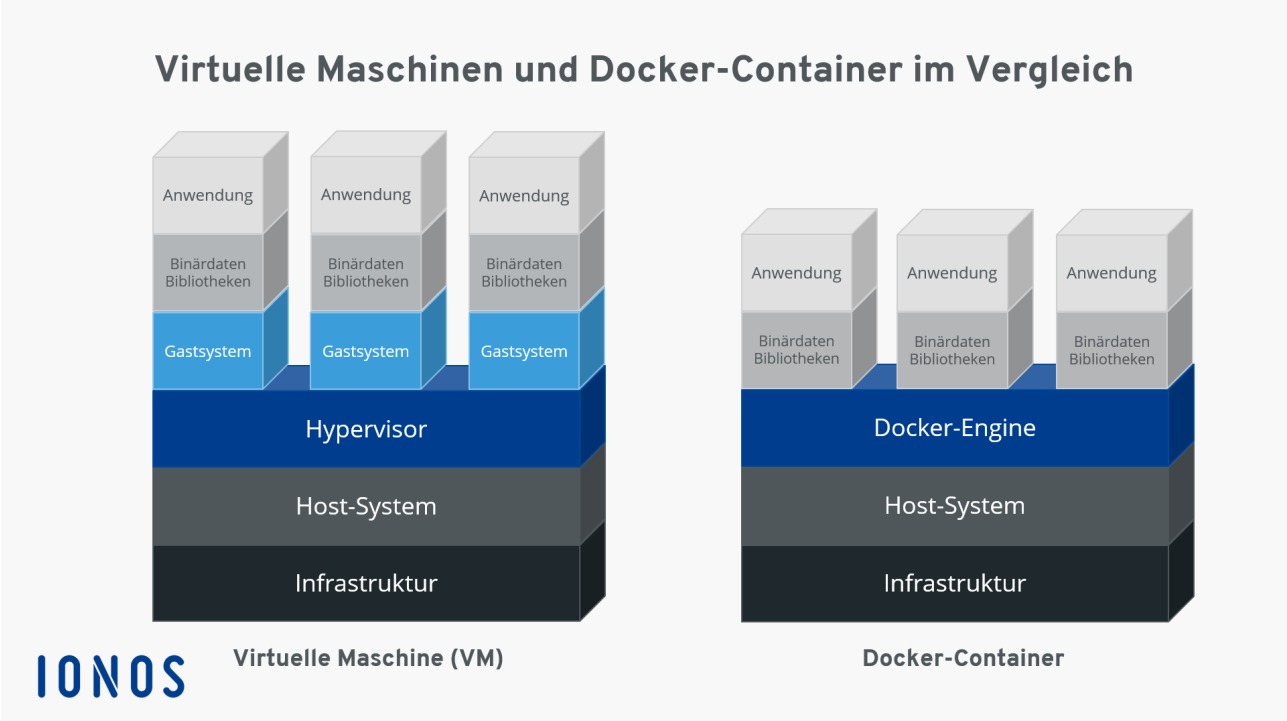
Docker Zusammenfassung

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).**



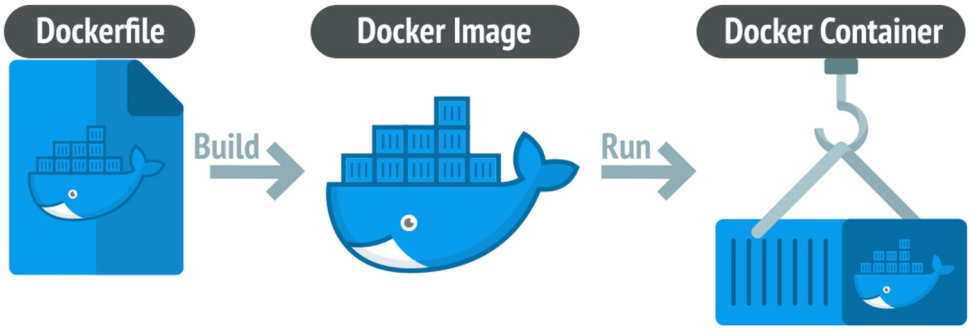
Cgroups:

- 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

Docker Zusammenfassung



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 Zusammenfassung

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 "gepullt".
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 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. docker0), 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
bentzerdefiniert	Vom Nutzer angelegtes Bridge-Netzwerk	Ermöglicht DNS-Namen, Container-Kommunikation über Namen

Docker Zusammenfassung

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.

Docker Zusammenfassung

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:

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

Imageverwaltung:

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

Netzwerk & Ports:

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

Docker Zusammenfassung

Volumes & Speicher:

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

System cleanup:

docker system prune	Löscht ungenutzte Container, Netzwerke und Images
docker image prune	Entfernt ungenutzte Images
docker container prune	Entfernt gestoppte Container
docker volume prune	Entfernt unbenutzte Volumes

Sonstiges:

docker stats	Zeigt Ressourcenverbrauch aller Container
docker top CONTAINER	Zeigt Prozesse im Container
docker cp CONTAINER:pfad ./ziel	Kopiert Datei vom Container auf den Host
docker info	Zeigt Systeminformationen zu Docker
docker version	Zeigt Docker-Client- und Server-Version