HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN MÜNCHEN

FAKULTÄT FÜR INFORMATIK UND MATHEMATIK

Containisierung von Java Apps mit Docker

Kalugin Vera

Martikel-Nr. 15551015

Schriftliche Ausarbeitung angefertigt im Rahmen des Seminars

Aktuelle Technologien zur Entwicklung verteilter Java-Anwendungen

Studiengang Bachelor Wirtschaftsinformatik

Sommersemester 2018

Betreuer

M. Theis

Abgabe

11.Mai 2018

Versicherung

„Ich versichere, dass ich die vorstehende Arbeit selbständig angefertigt und mich fremder Hilfe nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß veröffentlichtem oder nicht veröffentlichtem Schrifttum entnommen sind, habe ich als solche kenntlich gemacht."

_____

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

DevOps	Development and Operations
OS	Operating System
LXC	Linux Containers
UFS	Unit File System
CI -Tool	Continiuous Integration Tool

Kurzfassung

Dieses Dokument zielt darauf ab, ein grundlegendes Verständnis in Bezug auf die Containisierung von Java Applikationen mit Docker zu geben.

1. Einleitung

In der Welt der Softwareentwicklung gewann das Thema Virtualisierung in den letzten Jahren immer mehr an Bedeutung. Die ursprüngliche Motivation die die Virtualisierung antrieb, waren die Rechnerkapazitäten die nur teilweise ausgelastet waren. Aufgrund dessen wurden die bekannten Informatiker Gerald J. Popek und Robert P. Goldberg[1] tätigt und entwickelten die systembasierte virtuelle Maschine. Dabei wurde zunächst auf die vollständige Virtualisierung eines Rechnersystems abgezielt. Im Nachgang befasste man sich mit einer Lösung die es ermöglichen sollte eine Anwendung die für eine bestimmte Rechnerarchitektur erstellt wurde bei Wechsel auf einen andren Rechner problemlos ausführen zu können. Mit diesen zwei Erfindungen wurde bereits ein großer Meilenstein in der IT – Welt gesetzt.

Doch die Prioritäten der Unternehmen haben sich mit der Zeit verändert. Durch die stetig steigende Anzahl an Daten und komplexen Anwendungen war eine Umstrukturierung bzw. Weiterentwicklung der bisher vorhanden Virtualisierungslösungen von Nöten. Kunden legen immer mehr Wert darauf, dass Applikationen schnell, mit hoher Qualität und kostengünstig bereitgestellt werden. Zudem wird eine hohe Skalierbarkeit, Hochverfügbarkeit und Multi-Mandanten-Fähigkeit dieser vorausgesetzt. Vor allem die cloudbasierten Lösungen und die Zusammenarbeit von Entwicklung und Betrieb, auch DevOps genannt, gaben den Anstoß für den Entwurf neuer Strategien. Um all diesen Anforderungen die an die Software gestellt werden gerecht zu werden entstand der Ansatz mittels der Containertechnologie Docker, die virtuellen Maschinen zu erweitern.

Diese Arbeit zielt darauf ab, zunächst einen Einblick über die Entstehung von Virtualisierungstechnologien bis hin zur Weiterentwicklung dieser zu geben. Der Fokus des Themas liegt auf der Containertechnologie Docker, die heutzutage aus der IT nicht mehr weggedacht werden kann. Dabei wird auf die Architektur und die Terminologie eingegangen. Abschließend wird die Arbeit mit einem Anwendungsbeispiel und einem Ausblick abgerundet.

[1] Popek, G., & Goldberg, R. (1974). *Formal Requirements for Virtualizable Third Generation Architectures*. Abgerufen am 01. Mai 2018 von citeseerx.ist.psu.edu: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.4815&rep=rep1&type=pdf

2. Virtualisierung

2.1 Problemstellung – Entstehung der Virtualisierungstechnologie

Der Ursprung des Begriffes Virtualisierung lässt sich bereits in die frühen 1960 Jahre zurückverfolgen. Auslöser für die Entstehung dieser Technologie war die Rechenzeit die in einem Rechnersystem nur zu einem gewissen Grad genutzt werden konnte. Obwohl Rechenzentren für die damalige Zeit über leistungsfähige Rechnersysteme verfügten, konnten weder die vorhandenen Ressourcen noch die Prozessorleistung vollständig ausgeschöpft werden.[2]

Im Folgenden soll die Problematik anhand eins Anwendungsbeispiels mit einer Umgebung ohne den Einsatz einer Virtualisierungstechnologie verdeutlicht werden.

In einem Unternehmen wird mit einer Vielzahl an Anwendungen und Diensten gearbeitet. Damit diese optimal ausgeführt werden können, werden ausreichend Rechnerressourcen, wie z.B. Arbeitsspeicher, Prozessor, usw. benötigt. Nun muss aus unternehmerischer Sicht entschieden werden, wie die Ablaufumgebung aussehen soll.

Eine mögliche aber ineffiziente Lösung wäre alle Applikationen auf einen Rechner zu packen. Wird dieser nämlich überfüllt so nimmt auch die Rechnerleistung ab, da er nur eine bestimmte Kapazität besitzt mit der er andere Komponenten bedienen kann. Tritt diese Situation ein, so kann es dazu kommen, dass eine Anwendung nicht mehr ausgeführt werden kann und somit Teile eines Unternehmens außer Betrieb gesetzt werden können.
Um ein solches Ausmaß zu umgehen gibt es die Variante, jedem Rechner jeweils ein Programm zuzuordnen. Dadurch wird sichergestellt, dass ausreichend Ressourcen zur Verfügung stehen und somit die jeweiligen Anwendungen problemlos genutzt und ausgeführt werden können. Weiterhin ist zu erwähnen, dass bei Ausfall einer Komponente des Rechners nur die Anwendung die sich auf diesem befindet betroffen ist und alle anderen unberührt bleiben.

Doch auch hier überwiegen die Nachteile. Mit zunehmender Rechnerkapazität fallen hohe Kosten für den Betrieb, Wartung und Strom an. Zu erwähnen ist auch der zusätzliche Platzbedarf und die hohen Anschaffungskosten für neue Maschinen. Weiterhin wird die Flexibilität eingeschränkt, da sich die Hardwareressourcen eines bestehenden Servers nur sehr schwer anpassen lassen.[3]

An diesem Punkt kommt die Server-Virtualisierung zum Einsatz. Dabei wird jede Anwendung in eine virtuelle Maschine gepackt und alle virtuellen Maschinen können auf nur einem Server ausgeführt werden.

---

[2] Balmes, F. (2014). *Virtualisierungstechniken: Grundlagen und Anwendung im Serverbetrieb.* (D. Verlag, Hrsg.) Hamburg: Diplomica Verlag GmbH 2014.Abgerufen am 09. April 2018 von books.google.de: https://books.google.de/books?id=o12wBQAAQBAJ&pg=PP4&dq=ibm+virtualisierung+geschichte&hl=de&source=gbs_selected_pages&cad=2#v=onepage&q=ibm%20virtualisierung%20geschichte&f=false S. 11

[3] Ambroise, J.-E. (21. Dezember 2016). *Virtualisierung verständlich erklärt.* Abgerufen am 09. April 2018 von blog.beronet.com: https://blog.beronet.com/de/virtualisierung-verstaendlich-erklaert

2.2 Definition Virtualisierung

Der Vorreiter auf diesem Gebiet war das Unternehmen IBM, das die Virtualisierung auf Softwareebene erschuf. Mittels einer softwarebasierten Virtualisierungslösung, der virtuellen Maschine, die auf dem Großrechnersystem „System/360“ implementiert wurde, gelang es somit die Ressourcen eines Rechners optimal ausnutzen.[4]

Im Allgemeinen wird die Technologie dazu benutzt, die Software von der Hardware mittels einer Abstraktionsschicht zu entkoppeln. Dabei werden Anwendungen in mehrere virtuelle Umgebungen eingebettet, die sich wiederum mit der Hardware und dem Betriebssystem auf einem Rechner befinden. Die erzeugten virtuellen Umgebungen werden auch als Gastsysteme und der Rechner als Wirtsystem, der sogenannte Host, bezeichnet. Aufgrund dessen ist es möglich mehrere OS auf nur einer einzigen Maschine zu starten, was wiederum eine hohe Effizienz zur Folge hat. Besonders im Server-Umfeld wird der Nutzen der Virtualisierung deutlich. Da modern gebaute Serversysteme oftmals zu leistungsfähig sind um nur einen Server zu betreiben liegt ein Großteil der Ressourcen still und bleibt unbenutzt. Durch den Virtualisierungsmechanismus gelang es nun, dass auch die ungenutzten Rechnerkapazitäten optimal ausgelastet werden, indem physisch vorhandene Gegebenheiten in Software nachgebildet wurden.

Im nächsten Schritt soll die verallgemeinerte Definition der Virtualisierung konkretisiert werden.

Ein Serversystem besteht aus drei Schichten, der Software, der Hardware und Netzwerken. Grundsätzlich ist auf einem Computersystem ein Betriebssystem installiert, wobei Softwareanwendungen die Schnittstellen und Bibliotheken des OS mitbenutzen. Wiederum können auf der Hardware-, und Netzwerkebene informationstechnologische Operationen ausgeführt werden. Mit der Zuhilfenahme der Abstraktionsschicht, dem sogenannten Abstraktions-Layers, die sich zwischen dem Anwender (z.B. Betriebssystem) und der physisch vorhandenen Ressource (z.B. Computerhardware) befindet, werden diese auf dem Gastsystem simuliert. Konkret ausgedrückt wird ein Abbild von physisch vorhanden Gegebenheiten geschaffen, wobei diese nicht real, sondern nur virtuell zur Verfügung stehen.

Mittels der Virtualisierungsebene werden dem Nutzer folglich zwei mögliche Situationen präsentiert. Entweder werden mehrere Ressourcen in einer einheitlichen Umgebung gebündelt, was wiederum dem Anwender als Ganzes vorgespielt wird oder die tatsächlich vorhandenen Ressourcen werden auf gesplittet, sodass dieser mehrere logische Komponenten zur Verwendung gestellt bekommt - *siehe Abbildung 1*.[5]

---

[4] Balmes, F. (2014). *Virtualisierungstechniken: Grundlagen und Anwendung im Serverbetrieb.* (D. Verlag, Hrsg.) Hamburg: Diplomica Verlag GmbH 2014.Abgerufen am 09. April 2018 von books.google.de: https://books.google.de/books?id=o12wBQAAQBAJ&pg=PP4&dq=ibm+virtualisierung+geschichte&hl=de&source=gbs_selected_pages&cad=2#v=onepage&q=ibm%20virtualisierung%20geschichte&f=false S. 12

[5] Lipinski, D.-I. (09. November 2017). *Virtualisierung*. Abgerufen am 11. April 2018 von www.itwissen.info: https://www.itwissen.info/Virtualisierung-virtualization-technology-VT.html

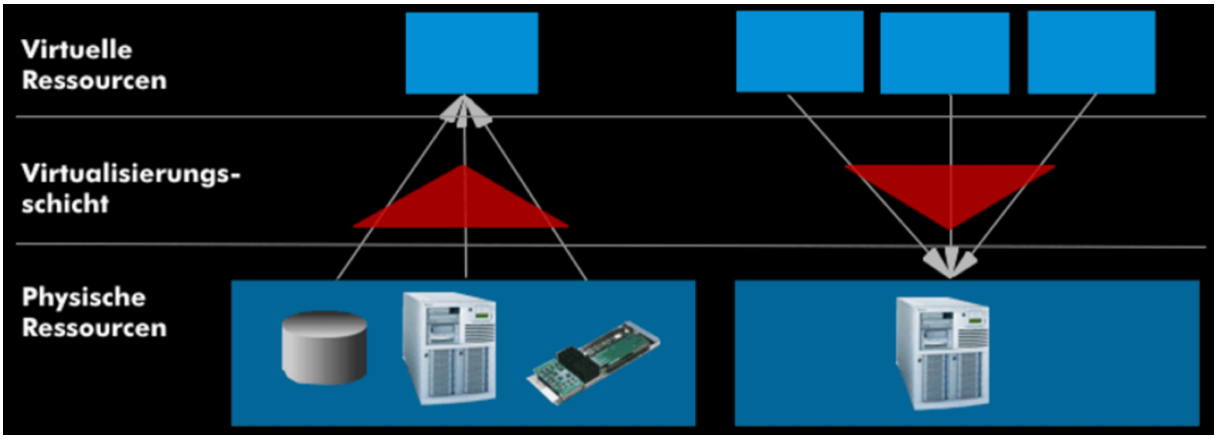


Abb. 1: Abstraktionsebene für die Zusammenfassung und Aufteilung von Ressourcen

Quelle: ITWissen.info (09. November 2017): https://www.itwissen.info/Virtualisierung-virtualization-technology-VT.html

Die Hauptaufgabe des Layers besteht darin die Kommunikation zwischen dem Wirtsystem und dem Gastsystem aufrechtzuerhalten, indem Ressourcenanfragen die von einem Gast gestellt werden aufgefangen und diese an den Host weiterleitet werden. Sobald ein Ergebnis vorliegt nimmt er dieses entgegen, schickt es zurück und speichert alle vorgenommenen Änderungen.

2.3 Arten der Virtualisierung[6]

Im Zuge der fortschreitenden Digitalisierung nahm im Laufe der Zeit die Komplexität von Computersystemen kontinuierlich zu. Folglich musste sich auch die Virtualisierung an den Wandel anpassen und setzt mittlerweile mehr als nur an einer Schicht in einem Rechnersystem an.

In diesem Abschnitt wird die Ausprägung der Virtualisierung, auf Hardware-, Software- und Netzwerkebene näher betrachtet. Hierbei wird darauf abgezielt ein grundsätzliches Verständnis über die Erscheinungsmöglichkeiten dieser Technologie zu schaffen.

2.3.1 Hardware Virtualisierung

Diese Art der Virtualisierung erlaubt es die real vorhandene Hardware in mehreren Komponenten nachzubilden, die wiederum unabhängig voneinander in einer virtuellen Umgebung ausgeführt werden. Diese Komponenten enthalten die gleichen Eigenschaften und können somit genau wie die tatsächlich vorhandene Hardware genutzt werden. Man unterscheidet hier zwischen der Partitionierung eines Servers oder der Computerressourcen und der Virtualisierung eines Speichermediums, sowie des Prozessors.

[6] *Virtualisierung (Informatik)*. (30. Dezember 2017). Abgerufen am 10. April 2018 von https://wikipedia.de: https://de.wikipedia.org/wiki/Virtualisierung_(Informatik)

2.3.2 Software Virtualisierung

Auf Softwareebene kann die Virtualisierung auf mehreren Wegen realisiert werden. Eingesetzt wird diese um entweder ein Betriebssystem oder nur einzelne Anwendung nachzubilden. Wie auf der Hardwareebene wird auch hier ein Hypervisor eingesetzt. Eine weitere Möglichkeit wäre der Einsatz von Containern.

- Mittels Hypervisor

  Der Hypervisor stellt eine Virtualisierungsschicht dar die es erlaubt neben dem real vorhanden Betriebssystem weitere Betriebssysteme mittels einer virtuellen Maschine auf nur einem Rechner auszuführen. Damit wird der parallele Betrieb mehrerer Betriebssysteme ermöglicht. Dabei bedient sich das Gast-OS mittels einer Schnittstelle an den Ressourcen des Host-OS.

- Mittels Container

  Hier werden Anwendungen anstatt in einer virtuellen Maschine, in einem Container organisiert. Kommt der Container auf einem System zur Ausführung, so wird diesem die Laufzeitumgebung vom Host-System zur Verfügung gestellt. Im *Kapitel 3 Containerbasierte Virtualisierung* wird die Nutzung eines Containers im Detail erläutert.

- Anwendungsvirtualisierung

  Die Anwendungsvirtualisierung ermöglicht, dass Anwendungen nicht nur von einem Betriebssystem abhängig sind. Diese werden je nach Bedarf komplett oder nur bis zu einem gewissen Grad in einer virtuellen Umgebung isoliert. Die Zwischenschicht, die sich zwischen den Anwendungen und dem Betriebssystem befindet, stellt diesen die Laufzeitumgebung zur Verfügung. Eine Anwendung die nur unter einem bestimmten Betriebssystem, z.B. Windows, ausgeführt werden kann, verhält sich so als ob sie direkt auf dem jeweiligen Betriebssystem implementiert wäre, obwohl sie eigentlich in einer virtuellen Umgebung eingebettet ist.

2.3.3 Netzwerkvirtualisierung

Grundsätzlich wird hier angestrebt, dass die verfügbare Netzwerkbandbreite in mehrere unabhängige Kanäle aufgeteilt wird. Dabei kann für jeden Kanal ein Server oder ein Gerät in Echtzeit zugewiesen werden. Mittels dieser Technik soll erreicht werden, dass der Administrationsaufwand im Netzwerk-Management abnimmt und die Geschwindigkeit eines Netzwerkes, sowie die Zuverlässigkeit, Flexibilität und die Sicherheit verbessert werden.

2.4 Gründe die für Virtualisierung sprechen

Je nach Einsatzgebiet bietet die Technologie vielversprechende Vorteile, die in diesem Abschnitt erläutert werden.

Ausgehend von der Nutzung einer virtuellen Maschine, die es ermöglicht eine Vielzahl an verschiedenen Betriebssystemen auf einer physisch vorhandenen Maschine ablaufen zu lassen, wird der Aufbau von komplexen Testumgebungen mit mehreren Rechnern ermöglicht.

Des Weiteren kann eine Software unabhängig vom Betriebssystem des Host-Rechners in einer virtuellen Maschine ausgeführt werden.

Da virtuelle Maschinen isoliert voneinander ablaufen wird damit wird gewährleistet, dass der laufende Betrieb anderer Maschinen nicht gestört wird.

Eine bedeutende Rolle nimmt der Umweltfaktor ein. Durch die Virtualisierung kommen weniger physische Rechner in Rechenzentren zum Einsatz. Somit fällt der Stromverbrauch geringer aus.

Zu erwähnen ist auch der Platzbedarf und die Kostenersparnis. Je weniger Server nachgerüstet werden müssen, desto weniger Platz wird eingenommen und die Anschaffungskosten fallen für zusätzliche Maschinen weg. Kommt es bei einer Hardware zum Ausfall oder es werden Wartungsarbeiten ausgeführt, dann kann die virtuelle Maschine auf einem anderen Host-System wieder ausgeführt werden.

3. Containerbasierte Virtualisierung

Im Digitalisierungszeitalter lässt sich beobachten, dass die Menge an Softwareanwendungen kontinuierlich angestiegen ist und auch in Zukunft mit einer Zunahme zu rechnen ist. Fokussiert werden immer mehr die Themen wie Flexibilität, Geschwindigkeit und Agilität einer Software. Damit ein Unternehmen am Markt konkurrenzfähig bleibt müssen geeignete Technologien angewandt werden, die eine Softwareanwendung in einem schnelleren Tempo und mit geringem Aufwand zur Verfügung stellt.

In diesem Zusammenhang bieten containerbasierte Technologien eine alternative Möglichkeit zu Hypervisorbasierten virtuellen Maschinen. Was unter diesem Begriff verstanden wird und welche Unterschiede sowie Vorzüge diese Virtualiserungsmethode mit sich bringt, wird im folgenden Abschnitt erörtert.

3.1 Was sind Container

Container-Technologien sind bereits seit geraumer Zeit im IT-Umfeld präsent und nehmen in der heutigen Zeit eine besondere Rolle in vielen Unternehmen ein. In erster Linie wird dieser Mechanismus dazu verwendet, um Prozesse besser gegeneinander zu isolieren. Zudem wird der Fokus auf eine deutlich einfachere Entwicklung und zugleich schnellere Bereitstellung von Anwendungen gelegt.[7]

Der Begriff Container wurde aus der industriellen Ebene in die Informatik übertragen. In der klassischen Industrie, dem Frachtverkehr, werden Waren in Container gelagert und von einem Ort zum anderen verfrachtet. Dabei wird darauf abgezielt, dass die zu verfrachtenden Güter „[…] zuverlässig, kostengünstig und sicher verarbeite[t] […]“[8] werden, ohne dass das Format oder die Beschaffenheit derer eine große Rolle dabei spielt.

Dasselbe Ziel verfolgen auch IT-Unternehmen für ihre Softwareanwendungen. Bei der Nutzung von Softwarecontainern ist die Vorgehensweise prinzipiell die gleiche. Die Anwendungen stellen die Güter die ausgeliefert werden müssen dar und der Container dient als Behältnis, welches in seiner Gesamtheit transportiert werden soll. Dabei wird die Anwendung samt ihrer Abhängigkeiten, die dazu benötigt werden um diese lauffähig zu gestellten, in den Behälter eingebettet. Somit wird gewährleistet, dass bei einem Umgebungswechsel die Software zuverlässig zum Laufen gebracht werden kann.[9]

Besonders im Linux-Bereich sind Container weit verbreitet. Möchte man einen Container auf dem Linux-System ausführen, so ist nur die Installation einer passenden Containerplattform notwendig.

---

[7] COMPAREX, R. (26. Oktober 2017). *Es geht nicht ohne: Neue TEchnologien für den digitalen Wandel - DevOps, Container & Co.* Abgerufen am 15. April 2018 von www.comparex-group.com: https://www.comparex-group.com/web/de/de/blog/topics/software/neue-technologien-fuer-den-digitalen-wandel-devop-container-und-co.htm

[8] Skerlec, M. (12. Juli 2017). *Was haben Frachtcontainer mit IT zu tun?* Abgerufen am 20. April 2018 von blog.bwi.de: https://blog.bwi.de/was-haben-frachtcontainer-mit-it-zu-tun/

[9] Geißler, O., & Ostler, U. (06. September 2017). *Was sind (Software-)Container?* Abgerufen am 16. April 2018 von www.datacenter-insider.de: https://www.datacenter-insider.de/was-sind-software-container-a-638949/

3.2 Containisierung versus klassische Virtualisierung

Allgemein betrachtet gewährleisten sowohl Container als auch virtuelle Maschinen voneinander isolierte Umgebungen. Beide können dazu verwendet werden um eine Software zu verpacken oder diese zu verteilen. Dabei liegt der wesentliche Unterschied darin, dass Container einzelne Anwendungen und virtuelle Maschinen ein komplett nachgebildetes System vom eigentlichen System isolieren.

Weiterhin ist zu erwähnen, dass für Container kein Hypervisor, der für die Ressourcenverteilung auf Hardwareebene zuständig ist, erforderlich ist. Bei Containern erfolgt die Virtualisierung auf der Ebene des Host - Betriebssystems, somit können sich alle Anwendungen jeweilige Ressourcen teilen.

In Hinblick auf die Sicherheit haben virtuelle Maschinen gegenüber Containern einen klaren Vorteil. Dadurch, dass sie stark voneinander abgekapselt auf einem System ablaufen, können Hardwarekomponenten getrennt voneinander virtualisiert und unabhängig voneinander den jeweiligen virtuellen Umgebungen zur Verfügung gestellt werden. Somit können bei einer Hackerattacke nur einzelne VM´s befallen werden. Wiederum bedienen sich alle Container gemeinsam an den Betriebssystem-Kernel-Ressourcen, wobei diese über privilegierte Rechte für den Zugriff besitzen. Ein Angreifer hat damit die Möglichkeit nicht nur die einzelnen Container, sondern auch den Host auf dem die Container ausgeführt werden zu beschädigen.

Werden mehrere virtuelle Maschinen auf einem System betrieben, so müssen auch notwendige Ressourcen vom OS zur Verfügung gestellt werden, um den Betrieb dieser gewährleisten zu können. Dieser Vorgang hat einen hohen Overhead zur Folge, denn allein für die generelle Bereitstellung einer virtuellen Maschine werden Unmengen an Ressourcen aufgebraucht. In Hinblick auf Container, stehen viele Ressourcen, wie das Operating System, auf dem Container - Host bereit. „[ ] Startet ein Container, muss dieser nicht das ganze Betriebssystem booten, Bibliotheken laden und Ressourcen für das eigene Betriebssystem zur Verfügung stellen. [ ]“[10]

Sowohl Container als auch virtuelle Maschine haben ihre Vor- und Nachteile. Deshalb ist es für ein Unternehmen wichtig abzuwägen welche Technologie am besten geeignet ist. Zudem findet sich in der Praxis oftmals die Mischform wieder. Dabei kommt ein Container auf einer VM zum Einsatz. [11]

In *Abbildung 2* wird der grundsätzliche Aufbau einer virtuellen Maschine und der Containertechnologie gegenübergestellt.

[10] Joos, T., & Karlstetter, F. (21. September 2016). *Warum Unternehmen auf die Container-Technologie setzen sollten.* Abgerufen am 15. April 2018 von www.cloudcomputing-insider.de: https://www.cloudcomputing-insider.de/warum-unternehmen-auf-die-container-technologie-setzen-sollten-a-550570/

[11] Rossbach, P. (04. April 2017). Container vs. Virtuelle Maschinen (VM´s): Ein Blick auf die Sicherheit. Von bee42.com: https://bee42.com/de/blog/Container-vs-Virtuelle-Maschinen-Sicherheit/ abgerufen

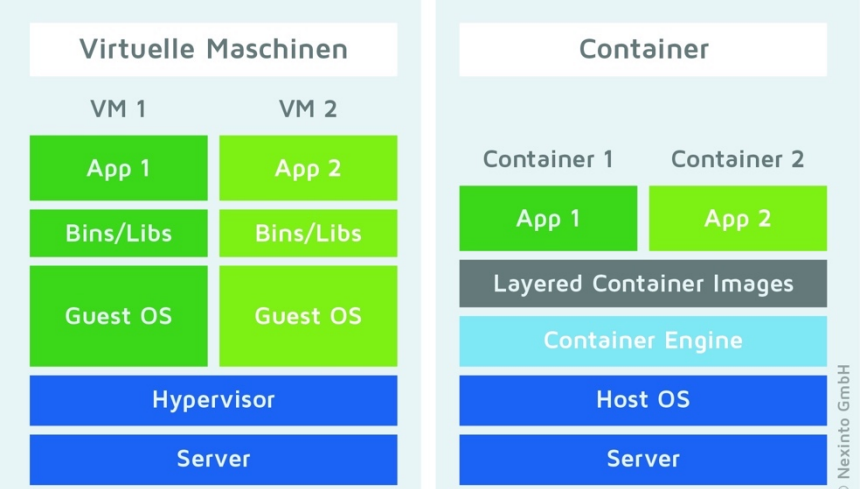


Abb. 2: Virtuelle Maschinen versus Container

Quelle: JaxEnter (28. August 2017): https://jaxenter.de/docker-vs-vm-54816

3.3 Einsatzgebiete von Containern

Anhand von möglichen Anwendungsfällen sollen die Vorzüge dieser Virtualisierungstechnologie nochmals untermauert werden.

3.3.1 Anwendung zur Unterstützung von Online-Transaktionen[12]

Ein Nutzer möchte auf einer Webseite ein Artikel bestellen. Bevor er die Zahlung tätigen kann, sind vorher noch einige Schritte die gemacht werden müssen notwendig. Zunächst muss er sich mit seinen Zugangsdaten auf der Webseite anmelden und anschließend auf den Artikel draufklicken. Folglich muss er weitere Anweisungen befolgen, bis er letzten Endes die Transaktion durchführen kann.

Diese vier Schritte die er durchläuft stellen einzelne Abschnitte, sogenannte Microservices, einer Anwendung dar. Jeder dieser Microservice befindet sich isoliert in einem Container. Kommt es nun bei einem Schritt zu einem Fehler, z.B. der gewünschte Artikel kann nicht angeklickt werden, so wird anhand der Verteilung sichergestellt, dass nicht die gesamte Anwendung ausfällt. In solch einem Fall greift der jeweilige Dienst auf ein alternatives System zurück und versucht von dort aus die Anfrage auszuführen. Jeder Container ist somit auf einen Abschnitt spezialisiert und stellt damit sicher, dass die Anwendung zuverlässig abläuft und problemlos genutzt werden kann.

[12]Kim, J. (23. August 2017). Die Vorteile der Container-Technologie richtig nutzen. Abgerufen am 12. April 2018 von www.it-daily.net: https://www.it-daily.net/it-management/software-development/16456-die-vorteile-der-container-technologie-richtig-nutzen

3.3.2 Umgebungswechsel einer Applikation[13]

Eine Softwareanwendung durchläuft mehrere Ebenen, bis diese letztendlich produktiv genutzt werden kann. Zunächst muss diese entwickelt und mit Hilfe eines CI – Tools gebaut werden. Sobald dieser Prozess abgeschlossen wurde, müssen Softwaretests durchgeführt werden. Dabei wird sichergestellt, dass die Anwendung den formulierten Anforderungen und den Qualitätsansprüchen gerecht wird. Dieser Vorgang wird mehrmals durchlaufen, wobei immer wieder neue Versionen von der Applikation entstehen. Anschließend kann die Software auf der Produktionsumgebung installiert werden.

Gerade bei komplexen Umgebungen mit vielen Abhängigkeiten können Deployments zu einer langwierigen Angelegenheit werden.

Der Einsatz von Containern schafft in diesem Fall Abhilfe und ermöglicht die Ausführung eines schnellen Deployment von neuen Releases, da Containerimages einen geringen Bedarf an Speicherplatz in Anspruch nehmen - *nähere Erläuterung in Abschnitt 4.4.1*. Zudem fällt die Netzwerklast für den Umgebungswechsel geringer aus und die Anwendung kann schneller gestartet werden, da nicht extra ein Betriebssystem hochgefahren werden muss.

4. Containerbasierte Virtualisierung mittels Docker

Nachdem in den vorangegangenen Kapiteln allgemein auf containerbasierte Virtualisierung eingegangen wurde, bedarf es im Folgenden auf Basis von Docker die Containertechnologie näher zu betrachten.

4.1 Was ist Docker

Die im Jahre 2013 erstmals publizierte Version von Docker gilt gegenwärtig als einer der bekanntesten Virtualisierungslösungen auf Containerbasis. Veröffentlicht wurde diese durch das Unternehmen dotCloud. Der Begriff Docker wird besonders mit den Eigenschaften leichtgewichtig und komfortabel assoziiert. Mit diesen Eigenschafften hebt sich Docker weitgehend von anderen Containertechnologien ab.[14]

Mit der Open-Source-Lösung zielt die Docker-Technologie auf die automatisierte Bereitstellung von Anwendungen, welche in Containern organisiert sind, ab.[15] Um das automatisierte Verfahren zu realisieren greift dieser auf die Eigenschaften des Linux-Kernels zu und trennt die in Containern

[13] Skerlec, M. (12. Juli 2017). *Was haben Frachtcontainer mit IT zu tun?* Abgerufen am 20. April 2018 von blog.bwi.de: https://blog.bwi.de/was-haben-frachtcontainer-mit-it-zu-tun/

[14] Willhöft. (16. Februar 2016). *Dienste mit Docker-Containern betreiben*. Abgerufen am 02. Mai 2018 von www.willhoeft-it.com: http://www.willhoeft-it.com/2016/02/16/docker-101.html

[15] Büst, R. (28. Januar 2016). *Wirtualisierung - Software Defined X*. Abgerufen am 13. April 2016 von www.computerwoche.de: https://www.computerwoche.de/a/docker-schafft-sich-ein-eigenes-oekosystem,3222086

verpackten Anwendungen komplett von der physischen Umgebung. Unter dieser Technik wird der parallele Betreib von Anwendungen, die unterschiedliche Anforderungen an das System stellen, ermöglicht.

Das Portieren eines Containers von einer Umgebung in die nächste, stellt mittels ihrer Leichtgewichtigkeit keine Schwierigkeiten dar, denn der virtuelle Behälter beinhaltet neben der jeweiligen Anwendung auch alle zur Ausführung notwendigen Abhängigkeiten.

Die immer populär werdende Containertechnologie ermöglicht es Anwendungen unabhängig von den Betriebsplattformen zu entwickeln. [16] Damit wird sichergestellt, dass die Applikation in jedem Fall auf den unterschiedlichsten Plattformen funktionieren wird. Zudem ist diese Technologie einfach zu bedienen und somit können Applikationen auf einem schnellen Weg anderen Anwendern bereitgestellt werden, was einen höheren Produktivitätsgrad zur Folge hat.

Im Vergleich zu anderen Virtualisierungslösungen, wie der virtuellen Maschine, wird mittels Docker nicht die komplette Hardware eines Rechners virtuell dargestellt. Auf diese Weise ist der Docker-Container auf weniger Ressourcen vom Host-System angewiesen, da nicht zusätzlich die in der virtuellen Maschine vorhandenen Betriebssysteme neben dem Hostbetriebssystem hochgefahren werden müssen.

Besonders in Unternehmen die Software für die Cloud entwickeln hat Docker einen hohen Stellenwert bezogen. Docker unterstützt durch seine hohe Skalierbarkeit, Flexibilität und Portierbarkeit, Anwendungen auf einer Cloud-Plattform bereitzustellen, ohne dass dabei ein hoher Mehraufwand[17] betrieben werden muss.[18]

4.2 Architektur von Docker[19]

In diesem Abschnitt sollen die Prinzipien die sich hinter dem Begriff Docker verbergen näher erläutert werden. Zunächst wird der grundsätzliche Aufbau von Docker und anschließend die Architektur des Linux Containers erläutert.

Docker stellt eine System-Level-Virtualisierung dar, die auf dem Betriebssystem des Hostsystems implementiert wird. In *Abbildung 3* wird der Grundaufbau von Docker und die Kommunikationswege der einzelnen Komponenten veranschaulicht dargestellt. Zunächst soll auf den Docker Host eingegangen werden. Er stellt das System dar auf dem Docker installiert ist. Auf dem Host ist der Docker

[16] Nemeth, A. (24. Mai 2016). *Mit Docker vom Container-Konzept in der IT profitieren.* Abgerufen am 04. Mai 2018 von cloud.telekom.de: https://cloud.telekom.de/blog/docker-softwareentwicklung/

[17] Linthicum, D. (Oktober 2015). *Swarm und Kubernetes: Zwei Tools zur Orchestrierung von Docker-Container.* Abgerufen am 25. April 2018 von www.searchdatacenter.de: https://www.searchdatacenter.de/tipp/Swarm-und-Kubernetes-Zwei-Tools-zur-Orchestrierung-von-Docker-Containern

[18] Nemeth, A. (24. Mai 2016). *Mit Docker vom Container-Konzept in der IT profitieren.* Abgerufen am 04. Mai 2018 von cloud.telekom.de: https://cloud.telekom.de/blog/docker-softwareentwicklung/

[19] Mouat, A. (September 2016). *Grundlagen von Docker*. Abgerufen am 30. April 2018 von www.dpunkt.de: https://www.dpunkt.de/common/leseproben//12553/5_Grundlagen%20von%20Docker.pdf. S. 36

Daemon implementiert. Unter seinen Aufgabenbereich fällt die Erstellung, die Ausführung und die Überwachung von IT-Containern. Zusätzlich ist er auch für die Images, die gebaut und gespeichert werden, zuständig. Regulär ist das Host - OS für den Start eines Daemons zuständig. Manuell kann er auch über das Kommando >>*docker daemon*<< gestartet werden.

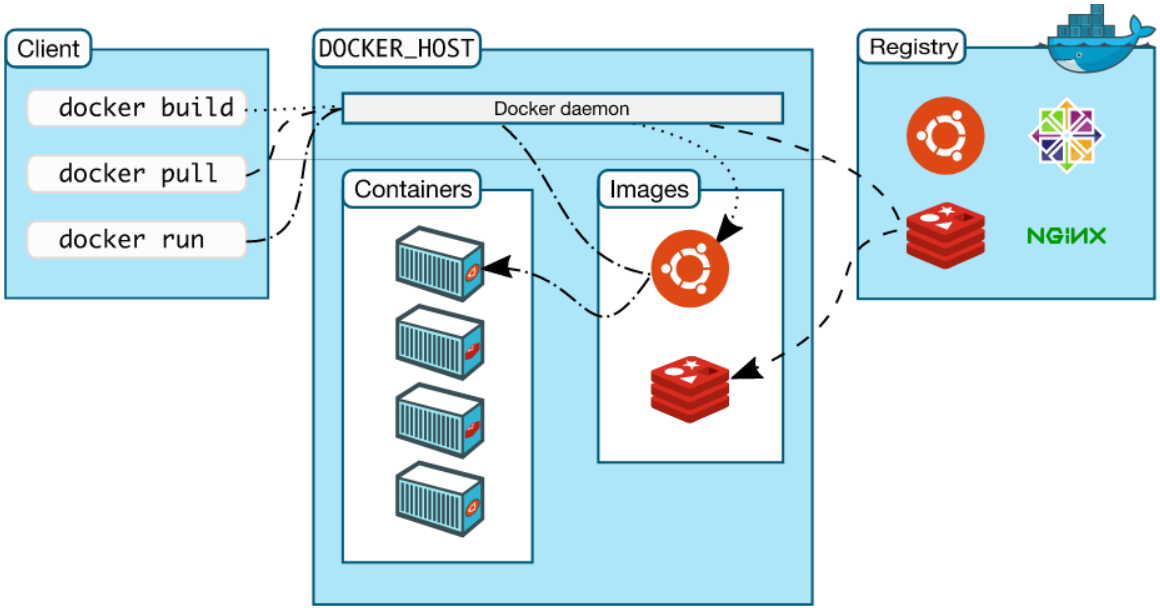


Abb. 3: Architektur von Docker

Quelle: docker docs: http://www.sebastianviereck.de/wp-content/uploads/2017/10/Bildschirmfoto-2017-10-01-um-14.14.42.png

Die Kommunikationsschnittstelle stellt die REST-API dar. Diese baut die Verbindung zwischen dem Docker Client und dem Docker Daemon auf und teilt diesen die notwendigen Anweisungen mittels HTTP mit. Die Schnittstelle selbst ist so ausgelegt, dass der Entwickler einer Anwendung anstatt zunächst auf den Client zurückzugreifen, direkt mit dem Daemon arbeiten kann.[20] Die Docker-Engine befindet sich auf dem Client, der wiederum den Docker Host steuert. Wird nun ein Programm entwickelt, so werden die notwendigen Befehle in der Kommandozeile des Clients eingegeben und per Rest-Schnittstelle an den Docker-Host weitergegeben, wobei der Daemon die Anforderungen des Entwicklers, wie z.B. das Bauen eines Image, erledigt.

Der letzte Bestandteil von Docker ist die Docker Registry. Diese kann als eine Bibliothek betrachtet werden, in der fertig erstellte Images abgelegt, verwaltet und verteilt werden. Im *Kapitel 4.3.3* wird die Registry ausführlicher beschrieben.

[20] Ebda.

4.3 Architektur von Linux

Bevor in die Tiefe eingegangen wird, müssen zunächst die Linux-Kernel-Funktionalitäten näher betrachtet werden.

Ein Linux-Container, auch LXC genannt, besteht aus vielen verschiedenen Funktionalitäten, derer sich Docker bedient, um einzelne Anwendungen in Containern voneinander zu isolieren[21] *siehe Abbildung 4*.

In den Anfängen der Docker-Generation bediente sich die Technologie an den Kernel-Features einer plattformunabhängigen Schnittstelle von LXC, bis diese Schnittstelle durch den von Docker entwickelten „libcontainer“, der als Interface fungiert, ersetzt wurde. Seit der Eigenimplementierung des libcontainers, gilt dieser als Standard für diese Funktionalitäten und wird von großen Unternehmen wie Redhat, Google, usw. unterstützt.

Im Rahmen dieser Arbeit wird hier nur auf die wichtigsten Linux-Kernel-Funktionalitäten eingegangen. Dazu zählen cgroup, namespaces und UFS.

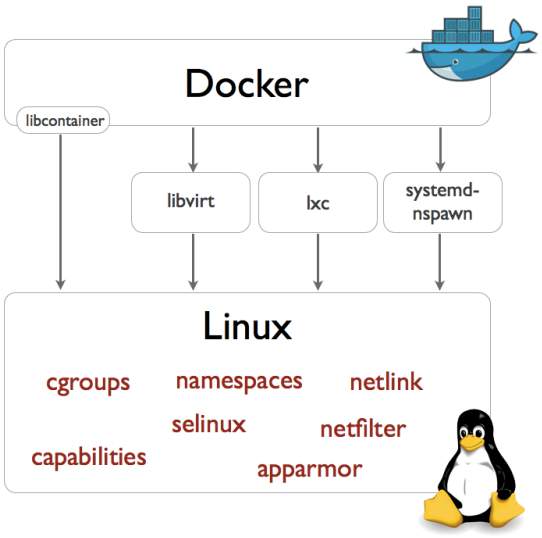


Abb. 4: Linux Funktionalitäten

Quelle: http://xmlandmore.blogspot.com (07. November 2017):

http://xmlandmore.blogspot.com/2015/11/security-and-isolation-implementation.html

[21] Pustina, L. (30. März 2015). *Docker Basics: Einführung in die System-Level-Virtualisierung.* Abgerufen am 08. Mai 2018 von entwickler.de: https://entwickler.de/online/development/docker-basics-system-level-virtualisierung-125514.html

Einer der wichtigsten Kernfunktionalitäten ist cgroups (control groups).[22] Mit dieser Funktion werden die Ressourcen für die jeweiligen Anwendungen verwaltet. Dazu zählen beispielsweise die CPU-Auslastung, Speichermengen die genutzt werden, etc. Grundsätzlich ist cgropus für die Isolation von Containern zuständig.

Eine weitere Kernfunktion stellt namespaces dar.[23] Namespaces ermöglicht, geschlossene Umgebungen gegenüber der jeweiligen ausgeführten Anwendung zu simulieren. „[…] [S]ie stellen sicher, dass Dateisystem, Hostname, Benutzer, Netzwerk und Prozesse vom Rest des Systems getrennt sind []"[24] Namespaces kümmert sich um die Sicherheitszugriffe für die Container.

Abschließend ist die Funktionalität UFS (Unit File System) zu erwähnen. Diese Funktion bietet die Verwaltung von hierarchischen Dateisystemen eines Images. Dabei werden besondere Dateisysteme eingesetzt, wie „[…] AUFS, devicemapper, BTRFS, ZFS, VFS oder Overlay. [ ]" [25]

4.4 Terminologie von Docker

4.4.1 Docker-Image

Eine Docker Image ist ein Abbild von einem Docker Container, die wiederum von einer Umgebung zur anderen leicht portierbar ist. [26] Die Docker Image enthält in Form einer Textdatei, dem sogenannten Dockerfile, eine Schritt-für-Schritt Anleitung für die Erzeugung eines Containers. Die Notwendigkeit eines Dockerfiles entfällt, wenn eine fertig erstellte Image aus der Registry runtergeladen wird.

Der Aufbau eines Image gleicht einem Schichtenmodel. Dabei werden mehrere Dateisysteme, die wiederum auch als Images bezeichnet werden, übereinandergestapelt. Jede dieser Schicht wird jeweils von der anderen getrennt erzeugt und ist zudem schreibgeschützt. In *Abbildung 5* wird ein solcher Aufbau beispielhaft dargestellt. Ausgehend von der Basis Image, die die Grundlage bildet, werden alle weiteren Images auf dieses nacheinander drauf gestapelt. Aus diesen drei Images (debian, emacs, apache) kann anschließend ein Container gestartet werden. Dazu wird während der Laufzeit eine weitere beschreibbare Ebene oben draufgelegt.

[22] Mouat, A. (September 2016). *Grundlagen von Docker*. Abgerufen am 30. April 2018 von www.dpunkt.de: https://www.dpunkt.de/common/leseproben//12553/5_Grundlagen%20von%20Docker.pdf

[23] Ebda.

[24] Ebda., S. 37

[25] Ebda.

[26] Augsten, S. (25. April 2017). *Was sind Docker-Container*. Abgerufen am 03. Mai 2018 von www.dev-insider.de: https://www.dev-insider.de/was-sind-docker-container-a-597762/

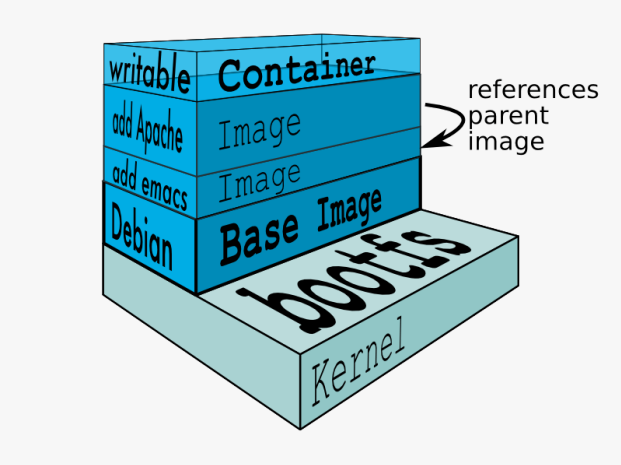


Abb. 5: Docker Dateisystem

Quelle: FreeCodeCamp (4. März 2016):

https://medium.freecodecamp.org/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b

Möchte man nun, nachdem aus der Image ein Container gestartet wurde, Änderungen vornehmen, so werden diese nicht innerhalb der Schicht vorgenommen,[27] sondern es wird ein neues Image erzeugt. Dieses referenziert auf den Vorgänger und enthält bzw. speichert ausschließlich die Unterschiede, die sich aus den beiden Schichten ergeben.

Um den grundsätzlichen Aufbau eines Dockerfiles zu verstehen, wird im Folgenden auf die wichtigsten Befehle[28] eingegangen. In der Textdatei wird Reihe für Reihe zunächst das Kommando und anschließend der gewünschte Parameter eingetragen, wobei der „FROM"-Befehl immer an erster Stelle stehen und mindestens vorhanden sein muss. Mit den anschließenden Befehlen wird das Image mit Leben gefüllt.

- FROM: [29]
  Dieser Befehl beschreibt, welches Image die Grundlage für das neu erzeugte Image bildet.

- RUN:
  Mittels dieses Befehls werden die Parameter durch Shell-Befehle ersetzt. Das Besondere hier ist, dass mehrere Befehle ausgeführt werden können. Zu beachten ist dabei, dass die Anzahl der „RUN"-Befehle möglichst geringgehalten werden soll, da

[27] Roden, G. (28. Februar 2014). *Anwendungen mit Docker transportabel machen*. Abgerufen am 28. April 2018 von www.heise.de: https://www.heise.de/developer/artikel/Anwendungen-mit-Docker-transportabel-machen-2127220.html?seite=all

[28] Tiersch, F. (21. September 2015). Docker – Teil 4: Das Dockerfile. Abgerufen am 05. Mai 2018 von www.ab-heute-programmieren.de: https://www.ab-heute-programmieren.de/docker-teil-4-das-dockerfile/

[29] Ebda.

für jeden dieser eine neue Schicht erstellt wird, was wiederum zu einem erhöhten Speicherbedarf führt. Demnach sollte besonders Acht darauf gegeben werden, dass hier mehrere Parameter in einem Befehl untergebracht werden.

- CMD:[30]
  Dem aus der Image erstellten Container wird anhand des „CMD“-Befehls ein Standardkommando das ausgeführt wird, zur Verfügung gestellt. Beim Start des Containers, kann „CMD“ überschrieben werden. In einer Docker Image sollte nur ein „CMD“-Befehl ausgeführt werden. Werden dennoch mehrere aufeinanderfolgende Befehle eingefügt, so wird nur der Letzte betrachtet und ausgeführt.

- ENTRYPOINT
  Mit dem Befehl „ENTRYPOINT“ wird festgelegt, welche ausführbare Datei und Argumente standardmäßig bei der Ausführung eines Containers laufen sollen. Konkret wird ein Start-Skript für den Containerstart angegeben.

- EXPOSE:[31]
  Dieser Befehl ist für die Portfreigabe zuständig. Mittels eines Ports wird ein Kommunikationskanal innerhalb des Containers und auch zur Außenwelt hergestellt.

- ADD[32]
  Eine Datei oder das Kontextverzeichnis, welches sich auf dem Host befindet, wird anhand dieser Anweisung dem Dateisystem eines Images hinzugefügt. Dabei muss zunächst die Datei oder Verzeichnis angegeben werden und im Anschluss das Ziel.

- WORKDIR[33]
  Die bereits genannten Befehle gehen immer erst vom Referenzverzeichnis aus. „WORKDIR“ ermöglicht ein neues Arbeitsverzeichnis anzugeben, dass für alle nachfolgenden Befehle bzw. beim nächsten Aufruf dieses Befehls ausgeführt wird.

[30] Mouat, A. (September 2016). *Grundlagen von Docker*. Abgerufen am 30. April 2018 von www.dpunkt.de: https://www.dpunkt.de/common/leseproben//12553/5_Grundlagen%20von%20Docker.pdf. S. 48

[31] Tiersch, F. (21. September 2015). Docker – Teil 4: Das Dockerfile. Abgerufen am 05. Mai 2018 von www.ab-heute-programmieren.de: https://www.ab-heute-programmieren.de/docker-teil-4-das-dockerfile/

[32] Ebda.

[33] Mouat, A. (September 2016). *Grundlagen von Docker*. Abgerufen am 30. April 2018 von www.dpunkt.de: https://www.dpunkt.de/common/leseproben//12553/5_Grundlagen%20von%20Docker.pdf. S. 51

- USER

  Hier wird ein Benutzer definiert, in dessen Kontext der im Container enthaltene Prozess gestartet wird.

Nun kann das Image aus dem Dockerfile gebaut werden. Dazu wird der „build"-Befehl verwendet. Jeder Befehl wird schrittweise ausgeführt und zwischengespeichert. Sollte es bei der Ausführung zu einem Fehler kommen, so wird der Container gestoppt und gelöscht. Die Zwischenspeicherung hat den positiven Effekt, dass bei Neustart eines Image bei dem Schritt weitergearbeitet werden kann, bei dem es zu der Unterbrechung gekommen ist.

Wurde ein Image erstellt, so kann es anderen Anwendern leicht über Docker Hub weitergereicht werden. Damit bildet dieses Image die Grundlage für die Erzeugung weiterer Container, ohne dass der original entwickelte Container zuvor gesehen wurde.

4.4.2 Docker Container

Docker Container sind isoliert voneinander laufende Prozesse, die aus einem Image heraus gestartet werden. Sobald also ein Image mit dem „run"-Befehl ausgeführt wird, ist von einem Container die Rede. Dieser Vorgang kann mit der objektorientierten Programmierung verglichen werden. Hier dient eine Klasse als Vorlage für ein konkretes Objekt. Dabei stellt das Image die Klasse dar und wiederum der Container eine Instanz eines Images.[34] In einem Container ist die Anwendungssoftware inkl. Abhängigkeiten organisiert. Dazu zählen unter anderem der Code, die Laufzeitumgebung, System-Tools und Bibliotheken.[35] Diese Abhängigkeiten werden von dem Image zur Verfügung gestellt.

4.4.3 Docker-Registry

Nachdem das Docker Image gebaut wurde, kann dieses in die Docker Registry (Docker Hub) transportiert werden. Docker Hub ist eine cloudbasierte Plattform, die eine große Menge an Repositories enthält und wiederum in diesem die Images verwaltet werden. Dabei gibt es einen privaten und einen öffentlichen Bereich, in dem die Images verwaltet werden können. Im privaten Bereich hat der Verantwortliche und der Personenkreis, der von ihm festgelegt wurde, die Möglichkeit diese zu nutzen. Wobei im öffentlichen Repository Jedermann auf die hochgeladenen Images zugreifen kann.

Das Repository ermöglicht die Arbeit von mehreren Anwendern an einem Image. Das Distributionssystem kann mit dem „Git Hub" verglichen werden. Hier werden Versionen eines Quellcodes in ein Repository hochgeladen und den Nutzern in der aktuellsten Version zur Verfügung gestellt.

---

[34] Tiersch, F. (07. September 2015). Docker – Teil 2: Das Arbeiten mit Containern. Abgerufen am 29. August 2018 von www.ab-heute-programmieren.de: https://www.ab-heute-programmieren.de/docker-teil-2-das-arbeiten-mit-containern/

[35] Caldie, S. (03. August 2017). *Docker-Container sind dabei die IT-Industrie zu verwandeln*. Abgerufen am 21. April 2017 von www.krollontrack.de: https://www.krollontrack.de/blog/docker-containers-transform-it/7330

4.4.4 Docker Volume[36]

Die Lebensdauer eines Containers ist von seiner Beschäftigung abhängig. Wurde ein Befehl ausgeführt, so wird auch der Container beendet. Kommt es zur Beendigung, so verschwinden auch die darin gehaltenen Daten.[37] Da aber in der Praxis mit Anwendungen gearbeitet wird, die ständig Verfügbar sein müssen, sogenannte persistente Daten, wird eine Speicherschicht zwischen dem Host-System und dem Container geschalten, die die Existenz der Daten sichert.[38]

Docker Volume wird dann initialisiert, wenn ein Docker Container eingerichtet wird. Dabei hat jedes Volume Einhängepunkte, sogenannte Mount Points, mit denen Dateisysteme verfügbar gemacht werden. Sollten in einem Image bereits Daten und Dateisysteme vorhanden sein, so werden diese in das Volume reinkopiert und können von dort aus vom Anwender genutzt werden.

Wurde ein Volume einmal erstellt, so kann dieses für mehrere Container hergenommen werden.

Sobald der aktuell ausgeführte Container gelöscht wird, bleiben die im Volume enthaltenen Daten erhalten und „[…] können an den nächsten Container angehängt werden. […]“[39]

Um ein neues Volume zu erstellen, muss bei der Ausführung des „run-Befehls“ der Zusatzparameter >> *-v* << angegeben werden. Zudem besteht noch die Möglichkeit mehrere Volumes miteinander zu verknüpfen. Alle Volumes werden anhand dieses Befehls an einen Container angehängt.

4.4.5 Docker Compose

Die Verwaltung von Systemen ist teilweise mit viel Arbeit verbunden. Sobald ein System auf dem Host ausgeführt werden soll, muss für jeden darin enthaltenen Service eine Docker Image erstellt, gebaut und anschließend zum Laufen gebracht werden. Möchte man einen Docker Container stoppen, dann muss darauf geachtet werden, dass bevor es zum nächsten Start kommt die zu löschenden Container entfernt werden.[40] Weiterhin ist es oftmals notwendig, dass zwischen mehreren Containern Kommunikationskanäle aufgebaut werden müssen, da z.B. ein Webserver von der Datenbank gewisse Daten benötigt.

---

[36] Bardy, C. (August 2017). *Docker: Nutzen Sie Optionen für persistenten Speicher*. Abgerufen am 06. Mai 2018 von www.searchstorage.de: https://www.searchstorage.de/lernprogramm/Docker-Nutzen-Sie-Optionen-fuer-persistenten-Speicher

[37] Tiersch, F. (07. September 2015). *Docker – Teil 2: Das Arbeiten mit Containern.* Abgerufen am 29. August 2018 von www.ab-heute-programmieren.de: https://www.ab-heute-programmieren.de/docker-teil-2-das-arbeiten-mit-containern/

[38] Evans, C. (Januar 2018). *Das ist beim Einsatz von Docker-Container wichtig*. Abgerufen am 28. April 2018 von www.searchstorage.de: https://www.searchstorage.de/tipp/Das-ist-beim-Einsatz-von-Docker-Containern-wichtig

[39] Arbezzano, G. (06. September 2017). *Grundkurs Docker: Eine praktische Einführung in die Welt der Container.* Abgerufen am 17. April 2018 von jaxenter.de: https://jaxenter.de/einfuehrung-docker-tutorial-container-61528

[40] Vitz, M. (04. April 2017). *Docker Compose - Komplette Systeme mit Docker managen*. Abgerufen am 13. April 2018 von www.innoq.com: https://www.innoq.com/de/articles/2017/04/docker-compose/

Um dies Vorgänge zu vereinfachen bzw. zu automatisieren, wird von Docker das Tool Docker-Compose zur Verfügung gestellt.[41] Anhand dieses Werkzeugs wird ermöglicht, dass mehrere Container innerhalb einer Konfigurationsdatei, der sogenannten YAML, erstellt werden können und wiederum mit einem einzigen Befehl (docker compose -up -d) ausgeführt werden können, wobei -d optional ist. Dieser Parameter sagt aus, dass die Container im Hintergrund ausgeführt werden.

Im Folgenden wird der Aufbau einer YAML Datei näher erläutert.

Auf oberster Ebene wird zunächst die aktuelle Version angegeben. Anschließend werden alle notwendigen Services eingetragen. Dabei wird erst der Container angegeben, danach folgen die Images, der Port, die Umgebungsvariablen und „[…] die richtigen Verlinkungen zwischen den Services werden hier eingetragen." Diese Eigenschaften sind notwendig, damit Docker eine Image finden oder auch bauen kann.[42] Das besondere an Docker Compose ist, dass das Netzwerk automatisch aufgebaut wird ohne, dass eine Verlinkung notwendig ist. Eine Beispieldeklaration wird anhand der *Abbildung 6* aufgezeigt. Wird nun beispielsweise anhand des Befehls >> *docker compose down* << versucht einen Container zu stoppen, übernimmt Docker Compose automatisch die Aufgabe des Entfernens der Container.[43]

```
version: '3'

services:
  database:
    image: redis
  frontend:
    build: .
    links:
      - database
    environment:
      - SPRING_REDIS_HOST=database
  load-balancer:
    image: dockercloud/haproxy
    links:
      - frontend
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    ports:
      - 80:80
```

Abb. 6: Docker Compose – Beschreibung des Systems

Quelle: INNOQ (04. April 2017):
https://www.innoq.com/de/articles/2017/04/docker-compose/

[41] Ebda.

[42] Vitz, M. (04. April 2017). *Docker Compose - Komplette Systeme mit Docker managen*. Abgerufen am 13. April 2018 von www.innoq.com: https://www.innoq.com/de/articles/2017/04/docker-compose/

[43] Ebda.

5. Anwendungsbeispiel

In diesem Abschnitt wird anhand eines Anwendungsbeispiels aufgezeigt, wie man aus einem Dockerfile ein Image erstellt und wie aus dieser anschließend ein Container gestartet wird. Dabei wird auf die einzelnen Kommandozeilen eingegangen.

5.1 Erste Schritte

Für das folgende Anwendungsbeispiel, wurde die Docker–Version 18.03.1 auf dem Arbeitsrechner eingerichtet.

Zudem wurde eine Spring-Boot-Applikation erstellt, die einen String bei Aufruf des Rest-Services zurückliefert. Diese Anwendung wird im nachfolgenden als JAR in einen Docker Container gepackt und abschließend zur Ausführung gebracht.

5.2 Erstellen eines Dockerfiles

Zunächst soll eine Anweisungsdatei „Dockerfile“ deklariert werden. In *Abbildung 7* wird abgebildet, in welchem Teil des Verzeichnisses sich das Dockerfile befindet.

Zu beachten ist, dass der Dateiname jedes Dockerfiles immer mit einem Großbuchstaben anfangen muss.

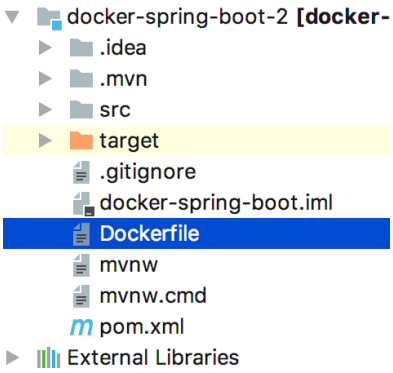


Abb. 7: Kontextverzeichnis

Sobald die Textdatei erstellt wurde, kann der Bauplan für das Erzeugen eines Containers aufgestellt werden. *Abbildung 8* zeigt das fertiggestellte Dockerfile.

Dockerfile

```
1  FROM openjdk:latest
2  ADD target/docker-spring-boot.jar docker-spring-boot.jar
3  EXPOSE 8085
4  ENTRYPOINT ["java", "-jar", "docker-spring-boot.jar"]
```

Abb. 8: Dockerfile

- Beginnend mit dem FROM-Befehl, wird hier die Auswahl der Basis Image getroffen. In diesem Beispiel bildet >> *openjdk* << die Grundlage für das zu erzeugende Image. Hinter dem Doppelpunkt wird ein Tag als weitere untergeordnete Spezifizierung gesetzt.[44] Dieser trennt die Basis Image von seiner Version. Wird der >> *tag* << auf >> *latest* << gesetzt, so wird immer die neueste Version von >> *openjdk* << hergenommen.

- Sollte sich die Basis Image nicht bereits lokal auf dem Rechner befinden, so wird diese von Docker automatisch aus der Registry „Docker Hub" herunterlageladen. Konkret sucht Docker zunächst im lokalen Repository nach der Image >> *openjdk* << mit der aktuellsten Version. Erst wenn diese lokal nicht gefunden wird, verbindet er sich mit der zentralen Registry und führt den Download durch.

- Im nächsten Schritt wird mit dem Kommando „ADD" die mit Maven gebaute JAR-Datei in das Image eingefügt. Hier wird der Verzeichnispfad angegeben, wo sich die JAR-Datei auf dem Hostsystem befindet >> *target/docker-spring-boot.jar* <<. Dieser Pfad wird in das gleichnamige
  >> *docker-spring-boot.jar* << Dateisystem vom Container reinkopiert.

- In der darauffolgenden Zeile, wird die Kommunikation innerhalb und außerhalb des Containers mit dem Port 8085 hergestellt. Hier wird Docker mitgeteilt, dass der Service auf den Port 8085 horchen soll.[45]

- Mittels „ENTRYPOINT" wird festgelegt, welches Start-Skript standardmäßig bei der Ausführung eines Containers ausgeführt werden soll. Dazu werden die Argumente *siehe Abbildung 9* festgelegt.

```
ENTRYPOINT ["java", "-jar", "docker-spring-boot.jar"]
```

Abb. 9: Entrypoint

[44] Vitz, M. (04. April 2017). *Docker Compose - Komplette Systeme mit Docker managen*. Abgerufen am 13. April 2018 von www.innoq.com: https://www.innoq.com/de/articles/2017/04/docker-compose/

[45] docs, d. (kein Datum). *Docker run reference*. Abgerufen am 03. Mai 2018 von docs.docker.com: https://docs.docker.com/engine/reference/run/#operator-exclusive-options

5.3 Image aus dem Dockerfile erzeugen

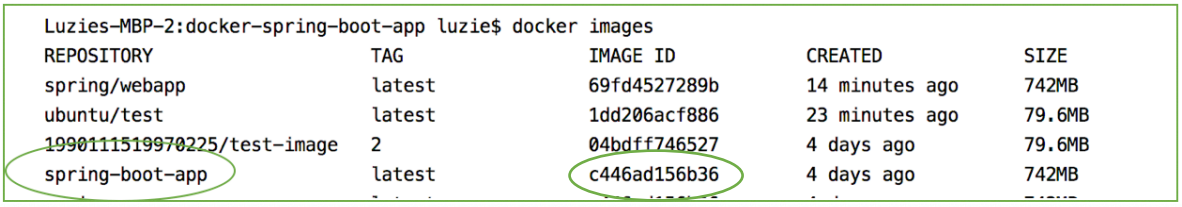
Anhand der fertigen Dockerfile kann nun das Image gebaut werden.

- Dafür muss das Kommando >> *docker build* << ausgeführt werden *siehe Abbildung 10*. Mit dem >> -f << - Parameter, kann der Ort der Dockerfile festgelegt werden. Wiederum wird mit dem >> *-t* << -Flag dem Image der Name „spring-boot“ vergeben.
- Besonders wichtig ist der Punkt, der an das Ende der Kommandozeile mit angefügt wird. Dieser stellt das aktuelle Verzeichnis dar, welches hergenommen wird um das Image zu erzeugen.
- Wird der >> *build* << -Befehl ausgeführt, so wird zunächst der Build-Kontext an den Daemon weitergeleitet. Danach werden die fünf Befehle aus dem Dockerfile der Reihe nach durchlaufen. Für jeden Schritt der ausgeführt wird, legt Docker einen temporären Container an, in dem der Befehl ausgeführt wird bzw. wie in *Abbildung 10* aufgeführt ist, ein „cache“ angelegt und die Befehle in diesem abgelegt. Sobald ein Schritt ausgeführt wurde, wird der Container anschließend wieder gelöscht. Nachdem der fünfte Befehl ausgeführt wurde, wird die endgültige Image erstellt, der wiederum eine ID zugewiesen wird.

```
Luzies-MBP-2:docker-spring-boot-app luzie$ docker build -f Dockerfile -t spring-boot-app .
Sending build context to Docker daemon  16.35MB
Step 1/4 : FROM openjdk:latest
 ---> c3e386dcd062
Step 2/4 : ADD target/docker-spring-boot.jar docker-spring-boot.jar
 ---> Using cache
 ---> 0e47eb6570fa
Step 3/4 : EXPOSE 8085
 ---> Using cache
 ---> cc38552fdd2a
Step 4/4 : ENTRYPOINT ["java", "-jar", "docker-spring-boot.jar"]
 ---> Using cache
 ---> c446ad156b36
Successfully built c446ad156b36
Successfully tagged spring-boot-app:latest
Luzies-MBP-2:docker-spring-boot-app luzie$
```

Abb. 10: Build Command

- Um sicherzustellen, dass das Image nach dem Download lokal vorhanden ist, wird der Befehl >> *docker images* << ausgeführt. Dabei werden alle auf dem Hostsystem befindlichen Images aufgelistet.



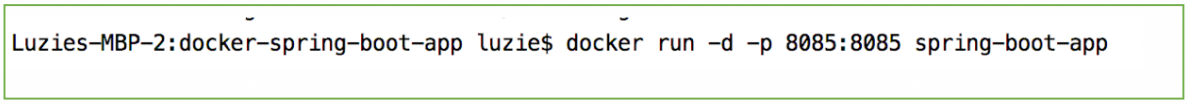
```
Luzies-MBP-2:docker-spring-boot-app luzie$ docker images
REPOSITORY                      TAG                 IMAGE ID            CREATED             SIZE
spring/webapp                   latest              69fd4527289b        14 minutes ago      742MB
ubuntu/test                     latest              1dd206acf886        23 minutes ago      79.6MB
1990111519970225/test-image     2                   04bdff746527        4 days ago          79.6MB
spring-boot-app                 latest              c446ad156b36        4 days ago          742MB
```

Abb. 11: Image Command

Wenn sich mehrere Container auf dem Host befinden, die die selbe Basis Image nutzen möchten, so wird nicht für jeden dieser Container eine einzelne Image angelegt, sondern alle Container teilen sich die benötigte Image. Anhand dieses Vorgangs wird Speicherplatz gespart.

5.4 Docker Container starten

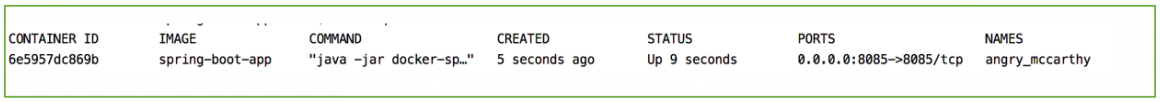
Nachfolgend kann aus der erstellten Image namens „spring-boot“ der Container gestartet werden. Dazu wird das Kommando >> *docker run* << ausgeführt. Mit >> -p << wird der Port angegeben auf dem die Webanwendung aufgerufen wird (8085) und hinter dem Doppelpunkt wird der Port vom Container angesprochen (8085).



```
Luzies-MBP-2:docker-spring-boot-app luzie$ docker run -d -p 8085:8085 spring-boot-app
```

Abb. 12: Run Command

Möchte man sich alle Container, die sich auf dem lokalen Repository befinden ansehen, so wird der Befehl >> *docker ps -a* << ausgeführt. Um überprüfen zu können, welcher Container gerade läuft benötigt man das Kommando >> *docker ps* <<.



```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
6e5957dc869b        spring-boot-app     "java -jar docker-sp…"   5 seconds ago       Up 9 seconds        0.0.0.0:8085->8085/tcp   angry_mccarthy
```

Abb. 13: Ps Command

Abschließend kann das Image vom lokalen Repository in die Registry hochgeladen werden.

Folgende Befehle müssen eingegeben werden:

- >> *docker login* << für die Anmeldung auf Docker Hub. Mit >> *docker logout* << meldet man sich wieder ab.

```
Luzies-MBP-2:docker-spring-boot-2 luzie$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: 1990111519970225
Password:
Login Succeeded
Luzies-MBP-2:docker-spring-boot-2 luzie$
```

Abb. 14: Login Command

- Nach erfolgreicher Anmeldung muss zunächst ein >> *tag* << für das Image gesetzt werden.
- Anhand des >> *commit* << -Kommandos, kann dieser gesetzt werden. „/test“ entspricht in diesem Beispiel dem >> *tag* <<.

```
Luzies-MBP-2:docker-spring-boot-app luzie$ docker commit 6e5957dc869b spring/test
```

Abb. 15: Commit Command

- Mit >> *docker push* << kann nun das Image in das Repository hochgeladen werden.

```
Luzies-MBP-2:docker-spring-boot-app luzie$ docker push spring/test
The push refers to repository [docker.io/spring/test]
```

Abb. 16: Push Command

6. Fazit

In diesem Zusammenhang lässt sich abschließend zusammenfassen, dass Docker vor allem in den immer komplexer werdenden IT-Landschaften eine feste Stellung eingenommen haben. Durch die Einfachheit die Docker mit sich bringt, wird den Anwendern dieser Plattform förmlich das Leben erleichtert. Besonders Systemadministratoren profitieren gänzlich von der Technologie, denn alle Container können mit den gleichen Werkzeugen behandelt werden. Zudem lässt sich ein Container einfach starten. Dazu ist lediglich nur notwendig, dass der Anwender Docker Hub besucht, im Repository sich ein beliebiges Image aussucht, dieses herunterlädt und anschließend mit einer Kommandozeile zum Laufen bringt. Außerdem startet Docker binnen weniger Sekunden. Nicht zu vergessen ist, dass Docker ein Open-Source-Projekt und damit für jeden frei zugänglich ist.

Besonders vorteilhaft ist, dass Docker auch ohne jegliche Vorkenntnisse genutzt werden kann, da die Reichweite der Communities und Foren, zum Austausch von Informationen, weit ausgedehnt ist. Zusätzlich werden in im Docker Hub zu allen Images Anleitungen mit beigefügt.

Innerhalb von wenigen Jahren hat es Docker geschafft ein riesiges Ökosystem aufzubauen, weshalb Unternehmen durch die Orchestrierungsmechanismen profitieren. Alle zwei Monate bringt Docker ein neues Release von Images raus, was den Beliebtheitsgrad immer mehr erhöht.

Der einzige Clou der sich hinter dieser Technologie verbirgt, zeichnet sich in den Sicherheitsanforderungen aus. Da Docker im Gegensatz zu z.B. virtuellen Maschinen noch relativ neu ist, versuchen Hacker immer wieder in das System einzubrechen. Deshalb ist es auch hier wie bei jedem anderen Programm wichtig, Sicherheitsupdates regelmäßig durchzuführen.

Literaturverzeichnis

Ambroise, J.-E. (21. Dezember 2016). *Virtualisierung verständlich erklärt*. Abgerufen am 09. April 2018 von blog.beronet.com: https://blog.beronet.com/de/virtualisierung-verstaendlich-erklaert

Arbezzano, G. (06. September 2017). *Grundkurs Docker: Eine praktische Einführung in die Welt der Container*. Abgerufen am 17. April 2018 von jaxenter.de: https://jaxenter.de/einfuehrung-docker-tutorial-container-61528

Augsten, S. (25. April 2017). *Was sind Docker-Container*. Abgerufen am 03. Mai 2018 von www.dev-insider.de: https://www.dev-insider.de/was-sind-docker-container-a-597762/

Büst, R. (28. Januar 2016). *Wirtualisierung - Software Defined X*. Abgerufen am 13. April 2016 von www.computerwoche.de: https://www.computerwoche.de/a/docker-schafft-sich-ein-eigenes-oekosystem,3222086

Balmes, F. (2014). *Virtualisierungstechniken: Grundlagen und Anwendung im Serverbetrieb.* Abgerufen am 09. April 2018 von books.google.de: https://books.google.de/books?id=o12wBQAAQBAJ&pg=PP4&dq=ibm+virtualisierung+geschichte&hl=de&source=gbs_selected_pages&cad=2#v=onepage&q=ibm%20virtualisierung%20geschichte&f=false

Balmes, F. (kein Datum). *Virtualisierungstechniken: Grundlagen und Anwendung im Serverbetrieb.* (D. Verlag, Hrsg.) Hamburg: Diplomica Verlag GmbH 2014.

Bardy, C. (August 2017). *Docker: Nutzen Sie Optionen für persistenten Speicher*. Abgerufen am 06. Mai 2018 von www.searchstorage.de: https://www.searchstorage.de/lernprogramm/Docker-Nutzen-Sie-Optionen-fuer-persistenten-Speicher

Caldie, S. (03. August 2017). *Docker-Container sind dabei die IT-Industrie zu verwandeln.* Abgerufen am 21. April 2017 von www.krollontrack.de: https://www.krollontrack.de/blog/docker-containers-transform-it/7330

COMPAREX, R. (26. Oktober 2017). *Es geht nicht ohne: Neue TEchnologien für den digitalen Wandel - DevOps, Container & Co.* Abgerufen am 15. April 2018 von www.comparex-group.com: https://www.comparex-group.com/web/de/de/blog/topics/software/neue-technologien-fuer-den-digitalen-wandel-devop-container-und-co.htm

docs, d. (kein Datum). *Docker run reference.* Abgerufen am 03. Mai 2018 von docs.docker.com: https://docs.docker.com/engine/reference/run/#operator-exclusive-options

Evans, C. (Januar 2018). *Das ist beim Einsatz von Docker-Container wichtig*. Abgerufen am 28. April 2018 von www.searchstorage.de: https://www.searchstorage.de/tipp/Das-ist-beim-Einsatz-von-Docker-Containern-wichtig

Literaturverzeichnis

Geißler, O., & Ostler, U. (06. September 2017). *Was sind (Software-)Container?* Abgerufen am 16. April 2018 von www.datacenter-insider.de: https://www.datacenter-insider.de/was-sind-software-container-a-638949/

Giese, M. (24. November 2015). *Multi-Tier-Applikation mit Docker Compose, Machine und Swarm*. Abgerufen am 18. April 2018 von www.heise.de: https://www.heise.de/developer/artikel/Multi-Tier-Applikationen-mit-Docker-Compose-Machine-und-Swarm-3014669.html?seite=all

Graefen, R. (24. März 2009). *Netzwerkvirtualisierung.* Abgerufen am April 2018 von www.storage-insider.de: https://www.storage-insider.de/network-virtualization-netzwerkvirtualisierung-netzwerk-virtualisierung-a-176591/

Joos, T., & Karlstetter, F. (21. September 2016). *Warum Unternehmen auf die Container-Technologie setzen sollten*. Abgerufen am 15. April 2018 von www.cloudcomputing-insider.de: https://www.cloudcomputing-insider.de/warum-unternehmen-auf-die-container-technologie-setzen-sollten-a-550570/

Kim, J. (23. August 2017). *Die Vorteile der Container-Technologie richtig nutzen*. Abgerufen am 12. April 2018 von www.it-daily.net: https://www.it-daily.net/it-management/software-development/16456-die-vorteile-der-container-technologie-richtig-nutzen

Linthicum, D. (Oktober 2015). *Swarm und Kubernetes: Zwei Tools zur Orchestrierung von Docker-Container.* Abgerufen am 25. April 2018 von www.searchdatacenter.de: https://www.searchdatacenter.de/tipp/Swarm-und-Kubernetes-Zwei-Tools-zur-Orchestrierung-von-Docker-Containern

Lipinski, D.-I. (09. November 2017). *Virtualisierung*. Abgerufen am 11. April 2018 von www.itwissen.info: https://www.itwissen.info/Virtualisierung-virtualization-technology-VT.html

Mouat, A. (September 2016). *Grundlagen von Docker*. Abgerufen am 30. April 2018 von www.dpunkt.de: https://www.dpunkt.de/common/leseproben//12553/5_Grundlagen%20von%20Docker.pdf

Nemeth, A. (24. Mai 2016). *Mit Docker vom Container-Konzept in der IT profitieren*. Abgerufen am 04. Mai 2018 von cloud.telekom.de: https://cloud.telekom.de/blog/docker-softwareentwicklung/

Popek, G., & Goldberg, R. (1974). *Formal Requirements for Virtualizable Third Generation Architectures*. Abgerufen am 01. Mai 2018 von citeseerx.ist.psu.edu: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.4815&rep=rep1&type=pdf

Prof.Dr. Leymann, F. (19. Februar 2018). *Hardware-Virtualisierung*. Abgerufen am 11. April 2018 von wirtschaftslexikon.gabler.de: https://wirtschaftslexikon.gabler.de/definition/hardware-virtualisierung-53364

Pustina, L. (30. März 2015). *Docker Basics: Einführung in die System-Level-Virtualisierung.* Abgerufen am 08. Mai 2018 von entwickler.de: https://entwickler.de/online/development/docker-basics-system-level-virtualisierung-125514.html

Literaturverzeichnis

Roden, G. (28. Februar 2014). *Anwendungen mit Docker transportabel machen*. Abgerufen am 28. April 2018 von www.heise.de: https://www.heise.de/developer/artikel/Anwendungen-mit-Docker-transportabel-machen-2127220.html?seite=all

Rossbach, P. (04. April 2017). *Container vs. Virtuelle Maschinen (VM´s): Ein Blick auf die Sicherheit*. Von bee42.com: https://bee42.com/de/blog/Container-vs-Virtuelle-Maschinen-Sicherheit/ abgerufen

Skerlec, M. (12. Juli 2017). *Was haben Frachtcontainer mit IT zu tun?* Abgerufen am 20. April 2018 von blog.bwi.de: https://blog.bwi.de/was-haben-frachtcontainer-mit-it-zu-tun/

Tiersch, F. (07. September 2015). *Docker – Teil 2: Das Arbeiten mit Containern*. Abgerufen am 29. August 2018 von www.ab-heute-programmieren.de: https://www.ab-heute-programmieren.de/docker-teil-2-das-arbeiten-mit-containern/

Tiersch, F. (21. September 2015). *Docker – Teil 4: Das Dockerfile*. Abgerufen am 05. Mai 2018 von www.ab-heute-programmieren.de: https://www.ab-heute-programmieren.de/docker-teil-4-das-dockerfile/

***Virtualisierung (Informatik)*.** (30. Dezember 2017). Abgerufen am 10. April 2018 von https://wikipedia.de: https://de.wikipedia.org/wiki/Virtualisierung_(Informatik)

Vitz, M. (04. April 2017). *Docker Compose - Komplette Systeme mit Docker managen*. Abgerufen am 13. April 2018 von www.innoq.com: https://www.innoq.com/de/articles/2017/04/docker-compose/

Willhöft. (16. Februar 2016). *Dienste mit Docker-Containern betreiben*. Abgerufen am 02. Mai 2018 von www.willhoeft-it.com: http://www.willhoeft-it.com/2016/02/16/docker-101.html