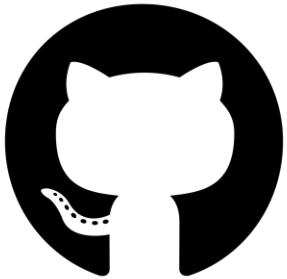
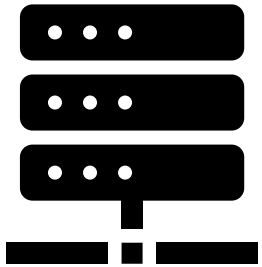
Ausgangspunkt

- Code in einem Repository
- Der Code ist lauffähig und kann erfolgreich in eine ausführbare Form gebracht werden
- Die Anwendung soll / muss in einer variablen Umgebung gestartet werden, um die Funktionalitäten der Anwendung bereitzustellen
- Dafür müssen je nach Programmiersprache / Anforderungen / Umgebung unterschiedliche Schritte ausgeführt werden





Zentrale Frage

Wie kommt unsere Anwendung auf das entsprechende Zielsystem?

Hürden

- Schritte zur Installation müssen in richtiger Reihenfolge ausgeführt werden
- Je nach Betriebssystem der Umgebung
  - Müssen andere Schritte ausgeführt werden
  - Existieren andere technische Abhängigkeiten
  - Gibt es Packages eventuell nicht
- Viel „händischer“ Aufwand bei jeder Installation

- → **Unterschiedliche Umgebungen, unterschiedliche Anforderungen („It work‘s on my machine“)**

Problemlösung durch Docker

- Docker hilft uns diese Probleme zu lösen
- Docker erlaubt uns:
  - Eine **isolierte** Laufzeitumgebung für unsere Software zu erstellen
  - Macht die **Erstellung** der Laufzeitumgebung **reproduzierbar**
  - Wenn **Docker** auf dem Zielsystem **läuft**, wird unsere **Anwendung** in Docker auch auf dem Zielsystem **laufen können**

Was macht Docker?

- Docker bildet eine Plattform zur Trennung von unseren Anwendungen und unserer Infrastruktur
- Mit Docker können wir unsere Anwendung in eine isolierte Umgebung (Container) verpacken und ausführen
- Container sind deutlich leichtgewichtiger als eine VM* und beinhalten alles, was zum Ausführen der Applikation benötigt wird
- Docker stellt uns alle Tools bereit, um unsere Applikation zu verpacken, auszuführen, zu teilen, zu testen, …

* Virtuelle Maschine

Docker vs. Virtuelle Machine

Merkmal	Virtuelle Maschine (VM)	Docker (Container)
Virtualisierungstyp	Hardwarevirtualisierung (Full Virtualization)	Betriebssystemvirtualisierung (OS-level)
Isolationsebene	Eigenes Betriebssystem (inkl. Kernel)	Prozesse teilen sich den Host-Kernel
Startzeit	Minuten	Sekunden
Ressourcenverbrauch	Hoch (RAM, CPU, Speicher)	Gering
Größe	Oft mehrere GB	MB-Bereich
Portabilität	Eingeschränkt	Image + Container laufen überall mit Docker Engine
Use Case	Komplette Infrastruktur, Legacy-Systeme	Microservices, CI/CD, DevOps

Zentrale Docker-Konzepte

- **Images**
  - Blueprint / Bauplan für einen Container
  - Bauplan besteht aus Befehlen, die ausgeführt werden um die notwendige Umgebung einzurichten
  - Besteht aus **Layern** (gecachete Befehle zur Beschleunigung des Bauens)
- **Container**
  - Instanz eines Images
  - Isolierte Laufzeitumgebung mit der Anwendung
  - Kann mit der Host-Maschine interagieren (Ports, Volumes, …)

Zentrale Docker-Konzepte

- **Dockerfile**
  - Skript, das beschreibt, wie das Image gebaut wird
  - Enthält alle notwendigen Befehle zur Einrichtung der Umgebung
- **Docker Engine**
  - Laufzeitumgebung für Container (starte, stoppen, verwalten, …)
- **Registry**
  - Zentrale Server-Anwendung die Images speichern kann
  - Von dort können Images heruntergeladen / hochgeladen werden
  - Wie GitHub für Images
  - (Docker Hub, GitHub Container Registry, Private Registries)

Images

```dockerfile
# 1. Verwende ein offizielles Python-Image als Basis (hier Python 3.11)
FROM python:3.11-slim

# 2. Setze das Arbeitsverzeichnis im Container auf /app
# Alle nachfolgenden Befehle werden relativ zu diesem Pfad ausgeführt
WORKDIR /app

# 3. Kopiere die Datei requirements.txt ins Arbeitsverzeichnis
# Diese Datei enthält die zu installierenden Python-Abhängigkeiten
COPY requirements.txt .

# 4. Installiere die Abhängigkeiten mit pip
# Das Flag --no-cache-dir verhindert das Zwischenspeichern, um das Image kleiner zu halten
RUN pip install --no-cache-dir -r requirements.txt

# 5. Kopiere den restlichen Anwendungscode ins Image (z. B. main.py, Templates, etc.)
COPY . .

# 6. Setze eine Umgebungsvariable (optional)
# Dies kann z. B. genutzt werden, um die App im Produktionsmodus zu starten
ENV FLASK_ENV=production

# 7. Öffne Port 5000 (Standardport von Flask)
# Dies hat keine direkte Auswirkung auf die Netzwerkkonnektivität, dient aber der Dokumentation
EXPOSE 5000

# 8. Definiere den Startbefehl für den Container
# Hier starten wir die Flask-App über das main.py-Skript
CMD ["python", "main.py"]

```

Weiterführende Konzepte

- **Volumes**
  - Persistente Datenspeicherung für Container
  - Für dauerhafte Speicherung von Daten aus dem Container bzw. genutzt vom Container
  - Gemanaged von Docker
- **Netzwerke**
  - Mehrere Container können untereinander mit Netzwerken kommunizieren
  - Namen der Container können, wie bei einem DNS, aufgelöst werden (Containername “db“ wird in IP-Adresse aufgelöst)
- **Umgebungsvariablen**
  - Können für Configs genutzt werden
  - Env Dateien oder einzelne Variablen möglich
- **Docker-Compose**
  - Starten einer Anwendung aus mehreren Containern
  - Docker verwaltet alle Container gleichzeitig