

6^{1/2}

OOP – Linux

Exkurs: Docker

Prof. Dr.-Ing. Tenshi Hara
tenshi.hara@ba-dresden.de



GLIEDERUNG DER VORLESUNG

Einführung: Geschichte von Unix zu Linux

Kapitel 1: Allgemeines und Grundlagen

Kapitel 2: Arbeit mit der Kommandozeile

Kapitel 3: Boot-Vorgang und Systeminitialisierung

Kapitel 4: Ausgewählte Themen der Systemadministration

Kapitel 5: Ausgewählte Themen der Netzwerkkonfiguration

Kapitel 6: Anwendungsentwicklung unter/für Linux
Exkurs: Docker

Kapitel 7: Ausgewählte Themen zu Web-Servern

GRUPPENRECHTE SETZEN

Vermeidung von **sudo**-Zwang durch Erstellen einer **docker**-Gruppe

```
sudo groupadd docker  
sudo usermod -aG docker $USER
```

AUSFÜHRUNG IM BASIS-SYSTEM (UNIX)

Docker-Container

- müssen aus einem Repository gezogen werden
- führen das übergebene Kommando aus und terminieren dann

```
docker pull busybox  
docker run busybox echo "Hello World!"
```

- können in einer interaktiven Shell weiter laufen

```
docker run -it busybox sh
```

- werden nach Terminierung auf den Ausgangszustand zurück gesetzt

```
docker run -it busybox sh  
rm -rf bin  
exit  
docker run -it busybox sh
```

UMGANG MIT CONTAINERN

- laufende Container auflisten

```
docker ps
```

- alle Container auflisten

```
docker ps -a
```

- bestimmten (gestoppten) Container löschen

```
docker rm containerID
```

- alle gestoppten Container löschen

```
docker rm $(docker ps -a -q -f status=exited)
```

oder

```
docker container prune
```

LANGLAUFENDE PROZESSE

- Loslösen vom aktuellen Terminal mit Parameter **-d**

```
docker run -d -P --name einName containerPfad
```

- Kombinieren mit weiteren Parametern

-P zufällige Zuweisung von Ports für Kommunikation

-P o:i zuweisen einer Abbildung von **äußerem** zu **innerem** Port

--name Namenszuweisung um nicht mit IDs hantieren zu müssen

- Pfad zum Container sollte immer angegeben werden
(oft im selben Verzeichnis, also „.“)

NETZWERKKOMMUNIKATION

- Docker-Container sind voneinander isoliert
- Kommunikation nur über Netzwerk möglich
 - Liste vorhandener virtueller Netzwerke anzeigen
`docker network ls`
 - Standard-Netzwerk **bridge** ist gemeinsam genutzt und unsicher
`docker network inspect bridge`
 - eigenes, separates, virtuelles Netzwerk anlegen
`docker network create eigenesNetz`
- Container an eigenes, virtuelles Netzwerk binden
(Angabe des Bindetyps ist optional)

```
docker run -d --name einName --net eigenesNetz  
-P ... -e "discovery.type=single-node" ...
```

NETZWERKZUGRIFF AUF CONTAINERISIERUNG

Hier nur am Beispiel von systemd

```
sudo systemctl edit docker.service
```

Beispieldateiinhalte:

```
[Service]
ExecStart=/usr/bin/dockerd -H fd:// -H tcp://127.0.0.1:2375
```

anschließend Daemon neu starten

```
sudo systemctl daemon-reload
sudo systemctl restart docker.service
```

AUTOMATISCHES STARTEN VON DOCKER NACH DEM BOOTEN

- systemd

```
sudo systemctl enable docker
```

- upstart

```
echo manual | sudo tee /etc/init/docker.override
```

- chkconfig

```
sudo chkconfig docker on
```

DOCKERFILE: AUTOMATISIERUNG DES CONTAINER-SETUPS

```
# Basis
FROM python:3-onbuild
MAINTAINER John Doe (john.doe@example.com)
# Komponenten installieren
RUN apt-get -yqq update
RUN apt-get -yqq install python-pip python-deb curl gnupg
RUN curl -sL https://deb.nodesource.com/setup_8.x | bash
RUN apt-get install -yq nodejs
# Anwendungscode kopieren
ADD flask-app /opt/flask-app
WORKDIR /opt/flask-app
# weitere Komponenten
RUN npm install
RUN npm run build
RUN pip install -r requirements.txt
# Port veröffentlichen
EXPOSE 5050
# Anwendung starten
CMD ["python", "./app.py"] # → $python ./app.py
```

BAUEN MIT DEM DOCKERFILE

mit dem Dockerfile und dem Basis-Image lässt sich schnell ein eigenes Container-Image für die eigene Anwendung bauen

```
docker build -t Tag PfadZumDockerfile
```

es gilt dabei:

- **Tag** eindeutige Kennung des eigenen Images
(bei Docker-Host v.d.F. username/imagename, bspw. hara/uxuvs)
- der Pfad zum Dockerfile muss angegeben werden, bspw. „.“

DOCKER-COMPOSE: CONTAINER GEMEINSAM BETREUEN

docker-compose.yml

```
version: "3"
services:
  es:
    image: docker.elastic.co/elasticsearch/elasticsearch:6.3.2
    container_name: es
    environment:
      - discovery.type=single-node
    ports:
      - 9200:9200
    volumes:
      - esdata1:/usr/share/elasticsearch/data
  web:
    image: prakhar1989/foodtrucks-web
    command: python app.py
    depends_on:
      - es
    ports:
      - 5000:5000
    volumes:
      - ./flask-app:/opt/flask-app
volumes:
  esdata1:
    driver: local
```

DOCKER-COMPOSE

- Verarbeiten der docker-compose.yml durch Aufruf im Verzeichnis

```
docker-compose up
```

oder losgelöst mittels Detach

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

→ Docker-Compose erstellt automatisch ein eigenes virtuelles Netzwerk für die Container der Komposition

- Stoppen des Clusters

```
docker-compose down -v
```

LITERATURHINWEIS

Ein gutes Tutorial von Prakhar Srivastav finden Sie hier:

<https://docker-curriculum.com>

etwas technischer: <https://docs.docker.com/get-started/>