see a list of all images on your system.

C:\Users\Family\git\docker-curriculum-master>docker images

```
C:\Users\Family\git\docker-curriculum-master>docker images
REPOSITORY                TAG       IMAGE ID       CREATED          SIZE
ubuntu                    latest    35a88802559d   2 months ago     78.1MB
busybox                   latest    65ad0d468eb1   15 months ago    4.26MB
hello-world               latest    d2c94e258dcb   15 months ago    13.3kB
wurstmeister/kafka        latest    a692873757c0   2 years ago      468MB
wurstmeister/zookeeper    latest    3f43f72cb283   5 years ago      510MB
```

Docker Run

Great! Let's now run a Docker **container** based on this image. To do that we are going to use the almighty docker run command.

C:\Users\Family\git\docker-curriculum-master>docker run busybox

```
C:\Users\Family\git\docker-curriculum-master>docker run busybox

C:\Users\Family\git\docker-curriculum-master>
```

Wait, nothing happened! Is that a bug? Well, no. Behind the scenes, a lot of stuff happened. When you call run, the Docker client finds the image (busybox in this case), loads up the container and then runs a command in that container. When we run docker run busybox, we didn't provide a command, so the container

booted up, ran an empty command and then exited. Well, yeah - kind of a bummer. Let's try something more exciting.

C:\Users\Family\git\docker-curriculum-master>docker run busybox echo "hello from busybox"

```
C:\Users\Family\git\docker-curriculum-master>docker run busybox echo "hello from busybox"
hello from busybox
```

Nice - finally we see some output. In this case, the Docker client dutifully ran the echo command in our busybox container and then exited it. If you've noticed, all of that happened pretty quickly. Imagine booting up a virtual machine, running a command and then killing it. Now you know why they say containers are fast! Ok, now it's time to see the docker ps command. The docker ps command shows you all containers that are currently running.

C:\Users\Family\git\docker-curriculum-master>docker ps

```
C:\Users\Family\git\docker-curriculum-master>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

Since no containers are running, we see a blank line. Let's try a more useful variant: docker ps -a

C:\Users\Family\git\docker-curriculum-master>docker ps -a

```
C:\Users\Family\git\docker-curriculum-master>docker ps -a
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS                       PORTS     NAMES
f05f7c1a4996   busybox                "echo 'hello from bu…"   2 minutes ago   Exited (0) 2 minutes ago               blissful_satoshi
3f0caee8f652   busybox                "sh"                     5 minutes ago   Exited (0) 5 minutes ago               trusting_hypatia
5e3b46a55f92   ubuntu                 "bash"                   13 hours ago    Exited (255) 11 minutes ago            musing_brown
9788be156918   hello-world            "/hello"                 13 hours ago    Exited (0) 13 hours ago                xenodochial_greider
41b412fb4d1c   hello-world            "/hello"                 13 hours ago    Exited (0) 13 hours ago                jovial_agnesi
c2003757acaa   wurstmeister/kafka     "start-kafka.sh"         5 days ago      Exited (143) 5 days ago                kafka-spring-boot-example-kafka-1
051be57272ca   wurstmeister/zookeeper "/bin/sh -c '/usr/sb…"   5 days ago      Exited (137) 5 days ago                kafka-spring-boot-example-zookeeper-1
```

So what we see above is a list of all containers that we ran. Do notice that the STATUS column shows that these containers exited a few minutes ago.

You're probably wondering if there is a way to run more than just one command in a container. Let's try that now:

C:\Users\Family\git\docker-curriculum-master>docker run -it busybox sh

```
C:\Users\Family\git\docker-curriculum-master>docker run -it busybox sh
/ # exit

C:\Users\Family\git\docker-curriculum-master>
```

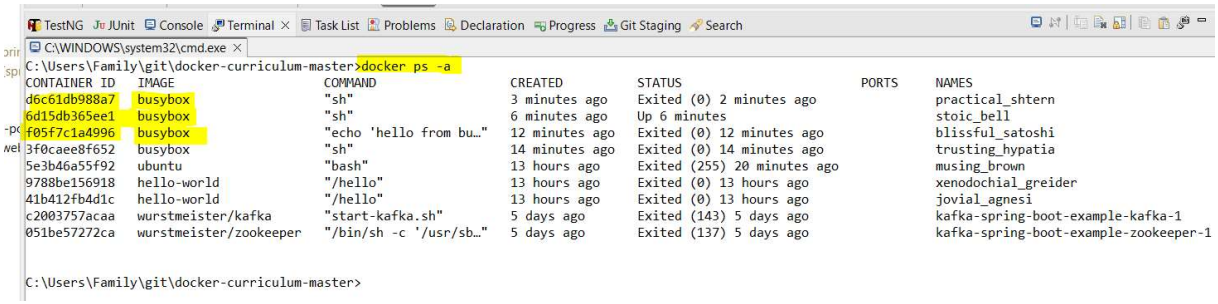
Running the run command with the -it flags attaches us to an interactive tty in the container. Now we can run as many commands in the container as we want. Take some time to run your favorite commands.

***Danger Zone**: If you're feeling particularly adventurous you can try rm -rf bin in the container. Make sure you run this command in the container and **not** in your laptop/desktop. Doing this will make any other commands like ls, uptime not work. Once everything stops working, you can exit the container (type exit and press Enter) and then start it up again with the docker run -it busybox sh command. Since Docker creates a new container every time, everything should start working again.*

That concludes a whirlwind tour of the mighty docker run command, which would most likely be the command you'll use most often. It makes sense to spend some time getting comfortable with it. To find out more about run, use docker run --help to see a list of all flags it supports. As we proceed further, we'll see a few more variants of docker run.

Before we move ahead though, let's quickly talk about deleting containers. We saw above that we can still see remnants of the container even after we've exited by running docker ps -a. Throughout this tutorial, you'll run docker run multiple times and leaving stray containers will eat up disk space. Hence, as a rule of thumb, I clean up containers once I'm done with them. To do that, you can run the docker rm command. Just copy the container IDs from above and paste them alongside the command.

C:\Users\Family\git\docker-curriculum-master>docker ps -a



C:\Users\Family\git\docker-curriculum-master>docker rm d6c61db988a7 6d15db365ee1 f05f7c1a4996 3f0caee8f652

On deletion, you should see the IDs echoed back to you. If you have a bunch of containers to delete in one go, copy-pasting IDs can be tedious. In that case, you can simply run -

$ docker rm $(docker ps -a -q -f status=exited)

This command deletes all containers that have a status of exited. In case you're wondering, the -q flag, only returns the numeric IDs and -f filters output based on conditions provided. One last thing that'll be useful is the --rm flag that can be passed to docker run which automatically deletes the container once it's exited from. For one off docker runs, --rm flag is very useful.

In later versions of Docker, the docker container prune command can be used to achieve the same effect.

C:\Users\Family\git\docker-curriculum-master>docker container prune

```
C:\Users\Family\git\docker-curriculum-master>docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N]
```

```
C:\Users\Family\git\docker-curriculum-master>docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
5e3b46a55f92519c5b292b145c246faadbd6606461ff9bd6f87d593728c1cacb
9788be1569187ac932071867a0aeeacae663181e0248be9b89e7322e551382d0
41b412fb4d1c91839e433b4378f6bdb2818553c2ea2e38335aa412d0cfd596cb
c2003757acaa58bad4a63ff8fe2aa1604d5bc07f280c6cdd6b8198a8f363f9ee
051be57272ca14e67e7cb1341c684e23dbd581be39da2ee4e0200858cff69f30

Total reclaimed space: 242.5kB
```

```
C:\Users\Family\git\docker-curriculum-master>docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS          PORTS     NAMES
6d15db365ee1   busybox   "sh"      21 minutes ago   Up 21 minutes             stoic_bell
```

Lastly, you can also delete images that you no longer need by running docker rmi.

```
C:\Users\Family\git\docker-curriculum-master>docker rmi
"docker rmi" requires at least 1 argument.
See 'docker rmi --help'.

Usage:  docker rmi [OPTIONS] IMAGE [IMAGE...]

Remove one or more images
```

Terminology

In the last section, we used a lot of Docker-specific jargon which might be confusing to some. So before we go further, let me clarify some terminology that is used frequently in the Docker ecosystem.

- *Images* - The blueprints of our application which form the basis of containers. In the demo above, we used the docker pull command to download the **busybox** image.
- *Containers* - Created from Docker images and run the actual application. We create a container using docker run which we did using the busybox image that we downloaded. A list of running containers can be seen using the docker ps command.
- *Docker Daemon* - The background service running on the host that manages building, running and distributing Docker containers. The daemon is the process that runs in the operating system which clients talk to.

- *Docker Client* - The command line tool that allows the user to interact with the daemon. More generally, there can be other forms of clients too - such as Kitematic which provide a GUI to the users.
- *Docker Hub* - A registry of Docker images. You can think of the registry as a directory of all available Docker images. If required, one can host their own Docker registries and can use them for pulling images.

Webapps with Docker

Great! So we have now looked at docker run, played with a Docker container and also got a hang of some terminology. Armed with all this knowledge, we are now ready to get to the real-stuff, i.e. deploying web applications with Docker!

Static Sites

Let's start by taking baby-steps. The first thing we're going to look at is how we can run a dead-simple static website. We're going to pull a Docker image from Docker Hub, run the container and see how easy it is to run a webserver.

Let's begin. The image that we are going to use is a single-page website that I've already created for the purpose of this demo and hosted on the registry - prakhar1989/static-site (https://github.com/prakhar1989/docker-curriculum). We can download and run the image directly in one go using docker run. As noted above, the --rm flag automatically removes the container when it exits and the -it flag specifies an interactive terminal which makes it easier to kill the container with Ctrl+C (on windows).

C:\Users\Family\git\docker-curriculum-master>docker run --rm -it prakhar1989/static-site

```
C:\Users\Family\git\docker-curriculum-master>docker run --rm -it prakhar1989/static-site
Unable to find image 'prakhar1989/static-site:latest' locally
latest: Pulling from prakhar1989/static-site
d4bce7fd68df: Pull complete
a3ed95caeb02: Pull complete
573113c4751a: Pull complete
31917632be33: Pull complete
77e66f18af1c: Pull complete
df3f108f3ade: Pull complete
d7a279eb19f5: Pull complete
e798589c05c5: Pull complete
78eeaf458ae0: Pull complete
Digest: sha256:bb6907c8db9ac4c6cadb25162a979e286575cd8b27727c08c7fbaf30988534db
Status: Downloaded newer image for prakhar1989/static-site:latest
Nginx is running...
```

Since the image doesn't exist locally, the client will first fetch the image from the registry and then run the image. If all goes well, you should see a Nginx is running... message in your terminal. Okay now that the server is running, how to see the website? What port is it running on? And more importantly, how do we access the container directly from our host machine? Hit Ctrl+C to stop the container.

Well, in this case, the client is not exposing any ports so we need to re-run the docker run command to publish ports. While we're at it, we should also find a way so that our terminal is not attached to the running container. This way, you can happily close your terminal and keep the container running. This is called **detached** mode.

C:\Users\Family\git\docker-curriculum-master>docker run -d -P --name static-site prakhar1989/static-site

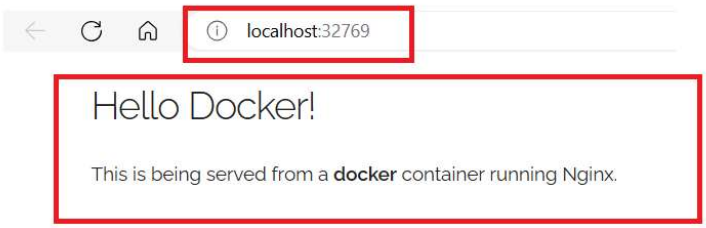
```
Nginx is running...
^C
C:\Users\Family\git\docker-curriculum-master>docker run -d -P --name static-site prakhar1989/static-site
9d6b80af14ce00cd72ffe7ce4fe4a4f5f29515148393086963cde84c2fb5fa02
```

In the above command, -d will detach our terminal, -P will publish all exposed ports to random ports and finally --name corresponds to a name we want to give. Now we can see the ports by running the docker port [CONTAINER] command

C:\Users\Family\git\docker-curriculum-master>docker port static-site

```
C:\Users\Family\git\docker-curriculum-master>docker port static-site
80/tcp -> 0.0.0.0:32769
443/tcp -> 0.0.0.0:32768
```

You can open http://localhost:32769 in your browser.



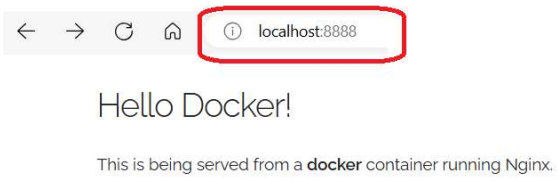
Note: If you're using docker-toolbox, then you might need to use docker-machine ip default to get the IP.

You can also specify a custom port to which the client will forward connections to the container.

C:\Users\Family\git\docker-curriculum-master>docker run -p 8888:80 prakhar1989/static-site

```
C:\Users\Family\git\docker-curriculum-master>docker run -d -P --name static-site prakhar1989/static-s
9d6b80af14ce00cd72ffe7ce4fe4a4f5f29515148393086963cde84c2fb5fa02

C:\Users\Family\git\docker-curriculum-master>docker port static-site
80/tcp -> 0.0.0.0:32769
443/tcp -> 0.0.0.0:32768

C:\Users\Family\git\docker-curriculum-master>docker run -p 8888:80 prakhar1989/static-site
Nginx is running...
```

You can open http://localhost:8888 in your browser.



And you can see more information under Nginx is running....

```
C:\Users\Family\git\docker-curriculum-master>docker run -p 8888:80 prakhar1989/static-site
Nginx is running...
172.17.0.1 - - [13/Aug/2024:08:27:27 +0000] "GET / HTTP/1.1" 200 2041 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) A
ppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36 Edg/127.0.0.0" "-"
172.17.0.1 - - [13/Aug/2024:08:27:27 +0000] "GET /css/normalize.css HTTP/1.1" 200 7797 "http://localhost:8888/" "Mozil
la/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36 Edg/127.0.0
.0" "-"
172.17.0.1 - - [13/Aug/2024:08:27:27 +0000] "GET /css/skeleton.css HTTP/1.1" 200 11452 "http://localhost:8888/" "Mozil
la/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36 Edg/127.0.0
.0" "-"
172.17.0.1 - - [13/Aug/2024:08:27:27 +0000] "GET /images/favicon.png HTTP/1.1" 200 1156 "http://localhost:8888/" "Mozi
lla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36 Edg/127.0.
0.0" "-"
172.17.0.1 - - [13/Aug/2024:08:29:45 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl
eWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36 Edg/127.0.0.0" "-"

C:\Users\Family\git\docker-curriculum-master>docker run -p 8888:80 prakhar1989/static-site
Nginx is running...
172.17.0.1 - - [13/Aug/2024:08:30:22 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) Appl
eWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36 Edg/127.0.0.0" "-"
```

To stop a detached container, run docker stop by giving the container ID. In this case, we can use the name static-site we used to start the container.

C:\Users\Family\git\docker-curriculum-master>docker stop static-site

```
C:\Users\Family\git\docker-curriculum-master>docker stop static-site
static-site
```

I'm sure you agree that was super simple. To deploy this on a real server you would just need to install Docker, and run the above Docker command. Now that you've seen how to run a webserver inside a Docker image, you must be wondering - how do I create my own Docker image? This is the question we'll be exploring in the next section.

Docker Images

We've looked at images before, but in this section we'll dive deeper into what Docker images are and build our own image! Lastly, we'll also use that image to run our application locally and finally deploy on AWS to share it with our friends! Excited? Great! Let's get started.

Docker images are the basis of containers. In the previous example, we **pulled** the *Busybox* image from the registry and asked the Docker client to run a container **based** on that image. To see the list of images that are available locally, use the docker images command.

C:\Users\Family\git\docker-curriculum-master>docker images

```
C:\Users\Family\git\docker-curriculum-master>docker images
REPOSITORY                  TAG       IMAGE ID       CREATED         SIZE
ubuntu                      latest    35a88802559d   2 months ago    78.1MB
busybox                     latest    65ad0d468eb1   15 months ago   4.26MB
hello-world                 latest    d2c94e258dcb   15 months ago   13.3kB
wurstmeister/kafka          latest    a692873757c0   2 years ago     468MB
wurstmeister/zookeeper      latest    3f43f72cb283   5 years ago     510MB
prakhar1989/static-site     latest    f01030e1dcf3   8 years ago     134MB
```

The above gives a list of images that I've pulled from the registry, along with ones that I've created myself (we'll shortly see how). The TAG refers to a particular snapshot of the image and the IMAGE ID is the corresponding unique identifier for that image.

For simplicity, you can think of an image akin to a git repository - images can be committed with changes and have multiple versions. If you don't provide a specific version number, the client defaults to latest. For example, you can pull a specific version of ubuntu image

C:\Users\Family\git\docker-curriculum-master>docker pull ubuntu:18.04

```
C:\Users\Family\git\docker-curriculum-master>docker pull ubuntu:18.04
18.04: Pulling from library/ubuntu
7c457f213c76: Pull complete
Digest: sha256:152dc042452c496007f07ca9127571cb9c29697f42acbfad72324b2bb2e43c98
Status: Downloaded newer image for ubuntu:18.04
docker.io/library/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. 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 application. For the purposes of this workshop, I've already created a fun little 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

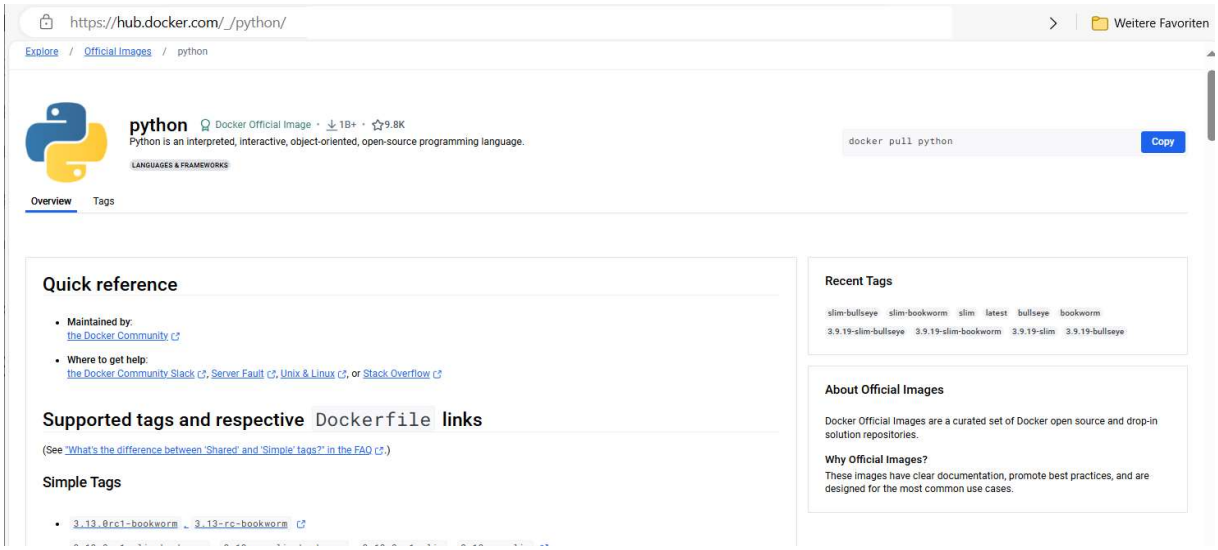
$ cd docker-curriculum/flask-app

```
C:\Users\Family\git\docker-curriculum-master>dir
 Datenträger in Laufwerk C: ist Windows
 Volumeseriennummer: 0E81-013A

 Verzeichnis von C:\Users\Family\git\docker-curriculum-master

12.08.2024  23:22    <DIR>          .
12.08.2024  23:22    <DIR>          ..
04.09.2023  23:43               116 .gitignore
12.08.2024  09:36               228 .project
12.08.2024  23:23    <DIR>          documentation
12.08.2024  09:35    <DIR>          flask-app
04.09.2023  23:43             1'074 LICENSE
04.09.2023  23:43               207 nodemon.json
04.09.2023  23:43           199'135 package-lock.json
04.09.2023  23:43             1'359 package.json
04.09.2023  23:43               195 README.md
               7 Datei(en),        202'314 Bytes
               4 Verzeichnis(se), 233'070'006'272 Bytes frei

C:\Users\Family\git\docker-curriculum-master>cd flask-app

C:\Users\Family\git\docker-curriculum-master\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 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 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:3.8
```

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 command to do that -

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

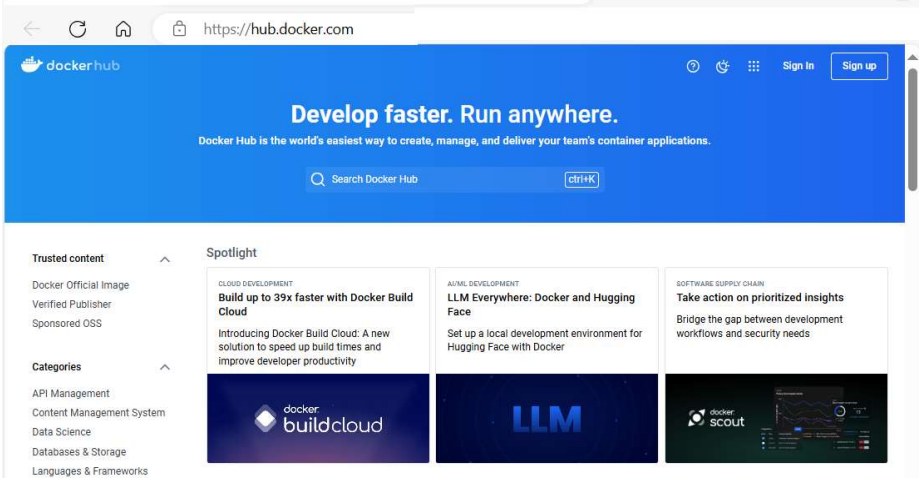
The primary purpose of CMD is to tell the container which command it should run when it is started. With that, our Dockerfile is now ready. This is how it looks –

```
FROM python:3.8

# 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"]
```

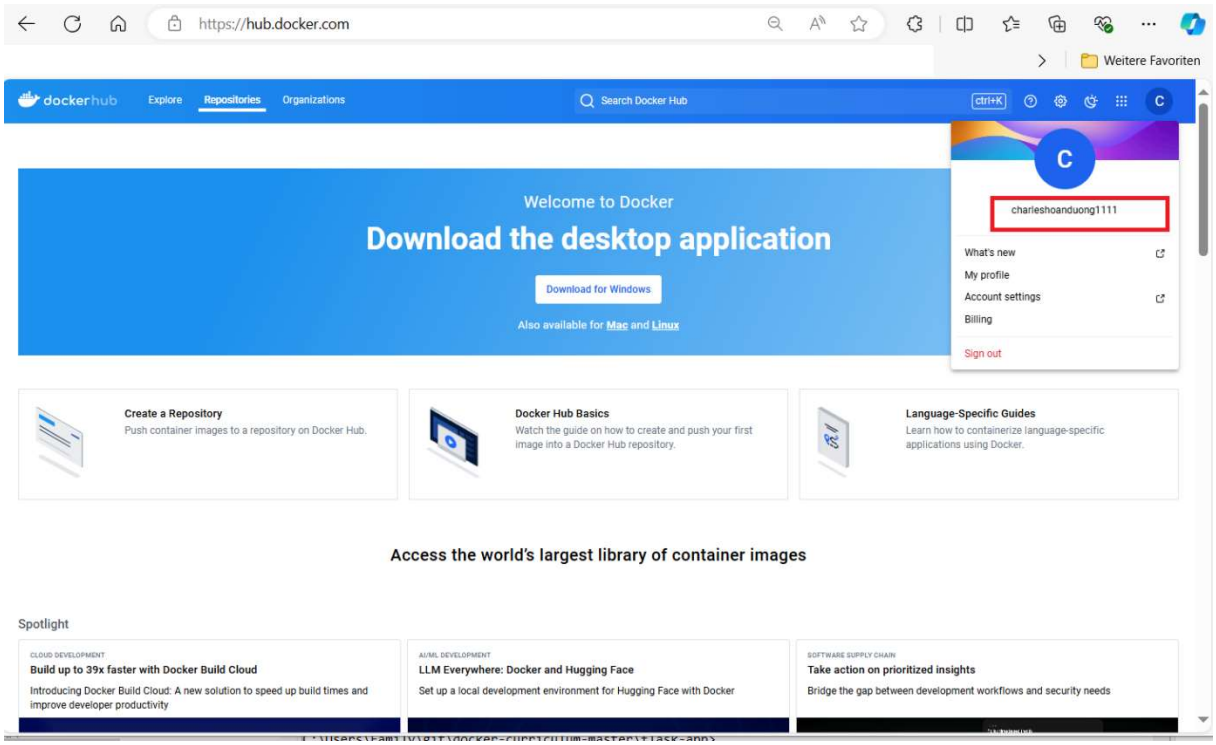

FROM python:3.8
set a directory for the app
WORKDIR /usr/src/app
copy all the files to the container
COPY . .
install dependencies
RUN pip install --no-cache-dir -r requirements.txt
define the port number the container should expose
EXPOSE 5000
run the command
CMD ["python", "./app.py"]

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/) →https://hub.docker.com/.



Login Docker Hub with my userid charleshoanduong1111



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 . → Sample: Should look as below

```
$ docker build -t yourusername/catnip .
Sending build context to Docker daemon 8.704 kB
Step 1 : FROM python:3.8
# 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
```

C:\Users\Family\git\docker-curriculum-master\flask-app>docker build -t yourusername/catnip .

```
C:\Users\Family\git\docker-curriculum-master\flask-app>docker build -t yourusername/catnip .
[+] Building 132.6s (9/9) FINISHED                                                                  docker:default
 => [internal] load build definition from Dockerfile                                                         0.1s
 => => transferring dockerfile: 340B                                                                         0.0s
 => => sha256:787c78da43830be6d988d34c7ee091f98d828516ce5478ca10a4933d655191bf 211.24MB / 211.24MB          98.3s
 => => sha256:5e0c543872f67eeba9c0deef12063485e241fa622be4b3f59602f5e28e1aa4f6 6.16MB / 6.16MB              38.1s
 => => extracting sha256:903681d87777d28dc56866a07a2774c3fd5bf65fd734b24c9d0ecd9a13c9f636                    4.4s
 => => extracting sha256:3cbbe86a28c2f6b3c3e0e8c6dcfba369e1ea656cf8daf69be789e0fe2105982b                    1.2s
 => => sha256:c85b6362d1e3707083ae57db029b0cce74c4401c2161e62a4ad1e304cbe7ea77 15.20MB / 15.20MB            52.0s
 => => sha256:71ef3208c9b5f7d3442c2ffdbafa874082a4c82b095e3d1f15890c7dd19e6fa3 230B / 230B                  52.5s
 => => sha256:f869877314cb6a0f7e74f38cc042710ea46a41e908bb53bb6fbb6b131f2d8c4b 2.70MB / 2.70MB              54.5s
 => => extracting sha256:6ed93aa58a52c9abc1ee472f1ac74b73d3adcccc2c30744498fd5f14f3f5d22c                    7.1s
 => => extracting sha256:787c78da43830be6d988d34c7ee091f98d828516ce5478ca10a4933d655191bf                   13.0s
 => => extracting sha256:5e0c543872f67eeba9c0deef12063485e241fa622be4b3f59602f5e28e1aa4f6                    0.6s
 => => extracting sha256:c85b6362d1e3707083ae57db029b0cce74c4401c2161e62a4ad1e304cbe7ea77                    1.0s
 => => extracting sha256:71ef3208c9b5f7d3442c2ffdbafa874082a4c82b095e3d1f15890c7dd19e6fa3                    0.0s
 => => extracting sha256:f869877314cb6a0f7e74f38cc042710ea46a41e908bb53bb6fbb6b131f2d8c4b                    0.5s
 => [internal] load build context                                                                            0.1s
 => => transferring context: 3.67kB                                                                          0.0s
 => [2/4] WORKDIR /usr/src/app                                                                              10.3s
 => [3/4] COPY . .                                                                                           0.1s
 => [4/4] RUN pip install --no-cache-dir -r requirements.txt                                                 5.0s
 => exporting to image                                                                                       0.3s
 => => exporting layers                                                                                      0.2s
 => => writing image sha256:e46a349721a3dd1aaa4251580006edb6bec1b54f3a909949f6397e5ae6db4c1d                 0.0s
 => => naming to docker.io/yourusername/catnip                                                               0.0s

 View a summary of image vulnerabilities and recommendations → docker scout quickview
```

Use my Docker Hub userid charleshoanduong1111 instead of yourusername

C:\Users\Family\git\docker-curriculum-master\flask-app>docker build -t charleshoanduong1111/catnip .

```
C:\Users\Family\git\docker-curriculum-master\flask-app>docker build -t charleshoanduong1111/catnip .
[+] Building 1.5s (9/9) FINISHED                                                                    docker:default
 => [internal] load build definition from Dockerfile                                                         0.0s
 => => transferring dockerfile: 340B                                                                         0.0s
 => [internal] load metadata for docker.io/library/python:3.8                                                1.2s
 => [internal] load .dockerignore                                                                            0.0s
 => => transferring context: 2B                                                                              0.0s
 => [1/4] FROM docker.io/library/python:3.8@sha256:5627587898d7946508ea314ca107a193a36606779e9d2ffa8d7d98a   0.0s
 => => resolve docker.io/library/python:3.8@sha256:5627587898d7946508ea314ca107a193a36606779e9d2ffa8d7d98a   0.0s
 => [internal] load build context                                                                            0.0s
 => => transferring context: 228B                                                                            0.0s
 => CACHED [2/4] WORKDIR /usr/src/app                                                                        0.0s
 => CACHED [3/4] COPY . .                                                                                    0.0s
 => CACHED [4/4] RUN pip install --no-cache-dir -r requirements.txt                                          0.0s
 => exporting to image                                                                                       0.0s
 => => exporting layers                                                                                      0.0s
 => => writing image sha256:e46a349721a3dd1aaa4251580006edb6bec1b54f3a909949f6397e5ae6db4c1d                 0.0s
 => => naming to docker.io/charleshoanduong1111/catnip                                                       0.0s

 View a summary of image vulnerabilities and recommendations → docker scout quickview
```

If you don't have the python:3.8 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)

C:\Users\Family\git\docker-curriculum-master\flask-app>python –version

```
C:\Users\Family\git\docker-curriculum-master\flask-app>python --version
Python 3.10.5
```

C:\Users\Family\git\docker-curriculum-master\flask-app>docker run -p 8888:5000 charleshoanduong1111/catnip

```
C:\Users\Family\git\docker-curriculum-master\flask-app>docker run -p 8888:5000 charleshoanduong1111/catnip
docker: Error response from daemon: driver failed programming external connectivity on endpoint gifted_albattani (218a882a7859843c114cf0a1de4536a2ba8
fc2abae1cf6cdc87487bd4859c647): Bind for 0.0.0.0:8888 failed: port is already allocated.
```

Use Administrator Command to delete the already allocated Port 8888.

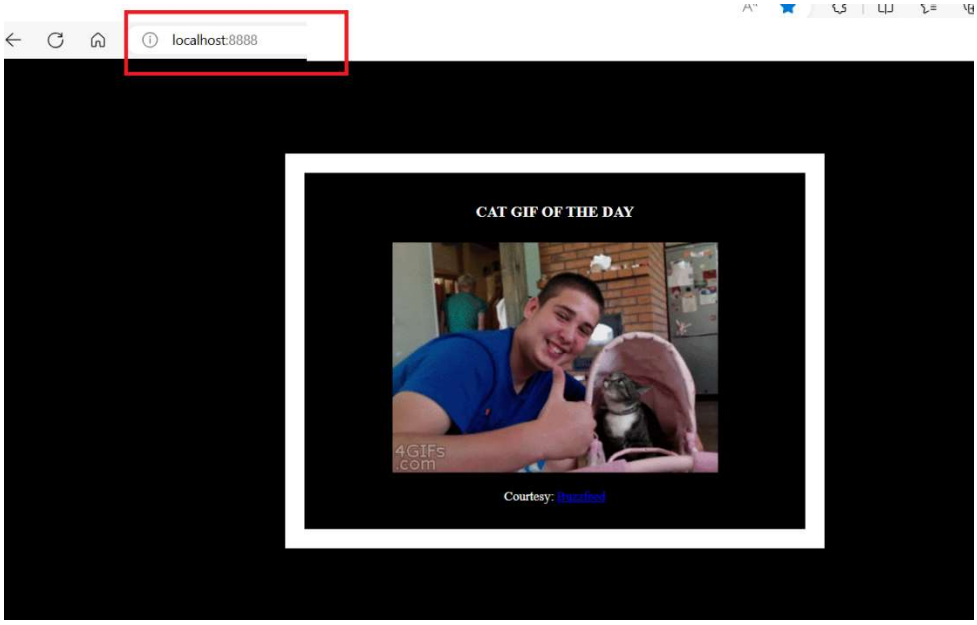
Use → netstat -ano | findstr :8888 → to find out the PID.

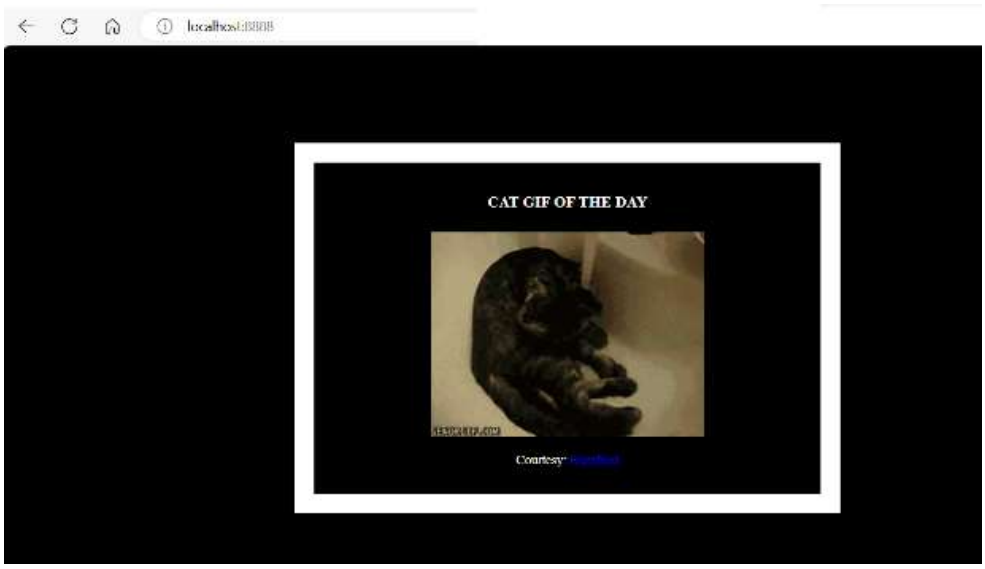
taskkill /PID 29756 /F → PID=29756 → kill the port 8888

```
Administrator: Eingabeaufforderung
Microsoft Windows [Version 10.0.19045.4651]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\WINDOWS\system32>
C:\WINDOWS\system32>netstat -ano | findstr :8888
  TCP    0.0.0.0:8888           0.0.0.0:0              ABHÖREN         29756
  TCP    [::]:8888              [::]:0                 ABHÖREN         29756
  TCP    [::1]:8888             [::]:0                 ABHÖREN         14172

C:\WINDOWS\system32>taskkill /PID 29756 /F
ERFOLGREICH: Der Prozess mit PID 29756 wurde beendet.

C:\WINDOWS\system32>
```

```
C:\Users\Family\git\docker-curriculum-master\flask-app>docker run -p 8888:5000 charleshoanduong1111/catnip
 * Serving Flask app 'app' (lazy loading)
 * Environment: development
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server ins
tead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 183-544-830
172.17.0.1 - - [14/Aug/2024 21:38:45] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [14/Aug/2024 21:39:02] "GET / HTTP/1.1" 200 -
```

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.

Docker on AWS

What good is an application that can't be shared with friends, right? So in this section we are going to see how we can deploy our awesome application to the cloud so that we can share it with our friends! We're going to use AWS Elastic Beanstalk to get our application up and running in a few clicks. We'll also see how easy it is to make our application scalable and manageable with Beanstalk!

Docker push

The first thing that we need to do before we deploy our app to AWS is to publish our image on a registry which can be accessed by AWS. There are many different Docker registries you can use (you can even host your own). For now, let's use Docker Hub to publish the image.

If this is the first time you are pushing an image, the client will ask you to login. Provide the same credentials that you used for logging into Docker Hub.

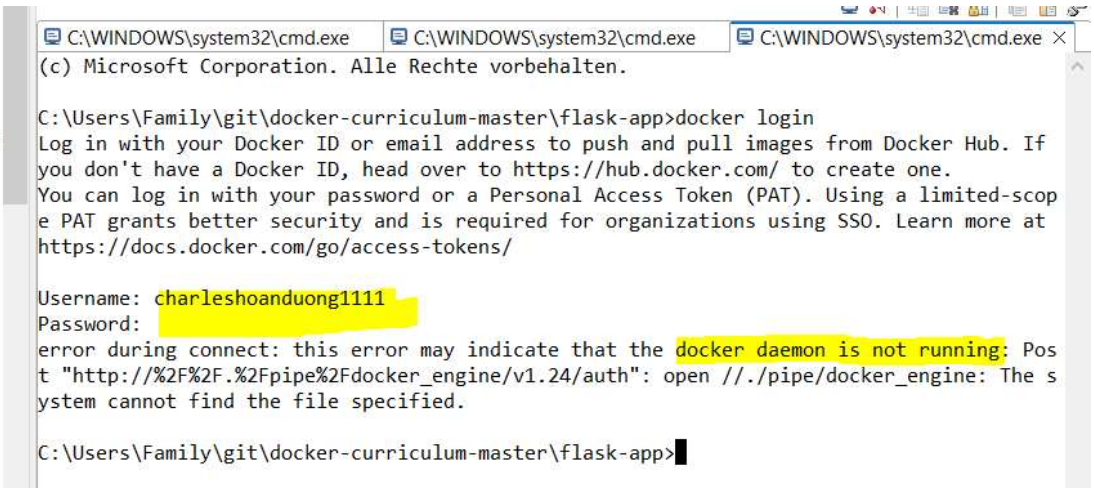
$ docker login

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

```
Username: yourusername

Password:

WARNING! Your password will be stored unencrypted in /Users/yourusername/.docker/config.json

Configure a credential helper to remove this warning. See

https://docs.docker.com/engine/reference/commandline/login/credential-store
```

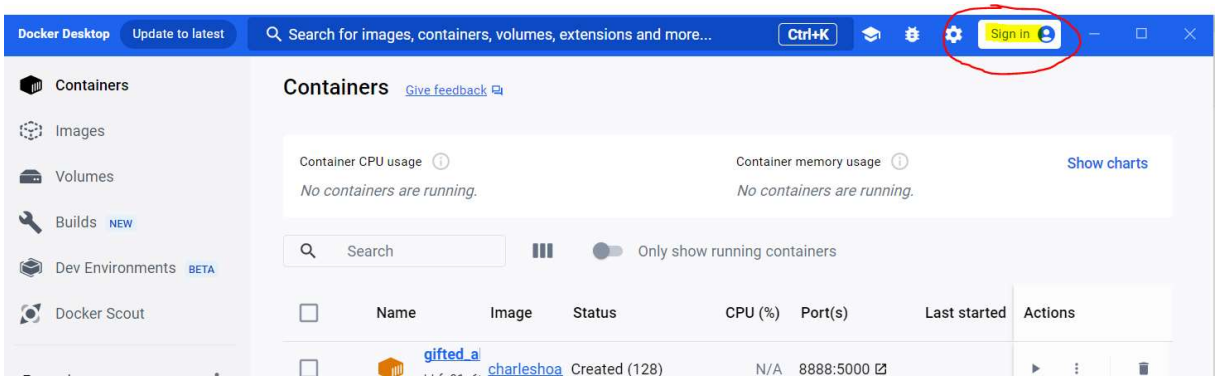
CLOSE my → Desktop Docker | Docker Engine → and try to run $ docker login

C:\Users\Family\git\docker-curriculum-master\flask-app>docker login
Username: charleshoanduong1111
Password: ****

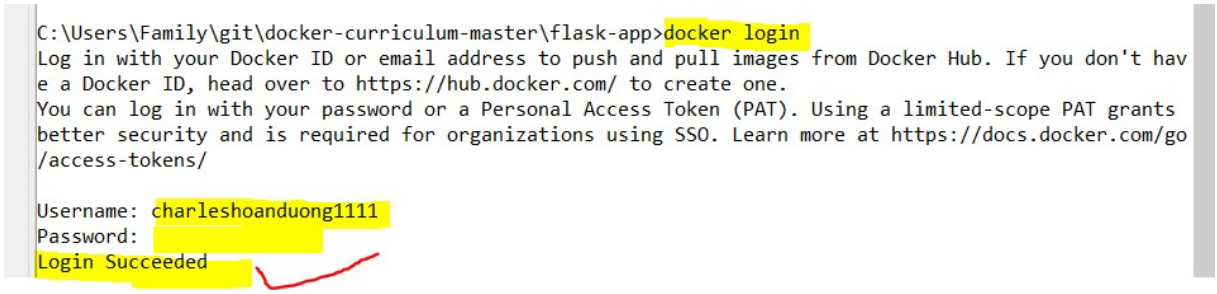


Login failed with message: "… docker daemon is not running…"

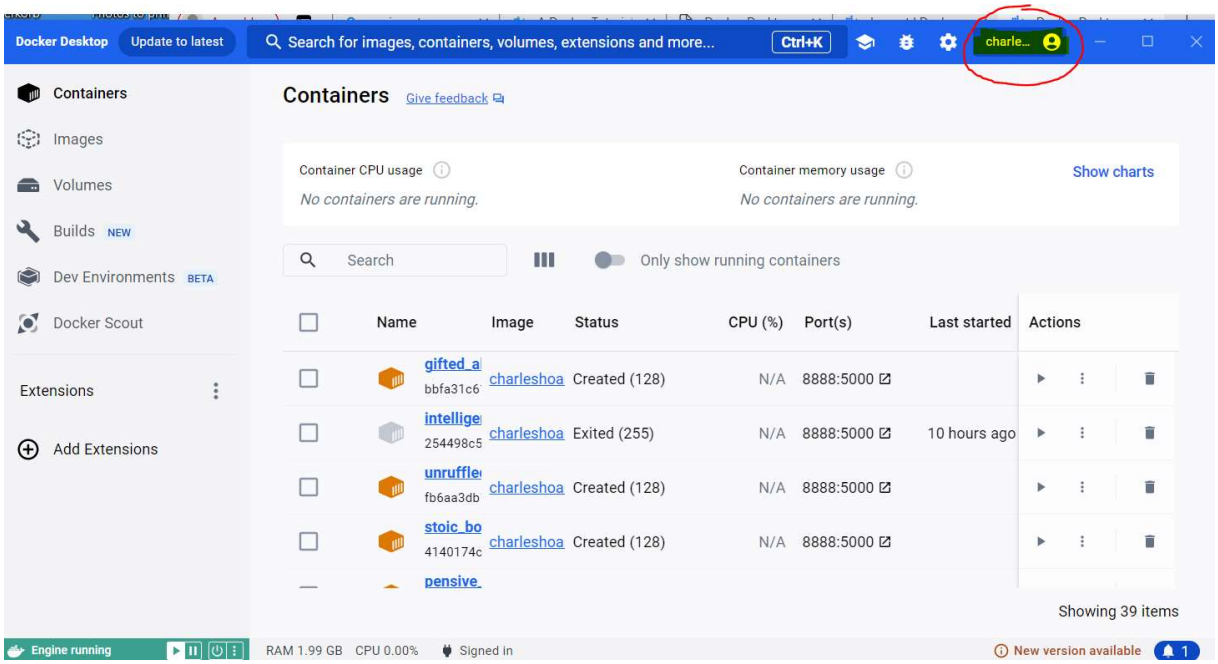
Open → Window OS → Desktop Docker | Docker Engine → without "Sign in" to Docker Hub



C:\Users\Family\git\docker-curriculum-master\flask-app>docker login
Username: charleshoanduong1111
Password: ****



Docker Desktop → Change "Sign in" to my userid "charleshoanduong1111"



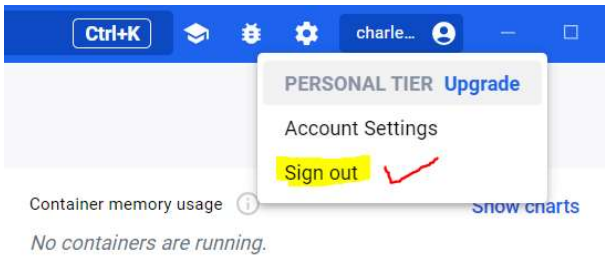
Docker Desktop → Now signed in with my userid “charleshoanduong1111”

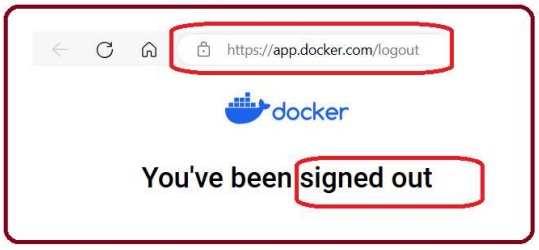
- Re-run → $docker login

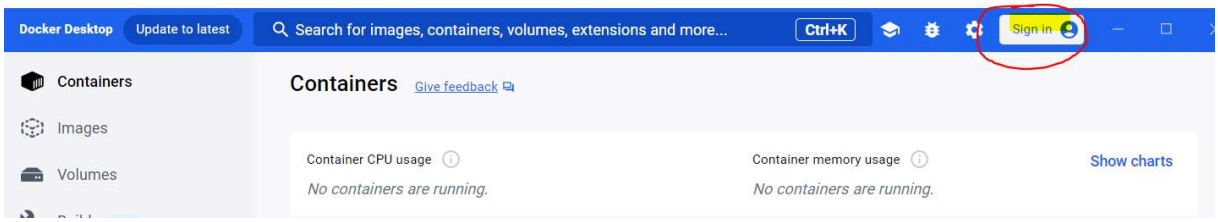
C:\Users\Family\git\docker-curriculum-master\flask-app>docker login

```
C:\Users\Family\git\docker-curriculum-master\flask-app>docker login
Authenticating with existing credentials...
Login Succeeded
```

Sign out → Desktop Docker | Docker Engine → and try to run $ docker login



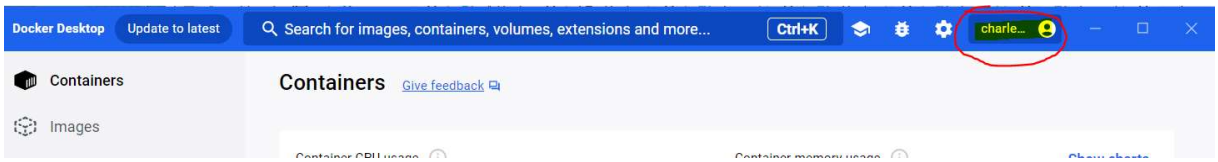




C:\Users\Family\git\docker-curriculum-master\flask-app>docker login

```
C:\Users\Family\git\docker-curriculum-master\flask-app>docker login
Log in with your Docker ID or email address to push and pull images from Docker Hub. If you don't hav
e a Docker ID, head over to https://hub.docker.com/ to create one.
You can log in with your password or a Personal Access Token (PAT). Using a limited-scope PAT grants
better security and is required for organizations using SSO. Learn more at https://docs.docker.com/go
/access-tokens/

Username: charleshoanduong1111
Password:
Login Succeeded
```

Docker Desktop → Change “Sign in” to my userid “charleshoanduong1111”

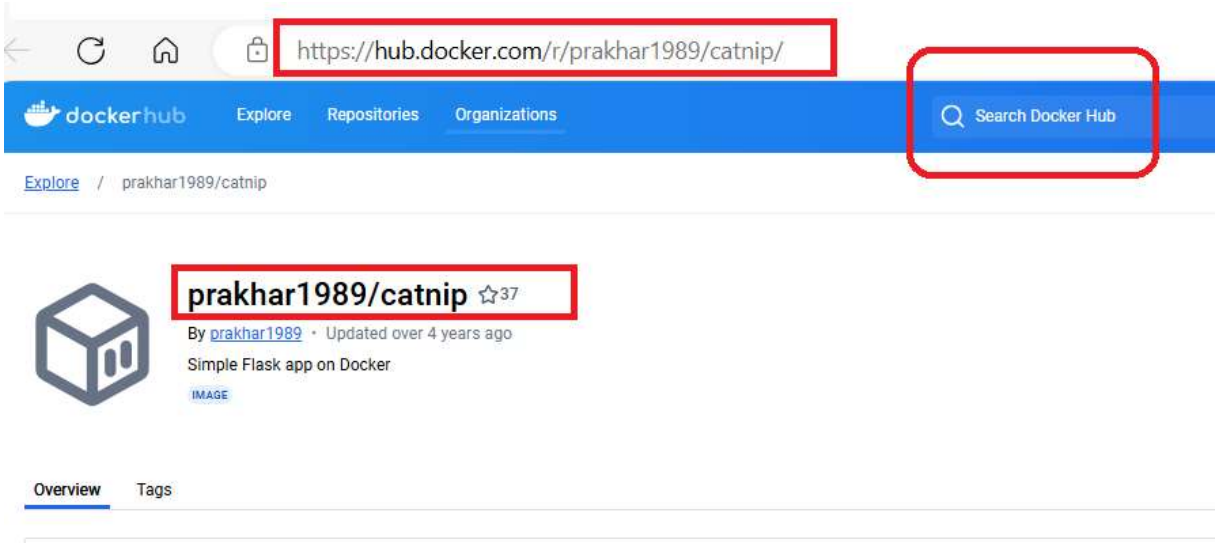


GREAT → Login to Docker Hub is Succeeded

To publish, just type the below command remembering to replace the name of the image tag above with yours. It is important to have the format of yourusername/image_name so that the client knows where to publish.

$ docker push yourusername/catnip

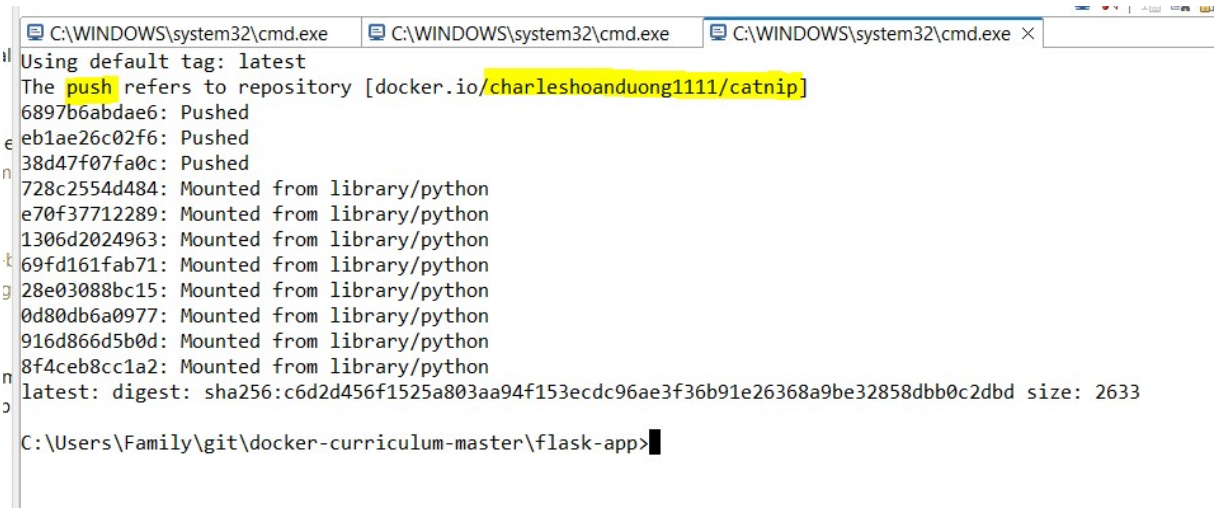
Once that is done, you can view your image on Docker Hub. For example, here's the web page for my image.



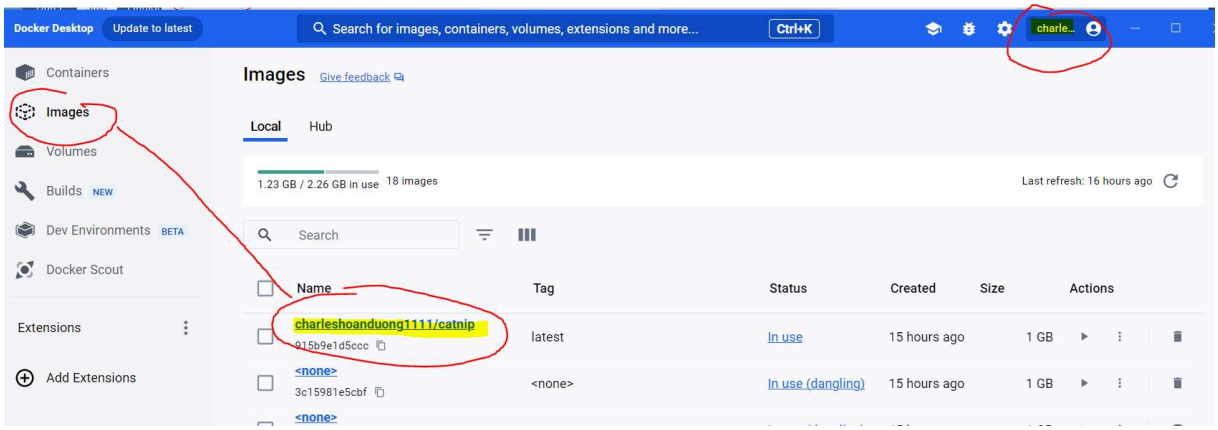
charleshoanduong1111/catnip image is not in my Docker Hub container.

- $Docker push

C:\Users\Family\git\docker-curriculum-master\flask-app>docker push charleshoanduong1111/catnip



charleshoanduong1111/catnip image is now in my Docker Hub container.



> *Note: One thing that I'd like to clarify before we go ahead is that it is not **imperative** to host your image on a public registry (or any registry) in order to deploy to AWS. In case you're writing code for the next million-dollar unicorn startup you can totally skip this step. The reason why we're pushing our images publicly is that it makes deployment super simple by skipping a few intermediate configuration steps.*

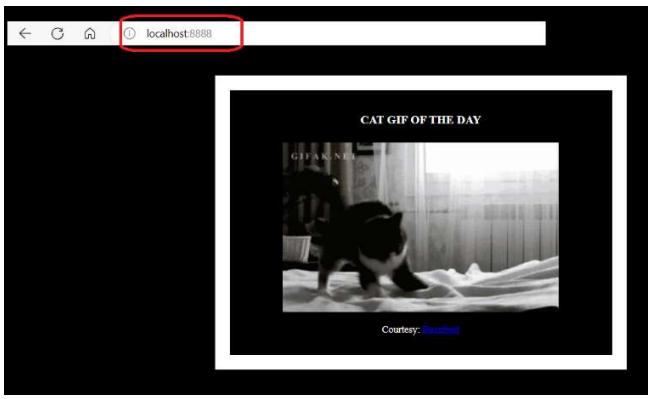
Now that your image is online, anyone who has docker installed can play with your app by typing just a single command.

```
$ docker run -p 8888:5000 yourusername/catnip
```

C:\Users\Family\git\docker-curriculum-master\flask-app>docker run -p 8888:5000 charleshoanduong1111/catnip

```
C:\Users\Family\git\docker-curriculum-master\flask-app>docker run -p 8888:5000 charleshoanduong1111/catnip
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server i
nstead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
172.17.0.1 - - [15/Aug/2024 12:42:13] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [15/Aug/2024 12:42:14] "GET /favicon.ico HTTP/1.1" 404 -
```

Web-browser → localhost:8888 → OK



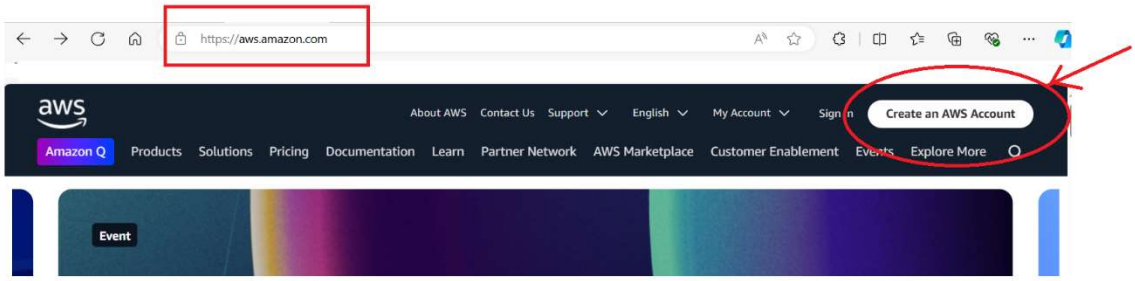
If you've pulled your hair out in setting up local dev environments / sharing application configuration in the past, you very well know how awesome this sounds. That's why Docker is so cool!

Beanstalk

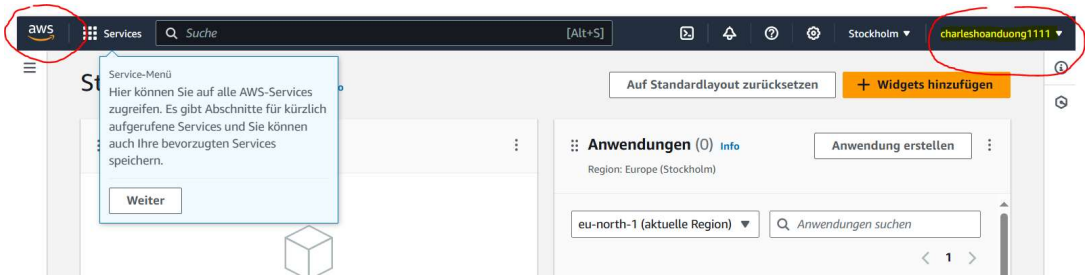
AWS Elastic Beanstalk (EB) is a PaaS (Platform as a Service) offered by AWS. If you've used Heroku, Google App Engine etc. you'll feel right at home. As a developer, you just tell EB how to run your app and it takes care of the rest - including scaling, monitoring and even updates. In April 2014, EB added support for running single-container Docker deployments which is what we'll use to deploy our app. Although EB has a very intuitive CLI, it does require some setup, and to keep things simple we'll use the web UI to launch our application.

To follow along, you need a functioning AWS account. If you haven't already, please go ahead and do that now - you will need to enter your credit card information. But don't worry, it's free and anything we do in this tutorial will also be free! Let's get started.

AWS webpace → https://aws.amazon.com/ → Click Create an AWS Account



AWS Account has been created with username: charleshoanduong1111

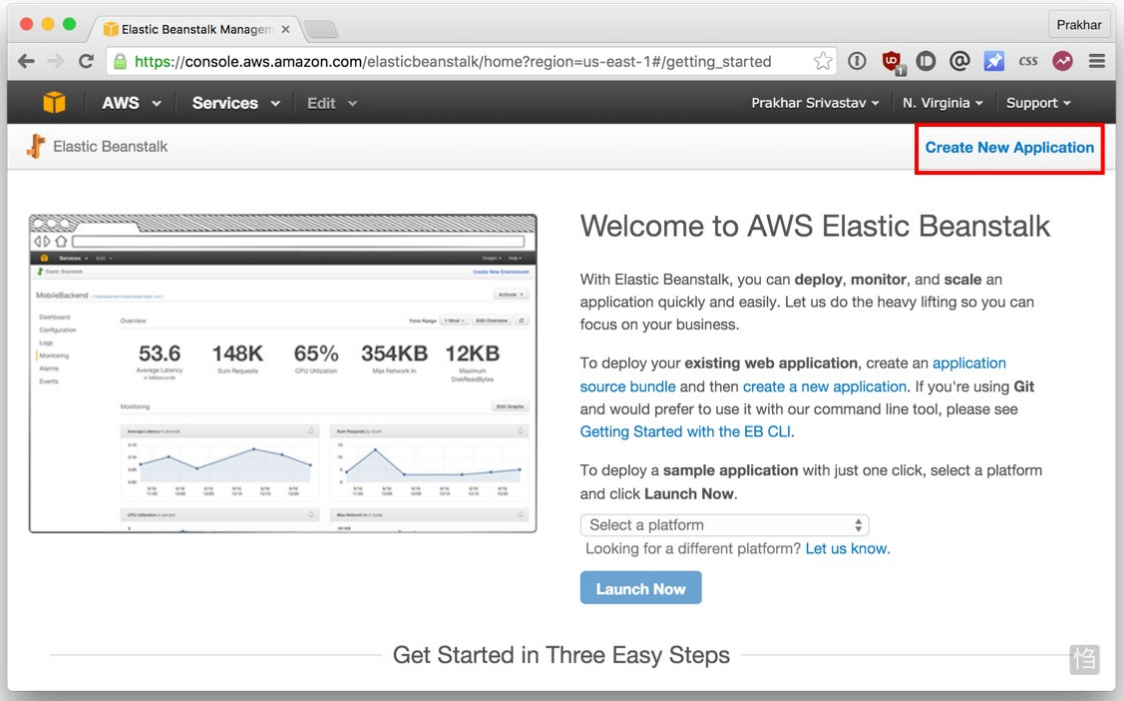


SAMPLE OF INSTRUCTION →

Here are the steps:

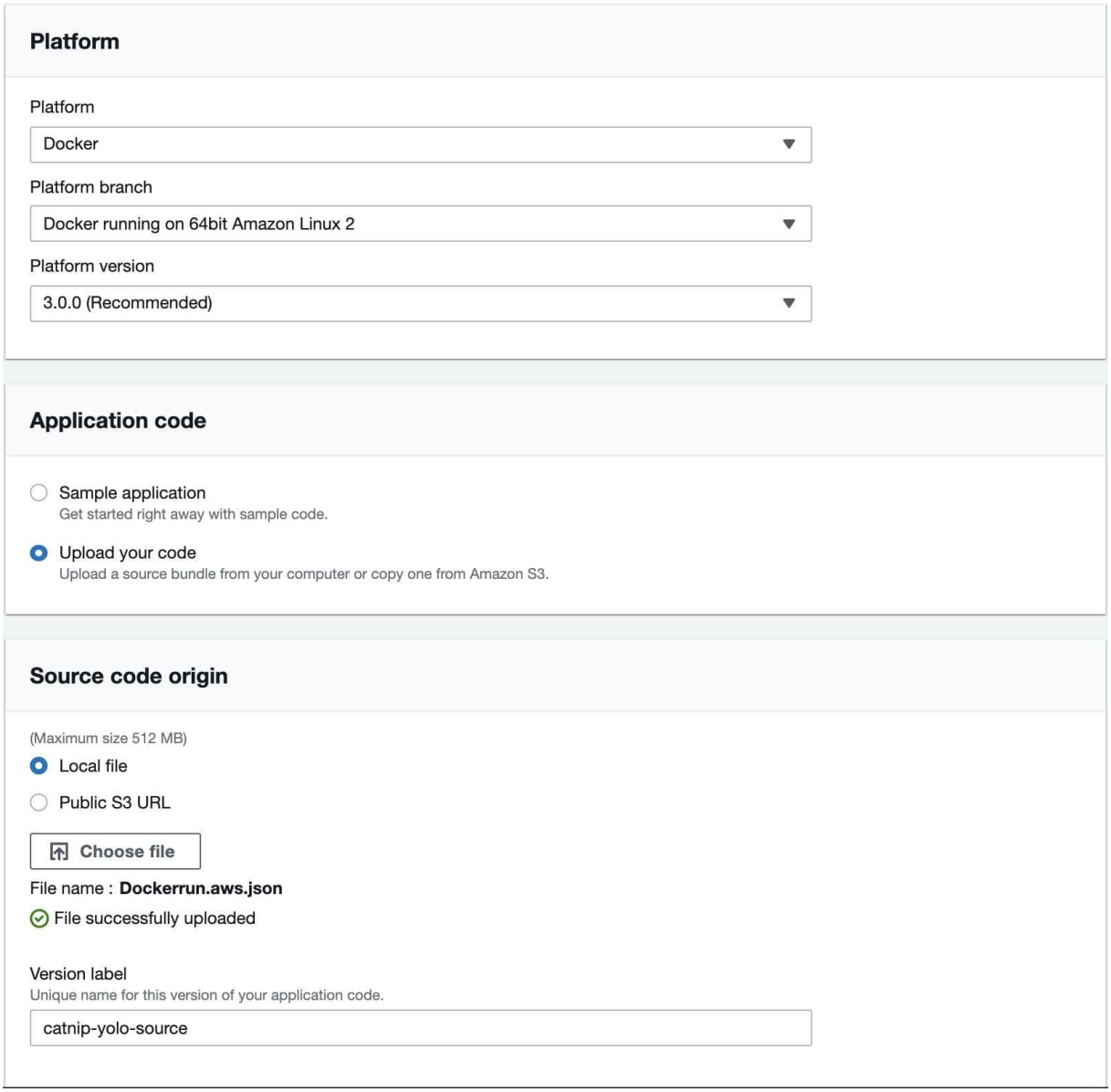
- Login to your AWS console.
- Click on Elastic Beanstalk. It will be in the compute section on the top left. Alternatively, you can access the Elastic Beanstalk console.

  ➔ https://eu-north-1.console.aws.amazon.com/elasticbeanstalk/home?region=eu-north-1#/welcome



- Click on "Create New Application" in the top right
- Give your app a memorable (but unique) name and provide an (optional) description
- In the **New Environment** screen, create a new environment and choose the **Web Server Environment**.
- Fill in the environment information by choosing a domain. This URL is what you'll share with your friends so make sure it's easy to remember.

  Under base configuration section. Choose *Docker* from the *predefined platform*.



- Now we need to upload our application code. But since our application is packaged in a Docker container, we just need to tell EB about our container. Open the Dockerrun.aws.json file located in the flask-app folder and edit the Name of the image to your image's name. Don't worry, I'll explain the contents of the file shortly. When you are done, click on the radio button for "Upload your Code", choose this file, and click on "Upload".

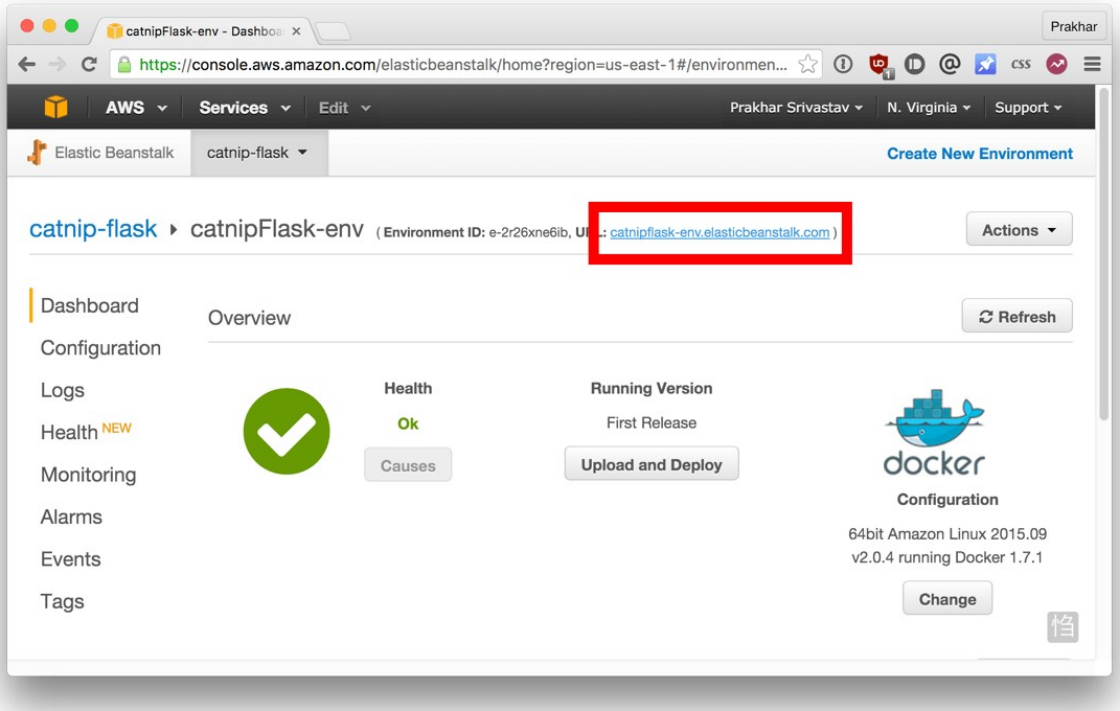
- Now click on "Create environment". The final screen that you see will have a few spinners indicating that your environment is being set up. It typically takes around 5 minutes for the first-time setup.

  While we wait, let's quickly see what the Dockerrun.aws.json file contains. This file is basically an AWS specific file that tells EB details about our application and docker configuration.

```
{
  "AWSEBDockerrunVersion": "1",
  "Image": {
    "Name": "prakhar1989/catnip",
    "Update": "true"
  },
  "Ports": [
    {
      "ContainerPort": 5000,
      "HostPort": 8000
    }
  ],
  "Logging": "/var/log/nginx"
}
```

The file should be pretty self-explanatory, but you can always reference the official documentation for more information. We provide the name of the image that EB should use along with a port that the container should open.

Hopefully by now, our instance should be ready. Head over to the EB page and you should see a green tick indicating that your app is alive and kicking.

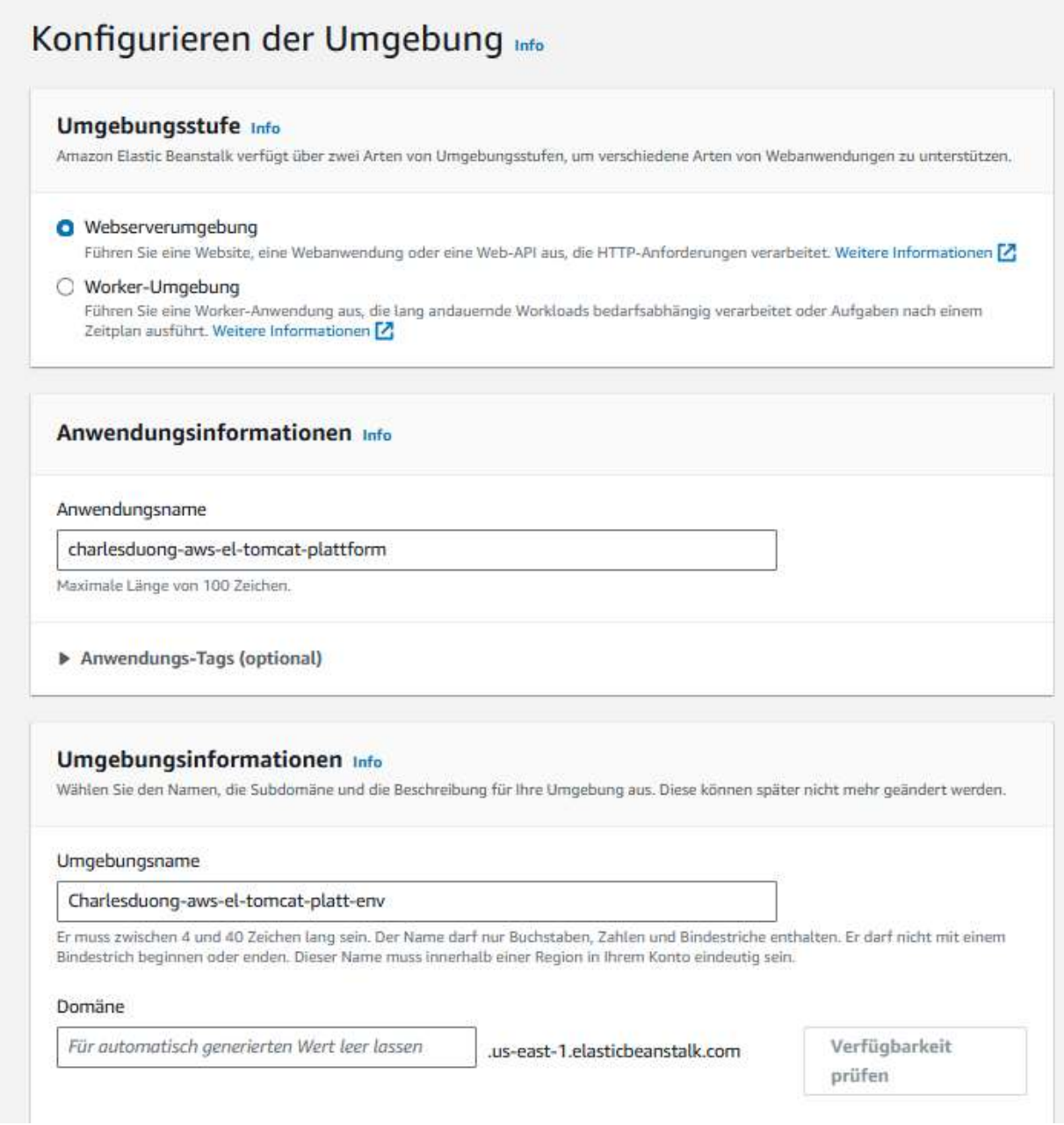


Go ahead and open the URL in your browser and you should see the application in all its glory. Feel free to email / IM / snapchat this link to your friends and family so that they can enjoy a few cat gifs, too.

Practice

AWS Account has been created with username: charleshoanduong1111.

1. Create a DEMO "charlesduong-aws-el-tomcat-plattform" used the following information and click "Create environment | German: **Absenden**"-Button



Plattform

Tomcat

Plattformverzweigung

Tomcat 10 with Corretto 21 running on 64bit Amazon Linux 2023

Plattformversion

5.3.1 (Recommended)

Anwendungscode Info

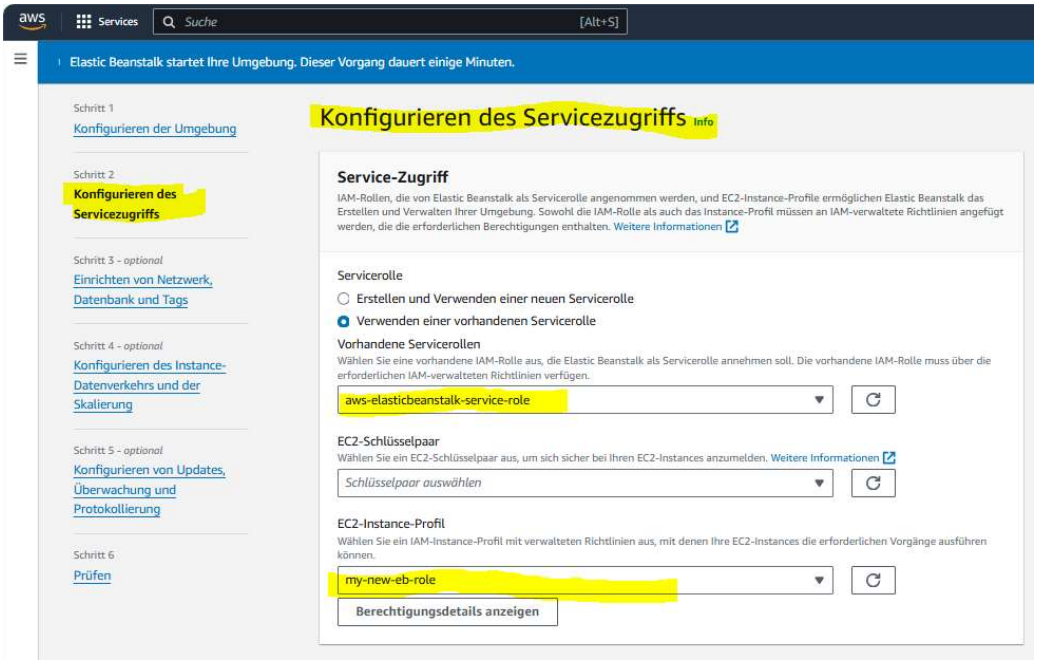
- (●) Beispielanwendung
- ( ) Vorhandene Version
  Anwendungsversionen, die Sie hochgeladen haben.
- ( ) Code hochladen
  Laden Sie ein Quell-Bündel von Ihrem Computer hoch oder kopieren Sie es aus Amazon S3.

Voreinstellungen Info

Beginnen Sie mit einer Voreinstellung, die Ihrem Anwendungsfall entspricht, oder wählen Sie eine benutzerdefinierte Konfiguration aus, um die empfohlenen Werte aufzuheben, und verwenden Sie die Standardwerte des Services.

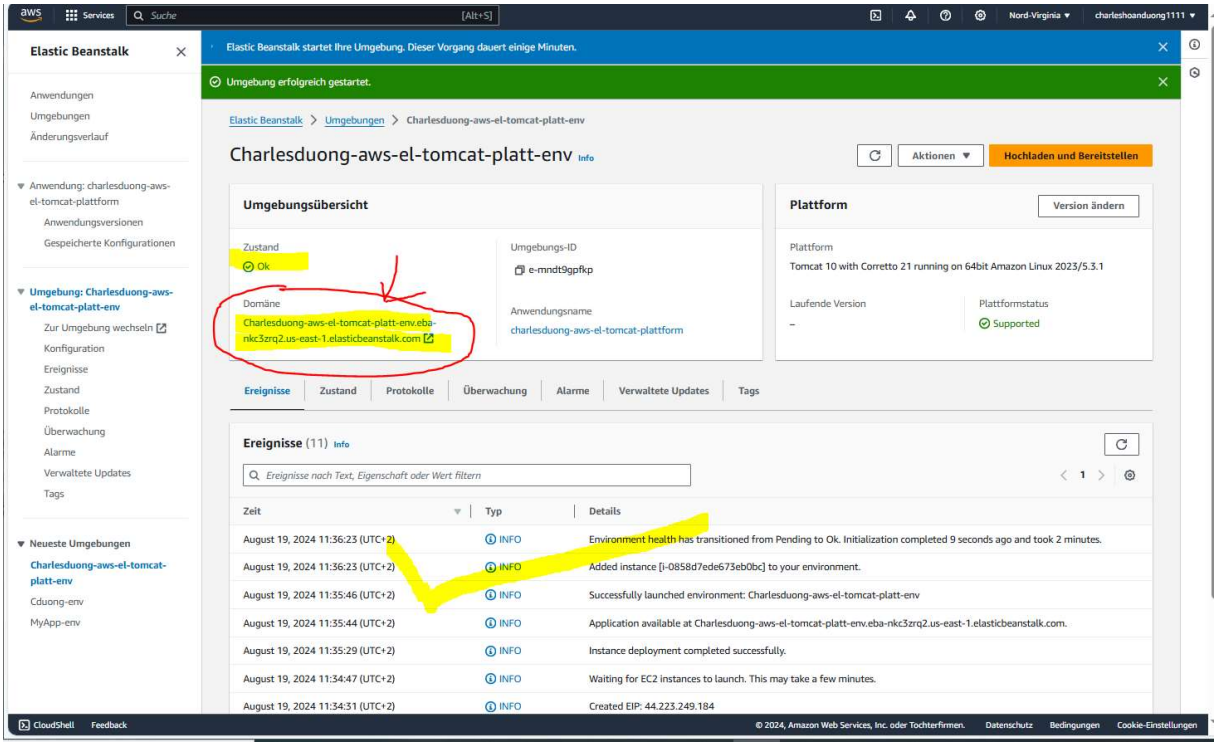
Konfigurationsvoreinstellungen

- (●) Single-Instance (kostenloses Kontingent verfügbar)
- ( ) Single-Instance (mit Spot Instance)
- ( ) Hohe Verfügbarkeit
- ( ) Hohe Verfügbarkeit (mithilfe von Spot- und On-Demand-Instances)
- ( ) Benutzerdefinierte Konfiguration

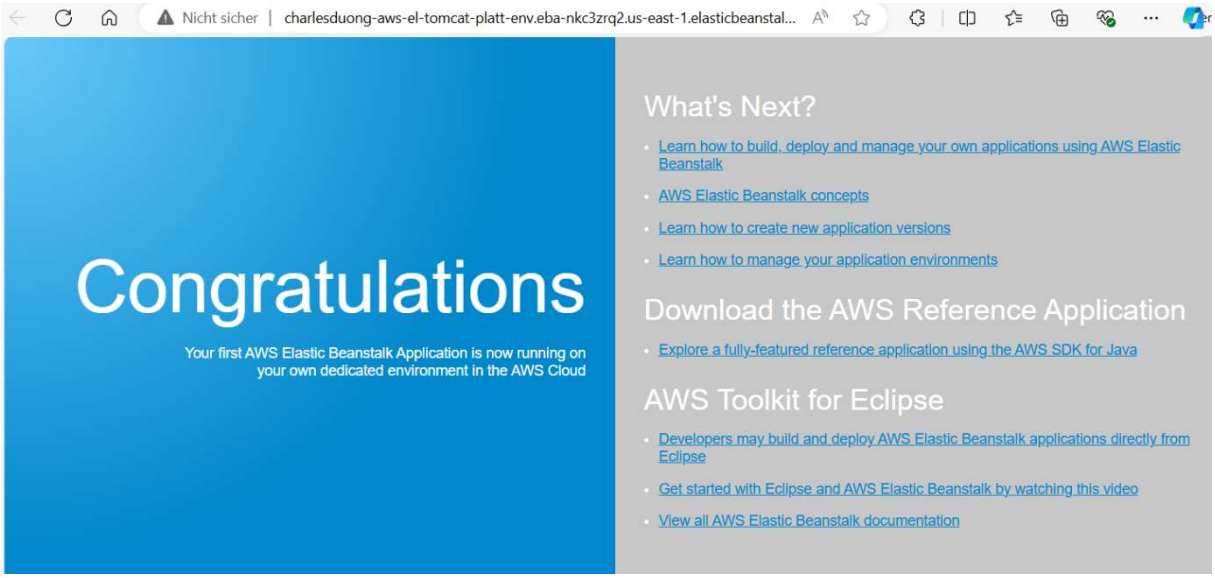


It works and click on Domain:

Domäne: Charlesduong-aws-el-tomcat-platt-env.eba-nkc3zrq2.us-east-1.elasticbeanstalk.com

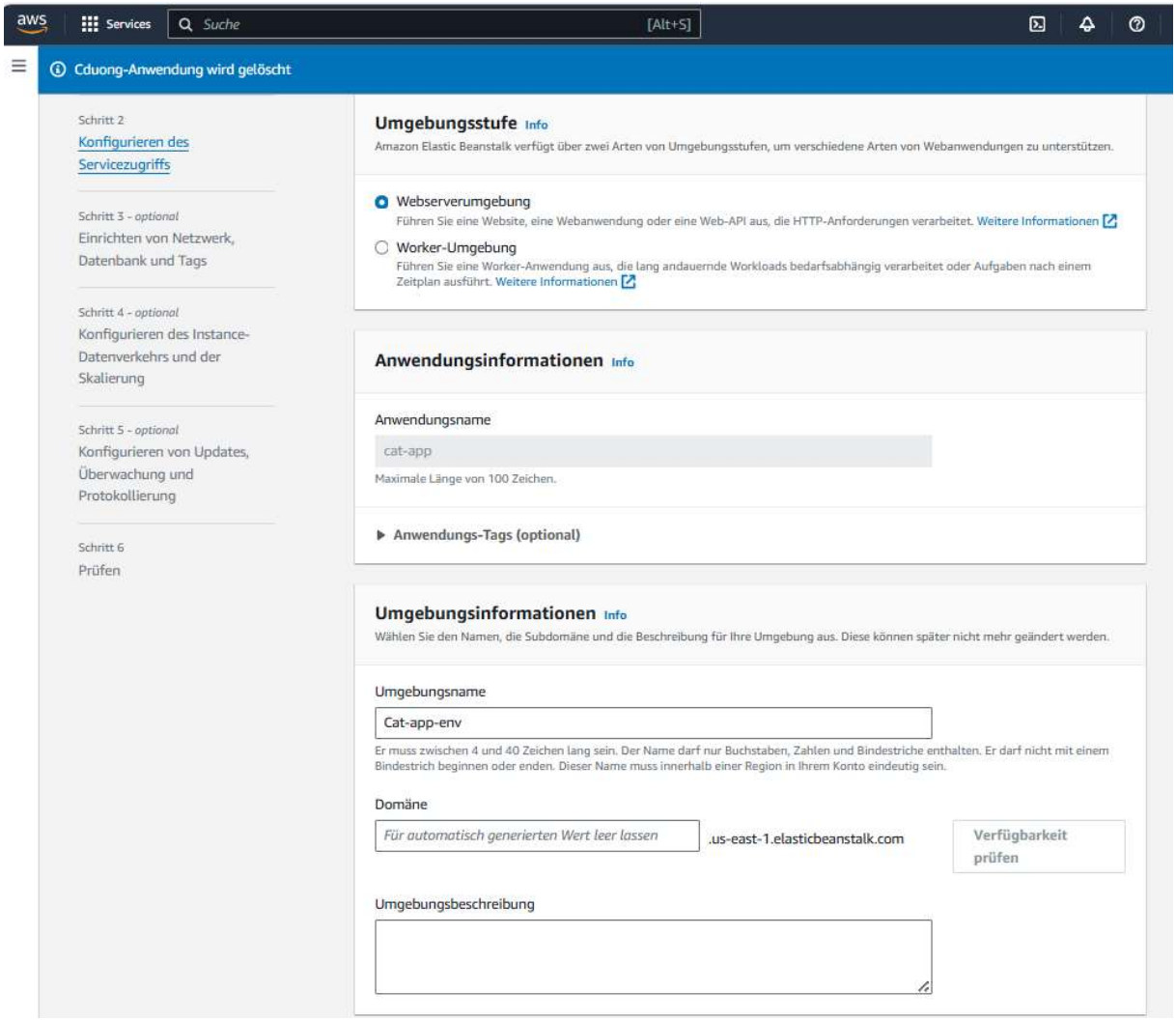


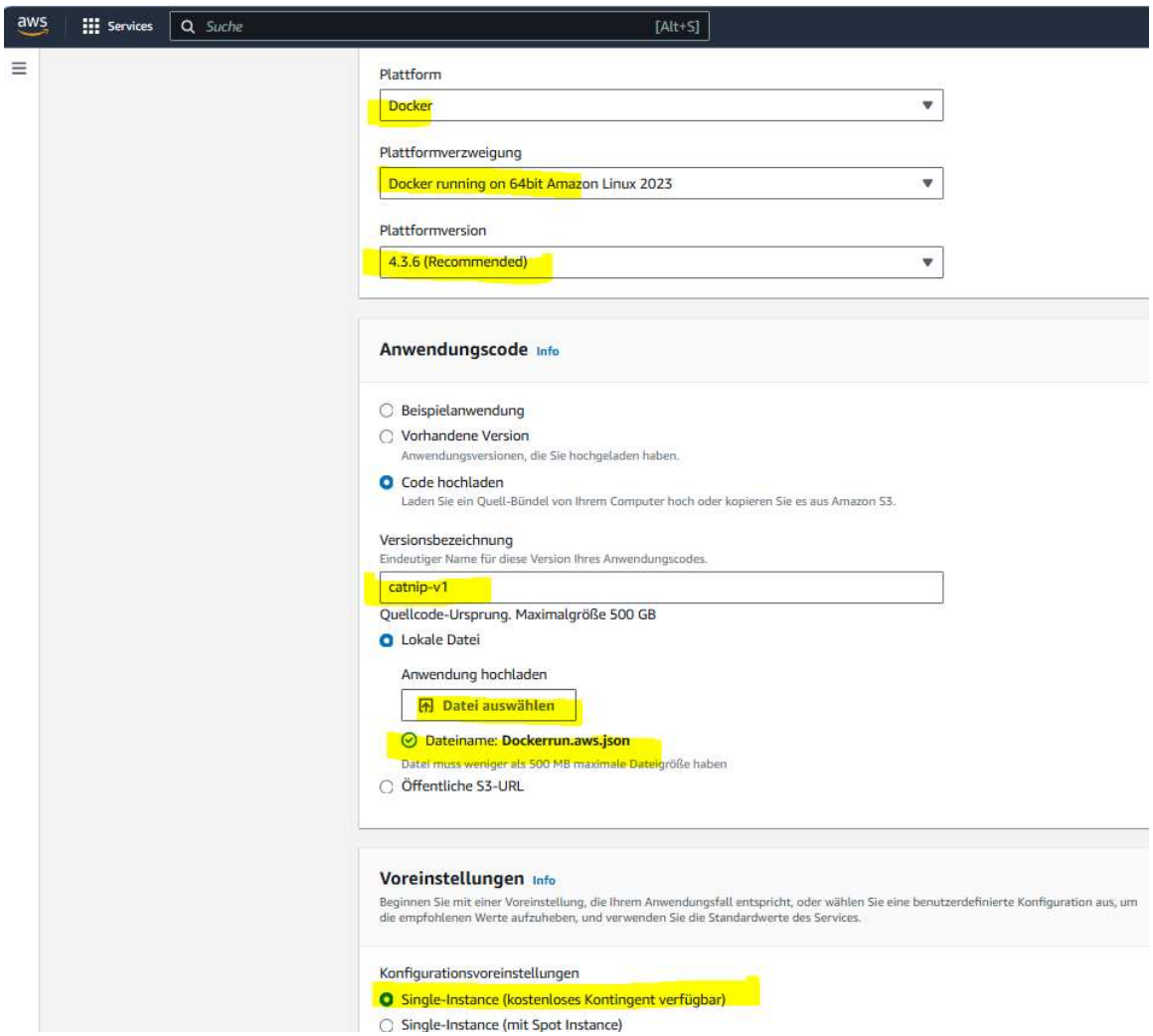
Charlesduong-aws-el-tomcat-platt-env.eba-nkc3zrq2.us-east-1.elasticbeanstalk.com shows as below:



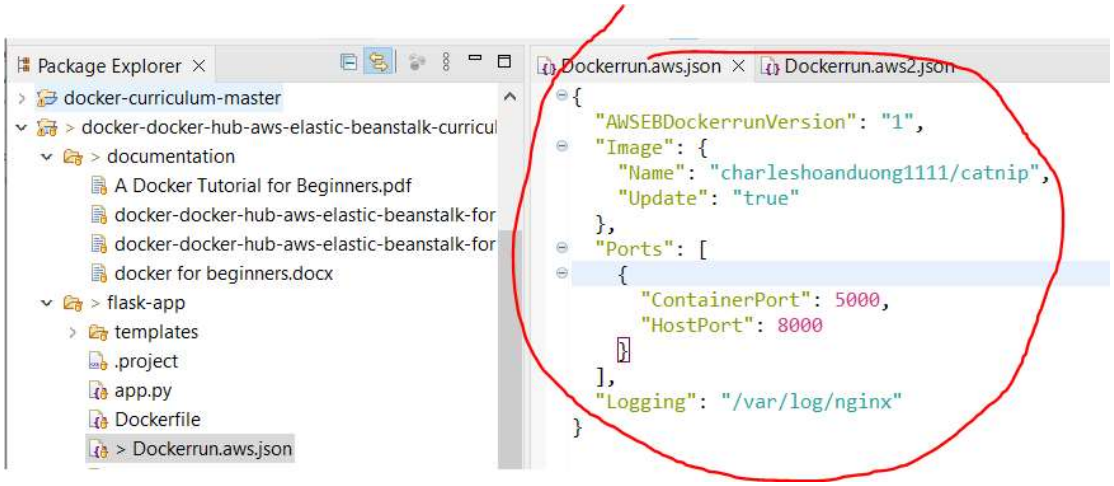
2. Create a

“Cat-app-env.eba-2qbnv7nm.us-east-1.elasticbeanstalk.com ” to push the Docker image from **Docker Hub** “charleshoanduong1111/catnip” to **AWS EB** by using the following information and click “**Absenden**”-Submit-Button

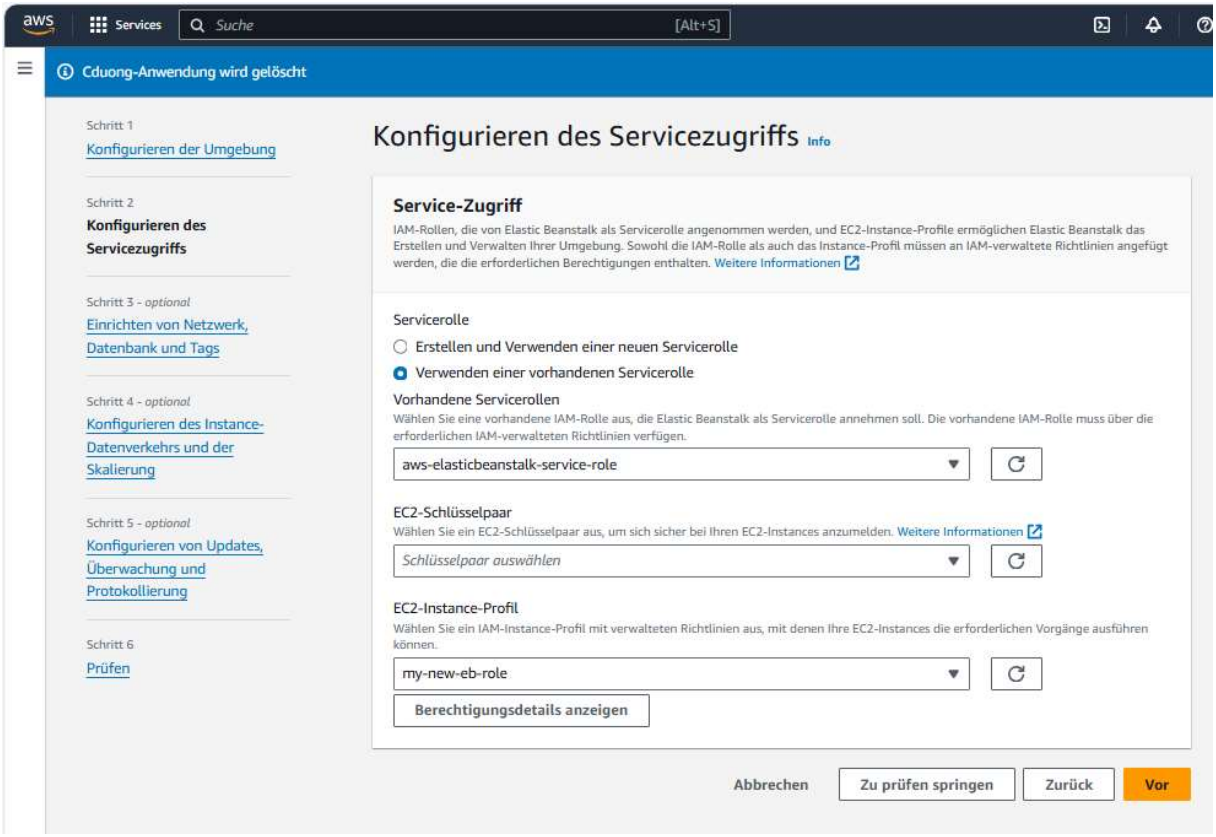




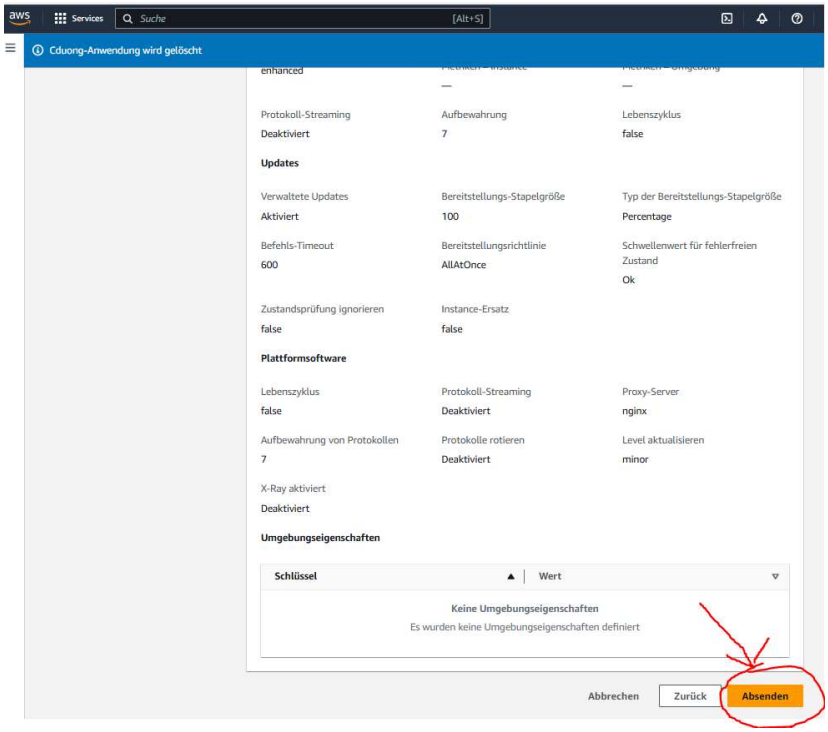
NOTE: Uploaded file info shows as below:



Click «zu prüfen springen» → to skip all the rest.

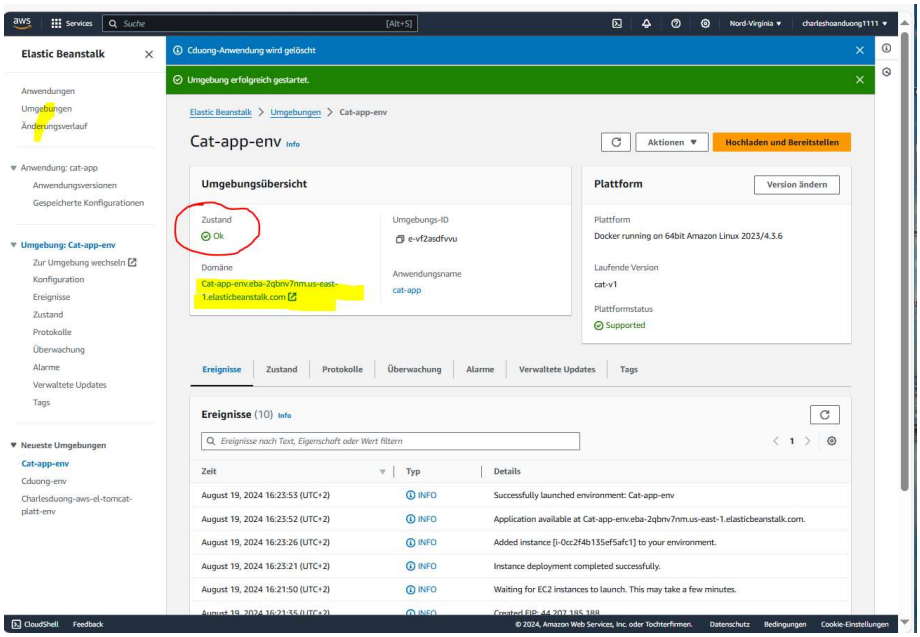


Click «Absenden» to submit.



It works and click on Domain:

Domäne: “Cat-app-env.eba-2qbnv7nm.us-east-1.elasticbeanstalk.com”



It works and the cat is running under:

Domäne: “Cat-app-env.eba-2qbnv7nm.us-east-1.elasticbeanstalk.com”