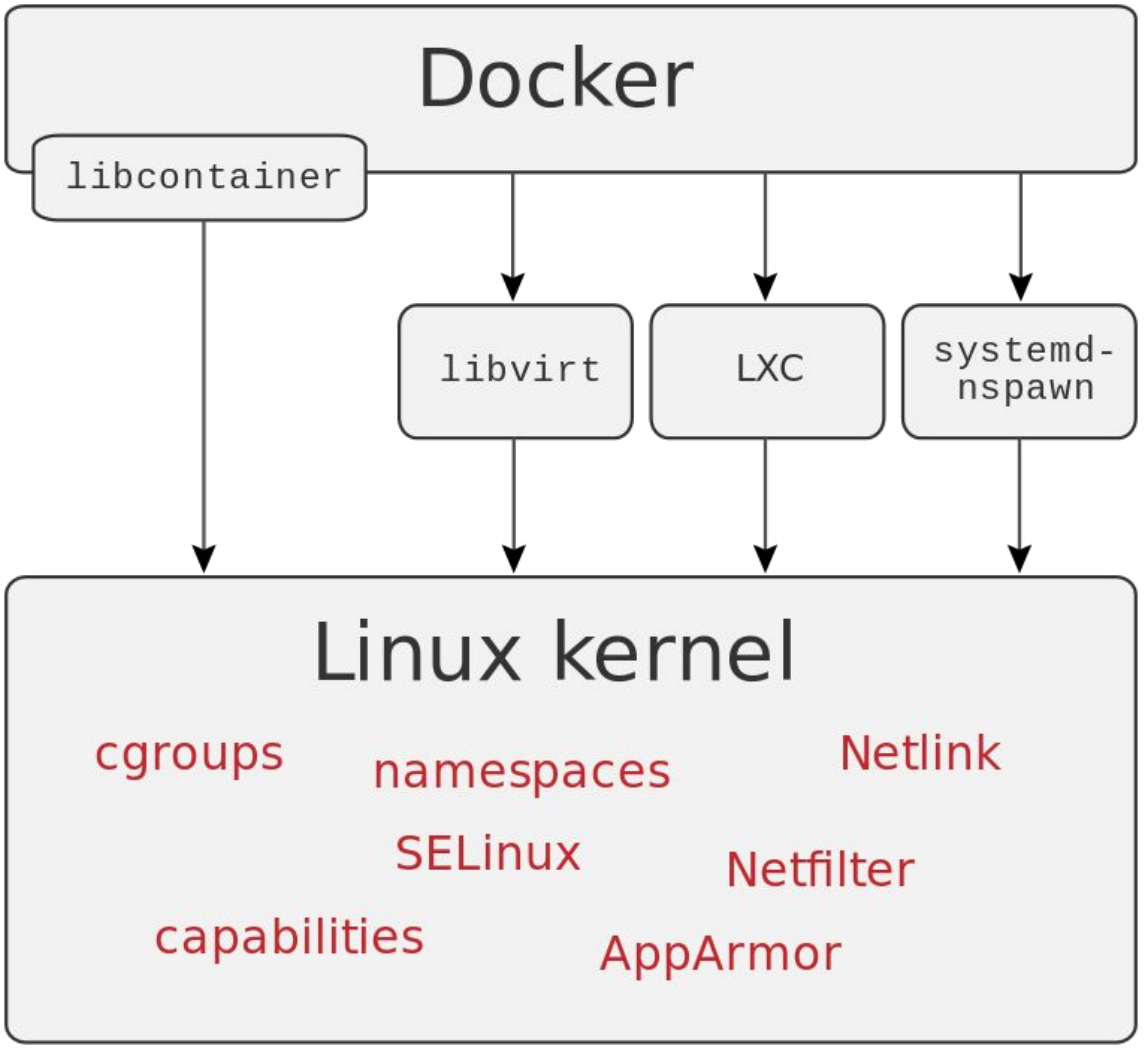
Was ist Docker?

Docker auf dem Server

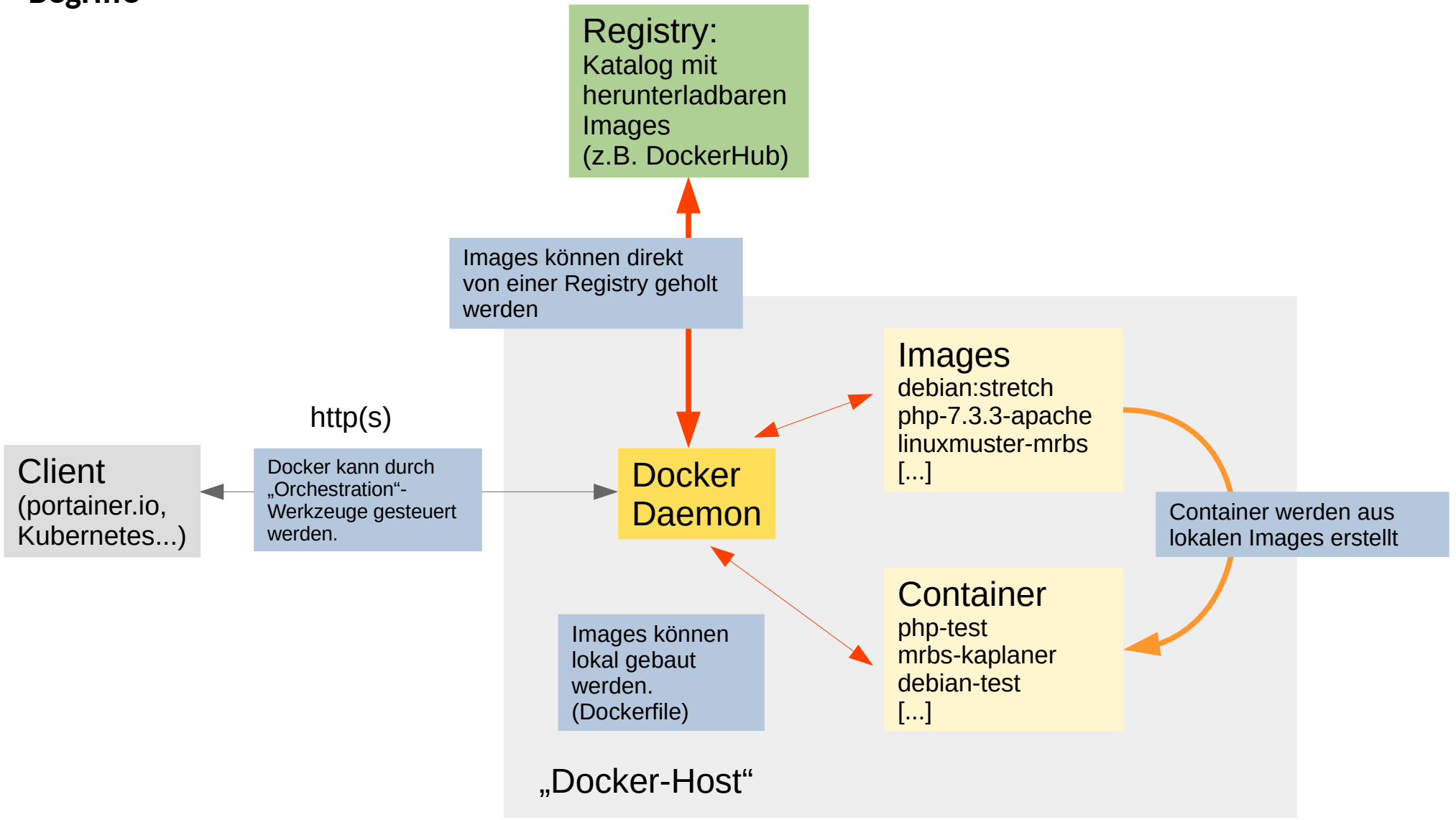
- Cgroups
- Namespaces
- AuFS
- Verwandt zu LXC („LXC+“)

Kernel des Hostsystems



https://commons.wikimedia.org/wiki/File:Docker-linux-interfaces.svg
Lizenz: Gemeinfrei

Begriffe



Wir bauen ein Image und starten einen Container – Zyklus 1

Dockerfile

```
From debian:stretch
Run apt-get -y update
Run apt-get -y dist-upgrade
```

„Image Bauplan“

docker build -t meintest/helloworld .

docker build -t meintest/helloworld:latest .

(Namensgebung: user/image:tag)

Image
meintest/helloworld

docker run meintest/helloworld echo "Hallo Welt"

Container erzeugen
Befehl ausführen
Container stoppen

Image Layers

```
docker history meintest/helloworld

IMAGE          CREATED              CREATED BY
a3b95c6e9c0e   About a minute ago   /bin/sh -c apt-get -y dist-upgrade
e86b954dbb0e   About a minute ago   /bin/sh -c apt-get -y update
2d337f242f07   5 days ago           /bin/sh -c #(nop)  CMD ["bash"]
<missing>      5 days ago           /bin/sh -c #(nop) ADD file:843b8a2a9df1a0730
```

Container Layers

admiring_curie | cocky_archimedes

```
a3b95c6e9c0e
e86b954dbb0e
2d337f242f07
<missing>
```

Gestoppte Container „liegen rum“

```
docker container ls -a
CONTAINER ID        IMAGE               NAME
3ce98719d388        2d337f242f07        admiring_curie
03ff4031304c        2d337f242f07        cocky_archimedes
```

Lebenszyklus eines Containers – Zyklus 2

Achtung: Updates von Upstream!

Image
meintest/helloworld

`docker run IMAGE`

Container
Führt Befehl aus oder
„Läuft“ (führt laufend Befehl aus,
z.B. `apache -DFOREGROUND`

Container stoppt oder
`docker stop CONTAINER`

Container liegt rum

Mit seinem Daten-Layer!

`docker start CONTAINER`

Container
Führt Befehl aus oder
„Läuft“ (führt laufend Befehl aus,
z.B. `apache -DFOREGROUND`

(Mit Daten von zuvor)

Container stoppt oder
`docker stop CONTAINER`

Container liegt rum

Mit seinem Daten-Layer!

`docker container rm CONTAINER`

Funktionalität: Entrypoint Skript

Dockerfile

```
From debian:stretch
Run apt-get -y update
Run apt-get -y dist-upgrade
Copy entrypoint.sh /usr/bin/entrypoint.sh
Entrypoint [“/usr/bin/entrypoint.sh“]
```

„Image Bauplan“

- **Copy** schafft das Skript ins Image
- **Entrypoint** legt es als „Startbefehl“ eines neuen Containers fest

entrypoint.sh

```
#!/bin/bash
if [ $# -eq 0 ]; then
        /usr/games/fortune |
/usr/games/cowsay
else
        # moo
        /usr/games/cowsay "$@"
fi
```

```
root@dockertest:~/first-image# docker run meintest/helloworld
 _____________________________________
/ Q: How did you get into artificial   \
| intelligence? A: Seemed logical -- I |
\ didn't have any real intelligence.   /
 -------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
root@dockertest:~/first-image# docker run meintest/helloworld "Hallo Linuxer!"
 ________________
< Hallo Linuxer! >
 ----------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
root@dockertest:~/first-image#
```

Daten persistent speichern: Volumes

```
docker run -it --name volumetest -v /data debian:stretch /bin/bash
```

Kann auch schon im Dockerfile definiert werden:

Volume /data

Das Verzeichnis /data im Container kann persistent Daten speichern. Diese landen im Dateisystem des Hosts, Wahl des Hostverzeichnisses durch Docker

```
docker inspect -f {{.Mounts}} volumetest
[{volume 8fb[...]16d6a /var/lib/docker/volumes/8fb[..]6d6a/_data /data local  true }]
```

```
→ docker volume ls
```

```
docker run -it --name volumetest -v /home/frank/cdata:/data debian:stretch /bin/bash
```

Kann nicht im Dockerfile definiert werden (Sicherheit, Portabilität)

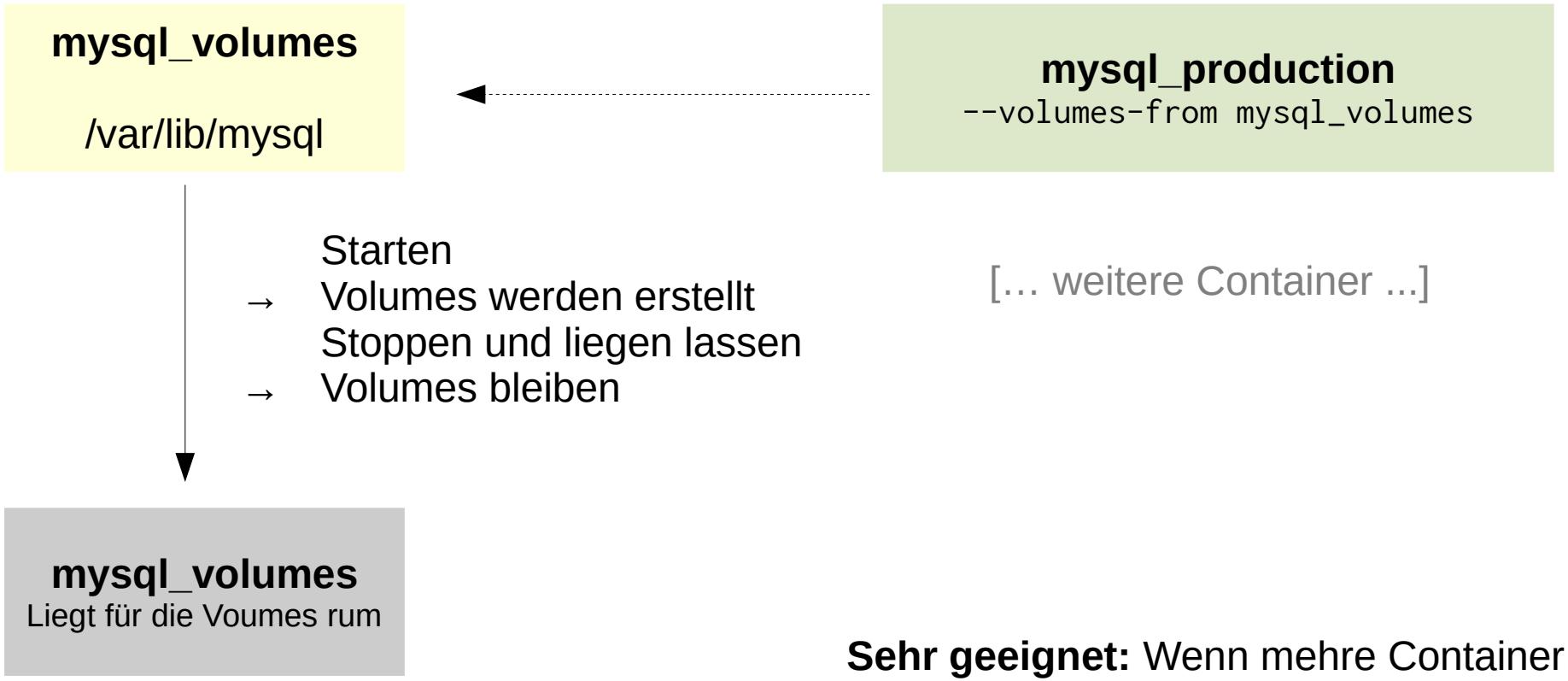
Das Verzeichnis *data im Container kann persistent Daten speichern. Diese landen im Dateisystem des Hosts im Verzeichnis* ***/home/frank/cdata/***

Daten persistent speichern: Datencontainer

Gestoppter Container mit Volumes: Weitere Container können gestartet werden mit

`--volumes-from container_name`

Solange der Volume-Container nicht mit der Option -v gelöscht wird, bleiben die Daten auch bei neuen Containern persistent.



Sehr geeignet: Wenn mehre Container Gemeinsame Datenhaltung betreiben!

Netzwerk 1: Docker macht Netz

```
docker run php:7.3.3-apache -d --name webserver -v /root/docroot:/var/www/html
```

php:7.3.3-apache
mit Volume für Content

`docker inspect webserver`

`→ NetworkSettings, IP: 172.0.0.2`

Internes Netz, von Docker gemanagt

Proof of Concept:

Auf dem Dockerhost funktioniert:
`ping 172.0.0.2`

Von Remote erreicht man den Apachen mit
`ssh -L8080:172.17.0.2:80 fsch@docker.ua25.de`

Und im Browser `http://localhost:8080`

Aber: Apache ist nicht unter

http://docker.ua25.de

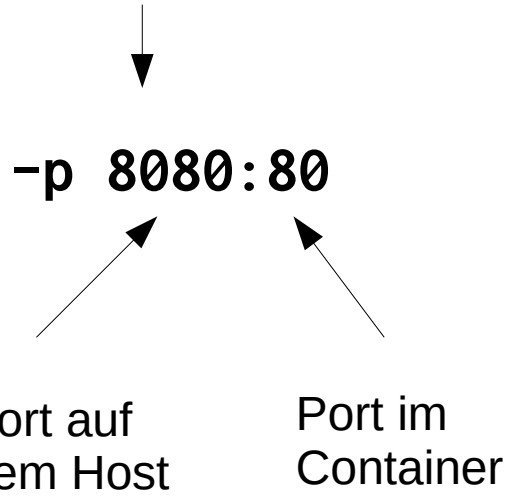
erreichbar.

Wir haben einen „sinnlosen“ Webserver im Container...

Netzwerk 2: Ports durchreichen

```
docker run php:7.3.3-apache -d --name webserver -p 8080:80 \
                    -v /root/docroot:/var/www/html
```

php:7.3.3-apache
mit Volume für Content



Jetzt: Apache ist nicht unter

http://docker.ua25.de:8080

erreichbar.

Wir haben einen Webserver mit PHP 7.3.3 im Container, dessen Document Root vom Host aus befüllt werden kann.

Das ist zum Testen super, weil man so die PHP Version in Sekunden wechseln kann.

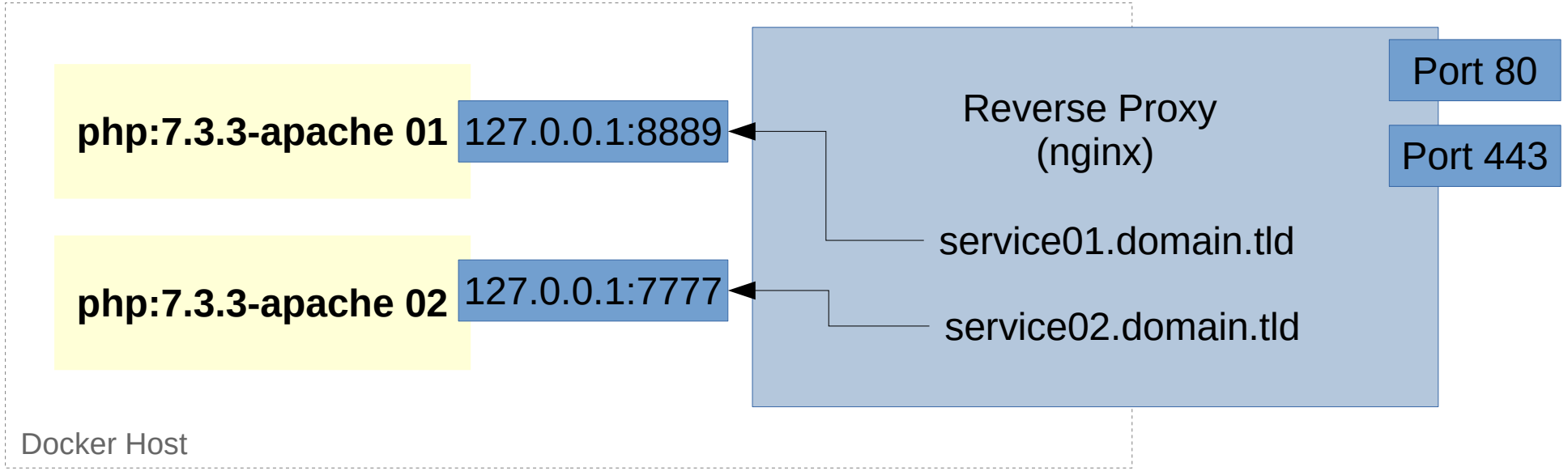
Netzwerk 3: Portkonflikte

```
docker run php:7.3.3-apache -d --name webserver -p 80:80 \
                    -v /root/docroot:/var/www/html
```

```
docker: Error response from daemon: driver failed programming external connectivity on endpoint
webserver (2549ec00cc14b151e04ff519fb8fc9246b29055cd583e1b76f5f3a26e06c1b48): Error starting userland
proxy: listen tcp 0.0.0.0:80: bind: address already in use.
```

Weil… Auf Port 80 des Docker hosts läuft schon **nginx**

Lösung dieses Poblems: **Reverse Proxy,** entweder auf dem Host oder in einem Docker Container



Am besten macht man das alles nicht per Hand sondern mit

docker-compose

- Automatische Erzeugung von einem oder mehreren Containern
- Konfiguration im einem yaml File
  - Definition von Volumes
  - Netzwerke
  - ….
- Übergabe von Umgebungsvariablen an gestartetet Container, diese können z.B. vom Entrypoint Skript ausgewertet werden
  - Autoconfig
  - Steuerung des Verhaltens

Docker-compose: Beispiel mrbs Docker App für lmn 6.2

```
version: '2'
services:
  db:
    container_name: mariadb
    restart: always
    image: mariadb:latest
    environment:
      MYSQL_ROOT_PASSWORD: hoonee0Uuch8bier9po2
    volumes:
      - ./mariadb-data:/var/lib/mysql
  mrbs1:
    depends_on:
      - db
    container_name: raumbuchung
    restart: always
    image: linuxmuster/mrbs:latest
    environment:
        MRBS_DB_HOST: db
        MRBS_DB_PORT: 3306
        MRBS_DB_USER: mrbsdbuser
        MRBS_DB_PASSWORD: gequ8ao3geeNgeecoh4a
        MRBS_DB_NAME: mrbs_raumbuchung
        MYSQL_ROOT_PASSWORD: hoonee0Uuch8bier9po2
    ports:
      - "127.0.0.1:7777:80"
    volumes:
      - ./config/raumbuchung.inc.php:/var/www/html/config.inc.php
      - ./dbdumps/:/var/linuxmuster-mrbs
```

Steuerung...

```
docker-compose up

docker-compose up -d

docker-compose down
```