
docker

Inhalt

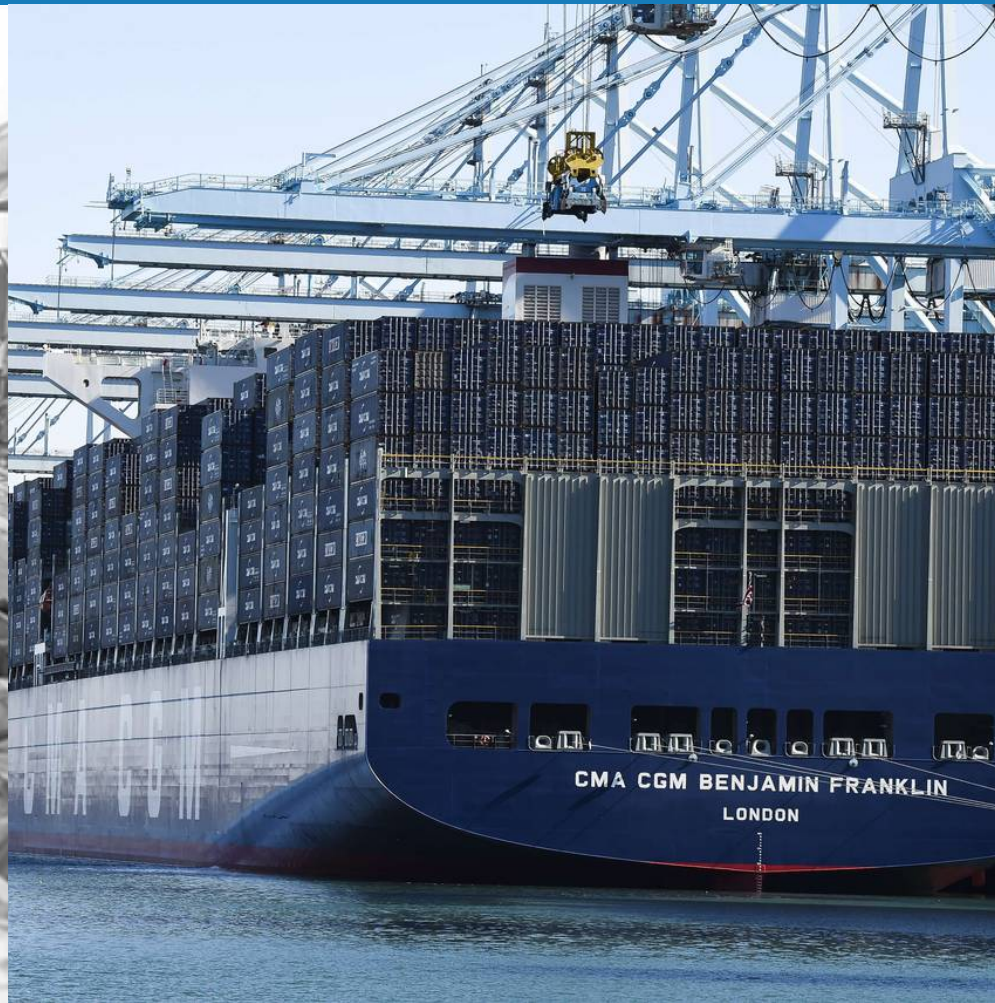
- Warum?
- Basics
- Docker Networking
- Arbeiten mit Docker
- Dockerfile
- Docker-Compose
- Best-Practices
- Troubleshooting
- Security

Allgemeine Infos

Demo & Beispiele (Source-Code):

https://github.com/christofluethi/docklands/tree/master/docker-techtalk

Warum?


CMA CGM BENJAMIN FRANKLIN
LONDON

Container
- Logistik revolutioniert
- Universeller genormter Transportbehälter
- Standardisierte Beschriftung

Hersteller / Absender
- Inhalt geschützt (Beschädigung, Sicherheit)
- Bekannte Abmessungen

Logistiker
- Hohe Umschlaggeschwindigkeit
- Hohe Automation
- Effiziente Lagerhaltung (Platz/Stapelbarkeit)
- Kombiverkehr (Schiff, Schiene, Lastwagen)

Warum Container?

- Standardisierung (Einheitliche Schnittstelle für Betrieb)
- Ressourcen-Limitierung
- Isolation der Umgebung
  - Filesystem-Isolation
  - Netzwerk-Isolation
- Wiederverwendbarkeit
  - Alle Umgebungen
  - Unterschiedliche Projekte
- Effizienz
  - Entwicklung (Technologie ist schnell einsatzbereit)
  - Technisch (Niedriger Footprint, schnelle Bereitstellung)


WORKED FINE
IN DEV
OPS PROBLEM NOW

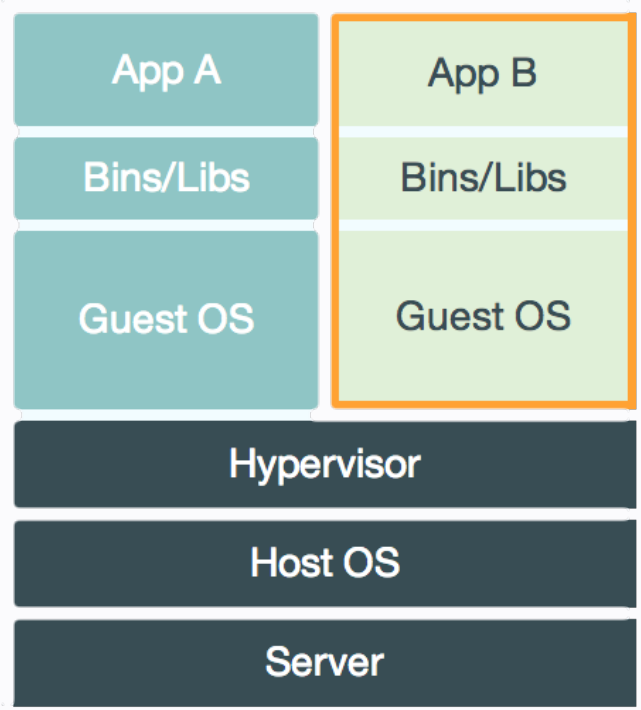
Warum Docker?

- Container gibt es schon lange (LXC 2008)
- Komplexität der unterliegenden Technologien
- Ecosystem (Images, Registry, Docker Hub)
- Massentaugliche Technologie
  - docker pull, run, build, push

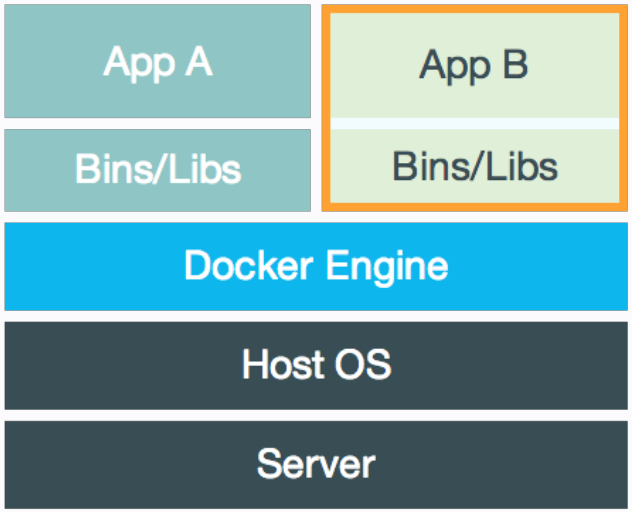
> *Simplicity is the art of hiding complexity. –Rob Pike, Google Inc., 2005*

Basics

VM vs. Container



System-Isolation

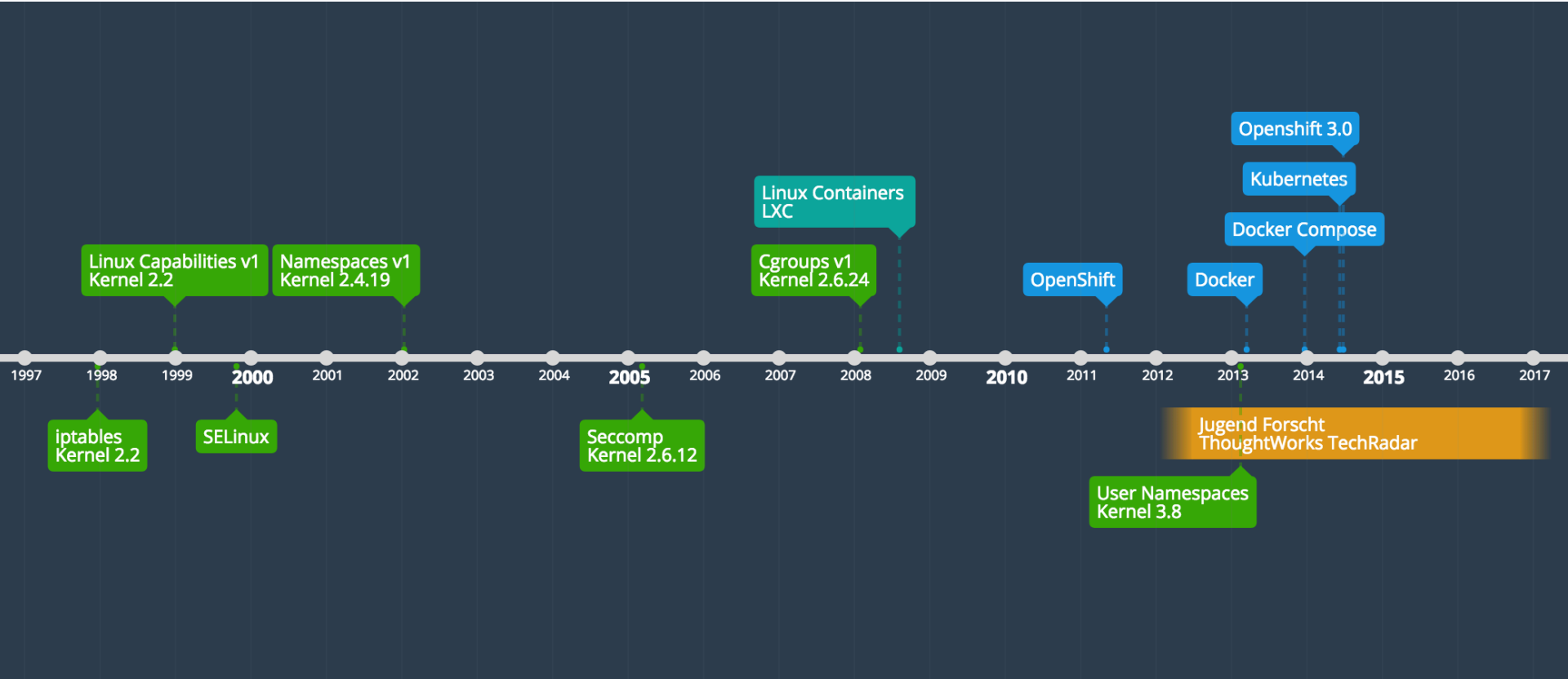


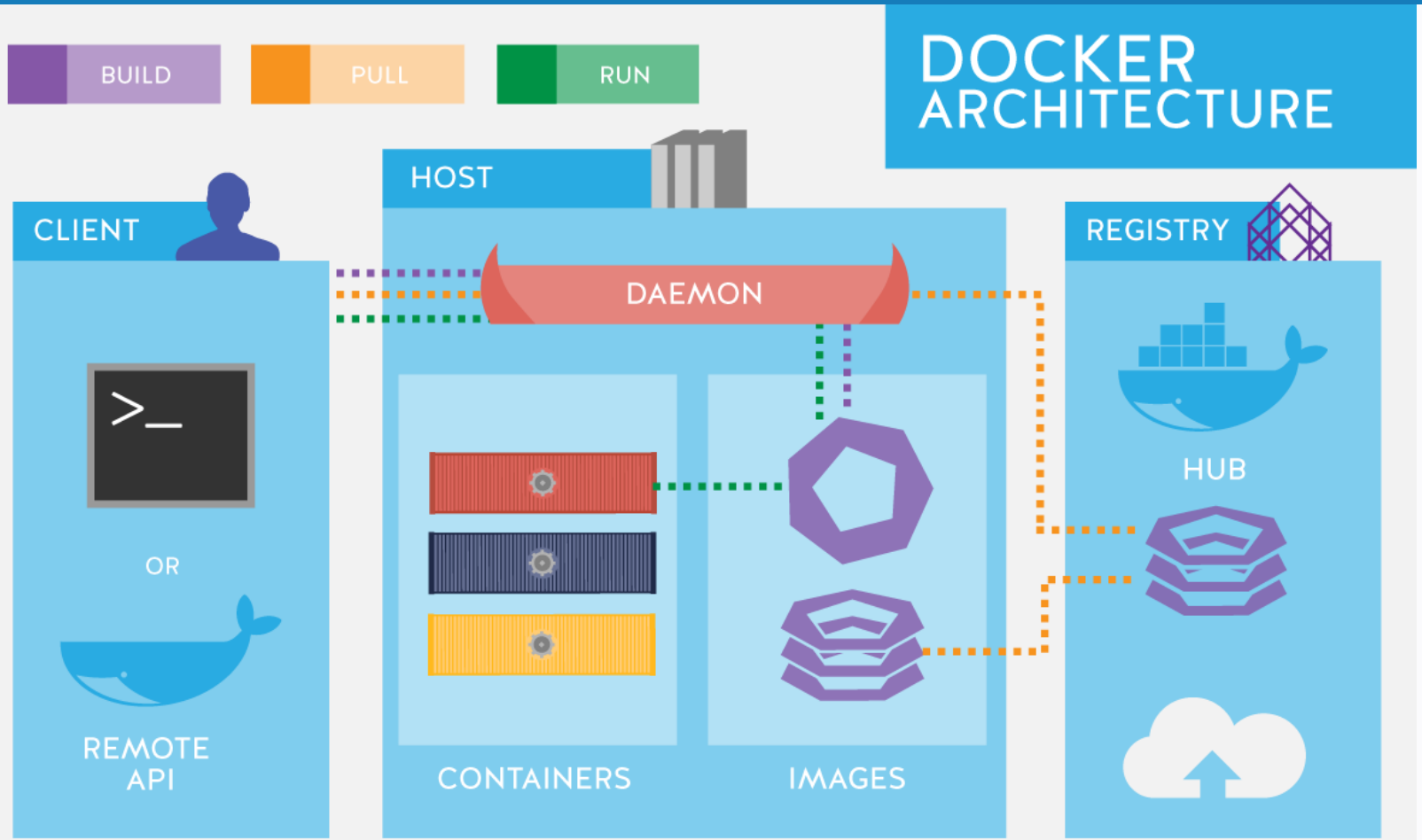
Prozess-Isolation

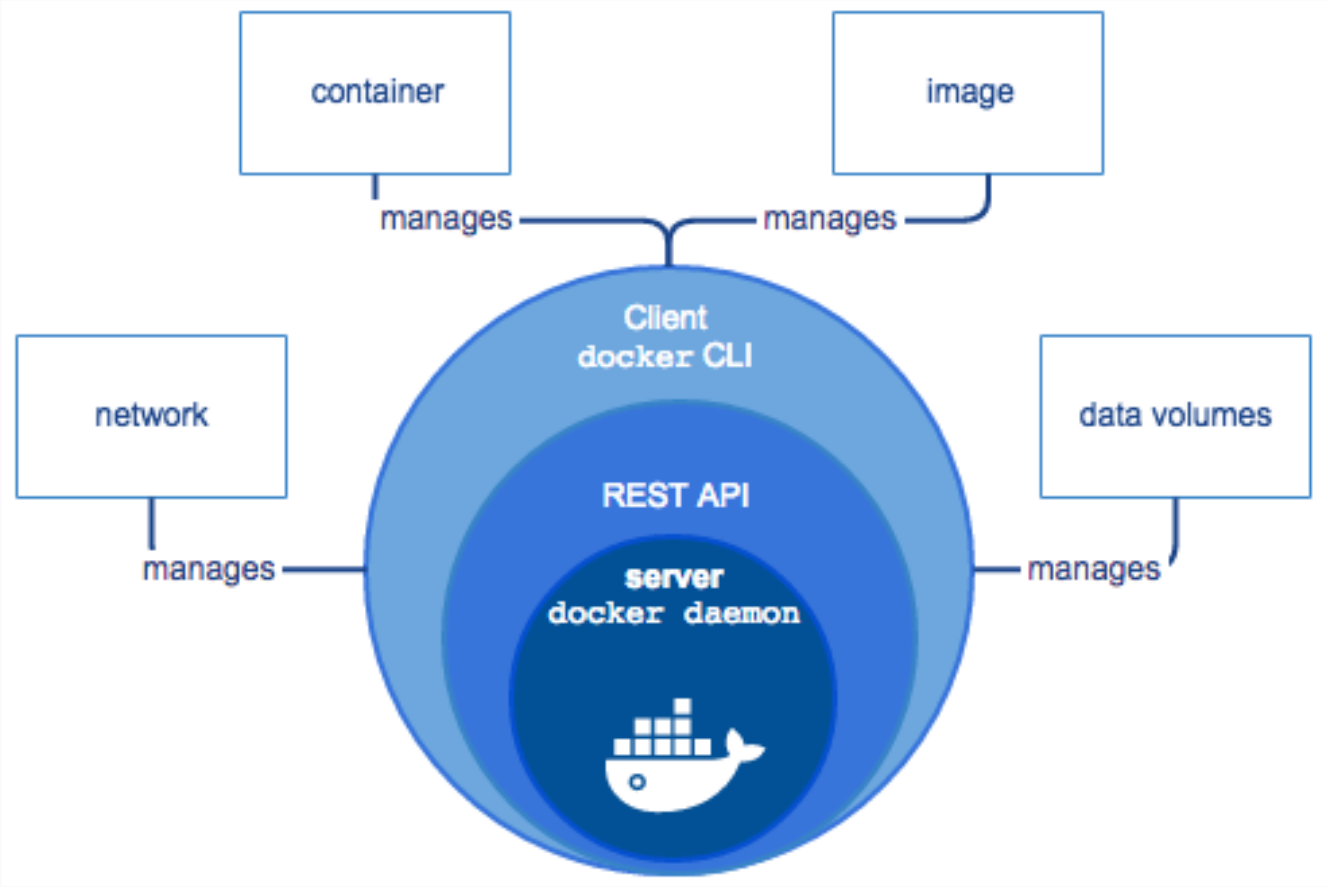
Verwendete Technologien

Control Groups *cgroups (2008)*	Teilen und Limitieren von Hardware Ressourcen - Memory, CPU, Block IO, Network
Namespaces *(2002)*	Isolation - Process IDs, Sockets, Network, Sysctls, Limits
Union File System *UnionFS*	Layered file system - Read-Only layers (image) - Read-Write container layer
Iptables *(1997)*	Netzwerk-Isolation, forwarding und NAT
Security Enhanced Linux *SELinux (1999)*	Zugriffskontrollen auf Ressourcen (Mandatory Access Control) Dateien, Verzeichnisse, Sockets, Devices, ...
Secure Computing Mode *seccomp (2005)*	Limitiert System Calls eines Containers auf den Kernel -> Aktuell 44 von 300 geblockt
Linux Capabilities *(1999)*	Fein-granulare Berechtigungen eines Prozesses für Kernel Features

History


Openshift 3.0
Kubernetes
Docker Compose
Linux Containers
LXC
Linux Capabilities v1
Kernel 2.2
Namespaces v1
Kernel 2.4.19
Cgroups v1
Kernel 2.6.24
OpenShift
Docker
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
iptables
Kernel 2.2
SELinux
Seccomp
Kernel 2.6.12
Jugend Forscht
ThoughtWorks TechRadar
User Namespaces
Kernel 3.8


BUILD
PULL
RUN
DOCKER ARCHITECTURE
CLIENT
HOST
REGISTRY
DAEMON
OR
REMOTE API
CONTAINERS
IMAGES
HUB
NORDICAPIS.COM

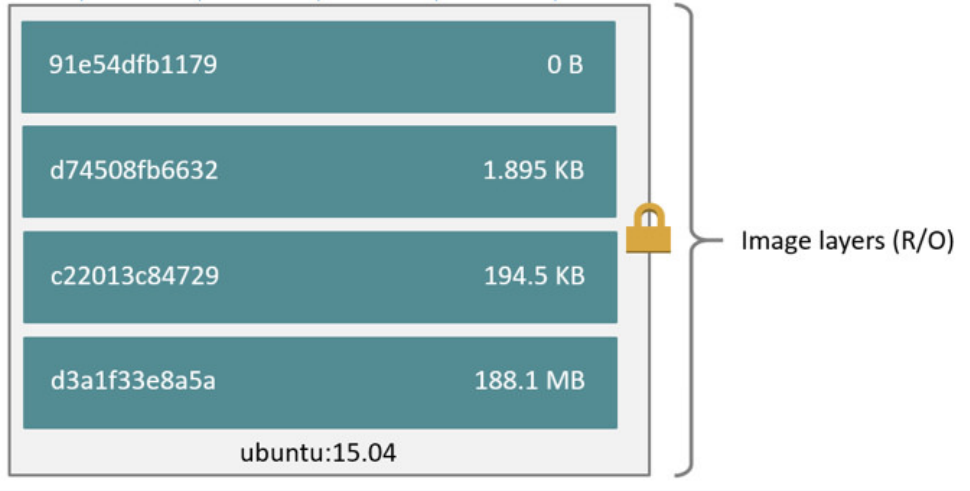

container
image
manages
manages
Client
docker CLI
network
data volumes
REST API
manages
server
docker daemon
manages

Was ist ein Image?

- Besteht aus 1-n Layern
- Hierarchisch
- Read-Only
- Template für Container

Layer

- FS Snapshot nach Befehl
  - Nur FS-Änderungen zum vorherigen Layer
- Wiederverwendbar

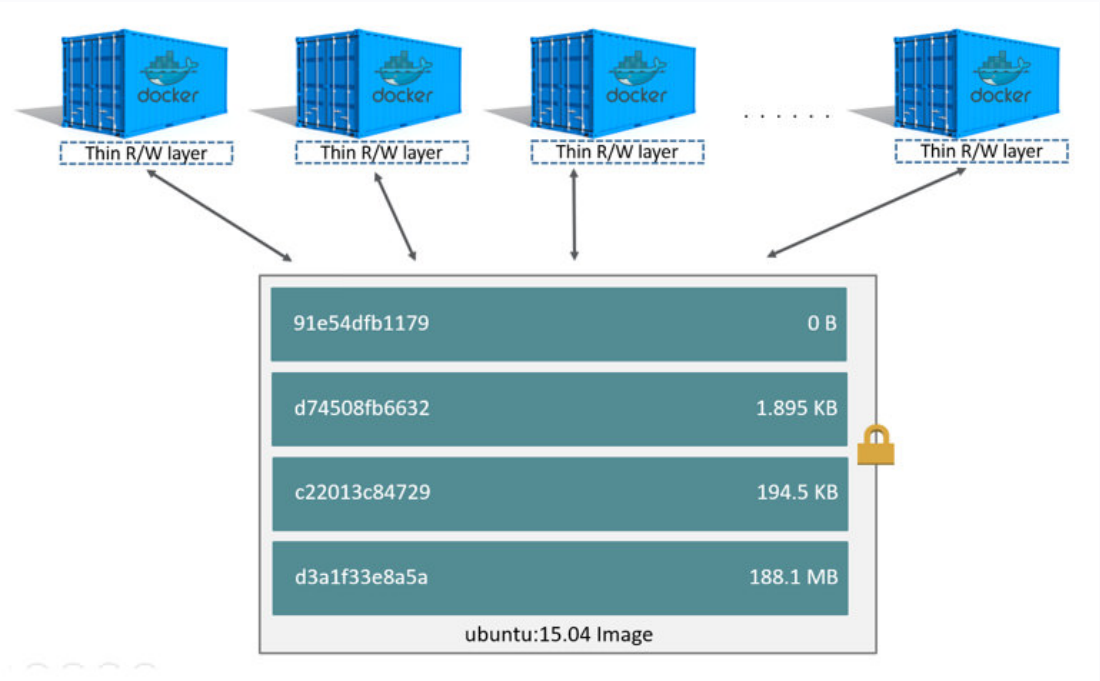


Was ist ein Container?

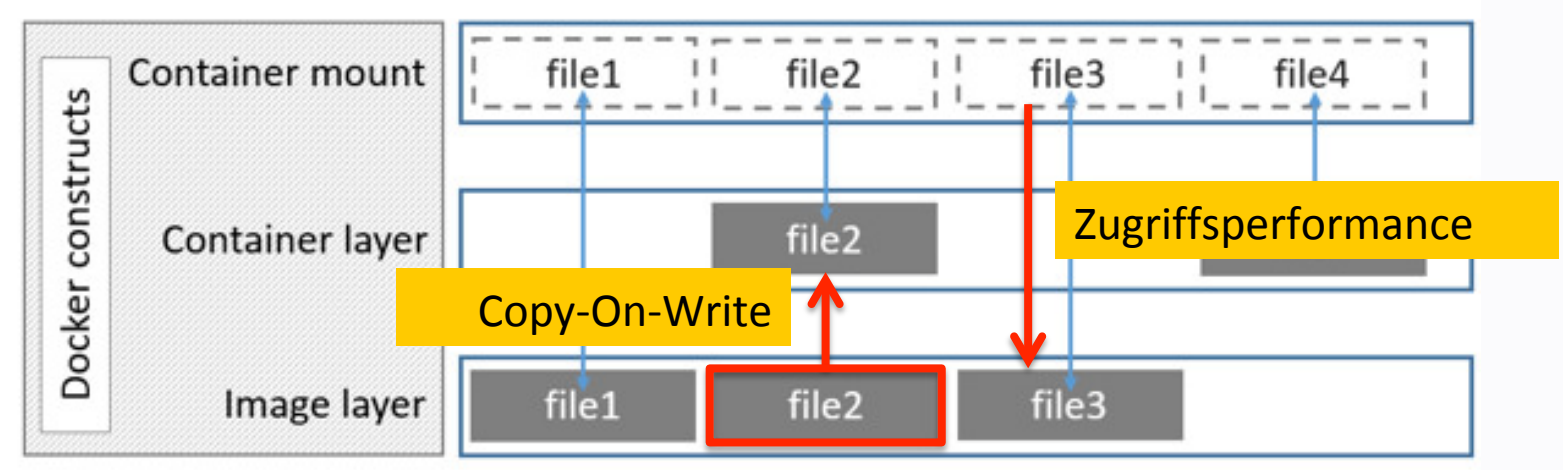
- Image + Container layer
- Runtime Konfiguration
  - Name
  - Netzwerk/Ports/IP
  - Limits (CPU/Ram)
  - Mounts
  - Volumes

Container layer

- Top layer
- Read-Write
  - Copy-On-Write
- Nicht persistent



Docker Layers




```
$ docker ps –s

NAME            SIZE
picapport       144MB (virtual 276MB)
prometheus      0B (virtual 112MB)
alertmanager    0B (virtual 31.9MB)
```

Size: Container Layer
Virtual: Image Layer + Container Layer

Image Name/Tag

```
REPOSITORY/NAME:TAG
```

Registry URL: docker.io, Official Image, Tag: latest

```
httpd
```

Registry URL: docker.io, Repository: centos, Name: httpd-24-centos7, Tag: 2.4

```
centos/httpd-24-centos7:2.4
```

Registry URL: docker.registryurl.com:5000, Name: xyz/xyz-abc, Tag: 1.0.0

```
docker.registryurl.com:5000/xyz/xyz-abc:1.0.0
```

Docker Directories

- /var/lib/docker

```
ls -l /var/lib/docker

drwx------  32 root root  4096 Jan 11 00:31 containers
drwx------   3 root root  4096 Aug  6 17:22 image
drwxr-x---   3 root root  4096 Aug  6 17:22 network
drwx------ 339 root root 45056 Jan 11 00:31 overlay2
drwx------  29 root root  4096 Aug 19 23:28 volumes
```

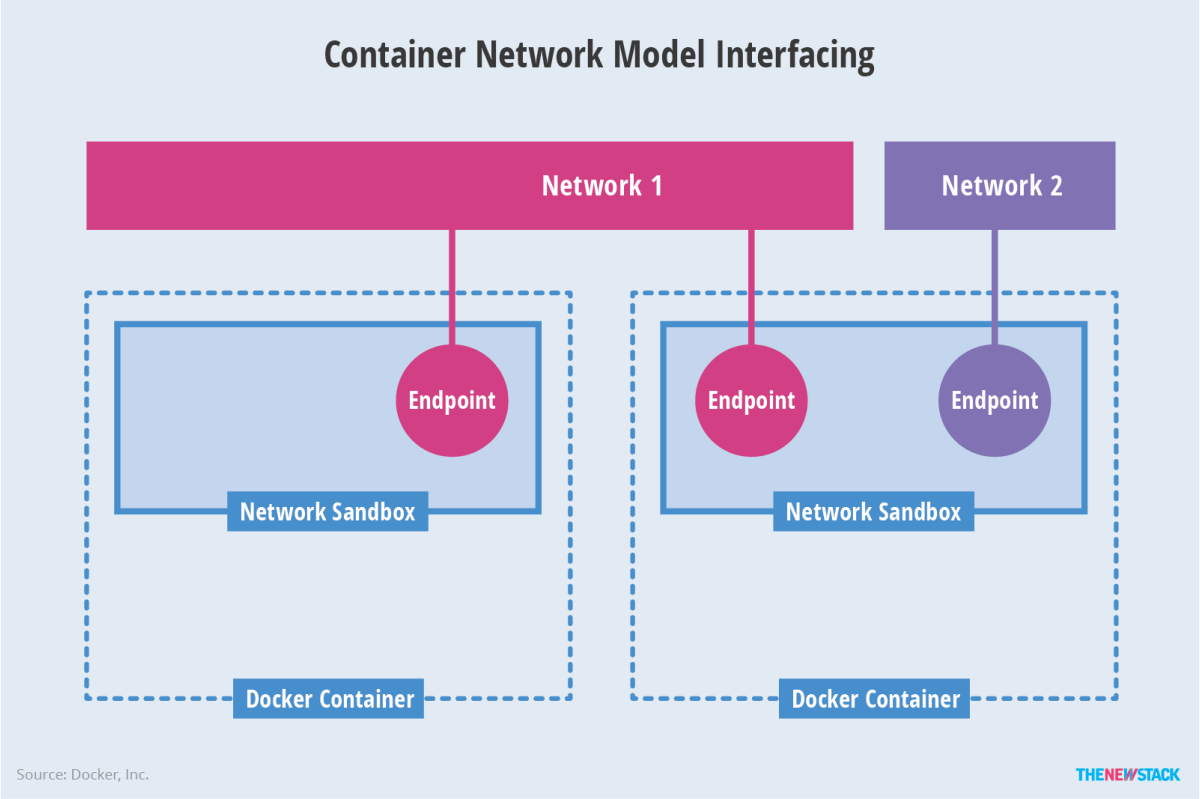
Bei Windows liegt dies in einer MobiLinuxVM

Docker Networking

Container Network Model (CNM)

- Abstraktion
- Standard

- Sandbox
- Endpoint
- Network

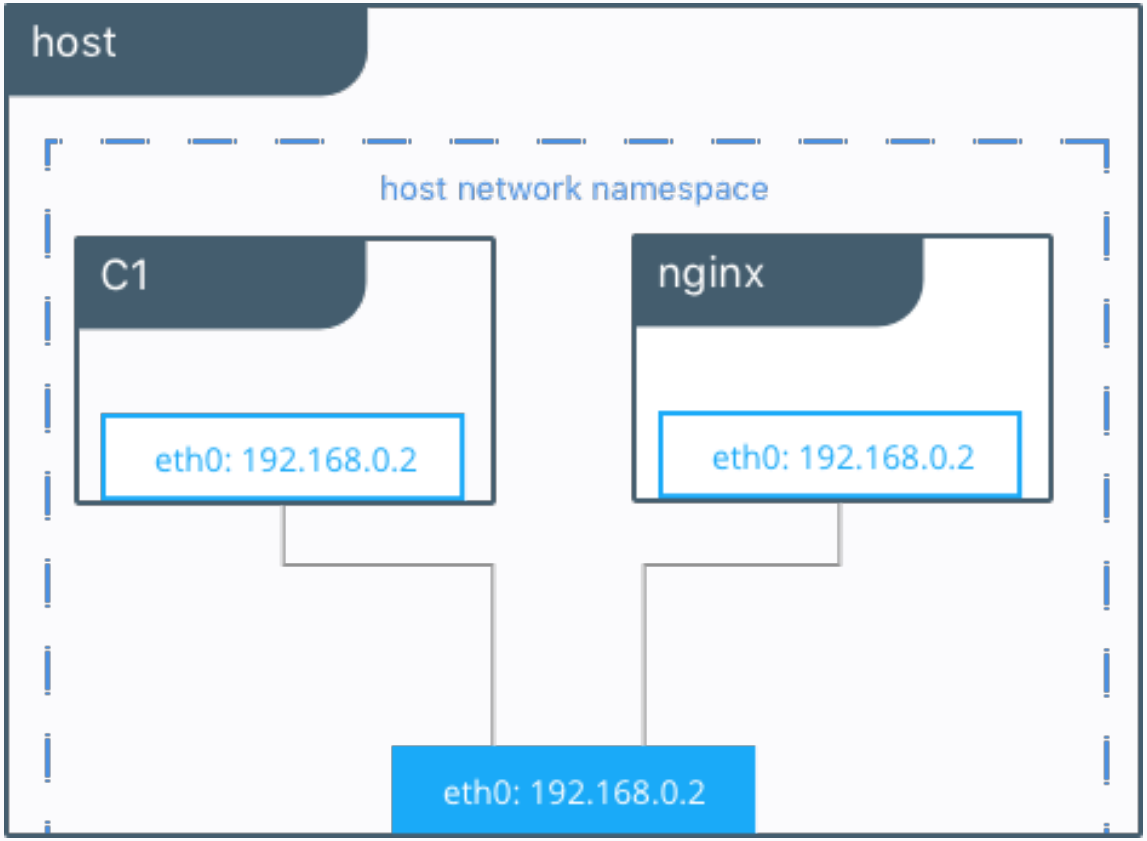


Netzwerk Typen

- Host
- Bridge (Standard, docker-compose)
- Overlay (Über mehrere Hosts – Swarm/Cluster)
- MacVlan (MAC Adresse wie richtige VMs)
- None

Host Netzwerk

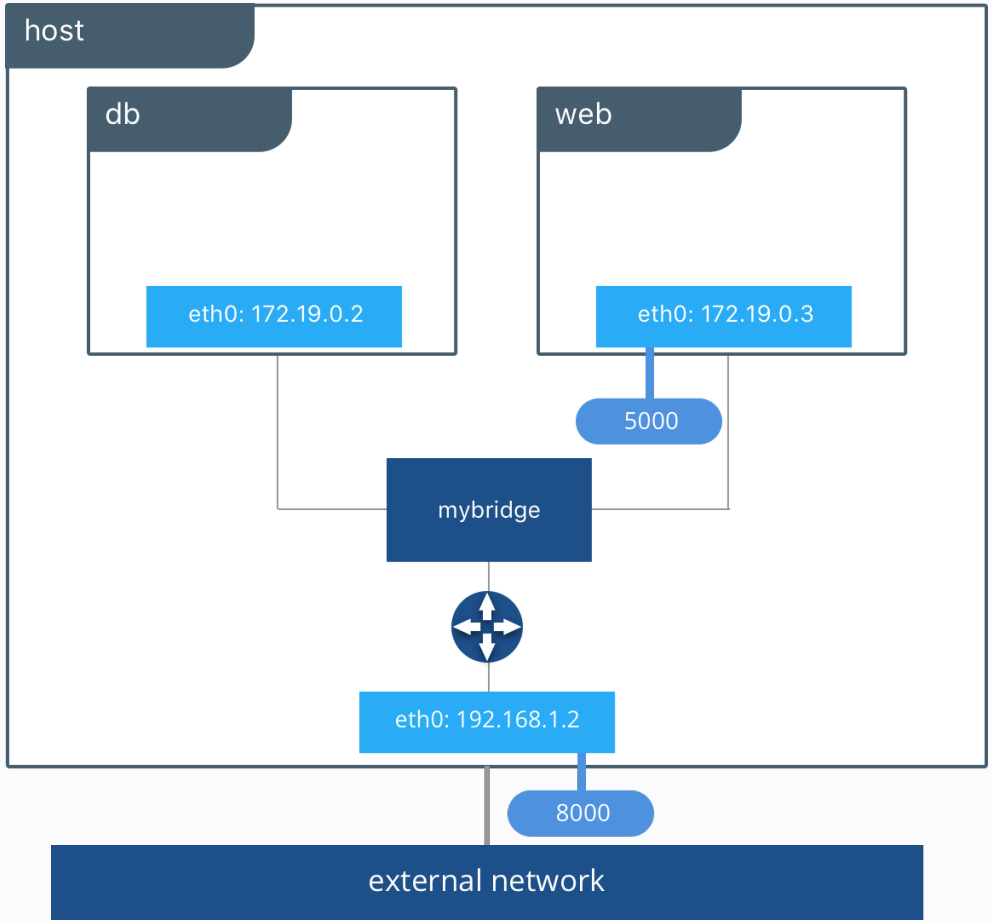
- Keine Isolation
- Host ports
- Performant



Bridge Netzwerk

- Isolation
- Port-Mapping nötig
- Interner DNS
- Interne Kommunikation

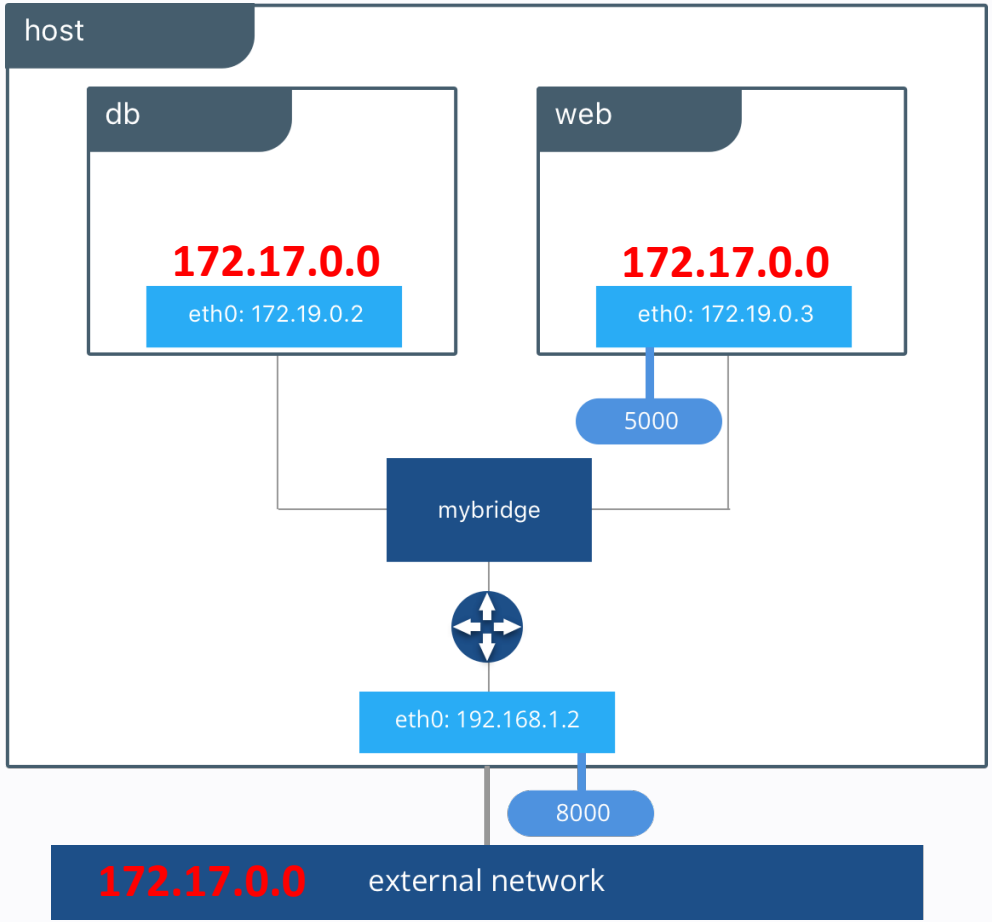
- Default Bridge docker0
  - Kein interner DNS



Bridge Netzwerk

- Default 172.17.0.0
- Netzwerk-“Konflikt“
  - Routing Problem

- Default Bridge docker0
  - BIP (daemon.json config)
- Custom Bridges
  - Explizit konfigurieren



Demo & Beispiele

«networking»

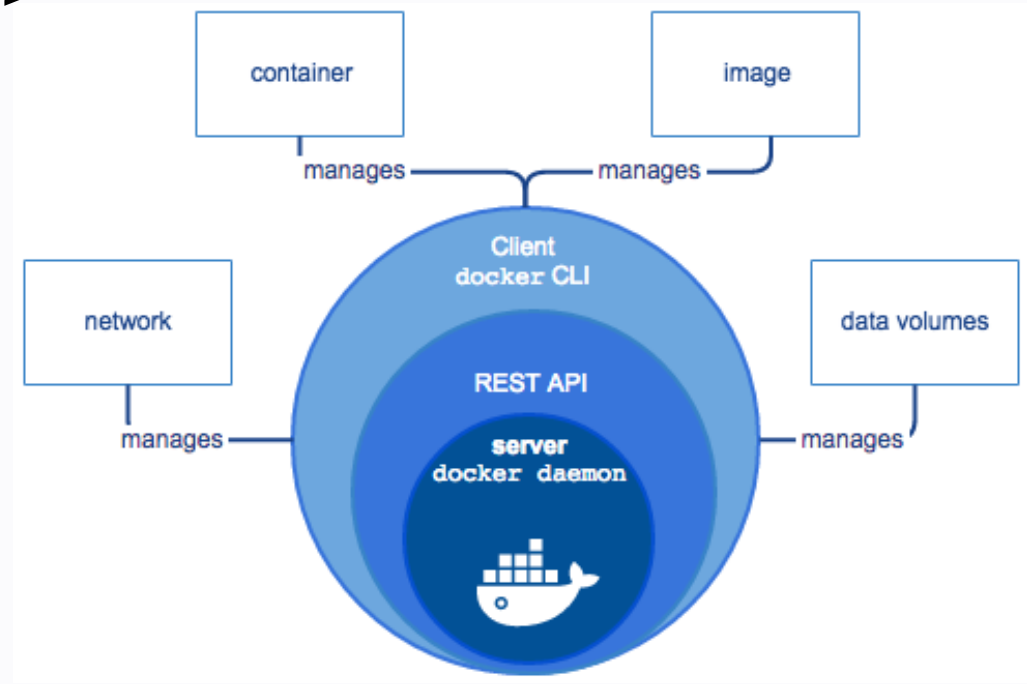
Arbeiten mit Docker

Docker CLI

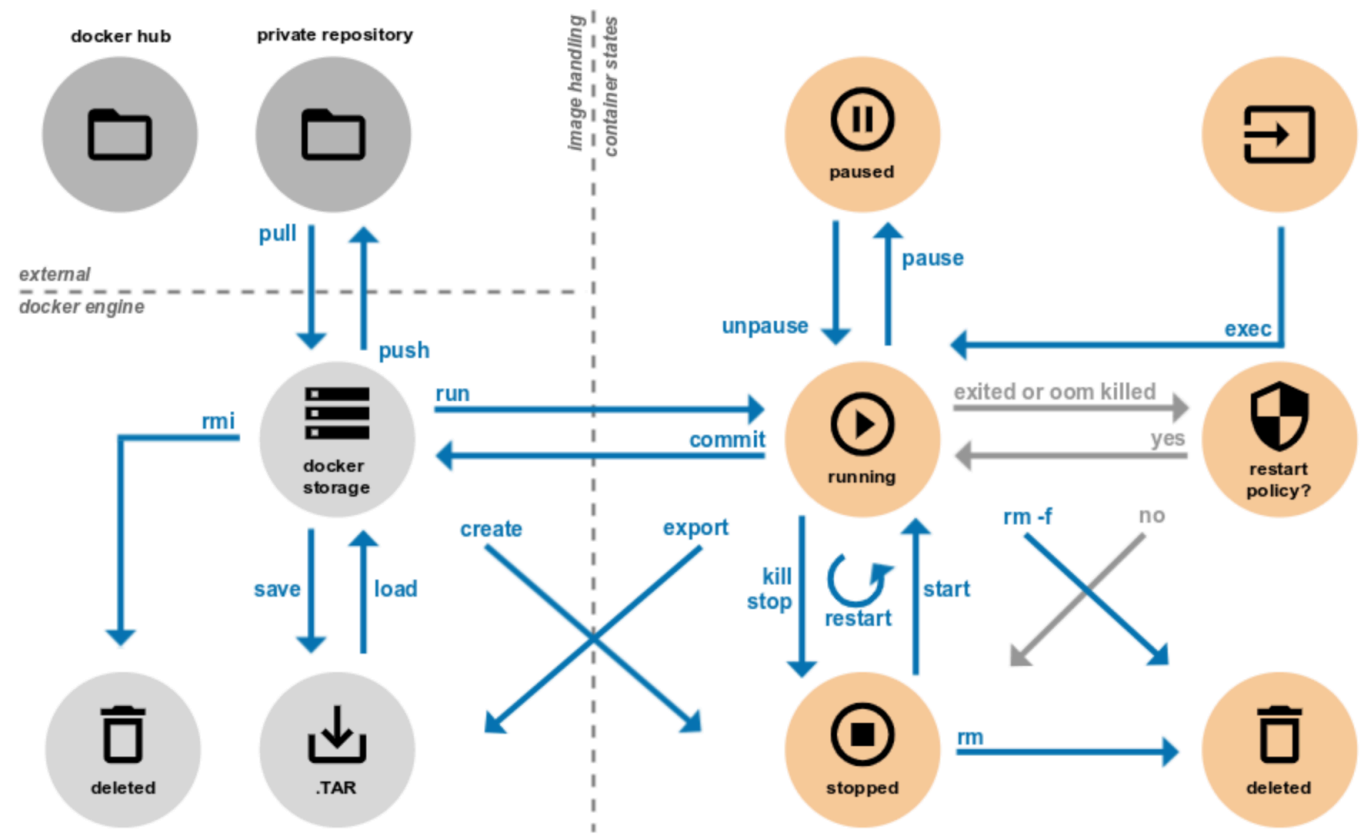
Docker CLI

docker <object> <action>

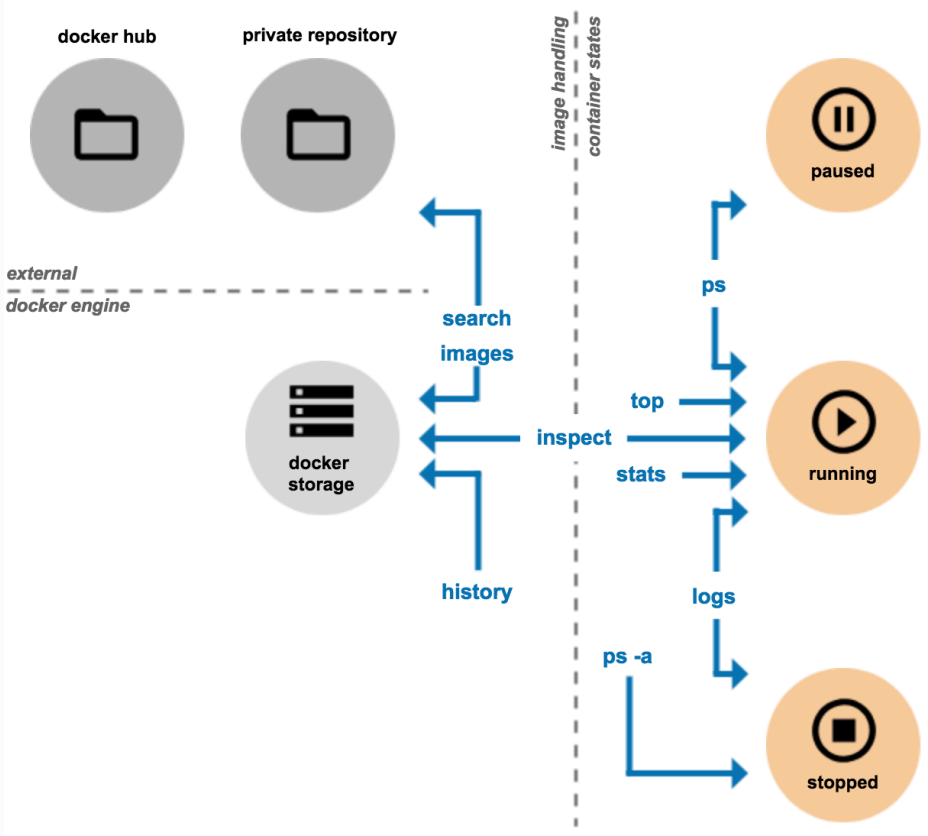
- container
- image
- volume
- network
- system
- ...



Docker CLI Actions / Container Lifecycle



Docker CLI Queries



Container Erstellen

- Benötigt ein Image (z.B. https://hub.docker.com/_/httpd)
- Pull erfolgt automatisch
- Run erstellt Container anhand Image und startet diesen
  - -d für detached

```
docker pull httpd
docker run –d –name cluweb httpd
docker ps
```

```
# pull von docker hub
# create, run container
# show containers
```

Image Erstellen

- Benötigt ein Dockerfile mit Anweisungen

```
docker build .
...
Successfully built 285ed64865aa
```

```
docker image ls
<none>      <none>     285ed64865aa      58 seconds ago     221MB
```

```
docker build . -t docker.registryurl.com:5000/christofluethi/cluweb:1.0.0
```

Image Taggen

- Benötigt ein Image

```
docker tag 285ed64865aa christofluethi/cluweb:1.0.0
docker tag 285ed64865aa hello/world:latest
docker tag hello/world:latest hello/world:1.0.0
docker tag hello/world:latest hello/world:1.1.0
```

```
docker image ls

REPOSITORY                TAG       IMAGE ID            CREATED              SIZE
christofluethi/cluweb     1.0.0     285ed64865aa        16 minutes ago       221MB
hello/world               1.0.0     285ed64865aa        16 minutes ago       221MB
hello/world               1.1.0     285ed64865aa        16 minutes ago       221MB
hello/world               latest    285ed64865aa        16 minutes ago       221MB
```

Container Exec

- Befehl in laufendem Container ausführen
- Debugging
- -i: interactive (stdin)
- -t: tty (stdout; pseudo terminal)

```
docker exec –it <container> <cmd>
```

Demo & Beispiele

«dockercli»

Dockerfile

Dockerfile

- Bauplan für Image

Wichtigste Befehle

- FROM: Basis Image
- COPY: Datei von Host in Image kopieren
- RUN: Befehl ausführen
- ENTRYPOINT: Eintrittspunkt im Container
- CMD: Default Befehl oder Argumente für Entrypoint

Dockerfile

- LABEL: Metainformation zum Image
- ENV: Environment Variable
- ARG: Build Argument (--build-arg var=val)
- VOLUME: Volume Spezifikation
- EXPOSE: Metainformation über offene Netzwerk-Ports
- WORKDIR: Aktuelles Verzeichnis
- USER: Laufzeit-User

Dockerfile - ENTRYPOINT und CMD

```
FROM alpine

COPY entrypoint.sh /
RUN chmod +x entrypoint.sh

ENTRYPOINT ["/entrypoint.sh"]
CMD ["hello“, "world"]
```

entrypoint.sh

```
#!/bin/sh
echo $@
```

```
$ docker run techtalk/entrypoint-demo
hello world

$ docker run techtalk/entrypoint-demo docker-techtalk
docker-techtalk
```

Dockerfile - ENTRYPOINT und CMD

	No ENTRYPOINT	ENTRYPOINT exec_entry p1_entry	ENTRYPOINT [“exec_entry”, “p1_entry”]
No CMD	*error, not allowed*	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry
CMD [“exec_cmd”, “p1_cmd”]	exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry exec_cmd p1_cmd
CMD [“p1_cmd”, “p2_cmd”]	p1_cmd p2_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry p1_cmd p2_cmd
CMD exec_cmd p1_cmd	/bin/sh -c exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd

Demo & Beispiele

«demo-dockerfile»

«chdir»

«healthcheck»

«entrypoint»

Docker Compose

Docker Compose

- Multi-Container setup
- Definition der Umgebung als YAML (docker-compose.yml)
- Restartet nur geänderte Container

Wichtigste Befehle

- docker-compose up -d
- docker-compose down
- docker-compose stop service
- docker-compose start service

CLI vs. Docker Compose

Docker CLI

```
$ docker network create –d bridge \
         --subnet 192.168.219.1/24 xyznet

$ docker run –d --network xyznet --name abc \
         –e POSTGRES_HOST=db –e ENVIRONMENT=development \
         –p 8080:8080 xyz/abc
```

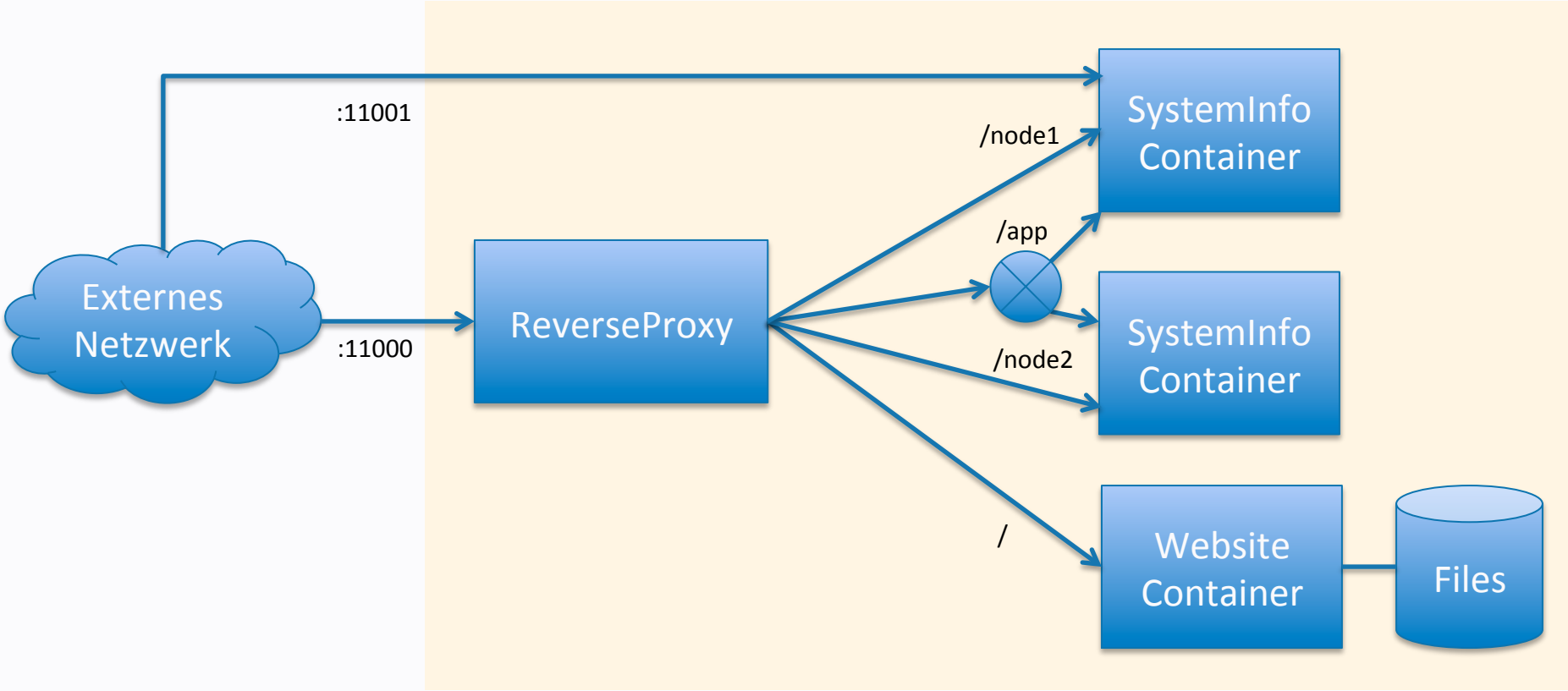
Docker Compose (docker-compose.yml)

```
version: "3"
services:
  consulting:
    image: xyz/abc
    networks:
      xyznet:
    ports:
      - 8080:8080
    environment:
      - ENVIRONMENT=development
      - POSTGRES_HOST=db
…
networks:
  xyznet:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 192.168.219.1/24
```

Demo & Beispiele

«demo-app»

Docker Compose – Demo App



Docker Compose - Override

Default Compose-File

- docker-compose.yml

Default Override-File

- docker-compose.override.yml

Weitere Compose-Files

- docker-compose –f docker-compose.yml –f docker-compose.prod.yml

Best Practices

Best Practices

Efficient Dockerfile

Simple Project

- Simple Java App „Hello World“
  - Sourcecode: 18 Zeilen, 380 Bytes
- Docker Container

```
$ ls -l
total 24
-rw-rw-r--  1 shaped  staff    1  9 Jan 23:42 Dockerfile
-rw-rw-r--  1 shaped  staff   61  9 Jan 23:41 README.md
drwxrwxr-x  3 shaped  staff   96  9 Jan 23:41 docs
-rw-rw-r--  1 shaped  staff  925  9 Jan 23:35 pom.xml
drwxrwxr-x  3 shaped  staff   96  9 Jan 23:35 src
drwxrwxr-x  8 shaped  staff  256  9 Jan 23:42 target
```

Dockerfile

```
FROM debian

COPY . /app
RUN apt-get update
RUN apt-get –y install openjdk-8-jdk ssh emacs curl

CMD ["java", "-jar", "/app/target/app.jar"]
```

```
Java app.jar:    16.7mb

Base Image:      101mb (2 Layers)
Custom Image:    804mb (6 Layers)
Build time:      1m55.615sec
```

Dockerfile – Reihenfolge

```
FROM debian

COPY . /app
RUN apt-get update
RUN apt-get –y install openjdk-8-jdk ssh emacs curl
COPY . /app

CMD ["java", "-jar", "/app/target/app.jar"]
```

```
Java app.jar:    16.7mb

Base Image:      101mb (2 Layers)
Custom Image:    804mb (6 Layers)
Build time:      0m4.704s                -1m50sec
```

Dockerfile – Copy

```
FROM debian

RUN apt-get update
RUN apt-get -y install openjdk-8-jdk ssh emacs curl
~~COPY . /app~~
COPY target/app.jar /app/

CMD ["java", "-jar", "/app~~/target~~/app.jar"]
```

Java app.jar: 16.7mb

Bessere Verwendung des Caches

Dockerfile – Gruppieren

```
FROM debian

RUN apt-get update
RUN apt-get –y install openjdk-8-jdk ssh emacs curl
RUN apt-get update && apt-get –y install openjdk-8-jdk ssh emacs curl
COPY target/app.jar /app/

CMD ["java", "-jar", "/app/app.jar"]
```

Java app.jar: 16.7mb

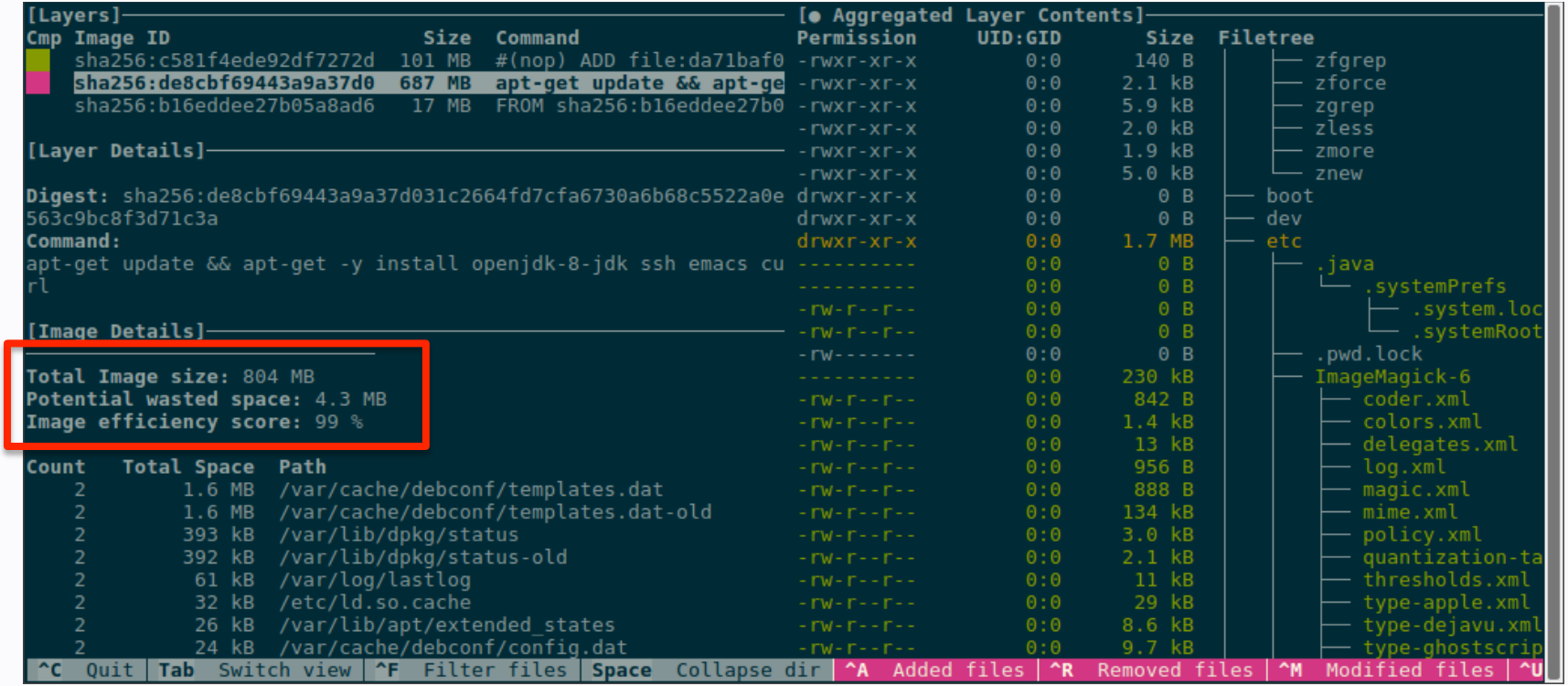
Bessere Verwendung des Caches – Korrektur für apt-get

Dockerfile – Grösse?

```
FROM debian

RUN apt-get update && apt-get –y install openjdk-8-jdk ssh emacs curl
COPY target/app.jar /app/

CMD ["java", "-jar", "/app/app.jar"]
```

```
docker history techtalk-demo:4

IMAGE               CREATED BY                                      SIZE
8f13824166ad        /bin/sh -c #(nop)  CMD ["java" "-jar" "/app/…   0B
39a78304bd4e        /bin/sh -c #(nop) COPY file:e7c88e055ee62868…   16.7MB
6e5997fd2779        /bin/sh -c apt-get update && apt-get -y inst…   687MB
de8b49d4b0b3        /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>           /bin/sh -c #(nop) ADD file:da71baf0d22cb2ede…   101MB
```

Dockerfile – Grösse?



dive: https://github.com/wagoodman/dive

Dockerfile – Überflüssige Pakete

```
FROM debian

RUN apt-get update && apt-get –y install openjdk-8-jdk ssh emacs curl
COPY target/app.jar /app/

CMD ["java", "-jar", "/app/app.jar"]
```

```
Java app.jar:    16.7mb

Base Image:      101mb (2 Layers)
Custom Image:    597mb (5 Layers)        -207mb
```

Dockerfile – --no-install-recommends

```
FROM debian

RUN apt-get -qq update && apt-get -y install --no-install-recommends \
    openjdk-8-jdk
COPY target/app.jar /app/

CMD ["java", "-jar", "/app/app.jar"]
```

```
Java app.jar:    16.7mb

Base Image:      101mb (2 Layers)
Custom Image:    494mb (5 Layers)        -103mb
```

Dockerfile – Package manager cache

```
FROM debian

RUN apt-get -qq update && apt-get -y install --no-install-recommends \
    openjdk-8-jdk \
    && rm -rf /var/lib/apt/lists/*
COPY target/app.jar /app/

CMD ["java", "-jar", "/app/app.jar"]
```

```
Java app.jar:    16.7mb

Base Image:      101mb (2 Layers)
Custom Image:    478mb (5 Layers)        -16mb
```

Dockerfile – Base-Images

```
FROM debian
FROM openjdk:8

RUN apt-get -qq update && apt-get -y install --no-install-recommends \
    openjdk-8-jdk \
    && rm -rf /var/lib/apt/lists/*
COPY target/app.jar /app/

CMD ["java", "-jar", "/app/app.jar"]
```

```
Java app.jar:    16.7mb

Base Image:      624mb (13 Layers)
Custom Image:    640mb (15 Layers)        +162mb
```

Dockerfile – Base-Images

```
FROM openjdk:8-jre

COPY target/app.jar /app/

CMD ["java", "-jar", "/app/app.jar"]
```

```
Java app.jar:    16.7mb

Base Image:      442mb (12 Layers)
Custom Image:    459mb (14 Layers)        -181mb
```

Dockerfile – Base-Images

```
FROM openjdk:8-jre-slim

COPY target/app.jar /app/

CMD ["java", "-jar", "/app/app.jar"]
```

```
Java app.jar:    16.7mb

Base Image:      204mb (10 Layers)
Custom Image:    221mb (12 Layers)        -238mb
```

Dockerfile – Base-Images

```
FROM openjdk:8-jre-alpine

COPY target/app.jar /app/

CMD ["java", "-jar", "/app/app.jar"]
```

Java app.jar:	16.7mb	
Base Image:	83mb (9 Layers)	
Custom Image:	100mb (11 Layers)	-121mb (-704mb)
Build time:	0m0.238sec	(-1m55sec)

Dockerfile

- Offizielle Images verwenden (wenn möglich)
  - Wurden getestet
  - Gemacht für Container
- Minimal images verwenden
- Die selben Basis-Images verwenden

REPOSITORY	TAG	SIZE
openjdk	8	624mb
openjdk	8-jre	442mb
openjdk	8-jre-slim	204mb
openjdk	8-jre-alpine	83mb

Demo & Beispiele

«effective-images»

Dockerfile Linting

- Lint your Dockerfiles with hadolint!
- Prüft Dockerfile anhand von 65+ Rules
- Hadolint: https://github.com/hadolint/hadolint

```
$ docker run --rm -i hadolint/hadolint < Dockerfile

/dev/stdin:8 DL3008 Pin versions in apt get install. Instead of `apt-get install <package>` use
`apt-get install <package>=<version>`
/dev/stdin:8 DL3015 Avoid additional packages by specifying `--no-install-recommends`
/dev/stdin:11 DL3020 Use COPY instead of ADD for files and folders
```

```
alias hadolint="docker run --rm -i hadolint/hadolint < $*"
```

Dockerfile Linting

```
$ hadolint Dockerfile
/dev/stdin:4 DL3008 Pin versions in apt get install. Instead of `apt-get install <package>` use `apt-get install
<package>=<version>`
/dev/stdin:4 DL3009 Delete the apt-get lists after installing something
/dev/stdin:4 DL3015 Avoid additional packages by specifying `--no-install-recommends`
/dev/stdin:15 DL3020 Use COPY instead of ADD for files and folders
/dev/stdin:18 DL3020 Use COPY instead of ADD for files and folders
/dev/stdin:20 DL3000 Use absolute WORKDIR
```

```
FROM nginx:1.13.9

RUN apt-get update && apt-get install -y vim
RUN ln -sf /usr/share/zoneinfo/Europe/Zurich /etc/localtime
RUN echo "Europe/Zurich" > /etc/timezone
RUN ln -sf /dev/stdout /var/log/nginx/access.log && ln -sf /dev/stderr /var/log/nginx/error.log

ADD build/www www/data
ADD nginx.conf /etc/nginx/nginx.conf

WORKDIR www/data
```

Dockerfile Linting

```
$ hadolint Dockerfile
```

```
FROM nginx:1.13.9

RUN set -x  && ln -sf /usr/share/zoneinfo/Europe/Zurich /etc/localtime && echo "Europe/Zurich" > /etc/timezone
RUN ln -sf /dev/stdout /var/log/nginx/access.log && ln -sf /dev/stderr /var/log/nginx/error.log

# hadolint ignore=DL3008
RUN apt-get -qq update && apt-get install -y --no-install-recommends vim && rm -rf /var/lib/apt/lists/*

COPY nginx.conf /etc/nginx/nginx.conf
COPY build/www /www/data

WORKDIR /www/data
```

Best Practices

Container Design

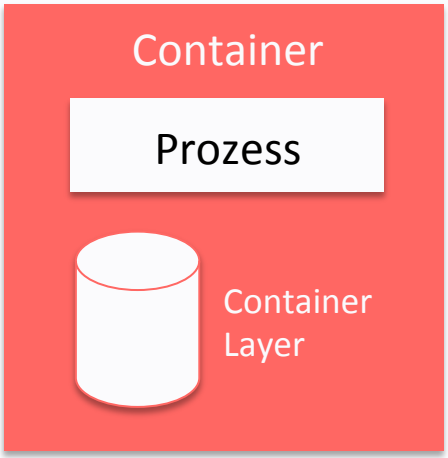
Single Process per Container

- Single Responsibility Prinzip
- Entspricht am ehesten Microservice-Architektur
- Resource-Limits einfacher (OOM Killer)
- Isolation nicht verletzt
- Healthchecks einfacher
- Signal-Handling einfacher
- Skalierung einfacher

Konfiguration im Container

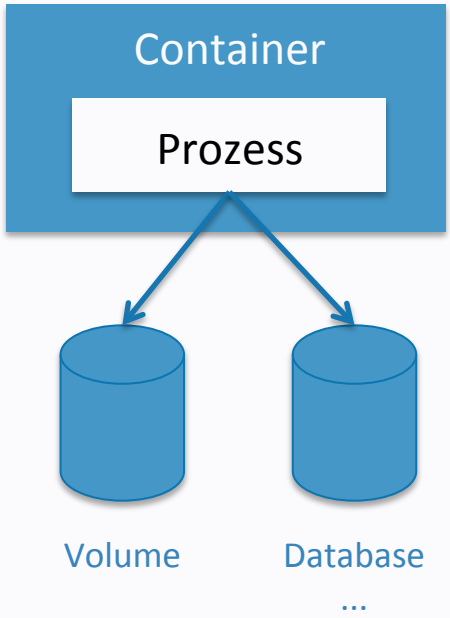
- Umgebungsspezifisch
  - ENV Variablen
  - *JAVA_OPTS, ENVIRONMENT, DB_PASSWORD*
- Dynamische Konfiguration
  - Config Files (Bind-Mount)
  - *Proxy nginx.conf, Java Properties-File*
- Statische Konfiguration
  - Config Files im Container
  - *Wildfly standalone.xml*

Stateless Container



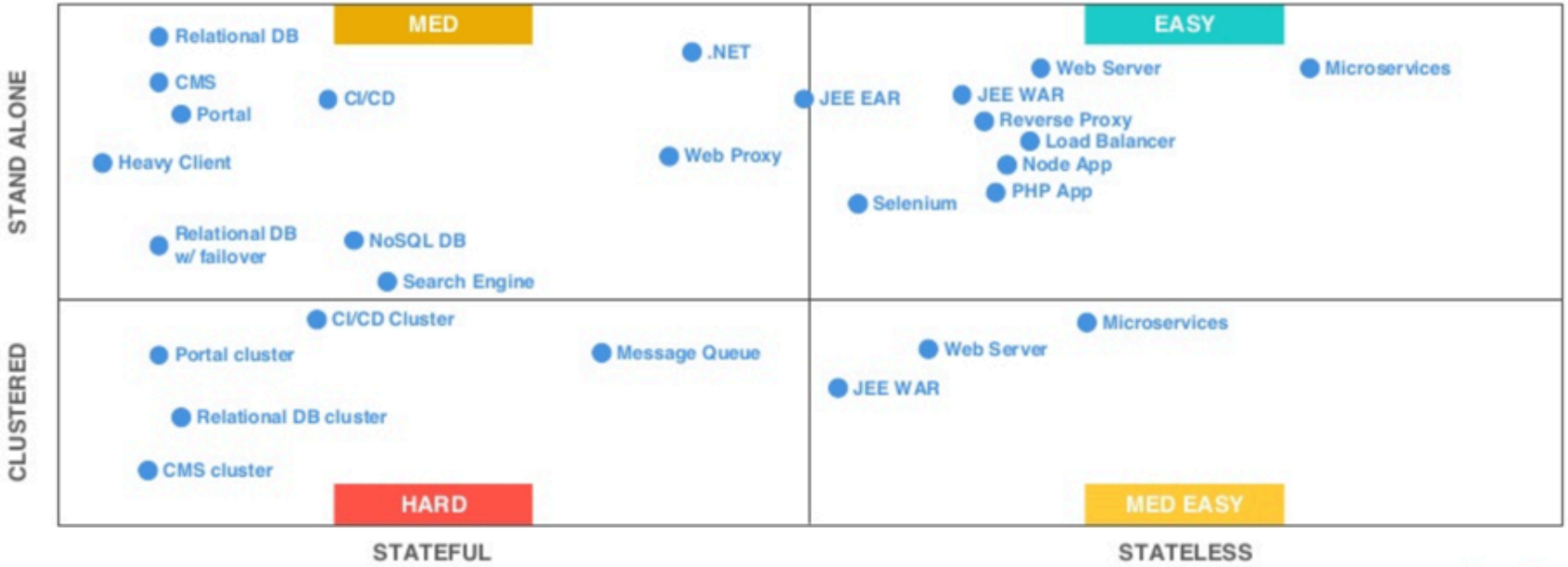
Zustand im Memory
Zustand auf der Disk
...

Container sind flüchtig
Container-Layer ist „langsam“



Stateless Container

Not all application require the same level of effort to Dockerize



Signal Handling

- Docker verwendet Signals zum Beenden
- Init Prozess im Container ist PID1 (Parent)
- PID1 muss die Signale beachten/weiterleiten

docker stop

- Sendet SIGTERM (15) an PID 1
- 10sec warten
- Sendet SIGKILL (9) wenn nicht bereits beendet

Signal Handling - PostgreSQL

```
$ docker stop xyz_db

$ docker ps -a --filter name=xyz_db
STATUS                          NAMES
Exited (137) 2 minutes ago      xyz_db
```

Exit code: 137
128 + 9 = killed by SIGKILL

Oder
Exit Code − 128 = SIGNAL

```
LOG:  received smart shutdown request
LOG:  autovacuum launcher shutting down

LOG:  database system was interrupted; last known up at 2019-01-09
09:05:42 CET
LOG:  database system was not properly shut down; automatic recovery in
progress
LOG:  redo starts at 0/86D3CB60
LOG:  invalid record length at 0/8722FBF0: wanted 24, got 0
LOG:  redo done at 0/8722FBC8
LOG:  last completed transaction was at log time 2019-01-09
09:06:31.486706+01
LOG:  MultiXact member wraparound protections are now enabled
LOG:  database system is ready to accept connections
```

Signal Handling - PostgreSQL

- SIGTERM: PostgreSQL waits for transactions to close
- SIGINT: terminate transactions and gracefully stop

```
db:
  image: xyz/db:latest
  stop_signal: SIGINT
  stop_grace_period: 1m30s
```

```
$ docker stop xyz_db
$ docker ps -a --filter name=xyz_db

STATUS                          NAMES
Exited (0) 2 minutes ago        xyz_db
```

```
LOG:  received fast shutdown request
LOG:  aborting any active transactions
FATAL:  terminating connection due to administrator command
LOG:  autovacuum launcher shutting down
LOG:  shutting down
LOG:  database system is shut down

LOG:  database system was shut down at 2019-01-09 09:10:49 CET
LOG:  MultiXact member wraparound protections are now enabled
LOG:  autovacuum launcher started
LOG:  database system is ready to accept connections
```

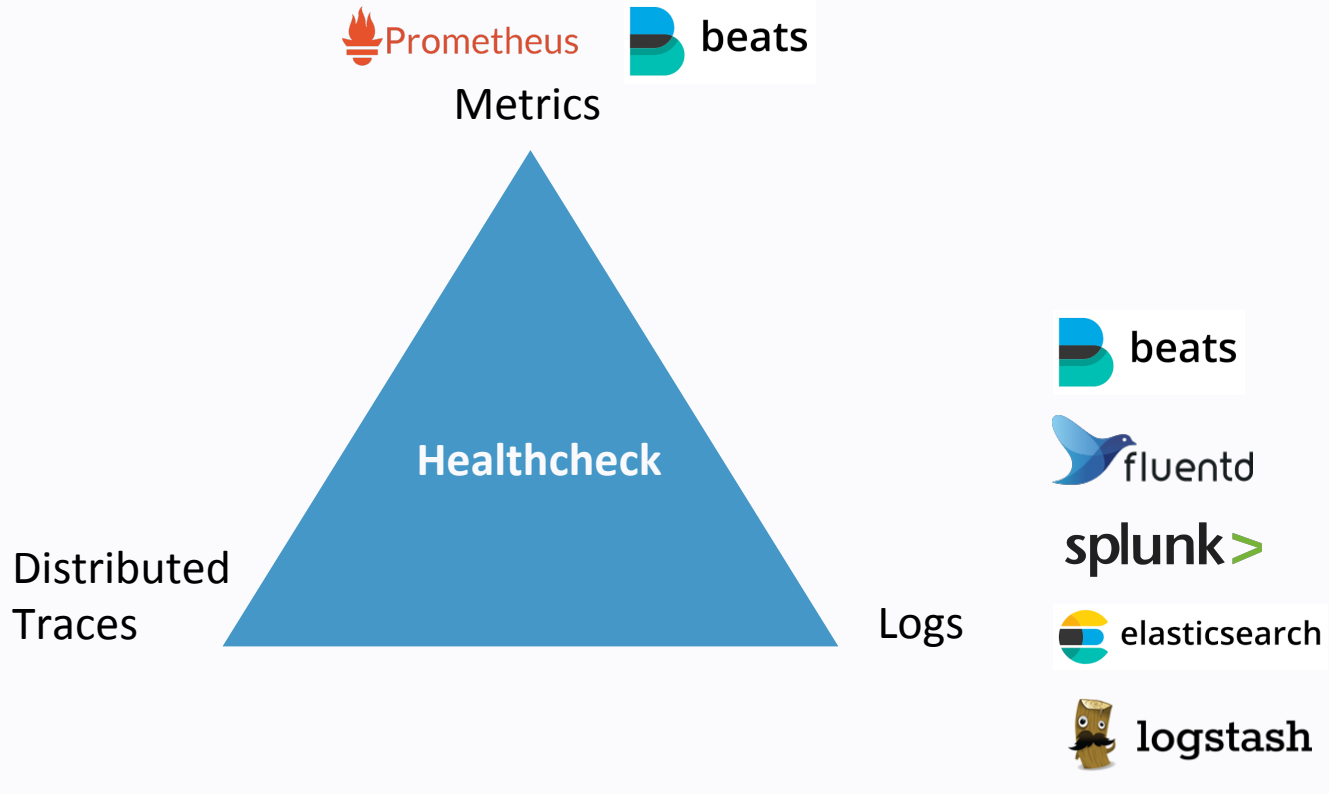
Demo & Beispiele

«signals»

Troubleshooting

Know your containers!

Diagnosability Triangle – Mögliche Technologien



Container nicht gestartet

- Container gestartet? docker ps
- Exit status? docker ps -a
- Logs? docker logs container
- Windows vs. Linux Line-Endings (v.a. Script)?
  - Kann zu „No such file or directory“ führen
- Fehler beim image pull? Image lokal vorhanden?

Dienst nicht erreichbar

- Logs? docker logs container
- Dienst intern erreichbar? docker exec, curl localhost:port
- Dienst von anderem Container erreichbar? docker exec, curl container:port
- Dienst von docker host erreichbar? curl ip:port
- Ports von host gemappt? curl localhost:port
- Logs von ReverseProxy?

Demo & Beispiele

«ide debugging»

Remote Debugging Java WildFly Container

Debug Mode/Transport konfigurieren (z.B. via JAVA_OPTS)

```
-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5005
```

WildFly unterstützt die Konfiguration via Environment-Variable

```
environment:
  - DEBUG=true
  - DEBUG_PORT=5005
```

Port-Forwarding für WildFly Container (docker-compose.yml)

```
ports:
  - 9031:5005
```

IntelliJ Remote Debug auf Port (9031)

Advanced Troubleshooting – Linux tools

Kein curl, wget oder nc?

```
$ exec 8<>/dev/tcp/<IP|HOSTNAME>/<PORT>
$ echo -e ”GET / HTTP/1.1\n\n” >&8
$ cat <&8
```

Kein nslookup oder dig?

```
$ getent hosts <IP|HOSTNAME>
```

Letzte Möglichkeit (debug):

```
$ docker run -it -v /bin:/bin image
```

Java und Docker

Java und Docker

- System mit 32GB Ram, 8 CPUs

Wie gross ist der maximale Java-Heap-Space (-Xmx) wenn nicht explizit konfiguriert?

→ ¼ des physischen RAMs

Was ist nun wenn 5 Java Prozesse laufen?

Java und Docker

32GB Ram *8 CPUs*	**--memory 1024mb**	**--cpus 4**	**--cpuset-cpus 0,2,4,6**
< 1.8.0_131	Falsch (~7.3gib)	Falsch (8)	Richtig
> 1.8.0_131	Falsch (~7.3gib)	Falsch (8)	Richtig
> 1.8.0_131 -XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap	Richtig	Falsch (8)	Richtig
Java 9	Falsch (~7.3gib)	Falsch (8)	Richtig
Java 9 -XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap	Richtig	Falsch (8)	Richtig
>= Java 10	Richtig	Richtig	Richtig

Java und Docker

- Vor Java 10 werden die CGroups von Java nicht korrekt verwendet
- Vor Java 1.8.0 131: -Xmx setzen
- Bis und mit Java 9:
  -XX:+UnlockExperimentalVMOptions
  -XX:+UseCGroupMemoryLimitForHeap

Generell

- JAVA_OPTS konfigurierbar machen
- Java -Xmx und -Xms immer explizit setzen

Demo & Beispiele

«systeminfo»

Security

Grundlagen

- Docker Daemon läuft als root
- Container laufen als root (default)
- Host-System kann gemounted werden: `-v /:/host`

Wer Docker Container starten kann ist faktisch root

Möglichkeiten zur Absicherung

- Container nicht als root starten (wenn möglich)
- Keine Container im `--privileged` mode
- Keine Ports öffnen die nicht gebraucht werden
- Extensions wie SELinux, Seccomp verwenden
- App-Kommunikation innerhalb Docker-Netzwerk
- CPU-Limits, PID-Limits einsetzen um DoS-Attacken zu vermeiden
- Docker Images prüfen (Docker Hub)

Capabilities

- Fähigkeiten eines Containers
- Feingranulare Berechtigungen
- Default bereits in Docker vorhanden (11/38 aktiviert)
- Hinzufügen –cap-add oder löschen –cap-remove möglich
- Überlappend mit Seccomp

Beispiel

CAP_SYS_TIME: Container können die System-Zeit nicht ändern.

CAP_CHOWN: Container können Owner nicht ändern.

Seccomp

- Kernel Features
- Seccomp muss vom Kernel unterstützt werden
- Limitiert die System-Calls
- Überlappend mit Capabilities
- Seccomp Profile definition im JSON-Format
- Spezifikation mit: --security-opt seccomp=default-no-chmod.json

Beispiel

acct: Verhindert dass Container ihre eigene Limitierungen aufheben können. Auch durch CAP_SYS_PACCT sichergestellt.

chmod, **fchmod**, **fchmodat**: Ändern der File-Permissions.

SELinux

- Docker deamon: `--selinux-enabled`
- Host Mounts: `context="system_u:object_r:container_file_t:s0"`
- Container Mounts: `-v volume:mountPoint:[zZ]`
- Docker-Verzeichnis: `system_u:object_r:container_var_lib_t:s0`
- Security Context wiederherstellen: `restorecon -R -v /var/lib/docker/`

Shared Content :z	Alle Container können lesen/schreiben	system_u:object_r:container_file_t:s0 _data
Private Content :Z	Nur aktueller Container kann lesen/schreiben	system_u:object_r:container_file_t:s0:**c683,c813** _data

SELinux – Context manuell Managen

- SELinux Context temporär ändern
  - chcon –t svirt_sandbox_file_t <directory>
- SELinux Context permanent ändern (Regel erstellen)
  - semanage fcontext –a
- SELinux Context wiederherstellen (Anhand der Regeln)
  - restorecon –R -v /directory
- SELinux Context anzeigen
  - ls –laZ <directory>

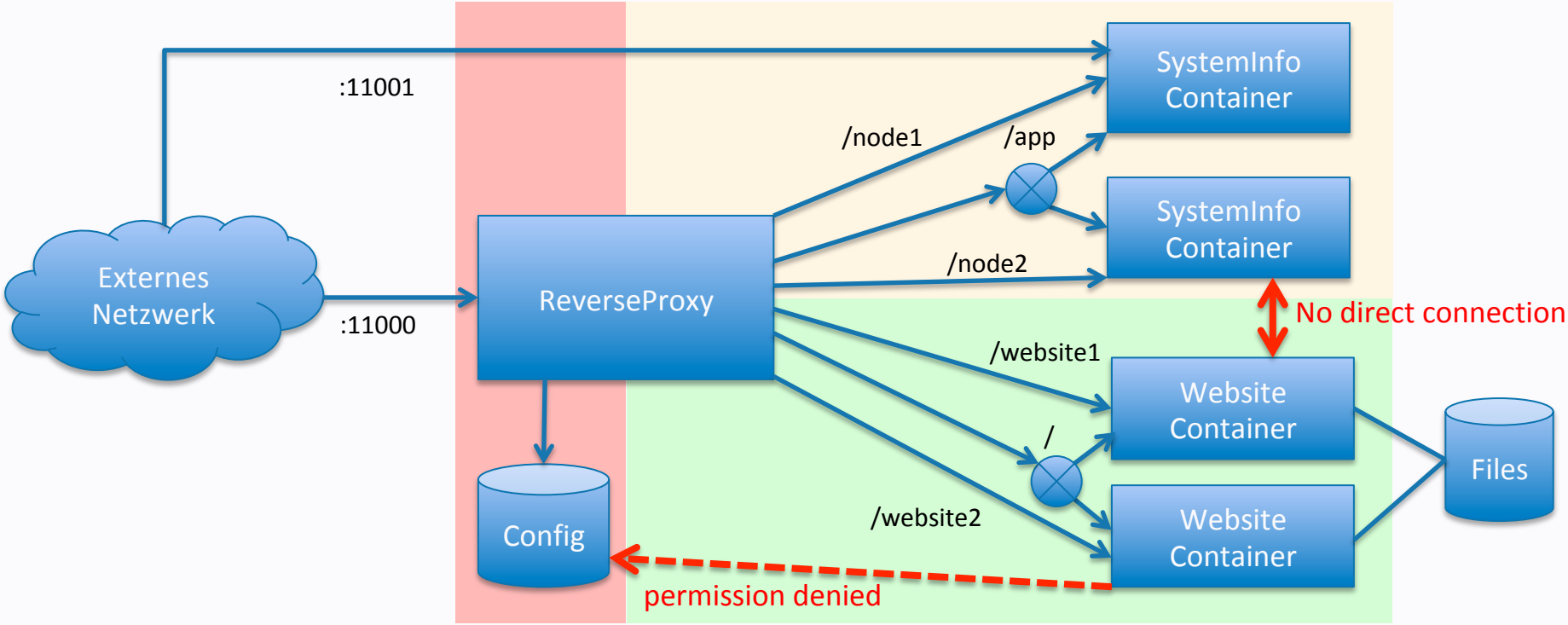
Demo & Beispiele

«secure»

«capabilities»

«seccomp»

Docker Compose – Demo App Secure



Nachdenken bei...

- Images von Docker-Hub
- Mounts in den Container (/, /etc/shadow)
- Mounts des Docker-Sockets (/var/run/docker.sock)
- Container --privileged sein möchte
- Container als root gestartet werden möchte
- Öffnen des Daemon APIs über TCP

Fragen?

Links

- Docker-Internals: http://docker-saigon.github.io/post/Docker-Internals
- Access MobyLinuxVM: https://forums.docker.com/t/how-can-i-ssh-into-the-betas-mobylinuxvm/10991/2
- Best Practices for writing Dockerfiles: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
- Java and Docker, the limitations: https://royvanrijn.com/blog/2018/05/java-and-docker-memory-limits
- Improved Docker Container Integration with Java 10: https://blog.docker.com/2018/04/improved-docker-container-integration-with-java-10/
- 9 Common Dockerfile Mistakes: https://runnable.com/blog/9-common-dockerfile-mistakes
- Design patterns for container-based distributed systems: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45406.pdf
- Docker und Kubernetes Patterns & Anti-Patterns: https://www.doag.org/formes/pubfiles/9953332/2018-NN-Josef_Adersberger-Kubernetes-_und_Docker-Patterns_und_-Antipatterns-Praesentation.pdf
- Openshift Creating Images Guidelines: https://docs.openshift.com/container-platform/3.11/creating_images/guidelines.html
- Docker Security https://github.com/docker/labs/blob/master/security/README.md

Nice to know
Docker on Windows

Access Docker Host on Windows

```
C:\Users\User> docker run --privileged -it \
-v /var/run/docker.sock:/var/run/docker.sock jongallant/ubuntu-docker-client

root@8b58d2fbe186:/# docker run --net=host --ipc=host --uts=host --pid=host –it \
 --security-opt=seccomp=unconfined --privileged --rm -v /:/host alpine /bin/sh

root@8b58d2fbe186:/# chroot /host

/ # ls -l /var/lib/docker/
total 128
…
drwx------  20 root root  4096 Jan  3 13:47 containers
drwx------   3 root root  4096 Jul 18 07:53 image
drwxr-x---   3 root root  4096 Jul 18 07:53 network
drwx------ 518 root root 77824 Jan  3 13:47 overlay2
…
drwx------  16 root root  4096 Jan  3 09:08 volumes
```