DOCKER TUTORIAL

Inverse pendulum
Env

Docker image -rm
↳ settings in Pycharm

What is Docker?

Container (own isolated environment)
- Way to package app with necessary dependencies + configuration
- Portable Artefact (packaged with all needed configuration)

Containers live in a container repository:
- Public repo: Docker Hub (contains Nodejs, redis, etc.)

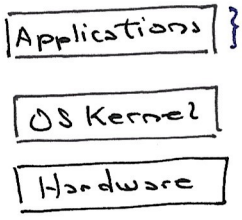
What is a Container?

- Layers of images ⇒ (only diff. layers are downloaded)
- Mostly Linux Base Image
- App Image on top

→ IMAGE = the actual package (Artefact : moved around)

→ CONTAINER = actually start the application

Docker vs. Virtual Machine



→ Size: Docker smaller
→ Speed: Docker faster
→ Compatibility: VM of any OS can run on any OS host.
(Ø with Docker)

Windows: Application + Kernel ⇸ Linux: App. ⇒ we must use Docker Toolbox.

Basic Commands on Docker

- Difference between Image and Container:
→ CONTAINER is a running environment for IMAGE

- docker pull redis ↳ image downloaded
- docker ps: list running containers
- docker run: starts new container with a command
↳ docker run -d redis.

→ View the existing images: docker images

{ · File system · Env configs · App image } CONTAINER port 5000

Restart the container:

1. docker stop <container-id>
2. docker start <container-id>

{ redis : 4.0 }
image / version

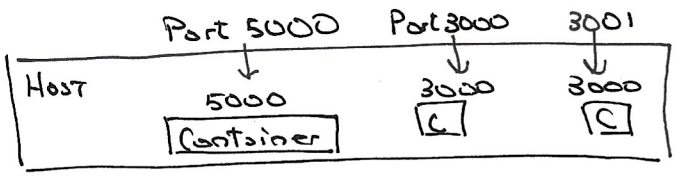
docker ps -a : lists of running and stopped containers

docker run : pull and start the container

if we have two containers with the same image - but with different versions:

Container Port vs. Host Port

- Multiple containers can run on your host machine
- Your laptop has only certain ports available.



- Example: redis 4.0 vs redis with port 6379 (container port)
  - → We need to link them to our host ports
  - → docker run -p6000:6379 -d redis
    (6000: host, 6379: container)

In short...

- docker pull <image:version>
- docker ps
- docker run -p<host-port>:<container-port> -d <image> (-d → detached mode)
- docker images
- docker stop / start <container-id>
- docker ps -a
- docker run -d <image> => pull + start.

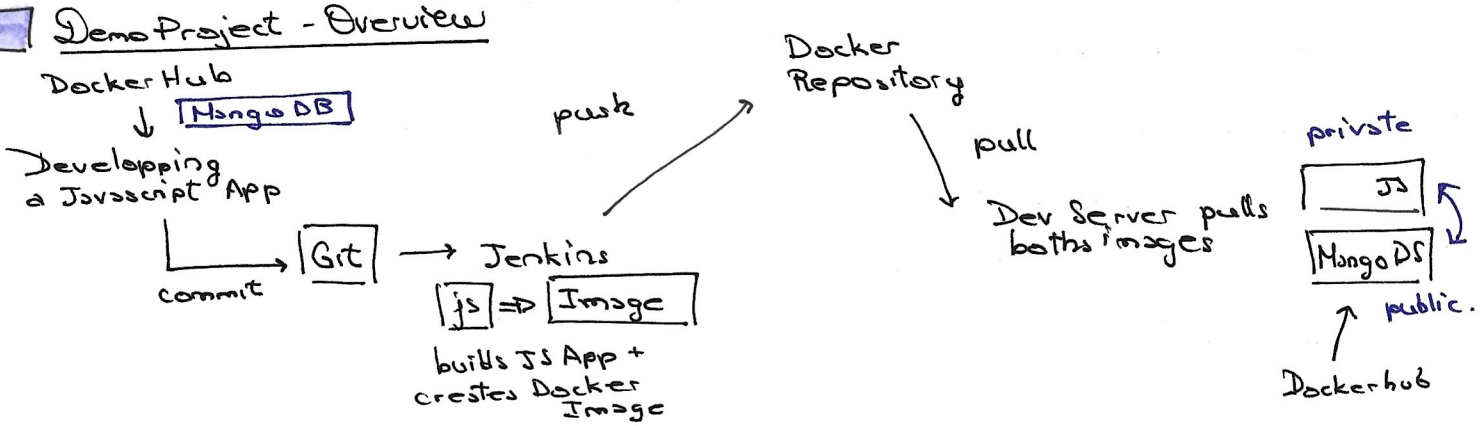
REMOVE IMAGE OR CONTAINER:

- docker rmi <image-id> ...
- docker rm <container-id>

Debugging a container

- docker logs <container-ID or name>
- stop the running container, use docker run --name in order to change the name
- docker exec -it ... in order to that terminal (inside the container => linux language)

Demo Project - Overview



- docker network ls
- docker network create <name>
- docker run -p 27017:27017 -d
  -e MONGO_INITDB_ROOT_USERNAME = admin } Env. variables (look documentation
  -e ...
  --name mongodb
  --net mongo-network
  mongo (name)

Docker Compose

1) Mongo: docker run command

```
docker run -d \
--name mongodb \
-p 27017:27017 \
-e MONGO_..._USERNAME
   = admin \
-e MONGO_..._PASSWORD
   = password \
--net mongo-network1
mongo
```

2) MongoExpress

[...]

mongo-docker-compose.yaml

→ Docker Compose takes care of creating a common Network (No need to create a network)

```
version: '3'
services:
  mongodb:
    image: mongo
    ports:
      - 27017:27017
    environment:
      - MONGO_...USERNAME = admin

  mongo-express:
    image: mongo-express
    ports:
      - 8080:8080
    env:
      - ME_CONFIG_...
```

docker-compose -f <file.yaml> up

Dockerfile ⇒ blueprint for any docker image.

FROM node → install node

ENV MONGO_DB_USERNAME=admin \
MONGO_DB_PWD=password

RUN mkdir -p/home/app ⇒ created INSIDE of the container, none in the host env.
↳ Any Linux command

COPY ./home/app ⇒ Executes on the HOST machine
↳ into our current host directory

CMD ["node", "server.js"] ⇒ start the app with "node server.js"
↳ ENTRYPOINT COMMAND

⇒ Node is pre-installed because of base image.

Image Layers:
- app:1.0 ⇒ FROM node...
- node:13-alpine ⇒ FROM Alpine:3:10
- alpine:3.10

⇒ When you adjust the Dockerfile, you MUST rebuild the image.