Introduction to Containerization

Containerization

- A container includes an application and its dependencies.
- **Easy to ship** and handle!

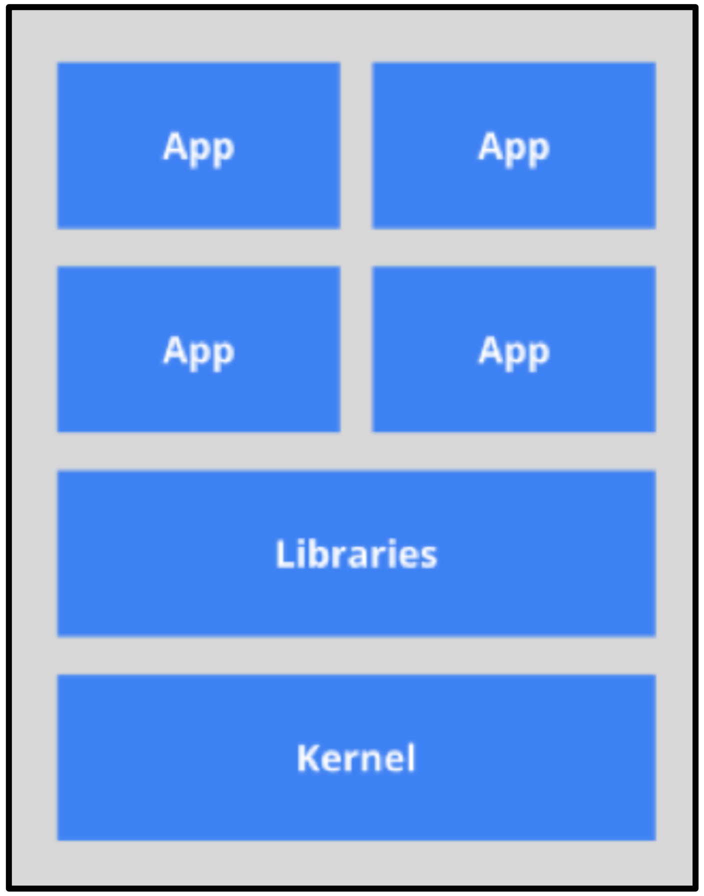
ITXU 109696 1
22G1
MAX. GROSS 30.480 KGS 67.200 LBS
TARE 2.185 KGS 4.820 LBS
NET 28.295 KGS 62.380 LBS
CU. CAP. 33.2 CU.M. 1.173 CU.FT.

Docker containers

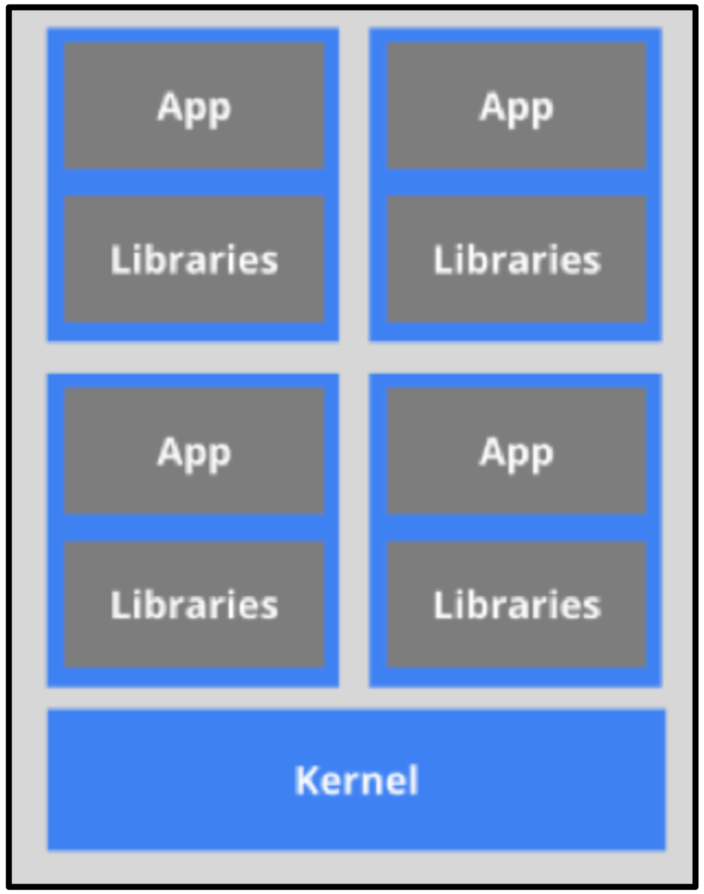
- **Docker containers** wrap up an application in a filesystem containing everything the application needs to run:
  - code
  - runtime libraries
  - system tools
  - configuration files
- The containerized application **will run identically** on any host.
  - 😀 no incompatibilities of any kind!

Why containerization

Traditional way



Using containers



Package manager installs apps.
Apps share libraries.

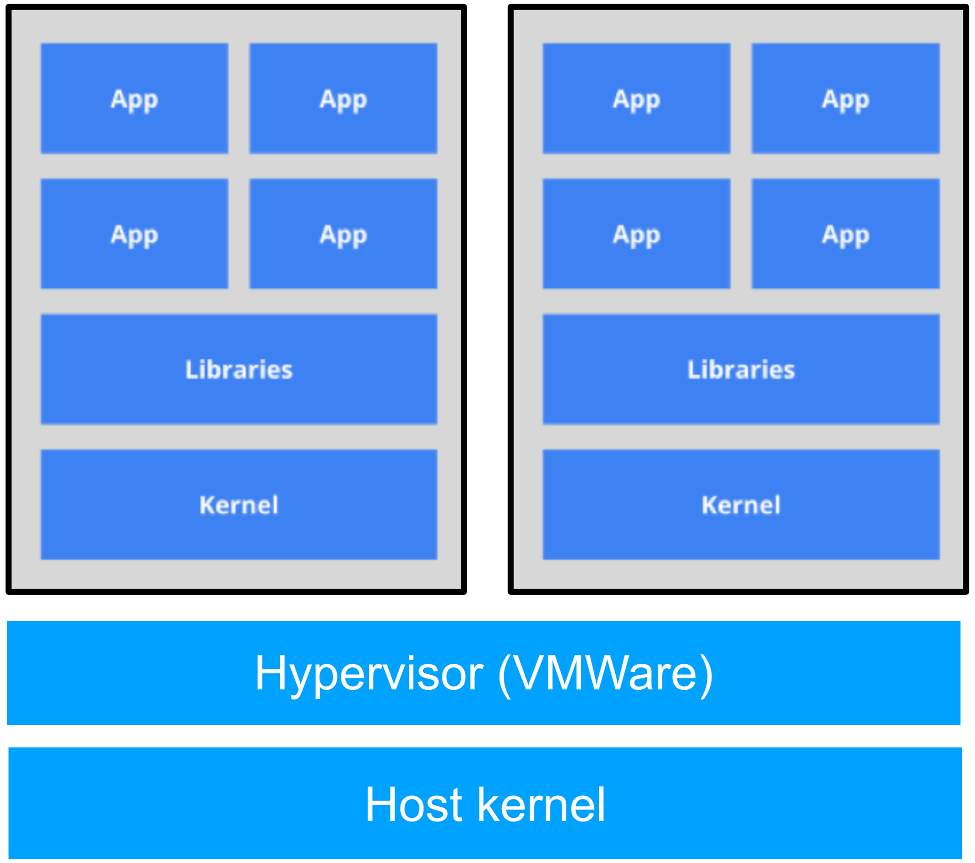
☹ Compatibility issues.

😀 Each container has its own libraries.

😀 Each container can be updated independently.

Difference between containerization and virtual machines

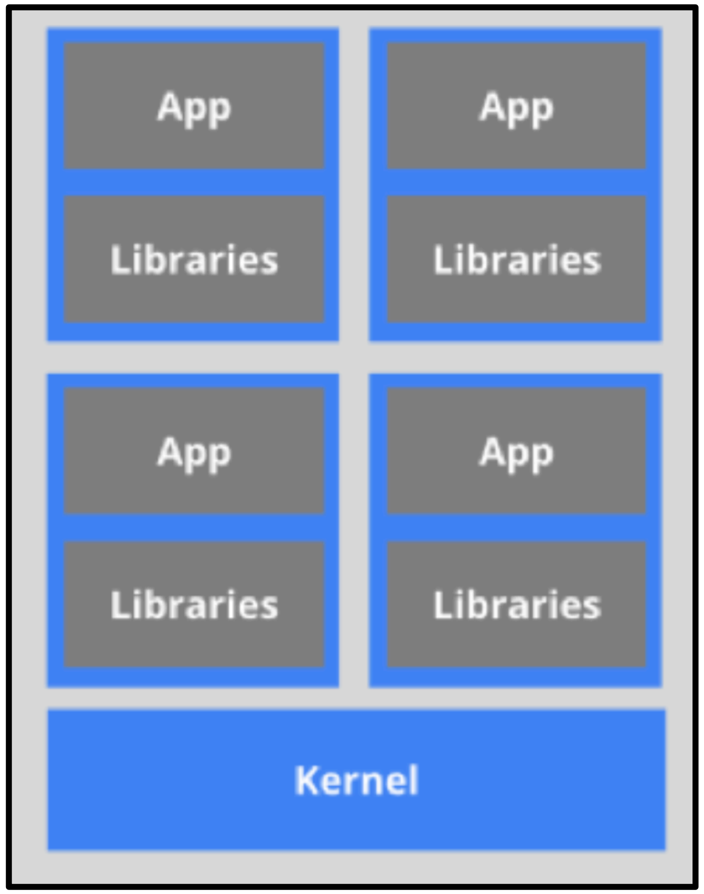
Virtual machines



☹ Large overhead

☹ Apps cannot communicate

Using containers

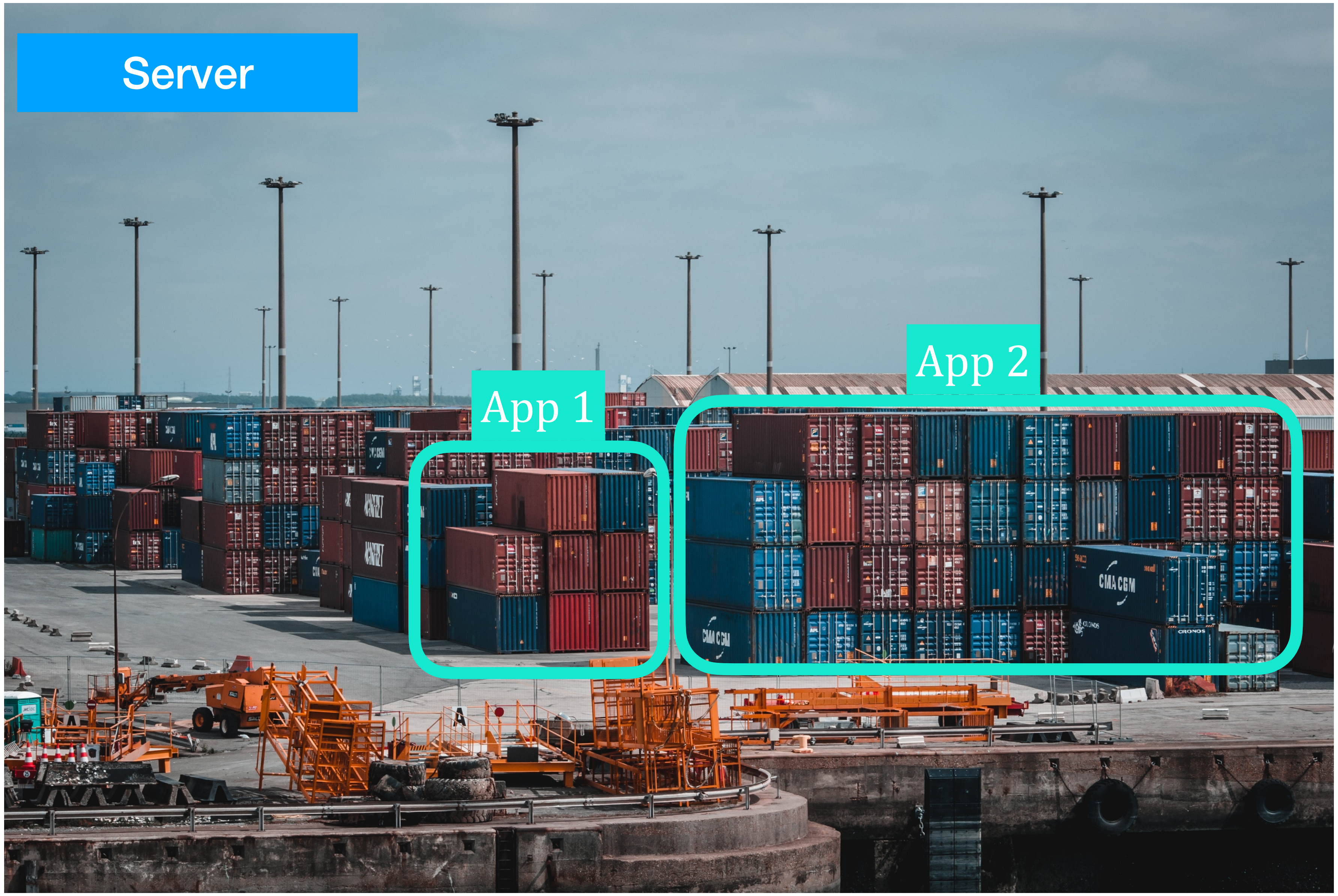


😀 Small overhead

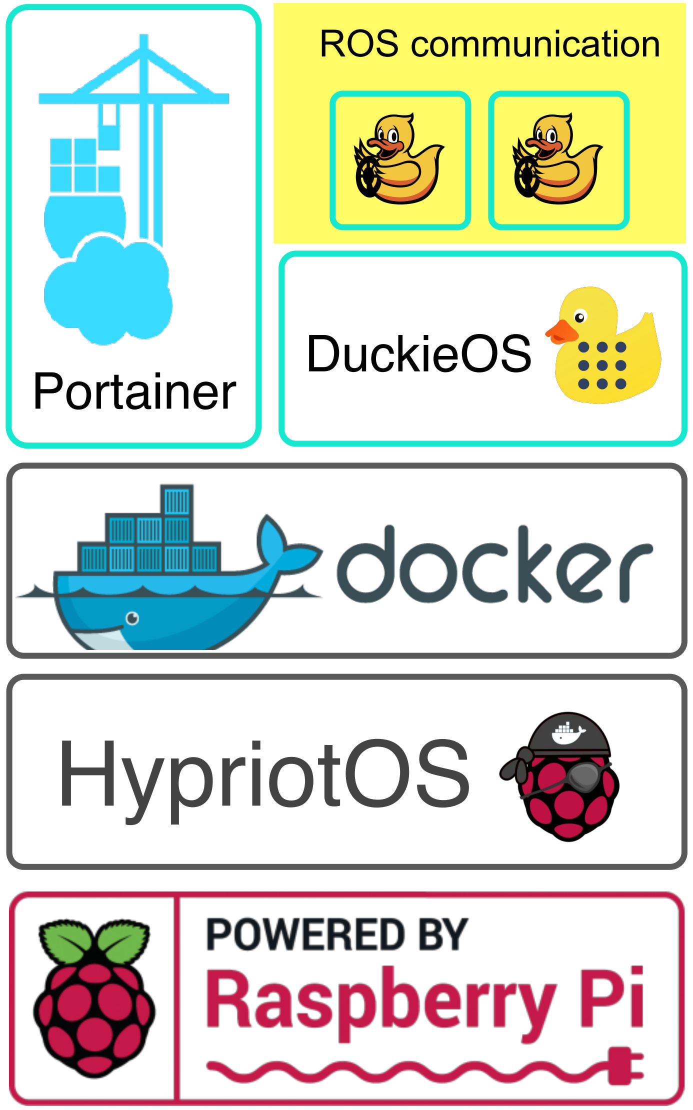
😀 Apps can communicate

Modern applications with containerization

- Modern application are organized in **stacks of containers working together.**



What ran on the Duckiebot 19



Portainer

ROS communication

DuckieOS

docker

HypriotOS

POWERED BY Raspberry Pi

containers

Demos and ROS nodes

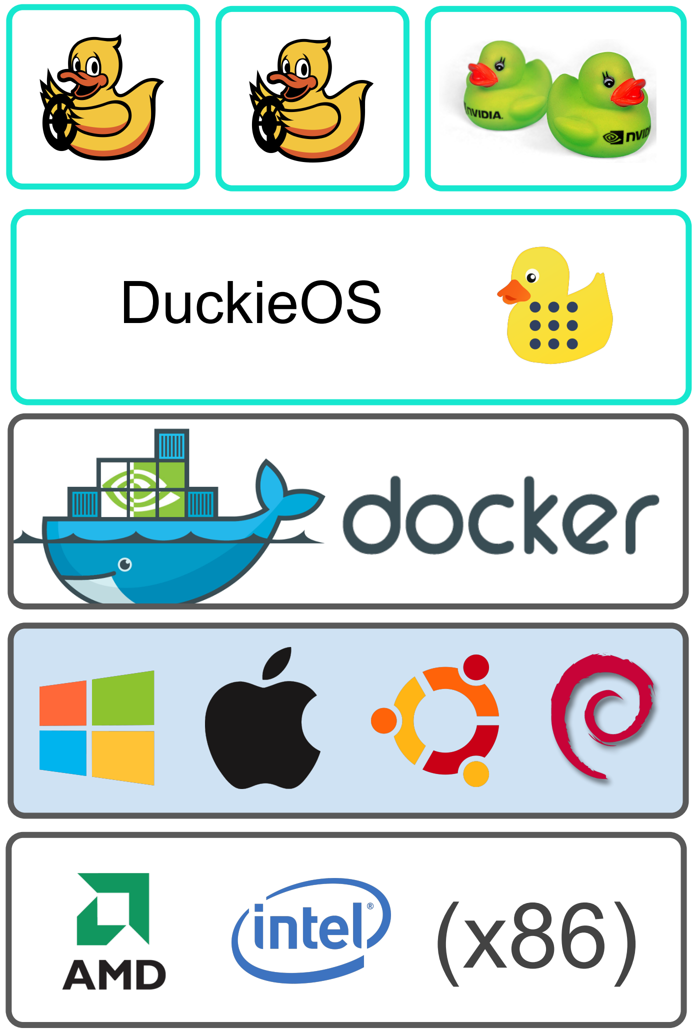
Base system based on Ubuntu

Docker Layer

Lightweight base operating system

ARM-based single board computer (SBC)

What runs on the laptop



Demos and ROS nodes

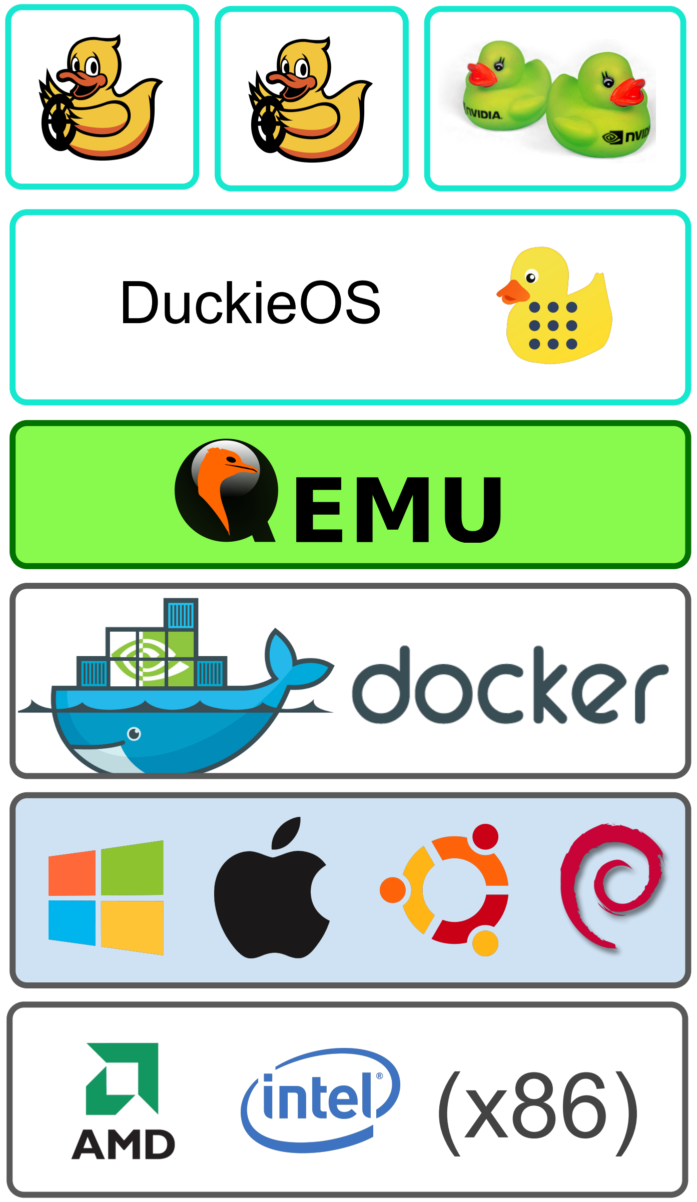
Based on Ubuntu

Docker layer

Any major OS (Windows/MacOS/Linux)

Any x86 compatible architecture

Running ARM code on the laptop



Demos and ROS nodes

Based on Ubuntu

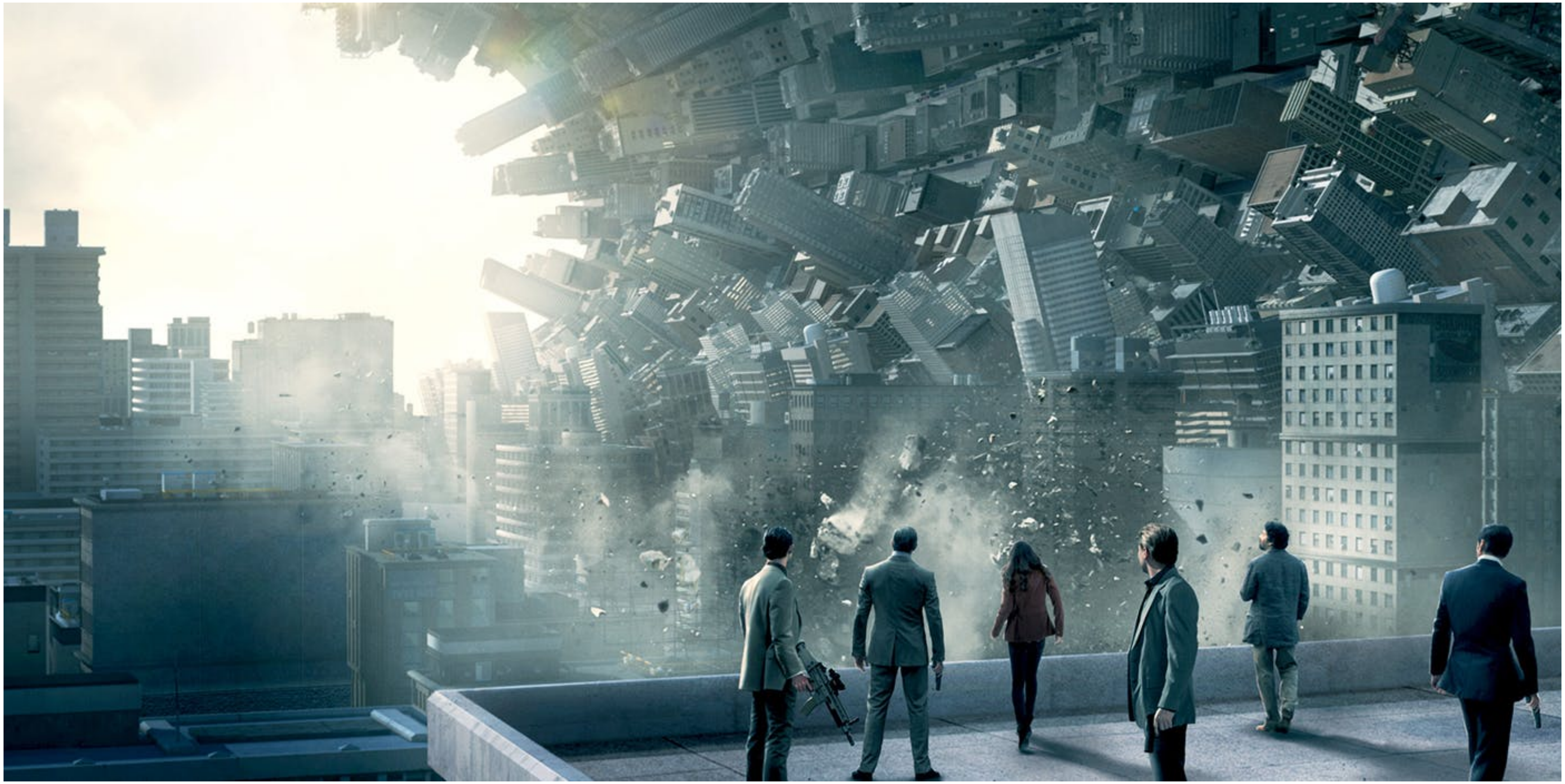
ARM32v7 emulator

Docker layer

Any major OS (Windows/MacOS/Linux)

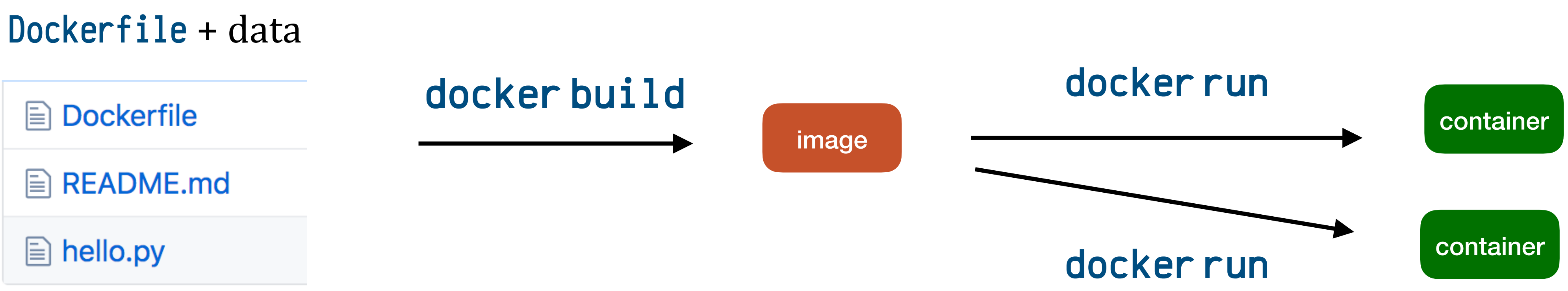
Any x86 compatible architecture

A dream in a dream in a dream in a dream...



Docker workflow overview

- Simplest workflow:
  - `docker build` - Builds an **image** from a `Dockerfile`.
  - `docker run` - Creates a **container** from an **image** and runs it.



`Dockerfile` + data

Dockerfile

README.md

hello.py

`docker build`

image

`docker run`

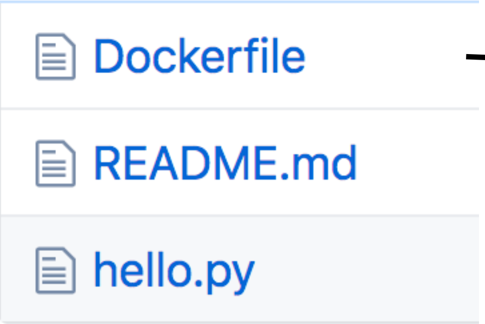
container

`docker run`

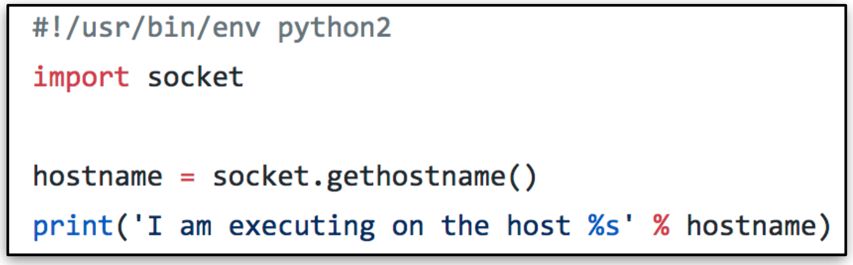
container

The recipes to create images

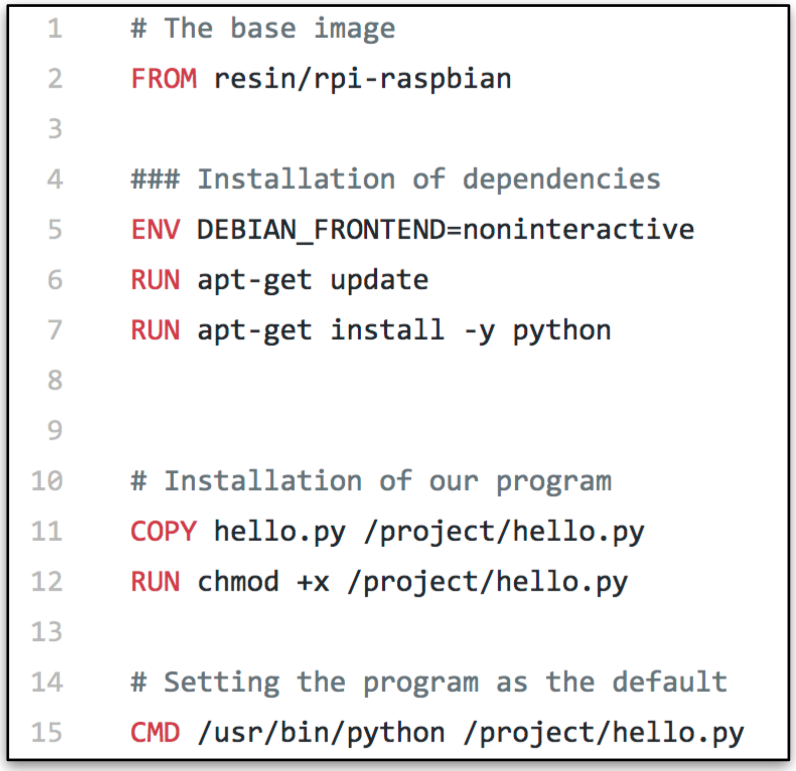
- The `Dockerfile` is the “recipe” to build a Docker **image**.



Dockerfile
README.md
hello.py



```python
#!/usr/bin/env python2
import socket

hostname = socket.gethostname()
print('I am executing on the host %s' % hostname)
```



```dockerfile
# The base image
FROM resin/rpi-raspbian

### Installation of dependencies
ENV DEBIAN_FRONTEND=noninteractive
RUN apt-get update
RUN apt-get install -y python


# Installation of our program
COPY hello.py /project/hello.py
RUN chmod +x /project/hello.py

# Setting the program as the default
CMD /usr/bin/python /project/hello.py
```

FROM declares the base image.

RUN runs a command

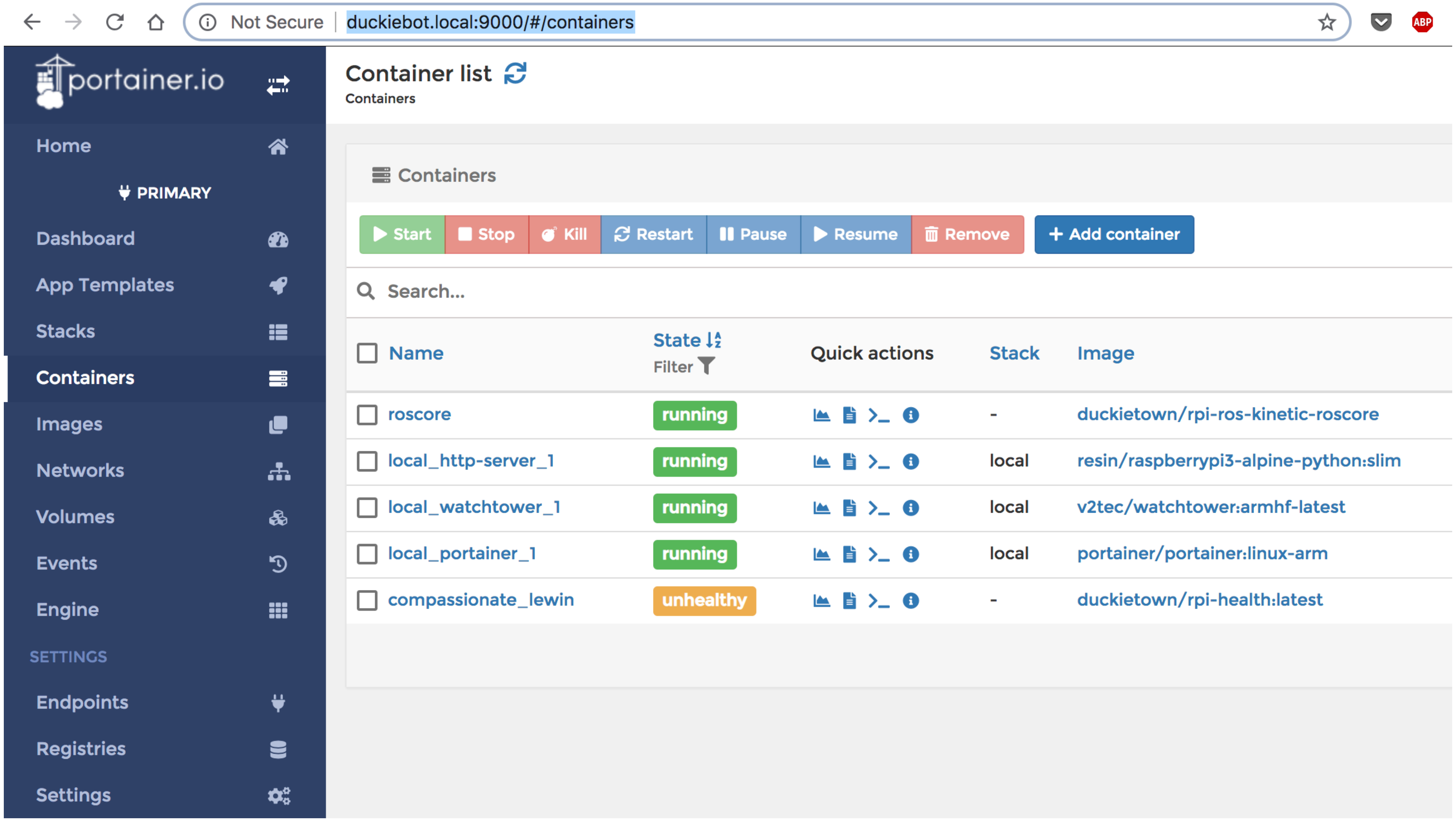
COPY files into the image

CMD declares what is the default command.

- It’s like you are **recreating an entire OS** inside the image.
  - You can *pin* dependencies.
  - No other program will mess with your environment.

Portainer

- Portainer allows to see which containers run on a host.



Container list

Containers

Containers

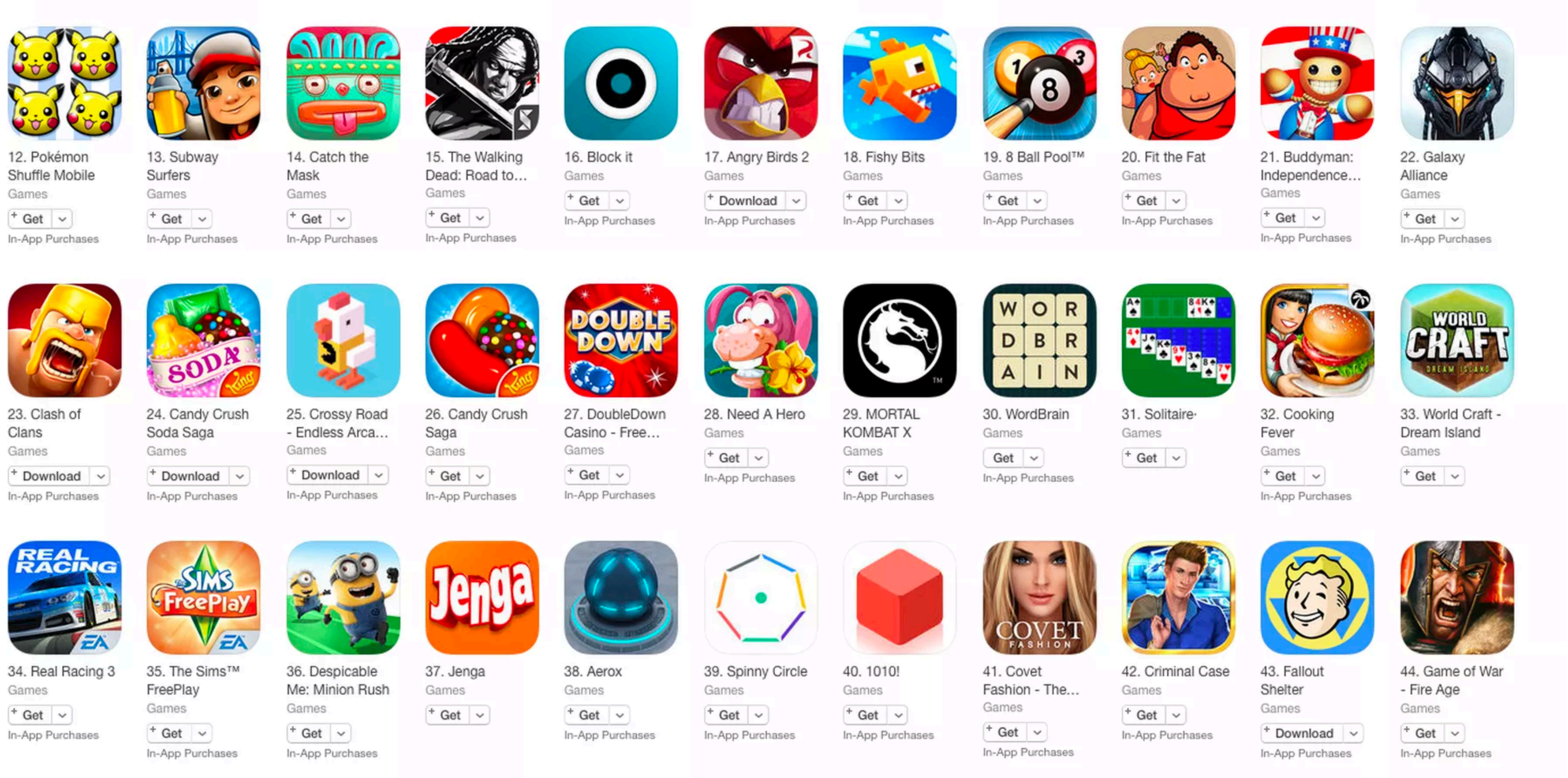
□	Name	State Filter	Quick actions	Stack	Image
□	roscore	running		-	duckietown/rpi-ros-kinetic-roscore
□	local_http-server_1	running		local	resin/raspberrypi3-alpine-python:slim
□	local_watchtower_1	running		local	v2tec/watchtower:armhf-latest
□	local_portainer_1	running		local	portainer/portainer:linux-arm
□	compassionate_lewin	unhealthy		-	duckietown/rpi-health:latest

Docker registries: Sharing is caring

- **Docker registries** are online databases of Docker images that anybody can use.
- The largest public registry is *Dockerhub*.
- You can run your own (private or public).

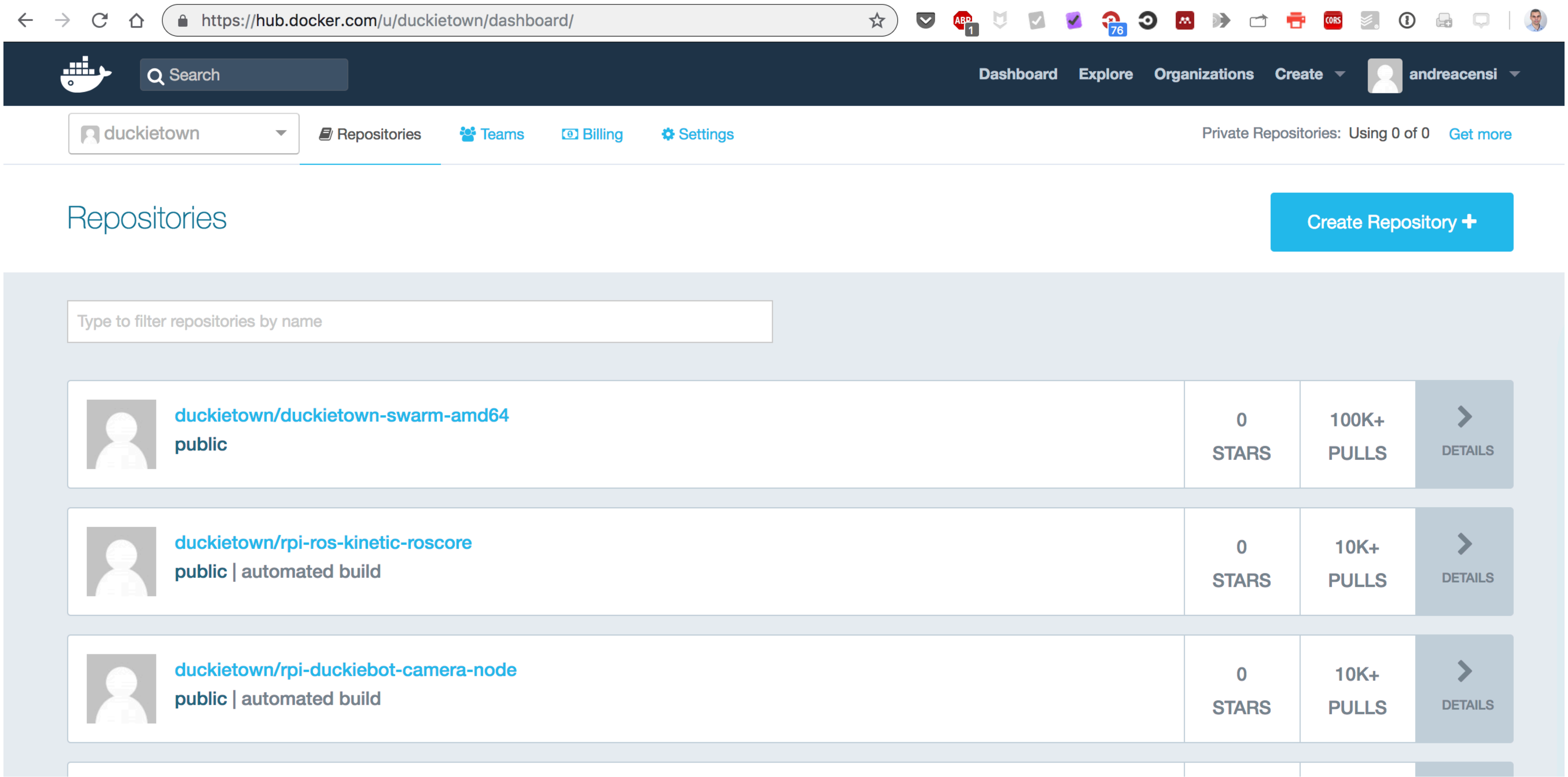
Docker registries

- Very similar to an “app store” used by servers.



Dockerhub

- Everybody can publish images for the world to use.
- You can browse the available images.



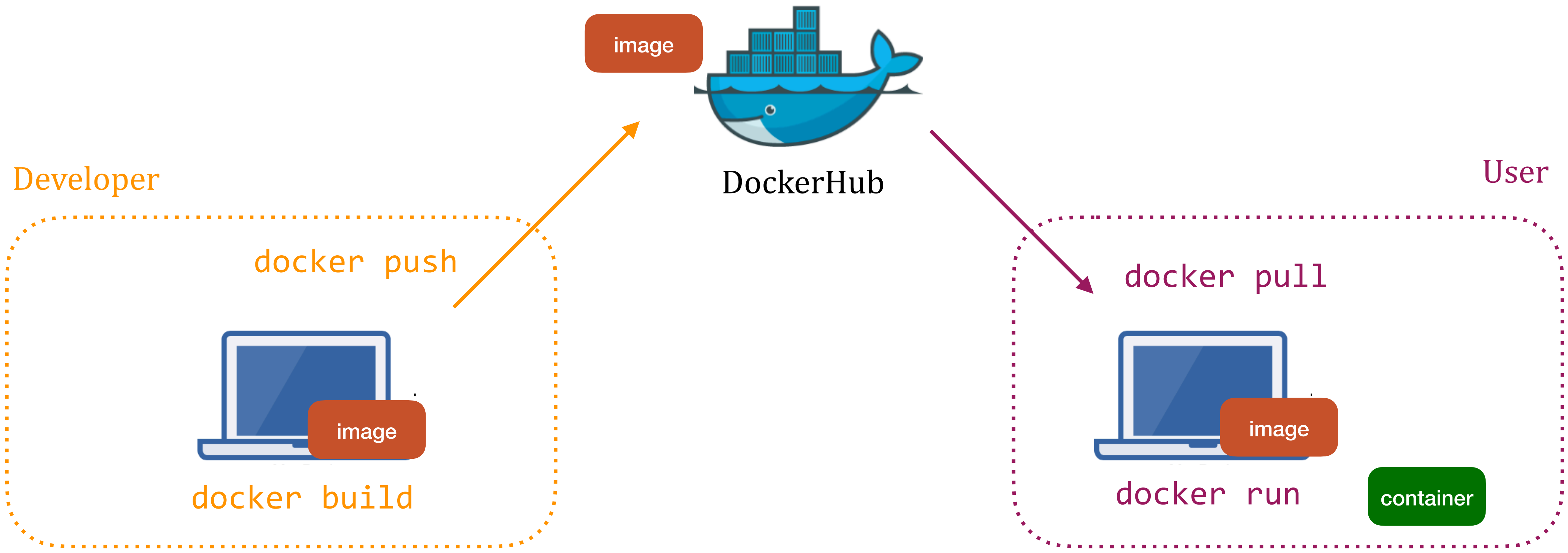
https://hub.docker.com/u/duckietown/dashboard/

Repositories

Repository	Visibility	Stars	Pulls
duckietown/duckietown-swarm-amd64	public	0 STARS	100K+ PULLS
duckietown/rpi-ros-kinetic-roscore	public | automated build	0 STARS	10K+ PULLS
duckietown/rpi-duckiebot-camera-node	public | automated build	0 STARS	10K+ PULLS

Docker workflow overview, with registry

- Operations to **develop containers**:
  - `docker build` - Builds an **image**
  - `docker push` - Uploads the **image** to the registry.
- Operations to **use containers**:
  - `docker pull` - Obtains or updates an **image** from the repository
  - `docker run` - Creates a **container** from an **image** and runs it.



What's nice about Docker

- **Reproducible and documented builds** with Dockerfiles.
- Full control over execution environment:
    - Know exactly what the **dependencies** are (e.g., *dependencies-apt.txt*).
    - Know exactly what **files** your application needs (*build context, docker diff*).
- Full support of **cross-application interaction:**
    - e.g., ROS, LCM
- **No conflict** between libraries.
- Full control over **networks and ports:**
    - Open only the ports and for the protocols you need.
- Full control over **resources** (X-Server, CPU, GPU, RAM).

Building Docker images

Docker Images hash and names

- An **image** is **uniquely identified by an hash:**

```
sha256:3448a24e6db0125ebbafefee0a355232fc533bd3a68c89dab3d450a8fa15d8ed
```

- **On a registry**, it is also (non-uniquely) **identified by a name:**

```
ubuntu/ubuntu:18.04

afdaniele/compose:0.9
```

  - Format of the name: `owner/image:tag`

Docker Image and layers

- An **image** is the combination of a sequence of **layers.**
- A **layer** is a **collection of files** (uniquely identified by an hash).

```
/my_file.dat        (user file)
/etc/hosts          (system file)
```

```
image hash = hash(
  layer 1 hash,
  layer 2 hash,
  layer 3 hash,
  layer 4 hash,
  layer 5 hash
)
```

An example Dockerfile

Dockerfile

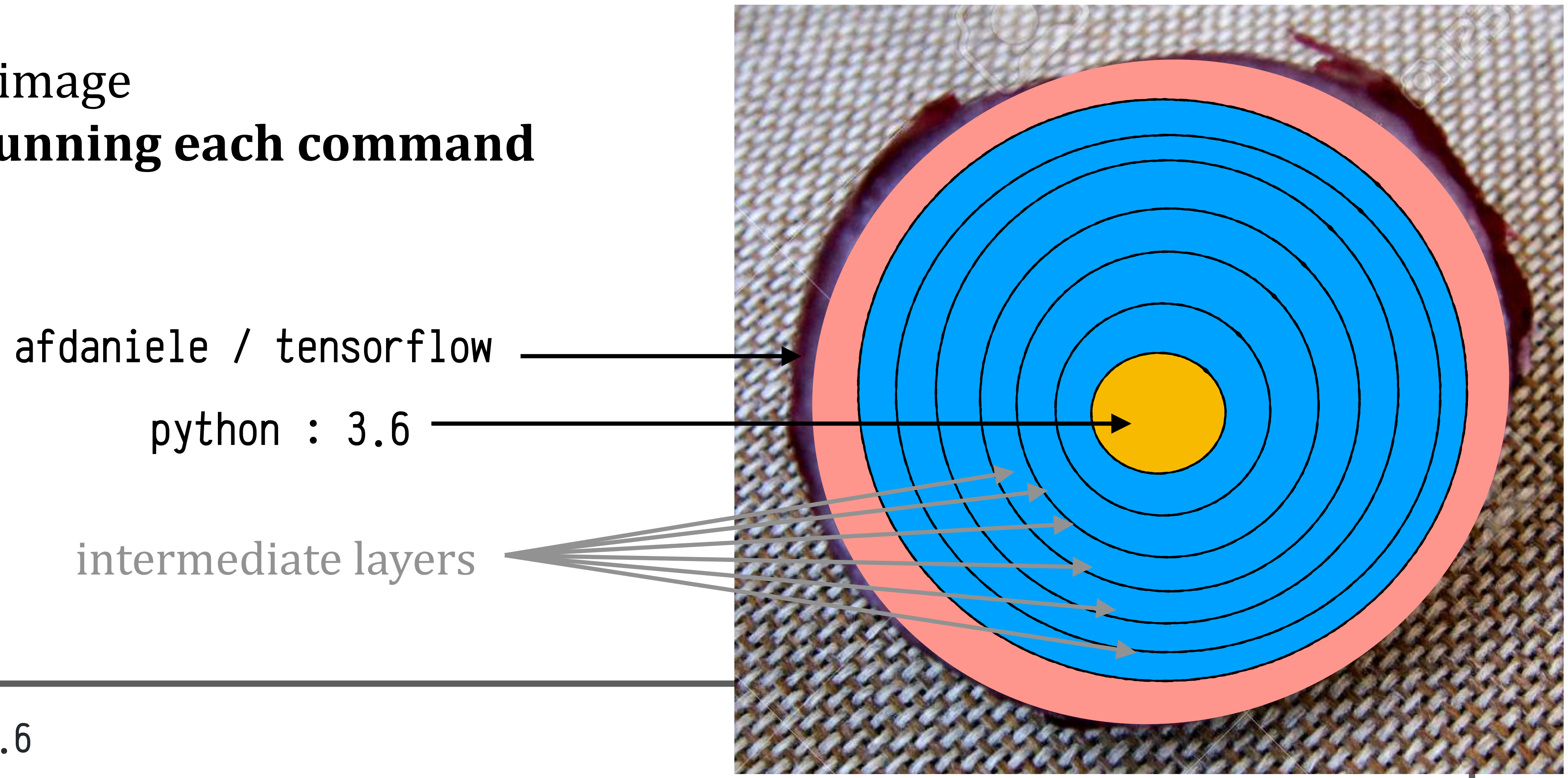
```
FROM python:3.6

MAINTAINER Andrea F. Daniele <afdaniele@ttic.edu>

RUN pip3 install tensorflow

…

EXPOSE 6006/tcp

CMD [”python3”, ”–m”, “tensorflow.tensorboard”, ”––logdir=/tflog”]
```

Common Dockerfile instructions

FROM	Define the **base image**
ARG	Define build-only arguments (non-persistent)
ENV	Define environment variables (persistent)
MAINTAINER	Set maintainer info
WORKDIR	Set working directory
USER	Set user ID
RUN	Run a command inside a container
ADD	Copy files and directories from the **build context**
COPY	Copy files and directories from the build context
VOLUME	Define a new **volume**
EXPOSE	Declare ports used by the image
CMD	Define default command
ENTRYPOINT	Define entrypoint executable

Useful documentation: docs.docker.com/reference

The **layers** of an image
are **created by running each command**
in a Dockerfile.



```
FROM python:3.6

MAINTAINER Andrea F. Daniele <afdaniele@ttic.edu>

RUN pip3 install tensorflow

...



EXPOSE 6006/tcp

CMD [″python3″, ″-m″, “tensorflow.tensorboard″, ″--logdir=/tflog″]
```

Build Context

- The **"build context"** is the directory from which Docker is allowed to copy files
  - In many scripts, it is "." (current directory)

mydir content:

MyDockerfile

```
FROM python:3.6

...

COPY my_image.jpg /data/my_image.jpg

...
```

- Build an **image** from custom Dockerfile and **build context path:**

```
> docker build -t my_image  -f /dir1/MyDockerfile  mydir
```

Image name Dockerfile path Build context

Running Docker Containers

Docker Containers

- A **container** is an instance of a Docker **image.**
- It is **uniquely identified by an alphanumeric string.**

```
94c5c6f50a7204b49c5cdfd662aa203f3af0b2e2eb6b449634738edfae77fbe3
```

- It is also assigned **a name.**
  - You can **choose the name** using the `—name` option:

```
>  docker run  --name my_container  my_image
```

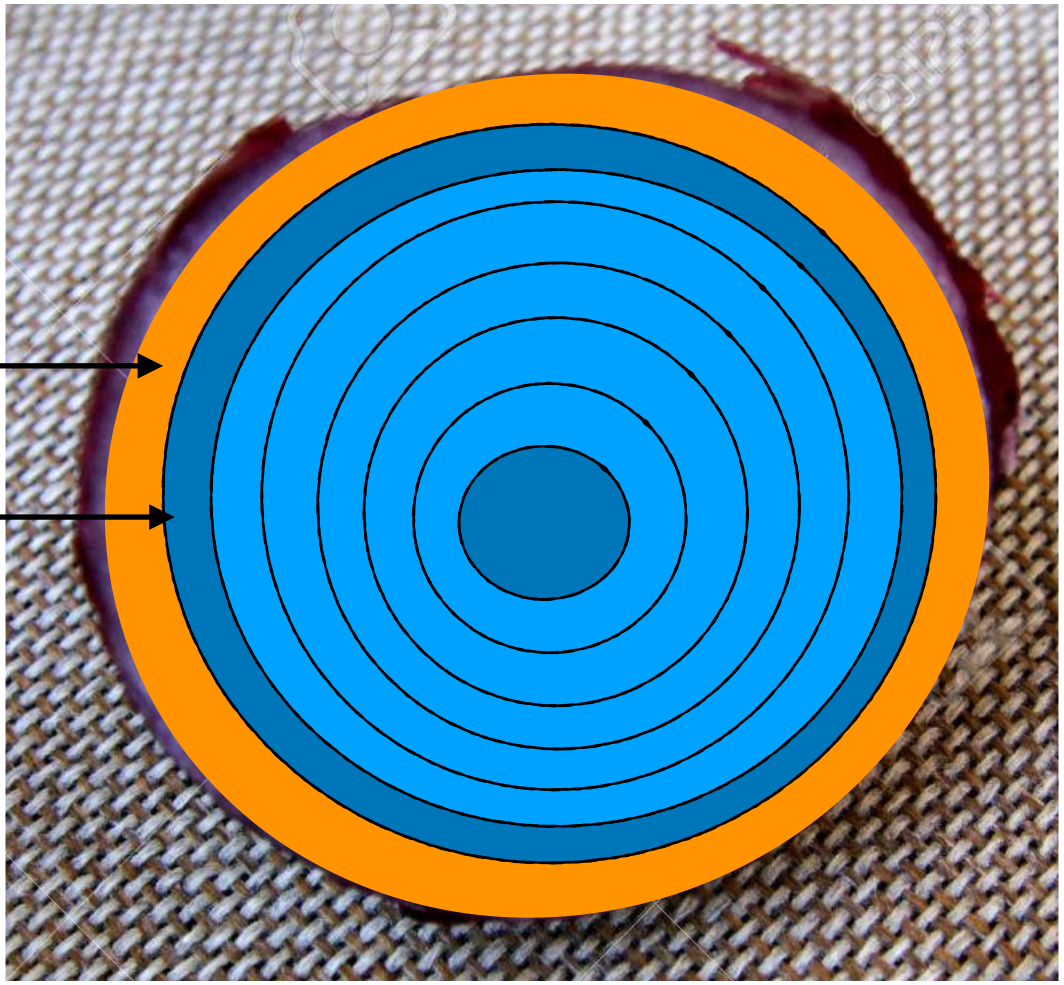
  - Otherwise, it will be **autogenerated** (`admiring_einstein`).

Docker Container execution

- When you run a **container** from an **image:**

```
>  docker run —name mycontainer afdaniele/tensorflow
```

- Docker creates a **writable volatile layer:** programs inside the container can write to their virtual disk.



volatile writeable layer for mycontainer

afdaniele/tensorflow

- This layer is **not persistent;** it is lost when the container is deleted.

```
docker stop mycontainer
docker rm mycontainer
```

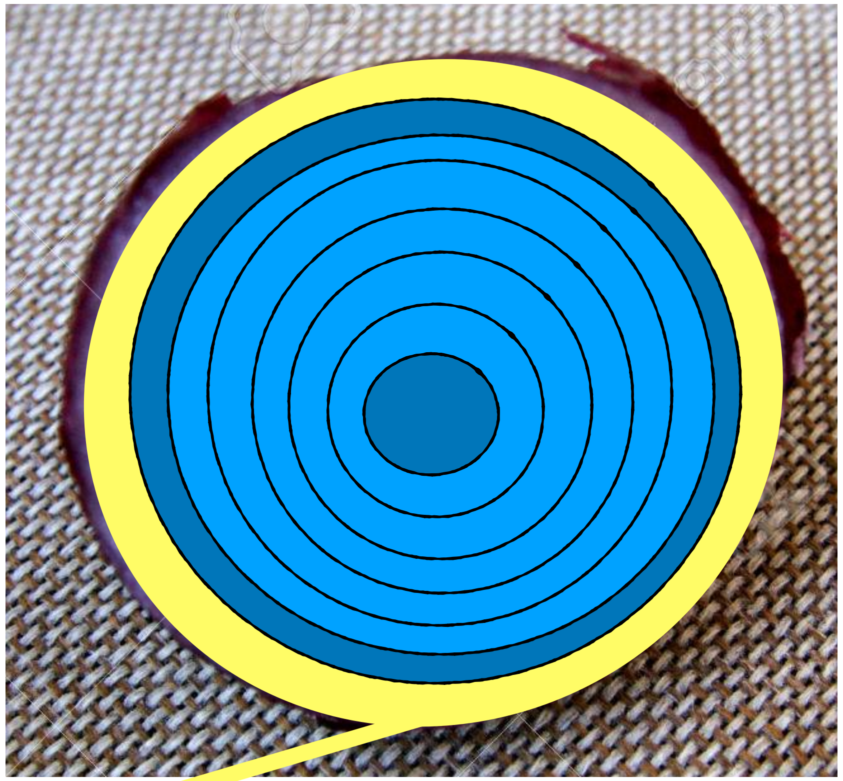
Combining layers - AUFS FileSystem

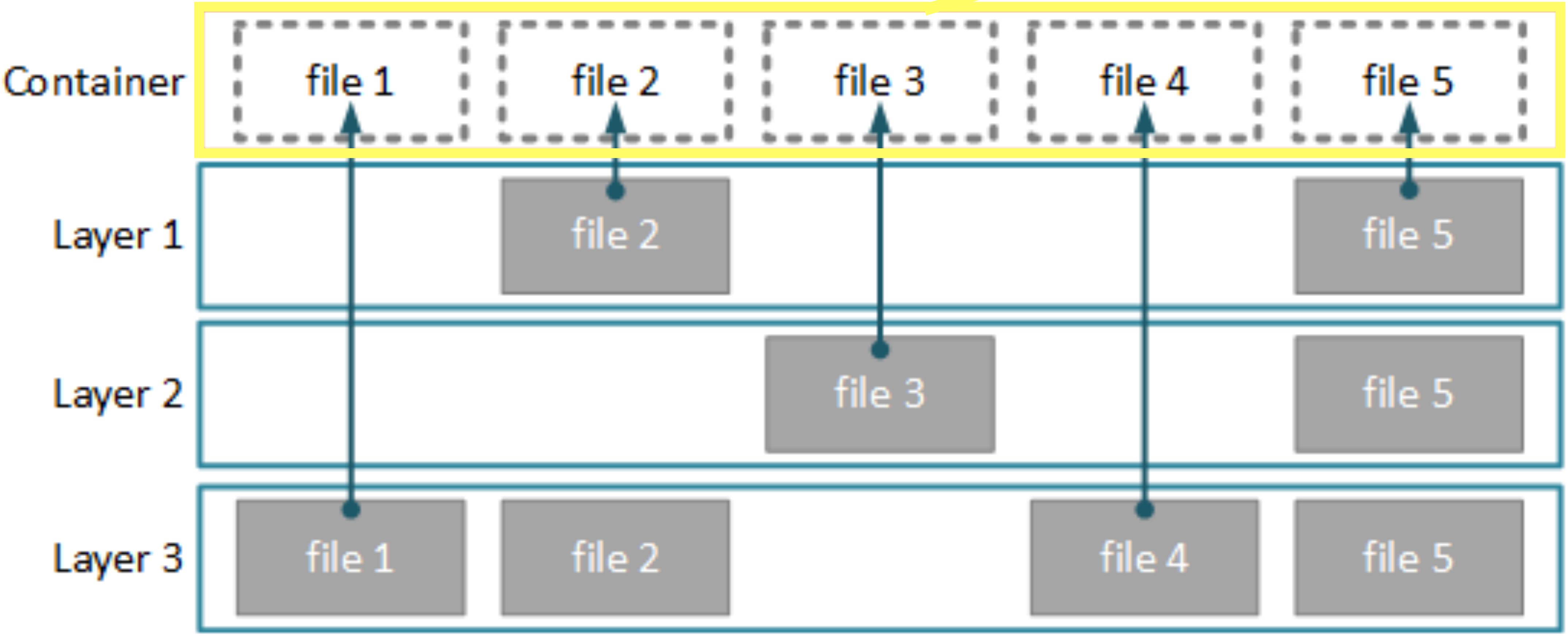
- Originally meaning,

  Another **U**nification **F**ile **S**ystem

- Later revised to,

  Advanced multi-layered **U**nification **F**ile **S**ystem





Container: file 1, file 2, file 3, file 4, file 5

Layer 1: file 2, file 5

Layer 2: file 3, file 5

Layer 3: file 1, file 2, file 4, file 5

Data persistency - Mounting directories

- You can share local directories with one or more containers using

```
> docker run -v [local_dir]:[container_dir] my_image
```

where,

`local_dir` path to a directory in the host file system

`container_dir` destination path to the directory in the container file system

Data persistency - Docker Volumes

- Create a Docker volume

```
>  docker volume create [volume_name]
```

- You can attach a volume to a container using

```
>  docker run -v [volume_name]:[container_dir] my_image
```

where,

`volume_name`	name of the volume
`container_dir`	destination path in the container file system

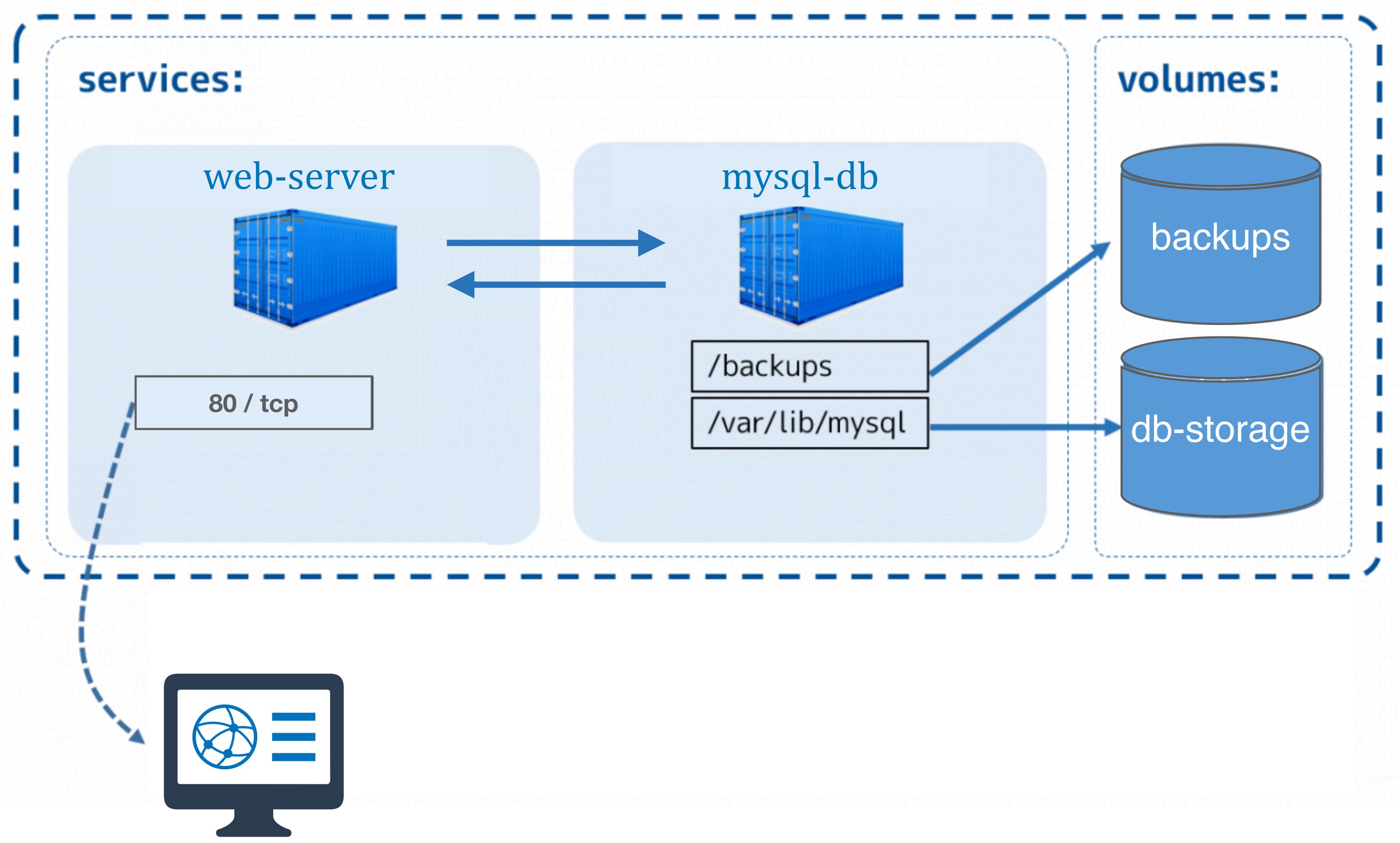
Example of using a volume

- Running the Dashboard on a Duckiebot:

```
>  docker volume create compose-data

>  docker run \
       -it \
       -p 8080:80/tcp \
       -v compose-data:/var/www/html \
       -v /data:/data \
       --hostname $(hostname) \
       --name dashboard \
       duckietown/dt-duckiebot-dashboard:daffy
```

Docker Compose

- An application can be split across multiple Docker images
  - The application runs when all the corresponding containers run



docker-compose.yaml

```
version: '2'
services:
  mysql-db:
    image: mysql:latest
    volumes:
      - db-storage:/var/lib/mysql
      - backups:/backups

  web-server:
    image: apache:latest
    ports:
      - "80:80"
    links:
      - mysql-db:mysql.db
    environment:
      - DBHost=mysql.db
      - DBUser=my_user
      - DBPassword=my_password

volumes:
  db-storage:
  backups:
```