

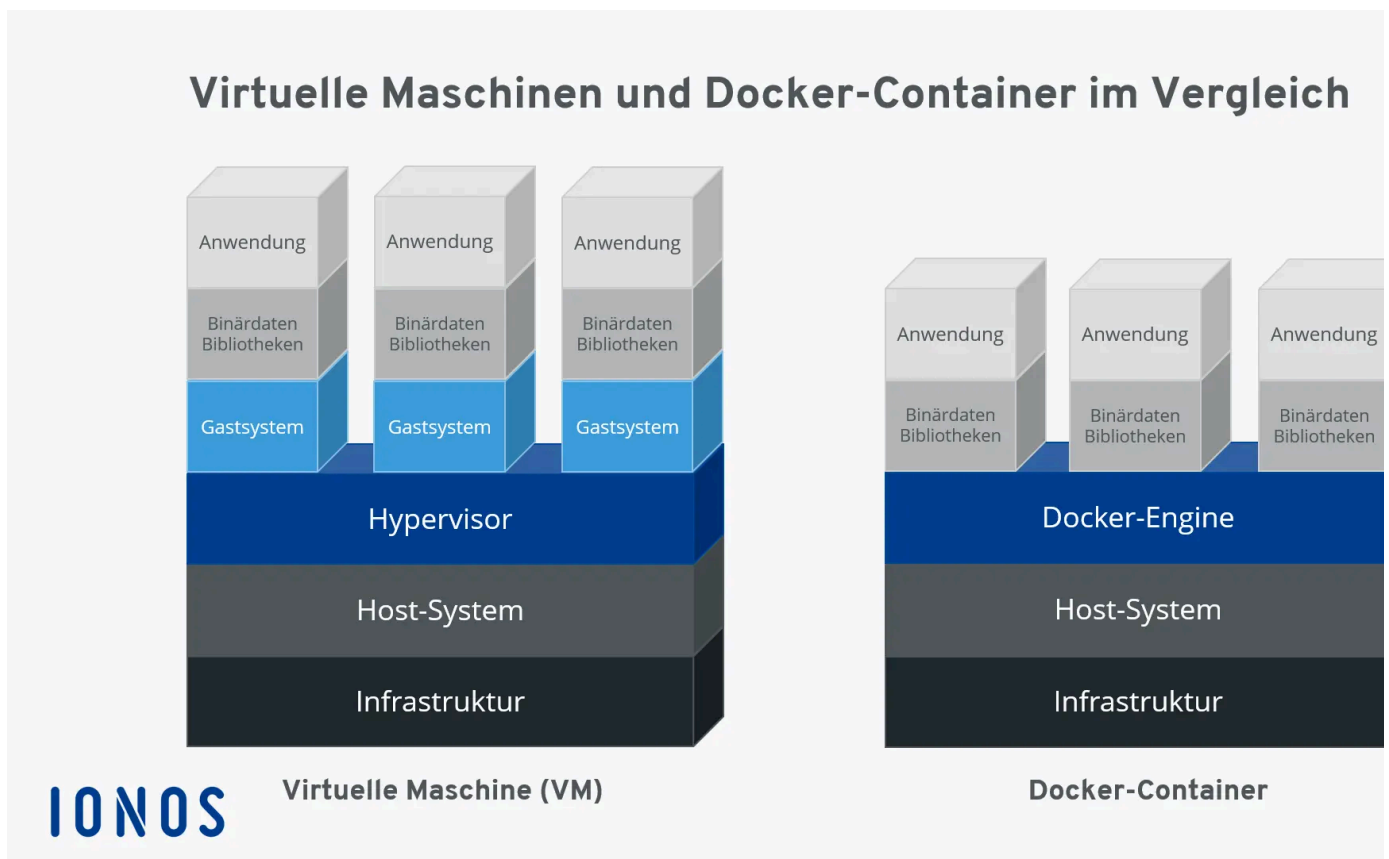
Docker

Was ist Docker?

Docker ist eine Plattform zur **Containerisierung** von Anwendungen. Das bedeutet: Docker ermöglicht es, Software inklusive aller Abhängigkeiten, Bibliotheken und Konfigurationen in sogenannten **Containern** auszuführen – isolierten, portablen Einheiten, die auf jedem System mit Docker gleich laufen.

Wie Funktioniert Docker?

Docker basiert auf **Linux-Containern**, die eine **prozessbasierte Virtualisierung** bereitstellen. Anders als virtuelle Maschinen (VMs) virtualisieren Container **nicht das gesamte Betriebssystem**, sondern nutzen den **Kernel des Host-Betriebssystems** und isolieren Prozesse durch **Namespaces** und **Control Groups (cgroups)**.



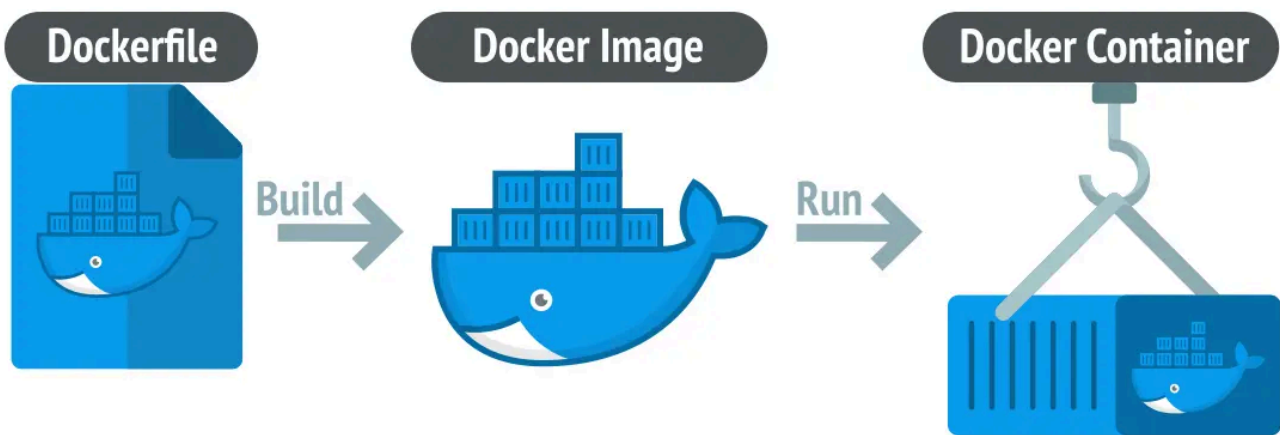
Cgroups:

- Definition von Quotas auf Prozesse (auf CPU- und RAM-Ebene)

Namespaces:

- Prozesse isolieren (denkt er ist der einzige Prozess)
- Netzwerkinterface zu Prozessen zuordnen

- virtuelles Dateisystem für Prozesse



Dockerfile

Um ein Image zu erstellen benötigt man zuerst ein Dockerfile. Dieses könnte beispielhaft so aussehen:

```
FROM python:3.11-slim          # Layer 1: Basisimage
WORKDIR /app                   # Layer 2: Arbeitsverzeichnis
COPY requirements.txt .        # Layer 3: Dateien kopieren
RUN pip install -r requirements.txt # Layer 4: Python-Abhängigkeiten
COPY . .                       # Layer 5: Anwendungscode
CMD ["python", "main.py"]      # Startbefehl für Container
```

Mit dem Befehl `docker build -t imagename .` wird dann das Docker Image erstellt.

Images

Ein **Docker Image** (ein ISO-File sozusagen) ist eine **unveränderliche Vorlage** (read-only), aus der ein oder mehrere **Container** erstellt werden können. Man kann es sich als **Schnappschuss eines Dateisystems** vorstellen – inklusive aller benötigten Dateien, Konfigurationen, Abhängigkeiten und der Anwendung selbst.

Ein Docker Image besteht aus mehreren **Layern**, die übereinandergestapelt sind:

```
[ Layer 4 ] ← Anwendungscode (z. B. Python-Skript)
[ Layer 3 ] ← zusätzliche Abhängigkeiten (z. B. pip install ...)
[ Layer 2 ] ← Systempakete (z. B. apt install ...)
[ Layer 1 ] ← Basis-Image (z. B. ubuntu:20.04)
```

Jeder dieser Layer ist **read-only**. Erst beim Starten eines Containers wird **ein schreibbarer Layer** darübergelegt.

Um Images herunterzuladen wird der Befehl `docker pull <imagename>` verwendet

Docker Container

Ein Docker Container ist die laufende Instanz eines Docker-Images.

Hier ein Beispiel eines Nginx Containerstarts:

```
docker run -d nginx
```

Wenn das `nginx` Image noch nicht am System verfügbar ist wird es automatisch vom Docker-Hub "gepult".

Die Flag `-d` lässt den Container im "detached" Modus laufen, dies bedeutet, dass keine direkte Interaktion mit dem Container per Kommandozeile möglich ist und keine Logs direkt ausgegeben werden

Hier ein paar der wichtigsten Flags:

Allgemeine Steuerung:

Flag	Beschreibung
<code>-d</code> , <code>--detach</code>	Container im Hintergrund starten (detached mode)
<code>--name</code> NAME	Benutzerdefinierter Name für den Container
<code>-it</code>	Kombiniert <code>--interactive</code> und <code>--tty</code> (z. B. für Bash)
<code>--rm</code>	Container nach dem Stoppen automatisch löschen
<code>--restart=...</code>	Automatischer Neustart (z. B. <code>always</code> , <code>on-failure</code>)

Ressourcenverwaltung:

Flag	Beschreibung
<code>--memory</code> , <code>-m</code>	Begrenzung des Speichers (z. B. <code>512m</code> , <code>2g</code>)
<code>--cpus</code>	Begrenzung der CPU-Kerne (z. B. <code>1.5</code>)
<code>--cpu-shares</code>	CPU-Gewichtung im Vergleich zu anderen Containern
<code>--pids-limit</code>	Begrenzung der maximalen Prozessanzahl

Netzwerk:

Flag	Beschreibung
<code>-p</code> HOST:CONTAINER	Portweiterleitung (z. B. <code>-p 8080:80</code>)
<code>--network</code>	Netzwerkmodus (<code>bridge</code> , <code>host</code> , <code>none</code> , benanntes Netz)
<code>--hostname</code>	Eigener Hostname im Container

Flag	Beschreibung
<code>--dns</code>	Benutzerdefinierte DNS-Server im Container

Hier noch ein komplexer run Befehl:

```
docker run -d \
  --name webapp \           # Name des Containers
  -p 8080:80 \             # Portweiterleitung ans Hostsystem
  -e NODE_ENV=production \  # Environmentvariablen
  -v $(pwd)/app:/usr/src/app \ # Volume einbinden
  --memory="512m" --cpus="1.0" \ # Maximaler RAM und CPU Kerne
  --restart=always \        # Automatischer Neustart
  my-node-image             # Image
```

Networking

Docker bringt beim Installieren automatisch drei Netzwerkmodi mit:

Name	Typ	Beschreibung
bridge	Standard für einzelne Container	Eigene virtuelle Bridge (z. B. <code>docker0</code>), Container bekommen eigene IP
host	Kein Netzwerk-Namespace	Container nutzt Host-Netzwerk direkt, keine Isolierung (schnell, aber unsicherer)
none	Komplett isoliert	Container hat kein Netzwerk , nutzbar für isolierte Tasks
benutzerdefiniert (<code>--driver bridge</code>)	Vom Nutzer angelegtes Bridge-Netzwerk	Ermöglicht DNS-Namen, Container-Kommunikation über Namen

bridge (Standard):

```
docker run -d --name web -p 8080:80 nginx
```

- Container läuft im Default-Bridge-Netzwerk.
- Docker nattet den Port `8080` auf `80` im Container. -> `<Hostport>:<Containerport>`
- Container hat eigene IP (z. B. `172.17.0.2`).
- Service erreichbar unter `localhost:8080`

host :

```
docker run --network host nginx
```

- Container verwendet direkt die Netzwerk-Interfaces des Hosts.
- Kein `-p` nötig – Ports sind direkt erreichbar.

none :

```
docker run --network none alpine
```

- Kein Internetzugang.
- Keine Kommunikation mit anderen Containern.

Benutzerdefinierte Netzwerke

Man kann eigene Netzwerke erstellen, um z. B. mehrere Container miteinander kommunizieren zu lassen:

```
docker network create my-net
```

Dann:

```
docker run -d --network my-net --name db postgres  
docker run -d --network my-net --name app myapp
```

- Jetzt kann `app` mit `db` über den DNS-Namen `db` kommunizieren.
- Docker erstellt automatisch DNS-Einträge und Routing.

Datenbanken

Der große Vorteil von Docker ist es unter anderem Datenbanksysteme in sekundenschnelle hochzufahren. Hier ein Beispiel zu einem MySQL Datenbanksystem:

```
docker run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=mysql --name  
mysqldb mysql
```

Commands:

Containerverwaltung:

Befehl	Beschreibung
<code>docker run IMAGE</code>	Startet einen neuen Container aus einem Image

Befehl	Beschreibung
<code>docker run -it IMAGE</code>	Startet interaktiv mit Terminal (z. B. für Bash)
<code>docker start CONTAINER</code>	Startet einen gestoppten Container
<code>docker stop CONTAINER</code>	Stoppt einen laufenden Container
<code>docker restart CONTAINER</code>	Startet Container neu
<code>docker rm CONTAINER</code>	Löscht einen (gestoppten) Container
<code>docker exec -it CONTAINER bash</code>	Führt Befehl im laufenden Container aus (z. B. Bash)
<code>docker logs CONTAINER</code>	Zeigt die Logs eines Containers
<code>docker ps</code>	Zeigt laufende Container
<code>docker ps -a</code>	Zeigt alle Container (auch gestoppte)
<code>docker inspect CONTAINER</code>	Zeigt Detailinfos zu einem Container

Imageverwaltung:

Befehl	Beschreibung
<code>docker pull IMAGE</code>	Lädt ein Image aus Docker Hub
<code>docker build -t NAME .</code>	Baut ein Image aus einem Dockerfile im aktuellen Verzeichnis
<code>docker images</code>	Listet alle lokalen Images
<code>docker rmi IMAGE</code>	Löscht ein Image
<code>docker tag IMAGE NEW_NAME</code>	Vergibt neuen Namen/Tag für ein Image
<code>docker save -o file.tar IMAGE</code>	Exportiert ein Image in eine Datei
<code>docker load -i file.tar</code>	Importiert ein Image aus Datei

Netzwerk & Ports:

Befehl	Beschreibung
<code>docker network ls</code>	Listet alle Netzwerke
<code>docker network create NAME</code>	Erstellt ein neues Netzwerk
<code>docker network inspect NAME</code>	Zeigt Details zu einem Netzwerk
<code>docker run --network=NAME IMAGE</code>	Startet Container in bestimmtem Netzwerk
<code>docker run -p HOST:CONTAINER IMAGE</code>	Portweiterleitung vom Host zum Container

Volumes und Speicher:

Befehl	Beschreibung
<code>docker volume ls</code>	Listet alle Volumes
<code>docker volume create NAME</code>	Erstellt ein neues Volume
<code>docker run -v VOLUME:/pfad IMAGE</code>	Mountet ein Volume in den Container
<code>docker volume inspect NAME</code>	Zeigt Detailinfos zu einem Volume
<code>docker volume rm NAME</code>	Löscht ein Volume

System aufräumen:

Befehl	Beschreibung
<code>docker system prune</code>	Löscht ungenutzte Container, Netzwerke und Images
<code>docker image prune</code>	Entfernt ungenutzte Images
<code>docker container prune</code>	Entfernt gestoppte Container
<code>docker volume prune</code>	Entfernt unbenutzte Volumes

Sonstiges:

Befehl	Beschreibung
<code>docker stats</code>	Zeigt Ressourcenverbrauch aller laufenden Container
<code>docker top CONTAINER</code>	Zeigt Prozesse im Container
<code>docker cp CONTAINER:pfad ./ziel</code>	Kopiert Dateien vom Container auf den Host
<code>docker info</code>	Zeigt Systeminformationen zu Docker
<code>docker version</code>	Zeigt Docker-Client- und Server-Version