

Docker

Konzept, Internals & Einsatzszenarien

TechTalk im Netz39 e.V.

Plan für heute

Die Idee hinter Docker

Begriffe, Bestandteile & Ökosystem

Geräte, Grafik, Sound

Einschub: Sicherheitsaspekte

Einsatzszenarien

„Die besten Pläne scheitern gemeinhin am Publikum.“
– Oskar Wilde

Idee

Ziele von Docker

Logische Trennung von Applikationen

Bündeln von Applikationen zu universellen „Paketen“

Dependency-Hölle vermeiden

Vereinfachung der Provisionierung von Servern

Infrastructure-as-Code

Die Idee

Docker ist ein Werkzeug zum...

- ... Erstellen & Verwalten von Container Images
- ... Erstellen & Verwalten von Containern

Docker ist kein(e)...

- ... Betriebssystem
- ... virtuelle Maschine ¹
- ... Allheilmittel

¹ Unter macOS und Windows teils schon

Grundbegriffe, Bestandteile & Ökosystem

Was sind Container?

chroot auf Steroiden

eigene Sicht auf Ressourcen

- Dateisystem
- Netzwerk
- Prozesstabelle

„Aber mit Jails oder chroot + cgroups + NetworkNS haben wir das doch schon voll lange!!1elf!“ – Ja.

Die Welt aus Sicht eines Alpine-Containers

```
okirmis@T470: ~  
Datei Bearbeiten Ansicht Suchen Terminal Hilfe  
/ # sh -c 'sleep 10 && echo Inside Container'  
Inside Container  
/ # ps aux  
PID    USER      TIME  COMMAND  
    1   root         0:00  sh  
   14   root         0:00  ps aux  
/ #  
  
root      11961  0.0  0.0   1584     4 pts/0    S+   14:36  
    0:00 sh -c sleep 10 && echo Inside Container  
okirmis@T470:~$  
  
[0] 0:docker* "T470" 14:41 18-Jan-19
```


Was sind Container-Images?

Dateisystem eines Containers

Metainformationen

- Umgebungsvariablen
- Startkommando & „Einstiegspunkt“
- Arbeitsverzeichnis
- Volumes
- Nutzer
- ...

Entspräche in der VM-Welt einer virtuellen Festplatte. Nur besser :)

Beispiel: PostgreSQL aufsetzen

```
okirmis@T470:~$ docker run -d \
  --name my-postgres-container \
  -e POSTGRES_PASSWORD=mypass \
  -e POSTGRES_USER=myuser \
  postgres:11
Unable to find image 'postgres:11' locally
11: Pulling from library/postgres
[...]
Digest: sha256:[...]
Status: Downloaded newer image for postgres:11
317f2d55[...]
okirmis@T470:~$ docker ps
```

CONTAINER ID	NAMES	IMAGE
317f2d55f80a	my-postgres-container	postgres:11

Beispiel: PostgreSQL aufsetzen

```
okirmis@T470:~$ docker run -d \
  --name my-postgres-container \
  -e POSTGRES_PASSWORD=mypass \
  -e POSTGRES_USER=myuser \
  postgres:11
Unable to find image 'postgres:11' locally
11: Pulling from library/postgres
[...]
Digest: sha256:[...]
Status: Downloaded newer image for postgres:11
317f2d55[...]
okirmis@T470:~$ docker ps
CONTAINER ID   NAMES                                IMAGE
317f2d55f80a   my-postgres-container               postgres:11
```

Container haben Namen & autom. IDs

Beispiel: PostgreSQL aufsetzen

Im Hintergrund
laufen

```
okirmis@T470:~$ docker run -d \
  --name my-postgres-container \
  -e POSTGRES_PASSWORD=mypass \
  -e POSTGRES_USER=myuser \
  postgres:11
```

```
Unable to find image 'postgres:11' locally
11: Pulling from library/postgres
[...]
```

```
Digest: sha256:[...]
```

```
Status: Downloaded newer image for postgres:11
317f2d55[...]
```

```
okirmis@T470:~$ docker ps
```

CONTAINER ID	NAMES	IMAGE
317f2d55f80a	my-postgres-container	postgres:11

Beispiel: PostgreSQL aufsetzen

```
okirmis@T470:~$ docker run -d \
  --name my-postgres-container \
  -e POSTGRES_PASSWORD=mypass \
  -e POSTGRES_USER=myuser \
  postgres:11
```

Versionen über
Tags
spezifizieren

```
Unable to find image 'postgres:11' locally
11: Pulling from library/postgres
[...]
Digest: sha256:[...]
Status: Downloaded newer image for postgres:11
317f2d55[...]
```

```
okirmis@T470:~$ docker ps
```

CONTAINER ID	NAMES	IMAGE
317f2d55f80a	my-postgres-container	postgres:11

Beispiel: PostgreSQL aufsetzen

```
okirmis@T470:~$ docker run -d \  
  --name my-postgres-container \  
  -e POSTGRES_PASSWORD=mypass \  
  -e POSTGRES_USER=myuser \  
  postgres:11
```

Unable to find image 'postgres:11' locally

11: Pulling from library/postgres
[...]

Digest: sha256:[...]

Status: Downloaded newer image for postgres:11
317f2d55[...]

```
okirmis@T470:~$ docker ps
```

CONTAINER ID	NAMES	IMAGE
317f2d55f80a	my-postgres-container	postgres:11

Autom.
Download,
offizielle &
inoffizielle
Images

Beispiel: PostgreSQL aufsetzen

```
okirmis@T470:~$ docker run -d \
  --name my-postgres-container \
  -e POSTGRES_PASSWORD=mypass \
  -e POSTGRES_USER=myuser \
  postgres:11
Unable to find image 'postgres:11' locally
11: Pulling from library/postgres
[...]
Digest: sha256:[...]
Status: Downloaded newer image for postgres:11
317f2d55[...]
okirmis@T470:~$ docker ps
CONTAINER ID    NAMES                                IMAGE
317f2d55f80a    my-postgres-container               postgres:11
```

Konfiguration
über
Umgebungsvariablen

Beispiel: PostgreSQL aufsetzen

```
okirmis@T470:~$ docker run -d \
  --name my-postgres-container \
  -e POSTGRES_PASSWORD=mypass \
  -e POSTGRES_USER=myuser \
  postgres:11
Unable to find image 'postgres:11' locally
11: Pulling from library/postgres
[...]
Digest: sha256:[...]
Status: Downloaded newer image for postgres:11
317f2d55[...]
okirmis@T470:~$ docker ps
```

CONTAINER ID	NAMES	IMAGE
317f2d55f80a	my-postgres-container	postgres:11

Images beziehen – Docker Registries

Speichern Images

On-Premise oder Managed

- hub.docker.com ist **die** zentrale Registry
- `docker run -d -p 5000:5000 --name my-reg registry:2`
- GitLab hat integrierte Registry
- u.v.m.

`docker run my.registry.de:1234/awesome-image`

Daten in Containern verwalten I - Grundidee

Konventionelle Datenverwaltung

„was nicht explizit temporär ist, ist persistent“

Container

„was nicht explizit persistent ist, ist temporär“

Container sind „flüchtig“ (ephemeral)

Explizites „das hätte ich gerne dauerhaft“

Datenbankverzeichnisse, Logs*, Konfigurationsdateien, ...

Daten in Containern verwalten II - Volumes

Einfachste Lösung: Host-Verzeichnisse binden

Besser: „echte Volumes“

- Lokal
- NFS, CEPH, SSHFS
- Kommerzielle Stagesysteme
- AWS, Azure, ...

Read-only oder auch schreibbar

Beispiel: PostgreSQL aufsetzen (+Persistenz)

```
okirmis@T470:~$ docker run -d \  
  --name my-postgres-container \  
  -e POSTGRES_PASSWORD=mypass \  
  -e POSTGRES_USER=myuser \  
  -v $(pwd)/mydb:/var/lib/postgresql/data \  
  postgres:11  
[...]  
okirmis@T470:~$ sudo ls mydb  
drwx----- 5 999 999  4096 Jan 18 16:36 base  
drwx----- 2 999 999  4096 Jan 18 16:36 global  
[...]
```

Beispiel: PostgreSQL aufsetzen (+Persistenz)

```
okirmis@T470:~$ docker run -d \  
  --name my-postgres-container \  
  -e POSTGRES_PASSWORD=mypass \  
  -e POSTGRES_USER=myuser \  
  -v $(pwd)/mydb:/var/lib/postgresql/data \  
  postgres:11
```

**Persistentes
Verzeichnis**

```
okirmis@T470:~$ sudo ls -la mydb  
drwx----- 5 999 999 4096 Jan 18 16:36 base  
drwx----- 2 999 999 4096 Jan 18 16:36 global  
[...]
```

Beispiel: PostgreSQL aufsetzen (+Persistenz)

```
okirmis@T470:~$ docker run -d \  
  --name my-postgres-container \  
  -e POSTGRES_PASSWORD=mypass \  
  -e POSTGRES_USER=myuser \  
  -v $(pwd)/mydb:/var/lib/postgresql/data \  
  postgres:11
```

[...]

```
okirmis@T470:~$ sudo ls mydb
```

drwx-----	5	999	999	4096	Jan 18 16:36	base
drwx-----	2	999	999	4096	Jan 18 16:36	global

[...]

What?!

Beispiel: PostgreSQL aufsetzen (+Persistenz)

```
okirmis@T470:~$ docker run -d \  
  --name my-postgres-container \  
  -e POSTGRES_PASSWORD=mypass \  
  -e POSTGRES_USER=myuser \  
  -v $(pwd)/mydb:/var/lib/postgresql/data \  
  postgres:11
```

[...]

```
okirmis@T470:~$ sudo ls mydb
```

```
drwx----- 5 999 999 4096 Jan 18 16:36 base
```

```
drwx----- 2 999 999 4096 Jan 18 16:36 global
```

[...]

Eigene UIDs & GIDs
im Container!

UID 999 = postgres
GID 999 = postgres

Eigene Images erstellen

Simpleste Lösung

- Container erstellen
- einrichten wie gewünscht
- committen & taggen: Container → Image

Besser: Dockerfiles

- Textuelle Beschreibung (skriptartig)
- `docker build`
- sinnvoll für CI-Unterstützung

Beispiel: Rails-Anwendung bauen

FROM ruby:2.6

my-rails-app/Dockerfile

RUN apt-get update -qq && \
apt-get install -y nodejs postgresql-client && \
useradd -ms /bin/sh server

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

COPY Gemfile* /usr/src/app/

RUN bundle install

COPY . /usr/src/app

RUN chown server log files/uploads -R

EXPOSE 3000

USER server

CMD ["rails", "server", "-b", "0.0.0.0"]

Geräte, Grafik, Sound

Technische Umsetzung unter Linux

Linux unterstützt...

- Overlay-Dateisysteme
- Bind-Mounts

Ein Linux-Prozess kann...

- ein eigenes Root-Verzeichnis haben
- Teil eines Network-Namespace sein
→ Sicht auf Netzwerkinterfaces einschränken
- Teil einer cgroup (v2) sein
→ CPU/RAM/IO limitierbar

I M A G E	Container Layer /var/lib/docker/overlay2/{shaHashCnt}
	Image-Layer N /var/lib/docker/overlay2/{shaHashN}
	Image-Layer 2 /var/lib/docker/overlay2/{shaHash2}
	Image-Layer 1 /var/lib/docker/overlay2/{shaHash1}

Technische Umsetzung unter Linux II

High-Level-Übersicht: Container-Start

- ... neues Overlay-Verzeichnis anlegen (über dem obersten Image-Layer)
- ... ggf. Network-Namespace anlegen (+ iptables-Magie)
- ... cgroup anlegen und konfigurieren (z.B. Network-NS zuordnen, Limits)
- ... Prozess in cgroup starten (mit Umgebungsvariablen, ...)
- ... Profit!

```
docker run --rm -t alpine echo Hello World
```

in deutlich unter einer Sekunde

Geräte- & Desktopintegration in Docker

Geräte sind Dateien

- `docker run -it \`
 `-v /dev/bus/usb...:/dev/bus/usb/... \`
 `myimage bash`

Sound via Pulseaudio

- Container mit installiertem PA
- `-v /run/user/$USER_UID/pulse:/run/pulse:ro`
- `/etc/pulse/client.conf: enable-shm=no`

Nützliches Repo: <https://github.com/terlar/docker-spotify-pulseaudio>

Geräte- & Desktopintegration in Docker

X11 – Socket + Umgebungsvariablen

```
xhost +local:docker  
  
docker run -it \  
    -v /tmp/.X11-unix:/tmp/.X11-unix \  
    -e DISPLAY \  
    myimage firefox
```

Wayland – Laufzeitverzeichnis + Umgebungsvariablen

```
docker run \  
    -e XDG_RUNTIME_DIR=/tmp -e WAYLAND_DISPLAY \  
    -v $XDG_RUNTIME_DIR/$WAYLAND_DISPLAY:/tmp/$WAYLAND_DISPLAY \  
    --user=$(id -u):$(id -g) \  
    myimage firefox
```

Einschub

Sicherheitsaspekte

Sicherheitsaspekte

Ressourcen-Limits (RAM, CPU, Netzwerk, Disk)

- Fehlerfälle abfangen
- DoS-Auswirkungen begrenzen

Isolation

- „besser als nichts“
- Hängt von Einhaltung der Best Practices ab
- Infrastructure-as-Code kann Updates, Reviews & Überblick verbessern



Docker ist nicht primär als Sicherheitssystem konzipiert.

Security: Checkliste (IT-Sec 1x1)

Updates, Updates, Updates

- Wenn möglich offizielle Images verwenden
- Eigene Images regelmäßig bauen & vorher Basis-Image pullen

Angriffsfläche minimieren

- Nur unbedingt benötigte Ordner & Devices mounten
- - - cap - add¹ nur bei Bedarf, - - cap - drop¹ wenn möglich
- Least Privilege Principle (Dateirechte, Nutzer, ...)

Docker nicht als Security-Ersatz ansehen

¹<https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities>

Einsatzszenarien

Einsatzszenarien

Für Entwickler

- Einheitliche Umgebung bereitstellen
- Verschiedene Umgebungen vorhalten

Für Hobbyadmins

- Nextcloud/Wordpress/... schnell & einfach aufsetzen
- Infrastruktur Zuhause spiegeln um Updates zu testen

Für Unternehmen/große Projekte

- CI/CD (Development/Testing/Production)
- Skalieren über beliebige* Hardware
- Automatisches Deployment auf N Servern

Einsatzszenarien – Für Entwickler

Komplexe Buildsysteme bereitstellen

- Feste Compiler-Versionen
- krude Drittanbieterabhängigkeiten
- passende Pfade
- Vorkonfiguriertes PostgreSQL/Redis/RabbitMQ/...

IDEs bereitstellen

- Richtige Version der IDE
- Compiler/Abhängigkeiten/... vorkonfiguriert
- 1-Klick-Run vorkonfiguriert

Einsatzszenarien – Für Hobbyadmins

Installation, Konfiguration & Daten trennen

- einfachere Backups
- weniger suchen
- logisch getrennte Verzeichnisse

Infrastructure-as-Code

- Backup einspielen: Konfiguration kopieren, Daten kopieren, Profit!
- Server-Umzug auf gleiche Weise durchführbar
- Bessere Übersicht, welche Services laufen
- Versionierung der Infrastruktur

Einsatzszenarien – Für Unternehmen

Siehe „Für Entwickler“ & „Für Hobbyadmins“

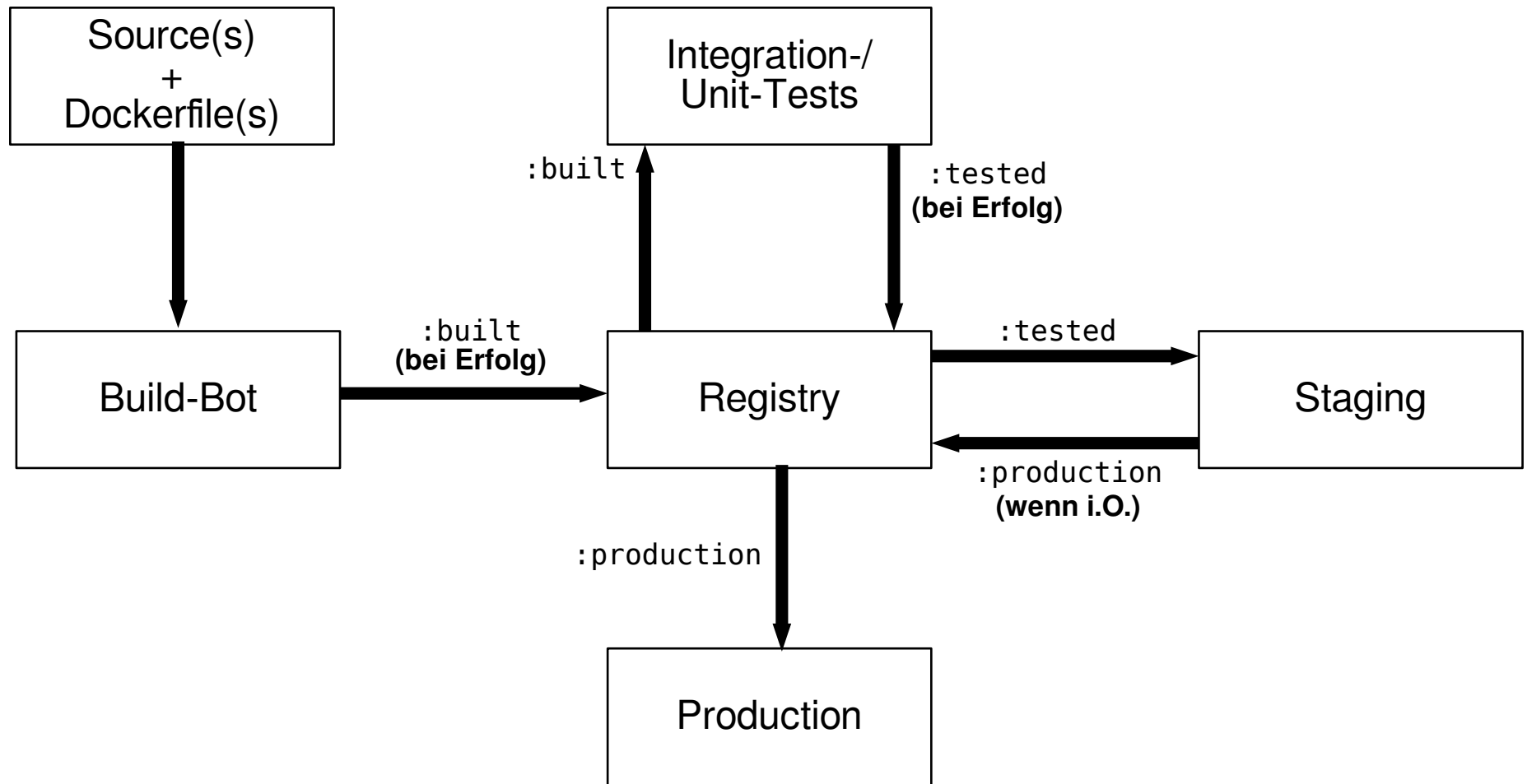
Skalieren

- Egal welcher Server: Dependencies sind alle vorhanden
- `docker-compose scale my-service=3 my-db=2`
- Services über mehrere Server hinweg verwalten (Rancher & Co.)

CI/CD

- Nicht nur Code, auch Infrastruktur automatisiert bauen
- Lokal eingerichtetes Build-System auf Build-Server kopieren

CI/CD mit Docker



Multi-Container-Anwendungen & Container Management

Serverdienste

Problem: Zusammenhängende Dienste

- Wordpress-Container + MariaDB
- Redmine + PostgreSQL
- ShareLaTeX + MongoDB + Redis

Lösung: docker-compose

- Erlaubt Deployment-Konfiguration („docker run“-Argumente)
- Erlaubt Konfiguration mehrere Container


Management-Interfaces wie Portainer

Beispiel: HackMD / CodiMD

docker-compose.yml

```
version: '2'
services:
  hackmd:
    image: hackmdio/hackmd:latest
    environment:
      HMD_DB_URL: ${DB_URL}
    volumes:
      - /storage/hackmd-uploads:/hackmd/public/uploads
    links:
      - hackmdPostgres:hackmdPostgres
  hackmdPostgres:
    image: postgres
    environment:
      POSTGRES_DB: ${DB_NAME}
      POSTGRES_PASSWORD: ${DB_PASS}
      POSTGRES_USER: ${DB_USER}
    volumes:
      - /storage/hackmd-postgres:/var/lib/postgresql/data
```

Portainer: Leichtgewichtiges Web-UI

 portainer.io

NAVIGATION

- Dashboard
- App Templates
- Containers
- Images
- Networks
- Volumes
- Swarm

Container list Containers

- Start
- Stop
- Kill
- Restart
- Pause
- Resume
- Remove
- Add container

☒ Show all containers

Filter...

	State	Name	Image	IP Address	Host IP	Exposed Ports
<input type="checkbox"/>	running	consul3	consul	-	10.0.7.12	-
<input type="checkbox"/>	running	consul2	consul	-	10.0.7.11	-
<input type="checkbox"/>	running	database02	mysql:latest	172.17.0.4	10.0.7.11	-
<input checked="" type="checkbox"/>	stopped	database01	mysql:latest	-	10.0.7.12	-
<input type="checkbox"/>	created	berserk_poitras	nginx	-	10.0.7.11	-
<input type="checkbox"/>	running	web04	nginx:latest	172.17.0.3	10.0.7.12	80 443
<input type="checkbox"/>	running	web03	nginx:latest	172.17.0.3	10.0.7.11	80
<input type="checkbox"/>	stopped	web02	nginx:latest	-	10.0.7.11	-
<input type="checkbox"/>	stopped	web01	nginx:latest	-	10.0.7.12	-
<input type="checkbox"/>	running	swarm_node2	swarm:latest	172.17.0.2	10.0.7.12	-
<input type="checkbox"/>	running	swarm_node1	swarm:latest	172.17.0.2	10.0.7.11	-

Portainer v1.9.0

Kubernetes: Cluster auf 11 aufdrehen

Komplexe Cluster mit Auto-Scale & Sauce & Scharf

Deklarativ

- „Was will ich haben“ → Engine kümmert sich

Weitere Abstraktionsebene für Networking, Storage etc.

Haupteinsatzgebiet: „Cloud“-Umgebungen

- Azure
- AWS
- GKE

Hands-on: Eure Szenarien & Fragen