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.

(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 Dockerfile's 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 <image-name>:<tag>
```

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

```
FROM nginx:1.11-alpine
```

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 entsprechend 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.

1. Erstelle eine .html Datei z.B. index.html und fülle sie mit etwas Inhalt. Diese Datei wollen wir im Anschluss in unserem Container bereitstellen.
2. 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 <port>* . 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 80 433
EXPOSE 7000-8000
```

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) verfügbar ist. Füge die Zeile (EXPOSE 80) deinem Dockerfile hinzu, um diesen Schritt auszuführen und speichere deine Datei.

4. Anwendung per Befehl starten

Fassen wir kurz zusammen was bis hierhin passiert ist. Du hast ein Docker-Image konfiguriert und festgelegt, auf welchen Port deine Anwendung zugreifen muss. Jetzt ist es an der Zeit den Befehl zu definieren, der deine Anwendung startet.

Der CMD-Befehl in einem Dockerfile definiert den Standardbefehl, der ausgeführt wird sobald dein Container gestartet wird. Wenn der Befehl zusätzliche Argumente benötigt, solltest du dafür ein Array verwenden.

CMD Befehl mit Argumenten als Array:

[„cmd“, „-a“, „arga value“, „-b“, „argb-value“] Die obige Zeile würde dann den Befehl **„cmd -a „arga value“ -b argb-value“** ausführen.

Jetzt bist du wieder gefragt: Der Befehl zum Ausführen von NGINX lautet nginx -g daemon off;.

Setze diesen Befehl als Standardbefehl in deinem Dockerfile.

5. Das Container-Image erstellen

Nachdem du dein Dockerfile definiert hast, musst du es mit docker build in ein Image umwandeln. Der Befehl build nimmt ein Verzeichnis mit dem Dockerfile auf, führt die Schritte aus und speichert das Image in deiner lokalen Docker Engine. Wenn einer der Schritte aufgrund eines Fehlers fehlschlägt, wird die Container-Erstellung abgebrochen.

Willst du diesen Schritt nachmachen?

Benutze den Befehl **docker build**, um das Image zu bauen. Du kannst dem Image einen lesbaren Namen geben, indem du das Argument -t <name> verwendest z.B.

```
docker build -t mein-image
```

Tipp: Mit dem Befehl docker images siehst du eine Liste deiner Images. Damit kannst du prüfen, ob der Build-Vorgang erfolgreich beendet wurde.

6. Dein neues Docker-Image starten. Wohooo!

Wenn das Image erfolgreich erstellt wurde, kannst du den Container nun auf die gleiche Weise starten, wie du es schon aus vorherigen Tutorials kennst. Solltest du noch nicht wissen, wie du ein Image startest, findest du nachfolgend den Befehl dafür.

Du möchtest diesen Schritt nachmachen? Starte eine Instanz deines brandneuen Images, indem du die ID des Build oder den Shortname (Nice-Name) verwendest, den du im letzten Schritt zugewiesen hast.

Der Webserver Nginx ist so konzipiert, dass er als Hintergrund-Prozess läuft. Du solltest daher die Option -d verwenden. Beim Ausführen solltest du als Argument ebenso den Port 80 an deinen Container binden (p 80:80).

Klingt kompliziert?

Hier der fertige docker run Befehl:

```
docker run -d -p 80:80 <Image ID oder Shortname>
```

Da ich mein Image im letzten Schritt „mein-image“ genannt habe würde der Befehl so aussehen:

```
docker run -d -p 80:80 mein-image
```

Du kannst überprüfen, ob dein Container läuft, indem du folgenden Befehl benutzt: docker ps