Zusammenfassung Docker Kompakt Schulung

02.12.2019 - 04.12.2019

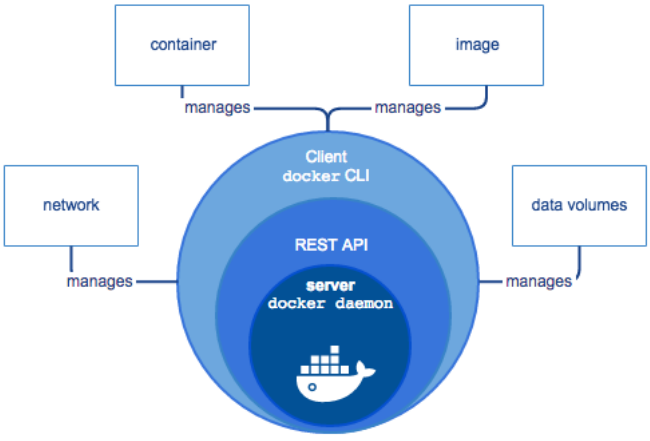
Tag 1

An diesem Tag haben wir uns hauptsaechlich beschaeftigt mit

- Docker Architektur
- Unterschied Container zu VM
- Images vs. Container
- docker CLI
  - build
  - run
  - push/pull
  - volume
  - ps
  - etc...
- Docker Persistent Storage
- Dockerfiles
- Images auf eigene Registry pushen

Docker Architektur

Docker besteht im Kern aus den drei Teilen:



Daemon Verwaltet Container/Netzwerke/Volumes

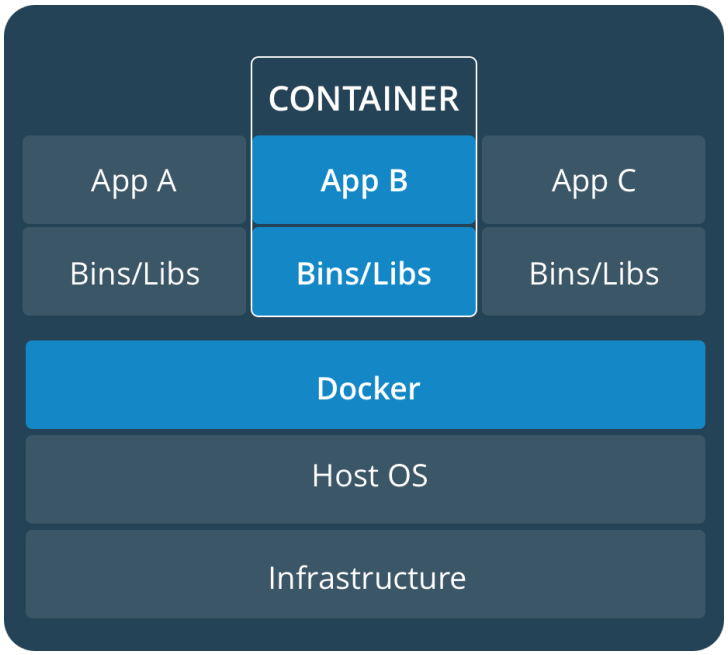
REST Api Wird von der Engine bereitgestellt, gewoehnlich als Socket und dient der Kontrolle dieser

CLI Das "docker" Kommando an sich welches die Befehle ueber die Rest API an die Engine absetzt

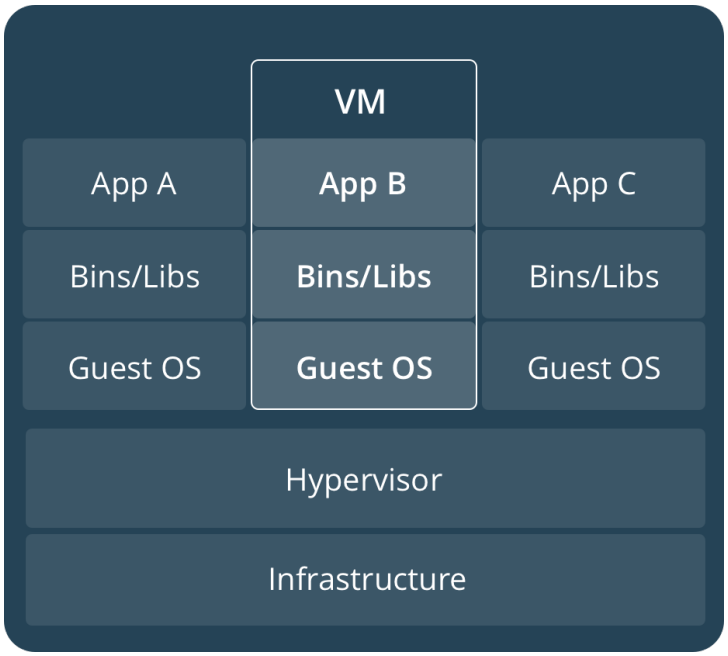
Weitere Infos:
https://docs.docker.com/engine/docker-overview/

Unterschied Container VM

Hier haben wir besonders hervorgehoben das Container im Kern erstmal ein Prozess sind und wir keine Interpretationsschicht wie bei VMs haben.
Ein Vergleich war: Haus (VM) vs. Wohnung im Mehrfamilienhaus (Container)



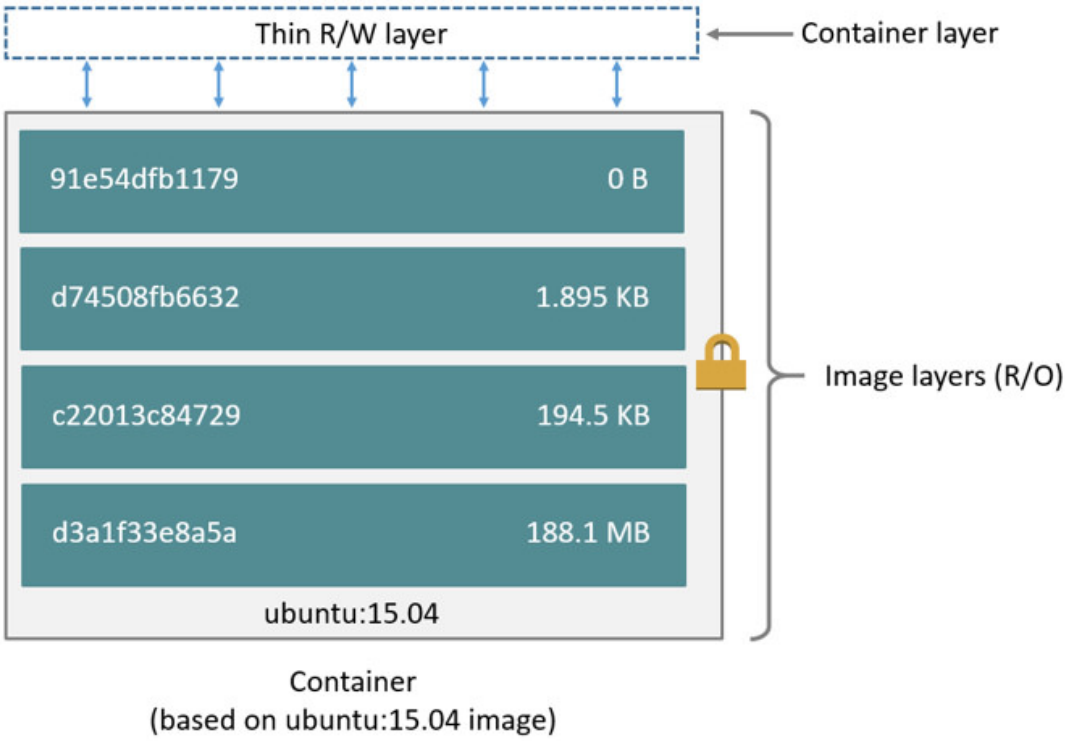
Container teilen sich die Infrastruktur des Hosts vor allem in Form des Kernels, besitzen aber ein eigenes Dateisystem.



VMs muessen ein komplettes Betriebssystem inklusive Kernel mitbringen und haben den zusaetzlichen overhead eines Hypervisors der sie und ihre Ressourcen verwaltet.

Images vs. Container

Images bestehen aus mehreren Layern und bilden die Basis von Containern. Images an sich werden nicht ausgefuehrt. Aus Images werden Container erzeugt, indem ueberhalb des Images ein beschreibarer Layer angelegt wird in dem dann wiederrum ein Prozess ausgefuehrt wird.



Docker CLI

Die docker CLI ist das Hauptwerkzeug um mit dem Docker daemon zu interagieren. Wir haben uns verschiedene Kommandos naeher angesehen. Einige Beispiele folgen.

Hello World

```
docker run hello-world
```

Erstellt einen Container auf Basis des hello-world Images (dabei impliziert :latest) laesst diesen laufen und haengt sich an den Standardoutput bis zu dessen Terminierung an.

Run Container

```
docker run -it ubuntu
```

Erstellt einen Container auf Basis des ubuntu Images (dabei impliziert :latest) und haengt sich nach start interaktiv ein (-it)

Run Container Background

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

Erstellt einen Container auf Basis des nginx Images (dabei impliziert :latest) laesst diesem im Hintergrund laufen (-d) und mappt den Host port 8080 auf den port 80 im Container (-p 8080:80). Benannt wird der Container "webserver" (–name=webserver)

Attach to container logs

```
docker logs -f webserver
```

Folgt dem log output des Containers mit Namen webserver

Show running containers

```
 docker ps
```

Zeigt alle momentan laufenden Container an

Show all containers

```
docker ps -a
```

Zeigt alle Container auf dem System an

Stop container

```
docker stop webserver
```

Stoppt den Container mit Namen webserver

Start container

```
docker start webserver
```

Startet den Container mit Namen webserver. Das Kommando dass der Container als Hauptprozess ausfuehrt kann hier nicht geaendert werden.

Remove container

```
docker rm -f webserver
```

Loescht den Container mit Namen webserver. -f impliziert dass dieser Notfalls wenn nicht schon gestoppt vorher gestoppt wird./

Pull image

```
docker pull mariadb:10.3
```

Zieht sich das Image mariadb mit Tag 10.3 von der Registry (default ist hier hub.docker.com)

Mount hostdirectory

```
docker run -d -e MYSQL_ROOT_PASSWORD=123 -p 5555:3306 \
--mount type=bind,source=/home/user/db-data,target=/var/lib/mysql mariadb
```

Laesst einen Container auf Basis des mariadb Images im Hintergrund mit Umgebungsvariable (-e) laufen und haengt das Hostverzeichnis /home/user/db-data in den Container unter /var/lib/mysql ein. Der Server wird unter Port 5555 auf dem Host bereitgestellt.

Create volume

```
docker volume create db-storage
```

Erstellt ein Volume (von docker verwalteter Speicherbereich).

Mount volume

```
docker run -d -e MYSQL_ROOT_PASSWORD=123 \
--mount type=volume,source=db-storage,target=/var/lib/mysql --name=database mariadb
```

Laesst einen Container auf Basis des mariadb Images im Hintergrund mit Umgebungsvariable (-e) laufen und haengt das Hostverzeichnis /home/user/db-data in den Container unter /var/lib/mysql ein.

Run command in Container

```
docker exec -it database mysql
```

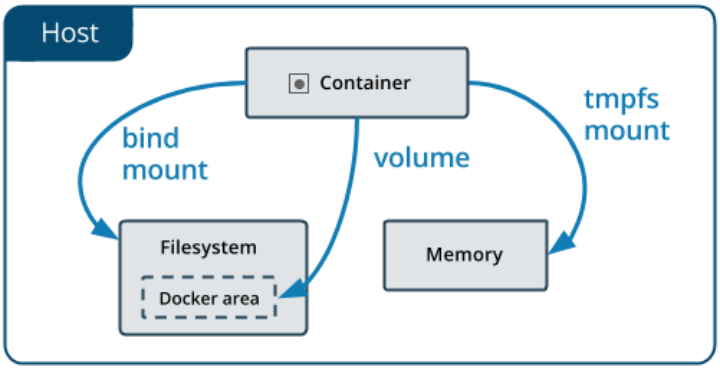
Fuehrt das Kommando mysql im Container database interaktiv (-it) aus. Wenn dieser eine lokale mysql Datenbank mit Socket bereitstellt wird direkt damit verbunden.

Weitere Informationen

Weitere Informationen ueber die CLI im allgemeinen entweder durch "docker –help" bzw. "docker **commandName** –help" oder in den docs https://docs.docker.com/engine/reference/run/

Docker Persistent Storage

Mit docker gibt es grundsaetzlich zwei Moeglichkeiten Daten auch ueber Container Lebenszeit hinaus zu erhalten. Das einbinden von Hostverzeichnis (--mount type=bind...) oder das einbinden von Volumes (--mount type=volume...). Der Unterschied ist das Volumes von Docker verwaltet werden und direkt per vergebenem Namen referenziert werden koennen. Hostverzeichnisse koennen sich auch ueberall auf dem Host befinden, Volumes leben auf dem Host immer unter /var/lib/docker/volumes.



Volumes bieten sich unter anderem dann an wenn der Speicher lediglich von Containern genutzt werden soll, oder das Layout auf den Hosts nicht bekannt ist man aber den selben deploy Befehl verwenden moechte.

Dockerfiles

Dockerfiles sind eine Moeglichkeit Images zu erstellen. Dabei wird eine Reihe von Kommandos nacheinander beginnend auf einem Basisimage ausgefuehrt und nach jedem Kommando ein Layer erzeugt. Dockerfiles werden vom Kommando "docker build" benutzt.

```
docker build . -t my-new-image
```

Das Kommando baut ein Image auf Basis des Dockerfiles im aktuellen Verzeichnis und vergibt den Tag my-new-image an das resultierende Image.
Weitere Details zu Dockerfiles und deren Aufbau

https://docs.docker.com/engine/reference/builder/
https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

Images auf eigene Registry pushen

Damit Images auf die eigene Registry gepusht werden koennen muessen sie erst entsprechend getaggt werden.

```
docker tag mariadb:10.4 myregistryhostname:5000/mariadb:10.4
```

Taggt das mariadb Image mit Tag 10.4 von DockerHub (oder lokal gebaut) auf den Tag myregistryhostname:5000/mariadb:10.4 der die URL der Zielregistry enthaelt.

```
docker push myregistryhostname:5000/mariadb:10.4
```

Pusht den neuen Tag auf die eigene Registry

Tag 2

An diesem Tag haben wir uns hauptsaechlich beschaeftigt mit

- Networking in Docker
- docker-compose
- Logging in Docker mit Graylog
- Healthchecks

Networking in Docker

Docker hat per default 3 Netzwerke

default Das Standardnetzwerk dem Container angehaengt werden wenn nichts anderes spezifiziert

host Spezielles Netzwerk das dem Container den Netzwerkstack des Hosts uebergibt (Container sieht sich aus Netzwerksicht wie Host)

none Kein Netzwerkzugang fuer den Container

Zu diesen Netzwerken koennen jetzt Benutzerdefinierte Netzwerke hinzugefuegt werden mit dem Kommando:

```
docker network create myNetwork
```

Erzeugt das Netzwerk mit Namen myNetwork. Neue und bestehende Container koennen nun an dieses Netzwerk angehaengt werden. Der Vorteil von eigenen Netzwerken ist eine gewisse Isolation gegenueber anderen Netzwerken und das anderen Container im selben Netzwerk mit Containername erreichbar sind.

Attach docker container to network

```
docker network connect myNetwork database
```

Haengt den Container database an das Netzwerk myNetwork an.

Run docker container on network

```
docker run -it --network=myNetwork --rm mariadb mysql -h database --password=123
```

Startet einen Container im Netz myNetwork der mit dem mysql client auf eine Datenbank auf dem host database verbindet.

docker-compose

docker-compose erlaubt es ganze Anwendungsarchitekturen durch ein Definitionsfile zu skripten. Dabei koennen automatisiert Container mit bestimmten Einstellungen erzeugt werden, sowie Netzwerke und Volumes.

docker-compose CLI

Der docker-compose Befehl an sich verarbeitet die .yml Datei und erzeugt/updated die darin enthaltene Architektur. Dabei ist der Name des Verzeichnises (nicht der Pfad dorthin) der Namespace unter dem die Container erzeugt werden (_ wird dabei ignoriert). Ausnahme sind explizit vergebene Namen an Container durch z.b. die container_name Direktive.

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

Das erzeugt eine Architektur aus einem .yml file und startet sie im Hintergrund (-d)

```
docker-compose logs
```

Zeigt die Logs der Services an.

```
docker-compose logs serviceName
```

Zeigt die logs eines Services an.

```
docker-compose exec serviceName command
```

Fuehrt ein Kommando im Service interaktiv aus.

```
docker-compose down
```

Zerstoert die Architektur. Volumes werden nicht zerstoert ausser man gibt -v als Option mit.

Compose Beispiel

```
version: "2"
services:
  # MongoDB: https://hub.docker.com/_/mongo/
  mongodb:
    image: mongo:3
  # Elasticsearch: https://hub.docker.com/_/elasticsearch/
  elasticsearch:
    image: elasticsearch:6.8.5
    environment:
      - http.host=0.0.0.0
      - transport.host=localhost
      - network.host=0.0.0.0
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    mem_limit: 1g
  # Graylog: https://hub.docker.com/r/graylog/graylog/
  graylog:
    image: graylog/graylog:3.1
    environment:
      # CHANGE ME (must be at least 16 characters)!
      - GRAYLOG_PASSWORD_SECRET=somepasswordpepper
      # Password: admin
      - GRAYLOG_ROOT_PASSWORD_SHA2=8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918
      - GRAYLOG_HTTP_EXTERNAL_URI=http://127.0.0.1:9000/
    links:
      - mongodb:mongo
      - elasticsearch
    depends_on:
      - mongodb
      - elasticsearch
    ports:
      # Graylog web interface and REST API
      - 9000:9000
      # Syslog TCP
      - 1514:1514
      # Syslog UDP
      - 1514:1514/udp
      # GELF TCP
      - 12201:12201
      # GELF UDP
      - 12201:12201/udp
```

Erstellt einen Anwendungsstack aus Mongodb, Elasticsearch und Graylog. Wobei die freigegeben Ports sowie eventuelle Speicherlimits mit angegeben sind. Damit Graylog auf 12201 gelf akzeptiert muss in Graylog dazu erst ein Input angelegt werden ueber as Webinterface.

Logging in Docker mit Graylog

Mit dem Graylog stack aus dem vorherigen Beispiel koennen wir nun Docker Container dorthin loggen lassen. Dazu muss zunaechst ueber das Graylog Webinterface ein Input fuer gelf auf 12201 TCP angelegt werden. Dann sind folgende Kommandos moeglich:

```
docker run -d --log-driver=gelf --log-opt gelf-address=tcp://localhost:12201 -p 8090:80 nginx
```

Laesst einen nginx Server auf port 8090 auf dem Host laufen. Anfragen an nginx werden nun in Graylog geloggt und sind dort sichtbar.

Logging in Compose

```
version: "2"
services:
```

```
  # MongoDB: https://hub.docker.com/_/mariadb/
  mariadb:
    image: mariadb
    environment:
      MYSQL_ROOT_PASSWORD: 123
    logging:
      driver: gelf
      options:
        gelf-address: "tcp://localhost:12201"
  nextcloud:
    image: nextcloud/nextcloud
    ports:
      - 9000:80
    logging:
      driver: gelf
      options:
        gelf-address: "tcp://localhost:12201"
```

Loggt eine mariadb und eine nextcloud Instanz in Graylog.

Healthchecks

Healthchecks sind Kommandos die in Regelmaesigen Abstaenden durch docker in Containern ausgefuehrt werden und deren Zustand (gesund/ungesund) wiedergeben sollen. Dieser Status kann dann durch Monitoring tools ausgelesen werden oder in docker ps eingesehen werden. Healthchecks koennen entweder im Dockerfile eingebaut werden:

```
FROM nginx

RUN apt-get update && apt-get install curl

HEALTHCHECK --interval=1s --retries=3 CMD curl localhost:80
```

oder nachtraeglich zur Laufzeit hinzugefuegt werden:

```
docker run --health-cmd "healthCommand" -d imageName
```

Tag 3

An diesem Tag haben wir noch einmal rekapituliert und sind dann noch vertiefend eingegangen auf

- Logging mit ELK-Stack (Logstash) oder ELF-Stack (fluentd)
- Monitoring
- Load-Balancing
- Cluster Applikation (docker swarm und Ausblick OpenShift)

Logging mit ELK und ELF Stack

Logging mit diesen Stacks funktioniert aehnlich dem Logging mit Graylog. Fuer den ELF Stack wird der fluentd logging driver verwendet. Logstash braucht das gelf Input plugin.

Monitoring

Docker internes Monitoring ist moeglich ueber die Kommandos

```
docker ps
```

Zeigt alle laufenden Container und deren Gesundheitszustand

```
docker top
```

Zeigt alle laufenden Prozesse in Containern

```
docker stats
```

Zeigt detaillierte Statistiken zum aktuellen Ressourcenverbrauch von Containern
Externe Monitoringtools wie cAdvisor oder Prometheus koennen ebenfalls angebunden werden

Load-Balancing

Per default werden Container gleichbehandelt und bekommen zu gleichen Teilen Ressourcen. Wenn einem Container eine feste Menge an Ressourcen zugewiesen werden soll geht das zur Laufzeit etwa mit:

```
docker run -it --cpus 1.5 ubuntu-stress stress -c 4
```

Laesst CPU Auslastung von 1.5 CPUs zu (in Unix Sprache eine maximale Auslastung von 150%)

```
docker run -it --memory 1G ubuntu-stress stress --vm-bytes 2G --vm-hang 0
```

Erlaubt maximal ein Gigabyte Speicherallokation bevor der Container beendet wird (dieses Beispiel beendet sich sofort)
Ist lediglich gewuenscht eine gewisse Verteilung zwischen den Containern einzustellen so ist das moeglich durch z.B.

```
 docker run -it --cpu-shares 2048 ubuntu-stress stress -c 4
```

Dabei bekommt dieser Container 2048 "shares" zugewiesen. Der default Wert ist 1024. Die CPU Auslastung wird jetzt nicht begrenzt solang CPU Ressourcen vorhanden sind. Sobald nicht genuegend Ressourcen fuer alle Container zur Verfuegung stehen werden diese nach den "shares" aufgeteilt.

Cluster Applikation

Bei Clusteranwedungen kann eine automatische Skalierung von Services ueber verschiedene Cluster Nodes hinweg erfolgen. Beispiele fuer solche Clusteranwendungen sind z.b. docker swarm (Docker nativer Cluster) oder aber OpenShift. Die Terminologie ist dabei recht unterschiedlich zwischen den verschiedenen Anwendungen obwohl einige Konzepte gemein sind.
Fuer Openshift self-learn:
https://learn.openshift.com/ Direktes lab:
https://learn.openshift.com/playgrounds/openshift311/ dort auf Dashboard wechseln falls man die Webkonsole will.