Docker

- Erstellen von Docker-Images
- Installing Docker, Docker Compose, and Portainer on Debian

Erstellen von Docker-Images

Container-Images erstellen in Docker (Tutorial für Anfänger)

Im ersten Tutorial zu Docker haben wir besprochen, wie du Container basierend auf bereits existierenden Images aus der Docker-Registry starten kannst. Dieses kurze Tutorial soll dir zeigen, wie du ein Image, passend zu deinen individuellen Anforderungen, erstellen kannst.

In diesem Beispiel-Fall wird der Container eine statische HTML-Website mit Nginx (einem Web-Server) ausführen. Der Name des Computers (oder Servers), auf dem Docker läuft, wird ganz einfach nur "Docker" genannt. Wenn du auf einen der Dienste zugreifen möchtest, dann benutze docker anstelle von localhost oder 0.0.0.0.0.

(Du benötigst kein Vorwissen zu Docker. Solltest du jedoch eine kurze Anleitung zu den Basics durchlesen wollen, findest du hier den Artikel "Docker lernen für Anfänger")

Wie funktionieren Docker-Images?

Docker-Images werden mithilfe eines Dockerfiles erstellt. Ein Dockerfile definiert alle Schritte, die nötig sind, um ein Docker-Image zu erstellen. Deine Anwendung wird dabei so konfiguriert, dass sie bereit ist als Container gestartet zu werden.

Das eigentliche Image enthält alle Dateien, vom Betriebssystem bis hin zur Konfiguration und allen Dependencies, die du benötigst, damit deine Anwendung fehlerfrei läuft.

Alle Dateien in dem Image parat zu haben hat den Vorteil, dass du deine Umgebungen überallhin migrieren kannst. Es stellt sicher, dass deine Anwendung überall funktioniert, wo du sie ausführst.

Das Dockerfile ermöglicht es, dass Images zusammengesetzt werden können, so dass Benutzer bestehende Images erweitern können. Das hat den Vorteil, dass du deine Images nicht jedes Mal von Grund auf neu bauen musst, sobald sich kleine Änderungen ergeben. Wenn du auf einem

bestehenden Image aufbaust, brauchst du nur die Schritte zur Einrichtung deiner Anwendung zu definieren. Die Basis-Images können bspw. Betriebssystem-Installationen oder bereits konfigurierte Systeme sein, die lediglich einige zusätzliche Anpassungen benötigen.

1. Ein Base-Image erstellen

Alle Docker-Images beginnen mit einem Base-Image.

Ein Base-Image ist identisch mit den Images aus der Docker-Registry, die zum Starten von Containern verwendet werden. Zusammen mit dem Image-Namen können wir auch ein Image-Tag angeben. Dadurch kannst du z.B. angeben, welche Version eines Programmes installiert wird. Wird keine Angabe zur Version gemacht, wird standardmäßig die aktuellste Version benutzt.

Diese Base-Images werden als Grundlage für deine zusätzlichen Änderungen verwendet. In diesem Fall benötigen wir zum Beispiel eine konfigurierte und lauffähige Nginx-Version, bevor wir die statischen HTML-Dateien bereitstellen können. Wir benutzten daher Nginx als Base-Image.

Dockerfile's sind einfache Textdateien, die pro Zeile einen Befehl enthalten. Damit du ein Base-Image definieren kannst, benutzt du folgenden Befehl:

```
FROM nginx
```

Da wir Nginx benötigen, sollte unsere erste Zeile im Dockerfile wie folgt aussehen:

```
FROM nginx
```

Schreibe diese Zeile in dein Dockerfile und speichere die Datei.

Hinweis zur Versionierung von Images:

Es ist so schön einfach :latest-Tag zu verwenden, um die aktuelle Image-Version zu erhalten. Jedoch ist hier Vorsicht geboten. Nicht immer bekommst du die Version des Image, die auf deine Anwendung abgestimmt ist. Ich empfehle dir daher immer die spezifische Versionsnummer als Tag anzugeben.

Sobald du eine Aktualisierung vorgenommen hast, solltest du dein Dockerfile auf die entsprechende Versionsnummer abändern.

Ja, das ist mehr Aufwand, aber es schützt auch vor Kopfschmerzen durch inkompatible Versionen!

2. Befehle ausführen

Wenn du das Base-Image angegeben hast, musst du weitere Befehle definieren, um dein Image zu konfigurieren. Hier gibt es mehrere Befehle, die dir dabei helfen z.B. RUN und COPY.

Hier eine kurze Erklärung zu RUN und COPY:

```
RUN <Befehl>
```

Der RUN-Befehl erlaubt es dir, jeden Befehl wie in der Kommandozeile auszuführen. Damit kannst du bspw. verschiedene Anwendungspakete installieren oder einen Build-Befehl ausführen.

Die Ergebnisse des RUN-Befehls werden im Image gespeichert, daher ist es wichtig, keine unnötigen oder temporären Dateien (z.B. Cache-Dateien) zu hinterlassen, da diese sonst in das Image eingebunden werden.

```
COPY <Quelle> <Ziel>
```

Mit COPY kannst du Dateien aus dem Verzeichnis, in dem sich dein Dockerfile befindet, in das Image des Containers kopieren. Das ist besonders nützlich für Quellcode und Assets, die du in deinem Container bereitstellen möchtest.

Hier eine einfache Übung zum COPY-Befehl:

Um etwas Praxis-Erfahrung zu bekommen führe Folgendes aus.

Erstelle eine .html Datei z.B. index.html und fülle sie mit etwas Inhalt. Diese Datei wollen wir im Anschluss in unserem Container bereitstellen. Benutze in der nächsten Zeile den Befehl COPY Befehl, um die index.html in das Nginx-Verzeichnis /usr/share/nginx/html zu kopieren. Hast du das gemacht? Gut! Dann geht es jetzt weiter mit der Freischaltung der Ports.

3. Ports freigeben (öffnen)

Wenn die Dateien in unser Image kopiert wurden und die Dependencies heruntergeladen sind, musst du noch den Port definieren. Genauer gesagt musst du spezifizieren, auf welchem Port die Anwendung zugänglich sein wird.

Dafür benutzt du den Befehl EXPOSE . Wir teilen Docker mit, welche Ports offen sein sollen und an welche Ports Docker gebunden werden kann. Es ist auch möglich gleich mehrere Ports einem einzigen Befehl zu definieren.

Docker-Beispiel, um mehrere Ports freizugeben:

```
EXPOSE 80
```

Wenn du diesen Schritt nachmachen möchtest, dann gib mit dem obigen Befehl den Port 80 frei. Der Port 80 muss geöffnet sein, damit unser Webserver (Nginx) verfüg

Installing Docker, Docker Compose, and Portainer on Debian

In this article, we will walk through a bash script that automates the process of installing Docker, Docker Compose, and Portainer on a Debian-based system. Docker is a popular platform for developing, shipping, and running applications using containerization. Docker Compose is a tool for defining and running multi-container Docker applications. Portainer is a lightweight management UI for Docker environments.

Let's dive into the script:

```
#!/bin/bash

# Function to uninstall old Docker version if found
uninstall_old_docker() {
    sudo apt-get purge -y docker-ce docker-ce-cli containerd.io
    sudo rm -rf /etc/apt/sources.list.d/docker.list
}

# Check if curl is installed
if ! [ -x "$(command -v curl)" ]; then
    echo "Installing curl..."
    sudo apt-get update
    sudo apt-get install -y curl
fi

# Check if Docker is already installed
if [ -x "$(command -v docker)" ]; then
    echo "Old Docker installation found. Uninstalling..."
    uninstall_old_docker
    echo "Old Docker installation removed."
fi
```

```
# Install Docker dependencies
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates gnupg-agent software-properties-common

# Add Docker’s official GPG key
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

# Add Docker repository
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

# Install Docker Engine
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io

# Install Docker Compose
if ! [ -x "$(command -v docker-compose)" ]; then
    echo "Installing Docker Compose..."
    sudo curl -L "https://github.com/docker/compose/releases/download/1.29.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
    sudo chmod +x /usr/local/bin/docker-compose
else
    echo "Docker Compose is already installed."
fi

# Install Portainer
docker volume create portainer_data
docker run -d -p 8000:8000 -p 9443:9443 --name portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ee:latest
```

Steps Explained:

1. **Uninstall Old Docker (if present):** This script first checks if an older version of Docker is installed. If found, it uninstalls it.

2. **Install curl:** It checks if `curl` is installed. If not, it installs it. `curl` is required for downloading files from the internet.
3. **Install Docker Dependencies:** Installs necessary dependencies for Docker.
4. **Add Docker GPG Key and Repository:** Adds Docker's official GPG key for package verification and adds the Docker repository to the system's package sources.
5. **Install Docker Engine:** Updates package lists and installs Docker Engine, Docker CLI, and Containerd.
6. **Install Docker Compose:** Checks if Docker Compose is installed. If not, it downloads the latest version and sets the necessary permissions.
7. **Install Portainer:** Creates a Docker volume for Portainer data, then runs Portainer as a Docker container with the necessary configurations.

Conclusion:

This script automates the installation process of Docker, Docker Compose, and Portainer on Debian-based systems, making it easier to set up a containerized environment for development or production use. Make sure to review and understand the script before executing it on your system.