Docker

Einige Grundlagen zu Docker und Docker Compose. Docker ermöglicht mit Containern eine leichtgewichtige Virtualisierung.

- Compose
- Netzwerke
- Variablen
- Docker Daemon Konfiguration
- Container automatisch aktualisieren

Compose

Mit Docker Compose gibt es eine Möglichkeit, die Konfiguration von Docker Containern komfortabel in einer Datei zu beschreiben und diese dann an den Docker Daemon zu übergeben, sodass diese den Container entsprechend der Konfiguration erstellt und ausführt. Hierbei können mehrere Container in einer Datei beschrieben werden. Zusätzlich wird automatisch ein gemeinsames internes Docker Netzwerk für die Container erstellt, sodass diese direkt miteinander kommunizieren können.

Die Konfiguration wird im sog. YAML Format eingegeben, dieses hat ein sehr striktes Format, welches einzuhalten ist. Eine solche Compose YAML Konfiguration kann z. B. wie folgt aussehen:

```
services:
  bookstack:
    image: lscr.io/linuxserver/bookstack
    container_name: bookstack
    environment:
      - DB_HOST=bookstack_db
      - DB_PORT=3306
      - DB_USER=bookstack
      - DB_PASS=Ein_langes_Passwort
      - DB_DATABASE=bookstackapp
    volumes:
      - ./bookstack_app_data:/config
    ports:
      - 80:80
    restart: unless-stopped
    depends_on:
      - bookstack_db
  bookstack_db:
    image: lscr.io/linuxserver/mariadb
    container_name: bookstack_db
    environment:
      - MYSQL_ROOT_PASSWORD=Noch_ein_extra_langes_Passwort
      - MYSQL_DATABASE=bookstackapp
      - MYSQL_USER=bookstack
      - MYSQL_PASSWORD=Ein_langes_Passwort
    volumes:
```

```
      - ./bookstack_db_data:/config
    restart: unless-stopped
```

Die Konfiguration wird in eine Datei mit dem Namen `docker-compose.yml` eingegeben. Es kann auch ein beliebiger Name vergeben werden, jedoch ist es übersichtlicher und einfacher, für jedes Projekt einen eigenen Ordner mit jeweils dieser Datei anzulegen. Dieser Name ist für Compose der Standard und wird standardmäßig bei anderen Befehlen verwendet (dann muss nicht extra der Name der Konfiguration nochmals angegeben werden).

Hierbei werden die selben Parameter verwendet, wie sie z. B. beim `docker run` Befehl übergeben werden. Sie müssen nur in das YAML Format übertragen werden. Einige Parameter können nicht 1:1 übernommen werden, da Compose einige zusätzliche Möglichkeiten bietet.

in diesem Beispiel werden 2 Container (Services) konfiguriert: *bookstack* und *bookstack_db*.

Alle Optionen sollten bereits vom docker run bekannt sein, auch das Image wurde dort verwendet, nur das es nicht expliziert als Image bezeichnet wurde (`sudo docker run --name test -d lscr.io/linuxserver/bookstack`).

Ein Parameter ist neu: `depends_on`. Mit diesem Parameter können Abhängigkeiten zwischen den Containern definiert werden. So muss in diesem Beispiel zuerst der Container *bookstack_db* erstellt werden und erst danach wird von Compose der Container *bookstack* erstellt.

Weiterhin konfiguriert Compose automatisch ein internes Default Netzwerk vom Typ Bridge für die beiden Container. Sodass diese sich untereinander mit den Namen aus der Config (*bookstack* und *bookstack_db*) ansprechen können. Diese Kommunikation erfolgt dann innerhalb des Docker Daemons und verlässt nicht den Host. D. h. in diesem Fall wäre die Datenbank (*bookstack_db*) nur für den anderen Container erreichbar. Weder von extern noch andere Container auf dem Docker Server können mit dem Container kommunizieren. Nur der Container *bookstack* wäre über Port 80 von extern und für andere Container erreichbar.

Um die Docker Compose Konfiguration auszuführen, kann am besten in das Verzeichnis der YAML Datei gewechselt werden. Danach wird je nach nach gewählter Installation `sudo docker-compose up -d` oder `sudo docker compose up -d` (keine Bindestrich zwischen docker und compose) eingegeben, um die Standard Konfiguration `docker-compose.yml` zu starten. Compose erstellt dann die gewünschten Container mit den angegeben Optionen. Sollten die Container bereits mit dieser Compose Konfiguration erstellt worden sein, so werden die Container in dieser neu erstellt, dessen Konfiguration geändert wurde.

Netzwerke

In Docker gibt es mehrere Möglichkeiten, um Container untereinander zu vernetzen und/oder mit der Außenwelt.

Container verbinden

Um Container untereinander auf dem Docker Host zu vernetzen, ist eine Möglichkeit ein eigenes Netzwerk vom Typ Bridge zu erstellen. Dazu kann folgender Befehl genutzt werden.

```
sudo docker network create mein-netzwerk
```

Anschließend können Container mit folgendem Befehl zu diesem Netzwerk hinzugefügt werden.

```
sudo docker network connect mein-netzwerk mein-container
```

> Es können laufende Container zum Netzwerk hinzugefügt werden, hierfür müssen die Container nicht gestoppt werden. Auch das Entfernen funktioniert ohne Container Neustart.

Um einen Container wieder aus einem Netzwerk zu entfernen, kann der folgende Befehl verwendet werden.

```
sudo docker network disconnect mein-netzwerk mein-container
```

Wird ein Netzwerk nicht mehr benötigt, lässt es sich mit folgendem Befehl entfernen.

```
sudo docker network rm mein-netzwerk
```

> Wenn mit Docker Compose Container verwaltet werden, dann wird automatisch immer ein Default Netzwerk erstellt, in das alle Container aus der Compose Konfiguration eingefügt werden. Ausnahme: wenn Container in der Compose Konfiguration explizit einem Netzwerk zugewiesen werden, dann werden diese nicht dem Default Netzwerk zugewiesen.

Compose Container verbinden

Wenn die Container aus verschiedenen Docker Compose Konfigurationen miteinander verbunden werden sollen, dann wird am besten mit den oben genannten Befehlen ein extra Netzwerk erstellt.

Erstellen wir beispielweise das Netzwerk *testnet*.

```
sudo docker network create testnet
```

Anschließend werden die gewünschten Konfigurationen wie folgt ergänzt.

Docker Compose A:

```
services:
  webserver-A:
    image: nginx
    networks:
      - testnet
networks:
  testnet:
    name: testnet
    external: true
```

Docker Compose B:

```
services:
  webserver-B:
    image: nginx
    networks:
      - testnet
networks:
  testnet:
    name: testnet
    external: true
```

Anschließend die beiden Konfigurationen neu starten und nun wären Webserver-A und Webserver-B in einem Netzwerk und könnten miteinander über den Docker Socket kommunizieren. Die Kommunikation läuft hierbei im virtuellen Docker Netzwerk ab und verlässt nicht den Host. Dadurch ist diese Verbindung besonders schnell und sicher.

Variablen

Docker Compose bietet die Möglichkeit Variablen aus einer Datei zu laden. Dadurch ergibt sich die Möglichkeit, bestimmte Informationen wie z. B. der Name einer Domäne in einer Datei abzulegen, sodass dieser von mehreren Compose Konfigurationen genutzt werden kann.

Der einfachste Weg ist das erstellen der Datei `.env`. Diese Datei wird von Docker Compose standardmäßig direkt akzeptiert und muss nicht mal in der Compsoe Konfiguration benannt sein, solange sie im selben Verzeichnis liegt. Daneben können auch eigene Dateien verwiesen werden, die einen anderen Namen haben und an einem anderen Ort liegen. Folgend die genauen Erklärungen.

Variablen mit .env

Bevor wir Variablen anlegen, erstellen wir uns erstmal eine Compose Konfiguration, in welcher wir später einige Werte durch Variablen austauschen.

```
services:
  webserver:
    image: nginx:latest
    container_name: webserver
    hostname: webserver
    ports:
      - 80:80
```

Angenommen, wir würden mehrere Webserver mit Nginx bereitstellen. Dann müssen diese alle einen eigenen Namen erhalten und die Ports müssten auch angepasst werden. Hier könnte ein Ordner als Template angelegt werden, welcher die Compose Konfiguration enthält und die *.env* Datei. Dann könnte dieser einfach kopiert, die *.env* Datei angepasst und die Konfiguration direkt gestartet werden.

Zunächst ändern wir die Compose Konfiguration wie folgt.

```
services:
  webserver:
    image: nginx
    container_name: ${DIENST_NAME}
    hostname: ${DIENST_NAME}
    ports:
      - ${DIENST_PORT}:80
```

Und schreiben nun folgendes in die *.env* Datei.

```
DIENST_NAME=webserver
DIENST_PORT=80
```

Dies wäre die gleiche Konfiguration wie oben, nur das diesmal nicht in der Compose Konfiguration editiert werden muss. Besonders bei komplexen Compose Konfigurationen ist der Einsatz von Variablen sehr hilfreich, um einfacher die gewünschten Parameter ändern zu können.

Um die Docker Compose Konfiguration auszuführen, kann am besten in das Verzeichnis der YAML Datei gewechselt werden. Danach wird je nach nach gewählter Installation `sudo docker-compose up -d` oder `sudo docker compose up -d` (keine Bindestrich zwischen docker und compose) eingegeben, um die Standard Konfiguration `docker-compose.yml` zu starten. Compose erstellt dann die gewünschten Container mit den angegeben Optionen. Sollten die Container bereits mit dieser Compose Konfiguration erstellt worden sein, so werden die Container in dieser neu erstellt, dessen Konfiguration geändert wurde.

.env Name über CLI ändern

Wenn die gewünschte .env Datei nicht im selben Ordner liegt wie die Compose Datei oder sogar einen anderen Namen hat, so kann diese auch beim *docker compose up* über die CLI mit angegeben werden. Dadurch wird die Standard *.env* überschrieben. Hierzu einfach folgenden Befehl anpassen, alle anderen Einstellungen bleiben wie im vorherigen Abschnitt beschrieben.

```
sudo docker compose -env-file /pfad/zur/.webserver.env up -d
```

Environment Variablen mit beliebigen Dateien

Wenn Variablen über mehrere Compose Konfigurationen hinweg gleich sind wie beispielsweise der Name einer Domäne, dann kann auch eine zentrale oder mehrere zentrale Dateien für Variablen erstellt und verwiesen werden. Ein Nachteil gegenüber der *.env* Datei ist, dass diese Dateien bei jedem Dienst hinterlegt werden müssen. Außerdem lassen sich hierüber nur Environment Variablen setzen, d. h. die oben gezeigten Anpassungen sind nicht möglich.

Um die Variablen unter Environment (in diesem Beispiel nicht vorhanden) zu setzen, kann die Compose Datei wie folgt angepasst werden.

```
services:
  webserver:
    image: nginx:latest
    container_name: webserver
    hostname: webserver
    ports:
      - 80:80
```

```
    env_file:
      - .webserver.env
```

Docker Daemon Konfiguration

Die Standardeinstellungen von Docker lassen sich anpassen, indem die Konfiguration unter */etc/docker/daemon.json* erstellt bzw. angepasst wird. Standardmäßig ist diese Datei nicht vorhanden, damit nutzt Docker die Standardeinstellungen. Wird nun in dieser JSON-Datei z. B. ein eigener Adress-Pool für die Vergabe von IP-Adressen definiert, so wird diese Einstellung überschrieben. Alle anderen Standards bleiben bestehen.

Wenn eine Docker Instanz völlig verbastelt ist und wieder die Standardeinstellungen erhalten soll, kann die Datei auch einfach wieder gelöscht werden.

> Damit die Einstellungen wirksam werden, muss der Docker Daemon neu gestartet werden.

Im folgenden werden einige Einstellungen vorgestellt. Weitere Einstellungen können in der offiziellen Docker Dokumentation nachgelesen werden: Docker daemon configuration overview

IP-Adressen Pool für Docker Netzwerke

Früher oder später kann es zu Konflikten kommen, wenn ein oder mehrere Docker Server in einem großen Netzwerk mit anderen Servern und vor allem anderen Subnetzen arbeiten. Denn die Subnetze werden von Docker nicht bearbeitet, sondern nur das Netzwerk, in dem der Docker Host arbeitet, wird nicht für die internen Netze verwendet. Sobald Docker ein Netzwerk erstellt, dessen IP-Bereiche mit einem externen Subnetz überlappen, mit welchem der Docker Host aber in Kontakt steht, dann ist die Kommunikation gestört. Aus diesem Grund bietet es sich an, die Vergabe der Netze z. B. mit einer Whitelist zu steuern.

Die folgende Konfiguration in würde z. B. nur IP-Adressen aus dem Raum 172.16.0.0 - 172.31.255.255 vergeben. Dabei umfasst jedes Netzwerk 254 nutzbare IP-Adressen, was über die Size festgelegt wird. Für die Berechnung der Subnetze kann z. B. die folgende Webseite verwendet werden: IP Subnet Calculator

```
{
  "default-address-pools" : [
    {
      "base" : "172.17.0.0/12",
      "size" : 24
```

```
    }
  ]
}
```

Container automatisch aktualisieren

Wer fleißig Container mit Docker bereitstellt, wird schnell feststellen, dass es sehr aufwendig ist, alle Container aktuell zu halten. Zum Glück gibt es Watchtower, das selbst als Container bereitgestellt wird und alle anderen Container aktuell hält. Indem der Docker Socket des Hosts an den Container weitergereicht wird, kann Watchtower andere Container überwachen, die Images auf Updates prüfen und diese dann einspielen.

Für eine minimale Installation kann folgende Docker Compose Konfiguration verwendet werden. Jeden Tag um 01:00 Uhr werden alle Containern geprüft und bei einem verfügbaren Update dieses heruntergeladen, der Container gelöscht und mit dem neuen Image gestartet. Zusätzlich werden alle alten Images gelöscht, um nicht Gefahr zu laufen, dass der Speicherplatz vollgemüllt wird.

```
services:
  watchtower:
    image: containrrr/watchtower
    container_name: watchtower
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    environment:
      - WATCHTOWER_CLEANUP=true
      - "WATCHTOWER_SCHEDULE=0 0 1 * * *"
      - TZ=Europe/Berlin
    restart: unless-stopped
```

Um die Docker Compose Konfiguration auszuführen, kann am besten in das Verzeichnis der YAML Datei gewechselt werden. Danach wird je nach nach gewählter Installation `sudo docker-compose up -d` oder `sudo docker compose up -d` (keine Bindestrich zwischen docker und compose) eingegeben, um die Standard Konfiguration `docker-compose.yml` zu starten. Compose erstellt dann die gewünschten Container mit den angegeben Optionen. Sollten die Container bereits mit dieser Compose Konfiguration erstellt worden sein, so werden die Container in dieser neu erstellt, dessen Konfiguration geändert wurde.

> Nun werden alle Container automatisch überwacht. Wer zusätzliche Anforderungen hat wie z. B. Benachrichtigungen (per Mail) oder nur einige Container überwachen lassen möchte, kann in der Liste der Argumente in der offiziellen Dokumentation weitere Einstellungsoptionen finden:

Watchtower Argumente