



Docker

Link script:

https://krspiced.pythonanywhere.com/chapters/project_pipeline/docker/README.html

Link teaching guide:

http://krspiced.pythonanywhere.com/teaching_guide/07_docker_pipeline/docker/

Goal

Run a container from a DOCKERFILE

Warm up: API

15": In groups of 2-3: pre-filled notebook **luftdaten_api_warmup.ipynb**

- Get sensor data from the luftdaten project (<https://sensor.community/en/>):
 - Pick one sensor IDs from as that interests you
 - Inspect the result of the response
 - Write a function to extract for the given sensor id:
 - time
 - P1 <- P10
 - P2 <- PM25
 - longitude
 - latitude

Question (from script)

A question: What does an operating system do?

MacOS, Linux, Window, even your phone has an operating system

A: They do the communication between the hardware and the software.

Hardware: screen, headphones, mouse, memory, network connection, etc. All of that is managed by the operating system. *If your computer is a flat: the operating system makes sure there is water, electricity and a door. Without it the flat is almost useless. The problem is that the configuration of those is sometimes a bit proprietary. For instance different apartments often have different power plugs, depending on the place you can drink the water in others you shouldn't. With doors sometimes we have a key, sometimes a chip, etc. The consequence: using that thing differs a bit. Back to the computer: if we want to install a program differs a bit in different operating system, installing Docker today is the best example. So if you write a program yourself you might have to take that into account. It's not so trivial to carry a program from one Os to another because there are different libraries installed.*

For a Software engineer this is a problem: they want to install something and they want things to work regardless of the computer or user. We want to make sure that some program like a python application or a Postgres installation works consistently.

For python, only python, there is already a solution.

Q: What is that? A: The virtual environments of anaconda.

Beyond python: If you also want to do other stuff, anaconda virtual environments don't help you. And this is where Docker comes in.

Docker

Image from script: the idea of a Docker container is the same as a physical container. We define containers that are

- **some kind of virtual environment** and I
- **install some libraries and the program that I want to run.**
- And then I take the containers and **plug them in the general Docker Engine which provides the generic properties** (power plug, water, keys) and connects it to your computer.
- And then you can **take this container and connect it to any machine, no matter the OS** as long as they are running Docker. **The cool thing is that I can do that many times with many different programs:** a database, python, etc, and they will be treated as different computers.

- So docker is simulating multiple computers inside your computer and it makes it in a very flexible and memory efficient way.



<https://www.weave.works/blog/a-practical-guide-to-choosing-between-docker-containers-and-vms>

Run a container

When you have docker you should be able to run:

```
docker run hello-world
```

If this works everything is going to work.

docker run : start a container in my computer.

Show all containers with status

```
docker ps -a
```

status: exited. Important to notice: in Docker when a process is finished, the container is stopped!

Why did the python container exit? It didn't have to do anything.

```
docker run python
```

```
docker ps
```

Shows all the running containers

```
docker ps -a
```

Show all containers with status.

Let's start a python container:

```
docker run -it -d --name habanero python
```

- `-it,` container in interactive mode
- `-d:` the container keeps running in the background

I started a virtual "computer" in the background and that computer is running python.

Q: What happens?

Docker looks for the image locally. If it's not found, downloads it. Then starts the container.

Let's interact with the container:

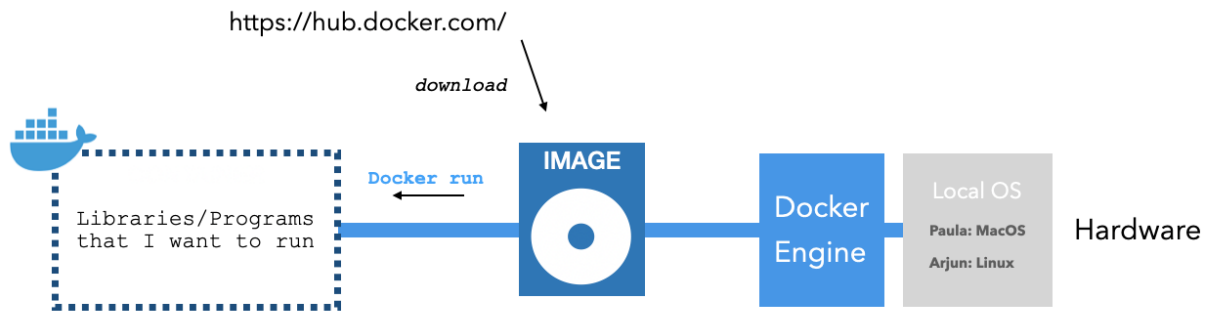
```
docker attach habanero
```

And then we get a standard python prompt. *"I have found myself a complicated way to start a python terminal" (-KR).*

```
from random import choices

students = ['Alisa', 'Catarina', 'Kiran', 'LiLi', 'Marko', 'Raphael', 'Rebecca', 'Reem', 'Simon']
choices(students)
```

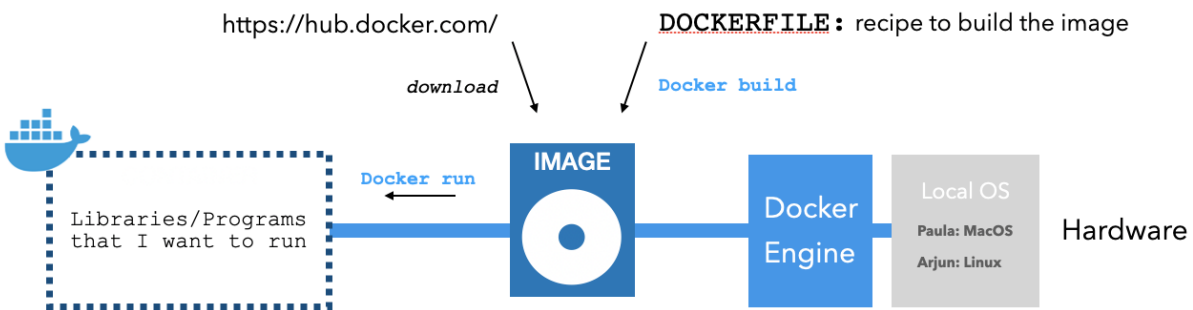
`ctrl+d` to exit the session. You could as well type the exit command.



Docker Images

`docker images`

- So, how did the docker container know how to install? The python == image. From Docker hub
- Or Alternative, we can define our own image with a Dockerfile



Dockerfile

Now the "python" that we were specifying is a docker image and we can build our own image with a docker script in a **Dockerfile**

```
# Use an official Python runtime as a parent image
FROM python:3.6-slim
```

```
# Set the working directory to /app
WORKDIR /my_folder

# Copy the requirements file into the container at /app
# (a text files with all the libraries you want to install)
COPY . /my_folder

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Run app.py when the container launches
CMD ["python", "luftdaten.py"]
```

The requirement.txt file can be:

```
requests==2.25.1
```

Now if we create a small python file like:

```
touch luftdaten.py
code luftdaten.py
```

```
import requests
import time

SENSOR_URL = "http://api.luftdaten.info/static/v1/sensor/{}/"

def pick_luftdaten_values(sensor_id):
    result = requests.get(SENSOR_URL.format(sensor_id))
    data = result.json()
    time_stamp = data[0]['timestamp']
    PM25 = data[0]['sensordatavalues'][0]['value']
    PM10 = data[0]['sensordatavalues'][1]['value']
    return time_stamp, PM25, PM10

if __name__ == '__main__':
    while True:
        time_stamp, PM25, PM10 = pick_luftdaten_values(21124)
        print(f'Air quality data at {time_stamp}: PM25 = {PM25}, PM10 = {PM10}')
        time.sleep(10)
```

And we build the image with:

```
docker build -t luftdaten_image .
```

And create the container:

```
docker run -it -d --name luftdaten_container luftdaten_image
```

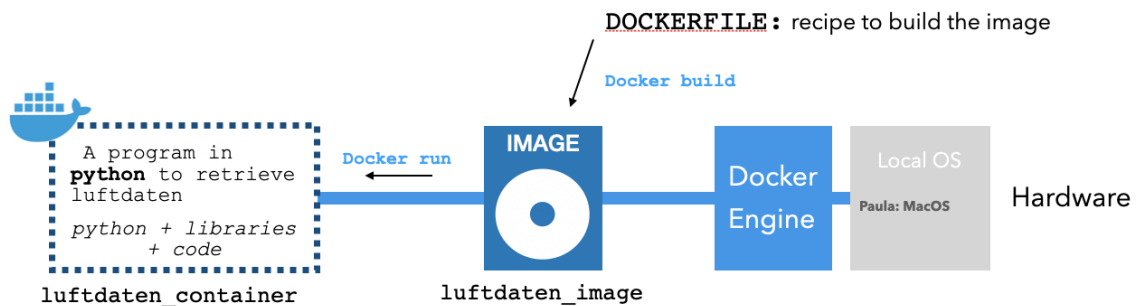
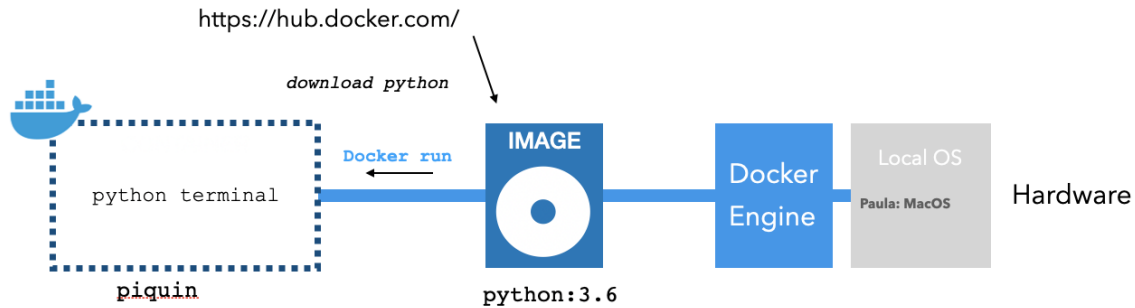
(If we wanted to mount a volume...)

```
docker run -it -v $PWD:/app/ spiced_py -t
```

For the project we have to add tweepy to the requirements, add the python file `get_tweets.py` and `credentials.py` with ADD.

WRAP UP

So what have we done? WE have taken our python program and have made a generic installation file that can run EVERYWHERE. This Dockerfile works everywhere, regardless of if python is inside, the only thing we need is to have Docker installed.



Useful Docker commands

List all containers (only IDs)

```
docker ps -aq
```

Stop all running containers


```
docker stop $(docker ps -aq)
```

Remove all containers

```
docker rm $(docker ps -aq)
```

Remove all images


```
docker rmi $(docker images -q)
```

 Cheatsheet for Docker CLI			
Run a new Container	Manage Containers	Manage Images	Info & Stats
<p>Start a new Container from an Image</p> <pre>docker run IMAGE</pre> <pre>docker run nginx</pre> <p>...and assign it a name</p> <pre>docker run --name CONTAINER IMAGE</pre> <pre>docker run --name web nginx</pre> <p>...and map a port</p> <pre>docker run -p HOSTPORT:CONTAINERPORT IMAGE</pre> <pre>docker run -p 8080:80 nginx</pre> <p>...and map all ports</p> <pre>docker run -P IMAGE</pre> <pre>docker run -P nginx</pre> <p>...and start container in background</p> <pre>docker run -d IMAGE</pre> <pre>docker run -d nginx</pre> <p>...and assign it a hostname</p> <pre>docker run --hostname HOSTNAME IMAGE</pre> <pre>docker run --hostname srv nginx</pre> <p>...and add a dns entry</p> <pre>docker run --add-host HOSTNAME:IP IMAGE</pre> <p>...and map a local directory into the container</p> <pre>docker run -v HOSTDIR:TARGETDIR IMAGE</pre> <pre>docker run -v ~/.usr/share/nginx/html nginx</pre> <p>...but change the entrypoint</p> <pre>docker run -it --entrypoint EXECUTABLE IMAGE</pre> <pre>docker run -it --entrypoint bash nginx</pre>	<p>Show a list of running containers</p> <pre>docker ps</pre> <p>Show a list of all containers</p> <pre>docker ps -a</pre> <p>Delete a container</p> <pre>docker rm CONTAINER</pre> <pre>docker rm web</pre> <p>Delete a running container</p> <pre>docker rm -f CONTAINER</pre> <pre>docker rm -f web</pre> <p>Delete stopped containers</p> <pre>docker container prune</pre> <p>Stop a running container</p> <pre>docker stop CONTAINER</pre> <pre>docker stop web</pre> <p>Start a stopped container</p> <pre>docker start CONTAINER</pre> <pre>docker start web</pre> <p>Copy a file from a container to the host</p> <pre>docker cp CONTAINER:SOURCE TARGET</pre> <pre>docker cp web:/index.html index.html</pre> <p>Copy a file from the host to a container</p> <pre>docker cp TARGET CONTAINER:SOURCE</pre> <pre>docker cp index.html web:/index.html</pre> <p>Start a shell inside a running container</p> <pre>docker exec -it CONTAINER EXECUTABLE</pre> <pre>docker exec -it web bash</pre> <p>Rename a container</p> <pre>docker rename OLD_NAME NEW_NAME</pre> <pre>docker rename 096 web</pre> <p>Create an image out of container</p> <pre>docker commit CONTAINER</pre> <pre>docker commit web</pre>	<p>Download an image</p> <pre>docker pull IMAGE[:TAG]</pre> <pre>docker pull nginx</pre> <p>Upload an image to a repository</p> <pre>docker push IMAGE</pre> <pre>docker push myimage:1.0</pre> <p>Delete an image</p> <pre>docker rmi IMAGE</pre> <p>Show a list of all Images</p> <pre>docker images</pre> <p>Delete dangling images</p> <pre>docker image prune</pre> <p>Delete all unused images</p> <pre>docker image prune -a</pre> <p>Build an image from a Dockerfile</p> <pre>docker build DIRECTORY</pre> <pre>docker build .</pre> <p>Tag an image</p> <pre>docker tag IMAGE NEWIMAGE</pre> <pre>docker tag ubuntu ubuntu:18.04</pre> <p>Build and tag an image from a Dockerfile</p> <pre>docker build -t IMAGE DIRECTORY</pre> <pre>docker build -t myimage .</pre> <p>Save an image to .tar file</p> <pre>docker save IMAGE > FILE</pre> <pre>docker save nginx > nginx.tar</pre> <p>Load an image from a .tar file</p> <pre>docker load -i TARFILE</pre> <pre>docker load -i nginx.tar</pre>	<p>Show the logs of a container</p> <pre>docker logs CONTAINER</pre> <pre>docker logs web</pre> <p>Show stats of running containers</p> <pre>docker stats</pre> <p>Show processes of container</p> <pre>docker top CONTAINER</pre> <pre>docker top web</pre> <p>Show installed docker version</p> <pre>docker version</pre> <p>Get detailed info about an object</p> <pre>docker inspect NAME</pre> <pre>docker inspect nginx</pre> <p>Show all modified files in container</p> <pre>docker diff CONTAINER</pre> <pre>docker diff web</pre> <p>Show mapped ports of a container</p> <pre>docker port CONTAINER</pre> <pre>docker port web</pre>

Known questions

Q: If I have 2 CPUS in my computer, can I run more than 2 containers?

A: Yes you can have as many containers as you want.

Q: Hi, can someone share how to get the requirements into the get-tweets python script? I think I got a bit confused

A: If you run `pip freeze` you will get all python libraries in your current environment printed to your terminal. if you then write that output directly into a file, e.g. `pip freeze >> requirements.txt`

but you may only want to do that for a few libraries to avoid overkill. So, for example, `pip freeze | grep pandas` will post-filter the result of `pip freeze` to only display the resulting rows that contain the word "pandas" in it.

Q: Difference between docker run and docker exec?

Docker **build** creates Docker images from Dockerfiles.

Docker **run** is for creating containers.

Docker **exec** only interacts with containers that are actively running.

- Show example with bash

```
docker exec -it habanero bash
```

Useful links

- <https://spin.atomicobject.com/2018/10/04/docker-command-line/...>

TO DOs

- Run once all commands in the lesson to see that it all works