Docker

Core Concepts

▼ CGroups

Beyond separating software concerns, we can separate the amount of machine power a container is going to use, such as CPU and RAM usage.

created by Google

▼ Containers

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

▼ Dockerfile

From: ImageName → builds the docker from the dependencies of this Image
other than that, you can use the Read/Write layer and alter an already running container

▼ Image Registry

Where we store our images

▼ Images

Your images work with dependencies, that can be used within another images, and when an error occur, it stays isolated in the dependency it was.
Are immutable by default and have a Read/Write layer

▼ Namespaces

It's a way to separate processes and cheat the containers to think they are the only thing running in the machine.

▼ Overlay File System

You are working with patches, when you create a new version, it's just an update on what has actually changed, works like git.

▼ Relationship

We have a Docker Client and a Docker Host, the Client make requests for the Host regarding:

- containers
- run - pull - push
- volume
- network

Docker Host:

- cache (holds image registry)
- volumes (data persistency)
- network → comunication between containers
- daemon (API)

Docker Compose

It’s a framework that helps docker and pushes all your containers automatically

```
version: '3'

services:

  laravel:
    build:
      context: ./laravel
      dockerfile: Dockerfile.prod
    image: marcus21/laravel:prod
    container_name: laravel
    networks:
      - laranet

  nginx:
    build:
      context: ./laravel
      dockerfile: Dockerfile.prod
    image: marcus21/nginx:prod #
    container_name: nginx
    networks:
      - laranet
    ports:
      - "8080:80"

networks:
  laranet:
    driver: bridge
```

Be careful with spaces on .yaml

Docker-compose commands:

`docker-compose up -d` -d flag will ensure you can still use your terminal

`docker-compose up -d --build` will rebuild your containers (important when changes are made)

`docker-compose ps` will show only the docker compose containers

`docker-compose down` will turn off the containers

Solutions for Dependencies:

▼ Dockerize

```
ENV DOCKERIZE_VERSION v0.7.0

RUN apk update --no-cache \
    && apk add --no-cache wget openssl \
    && wget -O - https://github.com/jwilder/dockerize/release
    && apk del wget
```

▼ Wait-for-it

https://github.com/codeedu/docker-wait-for-it

▼ Docker Healthcheck

https://github.com/devfullcycle/docker-healthcheck

Nginx as Reverse Proxy

Reverse proxies are commonly used to enhance security, load balancing, and to serve multiple web servers behind a single IP address. They can also handle tasks like SSL encryption, caching, and routing requests to different servers based on various criteria.

Nginx cannot pass images via php ftm, so you need them in your nginx container.

Multistage Building

You are basically building you application in steps

Docker Image

The smaller the image, the better. Since it will be:

- Better to Download and Upload
- Better security

So a very common docker image is one based in Alpine Linux since is a really small Linux Distro, but has all necessary dependencies.

Docker Hub

A hub where all docker images are stored, this is the default hub, but there are more hubs out there, for example amazon ones, that can be private. You can check the Dockerfile of each image in Docker Hub.

How does NGINX works? → link

If you push an image to docker hub and nobody pulls it during 90 days, it will be deleted

Docker File

Is a way to automate process of image creation

`docker built -t docker_hub_user_name/docker_image_name:latest .`

`docker run -it docker_hub_user_name/docker_image_name bash`

When we work with a docker file, our default user is ROOT

```
FROM nginx:latest

USER ALGUM_USER

WORKDIR /app

RUN apt-get update &&\
 apt-get install vim -y

COPY html/ /usr/share/nginx/html

ENTRYPOINT [ "echo", "Hello"]

CMD [ "World" ]
```

▼ CMD vs ENTRYPOINT

CMD can be overwritten in the command line

CMD is a parameter of my entry point

It’s a common practice to have 2 dockerfiles, 1 is the one you are apt-get and etc. the other one is going to have the COPY trait to get your files.

THIS IS A NORMAL DOCKER

```
FROM php:7.4-cli

WORKDIR /var/www

RUN apt-get update && \
    apt-get install libzip-dev -y && \
    docker-php-ext-install zip

RUN php -r "copy('https://getcomposer.org/installer', 'composer-
    php composer-setup.php && \
    php -r "unlink('composer-setup.php');"

RUN php composer.phar create-project laravel/laravel laravel

ENTRYPOINT [ "php", "laravel/artisan", "serve" ]
CMD ["--host=0.0.0.0"]
```

THIS WOULD BE A DOCKER.PROD

```
FROM node:15

WORKDIR /usr/src/app

COPY . .

EXPOSE 3000

CMD ["node", "index.js"]
```

To run this dockerfile, you could use something like this:

`docker build -t marcus21/hello-express . -f Dockefile.prod`

MySQL

`docker exec -it db bash` → opens mysql on bash

`mysql -uroot -p` → go to root user

`“root”` → About the password, you need to type it using quotes.

- **mysql commands**

  `show databases;`

  `use nodedb;`

  `create table people(id int not null auto_increment, name varchar(255), primary key(id));`

  `desc people;`

  `select * from people;`

  `delete from people;`

  `create table people(id int not null auto_increment, name varchar(255), ip_adress varchar(45), primary key(id));`

Network

Kinds of network in Docker:

- bridge : does not form anything, easy communication
  - is the default network when I set a container
- host : it entangles docker and it’s host
- overlay : when you have a lot of dockers in the same network
  - ex: Docker Swarm
- maclan : you can set a mac adress to a container
- none : when you want to seclude your container

Some commands:

1. List your networks

`docker network ls`

2. Remove your unused networks

`docker network prune`

3. Inspect for a certain kind of network

`docker network inspect bridge`

4. How to show IP:

`ip addr show`

5. How to ping into other machine in the same network

`ping ip_adress`

6. How to detach from a container

`ctrl + P` + `ctrl + Q`

7. How to create a network of type bridge

`docker network create --driver bridge my_network`

8. How to create a container that is hosted in your network

`docker run -d -it --name ubuntu1 --network my_network bash`

- containers that are on the same network can ping each other by name

9. How to connect a container to your already created network

`docker network connect my_network ubuntu3`

10. How to inspect your already created network

`docker network inspect my_network`

11. How to make a container connect to you machine as host

`docker run --rm -d --name nginx --network host nginx`

- now if you type local host, it should open nginx

12. You can get inside you local machine via container

`curl` `http://host.docker.internal:8000`

- remember to have curl in your container

13. Default Bridge Network

`docker network create laranet`

Commands and Structure

1. Basic command for listing (the second list even the closed ones)

`docker ps`

`docker ps -a`

2. Latest is a default entry point, `-i` (std:in) is responsible for interactivity and `-t` allows us to actually type in terminal

`docker run -it ubuntu:latest bash`

3. The -rm makes the process invisible after ended

`docker run -it --rm ubuntu:latest bash`

4. Expose the port 80 of your container to the port 8080 of you machine using ngnix

`docker run -p 8080:80 nginx`

5. Same command as above, but now using the Detached flag, you can use your terminal

6. `docker run -d -p 8080:80 nginx`

7. Now I gave it a name

`docker run -d --name nginx -p 8080:80 nginx`

8. You can start and stop containers

`docker start container_id`

`docker stop container_id`

9. You can delete containers

`docker rm container_id`

10. How to delete a running container

`docker rm -f container_id`

11. This way i can run commands into a container

`docker exec nginx ls`

12. Knowing the above, how to execute bash

`docker exec -it nginx bash`

13. Where our html is located

`/usr/share/nginx/html/`

14. When downloading something, you need to use apt-get update, before anything, because your image is really reduce by default

`apt-get update`

`apt-get install vim`

15. How to use Bind Mount with docker (`--v`)

```
docker run -d --name nginx -p 8080:80 -v
/mnt/c/Users/marcu/Documents/Code/FullCycle/Docker/html/:usr/share/nginx/html nginx
```

or basically:

`docker run -d --name nginx -p 8080:80 -v mount_path_from_wsl_file:path_to_docker_file nginx`

The code using `--v` is deprecated, another way of using is(`--mount`):

```
docker run -d --name nginx -p 8080:80 --mount
'type=bind,source="$(pwd)"/html,target=/usr/share/nginx/html nginx
```

Don't ever forget of the single quotes and don't space the arguments

The `--mount` won't create a new directory differently from `--v` .

15. How to use volumes via `--mount`

`docker run --name nginx -d --mount 'type=volume,source=meu_volume,target=/app' nginx`

16. For when you want to clean your machine of some docker files you don't know how to delete

`docker volume prune`

17. How to download some image from Docker Hub:

`docker pull image_name`

`docker pull image:version`

18. How to list docker images and names:

`docker images`

19. Remove images from your docker

`docker rmi image:version`

20. How to remove all containers in docker (running or not)

`docker rm $(docker ps -a -q) -f`

21. Login (Docker Hub)

`docker login`

21. Push my image to docker hub (must be logged in)

`docker push my_image_name`

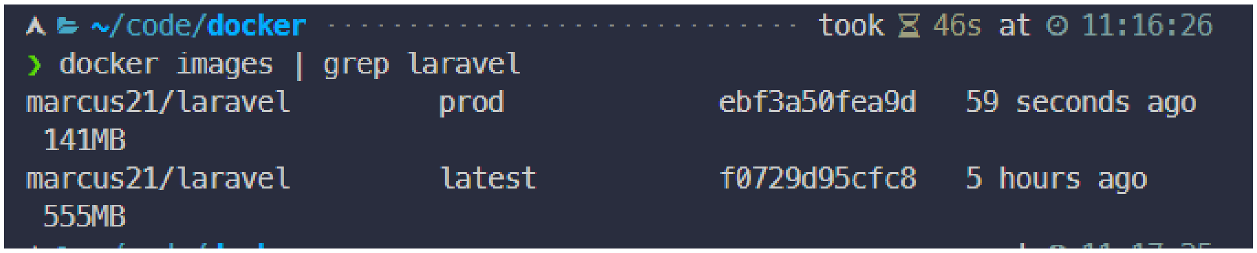
22. How to check what a container is doing:

`docker logs container_name`

23. Here we are building from a secondary docker file inside, that is the root of our CLI, while Laravel is the Folder inside the Root

`docker build -t marcus21/laravel:prod laravel -f laravel/Dockerfile.prod`

Here is a comparison in size:



```
~/code/docker ··························· took 46s at 11:16:26
❯ docker images | grep laravel
marcus21/laravel          prod             ebf3a50fea9d    59 seconds ago
 141MB
marcus21/laravel          latest           f0729d95cfc8    5 hours ago
 555MB
```

The Code for this Multistage Building

```
FROM php:7.4-cli AS builder

WORKDIR /var/www

RUN apt-get update && \
    apt-get install libzip-dev -y && \
    docker-php-ext-install zip

RUN php -r "copy('https://getcomposer.org/installer', 'composer-
    php composer-setup.php && \
    php -r "unlink('composer-setup.php');"
```

```
RUN php composer.phar create-project laravel/laravel laravel

FROM php:7.4-fpm-alpine
WORKDIR /var/www
RUN rm -rf /var/www/html
COPY --from=builder /var/www/laravel .
RUN chown -R www-data:www-data /var/www
EXPOSE 9000
CMD ["php-fpm"]
```

Another run, but inside the folder (in another project)

`docker build -t marcus21/nginx:prod . -f Dockerfile.prod`

Troubleshooting

1. WSL does not initialize docker daemon:

`sudo systemctl status docker` → checks if docker is running

`sudo systemctl start docker` → starts docker