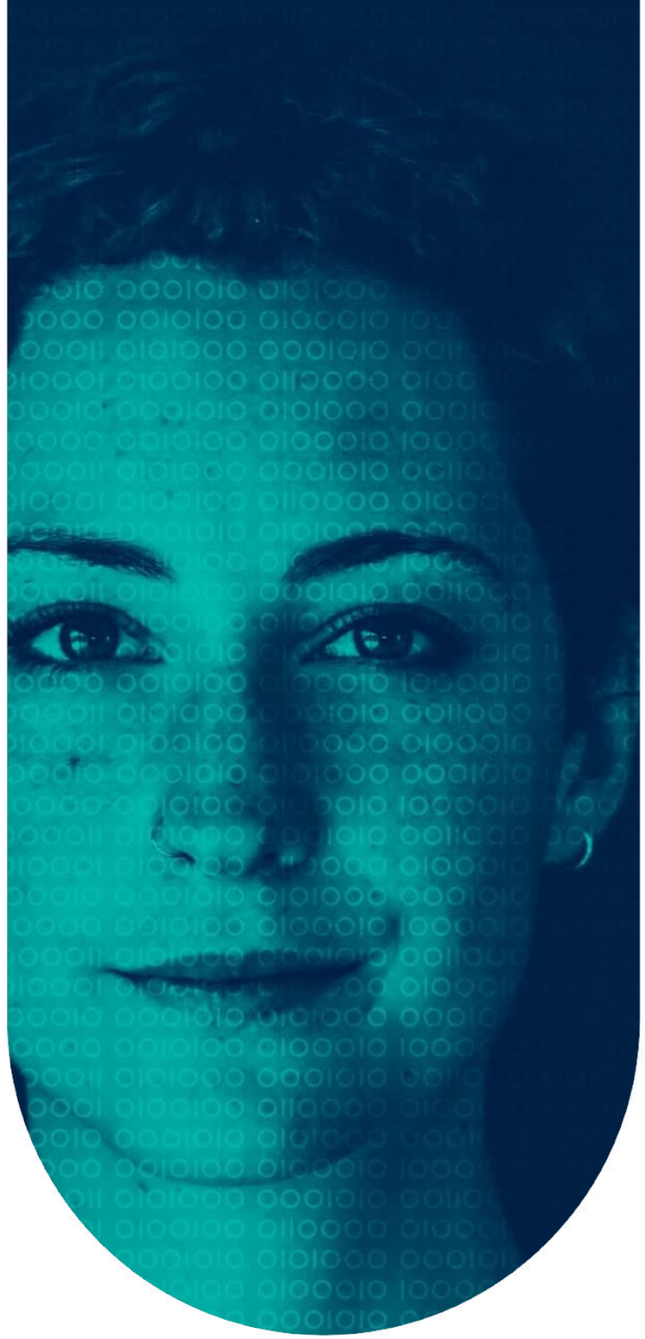

bigdata
republic

Docker for Data Scientists

Packaging your models & best practices

bigdata republic



After this webinar, you will know...

Why you need Docker

How Docker works

How to use Docker in your workflow

bigdata republic

bigdata republic

Data Scientists

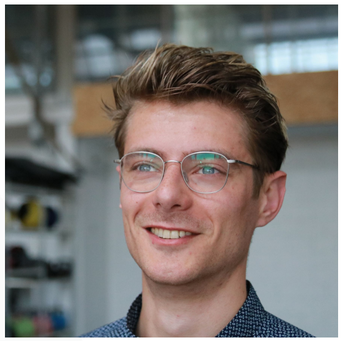
Data Engineers

Presenters and contributors

Tom de Ruijter

Data Scientist

/in/tomderuijter/



Joris Bukala

Data Scientist

/in/joris-bukala/

Jacobus Herman

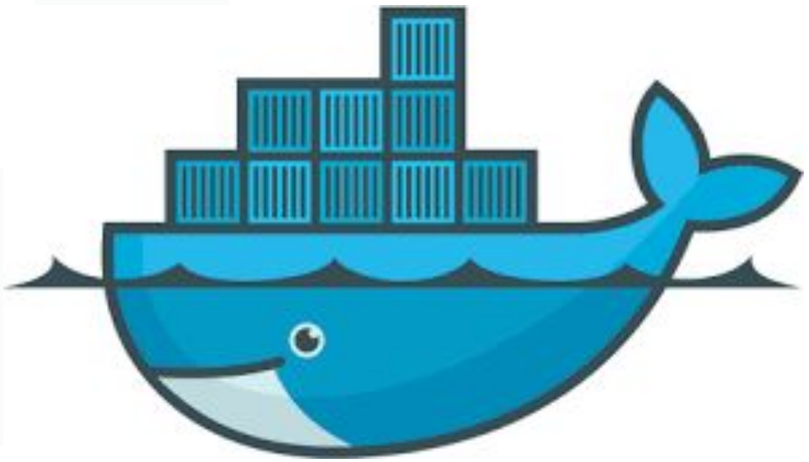
Data Engineer

/in/jacobus-herman/

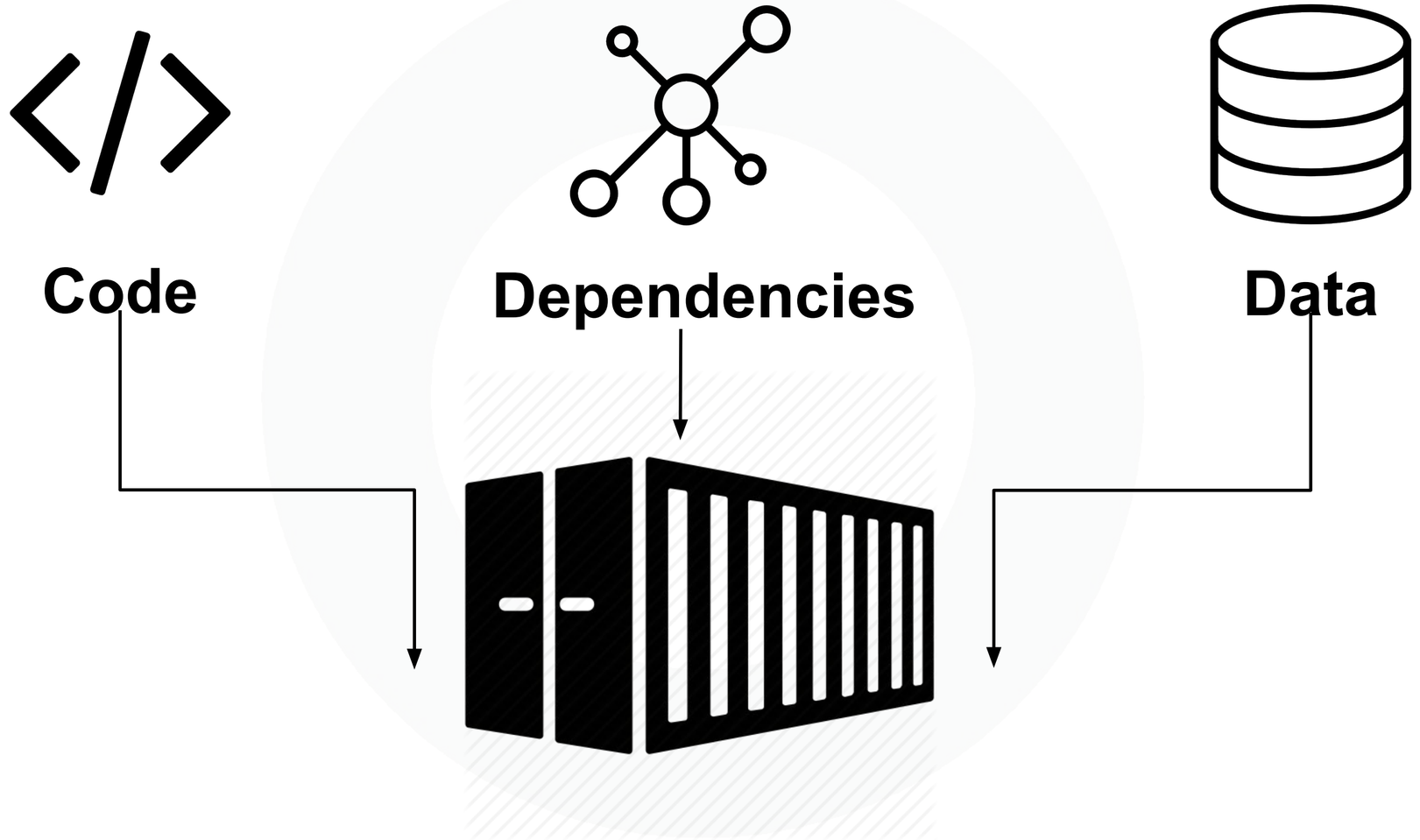
bigdata republic

What does Docker do?

Docker allows us to package and run applications in an isolated environment



Docker Containers



Problems Docker solves

Reproducing models/environments:

Did I need this library in this branch of my code?

Reproducing models/environments:

Did I need this library in this branch of my code?

What environment variables did I have to set when I started this project?

Reproducing models/environments:

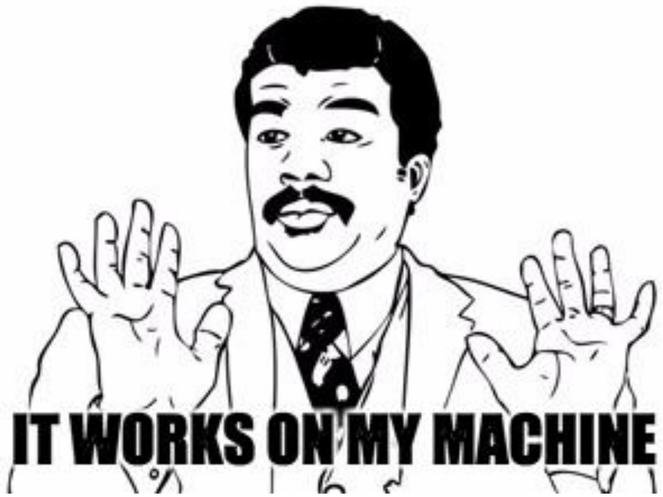
Did I need this library in this branch of my code?

What environment variables did I have to set when I started this project?

Did I use that confidential data when training this model?

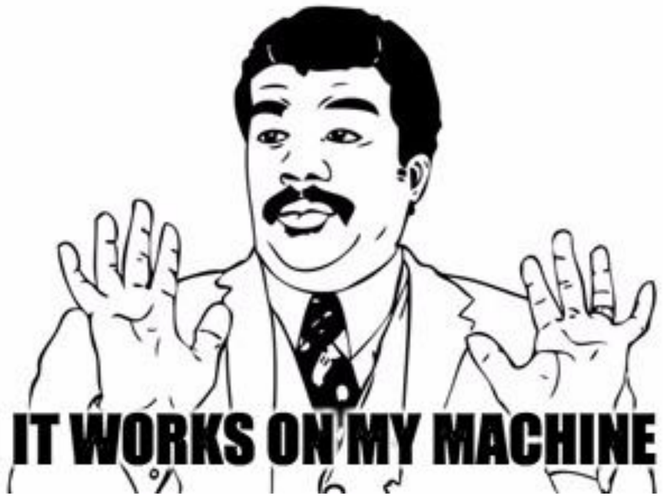
Sharing your work

- *Import error: no module named XXXX*



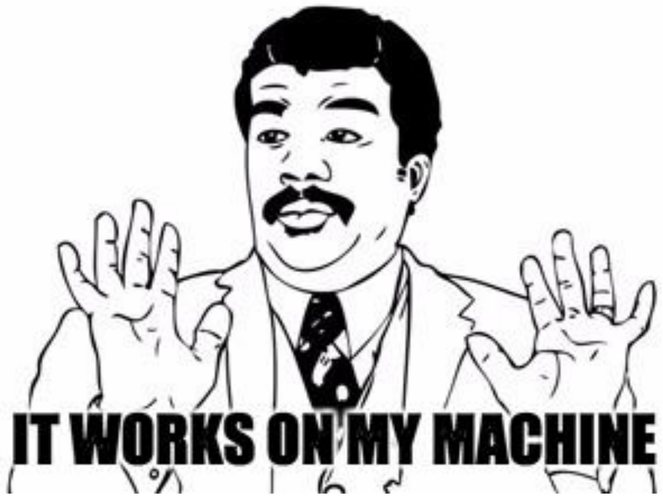
Sharing your work

- *Import error: no module named XXXX*
- *“Oh no all my devices only run Arch Linux...”*



Sharing your work

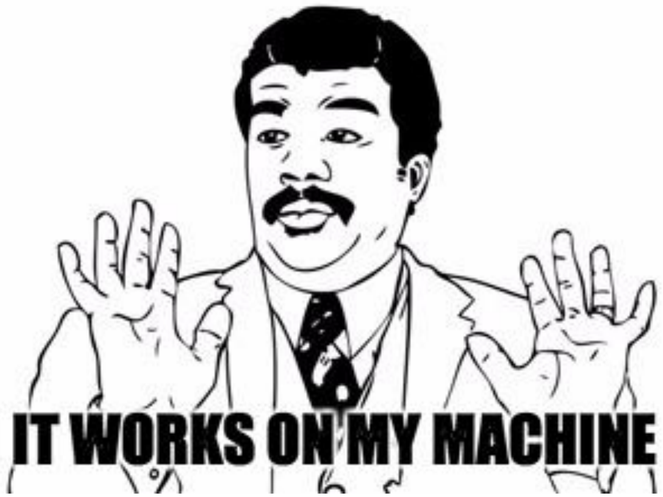
- *Import error: no module named XXXX*
- *“Oh no all my devices only run Arch Linux...”*
- *“Here is the Git repo, now you have to install X, Y, and Z. Oh right I forgot about W. Why don’t you have an up to date COBOL compiler?!”*



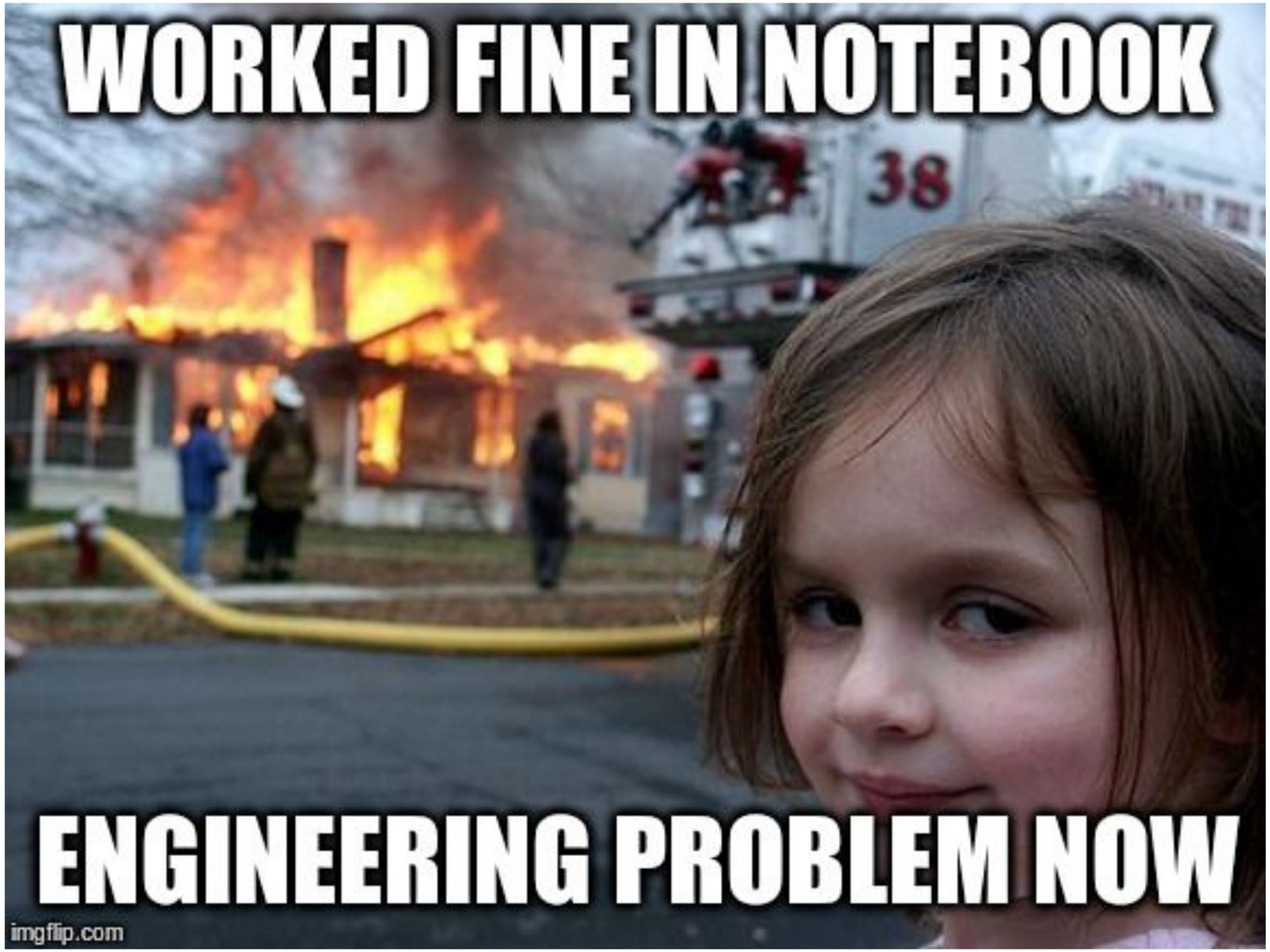
Sharing your work

- *Import error: no module named XXXX*
- *“Oh no all my devices only run Arch Linux...”*
- *“Here is the Git repo, now you have to install X, Y, and Z. Oh right I forgot about W. Why don’t you have an up to date COBOL compiler?!”*

→ Share code repo with a small text file (Dockerfile) and be done

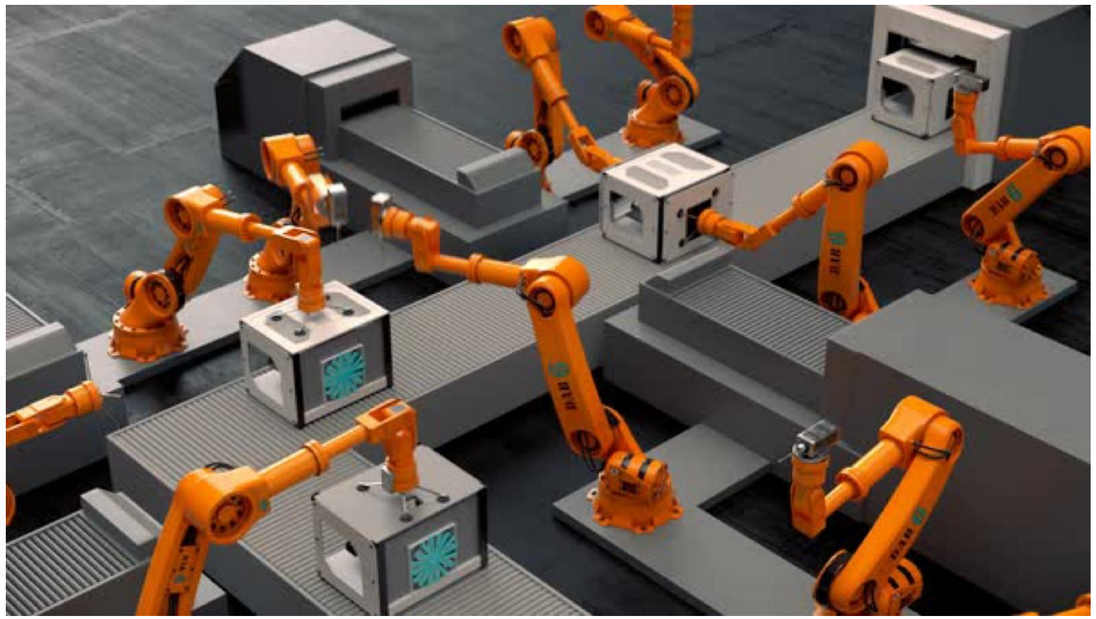


Deployment


WORKED FINE IN NOTEBOOK
38
ENGINEERING PROBLEM NOW
imgflip.com

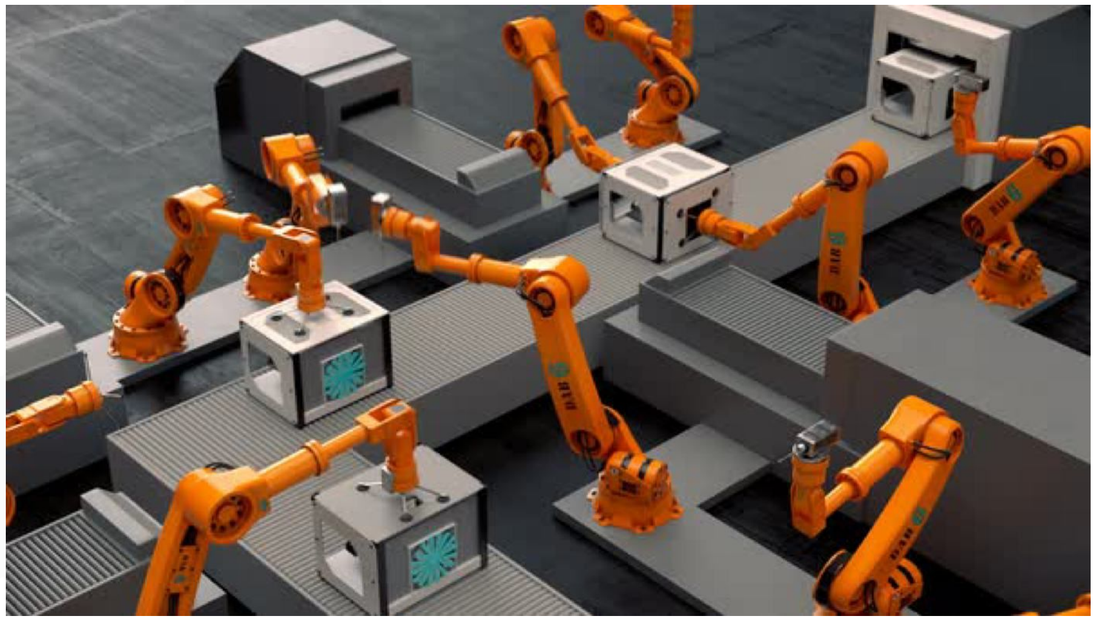
Deployment

- Independent of underlying hardware



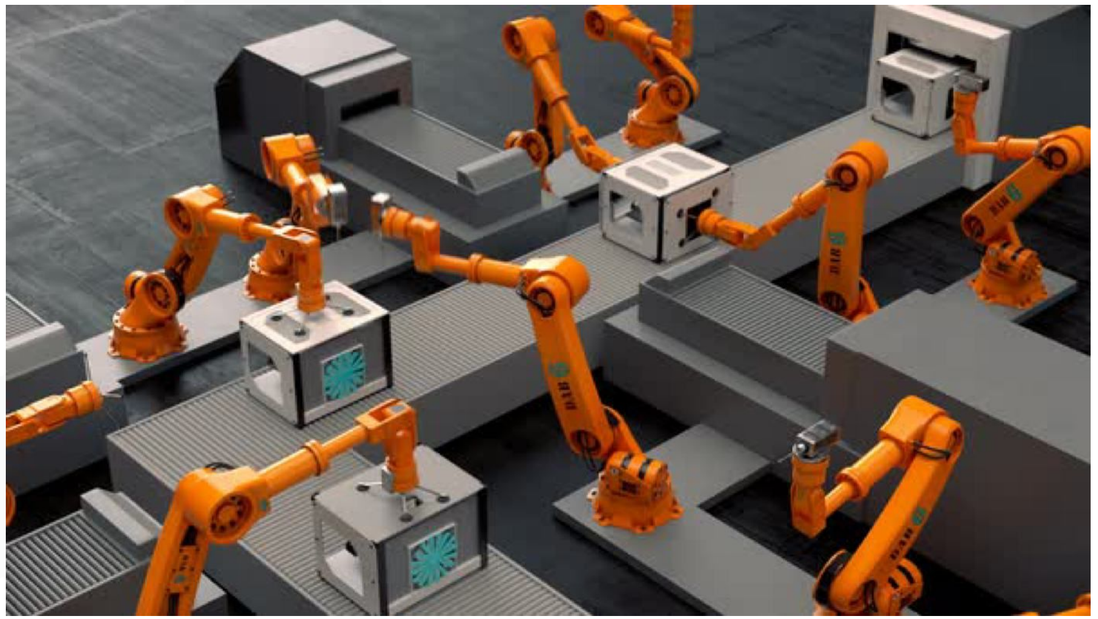
Deployment

- Independent of underlying hardware
- Easy to put anywhere, in any cloud environment



Deployment

- Independent of underlying hardware
- Easy to put anywhere, in any cloud environment
- Can be used in a cluster



How does Docker work

Definitions

Container

Image

Dockerfile

Volume

Definitions

Container: “Lightweight, standalone, executable software package;
includes everything: code, runtime, system tools, system libraries and settings.”

Image

Dockerfile

Volume

Definitions

Container: “Lightweight, standalone, executable software package;
includes everything: code, runtime, system tools, system libraries and settings.”

Image: A frozen snapshot of a container.

Dockerfile

Volume

Definitions

Container: “Lightweight, standalone, executable software package;
includes everything: code, runtime, system tools, system libraries and settings.”

Image: A frozen snapshot of a container.

Dockerfile: An (often) short text-file detailing how to make an image from another image.

Volume

Definitions

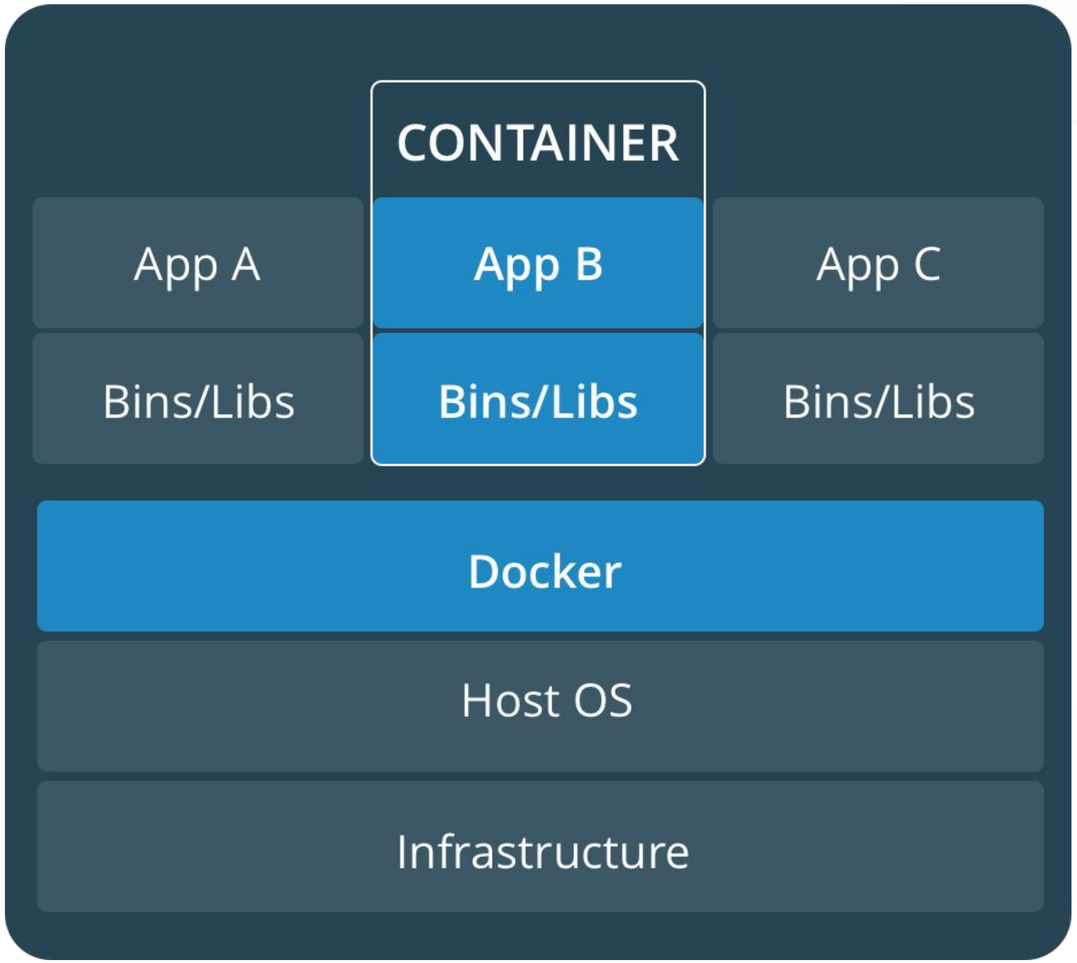
Container: “Lightweight, standalone, executable software package;
includes everything: code, runtime, system tools, system libraries and settings.”

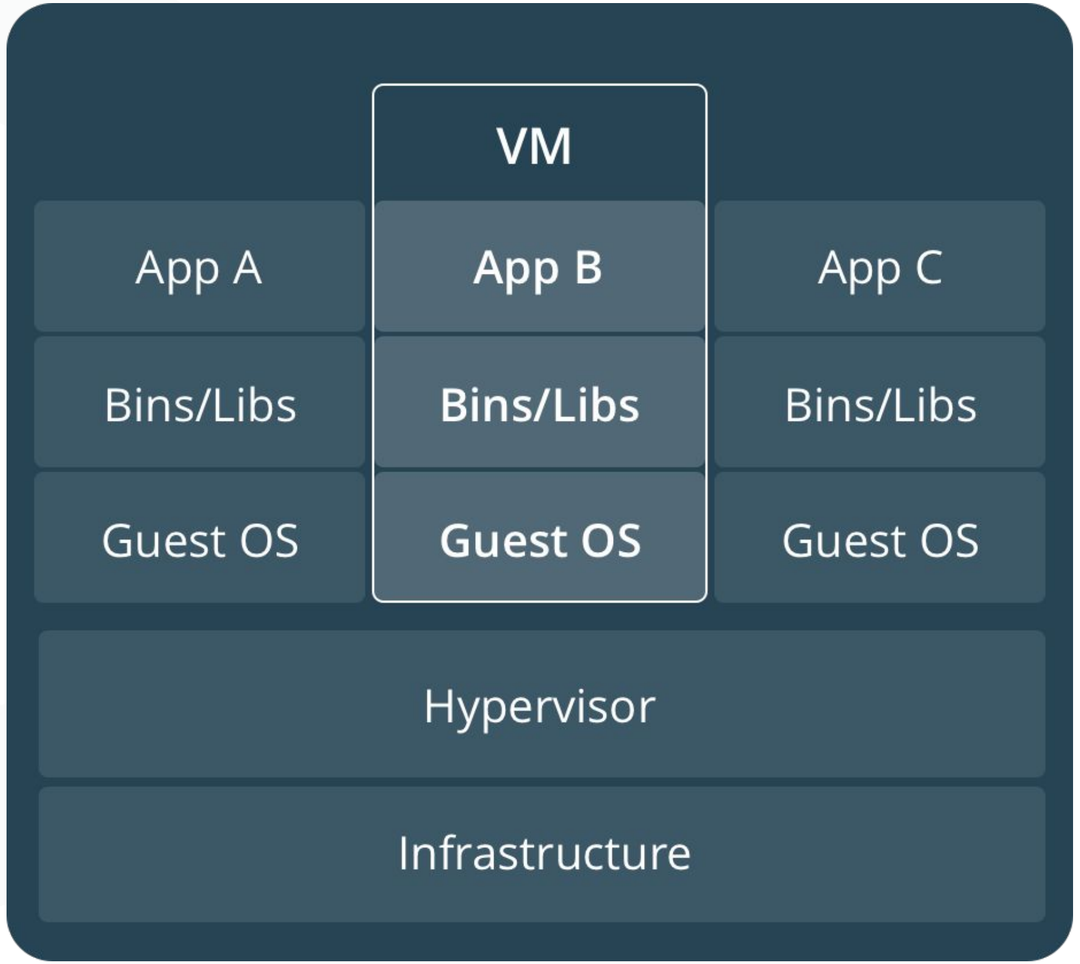
Image: A frozen snapshot of a container.

Dockerfile: An (often) short text-file detailing how to make an image from another image.

Volume: A chunk of filesystem (“folder”) that can be accessed by both containers as well as the host computer, and can persist.

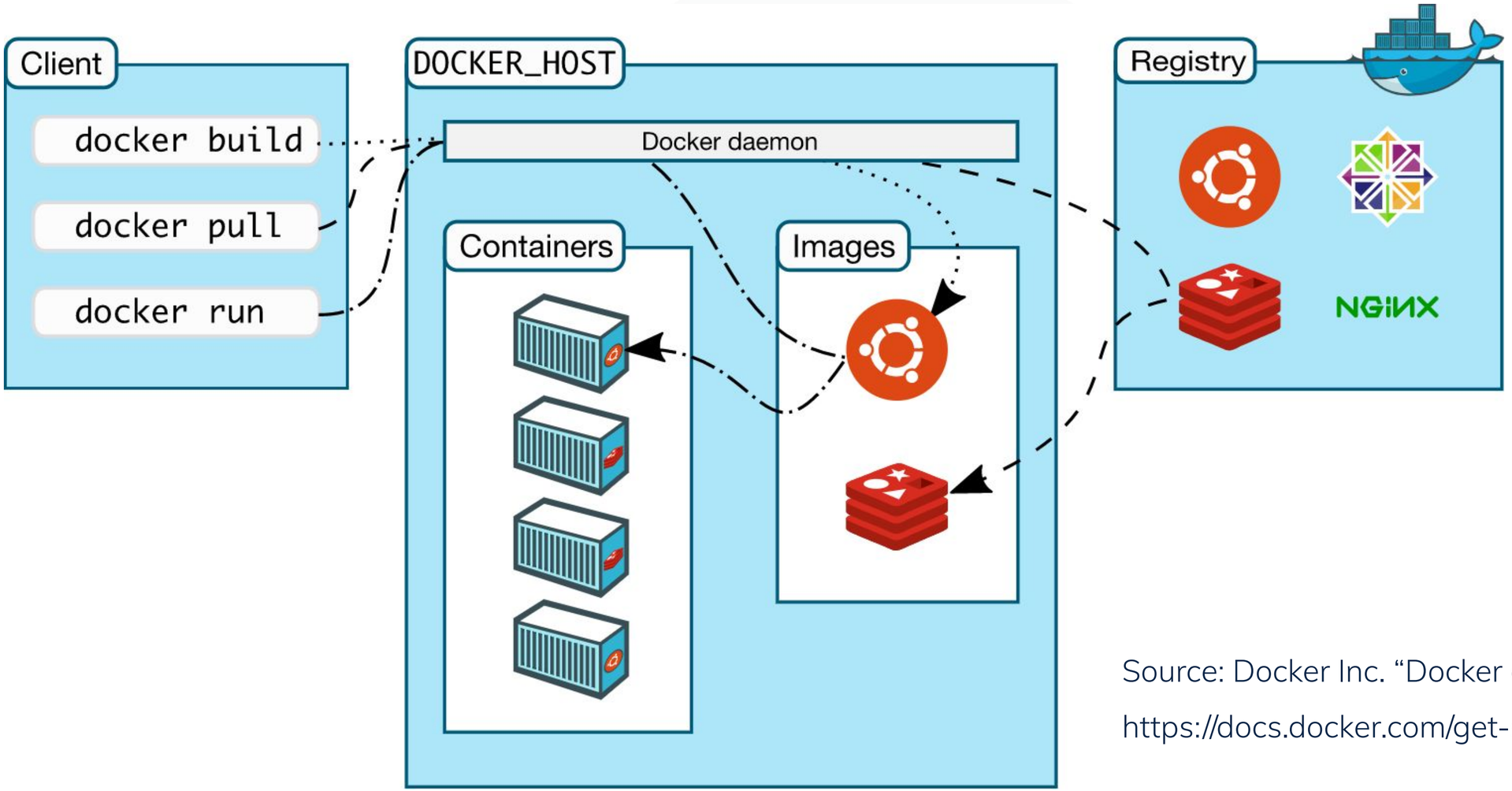
Definitions: Containers vs Virtual Machines





bigdata republic

How it actually works



Source: Docker Inc. “Docker overview” located at https://docs.docker.com/get-started/overview/

Search

OFFICIAL REPOSITORY

python ☆

Last pushed: 5 hours ago

Repo Info | Tags

Short Description

Python is an interpreted, interactive, object-oriented, open-source programming language.

Full Description

Supported tags and respective `Dockerfile` links

Simple Tags

- `3.7.0b3-stretch`, `3.7-rc-stretch`, `rc-stretch` *(3.7-rc/stretch/Dockerfile)*
- `3.7.0b3-slim-stretch`, `3.7-rc-slim-stretch`, `rc-slim-stretch`, `3.7.0b3-slim`, `3.7-rc-slim`, `rc-slim` *(3.7-rc/stretch/slim/Dockerfile)*
- `3.7.0b3-alpine3.7`, `3.7-rc-alpine3.7`, `rc-alpine3.7`, `3.7.0b3-alpine`, `3.7-rc-alpine`, `rc-alpine` *(3.7-rc/alpine3.7/Dockerfile)*

Building our first container

Using Docker

Dockerfile basics

```
1  FROM ubuntu:18.04
2  RUN apt-get update && apt-get install -y python python-pip
3  RUN pip install numpy
4  COPY ./calculate_pi.py /
5  ENTRYPOINT [ "python", "/calculate_pi.py" ]
6  CMD ["1000"]
7
```

https://github.com/BigDataRepublic/docker_webinar

bigdata republic

Dockerfile basics

- Constructed as layers
- Each instruction ~ 1 layer
- Start with base image - FROM
- Copy files to image - COPY or ADD
- Execute shell commands - RUN
- Run model - ENTRYPOINT with CMD

Image layers

- Each layer read-only
- Layer stores changes from previous layer

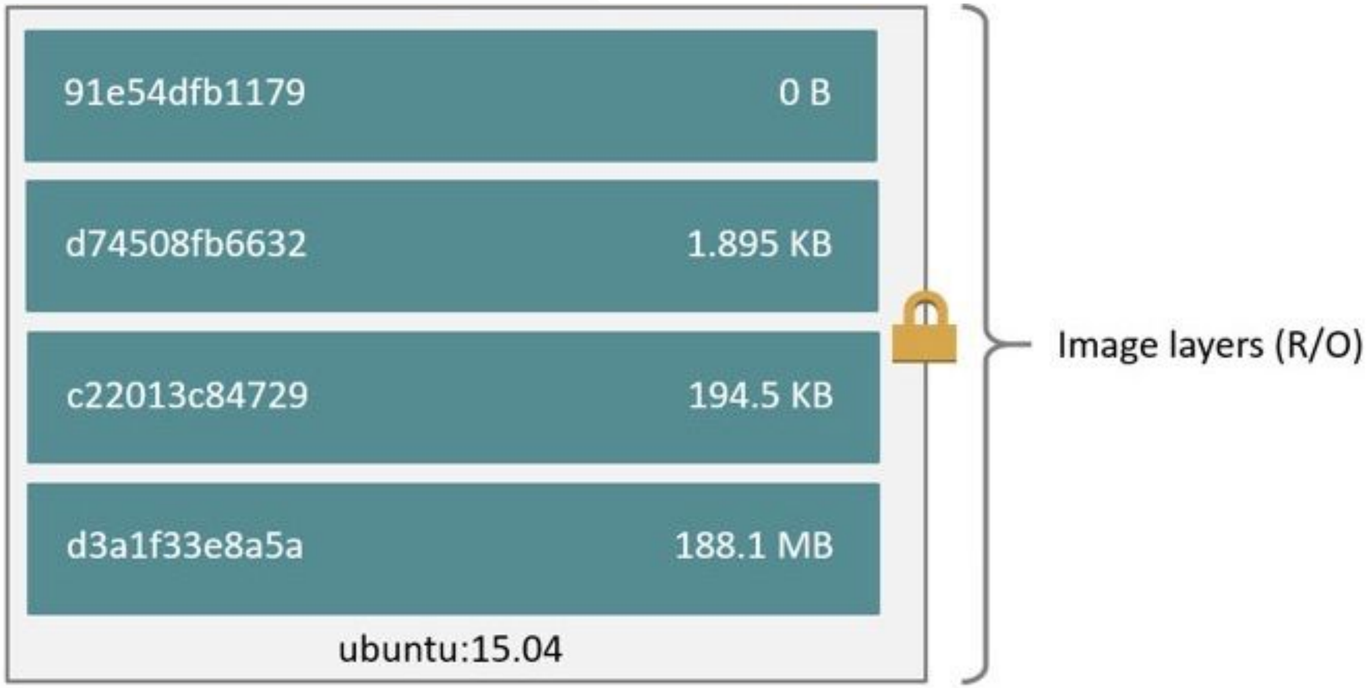
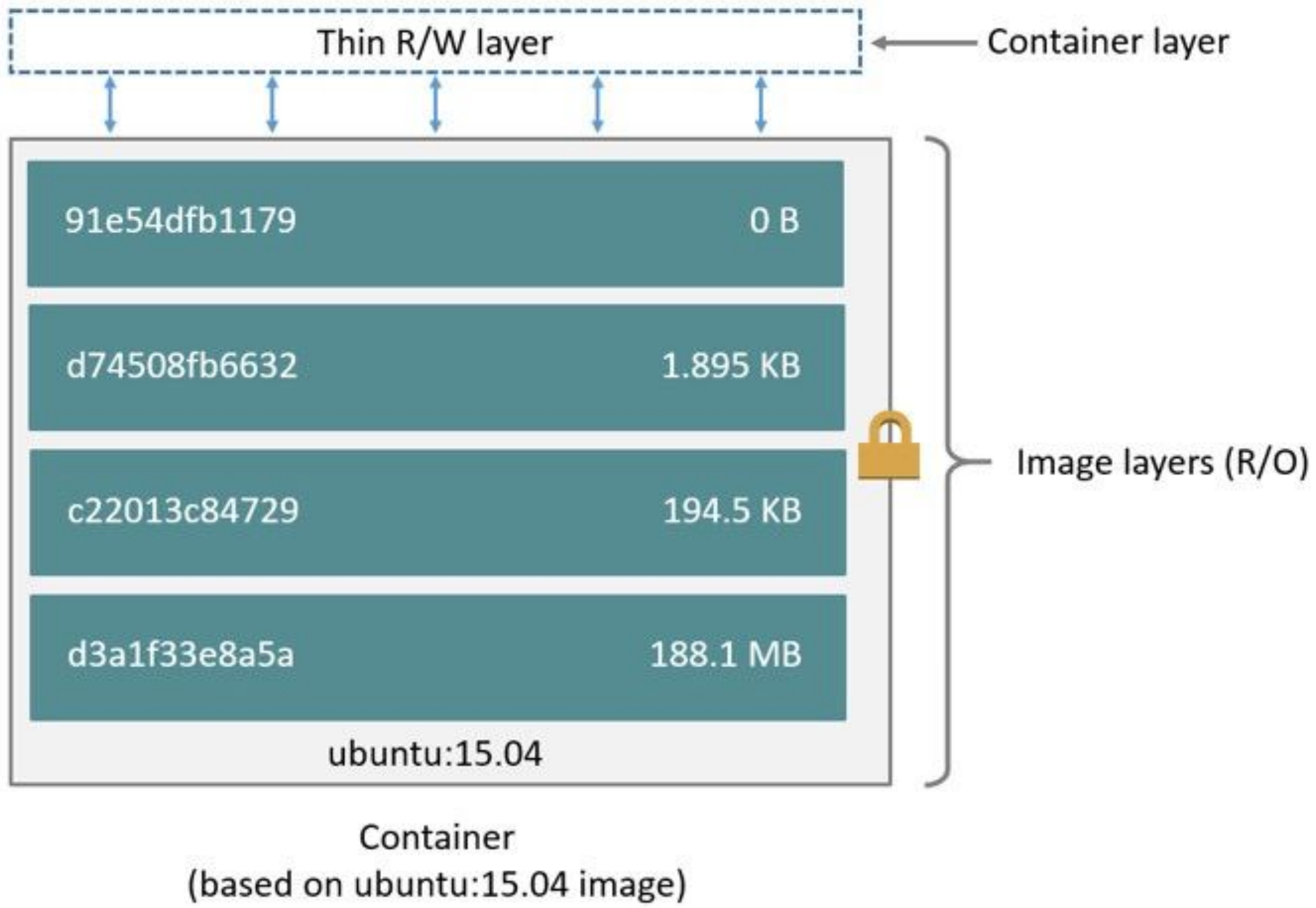


Image layers → container

- Each layer read-only
- Layer stores changes from previous layer
- Container > adds writeable layer on last image layer



Running it all

- docker build -t approx_pi -f Dockerfile_first .
- docker run approx_pi:latest

```
docker build -t approx_pi -f Dockerfile_first .
Sending build context to Docker daemon  587.8kB
Step 1/6 : FROM ubuntu:18.04
 ---> 4e5021d210f6
Step 2/6 : RUN apt-get update && apt-get install -y python python-pip
 ---> Using cache
 ---> 19ccdd3989ae
Step 3/6 : RUN pip install numpy
 ---> Using cache
 ---> 4551ccf0b8a2
Step 4/6 : COPY ./calculate_pi.py /
 ---> Using cache
 ---> 5c5d4bb4a615
Step 5/6 : ENTRYPOINT [ "python", "/calculate_pi.py" ]
 ---> Using cache
 ---> 2ef60e6036a5
Step 6/6 : CMD ["1000"]
 ---> Using cache
 ---> 523087ef7bd6
Successfully built 523087ef7bd6
```

Training a model with Docker

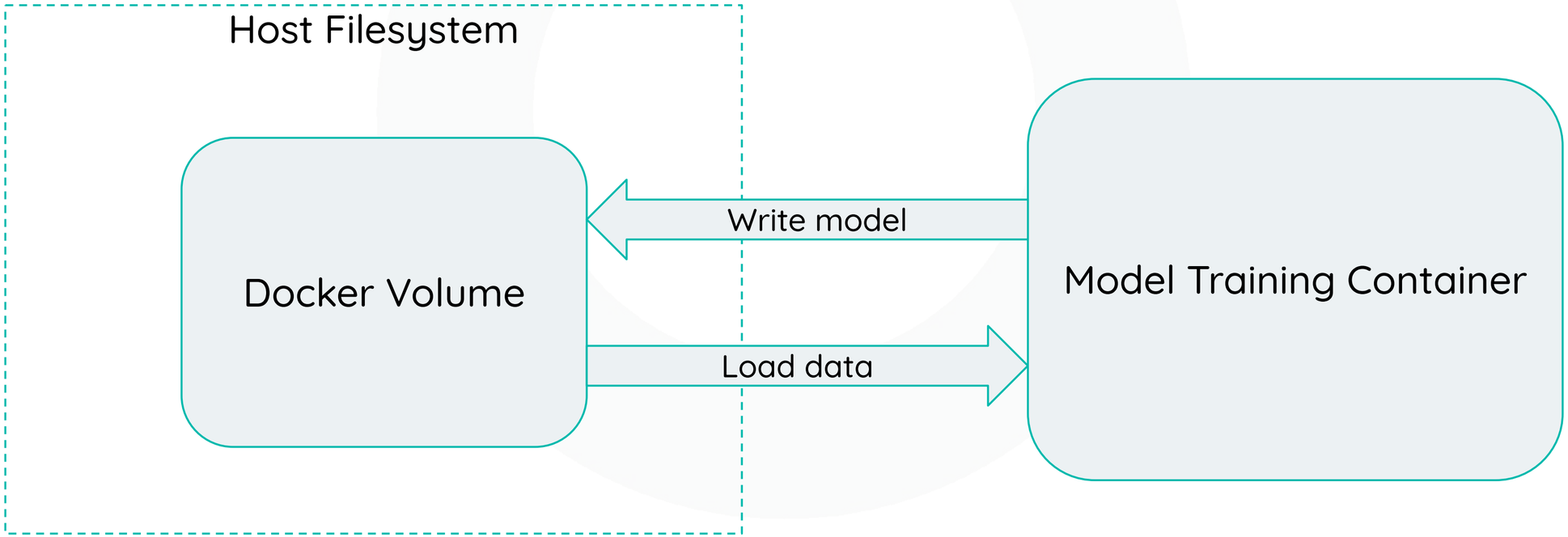
Using Docker for Data Science

Thursday, April 30, 2020

bigdata republic

Training & saving models

- Python model training in container
- Need to retrieve our model after training
- Docker volumes: sharing storage



Training & saving models

- To initialize with local filesystem folder:
  docker run -v /foo/data:/app/data my_container

- docker build -t iris_trainer -f Dockerfile_training .
- docker run -v /full/path/to/model:/app/model iris_trainer

Training & saving models continued

Dockerfile_training

```
FROM ubuntu:18.04

LABEL version="1.0"
LABEL description="This container trains a random forest on the iris dataset."
LABEL maintainer="joris.bukala@bigdatarepublic.nl,jacobus.herman@bigdatarepublic.nl"

COPY --chown=root:root ./security/add_user_group.sh ./training_app/requirements.txt /app/
RUN apt-get update -y && \
    apt-get install -y python-pip python-dev && \
    pip install -r /app/requirements.txt && \
    chmod 550 /app/add_user_group.sh && \
    /app/add_user_group.sh && \
    rm /app/add_user_group.sh && \
    rm -rf /var/lib/apt/lists/* && \
    mkdir -p /app/model && \
    chown -R dockerapp:dockerapp /app/model

USER dockerapp

COPY --chown=dockerapp:dockerapp ./training_app /app

VOLUME [ "/app/model" ]
ENTRYPOINT [ "python" ]
CMD [ "/app/train.py" ]
```

Serving predictions

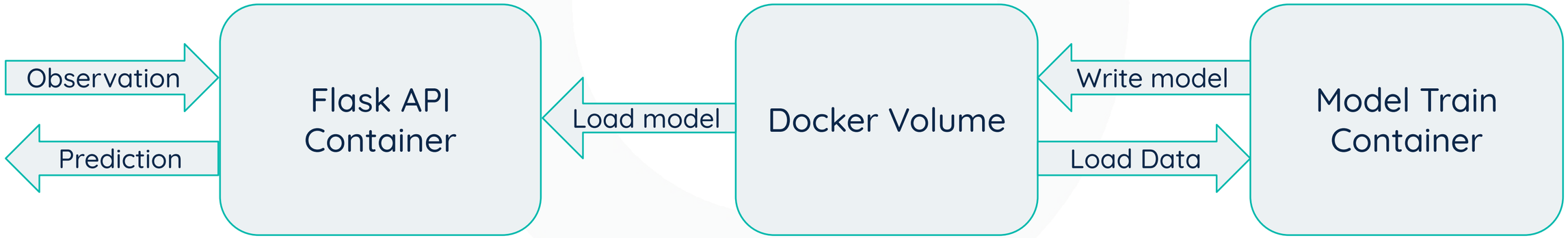
Using Docker for Data Science

Thursday, April 30, 2020

bigdata republic

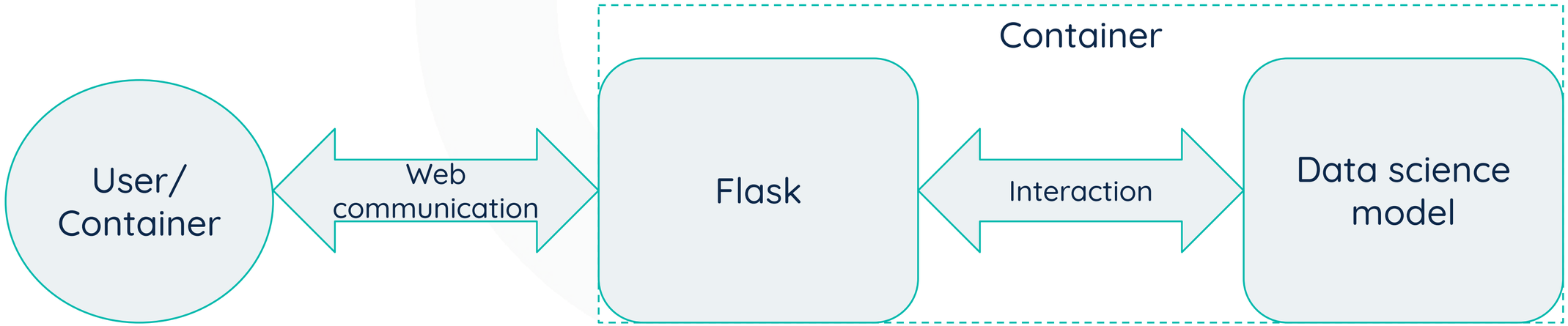
Serving models

- Use trained model with Flask API
- Share same volume
- Receives array with 2 numbers & returns number



What is Flask?

- Micro framework for building web applications
- Allows creation of interface to something through web communication
- Similar to website: user interacts with content
- End result (web) Application Programming Interface (API)



Flask source: https://palletsprojects.com/p/flask/

Serving models

- docker build -t iris_predictor -f Dockerfile_prediction .
- docker run -p 5000:5000 -v ~/model_volume:/app/model iris_predictor

- -p maps host port to container port
- Communication to 5000 on host goes to 5000 on container (like a mirror)
- Ports can be different, e.g. -p 80:5000

Serving models image

Dockerfile_prediction

```
FROM ubuntu:18.04

LABEL version="1.0"
LABEL description="This container exposes a fitted random forest model with Flask on the Iris dataset."
LABEL maintainer="joris.bukala@bigdatarepublic.nl,jacobus.herman@bigdatarepublic.nl"

COPY --chown=root:root ./security/add_user_group.sh ./prediction_app/requirements.txt /app/
RUN apt-get update -y && \
    apt-get install -y python-pip python-dev && \
    pip install -r /app/requirements.txt && \
    chmod 550 /app/add_user_group.sh && \
    /app/add_user_group.sh && \
    rm /app/add_user_group.sh && \
    rm -rf /var/lib/apt/lists/* && \
    mkdir -p /app/model && \
    chown -R dockerapp:dockerapp /app/model

USER dockerapp

COPY --chown=dockerapp:dockerapp ./prediction_app /app

ENV FLASK_APP=/app/main.py

VOLUME [ "/app/model" ]
EXPOSE 5000
ENTRYPOINT [ "flask" ]
CMD [ "run", "--host=0.0.0.0" ]
```

Making a front-end

Using Docker for Data Science

Thursday, April 30, 2020

bigdata republic

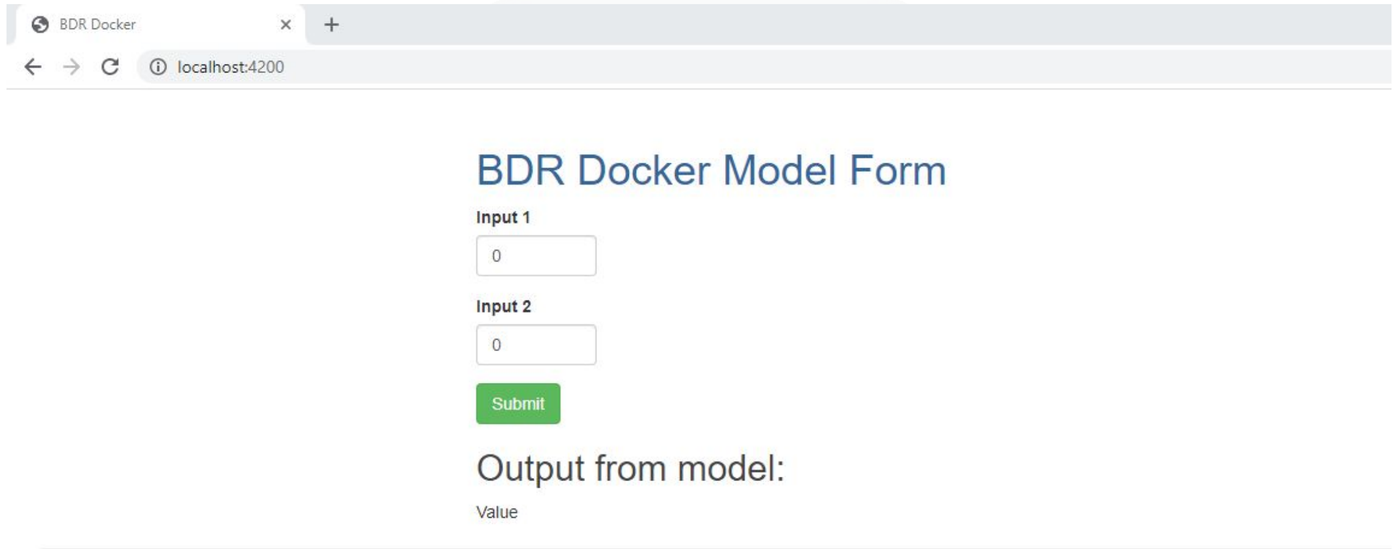
Add web UI

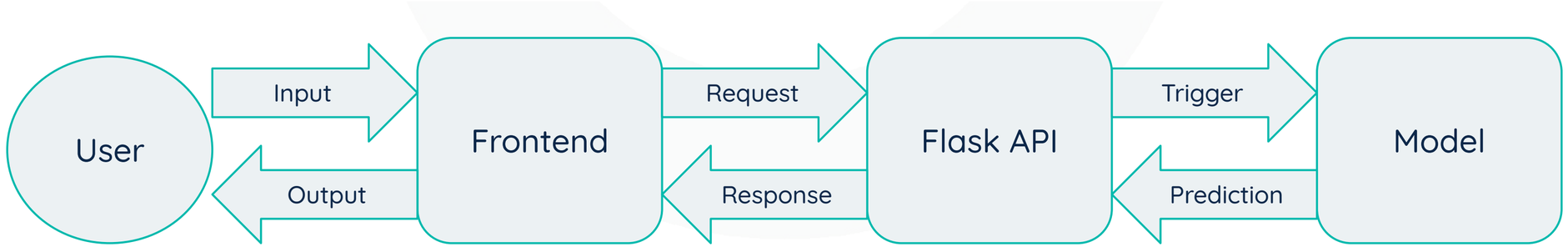
- AngularJS web application for access to production model
- Frontend container interacts with prediction container

<u>To run:</u>

- docker build -t iris_frontend -f Dockerfile_frontend .
- docker run -d -p 4200:4200 iris_frontend

Web UI flow


BDR Docker
localhost:4200
BDR Docker Model Form
Input 1
0
Input 2
0
Submit
Output from model:
Value


User
Input
Output
Frontend
Request
Response
Flask API
Trigger
Prediction
Model

Web UI image

Dockerfile_frontend

```
FROM node:12.16.2

LABEL version="1.0"
LABEL description="This container runs an Angular JS app that connects to a REST API on localhost:5000."
LABEL maintainer="joris.bukala@bigdatarepublic.nl,jacobus.herman@bigdatarepublic.nl"

WORKDIR /app

ENV PATH /app/node_modules/.bin:$PATH

COPY --chown=root:root ./security/add_user_group.sh ./frontend/model-display/package.json /app/
RUN npm install && \
    chmod 550 /app/add_user_group.sh && \
    /app/add_user_group.sh && \
    rm /app/add_user_group.sh

USER dockerapp

COPY --chown=dockerapp:dockerapp ./frontend/model-display /app/

EXPOSE 4200
ENTRYPOINT [ "ng", "serve" ]
CMD [ "--host", "0.0.0.0", "--disableHostCheck", "true" ]
```

Best practices

For writing Dockerfiles

bigdata republic

Docker-compose

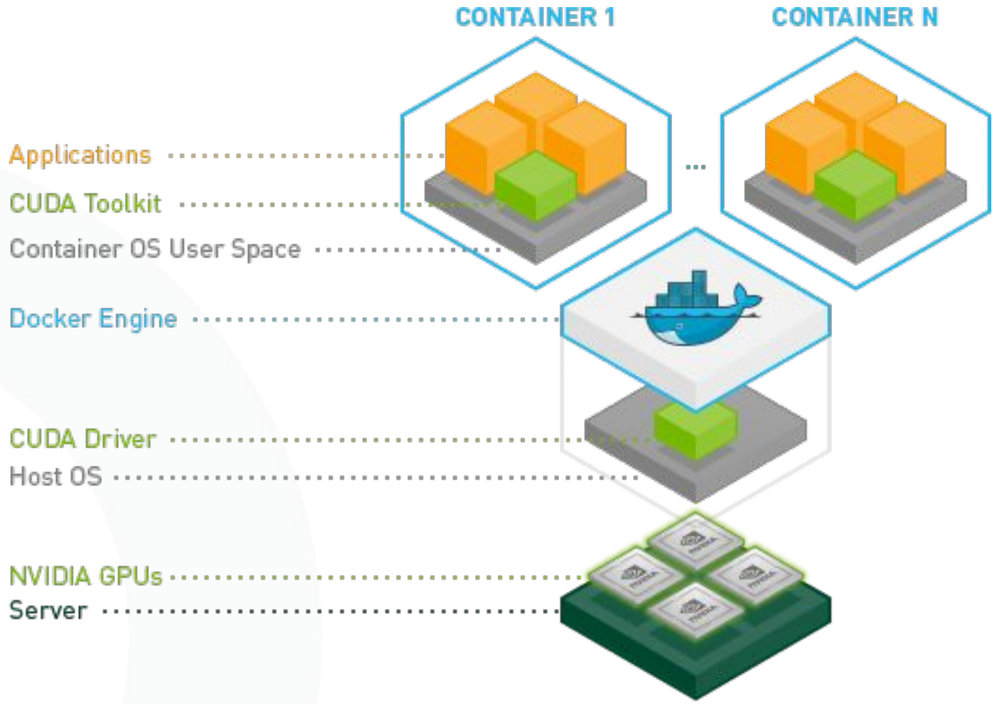
- All containers with arguments in one config file
- docker-compose up -d
- docker-compose down

```yaml
version: "3.4"
services:
    iris_frontend:
        build:
            context: .
            dockerfile: Dockerfile_frontend
        ports:
            - "4200:4200"
        depends_on:
            - iris_prediction
        networks:
            default:
                aliases:
                    - frontend
    iris_prediction:
        build:
            context: .
            dockerfile: Dockerfile_prediction
        ports:
            - "5000:5000"
        depends_on:
            - iris_training
        volumes:
            - "iris_model_vol:/app/model"
        networks:
            default:
                aliases:
                    - rest-api
    iris_training:
        build:
            context: .
            dockerfile: Dockerfile_training
        volumes:
            - "iris_model_vol:/app/model"
volumes:
    iris_model_vol:
networks:
    default:
```

Using a GPU with NVIDIA-Docker

- Training/ preparing DL model on your (Linux) laptop

- Quick deployment to Cloud service when necessary




```
#### Test nvidia-smi with the latest official CUDA image
docker run --gpus all nvidia/cuda:10.0-base nvidia-smi
```

Dockerfile writing

- Use .dockerignore: avoids unnecessary files, speeds up build process

```
**/.env
**/.idea
**/.vscode
**/.git
**/.cache
**/docker-compose.yml
**/*.md
**/node_modules
```

- Create stateless images: able to restart without data loss

- Single purpose: decoupling interacting parts for simple containers

- Multi-stage builds: single image file to build and deploy app

```
FROM golang:1.7.3
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=0 /go/src/github.com/alexellis/href-counter/app .
CMD ["./app"]
```

Dockerfile writing continued

- Minimize layers: combine RUN statements
- Frequently changing commands bottom of Dockerfile
- Reduce container size:
  - Use Alpine Linux base image for smallest size
  - Add rm -rf /var/lib/apt/lists/* to apt-get install
  - Try --no-install-recommends flag in apt-get install
- Use trusted official base images
- Specify image versions

```
$ docker build -t mynewimage .
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM python:3.8-slim-buster
 ---> 3d8f801fc3db
Step 2/3 : COPY build.sh .
 ---> 541b65a7b417
Step 3/3 : RUN ./build.sh
 ---> Running in 9917e3865f96
Building...
```

Summary & next steps

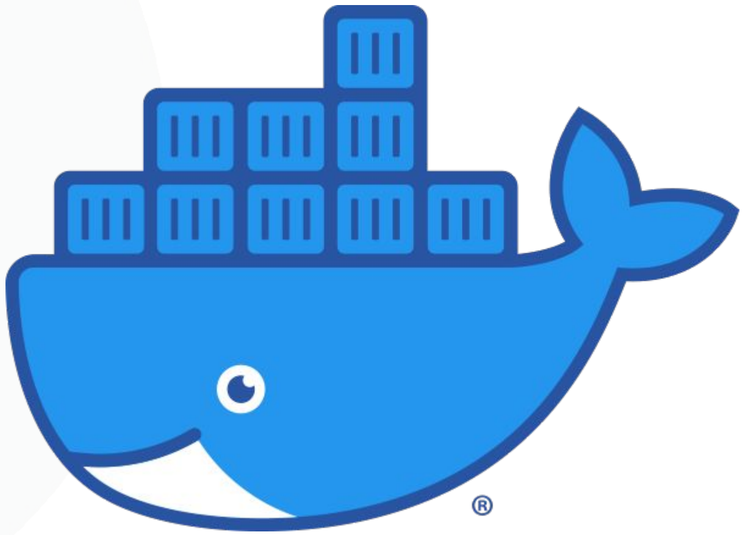
Docker usage

Summary

- Docker decouples project from environment → reproducibility
- Eases sharing of models → better collaboration
- Improves deployability → greater impact

- Be aware of security and optimizations

Next steps: Docker-swarm & Kubernetes



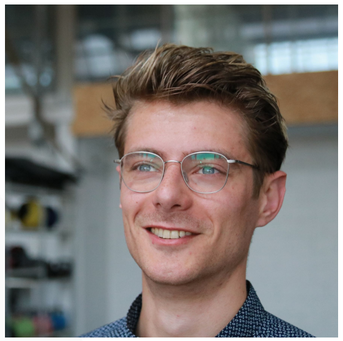
https://github.com/BigDataRepublic/docker_webinar

Presenters and contributors

Tom de Ruijter

Data Scientist

/in/tomderuijter/



Joris Bukala

Data Scientist

/in/joris-bukala/

Jacobus Herman

Data Engineer

/in/jacobus-herman/

bigdata republic

Phone +31 (0)168 479294 **Email** info@bigdatarepublic.nl **Address** Coltbaan 4C, 3439 NG Nieuwegein, The Netherlands