Docker

Docker
- Grundlagen
- Begriffe
  - Image
  - Container
  - Layer
  - Dockerfile
  - Repository
  - Registry
  - Docker Hub
- Docker CLI
  - Regestry Befehle
  - Image Befehle
  - Container Befehle
  - Beispiele
    - Hello World
    - Einzelnen Befehl in Container Ausführen
    - CLI zugriff in Laufenden Container
    - Erstellen eines Images mit einem Dockerfile
    - Ausführen eines Images und freigabe eines Ports
- Dockerfile
  - Befehle
  - Beispiele
    - Dockerfile Alpine SSH Server
    - Dockerfile zum ausführen einer jar Datei

Grundlagen

Bei Docker handelt es sich um eine Open Source Software zu Anwendungs Isolation mit Containervirtualisierung. Die eingesetzten Docker Container basieren auf Linux weshalb ein Linux Kernel zum ausführen nötig ist. Mithilfe des Hyper-V Standarts oder VirtualBox kann es auch auf anderen Systemen zum laufen gebracht werden.

Begriffe

Image

Bei einem Image handelt es sich um ein Speicher Abbild eines Containers. Es ist aus mehreren Speicher geschützten Layern aufgebaut und bildet die Grundlage für Container die man konstruieren kann. Images können auf mehreren Docker Systemen als Grundlage verwendet werden und sind daher portabel.

Container

Ein Container stellt ein gestartetes Image da. Hat der Container keine Aufgabe mehr auszuführen wird er beendet. Container haben die Fähigkeit untereinander zu kommunizieren oder Dienste nach außen anzubieten.

Layer

Ein Layer stellt eine Änderung an einem Image da die z.B eine hinzugefügte Datei oder das ausführen eines Befehls enthält. Mit Hilfe der Layer lassen sich also alle Änderungen an einem Image zurück verfolgen.

Dockerfile

Ein Dockerfile ist eine Textdatei in der Änderungen an einem Image beschrieben. Sie werden dem Image als Layer hinzugefügt und bilden so den Lauffähigen Container. Mit Hilfe des Dockerfiles kann so ein verändertes Image als Container auf verschiedenen Systemen laufen gelassen werden.

Repository

Ein Repository ist eine Sammlung an Images die ein Tag besitzen das meist der Versionsnummer entspricht.

Registry

Eine Registry bildet eine zentrale Bibliothek zur Verwaltung von Docker Images in Repositorys bekannt sind z.B. der Docker Hub oder die Docker Registry zum selber Hosten

Docker Hub

Der Docker-Hub ist eine Bibliothek für die Repositorys von Docker Images die von Software Entwicklern, Open Source Projekten und der Community bereit gestellt werden. Er kann auch selbst gehostet werden wenn man seine Images z.B. nicht auf die Server von Docker hoch laden möchte.

Docker CLI

```
docker info
```

Zeigt Systemweite Informationen über die Docker Instalation an

---

Regestry Befehle

```
docker login [OPTIONS] [SERVER]
docker logout [SERVER]
```

Login oder Logout von einer Docker Regestry

```
docker pull [OPTIONS] NAME[:TAG]
docker push [OPTIONS] NAME[:TAG]
```

Downloaded oder Uploaded ein Repository oder Image von einer Docker Regestry

---

```
docker search TERM
```

Durchsuche den DockerHub nach Images

Image Befehle

```
docker images
```

Liste alle Lokal verfügbarer Images

```
docker build [OPTIONS] PATH | URL |
```

Erstellen eines Images aus einem Dockerfile

> --no-cache Beim erstellen keinen Cache anlegen
>
> -t Benennen des erstellten Image

```
docker rmi [OPTIONS] IMAGE [IMAGE...]
```

Entfernen eines oder mehrerer Docker Images

Container Befehle

```
docker ps [OPTIONS]
```

Auflisten aller Container

> -a Anzeigen aller Container auch die die nicht ausgeführt werden
>
> -n Anzeigen der n letzten erstellten Contaienrn
>
> -l Anzeigen des letzten erstellten Containers
>
> -s Anzeigen der Container größe

```
docker create [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Erstellen eines neue Containers

```
docker rename CONTAINER NEW_NAME
```

Umbenennen eines Docker Containers

```
docker rm [OPTIONS] CONTAINER [CONTAINER...]
```

Entfernen eines Docker Containers

```
docker run  [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Ausführen eines Consolen Befehls in einem neu startenden Container

> --name string Festlegen des Namens eines Containers
>
> --rm Automatisches Entfernen des Contaienrs nach beendigung
>
> -i Interaktiver Shell zugriff in Container
>
> -t Verfenden einer Pseudo seriellen Schnittstelle
>
> -d Führt Container im Hitnergrund aus und gibt Container ID aus
>
> --publish=2222:22 Freigeben eines Container Ports zum Host
>
> --publish-all Freigeben aller ungeschützter Container Ports auf zufällige Host Ports
>
> --volum

```
docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
```

Ausführen eines Consolen Befehls im Container

> -i Interaktiver Shell zugriff in Container
>
> -t Verfenden einer Pseudo seriellen Schnittstelle
>
> -d Fürt Befehl im Hintergrund in Container aus

```
docker start [OPTIONS] CONTAINER [CONTAINER...]
```

Starten von Docker Containern

```
docker paus CONTAINER [CONTAINER...]
```

Pausieren von allen Prozessen in einem Docker Containern

```
docker unpause CONTAINER [CONTAINER...]
```

Weiter ausführen von allen Prozessen in einem Docker Container

---

```
docker restart CONTAINER [CONTAINER...]
```

Neustarten von Docker Containern

```
docker kill CONTAINER [CONTAINER...]
```

Beenden von Docker Containern erzwingen

```
docker port CONTAINER [PRIVATE_PORT[/PROTO]]
```

Anzeigen von Port zuweisungen für einen Container

```
docker top CONTAINER [ps OPTIONS]
```

Anzeigen der Laufenden Prozesse eines Containers

Beispiele

Hello World

```
$ docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:d58e752213a51785838f9eed2b7a498ffa1cb3aa7f946dda11af39286c3db9a9
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest

$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
hello-world         latest              bf756fb1ae65        6 months ago
13.3kB

$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
4e9a296f800f        hello-world         "/hello"            23 seconds ago
Exited (0) 22 seconds ago                       angry_gauss

$ docker rm angry_gauss
angry_gauss
```

```
$ docker rmi hello-world:latest
Untagged: hello-world:latest
Untagged: hello-
world@sha256:d58e752213a51785838f9eed2b7a498ffa1cb3aa7f946dda11af39286c3db9a9
Deleted: sha256:bf756fb1ae65adf866bd8c456593cd24beb6a0a061dedf42b26a993176745f6b
Deleted: sha256:9c27e219663c25e0f28493790cc0b88bc973ba3b1686355f221c38a36978ac63
```

Einzelnen Befehl in Container Ausführen

```
$ docker run -d --rm alpine-openssh-server

$ docker exec -d awesome_cray touch /file
```

CLI zugriff in Laufenden Container

```
$ docker run -d --rm alpine-openssh-server

$ docker exec -it inspiring_tharp /bin/ash
/ #
```

Erstellen eines Images mit einem Dockerfile

Erstellt ein Image namens alpine-openssh-server aus dem Dockerfile im aktuellen Verzeichnis

```
docker build --no-cache -t alpine-openssh-server .
```

Ausführen eines Images und freigabe eines Ports

Starten des Image alpine-openssh-server und freigabe des internen Ports 22 auf den Host Port 2222

```
docker run --rm --publish=2222:22 alpine-openssh-server
```

Dockerfile

Befehle

```
FROM alpine:latest
```

Auswahl des zugrunde liegenden Image für das neue zu erstellende Image

---

```
MAINTAINER Reiner Zufall <mail@nurks.do>
```

Autor des Image vermerken

```
RUN apk update
```

Führt einen übergebenen Parameter als Shell Befehl aus

> Dieser wird intern zu ""/bin/ash -c apk update" übersetzt wodurch absolute Pfadangaben sicherer sidn zu verwenden

---

```
CMD ["/usr/sbin/sshd", "-D"]
```

Legt das Standart Commando fest was beim ausführen des Containers ausgeführt wird

> Pro Container kann es nur einen CMD Commando geben. Bei mehreren ist nur das letzte gültig

---

```
EXPOSE 22
```

Gibt den Übergebenen Port des Containers frei

```
COPY identitys/identity.pub /home/user/.ssh/authorized_keys
```

Kopiert Eine Datei oder ein Verzeichnis aus dem Host File System in das des Containers

---

```
ADD http://source.file/url  /destination/path
```

Hat die selben fähigkeiten wie COPY kann als Source aber auch mit URLS umgehen.

```
WORKDIR /home
```

Ändert das aktuelle Working Directory da alle die Befehle RUN, CMD , ADD und COPY erstmal vom Home Directory ausgehen.

> Absolute Pfadangaben verbessern meist die wiederverwertbarkeit von Befehlen

Beispiele

Dockerfile Alpine SSH Server

```
FROM alpine:latest

#Update and install openssh
RUN apk update
RUN apk add openssh

#Change root Passwort or remove
RUN echo "root:|ichbineinpasswort|" | chpasswd
#RUN passwd -d root

#Add User
RUN adduser user -D
RUN echo "user:|ichbineinpasswort|" | chpasswd

#Add Public Key for user
COPY identitys/identity.pub /home/user/.ssh/authorized_keys
```

```
RUN chown -R user:user /home/user/.ssh
RUN chmod 700 /home/user/.ssh
RUN chmod 600 /home/user/.ssh/authorized_keys

#ssh_config ssh-banner ssh not sshd in alpine
COPY ssh-files/s* /etc/ssh/

#Copy Host-Keys
COPY hostkey/* /etc/ssh/

#Open Ports
EXPOSE 22

#Start openssh server
CMD ["/usr/sbin/sshd", "-D"]
```

Dockerfile zum ausführen einer jar Datei

Benötigt eine .jar Datei

Beispiel HelloWorld.jar Prints "HelloWorld" auf der Commando Zeile

```
#OpenJDK image
FROM openjdk:12

#Copy jar File
ADD HelloWorld.jar HelloWorld.jar

#Run jar on start up
CMD java -jar HelloWorld.jar
```