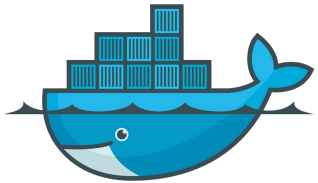


docker Workshop

Von der Installation bis zur Automatisierung

Agenda

neofonie*
Data. Technology. Services.

- Begrüßung / Kaffee
- kurze Vorstellung der Teilnehmer
- Erwartungen / Aufgabenstellungen

- Einführung Docker
- Docker Container
- Docker Images
- Docker Network
- Docker Compose
- Docker Volumes
- Docker Logging

Was ist Docker?

neofonie*
Data. Technology. Services.

- zentrales Produkt von Docker Inc.
  - initial von dotCloud entwickelt, erster GitHub-Commit 01 / 2013
  - 07 / 2014 Kooperation RedHat, MS, IBM, Google Kubernetes
  - 08 / 2014 an cloudControl verkauft
  - 04 / 2015 Investitionen in Höhe von $95 Millionen
  - 10 / 2018 Investitionen in Höhe von $92 Millionen, Venture Round
  - Docker Inc. Marktwert ca. $1 Billion

Was ist Docker?

neofonie*
Data. Technology. Services.

- DevOps-Tool
- Kapselt Anwendungen und deren Abhängigkeiten in sog. Images
- Versteht Instanzen der Images als sog. Container
- Open Source, geschrieben in Go
- keine Virtualisierung / VM
- Cluster Technologie, Docker Swarm “build in”
- Container Deployment wird von allen Cloud Anbietern unterstützt
  (MS Azure, Amazon, Google, Digital Ocean, Rackspace)

Was ist Docker ?

Deployment Schnittstelle

Was ist Docker ?

Deployment Schnittstelle

Basis-Komponenten

neofonie*
Data. Technology. Services.

Docker besteht aus mehreren Komponenten, die wichtigsten Begriffe sind:

- Docker Daemon
- Docker Client / Docker API
- Container
- Image
- Registry

Installation

```
~$ git clone https://github.com/Neofonie/DockerBasics.git
```

In `~/DockerBasics/workshop.md` findet ihr die URLs zu den Install-Ressourcen

Sollte der Docker Daemon nicht starten hilft folgender Check meist weiter

```
wget https://raw.githubusercontent.com/docker/docker/master/contrib/check-config.sh
bash check-config.sh
```

neofonie*
Data. Technology. Services.

Container

Container

neofonie*
Data. Technology. Services.

In diesem Abschnitt

- erstellen und starten wir unsere ersten Container
- loggen wir uns in laufende Container ein
- verschaffen wir uns Überblick
- beenden und löschen wir Container

Hello World!

```
~$ docker run busybox echo “hello world”
hello world
~$
```

Hello World!

```
/workshop:$ docker run busybox echo "hello world"

Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
90e01955edcd: Pull complete
Digest: sha256:2a03a6059f21e150ae84b0973863609494aad70f0a80eaeb64bddd8d92465812
Status: Downloaded newer image for busybox:latest
hello world
```

Hello World!

```
/workshop:$ docker ps

CONTAINER ID    IMAGE    COMMAND    CREATED    STATUS    PORTS    NAMES


/workshop:$ docker ps -a

CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                      PORTS
16920fd4b8fd   busybox   "echo 'hello world'"   10 seconds ago   Exited (0) 10 seconds ago

NAMES
eager_payne
```

unser Container hat sich nach der Ausführung von echo ‘hello world’ beendet. Mit docker ps kann man sich laufende Container oder mit der Option ‘-a’ alle Container auf dem Docker Host anschauen.

Starten eines Containers

Wir starten einen Debian Container und bekommen eine Shell im laufenden Container.

```
~$ docker run -it debian
root@18679d972c23:/# cat /etc/issue
Debian GNU/Linux 9 \n \l
root@18679d972c23:/# exit
exit
~$
```

die Option `-ti` verbindet uns mit stdin des Container Prozesses (-i) und gibt uns ein tty (-t).

Starten eines Containers

neofonie*
Data. Technology. Services.

Dockerfile + (~/asp-repos/neofonie/_DockerBasics/neofonie-helloworld) - VIM

Dockerfile + (~/asp-repos/neofonie/_DockerBasics/neofonie-helloworld) - VIM 121x41

```
FROM alpine:latest

MAINTAINER Dennis Winter (dennis.winter@neofonie.de) | Neofonie GmbH

RUN apk update && \
    apk add apache2 bash

EXPOSE 80

RUN mkdir -p /run/apache2

COPY httpd.conf /etc/apache2/httpd.conf
COPY index.html /var/www/localhost/htdocs/

ENTRYPOINT ["httpd", "-D", "FOREGROUND"]
```

Was passiert beim Start eines Containers ?

Container starten mit einem primären Prozess. Dieser wird im *Dockerfile* bei der Erstellung des Images mit den Anweisungen ENTRYPOINT oder CMD definiert.

Auf die beiden Anweisungen gehen wir später im Detail ein.

Starten eines Containers

Container können auch im Hintergrund “daemon mode” gestartet werden:

```
~$ docker run -p=7891:80 -d httpd
8cadebb85afe9164e67a807f3d09ef1d416b908052b224751a54fb73c544bd4b
~$
```

Docker zeigt lediglich die ID des gestarteten Containers.

Die Option ‘-p’ verbindet den lokalen TCP-Port 7891 mit dem Port 80 auf dem der httpd Prozess im Container lauscht.

Starten eines Containers

startet man den Apache ohne die Option ‘-d’ sieht man die Ausgabe des httpd Prozesses. Der ‘docker run’ Befehl kehrt nicht zurück und man kann mit Ctrl-C den httpd Prozess beenden.

```
~$ docker run -p=7890:80 httpd
AH00558: httpd: Could not reliably determine the server's fully qualified domain name
[...]
[Thu Apr 07 09:48:31.400690 2016] [mpm_prefork:notice] [pid 9] AH00163: Apache/2.4.17
(Unix) configured -- resuming normal operations
[Thu Apr 07 09:48:31.400707 2016] [core:notice] [pid 9] AH00094: Command line: 'httpd
-D FOREGROUND'
```

Container mit Volume starten

Möchte man die Daten in einem laufenden Container persistent speichern und wiederverwenden, dann benutzt man dafür ein Docker Volume.

Eine Variante dafür ist die Verwendung eines lokalen Verzeichnisses auf dem Docker Host.

```
~$ docker run -e MYSQL_ROOT_PASSWORD=geheim -v /data/workshop/mysql:/var/lib/mysql mysql
```

Container mit Volume starten

Die Option ‘-v’ erwartet zwei Argumente, die durch einen Doppelpunkt getrennt werden.

Links steht der lokale Pfad bzw. die Volume Definition und auf der rechten Seite, der Pfad im Container.

```
lokales Filesystem  <- -v /data/workshop/mysql:/var/lib/mysql  ->  Pfad im Container
```

Container mit Volume starten

Falls die Filesystem-Rechte es zulassen, wird das Verzeichnis gegebenenfalls mit der Docker UID und GID angelegt.

```
~$ ls -l /data/workshop

drwxr-xr-x 6        999 docker        4096 Dez   4 11:18 mysql
```

den Überblick behalten

Wie können wir schauen, ob unser dämonisierter Container noch läuft?

```
~$ docker ps
CONTAINER ID    IMAGE         COMMAND         CREATED            STATUS
f1228909bfa6    debian:8      "/bin/bash"     6 minutes ago      Up 6 minutes
~$
```

den Überblick behalten

mit dem Befehl ‘inspect’ erhält man ausführliche Informationen zum Docker Container. Das im Image hinterlegte CMD bzw. ENTRYPOINT Kommando findet man dort wieder.

```
~$ docker inspect 18679d972c23

[
        {
        "Id": "18679d972c236cc4981cedcf40a855e710442f2e33bbbd848a0acbec576d9b3c",
        "Created": "2018-12-03T14:47:34.737200648Z",
        "Path": "bash",
        "Args": [],
        "State": {
        "Status": "exited",
        "Running": false,
```

den Überblick behalten

Möchte man sich in einem Container umschauen, ohne das Startscript auszuführen, dann kann man einen alternativen Einstiegspunkt für den Start des Containers wählen.

Das ist z.B. sehr hilfreich zum debuggen, falls der reguläre Start des Containers nicht funktioniert.

```
~$ docker run -ti --entrypoint /bin/bash httpd
```

Stoppen eines Containers

Container können *graceful* gestoppt oder gekillt werden.

```
~$ docker stop f1228909bfa6
```
Hierdurch senden wir ein TERM Signal an den Container und geben ihm 10 Sekunden (default) Zeit, seinen im ENTRYPOINT laufenden Prozess zu beenden. Anschließend sendet Docker ein KILL -9.

```
~$ docker stop -t 600 f1228909bfa6
```
Wir geben dem Prozess 10 Minuten Zeit sich sauber zu beenden. Danach erfolgt ein Kill -9.

```
~$ docker kill f1228909bfa6
```
Wir können auch direkt ein KILL -9 senden. Damit wird umgehend der Prozess im ENTRYPOINT beendet.

Stoppen eines Containers

Das Stop-Timeout kann in der aktuellen Docker Version nicht global vorgegeben werden.

Im Docker Swarm kann die Einstellung für jeden Service konfiguriert werden:

```
~$ docker service update --stop-grace-period 600 <service_name>
```

Stoppen eines Containers

neofonie*
Data. Technology. Services.

Mit docker-exec können Befehle in einem laufenden Container ausgeführt werden. Das folgende Kommando startet die Bash.

Was passiert bei einem kill 1 ?

```
~$ docker exec -it 8cadebb85afe bash
bash-4.3# ps fax
[...]
bash-4.3# kill 1
```

Stoppen eines Containers

Ein Shell-Wrapper als Entrypoint läßt sich von außen mit “docker stop” nicht regulär beenden. Die Shell ignoriert das SIGTERM wenn ihre PID = 1 ist. Ein “trap” und “exit” hilft hier weiter.

```
#!/bin/bash
shutdown() {
     # kill all pids != 1
     kill $(ps --no-headers -eo pid | egrep -v "[[:space:]]*1$") >/dev/null 2>&1
     # terminate shell (pid 1)
     exit
}
trap shutdown 15
apachectl start
sleep infinity &
wait
```

Container Heaven?

Was geschieht mit gestoppten Containern?

```
~$ docker ps -a
CONTAINER ID    IMAGE                  COMMAND                  CREATED          STATUS
8cadebb85afe    neofonie/hello-world   "httpd -D FOREGROUND"    12 minutes ago   Exited (137) 1 seconds ago
6c82121db0fc    debian:8               "/bin/bash"              33 minutes ago   Exited (0) 23 minutes ago
~$
```

Sie sind weiterhin auf dem lokalen System vorhanden, und können mit `docker start <Container-id>` wieder gestartet werden.

Dabei wird immer das Start Kommando ausgeführt, also CMD oder ENTRYPOINT.

Führt das Kommando zu einem Fehler, kann der Container nicht mehr gestartet werden.

Container Heaven!

neofonie*
Data. Technology. Services.

Der Container muß zuvor gestoppt sein oder man verwendet die Option “-f”.

```
~$ docker rm <Container-ID>
```

Möchte man zusätzlich auch alle anonymen Volumes des Containers löschen , kann man das mit der Option “-v” machen (/var/lib/docker/volumes/<volume id>).

Selbstdefinierte Volumes werden dabei **nicht** gelöscht.

```
~$ docker rm -f -v <Container-ID>
```

Alle Container und alle Volumes löschen:

```
~$ docker rm -f -v $(docker ps -aq)
```

neofonie*
Data. Technology. Services.

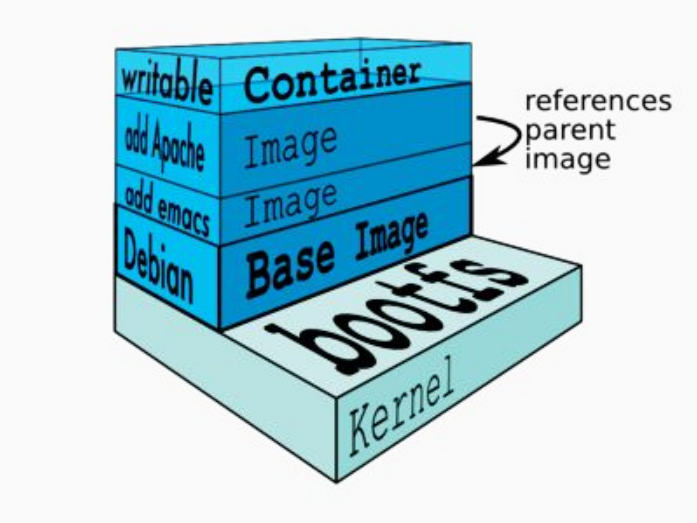
Images

Images

In diesem Abschnitt

- betrachten wir den Aufbau von Images
- erstellen ein Docker Image
- schauen wir uns die Namensvergabe & Namespaces an
- laden wir Images aus Repositories
- pushen wir Images in Repositories
- verwalten wir das lokale Repo

Was sind Images?



- Enthält alle Dateien, die für den Start der Prozesse im Container benötigt werden
- Eine Sammlung von Dateien & Meta-Daten
- Basiert auf übereinanderliegenden Layers
- In jedem Layer können Dateien ergänzt, geändert und gelöscht werden
- Layer können von mehreren Images verwendet werden

Was sind Images?

Image vs. Container:

Docker Container:

- der gestartete Container ist eine Instanz eines Images
- ein laufender Container enthält einen oder mehrere Prozesse
- Container laufen mit einem zusätzlichen Layer on-top eines Images, sozusagen als RW-Kopie
- Container starten mit dem Prinzip von *copy-on-write*
- aus einem laufenden Container kann ein Docker Image erstellt werden

Was sind Images?

neofonie*
Data. Technology. Services.

Image vs. Container:

Docker Image:

- das Docker Image stellt alle Filesystem-Resourcen für den Start eines Containers bereit
- ein Image ist ein sortierter Stapel an Layern im Docker Filesystem (overlay2)
- ein Image ist read-only
- Images werden lokal oder in einer Docker Registry gespeichert
- `docker run <Image-Name>` startet einen Container von einem bestimmten Image
- von einem Image können beliebig viele unabhängige Container gestartet werden

Erstellung von Images

neofonie*
Data. Technology. Services.

`docker build`

- baut aus einem Dockerfile ein Image
- sollte immer die bevorzugte Methode sein

`docker commit <container ID>`

- Erstellt eine Kopie eines Containers inklusive aller dort vorgenommenen Änderungen in Form eines Images
- ist schwer reproduzierbar

Wir erklären beides in Kürze.

Erstellung von Images

Neue Images leiten in aller Regel von so genannten BASE IMAGEs ab. Diese werden bei der Verwendung von `docker build` im Dockerfile mit der Instruktion `FROM` angegeben.

```
FROM ubuntu:latest
```

oder

```
FROM neofonie/hello-world:latest
```

oder

```
FROM registry.mycompany.com:40000/wild-app:beta
```

Erstellung von Images

neofonie*
Data. Technology. Services.

Möchte man mit einem leeren Container beginnen und alle Dateien selbst in das Image kopieren, dann beginnt das Dockerfile mit:

```
FROM scratch
```

Image-Namespaces

Es gibt drei Namespaces

- Basis- oder Root-Namespace

  `ubuntu`

  `-> https://hub.docker.com/u/library`

- User und Organisationen auf hub.docker.com

  `neoworkshop/hello-world`

- Self-Hosted

  `registry.example.com:5000/mein-image`

Root-Namespace

Hier finden sich offizielle Images. Docker Inc. veröffentlicht sie, sie sind aber größtenteils von Dritten gepflegt.

Dazu gehören:

- Mini-Images wie busybox oder alpine (für bspw. Micro-Services)
- Distro-Images wie Debian (zur einfachen Dockerisierung mit allen Möglichkeiten der Distros)
- fertige Komponenten & Dienste wie redis, postgresql etc.

User-Namespace

Der User Namespace enthält die Images für Docker Hub-Users und -Organisationen.

Beispielsweise `neoworkshop/hello-world`

`neoworkshop` ist ein User-Account auf hub.docker.com

`hello-world` ist der Image Name.

Self-Hosted-Namespace

Dieser Namespace enthält die Images in einer eigenen Registry. Teil der Namen ist dann immer die Adresse und ggf. der Port des Registry Servers.

Beispielsweise `registry.neofonie.de:40000/hello-world`

Image Tag

Zusätzlich zum Image-Namen hat jedes Image einen Tag. Das entspricht einer Versions-Nummer bzw. einem Release Tag. Ein Tag wird mit doppeltpunkt getrennt dem Image-Namen angehängt:

Falls kein Tag definiert wird, handelt es sich immer um den Tag “latest”

Beispiele:

```
~$ docker pull debian:9
~$ docker run my_app:bugfix-ticket-4711
```

neofonie*
Data. Technology. Services.

Image Management

Images können auf drei Arten heruntergeladen werden:

- explizit mit `docker pull`
- implizit mit `docker run`, wenn das Image lokal nicht gefunden wird
- implizit mit `docker build`, wenn das Image als BASE IMAGE mit `FROM` im Dockerfile angegeben wurde

Image Management

Images können auf zwei Arten gespeichert werden:

- in einer Registry
- auf dem lokalen System

Mit dem Docker Client können Images von einer Registry sowohl heruntergeladen (`docker pull <Image-Name>`) als auch dorthin übertragen werden (`docker push <Image-Name>`).

Image Management

Ein lokal erstelltes Image können wir mit einer Bezeichnung *taggen* (benennen) und in eine Registry *pushen*.

Die Image-Namen sind lokal frei wählbar.

Erst beim “push” oder “pull” wird der volle Image-Name inkl. Hostname aufgelöst.

```
~$ docker tag my_apache registry.my-company.com/project-x/apache:v1.0
```

```
~$ docker push registry.my-company.com/project-x/apache:v1.0
```

Image Management

Die lokalen Images kann man sich anzeigen lassen

```
~$ docker images

REPOSITORY          TAG         IMAGE ID        CREATED          SIZE
fedora              latest      ddd5c9c1d0f2    3 days ago       204.7 MB
centos              latest      d0e7f81ca65c    3 days ago       196.6 MB
ubuntu              latest      07c86167cdc4    4 days ago       188 MB
redis               latest      4f5f397d4b7c    5 days ago       177.6 MB
postgres            latest      afe2b5e1859b    5 days ago       264.5 MB
alpine              latest      70c557e50ed6    5 days ago       4.798 MB
debian              latest      f50f9524513f    6 days ago       125.1 MB
busybox             latest      3240943c9ea3    2 weeks ago      1.114 MB
```

Image Management

Images können natürlich auch vom lokalen System gelöscht werden.

```
~$ docker rmi <Image-ID>
```

Nicht mehr benutzte Daten alter Images können mit diesem Befehl gelöscht werden.

```
~$ docker system prune
```

Image Management

Docker-Hub kann mit dem Docker-Client durchsucht werden:

```
~$ docker search neoworkshop

NAME                                DESCRIPTION         STARS     OFFICIAL   AUTOMATED
neoworkshop/demo-loadbalancer                           0

neoworkshop/demo-server                                 0

neoworkshop/hello-world                                 0

neoworkshop/demo-auto-lb                                0

neoworkshop/demo-events                                 0
```

Image Builds

neofonie*
Data. Technology. Services.

Build mit einem Commit

Von einem laufenden Container kann ein neues Image erstellt werden, das dessen gesamte Änderungen enthält:

```
~$ docker commit <Container-ID>
123456789abcdef…
```

Build mit einem Dockerfile

Images können anhand einer Reihe von Anweisungen erstellt werden, die in einem sog. *Dockerfile* zusammengetragen werden:

```
~$ docker build -f Pfad/zum/Dockerfile
Pfad/zum/Build-Context
```

Image Builds

Build mit einem Commit

Von einem laufenden Container kann ein neues Image erstellt werden, das dessen gesamte Änderungen enthält:

```
~$ docker commit <Container-ID>
123456789abcdef…
```

So ist der Bau des Images nur schwer zu reproduzieren!

Build mit einem Dockerfile

Images können anhand einer Reihe von Anweisungen erstellt werden, die in einem sog. *Dockerfile* zusammengetragen werden:

```
~$ docker build -f Pfad/zum/Dockerfile
Pfad/zum/Build-Context
```

neofonie*
Data. Technology. Services.

Dockerfile

Dockerfile

In diesem Abschnitt

- finden wir heraus, was ein Dockerfile ist
- schreiben wir ein erstes Dockerfile
- lernen wir ENTRYPOINT und CMD kennen
- bauen wir ein Image
- optimieren wir ein Dockerfile

Image Builds

```
FROM alpine:latest

MAINTAINER Dennis Winter (dennis.winter@neofonie.de) | Neofonie GmbH

RUN apk update && \
    apk add apache2 bash

EXPOSE 80

RUN mkdir -p /run/apache2

COPY httpd.conf /etc/apache2/httpd.conf
COPY index.html /var/www/localhost/htdocs/

ENTRYPOINT ["httpd", "-D", "FOREGROUND"]
```

Dockerfile in a nutshell

Ein Dockerfile ist ein Build-Rezept.
Es enthält eine Serie von Anweisungen die Docker mitteilen, wie ein Image gebaut werden soll.

Image Builds

```
~$ mkdir figlet1
~$ cd figlet1
figlet1 $ vi Dockerfile
```

Das erste Dockerfile

Schreiben wir unser erstes Dockerfile. Es sollte hierzu zunächst in einem eigenen, leeren Verzeichnis sein.

Image Builds

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y figlet
```

Das erste Dockerfile

Wir schreiben diese Zeilen ins Dockerfile.

FROM definiert das Base Image für unseren Build.

RUN führt Befehle während des Builds aus und muss daher non-interactive sein.

Image Builds

```
figlet1 $ docker build -t figlet1 .
```

Das erste Dockerfile

Auf diese Weise bauen wir das Image.

- `-t` tagged das Image
- `.` definiert den Build-Context

Image Builds

```
figlet1 $ docker build -t figlet1 .
Sending build context to Docker daemon 2.048 kB
Step 1 : FROM ubuntu
 ---> c9ea60d0b905
Step 2 : RUN apt-get update
 ---> Running in 175beb205b1e
(... OUTPUT DES RUN-COMMANDS ...)
 ---> 74729b434e64
Removing intermediate container 175beb205b1e
Step 3 : RUN apt-get install figlet
 ---> Running in 7a76fc5d73f1
(... OUTPUT DES RUN-COMMANDS ...)
 ---> 45457b13fb57
Removing intermediate container 7a76fc5d73f1
Successfully built 45457b13fb57
```

Das erste Dockerfile

- Jeder Step stellt ein neues Image / einen Layer da.
- Die Steps werden in einem laufenden Container ausgeführt
- vor jedem Schritt schaut Docker, ob ein Image für diese Build-Sequenz schon existiert
- komplette Rebuilds können mit `--no-cache` und `--force-pull=true` erzwungen werden

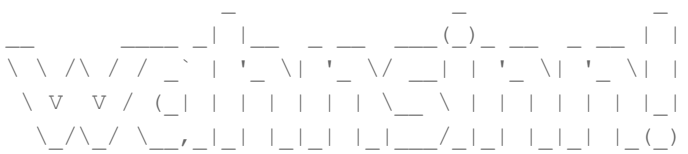
Image Builds

```
~$ docker run figlet1 figlet wahnsinn!
                 _                 _                 _
__      ____ _| |__  _ __  ___(_)_ __  _ __ | |
\ \ /\ / / _` | '_ \| '_ \/ __| | '_ \| '_ \| |
 \ V  V / (_| | | | | | | \__ \ | | | | | | |_|
  \_/\_/ \__,_|_| |_|_| |_|___/_|_| |_|_| |_(_)
```

Erfolg!

ENTRYPOINT & CMD

```
~$ docker run figlet1 figlet wahnsinn!
```



Was läuft da?

- jedes Image hat einen sog. ENTRYPOINT
- CMD funktioniert ähnlich, wird aber primär zur Definition von Default-Values verwendet

ENTRYPOINT & CMD

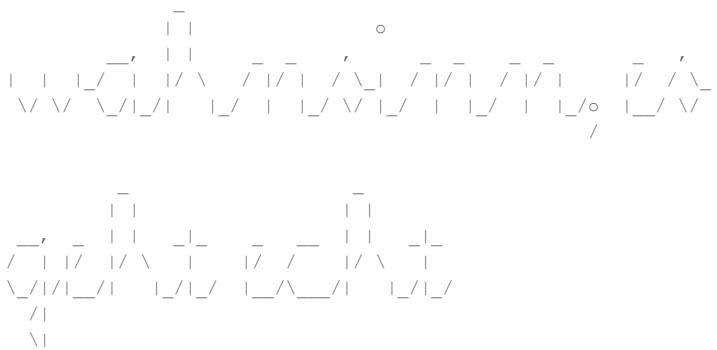
```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y figlet
ENTRYPOINT [“figlet”, “-f”, “script”]
```

```
figlet1 $ docker build -t figlet1 .
```

- ENTRYPOINT and CMD werden im JSON-Format angegeben, andernfalls wird dem String `sh -c` vorangestellt

ENTRYPOINT & CMD

```
~$ docker run figlet1 wahnsinn, es
geht echt
```



Was läuft da?

Hier startet der Container nun mit dem primären Prozess:

```
figlet -f script wahnsinn, es geht
echt
```

Das docker run Kommando interpretiert alles nach dem Image-Namen als Argumente für den ENTRYPOINT. Etwa wie

```
~$ docker run figlet1 $@
```

Dockerfile Referenz

`FROM ubuntu:14.04`	Das Base Image, auf das der Build aufbaut. Auf https://hub.docker.com finden sich die Images, auf GitHub häufig die zugehörigen Projekte.
`MAINTAINER Firma X`	Informationen zum Ersteller. Optional.
`RUN` *`Befehl`*	Führt den anschließenden Befehl im jeweils aktuellen Container des Builds aus. Achtung: hierdurch werden weder Prozess-Zustände gespeichert, noch können auf diese Weise daemons gestartet werden.
`EXPOSE 8080`	Definiert, welche Ports Container dieses Images publishen werden. Per default sind diese Ports nicht extern erreichbar.
`ADD assets/my.conf /etc/my.conf` `COPY assets/mybin /usr/local/bin/`	Instruktionen, um Files aus dem Build-Context in das Image zu kopieren. ADD ist etwas mächtiger, und kann auch mit URLs und Archiven umgehen.

Dockerfile Referenz

`VOLUME /pfad/im/Image`	Erstellt einen Mount Point im definierten Pfad. VOLUMES werden bei docker commit nicht gesichert, und können unterhalb von Containern geteilt werden, sogar VOLUMES gestoppter Container.
`WORKDIR /opt/webapp`	Setzt das Arbeitsverzeichnis für die folgenden Instruktionen.
`ENV WEBAPP_PORT 8080`	Setzt in Containern des zu bauenden Images eine Environment-Variable.
`USER nginx`	Setzt den Usernamen (oder UID), der beim Start des Containers verwendet werden soll.

Docker Images, Tips und Tricks

Daten können nur innerhalb eines Layers (Build-Step) gelöscht werden.
häufig geänderte Konfigurationsdateien ans Ende stellen und Kommandos, die viel Zeit brauchen an den Anfang.

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y figlet
ADD etc/app.config /opt/webapp/
ADD application/huge.zip
/opt/webapp/
ENTRYPOINT [“figlet”, “-f”,
“script”]
```

```
FROM ubuntu
RUN apt-get update && \
    apt-get install -y figlet && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/* /tmp/*
/var/tmp/*

ADD application/huge.zip /opt/webapp/
ENTRYPOINT [“figlet”, “-f”, “script”]
ADD etc/app.config /opt/webapp/
```

Docker Images, Tips und Tricks

neofonie*
Data. Technology. Services.

Mit “docker history” sieht man wie ein Image aus einem Dockerfile erstellt wurde.

~$ docker history my_apache

IMAGE	CREATED	CREATED BY	SIZE	COMMENT
ca623445f673	About an hour ago	/bin/sh -c apt-get -qq update && apt-get...		22.7MB
2a51bb06dc8b	2 weeks ago	/bin/sh -c #(nop) CMD ["httpd-foreground"]	0B	
<missing>	2 weeks ago	/bin/sh -c #(nop) EXPOSE 80/tcp	0B	
<missing>	2 weeks ago	/bin/sh -c #(nop) COPY file:761e313354b918b6...	133B	
<missing>	2 weeks ago	/bin/sh -c set -eux; savedAptMark="$(apt-m...	43.1MB	
<missing>	2 weeks ago	/bin/sh -c #(nop) ENV APACHE_DIST_URLS=http...	0B	

Docker Images, Tips und Tricks

neofonie*
Data. Technology. Services.

Der Build-Context, also alle Dateien aus dem Build-Verzeichnis, werden an den Docker-Daemon geschickt.

Es ist daher keine gute Idee einen Dockerfile direkt im Home-Verzeichnis anzulegen und dort das Image zu bauen.

```
/data: $ cd ~

~$ docker docker build -t my_image ./
```

Dieser Aufruf des “docker build” Kommandos würde das gesamte Home-Verzeichnis an den Docker-Daemon (Docker API) senden.

Docker Images, Tips und Tricks

neofonie*
Data. Technology. Services.

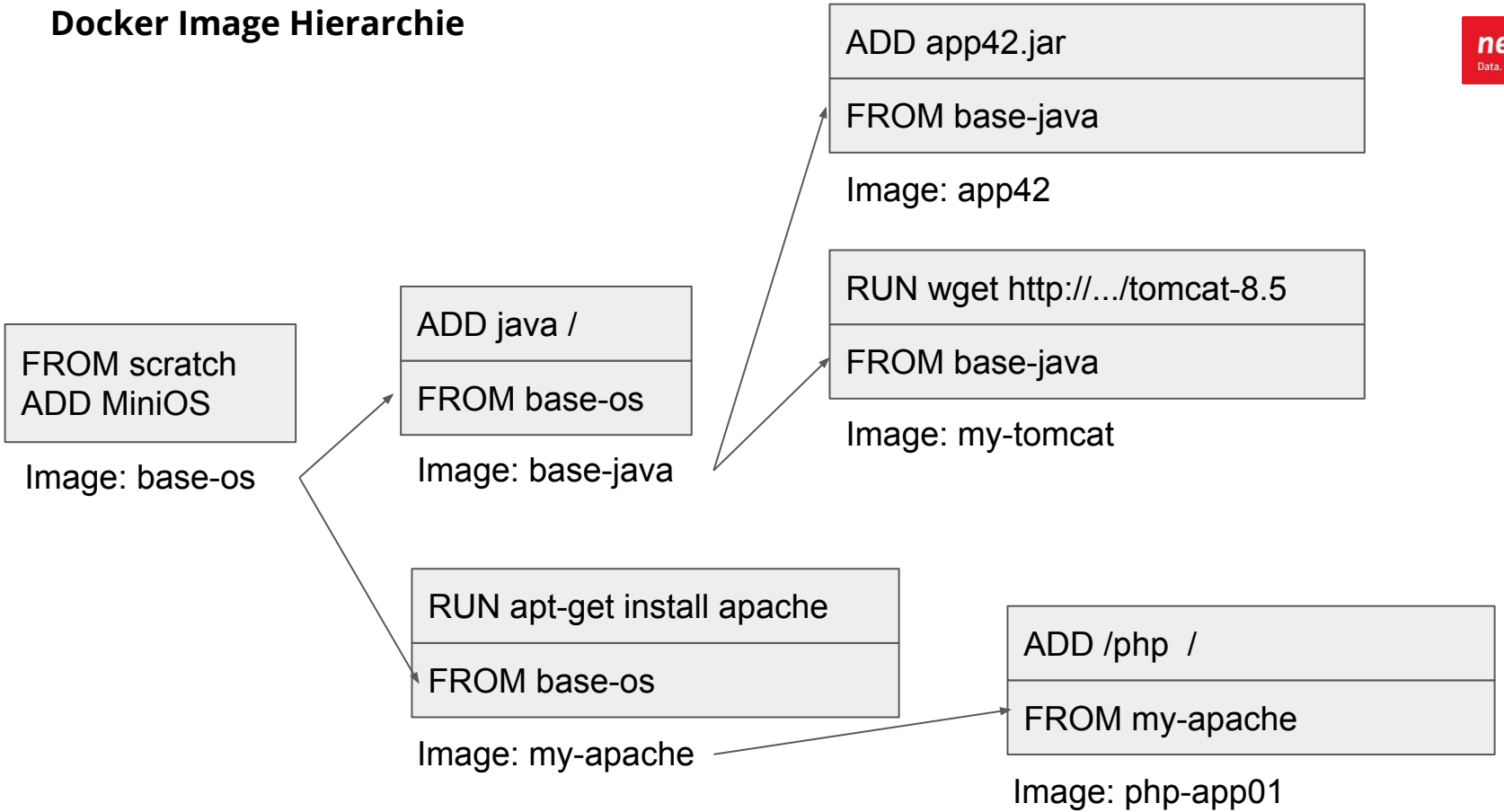
Mit “docker inspect” kann man sich die Metadaten zu einem Image anschauen.

~$ docker inspect my_apache:latest

...

```
"Layers": [
      "sha256:ef68f6734aa485edf13a8509fe60e4272428deaf63f446a441b79d47fc5d17d3",
      "sha256:143e72edeb6dcc7bd2c4d087650b48fda940e5f87c6e8c4e42232ff1303bb5bc",
```

Docker Image Hierarchie



Mittagspause!

```
~$ docker stop docker-workshop
```

Docker Network

neofonie*
Data. Technology. Services.

Kommunikation zwischen Containern
-
Docker Network

Docker Network

neofonie*
Data. Technology. Services.

In diesem Abschnitt

- konfigurieren das Docker Netzwerk
- verbinden wir Container untereinander

Docker Network

neofonie*
Data. Technology. Services.

- Docker Container übernehmen per default die Netzwerk-Konfiguration des Host-Systems.
  - DNS Konfiguration
  - Verbindung ins Internet
- Container bekommen ihre IP Adresse(n) dynamisch zugewiesen
- Netzwerkverbindungen zwischen Containern müssen explizit konfiguriert werden

Docker Network

Nach einer erfolgreichen Docker Installation sieht man ein Netzwerk-Interface mit dem Namen docker0.

Alle Container werden per default in diesem Netzwerk gestartet und befinden sich somit in einem IP Netz.

```
~$ ip a

docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
default
     link/ether 02:42:10:a0:e1:2d brd ff:ff:ff:ff:ff:ff
     inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
```

Docker Network

Das zugehörige Docker Netzwerk ist das “bridge” Netzwerk

```
~$ docker network ls

NETWORK ID          NAME                DRIVER              SCOPE
d754a38f74f9        bridge              bridge              local
228579c79ee5        host                host                local
a4820ee4ec81        none                null                local
```

Docker Network

Startet man zwei Container sieht man die IP Adressen im “bridge” Netzwerk:

```
~$ docker run -d httpd && docker run -d httpd

~$ docker network inspect bridge
      "Containers": {
            "00dd49cb3c911f5bb53b5851da43a5860620931e6eb7fee1fce3e2e2241c83b3": {
            "Name": "focused_saha",
            "EndpointID": "22c6019f24bd39d4836806286eb73e1cccb7363589eeab5497ad1aa0c0208a4c",
            "MacAddress": "02:42:ac:11:00:02",
            "IPv4Address": "172.17.0.2/16",
            "IPv6Address": ""
            },
            "36e92c78522698881d509ea89a0bed8613f3a91b726b4dc2b4f603bd2bb0c1df": {
            "Name": "suspicious_mayer",
            "EndpointID": "9e7e7a81827b0e657d46c8e453871d3bfb94173b30bfdfecd493a173832034f0",
            "MacAddress": "02:42:ac:11:00:03",
            "IPv4Address": "172.17.0.3/16",
            "IPv6Address": ""
```

Docker Network

Die Container IP Adressen sind auf dem Host erreichbar.

```
~$ wget -O/dev/null -S "http://172.17.0.2/"

--2018-12-05 15:17:14--  http://172.17.0.2/
Verbindungsaufbau zu 172.17.0.2:80 … verbunden.
HTTP-Anforderung gesendet, auf Antwort wird gewartet …
  HTTP/1.1 200 OK
  Date: Wed, 05 Dec 2018 14:17:14 GMT
  Server: Apache/2.4.37 (Unix)
...
```

Docker Network

neofonie*
Data. Technology. Services.

Möchte man mehrere Container im Netzwerk verbinden, dann legt man dafür explizit ein Docker Netzwerk an.

Zusätzlich können die Container dann über ihren Namen angesprochen werden

```
~$ docker network create workshop

~$ docker run -d --name apache01 --network workshop httpd
~$ docker run -d --name apache02 --network workshop httpd
```

Docker Network

Die Container können nun innerhalb des “workshop” Netzwerk auch per Namen angesprochen werden:

Im Container apache01 wird der Host bzw. Service-Name apache02 per DNS aufgelöst.

```
~$ docker exec -ti apache01 bash

root@8e46f3d5600f:/usr/local/apache2# getent hosts apache02
172.18.0.3      apache02
```

Docker IPv4 Adress Bereiche

Docker verwendet per default das IPv4 Netz 172.17.0.0/24 für das docker0 Interface.
Alle weiteren Docker Netzwerke werden hochgezählt. (172.18.0.0/24 usw.)

Die Einstellung kann über die Docker Daemon Konfiguration geändert werden (--bip) . Auf einem systemd-System geschieht das in der Datei:

```
~$ vi /etc/systemd/system/docker.service

[Service]
ExecStart=/usr/bin/dockerd -H tcp://0.0.0.0:2376 -H unix:///var/run/docker.sock
--bip=10.254.0.1/16

~$ systemctl daemon-reload
~$ service docker restart
```

Docker IPv4 Adress Bereiche

Alternativ können die Docker Daemon Einstellungen auch über die Datei ‘daemon.json’ erfolgen.

```
~$ vi /etc/docker/daemon.json

{
  "bip": "192.168.1.5/24",
  "dns": ["10.20.1.2","10.20.1.3"]
}

~$ service docker restart
```

Docker-Compose

neofonie*
Data. Technology. Services.

Sobald mehr als ein Docker Container gestartet werden muss verwendet man das Tool “docker-compose”.

Installation des docker-compose Tools:

```
~$ sudo curl -L
"https://github.com/docker/compose/releases/download/1.23.1/docker-compose-$(uname
-s)-$(uname -m)" -o /usr/local/bin/docker-compose

~$ chmod +x /usr/local/bin/docker-compose
```

Docker-Compose

Was ist docker-compose:

- ein Orchestration Tool für Docker
- basiert auf einer zentralen Definitionsdatei (docker-compose.yml)
- verwaltet ein Bundle aus mehreren Services (Docker Container)
- steuert den Docker Daemon
  - startet Container
  - legt Docker Netzwerke an
  - Image push, pull und build
- kennt auch Docker CLI Befehle wie z.B. docker logs -> docker-compose logs

Docker-Compose

```
~$ cat docker-compose.yml

version: '3'

services:

  apache01:
     image: httpd

  apache02:
     image: httpd
```

docker-compose verwendet die Konfigurationsdatei **docker-compose.yml**

Das Format der Datei unterscheidet sich ein wenig zwischen Version 1 und **version 2**.

Die Version 3 findet im Docker Swarm Verwendung.

Docker-Compose

```
~$ docker-compose up -d

Creating network "docker-compose_default" with the
default driver
Creating docker-compose_apache01_1_6ba35e69e6a9 ...
done
Creating docker-compose_apache02_1_f7acecd58555 ...
done
```

Mit der Option ‘up’ werden alle Services im docker-compose.yml File gestartet.

Das Docker Netzwerk ist nicht definiert. Compose startet daher ein “default” Netzwerk.

Docker-Compose

Mit docker-compose kann man sich auch die Container-Logs anzeigen lassen

```
~$ docker-compose logs

Attaching to docker-compose_apache01_1_7803ff1c9166, docker-compose_apache02_1_65ca6e7d8d08
apache01_1_7803ff1c9166 |  AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using
172.21.0.3. Set the 'ServerName' directive globally to suppress this message
apache01_1_7803ff1c9166 |  AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using
172.21.0.3. Set the 'ServerName' directive globally to suppress this message
apache01_1_7803ff1c9166 |  [Wed Dec 05 15:15:58.845863 2018] [mpm_event:notice] [pid 1:tid 140122119263424] AH00489:
Apache/2.4.37 (Unix) configured -- resuming normal operations
apache01_1_7803ff1c9166 |  [Wed Dec 05 15:15:58.845965 2018] [core:notice] [pid 1:tid 140122119263424] AH00094: Command line:
'httpd -D FOREGROUND'
apache02_1_65ca6e7d8d08 |  AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using
172.21.0.2. Set the 'ServerName' directive globally to suppress this message
apache02_1_65ca6e7d8d08 |  AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using
172.21.0.2. Set the 'ServerName' directive globally to suppress this message
apache02_1_65ca6e7d8d08 |  [Wed Dec 05 15:15:59.977165 2018] [mpm_event:notice] [pid 1:tid 140012087555264] AH00489:
Apache/2.4.37 (Unix) configured -- resuming normal operations
apache02_1_65ca6e7d8d08 |  [Wed Dec 05 15:15:59.977331 2018] [core:notice] [pid 1:tid 140012087555264] AH00094: Command line:
'httpd -D FOREGROUND'
```

Docker-Compose

Der Aufruf von docker-compose erzeugt nun zusätzlich ein lokales Netzwerk. Alle Container in diesem Netzwerk können sich per Container-Namen erreichen.

```
~$ docker network ls

NETWORK ID          NAME                      DRIVER              SCOPE
d754a38f74f9        bridge                    bridge              local
e7afd41b70ec        docker-compose_default    bridge              local
228579c79ee5        host                      host                local
a4820ee4ec81        none                      null                local
```

Docker-Compose

Docker build-in DNS: Seit der Docker Version 1.10 findet man dafür den Eintrag des internen DNS Servers in der /etc/resolv.conf.

```
/data/workshop: docker network create workshop

8e5f68a3bb12540e1f29378f790ac557fbe6b2185090fd8e995c883c425eb822

/data/workshop: docker run -ti --network workshop debian

root@b7508e7d6004:/# cat /etc/resolv.conf
search neofonie.de
nameserver 127.0.0.11
options ndots:0
```

Docker-Compose, Loadbalancer

Mit docker-compose kann man die Anzahl der gestarteten Instanzen dynamisch anpassen. Wir erweitern dafür unser Beispiel um einen “automatisierten” Loadbalancer.

Docker-Compose, Loadbalancer

```
~$ cd scale
~$ cat docker-compose.yml

version: '3'
services:

  apache:
     image: httpd
     labels:
     - "traefik.enable=true"
     - "traefik.backend=apache"
     - "traefik.port=80"
     - "traefik.frontend.rule=Host: apache"
```

Unser Service **apache** bekommt nun keinen Container-Namen. Diese werden beim Aufruf von docker-compose scale einfach numerisch hochgezählt.

Damit unsere Webserver Instanzen alle über einen Loadbalancer erreichbar sind, bekommt jeder Container ein paar Labels.

Der Loadbalancer ‘traefik’ erkennt an der Label-Konfiguration, wie der Service in das Loadbalancing integriert werden soll.

Docker-Compose, Loadbalancer

```
  lb:
     image: traefik
     command: --api --docker
     ports:
       - "8080:80"
       - "8042:8080"
     volumes:
       - /var/run/docker.sock:/var/run/docker.sock

networks:
  default:
     ipam:
     config:
       - subnet:  10.253.0.0/16
```

Der Loadbalancer Service bekommt den Namen **‘lb’** und verwendet das Docker Image **‘traefik’**.

Über den Port 8080 sind unsere apache-Service-Instanzen erreichbar.

Der Port 8042 zeigt uns das UI des Loadbalancers.

Abweichend von der default Konfiguration wird eine IP Range vorgegeben.

Docker-Compose, Loadbalancer

Jetzt starten wir einige Webserver und einen Loadbalancer:

```
~$ docker-compose up -d

Creating network "scale_default" with the default driver
Creating scale_lb_1_d69a0f2dc314   ... done
Creating scale_apache_1_ff743a3e76f0 ... done
```

Docker-Compose, Loadbalancer

Docker Compose generiert die Container-Namen nach dem Schema:

```
< Compose Projekt >_< Service-Name >_< Instanz Nr >_< id >
```

Der Projektname ist per default der Name des Arbeitsverzeichnis, in dem sich der docker-compose.yml File befindet.

Docker-Compose, Loadbalancer

Nun können wir eine Hand voll Webserver starten:

```
~$ docker-compose up -d --scale apache=5

scale_lb_1_851da9d7a1ab is up-to-date
Starting scale_apache_1_5feb9fe2766e ... done
Creating scale_apache_2_854a309a0d9e ... done
Creating scale_apache_3_45b5591ae795 ... done
Creating scale_apache_4_7419e61c7687 ... done
Creating scale_apache_5_7136d645a2ce ... done
```

Docker-Compose, Loadbalancer

Mit dem Host Header ‘apache’ im HTTP Request erreicht man die Apache Instanzen auf dem Port 8080 des Loadbalancers.

```
~$ wget -O/dev/null -S --header "Host: apache" "http://127.0.0.1:8080/"

--2018-12-06 15:12:23--  http://127.0.0.1:8080/
Verbindungsaufbau zu 127.0.0.1:8080 … verbunden.
HTTP-Anforderung gesendet, auf Antwort wird gewartet …
  HTTP/1.1 200 OK
  Accept-Ranges: bytes
  Content-Length: 45
  Content-Type: text/html
...
```

Docker Volumes, Layer Filesystem

neofonie*
Data. Technology. Services.

Ohne weiter Konfiguration eines Docker Volumes werden Daten im Layer-Filesystem des Containers geschrieben.

Was bedeutet das und was kann man damit machen ?

- alle Daten des Layers verschwinden beim Löschen des Containers
- schreibbarer Layer zusätzlich zu den readonly Image-Layern des Containers
- nicht effizient, das Überschreiben einer Datei erzeugt eine Kopie

Docker Volumes, Layer Filesystem

neofonie*
Data. Technology. Services.

Das Layer Filesystem des Containers:

- sollte nicht für Datenbanken oder häufige Schreibvorgänge verwendet werden
- ist für Konfigurationen und Daten, die einen Neustart “überleben” sollen geeignet
- verschiedene Filesystem-Treiber stehen zur Verfügung (aufs, overlay, devicemapper)
- wird beim Erstellen von Docker Images benutzt

Docker Volumes, persistenter Speicher

neofonie*
Data. Technology. Services.

Docker Volumes verwendet man, um Daten ausserhalb des Containers zu schreiben.

Was ist ein Docker Volume ?

- stellt persistenten, vom Container unabhängigen, Speicher zur Verfügung
- umgeht das Layer-Filesystem
- wird im Container als Verzeichnis bereitgestellt
- es gibt verschiedene Docker Volume Arten

Docker Volumes, persistenter Speicher

neofonie*
Data. Technology. Services.

Welche Docker Volumes gibt es ?

- vom Docker Daemon verwaltet:
  - Anonymes Volume
  - Named Volume
- Verzeichnis auf dem Docker Host
  - Host Volume
- Docker Volume Treiber
  - Docker Plugin: z.B. REX-Ray
  - zusätzliche Storage Software erforderlich

Docker Volumes

neofonie*
Data. Technology. Services.

Was ist ein anonymes Docker Volume ?

- dynamisch bereitgestelltes Docker Volume
- wird vom Docker Daemon unter /var/lib/docker/volumes angelegt
- erhält eine Volume ID, die dem Container fest zugeordnet ist
    - kann daher nicht wiederverwendet werden

Docker Volumes

neofonie*
Data. Technology. Services.

Was ist ein anonymes Docker Volume ?

- ist ein reguläres Verzeichnis im Filesystem des Docker Host
- ist Standard für Images wie mysql oder wordpress
- kann bereits im Image definiert sein, VOLUME ["/var/lib/mysql"]

Docker Volumes

neofonie*
Data. Technology. Services.

Was ist ein anonymes Docker Volume ?

- wird nicht automatisch mit dem Container gelöscht
- aufräumen mit `docker system prune` erforderlich

Docker Volumes

neofonie*
Data. Technology. Services.

Das MySQL Docker Image verwendet ein Volume für das Daten-Verzeichnis:

```
~$ docker run -d -e MYSQL_ROOT_PASSWORD=geheim mysql
27b435d1f873d1bfb70b108f856b481d7377af7ccad6c9ed5e7a90f37a03d82c
```

Man findet das Volume unter /var/lib/docker/volumes wieder wenn man z.B. nach dem mysql ibdata file sucht.

```
~$ sudo find /var/lib/docker/volumes -name "ibdata*"

/var/lib/docker/volumes/0a8a42d44e58d724537c8b96aa1f64c0d9d4815a1f2949f827b5201145793a
d1/_data/ibdata1
```

Docker Volumes

neofonie*
Data. Technology. Services.

Löscht man den Container, so bleibt das Volume erhalten. Es kann aber nicht mehr wiederverwendet werden.

```
~$ docker rm -f 27b435d1f873d1b

~$ sudo find /var/lib/docker/volumes -name "ibdata*"

/var/lib/docker/volumes/0a8a42d44e58d724537c8b96aa1f64c0d9d4815a1f2949f827b5201145793ad1/_data/ibdata1
```

Docker Volumes

neofonie*
Data. Technology. Services.

Nach dem “Aufräumen” ist das Volume verschwunden:

```
~$ docker volume prune

WARNING! This will remove all local volumes not used by at least one container.
Are you sure you want to continue? [y/N] y

Deleted Volumes:
0a8a42d44e58d724537c8b96aa1f64c0d9d4815a1f2949f827b5201145793ad1
```

Docker Volumes

neofonie*
Data. Technology. Services.

Was ist ein Named Volume ?

- verhält sich genauso wie ein anonymes Volume
- bekommt einen Namen statt einer ID
- ist wiederverwendbar
- wird mit `docker volume prune` gelöscht, wenn kein Container mehr damit verknüpft ist

Docker Volumes

neofonie*
Data. Technology. Services.

Wir starten unsere Datenbank mit einem Named Volume

```
~$ docker volume create mysql
mysql

~$ docker run -d -e MYSQL_ROOT_PASSWORD=geheim -v mysql:/var/lib/mysql mysql
9a8fd2fd24bd5a00d06c476386bdb8d579b03be51e25f54ac853af080cc8c5a8

~$ sudo find /var/lib/docker/volumes -name "ibdata*"
/var/lib/docker/volumes/mysql/_data/ibdata1
```

Docker Volumes

neofonie*
Data. Technology. Services.

Nach dem Löschen des Containers bleibt das Volume erhalten und kann wiederverwendet werden.

```
~$ docker rm -f 9a8fd2fd24bd5a
9a8fd2fd24bd5a

~$ sudo find /var/lib/docker/volumes -name "ibdata*"
/var/lib/docker/volumes/mysql/_data/ibdata1

~$ docker run -d -e MYSQL_ROOT_PASSWORD=geheim -v mysql:/var/lib/mysql mysql
889b0736490035501dea7d595f80efd06f9c2b7b5030d28dd2cde10858f16387
```

Docker Volumes

neofonie*
Data. Technology. Services.

Was ist ein Host Volume ?

- ein Verzeichnis auf dem Docker Host
- wird vom Docker Daemon angelegt, falls es nicht vorhanden ist
- kann nicht mit `docker volume prune` gelöscht werden

Docker Volumes

neofonie*
Data. Technology. Services.

Wir starten unsere Datenbank mit einem Host Volume

```
~$ docker run -d -e MYSQL_ROOT_PASSWORD=geheim -v /data/mysql:/var/lib/mysql mysql
9a8fd2fd24bd5a00d06c476386bdb8d579b03be51e25f54ac853af080cc8c5a8

~$ sudo find /data/mysql -name "ibdata*"
/data/mysql/_data/ibdata1
```

Docker Volumes

neofonie*
Data. Technology. Services.

Wir löschen den Container und schauen was passiert:

```
~$ docker rm -f 9a8fd2fd24bd5a
9a8fd2fd24bd5a

~$ docker volume prune
WARNING! This will remove all local volumes not used by at least one container.
Are you sure you want to continue? [y/N] y

~$ sudo find /data/mysql -name "ibdata*"
/data/mysql/_data/ibdata1
```

Docker Volumes

neofonie*
Data. Technology. Services.

Was sind Docker Volume Treiber ?

- Storageanbindung über Docker Volume Plugins oder externe Software
- direkte Anbindung einer Storage Lösung
- viele verschiedene Treiber vorhanden
- EMC REX-Ray
- NFS, Samba usw.
- https://docs.docker.com/engine/extend/plugins_volume

Docker Logging

neofonie*
Data. Technology. Services.

Logging Informationen aus einem Container:

- sollten über stdout geschrieben werden.
- nicht im Layer-Filesystem des Containers geschrieben werden
- große Logdateien sollten in ein Docker Volume geschrieben werden

Docker Logging

neofonie*
Data. Technology. Services.

Container Logging:

- stdout und stderr werden vom Docker Daemon per default in eine Datei geschrieben
  - in /var/lib/docker/containers/<ID>/<ID>-json.log

- es gibt eine Reihe alternativer Logging Treiber
  - none
  - syslog
  - fluentd

Docker Logging

neofonie*
Data. Technology. Services.

Die Logging-Konfiguration kann zentral über die Datei /etc/docker/daemon.json erfolgen.

Syslog für das Container Logging verwenden:

```
~$ vi /etc/docker/daemon.json

{
  "log-driver": "syslog"
}

~$ sudo service docker restart
```

Docker Logging

neofonie*
Data. Technology. Services.

Syslog für das Container Logging verwenden:

```
~$ docker run -ti debian

root@a06a9ef7ff89:/# echo moin

~$ sudo tail /var/log/syslog
...
Dec  7 14:16:45 Hamm a06a9ef7ff89[13015]: root@a06a9ef7ff89:/# echo moin
```

Docker Logging

neofonie*
Data. Technology. Services.

Log-Rotation über docker-compose.yml einstellen:

```
apache:
image: httpd
     logging:
          driver: "json-file"
          options:
               max-size: "1k"
               max-file: "3"
```

Docker Logging

neofonie*
Data. Technology. Services.

Log-Rotation zentral konfigurieren:

```
~$ vi /etc/docker/daemon.json

{
  "log-driver": "json-file",
  "log-opts": {
     "max-size": "10m",
     "max-file": "10"
  }
}

~$ sudo service docker restart
```