Docker unter Debian und Linux Mint

Hinweis nach AI-VO: Diese Anleitung wirde mit Hilfe von KI generiert und manuell überarbeitet.

Installation und Nutzung

.

1. Installation unter Debian/Linux Mint

Hinweis: Die Docker-Pakete unter Debian sind allgemein schnell veraltet, daher sollte in diesem Schritt das aktuelle Repository hinzugefügt, bereits installierte docker-Pakete gelöscht und die offiziellen Versionen installiert werden.

Vorbereitung

```
# Alte Docker-Versionen entfernen
sudo apt-get remove docker docker-engine docker.io containerd runc

# System aktualisieren
sudo apt-get update
```

Docker installieren

```
# Abhängigkeiten installieren
sudo apt-get install ca-certificates curl gnupg lsb-release

# Docker GPG-Schlüssel hinzufügen
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# Repository hinzufügen
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/debian $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null

# Docker installieren
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd docker-buildx-plugin docker-compose-plugin
```

Docker ohne sudo verwenden

```
# Benutzer zur docker-Gruppe hinzufügen
sudo usermod -aG docker $USER

# Abmelden und wieder anmelden
```

Installation verifizieren

```
docker --version
docker run hello-world
```

.

2. Wichtige Docker-Befehle

Container verwalten

```
# Laufende Container anzeigen
docker ps

# Alle Container anzeigen (auch gestoppte)
docker ps -a

# Container starten
docker start <container-id>

# Container stoppen
docker stop <container-id>
```

```
# Container neu starten
docker restart <container-id>

# Container entfernen
docker rm <container-id>

# Container entfernen (auch laufende)
docker rm -f <container-id>
```

Images verwalten

```
# Vorhandene Images anzeigen
docker images

# Image herunterladen
docker pull <image-name>

# Image entfernen
docker rmi <image-id>

# Nicht verwendete Images entfernen
docker image prune
```

Container ausführen

```
# Container im Vordergrund starten
docker run <image-name>

# Container im Hintergrund starten (detached)
docker run -d <image-name>

# Container mit Port-Mapping
docker run -p 8080:80 <image-name>

# Container mit Volume-Mount
docker run -v /host/path:/container/path <image-name>

# Container mit Umgebungsvariablen
docker run -e VAR=value <image-name>

# Container mit Namen starten
docker run --name mein-container <image-name>

# Interaktiver Container mit Shell
docker run -it <image-name> /bin/bash
```

Container-Informationen

```
# Container-Logs anzeigen
docker logs <container-id>

# Live-Logs anzeigen
docker logs -f <container-id>

# Prozesse im Container anzeigen
docker top <container-id>

# Container-Details anzeigen
docker inspect <container-id>

# In laufenden Container exec ausführen
docker exec -it <container-id> /bin/bash
```

Aufräumen

```
# Nicht verwendete Ressourcen entfernen
docker system prune

# Alle gestoppten Container entfernen
docker container prune

# Alle nicht verwendeten Images entfernen
```

```
docker image prune -a

# Alle Volumes entfernen
docker volume prune
```

.

3. Docker Compose

Installation

Docker Compose ist im Docker-Paket enthalten (ab Version 2.x):

```
docker compose version
```

YAML-Datei erstellen

Eine `docker-compose.yml` Datei im Projektverzeichnis:

```
version: '3.8'

services:
  web:
    image: nginx:latest
    ports:
      - "8080:80"
    volumes:
      - ./html:/usr/share/nginx/html
    depends_on:
      - db

  db:
    image: postgres:15
    environment:
      POSTGRES_PASSWORD: example
      POSTGRES_DB: myapp
    volumes:
      - pgdata:/var/lib/postgresql/data

volumes:
  pgdata:
```

Wichtige docker-compose Befehle

```
# Services im Hintergrund starten
docker compose up -d

# Services im Vordergrund starten
docker compose up

# Services stoppen
docker compose down

# Services neu bauen
docker compose build

# Status der Services anzeigen
docker compose ps

# Logs anzeigen
docker compose logs

# Logs live anzeigen
docker compose logs -f

# In Service-Container exec ausführen
docker compose exec <service-name> /bin/bash

# Services neu starten
docker compose restart
```

```
# Alle Ressourcen entfernen (inkl. Volumes)
docker compose down -v
```

Beispiel: Komplette Anwendung

```
version: '3.8'

services:
  app:
    build: .
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
      - DB_HOST=db
    depends_on:
      - db
    networks:
      - app-network

  db:
    image: mysql:8
    environment:
      MYSQL_ROOT_PASSWORD: secret
      MYSQL_DATABASE: appdb
    volumes:
      - mysql-data:/var/lib/mysql
    networks:
      - app-network

  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    depends_on:
      - app
    networks:
      - app-network

networks:
  app-network:
    driver: bridge

volumes:
  mysql-data:
```

.

4. Nützliche Tipps

Dockerfile erstellen

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

Image bauen

```
docker build -t mein-image .
```

Häufig verwendete Images

```
# Webserver
docker run -d -p 80:80 nginx

# Datenbank
docker run -d -e MYSQL_ROOT_PASSWORD=secret mysql:8

# Entwicklungsumgebung
docker run -it -v $(pwd):/workspace node:18
```

```
# Datenbank-Client
docker run -it --rm postgres:15 psql -h <host> -U <user>
```

.

5. Zusammenfassung

Befehl	Beschreibung
`docker ps`	Laufende Container anzeigen
`docker images`	Vorhandene Images anzeigen
`docker run`	Neuen Container starten
`docker build`	Image aus Dockerfile bauen
`docker compose up`	Services aus YAML starten
`docker compose down`	Services stoppen

.

Handout erstellt für Docker unter Debian/Linux Mint