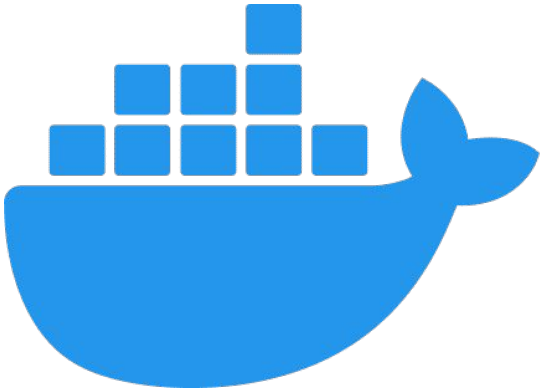


docker

in 60 minutes

KOM KWT 2022 / Workshop

WHY?

..and how?

What is Docker?

> “ Docker is an open platform for developing, shipping, and running applications.”
>
> https://docs.docker.com/get-started/overview/

Main idea

- Put applications into lightweight containers
- Use containers in all stages: development, testing and production
- Hardware-independent deployment (target only needs docker)
- Orchestration and scaling of modular applications

Points based on https://docs.docker.com/get-started/overview/

Why should you care?

As an independent developer

- Specify your whole setup at one place
- Build once, **run anywhere** (local machine, VM…)
- Containerized performance nearly identical to native*
  (also for ML workloads with GPUs)

As a researcher

- Isolated and safe environment for executing apps
- Use exactly the same setup between collaborators
- **Boost reproducibility of your research**

Most arguments are taken from https://pointful.github.io/docker-intro/
* depending on your workloads, see e.g. https://ieeexplore.ieee.org/document/7095802

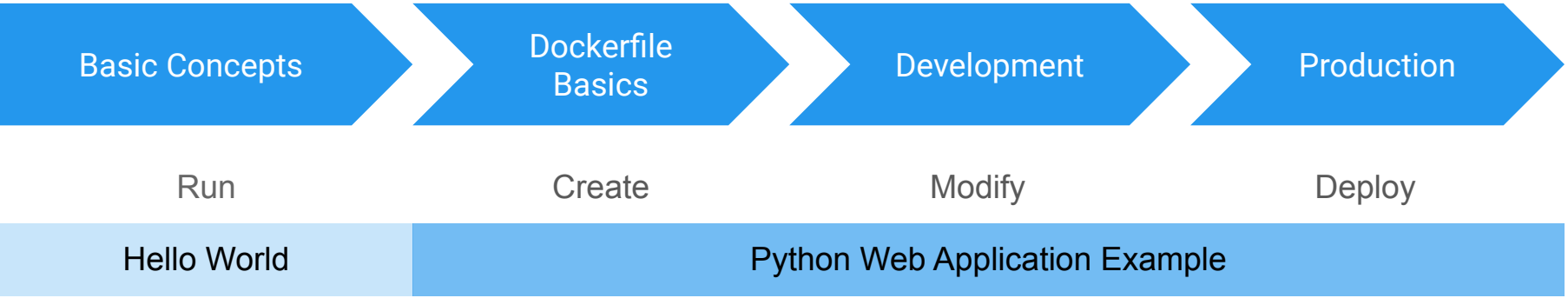
This Workshop on GitHub

```
git clone https://github.com/jw3il/kwt22-docker
```


QR code generated with https://www.qrcode-monkey.com/

What you will learn in the next ~ 55 minutes..

Concepts required for (basically) all projects

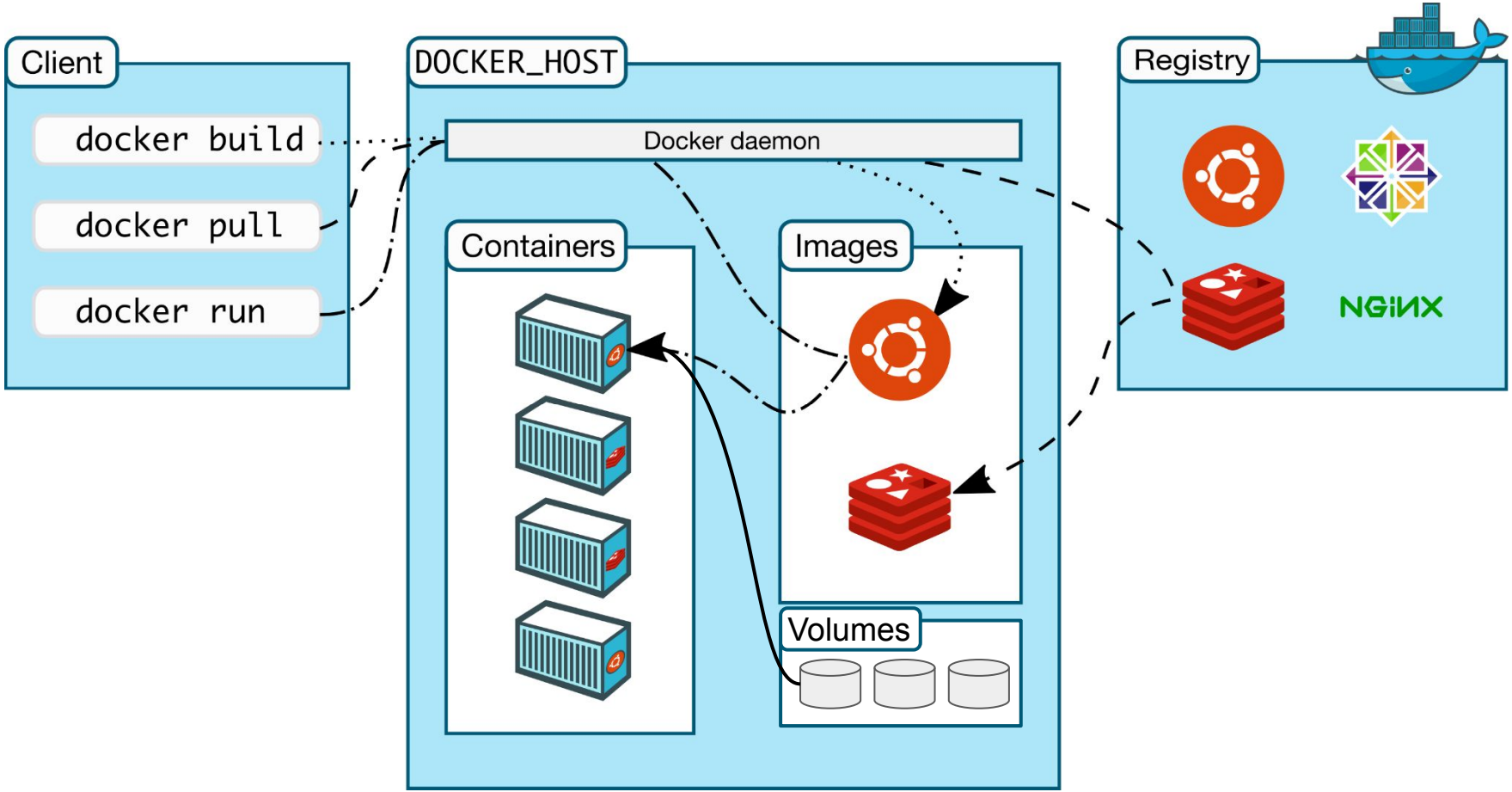


Basic Concepts

Images

Containers

Volumes



Architecture image from https://docs.docker.com/engine/images/architecture.svg (modification: added volumes)

Hands-On Session

Complete step 1

(see README.md in the git repository)

Dockerfile Basics

A recipe to build and run your application!

Dockerfile Basics

```
FROM python:3.9-slim-bullseye

# set the work dir in the image to /app
WORKDIR /app

# install requirements
RUN pip install --no-cache-dir flask

# copy files from local system to /app
COPY ./app ./

# set path to flask app and enable development mode
ENV FLASK_APP=app.py
ENV FLASK_DEBUG=1

# start flask when running containers based on this image
ENTRYPOINT [ "flask", "run" ]
CMD ["--host", "0.0.0.0", "--port", "5000"]
```

base image `name:tag`

install requirements & **remove garbage**

run command and default args

python™ (on debian bullseye)

We **build** an image with a Dockerfile

Docker creates *layers* after each statement

- Each layer is an intermediate image based on the previous image
- Allows for caching & faster downloads of modified images
- But: each layer increases the size of the image (like a git commit)

Logo from https://www.python.org/static/community_logos/python-logo-generic.svg

Hands-On Session

Complete step 2

Development

How to use docker
for development

Development with Docker

We need persistent storage for our code..

Volumes

- Let docker manage storage
- Isolated storage units
- Mount in multiple containers

https://docs.docker.com/storage/volumes/

Bind Mounts

- Mount local filesystem
- Easy to use
- Do not use in production

https://docs.docker.com/storage/volumes/

Development with Docker - Bind Mounts

local filesystem

Mount the directory `/$(pwd)/app` as `/app`

docker filesystem

```
$ docker run --rm --name kwt22-container
-v "/$(pwd)/app":"/app" -p 8080:5000 kwt22-image
```


Alternatively, you can also use `--mount`, which is more verbose (see https://docs.docker.com/storage/bind-mounts/ for further details and options)

Hands-On Session

Work on step 3

Production

Create different development and production images

Getting ready for production

Goal: create a (small) image to run your application

Main options

1. Create multiple Dockerfiles (e.g. development, production)
2. Create multi-stage builds with a single Dockerfile

Multi-Stage Builds by Example

```
# development image
FROM base-image as dev

# ... do some stuff

ENTRYPOINT ...
CMD ...

# production image
FROM dev as prod

# ... do some stuff

ENTRYPOINT ...
CMD ...
```

Stage 1: `--target dev`

- Like our previous image
- Keyword **as** defines build target name
- Base stage could e.g. be slim build target with all common and installed dependencies
- In our case: development target
  (we have no unnecessary dependencies)

Stage 2: `--target prod`

- Defines own build target
- Can be based on previous (**dev**) or different image
- We could also copy files from previous stages like

```
COPY --from=dev DEV_PATH PROD_PATH
```

Note that an arbitrary number of stages can be defined according to your needs
More info at https://docs.docker.com/develop/develop-images/multistage-build/

Hands-On Session

Complete step 4

Congratulations!

You are ready to use docker
in your own projects