

How to docker?

# Docker something

Julia Winkler

19.06.2024

# Disclaimer

- bei weitem nicht alles zum Thema Docker
- nur Allgemeine Grundlage



Befehle im Terminal ausführen  
Ordner `examples`, wenn  
nichts da steht



CodeTour (VSCode Plugin)  
Code im Repo

# Gliederung

Einführung

Dockerfile

Einfache Container

Python (FastAPI)

Volumes und Mounts

React

Multistage Builds

Weitere Commands

# Wiso, Weshalb, Warum?

Why use Docker?

**Trusted by developers.  
Chosen by Fortune 100 companies.**

Docker provides a suite of development tools, services, trusted content, and automations, used individually or together, to accelerate the delivery of secure applications.

# Wiso, Weshalb, Warum?

Why use Docker?

**Trusted by developers.  
Chosen by Fortune 100 companies.**

Docker provides a suite of development tools, services, trusted content, and automations, used individually or together, to accelerate the delivery of secure applications.

”a sandboxed process on your machine that is isolated from all other processes on the host machine”

# Wiso, Weshalb, Warum?

Why use Docker?

**Trusted by developers.  
Chosen by Fortune 100 companies.**

Docker provides a suite of development tools, services, trusted content, and automations, used individually or together, to accelerate the delivery of secure applications.

”a sandboxed process on your machine that is isolated from all other processes on the host machine”

”It works on my computer”

Why use Docker?

## **Trusted by developers. Chosen by Fortune 100 companies.**

Docker provides a suite of development tools, services, trusted content, and automations, used individually or together, to accelerate the delivery of secure applications.

"a sandboxed process on your machine that is isolated from all other processes on the host machine"

"It works on my computer"

"faster onboarding and testing while also simplifying the deployment of services"



Wiso, Weshalb, Warum?

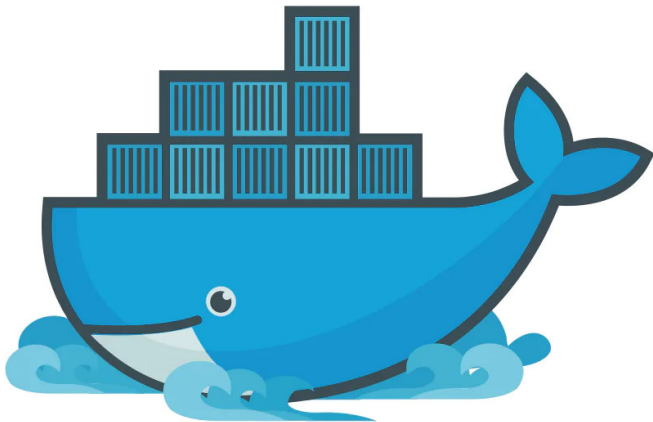


Wiso, Weshalb, Warum?



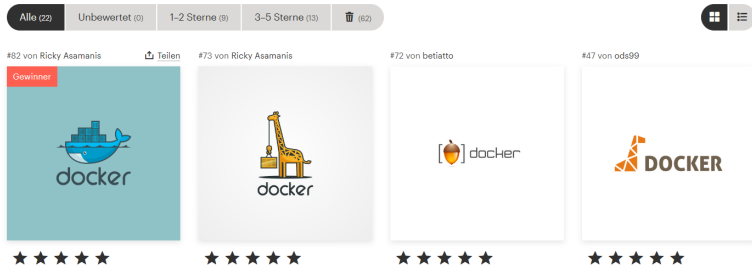
Wer ist Moby Dock?

Wer ist Moby Dock?



# Wer ist Moby Dock?

## Entwürfe



Wettbewerb zum Icon für Docker

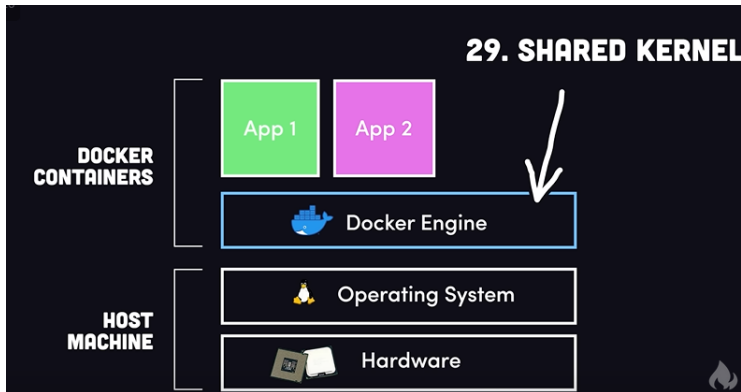
# Was ist Docker?

## Docker

freie Software zur Isolierung von Anwendungen

Containervirtualisierung

"light weight" Virtual Maschine



# Wichtige Begriffe

## Container

Umgebung in der die tatsächliche Anwendung läuft

## Image

Blaupausen, um einen Container zu erstellen

## Dockerfile

Anleitung, um ein Image zu erstellen

# Wichtige Begriffe

## Container

Umgebung in der die tatsächliche Anwendung läuft

## Image

Blaupausen, um einen Container zu erstellen

## Dockerfile

Anleitung, um ein Image zu erstellen

## Registry

z.B. Docker Hub, EAC.... Ort an dem viele verschiedene Images gespeichert und geteilt werden können

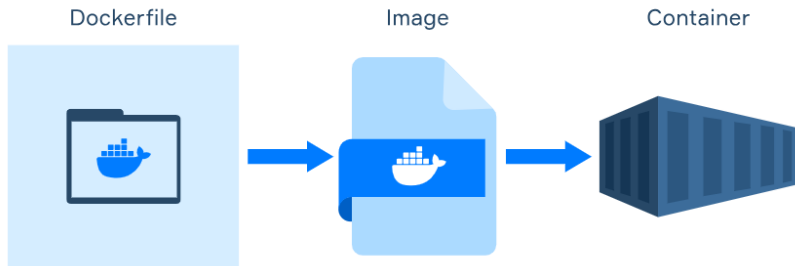
## Docker Compose

Orchestrierungstool für Dockerfile

Wrapper für einen oder mehrere Container

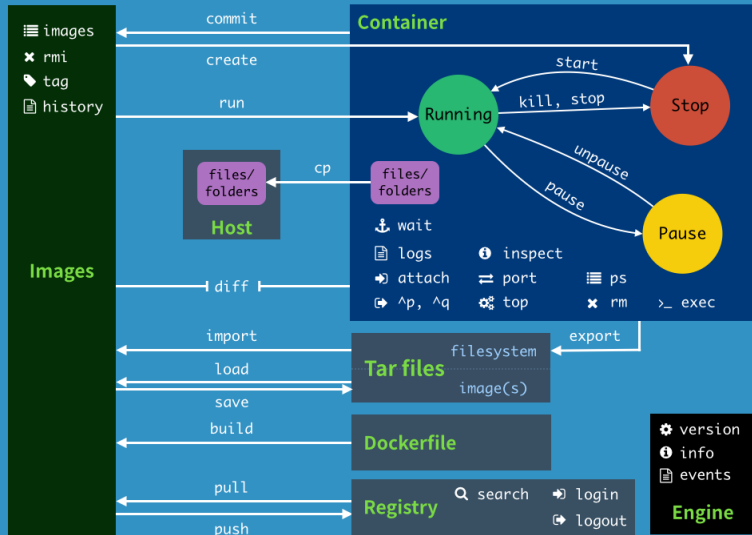


# Zusammenhang der Docker Komponenten



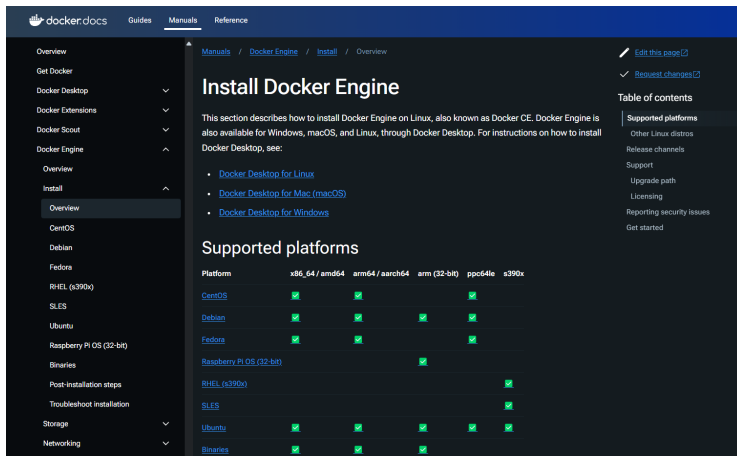
# Zusammenhang der Docker Komponenten

## Docker Commands Diagram



@fntsrlike

# Wie kriege ich dieses "Docker"?



The screenshot shows the Docker documentation website. The left sidebar contains a navigation menu with categories like Overview, Get Docker, Docker Desktop, Docker Extensions, Docker Scout, Docker Engine, Overview, Install, CentOS, Debian, Fedora, RHEL (s390x), SLES, Ubuntu, Raspberry Pi OS (32-bit), Binaries, Post-installation steps, Troubleshoot installation, Storage, and Networking. The main content area is titled 'Install Docker Engine' and includes a table of contents on the right. The table lists supported platforms and their compatibility with various architectures.

## Install Docker Engine

This section describes how to install Docker Engine on Linux, also known as Docker CE. Docker Engine is also available for Windows, macOS, and Linux, through Docker Desktop. For instructions on how to install Docker Desktop, see:

- [Docker Desktop for Linux](#)
- [Docker Desktop for Mac \(macOS\)](#)
- [Docker Desktop for Windows](#)

### Supported platforms

Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	ppc64le	s390x
<a href="#">CentOS</a>	✓	✓		✓	
<a href="#">Debian</a>	✓	✓	✓	✓	
<a href="#">Fedora</a>	✓	✓		✓	
<a href="#">Raspberry Pi OS (32-bit)</a>			✓		
<a href="#">RHEL (s390x)</a>					✓
<a href="#">SLES</a>					✓
<a href="#">Ubuntu</a>	✓	✓	✓	✓	✓
<a href="#">Binaries</a>	✓	✓	✓		

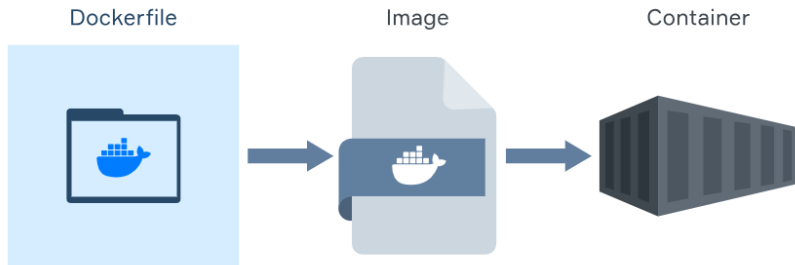
Doku

# Hello World

```
> docker -v  
> docker --help  
> docker run hello-world
```



# Zusammenhang der Docker Komponenten



# Dockerfile

- Anleitung um ein Image zu erstellen
- heißt standardmäßig 'Dockerfile'

ein beispielhaftes Dockerfile :

---

```
FROM alpine:latest
```

```
CMD [ "echo", "Hello World" ]
```

---

# Dockerfile

- Anleitung um ein Image zu erstellen
- heißt standardmäßig 'Dockerfile'

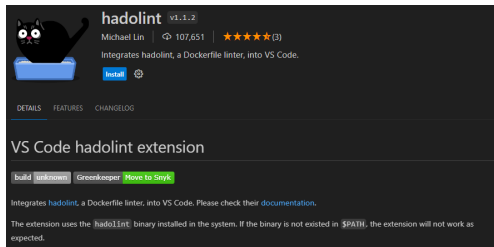
ein beispielhaftes Dockerfile :

---

```
FROM alpine:latest
```

```
CMD [ "echo", "Hello World" ]
```

---



Plugin für die Arbeit mit Docker



- Anleitung um ein Image zu erstellen
- heißt standardmäßig 'Dockerfile'

ein beispielhaftes Dockerfile :

---

```
FROM alpine:latest
```

```
CMD [ "echo", "Hello World" ]
```

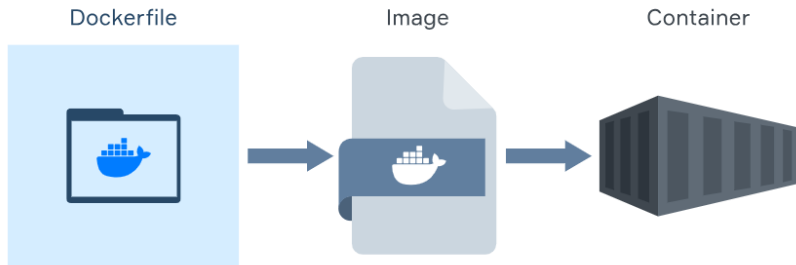
---

Weitere Informationen und Instruction

<https://docs.docker.com/reference/dockerfile/>



# Zusammenhang der Docker Komponenten



# docker build Befehl

```
docker build [OPTIONS] PATH | URL | -
```

Erstelle ein Image aus einem Dockerfile

[OPTIONS]

**-t, --tag *stringArray*** Name und optionaler Tag für das Image  
(format: "name:tag")

**-f, --file *string*** Name des Dockerfile

...

**PATH** Pfad zum Build Kontext (Ordner), meistens **.**

Beispiele

```
docker build . # 'Dockerfile' im aktuellen Ordner
```

```
docker build -t myimage:v1 .
```

```
docker build -f Docker.cmd .
```

```
docker build FastAPI
```

weitere Optionen mit `docker buildx build`



---

```
FROM ubuntu:22.04
```

```
LABEL author=HyperUser
```

```
RUN apt-get update -y \  
    && apt-get upgrade -y \  
    && apt-get install iputils-ping -y \  
    && apt-get install net-tools -y
```

```
ENTRYPOINT ["/bin/bash"]
```

---

---

```
FROM ubuntu:22.04
```

```
LABEL author=HyperUser
```

```
RUN apt-get update -y  
RUN apt-get upgrade -y  
RUN apt-get install iputils-ping -y  
RUN apt-get install net-tools -y
```

```
ENTRYPOINT ["/bin/bash"]
```

---

```
> docker build -t example:single -f Dockerfile.single .
```

```
> docker build -t example:multi -f Dockerfile.multi .
```

*# Vergleicht die Build-time*

*# Vergleicht die Größe - Wie?*

```
> docker images # Entstandene Images anschauen
```

# RUN

---

FROM ubuntu:22.04

LABEL author=HyperUser

RUN apt-get update -y \  
 && apt-get upgrade -y \  
 && apt-get install iputils-ping -y \  
 && apt-get install net-tools -y

ENTRYPOINT ["/bin/bash"]

---

---

FROM ubuntu:22.04

LABEL author=HyperUser

RUN apt-get update -y  
RUN apt-get upgrade -y  
RUN apt-get install iputils-ping -y  
RUN apt-get install net-tools -y

ENTRYPOINT ["/bin/bash"]

---

- pro `RUN` baut Docker einen Layer

# RUN

---

FROM ubuntu:22.04

LABEL author=HyperUser

RUN apt-get update -y \  
 && apt-get upgrade -y \  
 && apt-get install iputils-ping -y \  
 && apt-get install net-tools -y

ENTRYPOINT ["/bin/bash"]

---

---

FROM ubuntu:22.04

LABEL author=HyperUser

RUN apt-get update -y  
RUN apt-get upgrade -y  
RUN apt-get install iputils-ping -y  
RUN apt-get install net-tools -y

ENTRYPOINT ["/bin/bash"]

---

- pro **RUN** baut Docker einen Layer
- mehr Layer vergrößern das Image
- Layer werden gecached und nach Möglichkeit wiederverwendet

# RUN

---

```
FROM ubuntu:22.04
```

```
LABEL author=HyperUser
```

```
RUN apt-get update -y \  
    && apt-get upgrade -y \  
    && apt-get install iputils-ping -y \  
    && apt-get install net-tools -y
```

```
ENTRYPOINT ["/bin/bash"]
```

---

---

```
FROM ubuntu:22.04
```

```
LABEL author=HyperUser
```

```
RUN apt-get update -y  
RUN apt-get upgrade -y  
RUN apt-get install iputils-ping -y  
RUN apt-get install net-tools -y
```

```
ENTRYPOINT ["/bin/bash"]
```

---

- pro **RUN** baut Docker einen Layer
- mehr Layer vergrößern das Image
- Layer werden gecached und nach Möglichkeit wiederverwendet
- verbinden von **RUN** instructions verbessert built time und Image Größe

# CMD vs. ENTRYPOINT



---

```
FROM alpine
```

```
# Exec form
```

```
CMD ["echo", "Hello World."]
```

```
#shell form
```

```
CMD echo Hello Students
```

---

---

```
FROM alpine
```

```
# ENTRYPOINT ["echo"]
```

```
# CMD ["Hello", "Students."]
```

```
ENTRYPOINT ["echo", "Hello World"]
```

---

```
> docker build -t example:cmd -f Dockerfile.cmd .
```

```
> docker build -t example:entry -f Dockerfile.entry .
```

# CMD vs. ENTRYPOINT



---

```
FROM alpine
```

```
# Exec form
```

```
CMD ["echo", "Hello World."]
```

```
#shell form
```

```
CMD echo Hello Students
```

---

---

```
FROM alpine
```

```
# ENTRYPOINT ["echo"]
```

```
# CMD ["Hello", "Students."]
```

```
ENTRYPOINT ["echo", "Hello World"]
```

---

```
> docker build -t example:cmd -f Dockerfile.cmd .
```

```
> docker build -t example:entry -f Dockerfile.entry .
```

```
> docker run example:cmd
```

```
> docker run example:cmd echo hello
```

```
> docker run example:entry hello
```



# CMD vs. ENTRYPOINT

---

```
FROM alpine
```

```
# Exec form
```

```
CMD ["echo", "Hello World."]
```

```
#shell form
```

```
CMD echo Hello Students
```

---

---

```
FROM alpine
```

```
# ENTRYPOINT ["echo"]
```

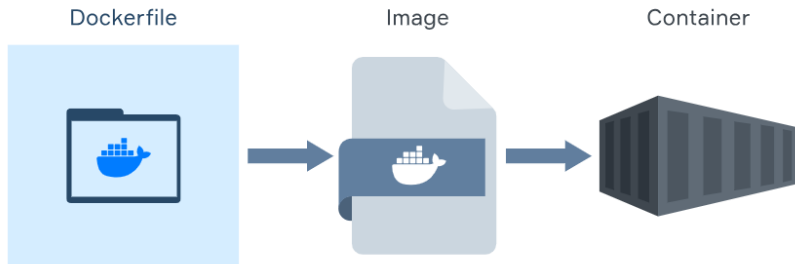
```
# CMD ["Hello", "Students."]
```

```
ENTRYPOINT ["echo", "Hello World"]
```

---

- beide definieren den, was nach Container start ausgeführt wird
- `CMD` kann überschrieben werden
- `ENTRYPOINT` bestimmt den Befehl, neue Parameter werden angehängen

# Zusammenhang der Docker Komponenten



## docker run Befehl

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Erstelle und starte einen Container von einem Image

[OPTIONS]

- d** Container vom Terminal lösen (im Hintergrund laufen lassen)
- it** Interaktives Terminal im Container öffnen
- e** Umgebungsvariablen setzen
- w** Provides an execution directory inside the container
- p [host]:[port]** Port(s) veröffentlichen
- P** Alle Ports veröffentlichen
- rm** Container automatisch nach Beenden entfernen
- mount mount** Dateisystem and den Container mounten
- v, -volume list** Bind mount ein Volume

...

**IMAGE** Referenz zum Image (Tag oder Id/Hash)

# Python / FastAPI

FastAPI 0.1.0 OAS 3.1

/openapi.json

## root

GET	/	Read Root	⌵
-----	---	-----------	---

## songs

GET	/song/get	Get Songs	⌵
-----	-----------	-----------	---

POST	/song/add	Add Song	⌵
------	-----------	----------	---

DELETE	/song/{id}	Delete Song	⌵
--------	------------	-------------	---

## Schemas

HTTPValidationError	>	Expand all	object
Song	>	Expand all	object
ValidationError	>	Expand all	object

# Python FastAPI im Conatiner



---

```
FROM python:3.10.11
```

```
WORKDIR /code
```

```
COPY ./requirements.txt /code/requirements.txt
```

```
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt
```

```
COPY ./app /code/app
```

```
CMD ["uvicorn", "app.api:app", "--host", "0.0.0.0", "--port", "80"]
```

---

# Python FastAPI im Container



```
> cd examples/FastAPI
> docker build -t fastapiapp:v1 .
> docker run -d --name backend -p 8000:80 fastapiapp:v1
# Open http://localhost:8000/docs
> docker exec -it backend bash
# 'exit' um den Container zu verlassen
> docker stop backend
> docker start backend
> docker rm backend
> docker run -it --name backend \
    -p 8000:80 fastapiapp:v1 bash
```

# docker exec Befehl

```
docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
```

Befehl in einem laufenden Container ausführen

[OPTIONS]

- d im Hintergrund ausführen
- e env Variablen setzen
- it Interaktives Terminal öffnen
- w, --workdir string Aktuelles Verzeichnis im Container ändern

...

Beispiele:

```
docker exec -it backend bash # Interaktives Terminal öffnen
```

```
docker exec -d backend touch /code/README.md
```

```
docker exec -e VAR_A=1 -e VAR_B=2 backend env
```

## docker container control Befehl

```
docker create [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Erstelle einen Container ohne ihn zustarten

```
docker start [OPTIONS] CONTAINER [CONTAINER...]
```

Starte einen oder mehrere existierende Container

```
docker stop [OPTIONS] CONTAINER [CONTAINER...]
```

Stoppe einen oder mehrere laufende Container (SIGTERM)

```
docker pause CONTAINER [CONTAINER...]
```

Stoppe alle Prozesse innerhalb eines oder mehrerer Container

```
docker kill [OPTIONS] CONTAINER [CONTAINER...]
```

Stoppe einen oder mehrere laufende Container (SIGKILL)



## docker remove Befehle

```
docker rm [OPTIONS] CONTAINER [CONTAINER...]
```

Entferne einen oder mehrere Container

[OPTIONS]

**-f, -force** Erzwingen das Entfernen

**-v, -volumes** Verbundene anonyme Mounts auch entfernen

```
docker rmi [OPTIONS] IMAGE [IMAGE...]
```

Entferne ein oder mehrere Images

[OPTIONS]

**-f, -force** Erzwingen das Entfernen

**-no-prune** Do not delete untagged parents

# Wo ist mein Song?

Problem reproduzieren:

```
> cd examples/FastAPI
> docker build -t fastapiapp:v1 .
> docker run -d --name backend -p 8000:80 --rm fastapiapp:v1
# Öffne http://localhost:8000/docs + add_song() ausführen
> docker stop backend # Container automatisch gelöscht
> docker run -d --name backend -p 8000:80 --rm fastapiapp:v1
# get_songs() ausführen -> Song fehlt :/
```

# Wo ist mein Song?

Problem reproduzieren:

```
> cd examples/FastAPI
> docker build -t fastapiapp:v1 .
> docker run -d --name backend -p 8000:80 --rm fastapiapp:v1
# Öffne http://localhost:8000/docs + add_song() ausführen
> docker stop backend # Container automatisch gelöscht
> docker run -d --name backend -p 8000:80 --rm fastapiapp:v1
# get_songs() ausführen -> Song fehlt :/
```

- der Song ist im Container gespeichert, nicht im Image
- `--rm` löscht den Container nach Beendigung

Wie bekomme ich den Song permanent gespeichert?

# Python FastAPI im Conatiner

- Option 1: json anpassen, Image neu erstellen

# Python FastAPI im Container



- Option 1: json anpassen, Image neu erstellen
- Option 2: Änderungen commiten

```
> docker commit backend fastapiapp:v2
```

```
> docker run -d --name backend2 --rm \  
    -p 8080:80 fastapiapp:v2
```

*# Öffne localhost:8080/docs -> get\_songs() hat neue Songs*

```
> docker run -d --name backend --rm \  
    -p 8000:80 fastapiapp:v1
```

*# Öffne localhost:8000/docs -> get\_songs() hat keine*

# Python FastAPI im Container



- Option 1: json anpassen, Image neu erstellen
- Option 2: Änderungen commiten
- Option 3: Volumes und Mounts verwenden

```
> docker commit backend fastapiapp:v2
```

```
> docker run -d --name backend2 --rm \  
    -p 8080:80 fastapiapp:v2
```

*# Öffne localhost:8080/docs -> get\_songs() hat neue Songs*

```
> docker run -d --name backend --rm \  
    -p 8000:80 fastapiapp:v1
```

*# Öffne localhost:8000/docs -> get\_songs() hat keine*

## Volumes und Mounts

- Docker containers are stateless by default, data inside is lost after shutdown.
- both map data/storage from the host machine to data/storage in the Container for persistent storage.

# Volumes und Mounts

- Docker containers are stateless by default, data inside is lost after shutdown.
- both map data/storage from the host machine to data/storage in the Container for persistent storage.

## Volume

- are managed by Docker and stored default at `var/lib/docker/volumes/VOLUMENAME`
- don't increase the size of the containers
- simplify and allow sharing data between containers



# Volumes und Mounts

- Docker containers are stateless by default, data inside is lost after shutdown.
- both map data/storage from the host machine to data/storage in the Container for persistent storage.

## Volume

- are managed by Docker and stored default at `var/lib/docker/volumes/VOLUMENAME`
- don't increase the size of the containers
- simplify and allow sharing data between containers

## Mount

- a file or directory on the host machine is attached to the containers filesystem
- dependant on the host machine while volumes are managed by docker

# Python FastAPI im Container mit Volume



- Option 1: json anpassen, Image neu erstellen
- Option 2: changes commiten
- Option 3: Volume, wenn man die json changes behalten möchte, aber den container per se nicht

*# Quiz: why is this not working*

```
> docker run -d --rm --name backend \  
    -v ${PWD}/app/songs.json:/app/songs.json \  
    -p 8000:80 fastapiapp:v1
```

# Python FastAPI im Container mit Volume



- Option 1: json anpassen, Image neu erstellen
- Option 2: changes commiten
- Option 3: Volume, wenn man die json changes behalten möchte, aber den container per se nicht

*# Quiz: why is this not working*

```
> docker run -d --rm --name backend \  
    -v ${PWD}/app/songs.json:/app/songs.json \  
    -p 8000:80 fastapiapp:v1
```

*# Note: be aware of the working directory of your app*

*# in this case code 'WORKDIR /code'*

```
> docker run -d --rm --name backend \  
    -v ${PWD}/app/songs.json:/code/app/songs.json \  
    -p 8000:80 fastapiapp:v1
```

# React

## My Favorit Songs

**Title:** A Change Is Gonna Come | **Artist:** Sam Cook | **Year:** 1964

Delete Song

**Title:** Like a Rolling Stone | **Artist:** Bob Dylan | **Year:** 1965

Delete Song

**Title:** string | **Artist:** string | **Year:** 2020

Delete Song

Add new Song

# React - Dockerfile

---

```
# pull official base image
FROM node:18.16.0-alpine

# set working directory
WORKDIR /app

# add `/app/node_modules/.bin` to $PATH
ENV PATH /app/node_modules/.bin:$PATH

# install app dependencies
COPY package.json ./
COPY package-lock.json ./
RUN npm install --silent
RUN npm install react-scripts@3.4.1 -g --silent

# add app
COPY . ./

# start app
CMD ["npm", "start"]
```

---

# React im Container



```
> cd examples/React
> docker build -t reactapp:dev .
> docker run -it --rm --name frontenddev \
    -v ${PWD}:/app -v /app/node_modules \
    -e CHOKIDAR_USEPOLLING=true \
    -p 3000:3000 reactapp:dev
# Öffne localhost:3000
```

(Der Container `backend` sollte laufen, damit die Webseite richtig funktioniert)

# Multistage builds

Idee: Image aufeinanderbauende Teile teilen, zwischen den Teilen nur die nötigen Dinge kopieren

z.B. Stage 1: App compile, Stage 2: Compilierte App ausführen  
(kein Build context) Vorteile

- Smaller image size
- faster build times
- improved security (only runtime artifacts and dependencies)
- code isolation and reusability
- Easier debugging and troubleshooting

# React - Multistage

---

*# build environment*

FROM node:18.16.0-alpine as build

WORKDIR /app

ENV PATH /app/node\_modules/.bin:\$PATH

COPY package.json ./

COPY package-lock.json ./

RUN npm ci --silent

RUN npm install react-scripts@3.4.1 -g --silent

COPY . ./

RUN npm run build

*# production environment*

FROM nginx:stable-alpine

COPY --from=build /app/build /usr/share/nginx/html

EXPOSE 80

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

---





```
> docker build -f Dockerfile.prod -t reactapp:prod .  
> docker run -it --rm --name frontend \  
    -p 1337:80 reactapp:prod
```

Vergleiche die Größe der Images:



```
> docker build -f Dockerfile.prod -t reactapp:prod .  
> docker run -it --rm --name frontend \  
    -p 1337:80 reactapp:prod
```

Vergleiche die Größe der Images:

frontenddev: 832 MB

frontend: 50.9 MB

# Dockerfile Best Practices

- `RUN` instructions mit `&&` zusammenfassen
- `COPY` sinnvoll platzieren, damit Cache best möglich genutzt werden kann
- `ADD` nur für `ADD` spezifische Funktionen
- Volumes und Mounts für persistenten Speicher nutzen
- Multistage builds verwenden

## Weiteres zu docker

### `.dockerignore`

- vgl. `.gitignore` für Docker
- bestimmte Dateien/Ordern ausschließen
- geringere Image Größe
- kein Cache invalidation

### *# interaktion mit Registries*

> docker push

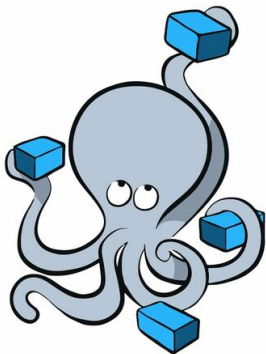
> docker pull

### *# Scan auf Sicherheitslücken*

> docker scout

Docker Desktop bietet eine Benutzeroberfläche für die meisten Befehle

Weitere Befehle am Ende



docker  
Compose

# Docker Compose

## Was?

Tool um Multi-Container Anwendungen einfach auszuführen  
eine Konfiguration, statt in mehreren Terminals alle verschiedene Container laufen zu lassen

## Warum?

- leichter mit mehreren Container zuarbeiten
- gute Portabilität
- schnelle Anwendungsentwicklung

## Wie?

- Definition von Services in `docker-compose.yml`
- ein Service entspricht einem Container

Anmerkung: Python in `examples/FastAPI`, React in `examples/React` und Full App in `examples` ausführen

# Docker Compose zu Python



---

```
version: '3.7'
```

```
services:
```

```
  fastapi:
```

```
    container_name: backend
```

```
    build:
```

```
      context: .
```

```
      dockerfile: Dockerfile
```

```
    # image: fastapiapp:v2
```

```
    ports:
```

```
      - '8000:80'
```

```
    volumes:
```

```
      - ./app/songs.json:/code/app/songs.json
```

---

```
docker compose up
```

**VS.**

# Docker Compose zu Python



```
version: '3.7'
```

```
services:
```

```
  fastapi:
```

```
    container_name: backend
```

```
    build:
```

```
      context: .
```

```
      dockerfile: Dockerfile
```

```
    # image: fastapiapp:v2
```

```
    ports:
```

```
      - '8000:80'
```

```
    volumes:
```

```
      - ./app/songs.json:/code/app/songs.json
```

---

```
docker compose up
```

VS.

```
> docker build -t fastapiapp:v1 .
```

```
> docker run -d --rm --name backend \
  -v ${PWD}/app/songs.json:/code/app/songs.json \
  -p 8000:80 fastapiapp:v1
```



# Docker Compose Webapp

---

```
version: '3.7'
```

```
services:
```

```
  frontend:
```

```
    container_name: frontend
```

```
    build:
```

```
      context: .
```

```
      dockerfile: Dockerfile.prod
```

```
    ports:
```

```
      - '1337:80'
```

---

```
> docker-compose -d -f docker-compose.prod.yml \  
  up
```

VS.

```
> docker run -it --rm -d --name frontend \  
  -p 1337:80 frontend:prod
```

# Docker Compose Full App

---

version: '3.7'

services:

frontend:

container\_name: frontend

build:

context: ./React/

dockerfile: Dockerfile.prod

ports:

- '3000:80'

fastapi:

container\_name: backend

build:

context: ./FastAPI/

dockerfile: Dockerfile

ports:

- '8000:80'

volumes:

- ./FastAPI/app/songs.json:/code/app/songs.json

---

> docker-compose up

## docker-compose up Befehl

```
docker-compose up [OPTIONS] [SERVICE...]
```

Erstellen, (neu)erstellen, starten und an Container eines Dienstes anhängen

[OPTIONS]

- d, **-detach** Führe die Container im Hintergrund aus
- build** Baue die Images, bevor die Container gestartet werden
- no-build** Baue kein Image, selbst wenn es fehlt
- force-recreate** Erstelle Container neu, auch wenn deren Konfiguration und Image sich nicht geändert haben
- no-recreate** Erstelle Container nicht neu, wenn sie bereits existieren
- remove-orphans** Entferne Container von Diensten, die nicht in der Compose-Datei definiert sind
- V**, **-renew-anon-volumes** Erstelle anonyme Volumes neu, anstatt Daten von vorherigen Containern zu übernehmen

# docker-compose down Befehl

```
docker-compose down [OPTIONS]
```

Stoppt und entfernt Container, Netzwerke, Images und Volumes

```
[OPTIONS]
```

- rmi type** Entfernt Images. Der Typ muss einer der folgenden sein:
  - all** Entfernt alle Images, die von einem Dienst verwendet werden
  - local** Entfernt nur Images, die kein benutzerdefiniertes Tag haben
- v, -volumes** Entfernt benannte Volumes, die im 'volumes' Abschnitt der Compose-Datei deklariert sind
- remove-orphans** Entfernt Container für Dienste, die nicht in der Compose-Datei definiert sind

## Weiter Docker Compose Befehle

```
docker-compose build [OPTIONS] [SERVICE...]
```

Build oder rebuild Services

```
docker-compose start [SERVICE...]
```

Starte existierende Containers

```
docker-compose stop [OPTIONS] [SERVICE...]
```

Stoppe laufende Containers ohne sie zu entfernen

```
docker-compose rm [OPTIONS] [SERVICE...]
```

Entferne gestoppte Container

```
docker-compose exec [OPTIONS] SERVICE COMMAND [ARGS...]
```

Führe einen Befehl in einem Container aus

...

Dockerfile



Image



Container



## Andere UseCases - Docker

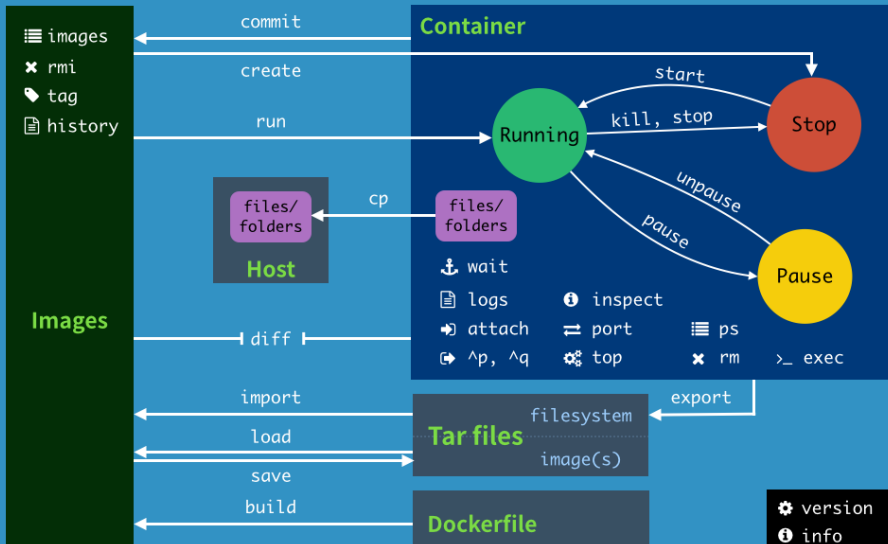
- OpenDrone Map
- Datenbanken in Containern
- Kubernetes Cluster
- Unternehmen, die wohl Docker nutzen: Airbnb, Spotify, PayPal, Uber, Netflix
- UseCases
- deploy your app on a cloud hosted frame work

## Andere UseCases - Docker Compose

- OpenDrone Map
- Nathalies Kubernetes Arbeit
- Felix Hiwi arbeit
- deploy your app on a cloud hosted frame work



# Docker Commands Diagram



## docker commit Befehl

```
docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]
```

Erstelle ein neues Image aus dem aktuellen Containerzustand  
[OPTIONS]

- c, **-change list** Verwende Dockerfile instruction zum erstellen
- m, **-message string** Commit Nachricht
- p, **-pause** Pausiere den Container während des commits  
(default true)

```
docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]
```

Erstelle einen neuen Tag für ein Image

```
docker image tag 9c62f3337754 ubuntu:v3
```

## docker cp Befehl und mehr

```
docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH|-
```

Kopiere Dateien zwischen Container und lokalem Speicher

[OPTIONS]

CONTAINER:SRC\_PATH DEST\_PATH|- Kopiere von Container zu lokal

SRC\_PATH|- CONTAINER:DEST\_PATH Kopiere von lokal zu Container

- Nutze tar archive

```
docker import [OPTIONS] file | URL | - [REPOSITORY[:TAG]]
```

Erstelle ein Image aus einer tar-Datei

```
docker export [OPTIONS] CONTAINER
```

Dateisystem eines Containers als tar speichern

## docker images Befehl

```
docker images [OPTIONS] [REPOSITORY[:TAG]]
```

Auflistung von Images

[OPTIONS]

- a, -all Zeige alle Images (default: intermediate Images versteckt)
- f, -filter filter Filter das Ergebnis
- format string Ausgabe formatieren

```
docker ps [OPTIONS]
```

Auflistung von laufenden Containern

[OPTIONS]

- a, -all Zeige alle Container (auch gestoppte)
- f, -filter filter Filter das Ergebnis
- format string Ausgabe formatieren
- n, -last int Zeige nur die letzten n Container
- s, -size Zeige die Größe an

## Cooler Quellen und so weiter

- <https://www.docker.com/>
- Offizielle Dokumentation:  
<https://docs.docker.com/get-started/>
- Wie man Docker auf Raspberry Pi einrichtet

## Weitere Themen für Lightning Talks und Full Talks

- Docker Networks
- Docker Compose (ausführlicher)
- Kubernetes