

**Kevin Klein**

**Thema der Bachelorarbeit**

* Die Extraktion und das Effektuieren von Software in die Cloud
* Die Verlagerung von Software
* Die Verlagerung von Programmen bis in die Cloud
* Die Verlagerung von Programmen anhand von Docker

**Stichworte**

**Kurzzusammenfassung**

Die schnell wachsenden technologischen Fortschritteermöglichen ständig neue Wege im Umgang mit Software. Im Rahmen dieser Bachelorarbeit wird die Expansion von Programmen bis in die Cloud untersucht. Danach werden die Interaktionsmöglichkeiten mit den verlagerten Programmen und die momentan dafür essenziellen Softwarelösungen analysiert. Anschließend wird evaluiert, wie effizient das Verlagern von Software in die Cloud ist und welchen effektiven Nutzen diese Veränderung hervorbringt.

Die vorliegende Bachelorarbeit gibt einen Überblick über die funktionalen Möglichkeiten am Beispiel von Docker. Dabei wurden die Funktionsweisen von Containern unter verschiedenen Systemen und Lokalitäten untersucht. Anschließend wurden diese Erkenntnisse mit alternativen Anbietern für Containerisierung verglichen und die individuellen Vorteile dargelegt. Zur praktischen Umsetzung wurde ein Python Skript, das mit einer Webseite kommuniziert in einem Docker Container innerhalb der SAP-Cloud Plattform ausgeführt.

**Kevin Klein**

**Title of the paper**

**Keywords**

**Abstract**

**Inhaltsverzeichnis**

**1 Einleitung**

1.1 Zielsetzung

1.2 Gliederung der Arbeit

**2 Grundlagen**

2.1 Container-Technologie

2.2 Virtualisierungsmodelle

2.3 Linux Kernel

2.4 Cloudcomputing

2.4.1 Provider

2.4.2 Sicherheit

**3** **Systeme und Erweiterungen**

3.1 Docker

3.1.1 Docker unter Linux

3.1.2 Docker unter Windows

3.1.2.1 Docker Images // mehr Befehle der File angeben

3.1.2.2 Docker Hub

3.1.2.3 Docker Compose

3.1.2.4 Volumes

3.1.2.5 Container

3.1.2.6 Debugging Container

3.2 Alpine Linux

3.3 Nginx

3.4 Docker Swarm

3.5 Kubernetes

3.6 Webapplikationen und CMS

3.6.1 Nextcloud

3.6.2 Wordpress

3.7 Continuous delivery

3.8 Git Hub Anbindung

**4 Alternativen zu Docker**

4.1 LXC

4.2 Hyper-V Containers

4.3 rkt

4.4 Podman

4.5 runC

4.6 containerd

**5 Realisierung**

5.1 verwendete Technologien

5.1.1 Python

5.1.2 HTML

5.1.3 CSS

5.2 Technische Umsetzung

5.2.1 Flask

5.2.2 J Query

5.2.3 HighCharts

5.2.4 Python Librarys

5.2.5 Gunicorn

**6 Schluss**

5.1 Zusammenfassung

5.2 Fazit

1 Einleitung

In der heutigen Zeit sind Computer nicht mehr wegzudenken. Das Einsatzfeld ist enorm groß und zugleich auch die Vielfalt der verwendeten Software. Die stetig fortscheitenden Technologien ermöglichen daher neue Optionen im Bereich der Softwareverlagerung. Ziel ist es, die verschiedensten Programme nicht lokal auszuführen, sondern von einem externen Standpunkt, zum Beispiel einer Cloud. Dieses Szenario beinhaltet zwei wesentliche Aspekte: Zum einen baut dieses Konzept auf sogenannten Containern auf. Diese Container sind essenziell wichtig, um die Programme letztendlich verlagern zu können und stellen somit die Basis dar. Zum andern profitiert man von einer weiteren wichtigen Technologie, das sogenannte Cloudcomputing. Dieses System stellt eine sehr vielfältige Auswahl an Möglichkeiten zur Verfügung, welche der Nutzer in der Cloud anwenden kann. Der Wunsch der Entwickler nach mehr Flexibilität im Bereich der Softwareentwicklung besteht schon seit einiger Zeit. Es gab jedoch bis … keine richtige Lösung für das Problem, denn die bekannte Technologie der ‘‘virtuellen Maschine‘‘ war zu schwergängig und langsam, um sie in Clouds oder ähnliche Lokalitäten zu verlagern.

Dadurch wurde die Idee geboren, diesen Prozess wesentlich effizienter zu gestalten, um das Ziel zu erreichen. Anfang … entstand der erste Ansatz, die Virtualisierung zu verlassen und eine neue Technologie, die sogenannte Containerisierung, in die Softwarewelt zu bringen. Die Firma Docker hat sich genau dies zur Aufgabe gemacht und ist bis heute auch der unangefochtene Markführer. Das Zusammenspiel der Expansion von Software und der Cloud eröffnete sehr individuelle neue Möglichkeiten. Es werden aber auch sehr viele Ressourcen benötigt, so auch eine Cloud, die es ermöglicht, Rechenleistung zu beanspruchen, um diese Container ausführen zu können. Heutzutage gibt es einige Cloudprovider, die dieses Zusammenspiel ermöglichen, mitunter auch die SAP Cloud Platform.

1.1 Zielsetzung

Ziel der Arbeit ist die Expansion eines Programms unter der Verwendung von Containern in eine Cloud. Dazu werden die einzelnen Aspekte bezüglich der Containerisierung im Vergleich zur virtuellen Maschine analysiert. Des Weiteren wird evaluiert, wie skalierbar und ressourcenaufwändig die neue Visualisierungstechnologie ist. Nicht zu vernachlässigen ist die Eruierung der Effizienz, Sicherheit und vor allem der Realisierbarkeit des Prozesses.

Die populärsten Container-Technologien werden ebenfalls gegenübergestellt und die vor und Nachteile dargestellt. Um die Umsetzbarkeit und die Anwendungsoptionen zu demonstrieren, wird ein Webserver in der Cloud exekutiert (=hinrichten). Dieser Server besteht aus einer HTML-Seite mit PHP, worüber der interaktive Austausch mit dem Nutzer stattfindet und einem JAVA Programm, womit die Daten über die Rest-API verarbeitet werden. Der enorme Vorteil dieser Vorgehensweise liegt darin, dass mehrere Nutzer von unterschiedlichen Lokalitäten auf den Webserver zugreifen können und somit den lokalen Anwendungsbereich verlassen. Dazu werden die ressourcenschonendsten Konzepte verwendet, um eine weiterhin gesteigerte Effizienz zu erlangen. Die Isolierung von mehreren Programmiersprachen in einem Programm wird damit auch ermöglicht.

1.2 Gliederung

2 Grundlagen

2.1 Container-Technologie

Schon 1979 entstand der erste Ansatz zur Virtualisierung. Einige Entwickler von Unix hatten damals den Befehl chroot eingeführt und ermöglicht einen kritischen Prozess des Dateisystems vom Rest zu isolieren.

(<http://openbook.rheinwerk-verlag.de/linux_unix_programmierung/Kap18A-009.htm>).

Die Firma Virtuozzo entwickelte 2001 als erstes Unternehmen eine verfügbare Container Technologie. Bis heute liefen über 5 Millionen virtuelle Umgebungen in der Cloud von Virtuozzo (<https://www.virtuozzo.com/about/our-story.html>). Mit LXC wurde der nächste Meilenstein zur Containerisierung gelegt. Oracle hatte es geschafft, einige isolierte Linux Instanzen auf dem gleichen Host laufen zu lassen. Dabei wurde auch schon die Ressourcenschonung beachtet. Das System LXC war jedoch noch relativ komplex und fand aus diesem Grund bei den Entwicklern noch nicht sehr viel Anklang. Dies sollte sich 2013 schlagartig ändern, denn die Firma Docker hat es geschafft, sehr einfach Software in Containern auszuführen. Des Weiteren hat die Firma sehr stark mit Cloud, Linux und Windows Anbietern zusammengearbeitet und das Ausführen der Container auf allen Betriebssystemen ermöglicht. Die einfache, ressourcenschonende und äußerst effiziente Technologie führte dazu, dass das Unternehmen Docker bis heute unangefochten der Marktführer in diesem Bereich ist.

(<https://docs.microsoft.com/de-de/dotnet/architecture/microservices/container-docker-introduction/docker-defined>). Es gibt jedoch auch einige gute Alternativen zu Docker. Darunter auch LXC, Podman, rkt und Windows Hyper-V. Diese Technologien haben unter anderem einen wesentlichen Vorteil in der Sicherheit des Systems gegenüber Docker.

2.2 Virtualisierungsmodelle

Virtualisierung ermöglicht es auf das physische Pendant virtuell zuzugreifen und die Funktionen zu nutzen. Es gibt einige Formen der Virtualisierung, zum Beispiel die Hardware Virtualisierung, die Software Virtualisierung, die Speicher Virtualisierung, die Daten Virtualisierung und die Netzwerk Virtualisierung. Die Hardware Virtualisierung ermöglicht es, Hardware-Komponenten unabhängig von der physischen Basis laufen zu lassen. Dazu werden meistens sogenannte VM (Virtuelle Maschinen) benutzt. Diese Technologie wird oft mit der Containerisierungstechnologie verglichen. Das Grundprinzip beider Technologien ist das gleiche, jedoch sind Container wesentlich besser optimiert als VMs. Der größte Nachteil von VMs gegenüber Containern ist, dass diese sehr schwergewichtig sind und deswegen auch wesentlich langsamer als Container sind. Dies liegt daran, dass VMs ein komplettes Betriebssystem integriert haben. Container hingegen lösen dieses Problem sehr effizient, denn es werden nur die Libraries benutzt, welche auch wirklich benötigt werden. In einem Docker-Container wird dann über einen Bootloader mit einem Docker-Service kommuniziert und dieser kommuniziert wiederrum mit dem Host Betriebssystem.

<https://www.ionos.de/digitalguide/server/konfiguration/virtualisierung/>

<https://screengui.de/31/docker>

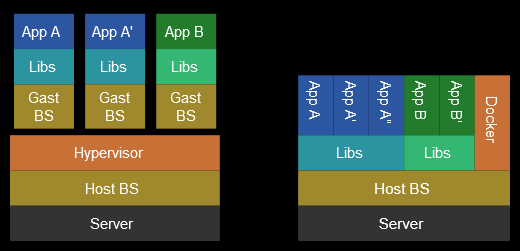


Abb. 1

In der Abb. 1 sieht man links im Bild sehr deutlich den von VMs erzeugten Overhead und rechts die effizientere Lösung mit Containern.

2.3 Linux Kernel

Container benötigen für ihre volle Realisierung einen Kernel. Dies muss nicht unbedingt ein Linux-Kernel sein, jedoch hat ein Linux-Kernel unter Docker die Möglichkeit, auf allen verfügbaren Betriebssystemen Linux-Container zu starten. Microsoft ist es mit der Hyper-V Technology jedoch auch gelungen, Linux Container unter Windows auszuführen, ohne den Linux Kernel zu verwenden. Ein Kernel ist ein essenziell wichtiger Baustein für jedes Betriebssystem. Dabei unterscheiden sich die Kernel der verschiedenen Betriebssysteme deutlich. Er hat die Funktion zwischen Hardware und Software zu agieren und zum Beispiel Treiber bereit zu stellen oder darauf zu achten, dass das System so effizient und fehlerfrei wie nur möglich funktioniert. Der Linux-Kernel verwaltet auch die Ressourcen, den Speicher und viele weitere Komponenten. Nur mit einem Kernel läuft ein Betriebssystem allerdings auch nicht, deswegen werden noch weitere Anwendungen, wie zum Beispiel eine Shell, benötigt, um das Betriebssystem vollwertig nutzen zu können. Eine Linux Distribution stellt eine Kombination aus dem Kernel, einem Boot-Manager, einem GNU-System und aus vielem mehr dar. Das GNU-System beinhaltet einige Softwarepakete und ist im Grundsatz ähnlich zu Unix. Manche Distributionen inkludieren auch Programme, wie zum Beispiel Browser oder Open Office. Es gibt mittlerweile mehrere hundert verschiedene Linux Distributionen, wobei Debian und Red Hat die zwei sind, worauf die meisten anderen Distributionen aufbauen. Die populärsten Distributionen neben Debian und Red Hat sind Ubuntu, Fedora, MX Linux, Linux Mint und Manjaro Linux. Docker kann diese Distributionen in seine Container einbinden und aufgrund der enormen Auswahl an Distributionen ist es möglich, einen sehr leichtgewichtigen Container zu erzeugen. Des Weiteren muss Docker auch mit einem Kernel kommunizieren. Im folgenden Bild wird veranschaulicht, welche Ebenen Docker beansprucht und wie der Ablauf ist.

FALSCHES BILD KEIN KERNEL ZU SEHEN

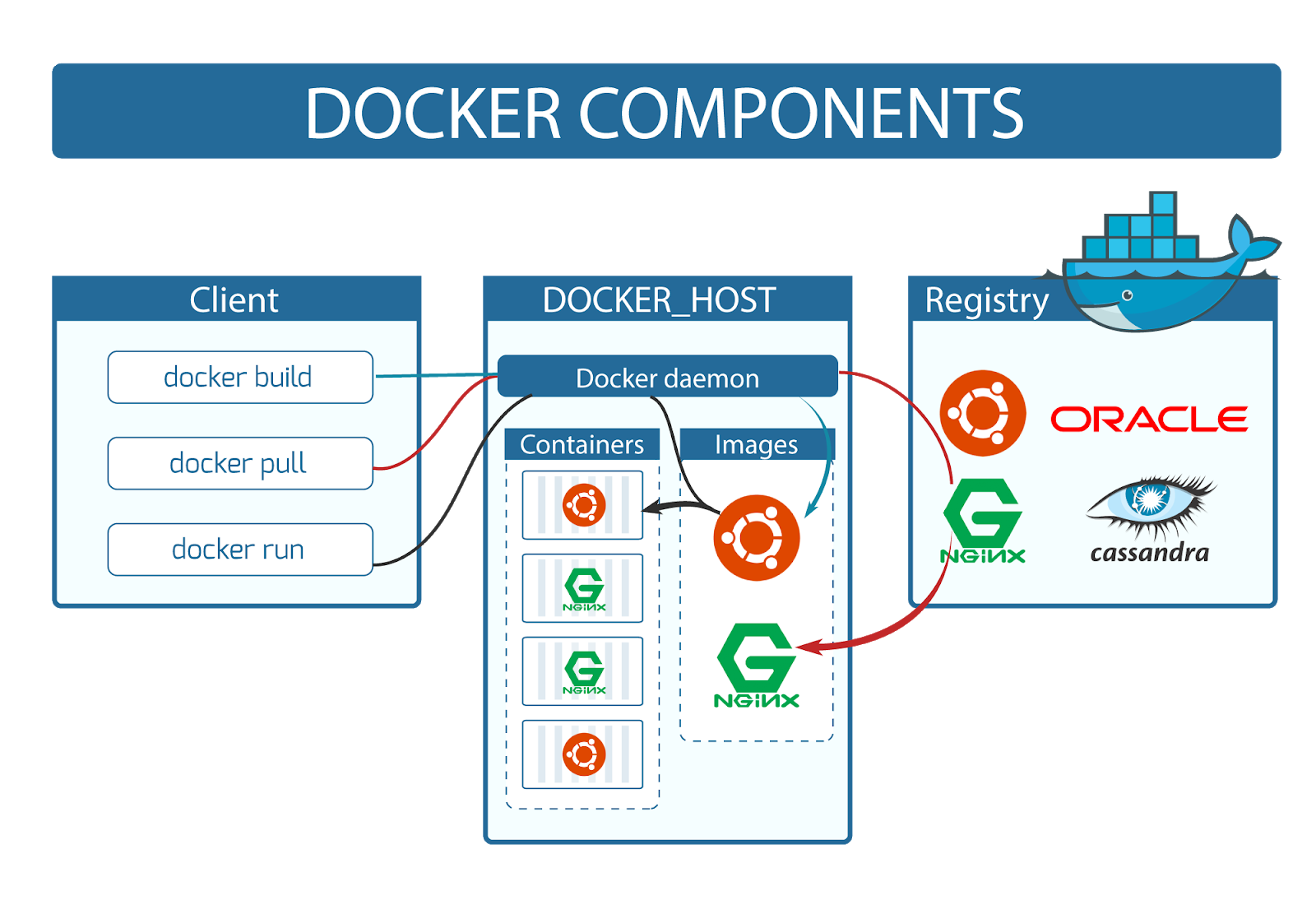


Abb. 2

<https://de.tipsandtrics.com/linux-kernel-an-explanation-layman-s-terms-763141>

<http://www.linux-kurs.eu/distributionen.php>

<https://www.chip.de/news/Bestes-Linux-OS-Top-10-der-beliebtesten-Distributionen_100413102.html>

2.4 Cloudcomputing

Die Idee des Cloudcomputing existiert bereits seit den 1960er Jahren. Mehrere Entwickler hatten die Idee, einige Ressourcen für eine breite Masse zur Verfügung zu stellen. Jedoch waren damals die Gegebenheiten vonseiten des Internets und anderen technischen Ansprüchen nicht gegeben bzw. nicht gut genug ausgereift. 1970 kamen mit der Firma Intel die ersten CPUs auf den Markt und es konnten zu damaliger Zeit schnelle PCs gebaut werden. Einige Jahre darauf kam mit der Firma Apple und Microsoft ein enormer Aufschwung in die Branche und die ersten Betriebssysteme konnten bereitgestellt werden. Tim Berners-Lee entwickelte das uns bekannte Protokoll http (Hypertext Transfer Protocol) und die Auszeichnungssprache HTML (Hypertext Markup Language). Durch diesen enormen technologischen Erfolg waren nun die Gegebenheiten für Cloudcomputing sehr gut. Mit der **Multi-Tenant-Software-Architektur wurde es Ende der 90er Jahre den Unternehmen ermöglicht, über einen Browser von verschiedenen Lokalitäten auf Software zuzugreifen. Dieses System wendete erstmals die** CRM-Software Salesforce **in den USA an und in Deutschland die Firma myfactory. Dies war der Durchbruch für die Cloudsysteme. Die Technology wurde von da an immer ausgereifter und seit 2010 ist jeder It-Service 24/7 nutzbar. Daraus folgt, dass immer mehr Firmen und auch Privatpersonen Clouds nutzen.**

Unbedingt noch anführen prvate, hybride und public cloud …

<https://www.smartbusinesscloud.de/geschichte-des-cloud-computing/>

2.4.1 Provider

Es gibt einige Anbieter für Public Cloud-Services. Zu den größten Anbietern gehören AWS (Amazon Web Service), Microsoft Azure und die Google Cloud Plattform. Sie unterscheiden sich jedoch in einigen elementaren Punkten. Die Speicherbegrenzung von Ressourcen bei AWS liegt zum Beispiel bei 0.5-244 GB. Bei Microsoft Azure sogar bei 0.75-448 GB und bei Google nur bei 0.6-208 GB. Die Kosten für den reinen Speicherplatz liegen bei Amazon bei 0,023 $ pro GB, bei Mircosoft bei 0,0002 $ pro GB und bei Google bei 0,007-0,014 $ pro GB

Nochmal überarbeiten

<https://www.infopulse.com/de/blog/das-beste-wahlen-vergleichende-analyse-zu-aws-vs-azure-vs-google-cloud/>

2.4.2 Sicherheit

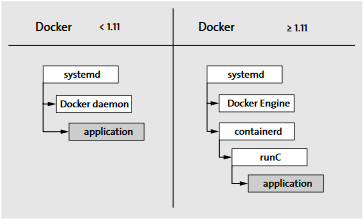
3. Systeme

Das momentan populärste Open-Source-Programm für Containerisierung ist Docker. Es gibt jedoch noch einige Erweiterungen im Zusammenhang mit Docker. Darunter auch Kubernetes. Es wird durch diese neue addierte Expansion ermöglicht, dass Docker noch effizienter in komplexeren Systemen ausgeführt werden kann und eine Skalierung der Container ermöglicht wird. Des Weiteren gibt es noch andere Stellschrauben, an denen gedreht werden kann, um vor allem die Containerisierung so ressourcenschonend wie nur möglich zu gestalten. Dafür gibt es einige sehr kleine Linux-Distributionen, die in der Docker File angegeben werden können. Diese Distributionen enthalten nur die wichtigsten technischen Grundbausteine, um mit der Containertechnologie perfekt zu harmonieren. Somit wird der Overhead stark reduziert, was sich auch in der Leistung bemerkbar machen kann. Eine kleine Linux-Distribution, die sehr oft verwendet wird, ist zum Beispiel Alpine Linux. Da mittlerweile in Docker auch immer mehr Anwendungen in mehrere Microservices aufgeteilt werden können, entsteht dadurch mehr Aufwand bei der Orchestrierung von Containern. Um dieses Problem zu lösen, entstand Docker Swarm.

<https://azure.microsoft.com/de-de/topic/kubernetes-vs-docker/>

3.1 Docker

Der Aufbau von Docker ist relativ einfach gehalten. Trotzdem ist es wichtig, die funktionalen Abhängigkeiten in Docker zu kennen, um mit dem System bestmöglich arbeiten zu können. Das Unternehmen hat es sich zur Aufgabe gemacht, die Prozesse und Befehle so einfach wie nur möglich zu gestalten. Dieses Vorhaben fand starken Anklang bei den Nutzern.



ALLES ROTE UMSCHREIBEN

Systemd verwaltet alle Prozesse auf Linux systemen und ist zum Starten, überwachen und beenden der Prozesse verantwortlich. Containerd is a container runtime which can manage a complete container lifecycle - from image transfer/storage to container execution, supervision and networking.. [runc](http://runc.io) is lightweight universal run time container, which abides by the OCI specification. runc is used by containerd for spawning and running containers according to OCI spec. It is also the repackaging of libcontainer.. [OCI](https://www.opencontainers.org/) maintains the OCI specification for runtime and images. The current docker versions support OCI image and runtime specs.

RunC implementiert das OCi Interface und ist zum starten der Container verantwortlich.

the Docker daemon prepares the image as an Open Container Image (OCI) bundle and makes an API call to containerd to start the OCI bundle. containerd then starts the container using runC. Note, the runtimes have to be OCI compliant, (like runC is), that is, they have to expose a fixed API to managers like containerd so that they(containerd) can make life easy for them(runC) (and ask them to stop/start containers)

OCI wurde 2015 von Docker und anderen großen Firmen im Bereich der Containerisierung verwendet und wurde von da an zu einem Standart in der Branche..

3.1.1 Docker unter Linux

(Ab hier Inhaltsverzeichnis vergleichen)

3.1.2 Docker unter Windows

3.1.2.1 Docker Images

Die gesamte Funktionsweise in Docker baut auf (read only) Docker Images auf. Sie sind die Basis, um später in Container eingebunden zu werden und damit arbeiten zu können. Docker Images beinhalten immer sehr individuelle Dateien, die man in einem Container ausführen möchte bzw. darauf zugreifen möchte. Zu einem Docker Image gehört auch immer eine sogenannte Docker File. Diese beinhaltet auch individuelle Konfigurationen, wie zum Beispiel dass Docker Image ausgeführt werden oder welche Linux Distribution verwendet werden soll. Des Weiteren kann man in der Docker File mit dem Befehl “COPY“ angeben, welche lokale Datei in die virtuelle Docker Directory kopiert werden soll, um eventuell später darauf zugreifen zu können. In der Docker File wird auch die benötigte Laufzeitumgebung angegeben, um zum Beispiel Python dort ausführen zu können. Um von einer Docker File, in der alle individuellen Dateien angegeben sind, ein Image zu erzeugen, gibt es mehrere Befehle von Docker. Dafür navigiert man mit einer Kommandozeile in den Zielordner und gegebenenfalls den zusätzlichen Dateien. Dort kann man dann mit dem Befehl “docker build .“ die Erstellung des Docker Images einleiten. Mit dem Befehl “docker build -t repositoryname/tagname .“ kann man direkt dem Image einen Repository-Namen und einen Tag-Namen zuweisen. Wichtig dabei ist es, den Punkt am Ende nicht zu vergessen. Mit “docker Images“ können alle erstellten Images angezeigt werden. Dort sieht man auch, dass jedes Image eine eindeutige Nummer beinhaltet, über die man auch im Nachhinein noch den Tag und andere Änderungen vornehmen kann. Durch den Befehl “docker run -d -p 8080:80 imagename“ kann ein Container mit dem Image erstellt und gestartet werden. Dabei legt -d fest, dass der Container im Hintergrund läuft und die Konsole nicht blockiert. Das -p in dem Befehlt legt dabei den Host Port 8080 im Container auf Port 80.

<http://www.anecon.com/blog/docker-basics-befehle-und-life-hacks/>

3.1.2.2 Docker Hub

Docker Hub ist eine von Docker bereitgestellte Library in der Cloud. Dort werden Images verwaltet und diese kann man auch in sein eigenes Projekt implementieren. Zudem ist es möglich, eigene Images in das (den???) Docker-Hub zu pushen und dort kann man einstellen, ob dieses Image öffentlich oder nur privat gemacht werden soll. Der Vorteil dieses Prinzips ist es, dass die Nutzer sehr viel Auswahl an vorgefertigten Images haben und dies sehr viel Zeit sparen kann, falls man das gesuchte Image auch findet. Es kann aber auch einen Nachteil darstellen, da einige unbrauchbare Images hochgeladen werden. Docker hat dafür auch eine Lösung gefunden und stellt eigene von Docker offizielle Images in die Cloud, die auch überprüft wurden. Des Weiteren besteht die Möglichkeit, Images von GitHub zu erstellen und diese dann auf Docker-Hub zu pushen. Nachdem man sich auf Docker-Hub angemeldet hat und ein neues Repository erstellt hat, wird schon der Befehl zum pushen angezeigt: “docker push hubname/reponame:tagname“. Um diesen Befehl in der Kommandozeile ausführen zu können, muss man sich zuerst mit seinem Account verbinden. Dies funktioniert mit dem Befehl (docker login) und danach kann das Image dann in das Docker-Hub gepusht werden.

<https://docs.docker.com/docker-hub/>

3.1.2.3 Docker Compose

Docker Compose ermöglicht es, mehrere Container auszuführen. Dafür werden alle benötigten Ressourcen in der Dockerfile angegeben. Dazu gehört auch gegebenenfalls der Port oder zu installierende Abhängigkeiten. Im gleichen lokalen Verzeichnis kann man eine neue File anlegen “docker-compose.yml“. In dieser Docker Compose File wird angegeben, welche Services benutzt und welche Images verwendet werden sollen. Es besteht die Möglichkeit, auch Images von Docker-Hub zu implementieren. Damit der Dockerfile ein eigenes Image erstellt wird, kann in der Docker Compose File unter einem beliebigen Service einfach “build .“ angegeben werden, da die Dockerfile und die Compose File im gleichen Verzeichnis liegen. Des Weiteren kann der Host Port festgelegt werden und der Exposed (expose?) Port. Der exposed Port wurde in der Dockerfile angegeben und besagt, dass der Container auf Port xy hört. Mit dem Befehl “docker-compose up“ wird Docker Compose gestartet und damit auch alle Container. Unter localhost:xy würde das Projekt laufen. Um Docker Compose wieder zu stoppen, gibt es zwei Möglichkeiten. Die erste ist, in der gleichen Kommandozeile STRG und c zu drücken. Die zweite ist, in einem separaten Terminal den Befehl “docker-compose down“ zu verwenden. Dabei ist es essenziell, dass man sich in dem Projektverzeichnis befindet. Es ist auch möglich, in der Docker Compose File sogenannte Volumes und Umgebungsvariablen zu verwenden. Die Volumes ermöglichen zusammen mit der Umgebungsvariable den Code zu verändern, ohne dass dieses Image neu erstellt werden muss.

<https://docs.docker.com/compose/gettingstarted/>

3.1.2.4 Volumes

Sofern bei Docker ein Update der Images vollzogen werden musste, wurde ein separater Container erstellt. In diesen Container werden alle nötigen geupdateten Elemente geladen. Dieses Konzept funktioniert auch sehr gut. Jedoch können in einer Anwendung verschiedenste Dateien erstellt worden sein, welche nicht automatisch in den neuen geupdateten Container übernommen wurden, sondern nur der geupdatete Programmcode würde übernommen werden. Das Problem liegt darin, dass jeder Container sein eigenes Verzeichnis hat. Um dieses Problem zu lösen, hat Docker das Konzept der Volumes eingeführt. Dieses besagt, dass durch Volumes ein separates Verzeichnis unabhängig von Containern erstellt wird und in diesem Verzeichnis alle erstellten Dateien gespeichert werden. Somit kann mit dem geupdateten Container direkt auf die Dateien zugegriffen werden. Das Standardverzeichnis unter Linux lautet dabei “/var/lib/docker/volumes“ und kann durch eine UUID identifiziert werden. Des Weiteren besteht die Möglichkeit, einen eigenen Namen für die Volumes sowie die Lokalität des Verzeichsses festzulegen. Die Lokalität des Verzeichnisses kann mit dem Befehl “docker inspect“ ermittelt werden.

(vgl. Buch Docker)

Für die Angaben brauchst du in der Regel den Vornamen und Nachnamen des Autors oder Autorin, den Titel des Werkes, den Erscheinungsort, den Verlag, das Erscheinungsjahr und die Seite. Die **Quellenangabe** für ein Buch nach der deutschen Zitierweise sieht so aus: ¹Vorname, Name, Titel des Werkes. Ort: Verlag Jahr, Seite.

3.1.2.5 Container

Docker Container können aus Docker Images erstellt werden. Um die Containerisierung auch praktisch anwenden zu können, gibt es einige wichtige Befehle. Mit dem Befehl docker create und dem zugehörigen Imagenamen kann ein Container erstellt werden. Durch den Befehl docker start und der Container-ID kann dieser Container gestartet werden. Eine konventionellere Art und Weise einen Container direkt zu erstellen und zu starten, ist mit dem Befehl docker run und der Image id möglich. Durch docker stop inklusive der Container-ID kann der Container auch wieder gestoppt werden. Der Befehl docker ps zeigt alle aktuell laufenden Container an. Mit docker rm und der Container-ID kann der gestoppte Container komplett gelöscht werden. Durch docker inspect und der Container-ID erlangt man Informationen über diesen Container, wie zum Beispiel das Erstellungsdatum oder Informationen über den Status.

<https://www.hosteurope.de/blog/diese-wichtigen-docker-befehle-sollten-sie-kennen/>

Container besitzen drei Speicherarten: Sicherer Speicher, Schichtspeicher und permanenter Speicher. Der “sichere“ Speicher gilt als relativ sicher, da er nur für den Container bereitgestellt wird und von dem Host-System abgekapselt ist. Sofern der Container gelöscht wird, werden auch alle gespeicherten Inhalte von diesem Container eliminiert. Dieses Konzept verhindert Angriffe vom Host zum Container-Speicher. Der permanente Speicher wird durch die Volumes eingerichtet und legt einen Speicherplatz für alle benötigten Container fest. Diese Volumes greifen jedoch auf das lokale Verzeichnis zu. Dazu gibt es auch einige Befehle, die nur das Lesen oder nur das Schreiben erlauben.

 docker run -v c:\ContainerData:c:\data:RO für den schreibgeschützten Zugriff

 docker run -v c:\ContainerData:c:\data:RW für den Schreibzugriff

 docker run -v c:\ContainerData:c:\data für den Schreibzugriff (Standard)

Schichtspeicher noch text

Die Speicherbegrenzung von Containern beläuft sich standartmäßig auf eine Größe von 20 GB. Diese Begrenzung kann man beliebig ändern. Dazu stellt Docker den Befehl “docker run --storage-opt "size=50GB" mcr.microsoft.com/windows/servercore:ltsc2019 cmd“ bereit. In diesem Fall wird die Speicherbegrenzung des Containers auf 50 GB erhöht.

<https://docs.microsoft.com/de-de/virtualization/windowscontainers/manage-containers/persistent-storage>

<https://docs.microsoft.com/de-de/virtualization/windowscontainers/manage-containers/container-storage>

3.1.2.6 Debugging Container

3.2 Alpine Linux

Alpine Linux hat seinen Ursprung in dem so genannten LEAF-Projekt (Linux Embedded Appliance Framework Project). Dieses Projekt baut wiederum auf dem „Router on a Floppy“ (Linux Router Project) auf, das Ende der 90er Jahre entstanden ist. Alpine Linux ist eine extrem kleine Linux Distribution, die auf Performance ausgelegt ist. Die Größe beträgt in einem Docker-Container ca. 4 MByte und wurde laut eigenen Angaben für eine effiziente, sichere und einfache Anwendung entwickelt. In der folgenden Tabelle kann der Speicherplatzvorteil von Alpine Linux gegenüber anderen Distributionen entnommen werden.

|  |  |
| --- | --- |
| Alpine Linux | Ca. 4 MByte |
| Cent OS 7 | Ca. 210 MByte |
| Debian 9 | Ca. 100 MByte |
| Ubuntu 16.04 | Ca. 110 Mbyte |
| Ubuntu 18.04 | Ca. 80 MByte |

Die leichte Implementierbarkeit über zum Beispiel Docker-Hub in eine Dockerfile ist sehr einfach gehalten. Durch die extrem effiziente Arbeitsweise von Alpine Linux, ist die Distribution bei den Nutzern von Docker sehr gefragt. Sie hat bislang alle anderen Distributionen auf Docker-Hub in den Schatten gestellt. Die Downloadzahlen belegen dies eindeutig.

Zahlen sind alt aus dokument

Alpine 1 986 214 848

Ubuntu 915 265 673

Debian 177 416 107

Alpine Linux verwendet die C-Standartbibliothek musl, die kompakter als glibc ist. Dabei werden Schlüsselwörter wie z. B. “domain“ und “search“ nicht verwendet. Des Weiteren wird das Init-System OpenRc anstatt systemd verwendet, um die Effizienz zu verbessern. Dieses System startet Hintergrund und Netzwerkdienste. Bei den meisten Dockerimages wird dies jedoch nicht benötigt.

BusyBox reduziert in Alpine Linux die Linux Kommandos, um die Distribution so klein wie möglich zu halten und bietet deshalb nicht ganz so viele Möglichkeiten wie die Vollversion. Die Shell bin/sh ist eine ebenfalls sehr kompakte Variante, die Alpine Linux verwendet. Bei größeren Linux Distributionen wird meistens die Bash verwendet. Dieses bietet mehr Optionen, ist dafür aber auch schwergewichtiger. (Bernd Öggl, 2018, S. 121)[Ber18]

<https://alpinelinux.org/about/>

<https://www.heise.de/select/ix/2019/3/1551612550867020>

3.3 Nginx

**Überprüfen auf Zitat oder nicht.**

Nginx ist ein Webserver, der 10 Jahre nach Apache entwickelt wurde. Das Projekt entstand durch ein sogenanntes [C10k-Problem](https://en.wikipedia.org/wiki/C10k_problem) das besagt, dass sobald 10000 Personen gleichzeitig auf den Server zugegriffen haben, er stark verlangsamt oder sogar abgestürzt ist. Dieses Problem soll Nginx mit einem neuen technologischen Ansatz verhindern. Durch die stetig wachsenden Zugriffe auf Webseiten wird diese Technology immer wichtiger. Vor allem sehr bekannte Webseiten rüsten immer mehr auf Nginx um. Insgesamt laufen momentan 46% der bekanntesten Webseiten auf Nginx. Dazu zählen zum Beispiel Netflix, Twitter und Apple. Apache wird trotzdem noch am meisten genutzt, vor allem bei weniger populären Webseiten.

Zu den Funktionen von Nginx zählen:

ERKLÄREN

1. Reverse Proxying stellt die Schnittstelle zum Backendserver dar.
2. Load Balancing ist die Verteilung der Lasten.
3. Video Streaming
4. **FastCGI-Unterstützung mit Caching**
5. **TLS/SSL**

Der größte Unterschied gegenüber anderen Webservern ist, dass Nginx die Anfragen über einen einzigen Thread den sogenannten Master-Thread annimmt. Von diesem werden dann einige Worker-Threads gleichzeitig und asynchron bearbeitet. Dies verhindert das blockieren zwschen den einzelen Threads. Bei Apache funktioniert die Annahme der Anfragen nicht besonders effizient, denn es wird für jede Anfrage ein neuer Prozess gestartet, was dazu führen kann, dass der Server überlastet wird. Desweiteren lädt Apache selber statische und dynamische Inhalte, diese erhöhen zudem die Serverauslastung. Nginx verfolgt diesbezüglich einen anderen Ansatz, nämlich werden die Anfragen an einen externen Prozess zum Beispiel **FastCGI weitergeleitet dort bearbeitet und wieder zu Nginx zurückgeschickt. Dieses Vorgehen sorgt ebenfalls für eine bessere und ressourcenschonender Arbeitsweise von Nginx. Desweiteren besteht die Möglichkeit des Reverse Proxying. Dies bedeutet, dass Nginx die daten vom Client empfängt, diese zum beispiel bestmöglichst verteilt und zum Backend Server schickt. Dabei kann Nginx nicht nur die Daten bestmöglich verteilen sondern auch die Sicherheit verbessern, indem die Verbindung zum Backend Server über SSL abgewickelt wird. Zudem minimiert Nginx Daten und speichert sie in einem Cache, dies dient ebenfalls der Performace verbesserung**

<https://blog.hubspot.de/marketing/wie-funktioniert-nginx>

**3.4 Docker Swarm**

Die Entstehung von Docker Swarm entstand im Jahr 2014. Zu dieser Zeit lief Docker(native) Swarm extern als selbstständiges Tool. Zur Nutzung musste man dieses manuell integrieren.

Die Technologie war zu diesem Zeitpunkt noch nicht optimiert und sehr komplex in der Verwendung von TLS (Transport Layer Security) und sog. Sidecar-Applications (stellt einer Hauptanwendung nützliche Funktionen bereit). Erst ab der Docker-Version 1.12 wurde das Konzept von Docker Swarm mode implementiert. Docker Swarm beinhaltet einige Funktionen, damit die Kommunikation mit anderen Softwarelösungen reibungslos funktioniert. Ein markanter Verwendungszweck von Docker Swarm ist die Lastenverteilung durch integrierte **Load-Balancing-Funktionen.**

(526 oliver)

https://www.ionos.de/digitalguide/server/knowhow/docker-orchestration-mit-swarm-und-compose/

3.4.1 Docker Swarm Mode

Durch das Tool Swarmkit entstand ab der Docker-Version 1.12 Docker Swarm. Damit die Technologie der Container-Orchestrierung umgesetzt werden konnten wurden einige Funktionen von Swarmkit in Docker Swarm übernommen. (Liebel, 2017, S. 526)[CSI17] Dieses Konzept löste im Herbst 2016 den älteren native Swarm ab.

Ein Nachteil war zu diesem Zeitpunkt, dass die Zuteilung von Swarm Services zu Namespaces noch nicht realisiert wurde. Mit der Docker Version 1.12 wurde die Komplexität enorm reduziert. Es wurde sich stark an dem KISS-Prinzip (Keep It Simple and Stupid) orientiert. Ein Docker-Cluster (Swarm) umfasst mindestens einen Manager- und beliebig viele Worker-Knoten. Durch den Swarm-Manager werden die jeweiligen Aufgaben an die Swarm-Worker vergeben. Die Swarm-Worker beschäftigen sich mit der Durchführung der jeweiligen Aufgaben. Ein „Service“ beinhaltet eine Container-Anwendung, die auf beliebig viele Worker-Knoten verteilt werden kann.

Dabei werden Container-Anwendungen als sogenannte „Services“ auf beliebig viele Docker-Konten verteilt.

In der Docker-Terminologie bezeichnet der Begriff „Service“ eine abstrakte Struktur, mit der Sie Aufgaben definieren, die im Cluster ausgeführt werden sollen. Jeder Service besteht aus einem Set einzelner Tasks, die jeweils in einem eigenen Container auf einem der Knoten im Cluster bearbeitet werden. Wenn Sie einen Service erstellen, bestimmen Sie, welches Container-Image diesem zugrunde liegt und welche Befehle in dem Container laufen, der auf Basis des Images ausgeführt wird. Docker Swarm unterstützt zwei Modi, in denen Swarm-Services definiert werden: Wählen Sie zwischen replizierten und globalen Services.

* **Replizierte Services:** Bei einem replizierten Service handelt es sich um einen Task, der in einer benutzerdefinierten Anzahl von Replikaten ausgeführt wird. Jedes Replikat ist eine Instanz des im Service definierten Docker-Containers. Replizierte Services lassen sich skalieren, indem Nutzer weitere Replikate erzeugen. Ein Webserver wie NGINX lässt sich beispielsweise je nach Bedarf mit einer einzigen Befehlszeile auf 2, 4 oder 100 Instanzen skalieren.
* **Globale Services:** Wird ein Service im globalen Modus ausgeführt, startet jeder verfügbare Knoten im Cluster einen Task für den entsprechenden Service. Wird dem Cluster ein neuer Knoten hinzugefügt, teilt der Swarm-Manager diesem unverzüglich einen Task für den globalen Service zu. Globale Services eignen sich beispielsweise für Monitoring-Anwendungen oder Antivirenprogramme.

https://www.ionos.de/digitalguide/server/knowhow/docker-orchestration-mit-swarm-und-compose/

Docker Swarm hat des Weiteren KV Stores integriert und ermöglicht über TLS (Transport Layer Security) eine schnellere Installation. TLS dient dazu Daten verschlüsselt zu übertragen und ist der Nachfolger von SSL (Secure Socket Layer). Dazu wird eine neue Zertifikatsautorität (CA) beim Start des erstem Managerknotens angelegt. Sobald ein neuer Knoten dem Swarm beitritt, wird automatisiert ein neues Zertifikat mit zufälliger id diesem zugewiesen. Diese Zertifikate werden von jedem Knoten regelmäßig aktualisiert, um gegebenenfalls abgelaufene oder gefährdete Zertifikate auszusortieren. Die Aktualisierungsintervalle können von dem Entwickler selbst festgelegt werden. (529)

S.526,527,528

(KVSTORES ERKLÄREN)

3.4.2 Swarm-Kommandos

Zu den relevantesten Kommandos für Docker Swarm zählen:

* 1. docker swarm - Erzeugt den Swarm und das Management
* 2. docker node - Verwaltung der Manager und Worker Nodes
* 3. docker service - Service Management

3.4.3 Swarmkit

**Literaturverzeichnis**

[Ber18] Bernd Öggl, M. K. (2018). *Docker.* Bonn: Rheinwerk Computig.

Liebel, D. I. (2017). *Skalierbare Container-Infrastrukturen.* Bonn: Rheinwerk Verlag.