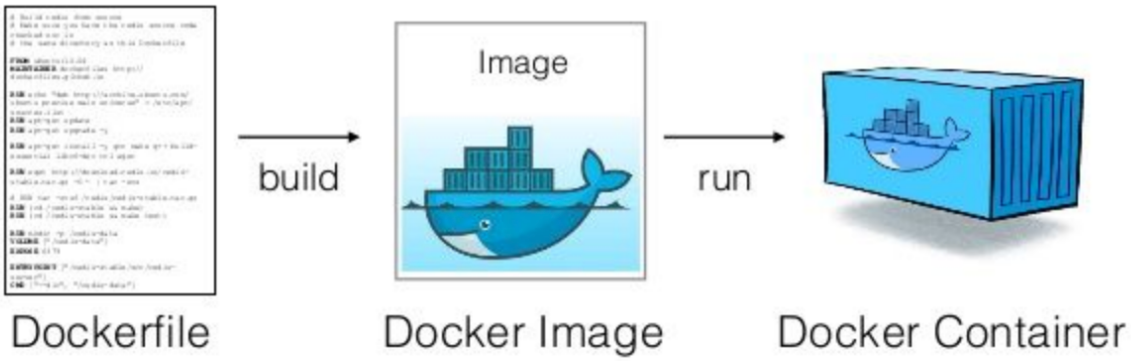
https://github.com/gerhynes/docker-notes

DOCKER



Introduction

Docker provides a consistent environment to run applications without interference and underlying differences in operating systems, ram, system specifications etc. Docker uses containers to run in an isolated environment in the system.

Docker Image

This is analogous to the blueprints of the container. They store:

- the runtime environment
- application code
- any dependencies
- extra configuration, such as environment variables
- commands

The docker images are read-only and t change them, you need to create a new image. The docker image can be shared and deployed across different environments. Running the docker image creates a container which is capable of running the application as outlined in the docker image.

Docker Container

These are isolated processes and are independent of the machine they are running on. They provide an isolated environment and ensure that the application runs as per the desire across different systems.

Dockerfile

They are a set of instructions or commands that defines how a docker image is built. They automate the process of creating a docker image. They provide the set of information that is used by the docker image to create the container.

The commonly used commands in a Dockerfile:

FROM: This specifies the parent image that will be used. E.g. python:3:11:4-slim

WORKDIR: This sets the working directory for the commands that follows it like RUN, COPY, ADD etc. like /app

COPY: it copies the file from working directory to the image.

RUN: This specify which commands are run when building the docker file. Some dependencies like tensorflow etc can be installed using this.

EXPOSE: this is required for docker desktop port mapping

CMD: this is the default command that runs when creating the container. E.g. ["python", "app.py"]

To build the docker file, use this -

```
docker build -t IMAGE_NAME .
```

To run the docker image, use this -

```
docker run IMAGE_NAME/ID
```

Some useful docker commands

Managing Images and Containers

`docker images` lists all the images you have.

`docker processes` lists all the running containers.

`docker ps-a` lists all containers.

`docker image rm IMAGE_NAME` deletes an image (if it isn't being used by a container).

`docker image rm IMAGE_NAME -f` will delete an image even if it being used by a container.

`docker container rm CONTAINER_NAME` deletes a container.

`docker system prune -a` will remove all containers, images and volumes.

In Docker, versions are denoted by tags, letting you create multiple versions of images with certain variations.

To create an image with a tag, use `docker build -t IMAGE_NAME:TAG .`

To run a container for a specific image version, specify the tag `docker run --name CONTAINER_NAME -p 4000:4000 IMAGE_NAME:TAG`

Layer caching

Docker uses a layered filesystem to manage and build images. When building an image, docker looks into the previous layers of cached images and sees if a layer with same instructions has been created previously. If there is a layer present, then docker reuses that instead of creating a new layer.

This caching mechanism significantly reduces the build time, especially when you make changes to your code or dependencies without modifying the earlier instructions in the Dockerfile.

Docker compose

This is usd to run multi-container docker applications. Sometimes, the application might demands multiple containers running simultaneously and communicate with each e.g. API, frontend and database. Docker compose makes a docker-compose.yml file that contains configurations of all the containers.

To run the Docker Compose file, use

```
docker-compose up
```

To stop and delete the container, while keeping the images and volumes, use

```
 docker-compose down
```

To remove all images and volumes, use

```
docker-compose down --rmi all -v.
```