Hochschule
Albstadt-Sigmaringen

Albstadt-Sigmaringen University

Praktische Arbeit zur vorbereitenden Blockveranstaltung

Software-Container und Software-Development

Bedeutung und Risiken von Software-Containern, sowie deren Einsatz in der Softwareentwicklung und das Zusammenspiel in Clustern

Autoren:

Maximilian Rieger
Technische Informatik
85581

Florian Lubitz
Technische Informatik
85900

Thomas Schöller
Technische Informatik
87113

Marc Bitzer
Technische Informatik
87117

Jonas Acker
Technische Informatik
85583

Inhaltsverzeichnis

1 Einleitung

Bis kurz vor der Jahrtausendwende führte die Virtualisierung von Servern ein Schattendasein und jeder Service wurde auf einem dedizierten Server zur Verfügung gestellt. Dabei war es keine Seltenheit, dass Server sehr gering ausgelastet waren, da der laufende Service nicht die gesamte Leistung der Hardware benötigte und der Ausfall eines nicht redundanten Servers einen Totalausfall eines Services bedeutete. Eine beispielhafte dedizierte Serverkonstellation stellt Abbildung 1 dar.

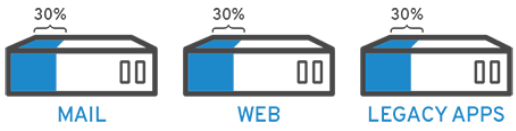

Abbildung 1: Serverauslastung ohne Virtualisierung [1]

Um diese und weitere Probleme zu lösen, gewann die Virtualisierung von Servern zum Anfang des neuen Jahrtausends immer mehr an Bedeutung und ist heutzutage ein fester Bestandteil der IT-Infrastruktur vieler Unternehmen. Dabei werden auf einem physikalischen System mehrere Dienste zusammengefasst, die sonst nur einen Bruchteil der Leistung benötigen würden. Dadurch kommen noch andere Vorteile wie z.B. das Erstellen von Snapshots und das dynamische Verschieben der virtuellen Maschinen zum Tragen. Abbildung 2 zeigt die Auslastung der virtualisierten Server.

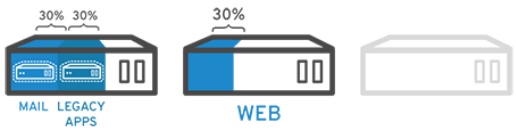

Abbildung 2: Serverauslastung mit Virtualisierung [2]

[1] Quelle: https://www.redhat.com/cms/managed-files/server-usage-500x131.png
[2] Quelle: https://www.redhat.com/cms/managed-files/server-usage-for-virtualization-500x131.png

Doch auch die Virtualisierung von Servern birgt noch Probleme, die einer Lösung bedürfen. So entsteht durch das Betriebssystem der virtuellen Maschinen ein deutlicher Overhead, da diese zur Laufzeit etliche Services benötigen. Außerdem beanspruchen die virtualisierten Betriebssysteme deutlich mehr Hardwareressourcen und die Startzeit ist relativ lang. Somit war die IT-Branche nicht in der Lage, wozu die Transportbranche längst in der Lage war – Güter in Container zu verpacken und diese Container aufgrund des standardisierten Formats auf den verschiedensten Verkehrswegen zu transportieren. Die Technologie- und Methodenplattform für die vernetzte medizinische Forschung e.V. veröffentlichte in einem Bericht ([TUM]) noch weitere Parallelen zwischen Software- und Transportcontainer:

Transport-Container	**Software-Container**
„Bessere Raumausnutzung“	Verzicht auf ein Betriebssystem im Container, Ressourcenallokation über cgroups, Overlay-FS
„Schutz gegen Beschädigungen und Diebstahl“	Virtuelle Netzwerke, wenige offene Ports, derzeit keine Signierung
„Effiziente Beladung, Transport, Entladung“	Sowohl einfache Shellkommandos wie auch komplexe Deploymenttools, zentrales Verzeichnis
„Beschleunigte Abfertigung“	Nachvollziehbarkeit von Änderungen durch Modifikationsskripte (Dockerfiles)
„Geschlossene Transportkette“	Gewisse Betriebssystemunabhängigkeit, Unterstützung durch Cloud-Provider
„Höhere Transportsicherheit“	Isolierung der Prozesse durch Linux Namespaces

Tabelle 1: Transport- und Software-Container

Um diese Lösung in die IT zu portieren, wurden auch für diese Problemstellung Container (in dem Fall für Software) entwickelt. Software-Container setzen wie die Schiffscontainer an den Punkten Kapselung und Portabilität an. Es soll nicht für jeden Service ein zusätzliches Betriebssystem virtualisiert werden, sondern ein Container soll nur das beinhalten, was er für die Bereitstellung des Services benötigt und trotzdem isoliert von den anderen Containern auf der Hardware laufen. Außerdem soll es wie bei den virtuellen Maschinen möglich sein, dynamisch Ressourcen zuzuweisen. [EDWARDS 2016; REDHAT] Die Abbildung 3 verdeutlicht nochmals den eingesparten Overhead bei Containern verglichen mit virtuellen Maschinen.

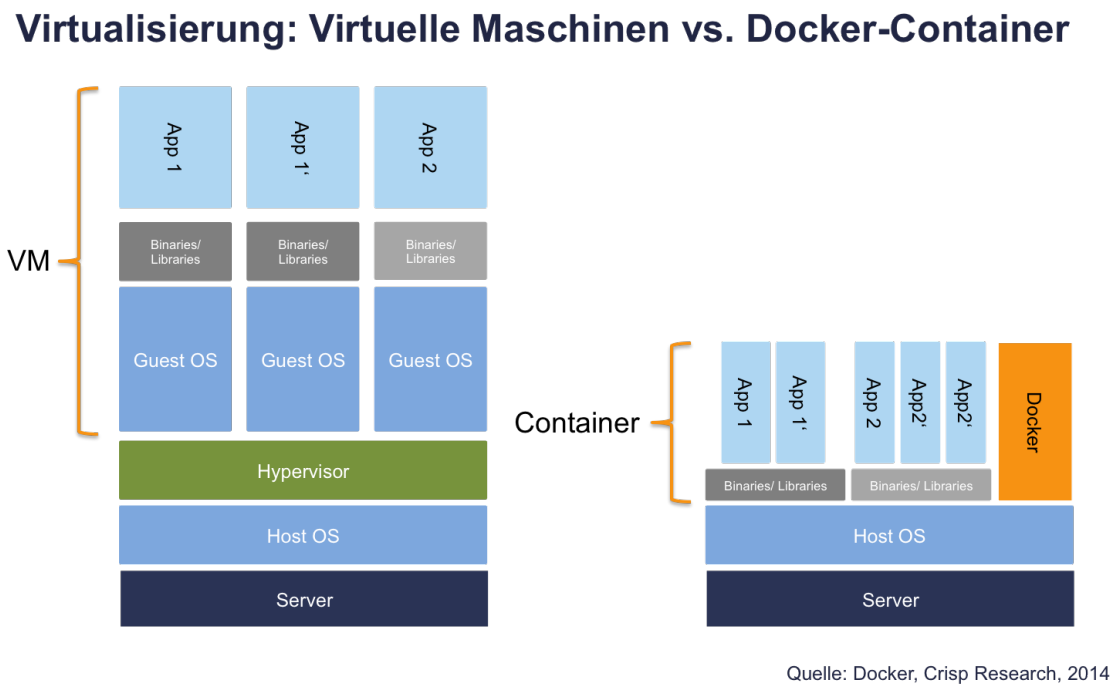

Abbildung 3: Vergleich Container und VM [3]

[3] Quelle: https://images.computerwoche.de/bdb/2668601/738x415_f5f5f5.jpg

2 Funktionalität von Containern

Container setzen meist direkt auf dem Kernel eines Linux-Betriebssystems auf. Um auf den Kernel durchgreifen zu können, verwenden Container native Kernelfuktionen wie Cgroups und Namespaces oder selbst entwickelte Schnittstellen. Dadurch wird das Betriebssystem innerhalb des Containers, ohne einen Hypervisor und eine Kopie des Betriebssystems, zwischen der Anwendung und der Hardware abstrahiert. Alles was die Anwendung zusätzlich benötigt wird mit in den Container gepackt. [ANDERSON 2015] Abbildung 4 verdeutlicht den Kernelzugriff.

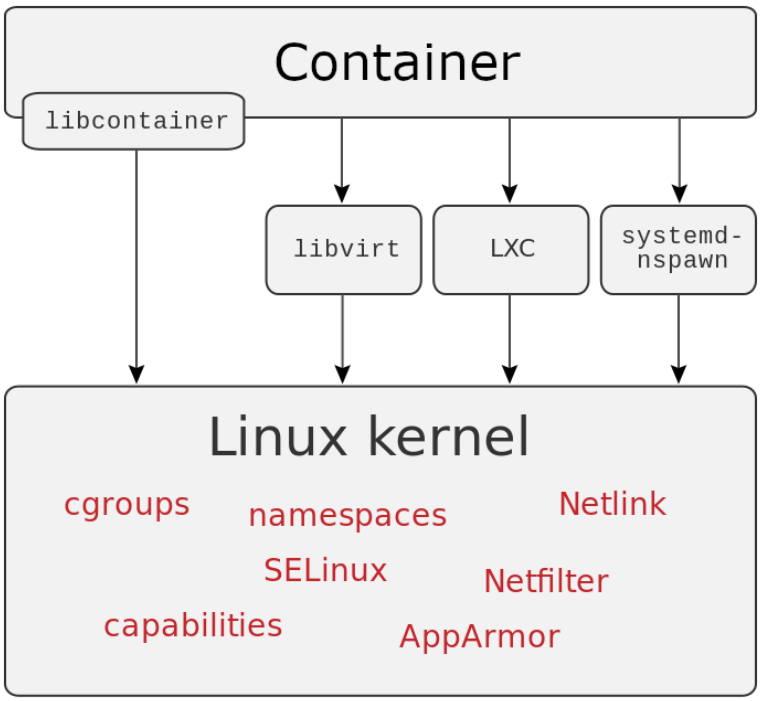

Abbildung 4: Schnittstelle vom Container zum Kernel [4]

[4] Quelle: https://www.datacenter-insider.de/container-technik-docker-co-a-480855/index2.html

Somit können CPU-Zyklen, Arbeitsspeicher, Blockspeicher und sonstige Schnittstellen über den Kernel angefordert und isoliert in dem jeweiligen Container zur Verfügung gestellt werden. [Riskhan u. a. 2017]

Die Kommunikation mit Containern funktioniert mithilfe einer virtuellen Netzwerkschnittstelle für jeden Container. Außerhalb der Container können die Ports dann auf die Netzwerkkarte gemappt werden, wobei auf dem Host dann natürlich jeder Port pro Netzwerkkarte nur einmal genutzt werden kann. [Anderson 2015]

Da Container nur als einzelnes Image abgelegt sind und kein Betriebssystem beinhalten, welches aktualisiert und gewartet werden müsste, beschränken sich die Installation und Deinstallation auf ein einfaches Kopieren oder Löschen des Containers. Aus einem Image können beliebig viele Container-Instanzen aufgerufen werden, da Schreibzugriffe nicht auf das Image zugreifen, sondern auf ein eigenes Dateisystem des Containers. Dieses Verhalten sorgt für eine sehr hohe Skalierbarkeit, da bei Bedarf einfach neue Instanzen der Anwendung gestartet werden können. [LEUNG u. a. 2018] Durch diese dynamische Skalierung und da die Container mit einem Bruchteil einer Sekunde im Vergleich zu virtuellen Maschinen oder dedizierten Servern sehr schnell gestartet und beendet werden können, haben sie eine deutlich kürzere durchschnittliche Lebensdauer.

Die genaue prozentuale Verteilung der Ausführungszeiten von Containern (Dauer zwischen Containerstart und Containerende) kann der Abbildung 5 entnommen werden.

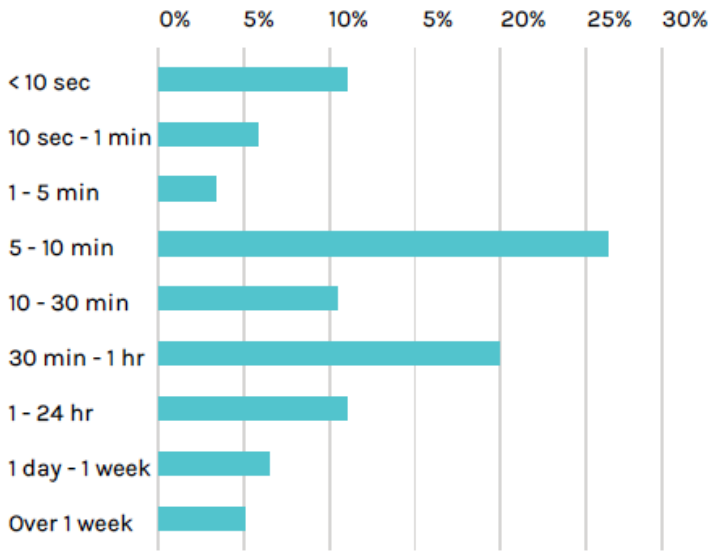

Abbildung 5: Lebensdauer eines Containers [5]

In den letzten 10 Jahren haben Container einen großen Wandel durchlebt, welcher in Abschnitt 3: Containertechnologien auf der nächsten Seite näher erläutert wird.

[5] Quelle: https://www.dailyhostnews.com/wp-content/uploads/2018/05/d3.png

3 Containertechnologien

In der Geschichte der Containertechnologie traten verschiedene Implementierungsformen auf. Hierbei waren die ersten Umsetzungen noch sehr einfach aufgebaut und wurden mit den Anforderungen an die Containerdienste immer komplexer. Im Folgenden findet sich eine Übersicht über die wichtigsten Technologien der Containerisierung.

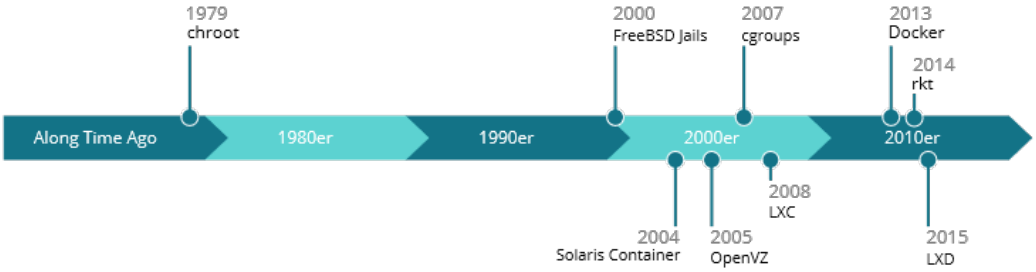

Abbildung 6: Containertechnologie im Laufe der Zeit

chroot

Chroot ist ein Befehl, der schon früh in Unix-Systemen eingebaut wurde. Er ermöglicht es, einem Prozess ein anderes Rootverzeichnis zu geben. Wird in einem Programm `chroot()` aufgerufen, wechselt es das Verzeichnis und kann nicht auf Dateien außerhalb der zugewiesenen Struktur zugreifen. Diese Abschottung eines Prozesses war nie als Sicherheitsfeature vorgesehen und wird hauptsächlich zur Virtualisierung eingesetzt. Mit dem Befehl können einzelne Prozesse auf Dateiebene von anderen Anwendungen getrennt werden, weitere Sicherheitsmechanismen oder Isolierungen gibt es nicht. [KAUR UND SINGH 2016; SMITH 1996; MANPAGES]

OpenVZ

Abbildung 7: Logo OpenVZ [6]

Im Jahr 2005 veröffentlichte die Firma SWsoft (später umbenannt zu Parallels) ihr Projekt OpenVZ unter der GNU GPL Lizenz. OpenVZ basierte auf der Idee der Container, ermöglicht es jedoch in jedem Container eine eigene Linux-Distribution auszuführen. Die durch die Containerumgebung abgegrenzten Betriebssysteme teilen sich einen Kernel. Dadurch ist der Overhead von OpenVZ deutlich geringer als bei der klassischen Vollvirtualisierung eines Betriebssystems. In den einzelnen Containern gibt es jeweils einen eigenen root-User und eine eigene Dateistruktur. Sie können unabhängig voneinander gestartet und gestoppt werden. Da sich die Betriebssysteme einen Kernel teilen, müssen die Gastsysteme ebenfalls Linux-Systeme sein. Da viele der Änderungen von OpenVZ den Linux Kernel betreffen, werden regelmäßig Änderungen von OpenVZ-Patches in diesen übernommen. [OpenVZ b; Ahmed u. a. 2008; OpenVZ a]

FreeBSD Jails

Mit der Veröffentlichung von FreeBSD 4.0 im Jahr 2000 war FreeBSD Jails das erste richtige System für Containervirtualisierung. Die FreeBSD Jails basieren auf dem Konzept von chroot. Auch hier wird das root-Verzeichnis eines Prozesses geändert. Zusätzlich verbessert Jails das Konzept um einige Aspekte.Jede Jail erhält einen eigenen Hostnamen und eine eigene IP-Adresse. Sie hat auch ihre eigenen Benutzer, inklusive einem root-Benutzer. [FreeBSD] Durch diese Prozessisolation ergibt sich eine Art Containersystem. Da die Jails als eigener Prozess laufen, können sie unabhängig voneinander gestartet und gestoppt werden. Jails wird gerne für den Einsatz in Netzwerkaufgaben eingesetzt, da die Performance sehr gut ist. Allerdings besitzt Jails kein so großes Ökosystem wie beispielsweise Docker oder OpenVZ.

[6] Quelle: https://upload.wikimedia.org/wikipedia/commons/b/bb/OpenVZ-logo.png?download

Daher wird es in der Containervirtualisierung von diesen Gegenspielern verdrängt.

Linux Containers (LXC)

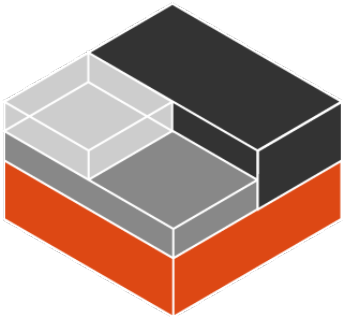

Abbildung 8: Logo LXC [7]

LXC ist seit der erstmaligen Veröffentlichung 2008 ein offizielles Kernelfeature und in den meisten Distributionen von Linux enthalten. LXC ist eine User Space-Schnittstelle für die Erstellung von isolierten Umgebungen innerhalb eines Systems. Dies geschieht durch die Nutzung von Kernel namespaces, Apparmor und SELinux-Profilen sowie chroots und cgroups. Diese Features standen schon vor LXC zur Verfügung, jedoch vereinigte sie LXC zu einer Schnittstelle für die Erzeugung von Containern. Zu Beginn der Entwicklung von LXC war die Isolation der Container nicht gut, sondern glich eher einer Abwandlung der chroot-Funktion. Mit der Zeit wurde die Abschottung jedoch immer besser und die LXC-Container wurden zu richtigen virtualisierten Umgebungen. Dies geschah unter anderem dadurch, dass ab Version 1.0 die einzelnen Container als unpriviliegierte Benutzer ausgeführt werden können. Zuvor war dies nicht möglich und eine Abgrenzung der Container nur bedingt gegeben. LXC ist eine Technologie, die von vielen weiteren Projekten eingesetzt wird, unter anderem von Proxmox oder Docker (bis Version 1.1)[BERNSTEIN 2014; BESERRA U. A. 2015; RIZKI U. A. 2016; UEHARA 2017; LXC]

LXD

Um die Verwendung von LXC zu vereinfachen, wurde das Tool LXD entwickelt. Es besteht aus drei Elementen: Einem Daemon, der eine REST-API zur

[7] Quelle: https://upload.wikimedia.org/wikipedia/commons/4/40/Linux_Containers_logo.png?download

Verfügung stellt, einem Befehlszeilenclient sowie einem Open-Stack Nova Plugin. Die vom Daemon bereitgestellte Schnittstelle ermöglicht es, über das Netzwerk auf das Management der Container zuzugreifen. LXD ist somit eine Erweiterung, die eine Schnittstelle zu LXC-Containern schafft. Über das Nova Plugin können die einzelnen LXD-Maschinen als Rechenknoten verwendet werden. [LXD]

Solaris Containers

Im Jahr 2004 veröffentlichte Oracle im Build 51 von Solaris 10 zum ersten Mal ein Feature mit dem Namen Solaris Containers. Solaris Containers stellt eine Technologie dar, mit der auf x86 und SPARC-Systemen Betriebssystemlevelvirtualisierung durchgeführt werden kann. Später zusammengelegt zu Solaris Zones, bestanden die beiden Technologien Solaris Containers und Solaris Zones parallel zueinander. Dabei war Zones eine klassische Virtualisierungsplattform mit Hypervisor und Containers eine Containertechnologie, die analog zu chroot funktionierte. Mit der Zusammenlegung von Containers und Zones zum neuen Zones wurde daraus eine Containerumgebung, in der die Container sicher voneinander und dem Host getrennt sind und von einem Resourcenmanagement kontrolliert werden. [Oracle; Drewanz und Grimmer]

Docker

dotCloud veröffentlichte im März 2013 das Projekt mit dem Namen Docker, dieses Projekt stellte Solomon Hykes auf der PyCon 2013 zum ersten Mal der Öffentlichkeit vor. [Hykes b] Ein paar Monate später kündigte dotCloud Inc. an, den Firmennamen zu Docker Inc. zu ändern und sich hauptsächlich der Entwicklung des Docker Ökosystems zu widmen. [Golub] Die Firma Docker Inc. (im Folgenden „Docker Inc.“ oder „Hersteller“) hat bis zum heutigen Tag

das Projekt Docker (im Folgenden „Docker“) weiterentwickelt und das zugehörige Ökosystem ausgebaut. So wurde unter anderem der DockerHub eingerichtet, eine Plattform um Images zu teilen und auszutauschen. [Docker-Inc. a]

Zu Beginn war Docker lediglich eine Werkzeugsammlung um LXC-Container zu verwalten. Jedoch baute Docker Inc. diese Sammlung immer weiter aus und erweiterte das System um Funktionen, die vom unterliegenden Linux-Betriebssystem nicht gegeben waren. Mit der Version 0.9 veröffentlichte Docker Inc. den neuen Treiber libcontainer und nutzte ihn ab diesem Zeitpunkt als native Umgebung für Docker-Container. [Hykes a] Zu Beginn unterstützte Docker lediglich Linux-Container und nutzte dazu unter Windows eine virtuelle Maschine (Windows 7 & 8) oder das Linux-Subsystem (Windows 10). Ab der Version 17.11 von Docker für Windows und dem Windows 10 Fall Creators Update konnten erstmals Windows Container genutzt werden. [Fischer] Auch entwickelte Docker Inc. weitere Zwischebenen, um sich von LXC zu lösen und die Umgebung in Module zu teilen. So entstand containerd, ein Container-Daemon, mit dem die Docker Engine kommuniziert. Dieser Daemon wiederum kann mit einem OCI-konformen Container-Tool umgehen und über dieses Container starten. Ein solches Tool ist das eigene runC. Auf diese Art können Docker-Container auch durch andere Orchestrierungstools wie Kubernetes oder Swarm verwaltet werden (Vgl. Abschnitt 5: Cluster auf Seite 19). [Liebel]

Zum heutigen Zeitpunkt ist Docker die führende Containerumgebung (Vgl. Abbildung 12), daher wird die Funktion derselben im Folgenden anhand eines Beispielcontainers aufgezeigt:

In diesem Beispiel soll innerhalb eines Docker-Containers ein Python-Skript ausgeführt werden. Als Basis für eine Container-Instanz dient ein schreibgeschütztes Image. Dieses enthält alle benötigten Teile des OS abgesehen vom Kernel, denn dieser wird bereits durch den Host zur Verfügung gestellt. Zusätzlich gehören auch benötigte Abhängigkeiten zu einem Image. Auf diesem schreibgeschützten Teil wird dann ein schreibbarer Layer aufgebaut, wenn von dem Image eine Container-Instanz abgeleitet wird. Wird die Container-

Instanz beendet, sind alle Änderung innerhalb des schreibbaren Layer verloren. Um die Änderungen zu sichern, kann ein sogenannter Snapshot angelegt werden, der dem Image einen weiteren read-only-Layer hinzufügt. Die Anzahl der Layer ist, je nach Docker-Version, auf 127 beschränkt. [Liebel; Docker-Inc. a]

Der Aufbau des Images unseres Beispielcontainers sieht aus wie in Abbildung 9 dargestellt:

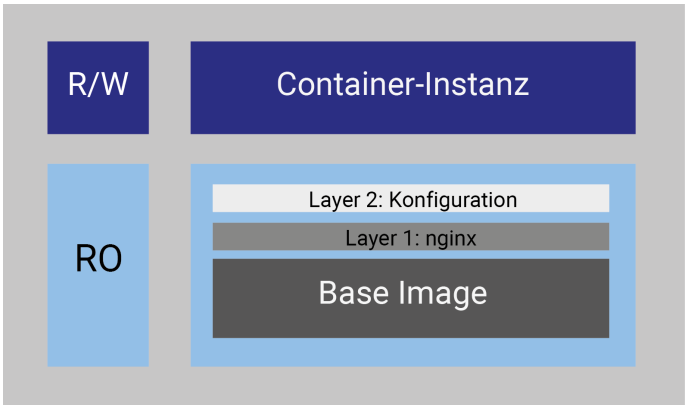

Abbildung 9: Image des Beispielcontainer

Als Basis dient ein Image, in dem Alpine Linux installiert ist. Python wird als weiterer Layer hinzugefügt. Zu diesen Layern wird noch das Python-Skript hinzugefügt. Dies zusammen ergibt dann den schreibgeschützten Teil, von dem die Container-Instanz abgeleitet wird. In dieser legt das Python-Skript Dateien an und schreibt in diese. Wird der Container beendet, werden alle angelegten Dateien verworfen.
Der Aufbau des Images kann in einem Dockerfile beschrieben werden. Für unser Beispiel sieht dieses wie folgt aus:

```
1 FROM python:3
2
3 WORKDIR /usr/src/app
4
5 COPY ./app.py ./app.py
6
7 CMD ["python", "./app.py"]
```

Listing 1: Dockerfile

In dem Dockerfile wird zuerst definiert, dass das Image auf dem vorhanden Python-Image aufbauen soll. Dieses wiederum baut auf einem Alpine-Linux-Image auf. Daraufhin wird das aktuelle Arbeitsverzeichnis im Container geändert. Anschließend wird die im Arbeitsverzeichnis des Host abgelegte Datei „app.py“ in das Arbeitsverzeichnis des Container übertragen und schließlich der Befehl definiert, der beim Start des Containers ausgeführt werden soll. Das Python-Skript legt eine Datei an und gibt deren Dateiname und Inhalt in der Konsole aus:

```
1 #!/usr/bin/python3
2
3 print("Ausgabe zur Kommandozeile")
4 file=open("datei.txt", "a+")
5 print("Dateiname: ", file.name)
6 file.write("Eine neue Zeile")
7 file.close()
8 print(open("datei.txt","r").read())
```

Listing 2: app.py

Nun kann mit dem Befehl `docker build` das Image erstellt werden. Dabei lädt Docker zunächst das Image von python aus dem oben genanten Docker-Hub herunter und legt darauf den Layer mit dem Python-Skript:

```
1  user@dockerpc:~$ docker build -t pythontest .
2  Sending build context to Docker daemon 311.4MB
3  Step 1/4 : FROM python:3
4   ---> 638817465c7d
5  Step 2/4 : WORKDIR /usr/src/app
6  ---> 8d3ab23442c9
7  Step 3/4 : COPY ./app.py ./app.py
8  ---> 2b0882cadee4
9  Step 4/4 : CMD ["python", "./app.py"]
10 ---> 5a8a392a0856
11 Successfully built 5a8a392a0856
12 Successfully tagged pythontest:latest
```

Listing 3: Terminalausgabe docker build

Nun liegt das Image bereit und es kann ein Container davon abgeleitet werden. Dazu reicht der einfache Befehl `docker run`, um den Container zu starten:

```
1 user@dockerpc:~$ docker run pythontest
2 Ausgabe zur Kommandozeile
3 Dateiname: datei.txt
4 Eine neue Zeile
```

Listing 4: Terminalausgabe docker run

Hier wird nun das Python-Skript in dem Container ausgeführt und der Container daraufhin beendet und somit auch die beschriebene Datei gelöscht. Wird der Container erneut ausgeführt, so wird eine neue Datei erstellt:

```
1 user@dockerpc:~$ docker run pythontest
2 Ausgabe zur Kommandozeile
3 Dateiname: datei.txt
4 Eine neue Zeile
5
6 user@dockerpc:~$ docker run pythontest
7 Ausgabe zur Kommandozeile
8 Dateiname: datei.txt
9 Eine neue Zeile
```

Listing 5: Terminalausgabe docker run mehrfach

Die Layer, aus denen Docker die verschiedenen Images aufbaut wurden von Docker Inc. in den ursprünglichen LXC-Container hinzugefügt und später zum hauseigenen libcontainer übernommen. Durch diese Layer ist es einfacher Ressourcen zu teilen. Wenn mehrere Images auf Python oder Alpine-Linux aufbauen sollen, so müssen diese beiden Ressourcen nicht mehrfach heruntergeladen werden, sondern stehen jedem Image zur Verfügung. Dies ist

einer der Gründe, warum Docker nach der Veröffentlichung große Popularität erreichte und heute die führende Containertechnologie ist. [Docker-Inc. b]

rkt

Abbildung 10: [8]

rkt (Ausprache wie "rocket") ist eine Containerengine, die sich als Alternative zu Docker etabliert hat und von CoreOS veröffentlicht wurde und weiterentwickelt wird. Das Projekt ist ein Open-Source-Projekt und unter der Apache License 2.0 veröffentlicht. [CoreOS b] Unter rkt werden viele Grundgedanken von UNIX umgesetzt. Z.B. liegen alle Container als Dateien vor, die einfach verwaltet werden können. Auch legt rkt einen großen Wert auf Sicherheit und setzt dazu verschiedene Techniken ein, die inzwischen von den meisten Konkurennten übernommen wurden. So kann rkt für jeden Container entscheiden, ob dieser auf Basis von KVM oder einer virtuellen Maschine isoliert wird und führt alle Prozesse, auch den Download von Images, als nicht priviligierter Benutzer aus. In rkt wird die kleinste Einheit Pod genannt. Sie kann aus einem oder mehreren Containeren bestehen, welche sich die Ressourcen teilen. So passt das Konzept von rkt direkt zu den Konzepten von Cluster-Managern. Auch besitzt rkt keinen zentralen Service, der alle Container überwacht, sondern arbeitet direkt mit dem systemeigenen systemd zusammen, um die Container zu verwalten. Somit lässt sich rkt auch direkt mit Kubernetes verknüpfen, für das der Herausgeber von rkt, CoreOS, die kommerzielle Implementierung Tectonic entwickelt. rkt unterstützt auch die Konvertierung von Docker-Containern zu rkt-Pods. [CoreOS a; Yanar]

[8] Quelle: https://github.com/rkt/rkt/raw/master/logos/rkt-horizontal-color.png

4 Container in der Softwareentwicklung

Der Einsatz von Containern erleichtert die Entwicklung von Software in vielerlei Hinsicht. So müssen Entwickler ihre Applikationen für verschiedene Plattformen nicht grundlegend verschieden entwerfen und die Programmiersprache kann meist frei gewählt werden. Ob für Windows, Linux, MacOS, Cloud-Plattformen oder andere, der Fokus der Entwicklung kann deutlich stärker auf die Funktionalitäten der Applikation gerichtet werden, wenn die Eigenheiten der Ziel-Plattform in den Hintergrund rücken. Das macht den gesamten Entwicklungsprozess einfacher und somit effizienter. Mithilfe der Abstraktion durch Container vermeidet man Inkompatibilitätsprobleme auf den Host-Geräten und auch die Entwicklung sowie Softwaretests gestalten sich dadurch leichter, schneller und effizienter, denn alle für die Applikation wichtigen Daten, Tools und Systembibliotheken sind im Container vorhanden. [BURNS u. a. 2016]

Entwickler können davon ausgehen, dass ihre Applikation auf verschiedenen Systemen funktionieren wird und können sie immer unter konsistenten Bedingungen testen, egal wie später die Umgebung aussehen mag. Dies erhöht die Zuverlässigkeit enorm. Man ist nicht länger Abhängig von der Verfügbarkeit von identischen Entwicklungs- und Testsystemen und der Entwickler kann auf seinem eigenen Rechner auch schnelleres Feedback erhalten, wenn er den Container lokal ausführt und debuggt.

Auch ermöglichen Container die Verwendung von Microservices. Sonst als monolithische Applikation entworfene Software kann von Entwicklern unabhängig in mehreren Teilen erstellt werden, was die Agilität deutlich fördert. Außerdem ist das Software Deployment sehr simpel, da es nur gilt, ein Container Image zu erzeugen und zu verteilen. Dies kann auch über Container-Orchestration-Tools wie Kubernetes nach dem Prinzip von Continous Delivery automatisiert werden. Die Ausführung läuft auf jedem System dann jedes Mal gleich ab. [IT AGILE]

Dementsprechend benötigt man auch für Weiterentwicklung und Wartung der Applikationen weniger Zeit und Personal als wenn man für jedes Sys-

tem eigene Entwickler mit Fachkenntnissen bräuchte. Bei Veränderungen an der Hardware, kurzfristigem Wechsel, Neuanschaffungen aber auch bei Upgrades des Betriebssystems hat eine Firma keine größeren Schwierigkeiten durch Inkompatibilitäten zu befürchten. Somit ist sie auch freier in der Wahl ihrer Geräte.

Auch das sogenannte Monitoring, die laufende Überwachung der Systeme, über Schnittstellen (APIs) ist mit Containern kein Problem. Logs können von jeder Applikation erstellt, dann einfach gesammelt und in ein Management-System übertragen werden. Die Erkennung und Eingrenzung von Fehlerquellen beschleunigt sich dadurch, dass die Applikation im Container gekapselt ist und keine weiteren Programme oder Betriebssystemteile die Fehlersuche erschweren. Auch können die Container-Applikationen einfach mit ihrem vordefinierten Idealzustand neugestartet werden, sobald ein Problem erkannt wird. Diese Vereinfachung durch Abstraktion hilft dann nicht nur dem Entwickler, sondern trägt zur Zufriedenheit der Nutzer bei.

Besonders wenn es darum geht, neue Applikationen zu entwerfen, deren Zielplattformen noch nicht endgültig festgelegt sind, oder bei einem Umzug in die Cloud. Gerade bei Cloud-Diensten sind Container unter anderem wegen ihres geringeren Ressourcen-Umfangs beliebt. [LEUNG u. a. 2018]

"Container eignen sich optimal für dienstbasierte Architekturen. Im Gegensatz zu monolithischen Architekturen, bei denen alle Teile einer Anwendung miteinander verknüpft sind [...], werden diese Komponenten bei einer dienstbasierten Architektur getrennt. Durch eine Trennung und Arbeitsteilung werden Ihre Dienste auch dann weiter ausgeführt, wenn andere fehlschlagen. Damit bleibt Ihre gesamte Anwendung zuverlässiger."[Google]

Tools, die sich speziell um das Ressourcen-Management kümmern, sind in vielen Containern mit inbegriffen, sodass beispielsweise der zur Verfügung stehende Speicher sinnvoll begrenzt werden kann, um Out-of-Memory-Abstürzen vorzubeugen. Das schont die Server, auf denen die Applikationen laufen, und reduziert den Hardware-Bedarf und die Kosten, wenn weniger virtuelle Maschinen mit eigenem vollwertigen Betriebssystem aufgesetzt werden müssen. [BURNS u. a. 2016]

5 Cluster

Wie in Abschnitt 1: Einleitung auf Seite 1 genannt, wurden in der Vergangenheit dedizierte Server für jeweils einen Service genutzt. Dies hatte den Nachteil einer geringen Serverauslastung sowie bei nicht redundanten Servern die Gefahr eines Totalausfalls eines Services. Applikationen ließen sich nicht ohne weiteres von einem Server auf einen anderen umziehen, da sie tief in das Hostsystem integriert waren.

Cluster Manager verbinden mehrere Maschinen zu einer Einheit. Während Lösungen wie Apache Mesos eine Abstraktion der Hardware vornehmen, basieren Kubernetes und Docker Swarm auf der Container-Architektur. Diese Cluster Manager übernehmen die Verwaltung der Container sowie ihre Zuordnung zu den jeweiligen Maschinen.

Clustering sorgt für eine verbesserte Redundanz und erhöht somit die Ausfallsicherheit. Außerdem lässt sich so eine bessere Ressourcen-Allokation vornehmen.

Thema aktueller Forschungsarbeiten ist die Verbesserung des Schedulings, um die Ressourcennutzung zu optimieren. [Liu u. a. 2018]

6 Risiken der Containertechnologie

Die Containertechnologie erobert in den letzten Jahren immer mehr die Rechenzentren. Doch welche Risiken verbergen sich dahinter und wie kann man sich schützen?

Durch die hohe Anzahl an Container pro Server ist das Risiko bei einer Sicherheitslücke deutlich höher, da sich diese dann in beispielsweise 80 Containern, anstatt in vier virtuellen Maschinen oder einem dedizierten Server ausnutzen lässt. [LANLine]

Um sich den Aufwand für die Konfiguration der Images zu sparen (diese kann sehr aufwendig sein), verwenden viele Administratoren vorgefertigte Container-Images aus einem Respository. Dabei muss dem Ersteller vertraut werden, dass das Image keinen Schadcode oder Hintertüren enthält, da der Aufwand für eine genaue Prüfung des Container-Inhalts sehr aufwendig wäre. Im Juni 2018 hatte die Sicherheitsfirma Kromtech berichtet, dass über das Repository Docker Hub mehrere Images über ein Jahr lang verfügbar waren, die Schadcode zum Minen von Kryptowährungen enthielten. Diese wurden insgesamt fünf Millionen mal installiert, bevor die Betreiber von Docker Hub reagierten und diese entfernten. [Kromtech] Die betroffenen Administratoren hätten das Risiko minimieren können, indem sie nur über das offizielle Docker Repository die Container bezogen hätten. Dort werden Images vor ihrer Veröffentlichung geprüft. [Docker] Ein Angreifer müsste zur Verteilung eines infizierten Images den Schadcode verstecken, sodass er bei der Prüfung nicht sichtbar wird. Dies stellt eine wesentlich höhere Hürde dar.

Werden Applikationen in Containern richtig verpackt, so sind die einzigen Abhängigkeiten nach außen hin die Systemaufrufe des Betriebssystems. Dies verbessert die Portabilität der Anwendungen ungemein, allerdings sind auch Systemaufrufe wie z.B. Socket-Schnittstellen sowie hardwarespezifische Systemaufrufe nicht auf allen Systemen einheitlich, wodurch die Portabilität eingeschränkt wird. Die Open Container Initiative der Linux Foundation arbeitet neben einem Standard für Container Formate auch an einem Standard für Container Runtimes. Dieser könnte helfen, die Schnittstelle zwischen Container und Betriebssystem besser festzulegen.

Container können nicht gegen Einflüsse schützen, die nicht vom Betriebssystem verwaltet werden. Hierzu sind virtuelle Maschinen als zusätzliche Sicherheitsschicht notwendig. [BURNS u. a. 2016]

Nicht zuletzt haben die Sicherheitslücken Meltdown ([Lipp u. a. 2018]) und Spectre ([Kocher u. a. 2018]) gezeigt, dass über Sicherheitslücken in Prozessoren containerübergreifende Angriffe auf Applikationen möglich sind. Hiergegen schützten virtuelle Maschinen allerdings ebenfalls nicht.

7 Aktuelle Lage

Abbildung 11 zeigt, dass trotz einiger Risiken der Container Technologie Unternehmen weltweit immer mehr Geld in die Containerisierung ihres Unternehmens investieren. Laut einer Umfrage, welche auf der DockerCon durchgeführt wurde, investierten 32% der Unternehmen mindestens 500.000$ jährlich, um die Containerisierung in ihrer Organisation voranzutreiben. [PORTWORX]

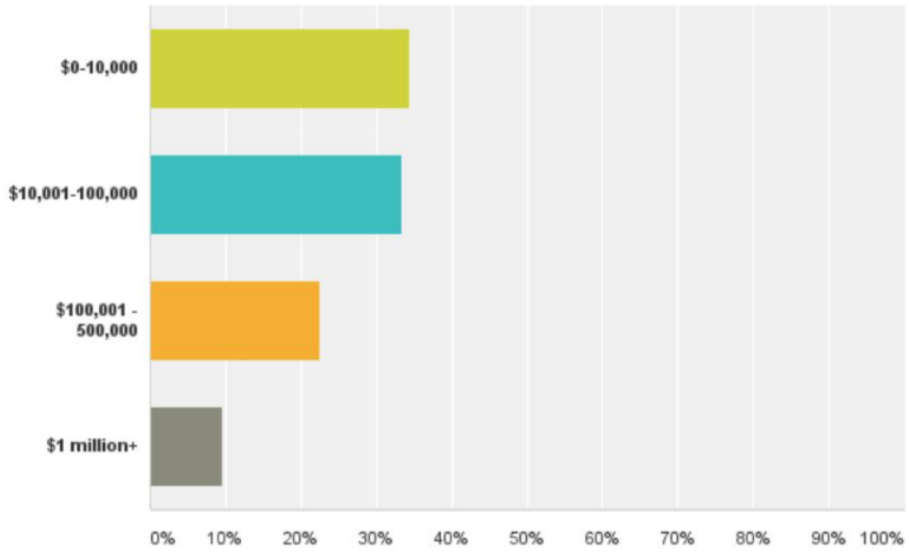

Abbildung 11: Investitionen in die Containerisierung [9]

[9] Quelle: https://portworx.com/wp-content/uploads/2017/04/survey-investment-768x471.png

Aus Abbildung 12 ist zu entnehmen, dass 83% der produktiv eingesetzten Container von Docker stammen. An zweiter Stelle der meist verwendeten Container findet sich CoreOS, welches von der Firma Red Hat übernommen wurde. Mesos Containerizer und LXC machen zusammen nur 5% aller eingesetzten Container aus. [CARTER]

Abbildung 12: Benutzung von Containertechnologien [10]

Laut eines Berichts der Container-Monitoring-Firma Sysdig stieg die Anzahl der durchschnittlich verwendeten Container pro Host 2018 um 50% im Vergleich zum Vorjahr. Das entspricht nun etwa 15 Containern. Laut des Berichts ist 154 die maximale Anzahl von Containern, die bisher gleichzeitig auf einer Maschine laufen. [CARTER]
Abbildung 13 stellt dies grafisch dar.

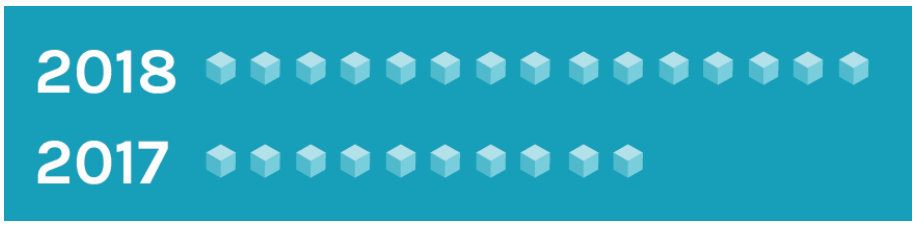

Abbildung 13: Container je Maschine [11]

[10] Quelle: https://www.dailyhostnews.com/wp-content/uploads/2018/05/d2.png
[11] Quelle: https://www.dailyhostnews.com/wp-content/uploads/2018/05/d1.png

Kubernetes sei die meist genutzte Plattform, um Container zu orchestrieren und wird von Software-Unternehmen wie Microsoft und IBM verwendet. Das beliebteste Tool, um Container-Cluster für große Firmen auszurollen, sei jedoch Mesos Containerizer. [CARTER]
Die genaue Verteilung ist Abbildung 14 zu entnehmen.

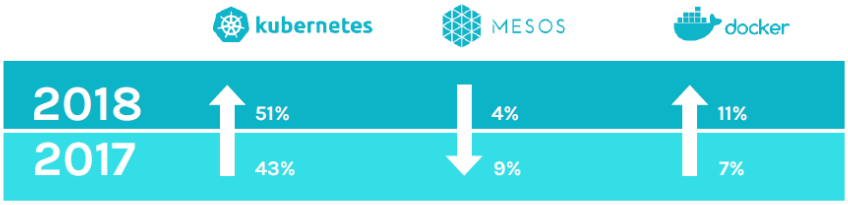

Abbildung 14: Cluster-Manager [12]

8 Einsatzszenarien von Containern an der Hochschule

Im folgenden soll betrachtet werden, welche Einsatzmöglichkeiten sich für Container in der IT-Infrastruktur der Hochschule Albstadt-Sigmaringen anbieten und welche Vorteile dies mit sich bringen würde.

Prädestiniert für den Einsatz sind die Server-Dienste der Hochschule. Hier wäre denkbar den in Abschnitt 5: Cluster auf Seite 19 vorgestellten Ansatz eines Serververbunds in Verbindung mit einem Cluster Manager wie Docker Swarm oder Kubernetes zu verwenden. Die einzelnen Applikationen wie z.B. das E-Learning System Ilias, die CentOS Instanzen, die Bibliotheksdienste und die Website könnten dann in mehreren Container-Instanzen laufen. Der

[12] Quelle: `https://www.dailyhostnews.com/wp-content/uploads/2018/05/d4.png`

Cluster-Manager würde dynamisch die Last auf die verschiedenen Maschinen verteilen und könnte Instanzen mit Programmfehlern erkennen sowie schnell neustarten.

Außerdem wäre es möglich, Lastspitzen abzufangen, indem Serverkapazitäten anderer Bildungseinrichtungen genutzt werden, um dort bei Bedarf Containerinstanzen zu starten.

Beispiel:
Ein konkreter Anwendungsfall für die Hochschue Albstadt-Sigmaringen ist die Containerisierung der Oracle Datenbank. Die Datenbank wird im Rahmen mehreren Vorlesungen und Praktika von Professoren und Studenten genutzt. Dabei ist die Zahl der Zugriffe auf die Datenbank außerhalb von Vorlesungen oder Praktika sehr gering. Während Praktika und Vorlesungen wird die Datenbank rege genutzt, wodurch die Performance spürbar leidet. Dieser starke Unterschied der benötigten Leistung könnte perfekt durch Containerisierung der Datenbank ausgeglichen werden, da je nach Nachfrage innerhalb von Sekunden Container hoch- oder runtergefahren werden können. Diese Vorgehensweise spart nicht nur Strom, sondern garantiert auch Professoren und Studierenden eine gute Performance der Datenbank. Oracle bietet im Docker Store ein vorgefertigtes Docker Image an, was die Nutzung von Oracle auf Docker sehr vereinfacht. Eine detaillierte Anleitung für die Containerisierung von Oracle Datenbanken findet man auf der Oracle Homepage. [13] Um die Leistungsfähigkeit der Datenbank weiter zu optimieren, könnte ein in Abschnitt 5: Cluster auf Seite 19 vorgestellter Cluster-Manager verwendet werden.

9 Fazit und Ausblick

In den letzten Jahren stieg die Nutzung von Containern rasant an und viele Unternehmen investieren massiv in diese Technologie (vgl. Abschnitt 7: Aktuelle Lage auf Seite 21). Der ausschlaggebende Grund hierfür sind der verrin-

[13] https://apex.oracle.com/pls/apex/germancommunities/dbacommunity/tipp/6241/index.html

gerte Overhead, die Performance-Vorteile, und das vereinfachte Deployment von Anwendungen. Im Bereich der Softwareentwicklung bringen Container einen großen Vorteil. Es ist möglich innerhalb von Sekunden ein System mit den nötigen Eigenschaften aufzusetzen, um die Anwendung darauf auszuführen. Außerdem kann so sichergestellt werden, dass Test- und Produktivsystem absolut identisch sind und keine unerwarteten Effekte auftreten. Durch die immer simpleren Containertechnologien können Unternehmen mit wenig Budget und Personal die Containerisierung ihres Unternehmens vorantreiben und die Vorteile dieser ausschöpfen. Somit ist auch in den nächsten Jahren mit dem vermehrten Einsatz von Containertechnologien im Businessbereich zu rechnen.

Container sind kein Ersatz für virtuelle Maschinen, sie können sogar sinnvoll kombiniert werden. Dabei ist es vor allem zu empfehlen den Docker-Host zu virtualisieren, sodass dieser bei einem Hardwareausfall trotzdem weiter laufen kann und bei Wartungen der Dienst weiter zur Verfügung gestellt werden kann. Somit können die in Abschnitt 1: Einleitung auf Seite 1 genannten Vorteile ausgeschöpft werden.

Durch die Diversität bei den Containertechnologien können die Anwender Container auf den verschiedensten Betriebssystemen nutzen. Zudem wird durch die Konkurenz der Technologien untereinander die Weiterentwicklung und Optimisierung der Technik gefördert. Ein Beispiel dazu ist die Open Container Initiative (OCI), die gegründet wurde, um die Technologie zu Standardisieren und Entwicklung zu fördern.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

LXC Linux Containers

OCI Open Container Initiative

Literaturverzeichnis

it agile

AGILE it: *Container im Agilen Entwicklungsprozess*. https://www.it-agile.de/fileadmin/docs/Whitepaper_ContainerImAgilenEntwicklungsprozess_it-agile.pdf, Abruf: 25.07.2018

Ahmed u. a. 2008

Ahmed, M. ; Zahda, S. ; Abbas, M.: Server consolidation using OpenVZ: Performance evaluation. In: *2008 11th International Conference on Computer and Information Technology*, 2008, S. 341–346

Anderson 2015

Anderson, Charles: Docker. In: *IEEE Software* 32 (2015), Nr. 3, 102 - c3. http://www.redi-bw.de/db/ebsco.php/search.ebscohost.com/login.aspx%3fdirect%3dtrue%26db%3degs%26AN%3d102288020%26site%3dehost-live. – ISSN 07407459

Bernstein 2014

Bernstein, D.: Containers and Cloud: From LXC to Docker to Kubernetes. In: *IEEE Cloud Computing* 1 (2014), Sept, Nr. 3, S. 81–84. http://dx.doi.org/10.1109/MCC.2014.51. – DOI 10.1109/MCC.2014.51. – ISSN 2325–6095

Beserra u. a. 2015

Beserra, D. ; Moreno, E. D. ; Endo, P. T. ; Barreto, J. ; Sadok, D. ; Fernandes, S.: Performance Analysis of LXC for HPC Environments. In: *2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems*, 2015, S. 358–363

BURNS u. a. 2016

BURNS, BRENDAN ; GRANT, BRIAN ; OPPENHEIMER, DAVID ; BREWER, ERIC ; WILKES, JOHN: Borg, Omega, and Kubernetes. In: *Communications of the ACM* 59 (2016), Nr. 5, 50 - 57. http://www.redi-bw.de/db/ebsco.php/search.ebscohost.com/login.aspx%3fdirect%3dtrue%26db%3degs%26AN%3d115178361%26site%3dehost-live. – ISSN 00010782

Carter

CARTER, Eric: *2018 Docker Usage Report*. https://sysdig.com/blog/2018-docker-usage-report/, Abruf: 26.07.2018

CoreOS a

COREOS: *Homepage rkt*. https://coreos.com/rkt/, Abruf: 25.07.2018

CoreOS b

COREOS: *Projektrepository rkt*. https://github.com/rkt/rkt, Abruf: 25.07.2018

Docker

DOCKER: *Official repositories on Docker Hub*. https://docs.docker.com/docker-hub/official_repos/, Abruf: 25.07.2018

Docker-Inc. a

DOCKER-INC.: *Docker Documentation*. https://docs.docker.com/, Abruf: 25.07.2018

Docker-Inc. b

DOCKER-INC.: *Docker Milestones*. https://www.docker.com/company, Abruf: 26.07.2018

Drewanz und Grimmer

DREWANZ, Detlef ; GRIMMER, Lenz: *The Role of Oracle Solaris Zones and Linux Containers in a Virtualization Strategy*. http://www.oracle.com/technetwork/articles/servers-storage-admin/zones-containers-virtualization-1880908.html, Abruf: 25.07.2018

Edwards 2016

EDWARDS, Chris: Containers Push Toward the Mayfly Server. In:

Communications of the ACM 59 (2016), Nr. 12, 24 - 26. http://www.redi-bw.de/db/ebsco.php/search.ebscohost.com/login.aspx%3fdirect%3dtrue%26db%3degs%26AN%3d120050683%26site%3dehost-live. – ISSN 00010782

Fischer

FISCHER, Carl: *Docker for Windows 17.11 with Windows 10 Fall Creators Update*. https://blog.docker.com/2017/11/docker-for-windows-17-11/, Abruf: 25.07.2018

FreeBSD

FREEBSD: *FreeBSD Handbuch*. FreeBSD. – 277 – 293 S. https://download.freebsd.org/ftp/doc/de/books/handbook/book.pdf, Abruf: 25.07.2018

Golub

GOLUB, Ben: *dotCloud, Inc. is Becoming Docker, Inc.* https://blog.docker.com/2013/10/dotcloud-is-becoming-docker-inc/, Abruf: 25.07.2018

Google

GOOGLE: *Container bei Google*. https://cloud.google.com/containers/, Abruf: 25.07.2018

Hykes a

HYKES, Solomon: *Docker 0.9: introducing execution drivers and libcontainer*. https://blog.docker.com/2014/03/docker-0-9-introducing-execution-drivers-and-libcontainer/, Abruf: 25.07.2018

Hykes b

HYKES, Solomon: *The future of Linux Containers*. https://www.youtube.com/watch?v=wW9CAH9nSLs, Abruf: 25.07.2018

Kaur und Singh 2016

KAUR, N. ; SINGH, M.: Improved file system security through restrictive access. In: *2016 International Conference on Inventive Computation Technologies (ICICT)* Bd. 3, 2016, S. 1–5

Kocher u. a. 2018

KOCHER, Paul ; GENKIN, Daniel ; GRUSS, Daniel ; HAAS, Werner ; HAMBURG, Mike ; LIPP, Moritz ; MANGARD, Stefan ; PRESCHER, Thomas ; SCHWARZ, Michael ; YAROM, Yuval: Spectre Attacks: Exploiting Speculative Execution. In: *CoRR* abs/1801.01203 (2018). http://arxiv.org/abs/1801.01203

Kromtech

KROMTECH: *Cryptojacking invades cloud. How modern containerization trend is exploited by attackers*. https://kromtech.com/blog/security-center/cryptojacking-invades-cloud-how-modern-containerization-trend-is-exploited-by-a Abruf: 25.07.2018

LANLine

LANLINE: *Container sicher nutzen*. https://www.lanline.de/container-sicher-nutzen/, Abruf: 24.07.2018

LEUNG u. a. 2018

LEUNG, ANDREW ; SPYKER, ANDREW ; BOZARTH, TIM: Titus: Introducing Containers to the Netflix Cloud. In: *Communications of the ACM* 61 (2018), Nr. 2, 38 - 45. http://www.redi-bw.de/db/ebsco.php/search.ebscohost.com/login.aspx%3fdirect%3dtrue%26db%3degs%26AN%3d127712851%26site%3dehost-live. – ISSN 00010782

Liebel

LIEBEL, Oliver: *Skalierbare Container-Infrastrukturen*. ebook. Rheinwerg Verlag Gmbh. – ISBN 97–3–8362–4367–4

Lipp u. a. 2018

LIPP, Moritz ; SCHWARZ, Michael ; GRUSS, Daniel ; PRESCHER, Thomas ; HAAS, Werner ; MANGARD, Stefan ; KOCHER, Paul ; GENKIN, Daniel ; YAROM, Yuval ; HAMBURG, Mike: Meltdown. In: *CoRR* abs/1801.01207 (2018). http://arxiv.org/abs/1801.01207

Liu u. a. 2018

LIU, Bo ; LI, Pengfei ; LIN, Weiwei ; SHU, Na ; LI, Yin ; CHANG, Victor:

A new container scheduling algorithm based on multi-objective optimization. In: *Soft Computing* (2018), Jul. http://dx.doi.org/10.1007/s00500-018-3403-7. – DOI 10.1007/s00500–018–3403–7. – ISSN 1433–7479

LXC

LXC, Offizelle H.: *Linux Containers - LXC*. https://linuxcontainers.org/lxc/, Abruf: 25.07.2018

LXD

LXD, Offizelle H.: *Linux Containers - LXD*. https://linuxcontainers.org/lxd/, Abruf: 25.07.2018

Manpages

MANPAGES, Linux: *chroot - Wurzelverzeichnis wechseln*. https://manpages.debian.org/stretch/manpages-de-dev/chroot.2.de.html, Abruf: 24.07.2018

OpenVZ a

OPENVZ, Homepage: *History*. https://wiki.openvz.org/History, Abruf: 24.07.2018

OpenVZ b

OPENVZ, Homepage: *News*. https://wiki.openvz.org/News, Abruf: 24.07.2018

Oracle

ORACLE: *Oracle Solaris Zones Introduction*. https://docs.oracle.com/cd/E36784_01/html/E36848/zones.intro-1.html#scrolltoc, Abruf: 25.07.2018

portworx

PORTWORX: *2017 Annual Container Adoption Survey: Huge Growth in Containers*. https://portworx.com/2017-container-adoption-survey/, Abruf: 25.07.2018

redhat

REDHAT: *Was ist Virtualisierung?* https://www.redhat.com/de/topics/virtualization/what-is-virtualization, Abruf: 24.07.2018

Riskhan u. a. 2017

RISKHAN, Basheer ; KE, Zhou ; MUHAMMAD, Raza: Energy Management of the System: An Empirical Investigation of Virtualization Approaches in Static and Dynamic Modes. In: *Information Technology Journal* 16 (2017), Nr. 1, 1 - 10. http://www.redi-bw.de/db/ebsco.php/search.ebscohost.com/login.aspx%3fdirect%3dtrue%26db%3degs%26AN%3d120592540%26site%3dehost-live. – ISSN 18125638

Rizki u. a. 2016

RIZKI, R. ; RAKHMATSYAH, A. ; NUGROHO, M. A.: Performance analysis of container-based hadoop cluster: OpenVZ and LXC. In: *2016 4th International Conference on Information and Communication Technology (ICoICT)*, 2016, S. 1–4

Smith 1996

SMITH, R. E.: Mandatory protection for Internet server software. In: *Proceedings 12th Annual Computer Security Applications Conference*, 1996. – ISSN 1063–9527, S. 178–184

TUM

TUM: *Grundlagen Container-Virtualisierung*. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&cad=rja&uact=8&ved=0ahUKEwjR_Lq79LncAhVQjqQKHbGqCVsQFgg6MAE&url=https%3A%2F%2Fwww.tmf-ev.de%2FDesktopModules%2FBring2mind%2FDMX%2FDownload.aspx%3FMethod%3Dattachment%26Command%3DCore_Download%26EntryId%3D28917%26PortalId%3D0&usg=AOvVaw26Jn9693iPmp8xCk4w9e9g, Abruf: 25.07.2018

Uehara 2017

UEHARA, M.: Performance Evaluations of LXC Based Educational Cloud in a Bare Metal Server. In: *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, 2017, S. 415–420

Yanar

YANAR, Erkan: Im Steigflug. , 07, 94. https://www.heise.de/ix/heft/Im-Steigflug-3754458.html, Abruf: 25.07.2018

A Anhang

A.1 Begründung der ausgewählten Literatur

Zur Verfügung stand lediglich sehr aktuelle Literatur, da die Containervirtualisierung erst seit Erscheinen von Docker im Jahr 2013 in der IT-Branche an Bedeutung gewonnen hat. Daher sind auch viele der hier betrachteten Werkzeuge erst in den vergangen Jahren entwickelt worden.

In Anbetracht des kurzen Zeitraumes, der den Autoren zur Verfügung stand, konnte keine Fernleihe durchgeführt werden. Eine Vorbestellung der Literatur war daher ebenfalls nicht möglich. Am ersten Tag der Bearbeitung des Artikels stand außerdem die Bibliothek aufgrund des Betriebsausflugs nicht zur Verfügung, weshalb auch nicht auf die physischen Medien zurückgegriffen werden konnte. Deshalb hat sich die Bücher- bzw. Artikelauswahl auf die über die Hochschule verfügbaren digitalen Medien beschränkt.

Die Literaturauswahl umfasst außerdem Dokumentationen der gängisten Software zum Thema Container-Technologie. Diese wurde zum Verständnis des Aufbaus und der Nutzung des jeweiligen Werkzeugs genutzt. Die Dokumentationen sind online bzw. zusammen mit dem jeweiligen Source Code verfügbar und werden von den Entwicklern zur Verfügung gestellt. Daher handelt sich bei den Dokumentationen um eine verlässliche Quelle über das jeweilige Werkzeug.

Auf den offizellen Webseiten der verschiedenen Hersteller und Projekten werden von den Entwicklern oder Firmen offizelle Informationen publiziert oder auch oben genannte Dokumentationen veröffentlicht. Der Inhalt der Webseiten kann als verlässliche Quelle angesehen werden, da hier der Ersteller des Produkts direkt veröffentlicht.

Blog Einträge dienten den Autoren als Ideengeber für einen Teil des Inhalts der vorliegenden Arbeit. Da diese am Puls der Zeit sind, zeigen sie aktuelle Trends und populäre Software zum Thema Container-Technologie auf. Ein

Blog wird nicht überprüft und stellt daher selbstverständlich keine zuverlässige Quelle dar. Zur weiteren Recherche wurden aufgrund dessen wissenschaftlich verlässliche Quellen verwendet.