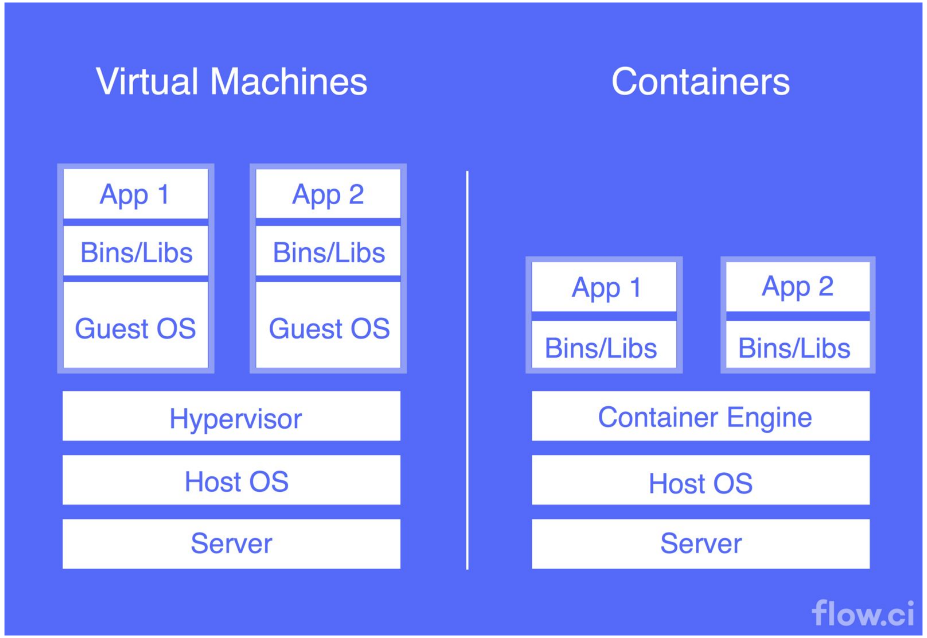
Dockers

Alejandro Jurnet
Pau Marcer

Introduction to Dockers

- Operating system level virtualization.
- Pack the minimum set of tools to allow us to create virtualized apps.
- Run directly onto the operating system, there is no hypervisor.



Introduction to Dockers

- Dockers are based on **images** and **containers**.
- Images can be build with various methods from source code.
- Containers are the running image on the docker machine.
- This Structure allows any image to be runned as a container on any docker capable machine without the need of modifying the code, thus docker allows our applications to be portable.
- Docker allows for application dependencies, so we can have an app distributed in various images.
- There is a centralized repository where we can obtain a huge variety of images, then depending on our needs we can choose the most appropriate for our case

Introduction to Dockers

Pros/Cons

- Good for deploying a lot of services on the same machine with low overhead cost.

- Dockers doesn’t require any hypervisor, this allows for smaller images, and faster startup/replication time. But this affects the way the container communicates to the OS, restricting the possible operations.

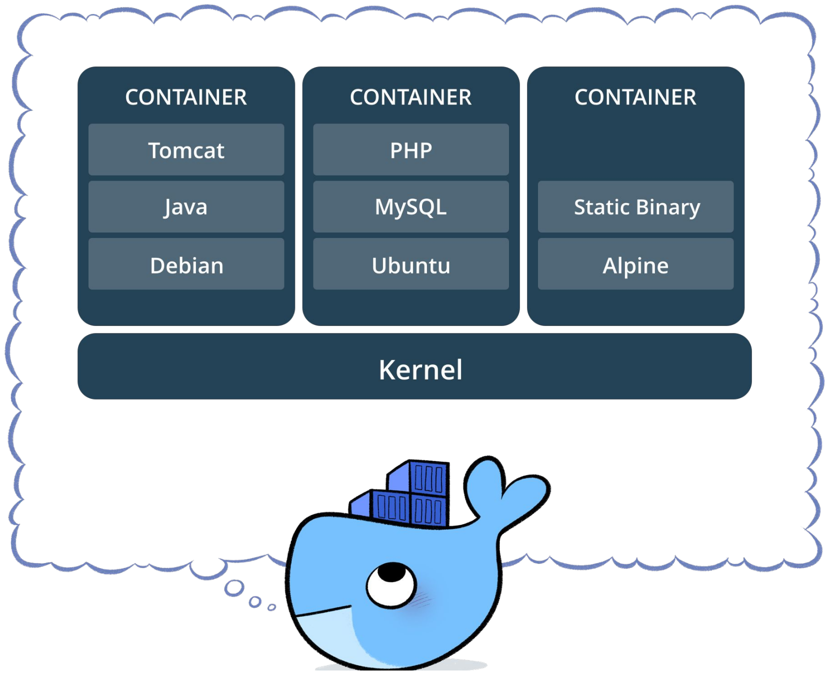
- Dockers share the OS, hypervisors have a copy of the OS, so the isolation is worst.

- Docker linux images are restricted to linux and cannot be used on windows.

Dockers: Container

- Configured when created from the base image (static) OR configured when starting an image (dynamic)

- Containers are stateless (no data is saved when the container stops).



Dockers: Container

- Containers can be stacks of images(we can have multiple base images in our container).
- Containers can have different level of isolation
  - Network level with private virtual subnets
  - Hardware level with privileged access.
- No need to use the whole OS if not needed by our app.

Dockers: Images

Docker images are the most elemental of docker structures

- Read only template with the instructions on how to build the container and the data.
- Can be obtained from public or private repositories.
- Compatible between docker hosts with the same architecture (unix -> unix)
- Usually provide a full service (SQL, Web Server).

Dockers: Images

When building a new image we always will use lots of images

- Every new file or modification that we add into the final image will be a new image (containing just the new data) that will be added on top of the base one.

Images are build with Dockerfiles

Dockers: Dockerfile

Dockerfiles are a set of commands that allow the user to configure new build images

- **ENV**: Set an environment variable in the container.
- **EXPOSE**: Expose a port or range of the container.
- **FROM**: Base image of the container.
- **WORKDIR**: Base directory to work
- **MKDIR**: Create a directory in the container.
- **RUN**: Run a command in the default shell.
- **CMD**: Run a command without shell. Also is used normally to specify the application we want to run.

Dockers: Dockerfile

Dockerfile only allows us to specify the image build and base run point

- Dockerfile does not specify how the image will be run
  - Network, volumes

- We need Docker Compose for that.

Dockers: Docker compose

Docker Compose allows for:

- Run multiple images, from repos or Dockerfiles at once.
- Assign dependencies between containers.
- Restart containers on failure.
- Create and attach the containers to our private networks.
- Control the open ports and bindings between containers.
- Publish ports in order to open the access to a service.

Dockers: Docker compose

Docker compose steps:

1. Define your app’s environment with a Dockerfile so it can be reproduced anywhere.
2. Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
3. Run docker-compose up and Compose starts and runs your entire app

Network

The docker networking is fully configurable, we can use the existing networks or create new networks, the main features of the network are defined by the used driver.

- Bridge:
  - Usually used when your applications run in standalone containers that need to communicate
  - Creates an interface between the outside world and the container.

Network

- Host:
  - Use the host’s networking directly, meaning that they share the IP and reserved ports.
  - Some precautions must be take to avoid collisions with same ports in the host and container.

- Overlay:
  - Containers running on different Docker hosts to communicate, or when multiple applications work together using swarm services

Swarm

Docker machines can be setup in swarm mode in order to create a cluster of dockers and share resources.

- On swarm mode, we can run services instead of containers, a service is a container replicated N times on the swarm.

- The swarm overlay type networking is capable of reaching the container/service on any of the docker machines of the swarm.

- Swarm is build with manager and worker nodes.

Quick start to dockers

- Running a container:
  - $ docker run --name debian-it -it debian
    - This runs a new debian container with the given name and an interactive shell.
- Creating a volume
  - $ docker volume create my-vol
- Mounting a volume when creating a container
  - $ --mount source=my-vol,target=/app
- To create a network
  - $ docker network create --driver bridge --subnet=192.168.0.1/24 --gateway=192.168.0.1 --ip-range=192.168.0.128/24 [my_network]

Quick start to dockers

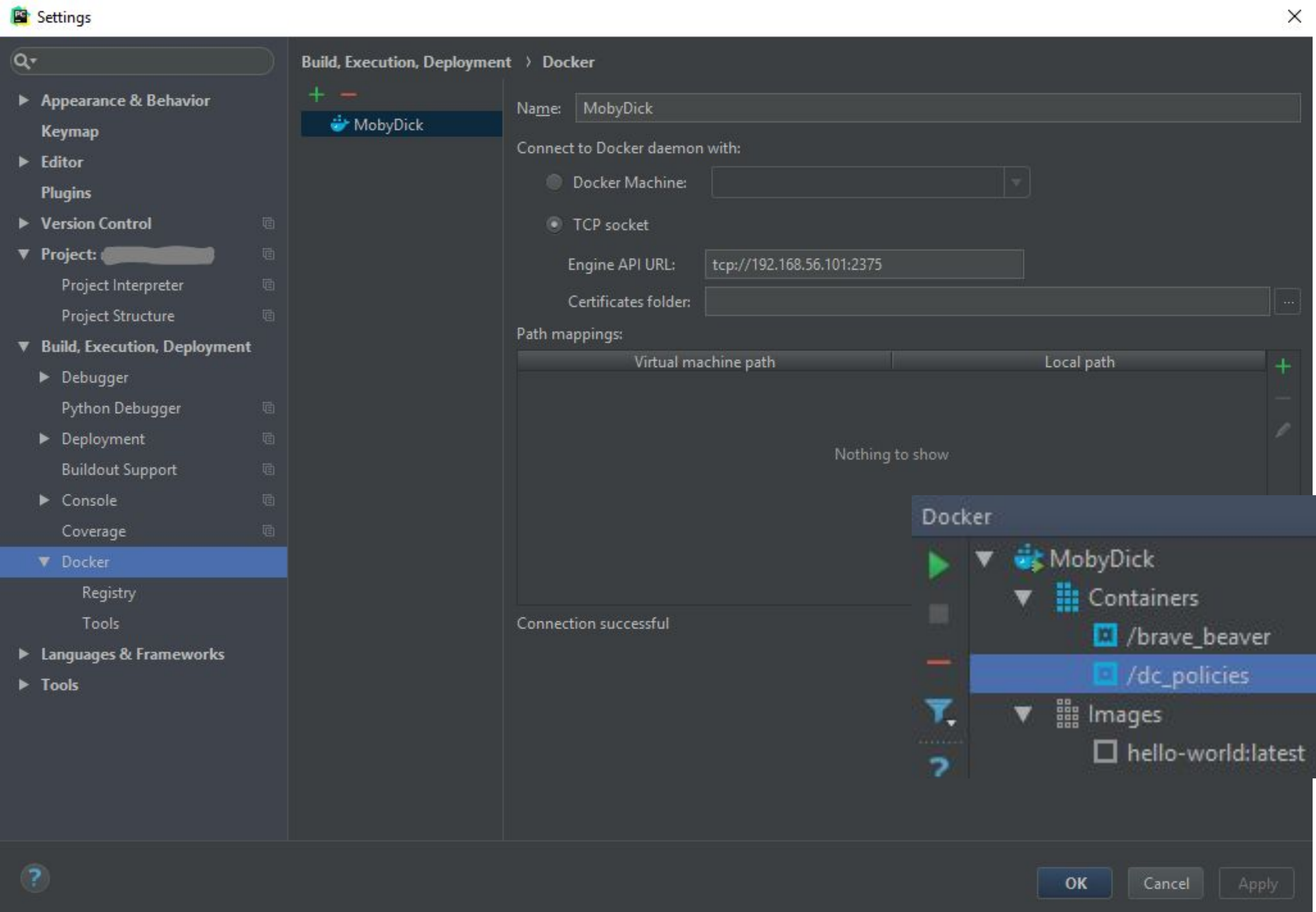
Dockerfile:

```
FROM python:3

ADD script.py /

ADD requirements.txt

ENV env=test

RUN pip install -r requirements.txt
```

Quick start to dockers

```
version: '3'
    services:
    db:
    image: postgres
    web:
    build: .
    command: python3 manage.py runserver 0.0.0.0:8000
    volumes:
        - .:/code
    ports:
        - "8000:8000"
    depends_on:
        - db
```

Pycharm plugin



Settings

Build, Execution, Deployment › Docker

MobyDick

Name: MobyDick

Connect to Docker daemon with:

Docker Machine:

TCP socket [✓]

Engine API URL: tcp://192.168.56.101:2375

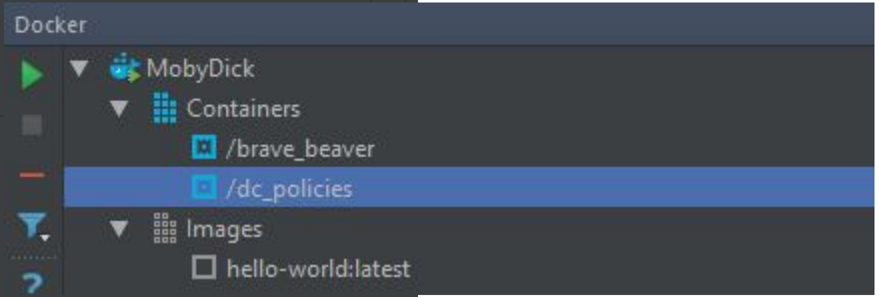
Certificates folder:

Path mappings:

Virtual machine path	Local path

Nothing to show

Connection successful



Docker

```
MobyDick
  Containers
    /brave_beaver
    /dc_policies
  Images
    hello-world:latest
```

Conclusions

- Dockers are very useful to deploy microservices and create some distributed environments.
- Very easy and well documented.
- Multi-platform and huge community.
- It's little difficult to start using dockers.
- Strong technology that is growing.