

## Fallstudie Entwicklungswerkzeuge (281761)

# Docker

Jerome Tagliaferri\*

6. Dezember 2016

Eingereicht bei Paul Lajer

\*190530, [jtagliaf@stud.hs-heilbronn.de](mailto:jtagliaf@stud.hs-heilbronn.de)

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>iii</b>
<b>Abbildungsverzeichnis</b>	<b>iv</b>
<b>Tabellenverzeichnis</b>	<b>v</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Ziel der Arbeit . . . . .	1
1.3 Vorgehensweise . . . . .	2
<b>2 Docker Grundlagen</b>	<b>3</b>
2.1 Definition . . . . .	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
<b>3 Docker im Produktiveinsatz</b>	<b>10</b>
3.1 Anwendungsbereiche - Wer nutzt Docker ? . . . . .	10
3.2 Tools - Kubernetes / Docker Swarm . . . . .	10
3.3 Alternativen und Unterschiede ? . . . . .	13
3.4 Wieso gerade Docker ? . . . . .	13
3.5 Weshalb kommt der Durchbruch erst jetzt ? . . . . .	13
<b>4 Fazit - Ausblick</b>	<b>14</b>
<b>Literaturverzeichnis</b>	<b>15</b>

# Abkürzungsverzeichnis

**ABK:** ABKÜRZUNG

# Abbildungsverzeichnis

2.1	VM vs Container	5
2.2	Treiber	6
2.3	Container-Images	7
3.1	docker-adoption	11
3.2	docker-swarm	12

# Tabellenverzeichnis

# 1 Einleitung

## 1.1 Motivation

In einer extrem kompetitiven wirtschaftlichen Umgebung, welche sich durch die Weiterentwicklung von technischen Systemen immer schneller und vor allem grundlegend verändert, ist es notwendig, seine Dienste auf einer flexiblen und skalierbaren Basis zu betreiben.

Auf diesem Hintergrund, sind viele Technologien im Big Data und Cloud Umfeld entstanden und gewachsen. Allen voran die Virtualisierung von Ressourcen und Systemen als Grundlage des Cloud Computing, mit dem Ziel auf individuelle Wünsche und eine stetig schwankende Ressourcenverteilung zu reagieren. So wie diese Technologie ihren Weg in viele weitere Bereiche geebnet hat, findet die Container Virtualisierung eine immer vielseitigere und intensivere Anwendung. Getragen und weiterentwickelt von Branchen-Größen wie Google oder IBM ist die Container Virtualisierung momentan eine der sich am schnellsten wachsenden Bereiche der Informatik welche eine noch stärkere Flexibilität und Automatisierung von Systemen und Ressourcen verspricht.

Aus diesem Grund, stellt die Container Virtualisierung ein relevantes Thema für jede Branche dar, welche mit Soft- und Hardwaresystemen in Kontakt kommt um in einem stetig veränderten Markt zu bestehen.

## 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. Dabei werden Anregungen für weitere Tools und Systeme gegeben welche diese Funktionalitäten erweitern und teilweise automatisieren.

## **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 darstellen, erläutert. Weiter wird die Umsetzung dieser Konzepte und deren Erweiterungen betrachtet und auf Basis eines einfachen Anwendungsbeispiels erläutert. Anschließend werden die Ergebnisse gebündelt und in Vor- und mögliche Nachteile unterteilt.

Aufbauend auf diesen Grundlagen werden Anwendungsbereiche vorgestellt, in denen 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 abschließen.

## 2 Docker Grundlagen

### 2.1 Definition

Die Bezeichnung Docker findet schon lange nicht nur im Cloud spezifischen Umfeld Erwähnung. Der aufmerksame Nutzer stößt 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 und auf unterschiedlichste Bibliotheken und Dienste 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 :

<sup>1</sup> „*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.

Hierbei tauchten erste Ansätze, welche später den Grundstein darstellen sollten, schon 1979 auf. Es begann dabei alles mit der Idee Services untereinander und vom eigentlichen Host isolieren zu können. Dieses Konzept wurde erstmals unter UNIX in der Form des "chroot" umgesetzt. Diese Funktion war in der Lage das Hauptverzeichnis eines Prozesses

---

<sup>1</sup>Vgl. <https://www.docker.com/what-docker>



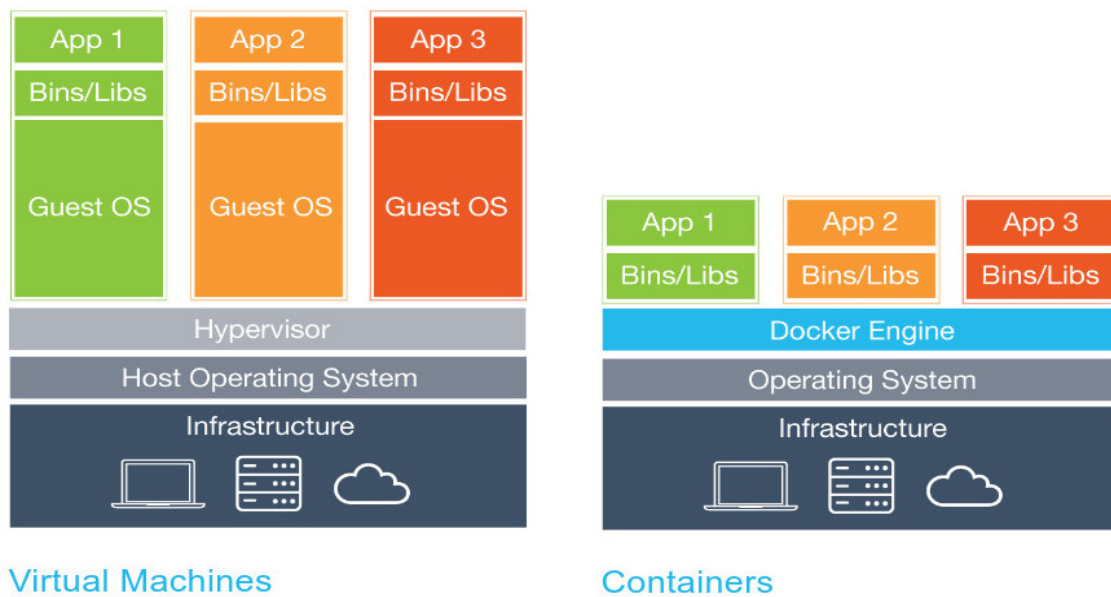
zu isolieren indem ihm ein neuer Ort zugewiesen wurde. Erst im Jahre 2000 wurde diese Funktionalität unter FreeBSD mit dem Namen Jails erneut aufgenommen und erweitert. Diesmal konnte nicht nur das Dateisystem isoliert werden, sondern auch dazugehörige Benutzer, Netzwerk und die zugehörigen Prozesse. Über die nächsten acht Jahre entstanden somit viele weitere Technologien welche diese Funktionalitäten integrierten. Darunter unter anderem im Linux VServer, Oracle Solaris Zones, OpenVZ oder auch die von Google entwickelten Process Container. Später wurde dann mit dem Namen ControlGroups ein weiterer wichtiger Baustein Teil des Linux Kernels. Erst 2008 entstand dann durch die Entwicklung bei IBM das Linux Containers project (LXC) welches die unterschiedlichen 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 dazu zusätzliche Software zu benötigen und somit den Einsatz enorm erleichterte. Am 15.03.2013 wurde dann Docker, durch den dotCloud Gründer Solomon Hykes zum ersten mal der Öffentlichkeit vorgestellt. Wobei diese Vorstellung nur aus einem fünf minütigen Vortrag auf der Python Entwickler Konferenz in Kalifornien bestand, verbreitete sich diese Information weltweit mit enormer Geschwindigkeit. Dies hatte zur Folge, dass große Firmen, allen voran Google großes Interesse an diesem Projekt verkündeten und es daraufhin auf GitHub offen gelegt wurde. Diese offene Entwicklung auch gegenüber drittanbietern, welche eigene Interpretationen und Tools beisteuern,

## 2.3 Wie funktioniert Docker ?

Abgesehen von der schnellen Verbreitung von Docker, war vielen Interessenten die eigentliche Funktionsweise nicht vollständig bewusst. Docker versprach, ein Stück Software in ein komplettes Dateisystem zu verpacken, welches alles beinhaltete was für das Ausführen dieser nötig ist. Dazu gehört der Quellcode, System Bibliotheken/Tools und die Ausführungsumgebung. Dies sorgte dafür, dass sich die Software immer gleich verhält egal auf welcher Umgebung sie ausgeführt wird. Hinzu kam, das Container aufgrund ihrer 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 ma-

Abbildung 2.1: VM vs Container



Quelle: [www.docker.com/what-docker](http://www.docker.com/what-docker)

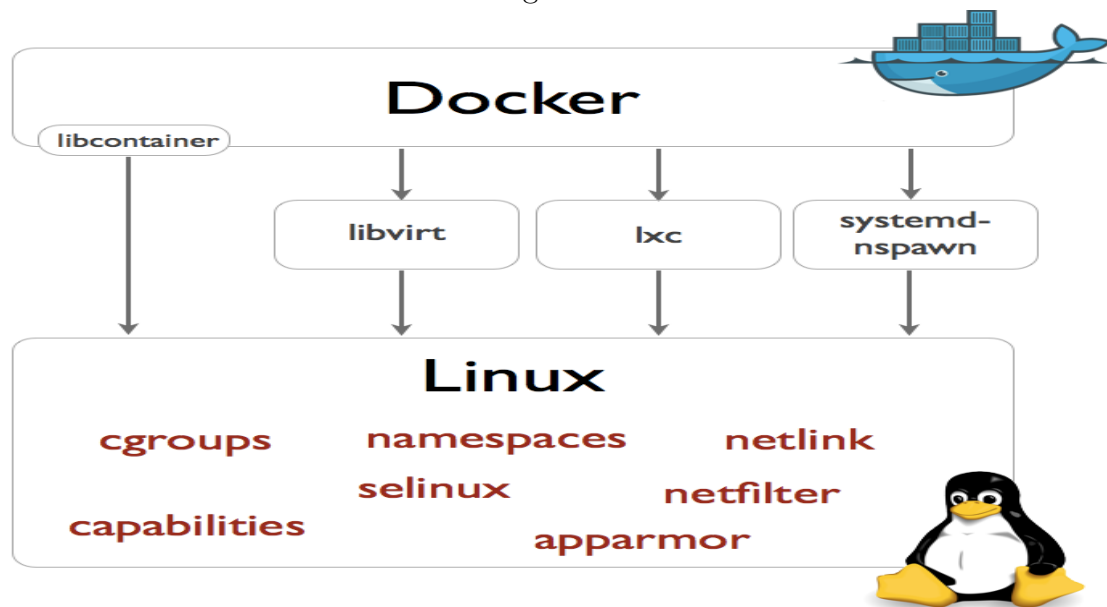
chen darauf aufmerksam, dass Docker sehr viel mehr ist. Um jedoch die Grundlegenden funktionsweisen zu erläutern, macht es Sinn diesen Vergleich zu bemühen.

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 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 Kom-

Abbildung 2.2: Treiber



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

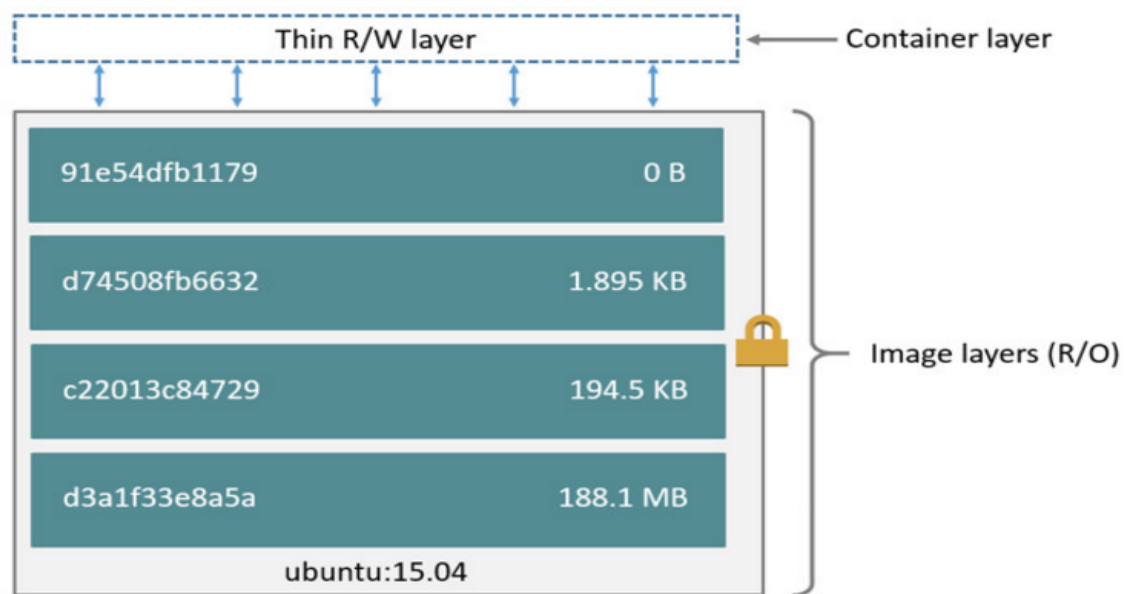
ponenten veröffentlicht wurden und allgemein als Standard angesehen werden. Eine weitere sehr essentielle Komponente stellt UnionFS dar. Dies ist ein Dateisystem, welches die Fähigkeit besitzt unterschiedliche Schichten (Standorte) von Dateien zu bündeln und einem Prozess zur Verfügung zu stellen. In 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.

## 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 vorgesetzt 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.

Abbildung 2.3: Container-Images



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

- **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.
- **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. Hierfür 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 Arbeitererstellt.

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 Verzeichnisses 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 [usr/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

Um die im Kapitel Grundlagen zusammen getragenen Informationen zu bündeln wird nun auf die allgemeinen Vor- und Nachteile der Docker Container eingegangen.

Vorteile :

- Container sind extrem klein und skalierbar wodurch sie innerhalb kürzester Zeit erzeugt und ausgeführt werden können um veränderten Anforderungen gerecht zu werden.
- Sie verhalten sich auf jeder Umgebung immer gleich und man kann dadurch sicher sein bei einem lauffähigen Container auf keine Probleme zu stoßen.
- Beinhaltenden im Vergleich zu VMs keinen Overhead im Bezug auf das Betriebssystem - Prozessen können gezielt Ressourcen zugewiesen werden
- Viele vorgefertigte Applikationen z.B. Jenkins, PostgreSQL, Ubuntu, Wordpress stehen über DockerHub zur Verfügung und sind innerhalb weniger Sekunden einsatzbereit.
- Durch den Einsatz von Containern, können Entwickler Software schneller entwickeln und sie in einer Produktivumgebung testen.
- Verringert die Komplexität der Entwicklung, da jeder mit der exakt selben Umgebung arbeitet.
- Sind durch ihre Größe und Isolation sehr gut für Microservices geeignet
- Umgebungen werden automatisiert erstellt wodurch Zeit eingespart wird

Nachteile :

- Leichtsinnigkeit kann zu Sicherheitsproblemen führen, da alle Container auf denselben Kernel zugreifen
- Da Docker Images nur lesbar sind müssen Daten extern über Volumes zur Verfügung gestellt werden
- Docker kann auf Windows und MacOSX ausgeführt werden, verliert jedoch auf diesen Plattformen an Geschwindigkeit

Es sei jedoch gesagt, dass Docker einer wahnsinnig schnellen Entwicklung unterliegt und einige der genannten Negativ Aspekte bereits bearbeitet werden. Man kann davon ausgehen, dass auch im Bereich Monitoring und Usability noch viel geschehen wird.

## 3 Docker im Produktiveinsatz

### 3.1 Anwendungsbereiche - Wer nutzt Docker ?

Es ist nicht verwunderlich, dass Docker von einer Firma wie dotCloud entwickelt wurde. Im Bereich von PAAS war der Druck wie auch der Wunsch, auf Veränderungen im Anforderungsverhalten der Nutzer, schnell und automatisiert reagieren zu können, am stärksten spürbar. Ein Blick auf die Unternehmen, welche Docker am stärksten unterstützen bestätigt diese Behauptung : Cisco - Google - Huawei - IBM - Microsoft - Red Hat Solch ein Unterstützer Stamm, zeigt deutlich wie wichtig diese Technology in Zukunft sein wird.

Betrachtet man Abbildung 3.1 wird deutlich, dass gerade im Webservice Bereich in Verbindung mit BigData Anwendungen, Docker am stärksten eingesetzt wird. Um jedoch eine Vorstellung des Umfangs zu bekommen in wie weit Container eingesetzt werden, hilft ein Zitat von Joe Beda, Software Entwickler bei Google, welcher im Jahre 2014 folgendes verkündete.

<sup>1</sup> „*Everything at Google runs in a Container*“  
„*We start over 2 billion containers per week.*“

Anwendungsbereiche, welche Docker bereits im Produktiveinsatz nutzen, beschränken sich jedoch nicht nur auf große Cloud Dienstleister. Gerade im Bereich der Software Entwicklung beginnen viele Teams Docker für sich zu entdecken um Probleme und Barrieren bei DevOps zu überwinden. Der Grund hierbei liegt in der einfachen Art und Weise unterschiedlichste Umgebungen und Tools zur Verfügung zu stellen. Diese unterstützen den Entwickler nicht nur , sondern geben ihm die Möglichkeit Software in einer Produktivumgebung zu entwickeln und testen.

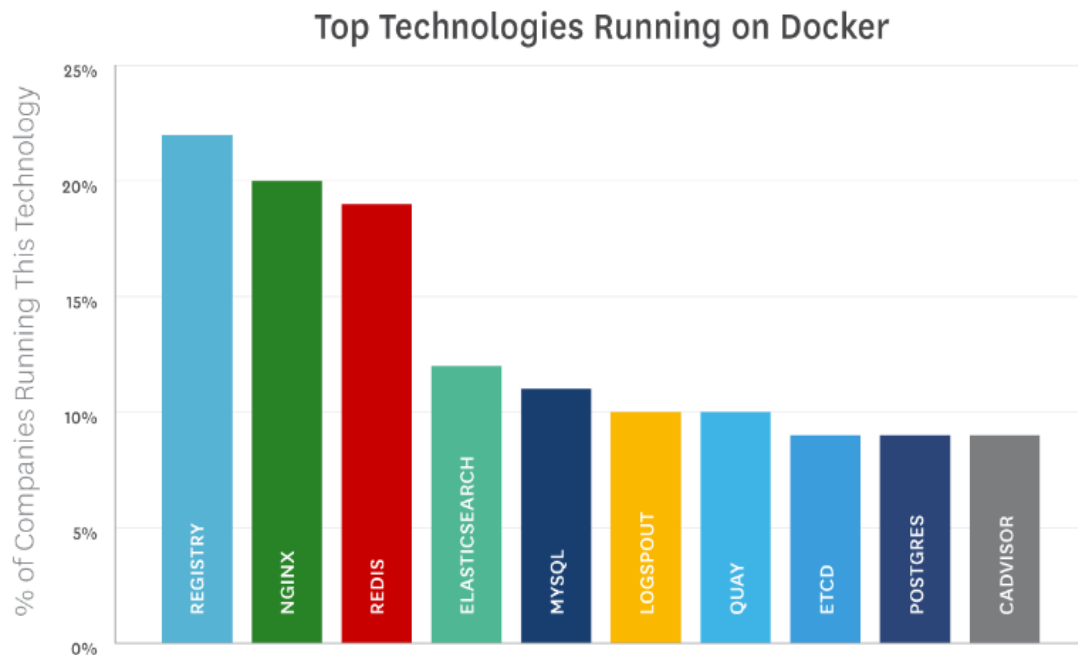
### 3.2 Tools - Kubernetes / Docker Swarm

Der Einsatz von großen Mengen an Containern erzeugt ganz natürlich einen enormen Verwaltungs und Konfigurationsaufwand. Aus diesem Grund entstanden sehr schnell unterschiedliche Lösungsansätze um große Teile davon zu automatisieren. Selbst kleine Web

---

<sup>1</sup>Vgl. <https://speakerdeck.com/jbeda/containers-at-scale>

Abbildung 3.1: docker-adoption



Quelle: <https://www.datadoghq.com/docker-adoption/>

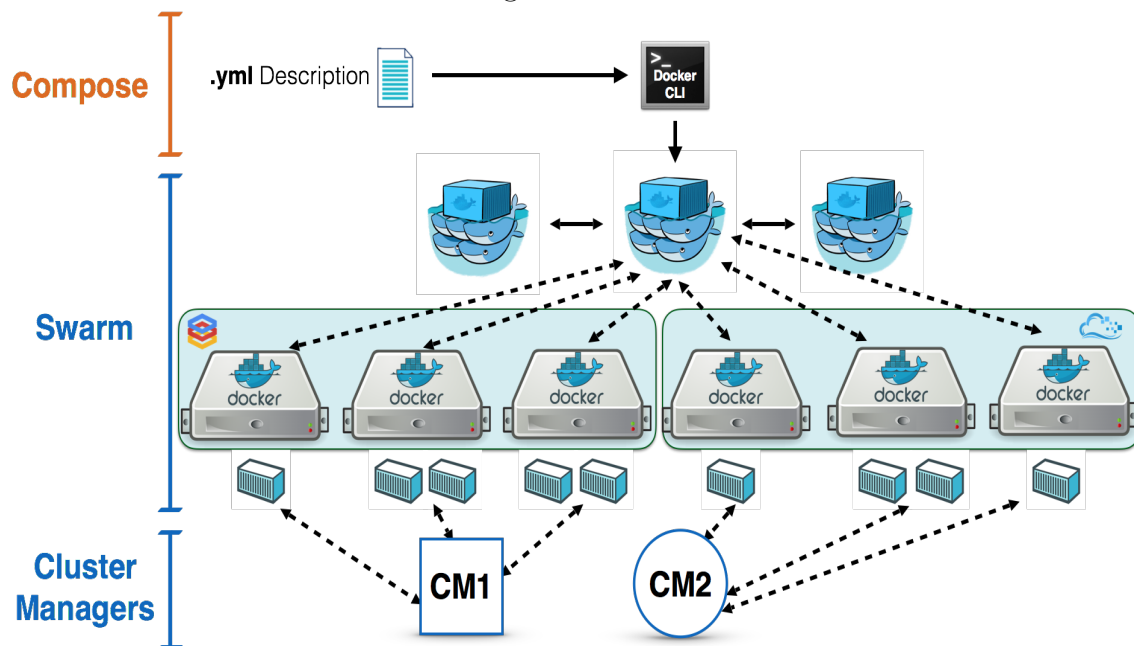
Projekte bestehen aus unterschiedlichen Services welche zusammen arbeiten müssen um eine lauffähige Applikation zu kreieren. Hierzu gehören Datenbanken, Authentifizierungsservices, Web Server und Load Balancer um nur wenige zu nennen. Um den Betrieb der Hauptapplikation im Falle von Problemen oder Änderungen nicht vollständig zu gefährden, empfiehlt es sich für jeden Service eine isolierte Umgebung zu schaffen. Dies führt jedoch in der Praxis sehr schnell zu Verwirrung und aufwändiger Dokumentation, wozu meist unnötige Konflikte folgen. Ein Tool welches diese Problematik lösen soll ist Docker Compose. Hierbei wird die gesamte Applikationsumgebung in einer Konfigurationsdatei genau definiert und kann dann mit nur einem Befehl in Betrieb genommen werden. Um den Pool an Containern von möglichen weiteren abzugrenzen, wird diesem ein Projekt Name zugewiesen, über welchen die einzelnen Services ansprechbar sind.

Vorteile beim Einsatz von Compose sind z.B.:

- Mehrere Webapplikation welche aus verschiedenen Containern bestehen, können parallel ausgeführt werden ohne miteinander in Konflikt zu geraten. Dies ist gerade im Bezug auf Entwicklungsumgebungen von Vorteil.
- Die Konfigurationsdatei dient zusätzlich als Dokumentation
- Durch die schnelle Inbetriebnahme werden automatisierte Tests stark vereinfacht.



Abbildung 3.2: docker-swarm



Quelle: <https://blog.docker.com/2015/11/deploy-manage-cluster-docker-swarm/>

Compose funktioniert jedoch nur auf einem Host und ist deshalb nicht für große und verteilte Applikationen einsetzbar. Auch hier entstand im Laufe der Entwicklung des Docker Frameworks eine Native Cluster Lösung. Bekannt unter dem Namen Docker Swarm sorgt dieses Tool dafür (siehe Abbildung 3.2), unterschiedliche Host Systeme zu einem virtuellen Docker Host zu bündeln.

Es bringt zusätzlich Funktionalitäten wie hohe Verfügbarkeit und Ausfallsicherheit mit sich. Dabei ist die Inbetriebnahme nicht weiter komplex und auch durch Drittanbieter Tools umsetzbar. Dies wird durch den kompletten Support der Docker API möglich und folgt auch hier folgendem Leitsatz :

<sup>1</sup> „*batteries included but removable*“

Seit Version 1.12.0 ist Swarm nun ein fester Bestandteil der Docker Engine. <https://docs.docker.com/engine/swarm/>

Bevor Docker überhaupt veröffentlicht wurde, experimentierte Google schon seit langem mit LXC Containern und entwarf hierfür eine Cluster Lösung. Mit der steigenden Beliebtheit von Docker stellte Google schließlich sein System Kubernetes als Open Source

<sup>1</sup>Vgl. <https://docs.docker.com/v1.6/swarm/>

Lösung zur Verfügung. Die Idee der Bündelung von Containern unterscheidet sich kaum zwischen Docker Swarm und Kubernetes. Jedoch kann Kubernetes auf einen Erfahrungsschatz von über 15 Jahren im produktiven Umfeld zurück greifen. Einige Funktionen sind z.B. :

- Einem Container Bündel (auch Pod genannt) können klare CPU und RAM vorgaben erteilt werden, wodurch automatisiert der optimale Knoten ausgewählt werden kann.
- Kubernetes entscheidet automatisiert wieviele Pods gerade aktiv sein müssen.
- Es ist möglich einen Rollout im laufenden Betrieb durchzuführen ohne den Betrieb hierfür zu unterbrechen. Sollte dabei ein Problem auftauchen, wird automatisch ein Rollback durchgeführt.
- Support für unterschiedlichste Speichermöglichkeiten. Egal ob Lokal oder von einem Cloud Dienstleister.
- Automatisierte Bereinigung von abgestürzten Containern und Knotenpunkten

Obwohl Docker Swarm und Kubernetes versuchen diesselben Probleme zu lösen, so sind die Lösungsansätze doch sehr verschieden und können je nach Anwendungsfall besser oder aber auch schlechter sein.

### **3.3 Alternativen und Unterschiede ?**

Wie anfangs betont ist die Geschichte der Container Virtualisierung weitaus älter und umfangreicher als man dies vermuten möchte. Deshalb stellt sich auch die Frage, welche Alternativen es zu Docker gibt. Aus diesem Grund, soll hier eine Auflistung einiger Alternativen und deren Unterschiede entstehen.

### **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