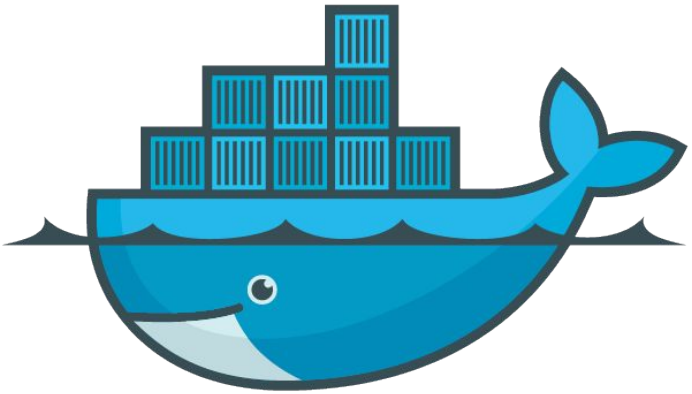


docker

Eine Einführung

Netresearch GmbH & Co. KG
2016-03-31

Norman Bestfleisch
Sebastian Mendel

Kategorie:	Virtualisierung
Lizenz:	Open-Source (Apache 2.0)
Erscheinunsjahr:	2013
Programmiersprache:	Go
Aktuelle Version:	1.10 (1.11 RC)
Beschreibung:	❏ erstellt portable Container für Anwendungen und deren Abhängigkeiten ❏ basiert auf verschiedenen Features des Linux-Kernels, um Container zu erstellen

Virtualisierung

Namespaces:

- bündeln Prozessgruppen und trennen sie voneinander
- leichtgewichtige Alternative zur Hardwarevirtualisierung
- virtuelle Umgebung zur isolierten Ausführung von Prozessen
- Container teilen sich einen gemeinsamen Kernel

Control Groups (cgroups)

- Ressourcenverwaltung
- limitieren Ressourcen wie CPU, RAM und I/O, die sie voneinander isolieren und kontrollieren können

Container vs. VM ein Vergleich

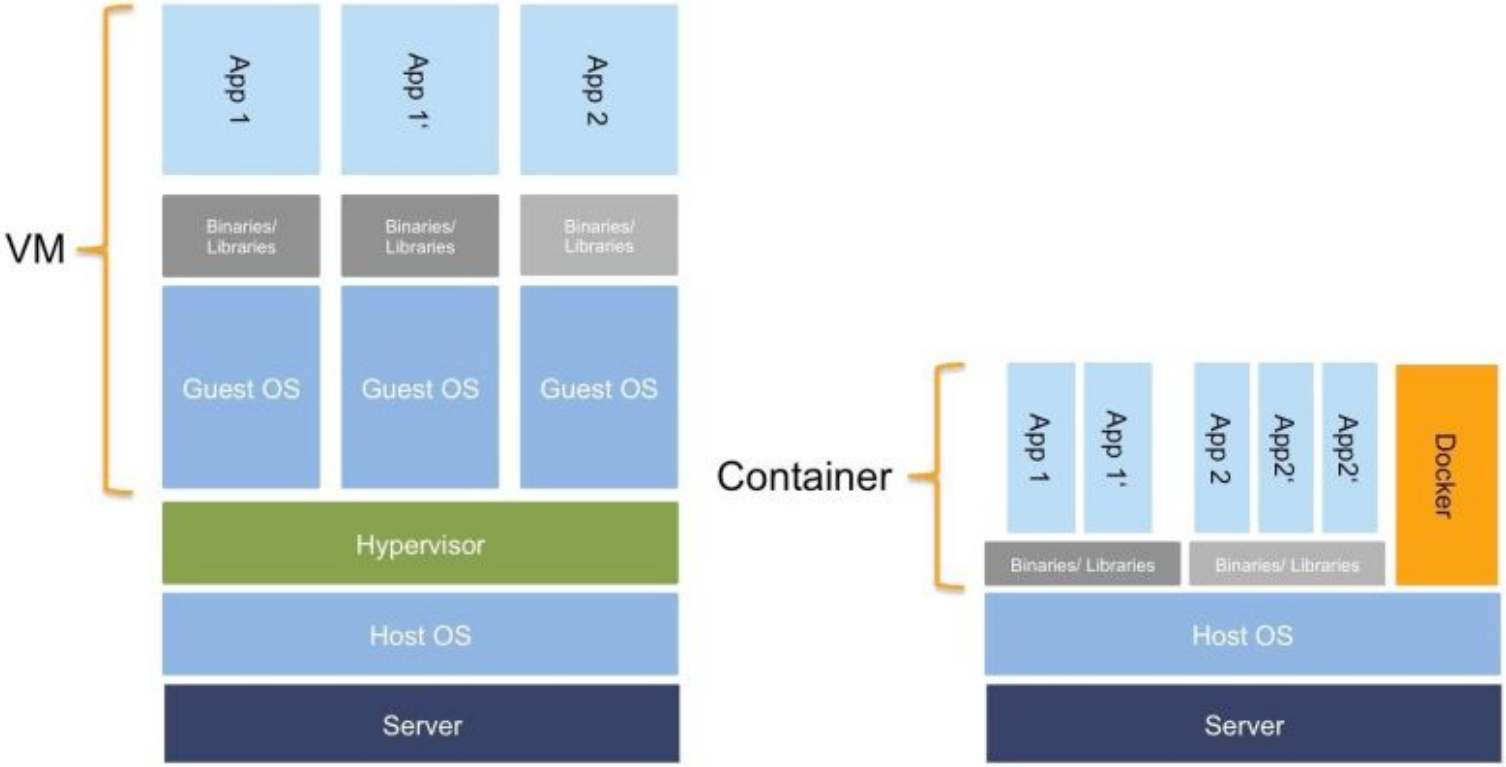
Container:

+ kein Overhead durch Hardwarevirtualisierung
+ ressourcenschonend, effektiv und schnell
+ portabilität
- keine tiefgreifende Isolation wie bei VM's, da alle Container auf demselben Kernel laufen

Virtuelle Maschine:

+ gute Isolation der VM's untereinander und vom Host
- hoher Bedarf an Ressourcen
- langwieriger Ladevorgang beim Start
- geringe Anzahl an zeitgleich ausführbaren VM's möglich

Virtualisierung: Virtuelle Maschinen vs. Docker-Container



Quelle: Docker, Crisp Research, 2014

Layer, Images, Container, ...

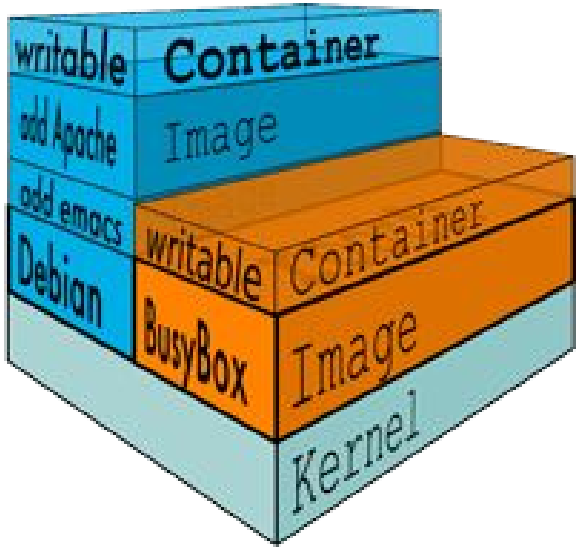
Image: Abbild eines Systems, das als Basis für den Container dient

Layer: Schichten, aus denen sich die Images zusammensetzen

Container: ein Image, das ausgeführt wird

Dockerfile: ‘Bauanleitung’ für ein Image

Registry: Plattform zum einfachen Austausch der Images



Vorteile auf einen Blick

+ Anwendung portabel
+ befreit von Abhängigkeiten/Isolierung
+ vereinfacht die Container-Virtualisierung wesentlich
+ etabliert ein Container-Format mit Layer-Struktur (Wiederverwendung von Layern)
+ geringer Ressourcenbedarf und hohe Ladegeschwindigkeit von Containern
+ hohe Anzahl an vordefinierten Images auf Docker Hub verfügbar
+ gleiche/selbe Umgebung während Entwicklung und auf Live
+ Schnelle Deployments
+ Dokumentierte “Server”-Konfiguration

Eigenen Container starten … interaktiv

```
$ docker run -ti php
Unable to find image 'php:latest' locally
latest: Pulling from library/php
fdd5d7827f33: Already exists
a3ed95caeb02: Pull complete
2f584a474c46: Pull complete
b185147d01c6: Pull complete
6663e6f16e95: Pull complete
fd91f8bf39df: Pull complete
6d7c00c29f57: Pull complete
Digest: sha256:
a98c4203f8d463ddc018948931ce70df4dd2204c3394cc183a5cc1f38
b799c81
Status: Downloaded newer image for php:latest
Interactive shell

php > echo phpversion();
7.0.4
php >
```

```
$ docker run -ti php:5.6.19
Unable to find image 'php:5.6.19' locally
5.6.19: Pulling from library/php
fdd5d7827f33: Already exists
a3ed95caeb02: Pull complete
2f584a474c46: Already exists
b185147d01c6: Already exists
6663e6f16e95: Already exists
b70135a50b0c: Pull complete
6531a1f6443e: Pull complete
Digest: sha256:
5a1fbc73b5f8cd4a88f56e9e5588c247b93200f48e002ce1c4571b55
00433545
Status: Downloaded newer image for php:5.6.19
Interactive shell

php > echo phpversion();
5.6.19
php >
```

`run` - container starten
- `-t` --tty - um ein Prompt zu bekommen
- `-i` --interactive - damit wir auch rumtippern können
- `--rm` - Container löschen nach Beendigung
- `php` - image aus dem der Container gestartet wird
- `:5.6.19` - getaggte Version des Images

Eigenen Container starten … ein Dämon

```
$ docker run -d -e MYSQL_ALLOW_EMPTY_PASSWORD=1 mariadb
11769c2766c0075c54c4fae5f31e2d322b4361d41d946e6cebd42ceb70017b1a
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS      NAMES
11769c2766c0   mariadb   "/docker-entrypoint.s"   7 seconds ago   Up 6 seconds   3306/tcp   prickly_cori

$ docker stop prickly_cori
prickly_cori
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS      NAMES
$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS      NAMES
11769c2766c0   mariadb   "/docker-entrypoint.s"   7 minutes ago   Exited                    prickly_cori
$ docker rm prickly_cori
prickly_cori
$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS      NAMES
```

`run` - Container starten
- `-d` --detach - der Container soll im Hintergrund laufen
- `-e` --env - Umgebungsvariablen im Container setzen
- `mariadb` - Image aus dem der Container gestartet wird

`stop` - Container stoppen

`ps` - Container auflisten
- `-a` --all - auch nicht laufende Container

`rm` - Container löschen

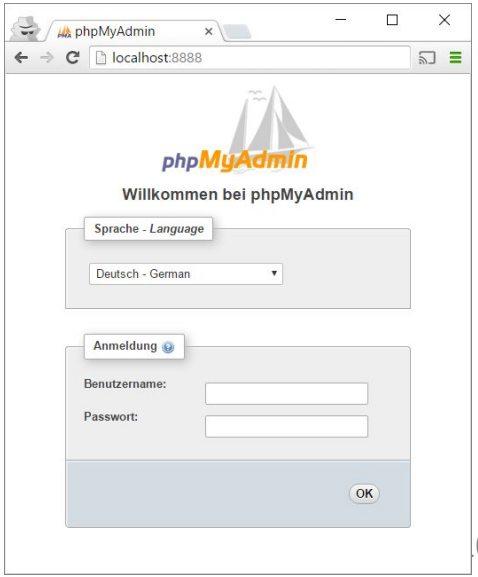
Eigenen Container starten … und noch einen

```
$ docker run --name mydb -d -e MYSQL_ALLOW_EMPTY_PASSWORD=1 mariadb
68d8b5371da6b141c94157c6cd59ad24e4e98fbe531c880bae52c0c69ca45655

$ docker run --name pma -d --link mydb:db -p 8888:80 phpmyadmin/phpmyadmin
62dd1e7ca48fb341282869d0e4b19e1b4df63b10dda922094a07643a558d46af

$ docker ps
CONTAINER ID IMAGE                 COMMAND                  CREATED         STATUS         PORTS                  NAMES
62dd1e7ca48f phpmyadmin/phpmyadmin "/run.sh"                16 seconds ago Up 15 seconds 0.0.0.0:8888->80/tcp pma
68d8b5371da6 mariadb               "/docker-entrypoint.s" 29 seconds ago Up 28 seconds 3306/tcp               mydb

$ docker stop pma mydb && docker rm pma mydb
pma
mydb
pma
mydb
```



`run` - Container starten
- `-d` --detach - der Container soll im Hintergrund laufen
- `-e` --env - Umgebungsvariablen im Container setzen
- `--name` - Name für Container setzen
- `--link` - Container verknüpfen
- `-p` --publish - Container Port nach außen verfügbar machen
- `mariadb` - Image aus dem der Container gestartet wird

Eigenen Container starten

… alle zusammen und miteinander (docker-compose)

```
$ cat docker-compose.yml
phpmyadmin:
  image: phpmyadmin/phpmyadmin
  ports:
    - "8888:80"
  links:
    - mariadb:db
  container_name: pma
  restart: always

mariadb:
  image: mariadb
  environment:
    -
MYSQL_ALLOW_EMPTY_PASSWORD=1
  container_name: mydb
  restart: always
```

```
$ docker-compose up -d
Creating mydb
Creating pma

$ docker-compose stop
Stopping pma ...
Stopping mydb …

$ docker-compose rm
Going to remove pma, mydb
Are you sure? [yN] y
Removing pma ... done
Removing mydb ... done
```

Dockerfile

Images bauen nach Rezept

```
$ cat Dockerfile

FROM alpine:3.3

RUN apk add --update redis && \
    rm -rf /var/cache/apk/* && \
    mkdir /data && \
    chown -R redis:redis /data

VOLUME /data
WORKDIR /data

EXPOSE 6379

CMD [ "redis-server" ]
```

Ausgangsimage / OS wählen

Gewünschte Software/Pakete hinzufügen

Konfiguration/Umgebung anpassen

Dienst nach außen bekannt machen

Dienst starten

docker build

Image backen aus Rezept

```
$ docker build --tag=myredis:testing .
Sending build context to Docker daemon 2.048 kB
Step 1 : FROM alpine:3.3 ──────────────────────────────────────────────────────
 ---> 90239124c352
Step 2 : RUN apk add --update redis && rm -rf /var/cache/apk/* && mkdir /data && chown -R──
redis:redis /data
 ---> Running in 2caccab0499b
fetch http://dl-4.alpinelinux.org/alpine/v3.3/main/x86_64/APKINDEX.tar.gz
fetch http://dl-4.alpinelinux.org/alpine/v3.3/community/x86_64/APKINDEX.tar.gz
(1/1) Installing redis (3.0.5-r1)
Executing redis-3.0.5-r1.pre-install
Executing busybox-1.24.1-r7.trigger
OK: 6 MiB in 12 packages
 ---> 0002190aa192
Removing intermediate container 2caccab0499b
Step 3 : VOLUME /data ─────────────────────────────────────────────────────────
 ---> Running in eac369fe061a
 ---> cae50bc0fe0b
Removing intermediate container eac369fe061a
Step 4 : WORKDIR /data ────────────────────────────────────────────────────────
 ---> Running in 50f6e2684f83
 ---> f2581fbfa3fb
Removing intermediate container 50f6e2684f83
Step 5 : EXPOSE 6379 ──────────────────────────────────────────────────────────
 ---> Running in 45eb8ef937e6
 ---> 28f1d4cf2d3d
Removing intermediate container 45eb8ef937e6
Step 6 : CMD redis-server ─────────────────────────────────────────────────────
 ---> Running in bdad696fd60c
 ---> 8d29dc018a7a
Removing intermediate container bdad696fd60c
Successfully built 8d29dc018a7a
```

```
$ cat Dockerfile
FROM alpine:3.3
RUN apk add --update redis && \
    rm -rf /var/cache/apk/* && \
    mkdir /data && \
    chown -R redis:redis /data
VOLUME /data
WORKDIR /data
EXPOSE 6379
CMD [ "redis-server" ]
```

Docker Registry

… wo sind all die Images hin

Docker Registry dient als Speicher für Docker Images.

Docker Hub, offizielle Registry, beinhaltet eine große Sammlungen an öffentlichen Images

Docker Registry gibt es auch als Enterprise - für private Images

Docker Registry gibt es auch als Docker Image für Daheeme

Ein Image aus der Docker Registry laden:

```
$ docker pull varnish
$ docker pull registry.example.org/myredis:3.0.2
```

Ein Image in der Docker Registry ablegen:

```
$ docker tag myredis:testing registry.example.org/myredis:stable
$ docker push registry.example.org/myredis:stable
```

Tipps

was sich bewährt hat

benutzt make (Makefile)

http://www.itnotes.de/docker/development/tools/2014/08/31/speed-up-your-docker-workflow-with-a-makefile/

Befehle in Dockerfile nicht zusammenhängen während Test-/Entwicklungsphase

beschleunigt Build-Prozess durch Cache-Nutzung

1 Service pro Container/Image

z. B. nicht httpd und PHP FPM in einem Image

Befehle in Dockerfile zusammenhängen für finale Version

spart Layer und Größe/Speicher -> docker history [image]

Aufräumen von alten gelöschten Images/Layern und Containern

Docker selber lässt die liegen, vermüllen die Platte

Verwendet Volumes für Daten die gesichert werden müssen oder Neustarts/Updates überleben sollen

Keine Benutzer-/Zugangsdaten oder allgemein sensible Daten im Image

z. B. Zertifkat-Schlüssel, DB Passwörter, ...

Was noch fehlt ...

Working within Containers

Volumes

Networking

Loghandling

Limits (CPU, RAM, Network, HDD, …)

Deployment Best Practices

Ende

Quellen und weitere Informationen:

Web:

http://docs.docker.com/
http://www.heise.de/developer/artikel/Anwendungen-mit-Docker-transportabel-machen-2127220.html
http://www.heise.de/developer/artikel/Mit-Docker-automatisiert-Anwendungscontainer-erstellen-2145030.html
http://wiki.nr/wiki/Docker
http://blog.nr/multi-container-anwendungen-mit-docker-compose-orchestrieren/

Print:

iX - Magazin für professionelle Informationstechnik 05/2014 und 04/2015
“Container Virtualisierung mit Docker”, c’t Linux 2015
“Docker - Wie sicher ist die Containertechnologie?”, PHPmagazin 6.15
“Build Your Own PaaS with Docker”, Oskar Hane, Packt Publishing, 2015

Fragen?