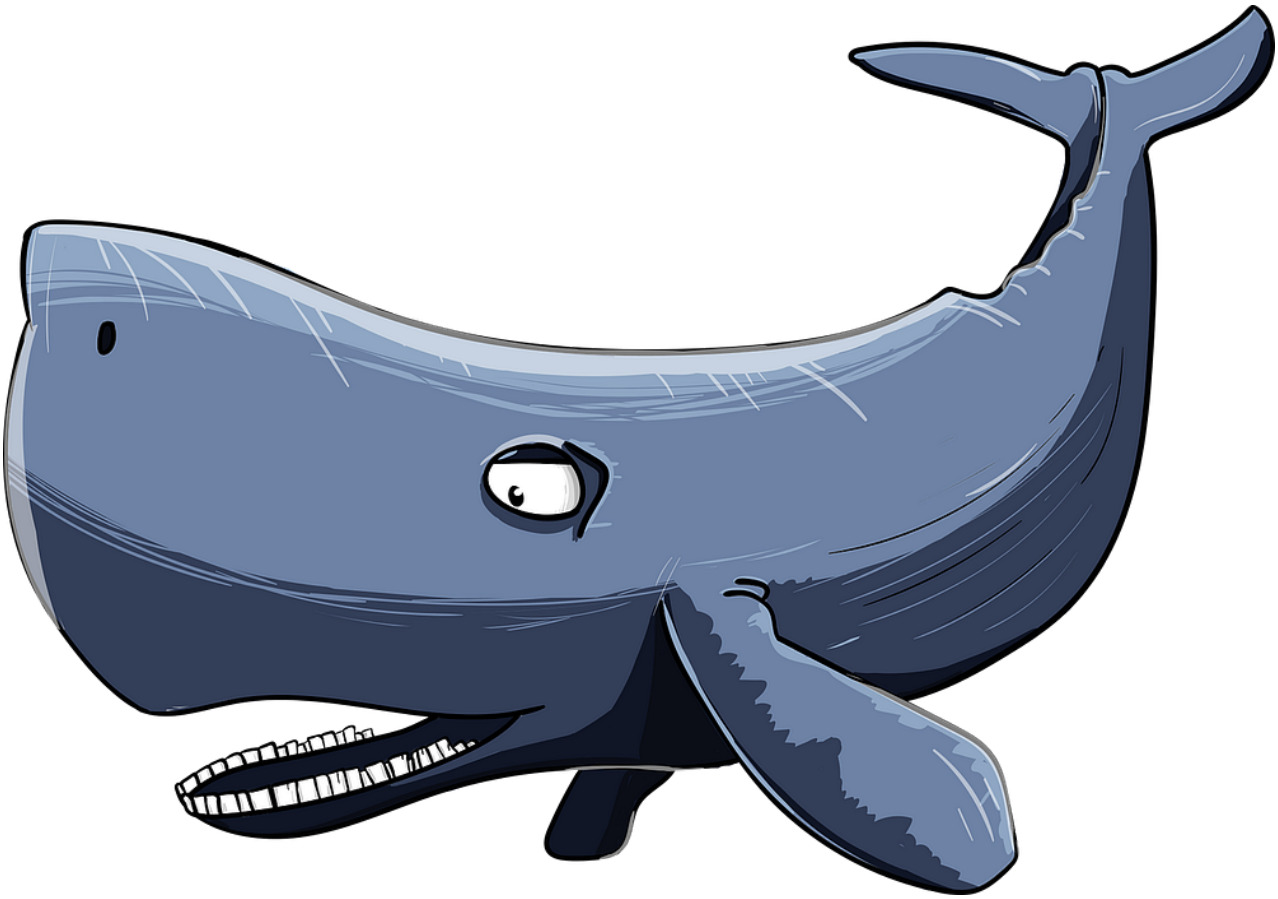
Docker + Flask | A Simple Tutorial

doedotdev.medium.com/docker-flask-a-simple-tutorial-bbcb2f4110b5

doedotdev April 1, 2018



Moby

This is a simple tutorial for getting started with Docker + Flask. There a lot of different methods I have found out there, but after going through my own struggles with it, I wanted to share the most concise version I could come up with. I hope this helps and as always the finished product/`code` can be found at the bottom of the page, no license, always free (claps are welcome) on my personal Github page.

You will learn how to create, run, build, push, pull, kill, prune and work in docker with flask as an api.

Setup Steps

Create a folder to hold the project. We will operate out of here for the most part. Use the `mkdir` command to create a folder.

```
$ mkdir hello_docker_flask
```

Navigate to that directory with `cd` .

```
$ cd hello_docker_flask
```

Make sure you have docker installed, the version is not particularly important as these basic commands are just about the same in all versions.

```
$ docker -vDocker version 17.12.0-ce, build c97c6d6
```

Don't have it installed? Here is the link to the docker official site, however you can use whatever method you like to install it.

Install Docker

docs.docker.com

Now that docker is ready lets see if you have any running containers.

```
$ docker psCONTAINER ID  IMAGE  COMMAND  CREATED  STATUS  PORTS  NAMES<I do not have have any running right now>
```

If you are just getting started, there shouldn't be any here. Either way it won't hurt to have another one running at the same time.

Have some currently running and want to `kill` them?

```
$ docker kill <CONTAINER ID>
```

You can also check to see if you have any containers even if they are not running.

```
$ docker imagesREPOSITORY              TAG     IMAGE ID      CREATED
SIZEmtngt/angular_docker  latest  ec5a8c5f01f1  2 hours ago   17MB
```

Again these won't hurt, but a good way to check to see what you already have (commands like this will come in handy later).

Want to clear out all the not running stuff as well?

```
$ docker system prune -a
```

That will delete everything you have in your local docker instance. So be careful.

Okay now that we know the basics, lets get started.

Creating the Files

First we need a simple flask file. I am going to make one that utilizes both `flask` and `flask_restful`.

Flask-RESTful - Flask-RESTful 0.3.6 documentation

flask-restful.readthedocs.io

Welcome | Flask (A Python Microframework)

flask.pocoo.org

At this point I am going to assume you know python basics and have python installed.

```
$ python --versionPython 2.7.6
```

Create a new file in your `hello_docker_flask` folder called app.py with some basic flask code in it.

app.py

```
   Flask   Resource, Apiapp = Flask(__name__)api = Api(app) (Resource):
get(self):         {'hello': 'world'}api.add_resource(HelloWorld, '/')if __name__
== '__main__':    app.run(debug=True, host='0.0.0.0')
```

You will notice the python file imports a few things. Although you might have them right now locally, the next person on the next machine won't. So we need to create a `requirements.txt` file to import them when our docker runs.

requirements.txt

```
flask
flask_restful
```

Now we need a `Dockerfile` in the same directory. It is just called `Dockerfile` , no extension, no suffix.

Dockerfile

```
# Dockerfile - this is a comment. Delete me if you want.FROM python:2.7COPY .
/appWORKDIR /appRUN pip install -r requirements.txtENTRYPOINT ["python"]CMD
["app.py"]
```

This `Dockerfile` copies our current folder, `.` , into our container folder `/app` . It sets that folder as the working directory, installs all our requirements with `pip install` from `requirements.txt`, and then runs the file using `python app.py`.

Thats it, these are the 3 files you need to get started. My `hello_docker_flask` folder looks like this.

```
hello_docker_flask|└──requirements.txt|└──Dockerfile|└──app.py
```

Docker Build

You should still be in your `hello_docker_flask` directory.

Now we can build our docker image.

```
$ docker build -t my_docker_flask:latest .
```

You will get a bunch of fancy output with loading bars, but what are looking for it it to end with the following confirmation.

```
Successfully built ddc23d92067eSuccessfully tagged my_docker_flask:latest
```

What does this do? We are building an image with the tag (`--tag` , `-t`) my_docker_flask:latest that includes everything in the current directory, `.` .

Want to know more about `tags`? This bloke is pretty passionate about it.

[The misunderstood Docker tag: latest]

[medium.com]

Now we can see what we created.

```
$ docker imagesREPOSITORY       TAG     IMAGE ID      CREATED
SIZEmy_docker_flask  latest  ddc23d92067e  45 seconds ago  687MB
```

But, is it running?

```
$ docker psCONTAINER ID  IMAGE  COMMAND  CREATED  STATUS  PORTS  NAMES
```

Nope! That is the next step.

Docker Run

You can run the build you just created with the `docker run` command.

```
$ docker run -d -p 5000:5000 my_docker_flask:latest
```

However this has a few pieces I think are important. First thing is `-d` which detatches from the run. This means you won't see any output. You can remove the `-d` if you would like to see the run process.

Next is `-p` which specifies the port it is going to run on. In out `app.py` file we used `app.run(debug=True, host='0.0.0.0')` so we needed to specify which port when using `flask run` , which above you can see I used 5000.

Okay, now we can see it is running!

```
$ docker psCONTAINER ID  IMAGE   COMMAND       CREATED   STATUS  PORTS9701
my_docker_flask python app.py 3 min ago Up 4 min 0.0.0.0:5000
```

I have shortened this output a bit because super long line snippets looks like awfulness on medium. ()

View Your Flask Restful Api

Now your api is running.

You can use a few methods to check it out.

Terminal

```
$ open -a "Google Chrome"  http://127.0.0.1:5000/
```

Or in your favorite browser go to http://127.0.0.1:5000/

Or use curl to do a get request.

```
$ curl http://127.0.0.1:5000/{"hello": "world"}
```

Or use Postman. A super great tool for working with endpoints you are constantly referencing and testing. You can save old requests and set up multiple environments.

Postman

www.getpostman.com

Push Your Image To Docker Hub

Now you can push your image to docker hub so you can pull it down later and use it, thats why docker is awesome.

Create an account over at docker hub.

https://hub.docker.com/

Then from your terminal window,

```
$ docker login -u mtngtPassword:Login Succeeded
```

But please use your username unless you can guess my password.

Now we re-tag the image with your username prefix, `<username>/`.

```
$ docker tag my_docker_flask mtngt/my_docker_flask
```

We could have named it this from the start, with the `<username>/` prefix. but I felt it was more clear to explain that is isn't necessary to do that until now.

And then you can push it to your docker hub.

```
$ docker push mtngt/my_docker_flask
```

This might take a minute but now `my_docker_flask` is available from your public docker hub. You can set this up private after the fact if you would like.

Pull Your Image From Docker Hub

Now lets pull down your image from docker hub. But first I want to prove that this works and we don't need all the stuff we have had previously.

So first, `kill` all the current running dockers.

```
$ docker psCONTAINER ID        IMAGE9701eed5868d        my_docker_flask:latest
```

Do it (Sheev Palpatine Voice).

```
$ docker kill 9701eed5868d
```

We have only stopped it. Remove it from our local docker instance completely.

```
$ docker system prune -a$docker imagesREPOSITORY   TAG   IMAGE   ID   CREATED
SIZE
```

All gone.

`Docker pull` the image you pushed.

```
$ docker pull mtngt/my_docker_flask
```

This might take some time and have a bit of output. But again, we are looking for `Status: Downloaded newer image for` mtngt`/my_docker_flask:latest`.

You have your image back!

```
$ docker imagesREPOSITORY                TAG    IMAGE ID      CREATED
SIZEmtngt/my_docker_flask  latest  ddc23d92067e 33 minutes ago  687MB
```

Now run like before, but remember, we have a new name. It is now prefixed with `<your_username>/`.

```
$ docker run -d -p 5000:5000 mtngt/my_docker_flask:latest
```

And it is live again!

```
$ docker psCONTAINER ID        IMAGE9e2c7644ee48
mtngt/my_docker_flask:latest
```

Curls for the gurls.

```
$ curl http://127.0.0.1:5000/{"hello": "world"}
```

Final Things To Remember

Now we have successfully created, ran, pushed, and pulled with docker.

However sometimes you want to edit. You can do this at anytime by making changes to the files and rebuilding the same way with `docker build` .

But remember, when running your new changes you will need to kill the current instance with `docker kill` , or specify a new port with `docker run -p <not the same port that is already running>` . This will avoid an error of the port already being in use.

Thanks!

Here is the code I promised.

mtngt/my_docker_flask

github.com

More to come with Docker + Angular and integrating those with the flask api with docker-compose, a way to link multiple dockers.