Docker

Why and How it works?

Vladyslav Rudenko (@VVRud)

Docker

Docker is a set of platform-as-a-service (PaaS) products that use operating-system-level virtualization to deliver software in packages called **containers**.

Containers are *isolated* from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating-system kernel and thus more lightweight than virtual machines.

Docker container vs docker image

An instance of an **image** is called a **container**.

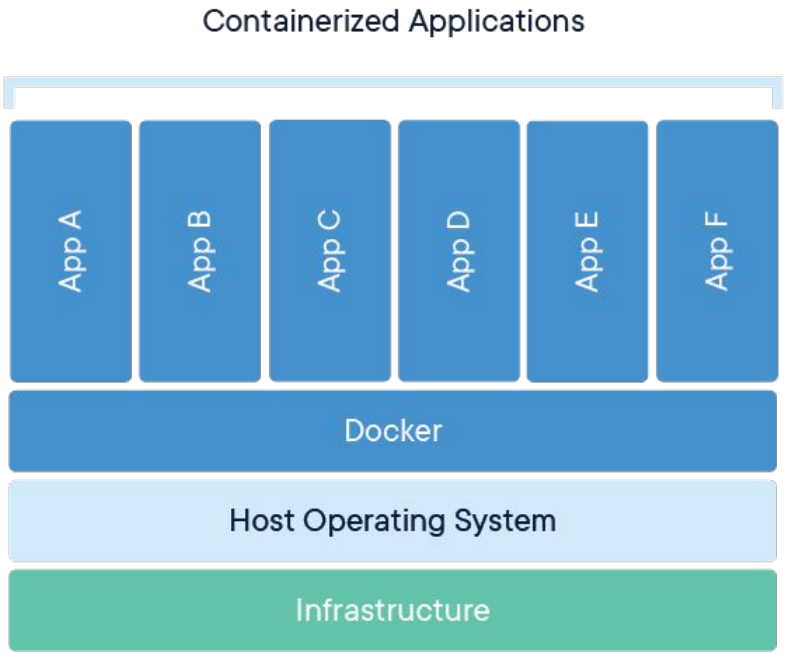
You have an **image**, which is a set of layers as you describe.

If you start this **image**, you have a running **container** of this image. You can have many running **containers** of the same **image**.

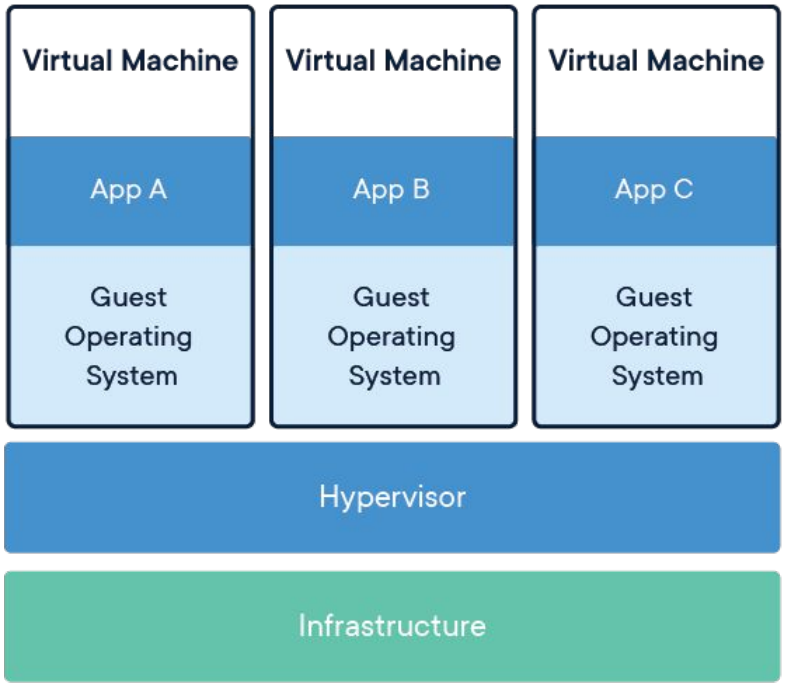
What problems Docker solves?

- missing or incorrect application dependencies such as libraries, interpreters, code/binaries, users;
- conflicts between programs running on the same computer such as library dependencies or ports;
- limiting the amount of resources such as cpu and memory an application can use;
- missing, complicated, or immature scripts to install, start, stop, and uninstall an application.

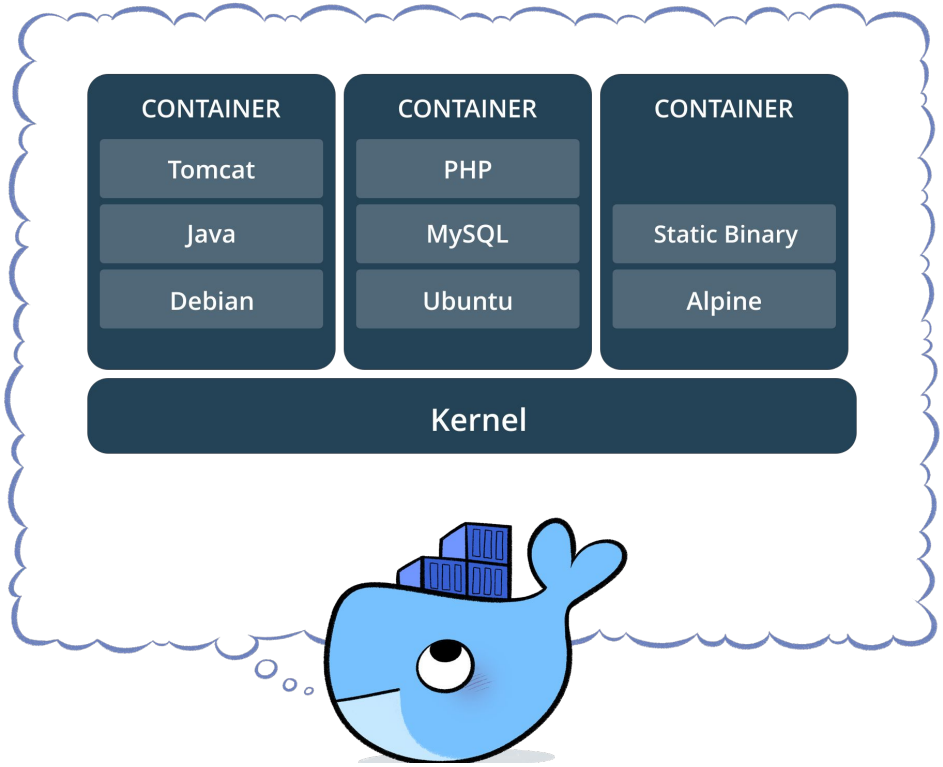
Docker



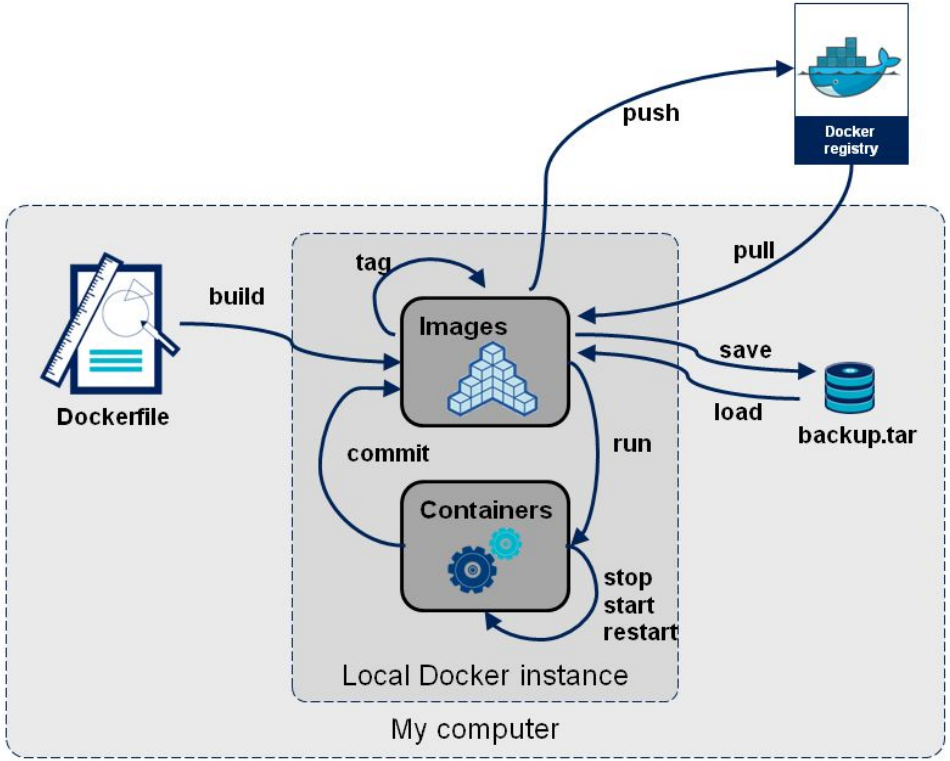
NOT Docker



Docker


CONTAINER
Tomcat
Java
Debian
CONTAINER
PHP
MySQL
Ubuntu
CONTAINER
Static Binary
Alpine
Kernel

Docker life cycle



Dockerfile example

Create an image with **python 3.7.3**

With environment variables **PG_HOST** and **SPLASH_IP** and if they are not set let them be **“localhost”**.

Copy file and **install** libraries it has.

Copy directory with code and **set** it **as a current** directory.

Run command to make file executable.

Run app from command line with parameters **PG_HOST** and **SPLASH_IP**.

```
FROM python:3.7.3

ENV PG_HOST     ${PG_HOST:-'localhost'}
ENV SPLASH_IP   ${SPLASH_IP:-'localhost'}

COPY requirements.txt /requirements.txt
RUN pip install -r requirements.txt

COPY . /crawl_app
COPY ./.env.prod ./crawl_app/.env
WORKDIR /crawl_app

RUN chmod +x ./production.sh

CMD ./production.sh ${PG_HOST} ${SPLASH_IP}
```

Docker commands example

```
docker build -t {container_name} {path_to_dockerfile}

docker run -p {port_outside}:{port_inside} {container_name}

docker ps

docker stop {container_id}

docker rm {container_id}

docker tag {container_name} {user_name}/{container_name}

docker push {user_name}/{container_name}
```

https://docs.docker.com/get-started/

Docker Compose

Docker Compose is a ***YAML file*** which contains details about the services, networks, and volumes for *setting up the Docker application*. So, you can use Docker Compose to create separate containers, host them and get them to communicate with each other.

Each container will expose a port for communicating with other containers.

The communication and up-time of these containers will be maintained by Docker Compose.

What problems Docker Compose solves?

- Managing multiple containers/volumes/networks;
- All application stack in one place;
- Easy to run/stop/debug;
- Versioning (in user-readable format).

Docker Compose example

```yaml
version: '3.5'

services:
  redis:
    image: redis:latest
    networks:
      - internal_net
    volumes:
      - './redis_conf/redis.conf:/usr/local/etc/redis/redis.conf:ro'
      - 'redis_data:/data'
    entrypoint: redis-server /usr/local/etc/redis/redis.conf
    restart: always

  postgres:
    image: postgres:latest
    environment:
      POSTGRES_USER: ${POSTGRES_USERNAME}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DATABASE}
    networks:
      - internal_net
    volumes:
      - 'postgresql_data:/var/lib/postgresql/data'
      - './postgres_conf/init_postgres.sql:/docker-entrypoint-initdb.d/init_postgres.sql:ro'
    restart: always
```

```yaml
volumes:
  redis_data: null
  postgresql_data: null

networks:
  internal_net:
    driver: bridge

```

Docker Compose example

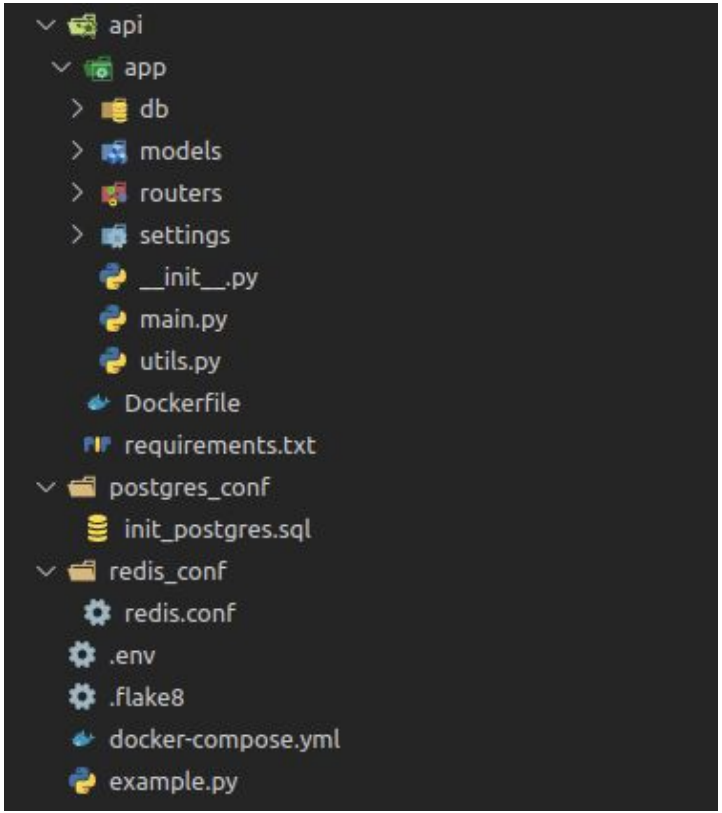
```
38    api:
39      build:
40        context: ./api
41        dockerfile: Dockerfile
42      environment:
43        POSTGRES_HOST: postgres
44        POSTGRES_PORT: 5432
45        POSTGRES_USERNAME: ${POSTGRES_USERNAME}
46        POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
47        POSTGRES_DATABASE: ${POSTGRES_DATABASE}
48
49        REDIS_HOST: redis
50        REDIS_PORT: 6379
51      networks:
52        - internal_net
53      ports:
54        - '8090:80'
55      depends_on:
56        - redis
57        - postgres
58      restart: always
```

```
60    volumes:
61      redis_data: null
62      postgresql_data: null
63
64    networks:
65      internal_net:
66        driver: bridge
67
```

Docker Compose commands example

```
docker-compose build

docker-compose up -d

docker-compose stop

docker-compose down -v --remove-orphans

docker-compose rm

docker-compose start
```

WORKSHOP TIME!



Project structure as follows:

- Configurations;
- Apps;
- Variables;

ALL IN ONE PLACE!

Thank you!

God saves the Queen.

Docker saves your time, money and nerves.