

Docker Compose

Commands für Docker Compose

- `docker-compose build` baut die Services
- `docker-compose up -d [SERVICENAME]` erstellt die Container und startet diese. Die Container werden in der korrekten Reihenfolge erstellt und gestartet. Wenn die Container bereits erstellt sind und laufen, bewirkt dieser Befehl einen Restart. `-d` : startet die Container im Hintergrund
- `docker-compose down` stoppt und entfernt alle Container, Netzwerke, Images und Volumes
- `docker-compose start [SERVICENAME]` startet die Container, dazu müssen diese aber vorher eingestellt worden sein.
- `docker-compose stop [SERVICENAME]` stoppt alle zu dem Stack gehörenden Container.
- `docker-compose rm -f` entfernt alle zu dem Stack gehörenden Container, diese müssen gestoppt sein. `-f` : erzwingt das Löschen der Container
- `docker-compose logs` zeigt die logs aller zu dem Stack gehörenden Container an. Wenn der Stack mit `docker-compose up -d` im Hintergrund gestartet wurde können so die Logfiles verfolgt werden.
- `docker-compose ps` zeigt die Liste aller dem Stack zugehörenden Container und deren Status an.
- `docker-compose exec SERVICENAME /bin/sh` gleich wie bei `docker exec` wird ein Befehl innerhalb des Containers ausgeführt. Da die Container, welche von `docker-compose` erstellt werden, immer den Namen des Ordners in dem sich das `docker-compose.yml` file befindet, enthält, ist der Name des Containers meist `ORDERNAME_SERVICENAME_1` (für die erste Instanz des Services) und könnte auch mit `docker exec ORDERNAME_SERVICENAME_1` angesprochen werden. Bei `docker-compose exec` kann der Container aber eben direkt über den `SERVICENAME` angesprochen werden. Hier werden die Parameter `-it` nicht benötigt.
- `docker-compose scale SERVICENAME=X` die Anzahl der laufenden Containerinstanzen des Service `SERVICENAME` wird auf `X` erhöht oder verringert. Dabei muss darauf geachtet werden, dass die Definition des Services im `docker-compose.yml` File skalierbar ist. Wenn zum Beispiel `ports: -80:80` angegeben ist kann es zu einem Konflikt kommen. Es kann nur ein Container von diesem Service erstellt werden, da ein zweiter ebenfalls auf den Port 80 des Hostsystem gemappt haben möchte, dieser aber bereits in Verwendung ist. In der `docker-compose.yml` muss `ports: -80:80` zu `ports: -80` geändert werden. Docker vergibt und managed die internen Ports selbst.

Einführung in Docker Compose

eine gute Einführung in die Syntax und Bedeutung der Befehle [findest du hier](#). oder [hier](#)

Installation von Docker Compose

Wie man `docker-compose` installiert und ein erstes `yml` file erstellt [findest du hier](#)

Docker Compose [erklärt](#)

Secrets und docker-compose - ein Beispiel

hier an einem Beispiel von Nextdcloud

```
version: '3.8'

services:
  mariadb:
    image: mariadb:10.11
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD_FILE: /run/secrets/db_root_password
      MYSQL_DATABASE: nextcloud
      MYSQL_USER: nextcloud
      MYSQL_PASSWORD_FILE: /run/secrets/db_user_password
    secrets:
      - db_root_password
      - db_user_password
    volumes:
      - db:/var/lib/mysql

  redis:
    image: redis:alpine
    restart: always
    command: ["redis-server", "--requirepass", "$(cat /run/secrets/redis_password)"]
    secrets:
      - redis_password
    volumes:
      - redis:/data

  nextcloud:
    image: nextcloud:31
    restart: always
    depends_on:
      - mariadb
      - redis
    environment:
      MYSQL_PASSWORD_FILE: /run/secrets/db_user_password
      MYSQL_DATABASE: nextcloud
      MYSQL_USER: nextcloud
      MYSQL_HOST: mariadb
      REDIS_HOST: redis
      REDIS_HOST_PASSWORD_FILE: /run/secrets/redis_password
      NEXTCLOUD_ADMIN_PASSWORD_FILE: /run/secrets/nextcloud_admin_password
      NEXTCLOUD_ADMIN_USER: admin
    secrets:
```

```
    - db_user_password
    - redis_password
    - nextcloud_admin_password
  volumes:
    - nextcloud:/var/www/html
  ports:
    - 8080:80

volumes:
  db:
  redis:
  nextcloud:

secrets:
  db_root_password:
    file: ./secrets/db_root_password
  db_user_password:
    file: ./secrets/db_user_password
  redis_password:
    file: ./secrets/redis_password
  nextcloud_admin_password:
    file: ./secrets/nextcloud_admin_password
```

Docker container basierend auf docker-compose.yml neu erstellen:

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

bedeutet:

up : Starte/erstelle Container basierend auf deinem docker - compose .yml (Service-Definitionen).

-d : *detached* – läuft im Hintergrund, du bekommst die Konsole sofort zurück.

–force-recreate : Container werden **neu erstellt**, auch wenn Docker denkt „ist eh schon aktuell“. → wichtig, wenn du z. B. **Volumes/Mounts, environment** oder **Config-Dateien** geändert hast und sicherstellen willst, dass das wirklich im Container landet.

app : Nur der **Service app** wird betroffen (nicht db/redis etc.). Was dabei typischerweise passiert:

Der **alte Container** für app wird gestoppt und entfernt.

Ein **neuer Container** wird mit derselben Service-Config erstellt (inkl. neuen Mounts/ENV).

Benannte Volumes wie nextcloud: bleiben erhalten (deine Daten sind nicht weg).

Nicht benannte/temporäre Container-Dateien sind weg (weil neuer Container). Praktisch als Merksatz:

→ „**Starte app neu, aber garantiert mit *frischer Container-Instanz* und aktueller Compose-Konfiguration.**“

From:

<http://wiki.waldhofer.at/> - **Wiki von Franz**

Permanent link:

<http://wiki.waldhofer.at/doku.php?id=docker:compose&rev=1770869302>

Last update: **2026/02/12 05:08**

