Docker

Docker is a software platform for building container-based applications, that is, they are small and lightweight execution environments that make shared use of the operating system kernel but run in isolation from each other.

The objective is to facilitate the creation and management of environments through containers.

Docker Container

The docker container groups together pieces of software from a complete file system that covers all the resources necessary for its execution.

Docker commands

docker search – to search for an image;

docker pull [image] – to download the image you want;

docker images – to list the images available;

docker rmi [nameimage] – to remove an image;

docker run – to execute the image. For this command it can be necessary to add some parameters.

docker ps - list the containers that are running;

docker ps –a - list all containers;

docker exec <container id> -it /bin/bash - to get inside the container we want;

docker stop <container id> - stop the execution of the container.

docker login - login to a Docker registry

docker tag - give useful information about a specific image version/variant;

docker commit - allows users to take a running container and save its current state as an image.

docker push - To share your images to the Docker Hub repository;

Dockerfile

Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using **docker build**, users can create an automated build that executes several command-line instructions in succession.

Dockerfile instructions

FROM - initializes a new build stage and sets the Base Image for subsequent instructions;

MAINTAINER - allows to set the name of Author of the generated images.

LABEL - a key-value pair, stored as a string;

RUN - lets you execute commands inside of your Docker image. These commands get executed once at build time and get written into your Docker image as a new layer;

CMD - specifies the instruction that is to be executed when a Docker container starts;

EXPOSE - informs Docker that the container listens on the specified network ports at runtime;

ENV - is mainly meant to provide default values for your future environment variables;

ADD or COPY - they let you copy files from a specific location into a Docker image;

ENTRYPOINT - allows you to configure a container that will run as an executable. It looks similar to CMD, because it also allows you to specify a command with parameters.

VOLUME - creates a new volume that containers can consume and store data in;

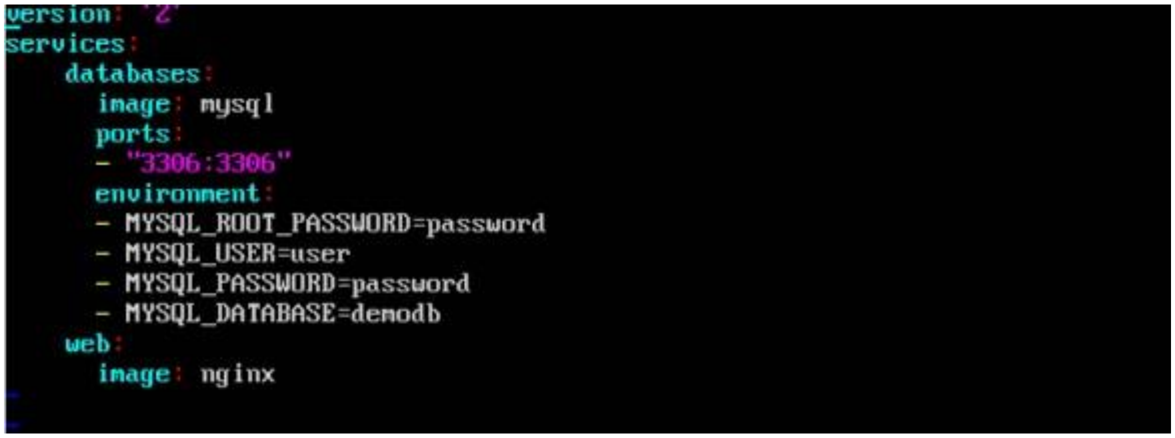
USER - sets the user name and optionally the user group to use when running the image and for any RUN, CMD and ENTRYPOINT instructions that follow it in the Dockerfile.

WORKDIR - is used to define the working directory of a Docker container at any given time.

Docker-Compose

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with **docker-compose up**, you create and start all the services from your configuration.

In the figure below I show the structure of Docker compose, which consists of two services: databases and web. For databases I use the mysql image and specify the variables, as well as their ports and in the web service I define nginx as the image for my web server.



```
version: '2'
services:
    databases:
      image: mysql
      ports:
      - "3306:3306"
      environment:
      - MYSQL_ROOT_PASSWORD=password
      - MYSQL_USER=user
      - MYSQL_PASSWORD=password
      - MYSQL_DATABASE=demodb
    web:
      image: nginx
```

Figure 1 – Docker-Compose Structure