

clone the repo

```
git clone git@github.com:mankings/workshop-docker.git
```

Docker 101

workshop

Miguel Matos

What is?

The Problem

any code that runs perfectly fine on your computer
will fail to run on somebody else's

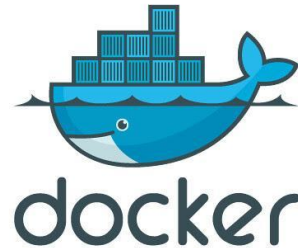


Containers

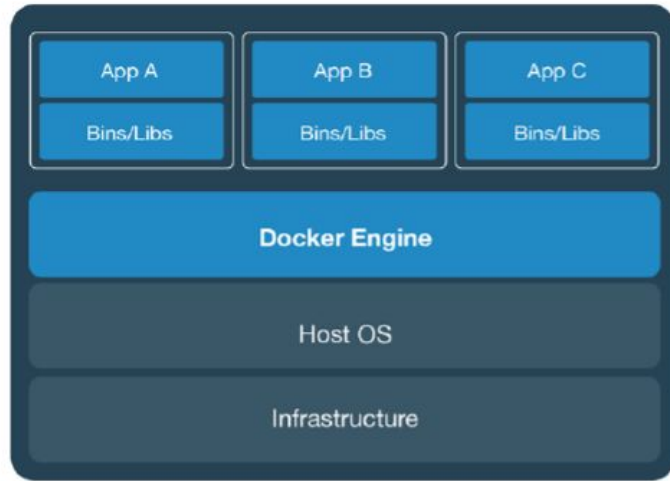
Container

- Lightweight, executable units of software that contain everything needed to run an application.

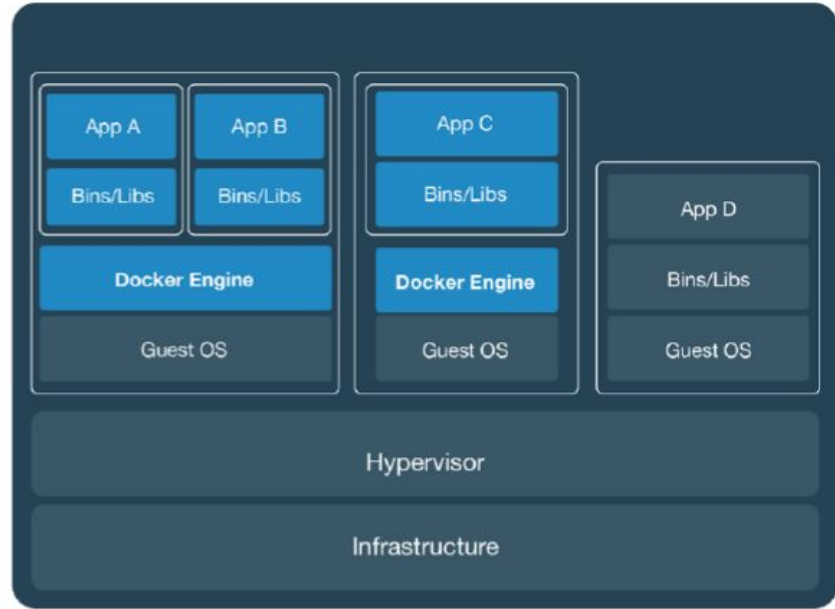
Docker is a platform designed to make it easier to create, deploy, and run applications by using containers.



Containers vs VMs



Containers



Virtual Machines

Why containers

Consistency

- "Works on my machine" issue resolved.

Isolation

- Applications run in separate containers without affecting each other.

Portability

- Run anywhere: on-premises, cloud, or hybrid environments.

Efficiency

- Lightweight compared to traditional VMs, uses fewer resources.

Docker real-world uses

- Netflix
- Spotify
- Pinterest
- The New York Times
- ...

- IES
- TQS
- CBD
- SIO
- TPW
- ...

How does it work?

Containers and Images



container



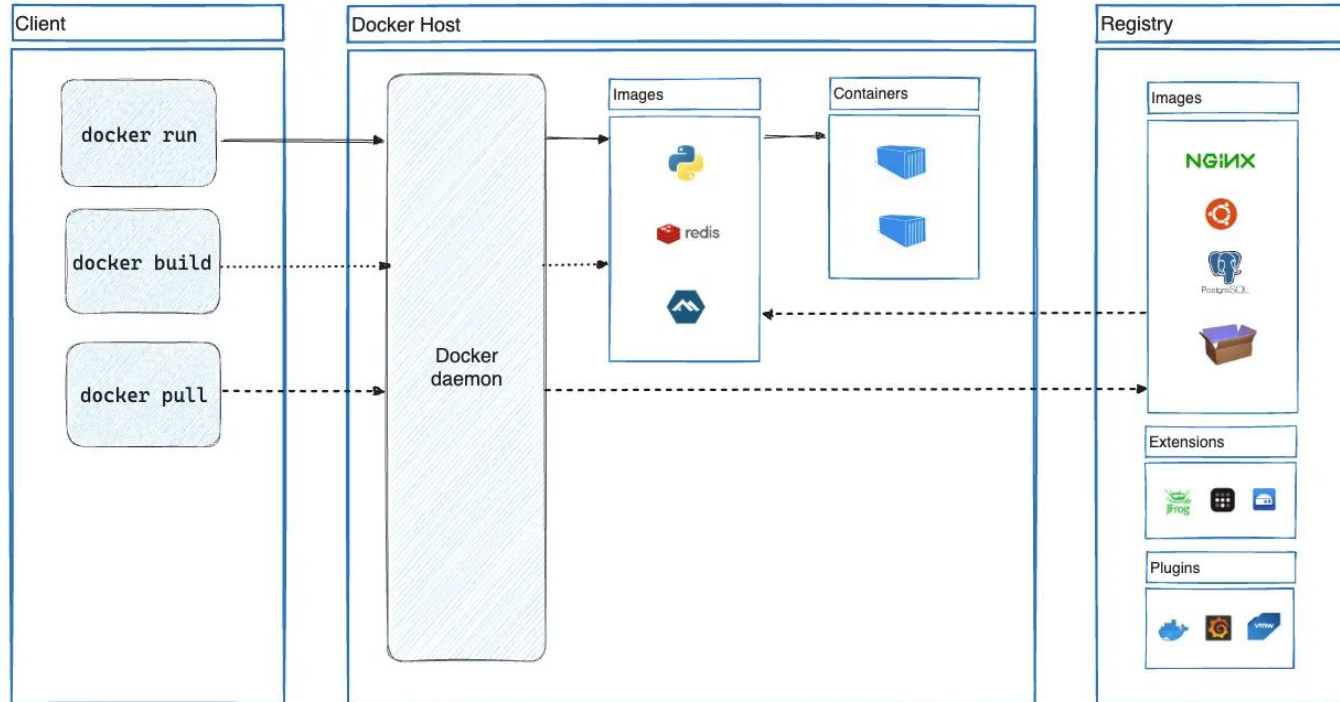
image

Dockerfile

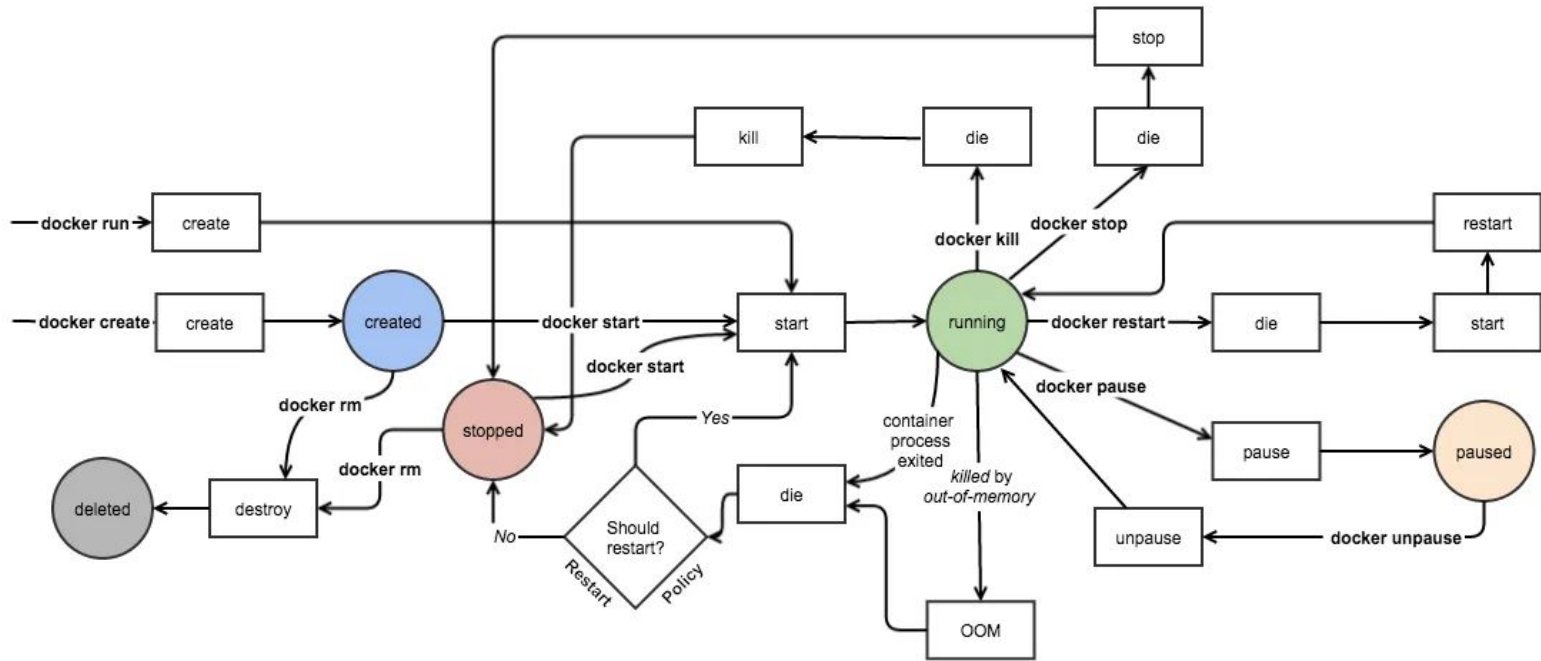
1. Base image
2. Modifications
3. Start command

```
1-app > Dockerfile > ...
1  # Base image
2  FROM python:3.8-alpine
3
4  # Image labels/tags
5  LABEL maintainer="Miguel Matos"
6
7  # Execute the command to create a folder for the application code
8  RUN mkdir /app
9
10 # Define the working directory inside the Docker container
11 WORKDIR /app
12
13 # Copy external files into the working directory
14 # In this case, copying python requirements file
15 COPY requirements.txt requirements.txt
16
17 # Install dependencies with pip
18 RUN pip install -r requirements.txt
19
20 # Copy the external code into the working directory
21 COPY app.py app.py
22
23 # Expose the port of the service
24 EXPOSE 8080
25
26 # Define the command to run when the container starts
27 CMD ["python3", "app.py"]
```

Docker Architecture



Container Lifecycle



Docker Basics

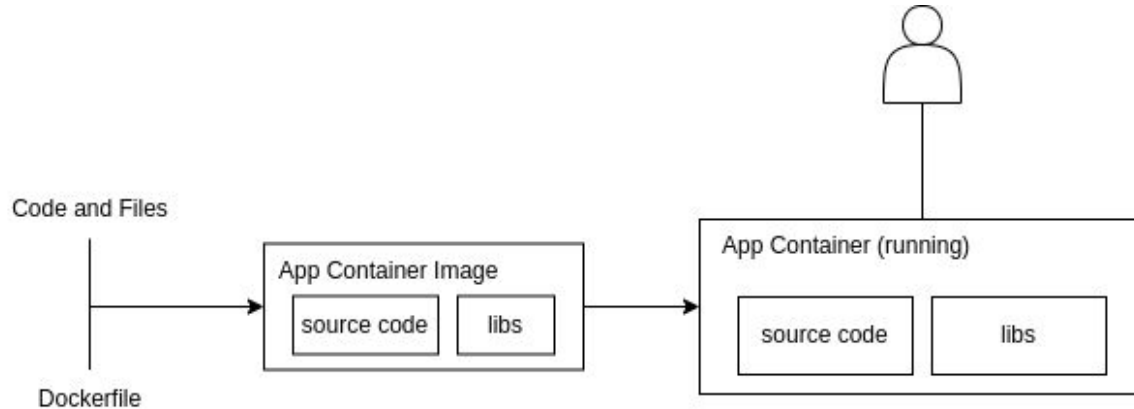
Basic Commands

- `docker pull` - pull an image from a registry
- `docker run` - run a container
- `docker ps` - list running containers
- `docker stop` - stop a container
- `docker rm` - remove/delete a container
- `docker build` - build an image
- `docker rmi` - remove an image
- `docker exec` - execute a command inside a container
- `docker logs` - check or follow the logs of a container

Hands-on!

Example 1 - Simple Application

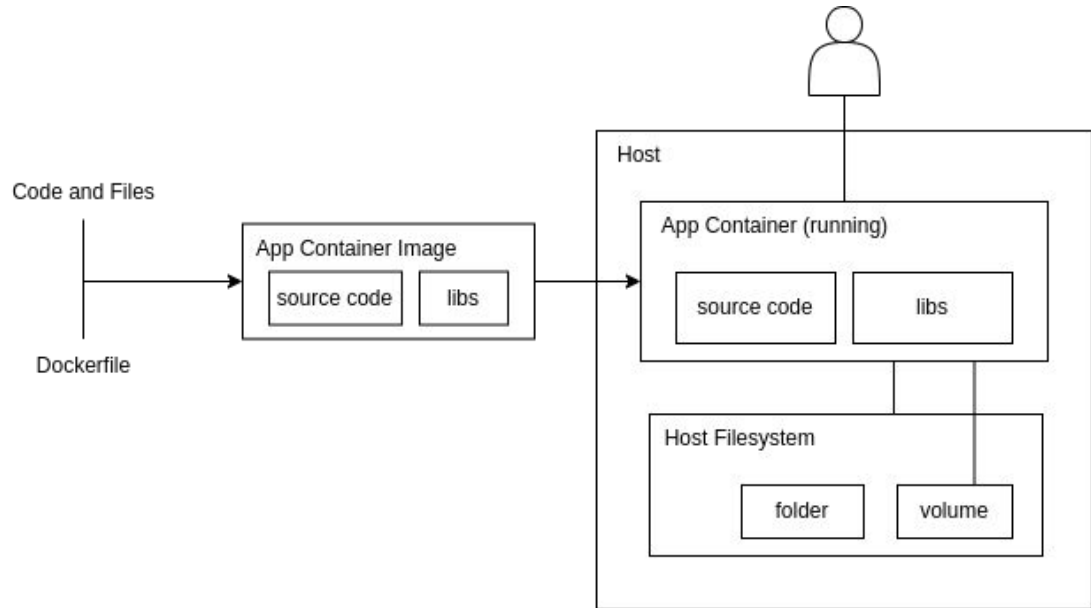
- *app.py* - application source code
- *requirements.txt* - required libraries/packages
- *Dockerfile* - instructions to build the container image



Example 2 - Persistence

Containers are ephemeral - data within them is lost upon stop or destruction

- Volumes
- Bind mounts



Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications.

Uses .yaml configuration files, defining the following:

- Services
- Networking
- Volumes
- Configuration

docker-compose.yml file

```
3-compose > 🚀 docker-compose.yml > {} services > {} nginx
docker-compose.yml - The Compose specification establishes a standard for the definition of multi-container platf
1  version: "3.8"
   ▶ Run All Services
2  services:
   ▶ Run Service
3      app:
4          image: fastapi-app
5          build:
6              context: .
7              dockerfile: Dockerfile.app
8          volumes:
9              - ./www:/app/www
10
   ▶ Run Service
11     nginx:
12         image: nginx-proxy
13         build:
14             context: .
15             dockerfile: Dockerfile.nginx
16         ports:
17             - 8081:80
```

Docker Compose basics

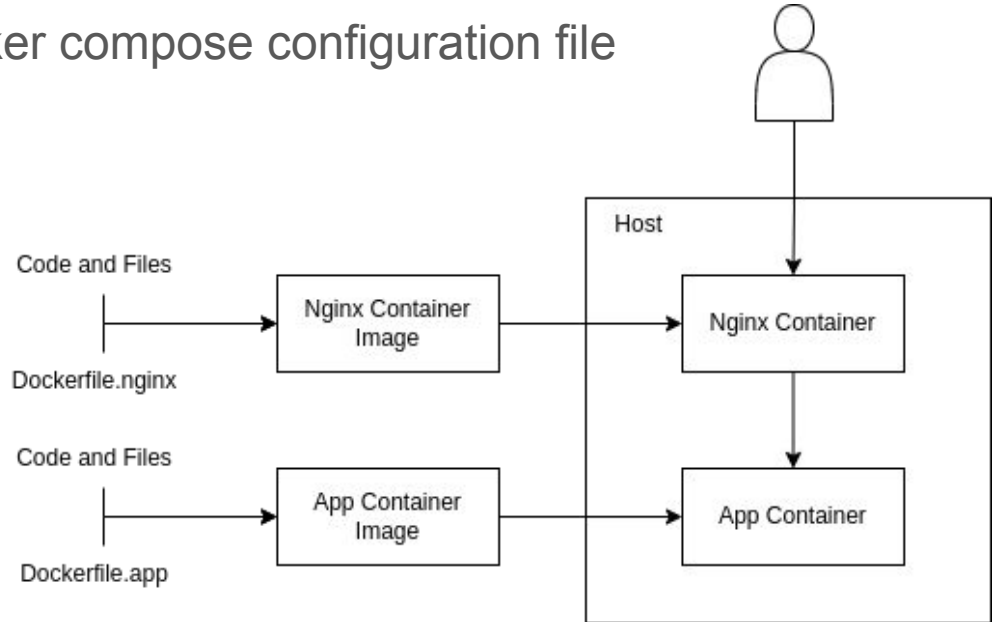
Basic Commands

- `docker compose up [-d]` - launch services
- `docker compose down` - stop services

- `docker compose ps` - list services
- `docker compose pull` - pull the image of a service
- `docker compose build` - build service images
- ...

Example 3 - Multi-container Application

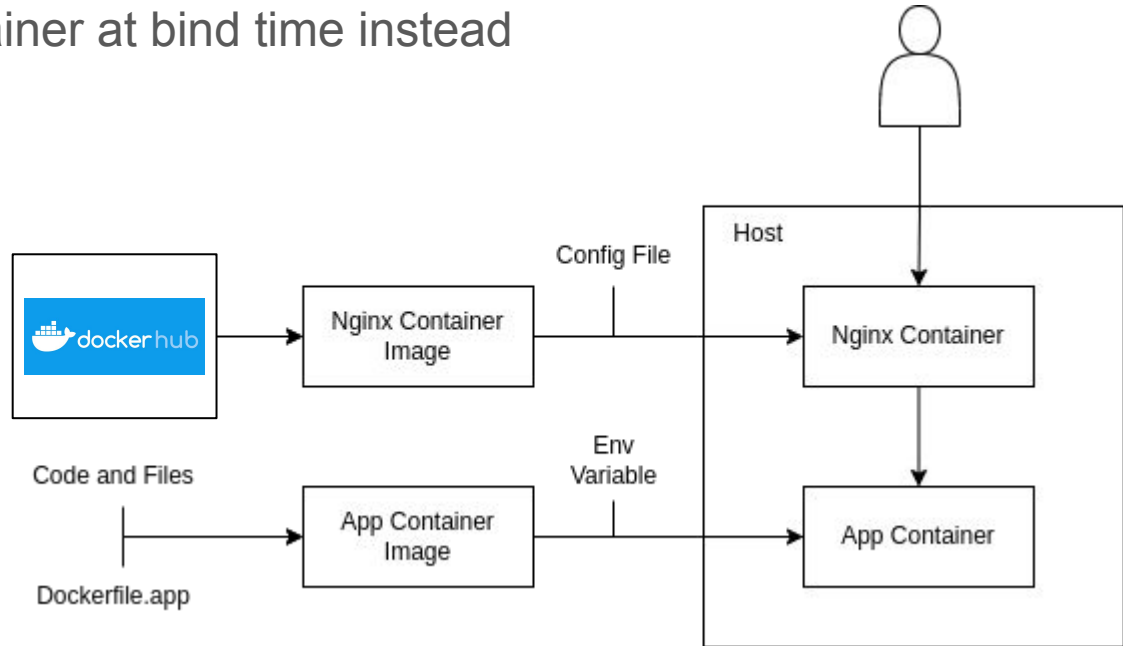
- *Dockerfile.app* - main application Dockerfile
- *Dockerfile.nginx* - Nginx service Dockerfile
- *docker-compose.yml* - docker compose configuration file



Example 4 - Configs

Used to manage data required by the application at runtime.

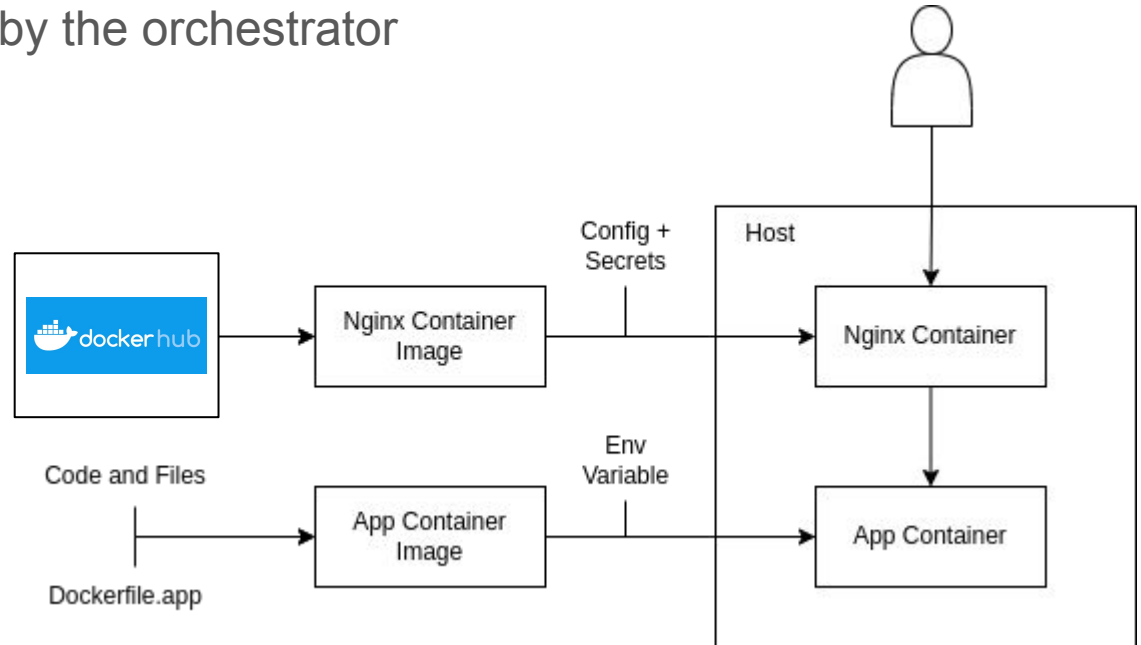
Add configuration to container at bind time instead of build time.



Example 5 - Secrets

Used to manage sensitive data required by the application at runtime.

Controlled and encrypted by the orchestrator



Conclusion



Docker Lunchable:

- Your trash code
- Code Dependencies
(frameworks/libraries, databases, caching, etc)
- Environment settings
- Everything else you need to run your trash code

[source](#)

- Docker simplifies the process of building, sharing, and running applications through containerization
- Containers offer consistency, portability, and efficiency across environments

Thanks!

questions?

and try to remember to delete your old images :)