Einführung Docker (V0.2)

DI. H. Lackinger

2018/2019

Contents

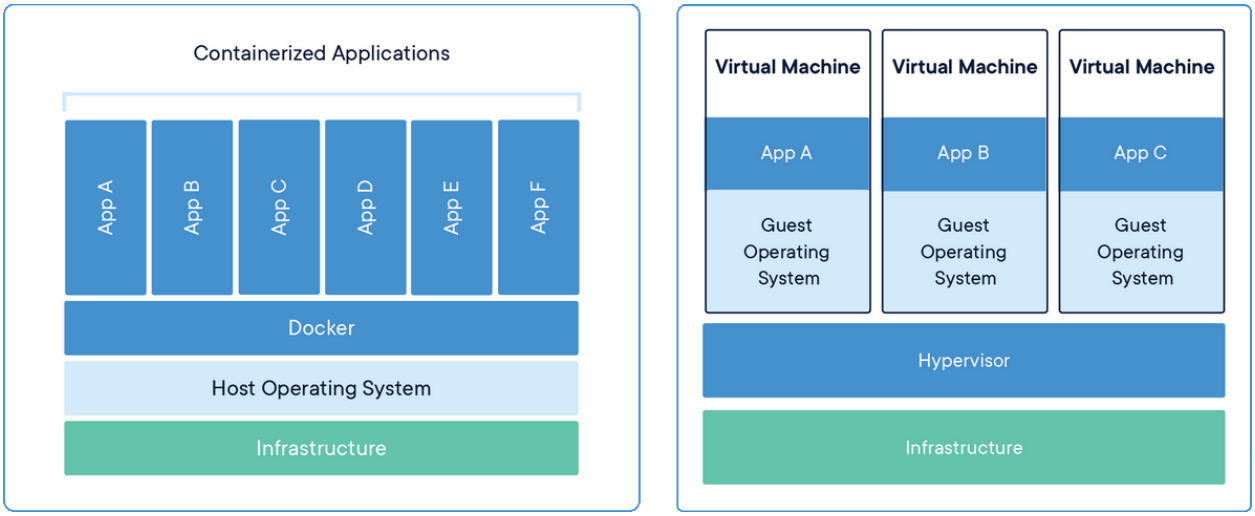
Links: https://docs.docker.com/

Docker Überblick

Was ist Docker?

- Docker erlaubt, Anwendungen in sogenannten **Containern** auszuführen, welche aufeinander aufbauen und untereinander kommunizieren können
- Docker basiert auf dem bereits älterem Konzept von LXC (Linux-Container), das für die **Isolation von Prozessen und Prozessgruppen** auf Kernel Ebene zuständig ist (jetzt libcontainer)
- Docker benötigt daher **nicht** für jeden Container ein eigenes vollständiges (virtualisiertes) Betriebssystem (im Vergleich zu Virtualbox, VMWare, . . .)
- Docker ist deshalb **ressourcenschonend** und erlaubt mehrere Container auf Commodity-Hardware laufen zu lassen

Container vs. Virtual Machines



Quelle: https://www.docker.com/resources/what-container

Verfügbarkeit

- Linux
- Mac (https://docs.docker.com/docker-for-mac/)
- Windows:
- Ab 64bit Windows 10 Pro, Enterprise und Education (https://docs.docker.com/docker-for-windows/install/)
- Bei älteren Windows-Versionen: Docker Toolbox, welches auf einem mittels VirtualBox virtualisiertem Linux basiert https://docs.docker.com/toolbox/overview/
- Diverse Cloud-Anbieter wie Amazon, Azure, . . .

Begriffe

- Docker-Host
  System, auf dem der Docker-Daemon und der Docker-Client läuft

- Docker-Client
  Setzt auf Docker-Daemon auf, der auf dem Docker-Host läuft. Wird durch die Binarydocker repräsentiert
- Dockerfile
  Textdokument mit der Beschreibung, wie ein Image erstellt werden soll. Dabei wird z.B. ein Basisbetriebssystem angegeben und eine Reihe von Kommandos, die ausgeführt werden sollen
- Docker-Image
  Schreibgeschützte Vorlagen, die eine Anwendung mitsamt allen Abhängigkeiten wie Bibliotheken, Hilfsprogrammen und statischer Daten zusammenfasst
- Container Sind lauffähige und nicht schreibgeschützte Varianten der Docker-Images
- Docker-Registry
  Enthält bereits fertige Docker-Images, die heruntergeladen und gestartet werden können (Docker-Hub)
- Docker Compose
  Tool, um mehrere Docker-Container mit einem Befehl bzw. Konfigurationsdatei zu administrieren|

Wichtige Docker Befehle

Starten/Stoppen von Docker engine

```
systemctl start/stop docker
```

Images

Image herunterladen (https://hub.docker.com/)

```
docker pull <image>
```

Images auflisten

```
docker images
docker image   ls
```

Ein Image löschen

```
docker rmi <image>
```

Alle Images löschen

```
docker rmi `sudo docker images -q`
```

Image inspizieren

```
docker image inspect
```

Container

Container erzeugen+starten

```
docker run ...
```

Beispiel: docker run -ti –entrypoint /bin/bash resin/raspberrypie-node

Container erzeugen

```
docker build
```

Container starten

```
docker start <id>
```

Container stoppen

```
docker stop <id>
```

Alle Container listen

```
docker ps -a
```

Laufende Container listen

```
docker ps
```

Container löschen

```
docker rm <id/name>
```

Alle Container löschen

```
docker rm $(docker ps -a -q)
```

Sonstige

command in einem laufenden Container starten

z.B. bash eines Containers starten

```
docker exec -it <id/name> bash
```

z.B. im Docker Container die mongo.cli starten

```
docker exec -it <id/name> mongo
```

Mit einem Container verbinden (login)

```
docker attach <id/name>
```

Vom Container wieder abmelden ohne den Container zu stoppen

```
<ctrl>P <ctrl>Q
```

Docker compose

Compose file

Mit einem Compose File können mehrere Container und deren Aufrufparameter verwaltet werden.

Beispiel: In diesem Compose File werden die 2 Container 'web' und 'redis' verwaltet. Der 'web'-Container wird mit dem Dockerfile im aktuellen Verzeichnis erstellt. Das Portmapping geht von 5000->5000. Der 'redis'-Container wird aus dem Standard-Imageredis:alpine von dockerhub gebuildet. Dier Versionsnummer gibt an, welche Version die Compose-File-Syntax ist.

Wichtig Es wird ein eigenes Netzwerk erzeugt, in dem sich die Services anmelden. Nun können die Container untereinander über die Service-Namen als nodename kommunizieren. z.B. http://web:5000

```
version: '3'
services:
  web:
    build: .
    ports:
     - "5000:5000"
  redis:
    image: "redis:alpine"
```

Wichtige Docker compose Befehle

Links: https://docs.docker.com/compose/reference/overview/#command-options-overview-and-help

```
docker-compose build # Bilden aller Services
docker-compose start # Starten aller Container
docker-compose stop  # Stoppen aller Container
docker-compose up    # Bilden und Starten aller Container
docker-compose down  # Stoppen und Löschen aller Container, Netzwerke, Images und Volumes
```

Beispiel: Angular App im Container

In das Projektverzeichnis wechseln

```
cd <angular project dir>
```

Dockerfile

```
FROM nginx:alpine

COPY nginx.conf /etc/nginx/nginx.conf

WORKDIR /usr/share/nginx/html
COPY dist/<project> .          # !!!!!!!! <project> nur ab Angular 6
```

Einache Konfiguration für nginx (nginx.conf) erstellen

```
http {
    server {
        listen 80;
        server_name  localhost;

        root   /usr/share/nginx/html;
        index  index.html index.htm;
        include /etc/nginx/mime.types;

        gzip on;
        gzip_min_length 1000;
        gzip_proxied expired no-cache no-store private auth;
        gzip_types text/plain text/css application/json application/javascript application/x-javascript

        location / {
            try_files $uri $uri/ /index.html;
        }
    }
}
```

Angular builden

```
ng build --prod
```

Eigenes Image (my-angular-app) builden

```
cd <angular-project-dir>  # muss Dockerfile und nginx.conf enthalten
docker image build -t my-angular-app .
```

Angular App im Container starten

Der Container-Port 80 wird auf den Host-Port 3000 gerouted

```
docker run -p 3000:80 my-angular-app
```

Angular App im Browser starten

```
http:\\localhost:3000
```

Mit Docker compose

Links: https://docs.docker.com/compose/overview/

docker-compose.yml erstellen

```
version: '3.1'

services:
    app:
        image: 'my-angular-app'
        build: '.'
        ports:
            - 3000:80
```

Angular App im Browser starten

```
docker-compose up
```

Beispiel: Java App im Container

Dockerfile

```
FROM openjdk:8-jre-alpine

RUN mkdir /app
COPY ${project.artifactId}.jar /app
COPY libs /app/libs

CMD ["java", "-jar", "/app/${project.artifactId}.jar"]
```

Java App builden

```
Netbeans -> build
```

Image (my-java-app) builden

```
cd <java-project-target-dir>  # muss Dockerfile und ./libs enthalten, ansonsten Dockerfile anpassen !!
docker image build -t my-java-app .
```

Java App im Container starten

```
docker run -p 8080:8080 my-angular-app
```

Mit Docker compose

docker-compose.yml erstellen

```
version: '3.1'

services:
    javaserver:
        image: 'my-java-app'
        build: '.'
        ports:
            - 8080:8080
```

Java App im Container starten

```
docker-compose up
```

Beispiel: Wildfly im Container

Dockerfile mit admin

```
FROM jboss/wildfly
RUN /opt/jboss/wildfly/bin/add-user.sh admin Admin --silent

# ADD standalone-custom.xml /opt/wildfly/standalone/configuration/     # add configuration
# ADD your-awesome-app.war /opt/jboss/wildfly/standalone/deployments/  # add application

CMD ["/opt/jboss/wildfly/bin/standalone.sh", "-b", "0.0.0.0", "-bmanagement", "0.0.0.0"]
```

Image builden

```
docker build --tag=jboss/wildfly-admin .
```

Deployment

Das Deployment von Applikationen (war-Files) kann entweder bereits im Dockerfile, oder später via Admin-Web-Console (bzw. CLI) durchgeführt werden

Wildfly im Container starten

```
docker run -p 8080:8080 -p 9990:9990 -it jboss/wildfly-admin
```

Aufruf im Browser

```
http://<docker-machine ip>:9990    // Admin
http://<docker-machine ip>:8080    // Public
```

Beispiel: MySQL und Adminer

docker-compose File: stack.yml

```
# docker-compose -f stack.yml up

# Use root/example as user/password credentials
version: '3.1'

services:

  db:
    image: mysql
    command: --default-authentication-plugin=mysql_native_password
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: example

  adminer:
    image: adminer
    restart: always
    ports:
      - 8080:8080
```

Images builden

```
docker-compose -f stack.yml up
```

Aufruf im Browser

```
http://<docker-machine ip>:8080    // Adminer
```

Beispiel: NodeJS im Container

Dockerfile

```
FROM node

WORKDIR /app

COPY . .
RUN npm install
EXPOSE 8080
CMD ["node", "server.js"]
```

Build Image

```
docker build -t my-node-app .
```

Container starten

```
docker run -p 8080:8080 my-node-app
```

Aufruf im Browser

```
http://<docker-machine ip>:8080    // Public
```

Beispiel: Application Stack

Docker-compose File

```
# run make from the commandline to build the java binaries. This will also start docker-compose

version: '2.0'

services:
  wildfly:
```

```
    build: docker/wildfly
    ports:
      - 8080:8080
      - 9990:9990
      - 8787:8787
    depends_on:
      - mysql
    environment:
      - MYSQL_ROOT_PASSWORD=root
  www:
    build: docker/nginx
    ports:
      - 80:80
    volumes:
      - ./www:/usr/share/nginx/html
    depends_on:
      - wildfly
  mysql:
    build: docker/mysql
    ports:
      - 3306:3306
    volumes:
      - mysql_data:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=root
  phpmyadmin:
    build: docker/phpmyadmin
    ports:
      - 5050:80
    depends_on:
      - mysql
    environment:
      - PMA_HOST=mysql
      - MYSQL_ROOT_PASSWORD=root
volumes:
mysql_data:
```

Beispiel: MongoDB im Container

Docker-compose File

Bei diesem Beispiel wird für die MongoDB kein eigenes Dockerfile angelegt, da alle benötigten Konfigurationen im Docker-compose File festgelegt werden.

```
version: '3'

services:
    webserver:
        build: ./webserver
        ports:
        - '5101:80'
        depends_on:
```

```
        - configserver
    configserver:
        build: ./configserver
        ports:
        - '5102:5102'
        depends_on:
        - mongodb
    mongodb:
        image: mongo
        ports:
        - '27017:27017'
```

Alle Container starten

```
docker-compose up
```

Beispiel: MariaDB im Container

Docker-compose File

Bei diesem Beispiel wird für die MariaDB kein eigenes Dockerfile angelegt, da alle benötigten Konfigurationen im Docker-compose File festgelegt werden.

```
version: "3.1"

services:
  time:
    image: authsampletimeservice
    container_name: time
    ports:
      - "8081:8081"
    networks:
      - back-tier
    # restart: always
    environment:
      DB_USER: test-user
      DB_PASS: test-pass
      DB_ADDR: mariadb
      DB_PORT: 3306
      DB_NAME: vinitortest
      USERVALIDATION_SERVICE_ADDR: auth
      USERVALIDATION_SERVICE_PORT: 8080

  auth:
    image: authsampleservice
    container_name: auth
    ports:
      - "8080:8080"
    networks:
      - back-tier
    # restart: always
```

```
    environment:
      DB_USER: test-user
      DB_PASS: test-pass
      DB_ADDR: mariadb
      DB_PORT: 3306
      DB_NAME: vinitortest
      USERVALIDATION_SERVICE_ADDR: localhost
      USERVALIDATION_SERVICE_PORT: 8080

  mariadb:
    image: mariadb:latest
    ports:
     - "3306:3306"
    container_name: mariadb
    networks:
      - back-tier
    # restart: always
    environment:
      MYSQL_ROOT_PASSWORD: 'root'

networks:
  back-tier:
```

Alle Container starten

```
docker-compose up
```

Beispiel: Go im Container

Dockerfile

```
FROM golang:1.10.0
RUN go get github.com/codegangsta/negroni \
           github.com/gorilla/mux \
           github.com/xyproto/simpleredis
WORKDIR /app
ADD ./main.go .
RUN CGO_ENABLED=0 GOOS=linux go build -o main .
```

Build Image

```
docker build -t my-go-app .
```

Container starten

```
docker run -p 8080:8080 my-go-app
```

Aufruf im Browser

```
http://<docker-machine ip>:8080    // Public
```

Beispiel: Programmieren in C

Starting the Container

Take the current working directory and mount it as a volume into the directory `/new` of the container. Finally log in and start an interactive session:

```
docker run -it -v "$PWD":/new prprubuntu:firsttry
```

Creating the Image

Docker File

Create a file called `Dockerfile` in a new directory and store the following lines there:

```
FROM ubuntu
RUN apt-get update
RUN apt-get upgrade -y
RUN apt install build-essential -y
```

Pushing the Image to DockerHub

1. A Docker login is needed
2. Log in on https://hub.docker.com/
3. Click on Create Repository.
4. Choose a name (e.g. `prprubuntu`) and a description for your repository and click *Create*
5. Log into DockerHub from the command line:

   `docker login --username=yourhubusername`

   `yourhubusername` is the name of your Docker login.
6. Tag your image:

   `docker tag bb38976d03cf yourhubusername/prprubuntu:firsttry`

   `bb38976d03cf` is the image id. `firsttry` is the tag name.
7. Push your image

   `docker push yourhubusername/prprubuntu`