Docker Presentation

How to drown in the sea of containers

Prepared by Amir Arsalan Yavari

Laurel

CONTENT

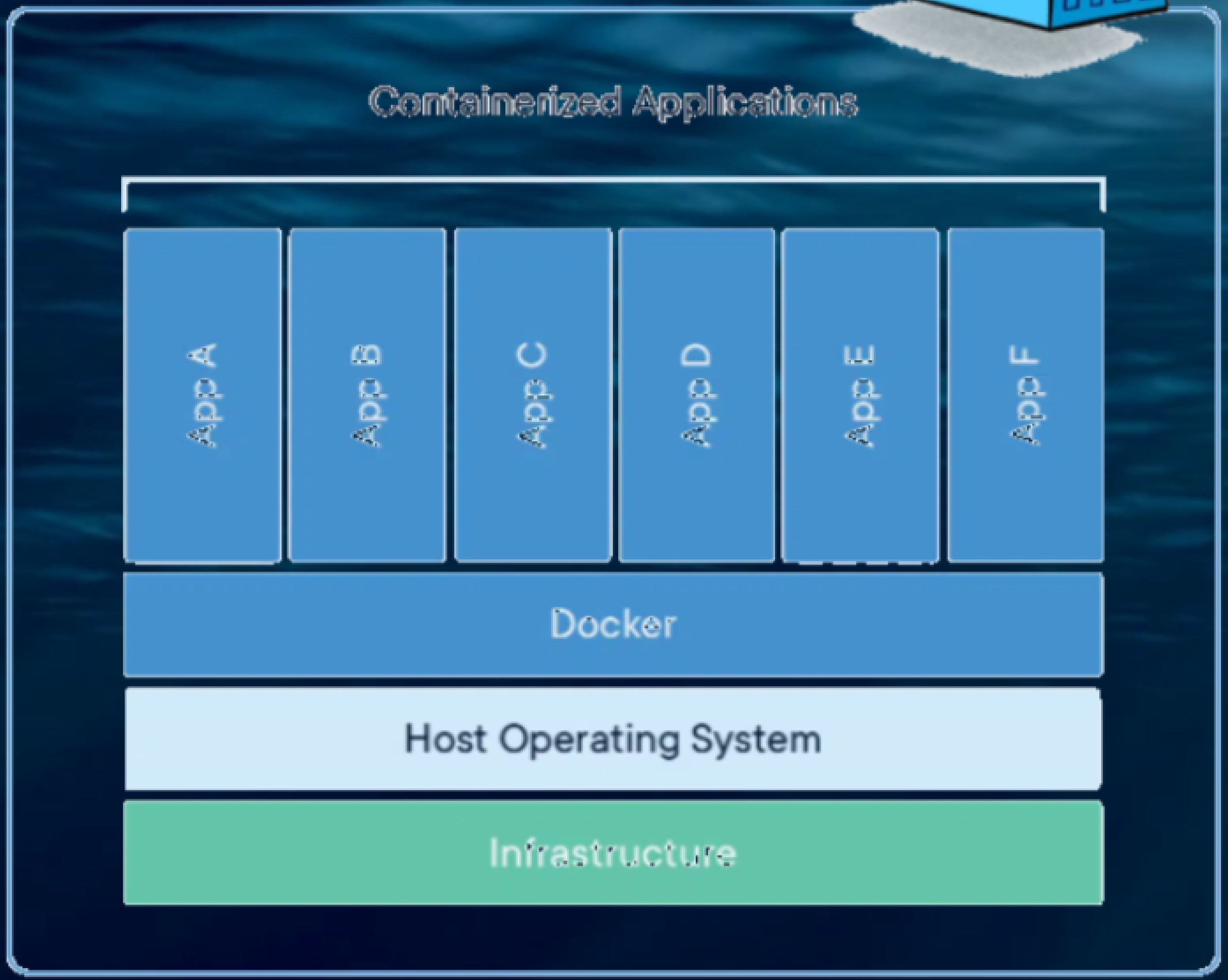
01

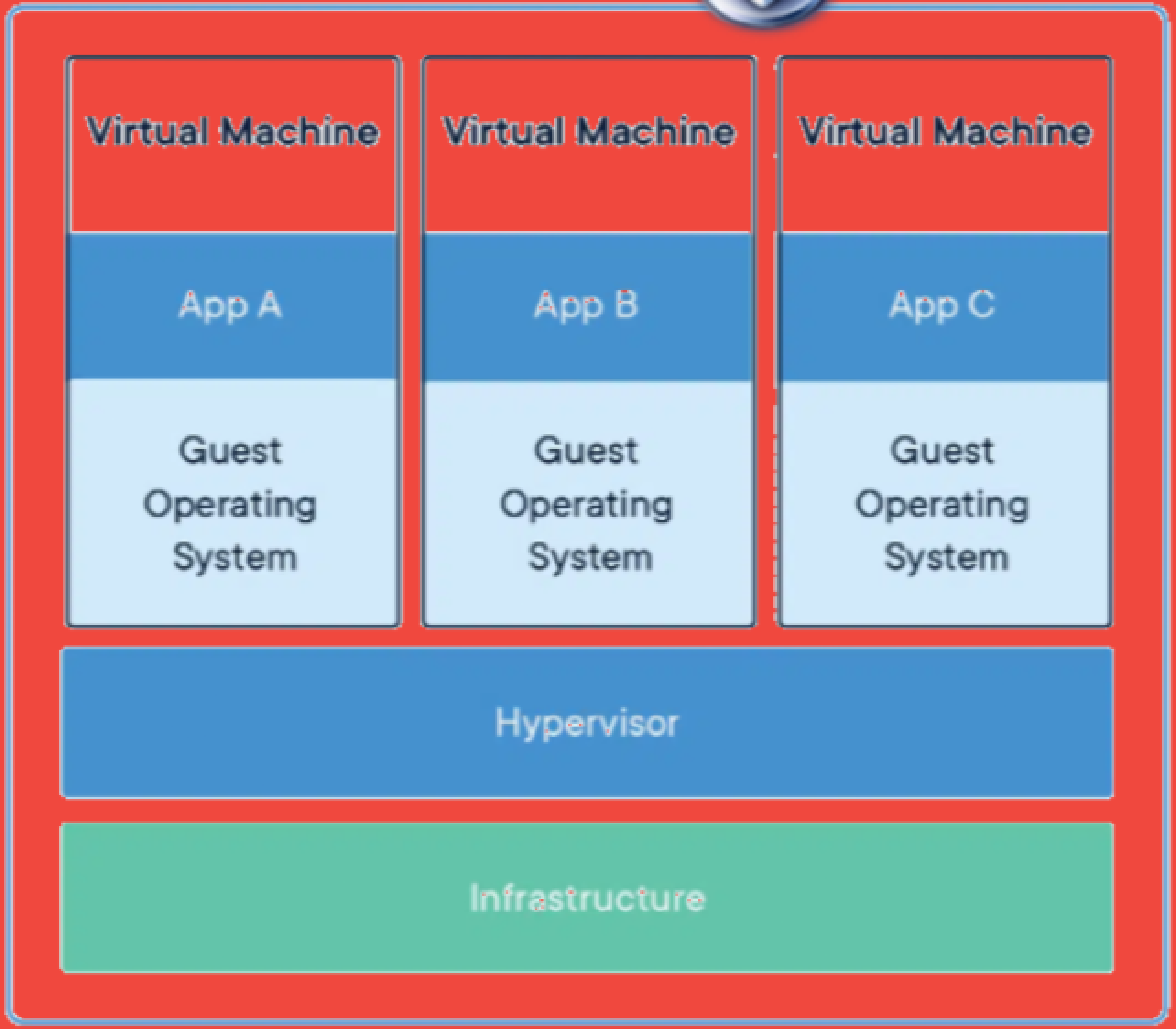
What is container and image

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.

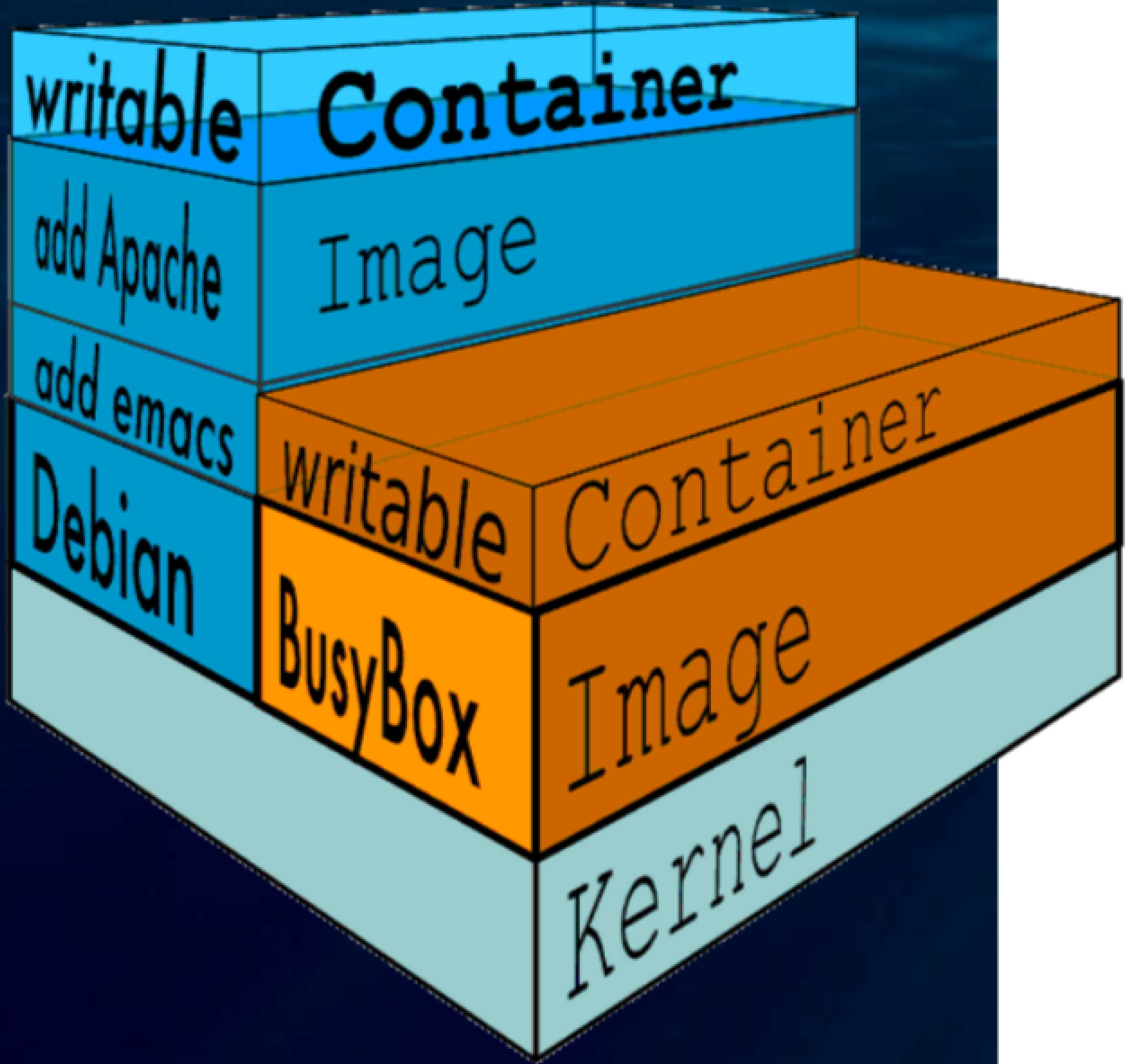
Container VS VM





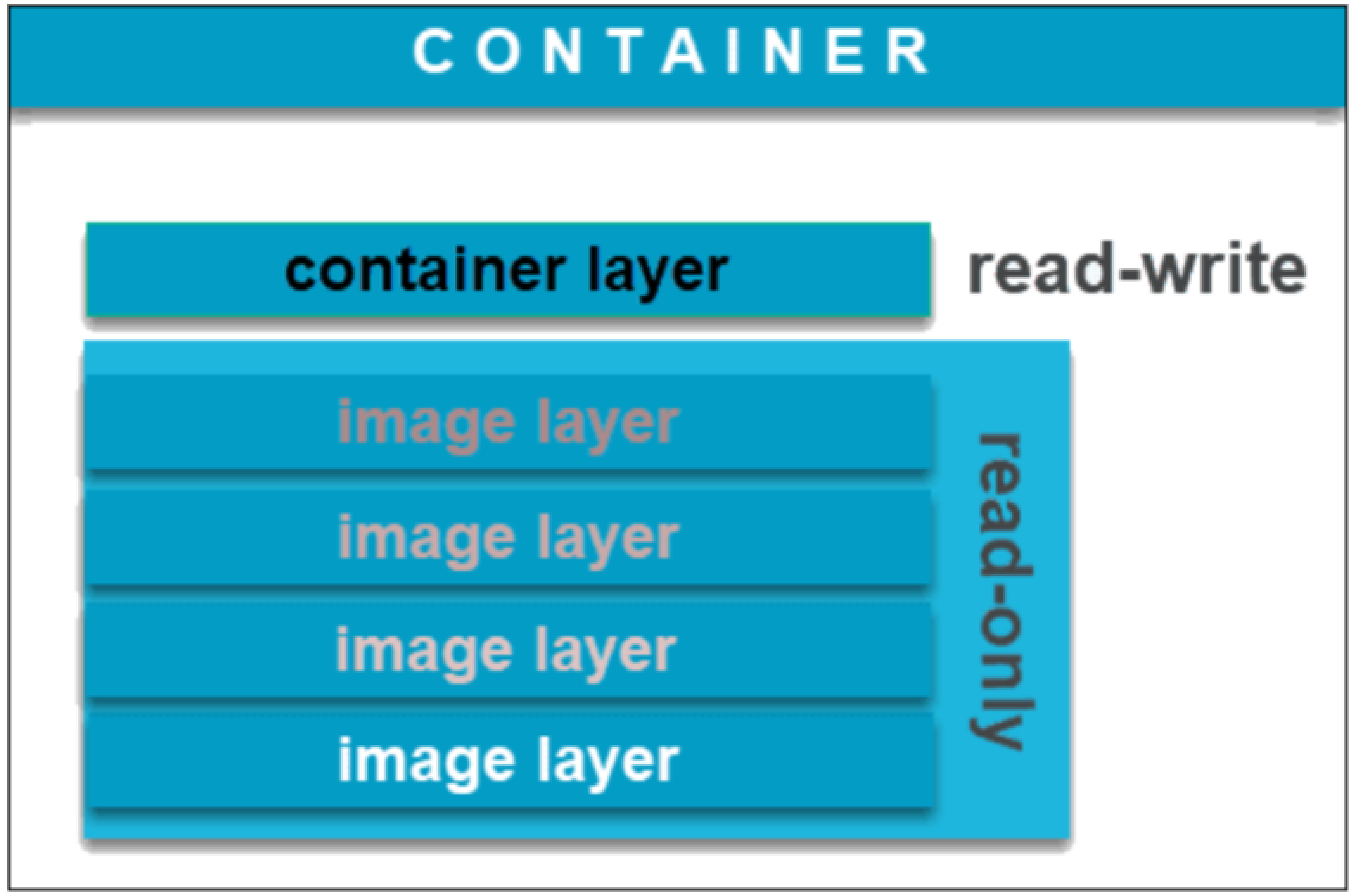
Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), can handle more applications and require fewer VMs and Operating systems.

Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, the application, necessary binaries and libraries – taking up tens of GBs. VMs can also be slow to boot.



Image

A Docker image is a file used to execute code in a Docker container. Docker images act as a set of instructions to build a Docker container, like a template. Docker images also act as the starting point when using Docker. An image is comparable to a snapshot in virtual machine (VM) environments.



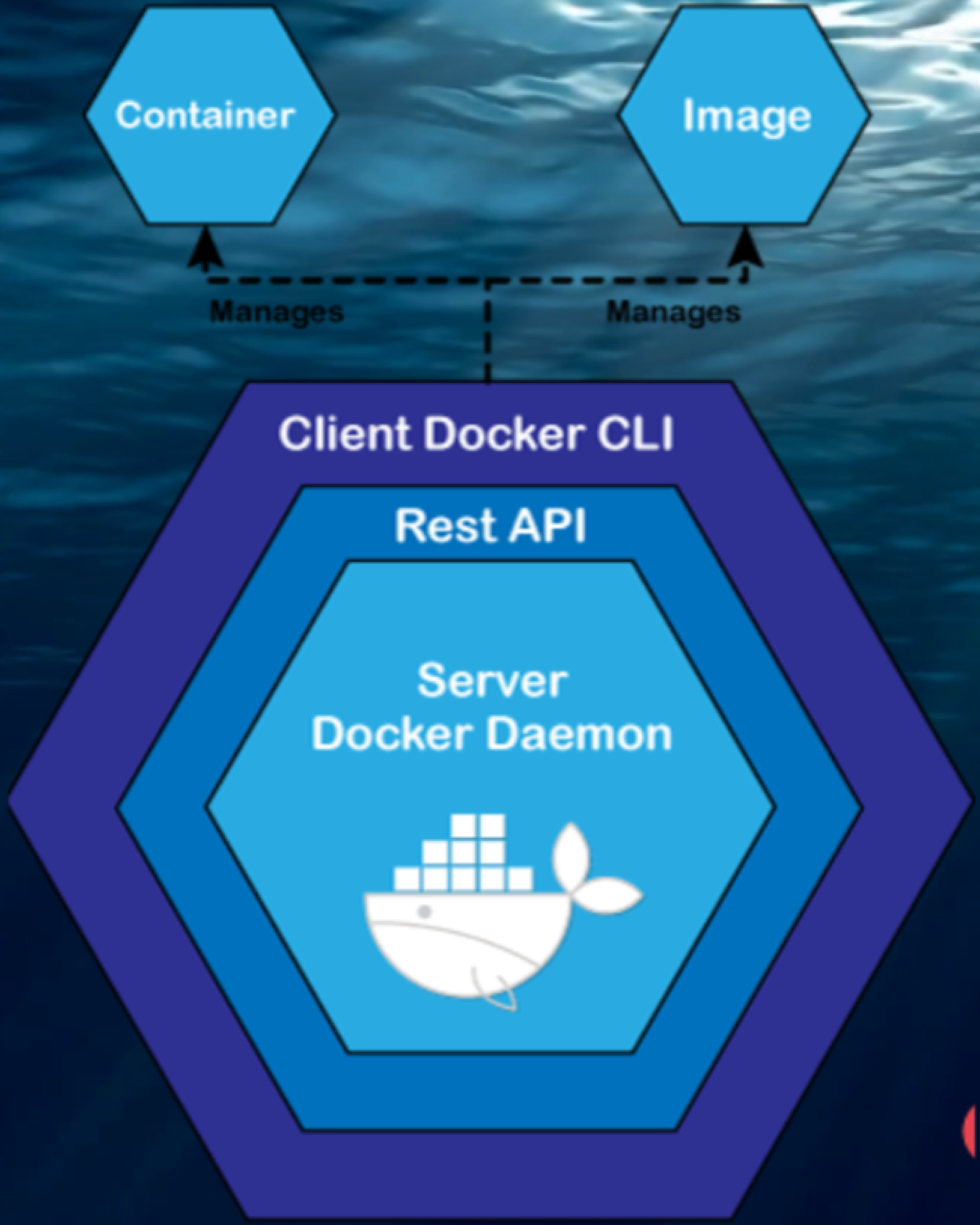
02

How docker work

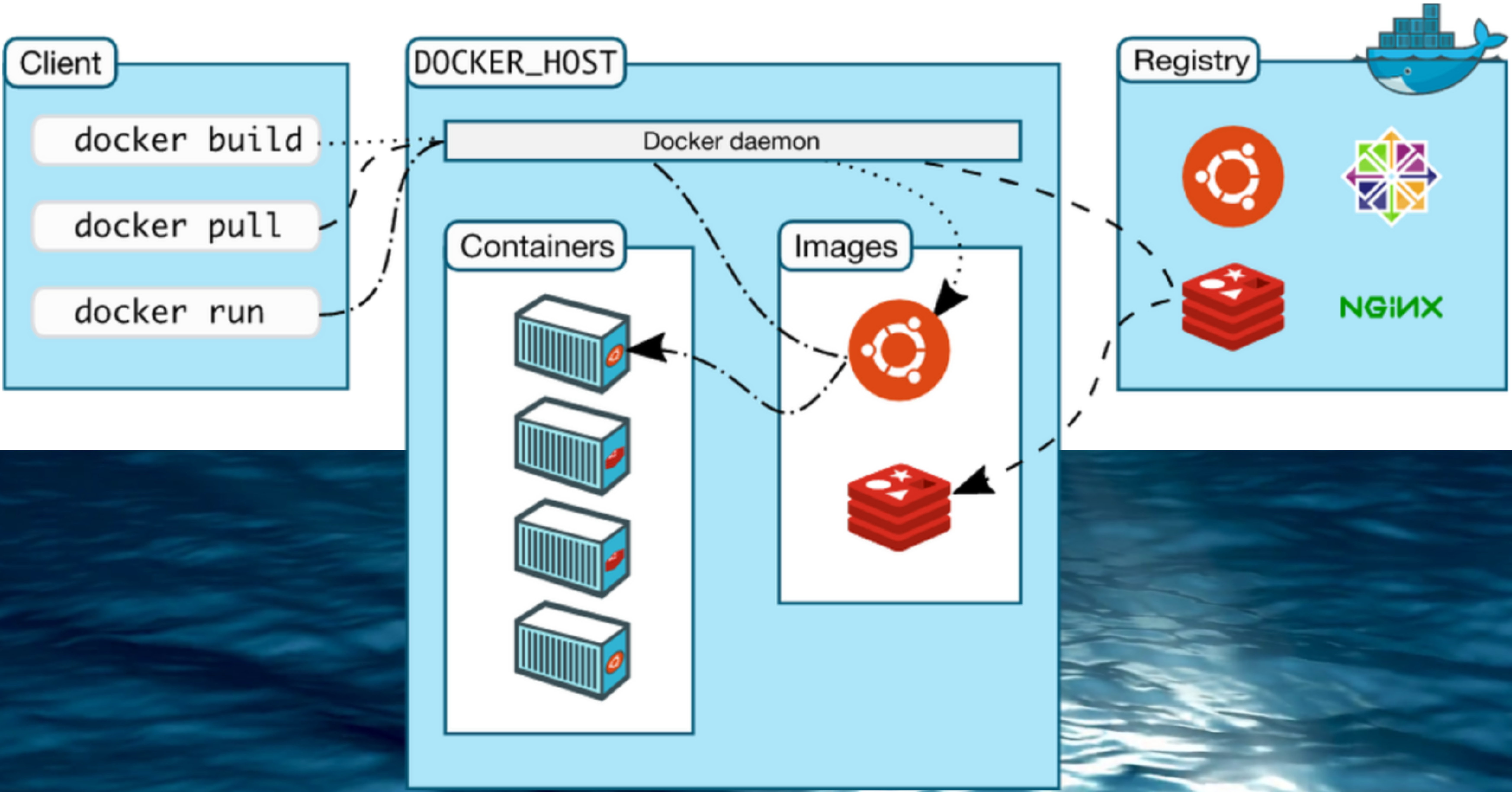
Docker Engine is an open source containerization technology for building and containerizing your applications. Docker Engine acts as a client-server application with:

- A server with a long-running daemon process `dockerd`.
- APIs which specify interfaces that programs can use to talk to and instruct the Docker daemon.
- A command line interface (CLI) client `docker`.

The CLI uses Docker APIs to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI. The daemon creates and manage Docker objects, such as images, containers, networks, and volumes.

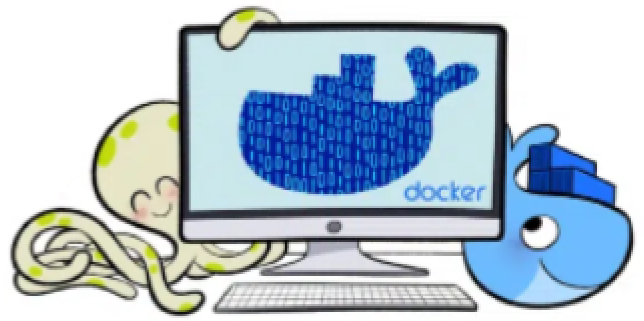


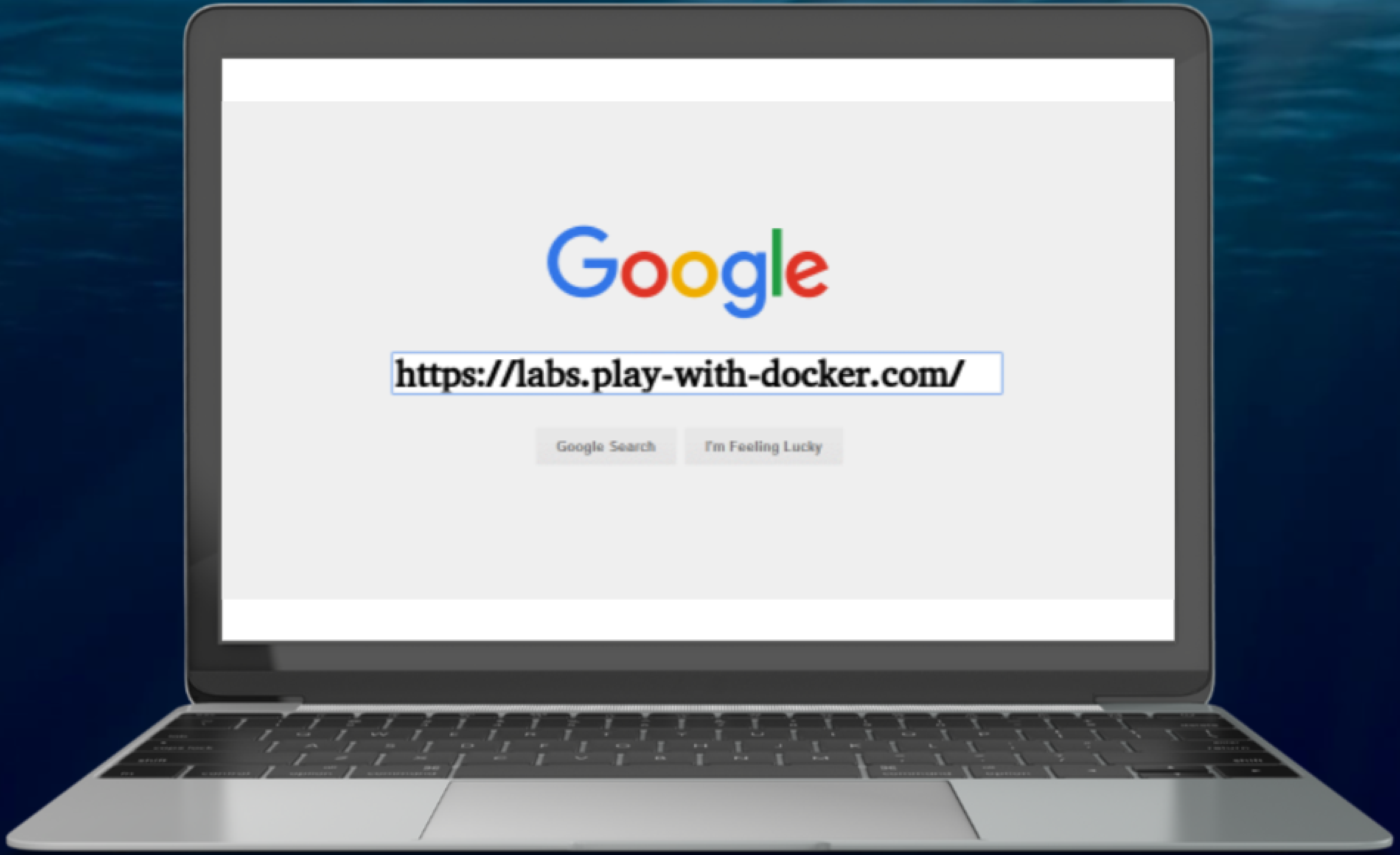
two different parts are volume and network


Client
docker build
docker pull
docker run
DOCKER_HOST
Docker daemon
Containers
Images
Registry
NGINX

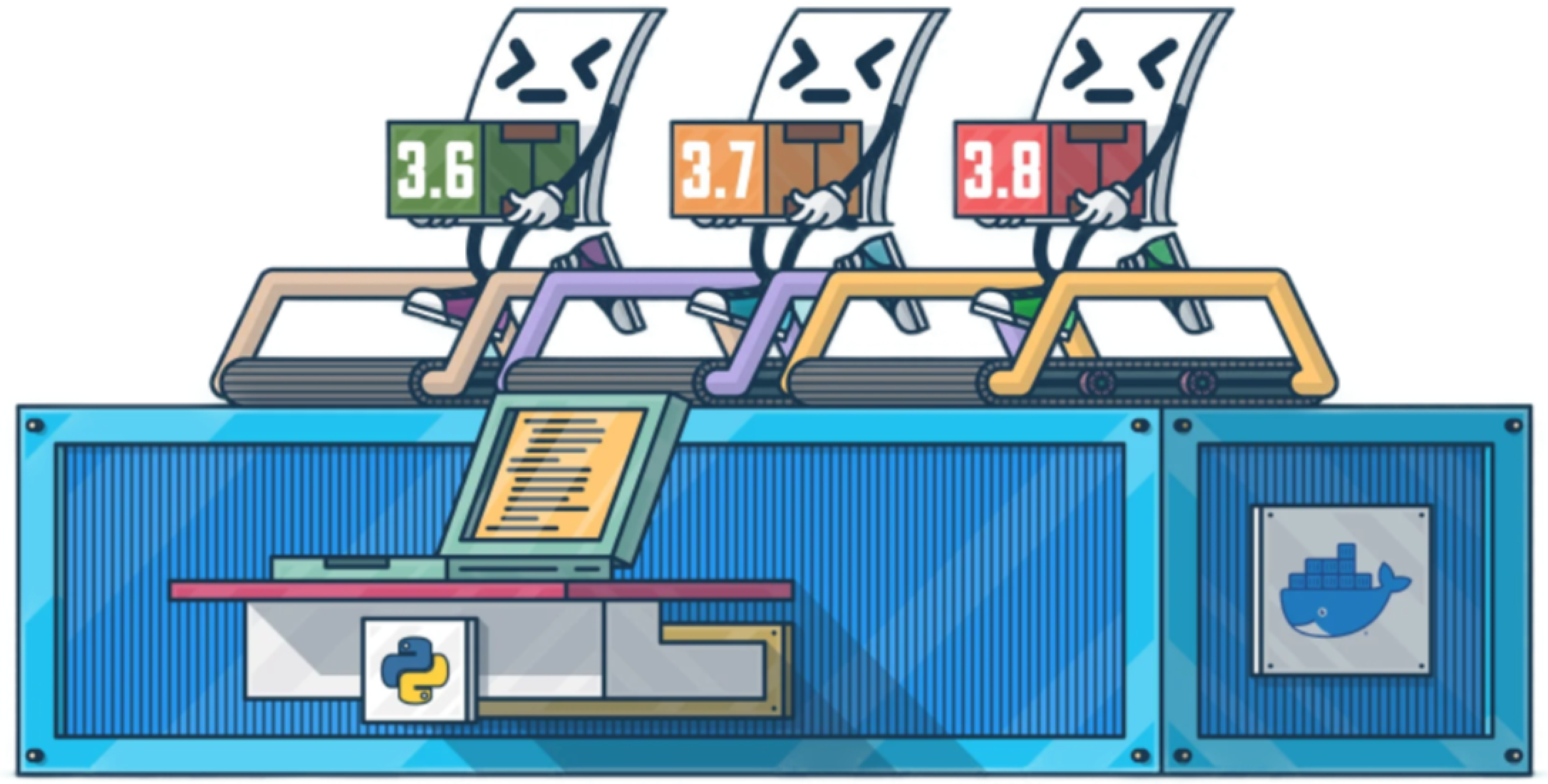
QA

let's play with docker cc

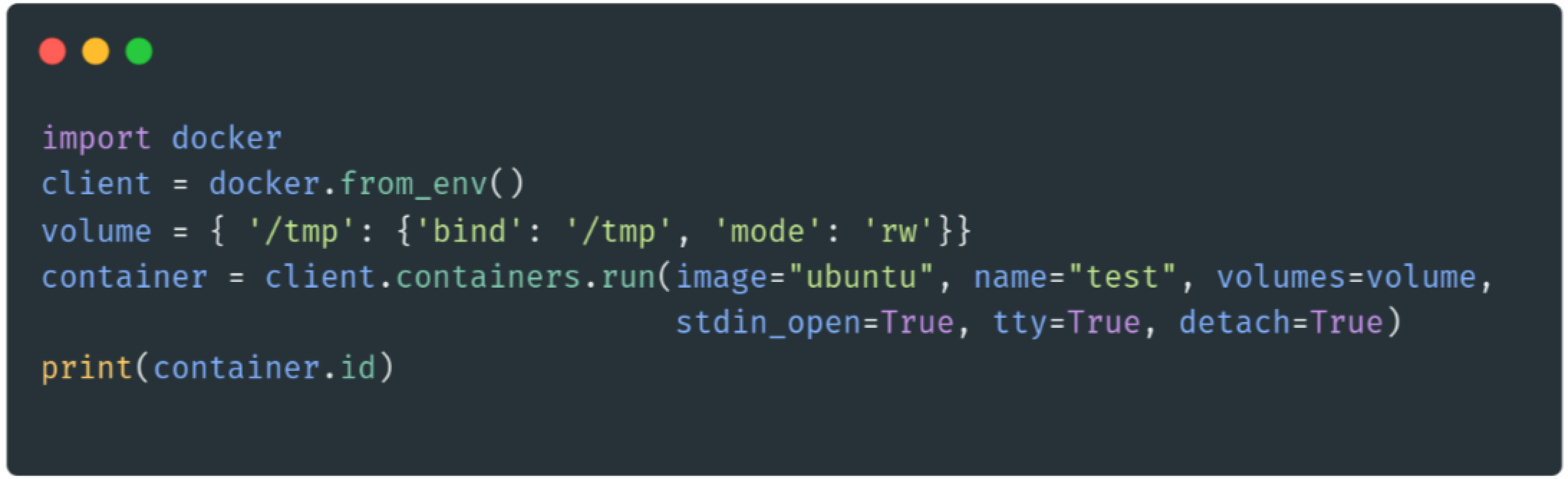




https://labs.play-with-docker.com/



Docker SDK



```python
import docker
client = docker.from_env()
volume = { '/tmp': {'bind': '/tmp', 'mode': 'rw'}}
container = client.containers.run(image="ubuntu", name="test", volumes=volume,
                                  stdin_open=True, tty=True, detach=True)
print(container.id)
```

1. Container and Image

remember images are read only but containers are changeable

2. How docker work

now we know about docker-CLI and docker-API

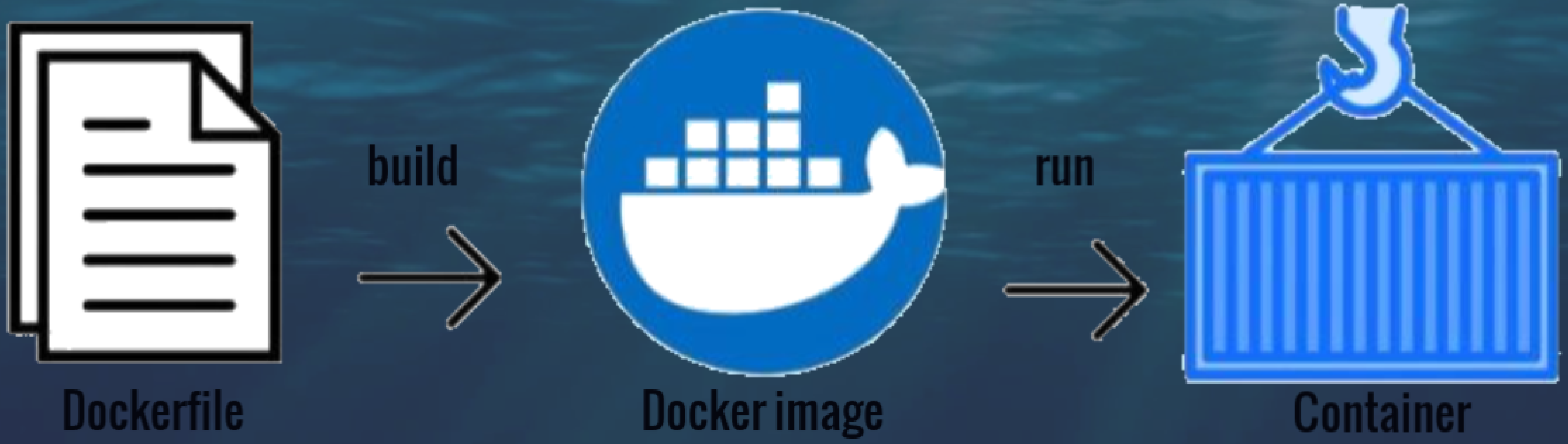
3. Docker file

using Dockerfile we can create images that so appropriate for our project

4. Docker-compose

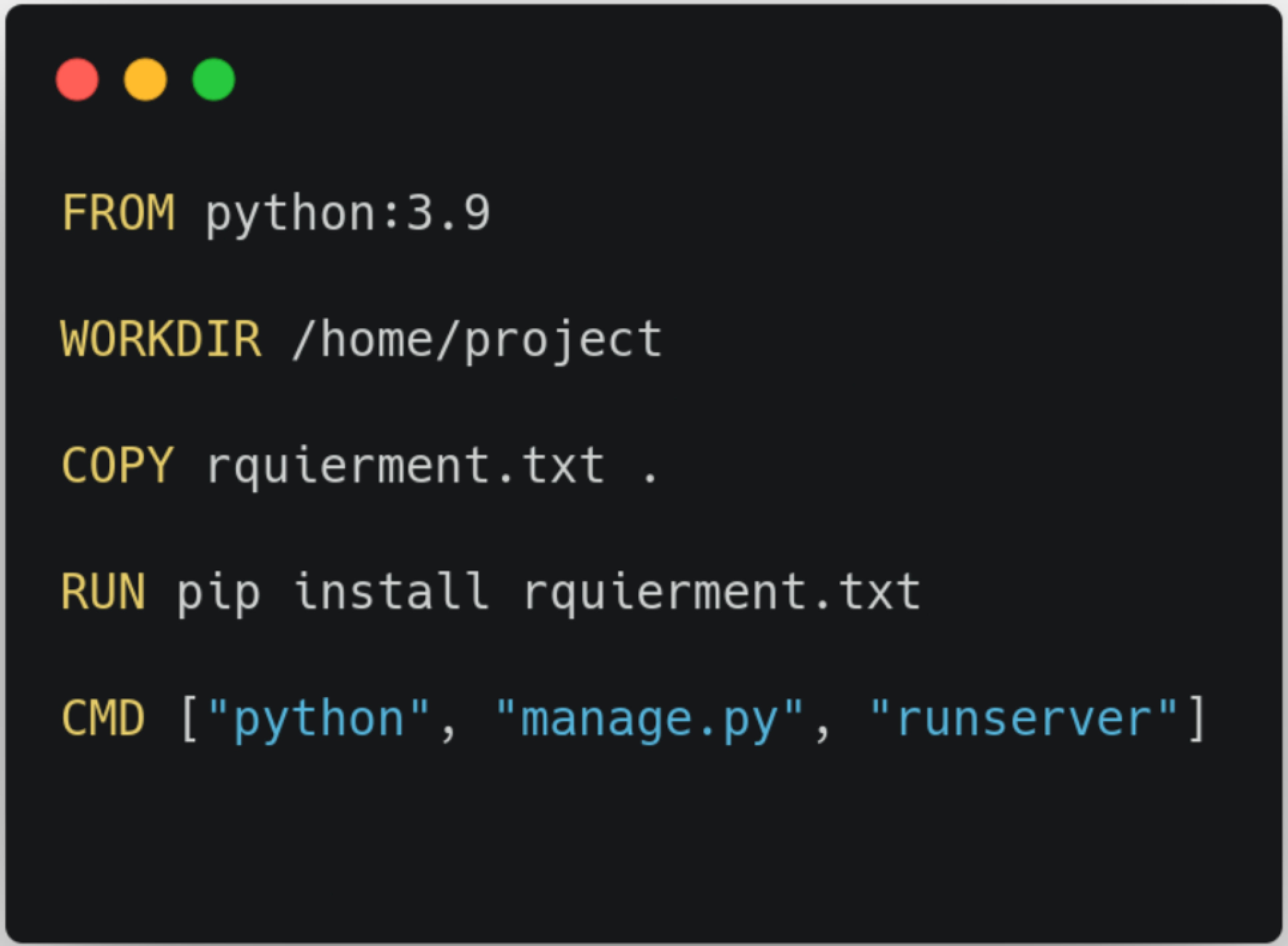
Waite more please... :)

Create your own image



Dockerfile

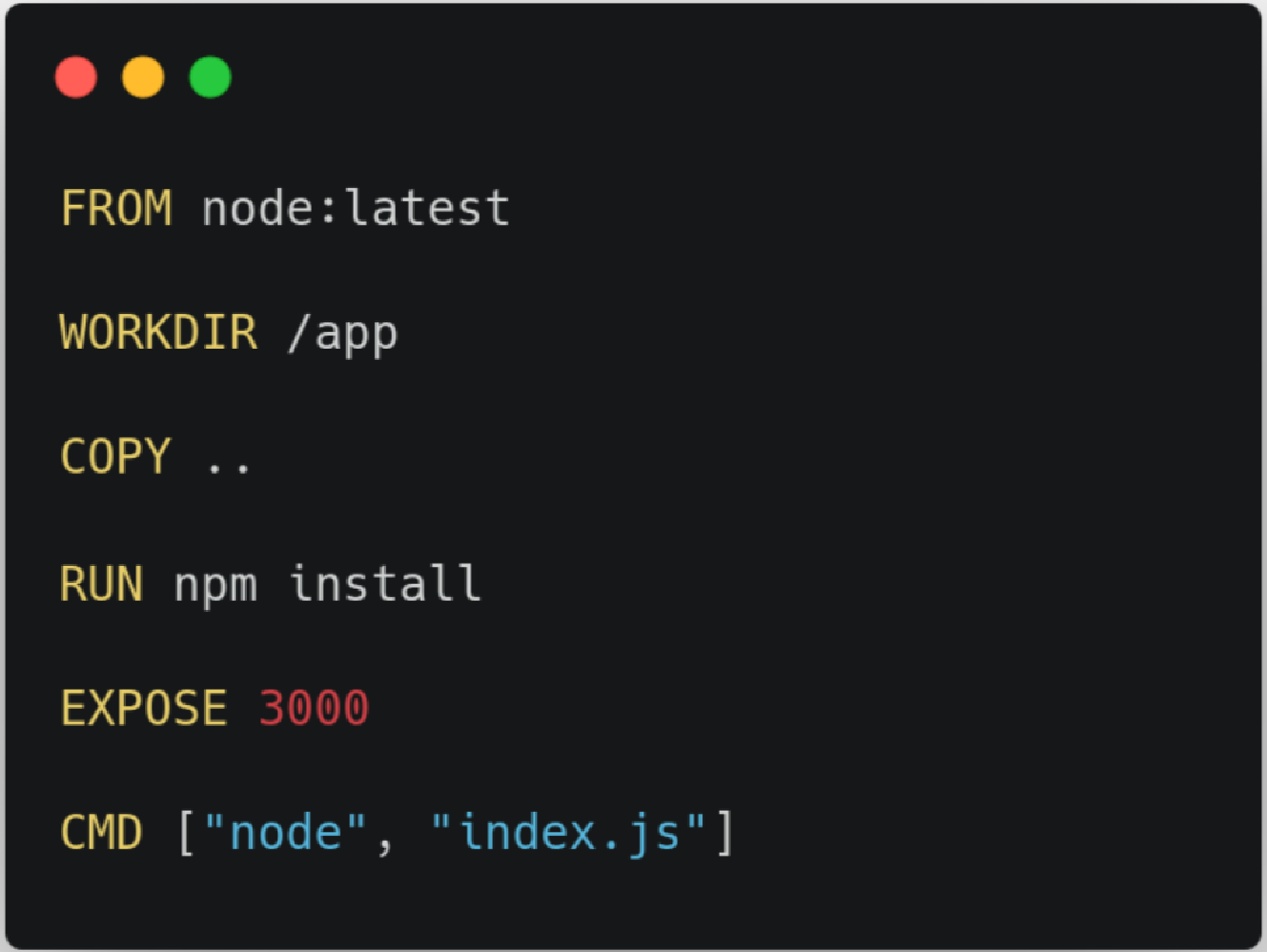
first of all you need write the instructions on a file that recommended be named Dockerfile; then using docker API build new image



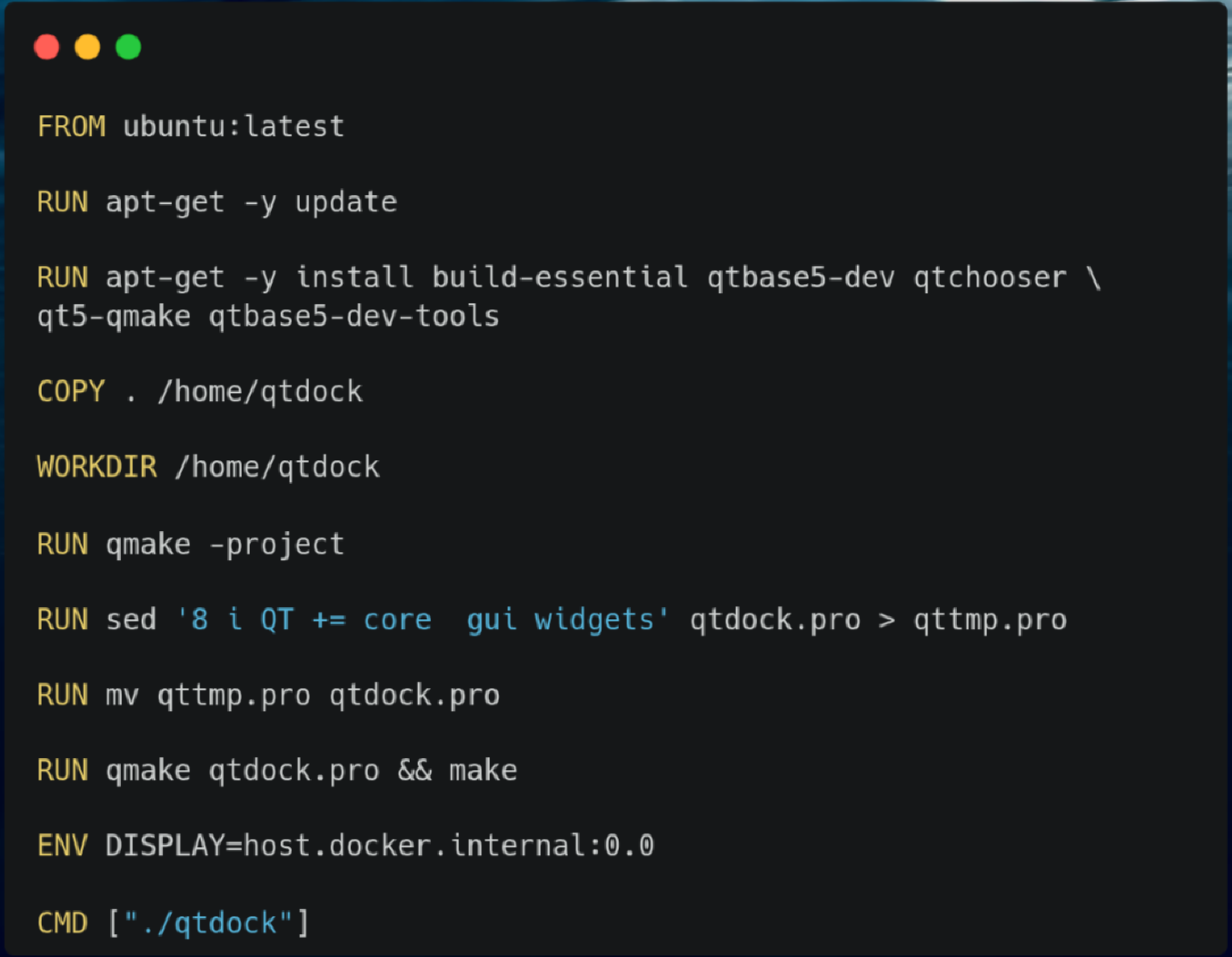
```
FROM python:3.9

WORKDIR /home/project

COPY rquierment.txt .

RUN pip install rquierment.txt

CMD ["python", "manage.py", "runserver"]
```

Dockerfile

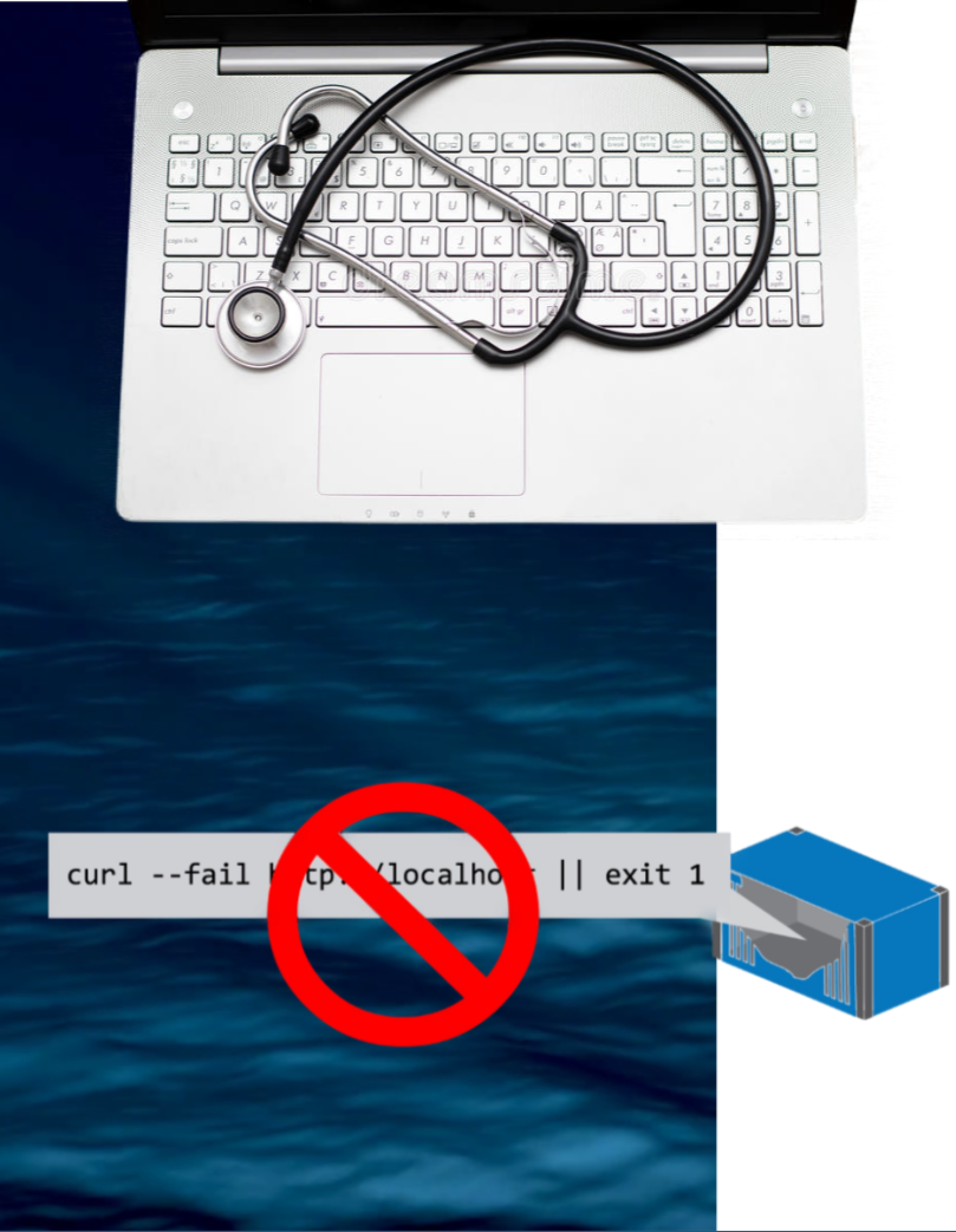
first of all you need write the instructions on a file that recommended be named Dockerfile; then using docker API build new image



```
FROM node:latest

WORKDIR /app

COPY ..

RUN npm install

EXPOSE 3000

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



```dockerfile
FROM ubuntu:latest

RUN apt-get -y update

RUN apt-get -y install build-essential qtbase5-dev qtchooser \
qt5-qmake qtbase5-dev-tools

COPY . /home/qtdock

WORKDIR /home/qtdock

RUN qmake -project

RUN sed '8 i QT += core  gui widgets' qtdock.pro > qttmp.pro

RUN mv qttmp.pro qtdock.pro

RUN qmake qtdock.pro && make

ENV DISPLAY=host.docker.internal:0.0

CMD ["./qtdock"]
```



```
curl --fail [illegible]tp[illegible]/localho[illegible] || exit 1
```

Health Check

Health checks allow a container to expose its workload's availability. This stands apart from whether the container is *running*. If your database goes down, your API server won't be able to handle requests, even though its Docker container is still running.

- **`--interval`** – Set the time between health checks. This lets you override the default value of 30 seconds. Use a higher interval to increase the time between checks. This helps if you have a low-priority service where regular health checks might impact performance. Use a more regular frequency for critical services.
- **`--start-period`** – Set the duration after a container starts when health checks should be ignored. The command will still be run but an unhealthy status won't be counted. This gives containers time to complete startup procedures.
- **`--retries`** – This lets you require multiple successive failures before a container is marked as `unhealthy`. It defaults to 3. If a health check fails but the subsequent one passes, the container will not transition to `unhealthy`. It will become unhealthy after three consecutive failed checks.
- **`--timeout`** – Set the timeout for health check commands. Docker will treat the check as failed if the command doesn't exit within this time frame.

04

docker-compose

Docker Compose

django is a backend framwork

Postgres is a db tool

Sunshine Optimism Logical Positive

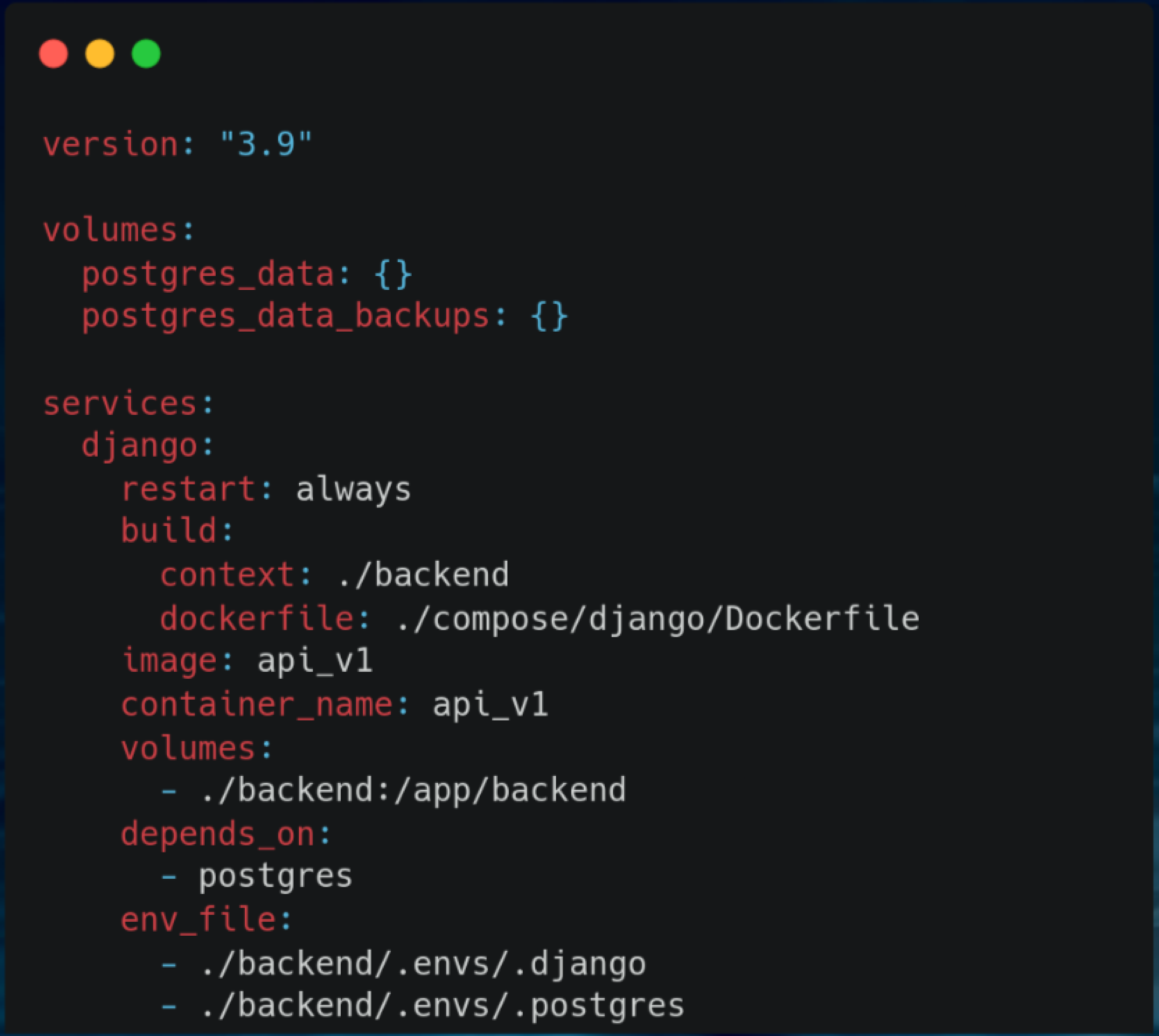
Nginx is a webserver tool

django

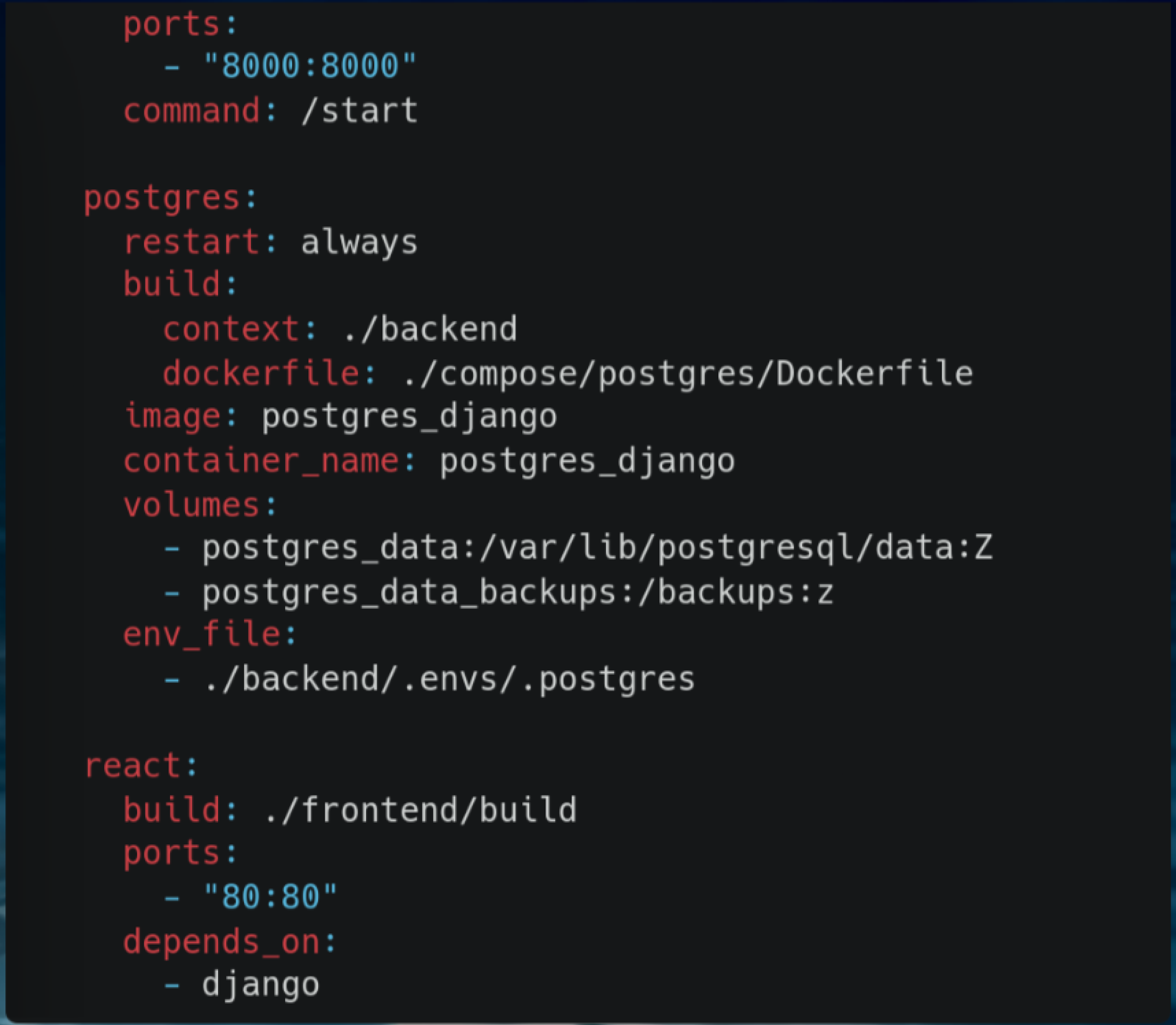
PostgreSQL

React

NGINX



```yaml
version: "3.9"

volumes:
  postgres_data: {}
  postgres_data_backups: {}

services:
  django:
    restart: always
    build:
      context: ./backend
      dockerfile: ./compose/django/Dockerfile
    image: api_v1
    container_name: api_v1
    volumes:
      - ./backend:/app/backend
    depends_on:
      - postgres
    env_file:
      - ./backend/.envs/.django
      - ./backend/.envs/.postgres
```



```yaml
    ports:
      - "8000:8000"
    command: /start

  postgres:
    restart: always
    build:
      context: ./backend
      dockerfile: ./compose/postgres/Dockerfile
    image: postgres_django
    container_name: postgres_django
    volumes:
      - postgres_data:/var/lib/postgresql/data:Z
      - postgres_data_backups:/backups:z
    env_file:
      - ./backend/.envs/.postgres

  react:
    build: ./frontend/build
    ports:
      - "80:80"
    depends_on:
      - django
```

THANK YOU FOR WATCHING

BE KING, FOR EVERYONE YOU MEET IS FIGHTING A BATTLE YOU KNOW NOTHING ABOUT.