Docker-Praxis mit Ubuntu und Nextcloud

Netzwerke/Linux
Thomas Schmitt
17. Juli 2018

Lehrerinnenfortbildung Baden-Württemberg

1. Überblick / Eigenschaften von Docker

Abbildung 1: Docker Logo / Author: dotCloud, Inc. / lizenziert unter Apache License Version 2.0

1.1. Was ist Docker?[1]

Einführendes Video (Youtube, c't magazin):

- nachgehakt: Was hat es mit Containern, Docker & Co auf sich?

Merkmale von **Docker**:

- Produkt der Firma **Docker Inc.**, San Francisco, Kalifornien.
- Open-Source-Projekt, lizenziert unter Apache-2.0-Lizenz.
- Vereinfachte, vom Betriebssystem unabhängige Installation und Betrieb von Server-Anwendungen.
- Minimale Anforderungen bzgl. Abhängigkeiten an Host-System.
- Abschottung vom Betriebssystem durch bestimmte Kernelfunktionen:
  - *Control Groups* (cgroups):

1 Quelle: c't 16.05: Dr. Oliver Dietrich, Container - Apps für Server, S. 108ff

- beschränken den Zugriff von Prozessen auf Speicher-, I/O- und CPU-Ressourcen.
- *Namespaces*:
  - Prozess „sieht" nur einen Ausschnitt des Systems (vergleichbar *chroot*).
  - Prozesse, die außerhalb des Containers laufen, sind nicht erreichbar.

- Abschottungsmechanismen sind die technische Voraussetzung für den Betrieb von Containern.
- **Docker** stellt Plattform bereit, um containerisierte Anwendungen zu erstellen, zu verteilen und zu betreiben:
  - einheitliches Containerformat.
  - Verwaltungswerkzeuge.

1.2. Unterschied zu virtuellen Maschinen

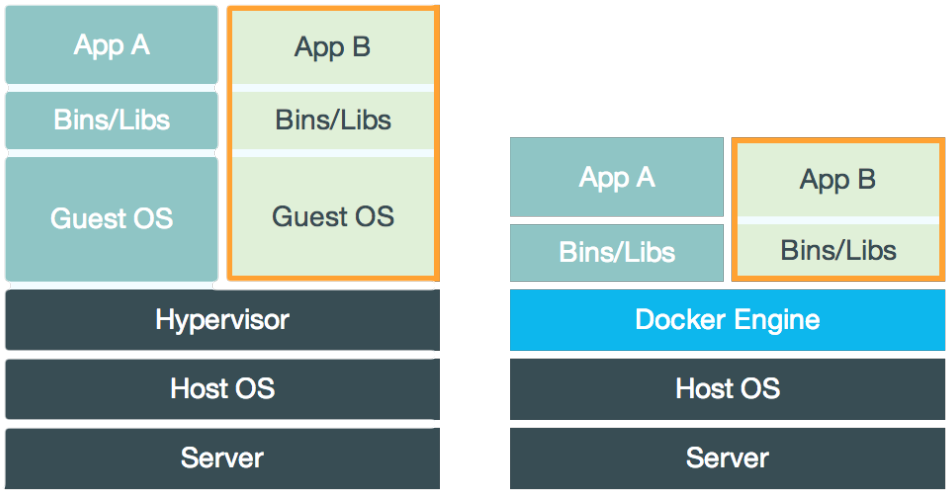

Abbildung 2: Quelle Docker – Eine Einführung, Oliver Nautsch (CC BY 4.0)

Virtualisierung light:

- Docker nutzt den Kernel des Host-Systems.
- Hypervisor-Schicht entfällt.
- Keine Hardwareemulation notwendig.
- Kein eigenes OS wird gestartet.

1.3. Begriffe

- **Image**:
  - Bringt komplette Laufzeitumgebung für eine Anwendung mit.
  - Wird mit einem sogenannten Dockerfile[2] definiert.
  - Das Dateisystem ist nur lesbar.
  - Ein Image ist die Elterninstanz eines Containers.
  - Kann aus verschiedenen Layern bestehen, die über die Git-Versionsverwaltung zusammengesetzt werden.
- **Container**:
  - Wird aus einem Image gestartet (Laufzeitinstanz).
  - Wird schreibbar durch Overlay-Dateisystem (unionfs).
  - Über spezielle Techniken vom Host-Systems isoliert (cgroups & namespaces, s.o.).

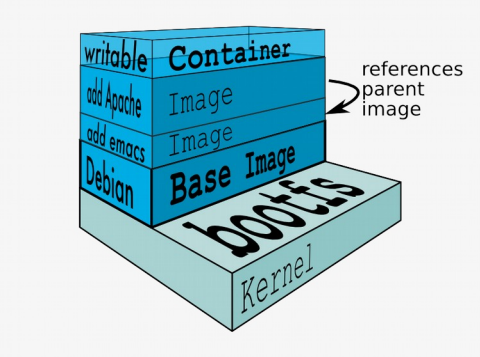

Abbildung 3: Quelle Docker – Eine Einführung, Oliver Nautsch (CC BY 4.0)

- **Volume**:
  - Persistenter Datenspeicher auf dem Host-Dateisystem, der in den Container gemappt wird.
  - Vereinfacht Updates, da der Container einfach ausgetauscht werden kann, ohne dass Daten verlorengehen.
- **Docker-Hub**[3]:
  - Umfangreiches Git-Repository für Docker-Images.

2 Siehe zum Beispiel das Nextcloud-Dockerfile.
3 Docker Hub: https://hub.docker.com/

2. Docker-Praxis

2.1. Installation von Docker[4]

Wir verwenden Ubuntu Server 18.04 LTS als Dockerhost.

- Installation als Benutzer `root` auf der Konsole mit einem Befehl:

```
# apt install docker.io
```

- Erfolgreiche Installation testen:

```
# docker run hello-world
# docker run -it ubuntu bash
```

2.2. Einfaches Nextcloud-Setup mit SQLite-Datenbank

Für die Übung verwenden wir das offizielle Nextcloud-Docker-Image[5].

Die Container-Daten werden auf dem Host-System jeweils in einem eigenen Volume unter */srv/docker/nextcloud* (außerhalb des Nextcloud-Containers) abgelegt und in den Container gemappt:

- User-Daten: */srv/docker/nextcloud/data* -> */var/www/html/data*
- Konfiguration: */srv/docker/nextcloud/config* -> */var/www/html/data/config*
- Nextcloud-Docker-Image suchen:

```
# docker search nextcloud
```

- Nextcloud-Image herunterladen:

```
# docker pull nextcloud
```

- Daten- und Konfigurationsverzeichnisse erstellen:

```
# mkdir -p /srv/docker/nextcloud/data
# mkdir -p /srv/docker/nextcloud/config
```

- Nextcloud-Instanz starten:
  - Image **nextcloud** wird unter dem Namen **nextcloud** gestartet,
  - Hostport 8080 wird auf den Containerport 80 weitergeleitet,
  - Hostvolumes werden in den Container gemappt:

```
# docker run -d --name nextcloud -p 8080:80 \
  -v /srv/docker/nextcloud/data:/var/www/html/data \
  -v /srv/docker/nextcloud/config:/var/www/html/config nextcloud
```

4 Installationsanleitungen siehe https://docs.docker.com/engine/installation/linux/
5 Nextcloud-Docker-Image auf github: https://github.com/nextcloud/docker

- Docker-Nextcloud-Prozess anzeigen:

```
# docker ps
```

- Nextcloud-Installation unter http://<Dockerhost-IP>:8080 abschließen[6].

```
# docker stop nextcloud
```

- Container löschen:

```
# docker rm nextcloud
```

- Image löschen:

```
# docker rmi nextcloud
```

2.3. Nextcloud-Setup mit MariaDB-Container

Wir stellen dem Nextcloud-Container einen zusätzlichen Datenbank-Container mit MariaDB zur Seite. Das ist aus Performanz-Gründen empfehlenswert. Dazu benötigen wir **Docker Compose**, ein Werkzeug zur Definition und Ausführung von Multi-Container-Docker-Anwendungen.

- Docker Compose installieren:

```
# apt install docker-compose
```

- In unserem Nextcloud-Docker-Verzeichnis erstellen wir die compose-Datei[7] *docker-compose.yml* mit folgendem Inhalt:

```
version: '3'

volumes:
  data:
  config:
  db:

services:
  nextcloud-db:
    image: mariadb
    restart: always
    volumes:
      - ./db:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=Muster!
      - MYSQL_PASSWORD=Muster!
      - MYSQL_DATABASE=nextcloud
      - MYSQL_USER=nextcloud
  nextcloud:
    image: nextcloud
    ports:
      - 8080:80
```

6 Die *Dockerhost-IP* ist die IP-Adresse, die die virtuelle Maschine per NAT-DHCP zugewiesen bekommt.

7 Siehe https://github.com/nextcloud/docker#base-version---apache

```
    depends_on:
      - nextcloud-db
    volumes:
      - ./config:/var/www/html/config
      - ./data:/var/www/html/data
    restart: always
```

- Alte Nextcloud-Daten löschen:

  ```
  # rm -rf config/* data/*
  ```

- Verzeichnis für das db-Volume erstellen:

  ```
  # mkdir db
  ```

- Container starten:

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

  Der Befehl lädt die Images herunter, startet die Container und richtet Datenbank- und Nextcloud-Container entsprechend den Vorgaben aus der *`docker-compose.yml`* ein.

- Im Nextcloud-Setup, das nun unter der URL *http://<Dockerhost-IP>:8080* aufgerufen wird, ist im letzten Feld der Name des MariaDB-Containers *nextcloud-db* einzutragen:

Abbildung 4: Nextcloud-Setup

2.4. Nextcloud-Container automatisiert per systemd starten[8]

Wir erstellen Start- und Stopskripte und stellen sicher, dass der Nextcloud-Container beim Booten des Dockerhosts automatisch startet.

- Unter */srv/docker/nextcloud* Start- und Stop-Skripte erstellen und ausführbar machen:
  - Start-Skript *start.sh*:

```
#!/bin/sh

RC=0
cd /srv/docker/nextcloud
/usr/bin/docker-compose up -d || RC=1
exit $RC
```

  - Stop-Skript *stop.sh*:

```
#!/bin/sh

RC=0
/usr/bin/docker stop nextcloud || RC=1
/usr/bin/docker stop nextcloud-db || RC=1
/usr/bin/docker rm nextcloud || RC=1
/usr/bin/docker rm nextcloud-db || RC=1
exit $RC
```

- Datei */etc/systemd/system/nextcloud.service* mit folgendem Inhalt anlegen:

```
[Unit]
Description=Docker - Nextcloud container
Requires=docker.service
After=docker.service

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/srv/docker/nextcloud/start.sh
ExecStop=/srv/docker/nextcloud/stop.sh

[Install]
WantedBy=default.target
```

- Service aktivieren (muss einmal gemacht werden, danach wird automatisch gestartet):

```
# systemctl enable nextcloud.service
```

- Service starten|stoppen:

8 Siehe https://www.freedesktop.org/software/systemd/man/systemd.service.html

```
# systemctl start|stop nextcloud.service
```

- Status des Services anzeigen:

```
# systemctl status nextcloud.service
```

2.5. SSL-Verbindung mit Reverse-Proxy

Um die Verbindung zur Nextcloud-Instanz mit SSL abzusichern, erstellen wir zunächst selbstsignierte Serverzertifikate und richten danach einen Reverse-Proxy mit nginx ein.

2.5.1. Zertifikate mit openssl erstellen[9]

- Zunächst erstellen wir ein Verzeichnis, das unsere Zertifikatsdateien aufnimmt:

```
# mkdir -p /srv/docker/nginx-proxy/ssl
```

- Wir wechseln in dieses Verzeichnis und erstellen den privaten Schlüssel *proxy.key*:

```
# openssl genrsa -out proxy.key 4096
```

- Dann erstellen wir den sogenannten *Certificate Signing Request* *proxy.csr* unter Verwendung des zuvor erstellten privaten Schlüssels:

```
# openssl req -new -key proxy.key -out proxy.csr
```

Dabei sind ein Paar Eingaben zu machen:

```
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:BW
Locality Name (eg, city) []:Esslingen
Organization Name (eg, company) [Internet Widgits Pty Ltd]:LFB
Organizational Unit Name (eg, section) []:FB
Common Name (e.g. server FQDN or YOUR name) []:ubuntu
Email Address []:
Please enter the following 'extra' attributes to be sent with your
certificate request
A challenge password []:
An optional company name []:
```

Bei *Common Name* gibt man den Hostnamen ein, den Rest lässt man leer, indem man einfach [Enter] drückt. Es wird kein Passwort vergeben, da man es sonst bei jedem Start des Proxy-Containers an der Konsole eingeben müsste.

- Schließlich erstellen wir mit Hilfe des privaten Schlüssels und des Requests das 365 Tage gültige selbstsignierte X.509-Zertifikat[10]:

```
# openssl x509 -req -days 365 -in proxy.csr \
    -signkey proxy.key -out proxy.crt
```

9 Vorgehensweise nach https://serversforhackers.com/c/self-signed-ssl-certificates
10 Wikipedia-Artikel zu X.509: https://de.wikipedia.org/wiki/X.509

2.5.2. Reverse-Proxy einrichten

Für den Reverse-Proxy verwenden wir das nginx-Dockerimage von **jwilder**[11].

- Wir wechseln in das Verzeichnis */srv/docker/nginx-proxy* und erstellen dort eine Datei *docker-compose.yml* mit folgendem Inhalt:

```
version: '2'

volumes:
  docker.sock:
  ssl:
  conf.d:

services:
  nginx-proxy:
    image: jwilder/nginx-proxy
    container_name: nginx-proxy
    ports:
     - "80:80"
     - "443:443"
    volumes:
      - /var/run/docker.sock:/tmp/docker.sock:ro
      - ./ssl:/etc/nginx/certs:ro
      - ./conf.d:/etc/nginx/conf.d
  whoami:
    image: jwilder/whoami
    container_name: whoami
    environment:
      - VIRTUAL_HOST=whoami.local
```

- Danach erstellen wir ein Unterverzeichnis *conf.d* und erzeugen darin die nginx-Konfigurationsdatei *nextcloud.conf* (die Dockerhost-IP muss entsprechend ersetzt werden):

```
upstream <Dockerhost-IP> {
        server <Dockerhost-IP>:8080;
}

server {
    listen 443 ssl;
    ssl_certificate           /etc/nginx/certs/proxy.crt;
    ssl_certificate_key       /etc/nginx/certs/proxy.key;
    location / {
        proxy_pass http://<Dockerhost-IP>;
    }
}
```

11 Automated nginx proxy for Docker containers using docker-gen: https://github.com/jwilder/nginx-proxy

- Jetzt kann der Proxy-Container gestartet werden:

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

- Danach ist die Nextcloud-Instanz per *https* aufrufbar. Allerdings bekommt man beim ersten Aufruf eine Warnmeldung, weil die Instanz nicht über Port 8080 aufgerufen wird:

Abbildung 5: Nextcloud-Warnung "vertrauenswürdige Domain hinzufügen"

- Leider lässt sich die vertrauenswürdige Domain nicht über die angebotene Schaltfläche hinzufügen, da sie in der URL das https-Protokoll mit der Portnummer 8080 kombiniert, was nicht funktionieren kann.
- Das lässt sich beheben, indem man in der Nextcloud-Konfigurationsdatei */srv/docker/nextcloud/config/config.php* die Portnummer 8080 der Dockerhost-IP (im Beispiel *172.16.52.130*) unter *trusted_domains* entfernt:

```
'trusted_domains' =>
array (
  0 => '172.16.52.130:8080',
),

'trusted_domains' =>
array (
  0 => '172.16.52.130',
),
```

- Jetzt sollte der Aufruf der Nextcloud-Instanz per *https* ohne Probleme von statten gehen.

2.5.3. Reverse-Proxy automatisch per systemd starten

Analog zu Abschnitt 2.4 richten wir für den Nginx-Proxy-Container die entsprechenden Skripte ein.

- Im Verzeichnis */srv/docker/nginx-proxy*:
  - *start.sh*:

```
#!/bin/sh

RC=0
cd /srv/docker/nginx-proxy
/usr/bin/docker-compose up -d || RC=1
exit $RC
```

  - *stop.sh*:

```
#!/bin/sh

RC=0
/usr/bin/docker stop nginx-proxy || RC=1
/usr/bin/docker stop whoami || RC=1
/usr/bin/docker rm nginx-proxy || RC=1
/usr/bin/docker rm whoami || RC=1
exit $RC
```

    Vergessen Sie nicht die beiden Skripte ausführbar zu machen!

- Das Systemd-Skript */etc/systemd/system/nginx-proxy.service*:

```
[Unit]
Description=Docker - Nginx-Proxy container
Requires=docker.service
After=docker.service

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/srv/docker/nginx-proxy/start.sh
ExecStop=/srv/docker/nginx-proxy/stop.sh

[Install]
WantedBy=default.target
```

- Damit der Dienst beim Booten startet, muss er noch aktiviert werden:

```
# systemctl enable nginx-proxy.service
```

3. Weitere Quellen

- c't-Video, nachgehakt: Docker-Praxis mit Linux - Wo Container punkten
- c't 14.17, Thorsten Leemhuis: Hafenarbeiter - Container-Virtualisierung mit Docker, S. 146ff
- c't 16.05, Thorsten Leemhuis: Warenverkehr - Container mit Docker bauen, umschlagen und betreiben, S. 112ff
- Oft benötigte Dockerbefehle und Best Practises: Docker Cheat Sheet
- Wikipedia-Artikel Docker (Software)