Docker Umgebungen / Provider

Spieleumgebung von Docker zum Auspobieren der Images:
(Registrierung notwendig)
https://labs.play-with-docker.com/

Docker Server/Hosting Provider:

* https://www.digitalocean.com/pricing/
* https://sloppy.io/de/pricing/
* https://www.orchardup.com/pricing
* https://aws.amazon.com/de/ecs/pricing/
* https://giantswarm.io
* https://cloud.google.com/pricing/list
* https://jelastic.com/public-cloud-pricing/
* https://www.loodse.com/
* https://upcube.io/#intro

Quelle: https://t3n.de/news/docker-hosting-613802/

Installation unter Ubuntu

```
curl -sSL https://get.docker.com/ | sh
curl -sSL https://get.docker.com/gpg | sudo apt-key add -
sudo docker run hello-world
; um nicht bei jedem Kommando sudo nutzen zu müssen
sudo usermod -aG docker mylocaluserName
; restart und dann läufts ohne sudo
; Falls dieser Fehler erscheint
;Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?
; Dann muss in der Regel noch der Daemon gestartet werden
; Temporär geht das so
sudo dockerd&
; Bei diesem Fehler
Error starting daemon: error initializing graphdriver: driver not supported
; half bei mir die Docker Installation zu aktualisieren und
; das Verzeichnis der aufs images zu löschen: sudo rm -rf /var/lib/docker/aufs
```

Installation unter Ubuntu (cont)

```
; (anschließend neustart)
```

Dein Kernel sollte mindestens 3.10 besser 3.13 oder höher sein.
Prüfe es mittels: *uname -r*
Weitere Infos unter: https://docs.docker.com/installation/ubuntulinux/

Docker File - Syntax

FROM	Base Image von dem abgeleitet wird
MAINTAINER	Produzent des Images
LABEL	Hinzufügen von Metainformationen
ARG	Definiert Build Parametervariablen deren Werte bereits ins Image geschrieben werden
ENV	Definiert Umgebungsvariablen welche beim Containerstart mit -e überschrieben werden - im Image werden nur Platzhalter angelegt
ADD	Hinzufügen von Dateien zum Image aus dem Host
COPY	Kopiert Dateien vom Host in das Image
RUN	Kommando welches beim Build ausgeführt wird
EXPOSE	Freigabe und Mapping von Ports zur Aussenwelt (Host und andere Container)
VOLUME	Definiert Volumen welche mit der Aussenwelt verknüpft werden können (Mountpoint)
WORKDIR	Festlegen des Arbeitsverzeichnisses
USER	Accountwechsel beim Build des Images z.b. von root auf jenkins
ENTRYPOINT	Kommando welches beim Containerstart ausgeführt wird (ergänzt den Entrypoint)

Docker File - Syntax (cont)

CMD	Kommando welches beim Containerstart ausgeführt wird (ergänzt den Entrypoint z.B. um Parameter)

Die Kommandos im Dockerfile werden in der Regel ausgeführt während des Build Prozesses und auf den im Image liegenden Verzeichnissen und Dateien.

Einzige Kommandos welche zur Laufzeit des Images also im Container ausgeführt werden sind: CMD und ENTRYPOINT

Details zu den Kommandos: https://docs.docker.com/v17.09/engine/reference/builder/

Command Line Befehle

docker build -t user/pojectname:tag projectfolder	Docker Image bauen aus einem Projekt mit Dockerfile
docker pull user/project	Lädt ein Image von Docker Hub herunter. Image ID funktioniert über User/Projekt
docker images	Listet alle lokal verfügbaren images
docker image inspect <imageId>	Zeigt Detailinformationen zum Image an
docker rmi user/repo:tag	Löscht lokal das zugeordnete Image.
docker image rm <imageId> [-f]	Löscht ein Image aus der lokalen Registry. Mit -f auch wenn noch Abhängigkeiten zu laufenden Containern bestehen.
docker run	Startet einen Container. Z.B. *docker run -it ubuntu bash* startet ein Ubuntu mit Shell

Command Line Befehle (cont)

docker stop <containerId>	Tötet den MainThread im Container und beendet ihn dadurch. Die Daten aus dem editierbaren Layer bleiben erhalten.
docker start <containerId>	Startet einen Container wieder an der Stelle an der er gestoppt wurde.
docker ps	Zeigt alle laufenden Container. Mit -a auch alle anderen.
docker rm <containerId|name:tag>	Löscht einen nicht laufenden Container
docker exec -it <containerId> bash	Startet eine zusätzliche Bash auf dem laufenden Container mit ID
docker commit <containerId> <mein--neuer-image-name>:-<tag-name>	Persistiert die Änderungen eines laufenden Containers als Image. z.B: docker commit c3f279d17e0a SvenDowideit/testimage:version3
docker export <containerId> -o <filename>.tar	Container als Tar File exportieren
docker save -o mein-image-save.tar <mein-image-name>	Image als Tar zur Weitergabe exportieren
docker load -i mein-image-save.tar	Docker Image aus Tar lokal einlesen
docker tag <imageId> user/project:tag	Taggen eines lokalen Images
docker login	Interaktive Eingabe der Login Daten für Docker Hub

By **Huluvu424242**
(FunThomas424242)

cheatography.com/funthomas424242/
stackoverflow.com/users/story/373498

Published 5th September, 2015.
Last updated 11th July, 2019.

Command Line Befehle (cont)

docker push user/repo:tag	Upload eines Images zum Docker Hub Portal
docker system prune	Räum lokal mal richtig auf - obsolete Dinge fliegen weg

Bitte beachten Sie, dass vor dem Upload von Images auf Docker Hub eine Registrierung bei dieser Platform notwendig ist.

Weiterhin ist für einen authorisierten Zugriff vor einem Push ein Login mittels docker login auf der Kommandozeile durchzuführen.

Best Praxis ist ein automatisch baubares Images auf Github als Projekt zu hosten und mit Docker Hub zu verbinden.

Docker in Docker (nested docker)

Du willst einen Docker Daemon in einem Docker Container starten und weißt nicht wie es geht? Dann kennst Du auch nicht die Probleme welche daraus resultieren. Zunächst lies also diese beiden Artikel:

* https://itnext.io/docker-in-docker-521958d34efd
* https://github.com/jpetazzo/dind

Jetzt weißt Du was Du wirklich willst und entscheidest zwischen blauer oder roter Pille?

A) Du willst eigentlich nur Container in einem Container starten z.B. um zu Testen oder zu bauen (Jenkins etc. ...) und diese dürfen durchaus im Hostsystem angelegt werden? Dann nimm ein von docker abgeleitetes image und mounte beim Start als Volume den docker Socket. Dein Docker Client im Container kann ab jetzt genau wie der Docker Client vom Host über den selben docker Deamon neue Container starten und Images pullen.

```
docker run -ti -v /var/run/docker.sock:/var/run/-
docker.sock docker
```

B) Du willst wirklich einen echten verschachtelten Docker - also im Container einen eigenen Docker deamon. Dann starte ein vom docker:dind Image abgeleitetes Image mit Sonderprivilegien. Alles wird gut - aber wundere Dich nicht wo Dein Speicher bleibt.

```
docker run --privileged -d docker:dind
```

Jenkins im Container

```
docker run -p 8080:8080 -p 50000:50000 -v /var/r-
un/docker.sock:/var/run/docker.sock -v /usr/bin/-
docker:/usr/bin/docker -v /var/lib/jenkins_h-
ome:/var/jenkins_home -e "DOCKER_GID_ON_HOST=do-
cker_group_id" oose/dockerjenkins
```

Quellen:

* https://www.oose.de/blogpost/jenkins-in-docker-und-mit-docker-und-fuer-docker/
* https://blogs.itemis.com/de/jenkins-mit-docker-virtualisieren-oder-doch-nicht

Troubleshooting

Kein Image im Manifest beim pull

Fehlerbild

```
> latest: Pulling from library/openjdk
> no matching manifest for unknown in the manifest
list entries
```

Ursache

Das angeforderte Image unterstützt die vorliegende Architektur nicht

Analyse

```
> docker info -f '{{.OSType}}/{{.Architecture}}'
```

Lösung (Beispielsuche nach openjdk)

```
> mkdir ~/docker-testconfig
> touch ~/docker-testconfig/config.json
{
"experimental": "enabled"
}
> docker --config ~/docker-testconfig manifest
inspect -v library/openjdk:latest | jq .[].Pl-
atform
```

Quelle

https://success.docker.com/article/error-pulling-image-no-matching-manifest

Netzwerke bilden: user-defined bridge

Im default Netzwerk (also ohne irgendwelche Anpassungen) finden sich die einzelnen Container (z.B. frontend und backend) einer App nicht. Hier konnte man früher mit dem jetzt nicht mehr unterstützten docker run Argument `--link` eine Verbindung schaffen. Der Weg wird explizit als veraltet und nicht empfohlen beschrieben.

Netzwerke bilden: user-defined bridge (cont)

Der neue Weg sind die Definition von "user-defined-brigde Netzwerken". Alle Container eines Netzwerkes können direkt untereinander kommunizieren (über den Containernamen welcher als Hostname im DNS fungiert). Hierbei wird von automatic service discovery gesprochen. Die Zuordnungen können on the fly erstellt und entfernt werden:

```
# ein Netzwerk my-net erstellen
docker network rm my-net
```

`# den Container my-nginx dem Netzwerk my-net zuordnen'

```
docker network connect my-net my-nginx
# den Container my-nginx aus dem Netzwerk my-net entfernen
docker network disconnect my-net my-nginx
```

Zusätzlich lassen sich über publish Portmappings herstellen:
`--publish 8080:80` mappt Port 80 im Container auf 8080 im Host.
Beispiele zum konkreten Aufbau von Netzwerken findet man hier:
https://docs.docker.com/network/network-tutorial-standalone/#use-user-defined-bridge-networks
Netzwerke über verschiedene Docker Deamon's nennen sich Overlay Netze:
https://docs.docker.com/network/overlay/

Quelle: https://docs.docker.com/network/bridge/##differences-between-user-defined-bridges-and-the-default-bridge

Allgemeine Infos

Docker ist ein System welches fertige Systeme (ähnlich einer virtuellen Machine) bereitstellen kann. Diese Systeme werden in Docker Images beschrieben (eine Art Template oder Klasse). Von diesen Images lassen sich dann beliebig viele Laufzeitinstanzen starten. Jede Laufzeitinstanz eines Images wird Container genannt (sozusagen die konkrete Ausprägung einer Systemklasse).
In jedem Container soll es nur einen Hauptthread geben. Solange dieser ausgeführt wird, gilt der Container als gestartet. Ist der Hauptthread beendet so geht der Container in den Zustand gestoppt über.
Die Images sind in Layer unterteilt. Jeder Layer enthält die Deltas welche auf dem Filesystem stattfanden. Images selbst sind TAR Archive. Die Layer im Image sind nur lesbar und nicht beschreibbar.

Allgemeine Infos (cont)

Beim Start eines Containers wird auch ein zusätzlicher beschreibbarer Layer on top angelegt. In diesem können die Anwendungen Änderungen am Filesystem vornehmen. Nach dem Stop eines Containers bleibt dieser Layer in diesem konkreten Container erhalten. Der Zustand lässt sich per docker commit persistieren.
Docker Images können automatisiert per Dockerfile über docker build erstellt werden. Um die Größe der Images klein zu halten (sie explodiert extrem schnell), sollte von kleinen Root Images (alpine) abgeleitet und ein Multistage Build genutzt werden.
Container mit Sicherheitsinformationen wie Passwörtern oder gültig konfigurierten GIT Zugängen sollte man niemals auf den HUB schieben!
Wird ein Container auf einem Remote Host gestartet (z.B. weil dort der Docker Daemon läuft) so beziehen sich die Parameter wie Port- und Volumenmappings auf diesen Remotehost.
Beim Mappen von Volumen werden die Berechtigungen wie User-ID und Gruppen-ID aus dem Container zum Schreiben auf das gemountete Volumen benutzt. Wenn die ID 100 im Container dem Nutzer jenkins gehört und der Hauptthread unter Nutzer jenkins läuft und dieser auf die gemounteten Volumen schreibt, dann wird auf dem Host mit der Nutzer ID 100 geschrieben auch wenn diese dann dem Nutzer Klaus oder root gehört. Zur Anwendung kommen natürlich dann die Berechtigungen im Hostsystem. Wenn Klaus also keine Rechte zum Schreiben besitzt wird der Jenkins Nutzer aus dem Container im Host nicht auf das Volumen schreiben können..
Letztlich muss auch noch beachtet werden, dass Docker Images Architektur spezifisch sind. Falls das vom Hub geladene Image nicht zur ausführenden Architektur passt kann man versuchen es selbst auf dieser Architektur zu bauen. Dazu benötigt man natürlich das Dockerfile aber das ist über den Hub und die Verlinkung zum Projekt meist zu bekommen.
Problematischer wird es wenn das im Dockerfile genutzte base Image nicht Architektur agnostisch ist. Die aktuellen offiziellen Base Images großer Tools versuchen zwar agnostisch bzgl. der Architektur zu sein aber das muss man evtl. prüfen und kann es nicht einfach als gegeben hinnehmen.
(https://stackoverflow.com/questions/52767983/docker-error-standard-init-linux-go185-exec-user-process-caused-exec-format-e)

Remotezugriff auf die Docker Engine

ACHTUNG - Remote Zugriff ist Böse - Clients spielen Root - ACHTUNG

Der Docker Deamon läuft normalerweise lokal und stellt den Socket `unix:///var/run/docker.sock` für Anfragen von docker cli bereit.

Das kann herausgefunden werden mittels:

```
sudo netstat -lnpx | grep docker.sock
```

Die Default Ports des Deamon sind 2375 (unverschlüsselt) und 2376 (verschlüsselt).

Der aktuelle Status des Deamon lässt sich so ermitteln:

```
systemctl status docker.service
```

Der Zugriff auf einen Remote Docker Deamon kann per REST API oder über Docker CLI oder einem Drittanbieter Client vorgenommen werden.

1) Zugriff über Docker-CLI in der Shell

Hier muss die Umgebungsvariable DOCKER_HOST auf den Docker socket des Remote Daemon gesetzt werden. Ab da sendet Docker CLI alle Befehle an den Remote Deamon. Beispiel:

```
export DOCKER_HOST=tcp://X.X.X.X:2376
```

2) Zugriff über das REST API

Dokumentation der Doker Engine REST API: https://docs.docker.com/engine/api/version-history/

Ein Beispiel für das Absetzen von Kommandos per REST Client:

GET http://x.x.x.x:2376/containers/json -> entspricht einem docker ps

Auf dem Rechner des Docker Deamon muss der REST Zugriff freigeschalten werden.

API aktivieren (https://success.docker.com/article/how-do-i-enable-the-remote-api-for-dockerd):

```
# /etc/systemd/system/docker.service.d/override.conf anlegen mit inhalt
[Service]
ExecStart=
ExecStart=/usr/bin/dockerd -H fd:// -H  tcp://0.0.0.0:2376
$ sudo systemctl daemon-reload
$ sudo systemctl restart docker.service
```

Docker Deamon bei Systemstart automatisch starten lassen

```
$ sudo systemctl enable docker
```

Chrome REST Erweiterung: http://bit.ly/2Q4hJ9h

Docker CLI für Windows: https://download.docker.com/win/static/stable/x86_64/

Docker Deamon Doku: https://docs.docker.com/v17.09/engine/reference/commandline/dockerd/#examples

docker run Options

-i	interacive Mode = -a STDIN
--sig-proxy=true	Alle Signale werden an den Containerthread weitergeleitet (nur ohne -t möglich)
-t	Stellt im Container eine Pseudo TTY bereit
-a stderr -a stdin -a stdout	-a verbindet stellt Streams des Hosts im Container zur Verfügung. Gülige Werte: STDIN, STDOUT, STDERR
-D	Schaltet den Debug Mode aktiv

Aufbau der Kommandozeile:

docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]

Details siehe hier: https://docs.docker.com/v17.09/engine/reference/run/#foreground

Multistage Builds (Kompaktiere das Image)

Kleine und kompakte Images zu erzeugen war bislang schwierig - der alte Trick alles in eine Zeile zu pappen und mit && zu verknüpfen ist schwer wartbar und docker erkennt nicht immer wenn etwas neu gebaut werden muss. Dann gibt es hier und da auch schon mal veraltete images in denen die Änderung von eben gar nicht gegriffen hat.

Die Lösung: Multistage Builds durch mehrere FROM Anweisungen im Dockerfile. Der Trick dabei: **COPY --from=0** kopiert nur die Deltas aus dem letzten Stage (Das 0. FROM).

Besser Du benennst die einzelnen Stages:

```
FROM golang:1.7.3 as builder
```

und nutzt die benannten Stages so:

```
COPY --from=builder ...
```

Quelle: https://docs.docker.com/v17.09/engine/userguide/eng-image/multistage-build/#use-multi-stage-builds

Docker Image Analyse

Lokale Ablage der Images

* https://stackoverflow.com/questions/19234831/where-are-docker-images-stored-on-the-host-machine

Image Formate

* aufs
* overlay2
* ...

Grobe Infos auslesen

```
>docker image inspect <imageId>
```

Image Explorer Werkzeuge

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

Image per Hand extrahieren

```
>docker save <imageId> -o <filename>.tar
```

Docker URL Aufbau

Allgemeine Syntax für Docker URL: [registry-url]/[namespace]/-[image]
zum Beispiel: docker.io/library/mysql:latest

Spezifische Eclipse Installation erstellen

```
>sudo docker run -it -e DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix batmat/docker-eclipse
; jetzt alle Plugins installieren und in einer anderen console folgendes eingeben
>docker ps
; Hiermit lässt sich die Container ID bestimmen und mit
>docker commit containerID username/imagename:version
; den Container als Image persistieren.
; Das neue Image lässt sich dann starten mit:
>sudo docker run -it -e DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix containerID:version
```

Der Hauptgrund für dieses Vorgehen ist ein bestehendes Linux System welches nicht verändert werden soll. Dennoch möchte man manchmal mit einer Eclipse-Luna neue DSLs entwickeln und mit einer Eclipse-Kepler neue RPC Anwendungen schreiben. Die benötigten Plugins und deren richtige Version können ein in die Hölle treiben. Daher ganz entspannt zukünftig nur den richtigen Docker Container starten. Klar vorher einmal bereitstellen :)

By **Huluvu424242**
(FunThomas424242)

cheatography.com/funthomas424242/
stackoverflow.com/users/story/373498

Published 5th September, 2015.
Last updated 11th July, 2019.
Page 6 of 6.