

Docker

- [Docker Image aus einem Dockerfile erstellen](#)
- [Docker und Docker-Compose installieren](#)
- [MariaDB mit docker-compose installieren](#)
- [Monitoring mit Uptime-Kuma und docker-compose installieren](#)
- [Alle laufenden Container neustarten](#)
- [Docker Container im Image am laufen halten](#)

Docker Image aus einem Dockerfile erstellen

Einleitung

In diesem Beitrag erkläre ich kurz wie du das einem Dockerfile ein Image erstellen kannst. Dieses kannst du dann weiter publizieren und verwenden. Docker ist kurz gesagt eine Software, die sogenannte **Container** verwendet. Mit diesen können abgeschottete Systeme erstellt werden.

Diese teilen sich im Gegensatz zu virtuellen Maschinen den Kernel mit dem Host und bauen Ihre Layer auf. In den mehreren Layern befinden sich dann die installierten Programme und Abhängigkeiten.

Docker unterstützt nur einen Linux Kernel. Wenn du Docker auf einem Windows System installierst, lässt Docker eine Linux Maschine auf dem Host laufen. Die Container greifen dann auf den Linux Kernel im Windows System zu.

In Windows:

Wenn du in Windows ein Docker-Image erstellen möchtest, öffnest du die PowerShell als **Administrator**. Nun navigierst du in das Verzeichnis, in dem sich das **Dockerfile** befindet.

Jetzt verwendest du den **docker build** Befehl. Die Syntax dieses Befehls ist folgend aufgebaut:

```
docker build [parameter] .
```

Das Image wird jetzt erstellt und lokal abgelegt. Du kannst dieses lokal jetzt verwenden.

Am besten gibst du gleich einen Tag für das Image mit. Dann kannst du das Image leichter wieder finden.

```
docker build -t phillipunzen/apache-php80:latest .
```

In Linux:

Wenn du in Linux ein Docker-Image erstellen möchtest, öffnest du das Terminal. Jetzt navigierst du in das Verzeichnis, in dem sich das **Dockerfile** befindet.

Jetzt verwendest du den **docker build** Befehl. Die Syntax dieses Befehls ist folgend aufgebaut:

```
docker build [parameter] .
```

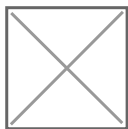
Das Image wird jetzt erstellt und lokal abgelegt.

Am besten gibst du gleich einen Tag für das Image mit. Dann kannst du das Image leichter wieder finden.

```
docker build -t phillipunzen/apache-php80:latest .
```

Parameter

Wenn du ein Docker-Image erstellst, kannst du den **Build Prozess** mit Parametern ein bisschen modifizieren. In dieser Tabelle findest du die Parameter, die du verwenden kannst.



Docker und Docker-Compose installieren

Einleitung

Mit Docker kannst du Anwendungen Containerisieren. Dies bedeutet, dass du auf einem Server auf Anwendungsebene Prozesse voneinander trennen kannst. So kannst du steuern, wer mit wem kommunizieren darf oder ob diese überhaupt von außen erreichbar sein dürfen. So kannst du z.B. mehrere Web-Server auf einem Server installieren.

Installation von Docker

Als Erstes installieren wir Docker. Dies brauchen wir, um erstmal Container zu starten.

Im ersten Schritt aktualisieren wir die Paketquellen und installieren notwendige Pakete

```
sudo apt-get update && \  
sudo apt-get install \  
    ca-certificates \  
    curl \  
    gnupg \  
    lsb-release -y
```

Im zweiten Schritt fügen wir den offiziellen Docker GPG Key hinzu.

```
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o  
/usr/share/keyrings/docker-archive-keyring.gpg
```

Im dritten Schritt setzen wir das benötigte Repository auf die **Stable** Version.

```
echo \  
    "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-  
keyring.gpg] https://download.docker.com/linux/debian \  
    $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Im letzten Schritt aktualisieren wir wieder die Paketquellen und installieren die Docker Pakete.

```
sudo apt-get update && \  
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Jetzt können wir überprüfen, ob die Docker Installation erfolgreich war

```
sudo docker -v
```

Installation von Docker-Compose

Jetzt installieren wir Docker-Compose. Mit Docker-Compose können wir vorab Konfigurationsdateien für Docker Container erstellen, aus denen wir dann starten. Wir können dann auch ganze Applikationen zu einem Stack zusammenfassen, die quasi als eine Einheit gestartet und gestoppt werden.

Im ersten Schritt laden wir das Skript herunter und verschieben es in das Docker Verzeichnis.

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Als Nächstes setzen wir die Berechtigung, um die Befehle ausführen zu können

```
sudo chmod +x /usr/local/bin/docker-compose
```

Falls du dann keine docker-compose Befehle absetzen kannst, führe den unten stehenden Code aus

```
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

Schnellinstallation

Falls wir **Docker** und **Docker-Compose** schnell installieren möchten, können wir den folgenden Befehl eingeben. Dann wird Docker komplett einmal installiert.

```
apt update && apt upgrade -y && apt install sudo -y
apt-get install ca-certificates curl gnupg -y
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg
echo "deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/debian "$(cat /etc/os-release && echo "$VERSION_CODENAME")"
stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-
compose-plugin -y
```


MariaDB mit docker-compose installieren

Einleitung

Wenn wir uns mit vielen Daten beschäftigen, die gespeichert werden sollen, werden relativ schnell Datenbanken verwendet. Datenbanken haben die Eigenschaft, dass diese sehr effizient mit vielen Daten umgehen können. Dazu kommt noch, dass mehrere Benutzer gleichzeitig auf die Daten zugreifen können. Datenbanken sind in fast jeder Firma und auch in jeglicher Software zu finden.

Installation

Zuerst müssen wir uns mit unserem Linux Server verbinden. Dies können wir über SSH / Telnet oder über eine serielle Verbindung machen.

Dann müssen wir sicherstellen, dass **docker** und **docker-compose** auf dem Server installiert ist. Falls nicht, habe ich in diesem Artikel beschrieben, wie man die Installation vornimmt: [Docker und Docker-Compose installieren](#)

Im nächsten Schritt erstellen wir in dem Ordner unserer Wahl, die **docker-compose.yml** und öffnen diese mit dem Editor unserer Wahl und fügen dort den folgenden Inhalt ein. Wir verändern daraufhin nur noch das Root Kennwort auf ein entsprechend sicheres Kennwort.

Ich erstelle für alle Services separate Ordner in denen die Daten der Container gespeichert werden können.

```
version: '3.1'

services:
  db:
    image: mariadb:latest
    container_name: db
    restart: always
    environment:
      MARIADB_ROOT_PASSWORD: <password>
    volumes:
      - ./db-data:/var/lib/mysql
    ports:
```

Im nächsten Schritt starten wir den Container. Es werden dann die Daten von der Datenbank in dem lokalen Verzeichnis **db-data** angelegt und mit dem Verzeichnis im Container gemountet. Dadurch stellen wir sicher, dass die Daten auch nach einem Neustart noch vorhanden sind. Im Folgenden wird dann auch der *root* Benutzer angelegt und mit dem angegebenen Kennwort versehen.

```
docker-compose up -d
```

Sobald der Container gestartet ist, das erkennen wir an dem **done** welches erscheint, sobald der Container gestartet ist, können wir uns in den Container hineinschalten um die entsprechenden Datenbanken und Benutzer anzulegen.

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

Wir sind dann als *root* in dem Container angemeldet und können jetzt die MySQL-Shell öffnen.

```
mysql -u root -p
```

Und wir geben in der Aufforderung dann unser festgelegtes *root* Kennwort ein. Wir können dann nach erfolgter Anmeldung unsere Datenbanken und Benutzer anlegen.

Datenbank und Benutzer direkt anlegen lassen

Sobald wir eine Datenbank mit einem Benutzer direkt bei der Initialisierung des Containers automatisch anlegen lassen wollen, können wir folgende **docker-compose.yml** verwenden. Wir müssen nur den Namen der Datenbank, des Benutzers und das Kennwort des Benutzer entsprechend anpassen.

Der Benutzer erhält dann alle Berechtigungen nur auf die eine angelegte Datenbank

```
version: '3.1'

services:
  db:
    image: mariadb:latest
    container_name: db
    restart: always
    environment:
      - MARIADB_ROOT_PASSWORD: <password>
```


- MARIADB_DATABASE: <datenbank>
- MARIADB_USER: <benutzer>
- MARIADB_PASSWORD: <passwort>

volumes:

- ./db-data:/var/lib/mysql

ports:

- 3306:3306

Monitoring mit Uptime-Kuma und docker-compose installieren

Einleitung

Zuverlässigkeit ist bei dem Angebot von Server Diensten das A & O. Es gibt Tools, die unterstützen uns bei der Überwachung der Dienste. Dazu gehört Uptime-Kuma. Uptime-Kuma ist ein kleines Programm, das verschiedene Dienste auf deren Erreichbarkeit überprüft.

Der Sinn von Überwachungslösungen ist der, dass man informiert wird, sobald etwas ausfällt. Und diese Aufgabe kann Uptime-Kuma auch übernehmen. Uptime-Kuma hat ein paar Integrationen, mit denen man bei Ausfällen informiert werden kann.

Installation

Im ersten Schritt müssen wir uns mit unserem Server per SSH / Telnet / Serielles Kabel verbinden. Im nächsten Schritt stellen wir sicher das **docker** und **docker-compose** installiert sind.

Jetzt wechseln wir in einen Ordner unserer Wahl und erstellen eine Datei **docker-compose.yml** und öffnen diese mit dem Editor unserer Wahl und fügen dort folgenden Inhalt ein. Wir können dort noch den Port ändern, mit dem wir das Web Interface öffnen können. Dazu müssen wir dann unter **Ports:** die Zahl nach dem - und vor dem : ändern.

```
version: '3.1'

services:
  uptimekuma:
    image: louislam/uptime-kuma:latest
    restart: always
    container_name: uptime-kuma
    volumes:
      - ./kuma-data:/app/data
    ports:
      - 3001:3001
```

Als nächstes starten wir den Container und initialisieren damit die Anwendung.

```
docker-compose up -d
```

Wenn wir jetzt einige Zeit warten können wir die IP-Adresse unseres Servers mit dem entsprechenden Port im Browser angeben und legen uns dann ein Administrator Konto an. Wir können dort dann jetzt unsere Services hinzufügen, die wir überwachen wollen.

Alle laufenden Container neustarten

Einleitung

In diesem Beitrag erkläre ich kurz, wie wir mit einem Befehl alle laufenden **Docker Container** neu starten können. Dies kann manchmal hilfreich sein, wenn Updates eingespielt werden und der Neustart möglichst schnell durchgeführt werden soll.

Container neu starten

Um die Container neu starten, müssen wir nur einmal den folgenden Befehl ausführen. Es werden beim Ausführen, nach und nach die entsprechenden Container ID's angezeigt, von dem Container, welcher gerade neu gestartet wird.

```
docker restart $(docker ps -q)
```

Docker Container im Image am laufen halten

Einleitung

Wenn wir eigene **Dockerimages** erstellen, kann es vorkommen, dass der Container nach einmaligen Ausführen sofort beendet wird. Dies kann ärgerlich sein, wenn auf dem Container ein Webserver oder ähnliches läuft. Damit der Container nach Ausführung des Codes, noch weiter läuft, müssen wir lediglich nur einen Absatz im Dockerfile hinzufügen. Sobald wir das Image dann neu bauen lassen, und dieses dann ausführen, sollte der Container nicht abstürzen.

Container laufen lassen

Im **Dockerfile** müssen wir nur den nachstehenden Code einfügen.

```
CMD tail -f /dev/null
```

Wenn wir jetzt das **Dockerimage** neu erstellen lassen, mit `docker build` wird beim Ausführen des Containers, der Code im **CMD Teil** ausgeführt. Mit dem Befehl produzieren wir eine Endlos-Schleife, wodurch unser Container nicht automatisch herunterfährt.