[Datum]

[Name des Autors]

[Firmenname]

[Firmenadresse]

[Dokumenttitel]

[Untertitel des Dokuments]

**Kevin Klein**

**Thema der Bachelorarbeit**

Die Isolierung von Anwendungen bis in die Cloud

**Stichworte**

**Kurzzusammenfassung**

Die rapide 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 essentiellen 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 Script, 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**

3.1 Docker

Docker unter Linux

Docker unter Windows

Hyper v

3.1.1 Docker Images // mehr befehle der file angeben

3.1.2 Docker Hub

3.1.3 Docker Compose

3.1.4 Volumes

3.1.5 Container

3.1.6 Debugging Container

Openshift

Security

t Skopeo und Atomic.

Container cluster

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

3.8 Continuous delivery

3.9 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**

5.1Verwendete Technologien

5.1.1 Python

5.1.2 Html

5.1.3 CSS

5.1 Technische Umsetzung

5.2 Ajax

5.3 J Query

5.4 Highcharts

5.5 verwendete Python Libarys

**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 somit 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 2 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. Eine weitere wichtige Technologie ist 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 Isolierung von Software und der Cloud eröffnete sehr individuelle neue Möglichkeiten. Es werden aber auch sehr viele Ressourcen benötig, 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 Plattform.

**1.1 Zielsetzung**

Überarbeten

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 Web Server in der Cloud exekutiert. Dieser Server besteht aus einer HTML Seite , 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 und die Verlagerung von mehreren Programmiersprachen in einem Programm wird somit auch ermöglicht.

**1.2 Gliederung**

**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 somit 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 somit das ausführen der Container auf allen Betriebssystemen ermöglicht. Die einfache, ressourcenschonende und 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 Virtualissierung zum Beispiel die Hardwarevirtualisierung Softwarevirtualisierung, Speichervirtualisierung, Datenvirtualisierung und Netzwerkvirtualisierung. Die Hardwarevirtualisierung ermöglicht es Hardware-Komponenten unabhängig von der Physischen Basis laufen zu lassen. Dazu werden meistens sogenannte VM(Virtuelle Maschinen) benutzt. Diese Technologie wir 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 effizent, denn es werden nur die Libraries benutzt, welche auch wirklich benötigt werden. In einem Dockercontainer wir dann über einen Bootloader mit einem Docker Service kommuniziert und dieser wiederrum mit dem Host Betriebssystem.

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

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

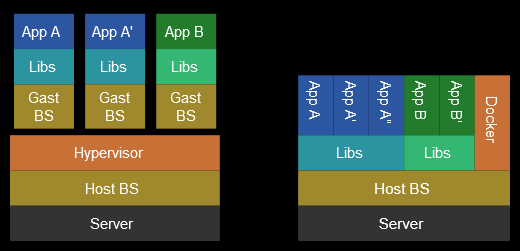


Abb.1

In der Abb1 sieht man sehr deutlich den von VMs erzeugten Overhead links im Bild 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 hat es mit der Hyper-V Technology jedoch auch geschafft 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 Anwendung 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. Mache Distributionen inkludieren auch Programme wie zum Beispiel Open Office oder Browser. 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.

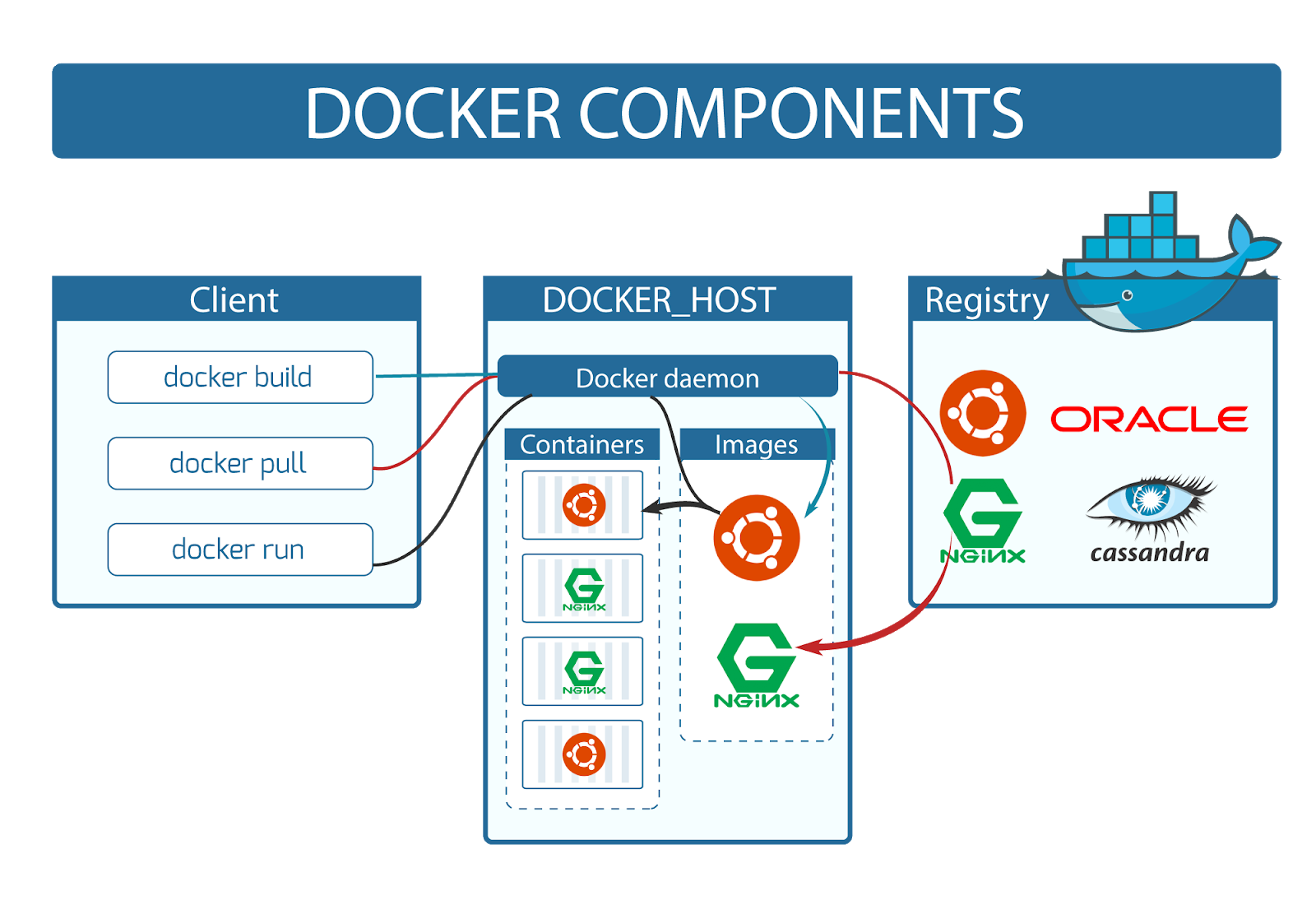


Abb2.

<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. Einige Entwickler hatten die Idee einige Ressourcen für eine Breite Masse zur Verfügung zu stellen, jedoch wahren damals die Gegebenheiten von Seiten des Internet und aneren 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 bakannte 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 Unternehmen ermöglicht über den Browser von verschiedenen Lokalitäten auf Software zuzugreifen. Dieses System wendete erstmals** CRM-Software Salesforce **in der 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 gehören AWS (Amazon Web Service), Microsoft Azure und 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/>

**2.4.1 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.

**3.1.1 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 ganz 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 das Docker Image ausgeführt werden soll 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 nun 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 welche 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 Docker Hub**

Docker Hub ist eine von Docker bereitgestellte Libary 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 Docker Hub zu pushen und dort kann man einstellen, ob dieses Image öffentlich gemacht werden soll oder nur privat. Der Vorteil dieses Prinzips ist es, dass die Nutzer somit sehr viel Auswahl an vorgefertigten Images haben und dies somit 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 auch 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 gepush werde.

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

**3.1.3 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 werden sollen und natürlich auch welche Images verwendet werden sollen. Es besteht die Möglickeit auch Images von Docker Hub zu implementieren. Da mit 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. Desweiteren kann der Host Port festgelegt werden und der Exposed 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 somit auch alle Container und unter localhost:xy würde das Projekt laufen. Um Docker Compose wieder zu stoppen gibt es 2 Möglickeiten. Die erste ist in der gleichen Kommandozeile STRG und c zu drücken. Die Zweite ist es in einem separaten Terminal den Befehl “docker-compose down“ zu verwenden. Dabei ist es essentiell, 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 und vorallem.

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

**3.1.4 Volumes**

Sofern Bei Docker ein Update der Images vollzogen werden musste, dann wurde ein seperater Container erstellt. In diesen Container werden alle nötigen geupdateteten 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 werden 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 er Volumes eingeführt. Dieses besagt, dass durch volumes ein seperates 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 standart Verzechnis unter Linux lautet dabei “/var/lib/docker/volumes“ und kann durch eine UUID identifiziert werden. Desweiteren besteht die Möglichkeit einen eigenen Namen für die Volumes festzulegen, sowie die Lokalität des Verzeichsses. Die Lokalität des Verzeichnisses kann mit dem Befehl “docker inspect“ ermittelt werden.

Quelle Buch Docker

**3.1.5 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 Containerid 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. 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 besitzer 3 Speicherarten. Sicherer Speicher, Schicht speicher 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 von dem Host zu dem Container Speicher. Der Permanente Speicher wird durch die Volumes eingerichtet und legt einen Speciherplatz 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 20GB. 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>

**3.2 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 Docker File ist sehr einfach gehalten. Durch die extrem effizeiente 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-Standartbibliotek musl, die kompakter ist als glibc ist. Dabei werden Schlüsselwörter nicht verwendet. So zum Beispiel “domain“ und “search“. Desweiteren 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 Varianten, die Alpine Linux verwendet. Bei größeren Linux Distributionen wird meistens die Bash verwendet. Dieses bietet mehr Optionen, ist dafür aber auch schwergewichtiger.

Buch seite 121

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

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

**3.3 Nginx**

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) Problem welches 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. Nginx verfolgt den Ansatz, dass alle Anfragen an den Server von einem einzgen Thread abgearbeitet werden muss und die weitere Arbeit auf mehrere Worker verteilt wird.