

[zurück](#)

# Docker - Grundlagen (Images, Container, Volumes, Networks, Compose)

Docker ist eine Container-Technologie, die Anwendungen leichtgewichtig, portabel und isoliert betreibt, ohne komplexe virtuelle Maschinen zu benötigen.

Container teilen sich den Host-Kernel, benötigen aber keine eigenen Betriebssysteme → dadurch starten sie extrem schnell und verbrauchen wenig Ressourcen.

---

## 1. Warum Docker?

Docker löst typische IT-Probleme:

- „Läuft auf meinem Rechner, aber nicht bei dir“
- unterschiedliche Abhängigkeiten / Bibliotheken
- komplizierte Installationen
- Versionskonflikte

Mit Docker bekommst du:

- **isierte Umgebung**
  - **reproduzierbare Builds**
  - **portierbare Anwendungen**
  - **schnelle Deployments**
  - **perfekt für Microservices**
- 

## 2. Container vs virtuelle Maschinen

### Container

- teilen sich den Kernel
- starten in Sekunden
- sehr leichtgewichtig
- ideal für Webapps, DBs, Dienste

## Virtuelle Maschinen

- vollständiges OS
- hohe Isolation
- größere Ressourcenlast

Vergleich:

Container:  
Host → Kernel → Container → App

VM:  
Host → Hypervisor → volles OS → App

---

## 3. Docker-Image

Ein Image ist eine **Vorlage**, aus der Container gestartet werden.

Besteht aus Schichten (Layers):

- Basis-Image (z. B. Debian, Alpine)
- App-Dateien
- Konfiguration
- Bibliotheken

Images sind:

- unveränderlich
- versionierbar
- portabel

Beispiele:

```
nginx:latest  
mariadb:11  
ubuntu:22.04
```

## 4. Container

Ein Container ist eine laufende Instanz eines Images.

Merkmale:

- isoliert vom Rest des Systems
- hat eigene Prozess-ID, eigenes Netzwerk
- nutzt das Image als Grundlage
- nicht persistent (ohne Volume gehen Daten verloren)

Befehle:

```
docker run  
docker ps  
docker stop  
docker logs
```

## 5. Volumes - Persistente Daten

Container-Daten sind flüchtig.

Für dauerhafte Speicherung nutzt man **Volumes**.

Beispiele:

- Datenbanken
- Konfigurationen
- Zertifikate
- Uploads (z. B. Nextcloud)

Volume-Typen:

- named volumes („docker-volume“)
- bind mounts (z. B. /opt/stacks/app/data)

```
Container → /data ← /opt/stacks/app/data
```

## 6. Docker Networks

Container kommunizieren über Netzwerke.

Arten:

### **bridge**

Standard-Netzwerk, Container → Container.

### **host**

Container nutzt Host-Netzwerk direkt.

### **macvlan**

Container bekommt eigene MAC-Adresse im LAN.

### **overlay**

Für Swarm/Kubernetes-Cluster.

Beispiel:

```
docker network create mynet
```

## 7. Docker Compose

Docker Compose definiert komplette Anwendungen als YAML-Datei.

Einfaches Beispiel:

```
version: "3" # kann weggelassen werden da veraltet

services:
  web:
    image: nginx
    ports:
      - "80:80"
    volumes:
      - ./html:/usr/share/nginx/html
```

Vorteile:

- mehrere Container auf einmal starten
- verständliche Struktur
- Netzwerke, Volumes, Umgebungsvariablen definierbar
- ideal für jede moderne Architektur

Starten:

```
docker compose up -d
```

## 8. Registries

Registries speichern Images.

Öffentliche:

- Docker Hub
- GitHub Container Registry

Private:

- Harbor
- GitLab Registry
- selbst gehosteter Registry-Container

## 9. Sicherheit in Docker

- Container nicht als root ausführen
  - eigene Netzwerke nutzen
  - Secrets sicher speichern (docker secrets)
  - nicht „latest“ verwenden
  - Images regelmäßig aktualisieren
  - wenig privilegierte Container starten (-cap-drop)
  - Traefik/Nginx vor Webservices setzen
- 

## 10. Realwelt-Beispiele (praktisch & TG-tauglich)

### Webserver + Datenbank

```
web (nginx)
db   (mariadb)
```

### Mein Home-Lab

- Traefik
- Portainer
- Vaultwarden
- Nextcloud
- Matrix
- Mailserver
- Monitoring (Grafana, Promtail, Loki)
- CrowdSec

### Unternehmensumgebung

- Microservices
- Load Balancer
- CI/CD Pipelines
- API Gateways

# 11. Best Practices

- eindeutige Verzeichnisstruktur (/opt/stacks/)
- alles in Docker Compose statt Einzellauf
- Secrets in .env oder docker secrets
- Healthchecks nutzen
- Backups der Volumes nicht vergessen
- Logs zentral erfassen (Loki, ELK)

## Zusammenfassung

- Docker nutzt Container statt VMs
  - Images als Vorlage → Container als laufende Instanz
    - \* Volumes speichern Daten permanent
    - \* Docker Networks verbinden Container
    - \* Compose verwaltet komplett Anwendungen
    - \* Container sind leicht, schnell und portabel
    - \* Sicherheit ist wichtig (root vermeiden, Updates)
    - \* Container sind Standard in DevOps & modernen IT-Umgebungen

From:

<http://wiki.nctl.de/dokuwiki/> - □ Veni. Vidi. sudo rm -rf / vici.

Permanent link:

<http://wiki.nctl.de/dokuwiki/doku.php?id=it-themen:grundlagen:netzwerkdienste:docker&rev=1764851876>

Last update: **04.12.2025 13:37**

