Docker

Bei Docker handelt es sich um ein open-source (Apache 2.0 Lizenz) System um sogenannte Container zu betreiben und zu verwalten. Dies dient im besonderen dazu Code effizient in kurzer Zeit auf diversen Systemen lauffähig zu machen. (Turnbull, 2019, S.7-8) So hilft Docker eine Applikation von der Infrastruktur zu separieren und so zu isolieren (Docker Inc., o.J.a).

Funktionsweise und Unterschied zu virtuellen Maschinen

Das Docker-System besteht aus drei essentiellen Teilen:

- Docker Images
- Docker Container
- Docker Engine

Die Docker Images sind die Bauanleitung für die Docker Container. Die Images enthalten dabei eine Schritt für Schritt-Anleitung, wie ein Container generiert beziehungsweise aufgebaut werden muss. Als Beispiel nennt Turnbull:

1. Füge eine Datei hinzu.
2. Führe einen Befehl aus.
3. Öffne einen Port.

Wie zusehen ist, sind diese Images sehr trivial aufgebaut. Das macht es besonders einfach sie zu teilen und zu modifizieren. (Turnbull, 2019, S.12) Ein weiterer Vorteil der Images ist es, dass sie aufeinander basieren können. So ist es beispielsweise möglich ein eigenes Image zu erstellen, welches auf dem Image von Ubuntu basiert. So ein Image wird dann in einer Dockerfile erstellt. Jeder weitere hinzugefügte Befehl wird als eine neue Ebene gesehen. Ein Vorteil, welcher Docker von anderen Container-Systemen unterscheidet, ist, dass Docker ein Image nicht komplett neu baut, wenn in der Dockerfile sich eine Ebene verändert. Es werden lediglich die veränderten Ebenen erneuert. Das macht, laut Docker Inc., die Images so klein und schnell. Die meisten der allgemein bekannten Docker Images werden in Registries veröffentlicht, wo sie für jeden zur Verfügung stehen. Eine der bekanntesten Registries ist Docker Hub. (Docker Inc., o.J.a) Die Docker Container werden zur Laufzeit von der Docker Engine mit Hilfe der Docker Images generiert. Da Docker Container standardisiert sind, sind sie mit anderen Container Umgebungen kompatibel. (Docker Inc., o.J.b) Auch das macht sie sehr portabel. Dies wird auch dadurch gefördert, dass die Container selber kein eigenes Betriebssystem und Kernel umfassen, da dies mit dem Host-System geteilt wird. Dies ist auch gleichzeitg einer der großen Unterschiede zu virtuellen Maschinen. Ein Container beinhaltet lediglich die Anwendung und deren Abhängigkeiten. So ist es auch möglich mehrere Container auf einem Host zu betreiben, wobei diese stets voneinander isoliert sind. Virtuelle Maschinen hingegen sind zwar auch voneinander isoliert, aber sie eignen sich besser dazu, physikalisch Hardware zu emulieren. Dafür enthält jede virtuelle Maschine ein eigenes Betriebssystem inklusive Kernel. Das macht sie deutlich ressourcenintensiver in der Bereitstellung, was sich auch auf die Perfomance negativ auswirkt. (Rad, Bhatti & Ahmadi, 2017) Auch wenn die Container isoliert vom Host-Betriebssystem arbeiten, bietet Docker die Möglichkeit, gewisse Teile des Containers zu öffnen und so eine einfachere Kommunikation zwischen Container und Host-Betriebssystem zu ermöglichen. Beispielsweise können Ports für einen Container freigegeben werden. Dies ist möglich, da jeder Container eine eigene Netzwerkschnittstelle besitzt. Die Docker Engine (später mehr dazu) kann diesen Port dann für das

Host-Betriebssystem auf einen anderen Port umleiten. Dies ermöglicht zum Beispiel zehn Container mit einer Anwendung, die Port 80 verwendet, zu betreiben und für das Host-Betriebssystem sind es dann die Ports 42001, 42002 und so weiter (siehe Abbildung HIER NOCH NE ABBILDUNG MACHEN). Jeder Container hat dann nach außen hin einen eigenen Port. In ihrem Container benutzt die Anwendung aber immer den selben Port. (Anderson, 2015, S.104f) Auch können für Docker Container sogenannte Volumes eingerichtet werden. Dies ist ein persistener Speicher für Daten, welcher nicht zwingend an einen Container gebunden sein muss. Auf diesen Speicher können sowohl mehrere Container als auch das Host-Betriebssystem direkt zu greifen. Dies wird beispielsweise häufig genutzt um Quellcode in den Container zu laden ohne das Image zu verändern oder um Ereignisprotokolle auch außerhalb des Containers verfügbar zu machen. Diese Funktion wird auch durch die Docker Engine ermöglicht. (Turnbull, 2019, S.113) Die Docker Engine, auch bekannt als Docker Daemon, stellt im Docker System quasi den Verwalter dar. Die Engine wird über eine RestAPI gesteuert. So können Images, Container, Netzwerke und Volumen verwaltet werden. Sie steuert auch die Kommunikation zwischen den laufenden Docker Containern und dem Host-Betriebssystem. Dazu zählen beispielsweise die Verteilung von Systemressourcen und der Zugriff auf Basisfunktionalität des Host-Betriebssystems. (Docker Inc., o.J.a)

Oftmals besteht eine Applikation nicht nur aus einer Anwendung. So können Webapplikationen beispielsweise einen Backend-Webserver und dazu noch eine oder mehrere Datenbanken enthalten. Dafür können natürlich mehrere Container einzeln über die Kommandozeile mit Hilfe der Docker Engine angelegt und verwaltet werden. Dies ist aber sehr aufwändig und auch schlecht replizierbar. Genau für solche Einsatzzwecke wurde das Tool Docker-Compose entwickelt. Docker-Compose ermöglicht es in einer Datei alle Services (eine Konfiguration eines Containers), Volumen, Netzwerke und Abhängigkeiten zu definieren. Diese Datei kann dann einfach und schnell geteilt werden, um eine Applikation in einer Entwicklungs-, aber auch in einer Produktionsumgebung lauffähig zu machen. Dabei können beispielsweise Portumleitungen oder Umgebungsvariablen schnell geändert werden, um die Applikation ihrer Umgebung anzupassen. (Smith, 2017)

Einsatzgebiete

Docker wird sowohl in der Entwicklung, als auch in der Produktion eingesetzt. Besonders von Vorteil ist, dass die Container eine standardisierte Umgebung, in welcher die Applikation betrieben wird, schaffen. Diese Umgebung ist dann gleich in der Entwicklung, beim Testen und in der Produktion. Dies verhindert das Auftreten von Fehlern, welche mit der Umgebung zusammenhängen. Auch ist es für Entwickler einfacher Aktualisierungen an einen Kunden zu übertragen, da nicht eine ganze Applikation aktualisiert werden muss, sondern nur ein Teil dieser. Dieser Ablauf wird Continuous Integration und Continuous Deployment genannt. (Docker Inc., o.J.a) Dies spart vor allem Zeit. Eine alternative Methode wäre es dies über virtuelle Maschinen aufzusetzen. Dabei würde jedoch alleine die Zeit, welche es benötigt eine solche virtuelle Maschine einzurichten, bei über 10 Minuten liegen. Weiterhin müsste dies für jede Umgebung einzeln getan werden. Sollte eine virtuelle Maschine fehlerhaft sein, muss diese natürlich auch wieder neu eingerichtet werden, was vor allem in einer Entwicklungsumgebung und der Testumgebung häufig vorkommt. Das andere Einsatzgebiet sind Umgebungen, in denen besonders viele Applikationen oder Instanzen einer Applikation betrieben werden. Würde dies über virtuelle Maschinen gelöst werden, dann würden mindestens zehn bis 15 % der verfügbaren Hardwareressourcen alleine durch den Hypervisor der virtuellen Maschinen verbraucht. Zusätzlich verbrauchen auch die virtuellen Maschinen noch mehr Ressourcen, als Docker Container. Für Unternehmen ist dies also ein einfacher Punkt Kosten zu sparen und gleichzeitig Hardwareressourcen effizienter zu verwenden. (Anderson, 2015, S.103)

From:
https://wiki.eolab.de/ - **HSRW EOLab Wiki**

Permanent link:
https://wiki.eolab.de/doku.php?id=user:jan001:ba:docker&rev=1612978307

Last update: **2021/08/24 17:34**