Container und Docker

Übersicht

- Container – was ist das
  - Übersicht OS Virtualisierung
- Docker
  - Container betreiben
  - Filesystem mit Schichten - Layers
  - Container bauen und verwalten
  - Anwendungen
  - Und was kommt als nächstes ?

Container I

- Virtualisierung auf OS-Ebene
  - Es wird **kein** 'Rechner' virtualisiert, das wäre eine 'Virtuelle Maschine' VM
  - Es wird einem *ix Prozess eine neue, virtuelle, Umgebung gegeben
- Ein Prozess ist bereits ein isoliertes Objekt:
  - Eigener Speicher (evtl. virtuell), Threads, Limits, etc.
- Global im System:
  - Dateibaum (/ und alle mount-points)
  - Netzwerk, Hostname, Filter
  - Prozesstabelle, IPC, Shared Memory etc.
- Container: Die 'globalen' OS Funktionen werden virtualisiert – verschiedene Prozesse haben eigene sicht auf systemweite Daten

Container – chroot

- Für einen Prozess, wird ein anderes Verzeichniss zu /
- Klassische Anwendung: Isolierung
  - FTP oder HTTP server um sie weiter zu isolieren
  - Auch systemweit lesbare files können nicht aus versehen nach aussen gelangen
- Anwendung: System-Entwicklung und Test
  - In der Umgebung hat der Bau-, Installations- und Test-Prozess freie Hand, z.b. um in /usr/lib/…
  - Kein Risiko für den Rest des Systems

Container Anwendungen

- Nächster Schritt: Netzwerk, Limits, Hostname, etc.
  - Kann ‘wie eine VM’ isoliert werden, oder nur auch nur teilweise
  - Jail, Sandbox, Container, Virtual Environment, Zone …
- Overhead sehr gering, skaliert gut
  - Pro Server-Host: 10-100 VMs,100-1000 Container
- Nur Container des gleichen OS, kein Linux-auf-Windows
  - Bei Linux: Kernel ist gemeinsamer Punkt der Distributionen
  - DistA-Container auf DistB-Host ist ok, z.B. Debian-Container auf RHEL
- Installation eines Containers oft ein ‘kleines System’
  - Übliche Größe: 100 – 500 MB auf disk
- Einfaches Teilen von Daten via ‘bind-mount’

Fast alle OS

Sort ascending	Operating system	License	Available since/between	Features								
				File system isolation	Copy on Write	Disk quotas	I/O rate limiting	Memory limits	CPU quotas	Network isolation	Partition checkpointing and live migration	Root privilege isolation
chroot	most UNIX-like operating systems	varies by operating system	1982	Partial[5]	No	No	No	No	No	No	No	No
Docker	Linux[6]	Apache License 2.0	2013	Yes	Yes	Not directly	Not directly	Yes	Yes	Yes	No	No
Linux-VServer (security context)	Linux	GNU GPLv2	2001	Yes	Yes	Yes	Yes[7]	Yes	Yes	Partial[8]	No	Partial[9]
lmctfy	Linux	Apache License 2.0	2013	Yes	Yes	Yes	Yes[7]	Yes	Yes	Partial[8]	No	Partial[9]
LXC	Linux	GNU GPLv2	2008	Yes[10]	Partial. Yes with Btrfs.	Partial. Yes with LVM or Disk quota.	Yes	Yes	Yes	Yes	No	Yes[10]
OpenVZ	Linux	GNU GPLv2	2005	Yes	No	Yes	Yes[11]	Yes	Yes	Yes[12]	Yes	Yes[13]
Parallels Virtuozzo Containers	Linux, Windows	Proprietary	2001	Yes	Yes	Yes	Yes[14]	Yes	Yes	Yes[12]	Yes	Yes
Solaris Containers	Solaris and OpenSolaris	CDDL	2005	Yes	Partial. Yes with ZFS	Yes	Partial. Yes with Illumos.[15]	Yes	Yes	Yes[16]	No[17]	Yes[18]
FreeBSD Jail	FreeBSD	BSD License	1998	Yes	Yes (ZFS)	Yes[19]	No	Yes[20]	Yes	Yes	No	Yes[21]
sysjail	OpenBSD, NetBSD	BSD License	No longer supported, as of March 3, 2009	Yes	No	No	No	No	No	Yes	No	?
WPARs	AIX	Proprietary	2007	Yes	No	Yes	Yes	Yes	Yes	Yes[22]	Yes[23]	?
HP-UX Containers (SRP)	HPUX	Proprietary	2007	Yes	No	Partial. Yes with logical volumes	Yes	Yes	Yes	Yes	Yes	?
iCore Virtual Accounts	Windows XP	Proprietary/Freeware	2008	Yes	No	Yes	No	No	No	No	No	?
Sandboxie	Windows	Proprietary/Shareware	2004	Yes	Yes	Partial	No	No	No	Partial	No	Yes

Quelle: http://en.wikipedia.org/wiki/Operating_system–level_virtualization

Docker

- Fokussiert auf Bau, Verteilung und Ausführung von Containern
  - Ist **kein** eigenes 'OS-virtualisierungs-system'
  - Nutzt verschiedene existierende (ursprünglich nur LXC)

- Philosophie:
  - Container-Image wird vom Entwickler gebaut und eingecheckt
  - Test und Deployment können das Image jederzeit laden
  - Images sind unveränderbar, neues Image -> neue Version

- Hat sich aus Managementlösung eines PaaS-Anbieter entwickelt

Begriffe

- Registry & Repository
  - Verzeichnis für Container und Images
  - Global: hub.docker.com Eigene Registry ist möglich
  - Kann man sich vorstellen als ‘GitHub für Images’
- Image
  - Fester (readonly) Satz von Datei-Layern die einen Container bilden
- Layer
  - Alle Daten die sich zu einem vorgänger-Layer geändert haben
- Container
  - Ein Image das mit Docker zum leben erweckt wurde
- Dockerfile
  - Bauanweisung für ein Image – analog zu Makefile

Run & Check

- **`docker run`** startet einen Container basierend auf einem Image
- **`docker ps`** zeigt laufende Container
- Andere: **`start/stop/restart, wait, create, pause / unpause, inspect, top`**

```
cbook3-3:~ mnbokaem$ docker run -ti debian:latest /bin/bash
root@2b91ba2142f0:/# more /etc/debian_version
7.7
root@2b91ba2142f0:/# ip addr show eth0
16: eth0: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 02:42:ac:11:00:07 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.7/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe11:7/64 scope link
       valid_lft forever preferred_lft forever
```

```
cbook3-3:~ mnbokaem$ docker ps
CONTAINER ID        IMAGE                  COMMAND             CREATED             STATUS              PORTS               NAMES
2b91ba2142f0        debian:latest          "/bin/bash"         26 hours ago        Up 3 minutes                            thirsty_fermi
a4afb3eb4d64        jpetazzo/dind:latest   "wrapdocker"        30 hours ago        Up 3 hours                              dreamy_yonath
e7baf0191241        debian:latest          "/bin/bash"         2 days ago          Up 31 hours                             lonely_darwin
```

Run - Exec

- **`docker exec`**
  - Erlaubt den Einstieg von der Seite in einen laufenden Container
  - Seit kurzem (docker 1.3) standard, davor extra tools wie nsenter
  - Sehr hilfreich für Fehlersuche und Maintenance

```
cbook3-3:~ mnbokaem$ docker exec -ti 2b91 /bin/bash
root@2b91ba2142f0:/# df -h
Filesystem      Size  Used Avail Use% Mounted on
rootfs           19G  562M   17G   4% /
none             19G  562M   17G   4% /
tmpfs          1005M     0 1005M   0% /dev
shm              64M     0   64M   0% /dev/shm
/dev/sda1        19G  562M   17G   4% /etc/resolv.conf
/dev/sda1        19G  562M   17G   4% /etc/hostname
/dev/sda1        19G  562M   17G   4% /etc/hosts
tmpfs          1005M     0 1005M   0% /proc/kcore
```

Run - Netzwerk

- Normalerweise eigene IP in virtuellem LAN
  - Ports können mit NAT nach aussen gemappt werden
  - Keine Port-Konflikte mit mehreren gleichen Containern
- Admin hat Kontrolle

```
# docker help run
 -h, --hostname=""          Container host name

 --net="bridge"             Set the Network mode for the container
                            'bridge': creates a new network stack for the
                                          container on the docker bridge
                            'none': no networking for this container
                            'container:<name|id>': reuses another container network stack
                            'host': use the host network stack inside the container.

-P, --publish-all=false         Publish all exposed ports to the host interfaces
-p, --publish=[]                Publish a container's port to the host
                                format: ip:hostPort:containerPort | ip::containerPort |
                                     hostPort:containerPort | containerPort
                                (use 'docker port' to see the actual mapping)
```

Run - Volumes

- Bindet Verzeichnisse des Hosts in den Container
  - Nutzt bind-mount
  - Ist gedacht für alle Arbeitsdaten, evtl. Logs und Config
  - Erlaubt teilen von Verzeichnissen zwischen Containern
  - 'Native Speed' – kein Copy-on-Write Layer overhead
- Jeder Container hat zwar einen read-write layer über dem statischen Image, aber es ist 'gutes Design' den persistenten state in ein Volume zu packen.

```
# docker help run
[…]
  -v, --volume=[]            Bind mount a volume (e.g., from the host: -v /host:/container,
                             from Docker: -v /container)
  --volumes-from=[]          Mount volumes from the specified container(s)
```

Run Docker in Docker

- **`Docker`** ermöglicht auch hierarchische Container, d-in-d
  - Benötigt einen Container mit privilegien
  - Interessant z.B. für lokale Testumgebungen

```
$ docker run --privileged -t -i jpetazzo/dind
root@a4afb3eb4d64:/# 2014/11/22 02:52:48 docker daemon: 1.3.1 4e9bbfa; execdriver: native;
graphdriver:
[d3085707] +job serveapi(unix:///var/run/docker.sock)
[d3085707] +job init_networkdriver()
[info] Listening for HTTP on unix (/var/run/docker.sock)
[d3085707] -job init_networkdriver() = OK (0)
[info] Loading containers:
[info] : done.
[d3085707] +job acceptconnections()
[d3085707] -job acceptconnections() = OK (0)
root@a4afb3eb4d64:/# root@a4afb3eb4d64:/# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 02:52 ?        00:00:00 bash
root        58     1  0 02:52 ?        00:00:01 docker -d
root       137     1  0 07:25 ?        00:00:00 ps -ef
```

Run - Image laden

- `Docker` lädt images automatisch wenn nötig und cached sie lokal
- Man kann gut die einzelnen Layer erkennen

```
$ bash-3.2# docker run --privileged -t -i jpetazzo/dind
Unable to find image 'jpetazzo/dind' locally
Pulling repository jpetazzo/dind
9c674feba890: Download complete
511136ea3c5a: Download complete
d497ad3926c8: Download complete
ccb62158e970: Download complete
e791be0477f2: Download complete
3680052c0f5c: Download complete
22093c35d77b: Download complete
5506de2b643b: Download complete
bb320698cdfd: Download complete
6666069f2e10: Download complete
31e94ee8207a: Download complete
6631d8213746: Download complete
785cb00738fb: Download complete
eb37cc94f089: Download complete
Status: Downloaded newer image for jpetazzo/dind:latest
root@a4afb3eb4d64:/# 2014/11/22 02:52:48 docker daemon: 1.3.1 4e9bbfa; execdriver: native;
graphdriver:
```

Bauen - Dockerfile

```
# VERSION               0.0.1
FROM     ubuntu
MAINTAINER Thatcher R. Peskens "thatcher@dotcloud.com"
# make sure the package repository is up to date
RUN echo "deb http://archive.ubuntu.com/ubuntu precise main universe" > /etc/apt/sources.list
RUN apt-get update
RUN apt-get install -y openssh-server
RUN mkdir /var/run/sshd
RUN echo 'root:screencast' |chpasswd
EXPOSE 22
CMD    /usr/sbin/sshd -D
```

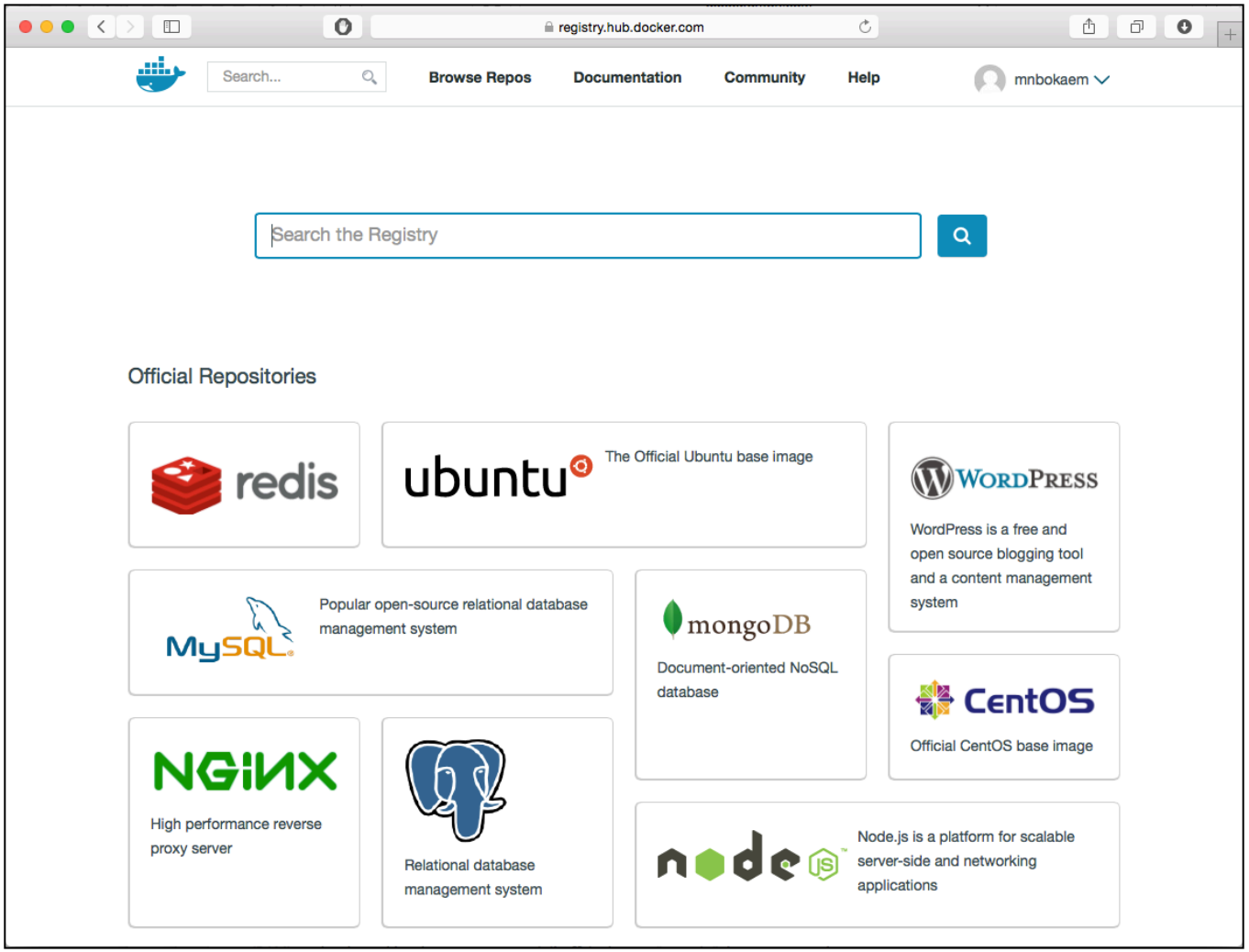
- **FROM**: Man fängt mit einem schon vorhandenen Image an
- **RUN**: Führt Kommandos innerhalb des Images aus
- Der grosse Trick: Layer-caching
  - Nach (fast) jedem statement wird ein neuer Layer angelegt
  - Bleiben start-image und Kommando gleich (und Caching ist erlaubt), existiert das Ziel-Image ja schon.

Bauen - Versionen

- **`docker tag`** – weist einem image ein tag zu
- **`docker push`** – lädt image zu Registry & Repository
- **`docker pull`** – download eines images
- **`docker login/logout`** – anmelden an (alternativer) Registry
- **`docker search`** – suchen nach Images

```
bash-3.2# docker search dind
NAME                                DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
jpetazzo/dind                       This lets you run Docker within Docker.         13                   [OK]
mattgruter/doubledocker             Run Docker inside Docker (aka. dind).           4                    [OK]
spiddy/dind-jenkins-slave                                                           2                    [OK]
getitlive0/trusty-dind              Docker-in-Docker (dind) on Ubuntu Trusty. ...   1
mazzolino/strider-dind              Strider Docker recipe (https://github.com/...   1
llamashoes/dind-kubernetes          Run kubernetes locally using docker in doc...   1                    [OK]
```

Verteilen - Hub



Docker – Hype ?

- Ja, ist aber berechtigt – denke ich
  - Probieren ist einfach, so kann man einiges filtern
- Container wird zu der Einheit in der ‘deployment’ stattfindet
  - Ersetzt OS-packaging + Anpassung
- Der Clou ist die Kombination von einigen existierenden features
  - Konsistentes bauen, verteilen und starten von Containern
  - Effizient durch Layers

- Genau was DevOps und andere brauchen

Docker – Schwächen ?

- Docker spezifisches
  - Docker kommt mit Vorstellungen von gutem oder weniger gutem Container-Design.
    'Häretiker' werden jedoch unterstützt
  - Braucht relativ neuen Linux-Kernel (3.8 / 3.10, mit Ausnahmen)
- Das Projekt ist SEHR aktiv – Probleme bleiben nicht lange offen
- Aber Container-basierte Installationen werfen neue Probleme auf
  - Containerdesign muss sich um einige Themen kümmern die in einer vollen OS-Umgebung schon gelöst sind, z.B. log-rotation
  - Wie findet eine Anwendung die über viele Container verteilt ist die Komponenten ?

Orchestrierung

- Wenn Anwendungen in zig Containern über X hosts verteilt werden, wie findet dann alles wieder zusammen ?
- Nicht wirklich adressiert von Docker selbst
- ‘Orchestrierungs-Tools’ – jede Woche was neues
  - Das ist eher ein Zeichen, daß das Thema noch nicht zufriedenstellend beantwortet ist

- CoreOS + etcd
- Docker links + Ambassador pattern
- Google Kubernetes
- Extensions zu Chef, Puppet, SaltStack und anderen

Wie geht es weiter

- Docker ist gereift, stabil und verwendbar !
- Aber Orchestrierung ist ein wunder Punkt
  - Nicht wirklich adressiert von Docker selbst
  - Und es gibt viele Anwendungen wo das nicht nötig ist
    - z.B. Entwicklungs- und Testumgebungen

- Google, RedHat und viele andere machen es zum de-fakto standard in diesem Feld

Vielen Dank!

Links:

- Docker: http://docker.com/

Gibt es noch Fragen ?