Inhaltsverzeichnis

Docker Tricks

Installieren

```
sudo apt-get install docker.io docker-compose
```

Den aktuellen Benutzer zur Gruppe docker hinzufügen, damit er auch als normaler Benutzer auf den Docker-Daemon und die laufenden Container einwirken kann.

```
sudo usermod -aG docker $USER
```

Dann **einmal ab- und wieder anmelden!**

Docker container laufen, aber sie sind nicht sichtbar in `docker ps -a` oder `docker images`? Dann ist docker vielleicht gleichzeitig per `apt` **und** als snap module installiert? Mit

```
snap list
```

kann man das überprüfen und ggf. mit

```
sudo snap remove docker
```

korrigieren

log into bash

```
docker exec -it schnipsl /bin/bash
```

log into bash in einem temporären Container aus einem Image

```
docker run --rm -it --entrypoint bash <image-id>
```

run a text editor

```
docker run --name=nano -it --rm -v=schnips-backup:/tmp/myvolume piegsaj/nano
nano
```

list content

```
docker run --rm -i -v=postgres-data:/tmp/myvolume busybox find /tmp/myvolume
```

```
# search for old container
docker ps -a

# remove it
docker rm dazzling_davinci
# find its old image
docker images
```

```
# remove it
docker rmi 48b7e7bba360

# create from scratch
docker -D build  -t schnipsl:v1 .

#run it first time
docker run -i -v schnips-backup:/app/devices/master/volumes/backup -v
schnips-runtime:/app/devices/master/volumes/runtime --network=host
0d72f342e1c2

# stop it
docker stop priceless_pare

# start it again
docker start -i priceless_pare
```

Gemeinsames Netzwerk zwischen getrennten Docker-Compose

Komischerweise muß man gemeinsame Netzwerke mit docker global definieren und kann sie dann erst von den Docker-Compose- Instanzen ansprechen?!?

```
docker network create external-example
```

```
version: '3'
services:
  service1:
    image: busybox
    command: sleep infinity

networks:
  default:
    external:
      name: external-example
```

Docker in einer Unix Client VM

Blöd, wenn man gezwungen ist, unter Windows zu entwickeln, aber keine Docker Desktop Lizenz bezahlt bekommt.

Lösungsansatz:

Docker in einer Ubuntu VM in Virtualbox laufen lassen und den Entwicklungsordner teilen.

Das Teilen findet dann entweder über die grafische Konfugurationsoberfläche von VB statt oder per

Kommandozeile [1] auf dem **Host**:

```
 VBoxManage sharedfolder add "VM name" --name sharename --hostpath "C:\test"
```

mit der Option `--transient` bleibt die Verbindung einmalig nur für diese Session, die option `--readonly` macht... guess what..

Damit das auch beim Start per Automount funktioniert, muss der Unix User Mitglied der Gruppe vboxsf sein

```
sudo usermod -aG vboxsf userName
```

mounten dann mit

```
mkdir /home/<user>/vboxshare
sudo mount -t vboxsf -o uid=1000,gid=1000 sharename /home/<user>/vboxshare
```

Docker-Compose, mehrere Instanzen und Umgebungsvariablen

Wieder mal ein erfülltes Wochenende...

Nach langem Gebastel habe ich begriffen, dass man ein Gebinde aus mehreren Containern mehrmals starten kann, wenn man den -p (–project-name) Parameter verwendet und `docker-compose` dann die Container um dem Projektnamen erweitert, weil sonst bei jedem Start die Container alle gleich heißen und man letztlich keine Neuen startet, sondern nur die bereits gestarteten anspricht.

Der Spaß fängt an, wenn die Container ins Filesystem des Hosts gemountet sind, um gemeinsam Konfig- Dateien zu lesen, aber unabhängig Dateien schreiben sollen, ohne sich in die Quere zu kommen - denn leider kann 'docker-compose' in der `docker-compose.yml` den aktuellen Projektnamen nicht als Variable benutzen. Und damit fing das Experimentieren an..

`Docker-compose` kann aber Umgebungsvariablen in die Config übernehmen, aber auch da nur zur Manipulation der Werte, aber leider nicht zur Manipulation des Key-Namens.

Der Trick ist nun, erst eine Umgebungsvariable zu definieren und die dann **gleichzeitig** als Projektname **und** in der Config selber zu benutzen. Da sich hier im Beispiel die unterschiedlichen Instanzen durch den Port unterscheiden sollen, unter dem sie erreichbar sind, ist der Port hier quasi der Schlüssel:

```
export PORT=8504 ; docker-compose --project-name ${PORT:-8504}  up
```

Damit kann man dann zum einen die Verzeichnisse manipulieren, wo die Instanzen ihre Daten speichern sollen

```
  logbook_maria:
    image: mariadb:10
    volumes:
```

```
      - ./logbookMaria/${PORT:-8504}:/var/lib/mysql
```

und man benutzt die Zahl auch gleich für den Port

```
  logbook_nginx:
     ports:
     - "${BINDADRESS:-127.0.0.1}:${PORT:-8504}:80"
```

Und damit's so richtig ins Eingemachte geht: Je nach Syntax sind die Variablen an den verschiedenen neuralgischen Stellen sichtbar- oder eben leider nicht...

Variante	im Docker Compose Aufruf	im `docker-config.yml`	im Container selber
VAR=etwas	X	-	X
export VAR=etwas	-	X	?
export VAR=etwas ;	X	X	?

Ich hoffe, ich hab das jetzt so richtig wiedergegeben..

unbenutzte Images löschen

Wird immer gerne vergessen: Der Befehl, um die Images zu löschen, die keinen Container mehr haben:

```
docker image prune [-a]
```

Remove all dangling images. If -a is specified, will also remove all images not referenced by any container.

Docker <2.0 um den Compose Befehl ergänzen

Manche Tools (z.B. Python_on_Whales) vertrauen auf das Vorhandensein des `docker compose` Befehls, was aber in vielen Distributionen noch gar nicht unterstützt wird, wo es noch das „pin-kompatible“ `docker-compose` gibt.

Zum Glück haben die Docker- Macher aber einen Weg gefunden, ältere Versionen aufzurüsten:

```
# add docker compose for docker version < 2.0
# create the docker plugins directory if it doesn't exist yet
mkdir -p ~/.docker/cli-plugins
# download the CLI into the plugins directory
curl -sSL
https://github.com/docker/compose/releases/download/v2.12.2/docker-compose-
linux-x86_64 -o ~/.docker/cli-plugins/docker-compose
# make the CLI executable
chmod +x ~/.docker/cli-plugins/docker-compose

# and do the same because the pyEdge- Service runs as root, so also root
```

```
need this extension
# create the docker plugins directory if it doesn't exist yet
sudo mkdir -p /root/.docker/cli-plugins
sudo cp ~/.docker/cli-plugins/docker-compose /root/.docker/cli-plugins
```

Docker und time permisson error

Auf dem Raspi 4 kam es beim Versuch, einen neuen Container zu installieren, zu einem nur aufwendig zu behebenden Fehler. Es beginnt mit dieser Fehlermeldung:

```
  docker build -t zuulac  https://github.com/stko/zuul-ac.git
```

```
Step 2/5 : RUN pip install --user --no-cache-dir Flask requests
---> Running in 9ccdcb51ccec
Traceback (most recent call last):
File "/usr/local/bin/pip", line 5, in <module>
from pip._internal.cli.main import main
File "/usr/local/lib/python3.10/site-packages/pip/_internal/__init__.py",
line 4, in
<module>
from pip._internal.utils import _log
File "/usr/local/lib/python3.10/site-packages/pip/_internal/utils/_log.py",
line 8, in
<module>
import logging
File "/usr/local/lib/python3.10/logging/__init__.py", line 57, in <module>
_startTime = time.time()
PermissionError: [Errno 1] Operation not permitted
```

Diese Website wußte Hilfe, hier die daraus extrahierte Zusammenfassung:

Ein paar Pakete aktuallisieren:

```
# Get signing keys to verify the new packages, otherwise they will not
install
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
04EE7237B7D453EC 648ACFD622F3D138
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
0E98404D386FA1D9 6ED0E7B82643E131
# Add the Buster backport repository to apt sources.list
echo 'deb http://httpredir.debian.org/debian buster-backports main contrib
non-free' | sudo tee -a /etc/apt/sources.list.d/debian-backports.list

sudo apt update
sudo apt install libseccomp2 -t buster-backports
```

und auch docker selbst braucht ein Update...

neue Datei

```
sudo nano /etc/apt/sources.list.d/bullseye-testing-docker.list
```

mit

```
deb http://mirrordirector.raspbian.org/raspbian/ bullseye main
```

Jetzt bloß kein apt upgrade starten!, denn erst muß das Update- Verhalten auf nur Docker eingeschränkt werden mit

```
sudo nano /etc/apt/preferences.d/bullseye-docker.pref
```

und dem Inhalt

```
Package: *
Pin: release n=bullseye
Pin-Priority: 50
```

Dann erst ein beherztes

```
sudo apt update
```

docker stoppen

```
systemctl stop docker
```

und

```
sudo apt install docker.io/bullseye
```

und jetzt zeigt docker auch hoffentlich eine Version >= 20. an

```
docker -v
```

und kann mit

```
sudo systemctl restart docker
```

wieder angestartet werden.

Danach ließ sich der neue Container „builden“

Aus einem Container aus Virtualbox auf den Host zugreifen

Was macht man, wenn man aus einem Container in einer Virtualbox- Linux- VM auf Daten des Virtualbox- Hosts zugreifen will?

(siehe auch hier)

Zuerst richtet man für die VM in Virtualbox einen gemeinsamen Ordner ein mit folgenden Parametern ein

Name	Der „Share“ Name, der später in der VM als Share des Mount- Befehls verwendet wird (`Development`)
Pfad	Der Pfad auf dem Host, der in den Container gemappt werden soll
Zugriff	Voll (oder halt weniger)
Automatisches Einbinden	Normalerweise „Ja“
Nach	Der Pfad, unter dem es in der VM selber gemountet wird (dies ist nicht der Pfad im endgültigen Container!) (`/media/host/development`)

Nach dem Starten der VM legt man einmalig den Mount- Ordner an

```
sudo -p /media/host/development
sudo chmod -R a+rwx  /media/host/development
```

und mountet man erstmal den geteilten Ordner in den VM client

```
sudo mount -t vboxsf Development /media/host/development
```

um diesen Ordner dann zu guter Letzt zum Beispiel so an einen Container als Volume zu übergeben

```
docker run --rm -it -v /media/host/development/:/tmp debian:10-slim
/bin/bash
```

Self certified certificates in Docker übernehmen

Wirft docker eine Fehlermeldung wie

```
$ docker run --rm -it burda/thunder-php:latest
Unable to find image 'burda/thunder-php:latest' locally
docker: Error response from daemon: Get "https://registry-1.docker.io/v2/":
tls: failed to verify certificate: x509: certificate signed by unknown
authority.
```

dann lieg's gerne mal am firmen-eigenen Certificate des Firmen- Proxies. Abhilfe

Eine externe HTTPS Website im Windows- Chrome- Browser öffnen, Zertifikat ansehen, das Firmen-eigene Root- CA lokalisieren, als company-CA.crt speichern und per scp auf den Zielrechner übertragen

```
  sudo apt-get install -y ca-certificates
  sudo cp company-ca.crt /usr/local/share/ca-certificates
  sudo update-ca-certificates
  sudo cp company-CA.crt /etc/docker/certs.d/docker.io/
  service docker restart
```

Einen Container testweise kopieren und mit anderem Entry- Point starten

Container stoppen

```
docker stop container_name
```

Container- id raussuchen

```
docker ps -a
```

den Container in ein neues „Copy-Image“ kopieren

```
docker commit <CONTAINER_ID> user/test_image
```

die Kopie mit neuem Entry- Point starten

```
docker run -ti --entrypoint=sh user/test_image
```

Anwendungen

Einen kleinen statischen Webserver einrichten

Ein Verzeichnis für die Pages anlegen

```
mkdir -p ~/docker/nginx-webserver/pages
```

dorthin wechseln

```
cd ~/docker/nginx-webserver/pages
```

und den Container dort laufen lassen (ggf. die Parameter anpassen)

```
 $ docker run -it --rm -d -p 8080:80 --name web -v .:/usr/share/nginx/html
nginx
```

[1)]
https://askubuntu.com/questions/161759/how-to-access-a-shared-folder-in-virtualbox