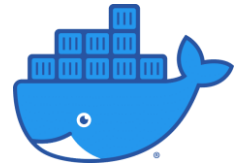
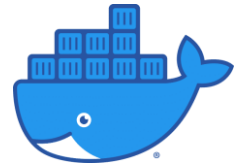


Docker

MEDT4

htl Hollabrunn



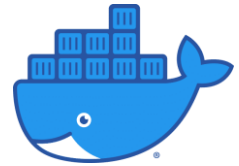
It works on my machine…

- Software läuft fehlerfrei…
  - am Rechner des Entwicklers
  - am Entwicklungssystem (ENTW-System)

- Fehler beim Testen…
  - am Qualitätssicherungssystem (QS-System)

- Risiko einer Bereitstellung (Deployment) am Produktivsystem (PROD-System)?
  - no risk, no fun!

htl Hollabrunn

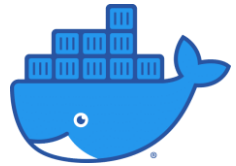


Umgebungen

- ENTW (dev)
- QS (test)
- PROD (prod)

- sind möglicherweise unterschiedlich
  - Hardware
  - OS
  - installierte Bibliotheken
  - Service Packs
  - Module

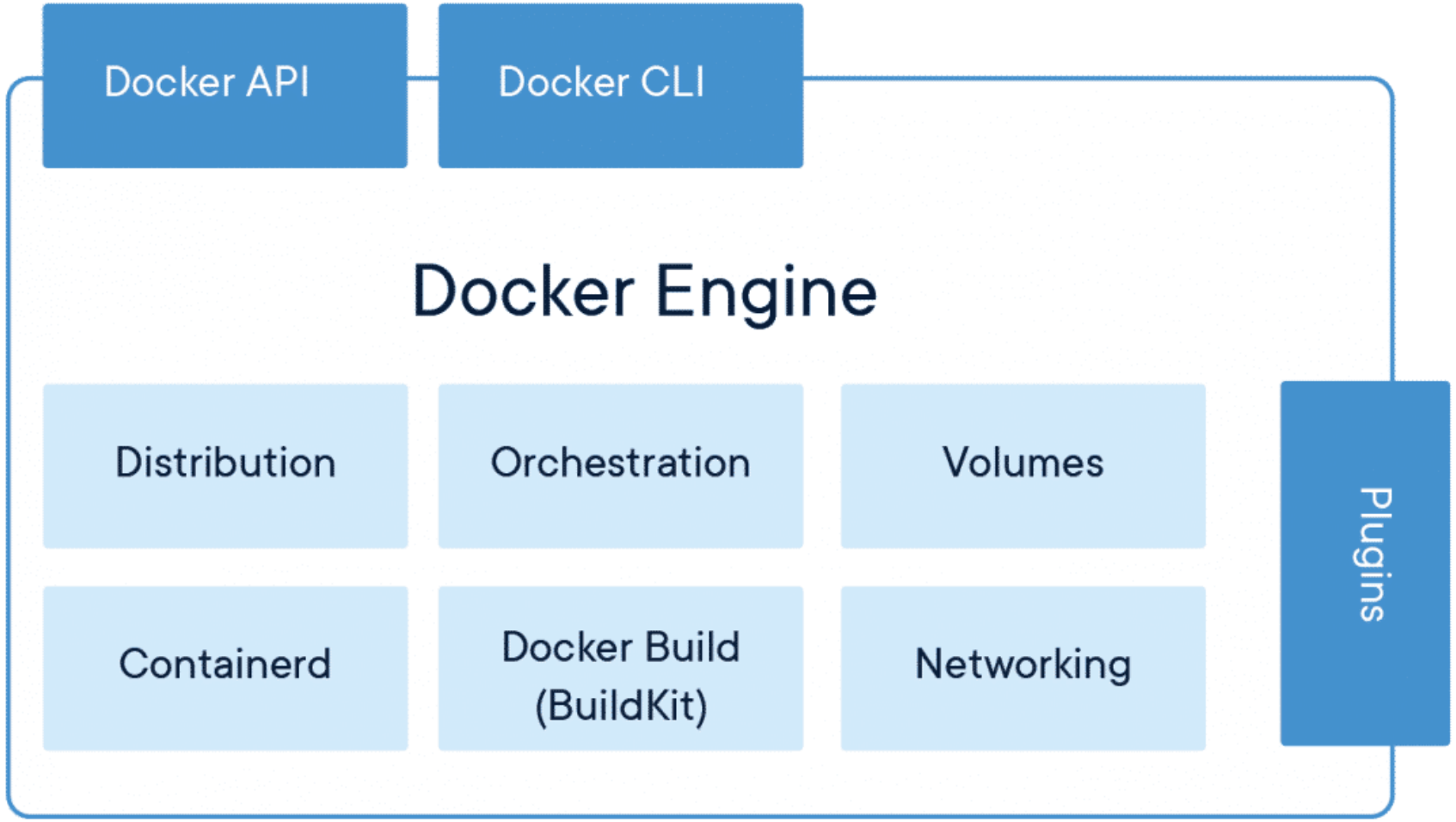
htl Hollabrunn



Docker

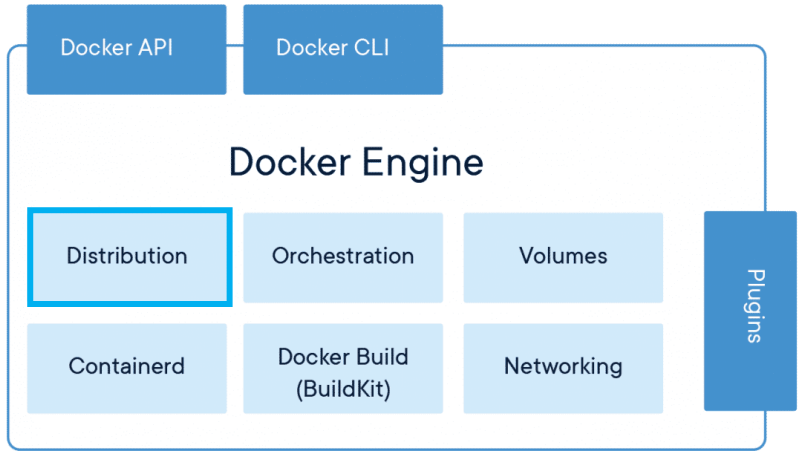
- dotCloud seit 2008, gegründet in Paris (mittlerweile im Silicon Valley)
- Docker wurde 2013 veröffentlicht (Umbenennung des Unternehmens)
- basiert auf
  - cgroups
    - Definition von Quotas auf Prozesse (auf CPU- und RAM-Ebene)
  - namespaces
    - Prozesse isolieren (denkt er ist der einzige Prozess)
    - Netzwerkinterface zu Prozessen zuordnen
    - virtuelles Dateisystem für Prozesse
- ähnlich wie VM
  - Isolation eines Prozesses
  - Leistungsressourcen begrenzen
  - ohne Overhead einer VM
- **läuft auf Linux**

Docker Engine



htl Hollabrunn

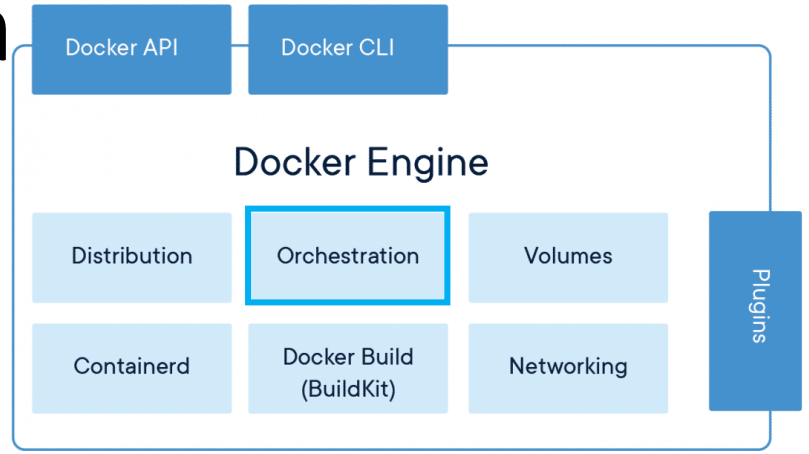
Docker Engine - Distribution



- to distribute – verteilen
- Werkzeug um Inhalte verpacken (pack), verschicken (ship), ablegen (store) und zustellen (deliver)
- Basis der **Container-Registry**
  - Unterteilung in Repositories
  - Ein Repository beinhaltet alle Versionen eines spezifischen Docker-Images
  - Docker-Images aus Registry lokal ablegen → pull
  - Neue Docker-Images ins Registry → push

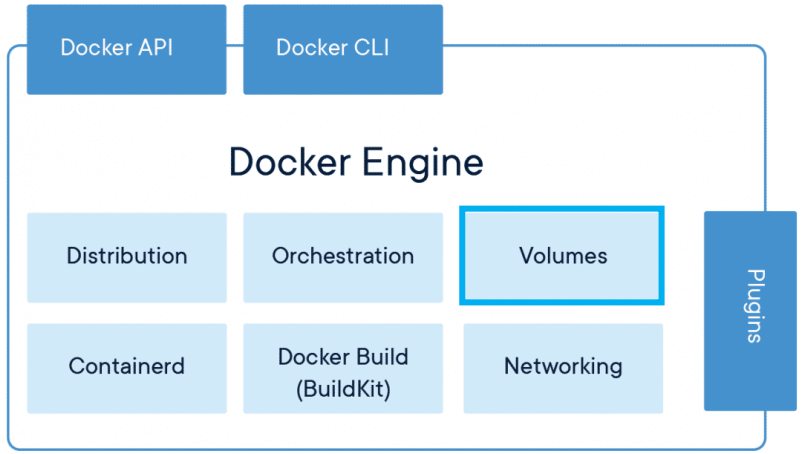
htl Hollabrunn

Docker Engine - Orchestration



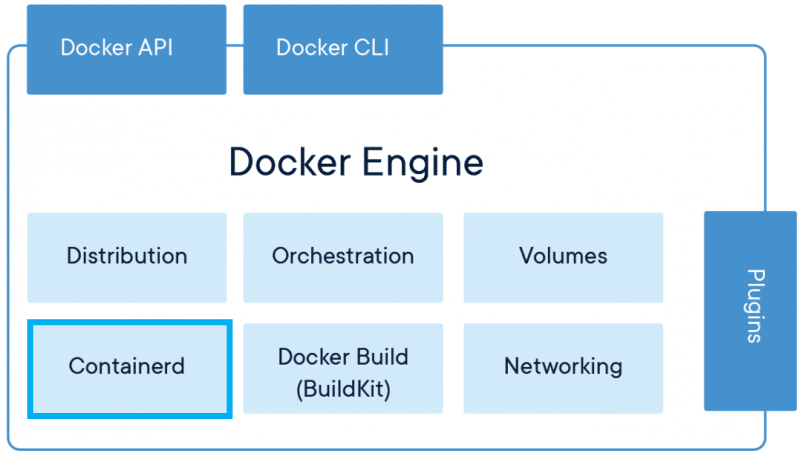
htl Hollabrunn

Docker Engine - Volumes



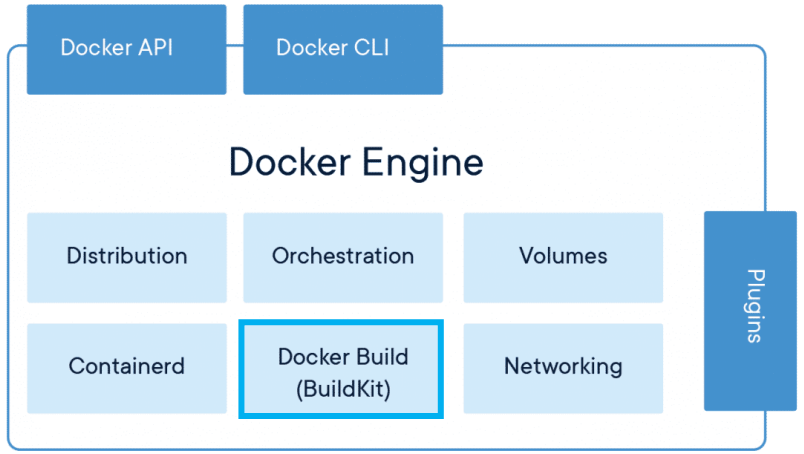
htl Hollabrunn

Docker Engine - Containerd



htl Hollabrunn

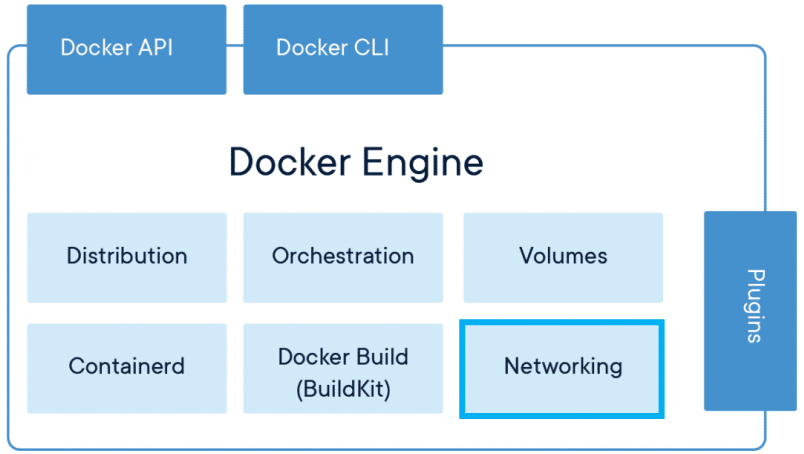
Docker Engine - Build



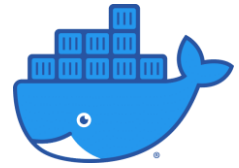
htl Hollabrunn

Docker Engine - Networking

- Bridged
- Overlay



htl Hollabrunn



Installation

- Linux
  - via Paketmanager (z.B. https://docs.docker.com/engine/install/ubuntu/)
- Windows & Mac
  - Docker Desktop
    Windows: https://docs.docker.com/desktop/install/windows-install/,
    Mac: https://docs.docker.com/desktop/install/mac-install/

- Installation-Check

```
> docker version
```

htl Hollabrunn

docker (Client)

- Images (für vorkonfigurierte Laufzeitumgebungen)

```
> docker pull ubuntu
```

- Ausführung

```
> docker run –it <image> <application>
```

  - i … interaktiv (STDIN bleibt geöffnet)
  - t … TTY – "Terminal"

```
> docker run -it ubuntu bash
```

htl Hollabrunn

docker (Client)

- Image-Versionen (Tags)
  - im Falle von Ubuntu Monate 04 und 10

```
> docker pull ubuntu:YYMM
```

  - Auflistung der geladenen Images/Tags

```
> docker images
```

```
> docker run –it ubuntu:20.10 bash
```

htl Hollabrunn

docker (Client)

- Webserver nginx
  - Ohne Prozessname wird der definierte Standard-Prozess gestartet
  - `> docker run -it nginx`
  - entkoppelter Start eines Containers (detached)
  - `> docker run -d nginx`
  - Docker Prozesse auflisten (Name Adjektiv + IT-Bekanntheiten)
  - `> docker ps`

htl Hollabrunn

docker (Client)

- Logs mit Containernamen überprüfen

```
> docker logs adjective_name
```

- Container mit benutzerdefinierten Namen starten (Reihenfolge wichtig! Imagename zum Schluss. Überprüfung mit `docker ps –a`)

```
> docker run –d --name webserver nginx
```

- Live-Weiterleitung der Logs mit der Option `--follow` bzw. `-f`

- Logs mit Container-Id überprüfen

```
> docker logs <id>
```

htl Hollabrunn

docker (Client)

- Docker-Container stoppen (grazeful shutdown)

```
> docker stop <id/name>
```

- Docker-Container ohne Wartezeit beenden

```
> docker kill <id/name>
```

- Docker-Container entfernen
- `> docker rm <id/name>`

- Docker-Images entfernen

```
> docker rmi ubuntu:20.10
```

- Docker-Container und Images entfernen (nur nicht verwendeten Objekte)

```
> docker system prune --all --volumes
```

htl Hollabrunn

docker (Client)

- Docker-Container Portforwarding `-p <Hostport>:<Containerport>`

```
> docker run -d --name webserver -p 8080:80 nginx
```

- Check der Weiterleitung

```
> docker ps
```

PORTS 0.0.0.0:8080->80/tcp

- Weiterleitung eines Volumes (Verzeichnis vom Host)

```
-v <Absoluter Pfad>:/usr/share/nginx/html
```

Volume-Pfad unter Windows: `/c/users/admin/www`

htl Hollabrunn

Docker-Images erstellen

- Node.js App im Docker-Container
- Dockerfile

  FROM node:19.0.0

  ADD . .

  CMD [„node“, „app.js“]

```
> docker build <pathOfDockerfileDirectory>
```

- Schlankeres Docker-Image das nicht auf Debian basiert

  FROM node:19.0.0-alpine

```
> docker run –d –p 3000:3000 -e MESSAGE='Hello World from
Docker!' <imageId>
```

htl Hollabrunn

Docker-Image mit `npm install`

- Ausnahmen der Volume
- .dockerignore

  node_modules

- Dockerfile

  USER node

  WORKDIR /home/node

  ADD --chown=node:node . /home/node

  RUN npm install

```
> docker run –d –p 3000:3000 --name my-app -e
MESSAGE='Hello World from Docker!' <imageId>
```

htl Hollabrunn

postgres

- Run Container
  - Port-Forwarding für Port 5432 im Container
  - Passwort für User "postgres" in Umgebungsvariable POSTGRES_PASSWORD setzen

```
> docker run -d -p 5432:5432
   -e POSTGRES_PASSWORD=postgres
   -v /c/db_data/postgresdb:/var/lib/postgresql/data
   --name postgresdb postgres
```

- bash im Container ausführen und mit DB verbinden

```
> docker exec –it postgresdb bash
> psql –U postgres
> SELECT version();
> \q
```

htl Hollabrunn

mysql

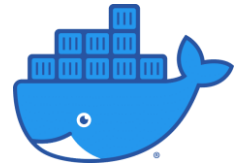
- Run Container
  - Port-Forwarding für Port 3306 im Container
  - Root-Passwort in Umgebungsvariable MYSQL_ROOT_PASSWORD setzen

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

- bash im Container ausführen und mit DB verbinden

```
> docker exec -it mysqldb bash
> mysql -u root -pmysql
> SHOW VARIABLES LIKE '%version%’;
> exit
```

htl Hollabrunn



mariadb

- Run Container
  - Port-Forwarding für Port 3306 im Container
  - Root-Passwort in Umgebungsvariable MYSQL_ROOT_PASSWORD setzen

```
> docker run -d -p 3307:3306 -e MYSQL_ROOT_PASSWORD=mariadb --
name mariadb mariadb
```

- bash im Container ausführen und mit DB verbinden

```
> docker exec -it mariadb bash
> mysql -u root -pmariadb
> SELECT version();
> exit
```

htl Hollabrunn

mongo

- Run Container
  - Port-Forwarding für Port 27017 im Container
  - Authentication mit Umgebungsvariablen konfigurieren

    MONGO_INITDB_ROOT_USERNAME

    MONGO_INITDB_ROOT_PASSWORD

```
> docker run -d -p 27017:27017
    -e MONGO_INITDB_ROOT_USERNAME=root
    -e MONGO_INITDB_ROOT_PASSWORD=mongodb
    --name mongodb
    mongo
```

htl Hollabrunn

influxdb

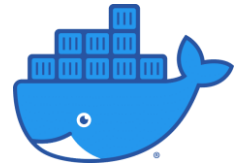
- Run Container
  - Port-Forwarding für Port 8086 im Container
  - Authentication mit Umgebungsvariablen konfigurieren

```
> docker run -d -p 8086:8086 --name influxdb influxdb

> docker exec influxdb influx setup `
      --username admin `
      --password influxdb `
      --org influxorg `
      --bucket databucket `
      --force
```

http://localhost:8086

htl Hollabrunn



DB clients

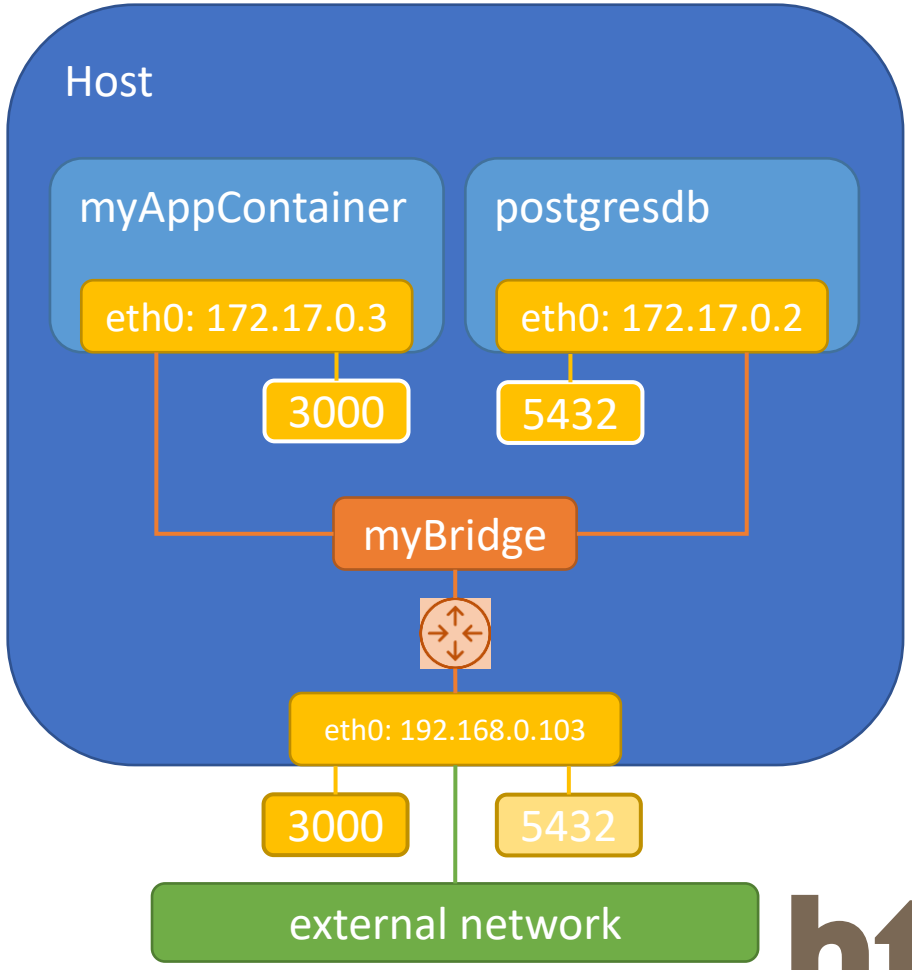
- DBeaver - https://dbeaver.io/download/
  - free multi-platform database tool
- HeidiSQL - https://www.heidisql.com/
  - MySQL, MariaDB, PostgesSQL, MS SQL, SQLite, …
- MongoDB Compass, CLI
  https://www.mongodb.com/try/download/compass

htl Hollabrunn

network drivers

- Container kommunizieren mit der Zuordnung eines Netzwerks.

- **network driver**:
  - bridge
    - privates internes Netzwerk von Containern
    - Isolation von Containern die nicht mir dem bridge-network verbunden sind
    - nur am selben docker-deamon-host möglich



htl Hollabrunn

Network

```
> docker run -d -p 5432:5432
-e POSTGRES_PASSWORD=postgres
-v /c/db_data/postgresdb:/var/lib/postgresql/data
--network myBridge
--name postgresdb
postgres

> docker run -d -p 3000:3000
-e message='Holidays!'
-e dbhost='172.17.0.2'
--network myBridge
--name myAppContainer
mynodeimg
```