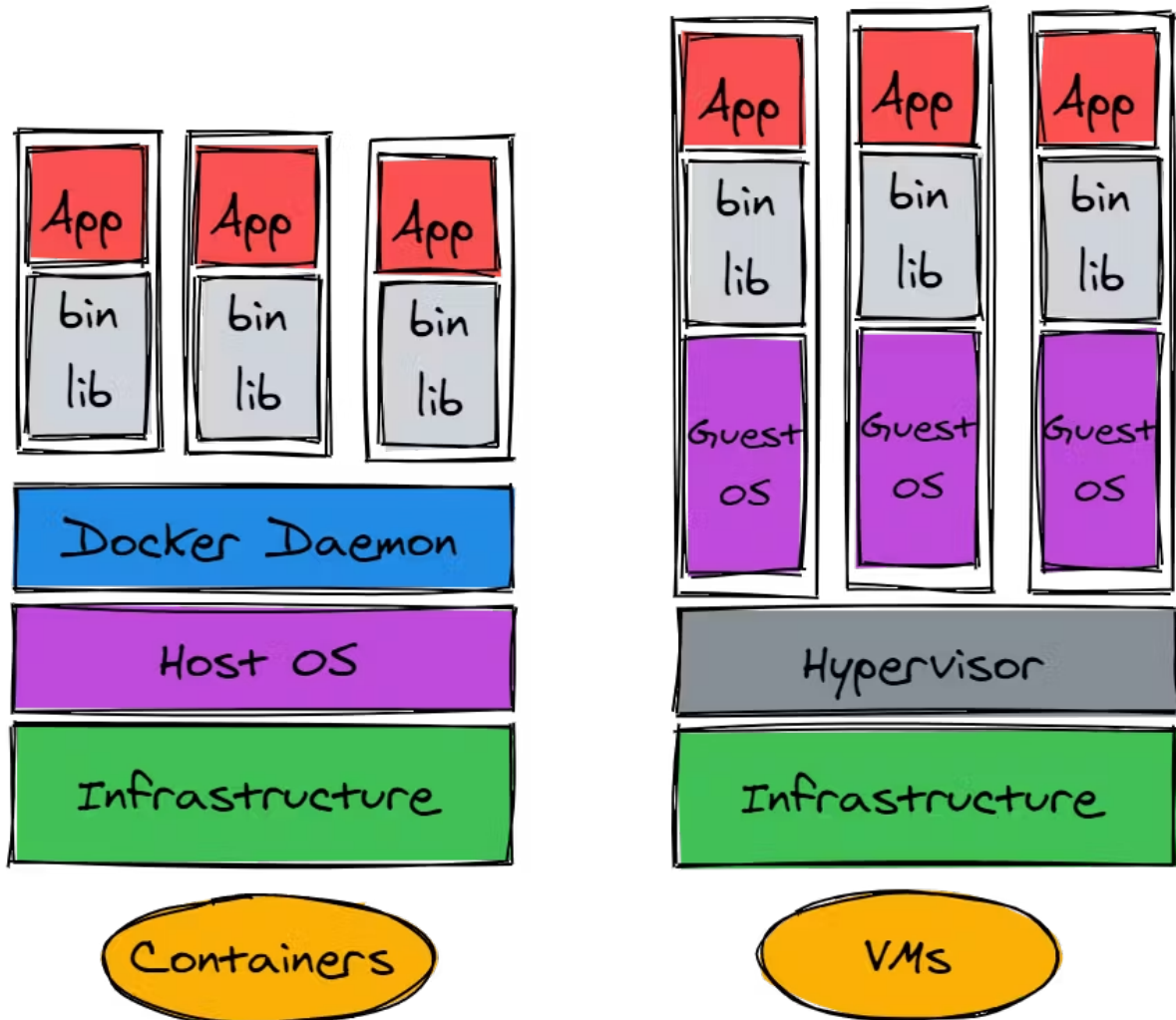


# M347 Dokumentation

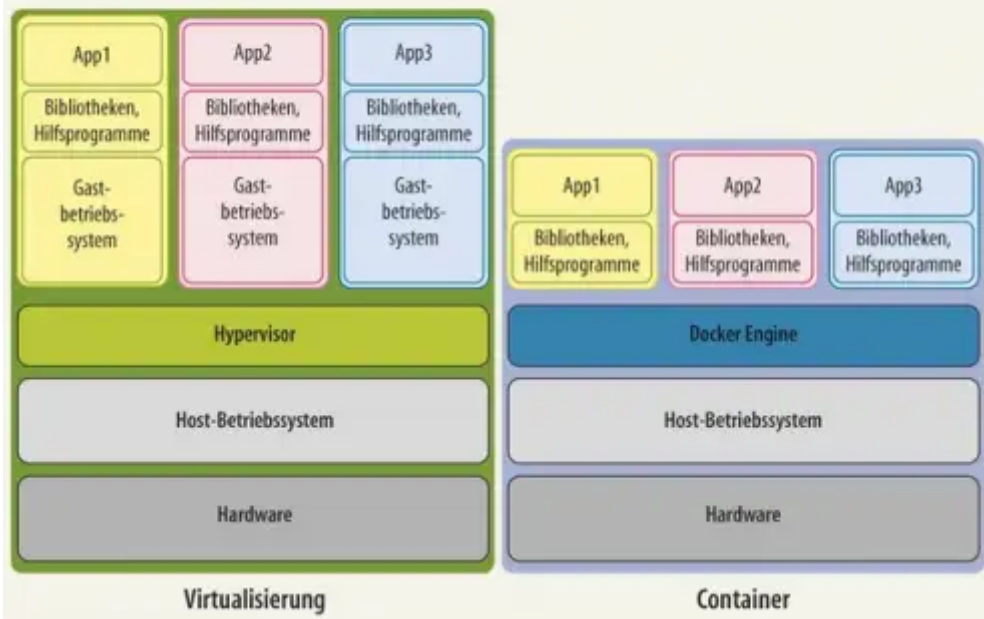
## Grundlagen

**Containerisierung** ist sozusagen die Virtualisierung mit Hilfe von **Containern**. Die einzelnen Container sind laufende Rechenumgebungen.



## Container vs. virtuelle Maschinen

Virtuelle Maschinen haben einen deutlich höheren Ressourcenbedarf als Container, da jede VM ein komplettes Betriebssystem booten muss. Allerdings kann in Containern kein anderes Betriebssystem laufen als auf dem Host.

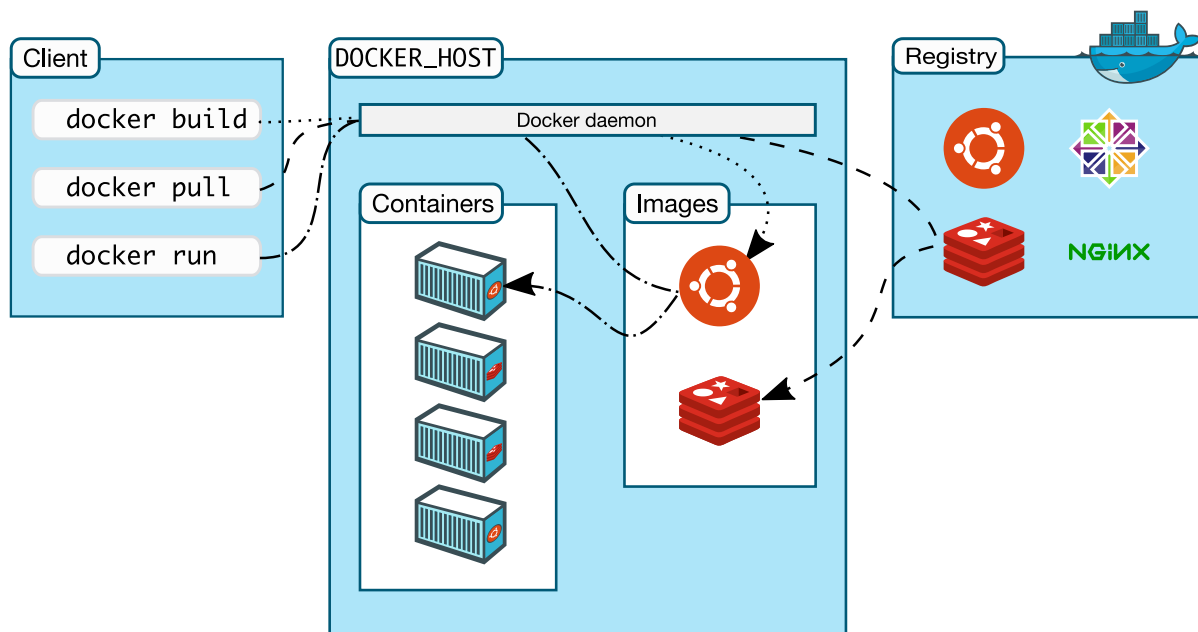


Wenn wir einen Container starten haben wir einen bestimmten Prozess am Laufen, bei einer VM muss immer ein ganzes Betriebssystem laufen. Hauptunterschied ist also, dass Container weniger Ressourcen benötigen. Ebenfalls sind Container auf jedem Betriebssystem verfügbar. Docker engine übernimmt die Verteilung der Ressourcen.

Hier einige wichtige Begriffe

Begriff	Definition
Image	Schreibgeschützte Vorlage mit dem Speicherabbild eines Containers.
Container	Aktive Instanz eines Images.
Layer	Teil eines Images. Enthält einen Befehl oder eine Datei, die dem Image hinzugefügt wurde.
Repository	Satz gleichnamiger Images mit verschiedenen Tags, zumeist Versionen.
Registry	Verwaltet Repositories. z.B. DockerHub
Dockerfile	Textdatei mit allen Befehlen, um ein Image zusammenzustellen.

Der Client ist z.B. eine Konsole. Das Betriebssystem kann als DOCKER\_HOST bezeichnet werden. Im DOCKER\_HOST hat der Docker daemon das Sagen. Registry ist vergleichbar mit Git wo man Images pusht und pullt.



## Konzepte

---

### Image

Ein Image kann als eine Art von "virtuellem Container" betrachtet werden, der es ermöglicht, Anwendungen oder Betriebssysteme auf verschiedenen Computern oder Servern auszuführen, ohne dass es notwendig ist, jedes Mal eine separate Installation durchzuführen. Das bedeutet, dass ein Image in einem Docking-System verwendet werden kann, um eine Anwendung oder ein Betriebssystem schnell bereitzustellen, ohne dass es notwendig ist, es auf jedem einzelnen Computer oder Server zu installieren.

### Microservices

Jeder Container hat eine spezifische Aufgabe. Man spricht deshalb von Microservices.

### Orchestrierung

Mithilfe der Container-Orchestrierung werden Deployment, Management, Skalierung und Vernetzung von Containern automatisiert. Heisst es hilft Container zu verwalten.

## Container ausführen

---

### Image herunterladen

```
docker pull ubuntu:latest
```

## Container starten

```
docker run -it --name my-ubuntu-container ubuntu:latest
```

Um zu überprüfen, **welche Container gestartet** sind können Sie `docker ps` (2. Terminal) verwenden:

```
docker ps
```

## Container stoppen

```
docker stop my-ubuntu-container
```

## Container Löschen

`docker rm` löscht einen gestoppten Container, das Image bleibt jedoch vorhanden.

```
docker rm my-ubuntu-container
```

## Image löschen

```
docker rmi ubuntu:latest
```

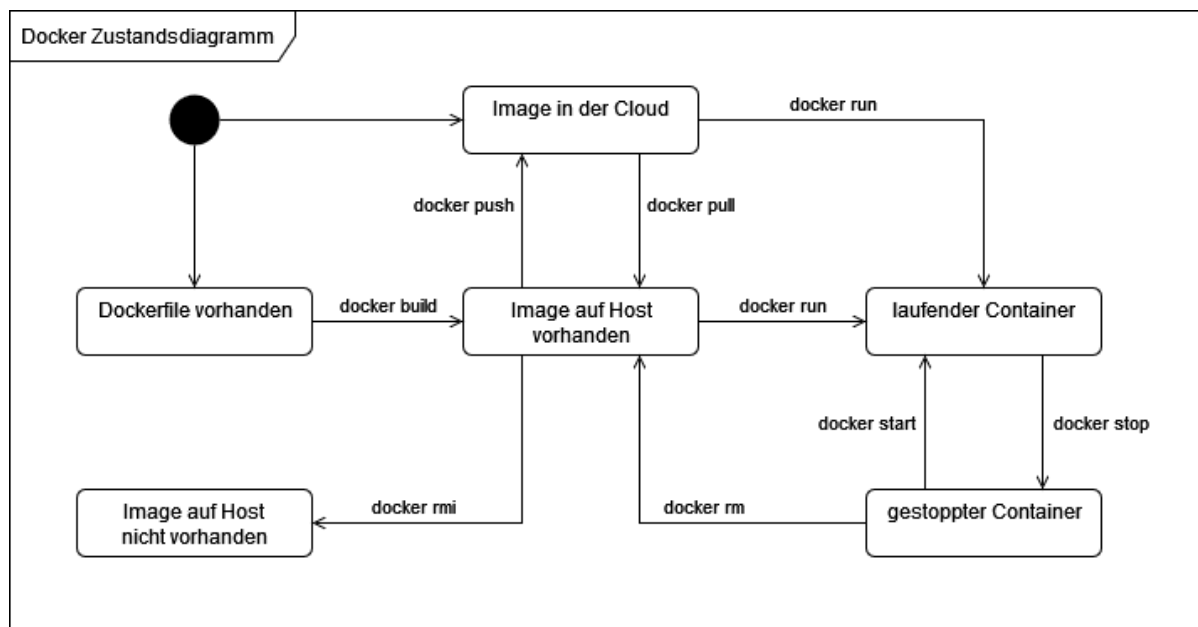
## Image Namen

Docker-Image-Namen setzen sich aus drei Teilen zusammen:

```
source/imagename:tag
```

- **source** gibt den Namen der Organisation (oder Person) an, die das Image erstellt hat, z.B. `docker`
- **imagename** der Name des Images, z.B. `getting-started`
- **tag** die Versionsnummer des Images, z.B. `22.04`

Wird keine source angegeben, nimmt docker an, dass eines der offiziellen Dockerimages gemeint ist. Wird kein tag angegeben, wird automatisch das tag `latest` verwendet.



## Docker pull

```
docker pull nginx
```

## Docker stop

```
docker stop my-nginx-container
```

## Docker start

```
docker start my-nginx-container
```

## Docker rm

```
docker rm my-nginx-container
```

## Docker rmi

Zweck: Ein Image auf dem Host wird gelöscht. Es dürfen keine abgeleiteten Container von diesem Image vorhanden sein (weder laufend noch gestoppt)

```
docker rmi nginx
```

## Docker ps

Zeigt laufende Container, wenn -a auch noch gestoppte.

```
docker ps -a
```

## Docker images

```
docker images
docker image ls
```

# Portweiterleitungen

Stellt ein Container einen Serverdienst über einen bestimmten Port zur Verfügung kann dieser mit einem anderen Port (oder auch dem gleichen) Port auf dem Host verknüpft werden.

Läuft beispielsweise im Container eine Webanwendung auf Port 80 (http), lässt sich dieser Port beim Start des Containers mit dem Parameter `-p` mit dem Port 80 des Hostrechners so verbinden

```
docker run -p 80:80 <image>
```

Auf dem Host lässt sich somit die Webseite des Containers mit `http://localhost` öffnen

Die Syntax für `-p` lautet

```
-p hostport:containerport
```

Für Webdienste üblich ist z.B. Port 8080:

```
docker run -p 8080:80 <image>
```

Die Containerwebseite ist dann unter `http://localhost:8080` erreichbar.

## Aufgabe Portweiterleitung

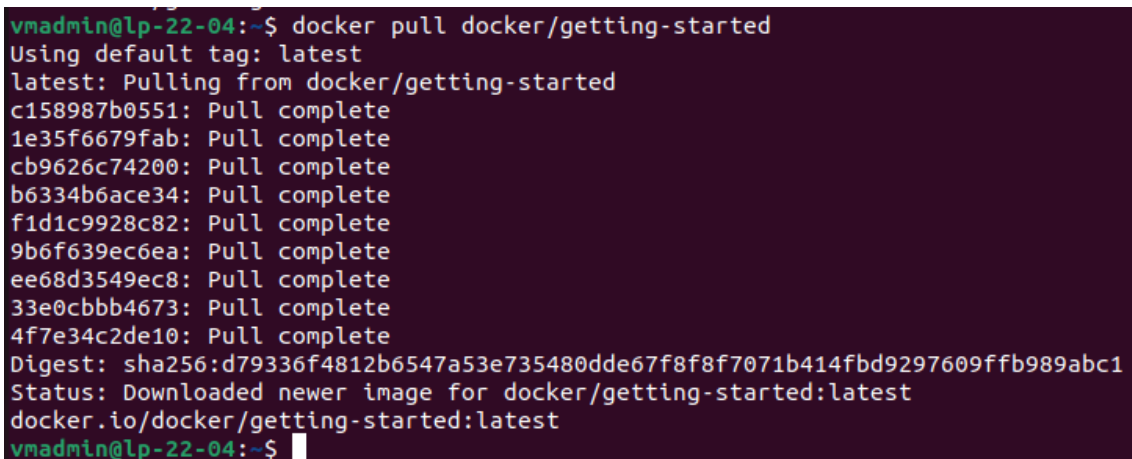
Auf welchem Port läuft die Webseite im Container?

Auf dem Container-Port

Starten Sie einen Container aus `getting-started`, der auf dem Host auf <http://localhost:8080> erreichbar ist

1. "getting-started" herunterladen

```
docker pull docker/getting-started
```

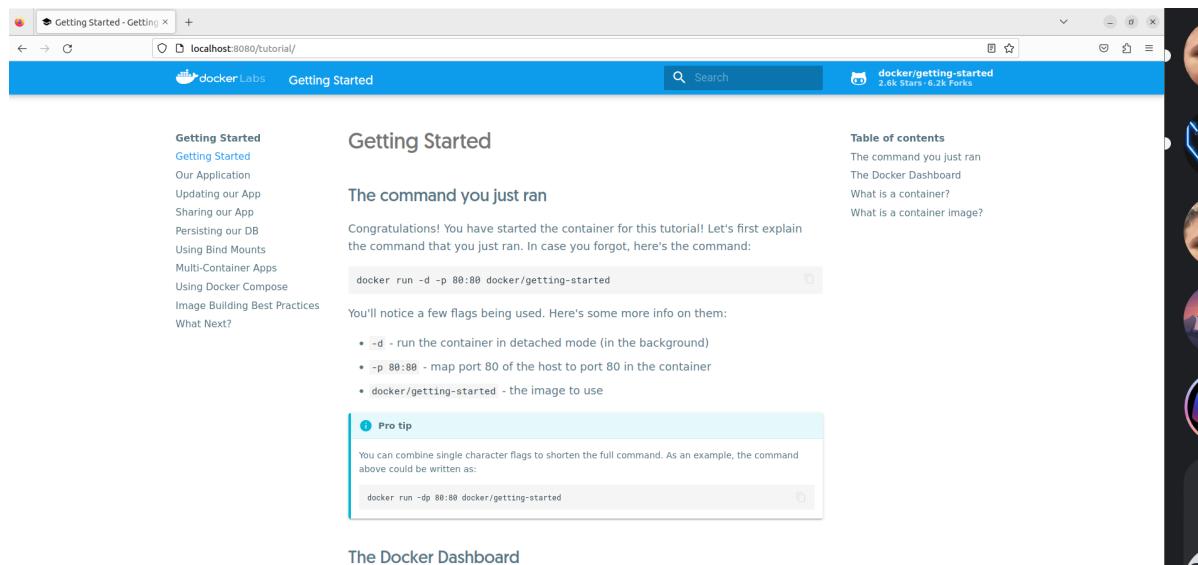


```
vmadmin@lp-22-04:~$ docker pull docker/getting-started
Using default tag: latest
latest: Pulling from docker/getting-started
c158987b0551: Pull complete
1e35f6679fab: Pull complete
cb9626c74200: Pull complete
b6334b6ace34: Pull complete
fd1c9928c82: Pull complete
9b6f639ec6ea: Pull complete
ee68d3549ec8: Pull complete
33e0cbbb4673: Pull complete
4f7e34c2de10: Pull complete
Digest: sha256:d79336f4812b6547a53e735480dde67f8f8f7071b414fbd9297609ffb989abc1
Status: Downloaded newer image for docker/getting-started:latest
docker.io/docker/getting-started:latest
vmadmin@lp-22-04:~$
```

2. Container starten. Mit `-p` den Host-Port 8080 und den standardmäßigen Container-Port von 80 angeben

```
docker run -p 8080:80 docker/getting-started
```

```
vmadmin@lp-22-04:~$ docker run -p 8080:80 docker/getting-started
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2023/04/23 15:12:56 [notice] 1#1: using the "epoll" event method
2023/04/23 15:12:56 [notice] 1#1: nginx/1.23.3
2023/04/23 15:12:56 [notice] 1#1: built by gcc 12.2.1 20220924 (Alpine 12.2.1_git20220924-r4)
2023/04/23 15:12:56 [notice] 1#1: OS: Linux 5.15.0-58-generic
2023/04/23 15:12:56 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2023/04/23 15:12:56 [notice] 1#1: start worker processes
2023/04/23 15:12:56 [notice] 1#1: start worker process 30
2023/04/23 15:12:56 [notice] 1#1: start worker process 31
```



Beenden Sie am Schluss den Container, löschen Sie ihn und auch das Image.

```
docker stop b967ea729bcd
```

```
docker rm b967ea729bcd
```

```
docker images
```

```
docker rmi 3e4394f6b72f
```

```
vmadmin@lp-22-04:~$ docker rm b967ea729bcd
b967ea729bcd
```

```
vmadmin@lp-22-04:~$ docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
httpd                latest      192d41583429  4 weeks ago   145MB
docker/getting-started latest      3e4394f6b72f  4 months ago  47MB
portainer/portainer-ce latest      5f11582196a4  5 months ago  287MB
hello-world          latest      feb5d9fea6a5  19 months ago 13.3kB
```

```
vmadmin@lp-22-04:~$ docker rmi 3e4394f6b72f
Untagged: docker/getting-started:latest
Untagged: docker/getting-started@sha256:d79336f4812b6547a53e735480dde67f8f7071b414fbd9297609ffb989abc1
Deleted: sha256:3e4394f6b72fccefa2217067a7f7ff84d5d828afa9623867d68fce4f9d862b6c
Deleted: sha256:cdc6440a971be2985ce94c7e2e0c2df763b58a2ced4ecdb944fcd9b13e7a2aa4
Deleted: sha256:041ac26cd02fa81c8fd73cc616bdeee180de3fd68a649ed1c0339a84cdf7a7c3
Deleted: sha256:376baf7ada4b52ef4c110a023fe7185c4c2c090fa24a5cbd746066333ce3bc46
Deleted: sha256:d254c9b1e23bad05f5cde233b5e91153a0540fa9a797a580d8a360ad12bf63a9
Deleted: sha256:dd5c79fa9b6829fd08ff9943fc1d66bebbba3e04246ba394d57c28827fed95af0
Deleted: sha256:8d812a075abf60a83013c37f49058c220c9cdf390266952126e7e60041b305dc
Deleted: sha256:ff1787ee3dcae843dc7dd1933c49350fc84451cf19ed74f4ea72426e17ee7cd1
Deleted: sha256:85ebd294be1553b207ba9120676f4fd140842348ddf1bb5f7602c7a8401f0a13
Deleted: sha256:ded7a220bb058e28ee3254fbba04ca90b679070424424761a53a043b93b612bf
```

```
vmadmin@lp-22-04:~$ docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
httpd                latest      192d41583429 4 weeks ago   145MB
portainer/portainer-ce latest      5f11582196a4 5 months ago  287MB
hello-world          latest      feb5d9fea6a5 19 months ago 13.3kB
```

## Volumes

Sollen Speicherort der Daten, die in einem Container verwendet werden, ändern. Sie werden nämlich standardmässig im Container gespeichert, was heisst, dass bei einer Neuinstallation des Containers alle Daten weg sind.

Hier geht es darum, wo ein Container seine Daten speichert. Das [Docker-Zustandsdiagramm](#) legt nahe, dass Daten die in einem laufenden Container gespeichert sind, erhalten bleiben, wenn der Container gestoppt und wieder gestartet wird, nicht jedoch wenn ein Container gelöscht und wieder neu erzeugt wird.

Um das zu demonstrieren verwenden wir einen Container für den mariadb-Server: Wir erzeugen einen Container (-d bewirkt, dass der Container im Hintergrund läuft, ausserdem muss ein root-Passwort gesetzt werden):

```
docker run -d --name mariadb-test -e MYSQL_ROOT_PASSWORD=geheim mariadb:latest
```

Öffnen Sie eine root-Shell innerhalb des Containers mit

```
docker exec -it mariadb-test /bin/bash
```

dann legen wir mit touch eine neue Datei abc.txt an

```
vmadmin@ubuntu:~$ docker exec -it mariadb-test /bin/bash
root@5de9ed33a981:/# touch abc.txt
root@5de9ed33a981:/# ls
abc.txt  bin ...
root@5de9ed33a981:/# exit
```

wenn man jetzt diese Commands ausführt sind die Daten immer noch vorhanden (bzw. die Datei)

```
docker stop mariadb-test
docker start mariadb-test
```



wenn man den Container allerdings löscht und neu erstellt ist es nicht mehr vorhanden.

```
docker stop mariadb-test
docker rm mariadb-test
docker run -d --name mariadb-test -e MYSQL_ROOT_PASSWORD=geheim mariadb:latest
```

## Unbenannte Volumes

```
docker run -d --name mariadb-test -e MYSQL_ROOT_PASSWORD=geheim mariadb
```

Um herauszufinden wo nun die Daten aus /var/lib/mysql gelandet sind können Sie das Kommando

```
docker inspect -f '{{.Mounts}}' mariadb-test
```

Das Resultat daraus wäre:

```
vmadmin@ubuntu:~/bsp-apache-php$ docker inspect -f '{{.Mounts}}' mariadb-test
[{"volume": "752507751a42f0c781b96adacb4a3d73bdbbf2184ead3bd4874b4b5f065ee4eb", "source": "/var/lib/docker/volumes/752507751a42f0c781b96adacb4a3d73bdbbf2184ead3bd4874b4b5f065ee4eb/_data", "target": "/var/lib/mysql", "type": "local", "readOnly": true}]
vmadmin@ubuntu:~/bsp-apache-php$
```

Wenn beim Erstellen des Containers also nichts eingestellt wird, liegt das Verzeichnis in einem zufällig benannten Unterverzeichnis von /var/lib/docker/volumes des Hostrechners. Die zufällige Bezeichnung verunmöglicht eine praktikable Verwaltung von Volumes nahezu. Sollte z.B. ein Volume gelöscht werden, müsste der vollständige Verzeichnisname angegeben werden: `docker volume rm 7525077...`

Eine Wiederverwendung des Volumes nach einem Löschen des Containers ist ebenfalls nicht möglich, da bei jedem Neuerstellen eines Containers eine neue zufällige Bezeichnung des Volumes erzeugt wird, d.h. die Daten sind nach einem Upgrade auf eine neuere Version nicht mehr vorhanden.

## Benannte Volumes

```
docker run -d --name mariadb-test2 -v myvolume:/var/lib/mysql -e
MYSQL_ROOT_PASSWORD=geheim mariadb
```

Auf der Linux-Konsole sieht dies so aus:

```
vmadmin@ubuntu:~/bsp-apache-php$ docker run -d --name mariadb-test2 -v
myvolume:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=geheim mariadb
2ccb55f675a5867efeeffaa8bfd02e896cf0e0e1b66fe098465d6d7cc3e6a9bc6
vmadmin@ubuntu:~/bsp-apache-php$ docker inspect -f '{{.Mounts}}' mariadb-test2
[{"volume": "myvolume", "source": "/var/lib/docker/volumes/myvolume/_data", "target": "/var/lib/mysql", "type": "local", "readOnly": true}]
```

Die Syntax für ein Benanntes Volume ist also:

```
-v volumename:containerverzeichnis
```

## Volumes in eigenen Verzeichnissen

Anstelle eines Namens für das Hostvolume kann auch ein Verzeichnis angegeben werden die Syntax dazu lautet:

```
-v hostverzeichnis:containerverzeichnis
```

Damit wird das Volume ganz aus der Dockerumgebung herausgelöst und kann an einer beliebigen Stelle platziert werden:

```
mkdir /home/vmadmin/databases  
docker run -d --name mariadb-test3 -v /home/vmadmin/databases:/var/lib/mysql -e  
MYSQL_ROOT_PASSWORD=geheim mariadb
```

## Aufgabe Volumes

- Starten Sie drei Container aus dem mariadb-Image und verbinden Sie die Containerports mit den beiden Hostports 3306, 3307 und 3308.
- Das Rootpasswort kann mit dem Parameter -e angegeben werden: -e MYSQL\_ROOT\_PASSWORD=geheim
- Verwenden Sie im ersten Container ein unbenanntes Volumen für die Datenbanken
- Verwenden Sie im zweiten Container ein benanntes Volumen
- Verwenden Sie im dritten Container ein Volumen, welches auf dem Hostrechner in /home/vmadmin/mysql liegt. (Der Ordner muss zuerst mit mkdir erstellt werden)
- Lassen Sie sich in allen Fällen den Inhalt des Volumes mit ls -l anzeigen
- Installieren Sie auf dem Host den mysql-client mit sudo apt install mariadb-client
- Verbinden Sie sich bei beiden Containern mit mysql -u root -p -P 3306 -h 127.0.0.1 mysql -u root -p -P 3307 -h 127.0.0.1 mysql -u root -p -P 3308 -h 127.0.0.1 und legen Sie je eine neue Datenbank an. (-h 127.0.0.1 ist nötig um TCP zu erzwingen)
- Stoppen und löschen Sie nun die Container (Simulation eines Upgrades vom mariadb)
- Erstellen Sie drei neue Container mit den selben Volumes und Ports wie vorher.
- Überprüfen Sie ob die zuvor erstellten Datenbanken noch vorhanden sind.
- Löschen Sie alle Container, Images und Volumes

```

vmadmin@lp-22-04:~$ docker run -d -p 3306:3306 --name mariadb-test -e MYSQL_ROOT_PASSWORD=geheim mariadb
be33a9eff45370da857dfff76c0ca2119bf0824e7fb892313e856b9968d52356b
vmadmin@lp-22-04:~$ docker run -d -p 3306:80C --name mariadb-test -e MYSQL_ROOT_PASSWORD=geheim mariadb

vmadmin@lp-22-04:~$ mysql -u root -p -P 3306 -h 127.0.0.1
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 10.11.2-MariaDB-1:10.11.2+maria-ubu2204 mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> Ctrl-C -- exit!
Aborted
vmadmin@lp-22-04:~$ docker run -d -p 3307:3306 --name mariadb-test2 -v myvolume:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=geheim mariadb
d755c86950253e48caa284c1a6f1f4210879d5c1a3a1db62ef2e745d44bd72e5
vmadmin@lp-22-04:~$ mysql -u root -p -P 3307 -h 127.0.0.1
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 10.11.2-MariaDB-1:10.11.2+maria-ubu2204 mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> Ctrl-C -- exit!
Aborted
vmadmin@lp-22-04:~$ docker run -d -p 3308:3306 --name mariadb-test3 -v /home/vmadmin/mysql:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=geheim mariadb
c1e467fc8befb30d4de65ae441769d23c58583629d94ed60119b3e8b3ad128438
vmadmin@lp-22-04:~$ mysql -u root -p -P 3308 -h 127.0.0.1
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 10.11.2-MariaDB-1:10.11.2+maria-ubu2204 mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> Ctrl-C -- exit!
Aborted
vmadmin@lp-22-04:~$ docker inspect -f '{{.Mounts}}' mariadb-test3
[{{bind /home/vmadmin/mysql /var/lib/mysql true rprivate}}]
vmadmin@lp-22-04:~$

```

```

vmadmin@lp-22-04:~$ mysql -u root -p -P 3307 -h 127.0.0.1
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 4
Server version: 10.11.2-MariaDB-1:10.11.2+maria-ubu2204 mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> CREATE DATABASE TEST;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> exit
Bye
vmadmin@lp-22-04:~$ mysql -u root -p -P 3308 -h 127.0.0.1
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 4
Server version: 10.11.2-MariaDB-1:10.11.2+maria-ubu2204 mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> CREATE DATABASE TEST;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> exit
Bye

```

## Netzwerke

Hier geht es darum wie mehrere Dockercontainer untereinander kommunizieren können. Beispielsweise muss ein Web-Container mit einem Datenbank-Container kommunizieren können und an seine Daten zu kommen.

vorhandenen Netzwerke:

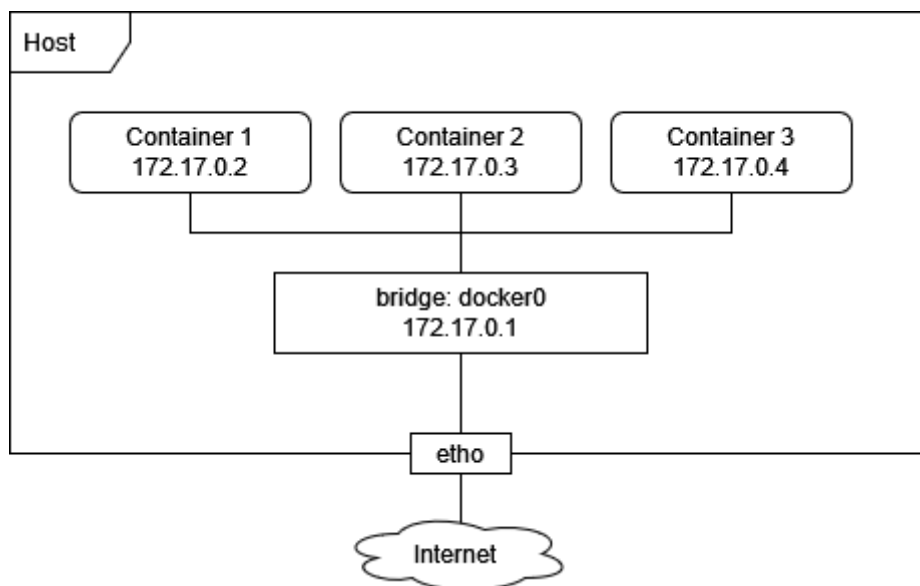
```
docker network ls
```

```
vmadmin@ubuntu:~$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
65593a9ebb3b	bridge	bridge	local
364521a9aa2	host	host	local
c69c18f0a974	none		

## Standardnetzwerk

Das Netzwerk mit dem Namen bridge ist das Standardnetzwerk und wird verwendet wenn nichts anderes angegeben wird. Die Netzwerkarchitektur lässt sich wie folgt darstellen:



Folgendes um diese Architektur nachvollziehen zu können:

```
vmadmin@ubuntu:~$ ip addr
```

```
...
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 02:42:c3:53:04:66 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
...
```

```
docker network inspect bridge
```

# Eigene Netzwerke

Alle Container landen standardmässig im selben Netzwerk, dem bridge-Netzwerk. Dies ist aus sicherheitstechnischen Gründen nicht ideal, wenn unterschiedliche Anwendungen voneinander isoliert sein sollen. Es lassen sich deshalb eigene Netzwerke definieren und diese den Containern zuordnen.

```
docker network create \
  --driver=bridge \
  --subnet=10.10.10.0/24 \
  --gateway=10.10.10.1 \
  my_net
```

Ein Container kann nun beim Start diesem Netzwerk zugeordnet werden, indem man dies ausführt:

```
docker run -it --name ubuntu_2 --network=my_net ubuntu:latest
```

Um die IP anzuzeigen:

```
docker network inspect my_net
```

Die IP-Adresse für den Container wird dabei von docker via DHCP aus dem definierten Netzwerk vergeben. Alternativ kann eine fixe IP-Adresse beim Start des Containers angegeben werden.

```
docker run -it --name ubuntu_2 --ip="10.10.10.10" --network=my_net ubuntu:latest
```

Als nächstes soll nun der weiteroben dem Netzwerk bridge zugeordnete Container ubuntu\_1 dem Netzwerk my\_net zugeordnet werden. Dazu trennen wir ihn zuerst von bridge mit

```
docker network disconnect bridge ubuntu_1
```

anschliessend wird er zu my\_net hinzugefügt und neu gestartet

```
docker network connect my_net ubuntu_1
docker start -i ubuntu_1
```

Um zu überprüfen, ob die beiden Container tatsächlich mit einander kommunizieren können, installieren wir auf ubuntu\_1 das Paket iputils-ping:

```
apt update
apt install iputils-ping
```

Dabei ist es nicht einmal nötig die IP-Adresse von ubuntu\_2 zu kennen, da man auch den Namen direkt verwenden kann

```
root@5fe876094647:/# ping ubuntu_2
PING ubuntu_2 (10.10.10.2) 56(84) bytes of data.
64 bytes from ubuntu_2.my_net (10.10.10.2): icmp_seq=1 ttl=64 time=0.099 ms
...
```

Netzwerk löschen mit:

```
docker network rm my_net
```

## Aufgabe Netzwerke

- Definieren Sie das docker-Netzwerk 192.168.100.0/24 mit Gateway 192.168.100.1

```
docker network create --driver=bridge --subnet=192.168.100.0/24 --
gateway=192.168.100.1 my-network
```

```
vmadmin@lp-22-04:~$ docker network create --driver=bridge --subnet=192.168.100.0/24 --gateway=192.168.100.1 my-network
1707ddf1604e56675891fd42272107ce608e48d46c40c144cf42243794f0ffe2
vmadmin@lp-22-04:~$ docker network ls
NETWORK ID        NAME                DRIVER              SCOPE
c293df2a4b65     bridge             bridge             local
5a558bb27117     host               host               local
1707ddf1604e     my-network         bridge             local
8a532b048872     none              null               local
```

```
vmadmin@lp-22-04:~$ docker network inspect my-network
[
  {
    "Name": "my-network",
    "Id": "1707ddf1604e56675891fd42272107ce608e48d46c40c144cf42243794f0ffe2",
    "Created": "2023-04-24T12:08:28.061428336+02:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "192.168.100.0/24",
          "Gateway": "192.168.100.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {},
    "Options": {},
    "Labels": {}
  }
]
vmadmin@lp-22-04:~$
```

- Starten Sie 2 ubuntu-Container und ordnen Sie diese dem oben erstellten Netzwerk zu: Der erste Container soll seine IP-Adresse via DHCP erhalten. Der zweite soll die IP-Adresse 192.168.100.100 erhalten

```
docker run -it --name ubuntu_1 --network=my-network ubuntu:latest
```

```
docker run -it --name ubuntu_2 --ip="192.168.100.100" --network=my-network
ubuntu:latest
```

```
vmadmin@lp-22-04:~$ docker network inspect my-network
[
  {
    "Name": "my-network",
    "Id": "1707ddf1604e56675891fd42272107ce608e48d46c40c144cf42243794f0ffe2",
    "Created": "2023-04-24T12:08:28.061428336+02:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "192.168.100.0/24",
          "Gateway": "192.168.100.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "2382e67477889db674ffd9f945ee2073c001dc20bd7730cfa7ecd68155a21311": {
        "Name": "ubuntu_1",
        "EndpointID": "e393b19b9b50c3605106cdb2188ed5e294deb3a6e185b1eef8306116fb635efa",
        "MacAddress": "02:42:c0:a8:64:02",
        "IPv4Address": "192.168.100.2/24",
        "IPv6Address": ""
      },
      "86a07845c4c83961f531c044cee97400a89ab42c38ed39be5bd086c4956fb66a": {
        "Name": "ubuntu_2",
        "EndpointID": "532ea219db9f0474f1da4eb2d376103e365a3559eb6fac4fd48585a20ae657b0",
        "MacAddress": "02:42:c0:a8:64:64",
        "IPv4Address": "192.168.100.100/24",
        "IPv6Address": ""
      }
    },
    "Options": {},
    "Labels": {}
  }
]
vmadmin@lp-22-04:~$
```

Die Container sollen sich gegenseitig anpingen können

```
apt update
apt install iputils-ping
apt install net-tools
```

```
ping ubuntu_2
```

```
root@49ea249db469:/# ping ubuntu_2
PING ubuntu_2 (192.168.100.100) 56(84) bytes of data.
64 bytes from ubuntu_2.my-network (192.168.100.100): icmp_seq=1 ttl=64 time=0.066 ms
64 bytes from ubuntu_2.my-network (192.168.100.100): icmp_seq=2 ttl=64 time=0.136 ms
64 bytes from ubuntu_2.my-network (192.168.100.100): icmp_seq=3 ttl=64 time=0.100 ms
64 bytes from ubuntu_2.my-network (192.168.100.100): icmp_seq=4 ttl=64 time=0.069 ms
64 bytes from ubuntu_2.my-network (192.168.100.100): icmp_seq=5 ttl=64 time=0.074 ms
64 bytes from ubuntu_2.my-network (192.168.100.100): icmp_seq=6 ttl=64 time=0.092 ms
64 bytes from ubuntu_2.my-network (192.168.100.100): icmp_seq=7 ttl=64 time=0.116 ms
64 bytes from ubuntu_2.my-network (192.168.100.100): icmp_seq=8 ttl=64 time=0.081 ms
^C
--- ubuntu_2 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7141ms
rtt min/avg/max/mdev = 0.066/0.091/0.136/0.023 ms
```

```

root@9f9482da86aa:/# ping ubuntu_1
PING ubuntu_1 (192.168.100.2) 56(84) bytes of data.
64 bytes from ubuntu_1.my-network (192.168.100.2): icmp_seq=1 ttl=64 time=0.079 ms
64 bytes from ubuntu_1.my-network (192.168.100.2): icmp_seq=2 ttl=64 time=0.077 ms
64 bytes from ubuntu_1.my-network (192.168.100.2): icmp_seq=3 ttl=64 time=0.077 ms
64 bytes from ubuntu_1.my-network (192.168.100.2): icmp_seq=4 ttl=64 time=0.078 ms
64 bytes from ubuntu_1.my-network (192.168.100.2): icmp_seq=5 ttl=64 time=0.115 ms
64 bytes from ubuntu_1.my-network (192.168.100.2): icmp_seq=6 ttl=64 time=0.120 ms
64 bytes from ubuntu_1.my-network (192.168.100.2): icmp_seq=7 ttl=64 time=0.098 ms
64 bytes from ubuntu_1.my-network (192.168.100.2): icmp_seq=8 ttl=64 time=0.087 ms
^C
--- ubuntu_1 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7167ms
rtt min/avg/max/mdev = 0.077/0.091/0.120/0.016 ms

```

Stoppen und Löschen sie alles wieder

```

vmadmin@lp-22-04:~$ docker stop ubuntu_1
ubuntu_1
vmadmin@lp-22-04:~$ docker stop ubuntu_2
ubuntu_2
vmadmin@lp-22-04:~$ docker rm ubuntu_1
ubuntu_1
vmadmin@lp-22-04:~$ docker rm ubuntu_2
ubuntu_2
vmadmin@lp-22-04:~$ docker network rm my-network
my-network

```

```
docker network rm my-network
```

## Einrichten einer Wordpress-Applikation

Definieren Sie das docker-Netzwerk wp\_net 192.168.200.0/24 mit Gateway 192.168.200.1

```
docker network create --driver=bridge --subnet=192.168.100.0/24 --gateway=192.168.100.1 wp_net
```

```

vmadmin@lp-22-04:~$ docker network create --driver=bridge --subnet=192.168.100.0/24 --gateway=192.168.100.1 wp_net
b3bfdb6a6a4a8e0c201834a43286de81529ea7427ed19a6d0ffdeecf84e1fe9

```

Starten Sie einen mariadb-Container in diesem Netzwerk:

```

docker run -d --name wp_mariadb --network wp_net -e
MYSQL_ROOT_PASSWORD=strengeheim -e MYSQL_DATABASE=wp -e MYSQL_USER=wpuser -e
MYSQL_PASSWORD=geheim -v wp_dbvolume:/var/lib/mysql mariadb

```

```

vmadmin@lp-22-04:~$ docker run -d --name wp_mariadb --network wp_net -e MYSQL_ROOT_PASSWORD=strengeheim -e MYSQL_DATABASE=wp -e MYSQL_USER=wpuser -e MYSQL_PASSWORD=geheim -v wp_dbvolume:/var/lib/mysql mariadb
mariadb
Unable to find image 'mariadb:latest' locally
latest: Pulling from library/mariadb
74ac377868f8: Already exists
9f8acc28aa1: Pull complete
11b336495e01: Pull complete
20ab1641dd41: Pull complete
eaf8c5c99886: Pull complete
23933549a097: Pull complete
931baaab2c80: Pull complete
f2e86cc8f052: Pull complete
Digest: sha256:9ff479f244cc596aed9794d035a9f352662f2caed933238c533024df64569853
Status: Downloaded newer image for mariadb:latest
5b4d8f4dd833cfa53454cc92106d37b2e3e099f0cf0d8aa8479b9ad45ea0fa2e

```

Starten Sie einen phpmyadmin-Container ebenfalls in diesem Netzwerk:

```

docker run -d --name wp_pma --network wp_net -p 8080:80 -e PMA_HOST=wp_mariadb
phpmyadmin/phpmyadmin

```



```

vmadmin@lp-22-04:~$ docker run -d --name wp_pma --network wp_net -p 8080:80 -e PMA_HOST=wp_mariadb phpmyadmin/phpmyadmin
Unable to find image 'phpmyadmin/phpmyadmin:latest' locally
latest: Pulling from phpmyadmin/phpmyadmin
f1f26f570256: Already exists
ee0a4e40ccac: Pull complete
5ca9fb408faa: Pull complete
5baa808a48ff: Pull complete
6e8d74e4d8ee: Pull complete
fac8e70fcf67: Pull complete
b3b7906fb177: Pull complete
cb4935bbeb83: Pull complete
c9e00ef337e3: Pull complete
cfe495c8d695: Pull complete
dcc3fd107f0c: Pull complete
fe3c587d1f07: Pull complete
677f27d94442: Pull complete
4d778a8cb653: Pull complete
5f0f7b557ecd: Pull complete
6ad259d60f7c: Pull complete
41acd705cbc4: Pull complete
912204d5a7e6: Pull complete
Digest: sha256:ed87921184b59f7d8fc85c6a5f041c22758a4d4419c0ee3bac38eb7e133eae63
Status: Downloaded newer image for phpmyadmin/phpmyadmin:latest
da967c4ab68102cbee7b4b586296a1aebc9ba15d761261ae62c50fd84343ef72

```

Starten Sie nun den wordpress-Container mit:

```

docker run -d --name wp_wordpress --network wp_net -v
wp_htmlvolume:/var/www/html/wp-content -p 8081:80 -e WORDPRESS_DB_HOST=wp_mariadb
-e WORDPRESS_DB_USER=wpuser -e WORDPRESS_DB_NAME=wp -e
WORDPRESS_DB_PASSWORD=geheim wordpress

```

```

vmadmin@lp-22-04:~$ docker run -d --name wp_wordpress --network wp_net -v wp_htmlvolume:/var/www/html/wp-content -p 8081:80 -e WORDPRESS_DB_HOST=wp_mariadb -e WORDPRESS_DB_USER=wpuser -e WORDPRESS_DB_NAME=wp -e WORDPRESS_DB_PASSWORD=geheim wordpress
E=wp -e WORDPRESS_DB_PASSWORD=geheim wordpress
Unable to find image 'wordpress:latest' locally
latest: Pulling from library/wordpress
26c5c85e47da: Pull complete
39c8021d1258: Pull complete
dff43c2de684: Pull complete
303987c596ea: Pull complete
3fd742e8a904: Pull complete
ccf9807e8362: Pull complete
11cc7ce10028: Pull complete
1146c81172a9: Pull complete
c70fa21d385f: Pull complete
7e0980570457: Pull complete
0706e40f0b9a: Pull complete
244a64c96bb8: Pull complete
c43af3ab38bf: Pull complete
9e441c3c0a9e: Pull complete
9c180e133219: Pull complete
87b180e37365: Pull complete
1252d07e4aa8: Pull complete
71107eb26029: Pull complete
370ec22af12b: Pull complete
0f05714e9aea: Pull complete
eb8ee61b56d: Pull complete
Digest: sha256:df36d02190833fd4cd09edac11c12efad408a046ae5c5721329ec2dacdace0b
Status: Downloaded newer image for wordpress:latest
2da56000dc9d9306f68bd9e040d76841257ad23ac1b33b3c77ba6638b7b5288f

```

WordPress Installation

localhost:8081/wp-admin/install.php?step=2

## Willkommen

Du musst eine E-Mail-Adresse angeben.

**Titel der Website**

**Benutzername**   
Benutzernamen dürfen nur alphanumerische Zeichen, Leerzeichen, Unterstriche, Bindestriche, Punkte und das @-Zeichen enthalten.

**Passwort**  [Hide](#)  
Strong

**Wichtig:** Du wirst dieses Passwort zum Anmelden brauchen. Bitte bewahre es an einem sicheren Ort auf.

**Deine E-Mail-Adresse**   
Bitte überprüfe nochmal deine E-Mail-Adresse auf Richtigkeit, bevor du weitermachst.

**Sichtbarkeit für Suchmaschinen** ☐ Suchmaschinen davon abhalten, diese Website zu indizieren  
Es ist Sache der Suchmaschinen, dieser Bitte nachzukommen.

[WordPress installieren](#)

IP-Adressen der drei Container:

```

"Containers": {
  "2da56000dc9d9306f68bd9e040d76841257ad23ac1b33b3c77ba6638b7b5288f": {
    "Name": "wp_wordpress",
    "EndpointID": "e6f22ff56aae6675971f85da3d1942f296b49d16d074698b9fb1ef12e732304b",
    "MacAddress": "02:42:c0:a8:64:04",
    "IPv4Address": "192.168.100.4/24",
    "IPv6Address": ""
  },
  "5b4d8f4dd833cfa53454cc92106d37b2e3e099f0cf0d8aa8479b9ad45ea0fa2e": {
    "Name": "wp_mariadb",
    "EndpointID": "e72c40d2fa1111102152f391cd4fb4efd826aafd237f56ec06b996f16bbb4158",
    "MacAddress": "02:42:c0:a8:64:02",
    "IPv4Address": "192.168.100.2/24",
    "IPv6Address": ""
  },
  "da967c4ab68102cbee7b4b586296a1aebc9ba15d761261ae62c50fd84343ef72": {
    "Name": "wp_pma",
    "EndpointID": "883e8d43efc549ffa376dca5ec7f8e1ecc9f1e9893a83a2d69ef7b324a08ca2f",
    "MacAddress": "02:42:c0:a8:64:03",
    "IPv4Address": "192.168.100.3/24",
    "IPv6Address": ""
  }
},
},
},

```

## Images

### Standardimages

Das **Alpine**-Image ist im Gegensatz zu anderen Images klein und ist deshalb auch schnell gestartet.

Distribution	Image Grösse
alpine	6 MB
ubuntu	80 MB
debian	125 MB
oracle	250 MB

Standardmässig kommt die rudimentäre `/bin/sh` Shell zum Zug. Bash lässt sich aber nachinstallieren:

```
apk add --update bash bash-completion
```

So probiert man es aus:

```
docker run -it --rm -h alpine --name alpine alpine
```

Man gelangt in eine interaktive root-Shell und kann z.B. die Version abfragen:

```
cat /etc/os-release
```

**Ubuntu** lässt sich starten mit:

```
docker run -it --name ubuntu-test ubuntu:latest
```

Das offizielle Image von **apache** enthält nur den apache http Server, also kein php. Um den Server zu testen können Sie dieses Kommando verwenden:

```
docker run -dit --name my-apache-app -p 8080:80 -v  
"$PWD":/usr/local/apache2/htdocs/ httpd:2.4
```

Dieses verbindet das lokale Arbeitsverzeichnis (\$PWD) mit dem Serverroot. Die Webseite ist unter <http://localhost:8080> verfügbar.

**mariadb** wird von vielen Applikationen als Datenbankserver verwendet. Sie können ein mariadb Container starten mit:

```
mkdir dbvolume  
docker run -d --name mariadb -e MYSQL_ROOT_PASSWORD=geheim \  
-v $(pwd)/dbvolume:/var/lib/mysql mariadb
```

Dabei wird über eine Umgebungsvariable das root-Passwort gesetzt und das Datenbankverzeichnis im Container /var/lib/mysql auf das aktuelle Arbeitsverzeichnis/dbvolume gemountet.

Weitere Umgebungsvariablen für dieses Image wären:

Variable	Bedeutung
MYSQL_ROOT_PASSWORD	Das mysql root Passwort
MYSQL_DATABASE	Wenn diese Variable gesetzt ist, wird beim ersten Start des Containers eine leere Datenbank mit dem angegebenen Namen erstellt.
MYSQL_USER	Der angegebene Benutzer wird beim ersten Start des Containers erstellt und erhält alle Rechte (GRANT ALL) auf der in MYSQL_DATABASE angegeben Datenbank.
MYSQL_PASSWORD	Das Passwort für MYSQL_USER
MYSQL_ALLOW_EMPTY_PASSWORD	Wenn diese Variable auf yes gesetzt wird, kann bei MYSQL_ROOT_PASSWORD ein leeres Passwort gesetzt werden. Dies ist nicht zu empfehlen.
MYSQL_RANDOM_ROOT_PASSWORD	Wird diese Variable auf yes gesetzt, wird ein zufällig erzeugtes root Passwort gesetzt.

# Eigene Images

Kurz gesagt wird in einem Arbeitsverzeichnis die Datei mit Namen **Dockerfile** erstellt. Diese enthält in einer speziellen Syntax das Rezept um das Image anschliessend mit `docker build` zu erstellen. Mit `docker push` kann das Image bei Bedarf in den Dockerhub geladen werden.

## Erstes Beispiel

Bei diesem Beispiel wird ein Image erstellt, welches einen Webserver startet und eine Seite ausliefert mit dem aktuellen Datum und Zeit.

```
cd bsp-apache-php
```

Erstellen Sie in diesem Verzeichnis eine neue Datei mit Namen Dockerfile und dem abgebildeten Inhalt:

```
# Datei: bsp-apache-php/Dockerfile
FROM php:8-apache
COPY index.php /var/www/html
```

Diese Datei gibt das Rezept an, wie ein Image erstellt wird. **FROM** gibt das verwendete Basisimage an. **COPY** kopiert die Datei index.php ins Verzeichnis /var/www/html im Container. Dieses Verzeichnis ist bei Apache das Standardverzeichnis für Webseiten, d.h. beim Aufruf der Webseite wird index.php aufgerufen von php bearbeitet und an den aufrufenden Browser ausgeliefert.

Die Datei index.php muss natürlich noch erstellt werden und hat folgenden Inhalt:

```
<!DOCTYPE html >
<!-- Datei index.php -->
<html >
<head >
<title >Beispiel</title >
<meta charset ="utf-8" />
</head >
<body >
<h1>Beispiel apache/php</h1>
Serverzeit : <?php echo date("j. F Y, H:i:s, e "); ?>
</body >
</html >
```

Nun wird das Image mit folgendem Befehl erstellt:

```
docker build -t bsp-apache-php .
```

Überprüfen Sie den Erfolg mit dem Kommando:

```
docker image ls
```

Hier sollte (unter anderem) nun 2 Images aufgeführt werden:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
bsp-apache-php	latest	c5a049e80c58	About a minute ago	458MB
php	8-apache	af944036d594	2 days ago	458MB

Nun kann aus dem Image ein Container gestartet werden:

```
docker run -d --name bsp-apache-php-container -p 8080:80 bsp-apache-php
```

Nun können Sie die Webseite <http://localhost:8080> aufrufen

## Zweites Beispiel

Hier sehen Sie ein etwas umfangreicheres Beispiel eines Dockerfiles für ein eigenes Webserver Image:

```
# Datei Dockerfile
FROM ubuntu:latest

LABEL maintainer "name@meine-webseite.ch "
LABEL description "Webserver Image für www.meine-webseite.ch"

# Umgebungsvariablen und Zeitzone einstellen
# (erspart interaktive Rückfragen )
ENV TZ="Europe/Zuerich" \
    APACHE_RUN_USER=www-data \
    APACHE_RUN_GROUP=www-data \
    APACHE_LOG_DIR=/var/log/apache2

# Zeitzone einstellen, Apache installieren, unnötige Dateien
# aus dem Paket-Cache gleich wieder entfernen, HTTPS aktivieren
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && \
    echo $TZ > /etc/timezone && \
    apt-get update && \
    apt-get install -y apt-utils apache2 && \
    apt-get -y clean && \
    rm -r /var/cache/apt /var/lib/apt/lists/* && \
    a2ensite default-ssl && \
    a2enmod ssl

# Ports 80 und 443 freigeben
EXPOSE 80 443

# Das Webroot-Verzeichnis als Volume definieren
VOLUME /var/www/html

# Startkommando für den Apache Webserver
CMD ["/usr/sbin/apache2ctl" , "-D" , "FOREGROUND"]
```

Das Image wird nun erstellt mit

```
docker build -t meine-webseite-image
```

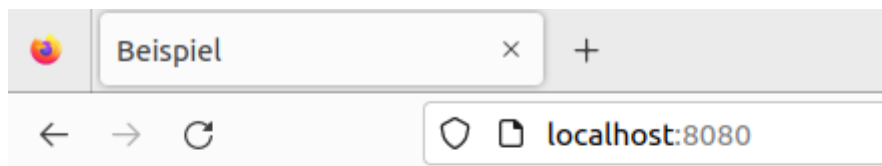
Ein aus dem Image abgeleiteter Container kann nun mit folgendem Kommando gestartet werden

```
docker run -d --name meine-webseite-container -p 8888:80 \
-v /home/vmadmin/meine-webseite/site:/var/www/html meine-webseite-image
```

Das Verzeichnis site muss vorgängig erstellt werden und wird auf das Webroot-Verzeichnis abgebildet. Wenn Sie darin eine Indexdatei index.html erstellen, wird diese nun beim Aufruf von <http://localhost:8888> angezeigt

## Aufgabe

```
vmadmin@lp-22-04:~/bsp-apache-php$ docker build -t bsp-apache-php .
Sending build context to Docker daemon 3.072kB
Step 1/2 : FROM php:8-apache
8-apache: Pulling from library/php
26c5c85e47da: Already exists
39c8021d1258: Already exists
dff43c2de684: Already exists
383987c505e8: Already exists
3fd742e8a904: Already exists
ccf9807e8362: Already exists
11cc7ce10028: Already exists
7c9a93edd83f: Pull complete
60001876b7f0: Pull complete
02e828362dd6: Pull complete
61967aaed772: Pull complete
ac804962968c: Pull complete
b9dda7bc4893: Pull complete
Digest: sha256:e28189a95082561bd68fa1424217909cdbc15295a15f39bec1a72746a6a7112b
Status: Downloaded newer image for php:8-apache
--> 577869563efd
Step 2/2 : COPY index.php /var/www/html
--> 44806a26e035
Successfully built 44806a26e035
Successfully tagged bsp-apache-php:latest
```



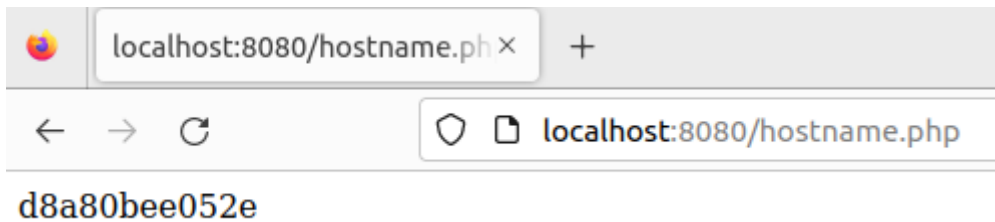
# Beispiel apache/php

Serverzeit : 1. May 2023, 08:33:24, UTC

# Beispiel apache/php

[Zeige Hostnamen](#)

Serverzeit : 1. May 2023, 09:01:13, UTC



Was ist der Unterschied zwischen COPY und ADD

Der `ADD`-Befehl ist ähnlich dem `COPY`-Befehl, hat aber einige zusätzliche Funktionen. Der `ADD`-Befehl kann auch URLs als Quelle für das Kopieren von Dateien akzeptieren und kann auch Archive (wie `.tar` oder `.zip`) extrahieren und in das Docker-Image kopieren. Der Befehl hat das folgende Format:

Unterschied zwischen CMD und ENTRYPOINT

`CMD` definiert standardmäßig den Befehl oder die Anwendung, die beim Starten des Containers ausgeführt wird.

Wenn der Docker-Container beim Starten ausgeführt wird, kann das `CMD` überschrieben werden, indem ein anderer Befehl als Argument an das `docker run`-Kommando übergeben wird.

Im Gegensatz dazu definiert `ENTRYPOINT` den ausführbaren Befehl, der beim Start des Containers ausgeführt wird.

## Docker compose

### Docker compose Aufgabe 1

docker-compose.yml:

```
services:
  wp_mariadb:
    image: mariadb:latest
    networks:
      - wp_net
    environment:
      - MYSQL_ROOT_PASSWORD=strenggeheim
      - MYSQL_DATABASE=wp
      - MYSQL_USER=wpuser
      - MYSQL_PASSWORD=geheim
    volumes:
      - wp_dbvolume:/var/lib/mysql

  wp_wordpress:
    image: wordpress:latest
    networks:
      - wp_net
    environment:
      - WORDPRESS_DB_HOST=wp_mariadb
      - WORDPRESS_DB_USER=wpuser
      - WORDPRESS_DB_NAME=wp
```

```

- WORDPRESS_DB_PASSWORD=geheim
volumes:
- wp_htmlvolume:/var/www/html/wp-content
ports:
- "8081:80"

networks:
  wp_net:
    ipam:
      driver: default
      config:
        - subnet: 192.168.200.0/24
          gateway: 192.168.200.1

volumes:
  wp_dbvolume:
  wp_htmlvolume:

```

```

vmadmin@lp-22-04:~/myapp$ docker compose up -d
[+] Running 5/5
 ✓ Network myapp_wp_net          Created
 ✓ Volume "myapp_wp_dbvolume"    Created
 ✓ Volume "myapp_wp_htmlvolume"  Created
 ✓ Container myapp-wp_wordpress-1 Started
 ✓ Container myapp-wp_mariadb-1  Started
vmadmin@lp-22-04:~/myapp$ docker compose down
[+] Running 3/3
 ✓ Container myapp-wp_mariadb-1    Removed
 ✓ Container myapp-wp_wordpress-1  Removed
 ✓ Network myapp_wp_net           Removed
vmadmin@lp-22-04:~/myapp$

```

## Docker compose Aufgabe 2

docker-compose.yml:

```

version: "3.8"

services:
  mariadb:
    image: mariadb:latest
    container_name: mymariadb
    volumes:
      - wp_dbvolume:/var/lib/mysql
    environment:
      - WORDPRESS_DB_HOST=wp_mariadb
      - WORDPRESS_DB_USER=wpuser
      - WORDPRESS_DB_NAME=wp
      - WORDPRESS_DB_PASSWORD=geheim
    ports:
      - "8081:80"
volumes:
  wp_dbvolume:

```



```

vnadmin@lp-22-04:~/myapp2$ docker compose up -d
[+] Running 2/2
 ✓ Network myapp2_default Created 0.2s
 ✓ Container mymariadb Started 2.5s
vnadmin@lp-22-04:~/myapp2$ docker logs mariadb-container
Error: No such container: mariadb-container
vnadmin@lp-22-04:~/myapp2$ docker logs mymariadb
2023-05-15 09:31:20+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.11.2+maria-ubu2204 started.
2023-05-15 09:31:21+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2023-05-15 09:31:21+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.11.2+maria-ubu2204 started.
2023-05-15 09:31:22+00:00 [ERROR] [Entrypoint]: Database is uninitialized and password option is not specified
You need to specify one of MARIADB_ROOT_PASSWORD, MARIADB_ROOT_PASSWORD_HASH, MARIADB_ALLOW_EMPTY_ROOT_PASSWORD and MARIADB_
RANDOM_ROOT_PASSWORD
vnadmin@lp-22-04:~/myapp2$ mysql -u root -p -h 127.0.0.1
Enter password:
ERROR 2002 (HY000): Can't connect to server on '127.0.0.1' (115)
vnadmin@lp-22-04:~/myapp2$ docker compose down
[+] Running 2/2
 ✓ Container mymariadb Removed 0.0s
 ✓ Network myapp2_default Removed 0.2s
vnadmin@lp-22-04:~/myapp2$

```

## Aufgabe 4.3-2

File erstellen

```
nano docker-compose.yml
```

docker-compose Inhalt

```

version: '3.8'
services:
  wp_mariadb:
    image: mariadb
    container_name: wp_mariadb
    networks:
      - wp_net
    environment:
      - MYSQL_ROOT_PASSWORD=strengeheim
      - MYSQL_DATABASE=wp
      - MYSQL_USER=wpuser
      - MYSQL_PASSWORD=geheim
    volumes:
      - wp_dbvolume:/var/lib/mysql

  wp_pma:
    image: phpmyadmin/phpmyadmin
    container_name: wp_pma
    networks:
      - wp_net
    ports:
      - 8080:80
    environment:
      - PMA_HOST=wp_mariadb

  wp_wordpress:
    image: wordpress
    container_name: wp_wordpress
    networks:
      - wp_net
    volumes:
      - wp_htmlvolume:/var/www/html/wp-content

```

```

ports:
  - 8081:80
environment:
  - WORDPRESS_DB_HOST=wp_mariadb
  - WORDPRESS_DB_USER=wpuser
  - WORDPRESS_DB_NAME=wp
  - WORDPRESS_DB_PASSWORD=geheim

networks:
  wp_net:

volumes:
  wp_dbvolume:
  wp_htmlvolume:

```

## Richtige Lösung

```

version: "3.8" services:
  wp_mariadb:
    image: mariadb:latest volumes: - wp_dbvolume:/var/lib/mysql environment:
    MYSQL_ROOT_PASSWORD: strenggeheim MYSQL_DATABASE: wp MYSQL_USER: wpuser
    MYSQL_PASSWORD: geheim networks: - wp_net
  wp_pma: image: phpmyadmin/phpmyadmin environment:
    PMA_HOST: wp_mariadb
    ports: - "8080:80"
    networks: - wp_net
  wp_wordpress:
    image: wordpress volumes: - wp_htmlvolume:/var/www/html/wp-content environment:
    WORDPRESS_DB_HOST: wp_mariadb WORDPRESS_DB_USER: wpuser WORDPRESS_DB_NAME: wp
    WORDPRESS_DB_PASSWORD: geheim
    ports: - "8081:80"
    networks: - wp_net
    volumes:
      wp_dbvolume: wp_htmlvolume:
    networks: wp_net: ipam:
      config: - subnet: 192.168.200.0/24
      gateway: 192.168.200.1

```

```
docker-compose up -d
```

```

vmadmin@lp-22-04: $ docker-compose up -d
Creating wp_pma ... done
Creating wp_mariadb ... done
Creating wp_wordpress ... done
vmadmin@lp-22-04: $ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
faafb970c4f5   wordpress                           "docker-entrypoint.s..." 17 seconds ago Up 12 seconds 0.0.0.0:8081->80/tcp, :::8081->80/tcp
4f42b714d4cc   mariadb                             "docker-entrypoint.s..." 18 seconds ago Up 13 seconds 3306/tcp
a15b12e49807   phpmyadmin/phpmyadmin              "/docker-entrypoint..." 18 seconds ago Up 13 seconds 0.0.0.0:8080->80/tcp, :::8080->80/tcp
2cd9b903b1db   portainer/portainer-ce:latest      "/portainer"             5 months ago  Up About a minute 8000/tcp, 9000/tcp, 0.0.0.0:9443->9443/tcp, :::9443->9443/tcp
vmadmin@lp-22-04: $

```

## Aufgabe 4.3-3

```

version: "3.8"

services:
  wp_mariadb:

```

```

image: mariadb:latest
volumes:
- wp_dbvolume:/var/lib/mysql
environment:
- MYSQL_ROOT_PASSWORD_FILE=/run/secrets/mariadb_root_password
- MYSQL_DATABASE=wp
- MYSQL_USER=wpuser
- MYSQL_PASSWORD_FILE=/run/secrets/mariadb_password
networks:
- wp_net

wp_pma:
image: phpmyadmin/phpmyadmin
environment:
- PMA_HOST=wp_mariadb
ports:
- "8080:80"
networks:
- wp_net

wp_wordpress:
image: wordpress
volumes:
- wp_htmlvolume:/var/www/html/wp-content
environment:
- WORDPRESS_DB_HOST=wp_mariadb
- WORDPRESS_DB_USER=wpuser
- WORDPRESS_DB_NAME=wp
- WORDPRESS_DB_PASSWORD_FILE=/run/secrets/wordpress_db_password
ports:
- "8081:80"
networks:
- wp_net

volumes:
wp_dbvolume:
wp_htmlvolume:

networks:
wp_net:
ipam:
config:
- subnet: 192.168.200.0/24
gateway: 192.168.200.1

secrets:
mariadb_root_password:
file: mariadb_root_password.txt
mariadb_password:
file: mariadb_password.txt
wordpress_db_password:
file: wordpress_db_password.txt

```

Dateien mit Passwort darin erstellen

```
docker-compose up -d
```

```
vmadmin@lp-22-04:~$ docker-compose up -d
Creating vmadmin_wp_mariadb_1    ... done
Creating vmadmin_wp_wordpress_1 ... done
Creating vmadmin_wp_pma_1       ... done
```

## Lernziele

**Sie können die Begriffe "Containerisierung", "Image", "Layer", "Container", "Repository", "Registry" und Dockerfile erklären**

1. Containerisierung: Containerisierung ermöglicht es, Anwendungen in isolierten Behältern (Containern) auszuführen, die alle benötigten Dateien und Einstellungen enthalten. Dies sorgt für Konsistenz und Portabilität der Anwendung.
2. Image: Ein Image ist eine Datei, die als Vorlage für die Erstellung von Containern dient. Es enthält alle benötigten Dateien und Konfigurationen, um eine Anwendung auszuführen.
3. Layer: Ein Layer ist eine einzelne Schicht innerhalb eines Docker-Images. Images bestehen aus mehreren Schichten, die Änderungen und Ergänzungen zum Dateisystem enthalten. Layer ermöglichen eine effiziente Speicherung und Verwaltung von Images.
4. Container: Ein Container ist eine isolierte Ausführungsumgebung für eine Anwendung. Es basiert auf einem Image und enthält alles, was benötigt wird, um die Anwendung auszuführen. Container sind konsistent und können auf verschiedenen Systemen ausgeführt werden.
5. Repository: Ein Repository ist ein Speicherort für Docker-Images. Es ermöglicht das Speichern, Verwalten und Teilen von Images. Ein Repository enthält verschiedene Versionen und Varianten eines Images.
6. Registry: Eine Registry ist ein Dienst, der als zentraler Speicherort für Docker-Images dient. Registries ermöglichen das Speichern, Verteilen und Verwalten von Images. Es gibt öffentliche Registries wie Docker Hub und private Registries für die interne Nutzung.
7. Dockerfile: Ein Dockerfile ist eine Textdatei, die Anweisungen enthält, um ein Docker-Image zu erstellen. Es beschreibt den Aufbau des Images, wie das Basisimage ausgewählt wird, Dateien kopiert werden und welche Einstellungen vorgenommen werden. Dockerfiles automatisieren die Image-Erstellung und ermöglichen eine einfache Wiederholbarkeit und Anpassung.

**Sie können die Begriffe Virtualisierung und Containerisierung voneinander trennen.**

Virtualisierung:

- Bei der Virtualisierung wird eine virtuelle Maschine (VM) erstellt, die als eigenständiger Computer fungiert und eine vollständige Betriebssysteminstanz mit allen benötigten Ressourcen enthält.

- Jede virtuelle Maschine emuliert eine physische Hardwareebene und kann verschiedene Betriebssysteme und Anwendungen ausführen.
- Virtualisierung ermöglicht die Isolierung von Anwendungen und bietet eine hohe Flexibilität, da verschiedene Betriebssysteme und Versionen parallel ausgeführt werden können.
- Es erfordert jedoch mehr Ressourcen, da jede virtuelle Maschine ihre eigenen Betriebssystemressourcen benötigt.

Containerisierung:

- Bei der Containerisierung werden Anwendungen in isolierten Containern ausgeführt, die eine leichte und schnelle Alternative zur Virtualisierung bieten.
- Containerisierung basiert auf der Nutzung des Host-Betriebssystems und des Kernel-Sharings, wodurch mehrere Container auf einem einzigen Host-System ausgeführt werden können.
- Container teilen sich den Host-Kernel, sind jedoch in ihrer Umgebung isoliert, sodass jede Anwendung ihre eigenen Dateisysteme, Prozesse und Netzwerkschnittstellen hat.
- Container sind leichtgewichtiger und starten schneller als virtuelle Maschinen. Sie bieten eine bessere Ressourceneffizienz und Skalierbarkeit.
- Containerisierung ist ideal für die Bereitstellung von Microservices und die schnelle Entwicklung und Bereitstellung von Anwendungen.

Zusammenfassend lässt sich sagen, dass Virtualisierung eine vollständige Virtualisierung von Hardware und Betriebssystemen ermöglicht, während Containerisierung auf der Isolierung von Anwendungen basiert und eine leichtgewichtige Alternative mit besserer Skalierbarkeit und Effizienz bietet.

**Sie kennen die elementaren Commands, um Images und Container zu erzeugen, starten, beenden und löschen**

**Sie können die wichtigsten Optionen von `docker run` (`--name`, `--rm`, `--network`, `--ip`, `-d`, `-it`, `-p`, `-v`, `-e`) korrekt anwenden**

- `--name <name>`: Gibt dem Container einen Namen, mit dem er leicht identifiziert werden kann. Zum Beispiel: `docker run --name my-container image-name:tag`.
- `--rm`: Löscht den Container automatisch, sobald er beendet wird. Dies ist praktisch, um temporäre Container zu erstellen, die nach der Ausführung nicht mehr benötigt werden. Zum Beispiel: `docker run --rm image-name:tag`.
- `--network <network>`: Weist den Container einem bestimmten Docker-Netzwerk zu. Dadurch können Container miteinander kommunizieren, indem sie den Netzwerknamen verwenden. Zum Beispiel: `docker run --network my-network image-name:tag`.
- `--ip <ip>`: Weist dem Container eine bestimmte IP-Adresse zu, wenn er mit einem benutzerdefinierten Netzwerk verbunden ist. Zum Beispiel: `docker run --network my-network --ip 192.168.0.10 image-name:tag`.
- `-d`: Startet den Container im Hintergrund (detached mode), sodass er im Hintergrund ausgeführt wird und die Kontrolle an die Shell zurückgegeben wird. Zum Beispiel: `docker run -d image-name:tag`.

- **-it**: Startet den Container im interaktiven Modus und bindet die Standardein- und -ausgabe an die Shell. Dies ermöglicht die interaktive Kommunikation mit dem Container. Zum Beispiel: `docker run -it image-name:tag`.
- **-p <host-port>:<container-port>**: Leitet den Verkehr vom Host-Port zum Container-Port weiter, wodurch die Verbindung zur Anwendung im Container hergestellt wird. Zum Beispiel: `docker run -p 8080:80 image-name:tag` leitet den Verkehr vom Host-Port 8080 zum Container-Port 80 weiter.
- **-v <host-path>:<container-path>**: Bindet einen Host-Verzeichnispfad an einen Pfad innerhalb des Containers. Dadurch können Daten zwischen dem Host und dem Container ausgetauscht werden. Zum Beispiel: `docker run -v /host/path:/container/path image-name:tag`.
- **-e <key=value>**: Setzt eine Umgebungsvariable im Container. Diese Option kann verwendet werden, um Konfigurationswerte an die Anwendung im Container zu übergeben. Zum Beispiel: `docker run -e ENV_VARIABLE=value image-name:tag`.

### Sie können verschiedene Versionen (tags) eines Container-Images nutzen

```
docker run my-image:v1.0 // Startet den Container mit Tag v1.0
docker run my-image:v1.1 // Startet den Container mit Tag v1.1
docker run my-image:latest // Startet den Container mit dem als latest markierten Tag
```

Standardmässig nimmt es **latest**.

### Sie können Portweiterleitungen in Betrieb nehmen

hostport:containerport

Weitere Infos oben.

### Sie können benannte und gemountete Volumen korrekt einsetzen

Volumename:Containerverzeichnis

Hostverzeichnis:Containerverzeichnis

Weitere Infos oben.

### Sie verstehen den Standardaufbau eines Netzwerks mit Docker

Weitere Infos oben.

### Sie können Docker-Netzwerke definieren und diese den Containern zuweisen

Weitere Infos oben.

## Sie kennen die Syntax von Dockerfiles

- `FROM`: Gibt das Basisimage an, auf dem das neue Image aufgebaut wird.
  - `RUN`: Führt einen Befehl innerhalb des Images aus, um Pakete zu installieren, Abhängigkeiten zu konfigurieren usw.
  - `COPY` oder `ADD`: Kopiert Dateien oder Verzeichnisse vom Host in das Image.
  - `WORKDIR`: Setzt das Arbeitsverzeichnis für nachfolgende Anweisungen innerhalb des Containers.
  - `EXPOSE`: Deklariert die Ports, auf denen der Container Anwendungen verfügbar macht.
  - `CMD` oder `ENTRYPOINT`: Gibt den Befehl an, der beim Starten des Containers ausgeführt werden soll.
2. Variablen: Um die Lesbarkeit und Wiederverwendbarkeit des Dockerfiles zu verbessern, können Variablen verwendet werden. Sie werden mit dem Format `$VARIABLE_NAME` oder `${VARIABLE_NAME}` dargestellt und können entweder im Dockerfile selbst oder über Umgebungsvariablen gesetzt werden.
3. Schichtenaufbau: Dockerfiles verwenden eine schichtbasierte Struktur. Jede Anweisung in einem Dockerfile erzeugt eine neue Schicht im Image. Schichten sind effizient, da sie wiederverwendet werden können, wenn sich die vorherigen Schichten nicht ändern.

Beispiel:

```
# Kommentar: Dockerfile für meine Anwendung

# Setzen des Basisimages
FROM ubuntu:latest

# Ausführen von Anweisungen innerhalb des Images
RUN apt-get update && apt-get install -y \
    package1 \
    package2

# Kopieren von Dateien vom Host in das Image
COPY app /app

# Festlegen des Arbeitsverzeichnisses
WORKDIR /app

# Exponieren eines Ports
EXPOSE 8080

# Befehl, der beim Starten des Containers ausgeführt wird
CMD ["python", "app.py"]
```

## Sie können die 12 verschiedenen Anweisungen von Dockerfiles erklären

### Sie können zwischen ENTRYPOINT und CMD sowie COPY und ADD unterscheiden

`ENTRYPOINT` und `CMD` unterscheiden sich in ihrer Verwendung und ihrem Verhalten:

- **ENTRYPOINT**: Definiert den Befehl oder das Skript, das immer ausgeführt wird, wenn der Container gestartet wird. Es stellt den Hauptbefehl des Containers dar und kann nicht überschrieben werden. Wenn **ENTRYPOINT** zusammen mit **CMD** verwendet wird, fungiert **CMD** als Argumente für **ENTRYPOINT**.
- **CMD**: Gibt den Standardbefehl oder das Standardskript an, das beim Starten des Containers ausgeführt wird. Es kann überschrieben werden, indem beim Starten des Containers Befehle oder Argumente angegeben werden. Wenn **CMD** zusammen mit **ENTRYPOINT** verwendet wird, stellt **CMD** Argumente für den Befehl dar, der durch **ENTRYPOINT** angegeben ist.

Beispiel

```
FROM ubuntu:latest
ENTRYPOINT ["echo", "Hello"]
CMD ["world"]
```

- **COPY** und **ADD** werden verwendet, um Dateien oder Verzeichnisse vom Host in das Image zu kopieren, haben aber einige Unterschiede:
- **COPY**: Kopiert Dateien oder Verzeichnisse vom Host in das Image. Es unterstützt grundlegende Kopierfunktionen und ist die bevorzugte Option, wenn Sie einfach Dateien in das Image kopieren möchten.
- **ADD**: Ähnlich wie **COPY**, kann aber auch URLs oder TAR-Archive verarbeiten. Es hat zusätzliche Funktionen wie das Entpacken von TAR-Archiven und das Herunterladen von Inhalten von URLs. Es wird empfohlen, **ADD** nur zu verwenden, wenn Sie die spezifischen Funktionen benötigen.

Im Allgemeinen wird empfohlen, **COPY** für einfache Kopieroperationen zu verwenden und **ADD** nur dann einzusetzen, wenn die zusätzlichen Funktionen benötigt werden.

## Miniprojekt

1. Ein Lokales Verzeichnis **webserver** erstellen und auf das Verzeichnis wechseln.

```
mkdir webserver
cd webserver
```

```
vmadmin@lp-22-04:~$ mkdir webserver
vmadmin@lp-22-04:~$ cd webserver
vmadmin@lp-22-04:~/webserver$
```

2. Dockerfile und Dateistruktur erstellen

```
nano Dockerfile
```

Konfigurationen vornehmen



```
FROM nginx

WORKDIR /usr/share/nginx/html

COPY ./public/ /usr/share/nginx/html

EXPOSE 8080

CMD ["nginx", "-g", "daemon off;"]
```

**From** bestimmt das Basisimage

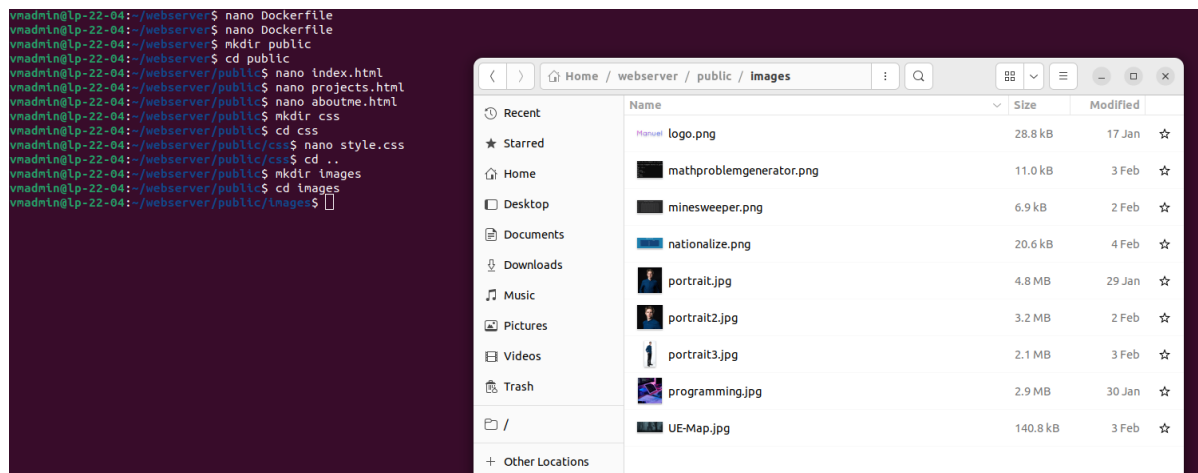
**Workdir** setzt das Arbeitsverzeichnis

**Copy** kopiert Dateien aus dem Hostsystem ins Image

**Expose** öffnet den angegebenen Port

Der letzte Befehl ist dafür verantwortlich, dass sich Nginx beim Containerstart automatisch startet

Dateistruktur für den **Copy** Befehl:



### 3. Image bauen

```
docker build -t mein-webserver .
```

-t steht für **Tag**, also zur Identifizierung des Images.

Der Punkt am Ende bedeutet einfach, dass sich das Dockerfile im aktuellen Verzeichnis befindet.

```

vmadmin@lp-22-04:~/webserver$ docker build -t mein-webserver .
Sending build context to Docker daemon 13.24MB
Step 1/5 : FROM nginx
latest: Pulling from library/nginx
9e3ea8720c6d: Pull complete
bf36b6466679: Pull complete
15a97cf85bb8: Pull complete
9c2d6be5a61d: Pull complete
6b7e4a5c7c7a: Pull complete
8db4caa19df8: Pull complete
Digest: sha256:480868e8c8c797794257e2abd88d0f9a8809b2fe956cbfbc05dcc0bca1f7cd43
Status: Downloaded newer image for nginx:latest
---> 448a08f1d2f9
Step 2/5 : WORKDIR /usr/share/nginx/html
---> Running in 792fc3b9057f
Removing intermediate container 792fc3b9057f
---> d20c975dbf8f
Step 3/5 : COPY ./public/ /usr/share/nginx/html
---> af17d48cfb6f
Step 4/5 : EXPOSE 8080
---> Running in 7ec6bae1bd6f
Removing intermediate container 7ec6bae1bd6f
---> 5a66cc03aec8
Step 5/5 : CMD ["nginx", "-g", "daemon off;"]
---> Running in a9168917b6ca
Removing intermediate container a9168917b6ca
---> 4be69c4f50c8
Successfully built 4be69c4f50c8
Successfully tagged mein-webserver:latest
vmadmin@lp-22-04:~/webserver$

```

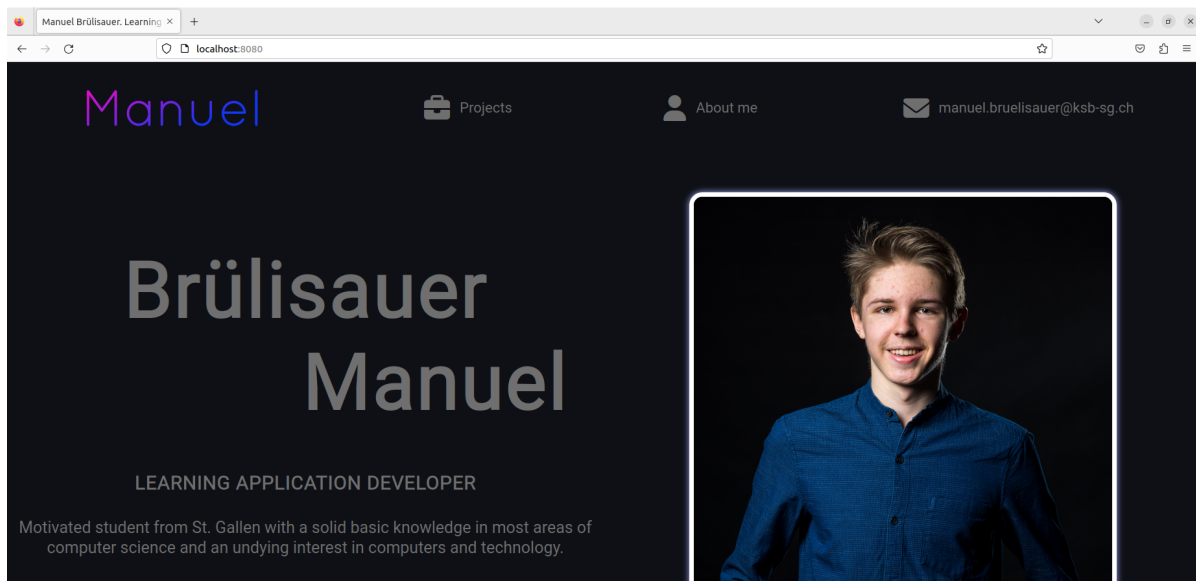
#### 4. Container starten

```
docker run -p 8080:80 -v /var/log:/var/log/nginx --name mein-container mein-webserver
```

```

mein-container
vmadmin@lp-22-04:/$ docker run -p 8080:80 -v /var/log:/var/log/nginx --name mein-container mein-webserver
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up

```



# Microservices, Entwicklung mit dotnet

## Aufgabe 7.4-1

.NET 7 installieren.

Dockerfile:

```
FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build-env
WORKDIR /build
COPY . .
RUN dotnet restore
RUN dotnet publish -c Release -o out

FROM mcr.microsoft.com/dotnet/aspnet:6.0
LABEL description="Minimal Api with MongoDB"
LABEL organisation="KSB St. Gallen"
LABEL author="Manuel Bruehlisauer"
WORKDIR /app
COPY --from=build-env /build/out .
ENTRYPOINT ["dotnet", "app.dll"]
```

```
docker run -p 5001:80 myminimalapi
```

Dieser Command war wichtig, da der Port auf dem Localhost sonst nicht richtig funktioniert hat.

Docker compose file:

```
version: "3.9"
services:
  myminimalapi:
    build: webApi
    ports:
      - 5001:80
```

## Aufgabe 7.4-2

Um Applikation auszuführen

```
dotnet run
```

```
code .
```

Mongodb Container:

```
docker run -d --name my-mongodb -p 27017:27017 -v mydata:/data/db mongo
```

```
dotnet add package MongoDB.Driver
```

Connection-String hardcode:

```
using MongoDB.Driver;

var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/", () => "Minimal API Version 1.0");

// app.MapGet("/check", () => { /* Code zur Prüfung der DB ...*/ return "Zugriff
auf MongoDB ok.";});

app.MapGet("/check", () =>
{
    try
    {
        var mongoDbConnectionString = "mongodb://localhost:27017";
        var mongoClient = new MongoClient(mongoDbConnectionString);
        var databaseNames = mongoClient.ListDatabaseNames().ToList();

        return "Zugriff auf MongoDB ok. Vorhandene DBs: " + string.Join(",",
databaseNames);
    }
    catch (System.Exception e)
    {
        return "Zugriff auf MongoDB funktioniert nicht: " + e.Message;
    }
});

app.Run();
```

Neues cs (kein hardgecodeter Connection-String mehr):

```
using MongoDB.Driver;

var builder = WebApplication.CreateBuilder(args);

var movieDatabaseConfigSection =
builder.Configuration.GetSection("DatabaseSettings");
builder.Services.Configure<DatabaseSettings>(movieDatabaseConfigSection);

var app = builder.Build();

app.MapGet("/", () => "Minimal API Version 1.0");

// app.MapGet("/check", () => { /* Code zur Prüfung der DB ...*/ return "Zugriff
auf MongoDB ok.";});
```

```

app.MapGet("/check", (Microsoft.Extensions.Options.IOptions<DatabaseSettings>
options) =>
{
    try
    {
        var mongoDbConnectionString = options.Value.ConnectionString;
        var mongoClient = new MongoClient(mongoDbConnectionString);
        var databaseNames = mongoClient.ListDatabaseNames().ToList();

        return "Zugriff auf MongoDB ok. Vorhandene DBs: " + string.Join(",",
databaseNames);
    }
    catch (System.Exception e)
    {
        return "Zugriff auf MongoDB funktioniert nicht: " + e.Message;
    }
});

app.Run();

```

appsettings.json:

```

{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "DatabaseSettings": { "ConnectionString": "mongodb://localhost:27017"}
}

```

neues Docker-compose file:

```

version: "3.9"
services:
  mongodb:
    image: mongo

  myminimalapi:
    build: webApi
    ports:
      - 5001:80
    depends_on:
      - mongodb
    environment:
      -
      MoviesDatabaseSettings__ConnectionString=mongodb://gbs:geheim@mongodb:27017

```

# Lernziele 2

## WebApi Projekt

Sie können mit [YAML-Dateien](#) korrekt umgehen

- Eine Liste wird mit `-` + Abstand gebildet, z.B.

```
# A list of tasty fruits
- Apple
- Orange
- Strawberry
- Mango
```

- Key-Value-Paare werden mit `:` + Abstand gebildet, z.B.:

```
name: Hans Martin
```

- 

Objekte bestehend aus mehreren Key-Value-Paaren, können durch Zeilenumbrüche gebildet werden

```
vorname: Hans Martin
nachname: Keller
alter: 20
```

- 

Ein Value kann seinerseits wiederum rekursiv aus Objekten, Key-Value-Paaren oder Listen bestehen:

```
# Employee records
- martin:
  name: Martin D'vloper
  job: Developer
  skills:
    - python
    - perl
    - pascal
- tabitha:
  name: Tabitha Bitumen
  job: Developer
  skills:
    - lisp
    - fortran
    - erlang
```

- Die Strukturierung der Elemente wird durch Einrückungen erreicht. Dabei **müssen Leerzeichen** verwendet werden. Tabulatoren auch am Zeilenende führen zu Fehlern.

Sie kennen die Syntax von `docker-compose.yml` und können die Schlüsselwörter korrekt anwenden

- **version:** zur Zeit "3.8"
- **services:** Zur Definition der einzelnen Container. Die Dienstnamen (Keys) für die Container können freigewählt werden.
- **networks:** Zur Definition eigener Netzwerke
- **volumes:** Angabe von benannten Volumes
- **secrets:** Angabe von Passwortdateien

```
# Grundsätzlicher Aufbau von docker-compose.yml
version: "3.8"
services:
  dienstname1:
    schlüsselwort1: einstellung1
    schlüsselwort2: einstellung2
    ...
  dienstname2:
    schlüsselwort1: einstellung1
    schlüsselwort2: einstellung2
    ...
volumes:
networks:
secrets:
```

Die wichtigsten Schlüsselwörter unterhalb der Dienstnamen sind:

- **image:** Das Basisimage eines Service oder
- **build:** Ein Verzeichnis mit Dockerfile, der Service wird dann nicht aus einem Image gestartet sondern aus einem Dockerfile
- **container\_name:** Der Name für den resultierenden Container
- **restart:** always, on-failure oder unless-stopped
- **environment:** Liste mit Umgebungsvariablen für einen Container
- **volumes:** Liste der gemounteten Volumes
- **ports:** Liste der Portweiterleitungen
- **expose:** Ports für die Kommunikation zwischen Containern
- **networks:** Verweis auf ein im Top-Level-Schlüsselwort networks definiertes Netzwerk
- **secrets:** Verweis auf die im Top-Level-Schlüsselwort secrets definierte Passwortdatei

Hier nun die Schlüsselwörter im Detail:

## image

Wenn das Basisimage in Dockerhub verfügbar ist, kann es direkt verwendet werden:

```
services:
  my-service:
    image: ubuntu:latest
    ...
```

## build

Wird ein eigenes Image benötigt, kann dieses aus dem angegebenen Dockerfile erstellt werden:

```
services:
  my-custom-app:
    build: /path/to/dockerfile/
    ...
```

### container\_name

Der Name für den resultierenden Container bei docker ps (analog --name bei docker run)

```
services:
  my-custom-app:
    container_name: my-container
    ...
```

Wird der container\_name nicht angegeben, resultiert für den Namen des Containers eine Zusammensetzung aus Arbeitsverzeichnis, Servicename und einer Laufnummer, z.B. workdir-my-custom-app-1. Der Name des Services entspricht also **nicht** dem Namen des Containers.

### restart

Dieser Eintrag definiert die Restart-Policy für einen Container. Folgende Werte sind möglich

- **no**: Der Standard, Container wird nicht automatisch gestartet
- **on-failure**: Der Container wird bei einem Fehler (Exit-Code ungleich 0) automatisch neu gestartet
- **always**: Der Container wird immer neu gestartet, insbesondere bei einem Reboot. Wenn der Container manuell gestoppt wird, startet er neu, wenn der Dockerdaemon neu gestartet wird.
- **unless-stopped**: Ähnlich wie always, wird der Container manuell gestoppt, startet er nach einem Neustart des Dockerdaemons oder einem Reboot nicht mehr neu.

```
services:
  my-custom-app:
    restart: always
    ...
```

### ports

Um einen Service vom Host aus zu erreichen, wird die Angabe der Portweiterleitungen benötigt:

```
services:
  my-custom-app:
    image: myapp:latest
    ports:
      - "8080:3000"
      - "8081:4000"
    ...
```

### expose

Angabe der Ports, welche die Container für die Kommunikation untereinander benötigen. Wenn ein Basisimage bereits einen Port exponiert, ist diese Angabe nicht nötig.



```
services:
  network-example-service:
    image: karthequian/helloworld:latest
    expose:
      - "80"
```

## networks

Hier kann ein unter dem Top-Level-Schlüsselwort definiertes Netzwerk referenziert werden.

```
services:
  network-example-service:
    image: karthequian/helloworld:latest
    networks:
      - my_network
    ...
  another-service-in-the-same-network:
    image: alpine:latest
    networks:
      - my_network
    ...

networks:
  my_network:
    ipam:
      config:
        - subnet: 172.17.0.0/24
          gateway: 172.17.0.1
```

In der Regel werden bei der Definition des Netzwerkes keine weiteren Optionen angegeben. Docker kümmert sich dann selber um die konkrete Definition:

```
services:
  network-example-service:
    image: karthequian/helloworld:latest
    networks:
      - my_network
    ...

networks:
  my_network:
```

## volumes

Gibt an wie Containervolumes auf den Host gemountet werden. Benannte Volumes müssen im Top-Level-Schlüsselwort volumes angegeben werden. Somit können benannte Volumes von mehreren Containern gleichzeitig verwendet werden. Der Zusatz :ro mountet ein Verzeichnis read-only.

```
services:
  volumes-example-service:
    image: alpine:latest
    volumes:
      - my-named-global-volume:/my-volumes/named-global-volume
```

```

- /tmp:/my-volumes/host-volume
- /home:/my-volumes/readonly-host-volume:ro
...
another-volumes-example-service:
  image: alpine:latest
  volumes:
    - my-named-global-volume:/another-path/the-same-named-global-volume
    ...
volumes:
  my-named-global-volume:

```

Unbenannte Volumes und Volumes in eigene Verzeichnisse müssen nicht im Top-Level volume aufgeführt werden.

Es können auch einzelnen Dateien gemountet werden.

### depends\_on

Um Abhängigkeiten zu definieren, kann depends\_on verwendet werden. Die betreffenden Container werden dann zuerst geladen:

```

services:
  kafka:
    image: wurstmeister/kafka:2.11-0.11.0.3
    depends_on:
      - zookeeper
    ...
  zookeeper:
    image: wurstmeister/zookeeper
    ...

```

### environment

Variablen können sowohl statisch als auch dynamisch mit \${} definiert werden

```

services:
  database:
    image: "postgres:${POSTGRES_VERSION}"
    environment:
      DB: mydb
      USER: "${USER}"

```

Die dynamischen Variablen können dabei u.a. in einer Datei .env im selben Verzeichnis als Key-Value-Paare definiert werden:

```

POSTGRES_VERSION=alpine
USER=foo

```

Die verfügbaren Umgebungsvariablen müssen in der Dokumentation zu einem Image auf Dockerhub nachgeschaut werden

### secrets

Sollen Passwörter aus sicherheitstechnischen Gründen nicht in der docker-compose Datei aufgeführt werden, kann man secrets verwenden. Das Top-Level-Schlüsselwort gibt an, wo sich die Passwortdatei befindet. Die Einträge bei den Services referenzieren dann die Top-Level-Definition

```
services:
  db:
    image: mysql:latest
    environment:
      MYSQL_ROOT_PASSWORD_FILE: /run/secrets/db_root_password
      MYSQL_PASSWORD_FILE: /run/secrets/db_password
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
    secrets:
      - db_root_password
      - db_password
    ...
secrets:
  db_password:
    file: db_password.txt
  db_root_password:
    file: db_root_password.txt
```

Die in MYSQL\_ROOT\_PASSWORD\_FILE und MYSQL\_PASSWORD\_FILE angegebenen Dateien innerhalb des Containers enthalten dann die in den unter dem Top-Level-Schlüsselwort secrets angegebenen Passwörter. Die verfügbaren Secretsvariablen müssen in der Dokumentation zu einem Image auf Dockerhub nachgeschaut werden.

### Sie können das Kommando docker compose up/down mit den Schaltern -d und --build korrekt anwenden

- `-d` wird verwendet, um die Container im Hintergrund (detach mode) auszuführen. Dadurch werden die Logs nicht auf dem Terminal ausgegeben.
- `--build` wird verwendet, um die Images der Dienste neu zu erstellen, falls Änderungen an den Dockerfile- oder Build-Kontextdateien vorgenommen wurden.

Beispiel:

- `docker-compose up -d`: Startet die Dienste im Hintergrund, ohne die Images neu zu erstellen.
- `docker-compose up --build`: Startet die Dienste und erstellt die Images neu, falls erforderlich.
- `docker-compose up -d --build`: Startet die Dienste im Hintergrund und erstellt die Images neu, falls erforderlich.

Das Kommando `docker-compose down` wird verwendet, um die Dienste in einem Docker-Compose-Projekt zu stoppen und die zugehörigen Container, Netzwerke und Volumes zu entfernen.

### Sie können Dockerfiles für Multistage Builds im Zusammenhang mit dotnet aufbauen

### Sie können mit dotnet Web- und Konsolen-Anwendungen korrekt umgehen

