HOCHSCHULE HEILBRONN
TECHNIK WIRTSCHAFT INFORMATIK

Fallstudie Entwicklungswerkzeuge (281761)

Docker

Jerome Tagliaferri*

4. Dezember 2016

Eingereicht bei Paul Lajer

*190530, jtagliaf@stud.hs-heilbronn.de

Inhaltsverzeichnis

Abkürzungsverzeichnis iii

Abbildungsverzeichnis iv

Tabellenverzeichnis v

1 Einleitung 1
- 1.1 Motivation . . . 1
- 1.2 Ziel der Arbeit . . . 1
- 1.3 Vorgehensweise . . . 1

2 Docker Grundlagen 3
- 2.1 Was ist Docker ? Was versteht man darunter ? . . . 3
- 2.2 Die Geschichte, der Ursprung und Entwicklung von Docker . . . 3
- 2.3 Wie funktioniert Docker ? . . . 4
- 2.4 Welche Funktionen enthält Docker /Docker Workflow ? . . . 6
- 2.5 Vorteile und Nachteile von Docker . . . 9

3 Docker im Produktiveinsatz 10
- 3.1 Anwendungsbereiche - Wer nutzt Docker ? . . . 10
- 3.2 Tools - Kubernetes / Docker Swarm . . . 10
- 3.3 Alternativen und Unterschiede ? . . . 10
- 3.4 Wieso gerade Docker ? . . . 10
- 3.5 Weshalb kommt der Durchbruch erst jetzt ? . . . 10

4 Fazit - Ausblick 11

Literaturverzeichnis 12

Abkürzungsverzeichnis

ABK: ABKÜRZUNG

Abbildungsverzeichnis

2.1 VM vs Container 5
2.2 Treiber 6
2.3 Container-Images 7

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Technologien welche im Big Data und Cloud Umfeld entstanden und gewachsen sind, werden immer häufiger auch in anderen Bereichen eingesetzt. So entstand die Virtualisierung von Betriebssystemen als Grundlage des Cloud Computing um auf individuelle Wünsche und eine stetig schwankende Ressourcenverteilung zu reagieren. So wie diese Technology ihren Weg in viele weitere Bereiche geebnet hat, findet die Container Virtualisierung eine immer vielseitigere Anwendung. Getragen und weiterentwickelt von Branchengrößen wie Google oder IBM ist die Container Virtualisierung momentan eine der sich am schnellsten wachsenden Bereiche der Informatik. Sie verspricht eine noch stärkere flexibilität und automatisierung von Systemen und Ressourcen und ist deshalb ein relevantes Thema für jede Branche, welche mit Soft- und Hardwaresystemen in Kontakt kommt.

1.2 Ziel der Arbeit

Diese Ausarbeitung soll als Einstieg in den Bereich Container Virtualisierung dienen und dabei Docker als zentrale Komponente behandeln und vorstellen. Es soll ein umfassender Einblick in die Thematik und deren unterschiedliche herangehensweisen erörtert werden, wodurch eingeschätzt werden kann, inwieweit sich die Container Virtualisierung für individuelle Einsatzzwecke eignet.

1.3 Vorgehensweise

Um dieses Ziel zu erreichen, werden die Grundlagen in Form der Idee und grundlegenden Historie und deren Konzepte, welche die Basis von Docker beinhalten, erläutert. Daraufhin wird die Umsetzung dieser Konzepte und deren Erweiterungen in Docker betrachtet und daraus auf mögliche Vor- und Nachteile geschlossen. Aufbauend auf diesen Grundlagen, werden Anwendungsbereiche vorgestellt, in dennen Docker in einem produktiven Umfeld eingesetzt wird und welche Tools den massiven und verteilten Einsatz von Containern erleichtern. Mögliche Alternativen und deren Unterschiede, sowie ein Ausblick

in die Zukunft der Container Virtualisierung sollen diese Arbeit abrunden.

2 Docker Grundlagen

2.1 Was ist Docker ? Was versteht man darunter ?

Die Bezeichnung Docker, findet schon lange nicht nur im Cloud spezifischen Umfeld erwähnung. Der aufmerksame Nutzer stößst immer öfter schon bei der Installation von Anwendungen auf diesen Term. Ein Beispiel hierfür wäre das Test und Entwicklungswerkzeug Jenkins, welches nun an erster Stelle die Option eines Docker Containers als mögliche Installationsquelle zur Verfügung stellt. [3] Viele weitere Hersteller welche Werkzeuge anbieten, welche auf unterschiedlichste Bibliotheken und Anwendungen angewiesen sind, greifen immer öfter zu Docker. Hierbei stellt sich jedoch die Frage, was ist Docker ?
Auf der Offiziellen Seite wird diese Frage mit folgendem Satz beantwortet :

> [1] „*Docker is the world's leading software containerization platform*"

Da diese Aussage wenig bis keinen Informationsgehalt bietet, macht es sinn, sich zuerst einmal die Grundlegenden Konzepte und historische Entwicklung der Container Virtualisierung anzuschauen.

2.2 Die Geschichte, der Ursprung und Entwicklung von Docker

Wie so viele Entwicklungen im technischen Umfeld sind die Konzepte und ersten Umsetzungen der Container Virtualisierung schon weitaus älter als man vermuten sollte. Es begann alles mit der Idee Services untereinander und vom eigentlichen Host isolieren zu können. Dieses Konzept wurde erstmals im Jahre 1979 unter UNIX mit der funktion des "chrootümgesetzt, diese Funktion isolierte das Hauptverzeichnis eines Prozresse an einem neuen Ort.
Erst im Jahre 2000 wurde diese Funktionalität unter FreeBSD mit dem Namen Jails erneut aufgenommen und erweitert. Diesmal konnte nicht nur das Dateisystem isoliert werden, sondern auch dazugehörige Benutzer, Netzwerk und dazugehörige Prozesse.

[1] Vgl. https://www.docker.com/what-docker

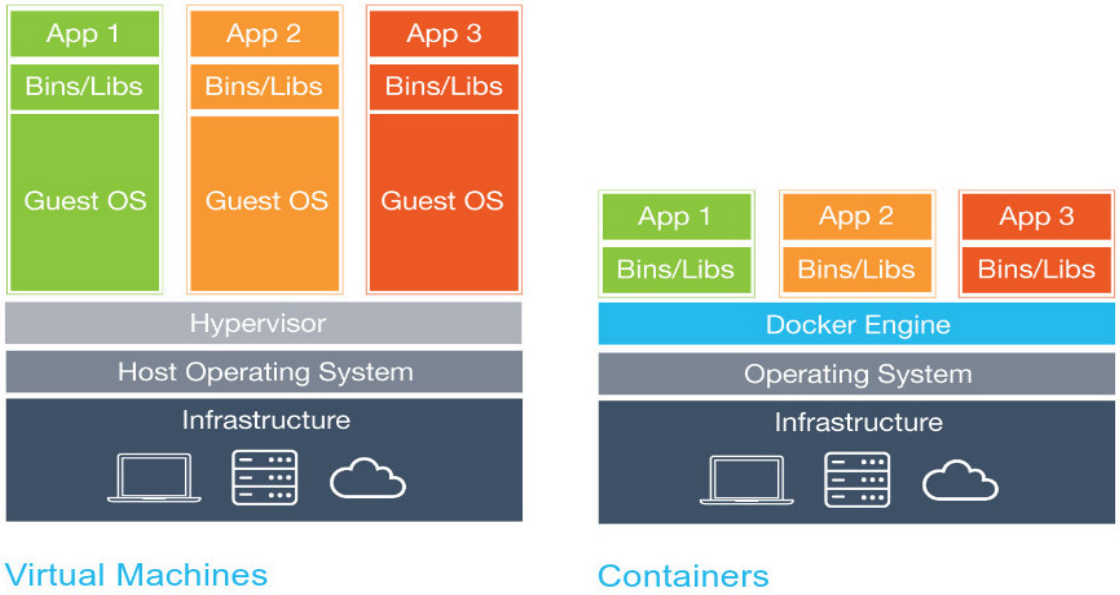
Über die nächsten acht Jahre können viele weitere Technologien gefunden werden, welche diese Funktionalitäten integrieren. Darunter Linux VServer, Oracle Solaris Zones, OpenVZ, Process Container (von Google entwickelt),ControlGroups im Linux Kernel und WPARS. 2008 entstand dann durch die Entwicklung bei IBM das Linux Containers project (LXC) welches die unterschiedlichsten Technologien im Container Umfeld zusammen brachte und somit die vollständigste Implementierung eines Linux Container Managements darstellte. Die besonderheit von LXC, bestand darin, dass es seine ressourcen komplett aus dem Linux Kernel bezog ohne zusätzliche Software zu benötigen. Am 15.03.2013 wurde Docker, durch den dotCloud Gründer Solomon Hykes zum ersten mal der Öffentlichkeit vorgestellt. Diese Vorstellung bestand nur aus einem fünf minütigen Vortrag auf der Python Entwickler Konferenz in Kalifornien. Verbreitete sich jedoch weltweit mit enormer Geschwindigkeit wodurch das Projekt schnellstmöglich auf GitHub offen gelegt wurde um die weitere Entwicklung voran zu treiben.

2.3 Wie funktioniert Docker ?

Abgesehen von der schnellen Verbreitung von Docker, war vielen Interessenten die eigenltiche Funktionsweise nicht vollständig bewusst. Docker versprach, ein Stück Software in ein komplettes Dateisystem zu verpacken, welches alles beeinhaltete was für das Ausführen dieser nötig ist. Dazu gehört der Quellcode, System Bibliotheken/Tools und die Ausführumgebung. Dies sorgte dafür, dass sich die Software immer gleich verhält egal auf welcher Umgebung sie ausgeführt wird. Hinzu kam, das Container aufgrund ihrere Größe und Isolation beliebig gestartet oder gestoppt werden konnten, ohne das Host System zu beeinflussen. Einige dieser Eigenschaften, werden oftmals sehr gerne mit denen Virtueller Maschinen verglichen, da es auf den ersten Blick sehr viele Ähnlichkeiten aufweist. Die Entwickler von Docker, distanzieren sich jedoch sehr klar von diesem Vergleich und machen darauf aufmerksam, dass Docker sehr viel mehr ist. Um jedoch die Grundlegenden funktionsweisen zu erläutern, macht es Sinn diesen Vergleich zu bemühen.

Wie in der Abbildung unschwer zu erkennen, liegt ein sehr wesentlicher Unterschied in der größe der Lösungen. Da eine Virtuelle Maschine über einen Hypervisor mit dem Hostsystem kommuniziert bringt es bei jeder Instanz ein komplettes Gast Betriebssystem mit sich. Dies führt dazu, dass nur wenige Instanzen auf einem Host ausgeführt werden können. Da jedoch jeder Docker Container den Kernel des Hostsystems nutzt, ist es möglich auf einer Maschine mehrere hundert Container parallel zu betreiben. Damit dies möglich wurde, nutzt Docker mehrere Funktionen des Linux Kernels wobei folgende

Abbildung 2.1: VM vs Container



Quelle: www.docker.com/what-docker

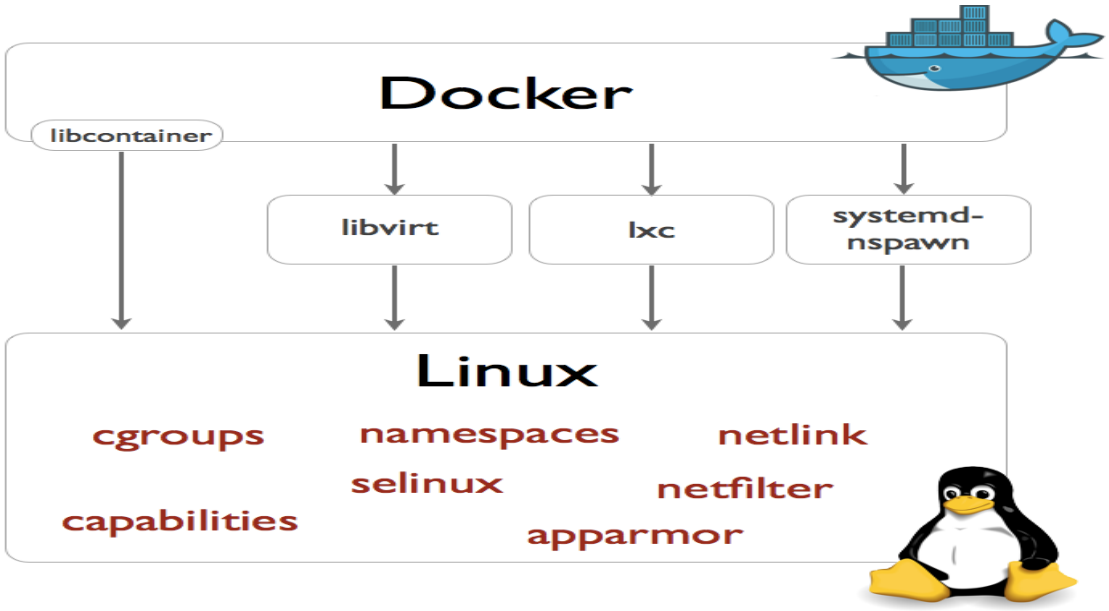
essentiell sind:

- CGroups : Limitiert und verwaltet die Ressourcen wie CPU , Arbeitsspeicher, Netzwerk und Festplattenzugriff welche einer Prozessgruppe zugewiesen werden können.

- Namespace isolation : Prozessgruppen sind nicht in der Lage die Ressourcen anderer Gruppen zu sehen.

Bis zur Version 0.9 war Docker von der LXC-Bibliothek abhängig, nutzte jedoch ab diesem Zeitpunkt eine eigens Entwickelte Bibliothek namens libcontainer. Diese neue Bibliothek wurde entwickelt um weitere Isolationstechnologien zu nutzen außerdem war Docker nun in der Lage alle nötigen Komponenten aus einer Hand zu bieten. Dies gab auch Microsoft die möglichkeit Docker unter Windows lauffähig zu machen da alle Komponenten veröffentlicht wurden und allgemein als Standard angesehen werden. Eine weitere sehr essentielle Komponente stellt UnionFS dar. Dies ist ein Dateisystem, welches die Fähigkeit besitzt unterschiedliche Schichten (Standorte) von Dateien zu bündeln und einem Prozess zur Verfügung zu stellen. In Docker wird dies genutzt, um nur neue oder veränderte Dateien dem Container Verfügbar zu machen um so die Geschwindigkeit zu erhöhen und beim Einsatz von großen Mengen an Containern Speicherplatz zu sparen.

Abbildung 2.2: Treiber



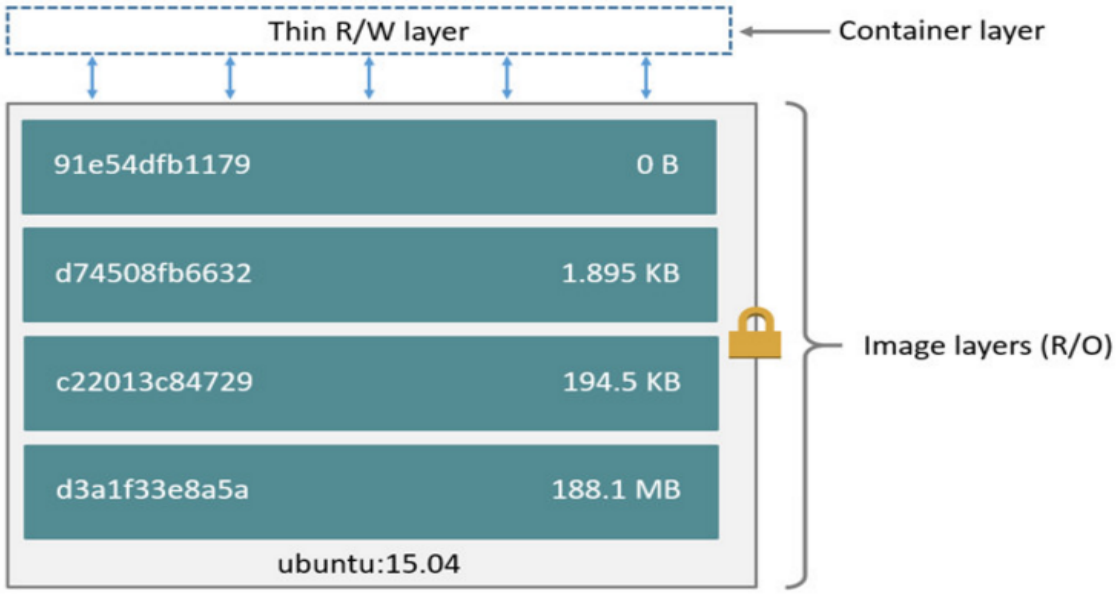
Quelle: http://jancorg.github.io/blog/2015/01/03/libcontainer-overview/

2.4 Welche Funktionen enthält Docker /Docker Workflow ?

Aufbauend auf diesen Grundlagen, soll nun eine mögliche Vorgehensweise erläutert werden, welche den Einsatz von Containern beschreibt. Zuerst sollten jedoch die einzelnen Komponenten vorgesetellt werden :

- **Docker daemon**: Docker nutzt eine Client-Server Architektur wobei der Daemon einen Serverseitigen Prozess darstellt, welcher die Aufgabe hat Container zu erstellen, auszuführen und zu verteilen.
- **Docker CLI**: Der Docker Client hat die Möglichkeit über eine REST API mit dem Docker daemon zu kommunizieren und Befehle zu übermitteln.
- **Docker Image Index**: Ein öffentliches oder Privates Verzeichnis über alle Docker Images welche vorhanden sind. Möchte man ein Image ausführen, wird meist zuerst lokal nach solchem gesucht, sollte der Prozess nicht fündig geworden sein, wird im Docker eigenen Repository DockerHub weiter gesucht, welche sehr viele spezielle Images anbietet.

Abbildung 2.3: Container-Images



Quelle: https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/

- **Docker Containers**: Ein Container, welcher sich meist aus mehreren Images zusammen stellt. Diese enthalten alle nötigen Bibliotheken/Tools und Quellcode um die Applikation auszuführen. Wenn ein solcher Container angelegt wird, wird wie in Abbildung 2.3 zu sehen ein neuer Layer angelegt. Dieser kann dann später alle Veränderungen aufnehmen.
- **Docker Images**: Stellt ein Dateisystem mit verschiedenen Parametern dar welches nur gelesen aber nicht beschrieben werden kann.
- **Dockerfiles**: Skripte welche den Bau eines Images vereinfachen und automatisieren.

Das zusammenspiel dieser Komponenten, sorgt dafür, dass die Erstellung , Ausführung und Administration von Docker Containern sehr schnell und einfach funktioniert. Dies soll nun anhand einer Webapplikation verdeutlicht werden. Hierführ gehen wir davon aus, dass Docker bereits installiert und wir eine simple "Hello World !"Django Applikation erstellt haben. Diese nutzt folgende Technologien : Python 3 mit Django und Gunicorn als Web Server.

1. Um die Erstellung des Containers zu erleichtern, wird im Hauptverzeichnis der Applikation eine requirements.txt angelegt welche unsere Bedingungen enthält.

In diesem Fall :

Django==1.9.4 gunicorn==19.6.0

2. Nun soll ein script erstellt werden, welches später die richtige Ausführung des Web Servers übernehmen soll. Hierzu wird eine start.sh Datei erstellt welche folgenden Inhalt besitzt :

#!/bin/bash
exec gunicorn helloworld.wsgi:application
–bind 0.0.0.0:8000
–workers 3

Über den exec Befehl wird hierbei Gunicorn mit unserer helloworld applikation gestartet, welche wir über die lokale Adresse 0.0.0.0 und den Port 8000 ansprechen wollen. Zur Sicherheit und verbesserten Bearbeitung von anfragen werden zusätzlich drei Ärbeiterërstellt.

3. Es kann damit begonnen werden eine Dockerfile im Hauptverzeichnis zu erstellen welche unterschiedliche Befehle beinhalten kann, um die Umgebung anzupassen.

Nutzt ein vorgefertigtes Python3 Image von DockerHub als Basis FROM python:3-onbuild

Kopiert den Inhalt des Verzeichnises an die angegebene Stelle innerhalb des Containers COPY . /usr/src

Öffnet den Port 8000 um eine Kommunikation zu ermöglichen EXPOSE 8000

führt das Skript des Web Servers aus CMD [üsr/src/start.sh"]

4. Das Image muss nun über folgenden Befehl gebaut werden :

sudo docker build -t helloworld .

Dies führt docker mit root rechten aus um innerhalb des momentanen Verzeichnisses die Dockerfile abzuarbeiten und einen Container mit dem namen helloworld zu erstellen.

5. Als letzter Schritt muss der Container über das Terminal gestartet werden :

sudo docker run -it -p 8000:8000 helloworld

Dies führt den Docker Container helloworld aus und verbindet sich über den Port 8000 mit dem Web Server.

Werden nun Änderungen an der Applikation vorgenommen müssen nur noch die letzten zwei Schritte ausgeführt werden. Hierdurch können Neuerungen an Software Systemen sehr schnell umgesetzt und in Betrieb genommen werden ohne mit Komplikationen konfrontiert zu werden. Dieses kleine Beispiel kann nicht alle Funktionalitäten von Docker umfassen, gibt aber einen ersten Einblick in die Arbeit mit Containern. Ansatzpunkte welche das dargestellte Beispiel erweitern , sind die Verlinkung von Containern

(sollte z.b. eine Datenbank hinzukommen) oder auch das erstellen von eigenen Basis Images welche dann veröffentlicht und verteilt werden können.

2.5 Vorteile und Nachteile von Docker

3 Docker im Produktiveinsatz

3.1 Anwendungsbereiche - Wer nutzt Docker ?

3.2 Tools - Kubernetes / Docker Swarm

3.3 Alternativen und Unterschiede ?

3.4 Wieso gerade Docker ?

3.5 Weshalb kommt der Durchbruch erst jetzt ?

4 Fazit - Ausblick

Im Schlusswort / Fazit („Was ich gesagt habe und was daraus folgt“) kann ein kurzer Rückblick auf das Thema erfolgen. Wenn das Thema dies zulässt, können auch Zukunftsperspektiven aufgezeigt und höchst subjektive Bewertungen des Verfassers eingebracht werden. - Wie soll es weiter mit Docker gehen, was sagt der momentane Plan ? (Roadmap)

Literaturverzeichnis

[1] Manuel René Theisen: *Wissenschaftliches Arbeiten: Technik – Methodik – Form*; 15. Auflage; Vahlen; München 2011; ISBN 978-3-8006-3830-7

[2] *Hochschule Heilbronn*; http://www.hs-heilbronn.de/; abgerufen am 14.08.2010

[3] *Jenkins*; https://jenkins.io/; abgerufen am 02.12.2016

[4] *Docker*; https://www.docker.com/what-docker; abgerufen am 02.12.2016