dataport

anderScore
trust in competence

Tag 3: GitOps, Docker in der Entwicklung und Deployment-Strategien

19.06.2024, Daniel Krämer & Malte Fischer

HECKER
CONSULTING

Agenda

- **Tag 1 – Einführung in Git und GitLab, Git-Workflow im Team**
  - Einführung & Kursüberblick
  - Grundlagen von Git
  - Git Rebase und Merge-Strategien
  - Git Remote
  - Grundlagen von GitLab
  - Git-Workflow im Team
- **Tag 2 – Vertiefung Git-Workflow, CI/CD & GitLab CI**
  - Gitflow-Workflow
  - Tags, Releases & deren Verwaltung
  - GitLab-Runner
  - Einführung in GitLab CI/CD & gitlab.yml
- **Tag 3 – GitOps, Docker in der Entwicklung und Deployment-Strategien**
  - GitOps Grundlagen
  - Lokale Entwicklung mit Docker
  - Container/Docker-Registry
  - Erstellen von Release- und Tagged-Images
  - Möglichkeiten des Deployments & Verwaltung von Konfiguration
  - Abschlussübung & Diskussion

Agenda

- **Tag 1 – Einführung in Git und GitLab, Git-Workflow im Team**
  - Einführung & Kursüberblick
  - Grundlagen von Git
  - Git Rebase und Merge-Strategien
  - Git Remote
  - Grundlagen von GitLab
  - Git-Workflow im Team
- **Tag 2 – Vertiefung Git-Workflow, CI/CD & GitLab CI**
  - Gitflow-Workflow
  - Tags, Releases & deren Verwaltung
  - GitLab-Runner
  - Einführung in GitLab CI/CD & gitlab.yml
- **Tag 3 – GitOps, Docker in der Entwicklung und Deployment-Strategien**
  - GitOps Grundlagen
  - Lokale Entwicklung mit Docker
  - Container/Docker-Registry
  - Erstellen von Release- und Tagged-Images
  - Möglichkeiten des Deployments & Verwaltung von Konfiguration
  - Abschlussübung & Diskussion

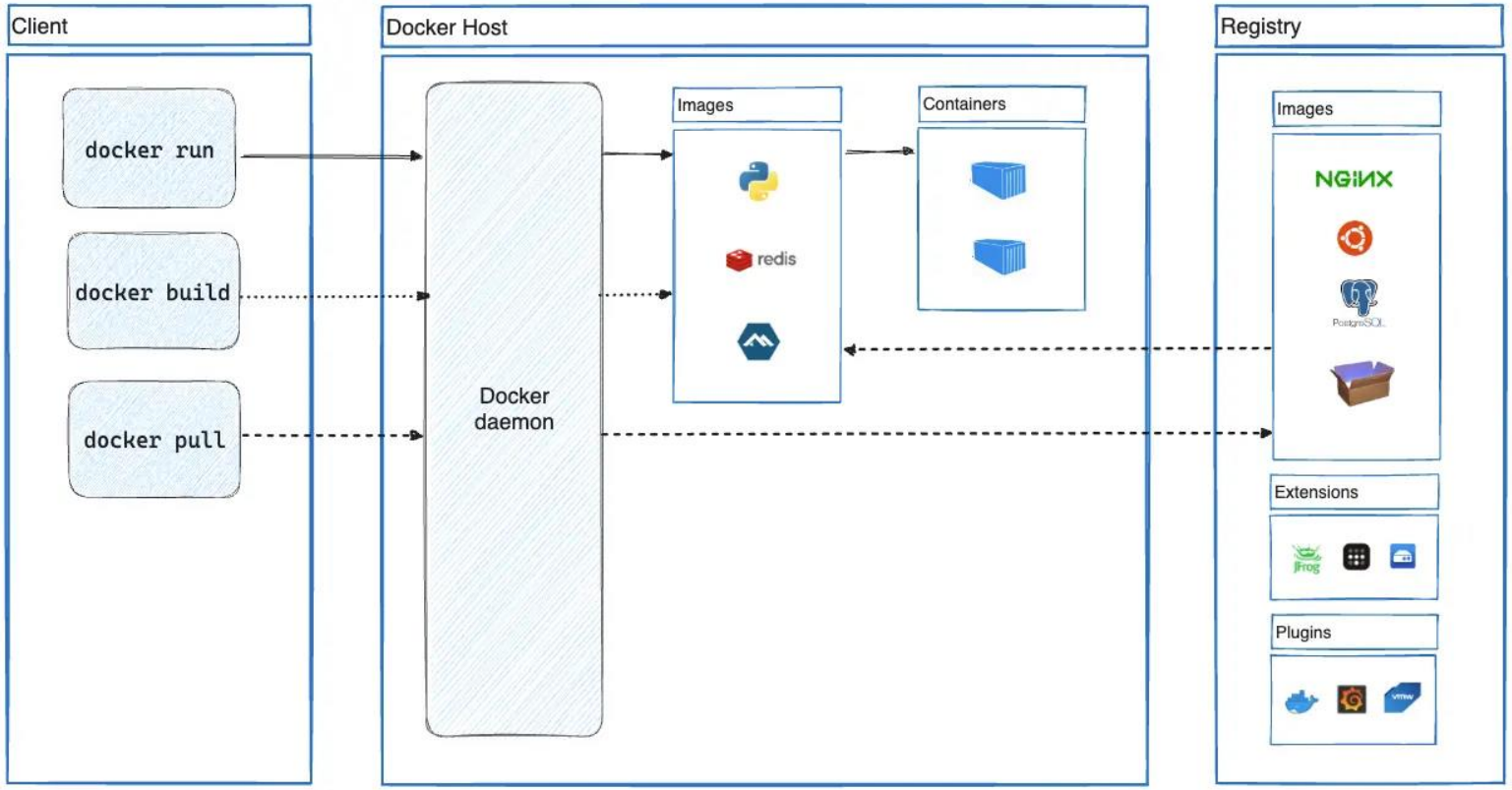
Lokale Entwicklung mit

DOCKER

Was ist Docker?

- Containerisierung von Software
- Isolierte Laufzeitumgebung
- Eigenes Netzwerk
- Hilft beim Bauen, Veröffentlichen und Ausführen
- Verringert die Arbeit für Umgebungsmanagement und –konfiguration
- Nutzt Linux-Boardmittel (u.a. cgroups)

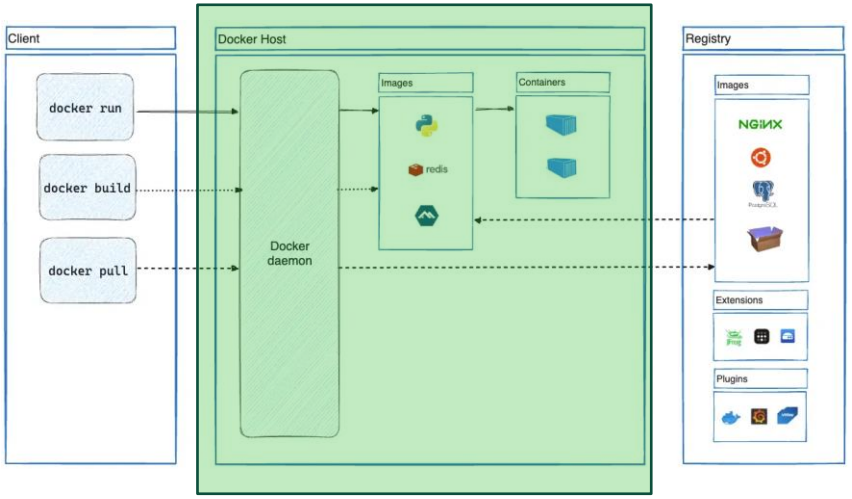
Docker Infrastruktur



https://docs.docker.com/get-started/overview/

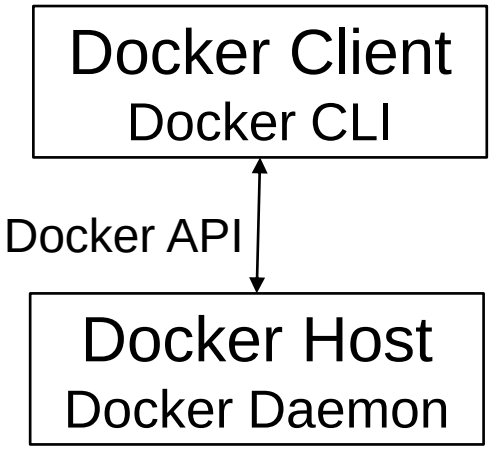
Docker Host

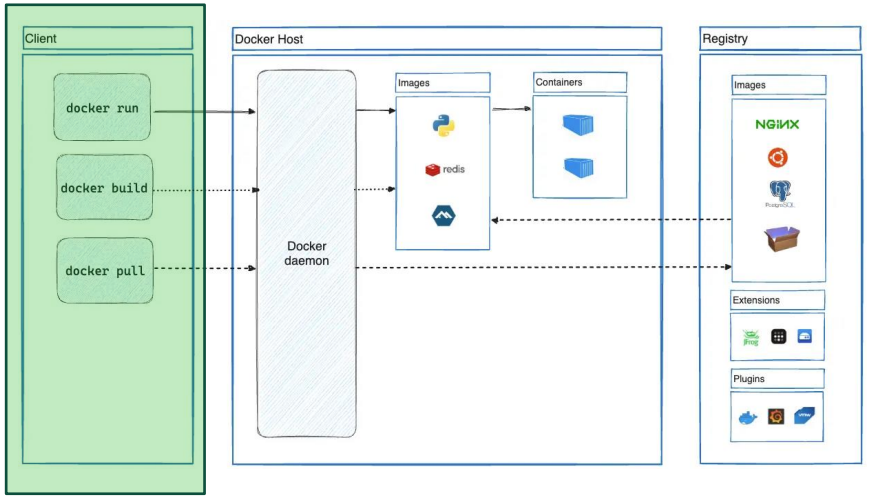
- Runs Docker Daemon (Aufruf: dockerd)
- Überwacht Docker-Objekte (Container, Images, Networks)
- Hört auf Docker API Anfragen
- Kann mit anderen Daemons kommunizieren



Docker Client

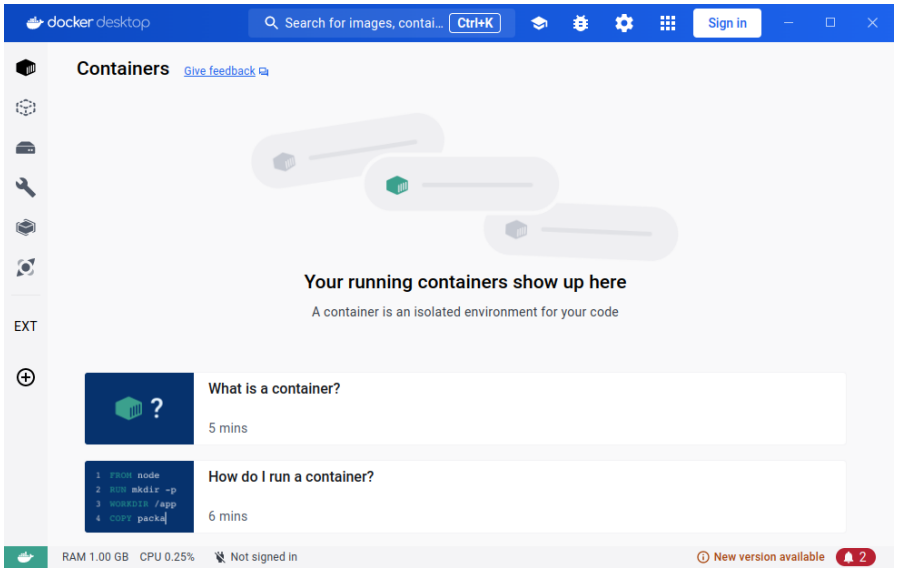
- Primärer Ansprechpunkt für Docker (Aufruf: docker)
- Sendet Anweisungen an Docker Host
- Kann mit mehr als einem Host kommunizieren
- Separater Client für *Docker Compose*



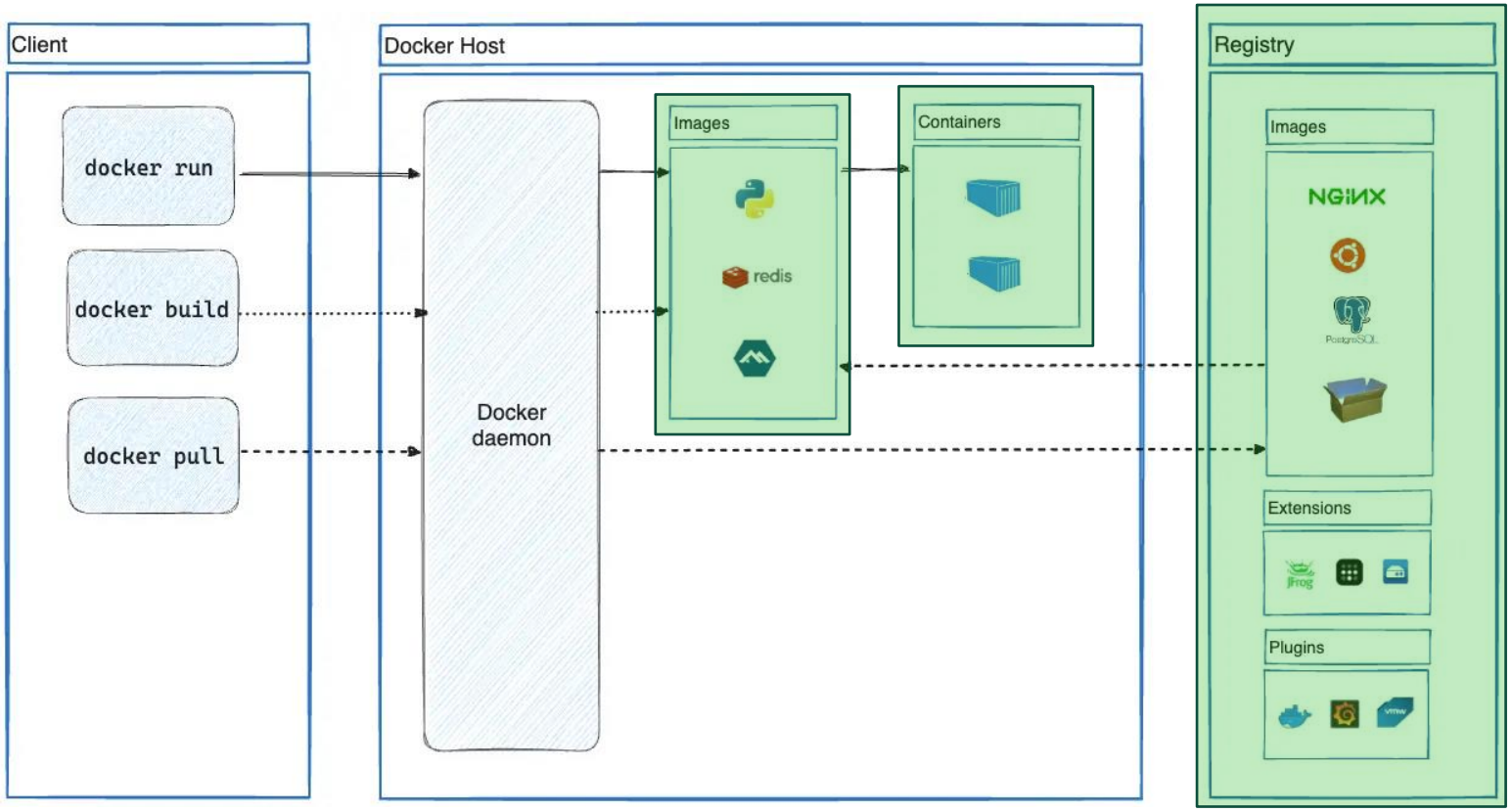


Docker Desktop

- Graphische Benutzeroberfläche
- Freeware, aber nicht OpenSource
- Beinhaltet
  - Docker Daemon
  - Docker Client
  - Docker Compose
  - Docker Content Trust
  - Anbindung an Kubernetes
  - Credentials Helper



Docker Infrastruktur



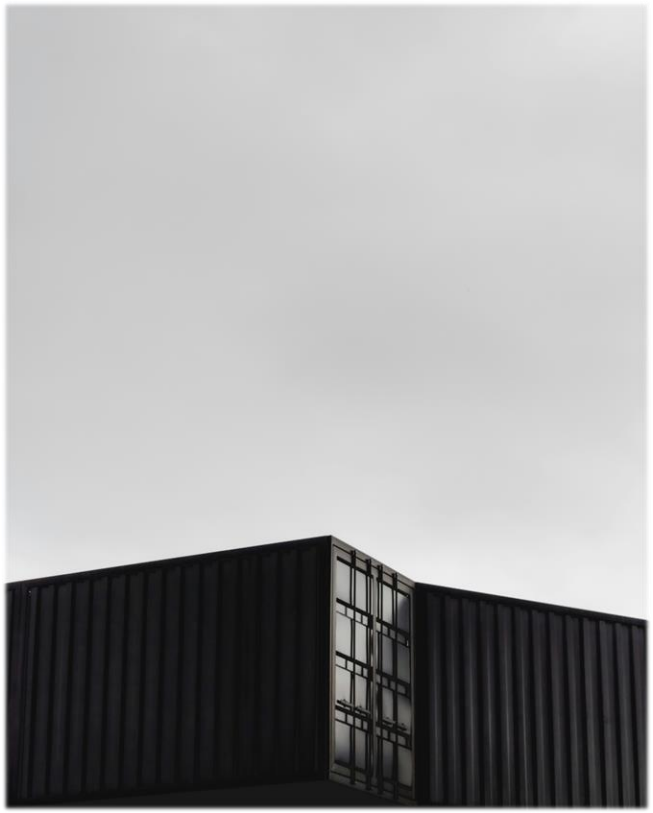
Lokale Entwicklung mit Docker

Was ist ein Container?

- Virtuelle Umgebung für Software
- Klein und leichtgewichtig
- Abgekapselt
- Eigenes Dateisystem
- Performant
- Betriebssystemunabhängig

Warum keine VM?

- Weniger Speicherplatz
  - MB im Vergleich zu GB
- Schnelleres Startup für Scaling
  - Sekunden im Vergleich zu Minuten
- Performanter

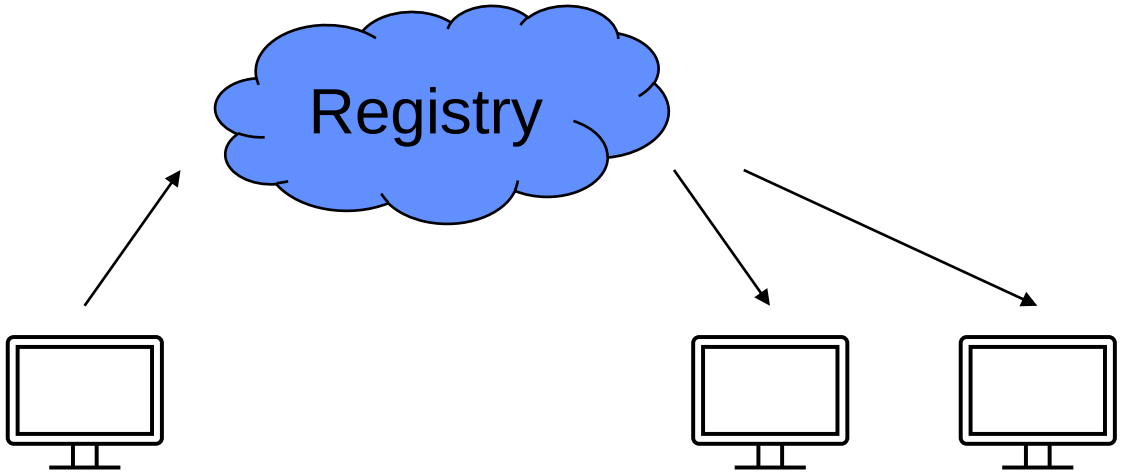


Was ist ein Image?

- „Bauanleitung für Container“
- Ein Image, viele Container
- Werden in Schichten gebaut
- Basieren auf anderen Images

Docker Registries

- Remote Storage für Images
- Können mit Tags gepusht und gepullt werden
- Öffentliche Registries (z.B. DockerHub)
- Private Registries können selbst gehostet werden



Installation

- Plattformabhängig
- Anleitungen in der Dokumentation
- https://docs.docker.com/desktop/install/linux-install/
- https://docs.docker.com/desktop/install/windows-install/
- https://docs.docker.com/desktop/install/mac-install/

Erstellen von Images

- Definition mittels *Dockerfile*
- Definiert die einzelnen Schichten
- Ähnlich wie ein Script

```
FROM bash:latest
WORKDIR /app
ENV MSG="Hello World"
COPY /app .
RUN ["bash","setup.sh"]
CMD ["bash","script.sh"]
EXPOSE 1234
```

FROM <Image>

- Basis für das neue Image
- Kann lokal oder in Registry sein
- Optional
- Wenn fehlt, wird das Image „scratch“ als Basis verwendet
- Nur eine Basis pro Dockerfile

```
FROM bash:latest
WORKDIR /app
ENV MSG="Hello World"
COPY /app .
RUN ["bash","setup.sh"]
CMD ["bash","script.sh"]
EXPOSE 1234
```

WORKDIR <Directory>

- Setzt Arbeitsverzeichnis
- Alle Befehle werden hier ausgeführt

```
FROM bash:latest
WORKDIR /app
ENV MSG="Hello World"
COPY /app .
RUN ["bash","setup.sh"]
CMD ["bash","script.sh"]
EXPOSE 1234
```

ENV <Name>=<Value>

- Setzt Umgebungsvariablen
- Beliebig viele pro Command
  - ENV <Name1>=<Value1> <Name2>=<Value2> …

```
FROM bash:latest
WORKDIR /app
ENV MSG="Hello World"
COPY /app .
RUN ["bash","setup.sh"]
CMD ["bash","script.sh"]
EXPOSE 1234
```

COPY <Source> <Destination>

- Kopiert Dateien und Verzeichnisse in den Container
- Destination ist relativ zum Arbeitsverzeichnis

```
FROM bash:latest
WORKDIR /app
ENV MSG="Hello World"
COPY /app .
RUN ["bash","setup.sh"]
CMD ["bash","script.sh"]
EXPOSE 1234
```

RUN <Command>

- Führt einen Befehl aus
- Wird einmalig beim Bauen des Images ausgeführt
- RUN <Command>
- RUN [<Program>,<Param1>, …]

```
FROM bash:latest
WORKDIR /app
ENV MSG="Hello World"
COPY /app .
RUN ["bash","setup.sh"]
CMD ["bash","script.sh"]
EXPOSE 1234
```

CMD <Command>

- Gibt den Standardstartbefehl an
- Wird standardmäßig beim Containerstart ausgeführt
- Nur **ein** CMD pro Image!
  - Wenn mehr als eins, wird das letzte ausgeführt
- CMD <Command>
- CMD [<Program>,<Param1>, …]

```
FROM bash:latest
WORKDIR /app
ENV MSG="Hello World"
COPY /app .
RUN ["bash","setup.sh"]
CMD ["bash","script.sh"]
EXPOSE 1234
```

EXPOSE <Port>

- Öffnet einen Port des Containers nach außen
- Muss einem Host Port zugewiesen werden

```
FROM bash:latest
WORKDIR /app
ENV MSG="Hello World"
COPY /app .
RUN ["bash","setup.sh"]
CMD ["bash","script.sh"]
EXPOSE 1234
```

Allgemein

- Dockerfiles heißen standardmäßig „Dockerfile“
  - Keine Dateiendung
  - Können anders heißen
- Zusätzliche Funktionalität: siehe Dokumentation
- Einzelne Schichten werden gecached
- https://docs.docker.com/reference/dockerfile/

Lokale Entwicklung mit Docker

Generiert mit Imgflip.com

Erstellen von Images

- docker build: Befehl zum Bauen von Images
- -t <tagname>: Gibt erstelltem Image einen Tag
- <directory>: Verzeichnis des Dockerfiles
- Dockerfile namens „Dockerfile“ im Verzeichnis

```
docker build [-t <tagname>] <directory>
docker build -t example .
```

Erstellen von Containern

- Container haben Namen und ID zur Identifikation
- mit --name können Container explizit benannt werden
- Viele weitere Optionen je nach Image und Applikation

```
docker run <Image> [--name <Name>]
docker create <Image> [--name <Name>]
docker run myimage:latest --name example
```

Starten von Containern

- Kann mit ID oder Name identifiziert werden

```
docker start <Identifier | Name>
docker start example
```

Stoppen von Containern

- Kann mit ID oder Name identifiziert werden
- Stop resultiert in SIGTERM
- Kill resultiert in SIGKILL
- Wenn möglich stop benutzen

```
docker stop <Identifier | Name>
docker kill <Identifier | Name>
docker stop example
```

Löschen von Containern

- Docker löscht den Container
- Kann mit ID oder Name identifiziert werden
- Container muss gestoppt sein

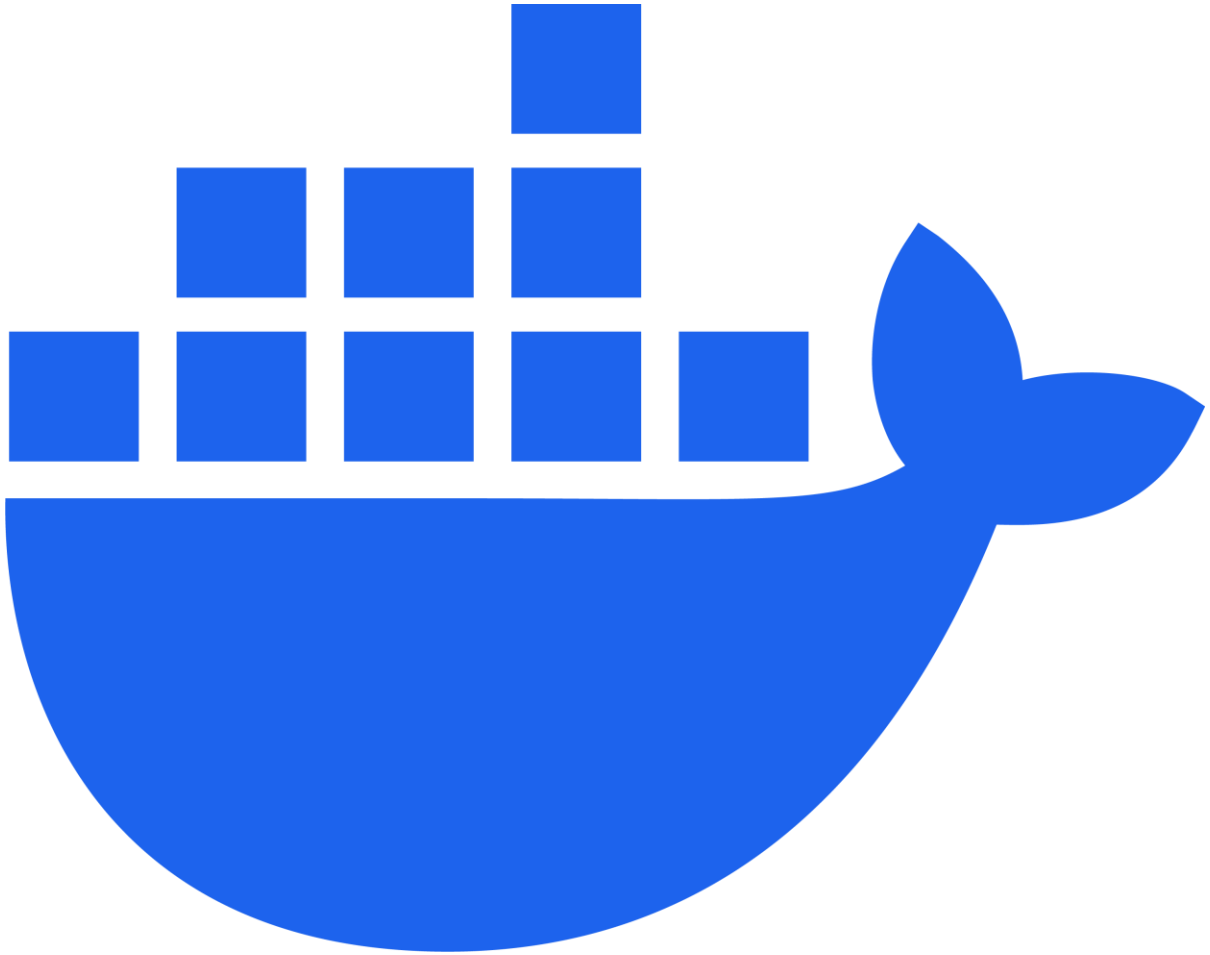
```
docker rm <Identifier | Name>
docker rm example
```

Auflisten von Containern

- Zeigt aktuell verfügbare Container
- Zeigt nur laufende Container an
  - Alle Container können mit -a angezeigt werden

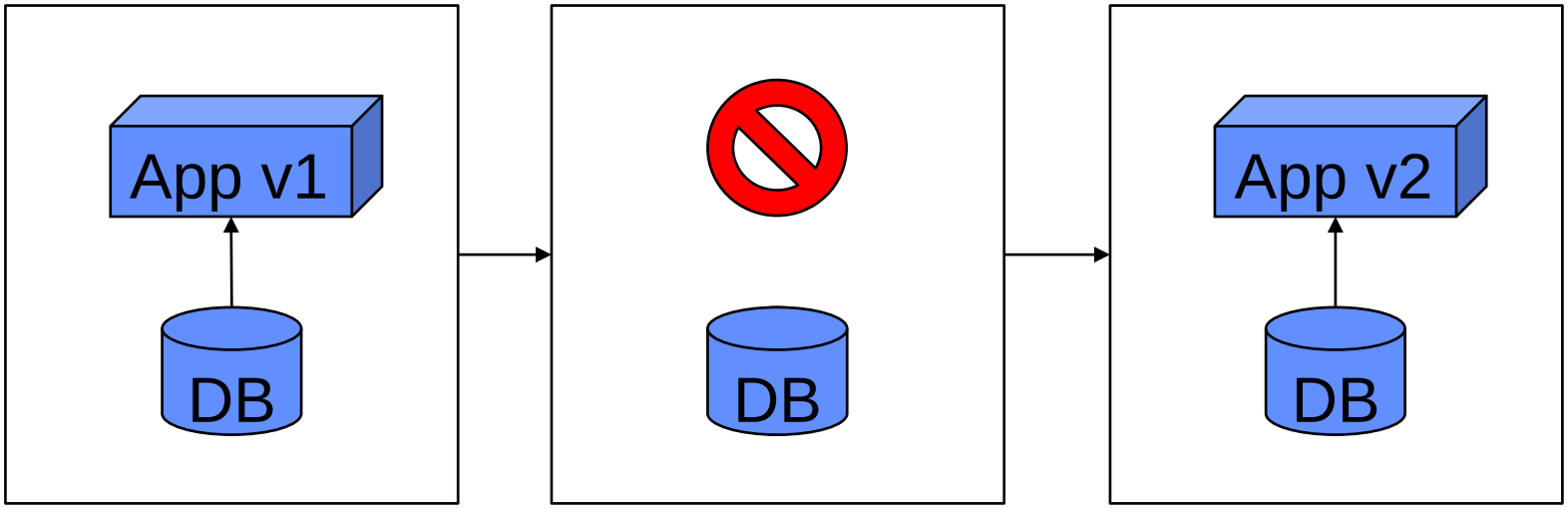
```
docker ls
docker ls -a
docker ps
```

Demo



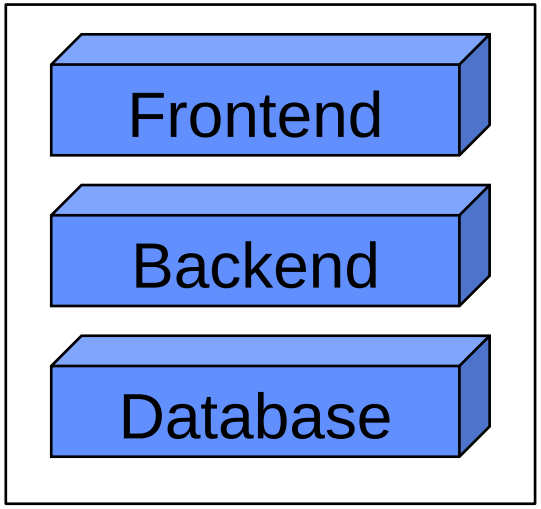
Volumes

- Daten werden gemeinsam mit Container **gelöscht**
- Persistente Daten sollten in Volumes gespeichert werden
- Volumes werden in Container eingebunden
- https://docs.docker.com/storage/volumes/



Docker Compose

- Mehrere Container in gemeinsamem Netzwerk
- Können miteinander kommunizieren
- Können zusammen gestartet und gestoppt werden
- Konkretes Setup (Container + Konfiguration)
- https://docs.docker.com/compose/



Integration mit GitLab CI

- Jobs einer Pipeline in Containern ausführen
- Zuverlässig
- Ein Container pro Step
- Verschiedene Umgebungen
- Abgekapselt

```
build-job:
  image: node
  script:
    - npm install
    - npm run build
  artifacts:
    paths:
      - "build/"
```

Zusammenfassung

- Auslieferung von Software als vorkonfigurierte Images
- Definition mittels Dockerfile
- Ausführung als isolierte Container
- Reproduzierbare Umgebung
- Plattformunabhängig
- Flexibel
- Vielfältiges Hosting möglich