

06 Docker

Multimedia Engineering (PS siehe 36609b)

M.Sc. Nils Hellwig

Lehrstuhl für Medieninformatik

FAKULTÄT FÜR INFORMATIK UND DATA SCIENCE



Universität Regensburg

Überblick Themen

- Was ist Docker?
- Docker Images / Container
- Dockerfile
- docker-compose

Was ist Docker?

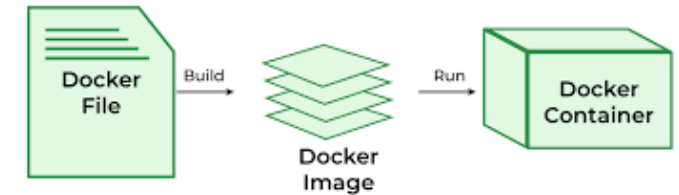
- Ein **Docker-Container** ist eine isolierte Laufzeitumgebung, die eine Anwendung samt aller benötigten Abhängigkeiten, Bibliotheken und Konfigurationen bündelt, sodass sie zuverlässig auf jedem System ausgeführt werden kann.
- **Plattformübergreifende Ausführung:** Ein Container läuft überall, wo Docker installiert ist: lokal, in der Cloud oder auf Servern.
- **Portabilität:** „Write once, run anywhere.“
- **Isolierung:** Anwendungen und ihre Abhängigkeiten laufen unabhängig.

Was ist Docker?

- **Auf Linux-Systemen** nutzen Docker-Container direkt den vorhandenen Linux-Kernel, ohne dass dieser virtualisiert werden muss.
- Auf **macOS oder Windows** startet Docker im Hintergrund eine kleine virtuelle Linux-Maschine, damit Container dort trotzdem funktionieren.

Ein **Kernel** ist der **Kern eines Betriebssystems** – also das zentrale Programm, das direkt mit der Hardware kommuniziert. Er sorgt dafür, dass Programme z. B. auf den Speicher, Prozessor, Festplatten oder Netzwerke zugreifen können, ohne sich gegenseitig zu stören. Der Kernel regelt also, **wer wann was darf**, und hält das ganze System stabil und sicher.“

Begriffe



- **Docker Image:** Vorlage für einen Container: Beschreibt, was in einem Container enthalten sein soll (z.B. Betriebssystem, Software).
- **Docker Container:** Ein Container ist eine **laufende Instanz** eines Images.
- **Docker Hub:** Große Sammlung vorgefertigter Docker-Images (siehe <https://hub.docker.com/>)

Beispiele: ubuntu, mongo, mysql.

Die (öffentliche) Bereitstellung eigener Images beim Docker Hub ist ebenfalls möglich.

Docker Hub





← → ↺ hub.docker.com ☆

Zum Aktualisieren neu starten ⋮





WhatsApp ChatGPT Notion Homesever Google Scholar Word Studium Bachelor Google Wetter >> Alle Lesezeichen

Frameworks
Integration & Delivery
Internet of Things
Machine Learning & AI
Message Queues
Monitoring &
Networking
Operating Systems
Security
Web Servers
Developer Tools
Web Analytics





Machine Learning & AI [View all](#)

-  **tensorflow/tens...**
the machine learning...
☆2.6K ↓50M+
-  **pytorch/pyto**
framework that puts...
☆1.2K ↓10M+
-  **langchain/lan...**
tions
with LLMs through...
☆205 ↓50K+
-  **ollama/ollama**
The easiest way to get up and running with large...
☆862 ↓5M+

Trending this week ↗

-  **homeassistant/...**
☆141 ↓5M+
-  **paketobuildp...**
☆43 ↓50M+
-  **vitess/lite**
A slimmed down version of Vitess containers, with jus...
☆45 ↓10M+
-  **friendica**
Welcome to the free social web.
☆103 ↓5M+

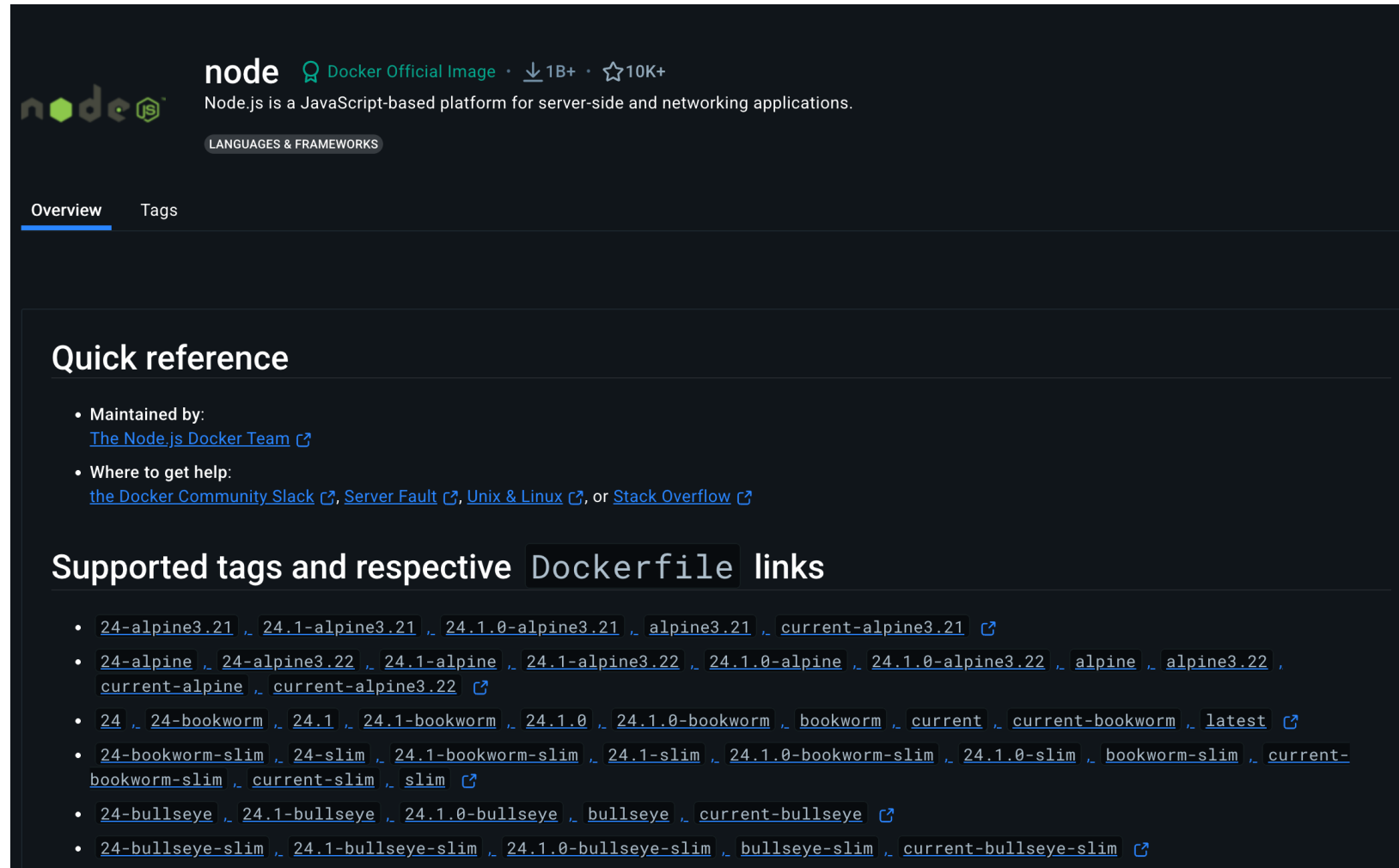
Most pulled images [View all](#)

-  **memcached**
Free & open source, high-performance, distributed...
☆2.3K ↓1B+
-  **nginx**
Official build of Nginx.
☆10K+ ↓1B+
-  **busybox**
Busybox base image.
☆3.3K ↓1B+
-  **alpine**
A minimal Docker image based on Alpine Linux wit...
☆10K+ ↓1B+

Docker Image Tags

Docker Image Tags:

Spezifische Versionen eines Images.



The screenshot shows the Docker Hub page for the 'node' image. At the top, it says 'node' with the Docker Official Image logo, 1B+ downloads, and 10K+ stars. Below this, it states 'Node.js is a JavaScript-based platform for server-side and networking applications.' and 'LANGUAGES & FRAMEWORKS'. There are tabs for 'Overview' and 'Tags'. The 'Quick reference' section lists links for 'Maintained by' (The Node.js Docker Team) and 'Where to get help' (the Docker Community Slack, Server Fault, Unix & Linux, or Stack Overflow). The 'Supported tags and respective Dockerfile links' section lists various tags for different base images (alpine, bookworm, bullseye) and their corresponding Dockerfile links.

node Docker Official Image · ↓ 1B+ · ☆ 10K+

Node.js is a JavaScript-based platform for server-side and networking applications.

LANGUAGES & FRAMEWORKS

Overview Tags

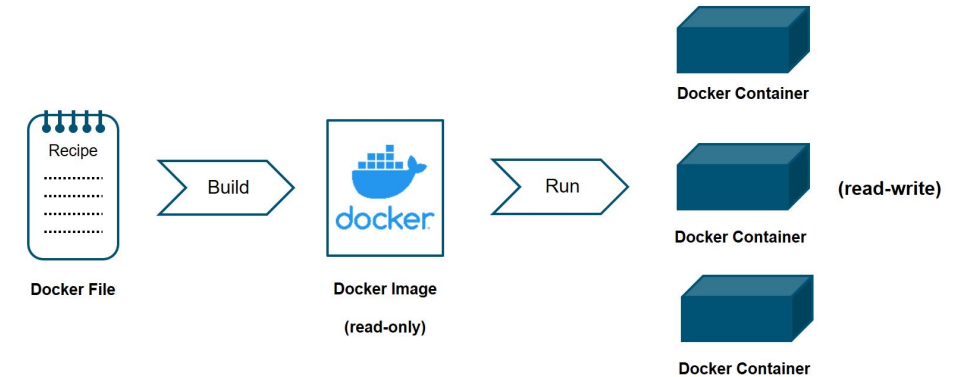
Quick reference

- Maintained by:
[The Node.js Docker Team](#)
- Where to get help:
[the Docker Community Slack](#), [Server Fault](#), [Unix & Linux](#), or [Stack Overflow](#)

Supported tags and respective Dockerfile links

- [24-alpine3.21](#) , [24.1-alpine3.21](#) , [24.1.0-alpine3.21](#) , [alpine3.21](#) , [current-alpine3.21](#)
- [24-alpine](#) , [24-alpine3.22](#) , [24.1-alpine](#) , [24.1-alpine3.22](#) , [24.1.0-alpine](#) , [24.1.0-alpine3.22](#) , [alpine](#) , [alpine3.22](#) , [current-alpine](#) , [current-alpine3.22](#)
- [24](#) , [24-bookworm](#) , [24.1](#) , [24.1-bookworm](#) , [24.1.0](#) , [24.1.0-bookworm](#) , [bookworm](#) , [current](#) , [current-bookworm](#) , [latest](#)
- [24-bookworm-slim](#) , [24-slim](#) , [24.1-bookworm-slim](#) , [24.1-slim](#) , [24.1.0-bookworm-slim](#) , [24.1.0-slim](#) , [bookworm-slim](#) , [current-bookworm-slim](#) , [current-slim](#) , [slim](#)
- [24-bullseye](#) , [24.1-bullseye](#) , [24.1.0-bullseye](#) , [bullseye](#) , [current-bullseye](#)
- [24-bullseye-slim](#) , [24.1-bullseye-slim](#) , [24.1.0-bullseye-slim](#) , [bullseye-slim](#) , [current-bullseye-slim](#)

Begriffe



- **Dockerfile:** Ein Dockerfile ist eine Textdatei, in der man definiert, wie ein Docker-Image aufgebaut wird. Es ermöglicht einem, ein individuelles Image zu erstellen (z.B. mit eigener Software).
- **Docker Desktop** muss gestartet werden, bevor Docker-Befehle ausgeführt werden können, wenn man auf einem **Windows-** oder **Mac-System** arbeitet. Docker Desktop ist eine Anwendung, die die **Docker Engine** und die notwendigen Komponenten wie den **Docker CLI** startet.

Zugriff auf Docker

- Der Zugriff auf Docker läuft über die Kommandozeile (docker).

```
nils_hellwig@MacBook-Pro-von-Nils-2 ~ % docker --version  
Docker version 28.1.1, build 4eba377
```

```
nils_hellwig@MacBook-Pro-von-Nils-2 ~ % docker ps  
Cannot connect to the Docker daemon at  
unix:///Users/nils_hellwig/.docker/run/docker.sock.  
Is the docker daemon running?
```

Mac/Windows: Docker Desktop starten!

Docker Command Line Interface

- `-d`: Der Container läuft im **Hintergrund** (detached mode).
- `-p 8080:80`: Ordnet einen Port des Hosts (8080) einem Port im Container zu (80). Diese Zuordnung sorgt dafür, dass du von außen auf die Anwendung im Container zugreifen kannst.
- `--name <container_name>`: Gibt dem Container einen **Namen**, damit du ihn leichter identifizieren kannst.

```
# 2. Container aus einem Image starten
```

```
docker run -d -p 8080:80 --name <container_name> <image_name>
```

Docker Command Line Interface

1. Docker Image aus dem Docker Hub ziehen

docker pull <image_name>

2. Container aus einem Image starten (im Hintergrund, Port 8080 auf 80 gemappt, mit Namen)

docker run -d -p 8080:80 --name <container_name> <image_name>

3. Alle Container anzeigen (-a: auch gestoppte)

docker ps -a

4. Einen Container stoppen

docker stop <container_name>

5. Einen gestoppten Container starten

docker start <container_name>

6. Einen Container löschen

docker rm <container_name>

7. Docker-Images anzeigen

docker images

8. Docker-Image löschen

docker rmi <image_name>

Dockerfile

- **Dockerfile:** Textdatei, die Aufbau eines Docker-Images beschreibt.
- Die **Dockerfile** sollte im Verzeichnis eines Projekts liegen, damit auf alle relevanten Dateien und Ordner innerhalb des Projekts zugegriffen werden kann.

Dockerfile

```
# Basis-Image
FROM node:16

# Arbeitsverzeichnis setzen
WORKDIR /app

# Kopiere die App-Dateien
COPY package*.json ./
RUN npm install

# Kopiere den Rest der App
COPY . .

# Exponiere Port 3000
EXPOSE 3000

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

Dockerfile

FROM: Legt das Basis-Image fest, auf dem das neue Image aufbauen soll.

WORKDIR: Setzt das Arbeitsverzeichnis für alle nachfolgenden Befehle im Container (wird ggf. neu erstellt).

COPY: Kopiert Dateien oder Verzeichnisse vom Host in das Image.

RUN: Führt einen Befehl im Container zur Build-Zeit aus (z. B. Installationen).

CMD: Gibt den Standardbefehl an, der beim Starten des Containers ausgeführt wird.

ENV: Setzt Umgebungsvariablen im Container.

Dockerfile

```
# Basis-Image
FROM node:16

# Arbeitsverzeichnis setzen
WORKDIR /app

# Kopiere die App-Dateien
COPY package*.json .
RUN npm install

# Kopiere den Rest der App
COPY . .

# Exponiere Port 3000
EXPOSE 3000

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

Dockerfile: Schichtweiser Aufbau

- **Neue Schichten durch Anweisungen:** Jede Anweisung im Dockerfile (z. B. RUN, COPY, ADD) erstellt eine neue Schicht.
- Jede Schicht enthält **nur die Änderungen** der jeweiligen Anweisung.
- Wenn eine Schicht gleich geblieben ist, benutzt Docker die gespeicherte Version (Cache).
- Erst wenn sich eine Schicht ändert, werden **alle folgenden Schichten** neu gebaut.

Dockerfile

```
# Basis-Image
FROM node:16

# Arbeitsverzeichnis setzen
WORKDIR /app

# Kopiere die App-Dateien
COPY package*.json .
RUN npm install

# Kopiere den Rest der App
COPY . .

# Exponiere Port 3000
EXPOSE 3000

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

Dockerfile

Schicht 1: Basis-Image mit Node.js

FROM node:18

Schicht 2: Arbeitsverzeichnis im Container
festlegen/erstellen

WORKDIR /app

Schicht 3: package.json und package-lock.json kopieren

COPY package.json ./

Schicht 4: npm install ausführen (abhängigkeiten
installieren)

RUN npm install

Schicht 5: Restlichen Quellcode kopieren

COPY . .

Schicht 6: Startbefehl

CMD ["node", "index.js"]

- Führe ich die Dockerfile nicht das erste Mal aus wurden alle Schichten gecached.
- Docker prüft, ob sich package.json geändert hat.
- Wenn nein, wird diese Schicht aus dem Cache genommen.
- In diesem Fall können alle Schichten bis einschließlich der 4. Schicht aus dem Cache entnommen werden.

Dockerfile

+-----+	
Node.js Runtime + App-Code	← Dein Node.js Image
+-----+	
Node.js Official Image	← Enthält Node.js & npm
+-----+	
Linux Basis-Image (z.B.	
Debian, Alpine oder Ubuntu)	← Minimal-OS-Image
+-----+	

Häufig verwendete Linux Distributionen bei Docker Images

	Alpine	Debian Slim	Ubuntu	BusyBox	Distroless
Größe (Image)	Sehr klein (~5 MB)	Klein (~22 MB)	Groß (~29–60 MB+)	Extrem klein (~1 MB)	Sehr klein (~10–20 MB)
Sicherheit	Gut, kleiner Angriffsvektor	Gut, regelmäßige Updates	Gut, aber größere Angriffsfläche	Minimal, aber limitiert	Sehr gut, minimale Angriffsfläche
Kompatibilität	Eingeschränkt (musl libc)	Hoch (glibc)	Sehr hoch (glibc, weit verbreitet)	Sehr eingeschränkt	Eingeschränkt (kein Shell)
Performance	Hoch, wegen Leichtgewicht	Durchschnittlich	Durchschnittlich	Hoch	Hoch
Paketmanager	apk	apt	apt	rudimentär	keiner
Benutzerfreundlichkeit	Eingeschränkt (minimal Tools)	Gut	Sehr gut	Schlecht	Sehr eingeschränkt
Geeignet für	Microservices, kleine Images	Allgemeiner Gebrauch, kleiner als Ubuntu	Entwicklung, Kompatibilität	Tiny Container Tests	Produktions-Deployments
libc-Typ	musl	glibc	glibc	uClibc/musl	keine/libc statisch gelinkt
Shell verfügbar?	Ja (sh)	Ja (sh, bash)	Ja (bash)	Ja (sh)	Nein
Verbreitung in Docker Hub	Sehr hoch	Hoch	Sehr hoch	Mittel	Steigend

Dockerfile - Debugging

- Jeder Container besitzt ein eigenes Linux-Dateisystem mit Root-Verzeichnis / und typischen Ordnern wie /bin (Systemprogramme) und /usr (Zusätzliche Bibliotheken, die vom User installiert wurden).
- Die Container-Shell (z. B. /bin/sh oder /bin/bash) ermöglicht eine Kommandozeile, ist aber nur in Images mit installierter Shell vorhanden.
- Mit `docker run -it <image> /bin/sh` oder `docker exec -it <container> /bin/bash` kann in die Kommandozeile eines Containers gewechselt werden.

Dockerfile – Building

- Um auf Basis einer Dockerfile ein Image zu erstellen, kann folgender Befehl benutzt werden:
`docker build -t dein-image-name:tag .`
- `docker build` startet den Build-Prozess und liest das Dockerfile im aktuellen Verzeichnis (deshalb der Punkt `.` am Ende).
- `-t dein-image-name:tag` gibt dem Image einen Namen und optional einen Tag (z.B. `myapp:1.0`).
- Nach dem Build kann man das Image im Terminal mit `docker images` sehen und mit `docker run` gestartet werden.

Dockerfile – Kleine Aufgabe (10-15 Minuten)

Erstelle eine kleine Express-Anwendung, die auf dem Pfad /hello-world den Text „Hello from Express on /hello-world!“ ausgibt. Die App soll in einem Docker-Container laufen und von außen über Port 8000 erreichbar sein.

Tipps:

- Starte zunächst Docker Desktop
- Starte das Terminal in dem Verzeichnis mit der Node Anwendung (Auf GRIPS bzw. GitHub verfügbar)
- Erstelle im Verzeichnis eine „Dockerfile“-Datei (keine Endung (nicht .txt o.Ä.))
- Verwende das offizielle Node.js-Image node:18 als Basis.
- Erstelle eine Datei index.js mit einer Express-App, die auf Port 8080 läuft und beim Zugriff auf /hello-world den Text ausgibt.
- Erstelle eine package.json mit der Express-Abhängigkeit.
- Schreibe ein Dockerfile, das: (1) Das Node-Image verwendet, (2) alle Abhängigkeiten installiert (npm install).
- Den Quellcode in den Container kopiert.
- Die App mit node index.js startet.
- Baue das Docker-Image und starte den Container mit Port-Mapping von 8000 (Host) zu 8080 (Container).
- Teste die Anwendung über <http://localhost:8000/hello-world>.

docker-compose

- Docker Compose ist ein Tool, für **Multi-Container-Docker-Anwendungen**.
- Mit **Docker Compose** kannst du mehrere Container konfigurieren und starten, indem du eine YAML-Datei namens docker-compose.yml erstellst.
- **Vorteil:** Alle benötigten Container (z.B. Datenbank, Frontend, Backend) können mit einem Befehl (`docker-compose up`) gestartet werden.

Einführung in Docker – docker-compose Beispiel

- **Ziel:** Wir erstellen eine Anwendung, bei der ein Node.js-Backend und ein einfacher HTTP-Server für das Frontend zusammenarbeiten.
- Unser Repository:

```
perl
my-app/
├── backend/
│   ├── Dockerfile
│   └── server.js
├── frontend/
│   └── index.html
├── docker-compose.yml
└── README.md
```

Einführung in Docker – docker-compose Beispiel

```
yaml

version: '3'
services:
  backend:
    build: ./backend
    ports:
      - "3000:3000" # Host-Port 3000 auf Container-Port 3000

  frontend:
    image: httpd:latest # Nutzt den Apache HTTP Server
    volumes:
      - ./frontend:/usr/local/apache2/htdocs/ # Bindet das Frontend-Verzeichnis
    ports:
      - "8080:80" # Host-Port 8080 auf Container-Port 80
```

Zum Starten kann dann in der Datei mit der `docker-compose.yaml` das Kommando `docker-compose up` ausgeführt werden.

Einführung in Docker – docker-compose Beispiel

```
yaml

version: '3'
services:
  backend:
    build: ./backend
    ports:
      - "3000:3000" # Host-Port 3000 auf Container-Port 3000

  frontend:
    image: httpd:latest # Nutzt den Apache HTTP Server
    volumes:
      - ./frontend:/usr/local/apache2/htdocs/ # Bindet das Frontend-Verzeichnis
    ports:
      - "8080:80" # Host-Port 8080 auf Container-Port 80
```

1. version

- Gibt die Version des compose-Formats an (z. B. "3")
- Bestimmt, welche Features verfügbar sind
- Wichtig für Kompatibilität mit Docker-Versionen

Zum Starten kann dann in der Datei mit der `docker-compose.yaml` das Kommando `docker-compose up` ausgeführt werden.

Einführung in Docker – docker-compose Beispiel

yaml

```
version: '3'
services:
  backend:
    build: ./backend
    ports:
      - "3000:3000" # Host-Port 3000 auf Container-Port 3000

  frontend:
    image: httpd:latest # Nutzt den Apache HTTP Server
    volumes:
      - ./frontend:/usr/local/apache2/htdocs/ # Bindet das Frontend-Verzeichnis
    ports:
      - "8080:80" # Host-Port 8080 auf Container-Port 80
```

2. services

- Definition der einzelnen Container, die zusammen eine Anwendung bilden
- Jeder Service entspricht einem Container
- Beliebige Benennung, Beispiele: "backend", "frontend", "database"

Zum Starten kann dann in der Datei mit der `docker-compose.yaml` das Kommando `docker-compose up` ausgeführt werden.

Einführung in Docker – docker-compose Beispiel

```
yaml

version: '3'
services:
  backend:
    build: ./backend
    ports:
      - "3000:3000" # Host-Port 3000 auf Container-Port 3000

  frontend:
    image: httpd:latest # Nutzt den Apache HTTP Server
    volumes:
      - ./frontend:/usr/local/apache2/htdocs/ # Bindet das Frontend-Verzeichnis
    ports:
      - "8080:80" # Host-Port 8080 auf Container-Port 80
```

3. image

- Gibt das Docker-Image an, das verwendet wird
- Kann ein öffentliches Image aus Docker Hub sein (z.B. httpd:latest)

Zum Starten kann dann in der Datei mit der `docker-compose.yaml` das Kommando `docker-compose up` ausgeführt werden.

Einführung in Docker – docker-compose Beispiel

```
yaml

version: '3'
services:
  backend:
    build: ./backend
    ports:
      - "3000:3000" # Host-Port 3000 auf Container-Port 3000

  frontend:
    image: httpd:latest # Nutzt den Apache HTTP Server
    volumes:
      - ./frontend:/usr/local/apache2/htdocs/ # Bindet das Frontend-Verzeichnis
    ports:
      - "8080:80" # Host-Port 8080 auf Container-Port 80
```

4. build

- Alternative zu image
- Pfad zu einem Dockerfile oder Verzeichnis, aus dem ein Image gebaut wird
- Beispiel: build: ./backend
Im Verzeichnis ./backend muss dann eine Dockerfile liegen

Zum Starten kann dann in der Datei mit der `docker-compose.yaml` das Kommando `docker-compose up` ausgeführt werden.

Einführung in Docker – docker-compose Beispiel

```
yaml

version: '3'
services:
  backend:
    build: ./backend
    ports:
      - "3000:3000" # Host-Port 3000 auf Container-Port 3000

  frontend:
    image: httpd:latest # Nutzt den Apache HTTP Server
    volumes:
      - ./frontend:/usr/local/apache2/htdocs/ # Bindet das Frontend-Verzeichnis
    ports:
      - "8080:80" # Host-Port 8080 auf Container-Port 80
```

5. ports

- Verbindet Ports des Containers mit Ports des Hosts
- Format: `hostPort:containerPort` (z. B. 8080:80)
- Erlaubt externen Zugriff auf Container-Dienste

Zum Starten kann dann in der Datei mit der `docker-compose.yaml` das Kommando `docker-compose up` ausgeführt werden.

Einführung in Docker – docker-compose Beispiel

```
yaml

version: '3'
services:
  backend:
    build: ./backend
    ports:
      - "3000:3000" # Host-Port 3000 auf Container-Port 3000

  frontend:
    image: httpd:latest # Nutzt den Apache HTTP Server
    volumes:
      - ./frontend:/usr/local/apache2/htdocs/ # Bindet das Frontend-Verzeichnis
    ports:
      - "8080:80" # Host-Port 8080 auf Container-Port 80
```

6. volumes

- Bindet Verzeichnisse oder Dateien vom Host ins Container-Dateisystem ein
- Für Datenpersistenz oder Entwicklung (z. B. Quellcode)
- Format: hostPath:containerPath

Zum Starten kann dann in der Datei mit der `docker-compose.yaml` das Kommando `docker-compose up` ausgeführt werden.

Weitere Ressourcen (Leseempfehlung!)

- Docker compose: <https://docs.docker.com/compose/gettingstarted/> / <https://www.freecodecamp.org/news/what-is-docker-compose-how-to-use-it/>
- Docker-compose Service Attributes: <https://docs.docker.com/reference/compose-file/services/>