Übung: Betrieb und Management von Docker-Containern

Docker Container, im Hintergrund, mit Namen c1 starten

In [1]:

```
! docker container run --detach --name c1 ubuntu /bin/bash -c "for i in {1..1000}; do echo
```

```
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu

d66f5408: Pulling fs layer
c27f669e: Pulling fs layer
21273955: Pulling fs layer
Digest: sha256:70fc21e832af32eeec9b0161a805c08f6dddf64d341748379de9a527c01b6
ca1
Status: Downloaded newer image for ubuntu:latest
2282181930d229c71f85fc0fa37a6944ebc50a55c6cd2c209d05bdfaced9911f
```

Laufende Container ausgeben

In [2]:

```
! docker container ps
```

```
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS                  PORTS               NAMES
2282181930d2        ubuntu              "/bin/bash -c 'for i…"   1 second ag
o        Up Less than a second                       c1
```

Container stoppen und Kontrolle ob nicht mehr vorhanden

In [3]:

```
! docker container stop c1
! docker container ps
```

```
c1
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

Container ist nicht mehr ersichtlich, die Daten und das Log ist noch vorhanden:

In [4]:

```
! docker logs c1
```

```
1
2
3
4
5
6
7
8
9
10
11
12
```

Gestoppten Container wieder starten und Kontrolle ob wieder vorhanden

In [5]:

```
! docker container start c1
! docker container ps
```

```
c1
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS                  PORTS               NAMES
2282181930d2        ubuntu              "/bin/bash -c 'for i…"   15 seconds
ago      Up Less than a second                       c1
```

Der Container läuft wieder. Der Prozess innerhalb des Containers wurde neu gestartet. Der Zähler fängt wieder bei 1 zu zählen.

In [6]:

```
! docker logs c1
```

```
1
2
3
4
5
6
7
8
9
10
11
12
1
2
```

Änderungen im Dateisystem des Containers

Erstellen einer Datei im Container

In [7]:

```
! docker container exec c1 /bin/bash -c "echo 'Ein Test um `date`' >/test.txt"
```

Ausgabe der geänderten Dateien `diff` , des Verzeichnisses `ls -l` und der Inhalt der Datei `cat`

In [8]:

```
! docker container diff c1
! docker container exec c1 ls -l
! docker container exec c1 cat /test.txt
```

```
A /test.txt
total 68
drwxr-xr-x   2 root root 4096 Apr 24 21:07 bin
drwxr-xr-x   2 root root 4096 Apr 24  2018 boot
drwxr-xr-x   5 root root  340 May 14 14:01 dev
drwxr-xr-x   1 root root 4096 May 14 14:01 etc
drwxr-xr-x   2 root root 4096 Apr 24  2018 home
drwxr-xr-x   8 root root 4096 May 23  2017 lib
drwxr-xr-x   2 root root 4096 Apr 24 21:07 lib64
drwxr-xr-x   2 root root 4096 Apr 24 21:07 media
drwxr-xr-x   2 root root 4096 Apr 24 21:07 mnt
drwxr-xr-x   2 root root 4096 Apr 24 21:07 opt
dr-xr-xr-x 260 root root    0 May 14 14:01 proc
drwx------   2 root root 4096 Apr 24 21:07 root
drwxr-xr-x   1 root root 4096 Apr 26 22:21 run
drwxr-xr-x   1 root root 4096 Apr 26 22:21 sbin
drwxr-xr-x   2 root root 4096 Apr 24 21:07 srv
dr-xr-xr-x  13 root root    0 May 14 14:01 sys
-rw-r--r--   1 root root   41 May 14 14:01 test.txt
drwxrwxrwt   2 root root 4096 Apr 24 21:07 tmp
drwxr-xr-x   1 root root 4096 Apr 24 21:07 usr
drwxr-xr-x   1 root root 4096 Apr 24 21:07 var
Ein Test um Tue May 14 14:01:25 UTC 2019
```

Stoppen des Containers. Die Datei test.txt und deren Inhalt bleibt erhalten.

Hinweis: Neben dem regulären Stopp haben wir auch die Möglichkeit, eine (z. B. hängende) Container-Instanz per `docker container kill` zu killen. Default wird SIGTERM (-15) angenommen, aber natürlich stehen alle üblichen Signale per Schalter (-s/--signal) zur Verfügung.

In [9]:

```
! docker container stop c1
```

```
c1
```

Löschen des Containers. Damit geht auch die Datei test.txt bzw. deren Inhalt verloren.

In [10]:

```
! docker container rm c1
```

```
c1
```

Prozessverwaltung im Container

Wir starten wieder den Container von vorher

In [11]:

```
! docker container run --detach --name c1 ubuntu /bin/bash -c "while true; do sleep 30000;
```

```
a6088ed04f10970f139c6df00ca9cbb091b0f5f0976529a27c774788afadfc32
```

In [12]:

```
! docker container exec c1 ps ax
```

```
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     0:00 /bin/bash -c while true; do sleep 30000; done
    8 ?        S      0:00 sleep 30000
    9 ?        Rs     0:00 ps ax
```

Der erste Prozess im Container bekommt die **Id: 1**.

Die Sichtweise auf die Prozesse des Containers ist von innen (innerhalb des Containers) und von aussen (Host), völlig unterschiedlich.

Die externe Sicht kann wie folgt ausgegeben werden:

In [13]:

```
! docker container top c1
```

```
PID                 USER                TIME                COMMAND
953                 root                0:00                /bin/bash -c whi
le true; do sleep 30000; done
986                 root                0:00                sleep 30000
```

Ausgabe weitere Informationen zum Container, wie Memory, CPU Verbrauch

In [14]:

```
! docker container stats --no-stream c1
```

```
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMI
T     MEM %               NET I/O             BLOCK I/O           PIDS
a6088ed04f10        c1                  0.00%               1.277MiB / 5.828
GiB   0.02%               0B / 0B             0B / 0B             2
```

Aufräumen

In [15]:

```
! docker container stop c1
! docker container rm c1
```

```
c1
c1
```

Docker logging

Start eines fehlerhaften Containers

In [16]:

```
! docker container run --detach --name c3 ubuntu /bin/bash "while true; do sleep 30000; don
! docker container ps
```

```
f76363d8d5cb4707c2cf23e602e72a4e9cb0ecc0c491359f5be204105a56c303
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

Ausgabe der Logging Informationen zum fehlerhalften Container

In [17]:

```
! docker container logs c3
```

```
/bin/bash: while true; do sleep 30000; done: No such file or directory
```

Fehler: die Option -c wurde nicht angegeben.

In [18]:

```
! docker rm c3
```

```
c3
```

In [ ]: