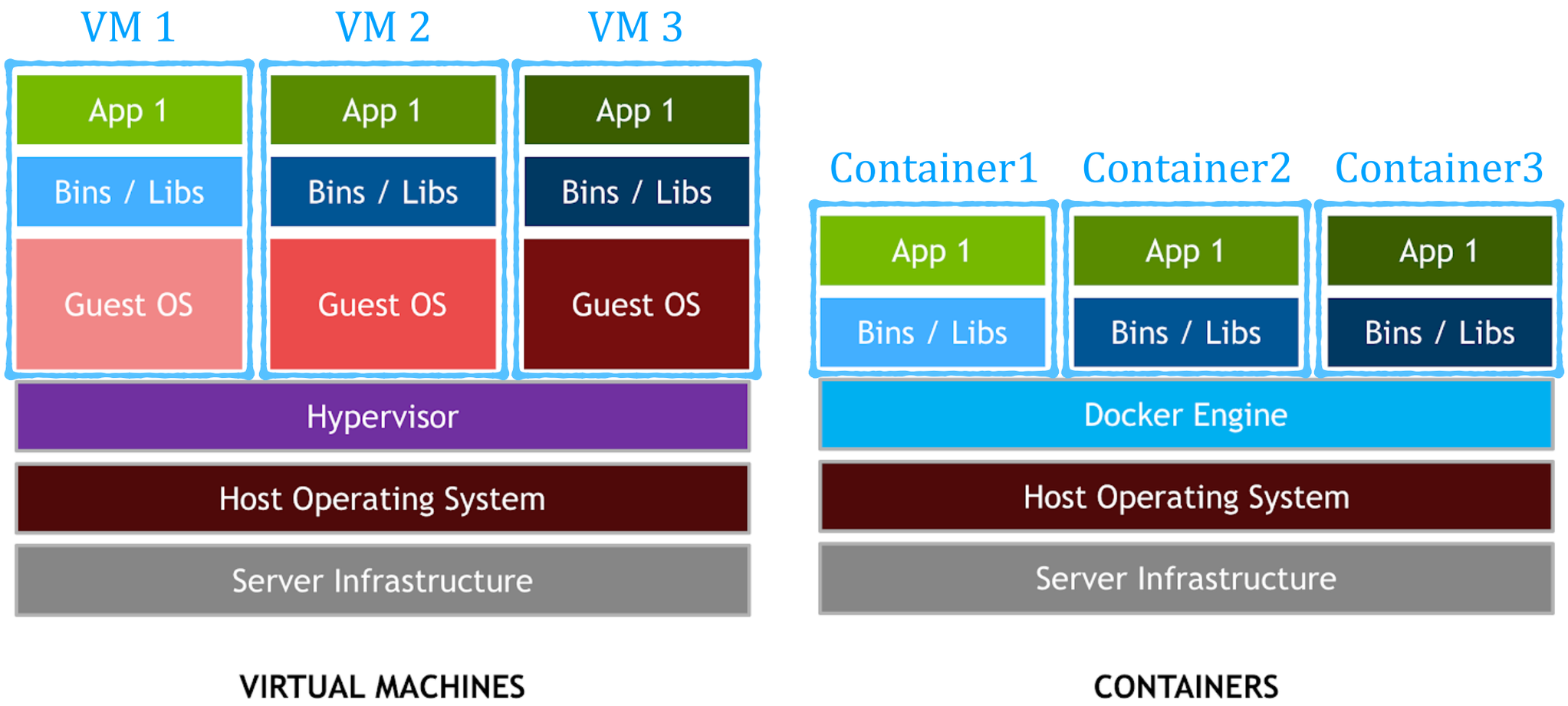

docker

What is Docker?

I got it, it is like VirtualBox…. Hem, not really!



- Docker is similar (in spirit) to Virtual Environments but at a File System level

Why Docker?

Common scenario

- You just started a new research project

    1. Freshly install Ubuntu for a clean environment
    2. Configure your environment
    3. Install all the libraries you need

```
#    ubuntu:18.04

>    NAME=Andrea

>    apt install python-numpy
```

Common scenario (with Docker)

- You just started a new research project
    1. Freshly install Ubuntu for a clean environment
    2. Configure your environment
    3. Install all the libraries you need

Dockerfile

```
FROM   ubuntu:18.04

ENV    NAME=Andrea

RUN    apt install python-numpy
```

Docker = A lot of new confusing terms…

For example:

- Docker Layer
- Docker Image
- Docker Container
- Docker Volume
- Docker Registry
- Docker Compose

Docker
Layer/Image/File

Docker Layer

- It is a **collection of files**
  - e.g.,

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

- It is uniquely identified by an hash

```
sha256:94c5c6f50a7204b49c5cdfd662aa203f3af0b2e2eb6b449634738edfae77fbe3
```

Docker Image

- It is the combination of a sequence of **Docker Layers**
- It is uniquely identified by an hash

```
sha256:3448a24e6db0125ebbafefee0a355232fc533bd3a68c89dab3d450a8fa15d8ed
```

- It is also (non-uniquely) identified by a name

```
library/ubuntu:18.04
afdaniele/compose:0.9
```

format:

```
[owner] / [image] : [tag]

library / [image] : latest      (default)
```

Docker Image VS Docker Layers

- You **CANNOT** create single layers
- You **CAN** create new images
- Layers are read-only
- The simplest (most beautiful) way to create an image is a **Dockerfile**

Dockerfile (an example)

```
FROM python:3.6

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

RUN pip3 install tensorflow

VOLUME /tflog

EXPOSE 6006/tcp

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

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

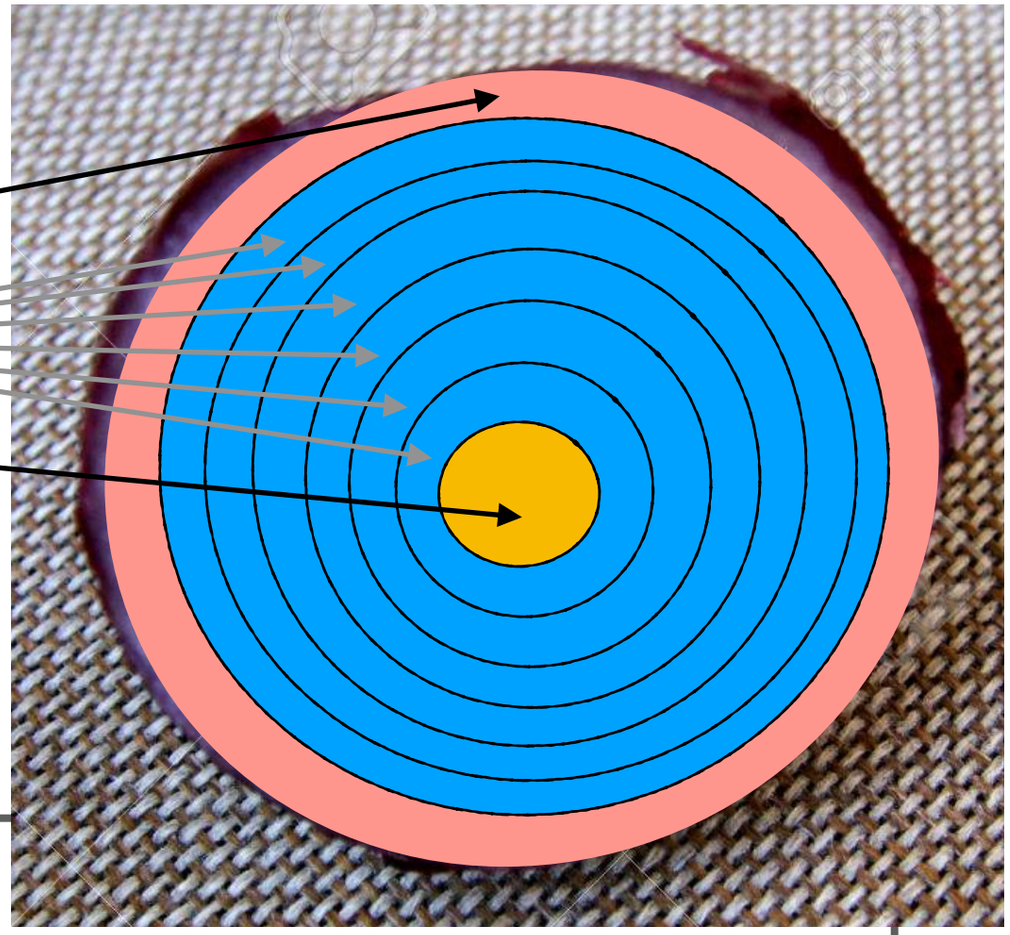
Useful documentation: docs.docker.com/reference

Docker Image

afdaniele / tensorflow

intermediate layers

python : 3.6



Dockerfile

```
FROM python:3.6

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

RUN pip3 install tensorflow

VOLUME /tflog

...

EXPOSE 6006/tcp

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

Building an Image

Build Context

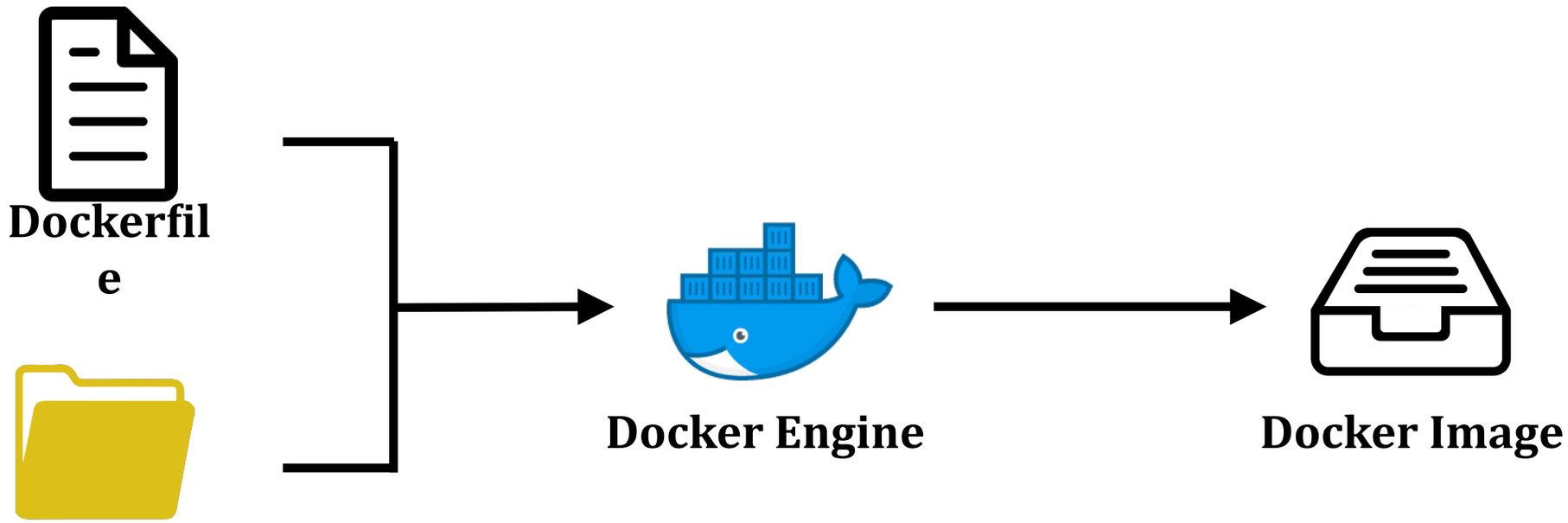
- A directory from which Docker is allowed to copy files to a layer

Directory

Dockerfile

```
FROM python:3.6

...

COPY my_image.jpg /data/my_image.jpg

...
```

Building a Docker Image (using Dockerfile)



- Build an image from a **Dockerfile** and a **Build Context**

```
>  docker build -t my_image [-f Dockerfile] ./
```

Image name Dockerfile path Build context

Docker Container

Docker Container

- An instance of a Docker image
- It is uniquely identified by an alphanumeric string. e.g.,

```
94c5c6f50a7204b49c5cdfd662aa203f3af0b2e2eb6b449634738edfae77fbe3
```

- It is also assigned a (possibly custom) name. e.g.,

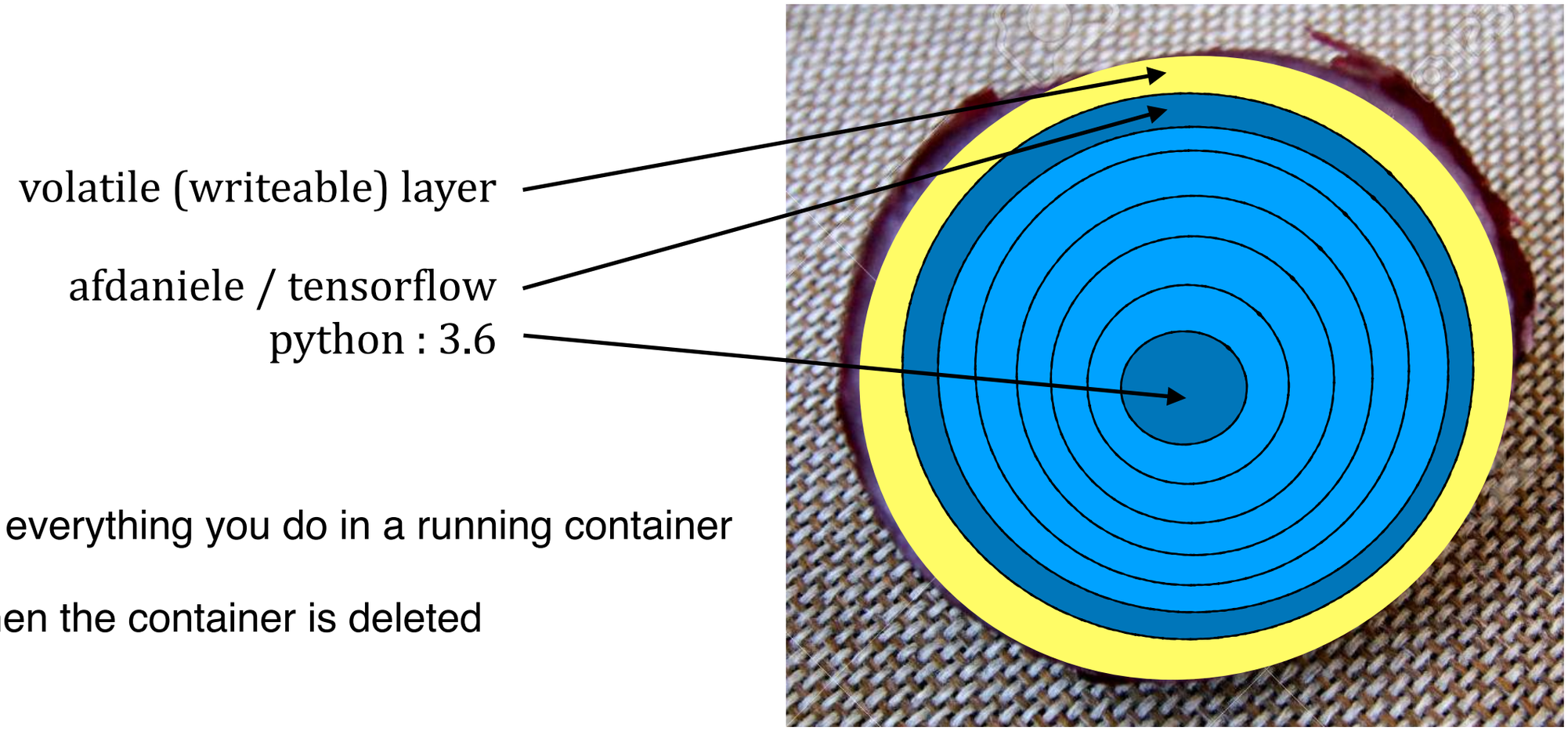
```
admiring_einstein
my_website
```

- Run a **container** from an **image**

```
> docker run [options] my_image
```

Image name

Docker Container



- Contains everything you do in a running container
- Is lost when the container is deleted

- Run a **container** from an **image**

```
> docker run [options] my_image
```

Image name

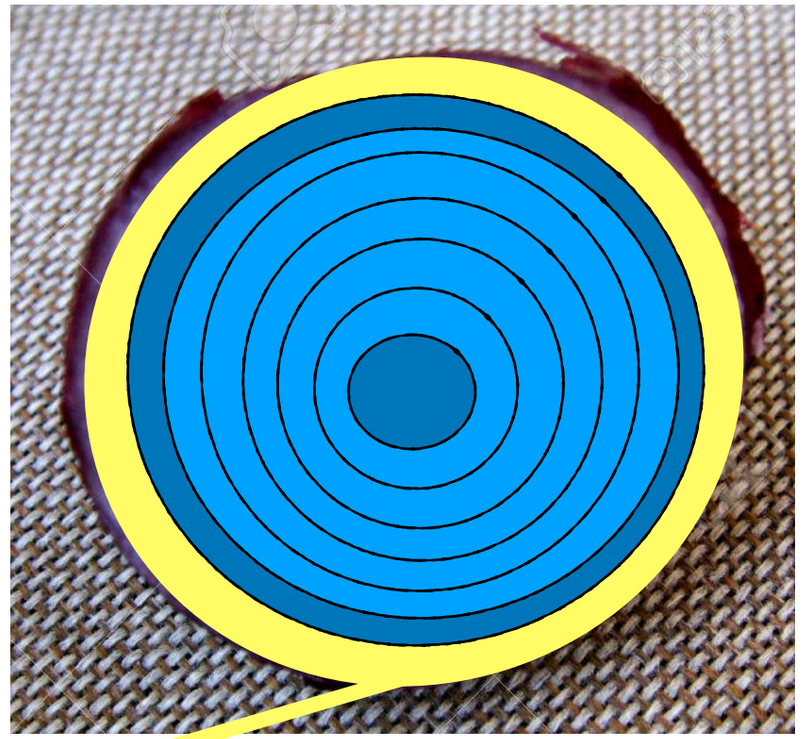
Combining layers - AUFS FileSystem

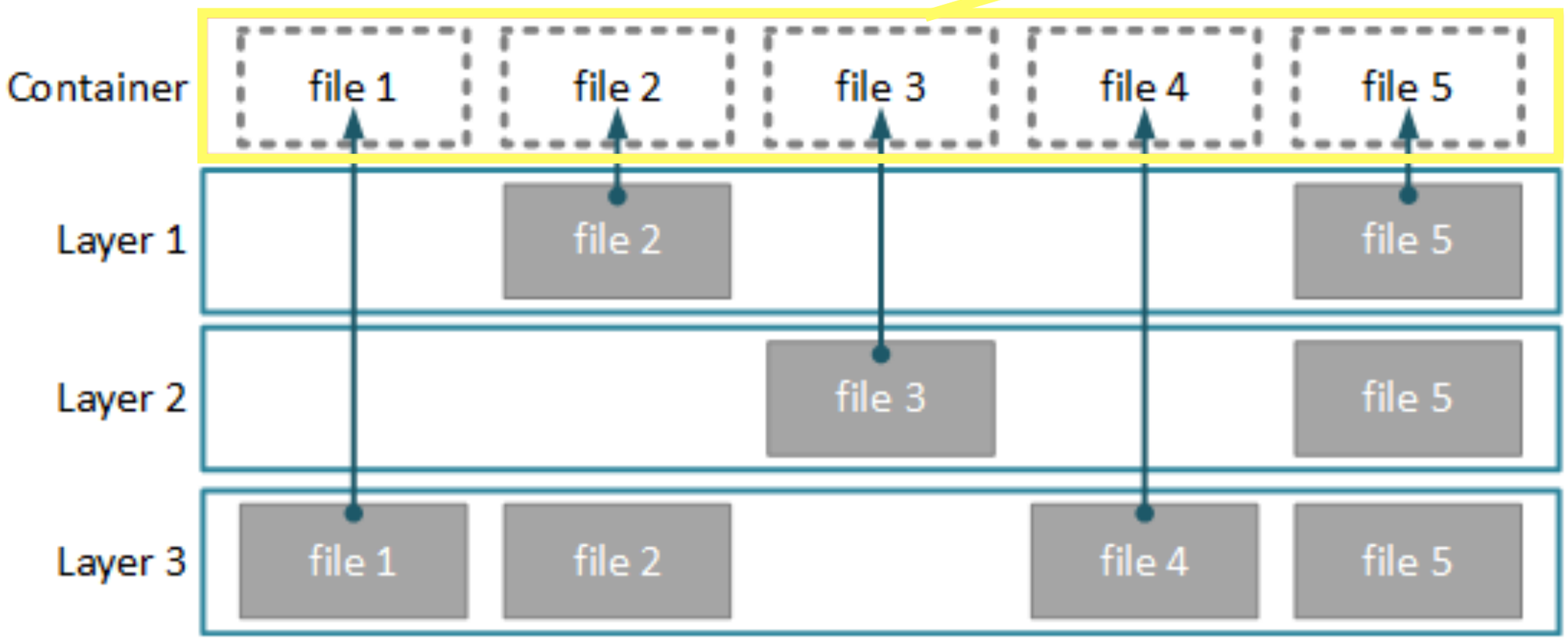
- Originally meaning,

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

- Later revised to,

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





Data Persistency

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

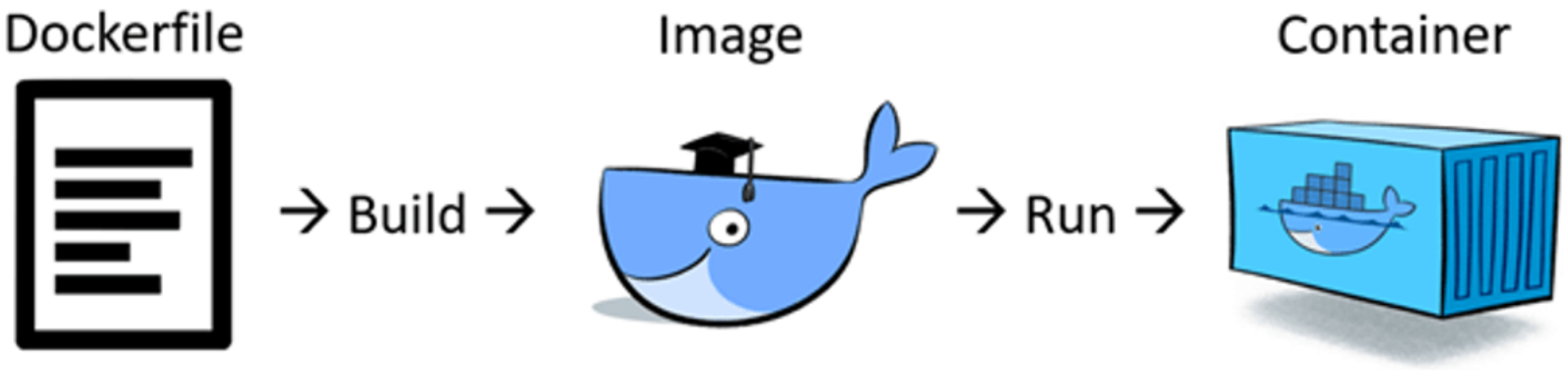
`container_dir` destination path in the container file system

Docker Container (an interesting one)

- 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
```

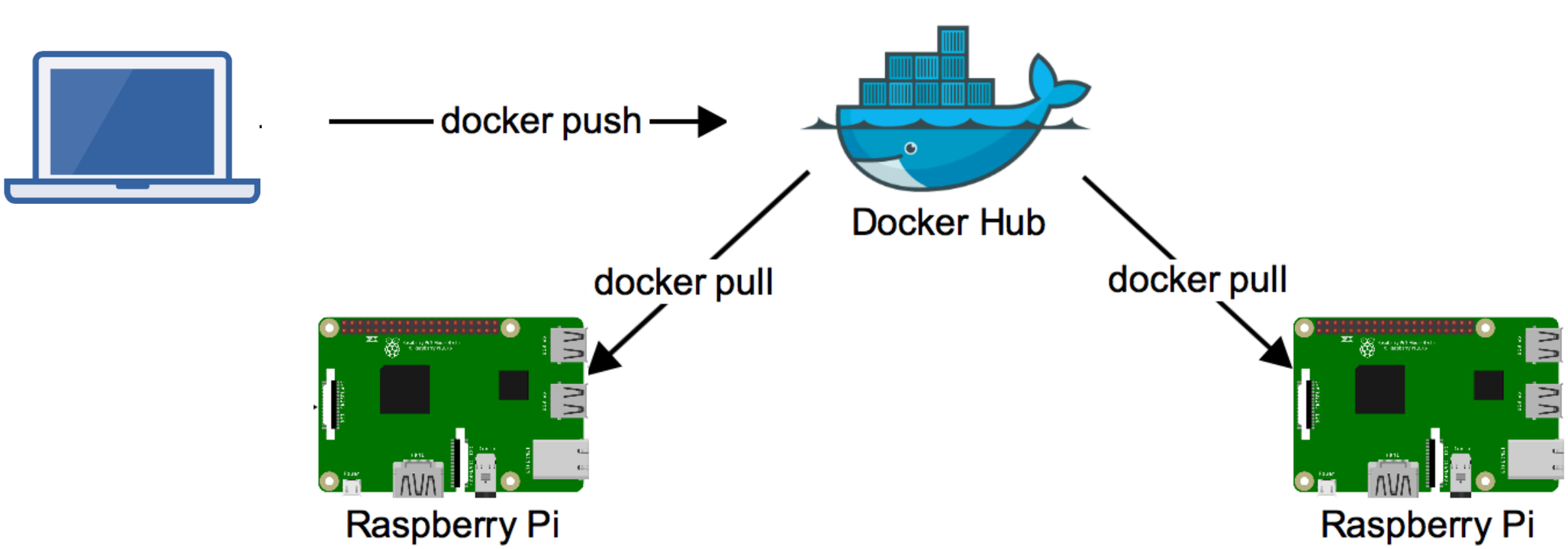
Dockerfile, Docker Image and Docker Container



Docker Registry

Docker Registry

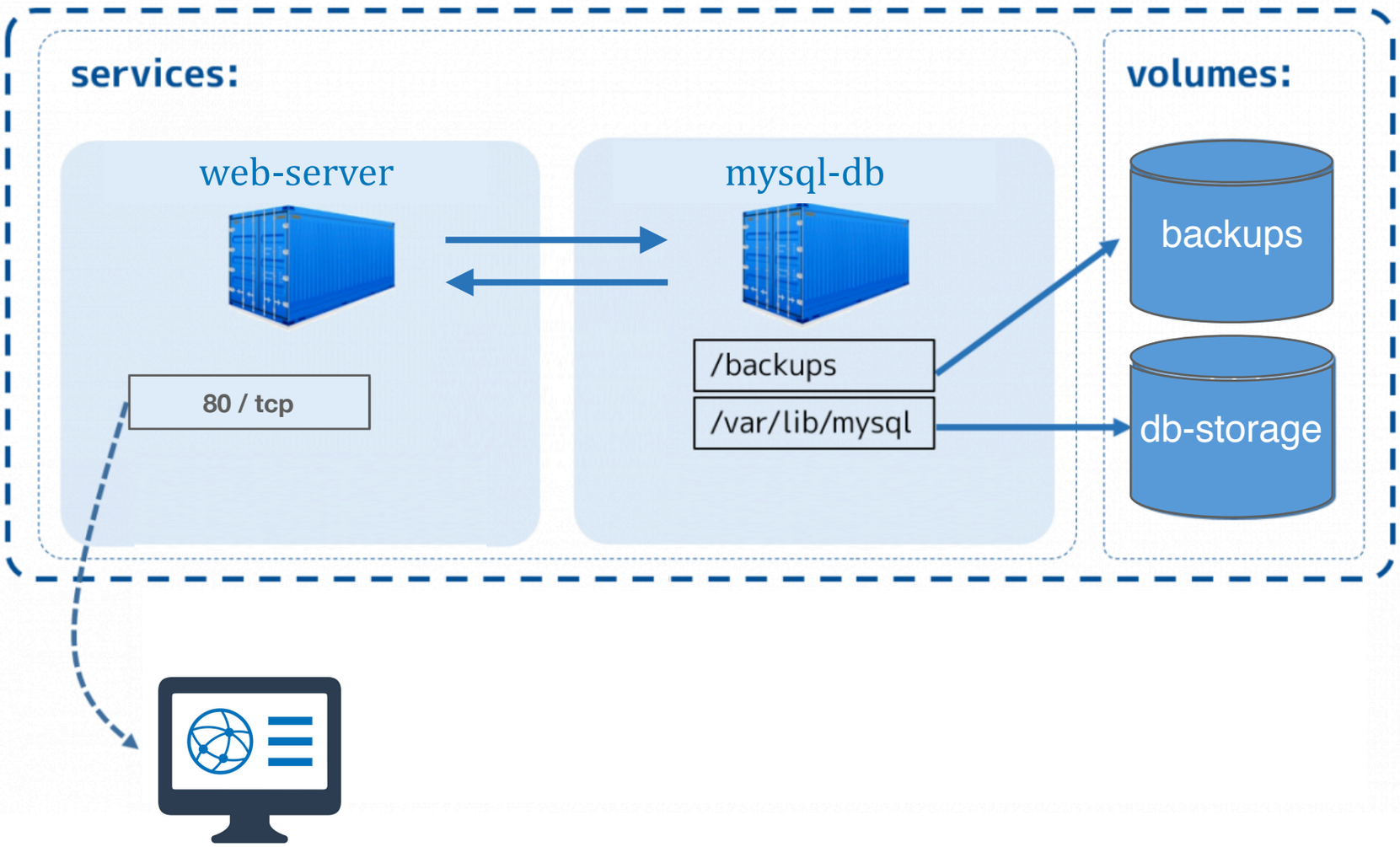
- A **storage** and **distribution** system for Docker images
- Two fundamental operations: **push** and **pull**



Docker Compose

Docker Compose

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

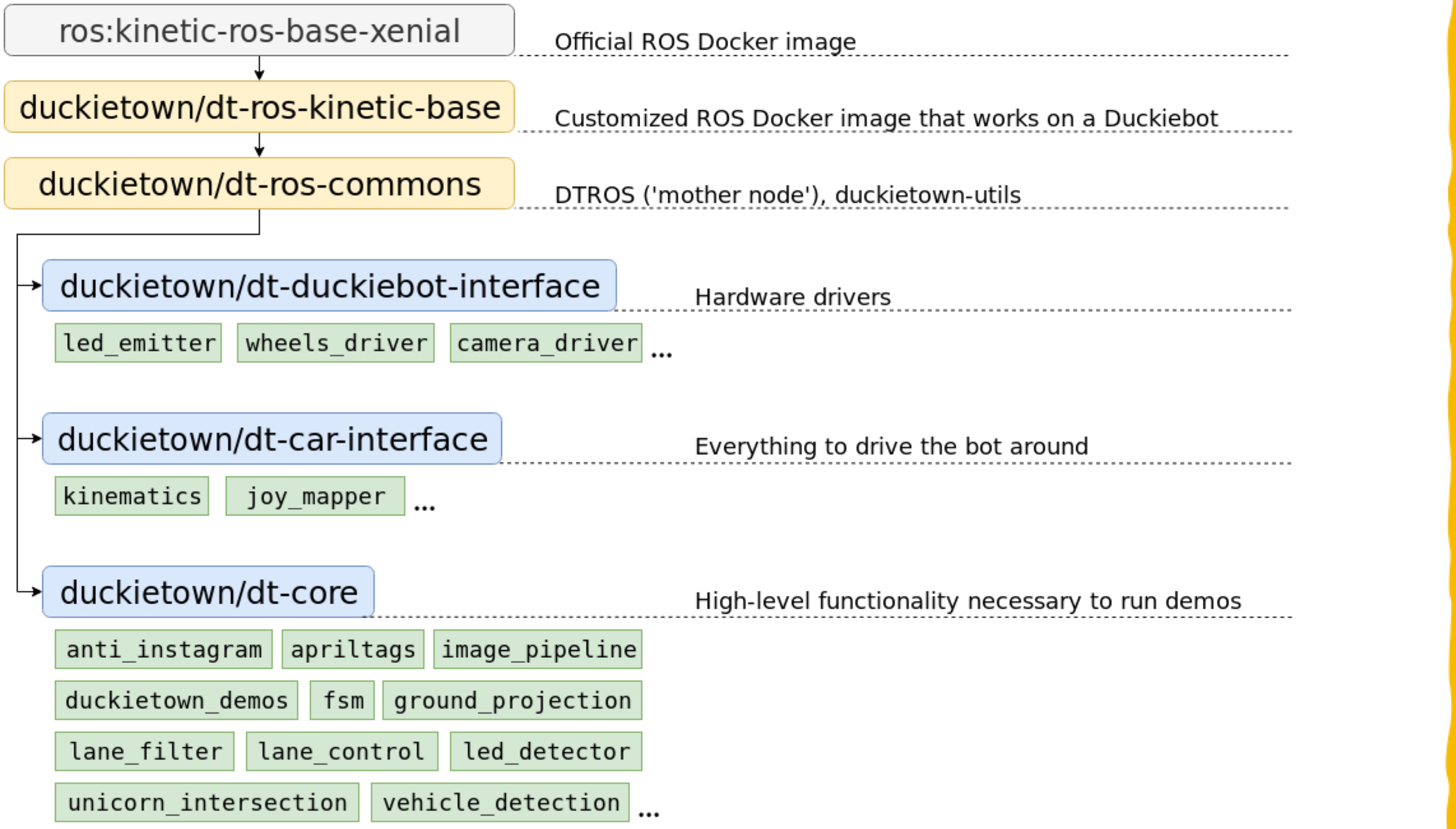


docker-compose.yaml

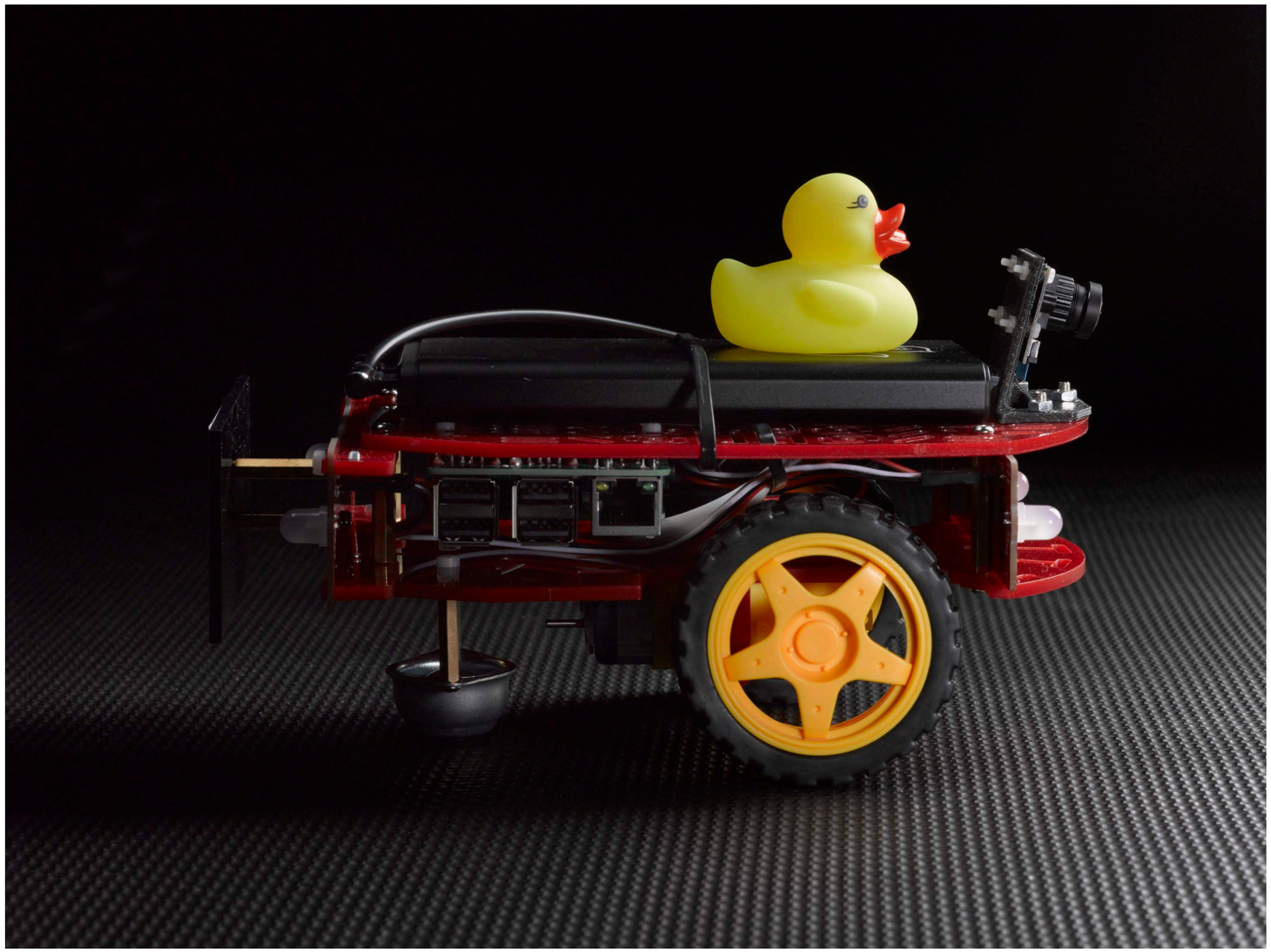
```yaml
version: '2'
services:
  mysql-db:
    image: mysql:latest
    volumes:
      - db-storage:/var/lib/mysql
      - backups:/backups

  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:
```

Docker in Duckietown

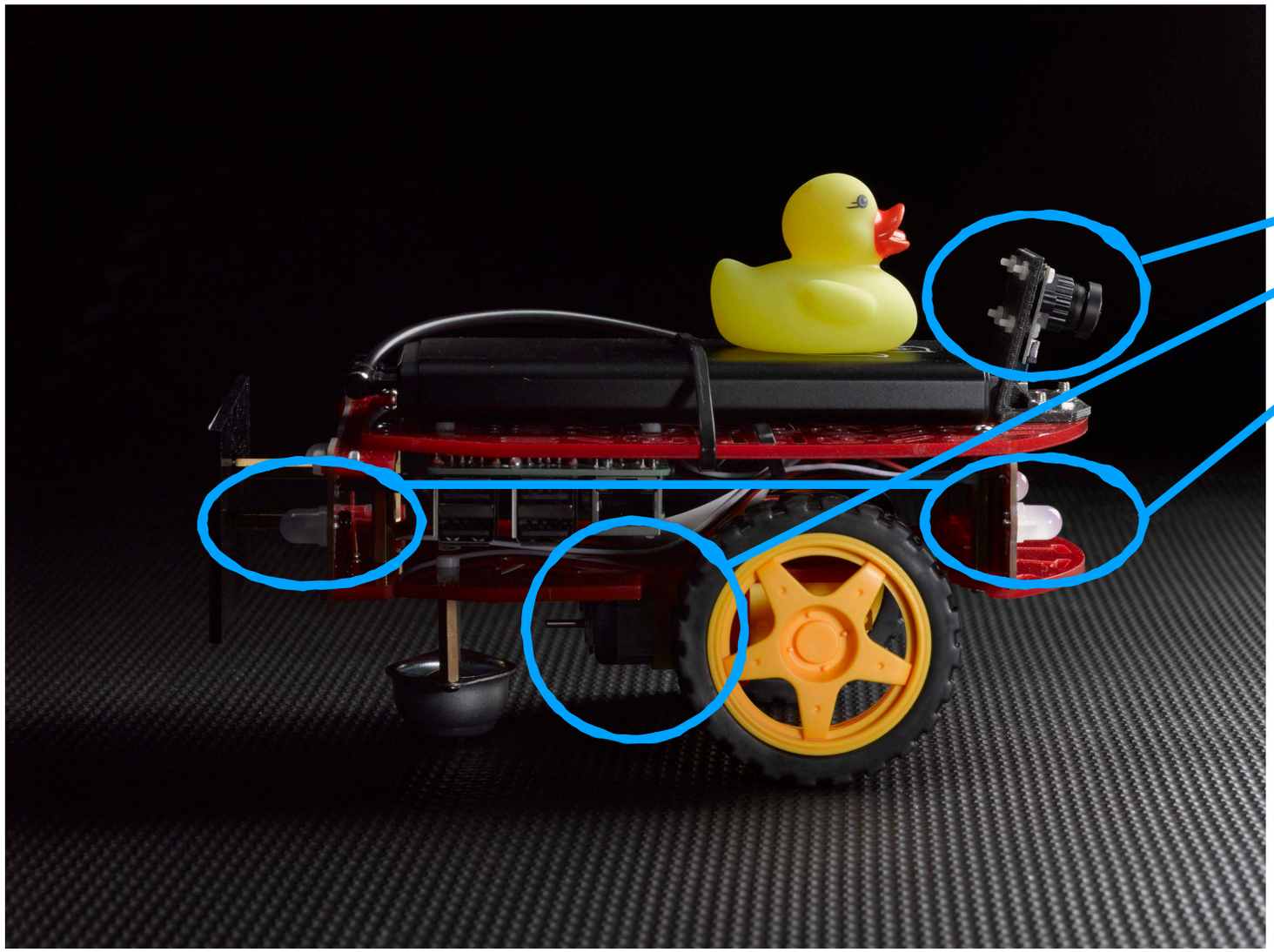
The Docker images family tree



Duckiebot Pipeline



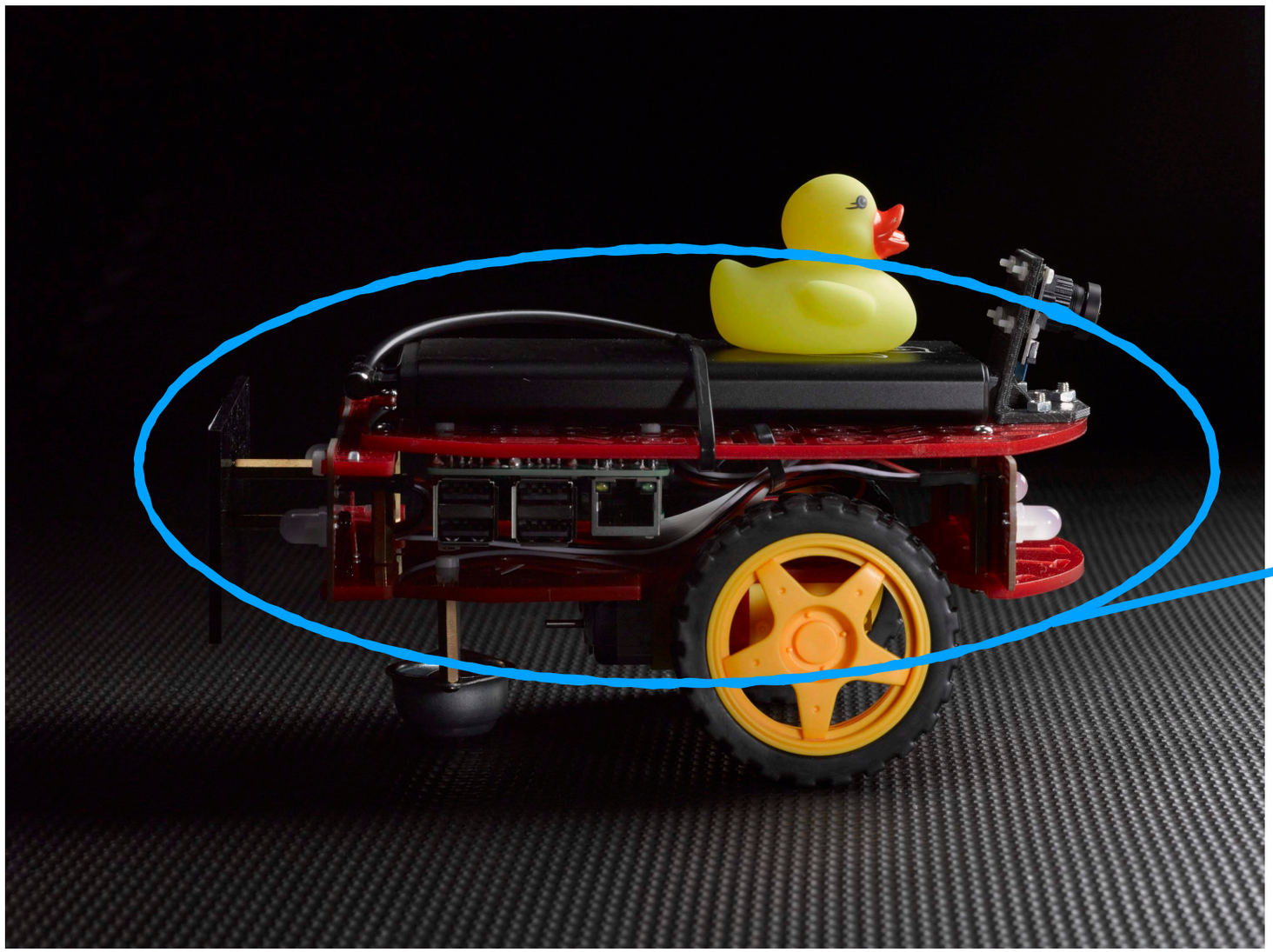
Duckiebot Pipeline



dt-duckiebot-interface

- left/right wheel speed
- raw camera image
- leds color and pattern

Duckiebot Pipeline



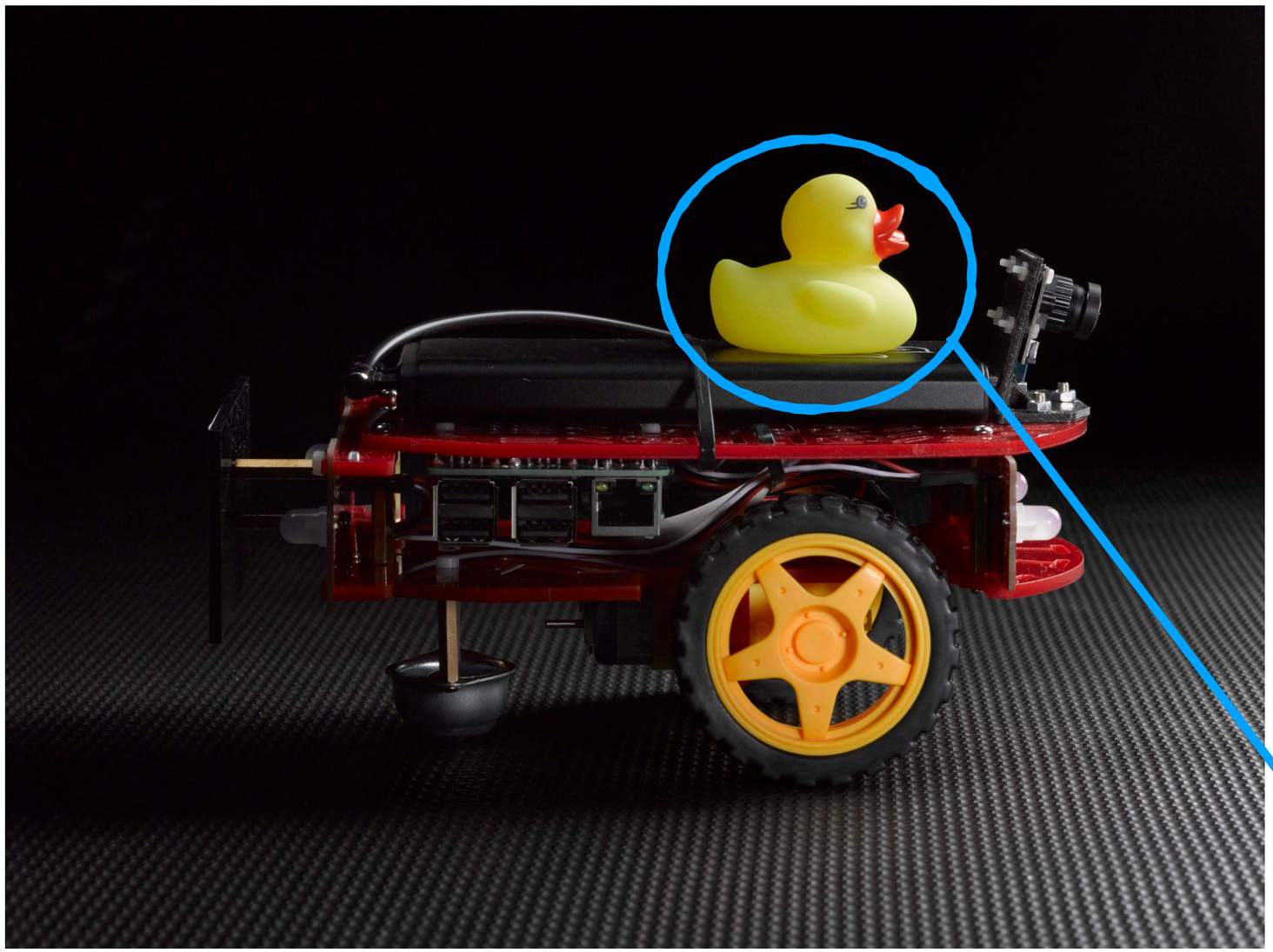
dt-duckiebot-interface

- left/right wheel speed
- raw camera image
- led color and pattern

dt-car-interface

- linear/angular velocity

Duckiebot Pipeline



dt-duckiebot-interface

- left/right wheel speed
- raw camera image
- led color and pattern

dt-car-interface

- linear/angular velocity

dt-core

- lane following
- coordination
- ...

What's nice about Docker

- Dockerfiles serve as **(required) documentation**
- 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)