

Docker

Nützliches zu Docker, Docker-Compose und Co.

- Container starten & direkt in eine Shell einsteigen
- kurze Erklärungen
- Docker Backup und Restore
 - Manuelles Backup und Restore

Container starten & direkt in eine Shell einsteigen

```
docker run --name ubuntu_bash --rm -i -t ubuntu bash
```

Hier sorgt `-t (-tty)` für das Zuweisen einer Pseudo-TTY und `-i (-interactive)` steht für den interaktiven-Modus, während `-rm` dafür sorgt dass der Container entsorgt wird, wenn der User die Bash beendet.

kurze Erklärungen

Container starten & Ports freigeben

```
docker run -p 127.0.0.1:80:8080/tcp ubuntu bash
```

Ports werden mit -p definiert nach dem Schema LOKALER PORT:CONTAINER PORT, somit wird in diesem Beispiel der Port 8080 im Container auf den Port 80 der Hostmaschine und der Adresse 127.0.0.1 freigegeben / weitergeleitet

Laufende Docker Container anzeigen

```
docker ps
```

Alle Docker Container anzeigen (laufende / angehaltene):

```
docker ps --all
```

Compose docker-compose.yml and run

```
docker compose up -d --force-recreate
```

Container Stats

```
docker container stats
```

oder

```
docker ps / docker ps -a
```

Alle Container anhalten

```
docker stop $(docker ps -a -q)
```

Alle Container löschen

```
docker rm $(docker ps -a -q)
```

Alle Images löschen

```
docker rmi $(docker images -q)
```

Oder wenn asap alles weg muss!

```
docker stop $(docker ps -a -q) &&
```

```
docker rm $(docker ps -a -q) &&
```

```
docker rmi $(docker images -q)
```

Docker Container neustarten in einem Befehl

```
docker-compose up --force-recreate
```

(c) c0by.de

Docker Backup und Restore

Manuelles Backup und Restore

Docker Volume

Zuerst sollte man wissen wie das zu sichernde Docker Volume heißt. Eine Übersicht ergibt der folgende Befehl:

```
docker volume ls
```

Ich möchte das Volume mit dem Namen `test-data` sichern, mein Backup Verzeichnis soll `/backup/volumes/` sein, welches vorher angelegt werden muss:

```
mkdir -p /backup/volumes/  
docker run --rm \  
    -v /backup/volumes:/backup \  
    -v test-data:/data:ro \  
    debian:stretch-slim bash -c "cd /data && /bin/tar -czvf /backup/test-data.tar.gz ."
```

Zum Restore das Ganze einfach umdrehen.

```
docker run --rm \  
    -v /backup/volumes:/backup \  
    -v test-data:/data \  
    debian:stretch-slim bash -c "cd /data && /bin/tar -xzvf /backup/test-data.tar.gz"
```

Nun sollten wieder alle Dateien an Ort und Stelle sein. Falls Dateien am Ort vorhanden sind, werden diese mit den Dateien aus dem Backup überschrieben.

MySQL

So kann man ganz einfach eine MySQL oder eine ältere MariaDB mittels `mysqldump` und `mysql` Sichern und Wiederherstellen:

```
# Sichern
```

```
docker exec CONTAINERNAME /usr/bin/mysqldump -u root --password=ROOTPASSWORD DATABASE > backup.sql
```

```
# Wiederherstellen
```

```
cat backup.sql | docker exec -i CONTAINERNAME /usr/bin/mysql -u root --password=ROOTPASSWORD DATABASENAME
```

und das Ganze nun noch komprimiert mit `gzip`:

```
# Sichern
```

```
docker exec CONTAINERNAME /usr/bin/mysqldump -u root --password=ROOTPASSWORD DATABASE | gzip > backup.sql.gz
```

```
# Wiederherstellen
```

```
zcat backup.sql.gz | docker exec -i CONTAINERNAME /usr/bin/mysql -u root --password=ROOTPASSWORD DATABASENAME
```

MariaDB

So kann man ganz einfach eine MariaDB mittels `mariadb-dump` sichern und mit dem `mysql` command wiederherstellen:

```
# Sichern
```

```
docker exec CONTAINERNAME /usr/bin/mariadb-dump -u root --password=ROOTPASSWORD DATABASE > backup.sql
```

```
# Wiederherstellen
```

```
cat backup.sql | docker exec -i CONTAINERNAME /usr/bin/mysql -u root --password=ROOTPASSWORD DATABASENAME
```

und das Ganze nun noch komprimiert mit `gzip`:

```
# Sichern
```

```
docker exec CONTAINERNAME /usr/bin/mariadb-dump -u root --password=ROOTPASSWORD DATABASE | gzip > backup.sql.gz
```

```
# Wiederherstellen
```

```
zcat backup.sql.gz | docker exec -i CONTAINERNAME /usr/bin/mysql -u root --password=ROOTPASSWORD
```

PostgreSQL

Um PostgreSQL zu sichern, verwenden wir einfach `pg_dump` oder für alle Datenbanken `pg_dumpall`. Die Befehle schicken wir direkt in den Datenbank Container:

```
# Backup Aller Datenbanken unkomprimiert
```

```
docker exec CONTAINERNAME pg_dumpall -c -U POSTGRESUSER > backup.sql
```

```
# Backup Aller Datenbanken komprimiert
```

```
docker exec CONTAINERNAME pg_dumpall -c -U POSTGRESUSER | gzip > backup.sql.gz
```

```
# Backup einer Bestimmten Datenbank unkomprimiert
```

```
docker exec CONTAINERNAME pg_dumpall -c -U POSTGRESUSER -d DATABASENAME > backup.sql
```

```
# Backup einer Bestimmten Datenbank komprimiert
```

```
docker exec CONTAINERNAME pg_dumpall -c -U POSTGRESUSER -d DATABASENAME | gzip > backup.sql.gz
```

Der Restore erfolgt dann so:

```
# Restore Aller Datenbanken unkomprimiert
```

```
cat backup.sql | docker -i exec CONTAINERNAME psql -U POSTGRESUSER
```

```
# Restore Aller Datenbanken komprimiert
```

```
zcat backup.sql.gz | docker -i exec CONTAINERNAME psql -U POSTGRESUSER
```

```
# Restore einer Bestimmten Datenbank unkomprimiert
```

```
cat backup.sql | docker -i exec CONTAINERNAME psql -U POSTGRESUSER -d DATABASENAME
```

```
# Restore einer Bestimmten Datenbank komprimiert
```

```
zcat backup.sql.gz | docker -i exec CONTAINERNAME psql -U POSTGRESUSER -d DATABASENAME
```

InfluxDB v2.x

Die InfluxDB in Version 2.x bringt mit der CLI influx einen Parameter `backup / restore` mit. Mit diesem kann man ganz einfach die InfluxDB Datenbank sichern. Voraussetzung hierfür ist, dass ihr ein Backup Folder in euren Container vorab nach `/backup` mountet.

volumes:

```
- "/backup/influxdb:/backup/"
```

dann kann man ein komplettes komprimiertes Backup der Datenbank ganz einfach mit dem folgenden Befehl ausführen:

```
docker exec CONTAINERNAME influx backup --compression gzip /backup/backup1
```

Das Backup file sollte dann im Ordner unter /backup/backup1 abgelegt werden. Den Backup Status sieht man im STDOUT. Der Restore kann dann durch den restore Parameter ausgeführt werden:

```
# Restore aller Timeseries
```

```
docker exec CONTAINERNAME influx restore /backup/backup1
```

```
# Restore und alles ersetzen
```

```
docker exec CONTAINERNAME influx restore /backup/backup1
```

Docker Compose Projekt Ordners

Um ein Compose Projekt zu sichern ist es am besten einfach ein `tar.gz` des gesamten Ordners zu erstellen:

```
mkdir /opt/backup-compose-projects/  
cd /opt/composeproject1  
tar -czvf /opt/backup-compose-projects/composeproject1.tar.gz .
```

Der Restore erfolgt dann durch Entpacken des `tar.gz`

```
mkdir /opt/composeproject1  
cd /opt/composeproject1  
tar -xzvf /opt/backup-compose-projects/composeproject1.tar.gz
```

schon ist alles wieder an Ort und Stelle.

Docker Container bzw. Image

Auch Docker Container oder Images selbst kann man sichern und wiederherstellen, oder besser gesagt, man kann so den Status eines Containers in ein Image abspeichern (wie einen Snapshot). Das erstellte Image bzw. der Snapshot kann dann gesichert werden. Dies kann Sinn machen, wenn man zu Debugging Zwecken einen Container mit dem aktuellen Stand in eine Testumgebung

kopieren möchte, um dort dann ausgiebig zu troubleshooten.

Wir wollen nun wissen wie unser Container heißt, mit diesem Befehl bekommt man eine Übersicht aller laufenden Container:

```
docker ps
```

nun sucht man sich den zu sichernden Container heraus und nutzt entweder dessen `CONTAINER ID` oder den Namen:

```
docker commit -p CONTAINER-ID BACKUP-NAME
```

verifizieren kann man das nun mit folgendem Befehl, in der Liste sollte nun irgendwo der `BACKUP-NAME` auftauchen

```
docker image ls
```

Um das Image nun auf der Festplatte abzuspeichern:

```
docker save -o backup-name.tar BACKUP-NAME
```

komprimieren kann man die Datei zum Schluss ebenfalls noch

```
gzip backup-name.tar
```

Nun hat man eine komprimierte Version des Docker Containers.

Der Restore des Images erfolgt dann mit `docker load`:

```
docker load -i backup-name.tar
```

Um den Container wieder herzustellen, kann dann einfach das Image mit `docker run` gestartet werden:

```
docker run -d --name NEWCONTAINERNAME BACKUP-NAME
```