Docker Workshop

Containerize all the things

Agenda

Workshop Content

- Docker in Theory
  - How it works
  - Dockerfiles - Getting started API Overview
  - .dockerignore
  - Docker CLI
  - Docker Compose

Docker in Theory

How it works

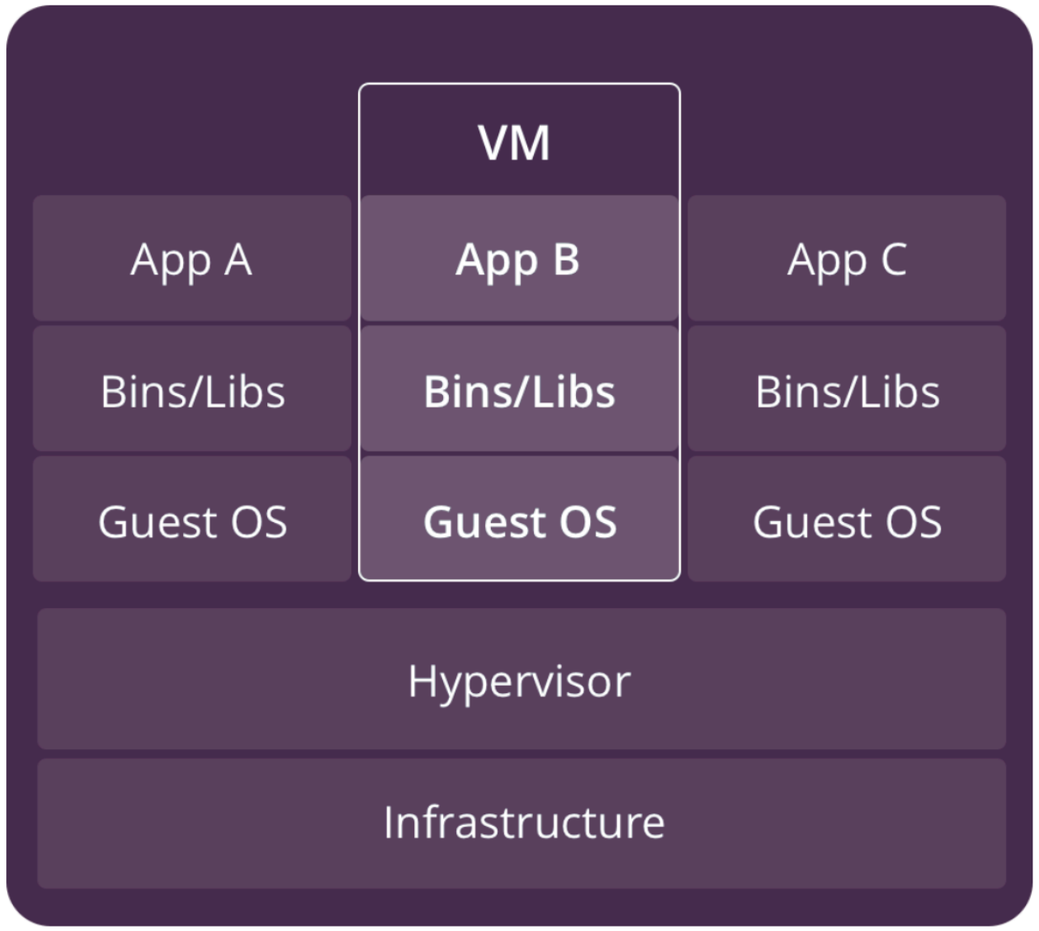
Docker in Theory

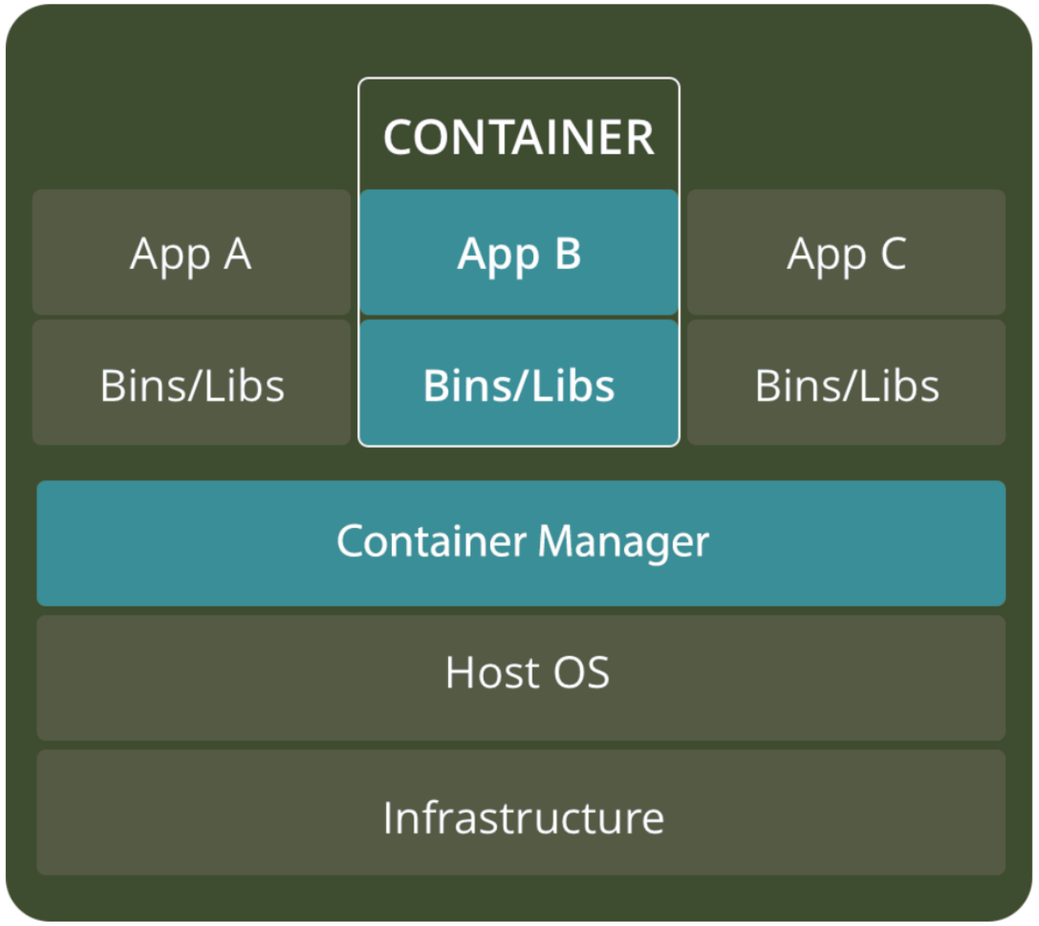
How it works

- Containers sit on top of a physical server and its host OS
- Each container shares the host OS kernel and the binaries and libraries, too
- Server can run multiple workloads with a single operating system installation
- Containers are exceptionally light and take just seconds to start
- create a portable, consistent operating environment for development, testing, and deployment

Docker in Theory

How this looks like





Dockerfiles

Getting started API Overview

Dockerfiles

Getting started API Overview

- Each Dockerimage is defined and described as Dockerfile
- A Dockerfile is a Textfile named as „Dockerfile“ or has the „dockerfile“ File-Extension
- Dockerfiles have a defined API with several Instructions, describe and explained in the official Docker Documentation
  - https://docs.docker.com/engine/reference/builder/
- Basic Instructions are explained on the following Slides

Dockerfile Instruction: FROM

FROM [--platform=<platform>] <image>[:<tag>] [AS <name>]

- initializes a new build stage and sets the Base Image
- a valid Dockerfile must start with a FROM instruction.
- The image can be any valid image
- [AS <name>] is used for multi stage images (later topic)

```
FROM golang:1.14 as builder

WORKDIR /var/www
COPY . .

RUN go get -d -v \
    && go install -v

RUN make build
```

```
FROM nginx:1-alpine

ENV TZ "Europe/Berlin"

COPY nginx.conf /etc/nginx/conf.d/default.conf
```

Dockerfile Instruction: ARG

ARG <name>[=<default value>]

- define variables that users can pass at build-time

```
FROM node:13-alpine as build

WORKDIR /var/www/app

ARG sentry=""
ENV VUE_APP_SENTRY=$sentry
```

```
ARG VERSION=14-alpine
FROM node:$VERSION as build

WORKDIR /var/www/app
```

Dockerfile Instruction: RUN

RUN <command> / RUN ["executable", "param1", "param2"]

- execute any commands in a new layer on top of the current image and commit the results.
- the resulting committed image will be used for the next step in the Dockerfile.
- exec form has no shell processing (e.g. RUN ["echo", "$HOME"] will not do variable substitution on $HOME)

```
RUN apk add --no-cache dbus postgresql-dev autoconf make g++ openrc oniguruma-dev \
    && docker-php-ext-install \
        mbstring \
        pgsql \
        pdo_pgsql \
        bcmath \
        sockets \
    && pecl install swoole redis \
    && apk del make autoconf \
    && docker-php-ext-enable swoole \
    && docker-php-ext-enable redis \
    && ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone
```

Dockerfile Instruction: WORKDIR

WORKDIR /path/to/workdir

- sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile

```
FROM alpine:latest

ENV TZ "Europe/Berlin"

WORKDIR /var/www

COPY --from=dependencies /deps/vendor ./vendor
COPY bin bin
COPY config config
COPY etc etc
COPY src src
COPY public public
COPY templates templates
COPY translations translations
COPY composer.json composer.json
COPY composer.lock composer.lock
COPY symfony.lock symfony.lock

RUN mkdir -p -m 0777 /var/www/var/log \
    && mkdir -p -m 0777 /var/www/var/cache \
```

Dockerfile Instruction: COPY

COPY [--chown=<user>:<group>] <src>... <dest>

- copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>
- Optionally COPY accepts a flag --from=<name> that can be used to set the source location to a previous build stage
- src is absolute or relativ to context dir
- dest is absolute or relative to work dir
- copy inside of context only
- copy invalidate followed
- RUN commands

```
COPY --from=dependencies /deps/vendor ./vendor
COPY bin bin
COPY config config
COPY etc etc
COPY src src
COPY public public
COPY templates templates
COPY translations translations
COPY composer.json composer.json
COPY composer.lock composer.lock
COPY symfony.lock symfony.lock
```

```
FROM node:11-alpine as dependencies

WORKDIR /var/www/app

COPY package*.json ./
```

Dockerfile Instruction: ADD

ADD [--chown=<user>:<group>] <src>... <dest>

- works like copy, except it can use an URL as src and unpack archives

```
ADD http://foo.com/bar.go /tmp/main.go
```

```
ADD /foo.tar.gz /tmp/
```

Dockerfile Instruction: EXPOSE

EXPOSE <port> [<port>/<protocol>...]

- informs Docker that the container listens on the specified network ports at runtime.
- You can specify whether the port listens on TCP or UDP
  - default is TCP if the protocol is not specified

```
EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

Dockerfile Instruction: ENV

ENV <key>=<value> ...

- sets the environment variable <key> to the value <value>.
- This value will be in the environment for all subsequent instructions
- The environment variables set using ENV will persist when a container is run from the resulting image.

```
ARG VERSION=14-alpine
FROM node:$VERSION as build

WORKDIR /var/www/app

ARG sentry=""
ENV VUE_APP_SENTRY=$sentry
```

Dockerfile Instruction: ENV

VOLUME <target> / VOLUME ["/data"]

- creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers.
- the docker run command initializes the newly created volume with any data that exists at the specified location within the base image.

```
FROM php:7.4.3-cli-alpine

COPY --from=composer /usr/bin/composer /usr/bin/composer

ENV TZ "Europe/Berlin"

WORKDIR /app

VOLUME ["/app"]

CMD ["/usr/bin/composer", "install", "--ignore-platform-reqs"]
```

Dockerfile Instruction: ENTRYPOINT

ENTRYPOINT ["executable", "param1", "param2"]

- allows you to configure a container that will run as an executable

```
ENTRYPOINT ["/var/www/server"]
```

Dockerfile Instruction: ENTRYPOINT

CMD ["executable", "param1", "param2"]

- The main purpose of a CMD is to provide defaults for an executing container.
- These defaults can include an executable, or they can omit the executable, in which case you must specify an ENTRYPOINT instruction as well

```
CMD ["/usr/bin/composer", "install", "--ignore-platform-reqs"]
```

```
ENTRYPOINT ["/usr/bin/composer"]
CMD ["install", "--ignore-platform-reqs"]
```

.dockerignore

How to simplify COPY and ADD instructions

.dockerignore

Simplify COPY and ADD

- COPY and ADD instructions by default copy all files and directories within the source if its a directory
- .dockerignore files work similar to .gitignore or .npmignore. All listed files will not copied by an COPY or ADD instruction within your Dockerfile
- You can define .dockerignore files per Dockerfile since Docker 19.03 with a special naming <Dockerfile>.dockerignore and .dockerignore works as fallback. (You need to enable BuildKit mode)

.dockerignore

You, a few seconds ago

```
var
config/jwt
live-postgres-*
Dockerfile
.github
README.md
Jenkinsfile
```

Docker CLI

build - monitor - run - delete

docker build

From a Dockerfile to an Dockerimage

- Usage: docker build [OPTIONS] PATH | URL | -
  - Example: docker build ./ —file Dockerfile —tag name[:version]
  - Common options for builds are
    - —file used Dockerfile (default „Dockerfile) used if multiple Dockerfiles in your project exist
    - —tag used to tag your image, the tag is used to run or publish the Dockerimage
    - —build-arg set or overwrite ARGs inside your Dockerfile
  - The first Argument for docker build is the so called build context and defines the root path within your Dockerfile for relative source paths in

docker run

Run created Dockerimages

- Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
  - Example: docker run name[:version]
  - Common options for run are
    - -d / —detach run containers in background
    - —rm remove the container when it exists
    - -p / —publish maps container ports to the host, e.g.: 8080:80 maps the exposed port 80 to 8080 on localhost
    - —env / —env-file set or overwrite ENV variables in the running container
    - —name assign a custom name to the container
    - -v / —volume bind a local path as mount volume into the container

docker logs / exec / ps

Debug your Container

- docker logs [OPTIONS] CONTAINER
  - Example: docker logs <name I CONTAINERID>
  - Common options for run are
    - -f / —follow follow log output
- docker exec [OPTIONS] CONTAINER COMMAND [ARG…]
  - Common usage: docker exec -it <name I CONTAINERID> sh
    - Connects you into the Container with an simple shell
  - Could also use to execute simple commands within a container
- docker ps [OPTIONS]
  - List created and running containers with additional Informations like port binding, sizes and names

docker system prune

Cleanup your Disk

- Usage: docker system prune [OPTIONS]
  - Example: docker system prune —all
  - Common options for builds are
    - —all removes all unused images
    - —volumes removes unused volumes

Docker Compose

Put it all together

Docker Compose

Put it all tiger

- docker-compose is an additional tool to organize and link multiple docker container together
- It is used with an configuration file called docker-compose.yaml
- this file provide an YAML based API to preconfigure docker images. You can define exposed ports, volumes, networks or override defaults like entrypoint, command and ENV variables
- You can group containers into networks to improve communication between multiple containers without port conflicts

Docker Compose

CLI interaction

- docker-compose CLI provide different commands to interact with the docker-compose.yaml or managed running containers
- Many commands are very similar to the related docker commands like run, stop, rm, exec, ps

Docker Compose

docker-compose up [OPTIONS] [SERVICE…]

- Start a list or all service definitions in your docker-compose.yaml
- If this service has a dependency to another service this service will also be started
- common options are
  - —build to force a rebuild of the image
  - —detach / -d to run the service in the background

Docker Compose

docker-compose stop [SERVICE…]

- Stop a list or all running containers defined in your docker-compose.yaml

docker-compose rm [SERVICE…]

- Remove a list or all stoped containers defined in your docker-compose.yaml