Forschungsbereich
MaschinenbauInformatik
und Virtuelle
Produktentwicklung

Betrachtung gängiger Softwarelösungen zur Isolierung von Applikationen auf Edge Devices im industriellen Umfeld anhand von Container-basierter Virtualisierung

Bachelorarbeit

Johannes Elsner

Matrikel.-Nr.: 01351461

BACHELORARBEIT

Betrachtung gängiger Softwarelösungen zur Isolierung von Applikationen auf Edge Devices im industriellen Umfeld anhand von Container-basierter Virtualisierung

ausgeführt zum Zwecke der Erlangung des akademischen Grades Bachelor of Science (B.Sc.)
unter der Leitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Manfred Grafinger

Univ.-Ass. Dipl.-Ing. Patrick Rosenberger

E307-04 – Institut für Konstruktionswissenschaften und Produktentwicklung

Forschungsbereich Maschinenbauinformatik und Virtuelle Produktentwicklung

eingereicht an der Technischen Universität Wien

Fakultät für Maschinenwesen und Betriebswissenschaften

von

Johannes Elsner

Matrikelnummer: 01351461

Kaasgrabengasse 5-B, 1190 Wien

Ort, Datum WIEN, 16.11.2019

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Aus fremden Quellen wörtlich oder sinngemäß übernommene Zitate sind als solche kenntlich gemacht. Die Arbeit hat noch nicht anderweitig in gleicher oder ähnlicher Form zu Prüfungszwecken vorgelegen.

Die Arbeit darf über die Hochschulbibliothek zugänglich gemacht werden.

WIEN, 16.11.2019
Ort, Datum


Unterschrift

Gender Erklärung

Die Inhalte dieser Bachelorarbeit wurden so weit wie möglich geschlechterneutral formuliert. War dies nicht möglich, wurde aus Gründen der besseren Lesbarkeit das generische Maskulinum verwendet. Es wird an dieser Stelle ausdrücklich darauf hingewiesen, dass entsprechende Formulierungen geschlechterunabhängig zu verstehen sind.

Kurzfassung

Container-basierte Virtualisierung birgt großes Veränderungspotential für Unternehmen im Sektor Informationstechnik. Treibende Kräfte sind unter anderem Konzerne wie Google, Microsoft und Amazon. Allerdings wird die Welt der Container auch von einer aktiven Entwickler-Community getragen, die von offenen Standards unterstützt, eine Grundlage für die Anwendung von Containern legen. Seit Docker 2013 die Container-Technologie populär machte, steigt die Anzahl der vorhandenen Angebote in diesem Bereich laufend. Die vorliegende Bachelorarbeit möchte einen Überblick über die unterschiedlichen Produkte geben und diese anschließend vergleichen. Um auch für einen interessierten Laien, der bisher noch wenig Kontakt zu Containern hatte, verständlich zu sein, steht eine Erläuterung der wichtigsten Begriffe rund um die Container-Technologie am Anfang der Arbeit. Folgend erhält der Leser einen Einblick in die verschiedenen Softwarelösungen, die in vier Kategorien eingeteilt werden. Diese Kategorien werden schließlich für einen Vergleich der beschriebenen Softwarelösungen verwendet, um die bestehenden Unterschiede eingängig darzustellen. Die dazu notwendigen Informationen wurden durch die qualitative Auswertung von Literatur gesammelt. Dadurch konnte die Bedeutung des OCI Projektes für die Standardisierung wichtiger Werkzeuge, von Docker als Container-Anbieter und Kubernetes als Container-Orchestrierung bestätigt werden. Hingegen fehlt es noch an aussagekräftigen Analysen zum Overhead der vorgestellten Container-Technologien. Auch Edge Computing hat bereits Bedeutung erlangt, obwohl die meisten Anbieter noch mit Cloud Computing werben. Anwendungsmöglichkeiten der Container-Technologie auf Edge Devices werden im letzten Kapitel der Arbeit anhand von Experimenten aufgezeigt.

Abstract

Container-based virtualization could be a game changer for companies in the IT-sector. After Docker released its container engine in 2013, this type of virtualization became popular very fast. Today there are quite a lot of software suppliers for different parts of the container ecosystem. Most of them are community-developed and benefit from strong software standards. Although most of the software is available as open-source, big tech companies like Google, Microsoft and Amazon are just as interested in the potential of this technology and are already investing not only money but also know-how by participating in the organizations for developing standards. The bachelor thesis at hand wants to provide an overview of the many offerings and to compare them to each other. First the thesis explains some basic concepts, which the reader would have to know to understand containers. After that there is a compilation of a selection of the current container software offerings. To have a better view on the subject, the offerings are divided into four categories, which represent groups of products with similar characteristics. Following, the offerings are compared to each other in these four groups, to see the differences clearly. The necessary data was obtained by a qualitative evaluation of various scientific resources. The main findings of this bachelor thesis are on the one hand the verification of the importance of the OCI project, Docker and Kubernetes in the field of container technology. On the other hand, there are no distinct results of the difference between container offerings, when evaluating their resource consumption. Although Edge Computing gained some importance already, providers rather advertise cloud computing solutions yet. However, the importance of container technology on Edge Devices is illustrated by some successful experiments in the last chapter.

Danksagung

An dieser Stelle möchte ich bei all jenen bedanken, die mich bei der Erstellung der vorliegenden Bachelorarbeit unterstützt haben. Ein besonderer Dank gilt Dipl. Ing. Stefan Dumss, der mich geduldig angeleitet hat und mir immer mit Rat und Tat zur Seite stand. Ebenso möchte ich mich bei allen Mitarbeitern des Instituts für Maschinenbauinformatik und Virtuelle Produktentwicklung bedanken, die mit hilfreichen Anregungen und konstruktiver Kritik die Richtung der Arbeit maßgeblich mitgestaltet haben.

Außerdem möchte ich mich bei meiner Großmutter und meinem Vater für das gewissenhafte Korrektur lesen ganz herzlich bedanken.

Abschließend soll hier auch meiner ganzen Familie ein großer Dank ausgesprochen werden, deren Unterstützung für den Abschluss dieses Studiums ausschlaggebend war.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ACS	Azure Container Service
AKS	Azure Kubernetes Service
API	Application Programming Interface
AWS	Amazon Web Service
CLI	Command Line Interface
CNCF	Cloud Native Computing Foundation
CNI	Container Network Interface
CPU	Core Processor Unit
CRIU	Checkpoint/Restore In Userspace
DoS-Attacken	„Denial of Service“-Attacken
EC2	(Amazons) Elastic Compute Cloud
ECR	(Amazons) Elastic Container Registry
ECS	(Amazons) Elastic Container Service
EKS	(Amazons) Elastic Kubernetes Service
et al.	„et alii“, und andere
etc.	et cetera („und so weiter“)
etcd	eine verteilte Schlüssel-Werte-Datenbank
GCE	Google Compute Engine
GKE	Google Kubernetes Engine
IaaS	Infrastructure as a Service
K8s	Kubernetes
keine Quellen	keine Angabe in den verfügbaren Quellen
Ltd.	Limited, Unternehmensform
LXC	**L**inu**X** **C**ontainer
LXD	**L**inu**X** Container **D**aemon
o. J.	ohne Jahresangabe
OCI	Open Container Initiative
OS	operating system („Betriebssystem“)
OSF	OpenStack Foundation
PaaS	Platform as a Service
RAM	Rapid Access Memory
u.a.	unter anderem
vgl.	vergleiche
VM	virtuelle Maschine
z.B.	zum Beispiel

1 Einleitung

Das „Digitale Zeitalter“ hat in den vergangenen Jahren in allen Bereichen des Lebens zahlreiche Veränderungen gebracht. Technische Neuerungen wie Smartphones, Smart Devices und Künstliche Intelligenz sind für jedermann sichtbar, werden vermehrt als Begriffe der Alltagssprache verwendet und prägen die Diskussionen in breiten Bevölkerungsschichten. Viele Veränderungen sind dabei auf neue Konzepte oder technische Entwicklungen zurückzuführen. Eine dieser neuen Möglichkeiten ist die Verwendung der in der vorliegenden Arbeit näher dargelegten Container-basierten Virtualisierung. Schon heute wenden viele große Technologiekonzerne diese praktische Technik auf die eine oder andere Weise an, um ihre Produkte schneller, sicherer und kostengünstiger anbieten zu können, oder bieten Container selbst als Produkt erfolgreich an.

1.1 Problemstellung

Container-basierte Virtualisierung gewinnt besonders im Bereich des Cloud Computing immer mehr an Bedeutung. Die Kombination der Container-basierten Virtualisierung mit dem „Internet of Things“ und „Edge Computing“ verspricht, Computersysteme schneller, stabiler und günstiger zu machen (vgl. Chen et al. 2019: 203–204).

Um diese Vorteile flächendeckend in der Industrie nutzen zu können, denken nun einige Firmen daran, Standards zu definieren, um Abhängigkeiten zu vermeiden. Die Open Container Initiative leistet dazu wichtige Beiträge und hat unter anderem einen De-facto-Standard für das Format und das Laufzeitverhalten von Container-Software definiert (vgl. The Linux Foundation 2015).

Besonders das Zusammenspiel von unterschiedlichen Softwarekomponenten kann jedoch kritisch sein, da oft nicht alle Funktionalitäten für eine industrielle Anwendung von einer einzigen Softwarelösung abgedeckt werden. Die vorliegende Arbeit stellt daher einige Produkte im Bereich der Container-Technologie vor, präsentiert Unterschiede der dargestellten Produkte prägnant und übersichtlich und versucht eine Bewertung hinsichtlich der Anwendbarkeit in einem industriellen Umfeld.

1.2 Zielsetzung

Mit dieser Arbeit soll dem interessierten Leser ein erster Einstieg in den Bereich der Container-Technologie ermöglicht werden. Dazu werden wichtige Begriffe im Zusammenhang mit Container-basierter Virtualisierung erläutert und für diese Arbeit definiert. Anschließend werden die gängigen Softwarelösungen beschrieben und übersichtlich vier Kategorien zugeordnet, die einen raschen Überblick über die Welt der Container erlauben. Schlussendlich werden die vorgestellten Produkte innerhalb dieser Kategorien anhand von aussagekräftigen Kriterien miteinander verglichen und in Bezug auf die Anwendung im industriellen Umfeld diskutiert und bewertet.

1.3 Struktureller Aufbau

Methodologie

Allgemeine Vorgehensweise

Die vorliegende Arbeit hatte das Ziel, mit einer umfassenden Literaturrecherche die für eine Isolierung von Anwendungen praktikable Methode der Container-basierten Virtualisierung zu beleuchten und dem interessierten Leser einen Überblick über den Hintergrund, die derzeitigen Softwarelösungen und mögliche neue Entwicklungen dieses Bereiches zu geben.

Dazu wurde auf verschiedene Weise Literatur gesammelt. Nach Möglichkeit wurden jeweils mehrere Quellen zu einem Thema verglichen und geprüft, ob diese inhaltlich übereinstimmen. Die Behandlung der Fragestellung wurde auf einer theoretischen Ebene durchgeführt und die Erhebung von relevanten und aussagekräftigen Daten wurde durch eine große Anzahl an Literatur erreicht.

Beim Sammeln der Daten wurde meistens ein ähnliches Schema verwendet. Zuerst wurde in allgemeinen Internetquellen ein Überblick zur Thematik gewonnen. Basierend auf dem gewonnenen Überblick konnten anschließend eindeutige Suchbegriffe gewählt werden, die bei der Suche in Universitätsbibliotheken, auf Google Scholar, bei Fachverlagen und Buchhandlungen passende Ergebnisse lieferten.

Erstellung des Exposees

Nach intensivem Einlesen in die Fachliteratur begann eine erste Auswahl relevanter Informationen, um das Themengebiet sinnvoll eingrenzen zu können. Die Literatur wurde anhand von Kriterien, wie Häufigkeit, Autoren-Empfehlungen oder Alleinstellungsmerkmale gegenüber anderen Produkten, durchsucht und mit den Ergebnissen wurde eine erste Liste von bedeutenden Softwarelösungen erstellt. Damit konnte ein vorläufiger Titel, mögliche Forschungsfragen und der Fokus für die vorliegende Bachelorarbeit formuliert werden.

Verschriftlichung der vorliegenden Arbeit

Das Verschriftlichen der vorliegenden Arbeit, insbesondere der Kapitel zwei und drei, folgte einem klaren Muster. Da die Arbeit aufbauend gestaltet ist, nämlich am Beginn Grundlagen für die später ausgearbeiteten Themen erläutert werden, wurde die Abschrift ebenfalls in dieser Reihenfolge ausgeführt. Die Vorgehensweise folgte einem einfachen, sich wiederholenden Algorithmus. Wenn zu einem Teilbereich auf kein Hintergrundwissen aufgebaut werden konnte, brachten nicht-wissenschaftlichen Quellen einen ersten Überblick. Anschließend wurden nach Möglichkeit wissenschaftliche Quellen gesucht, die das Detail, beispielsweise Edge Devices oder den Linux Befehl chroot, behandeln. Sobald mehrere Quellen gefunden wurden, die im Inhalt übereinstimmten, wurde von den ergiebigsten Quellen eine Zusammenfassung relevanter Fakten erstellt. Teilweise mussten bestimmte Details von nicht-wissenschaftlichen Quellen übernommen werden. Diese Quellen wurden mit wissenschaftlicher Literatur verglichen, um die Richtigkeit der Quelle anhand der überschneidenden Informationen zu überprüfen.

Die Ermittlung der Daten für die Tabellen in Kapitel vier gestaltete sich mitunter schwierig. Die Vorgehensweise orientierte sich trotzdem an demselben Muster, wobei oft die bereits genutzten

Quellen ausreichten. Die durch die tabellarische Aufbereitung ermöglichte Übersicht stellt gleichwohl einen erheblichen Mehrwert dar. Für die anschließende Diskussion wurden die vorliegenden Daten mit dem durch das eingehende Studium einschlägiger Literatur gewonnenen Wissen verglichen. Daraus wurden bestimmte Eigenschaften oder Tendenzen der Angebote ersichtlich, die, wenn nötig, mit entsprechenden Beispielen und Quellen untermauert wurden.

Fazit und Kurzfassung geben einen Überblick über die wesentlichen, gewonnenen Erkenntnisse und beleuchten das Potential für weiterführende, auf dieser Arbeit aufbauende Untersuchungen.

Aufbau der vorliegenden Arbeit

Der Aufbau der vorliegenden Arbeit hält sich weitgehend an eine logische Abfolge der behandelten Forschungsfragen. Zuerst wird der Leser in die theoretischen Grundlagen eingeführt. Anschließend werden die derzeit bedeutenden Produkte in einer Sammlung von Beschreibungen dargestellt, wobei diese übersichtlich vier Kategorien zugeordnet werden. Diese Kategorien werden gewählt, indem mehrere Quellen mit vergleichbaren Aufstellungen miteinander verglichen werden. Dabei sollen die Kategorien besonders zwei Eigenschaften erfüllen: Erstens eine klare Zuteilung ermöglichen und zweitens die Produkte auf eine Weise gruppieren, dass das Zusammenspiel der Angebote bei der Verwendung in Kombination miteinander offensichtlich wird. Deshalb werden die vorliegenden Kategorien verwendet und die Benennung an die verwendete Literatur angelehnt.

Die Vergleichstabellen folgen ebenfalls einer vorangegangenen Auswertung von einschlägiger Literatur. Ausgehend von der Überlegung, anhand welcher Kriterien sich ein Leser eine fundierte Meinung zu einem Produkt machen könne, werden zuerst mit Brainstorming viele mögliche Eigenschaften gesammelt und diese dann im Abgleich mit ähnlichen Gegenüberstellungen in der Literatur verglichen. Aus diesem Vergleich folgt sowohl die Formulierung der Kriterien als auch eine aussagekräftige Auswahl. Bei der Auswahl wird besonders darauf Wert gelegt, dass die Forschungsfragen mit den vorliegenden Kriterien beantwortet werden können. Die Diskussion der Vergleichstabellen wird direkt anschließend eingefügt, um dem Leser das Umblättern zwischen Daten und Diskussion zu ersparen und damit den Lesefluss zu verbessern.

Abgeschlossen wird die Arbeit mit einem Fazit, dass die Ergebnisse der Arbeit und deren Hintergrund zusammenfasst.

Kapitelübersicht

In vier inhaltlichen Kapiteln stellt diese Arbeit einen Leitfaden durch das weite Ecosystem der Container-Technologie dar.

Im zweiten Kapitel werden die Grundlagen dazu besprochen. Neben allgemeinen Erläuterungen zum Thema Virtualisierung und einer Gegenüberstellung der Eigenschaften von virtuellen Maschinen und Containern werden in diesem Kapitel einige Linux-Kernel-Funktionen erklärt, die die Basis für Linux Container bilden, die derzeit häufigste Container-Art. Im Anschluss werden die Begriffe Edge Devices und Edge Computing erklärt.

Das dritte Kapitel gibt zuerst eine Übersicht zu Container-bezogenen Vokabeln. Die weiteren Abschnitte beschäftigen sich mit einer Darstellung und Beschreibung bedeutender Softwarelösungen der Container-Technologie. Durch eine Einteilung in vier Hauptkategorien und eine grafische Übersichtskarte am Anfang der Auflistung soll dabei ein guter Überblick entstehen.

Unter Vergleich und Diskussion, dem vierten Kapitel, finden sich vergleichende Aufstellungen der Fakten zu den vorgestellten Softwarelösungen anhand ausgewählter Eigenschaften, die anschließend mit Bezug auf die Anwendung im industriellen Bereich diskutiert werden. Der letzte Abschnitt dieses Kapitels stellt einen Bezug zwischen Containern und Edge Devices her.

Das fünfte Kapitel fasst die Ergebnisse der Literaturrecherche übersichtlich zusammen und schließt die Ausführung mit einem Ausblick auf mögliche nachfolgende wissenschaftliche Arbeiten ab.

2 Theoretische Grundlagen

2.1 Virtualisierung

Virtualisierung ist ein Framework oder eine Methodik, anhand derer Computerressourcen in mehrere Ausführungsumgebungen aufgeteilt werden können, wie Adam Singh in seinem Artikel über Virtualisierung allgemein darlegt (Singh 2004). Wird ein Computer virtualisiert, spricht man von einer „Virtual Machine“ beziehungsweise virtuellen Maschine, kurz „VM“, die eine isolierte virtuelle Instanz des Computers mit eigenem Betriebssystem und Programmen darstellt. Eine solche VM ist vollkommen unabhängig und bekommt die benötigten Rechnerressourcen von einem „Hypervisor“ zugeordnet. Der Hypervisor ist eine kleine Softwareschicht auf der physischen Hardware oder einem beliebigen Betriebssystem, kurz dem „Host“, der die VM vom Host entkoppelt. Auf einem Host können auf diese Weise mehrere VMs laufen, die alle von einem Hypervisor die benötigten Computerressourcen zugewiesen bekommen. Zentrale Eigenschaften von VMs sind Partitionierung, Isolierung, Einkapselung und Unabhängigkeit von der Hardware. Durch diese Kerneigenschaften ist es möglich, mehrere, separate, sichere und isolierte Umgebungen auf einem Server zu schaffen, deren Ressourcen gut kontrollierbar sind und die sich außerdem einfach migrieren lassen (vgl. Red Hat, Inc. 2019b; VMware 2014).

2.1.1 Container-basierte Virtualisierung

Container-basierte Virtualisierung, „Container-Technologie“ oder auch unter Betriebssystem-Virtualisierung bekannt, gilt als leichtgewichtige Alternative zu virtuellen Maschinen, da im Allgemeinen weniger Ressourcen benötigt werden. Anstatt die vorhandenen Ressourcen vollständig zu virtualisieren, wie das bei VMs der Fall ist, bilden Container nur ein Abbild des benötigten Betriebssystems mit den verlangten Funktionen (vgl. Morishima et al. 2016: 226).

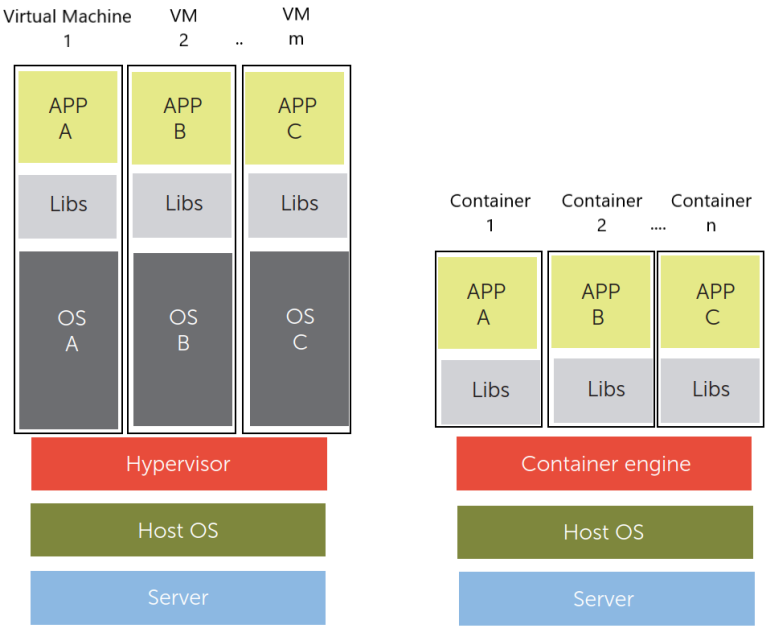
Container stellen damit eine standardisierte Einheit von Softwarecode dar, die den Quellcode der Anwendung, die notwendigen Abhängigkeiten, Bibliotheken und Datenpakete in ein Paket zusammenfasst. Dieses Paket wird durch den Container von der umliegenden Betriebsumgebung getrennt. Durch diese Entkoppelung läuft die darin verpackte Anwendung unabhängig vom Betriebssystem schnell und zuverlässig ab (vgl. Docker Inc. 2019).

Üblicherweise bestehen Container aus zwei Teilen. Der erste Teil beinhaltet ein Archiv mit dem benötigten Dateisystem inklusive installierten Software-Komponenten. Der zweite Teil definiert hingegen die Laufzeit-Einstellungen und steuert die Anbindung an externe Ressourcen. Obwohl alle Container auf sehr ähnliche technische Grundlagen zurückgreifen, sind die Unterschiede der am Markt befindlichen Angebote teils signifikant groß (vgl. Morishima et al. 2016: 226).

2.1.2 Container vs. virtuelle Maschine

Überblicksmäßige Gegenüberstellung

Container und virtuelle Maschinen unterscheiden sich grundlegend. Container-Technologie abstrahiert die Betriebssystemebene und deren Funktionen, wie zum Beispiel Dateimanager, Netzwerkzugang und laufende Programme. Im Gegensatz dazu werden bei einer virtuellen Maschine die Ressourcen selbst, u.a. CPU, RAM und Festplattenspeicher, virtualisiert (vgl. Morishima et al. 2016: 226).



Quelle: basierend auf (Bernstein 2014: 82)

Abbildung 1 - Gegenüberstellung virtuelle Maschine vs. Container

Abbildung 1 zeigt den unterschiedlichen Aufbau von virtuellen Maschinen und Containern, aus denen sich einige unterschiedliche Eigenschaften ergeben. Während Container einen geringen Speicherbedarf haben, portabel und flexibel sind und (zum Teil) beinahe keinen Overhead erzeugen, haben virtuelle Maschinen ein vollständiges Betriebssystem installiert, wodurch sie zwar eine gute Isolierung und Flexibilität aufweisen jedoch auch deutlich mehr Speicherplatz benötigen (vgl. Rathore et al. 2013: 88). Ein vollständiges Betriebssystem bezeichnet die Gesamtheit der für den anwendungsunabhängigen Betrieb notwendigen Programme (vgl. Brause 2017: 4). Auf den Punkt gebracht dienen virtuelle Maschinen besonders für Anwendungen im Bereich IaaS (Infrastructure as a Service) and PaaS (Platform as a Service), wo sie mit der Abstraktion der Hardware und Automatisierung helfen, Kosten zu sparen.

Weiters sind virtuelle Maschinen im Vorteil, wenn die auszuführenden Programme unterschiedliche Betriebssysteme oder Betriebssystemversionen benötigen. Da Container-Anwendungen sich das gleiche Betriebssystem beziehungsweise den gleichen Betriebssystem-Kernel teilen, können sie diese Aufgabe zwar nicht übernehmen, ersparen sich dafür aber einen Teil des für das Betriebssystem notwendigen Speicherbedarfes (vgl. Bernstein 2014: 82).

Anders ist dies bei der Verwendung mit Microservices, wo das langsame Hochfahren und der hohe Wartungsaufwand mehrerer Betriebssysteme von virtuellen Maschinen einen Nachteil bedeuten (vgl. Ramalho/Neto 2016: 3). Einen guten Überblick der Unterschiede bietet die folgende Tabelle.

Tabelle 1 - Vergleich der Eigenschaften von VM und Container-Virtualisierung

Quelle: (vgl. Abraham et al. 2019: 149)

Vergleichsfaktoren	virtuelle Maschinen	Container
Benutzung des Wirt-Betriebssystems	benötigt eigenes Betriebssystem (OS)	Betriebssystem des Wirtes wird mitbenutzt
Zeitdauer zum Starten einer Instanz	benötigt zum Hochfahren gleich viel Zeit wie ein traditionelles OS	fährt sehr schnell hoch
Standardisierung	sind generell Betriebssystem-spezifisch	sind generell anwendungsorientiert
Portabilität	schlechter übertragbar	leicht beweglich
benötigte Serveranzahl	wenige VM's pro Server	viele Container pro Server
Sicherheit	das OS einer VM kann Sicherheitsfunktion für die Benutzerdaten bieten; abhängig vom Hypervisor	Sicherheit kann über den geteilten Kernel beeinträchtigt sein; in gewisser Weise fehlen daher Sicherheitsmaßnahmen
Grad der Redundanz	viele redundante Informationen, da jede VM ein eigenes OS hat	da Teile des OS's geteilt werden, weniger Redundanz
Zugriff auf die Hardware	kein direkter Zugriff möglich	direkter Zugriff ist in einigen Fällen grundsätzlich möglich
Ressourcen-verbrauch	hoher Verbrauch	geringer Verbrauch
Anforderungen an Zwischenspeicher	hoher Bedarf, da jeweils eigenes OS betrieben werden muss	niedriger Bedarf, da OS geteilt wird
Mitbenutzung von Dateien und Verzeichnissen	gemeinsame Benutzung von Dateien grundsätzlich nicht möglich	gemeinsame Benutzung mittels Linux Befehlen möglich

Virtuelle Maschinen unterscheiden sich von Containern insbesondere durch eine höhere Belastung der vorhandenen Hardwareressourcen, wie oben dargelegt. Edge Devices besitzen hingegen oft nur begrenzte Ressourcen. Aus der Kombination dieser beiden Anforderungen ist klar ersichtlich, dass Container-basierte Virtualisierung dafür gut geeignet sein kann. Daher steht diese Form der Virtualisierung im Fokus dieser Arbeit (vgl. Ramalho/Neto 2016: 4).

2.1.3 Linux Kernel Funktionen als Container-Grundlage

Linux Container (LXC)

LXC ist eine Oberfläche des Linux Kernels, die dem Benutzer verschiedene Isolierungsmechanismen zur Verfügung stellt. Dadurch können auf einem einzigen Linux-Host mehrere voneinander separierte Abbilder/Container ausgeführt werden. LXC bietet auch die Option, unprivilegierte Container zu erstellen. Die Ausführung als unprivilegiert ist im Allgemeinen von Vorteil. Allerdings können hierbei auch Probleme mit der Sicherheitsfreigabe auftreten. Durch die Transportfähigkeit von Containern können die Abbilder zwar grundsätzlich ohne große Veränderungen auf verschiedenen Rechnersystemen laufen, allerdings wird dazu ein Linux-Kernel vorausgesetzt, wobei der Support von der Linux Distribution und deren Engagement abhängt (vgl. Canonical Ltd. o. J.-a; Martin et al. 2018: 5).

In erster Linie dienen die folgenden Funktionen zur Erstellung von LXC Containern, die teilweise nachfolgend beschrieben werden:

Kernel namespaces, CGroups, Apparmor und SELinux Profile, Seccomp Richtlinien, Chroots, Kernel capabilities (vgl. Canonical Ltd. o. J.-a).

Durch geschickte Kombination der vorgestellten Funktionen können verschiedenartige Container erstellt werden, die für die eine oder andere Aufgabe besser geeignet sind. Da manche Eigenschaften von unterschiedlichen Mechanismen erzeugt werden können, kann der Hersteller einer Container-Technologie wählen, für welchen Anwendungsbereich seine Container besonders gut angepasst sein sollen.

namespaces

Namespaces, Linux namespaces oder Kernel namespaces abstrahieren eine globale Systemressource, sodass diese für einen im namespace ausgeführten Prozess wie eine eigene, isolierte Instanz dieser globalen Ressource aussieht. Veränderungen an dieser neuen Instanz können nur Prozesse innerhalb des namespace sehen, außerhalb davon ist die instanziierte Ressource nicht sichtbar. Linux stellt die folgenden sieben namespaces bereit: „cgroup“, „IPC“, „Network“, „Mount“, „PID“, „User“, „UTS“ (vgl. Kerrisk/Biedermann 2017e).

Einer der Verwendungszwecke von namespaces sind Container. Durch die Prozess-Level Isolation von Linux Ressourcen der namespaces kann jeder Container nur mehr seinen eigenen Bereich der jeweiligen Ressource einsehen, obwohl diese für den Container wie die tatsächliche globale Ressource aussieht. Deshalb kann ein Prozess innerhalb eines Containers Applikationen außerhalb des eigenen Containers oder eines Tochter-Containers nicht beeinflussen oder einen Zugang darauf erhalten (vgl. Sabharwal/W 2015: 18).

Wichtige Einschränkung ist, wie Stéphane Graber in seinem Blog erläutert, dass die Sicherheit eines Containers stark von den gewählten Einstellungen abhängt. Insbesondere privilegierte Container können nicht als sicher gelten, da sich ein solcher Container jegliche Berechtigungen wieder erteilen kann. Der User namespace ist seiner Meinung nach aber ein probates Mittel, um LXC Container sicher zu gestalten (vgl. Graber 2014f).

cgroups

CGroups bieten Mechanismen, die dazu genutzt werden, um zusammengehörende Prozesse und alle daraus entstehenden Unterprozesse in eigene hierarchische Gruppen zusammenzufassen, die dann jeweils mit bestimmtem speziellen Verhalten ausgestattet werden können (vgl. Menage 2004).

Um dies zu erreichen, ordnet die cgroup einer Gruppe von Prozessen eigene Parameter zu. „subsystem“ genannte Module können über die zugeordneten Parameter die Prozessgruppen auf spezielle Art behandeln. Typischerweise werden dadurch die bereitstehenden Ressourcen für die Prozessgruppe gesteuert, allerdings kann es grundsätzlich jede Verhaltensweise steuern, die auf eine Gruppe von Prozessen angewendet werden soll. Die cgroups stehen zueinander in einer hierarchischen Beziehung, sodass zu jeder Zeit klar ist, welcher Prozess welcher cgroup beziehungsweise welchem „subsystem“ zugeordnet ist. Diese Hierarchie wird jeweils durch eine Instanz der cgroup aufgebaut, wobei es mehrere verschiedene solcher Hierarchien geben kann, die je einen Teil der im Gesamtsystem vorhandenen Prozesse abbilden. Solche cgroups können von den einzelnen Usern entsprechend ihrer Berechtigungen gebildet und verändert werden (vgl. Menage 2004).

Alleinstehend sind cgroups nur dazu geeignet Prozesse nachzuverfolgen (vgl. Menage 2004). Nützlich sind cgroups für die Beschränkung von Ressourcen für Linux-Benutzer und -Gruppen. Wirklich sicherheitsrelevant ist die Möglichkeit, mit cgroups festzulegen, welcher Benutzer welche Dateien oder Komponenten lesen, schreiben oder ausführen darf. Ebenso können DoS Attacken verhindert werden, indem ein Benutzer durch eine entsprechende cgroup nur auf limitierte Ressourcen (Arbeitsspeicher, CPU und PIDS) zugreifen kann. Diese Einstellungen können auch von höheren Hierarchiestufen vererbt werden (vgl. Canonical Ltd. o. J.-b; Graber 2014f).

chroot

Chroot ist ein Systemaufruf in Linux und anderen Unix Betriebssystemen, der es dem root-Benutzer erlaubt, das root-Verzeichnis für Prozesse und deren Kinder zu verändern. Dadurch wird die Sicht dieser Prozesse auf das neue Verzeichnis reduziert und ebenfalls der Zugriff auf externe Dateien verhindert. Der ursprüngliche Zweck war die Isolierung alter Programme und ähnlicher Anwendungen in einer Form des Kompatibilitätsmodus. Daher hat der chroot-Mechanismus aus der Sicherheitsperspektive diverse Schwachstellen, die durch Patches gelöst werden müssen. Da auch einige wichtige Systemdateien vom neuen Verzeichnis aus zugreifbar sein müssen, das unter Umständen nicht sicher ist, sollte der chroot-Befehl ausschließlich Administratoren vorbehalten sein (vgl. Haque et al. 2018: 1; van Tilborg/Jajodia 2011: 206–207).

Linux Security Modules:

Linux Security Modules sind ein Programmiergerüst, das dem Linux Kernel erlaubt, unterschiedliche Sicherheitsmodelle zu unterstützen, ohne ein konkretes zu favorisieren (vgl. Wikimedia Foundation Inc. 2019a). Zwei übliche Varianten sind im Folgenden vorgestellt.

- AppArmor

AppArmor ist ein Sicherheitsframework für Linux, das als Mandatory Access Control System Anwendungen einzeln kontrollieren kann. Zum Schutz von sicherheitskritischen Anwendungen

können in Profilen Zugriffsrechte festgelegt werden, die die systeminternen Dateirechte ergänzen. In diesen Profilen werden die zu schützenden oder zu kontrollierenden Dateien/Prozesse über ihren Speicherpfad benannt und dann die gewünschten Rechte zugeordnet. Seit 2009 unterstützte Canonical AppArmor verstärkt und sorgte schließlich für die Integrierung in den Linux-Kernel. Im Vergleich zu SELinux gilt AppArmor als anwenderfreundliche Alternative (vgl. ubuntuusers.de 2019; Wikimedia Foundation Inc. 2019a).

- SELinux

Wie AppArmor ist SELinux eine Erweiterung für Linux, die die Sicherheit durch stärkere Zugriffskontrolle erhöht. Anders als AppArmor wird bei SELinux die Zuordnung von Berechtigungen über mit den einzelnen Elementen verknüpfte Labels durchgeführt. Dazu werden “role-based access control” (RBAC), “domain and type enforcement” (DTE) und “multilevel security” (MLS) kombiniert (vgl. Schreuders et al. 2011: 3). So kann bestimmten Elementen eine Erlaubnis exakter erteilt oder entzogen werden. Außerdem bietet SELinux neben den bekannten Unterscheidungen lesen, schreiben, ausführen auch noch weitere Unterteilungen, wie zum Beispiel „Verknüpfung auflösen“ oder „Datei verschieben“, an. Dadurch ist eine genau auf die eigenen Bedürfnisse zugeschnittene Lösung möglich (vgl. Morris 2017i).

Ursprünglich war SELinux für die Anwendung durch IT-Experten und Administratoren vorgesehen, da SELinux-Regeln sehr komplex sein können und oft nur über das Command Line Tool eingerichtet werden. Heute ist es dennoch der am weitesten verbreitete Linux-Sicherheitsmechanismus und ist auch allein für viele Nutzer ausreichend. (vgl. Schreuders et al. 2011)

Kernel Capabilities

Linux Kernel Capabilities gehören zu den früh implementierten Sicherheitsfunktionen der Linux Betriebssysteme. Seit Linux Kernel 2.2 werden die Berechtigungen, die ursprünglich mit dem Superuser verbunden waren, in einzelne Listen, namentlich capabilities, aufgeteilt, die somit einzeln vergeben werden können. (vgl. Kerrisk 2002d)

Auf diese Weise kann Prozessen und Dateien vieles unmöglich gemacht werden. Dies wirkt sich jedoch neben dem gewünschten Sicherheitszugewinn oft auch auf die Funktionstüchtigkeit des Programmes aus. (vgl. Graber 2014f)

Seccomp

Seccomp, kurz für SECure COMPuting Mode, ist ein Sicherheitsfeature des Linux Kernels. Ursprünglich konnte damit ein Prozess in einen „sicheren“ Status gebracht werden, in dem nur mehr die Systemaufrufe exit(), sigreturn(), read() und write() verwendet werden konnten. Diese Systemaufrufe sollten die Sicherheit des restlichen Systems vor diesem Prozess garantieren, wurden in dieser Form jedoch bis zum Upgrade auf Linux 3.5 im Jahr 2012 kaum genutzt. Nun kann mithilfe eines kleinen Berkeley Packet Filter (BPF) Programms präzisiert werden, welche Systemaufrufe einem Prozess gestattet und welche untersagt werden (vgl. The Linux Foundation o. J.). Dadurch dient Seccomp dem Bilden von Sandboxes und wird durchaus dazu verwendet. Beispiele sind Chrome Browser, OpenSSH oder Firefox OS (vgl. Edge 2015).

2.2 Edge Devices im Internet of Things (IoT)

Edge Device

Edge Device ist ein Sammelbegriff für Geräte, die sich am Rande eines Netzwerkes befinden. Da dieses Wort von unterschiedlichen Autoren für unterschiedliche Geräte verwendet wird, ist eine eindeutige Definition nicht möglich. Varghese et al. (2016: 1) meinen mit Edge Devices Geräte, wie zum Beispiel Smartphones, Kameras, MP3-Player oder Spielkonsolen. Hingegen werden Begriffe wie Edge Router, Edge Computing Device oder Edge Host von Samaniego/Deters (2016: 116–119) mit Edge Device mitunter austauschbar verwendet. Mit dieser Bedeutung wird auch in dieser Arbeit bearbeitet, da im Internet der Dinge ein Edge Device als Verbindungsstück zwischen Cloud und IoT-Gerät dient und somit bei Verwendung passender Software auch verschiedene weitere Funktionen erfüllen kann. Insbesondere in Bezug auf Edge Computing erreichen Edge Devices Verbesserungen der Latenz und eine Verringerung der Datenübertragungsmenge dadurch, dass Rechnerressourcen näher an die IoT-Geräte gebracht werden (vgl. Villari et al. 2016: 82). Ein zentrales Problem bei der Verwendung von Edge Devices bleibt die sichere Übertragung und Installation von Code auf diese (vgl. Samaniego/Deters 2016: 116–117).

Quelle: (Villari et al., 2016, p. 77)

Abbildung 2 - "The Edge" im IoT

Ein typisches Beispiel für ein Edge Device ist der Edison-Arduino von Intel, der von Samaniego/Deters (2016: 117) ausgewertet wurde.

Edge Computing

Edge Computing bezeichnet eine Technologie, die es ermöglicht, Rechenlast von der Cloud auf den Netzwerkrand zu verschieben. Im Falle einer Cloudanwendung bedeutet Netzwerkrand die Computerressource, die als Verbindungsstelle zwischen Cloudserver und IoT-Gerät (einem End Device) dient. Mit Edge Computing wird nun, anstatt alle Daten von der Quelle zur Cloud zu übertragen, ein Teil der Berechnungen schon auf den Computerressourcen am Netzwerkrand ausgeführt. Dazu gehören Tätigkeiten wie die Verarbeitung und Lagerung von Daten, Caching und das Ausgleichen von Rechnerlasten (vgl. Shi/Dustdar 2016: 80). Unter anderem bietet dies die Vorteile, dass weniger Daten zur Cloud übertragen werden, die Latenzzeit und unter Umständen der Energieverbrauch verringert wird und die Sicherheit verbessert wird. Verwandte Begriffe sind zum Beispiel „fog computing“ (vgl. Shi et al. 2016: 638).

3 Softwarelösungen für Container-basierte Virtualisierung

3.1 Die Welt der Container – Überblick zum Vokabular

Programme in isolierten Umgebungen auszuführen, ist keine neue Erfindung. Spätestens in den 1970er Jahren aufgekommen, schaffte die Container-basierte oder OS-level Virtualisierung den großen Sprung Richtung Mainstream mit dem Aufkommen von Docker im Jahr 2013 (vgl. Osnat 2018). Seitdem sind die stetigen neuen Entwicklungen und Angebote auf diesem Gebiet kaum mehr zu überblicken. Insbesondere die unterschiedliche Funktionalität der einzelnen Angebote ist schwer zu durchschauen, da für verschiedene Objekte oft dieselben Begriffe verwendet werden.

Daher sollen zum Zwecke von Verständlichkeit und Eindeutigkeit für diese Arbeit einige Begriffe vorab definiert werden. Scott McCarty liefert dazu eine Übersicht in Form eines technischen Wörterbuches, das auf Red Hat Developers veröffentlicht wurde. In diesem Artikel bezieht McCarty sich in erster Linie auf Linux Container, die in der Tat den Großteil des Container-Marktes ausmachen und daher dazu geeignet sind als Definition herzuhalten (vgl. McCarty 2018).

Der Inhalt der folgenden Absätze ist McCarty (2018) entnommen, falls nicht anders gekennzeichnet.

Container:

Container gibt es in unterschiedlicher Form schon lange Zeit. In der hier verwendeten Art sind sie die Instanziierung der entsprechenden Abbilder. Im häufig verwendeten Linux-System sind Container wie ein standardmäßiger Linux Prozess, der häufig mit weiteren Schutzmaßnahmen isoliert und abgeschirmt wird. „Containerisierte Prozesse“ können aus diesem Grund mit Containern in ihrer Bedeutung gleichgesetzt werden. Diese wurden schon im Kapitel 2.1.4 LXC Container-Funktionen als Grundlage beschrieben. Auch andere manuell erstellte, isolierte Umgebungen werden mitunter als Container bezeichnet. Allerdings sind derart konfigurierte Umgebungen für eine industrielle Anwendung ungeeignet, da sie nicht automatisierbar sind, und werden daher nicht näher beleuchtet.

Container Image: (z.B. LXD-image, OCI-based-image)

Ein Container Image, Container-Abbild oder Abbild ist im einfachsten Fall eine Datei, die von einem Container Registry Server bezogen wird und die als Grundlage für die Erstellung eines Containers dient. Technisch kann dieses Abbild sehr unterschiedliche Formen annehmen. Allen Abbildern gleich ist jedoch, dass die jeweilige Ausformung einer Container Engine aus dem Abbild und den dazugehörigen Metadaten einen laufenden Container erstellt.

Container Image Format: (z.B. LXD-, rkt-, Docker-, OCI-format)

Das Container Image Format beschreibt den Aufbau eines Container Image und stellt damit den Rahmen auf, wie ein solches Abbild aussehen kann, welche Funktionen es unterstützt und wie andere Tools von Drittparteien darauf zugreifen können. Ursprünglich hatte jeder Container-Anbieter sein eigenes Container Image Format bis mit der Open Container Initiative ein standardisiertes Image Format definiert wurde und damit die plattformübergreifende Kollaboration verbesserte.

Container Engine: (z.B. LXD, LXC, rkt, Docker, Cri-O, …)

Eine Container Engine ist eine Software, die verschiedene Aufgaben übernimmt. Unter anderem nimmt die Container Engine Benutzereingaben an, zieht Abbilder vom Server, hält die notwendigen Informationen für die Inbetriebsetzung von Containern bereit und betreibt die Container aus Sicht der Benutzer. In der Realität kommt hier allerdings eine Container Runtime ins Spiel, siehe weiter unten, die sowohl für die tatsächliche Ausführung eines Containers als auch dessen Betrieb zuständig ist. Die Container Engine ist somit ein wichtiges Zwischenstück, das verschiedene Bestandteile der Container-Welt verknüpfen muss. Dadurch ist sie aber auch ein besonders kritisches Element in Bezug auf Sicherheit, Kompatibilität und Geschwindigkeit.

Container Host: (z.B. Red Hat Atomic Host, Windows, Linux UX, Mac OSx)

Mit „Container Host" ist das System gemeint auf dem die Container oder „containerisierten" Prozesse laufen. Dieser stellt alle verfügbaren Ressourcen und sollte durch die vorgenommenen Schutzmaßnahmen vor den laufenden Containern geschützt werden. Ein solches System kann sowohl eine virtuelle Maschine, ein einfacher Personal Computer oder ein Bare-Metal-Server sein.

Registry/Registry Server:

Der Registry Server, auch Container Registry oder einfach Registry genannt, ist ein öffentlicher oder privater Datenserver für Container Images, die in diesem Fall auch als Repository (dazu weiter unten) bezeichnet werden. Der Registry Server stellt einen bedeutenden Teil eines Container-Systems dar, da er die Abbilder zur Verfügung stellt, die nicht lokal verfügbar sind. Allerdings wird vorausgesetzt, dass man dem Registry Server vertraut. In Bezug auf Sicherheit, Lizenzierung und Compliance ist daher Vorsicht im Umgang mit den Daten eines Registry Servers geboten.

Container-Orchestrierung: (z.B. Kubernetes, Docker Swarm, Mesos, OpenShift)

Sobald Container in einem Verbund für eine industrielle Anwendung ausgeliefert werden sollen, braucht es für die Bewältigung der Aufgabe eine übergeordnete Struktur. Diese wird von einer passenden Container-Orchestrierung zur Verfügung gestellt, die folgende Aufgaben übernimmt:

- Die Rechenlast der Container innerhalb des vorhandenen Computerclusters wird dynamisch aufgeteilt. Dies wird oft als Distributed Computing bezeichnet.
- Sie stellt eine standardisierte Datei bereit, die die Services definiert, aus der die eigene Anwendung aufgebaut sein wird. (z.B. kube yaml, docker compose)

Dadurch wird die Container-Orchestrierung in die Lage versetzt, zum einen die Zuteilung der vorhandenen Ressourcen für einen Cluster bis auf den einzelnen Containern zu bestimmen und zum anderen für jeden einzelnen Container im Cluster festzulegen, wie lange dieser auf welchem Host ausgeführt werden soll. Weiters können auf dieselbe Weise große Cluster von Container Hosts verwendet werden, die ebenfalls individuell gesteuert werden können. Somit können Ausfälle auf Container-, Engine- und Host-Ebene ausgeglichen werden. Außerdem vereinfacht eine Container-Orchestrierung die Installation neuer Instanzen von bestehenden Anwendungen in weiteren Umgebungen. Nachdem zwei der drei großen Anbieter, Docker Swarm und Mesos von Mesosphere, ihre Unterstützung für Kubernetes angekündigt haben, wurde Kubernetes zum De-facto-Standard im Bereich der Container-Orchestrierung, den auch alle großen Cloud Anbieter unterstützen. Andere Optionen sind insbesondere im Unternehmensbereich weiterhin vorhanden.

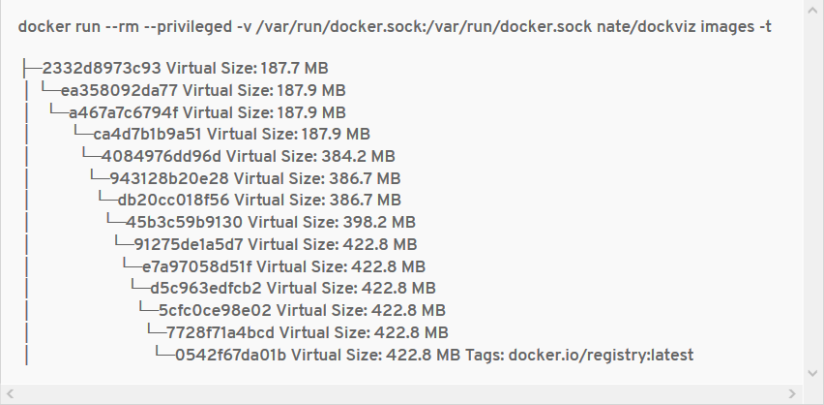
Container Runtime: (z.B. runc, crun, railcar, katacontainers)

Die Container Runtime ist üblicherweise Teil einer Container Engine auf einer tieferliegenden Ebene des Programms. Sie kann zu Testzwecken aber auch alleinstehend verwendet werden. Der von der Open Containers Initiative verwendete Runtime-Standard ist runc, der auch von vielen Container Engines voreingestellt genutzt wird. Die Aufgaben einer Container Runtime beinhalten das Einlesen der bereitgestellten Dateien (Container Image und Metadaten), die Kommunikation mit dem Kernel zum Starten des Containers und das Einstellen der gewünschten Sicherheitsfunktionen (cgroups, SELinux Richtlinien, AppArmor Regeln).

Image Layer:

Ein Container Image besteht aus einem oder mehreren Image Layern. Image Layer bezeichnen hierbei eine Sammlung von Abbildern in einem Repository, die zueinander in einer Eltern-Kind-Beziehung stehen. Wenn ein Abbild verändert wird, wird automatisch ein neues Image Layer erstellt, dass den Unterschied zwischen altem und neuem Abbild festhält und unterhalb angeordnet ist. Als Benutzer kann man jedes vorhandene Image Layer von einem Repository über seinen Universally Unique Identifier (UUID) oder sein Tag, eine manuell zugeordnete Bezeichnung, beziehen.

Mithilfe des Tools Dockviz kann die Beziehung einiger Image Layer so dargestellt werden.



Quelle: (McCarty 2018)

Abbildung 3 - Darstellung der Beziehung von Image Layern in einem Repository

In der Abbildung 3 ist die Struktur der Image Layer klar ersichtlich. Das jeweils darunter angeordnete Abbild wurde später erstellt und enthält nur die Änderungen, die gegenüber dem alten Abbild hinzugefügt wurden.

Repository:

Das Repository bezeichnet die Sammlung von Image Layern eines bestimmten Abbildes beziehungsweise Versionen von ähnlichen Images auf einem Registry Server. Wenn ein/e Benutzer/in ein Abbild herunterladen möchte, braucht er beispielsweise bei Docker nur das Repository zu benennen. Die Container Engine sucht daraufhin alle konfigurierten Server nach diesem Repository ab. Sobald er das passende Repository gefunden hat, lädt er vom Repository das spezifizierte Image Layer herunter. Wurde keine Spezifizierung getroffen, wird das „latest"-Image Layer geladen. Damit erhält der/die Benutzer/in ein Abbild, ohne sich zwingend Gedanken über die verschiedenen Image Layer machen zu müssen.

Open Container Initiative (OCI):

Die Open Container Initiative stellt eine technische Open-Source Community dar, die sich zum Ziel gesetzt hat, eine unternehmensneutrale, portable und offene Spezifikation und Runtime für die Anwendung von Containern als Trägerformat für beliebige Anwendungen zu definieren. Daraus entstehen konkret eine Container-Format- und Container-Laufzeit-Spezifikation und eine robuste eigenständige Runtime, die von allen Produkten, die den Spezifikationen entsprechen, angesteuert beziehungsweise verwendet werden können. Die Bedeutung dieser Initiative wird beim Blick auf die Liste der Mitglieder der OCI ersichtlich, die alle großen Container-affinen Unternehmen enthält: 99 Cloud, Alibaba Cloud, Amazon Web Services, Cisco, D2iQ, Dell EMC, Docker, EasyStack, Facebook, Fujitsu, Goldmann Sachs, Google, Huawei, IBM, InfoSiftr, Intel, Joyent, Microsoft, OpenStack, Oracle, Pivotal, Portworx, Red Hat, Replicated, SUSE, Sysdig, Twistlock, VMware, Weave.

3.2 Überblick der Container-Softwareangebote

Um eine funktionsfähige Anwendung einer Container-basierten Virtualisierung zu erhalten, müssen einige Tools passend miteinander verknüpft werden. Im Folgenden sollen einige Produktangebote vorgestellt werden, die entsprechend dem im vorigen Kapitel genannten Vokabular grob in jene Kategorien eingeteilt werden können: Container Runtime, Container Engine, Container-Orchestrierung.

Aus der Marktdynamik der letzten Jahre sind insbesondere im Bereich der Container Runtime und Orchestrierung mit runc und Kubernetes De-facto-Standards entstanden, die von der breiten Masse der Anbieter unterstützt oder hauptsächlich verwendet werden. Einen großen Beitrag dazu hat die Open Container Initiative (OCI) geleistet, der heute Branchengrößen wie Docker, Google, Microsoft, Amazon Web Services, Facebook oder Alibaba Cloud (vgl. The Linux Foundation 2016) angehören. Derzeit ist die Dichte an Produkten, die sich mit Container-Technologie befassen, stark angestiegen, wodurch sich eine Vielzahl an Angeboten in den verschiedenen Kategorien ergibt. Einige bedeutende Produkte werden im Folgenden überblicksmäßig erörtert. Falls Interesse an weiteren Produktvarianten besteht, wurde im Anhang unter dem Abschnitt weitere Angebote mit Container-Bezug eine Liste weiterer Produkte angeführt, deren Relevanz am Container-Markt derzeit noch vergleichsweise gering ist.

Die folgende Grafik stellt einen groben Zusammenhang der in den nächsten Abschnitten vorgestellten Auswahl an Angeboten dar (siehe Anhang für hochauflösende Darstellung).

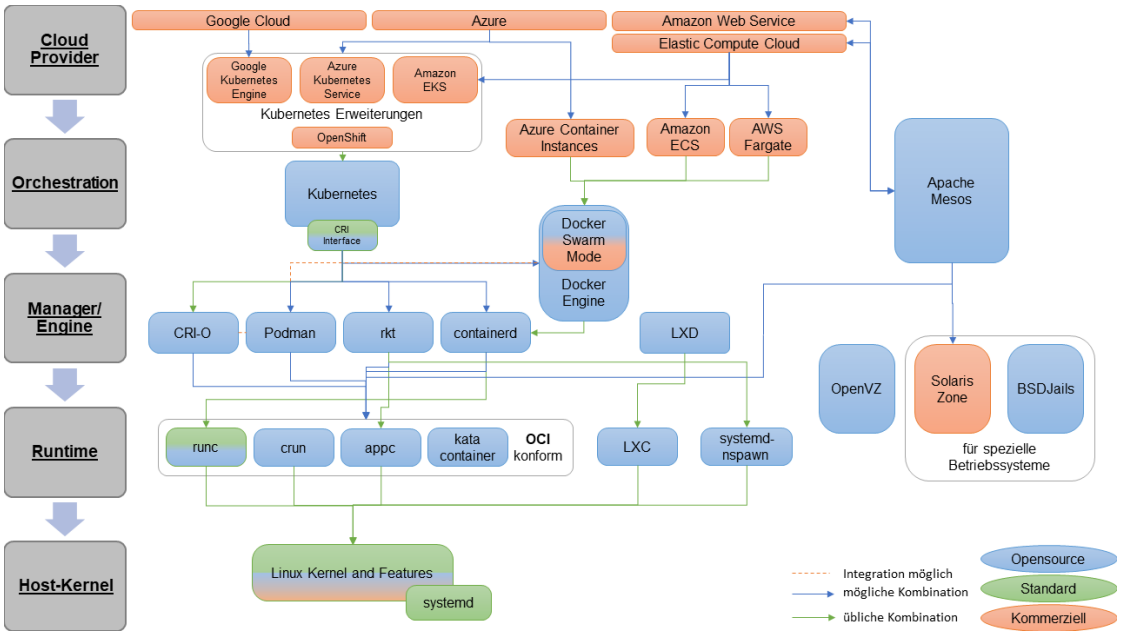


Abbildung 4 - Übersicht der vorgestellten Container-Softwareangebote

Für einen besseren Überblick ist hier ebenso die Reihenfolge dieser Angebote angeführt:

Tabelle 2 - Überblick Container-Softwareangebote

3.3 Container Runtime	.1 runc	.2 appc	.3 crun
	.4 systemd-nspawn	.5 LXC	.6 kata container
3.4 Container Engine	.1 containerd	.2 Docker	.3 rkt
	.4 LXD	.5 Cri-O	.6 Podman
	.7 OpenVZ/Virtuozzo	.8 BSD Jails	.9 Solaris Zonen
3.5 Container-Orchestrierung	.1 Kubernetes	.2 Docker Swarm	.3 Apache Mesos
	.4 OpenShift		
3.6 Container-Orchestrierung “as a Service”	.1 Amazon Web Services	.2 Azure	.3 Google Cloud

3.3 Container Runtime

Es gibt diverse Angebote, die unter dem Sammelbegriff „Container Runtime“ subsummiert werden können. Teilweise unterscheiden sich diese Produkte zwar deutlich voneinander. Gemeinsam ist jedoch allen, dass sie eine Umgebung schaffen, in der ein Container ausgeführt und isoliert werden kann. Unterschiede finden sich bei den genutzten Programmierkonzepten und in der Stärke der Isolierung sowie dem Grad der Standardisierung.

3.3.1 runC

Kurzbeschreibung runC

runC ist ein Command-Line-Interface Werkzeug, das zum Erschaffen und Betreiben von Containern verwendet wird. Bedeutend für die Verbreitung ist die Übereinstimmung mit den Spezifikationen des OCI Projektes (vgl. GitHub, Inc. 2019g).

Ursprünglich entwickelte Docker eine Codebasis, um die benötigten Linux Kernel Funktionen in einem Tool zu vereinen. Nachdem Docker diese an die OCI gespendet hatte, wurde in Kooperation mit den OCI Mitgliedern das eigenständige Tool runC entwickelt, das nun als standardisierte Container Runtime auf verschiedenen Plattformen nutzbar ist, unter anderem Linux Distributionen und Microsoft Windows (vgl. Estes 2016).

Die Container Runtime runC ist eine open source Software, die auf GitHub gehostet wird und regelmäßig Releases erhält, die nach Möglichkeit mit den (größeren) OCI Spezifikationsversi-

onen synchronisiert werden. Zu verwendende Container müssen im OCI Bundle Format vorliegen. Dieses besteht jedenfalls aus einem root-Dateisystem und einer JSON-Konfigurationsdatei (vgl. Estes 2016; GitHub, Inc. 2019g).

3.3.2 appc

Kurzbeschreibung Application Container (appc)

App Container (appc) ist eine Spezifikation für Anwendungscontainer, die von einer Community entwickelt wird. Definiert werden einige unabhängige Tools, die zusammengesetzt verwendet werden können. Neben einem Image Format sind eine Container Runtime und ein Container-Suchmechanismus enthalten. Seit 2016 wird appc jedoch nicht mehr aktiv weiterentwickelt, da die teilnehmenden Entwickler am OCI-Projekt mitarbeiten. Viele Eigenschaften der appc Spezifikationen wurden dafür in die OCI-Spezifikationen übernommen (vgl. GitHub, Inc. 2019b).

3.3.3 crun

Kurzbeschreibung crun

crun ist ebenfalls eine OCI-konforme Container Runtime, die vollständig in C geschrieben ist. Im direkten Vergleich mit runc hat crun eine schnellere Performance und eine geringere Auswirkung auf den Arbeitsspeicher. Die Grundlage bieten wiederum die Container-Funktionen von Linux. crun ist wie runc als command line Programm konzipiert ist (vgl. GitHub, Inc. 2019c).

3.3.4 systemd-nspawn

Kurzbeschreibung systemd-nspawn

systemd ist ein Programmpaket von grundlegenden Bausteinen eines Linux Systems. Zum einen bietet systemd einen System- und Service-Manager, der mit der Prozessnummer PID 1 läuft und für das Starten des restlichen Systems verantwortlich ist. Zum anderen stellt systemd auch verschiedene erweiterte Funktionen zur Verfügung (vgl. freedesktop.org LLC o. J.-a; Hesse 2019). Neben der Verwendung von cgroups zur Überwachung von Prozessen bedient sich systemd auch der beiden Tools systemd-nspawn und machinectl, die beide eine zentrale Rolle bei der Erstellung und Regulierung von Linux Containern haben (vgl. Wikimedia Foundation Inc. 2019b).

systemd-nspawn ist in systemd Paket enthalten und vergleichbar mit dem chroot Befehl von Linux. Allerdings virtualisiert systemd-nspawn, anders als chroot, eine volle Dateisystemhierarchie inklusive einem Prozessbaum und kann daher neben Befehlen auch Betriebssysteme in leichtgewichtigen Containern ausführen. Wichtiger Unterschied zu LXC-systemd ist die Einfachheit der Konfiguration (vgl. ArchWiki 2019).

3.3.5 LXC

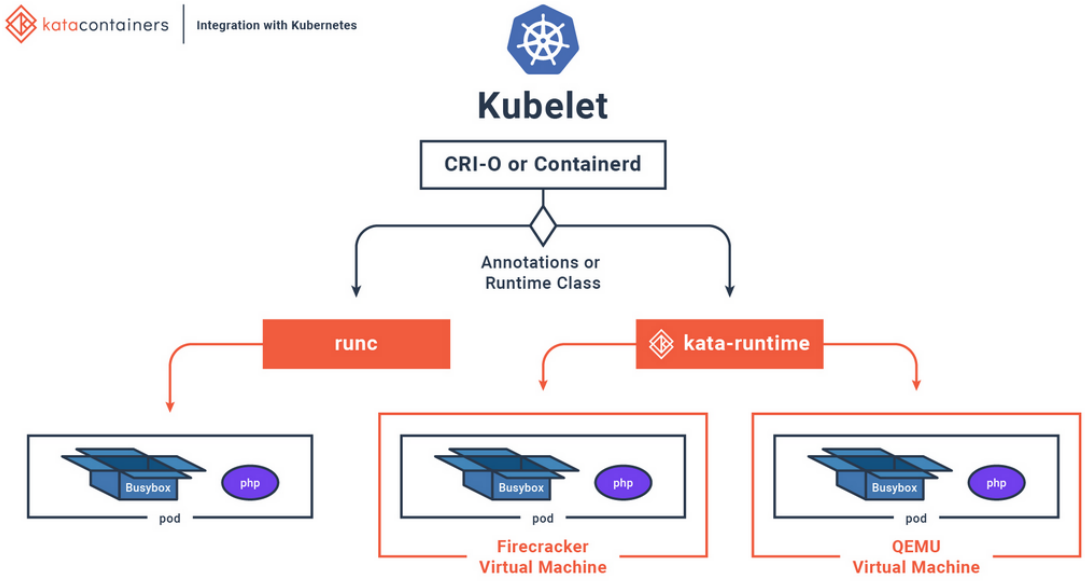
Kurzbeschreibung LXC

LXC, kurz für LinuX Container, ist eine getestete low-level Container Runtime, die seit 2008 in Entwicklung ist und schon in kritischen Produktionsumgebungen eingesetzt wurde. Der zentrale Fokus von LXC sind System-Container, die ungefähr dem entsprechen, was eine VM leistet. Ein geringerer Overhead entsteht bei diesen dadurch, dass keine separaten Kernels betrieben und die Hardwarekomponenten nicht virtualisiert werden müssen. Möglich wird ein LXC Container durch die Kernel Sicherheitsfunktionen (siehe Abschnitt Linux Container (LXC)) die der Host-Kernel unterstützen muss (vgl. Canonical Ltd. o. J.-a; Martin et al. 2018: 5).

3.3.6 Kata Container

Kurzbeschreibung Kata Container – eine microVM

Kata Container entstand aus dem Zusammenschluss der beiden Projekte Hyper.SH runV und Intel Clear Containers und wurde Ende 2017 innerhalb der OpenStackFoundation (vgl. OpenStack Foundation 2019) als open-source-Software verfügbar gemacht. Anders als die bisherigen Container Runtimes ist ein Kata Container eine virtuelle Maschine in Leichtbauweise, die ohne Probleme in die Container-Welt integriert werden kann und dabei die Vorteile beider Welten kombiniert. Neben der Schnelligkeit und Leichtigkeit von Containern (Boot-Zeit von <100ms) bieten Kata Container auch die volle Integration in die Management- und Orchestrierungsumgebung von beispielsweise Kubernetes, während die Vorteile von virtuellen Maschinen bezüglich Sicherheit und Isolation genutzt werden können. Damit geht ein gewisser Overhead einher, der allerdings, wie Graham Whaley in seiner Einführung erklärt, sehr gering ist, bei der Boot-Zeit im Vergleich zu runC fast vernachlässigbar ist und nur etwa 50MB Speicher benötigt (vgl. OpenStack Foundation 2018). Ebenfalls unterstützt werden Industriestandards wie OCI und CRI (vgl. OpenStack Foundation 2017).



Quelle: (katacontainers o. J.-a)

Abbildung 5 - Integration von Kata Container im Vergleich zu runC

Dadurch, dass Kata Container sowohl mit der OCI-Spezifikation als auch dem Container Runtime Interface kompatibel sind, können in der Abbildung 5 als „kata-runtime“ bezeichnete Kata Container nahtlos mit CRI-O, containerd oder Docker Engine (siehe folgende Abschnitte für Details) verwendet werden. Neben anderen Optionen stehen dort die Standard-Runtime runC und die kata-runtime zur Verfügung. Mit der kata-runtime kann für jeden Container oder Pod, der von der darüberliegenden Engine gestartet wurde, eine virtuelle Maschine erstellt werden, in der dieser vollständig isoliert wird. Die virtuelle Maschine wird entweder durch FireCracker, QEMU oder KVM erstellt (vgl. GitHub, Inc. 2019c).

Eine weitere Option eines microVMs ist Amazons FireCracker (vgl. katacontainers o. J.-a).

3.4 Container Engine

3.4.1 containerd

containerd ist ein für die Industrie standardisierter Container Daemon mit einem Fokus auf Einfachheit, Robustheit und Beweglichkeit. Als Daemon ist es sowohl für Linux als auch Windows erhältlich und kann den kompletten Container-Lebenszyklus für sein Gastsystem übernehmen, wobei die endgültige Ausführung eines Containers einer Runtime wie runC obliegt. Die Voraussetzungen von containerd an eine solche Runtime sind gering (vgl. GitHub, Inc. 2019e).

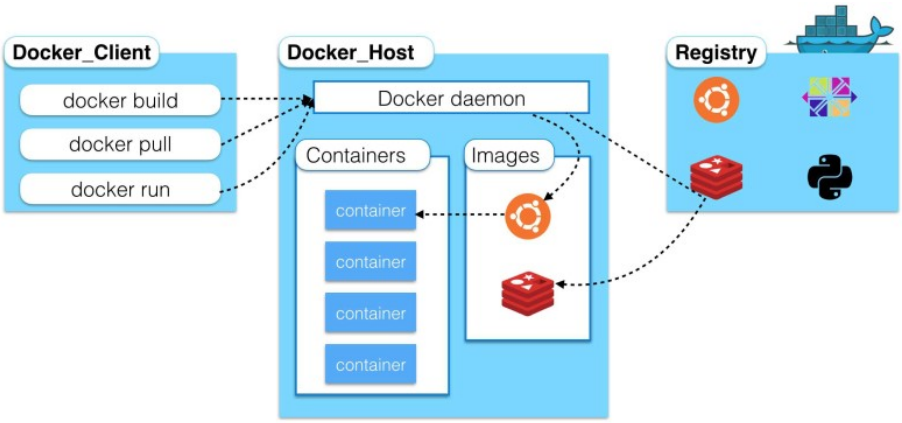
Ursprünglich wurde containerd zur Integration von OCI-kompatiblen Laufzeiten konzipiert und ist daher auch heute noch für die meisten Betriebssysteme verwendbar. Vereinfacht gesagt stellt containerd eine automatisierte Schnittstelle zwischen einer Container Engine und einer OCI-kompatiblen Runtime dar. Zu den Anwendern gehören unter anderem Kubernetes und Docker (vgl. Crosby 2017).

3.4.2 Docker

Kurzbeschreibung Docker Engine

Ein „Docker Container Image“ ist eine leichtgewichtige, unabhängig ausführbare Softwaredatei, die alles inkludiert, um eine Anwendung ablaufen zu lassen: Quellcode, Laufzeit, Datenbibliotheken, Systemeinstellungen und Systemwerkzeuge. Bei Ausführung mit der Docker Engine wird aus dem Container-Abbild in der Laufzeit ein Docker Container. Dieser besitzt zwei große Vorteile. Erstens isoliert der Container die darin liegende Anwendung von dem ihn umgebenden Betriebssystem. Zweitens läuft die Anwendung aufgrund des Containers immer gleich ab, unabhängig vom darunterliegenden Betriebssystem oder der Unterschiede zwischen Entwicklung und Auslieferung (vgl. Docker Inc. 2019).

Aufbau eines Docker Containers



(Paraiso et al. 2016: 3)

Abbildung 6 - Architektur von Docker Container-Systemen

Docker setzt sich aus mehreren Hauptkomponenten zusammen. Die Docker Engine, bestehend aus Docker Host (Mittelteil der Abbildung 6) und Docker Client (linker Teil der Abbildung 6), ist für die Abwicklung, Organisation und Kontrolle der Abbild-Dateien und der Container zuständig. Der Docker Host ist aus dem Docker Daemon und den eingesetzten Containern aufgebaut. Der Docker Daemon ist dafür zuständig, die Container zu erstellen, zu betreiben und zu kontrollieren. Außerdem erstellt und lagert er auch die Abbild-Dateien. Er selbst wird vom übergeordneten Betriebssystem gestartet. Die Aufgabe des Docker Client ist die Kommunikation mit dem Host, die Organisation der Container und die Veröffentlichung der Abbilder (vgl. Agarwal/Moreira 2014: 9; Paraiso et al. 2016: 3).

Das Docker Registry (rechter Teil der Abbildung 6) lagert die bestehenden Docker Abbilder und sorgt für deren Verteilung im Zusammenspiel mit dem Docker Client. Das Standardverzeichnis dafür ist das lokale Dateisystem mit einer Basis Konfiguration. Um weitere Abbilder herunterzuladen, wird standardmäßig der Docker Hub verwendet, der viele tausend öffentliche Abbilder online zum Download bereitstellt (vgl. Agarwal/Moreira 2014: 21; Paraiso et al. 2016: 3).

Docker verwendet grundsätzlich die vom Linux Kernel bereitgestellten Container-Funktionen zur Isolierung der Prozesse. Eng mit den Kernel Funktionen „namespaces“ und „cgroups“ verbunden ist das von Docker verwendete Tool runC. Die Kernel Funktionen wurden im Kapitel 2 im Abschnitt „Die Grundlagen – LXC Container aus Linux“ behandelt (vgl. Paraiso et al. 2016: 3).

3.4.3 rkt

Kurzbeschreibung rkt (CoreOS)

rkt (je nach Quelle „rock it“ oder „rocket“ ausgesprochen) ist eine Container Engine des Open-Source-Softwareproduzenten CoreOS, der zu RedHat gehört. Der Fokus von rkt liegt auf Sicherheit, Geschwindigkeit und einem einfachen, modularen Aufbau. Als Alternative zu Docker bietet sich rkt besonders für Anwendungen mit hoher Sicherheitsstufe und im industriellen Umfeld an. Zentrale Vorteile von rkt gegenüber dem Branchenprimus Docker sind, dass rkt als standalone Software, nämlich ohne eigenen Daemon, bei einem Crash nicht Gefahr läuft, alle erstellten Container ebenfalls zu stoppen, und gut in die Linux-eigenen Funktionen init, systemd und upstart integrierbar ist (vgl. Mocevicius 2015: 9.1).

Die kleinste Einheit in rkt ist ein „pod“. Ein „Pod“ besteht aus einem oder mehreren, miteinander verknüpften Container-Abbildern. Dies kann sofort für eine Verschachtelung von mehreren Containern verwendet werden und Einstellungen, zum Beispiel bezüglich der Isolierung, können sowohl auf der Ebene des „Pods“ als auch direkt auf der Ebene der Abbilder eingestellt werden. rkt implementiert den App Container Standard (appc), ein offenes, standardisiertes Container-Format, kann jedoch auch mit anderen Container-Formaten verwendet werden. Außerdem ist rkt auf den meisten Linux Distributionen lauffähig (vgl. Red Hat, Inc. 2019a).

Aufbau eines rkt Containers

Die Erzeugung von rkt Containern funktioniert zum Teil ähnlich wie bei Docker, mit einigen zentralen Unterschieden wie oben erläutert. Der Entstehungsprozess kann in drei Phasen unterteilt werden. In der ersten Phase wird das Dateisystem für den Container festgelegt und das gewünschte Abbild bezogen. Die Laufzeitumgebung wird im erstellten Dateisystem in Phase zwei erstellt. Hier können mithilfe von systemd-nspawn auch die entsprechenden cgroups definiert werden, wobei rkt auf möglichst großer Austauschbarkeit mit anderen Implementierungen ausgelegt ist. Nun wird der Pod mit den Containern und deren Laufzeitumgebung in Phase drei gestartet. Die Einfachheit der Erstellung ist rkts Architektur geschuldet, die die “Pods” direkt im Unix Prozess Modell ablaufen lässt (vgl. Makam 2016; Red Hat, Inc. 2019a).

3.4.4 LXD

Kurzbeschreibung LXD

LXD ist ein Container-Manager, der auf Linux Containern (LXC) basiert, jedoch wie eine virtuelle Maschine funktioniert. Die Basis bilden Abbilder, die für viele Linux Distributionen zur Verfügung stehen, und eine leistungsfähige, einfach aufgebaute Rest API. Das Projekt wurde von Canonical Ltd. gegründet und mit Unterstützung von anderen auch weiterhin von ihr geführt (vgl. Canonical Ltd. o. J.-c).

Aufbau eines LXD Containers

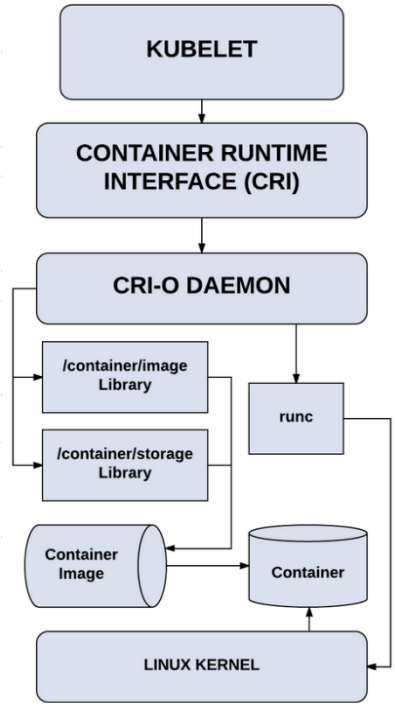
Im Zentrum von LXD steht ein Daemon mit der schon genannten Rest API, die die Kommunikation der Anwendungen in den LXC Containern nach draußen überträgt und damit volle Kontrolle darüber bietet. Die Verbindung mit einem „local host“ oder einem „remote server“ funktioniert daher ident (vgl. Canonical Ltd. o. J.-c).

Während LXC Container von LXD ausgeführt werden, übernimmt LXD organisatorische Aufgaben und speichert Informationen zur Konfiguration der Container selbst ab, weshalb innerhalb von LXD keine LXC-Befehle verwendet werden sollten, obwohl dies grundsätzlich möglich ist (vgl. Ubuntu Documentation Team o. J.). LXC hat alleinstehend aufgrund seiner Rückwärtskompatibilität einige Nachteile, die durch LXD zum Teil gelöst werden. Erstens bietet LXD eine dynamische Limitierung der Hardwareressourcen und Möglichkeiten zur effizienten Migration von Containern. Zweitens können sinnvolle Voreinstellungen ausgewählt werden, die es ermöglichen, Container standardmäßig sicher zu erstellen. Drittens macht LXD die Bedienung von Containern benutzerfreundlicher (vgl. Graber 2016).

3.4.5 CRI-O

Kurzbeschreibung CRI-O

CRI-O ist in gewisser Weise ein Spezialfall einer Container Engine (vgl. Gracey 2019). Aufbauend auf der CRI Schnittstelle von Kubernetes bildet CRI-O das Verbindungsstück zwischen dem Kubelet und einer OCI-konformen Container Runtime. Derzeit werden konkret runC und Kata Containers als Runtime unterstützt. Obwohl CRI-O als Leichtbaualternative für Docker oder rkt konzipiert ist, deckt es nicht alle Funktionen der anderen Engines ab, da weder ein eigenständiges CLI Werkzeug geplant ist, um mit CRI-O zu interagieren, noch Abbilder erstellt oder organisiert werden. Mit CRI-O ist es Kubernetes möglich, ohne ein weiteres Tool direkt Container zu starten und zu managen. Wichtige Unterstützer des auf GitHub gehostetem CNCF Projektes sind Red Hat, Intel, SUSE, Hyper und IBM (vgl. GitHub, Inc. 2019i, o. J.).



Quelle: (Brockmeier, 2017)

Abbildung 7 - Implementierung von CRI-O

Aufbau von CRI-O

Die Architektur von CRI-O besteht aus einem Daemon, der von kubelet angesprochen wird und dann über die gewählte Runtime den Container startet. Der CRI-O Daemon nutzt die „containers/image“ Bibliothek, um die Abbilder zu erhalten und lässt diese dann von runC oder einer anderen OCI-konformen Runtime ausführen. Für die Überwachung bedient sich CRI-O dem Überwachungsinstrument conmon und für die Einrichtung des Netzwerkes wird der CNI Standard gewählt (vgl. Batts 2019; Brockmeier 2017).

3.4.6 Podman

Kurzbeschreibung Podman

Podman ist ein Teil der Anwendungsbibliothek libpod, die das Container-Pod-Konzept von Kubernetes zur Anwendung bringt. Der Pod Manager, Podman, ist dafür zuständig, Pods, Container, Container-Abbilder und Container-Datenspeicher zu organisieren. Dabei fokussiert sich Podman darauf, OCI-kompatible Abbilder und viele Arten des Abbild Downloads zu unterstützen, die vollständige Organisation eines Containers (beziehungsweise Pod) Lebenszyklus zu übernehmen, für die Ressourcenzuteilung zu sorgen und die Kompatibilität mit Docker und CRI-O sicherzustellen. Bedeutender Vorteil von Podman ist die Möglichkeit, Container als normaler Benutzer, also rootless, zu erstellen. Dabei kann der Container höchstens die Berechtigungen des ausführenden Benutzers erhalten und hat damit keine Chance, root-Zugriff auf das Host-

system zu bekommen. Die entstehenden Einschränkungen sind gering, allerdings müssen davor von einem Administrator einige Konfigurationen im Hostsystem erstellt werden (vgl. Containers - Open Repository for Container Tools 2019).

Zusammensetzung von Podman

Podman baut, anders als der Konkurrent Docker, auf einem Fork-Exec-Modell auf, das jeden Container zu einem Kind-Prozess von Podman macht. Dadurch erspart sich Podman den Container Daemon, der gewisse Sicherheitsrisiken birgt. Wie der Name andeutet, verwendet Podman den von Kubernetes geprägten Begriff Pod, der eine Gruppe von nicht vollständig voneinander getrennten Containern bezeichnet. Damit können Entwickler komplexere Konstellationen von Containern realisieren (vgl. Rothberg 2019).

Eine weitere Besonderheit von Podman ist sein modularer Aufbau, indem bestimmte Funktionen in andere Anwendungen ausgelagert werden, die im Folgenden diskutiert werden.

Erweiterung durch Buildah, Skopeo, conmon und CRI-O

Der Plan für Podman ist eine breite Integration von OCI Projekten, um von möglichst vielen Standards bestmöglich zu profitieren. Daher werden viele Aufgaben, die nicht als Kernaufgabe von Podman gesehen werden, von den folgenden Anwendungen übernommen. Buildah, das ebenfalls viele Docker Befehle implementiert hat (vgl. GitHub, Inc. 2019g), ist auf das Erstellen und Hochladen von Container-Abbildern spezialisiert. Skopeo kümmert sich um die Verarbeitung von Abbildern im Backend-Bereich. Conmon dient als Werkzeug, um die OCI-Runtimes zu überwachen. Der Netzwerkbetrieb läuft über die CNI Schnittstelle, ein CNCF Projekt (vgl. GitHub, Inc. 2019f). CRI-O, das Kubernetes auch als eigenständige Container Engine dient (siehe unten), übernimmt für Podman die Kommunikation mit Kubernetes über das CRI Interface. (vgl. GitHub, Inc. 2019h)

3.4.7 OpenVZ/Virtuozzo

Kurzbeschreibung OpenVZ/Virtuozzo

OpenVZ ist eine container-basierte Form einer Virtualisierung auf Betriebssystemebene seit 2006. Mithilfe eines modifizierten Linux Kernels können durch OpenVZ mehrere isolierte Instanzen eines Betriebssystems auf einem physischen Server ausgeführt werden, die auch als Container, Private Server oder virtuelle Umgebungen bezeichnet werden. Ein solcher Container erhält sein eigenes Set an Prozessen, ein Dateisystem, Benutzer, Netzwerkschnittstellen, etc. Dadurch können viele Container nebeneinander getrennt voneinander betrieben werden, mit der Einschränkung, dass nur Linux Distributionen als Betriebssystem der Container ausgeführt werden können (vgl. Kovari/Dukan 2012: 336).

In der 2016 veröffentlichten Version wurden die beiden Virtualisierungslösungen der Firma Virtuozzo, OpenVZ (open-source) und Virtuozzo (kommerziell), auf eine Codebasis vereint (vgl. Bronnikov 2016).

Aufbau eines OpenVZ Containers

Ein OpenVZ Container baut auf einem modifizierten Kernel auf, der Virtualisierung und Isolierung von diversen Untersystemen sowie ein Ressourcenmanagement und „Checkpointing“ durchführt. Das Ressourcenmanagement stellt sicher, dass eine faire und praktikable Zuteilung der verfügbaren Hardwarekomponenten (CPU, RAM, Speicherplatz) erfolgt. Mit der „Checkpointing“-Funktion kann der Status eines zuvor gestoppten Containers abgespeichert und dann auf dem gleichen oder einem anderen Host wieder gestartet werden. Wie alle OS-level Virtualisierungen teilt sich ein OpenVZ Container das Betriebssystem und die Gerätetreiber mit dem Host und hat daher eine dünnere Virtualisierungsschicht als eine virtuelle Maschine (vgl. M. Vasconcelos et al. 2016: 321–322).

3.4.8 Solaris Zonen

Kurzbeschreibung Oracle Solaris Zonen

Oracle Solaris Zonen (Container), die erstmals mit Solaris Express 2005 erschienen sind (vgl. Drewanz/Grimmer 2012), stellen, ähnlich wie Linux Container, virtuelle Betriebssystemservices bereit, durch deren Isolierung verhindert werden soll, dass Prozesse aus einer Zone die Prozesse einer anderen Zone beeinflussen und/oder überwachen. Diese Isolation kann bei Solaris Zonen recht genau auf die eigenen Bedürfnisse eingestellt werden. Selbst als Superuser ausgeführte Prozesse können nicht außerhalb der eigenen Zone operieren, womit eine hohe Sicherheit gegeben ist. Die verwendete Abstraktionsschicht kann weiters für eine dynamische Ressourcenzuteilung verwendet werden. Dadurch können die Hardwareressourcen nach Bedarf den Containern zugeteilt und auch schlecht programmierte Anwendungen ohne Bedenken auf demselben Rechner ausgeführt werden (vgl. Oracle 2013: 217–226).

Aufbau von Solaris Zonen

Ab der Oracle Solaris-Version 10 können Solaris Zonen erstellt werden, wobei das Wort Zonen mit „nicht globalen Zonen“ gleichzusetzen ist. Durch die Zonenpartitionierungstechnologie wird neben der globalen Zone eine Anzahl an nicht globalen Zonen erstellt (bis zu einem Maximum von 8192 in der Version 10), die jeweils eine isolierte Anwendungsumgebung darstellen. Jeder Zone ist ein Zonenname, ein einmaliger numerischer Bezeichner und ein vom Zonennamen unabhängiger Knotenname zugeordnet. Außerdem erhält jede Zone ein eigenes Root-Verzeichnis, das über das Root-Verzeichnis der globalen Zone definiert ist. Auf diese Art können in einer Zone auch privilegierte Prozesse ausgeführt werden, die in der globalen Zone als Prozess zu sehen sind, allerdings keine Möglichkeit haben, in anderen Zonen außer der eigenen Änderungen zu veranlassen. In den Zonen gibt es daher naturgemäß Einschränkungen, wenn privilegierte Aktionen, wie zum Beispiel Systemaufrufe, durchgeführt werden sollen (vgl. Oracle 2013: 217–226).

3.4.9 Free BSD Jails

Kurzbeschreibung FreeBSD Jails

FreeBSD Jails sind in modernen FreeBSD Systemen zur Stärkung der Sicherheit implementiert und können als OS-level Virtualisierung angesehen werden. Dazu wird das schon bekannte chroot Konzept verwendet, um ein isoliertes root-Verzeichnis für eine Gruppe von Prozessen zu erzeugen, die dadurch nicht mehr auf Prozesse oder Dateien außerhalb des Jails zugreifen können. Da die alleinige Anwendung von chroot nicht für eine Isolierung reicht, virtualisiert der Jails Befehl außerdem den Dateisystemzugriff, die verwendbaren User und ein Netzwerkteilsystem (vgl. GitHub, Inc. 2019b; Riondato o. J.; Wikimedia Foundation Inc. 2019a).

BSD Jails gehören, ähnlich wie der Befehl chroot, zu den ersten Formen der Container-Technologie (vgl. Combe et al. 2016: 2).

Aufbau von FreeBSD Jails

Ein Jail ist aus vier Kernelementen aufgebaut: ein Unterverzeichnis (der Eintrittspunkt in das Jail und Rahmen, aus dem ein Prozess nicht mehr ausbrechen kann), ein Hostname (vom Jail genutzt), eine IP-Adresse (dem Jail zugeordnet und oft ein Alias einer bestehenden IP-Adresse) und ein Befehl (Dateipfad des im Jail auszuführenden Programms). Weiters haben Jails einen eigenen Satz User und einen eigenen root-Account, der auf das Jail begrenzt ist und ausschließlich Operationen innerhalb ausführen kann (vgl. Riondato o. J.).

3.5 Container-Orchestrierung

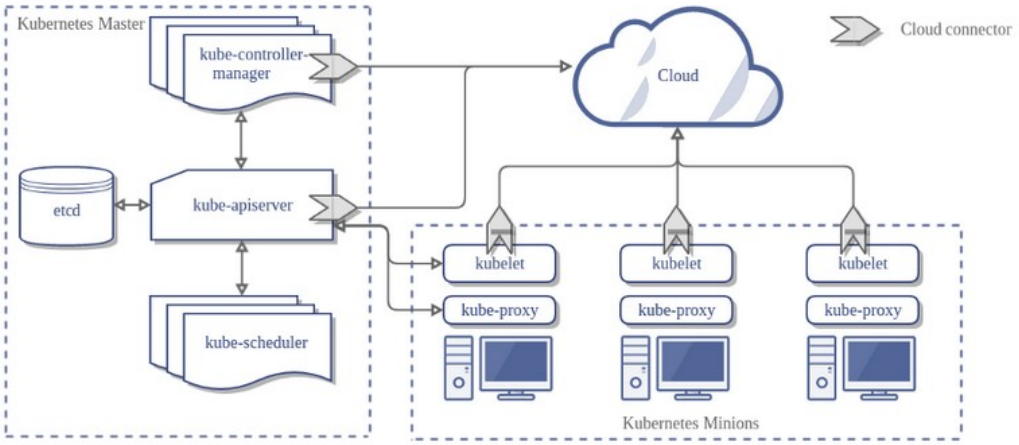
3.5.1 Kubernetes

Kurzbeschreibung Kubernetes (K8s)

Kubernetes ist ein von Google entwickeltes open-source Projekt zur Bereitstellung, Wartung und Skalierung von Programmen auf verschiedenen Hosts. Während der Entwicklung profitierte Kubernetes insbesondere von der jahrelangen Erfahrung von Google mit ähnlichen Projekten und der großen, aktiven Community. Später wurde Kubernetes an die Cloud Native Computing Foundation (CNCF) gespendet, von der es nun gehostet wird (vgl. GitHub, Inc. 2019d).

Eine herausragende Eigenschaft von Kubernetes ist die Unterstützung verschiedenster Container-Lösungen, unter anderem Docker, rkt, Cri-O und Windows Container, durch einen modularen Aufbau (vgl. Yang et al. 2019: 681). Eine bedeutende Änderung dazu wurde durch das Container Runtime Interface (CRI) herbeigeführt, das als Application Programming Interface (API) die interne Docker Integration ablöste und dadurch die Verwendung beliebiger CRI-kompatibler Container Runtimes ermöglicht (vgl. Baier et al. 2019: 85).

Aufbau eines Kubernetes Clusters



Quelle: (The Linux Foundation/The Kubernetes Authors 2019b)

Abbildung 8 - Architektur eines Kubernetes Clusters

Ein Kubernetes Cluster, der in der obigen Darstellung abgebildet ist, wird aus zwei Hauptkomponenten aufgebaut, einem oder mehreren Master-Komponenten und einer beliebigen Anzahl an Node-Komponenten, die früher Minions genannt wurden. Nodes sind Arbeitsmaschinen, die sowohl auf einer VM oder einer physischen Maschine laufen können, die notwendigen Services für das Ausführen von Pods enthalten und von einer Master-Komponente gesteuert werden (vgl. The Linux Foundation/The Kubernetes Authors 2019a). Die Master-Komponente bietet ein Kontrollzentrum für den Cluster, kann dementsprechend globale Entscheidungen für diesen durchführen und kann grundsätzlich auf jeder Maschine im Cluster laufen; einfachheitshalber werden jedoch alle Master-Komponenten auf einer Maschine ausgeführt (vgl. The Linux Foundation/The Kubernetes Authors 2019c). Weiters gibt es Pods, die auf den Nodes laufen und die gewünschten Container enthalten. Außerdem gibt es noch weitere Services, wie kubelet und Replication Controller, die für die Steuerung und Überwachung der Container beziehungsweise des Clusters und der Pods verwendet werden (vgl. Yang et al. 2019: 681).

Weitere Erläuterungen der einzelnen Komponenten sind hier zu finden (vgl. The Linux Foundation/The Kubernetes Authors 2019c).

3.5.2 Docker Swarm

Kurzbeschreibung Docker Swarm Mode

Docker Swarm, ursprünglich ein eigenständiges Produkt von Docker, ist seit Docker 1.12 ein Teil des Docker Clients mit dem Namen Docker Swarm Mode und kann direkt über die Docker CLI gesteuert werden. Die Funktionalität entspricht in weiten Teilen denen von Kubernetes. Mehrere Container können damit in einem Cluster zentral organisiert und upgedated werden. Zu den Eigenschaften von Docker Swarm Mode gehören ein dezentralisiertes Design mit mehreren Master Nodes, Möglichkeiten zum Lastausgleich, Bereitstellung eines Netzwerkes für mehrere Hosts und die Unterstützung von Docker Compose Konfigurationen. Docker Swarm

Mode eignet sich insbesondere für kleine Cluster und bietet als Vorteile einen einfachen Aufbau und gute Geschwindigkeit. Im Gegenzug können damit nur Docker Container verwaltet werden (vgl. Yang et al. 2019: 667).

Aufbau eines Docker Swarm Clusters

Ein Cluster in Docker Swarm Mode besteht aus einer beliebigen Anzahl von Nodes (Hardwareeinheiten, z.B. ein Server). Diese teilen sich in zwei Gruppen, Master oder Worker Nodes, die eng zusammenarbeiten. Dabei übernehmen Worker nur die Durchführung der benötigten Berechnungen, während Master den Swarm zusätzlich oder ausschließlich den Cluster pflegen und organisieren. Dazu beobachten sie die vorhandenen Services und Nodes, gleichen diese mit dem Soll-Zustand ab und können bei Änderungen oder Ausfällen entsprechend den gewählten Einstellungen sofort reagieren. Üblicherweise werden zur Sicherung der Funktionsfähigkeit mehrere Master-Nodes eingesetzt, um bei einem Ausfall handlungsfähig zu bleiben (vgl. Bhatia et al. 2018: 986; Gesellchen 2017).

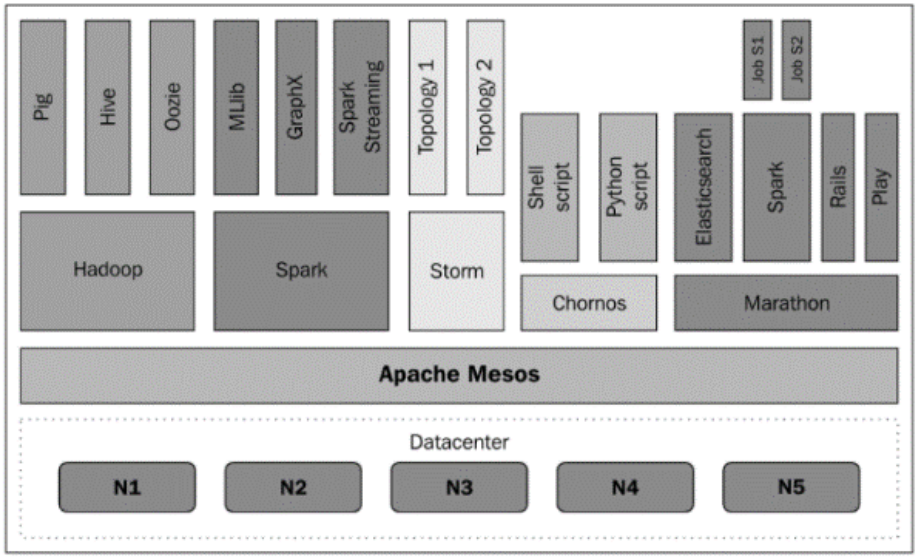
3.5.3 Apache Mesos

Kurzbeschreibung Apache Mesos

Apache Mesos ist ein Cluster-Management Werkzeug, das für eine bessere Ressourcennutzung auf verteilten Rechnerumgebungen mit unterschiedlichen Frameworks verwendet wird. Gestartet wurde es als Projekt an der University of California, Berkeley im Jahr 2009 und wurde 2013 von Apache als Top-Projekt aufgenommen. Vereinfacht kann Mesos als Kernel für ein Datencenter betrachtet werden, das einen Überblick über die Hardwareressourcen aller Nodes bietet und gleichzeitig diese Ressourcen wie ein Kernel eines Einzelrechners zuteilt. Dafür stehen unterschiedliche Funktionen zur Verfügung, inklusive der Isolierung von Ressourcen und der Strukturierung von sogenannten Slave-Nodes, eine Ablaufplanung für Programme, die Bereinigung von fehlerhaften Teilen und die Skalierung von Nodes. Zusätzlich wird Mesos durch Tools wie Apache Zookeeper und Marathon erweitert (vgl. Aggarwal 2018: 5; Kakadia 2015: 3).

Aufbau eines Mesos Clusters

Auch bei einem Mesos Cluster werden zwei Rollen unterschieden, ein Master und mehrere Slaves die von einem Framework genutzt werden. Dabei gibt es zu einer Zeit nur einen aktiven Master, der den Slaves die Arbeitspakete zuteilt. Frameworks stellen in diesem Rahmen die von Mesos ausgeführten Anwendungen dar. Durch die verschiedenen Erweiterungen bietet Mesos eine sehr große Vielfalt an möglichen Anwendungsszenarien und ist speziell für die Skalierung über große Datencenter entwickelt, wie man in Abbildung 9 erkennen kann (vgl. Kakadia 2015: 3–5).



Quelle: (Kakadia 2015: 3)

Abbildung 9 - Aufbau eines Apache Mesos Cluster

3.5.4 OpenShift (Red Hat)

Kurzbeschreibung OpenShift von Red Hat

Red Hat OpenShift ist eine Container-Plattform für die Anwendung auf einer hybriden Cloud im industriellen Bereich. Ziel von OpenShift ist es, Kubernetes und Docker bestmöglich anwendbar zu machen, sodass Entwickler und Administratoren sich auf das Entwickeln, Installieren und Ausführen von Programmen fokussieren können. OpenShift stellt damit keine eigene Container-Orchestrierungslösung dar, sondern ist ein Tool, mit dem es Unternehmen erleichtert wird, diese marktführenden Softwarelösungen für die eigenen Zwecke anzuwenden. Dazu werden von Red Hat verschiedene Varianten angeboten, die sich insbesondere am Vertriebsmodell, von open source bis kommerziell, unterscheiden (vgl. Cholewa 2019). Für den Anwender bietet OpenShift zwei Eingabemasken an. Über eine web-basierte Konsole, die mit vielen Funktionen ausgestattet ist, können alle gewünschten Aktionen mit graphischen Darstellungen ergänzt durchgeführt werden. Als zweite Option steht aber auch ein klassisches Command-line Tool bereit, von dem aus alle Operationen direkt möglich sind (vgl. Shipley/Dumpleton 2016: 3–5).

Der für die Container relevante Teil der Software von OpenShift ist deckungsgleich mit Kubernetes und Docker. Deshalb ist für den nachfolgenden Vergleich der konkrete Aufbau von OpenShift nicht relevant. Auf eine gesonderte Darstellung wird daher verzichtet.

3.6 Container-Orchestrierung „as a Service“

Bedeutung von „as a Service“

„as a Service“ bedeutet, anders als „on premise“, dass der Kunde Software nicht mehr als Produkt kauft und dieses dann auf seiner Hardware ausführt, sondern der Verkäufer die Software auf seiner Hardware bereitstellt und dem Kunden diese entsprechend dessen Bedarf den Zugriff darauf ermöglicht. Zum Beispiel kann Word entweder auf einem PC installiert und lokal ausgeführt werden oder über einen Browser aufgerufen und online bearbeitet werden. Im zweiten Fall wird Word als Service bereitgestellt. Dadurch wird allerdings nicht nur die Bereitstellung des konkreten Dienstes verändert, sondern das gesamte „Ecosystem“ muss sich entsprechend anpassen. Unter anderem können beim Kunden auf diese Weise Kosten für Hardware und Wartung verringert werden (vgl. Tyrväinen et al. 2010: 125–126).

3.6.1 Amazon Web Services (AWS)

Amazon Web Services

Amazon Web Services, kurz AWS, ist eine öffentliche, breitgefächerte Cloudcomputing-Plattform, die eine Vielzahl an webbasierten Produkten „as a Service“ anbietet, deren Bezahlung auf Basis von effektiver Benutzung beziehungsweise „on-demand“ abgerechnet wird. Meistens wird der Zugriff auf die Produkte über die „AWS Management Console“ genannte webbasierte Steuerkonsole durchgeführt. AWS ist seit 2006 eine eigene Entität und wird global angeboten mit Datencentern in den USA, Europa, Brasilien, Singapur, Japan, China und Australien. (vgl. Wadia 2016: 4–5).

Die AWS Plattform bietet viele Services mit einem bestimmten Funktionsumfang, die in drei Hauptkategorien unterteilt werden können. Basisprodukte stellen die Grundlage für alle weiteren Services von AWS zur Verfügung, wie die Datenspeicherung oder Netzwerkfunktionen. Anwendungsprodukte dienen der Ausführung von konkreten Aufgaben, die der Nutzer stellt. Die administrativen Services übernehmen schließlich alle Aufgaben, die für die Umgebung des Benutzers notwendig sind, zum Beispiel Identitäts- und Zugriffsmanagement, Werkzeuge oder Kontrollfunktionen (vgl. Wadia 2016: 7).

Drei Produkte mit Bezug zu Containern werden im Folgenden erläutert.

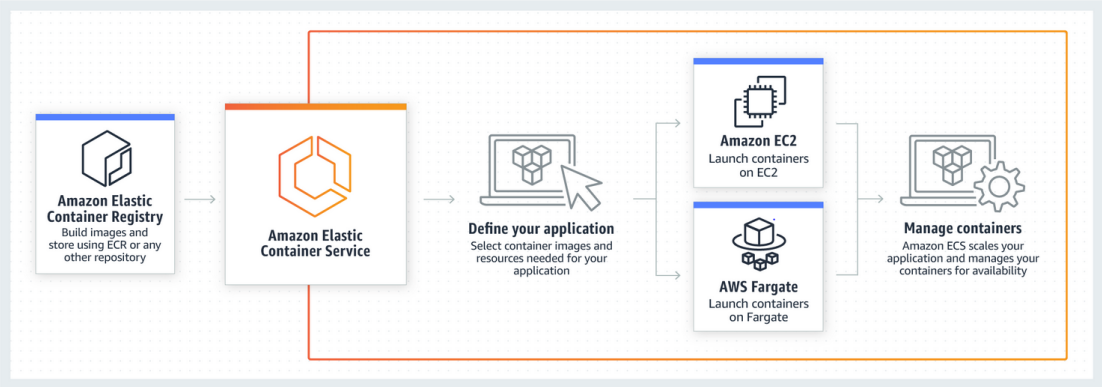
Amazon Elastic Compute Cloud (EC2)

Die Elastic Compute Cloud (EC2) ist ein Webservice, das Rechnerkapazitäten auf einer flexiblen und sicheren Basis anhand einer virtualisierten Plattform liefert. Mit EC2 können die benötigten Kapazitäten mit minimalem Einsatz konfiguriert und kontrolliert werden. Die angebotene Möglichkeit, die Rechnerkapazität entsprechend den eigenen Anforderungen sowohl vertikal (die Ressourcen des einzelnen Servers) als auch horizontal (mehr Instanzen pro Server) innerhalb von Minuten zu verändern, wird als Elastizität bezeichnet. Seit 2006 fungiert EC2 als Kernelement von AWS und gehört zu den Größen am Markt der Cloud Provider. Für die Nutzung

von Containern werden mehrere EC2 Instanzen von anderen Amazon Services verwendet (vgl. Sequeira 2019: 1; Wadia 2016: 8).

Amazon Elastic Container Service (ECS)

Elastic Container Service, kurz ECS, ist ein von AWS angebotenes Service, das als hoch performante und gut skalierbare Container-Orchestrierung fungiert und mit einigen Tools und AWS Services kombiniert werden kann. Als Container-Software werden Docker Container unterstützt. In Verbindung mit EC2 erlaubt ECS dem Kunden, seine Anwendungen auf einem Cluster aus EC2 Instanzen laufen zu lassen, wobei das Management der Infrastruktur von ECS übernommen wird. Dadurch können schon mit einfachen API Aufrufen Anwendungen in Docker Containern gestartet, gestoppt und überwacht werden. Ebenso kann der Status des Clusters abgerufen werden und es besteht Zugriff auf die von AWS bekannten Sicherheitsfunktionen. Bei Bedarf kann auch ein Steuerungsprogramm der eigenen Firma oder von einer Drittfirma verwendet werden (vgl. Menga 2018: 167–170; Sequeira 2019: 1).



Quelle: (Amazon Web Services, Inc. 2019b)

Abbildung 10 – Funktionsweise von Amazon ECS

Die obige Abbildung stellt das Zusammenspiel der verschiedenen Services von AWS übersichtlich dar. Das Amazon Elastic Container Registry (ECR) steht als verwaltetes Docker Container-Verzeichnis bereit (vgl. Sequeira 2019: 1). Das ECS ruft die benötigten Container vom ECR ab, definiert die gewünschte Anwendung und stellt dann, wie in den Abschnitten ECS und Fargate beschrieben, mithilfe von EC2 Instanzen einen voll funktionsfähigen und automatisch verwalteten Container Cluster bereit.

Amazon Elastic Container Service for Kubernetes (EKS)

Das Elastic Container Service for Kubernetes ist nach Sequeira (2019: 1) ein neu entwickeltes Service. Damit wird es dem Benutzer ermöglicht, Plug-Ins und Werkzeuge der Kubernetes Community in EKS zu verwenden. Anwendungen, die auf einer Kubernetes-Umgebung ausführbar sind, können auf diese Weise einfach in Amazon EKS migriert werden und sind jedenfalls kompatibel. Durch die Integration von Kubernetes durch EKS werden weiters Windows- und Linux Container unterstützt und somit verschiedenste Anwendungsfälle möglich gemacht. EKS führt Kubernetes automatisch mit drei Master Nodes aus, um einen „single point of failure“ zu vermeiden, erkennt und ersetzt kaputte Master automatisch und lädt für diese automatisiert

die aktuellen Versionen, Upgrades und Patches (vgl. Amazon Web Services, Inc. 2019a; Sequeira 2019: 1).

AWS Fargate

AWS Fargate ist eine weitere Art der Ausführung von Docker Containern auf AWS, die weiterhin von ECS oder EKS verwaltet und orchestriert werden. Auf diese Art wird die gesamte Infrastruktur von AWS Fargate zur Verfügung gestellt, der Kunde muss weder Server noch Cluster konfigurieren und die einzelnen EC2 Instanzen treten gegenüber dem Benutzer nicht mehr in Erscheinung. Der Anwender muss somit nur mehr die containerisierte Anwendung und die benötigten Rechnerressourcen angeben, alles andere wird von AWS Fargate automatisch bereitgestellt. Mit der EC2-Starttyp Variante ist es möglich, einige Einstellungen für die Infrastruktur, auf der die eigenen Container laufen, zu definieren und Statusdaten (CPU, Speicher und andere Ressourcen) abzurufen sowie neu festzulegen (vgl. Amazon Web Services, Inc. 2019c; Vohra 2018: 7).

3.6.2 Microsoft Azure

Container in Microsoft Azure

Azure ist, ähnlich wie AWS, eine Cloud Computing Plattform, die ihren Kunden viele Funktionen anbietet. Einen Teil davon bildet das Azure Container Service (ACS). Dieses ist eine Container-Service-Plattform, die drei unterschiedliche Container-Orchestrierungen zur Wahl stellt: Marathon mit DC/OS, Docker Swarm und Kubernetes. Alle Varianten können mit der API, der Azure CLI, oder mit dem ACS Portal gestartet und kontrolliert werden. Die beiden letzteren unterstützen weiters den Azure Ressource Manager (vgl. Awada 2018: 4).

Azure Container-Instanzen (ACI)

Container-Instanzen von Azure sind eine serverlose Variante, Container-Services in Azure zu nutzen. Im Fokus stehen dabei isolierte Container für einzelne Anwendungen, die schnell und einfach gestartet werden, wobei die darunterliegende Infrastruktur automatisch bereitgestellt wird. Dies steht im Gegensatz zum anschließend besprochenen Azure Kubernetes Service (AKS), das für selbst verwaltete Cluster von Containern gedacht ist. Neben Linux Containern unterstützt ACI auch Windows Container mit einigen Abschlägen. Interessant kann in der Anwendung insbesondere die Kombination mit AKS werden, da Leistungsspitzen mit Container-Instanzen abgefedert werden können (vgl. Microsoft 2019a, 2019c).

Azure Kubernetes Service (AKS)

Das Azure Kubernetes Service (AKS) ist der Container-Manager von Microsoft Azure, das für die einfache Erstellung eines betriebsbereiten Kubernetes Cluster genutzt werden kann. Die Infrastruktur wird durch Virtualisierung mittels Containern gebildet, die ein vereinfachtes Linux Abbild als Betriebssystem verwenden. Die Nutzerschnittstelle für alle Operationen mit dem Cluster bildet die Azure CLI. Eine Skalierung ist mit dieser über die Definition der Anzahl von

Worker Nodes möglich, wobei Azure kein automatisiertes Skalieren unterstützt (vgl. Barolli et al. 2020: 991).

3.6.3 Google Cloud

Google Cloud bietet, vergleichbar mit Amazon Web Services und Microsoft Azure, mehrere Möglichkeiten, Container als Service ausführen zu lassen. Google Kubernetes Engine wird hier als bedeutendste Variante davon näher beleuchtet. Ein solider Überblick zu weiteren Diensten findet sich unter Google Cloud (o. J.-c).

Google Kubernetes Engine (GKE)

Google Kubernetes Engine (GKE) nutzt eine herstellergebundene Version von Kubernetes, die auf der Google Compute Engine (GCE) ausgeführt wird. Bei der Erstellung eines Clusters kann der Benutzer einige Einstellungen vornehmen, um den Cluster nach seinen Wünschen zu formen. Unter anderem wird in diesem Vorgang festgelegt, wie groß der Cluster sein soll, wo sein Speicherort liegt, welches Basis Abbild für die Container und welche Kubernetes Version verwendet wird. Von GKE werden außerdem Werkzeuge zur Statuserfassung und Überwachung bereitgestellt, die während der Einrichtung aktiviert werden können. Eine automatische Skalierung anhand von Ressourcennutzung wird zwar unterstützt, allerdings ist diese nicht vergleichbar mit der regelbasierten Methode, die von Amazon ECS zur Verfügung gestellt wird (vgl. Christodoulopoulos/Petrakis 2019: 992–993).

4 Vergleich und Diskussion

In diesem Kapitel werden die in Kapitel 3 „Softwarelösungen für Container-basierte Virtualisierung“ vorgestellten Produkte einander gegenübergestellt und deren Eignung für die Verwendung in einem industriellen Umfeld diskutiert. Ebenfalls gute, jedoch nicht wissenschaftliche Übersichten zu Virtualisierung auf Betriebssystemebene sind unter Wikimedia Foundation Inc. (2019e) und 1&1 IONOS SE (2019) zu finden. Beide Webseiten stellen eine Auswahl der Softwarelösungen gegenüber und waren eine Inspiration für die Vergleichstabellen in diesem Kapitel.

Um die Tabellen übersichtlich zu gestalten, folgt die Zitierung der Tabelleninhalte folgender Überlegung. Wenn Daten aus einer Quelle stammen, die in der jeweiligen Kurzübersicht eines Produktes zitiert wird, dann wurde diese Quelle nicht nochmals zitiert. Bei Fakten, die sich auf neue Quellen berufen, wurde wie gewohnt zitiert. Einzige Ausnahme sind die mit (x) markierten Felder unter dem Vergleichskriterium „Sicherheit“ („Schutz gegen root-Zugriff“/“Dateisystem“/“Ressourcenlimitierung“), die Hassanien et al. (2020: 223) entnommen wurden.

4.1 Definition der Vergleichskriterien

Tabelle 3 - Erläuterung der Vergleichskriterien

Vergleichskriterien	Erklärung
Besonderheit, „Stichwort“	Nennung von einzigartigen Merkmalen, besonders prägenden Eigenschaften und/oder beschreibenden Schlagworten
Overhead (CPU, RAM, Speicher)	Ungefähre Beschreibung der entstehenden Mehrbelastung teilweise im Vergleich zu anderen Angeboten
Container-Format	Beantwortung, welche Art von Container eingesetzt wird (App-Container: pro Anwendung wird ein eigener Container genutzt, System-Container: im Container wird ein (reduziertes) Betriebssystem virtualisiert, ähnlich einer virtuellen Maschine, microVM: kein Container im herkömmlichen Sinn, sondern eine besonders kleine virtuelle Maschine mit Container-ähnlichen Eigenschaften)
unterstützte Plattformen	Aufzählung der Betriebssysteme, auf denen das Produkt ausgeführt oder genutzt werden kann
unterstütze Container-Abbilder	Aufzählung der Container-Abbilder, die mit dem Produkt verwendet werden können
unterstützte Container Runtime	Aufzählung der Container Runtimes, die mit dem Produkt verwendet werden können und in dieser Arbeit besprochen wurden

Vergleichskriterien		Erklärung
unterstützte Container Engine		Aufzählung der Container Engines, die mit dem Produkt verwendet werden können und in dieser Arbeit besprochen wurden
Lizenz/ Verfügbarkeit		Lizenz, unter der das Produkt verwendet werden kann/ Orte, von denen die Software bezogen werden kann
Geschrieben in		Nennung der wesentlichen Programmiersprachen
Eingabemaske		Aufzählung der Eingabearten, mit der die Software gesteuert werden kann
Standardisierung		Nennung der unterstützten Standards
Entwicklung		Der ursprüngliche Entwickler und das Erscheinungsjahr
kompatible Anbieter		Produkte, die in Kombination verwendet werden können und/ oder Schnittstellen bereitstellen
Sicherheit	Schutz gegen root-Zugriff	Fähigkeit gegen root-Benutzer zu schützen
	Dateisystem	Sicherung der verfügbaren Dateien gegen fremden Zugriff
	Ressourcen-limitierung	Verfügbarkeit von Ressourcenlimitierung und deren besonderen Eigenschaften

4.2 Container Runtime

4.2.1 Vergleichstabellen

Tabelle 4 - Vergleich Container Runtime Teil 1

	1. runc	2. appc	3. crun
Besonderheit, „Stichwort“	weit verbreitet und verwendet, einfach	keine Weiterentwicklung	schneller und leichter als runc
Container-Format	App-Container	App-Container	App-Container
unterstützte Plattformen	Linux, Windows	Linux, FreeBSD	Linux
unterstütze Container-Abbilder	OCI-konform	Application Container Image (ACI)	OCI-konform
Lizenz/ Verfügbarkeit	Apache 2.0/ GitHub	Apache 2.0/ GitHub	GNU GPL 3/ GitHub
Geschrieben in	Go	Go	C
Eingabemaske	CLI	ACI Builder (CLI)	CLI
Standardisierung	OCI	keine	OCI
Entwicklung	Docker/OCI, 2014	CoreOS, Entwicklung eingestellt 2016	GitHub Community, 2017
kompatible Anbieter	alle OCI konformen Anbieter (ua. Docker, rkt, Cri-O, containerd)	rkt	alle OCI konformen Anbieter (keine namentliche Nennung von crun)

Tabelle 5 - Vergleich Container Runtime Teil 2

	4. systemd-nspawn (siehe systemd)	5. LXC	6. katacontainer
Besonderheit, „Stichwort“	besseres „chroot“, wenige Funktionen	anwendungsgeprüft	Beste von 2 Welten
Container-Format	System-Container	System-Container	microVM
unterstützte Plattformen	Linux	Linux	Linux
unterstütze Container-Abbilder	eigene & teilweise OCI-Verzeichnisse	eigene	OCI-konform
Lizenz/ Verfügbarkeit	GNU LGPL 2.1/ GitHub	GNU LGPLv2.1+/ GitHub	Apache 2.0/ OpenStack, GitHub
Geschrieben in	C/C++	C	Go, Shell, C
Eingabemaske	Linux CLI	Linux CLI	Linux CLI
Standardisierung	keine	keine	OCI, CRI
Entwicklung	mit systemd, 2010	GitHub Community, 2008	Community (OSF), 2017
kompatible Anbieter	LXC, LXD	LXD, systemd	Docker, Kubernetes, containerd, Cri-O, runc, AWS

4.2.2 Diskussion

Unter den Container Runtimes hat sich schon im Juli 2017 (vgl. The Linux Foundation 2017) durch die Bemühungen des Open Container Initiative Projektes ein Standard herausgebildet. Heute ist fast jedes andere Produkt mit runc kompatibel, viele Anbieter haben runc in ihrem Produkt voreingestellt und in vielen Anwendungsfällen wird runc schon jetzt auch im industriellen Umfeld eingesetzt. appc, das CoreOS als Konkurrent entwickelte, wurde schließlich 2016 ebenfalls dem OCI Projekt hinzugefügt und floss in die Entwicklung von runc ein (vgl. GitHub, Inc. 2019e). Das ambitionierte Community Projekt crun dient dagegen noch heute als Alternative zu runc. crun kann es in Hinblick auf Beliebtheit und Verbreitung zwar noch lange nicht mit dem Branchenstandard aufnehmen und findet dementsprechend auch in der Literatur noch wenig Beachtung, allerdings zeigen sich die Entwickler ob der guten Performance (um etwa 45% schnellere Berechnungen in den Tests der Entwickler) optimistisch (vgl. GitHub, Inc. 2019f).

Nicht exakt in diese Kategorie fällt systemd-nspawn aus dem Linux-Funktionsset systemd. systemd-nspawn ist aufgrund seiner direkten Implementierung in systemd besonders Ressourcen-

schonend, kann einfach über die Kommandozeile bedient werden und kann im Gegensatz zum artverwandten Befehl chroot ein vollständiges Linux-System in einem Container starten. Allerdings fehlen durch den sparsamen Aufbau Werkzeuge zur Automatisierung, womit der Einsatz in einem betrieblichen Umfeld im großen Stil kaum von Interesse ist. Mit dem Tool machinectl kann zwar ein Teil der Steuerbefehle abstrahiert werden (vgl. freedesktop.org LLC o. J.-b), allerdings ist dies nicht vergleichbar mit Angeboten wie zum Beispiel Kubernetes (vgl. freedesktop.org LLC o. J.-c).

Ähnlich verhält es sich mit LXC. Obwohl LXC sowohl als eigenständiges Werkzeug als auch in Kombination mit LXD durchaus in Produktionsszenarien Verwendung findet, kann es nicht mit der Flexibilität und den Möglichkeiten mithalten, die eine OCI-Konformität mit sich bringt. Zu den Stärken gehören ein robuster, einfacher Code und der bereits getestete Einsatz in realen Produktionsumgebungen. Umgekehrt unterstützt LXC den OCI Standard nicht und kann daher mit beliebten Tools wie Kubernetes nicht kombiniert werden (vgl. Canonical Ltd. o. J.-a; GitHub, Inc. 2019m).

Auch wenn katacontainer im striktesten Sinne keine Container darstellen, erhält der Benutzer doch beinahe gleiche Eigenschaften, wenn er die OCI-konformen microVMs als Container-Ersatz einsetzt. Intel und Hyper.sh versuchten mit katacontainer das Beste aus diesen beiden Welten (virtuelle Maschinen und Container) zu erhalten. Da katacontainer eine relativ neue Entwicklung sind (Version 1.0 wurde im Mai 2018 gestartet), müssen sie sich erst in der Produktion beweisen, die bisherigen Tests zeigen jedenfalls, dass die Geschwindigkeit mit heutigen Container Runtimes vergleichbar ist und der Speicherbedarf ist mit rund 50 MB pro microVM ebenfalls vertretbar. Der große Vorteil wird jedoch die Verbindung aus hoher Isolation einer virtuellen Maschine mit der guten Integration von Management-Werkzeugen wie Docker und Kubernetes sein (vgl. GitHub, Inc. 2019c; katacontainers o. J.-b; OpenStack Foundation 2017).

4.3 Container Engine

4.3.1 Vergleichstabellen

Tabelle 6 - Vergleich Container Engine Teil 1

		1. containerd	2. Docker	3. rkt
Besonderheit, „Stichwort“		einfach, robust	Industrienorm	sicher, schnell, modular, wenig intuitiv
Container-Format		App-Container	App-Container	App-Container
unterstützte Plattformen		Linux, Windows	Linux, Windows	Linux
unterstütze Container Runtime		OCI-konform (runc)	OCI-konform (containerd)	OCI-konform (appc), Docker-kompatibel
Lizenz/ Verfügbarkeit		Apache 2.0/ GitHub (CNCF)	Apache 2.0/ GitHub, Docker Inc.	Apache 2.0/ GitHub (CNCF)
Geschrieben in		Go	Go	Go
Eingabemaske		containerd-Client, API	Docker Client	CLI
Standardisierung		OCI	OCI	OCI, CRI, CNI
Entwicklung		Docker, 2014	Docker, 2013	CoreOS, 2014
kompatible Anbieter		Docker, Kubernetes	containerd, Azure, AWS, Kubernetes	Kubernetes
Sicherheit	Schutz gegen root-Zugriff	keine Quellen	Ja (x)	Ja (x)
	Isolation Dateisystem	Teilweise	Ja (x)	Ja (x)
	Ressourcen-limitierung	keine Quellen	Ja (x)	Teilweise (x)

(x) … siehe Erläuterungen am Anfang des Kapitels

Tabelle 7 - Vergleich Container Engine Teil 2

		4. LXD	5. CRI-O	6. Podman
Besonderheit, „Stichwort“		verbessertes LXC	Link von Kubelet zu OCI Runtime	kompatibel, rootless, ohne Daemon, modular
Container-Format		System-Container	App-Container	App-Container
unterstützte Plattformen		Linux	Linux	Linux, remote client (Windows, MacOS)
unterstütze Container Runtime		LXC	OCI-konform	OCI-konform, Docker Image
Lizenz/ Verfügbarkeit		Apache 2.0/ GitHub	Apache 2.0/ GitHub	Apache 2.0/ GitHub
Geschrieben in		Go	Go, Shell	Go, Shell
Eingabemaske		REST API	keine	CLI, remote client, Podman API
Standardisierung		keine	OCI, CRI, CNI	OCI, CRI, CNI
Entwicklung		Canonical LTD., 2015	CNCF, 2017 (GitHub, Inc. 2019a)	Project Atomic, 2017 (Walsh 2018)
kompatible Anbieter		LXC, OpenStack	Kubernetes, runc, Docker Image, OpenShift	Docker, Kubernetes, runc, Buildah, Skopeo, Conmon, Cri-O
Sicherheit	Schutz gegen root-Zugriff	Ja (x)	Ja, da kein direkter Eingriff möglich, OCI-spezifiziert	Ja, da rootless möglich
	Isolation Dateisystem	Ja (x)	ausgelagert an containers/storage	ausgelagert an containers/storage
	Ressourcen-limitierung	Teilweise (x)	Teilweise, CRI-spezifiziert	Ja

(x) … siehe Erläuterungen am Anfang des Kapitels

Tabelle 8 - Vergleich Container Engine Teil 3

		7. OpenVZ/Virtuozzo	8. Solaris Zonen	9. Free BSD Jails
Besonderheit, „Stichwort“		Container frühzeitig übernommen	Isolierung	eine der ersten Container-Formen
Container-Format		System-Container	System-Container	keine Quellen
unterstützte Plattformen		Linux	Oracle, Sun, Fujitsu	FreeBSD
unterstütze Container Runtime		keine	keine	keine
Lizenz/ Verfügbarkeit		GNU GPL 2.0/ Downloadpage (OpenVZ) (Virtuozzo International GmbH o. J.)	Lizenzvertrag/ Downloadpage (Oracle)	Copyright 1992-2019 The FreeBSD Project/ als Teil von FreeBSD (GitHub, Inc. 2019b)
Geschrieben in		C	keine Quellen	C, C++
Eingabemaske		CLI	Zonenkonsole, CLI	Shell
Standardisierung		keine	keine	keine
Entwicklung		Virtuozzo, 2005 (Wikimedia Foundation Inc. 2018)	Oracle/illumos, 2005	Poul-Henning Kamp, 1999 (Wikimedia Foundation Inc. 2019c)
kompatible Anbieter		Virtuozzo, OpenStack (Lingayat et al. 2018: 1102)	Apache Mesos	verschiedene BSD Systeme
Sicherheit	Schutz gegen root-Zugriff	Ja (x)	Ja (x)	Ja (x)
	Isolation Dateisystem	Ja (x)	Ja (x)	Ja (x)
	Ressourcen-limitierung	Ja (x)	Teilweise (x)	Ja (x)

(x) … siehe Erläuterungen am Anfang des Kapitels

4.3.2 Diskussion

Im Bereich der Container Engines hat sich in den letzten Jahren eine bunte Ansammlung an Mitbewerbern gebildet, die sich mit verschiedenen Angeboten um Marktanteile bemühen. Die vorgestellten Produkte können in drei Kategorien eingeteilt werden. Cri-O und containerd sind einfache Varianten, die insbesondere für eine schnelle Automatisierung mit einer Orchestrierung eingesetzt werden. Docker, rkt, LXD, Podman und OpenVZ/Virtuozzo bieten etwas mehr Funktionen und können einige Einstellungen selbst vornehmen. Besonders für kleine Anwendungen, die keine größeren Cluster benötigen, sind sie von Vorteil, können mit Ausnahme von OpenVZ/Virtuozzo aber genauso mit den bekannten Container-Orchestrierungen kombiniert werden. In die dritte Kategorie fallen Solaris Zonen und BSD Jails. Diese zwei Angebote sind für bestimmte Betriebssysteme konzipiert und fallen damit aus dem Wettbewerb heraus, weil sie mit der „neuen“ Container-Welt nicht gut kompatibel sind.

Einige Eigenschaften haben fast alle Produkte gemeinsam. Bis auf Solaris Zonen sind die vorgestellten Softwarelösungen mit einer freien Lizenz versehen und zumindest auch auf GitHub erhältlich. Weiters bieten alle Varianten ähnliche Sicherheitsfunktionen, die bei manchen zwar erst in den letzten Jahren nachgebessert wurden, heute jedoch mit kleinen Unterschieden im Detail ähnlich guten Schutz bieten. In dieser Hinsicht stechen nur containerd und Cri-O hervor, da bei beiden Services aufgrund ihrer starken Integration in andere Systeme weniger Details bekannt sind.

Bezüglich der benötigten Ressourcen ähneln sich die vorgestellten Container Engines (vgl. Morabito et al. 2018: 105). Lingayat et al. (2018: 1096–1098) stellen Unterschiede zwischen den Systemen fest, die allerdings im gleichzeitigen Vergleich mit virtuellen Maschinen gering ausfallen. Außerdem kann sich kein Produkt in allen Kategorien behaupten, sondern ist eher für CPU-Leistung oder Lese-/Schreibgeschwindigkeit besser geeignet. Lingayat et al. kommen ebenfalls zu dem Schluss, dass alle Container-Lösungen im Vergleich zur direkten Ausführung auf dem Rechner einen gewissen Overhead erzeugen, der in bestimmten Fällen jedoch vernachlässigbar klein sein kann.

Klare Unterschiede gibt es hingegen, wie oben erwähnt, bei den unterstützten Betriebssystemen und bei der unterstützten Drittsoftware. Insbesondere die Kombination mit Container-Orchestrierungen ist für eine industrielle Anwendung von Interesse. Da LXD, OpenVZ/Virtuozzo, Solaris Zonen und BSD Jails keine Standardisierung aus dem Container-Ecosystem nutzen, sind die Kombinationsmöglichkeiten für diese vier Produkte stark eingeschränkt. LXD, Virtuozzo und Solaris Zonen haben zwar mit OpenStack und Apache Mesos Optionen hinsichtlich Clustermanagern, allerdings ist die fehlende Kompatibilität mit Kubernetes wegen dessen Marktbeherrschung ein Nachteil. Umgekehrt profitieren die fünf verbleibenden Lösungen, containerd, Docker, rkt, Cri-O und Podman von dem guten Zusammenspiel über die branchenweiten Standards OCI, CRI und CNI, wodurch gleichzeitig auch eine direkte Konkurrenz entsteht.

Diese fünf Kubernetes-kompatiblen Container Engines sollten keineswegs getrennt betrachtet werden, da sie meist ineinandergreifen. So baut Docker auf containerd auf, Cri-O wird von Podman für die Kommunikation mit CRI verwendet und alle fünf können neben OCI-Images auch mit Docker Images umgehen.

Für eine Nutzung im industriellen Bereich kann daher zusammenfassend gesagt werden, dass für die Wahl der Container-Software in erste Linie das verwendete Betriebssystem ausschlaggebend ist. Danach kommt es auf die Größe des Projektes an.

Sobald Drittsoftware in Form von Container-Orchestrierung genutzt werden soll, ist die Wahl jedoch oft von dieser abhängig, da die Unterschiede zwischen den Container Engines/Managern meist vernachlässigbar klein sind. Die letztendliche Entscheidung ist schließlich oft nur mehr dem Umfang nicht benötigter Zusatzfunktionen oder der Beliebtheit der Softwarelösung geschuldet.

Soll hingegen die Container-Anwendung vorerst zum Beispiel für kleine Softwaretests eingeführt werden, ist man mit Docker, rkt und LXD gut beraten, die neben einem funktionsreichen Softwarepaket auch eine gute Community-Unterstützung bieten. Unterschiede gibt es bei der Bedienung und der Geschwindigkeit. Docker hat bei der Bedienung die Nase vorn, wohingegen alle drei mit einer guten Geschwindigkeit aufwarten und im Vergleich zu den herkömmlichen virtuellen Maschinen eine deutlich Verbesserung bieten (vgl. Xie et al. 2017: 2140).

4.4 Container-Orchestrierung

4.4.1 Vergleichstabelle

Tabelle 9 - Vergleich Container-Orchestrierungen

	Kubernetes	Docker Swarm	Mesos	OpenShift
Besonderheit, „Stichwort“	De-facto-Standard	einfacher Aufbau, schnell	viele Erweiterungen	Kubernetes Erweiterung
Container-Format	verschieden	App-Container	verschieden	verschieden
unterstützte Plattformen	Plattform unabhängig	Linux, MacOS, Windows	Linux, MacOS	RHEL, Fedora Linux
unterstütze Container Engine	CRI-konform	Docker (integriert)	Docker, rkt, appc	CRI-konform
Lizenz/ Verfügbarkeit	Apache 2.0/ CNCF	Apache 2.0/ Docker Inc.	Apache 2.0/ Apache,GitHub	Apache 2.0 &kommerziell
Geschrieben in	Go	Go	C++	Go, AngularJS
Eingabemaske	WebUI, kubectl	Docker CLI	REST, Web UI, CLI	CLI
Standardisierung	CRI	keine	keine	CRI
Entwicklung	Google, 2014	Docker, 2013	UC Berkeley, 2009	Red Hat, 2011
kompatible Anbieter	Docker, Cri-O, rkt, OpenShift, Azure, AWS, Windows Container	nur in Docker Engine integriert	Docker, rkt, appc, Solaris Zonen, AWS	Kubernetes

4.4.2 Diskussion

Bei genauerem Hinsehen fällt auf, dass unter den Container-Orchestrierungen nur zwei echte Rivalen zu finden sind. Von diesen hat sich in den vergangenen Jahren mit Kubernetes ein De-facto-Standard entwickelt, der von den meisten anderen Anbietern unterstützt wird und weit verbreitet ist. Der Durchbruch gelang sicherlich auch aufgrund der CRI API-Schnittstelle, die es ermöglichte, Kubernetes auch mit anderen CRI-kompatiblen Container Engine/Runtimes zu verwenden; bis dahin war Kubernetes auf Docker zugeschnitten gewesen (vgl. Baier et al. 2019: 85–87). Docker Swarm, als einziger direkter Konkurrent zu Kubernetes, ist zwar direkt in

Docker integriert und einfach zu bedienen, allerdings bietet es dem Benutzer weniger Möglichkeiten und ist nicht so ausgereift wie der Branchenprimus. Die exklusive Unterstützung von Docker ist ebenfalls kein Vorteil von Docker Swarm, sondern schränkt die Flexibilität stark ein.

OpenShift ist hingegen nur eine Erweiterung von Kubernetes und fügt diesem einige nette Funktionen hinzu. In erster Linie vereinfacht OpenShift jedoch die Anwendung von Kubernetes, fügt praktische Automatisierungen hinzu und gilt als sicherheitsorientierte Kubernetes-Plattform (vgl. Red Hat, Inc. o. J.-b).

Das ursprünglich von der University of California entwickelte Apache Mesos hat schließlich ein anderes Leistungsspektrum. Zwei Eigenschaften stechen bei Apache Mesos besonders hervor. Erstens ist Mesos stark modular aufgebaut. Wird also eine bestimmte Funktion benötigt, kann diese üblicherweise über ein eigenes Tool verfügbar gemacht werden. Zweitens kann Apache Mesos wegen seiner Modularität auf verschiedenste Arten eingesetzt werden. Dadurch kann die Software exakt auf den Bedarf eines Unternehmens zugeschnitten werden (vgl. Kakadia 2015: 4–5). Gleichzeitig ist diese Eigenschaft auch ein Nachteil im Vergleich zu Kubernetes. Da so viele Optionen möglich sind, müssen diese bei der Konfiguration auch ausgewählt beziehungsweise eingestellt werden. Mesos gilt zwar als einfach bedienbar, im Gegensatz dazu ist ein Kubernetes Cluster jedoch mit weniger Befehlen startklar.

Für die industrielle Anwendung kommt es auch bei den Orchestrierungen auf die konkreten Bedürfnisse des Benutzers an. Soll eine IT-Landschaft entsprechend der eigenen Bedürfnisse erstellt werden, bietet Apache Mesos größtmögliche Flexibilität und diverse Optionen hinsichtlich verschiedener Softwarelösungen. Ist das Hauptaugenmerk hingegen die Wartung, kann Kubernetes mit einer potenziellen Erweiterung durch Red Hat OpenShift mit einer übersichtlichen Bedienung des eigenen Container Clusters punkten. Docker Swarm bietet insbesondere für kleine Cluster-Konstellationen alle benötigten Funktionen, die bei steigenden Anforderungen allerdings zu wenig ausgereift sein könnten.

4.5 Container-Orchestrierung “as a Service”

4.5.1 Vergleichstabelle

	AWS	Azure	Google Cloud
Besonderheit, „Stichwort“	Marktführer, sehr großes Angebot, vielfältige Variationen	besonders für Unternehmenskunden und geringen Aufwand	direkte Erweiterung von Kubernetes auf Cloud Ressourcen
Container-Format	verschiedene Arten	verschiedene Arten	verschiedene Arten
unterstütze Container-Abbilder	Kubernetes (Linux) - kompatible, Docker & Windows Container	Kubernetes (Linux) - kompatible, Docker & Windows Container	Kubernetes (Linux, Docker) - kompatible Container
Lizenz/ Verfügbarkeit	zwei Lizenzmodelle / „pay-as-you-go“ (on-demand, 1 Jahr, 3 Jahre) (Otava 2019: 11)	verschiedene Lizenzmodelle / „pay-as-you-go“ (on-demand, 1 Jahr, 3 Jahre) (Otava 2019: 11)	keine Quellen / Bezahlung on-demand, 1 Jahr, 3 Jahre (Google Cloud o. J.-b)
Geschrieben in	Java, JS, C++ (Kaur 2018)	C++, C#, C (Kaur 2018)	keine Quellen
Eingabemaske	WebUI, API	Azure CLI, Azure Portal (Microsoft 2019b)	GCP Console (Google Cloud o. J.-a)
Standardisierung	keine	keine	CRI
Entwicklung	Amazon, 2006	Microsoft, 2016 (Microsoft 2016)	Google, 2015 (Google Cloud o. J.-a)
kompatible Anbieter	Docker, Kubernetes	Kubernetes, Red Hat OpenShift, Docker und Windows Container,	Kubernetes, Docker

4.5.2 Diskussion

Im Gegensatz zu fast allen anderen hier diskutierten Softwarelösungen sind die drei Varianten der Container-Orchestrierungen „as a Service“ darauf ausgelegt, Umsätze zu generieren. Dabei bieten die drei größten Anbieter des Bereiches Cloud Infrastruktur (vgl. Synergy Research Group 2019) ähnliche Services für die Ausführung von Containern an. Unterschiede dazu finden sich erst in den Details.

AWS punktet mit einer besonders großen Palette an Services, hat aufgrund seines Marktanteiles einen sicheren Stand mit vielen Kunden und bietet verlässliche Qualität und Leistung. Microsoft Azure überzeugt mit der Integration seiner anderen Softwareangebote, die besonders für bestehende Microsoftkunden interessant ist, und bietet weiters eine solide Lösung für hybride Cloudansätze an. Google Cloud ist durch seinen deutlich kleineren Marktanteil zwar der „Underdog“, besticht allerdings mit der technischen Expertise in diesem Bereich sowie in den Feldern künstliche Intelligenz, maschinelles Lernen und Datenanalyse. Außerdem liegt Google Cloud im Kostenvergleich mit einer sekundengenauen Abrechnung und automatischen Rabatten vor den beiden Konkurrenten (vgl. Harvey/Patrizio 2019).

Auch bei der Wahl eines Cloud Anbieters für die eigenen Container-Services kann kein generelles Richtig und Falsch genannt werden. Abhängig von den eigenen Bedürfnissen kann sogar pro Projekt variieren, welcher Provider besser passt. Harvey/Patrizio (2019) fassen dies in ihrem Artikel übersichtlich so zusammen: AWS hat mit der großen Vielfalt an Services für jeden Kunden etwas im Angebot, jedoch fehlt aufgrund der schieren Größe meist die persönliche Beziehung mit dem einzelnen Kunden. Azure bietet besonders bestehenden Microsoft Kunden die beste Integration von Microsoftprogrammen und legt einen Fokus darauf, dass das bestehende Datencenter mit der Azure Cloud kombiniert tadellos als hybride Cloud bereitsteht. Google Cloud wird zwar noch laufend verbessert und kann nicht auf bestehenden Unternehmenskunden aufbauen. Auf der anderen Seite investiert Google stark in seine Cloud mit einem Fokus auf die Kernkompetenzen, Skalierung und maschinelles Lernen. Für webbasierte Startups könnte Google Cloud damit besonders interessant sein.

4.6 Bedeutung von Containern auf Edge Devices

4.6.1 Diskussion

Beim Auswerten der Literatur zu den verschiedenen Container-Softwarelösungen war es auffällig, dass die Verwendung auf Edge Devices kaum Erwähnung fand und offensichtlich Edge Computing bisher ein Thema der Zukunft ist, heute hingegen meistens Cloud Computing besonders erwähnt und beworben wird.

Dabei gibt es auch schon heute konkrete Anwendungen von Containern auf Edge Devices. Ha et al. (2017) stellen die Verwendung von Docker Containern in einer Smart Factory vor und gehen dabei darauf ein, wie ein Fabriksoperator durch besseres DevOps unterstützt wird. Ein anderes Beispiel ist die Nutzung von Containern, um Vitaldaten von Patienten im medizinischen Bereich mittels Edge Devices zu überwachen. Jaiswal et al. (2018) beschreiben, wie Docker Container auf RaspberryPi-Edge Devices ausgeführt und dadurch Latenz und Bandbreitennutzung verringert werden. Außerdem kann ein Edge Device auch weitere Funktionen übernehmen, wie zum Beispiel Säuberung, Lagerung und Analyse der aufgenommenen Daten oder Einrichtung eines Notfallwarnsystems durch Auswertung der Echtzeitdaten.

Neben Containern kommen jedoch auch andere LVs (lightweight virtualization) zum Betreiben von Edge Devices in Frage. Virtuelle Maschinen verbrauchen vergleichsweise große Mengen an Ressourcen und sind daher schlechter für die Verwendung auf Edge Devices geeignet. Allerdings wurden schon microVMs, wie katacontainer, entwickelt, die den Unterschied stark verringern. Container bekommen jedoch auch Konkurrenz durch sogenannte Unikernel, die mit noch weniger Ressourcen auskommen und ebenso geeignet sind, Programme auf Edge Devices auszuführen. Einen genauen Vergleich dieser beiden Technologien, Container und Unikernel, haben Morabito et al. (2018) durchgeführt.

Auch unter den Orchestrierungslösungen werden auf Edge Computing zugeschnittene Produkte entwickelt, die in Zukunft bedeutend werden könnten. Eines von diesen ist PiCasso, eine Plattform, die Services am Netzwerkrand möglichst automatisiert bereitstellt (vgl. Lertsinsrubtavee et al. 2017).

5 Fazit

Der Durchbruch im Bereich der Container-Technologie ist in den letzten Jahren gelungen und die Anzahl der entwickelten Container-Produkte steigt stetig. In dieser Situation erscheint es sinnvoll, einen Überblick zu verschaffen, das vorhandene Wissen zusammenzufassen und neues Potential zu erkennen. Die vorliegende Arbeit gibt dazu einige Antworten.

Für den interessierten Leser bietet die vorliegende Bachelorarbeit einen Einstieg in die Welt der Container und definiert vorab einige notwendige Fachbegriffe. Danach wird eine Auswahl der bisherigen Container-Softwarelösungen entsprechend der vorherrschenden Literatur vorgestellt und durch die Einteilung in wenige Kategorien übersichtlich eingeordnet. Diese Einteilung wird anschließend dazu genutzt, Softwarelösungen mit nicht deckungsgleichen Anwendungsbereichen in einem direkten Vergleich darzustellen und damit Unterschiede herauszuarbeiten. Schließlich tritt der Fokus auf Szenarien der industriellen Anwendung in den Vergleichen aus dem Hintergrund in den Vordergrund, wenn festgestellt wird, welche Software-Angebote sich für welche Nutzungsarten eignen. Schließlich wird die derzeitige Verwendung von Containern auf Edge Devices kurz beleuchtet. Allerdings müssen sich Container in diesem Bereich gegen die Konkurrenz von Unikernel Software behaupten.

Hingegen kann die Arbeit weder Leistungsanalysen zu den Container-Angeboten noch Reihungen der Container Engines nach Geschwindigkeit oder Speicherbedarf zur Verfügung stellen. Dies ergibt sich zum einen aus der Aufgabenstellung, nur eine Literaturrecherche durchzuführen, und ist zum anderen dem Mangel diesbezüglicher Ausarbeitungen in der Literatur geschuldet.

Anhand dieser Arbeit kann ein Start in die Welt der Container gelingen, die zurzeit stark an Relevanz gewinnt. Die Jahreszahlen der fachspezifischen Literatur sind ein Zeichen dafür, dass dieser Themenbereich erhebliche Dynamik hat und die Zukunft viele Veränderungen bringen kann. Aufbauend auf die dargelegten Erkenntnisse könnte in nachfolgenden Arbeiten behandelt werden, welche Leistungsdaten die vorgestellten Container-Technologien besitzen, welche der neuen Technologien in den kommenden Jahren die Produktreife erreichen werden oder welche Voraussetzungen Container erfüllen müssen, um für die Nutzung im Edge Computing einsetzbar zu sein.

Mit seiner gut verständlichen Einleitung in das Thema Container-Technologie gewährt diese Arbeit einen kurzen Einblick in die wichtigsten Container-Produkte und legt damit eine Grundlage für darauf aufbauende Forschungstätigkeiten.

6 Verzeichnisse

6.1 Literaturverzeichnis

1&1 IONOS SE (2019): Docker-Alternativen im Überblick. Text online abrufbar unter: https://www.ionos.at/digitalguide/server/knowhow/docker-alternativen-im-ueberblick/ (Zugriff am 26.9.2019).

Abraham, Ajith/Dutta, Paramartha/Mandal, Jyotsna Kumar/Bhattacharya, Abhishek/Dutta, Soumi (Hrsg.) (2019): Emerging Technologies in Data Mining and Information Security: Proceedings of IEMIS 2018, Volume 3, Bd. 814. Singapore: Springer Singapore.

Agarwal, Nitin/Moreira, Belmiro (2014): Docker on OpenStack. CERNopenlab.

Aggarwal, Manuj (2018): Learn Apache Mesos: A Beginner's Guide to Scalable Cluster Management and Deployment. Packt Publishing Ltd. Text online abrufbar unter: https://books.google.at/books?hl=de&lr=&id=N-l1DwAAQBAJ&oi=fnd&pg=PP1&dq=mesos+apache+marathon&ots=UTcLhLQUv8&sig=4QyzPrNhpese3mfgwG7Baq26lAI&redir_esc=y#v=onepage&q&f=false.

Amazon Web Services, Inc. (2019a): Amazon EKS. Text online abrufbar unter: https://aws.amazon.com/de/eks/ (Zugriff am 15.10.2019).

Amazon Web Services, Inc. (2019b): AWS | Amazon EC2 Container & Konfigurationsmanagement. Text online abrufbar unter: https://aws.amazon.com/de/ecs/ (Zugriff am 7.10.2019).

Amazon Web Services, Inc. (2019c): AWS Fargate. Text online abrufbar unter: https://aws.amazon.com/de/fargate/ (Zugriff am 15.10.2019).

Anand, Narendra/Chintalapally, Anuraag/Puri, Colin/Tung, Teresa (2017): Practical Edge Analytics: Architectural Approach and Use Cases. Präsentiert auf: 2017 IEEE International Conference on Edge Computing (EDGE), Juni 2017, *2017 IEEE International Conference on Edge Computing (EDGE)*, 236–239.

ArchWiki (2019): systemd-nspawn. Text online abrufbar unter: https://wiki.archlinux.org/index.php/Systemd-nspawn (Zugriff am 29.9.2019).

Awada, Uchechukwu (2018): Application-Container Orchestration Tools and Platform-as-a-Service Clouds: A Survey. In: *International Journal of Advanced Computer Science and Applications*,.

Baier, Jonathan/Sayfan, Gigi/White, Jesse (2019): The Complete Kubernetes Guide: Become an Expert in Container Management with the Power of Kubernetes. Packt Publishing Ltd. Google-Books-ID: 0JmZDwAAQBAJ.

Barolli, Leonard/Takizawa, Makoto/Xhafa, Fatos/Enokido, Tomoya (Hrsg.) (2020): Advanced Information Networking and Applications: Proceedings of the 33rd International Conference on Advanced Information Networking and Applications (AINA-2019), Bd. 926. Cham: Springer International Publishing.

Batts, Vincent (2019): Red Hat Contributes CRI-O to the Cloud Native Computing Foundation. Text online abrufbar unter: https://www.redhat.com/en/blog/red-hat-contributes-cri-o-cloud-native-computing-foundation (Zugriff am 4.10.2019).

Bernstein, D. (2014): Containers and Cloud: From LXC to Docker to Kubernetes. In: *IEEE Cloud Computing*, 1 (3), 81–84.

Bhatia, G./Choudhary, A./Dadheech, K. (2018): Behavioral Analysis of Docker Swarm Under DoS/ DDoS Attack. Präsentiert auf: 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), April 2018, *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, 985–991.

Brause, Rüdiger (2017): Betriebssysteme: Grundlagen und Konzepte. Springer-Verlag. Text abrufbar unter: https://books.google.at/books?hl=de&lr=&id=2WqYD-gAAQBAJ&oi=fnd&pg=PR19&dq=vollst%C3%A4ndiges+betriebssystem&ots=5l3SVGqMS3&sig=GYKGcppll776E2D3PjFdF5FIkjk&redir_esc=y#v=onepage&q=%22vollst%C3%A4ndiges%20betriebssystem%22&f=false. Google-Books-ID: 2WqYDgAAQBAJ.

Brockmeier, Joe (2017): Introducing CRI-O 1.0. Text online abrufbar unter: https://www.redhat.com/en/blog/introducing-cri-o-10 (Zugriff am 4.10.2019).

Bronnikov, Sergey (2016): OpenVZ 7.0 Released. Text online abrufbar unter: https://openvz.livejournal.com/ (Zugriff am 5.10.2019).

Canonical Ltd. (o. J.-a): Linux Containers - LXC - Introduction. Text online abrufbar unter: https://linuxcontainers.org/lxc/ (Zugriff am 10.9.2019).

Canonical Ltd. (o. J.-b): Linux Containers - LXC - Security. Text online abrufbar unter: https://linuxcontainers.org/lxc/security/ (Zugriff am 10.9.2019).

Canonical Ltd. (o. J.-c): Linux containers: LXD - Introduction. Text online abrufbar unter: https://linuxcontainers.org/lxd/introduction/ (Zugriff am 30.9.2019).

Chen, Jiann-Liang/Pang, Ai-Chun/Deng, Der-Jiunn/Lin, Chun-Cheng (Hrsg.) (2019): Wireless Internet: 11th EAI International Conference, WiCON 2018, Taipei, Taiwan, October 15-16, 2018, Proceedings, Bd. 264. Cham: Springer International Publishing.

Cholewa, Tomasz (2019): 10 Most Important Differences between OpenShift and Kubernetes. Text online abrufbar unter: https://cloudowski.com/articles/10-differences-between-openshift-and-kubernetes/ (Zugriff am 6.10.2019).

Christodoulopoulos, Christos/Petrakis, Euripides (2019): Advanced Information Networking and Applications - Article: Commodore: Fail Safe Container Scheduling in Kubernetes. New York, NY: Springer Berlin Heidelberg.

Combe, Theo/Martin, Antony/Di Pietro, Roberto (2016): To Docker or Not to Docker: A Security Perspective. In: *IEEE Cloud Computing*, 3 (5), 54–62.

Containers - Open Repository for Container Tools (2019): Podman.Io/Whatis. Text online abrufbar unter: https://podman.io/whatis.html (Zugriff am 4.10.2019).

Crosby, Michael (2017): What Is Containerd?. Text online abrufbar unter: https://www.docker.com/blog/what-is-containerd-runtime/ (Zugriff am 2.10.2019).

Cziva, Richard/Pezaros, Dimitrios P. (2017): Container Network Functions: Bringing NFV to the Network Edge. In: *IEEE Communications Magazine*, 55 (6), 24–31.

Docker Inc. (2019): What Is a Container?. Text online abrufbar unter: https://www.docker.com/resources/what-container (Zugriff am 9.9.2019).

Drewanz, Detlef/Grimmer, Lenz (2012): The Role of Oracle Solaris Zones and Linux Containers in a Virtualization Strategy. Text online abrufbar unter: https://www.oracle.com/technical-resources/articles/it-infrastructure/admin-zones-containers-virtualization.html (Zugriff am 3.10.2019).

Edge, Jake (2015): A seccomp overview [LWN.net]. Text online abrufbar unter: https://lwn.net/Articles/656307/ (Zugriff am 12.9.2019).

Estes, Phil (2016): RunC: The Little Container Engine That Could. Opensource.com. Text online abrufbar unter: https://opensource.com/life/16/8/runc-little-container-engine-could (Zugriff am 23.9.2019).

freedesktop.org LLC (o. J.-a): freedesktop.org: Systemd System and Service Manager. Text online abrufbar unter: https://www.freedesktop.org/wiki/Software/systemd/ (Zugriff am 29.9.2019).

freedesktop.org LLC (o. J.-b): machinectl. Text online abrufbar unter: https://www.freedesktop.org/software/systemd/man/machinectl.html# (Zugriff am 28.10.2019).

freedesktop.org LLC (o. J.-c): systemd-nspawn - Spawn a command or OS in a light-weight container. Text online abrufbar unter: https://www.freedesktop.org/software/systemd/man/systemd-nspawn.html (Zugriff am 28.10.2019).

Gesellchen, Tobias (2017): So funktioniert Docker Swarm: Eine Einführung in Continuous Deployment. Text online abrufbar unter: https://jaxenter.de/docker-swarm-einfuehrung-65263 (Zugriff am 6.10.2019).

GitHub, Inc. (2019a): Cri-o/Cri-o. Text online abrufbar unter: https://github.com/cri-o/cri-o (Zugriff am 30.10.2019).

GitHub, Inc. (2019b): freebsd/freebsd. Text online abrufbar unter: https://github.com/freebsd/freebsd/blob/master/COPYRIGHT (Zugriff am 27.10.2019).

GitHub, Inc. (2019c): Kata-Containers/Documentation. Text online abrufbar unter: https://github.com/kata-containers/documentation/blob/master/design/architecture.md (Zugriff am 1.10.2019).

GitHub, Inc. (2019d): Opencontainers/Runc. Text online abrufbar unter: https://github.com/opencontainers/runc (Zugriff am 23.9.2019).

GitHub, Inc. (2019e): appc/spec. Text online abrufbar unter: https://github.com/appc/spec (Zugriff am 25.9.2019).

GitHub, Inc. (2019f): containers/crun. Text online abrufbar unter: https://github.com/containers/crun (Zugriff am 26.9.2019).

GitHub, Inc. (2019g): kubernetes/kubernetes. Text online abrufbar unter: https://github.com/kubernetes/kubernetes (Zugriff am 27.9.2019).

GitHub, Inc. (2019h): containerd/containerd. Text online abrufbar unter: https://github.com/containerd/containerd (Zugriff am 2.10.2019).

GitHub, Inc. (2019i): containernetworking/cni. Text online abrufbar unter: https://github.com/containernetworking/cni (Zugriff am 4.10.2019).

GitHub, Inc. (2019j): containers/buildah. Text online abrufbar unter: https://github.com/containers/buildah (Zugriff am 4.10.2019).

GitHub, Inc. (2019k): containers/libpod. Text online abrufbar unter: https://github.com/containers/libpod (Zugriff am 4.10.2019).

GitHub, Inc. (2019l): cri-o/cri-o. Text online abrufbar unter: https://github.com/cri-o/cri-o (Zugriff am 4.10.2019).

GitHub, Inc. (2019m): lxc/lxc. Text online abrufbar unter: https://github.com/lxc/lxc (Zugriff am 28.10.2019).

GitHub, Inc. (o. J.): cri-o - Lightweight Container Runtime for Kubernetes. Text online abrufbar unter: https://cri-o.io/ (Zugriff am 4.10.2019).

Google Cloud (o. J.-a): Google Kubernetes Engine | Kubernetes Engine. Text online abrufbar unter: https://cloud.google.com/kubernetes-engine/?hl=de (Zugriff am 30.10.2019).

Google Cloud (o. J.-b): Alle Preise | Compute Engine-Dokumentation. Text online abrufbar unter: https://cloud.google.com/compute/all-pricing?hl=de (Zugriff am 30.10.2019).

Google Cloud (o. J.-c): Optimale Lösung für die Containerausführung auswählen. Text online abrufbar unter: https://cloud.google.com/container-options/?hl=de (Zugriff am 1.11.2019).

Graber, Stéphane (2014): Stéphane Graber's Website - LXC 1.0: Security Features [6/10]. Text online abrufbar unter: https://stgraber.org/2014/01/01/lxc-1-0-security-features/ (Zugriff am 10.9.2019).

Graber, Stéphane (2016): LXD 2.0: Introduction to LXD. Text online abrufbar unter: http://ubuntu.com/blog/lxd-2-0-introduction-to-lxd (Zugriff am 30.9.2019).

Gracey, Andrew (2019): Technical Deep-Dive of Container Runtimes. Text online abrufbar unter: https://www.suse.com/c/technical-deep-dive-of-container-runtimes/ (Zugriff am 4.10.2019).

Ha, Jihun/Kim, Jungyong/Park, Heewon/Lee, Jaehong/Jo, Hyuna/Kim, Heejung/Jang, Jaeheon (2017): A web-based service deployment method to edge devices in smart factory exploiting Docker. Präsentiert auf: 2017 International Conference on Information and Communication Technology Convergence (ICTC), Oktober 2017, *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, 708–710.

Haque, A./Ayyar, A./Singh, S. (2018): Chroot Hardening Through System Calls Modification. Präsentiert auf: 2018 8th International Conference on Cloud Computing, Data Science Engineering (Confluence), Januar 2018, *2018 8th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, 491–496.

Harvey, Cynthia/Patrizio, Andy (2019): AWS vs. Azure vs. Google: Cloud Comparison [2019 Update]. Text online abrufbar unter: https://www.datamation.com/cloud-computing/aws-vs-azure-vs-google-cloud-comparison.html (Zugriff am 1.11.2019).

Hassanien, Aboul Ella/Bhatnagar, Roheet/Khalifa, Nour Eldeen M./Taha, Mohamed Hamed N. (Hrsg.) (2020): Toward Social Internet of Things (SIoT): Enabling Technologies, Architectures and Applications: Emerging Technologies for Connected and Smart Social Objects, Bd. 846. Cham: Springer International Publishing.

Hesse, Christian (2019): Arch Linux - systemd 243.78-1 (x86_64). Text online abrufbar unter: https://www.archlinux.org/packages/core/x86_64/systemd/ (Zugriff am 25.10.2019).

IBM Deutschland (o. J.): IBM – Cloud Kubernetes Service – Details. Text online abrufbar unter: https://www.ibm.com/de-de/cloud/container-service/details (Zugriff am 16.11.2019).

Jaiswal, Kavita/Sobhanayak, Srichandan/Turuk, Ashok Kumar/Bibhudatta, Sahoo L/Mohanta, Bhabendu Kumar/Jena, Debasish (2018): An IoT-Cloud Based Smart Healthcare Monitoring System Using Container Based Virtual Environment in Edge Device. Präsentiert auf: 2018 International Conference on Emerging Trends and Innovations In Engineering And Technological Research (ICETIETR), Juli 2018, *2018 International Conference on Emerging Trends and Innovations In Engineering And Technological Research (ICETIETR)*, 1–7.

Kakadia, Dharmesh (2015): Apache Mesos Essentials. Packt Publishing Ltd. Text online abrufbar unter: https://books.google.at/books?hl=de&lr=&id=fPwKCgAAQBAJ&oi=fnd&pg=PP1&dq=mesos+apache&ots=lc20Cb6ow5&sig=Pbge9lRQpyr8S_TffoYb1z0omzU&redir_esc=y#v=onepage&q=mesos%20apache&f=false. Google-Books-ID: fPwKCgAAQBAJ.

katacontainers (o. J.-a): Software | Kata Containers. Text online abrufbar unter: https://katacontainers.io/software/ (Zugriff am 1.10.2019).

katacontainers (o. J.-b): Learn | Kata Containers. Text online abrufbar unter: https://katacontainers.io/learn/ (Zugriff am 28.10.2019).

Kaur, Kanwarpreet (2018): What Programming Languages Are Used to Build Amazon AWS and Microsoft Azure? - Quora. Text online abrufbar unter: https://www.quora.com/What-programming-languages-are-used-to-build-Amazon-AWS-and-Microsoft-Azure (Zugriff am 29.10.2019).

Kerrisk, Michael (2002): Linux Programmer's Manual - capabilities(7). Text online abrufbar unter: http://man7.org/linux/man-pages/man7/capabilities.7.html (Zugriff am 11.9.2019).

Kerrisk, Michael/Biedermann, Eric W. (2017): Linux Programmer's Manual - Namespaces(7). Linux Programmer's Manual. Text online abrufbar unter: http://man7.org/linux/man-pages/man7/namespaces.7.html (Zugriff am 9.9.2019).

Kovari, A./Dukan, P. (2012): KVM OpenVZ virtualization based IaaS open source cloud virtualization platforms: OpenNode, Proxmox VE. Präsentiert auf: 2012 IEEE 10th Jubilee International Symposium on Intelligent Systems and Informatics, September 2012, *2012 IEEE 10th Jubilee International Symposium on Intelligent Systems and Informatics*, 335–339.

Kratzke, Nane (2018): About the Complexity to Transfer Cloud Applications at Runtime and How Container Platforms Can Contribute? Präsentiert auf: 2018, Ferguson, Donald/Muñoz, Víctor Méndez/Cardoso, Jorge/Helfert, Markus/Pahl, Claus (Hrsg.)*Cloud Computing and Service Science*, Springer International Publishing, 19–45.

Lertsinsrubtavee, Adisorn/Ali, Anwaar/Molina-Jimenez, Carlos/Sathiaseelan, Arjuna/Crowcroft, Jon (2017): PiCasso: A lightweight edge computing platform. Präsentiert auf: 2017 IEEE 6th International Conference on Cloud Networking (CloudNet), September 2017, *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*, 1–7.

Lingayat, Ashish/Badre, Ranjana R./Gupta, Anil Kumar (2018): Integration of Linux Containers in OpenStack: An Introspection. In: *Indonesian Journal of Electrical Engineering and Computer Science*, 12 (3), 1094–1105.

M. Vasconcelos, Pedro Roger/Azevedo A. Freitas, Gisele/G. Marques, Thales (2016): KVM, OpenVZ and Linux Containers: Performance Comparison of Virtualization for Web Conferencing Systems. In: *International Journal of Multimedia and Image Processing*, 6 (1/2).

Makam, Sreenivas (2016): Rkt. In: Mastering CoreOS. Packt Publishing. Text online abrufbar unter: https://proquest.tech.safaribooksonline.de/book/operating-systems-and-server-administration/virtualization/9781785288128/firstchapter (Zugriff am 16.9.2019).

Martin, John Paul/Kandasamy, A./Chandrasekaran, K. (2018): Exploring the Support for High Performance Applications in the Container Runtime Environment. In: *Human-centric Computing and Information Sciences*, 8 (1), 1.

McCarty, Scott (2018): A Practical Introduction to Container Terminology. Text online abrufbar unter: https://developers.redhat.com/blog/2018/02/22/container-terminology-practical-introduction/ (Zugriff am 17.9.2019).

Menage, Paul (2004): CGroups Documentation. cgroups. Text online abrufbar unter: https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt (Zugriff am 10.9.2019).

Menga, Justin (2018): Docker on Amazon Web Services: Build, Deploy, and Manage Your Container Applications at Scale. Packt Publishing Ltd. Google-Books-ID: 7EZsDwAAQBAJ.

Microsoft (2016): Azure Container Service Is Now Generally Available | Blog | Microsoft Azure. Text online abrufbar unter: https://azure.microsoft.com/is-is/blog/azure-container-service-is-now-generally-available/ (Zugriff am 30.10.2019). 19.04.2016.

Microsoft (2019a): Azure Container Instances | Microsoft Azure. Text online abrufbar unter: https://azure.microsoft.com/en-us/services/container-instances/ (Zugriff am 19.10.2019).

Microsoft (2019b): Azure für Container: Schnellstartanleitungen, Tutorials, API-Referenz. Text online abrufbar unter: https://docs.microsoft.com/de-de/azure/containers/ (Zugriff am 30.10.2019).

Microsoft (2019c): What Is Azure Container Instances?. Text online abrufbar unter: https://docs.microsoft.com/en-us/azure/container-instances/container-instances-overview (Zugriff am 19.10.2019).

Mocevicius, Rimantas (2015): CoreOS Essentials: Develop Effective Computing Networks to Deploy Your Applications and Servers Using CoreOS. Birmingham, England; Mumbai, [India]: Packt Publishing.

Montero, Diego/Yannuzzi, Marcelo/Shaw, Adrian/Jacquin, Ludovic/Pastor, Antonio/Serral-Gracia, Rene/Lioy, Antonio/Risso, Fulvio/Basile, Cataldo/Sassu, Roberto/et al. (2015): Virtualized security at the network edge: a user-centric approach. In: *IEEE Communications Magazine*, 53 (4), 176–186.

Morabito, Roberto/Cozzolino, Vittorio/Ding, Aaron Yi/Beijar, Nicklas/Ott, Jorg (2018): Consolidate IoT Edge Computing with Lightweight Virtualization. In: *IEEE Network*, 32 (1), 102–111.

Morishima, Atsuyuki/Rauber, Andreas/Liew, Chern Li (Hrsg.) (2016): Digital Libraries: Knowledge, Information, and Data in an Open Access Society: 18th International Conference on Asia-Pacific Digital Libraries, ICADL 2016, Tsukuba, Japan, December 7–9, 2016, Proceedings, Bd. 10075. Cham: Springer International Publishing.

Morris, James (2017): SELinux Wiki. Text online abrufbar unter: http://selinuxproject.org/page/Main_Page (Zugriff am 11.9.2019).

OpenStack Foundation (2017): Kata Containers/1pager. Text online abrufbar unter: https://katacontainers.io/collateral/kata-containers-1pager.pdf (Zugriff am 1.10.2019).

OpenStack Foundation (2018): Kata Containers An Introduction and Overview. Text online abrufbar unter: https://www.youtube.com/watch?time_continue=6&v=4gmLXyMeYWI (Zugriff am 1.10.2019).

OpenStack Foundation (2019): Kata Containers - Project Updates May 2019. Text online abrufbar unter: https://www.youtube.com/watch?v=FZr1v08Oyic (Zugriff am 1.10.2019).

Oracle (2013): System Administrationshandbuch: Oracle Solaris Container – Ressourcen Administration und Solaris Zones. Text online abrufbar unter: https://docs.oracle.com/cd/E38896_01/pdf/820-2316.pdf (Zugriff am 3.11.2019).

Osnat, Rani (2018): A Brief History of Containers: From the 1970s to 2017. Text online abrufbar unter: https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016 (Zugriff am 17.9.2019).

Otava (2019): Guide to Building and Managing a Cost-Effective Hybrid Cloud. Text online abrufbar unter: https://tinyurl.com/ueg4hv6 (Zugriff am 16.11.2019).

Paraiso, Fawaz/Challita, Stéphanie/Al-Dhuraibi, Yahya/Merle, Philippe (2016): Model-Driven Management of Docker Containers. Präsentiert auf: IEEE CLOUD 2016, Juni 2016, *9th IEEE International Conference on Cloud Computing (CLOUD)*, San Francisco, USA.

Puliafito, Antonio (2018): Systems Modeling: Methodologies and Tools. New York, NY: Springer Science+Business Media.

Qanbari, Soheil/Pezeshki, Samim/Raisi, Rozita/Mahdizadeh, Samira/Rahimzadeh, Rabee/Behinaein, Negar/Mahmoudi, Fada/Ayoubzadeh, Shiva/Fazlali, Parham/Roshani, Keyvan/et al. (2016): IoT Design Patterns: Computational Constructs to Design, Build and Engineer Edge Applications. Präsentiert auf: 2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI), April 2016, *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 277–282.

Quint, Peter-Christian/Kratzke, Nane (2018): Towards a Lightweight Multi-Cloud DSL for Elastic and Transferable Cloud-Native Applications. Text online abrufbar unter: http://arxiv.org/abs/1802.03562 (Zugriff am 15.10.2019).

Ramalho, Flávio/Neto, Augusto (2016): Virtualization at the Network Edge: A Performance Comparison.

Rathore, Muhammad Siraj/Hidell, Markus/Sjödin, Peter (2013): KVM vs. LXC: Comparing Performance and Isolation of Hardware-assisted Virtual Routers. In: *American Journal of Networks and Communications*, Vol. 2 (No. 4), 88–96.

Red Hat, Inc. (o. J.): Red Hat OpenShift erleichtert die Container-Orchestrierung. Text online abrufbar unter: https://www.redhat.com/de/technologies/cloud-computing/openshift (Zugriff am 31.10.2019).

Red Hat, Inc. (2019a): rkt - Overview. Text online abrufbar unter: https://coreos.com/rkt/ (Zugriff am 16.9.2019).

Red Hat, Inc. (2019b): THEMA: Virtualisierung. Text online abrufbar unter: https://www.redhat.com/de/topics/virtualization (Zugriff am 13.9.2019).

Riondato, Matteo (o. J.): Chapter 14. Jails. Text online abrufbar unter: https://www.freebsd.org/doc/handbook/jails.html (Zugriff am 4.10.2019).

Rothberg, Valentin (2019): Podman: Linux-Container einfach gemacht, Teil 1. Text online abrufbar unter: https://www.heise.de/developer/artikel/Podman-Linux-Container-einfach-gemacht-Teil-1-4329067.html (Zugriff am 4.10.2019).

Sabharwal, Navin/W, Bibin (2015): Hands on Docker.

Samaniego, M./Deters, R. (2016): Hosting Virtual IoT Resources on Edge-Hosts with Blockchain. Präsentiert auf: 2016 IEEE International Conference on Computer and Information Technology (CIT), Dezember 2016, *2016 IEEE International Conference on Computer and Information Technology (CIT)*, 116–119.

Schreuders, Z. Cliffe/McGill, Tanya/Payne, Christian (2011): Empowering End Users to Confine Their Own Applications: The Results of a Usability Study Comparing SELinux, AppArmor, and FBAC-LSM. In: *ACM Transactions on Information and System Security*, 14 (2), 1–28.

Sequeira, Anthony (2019): AWS Certified Solutions Architect - Associate (SAA-C01) Cert Guide. Pearson IT Certification. Google-Books-ID: XHaCDwAAQBAJ.

Shah, A. A./Piro, G./Grieco, L. A./Boggia, G. (2019): A Qualitative Cross-Comparison of Emerging Technologies for Software-Defined Systems. Präsentiert auf: 2019 Sixth International Conference on Software Defined Systems (SDS), Juni 2019, *2019 Sixth International Conference on Software Defined Systems (SDS)*, 138–145.

Shi, W./Cao, J./Zhang, Q./Li, Y./Xu, L. (2016): Edge Computing: Vision and Challenges. In: *IEEE Internet of Things Journal*, 3 (5), 637–646.

Shi, W./Dustdar, S. (2016): The Promise of Edge Computing. In: *Computer*, 49 (5), 78–81.

Shipley, Grant/Dumpleton, Graham (2016): OpenShift for Developers: A Guide for Impatient Beginners. O'Reilly Media, Inc. Text online abrufbar unter: https://books.google.at/books?id=hKvODAAAQBAJ&printsec=frontcover&hl=de&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false. Google-Books-ID: qKvODAAAQBAJ.

Singh, Amit (2004): An Introduction to Virtualization. Text online abrufbar unter: http://www.kernelthread.com/publications/virtualization/.

Synergy Research Group (2019): Cloud Service Spending Still Growing Almost 40% per Year; Half of It Won by Amazon & Microsoft | Synergy Research Group. Text online abrufbar unter: https://www.srgresearch.com/articles/cloud-service-spending-still-growing-almost-40-year-half-it-won-amazon-microsoft (Zugriff am 1.11.2019).

The Linux Foundation (2015): Open Container Initiative Charter. Text online abrufbar unter: https://www.opencontainers.org/about/governance (Zugriff am 4.9.2019).

The Linux Foundation (2016): OCI Members. Text online abrufbar unter: https://www.opencontainers.org/about/members (Zugriff am 23.9.2019).

The Linux Foundation (2017): Open Containers Initiative. Text online abrufbar unter: https://www.opencontainers.org/release-notices/v1-0-0 (Zugriff am 28.10.2019).

The Linux Foundation (o. J.): SECure COMPuting with filters. Text online abrufbar unter: https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt (Zugriff am 12.9.2019).

The Linux Foundation/The Kubernetes Authors (2019a): Concepts: Nodes. Text online abrufbar unter: https://kubernetes.io/docs/concepts/architecture/nodes/ (Zugriff am 28.9.2019).

The Linux Foundation/The Kubernetes Authors (2019b): Concepts Underlying the Cloud Controller Manager. Text online abrufbar unter: https://kubernetes.io/docs/concepts/architecture/cloud-controller/ (Zugriff am 28.9.2019).

The Linux Foundation/The Kubernetes Authors (2019c): Kubernetes Components. Text online abrufbar unter: https://kubernetes.io/docs/concepts/overview/components/ (Zugriff am 28.9.2019).

van Tilborg, Henk C.A. (Hrsg.)/Jajodia, Sushil (Hrsg.) (2011): Encyclopedia of Cryptography and Security. Boston, MA, USA: Springer Science+Business Media, LLC 2011.

Tyrväinen, Pasi/Jansen, Slinger/Cusumano, Michael A. (Hrsg.) (2010): Software business: First International Conference, ICSOB 2010, Jyväskylä, Finland, June 21-23, 2010: proceedings. Berlin ; New York: Springer-Verlag.

Ubuntu Documentation Team (o. J.): help/ubuntu/LXD. Text online abrufbar unter: https://help.ubuntu.com/lts/serverguide/lxd.html?_ga=2.239342410.873766415.1569837967-582328969.1568192971 (Zugriff am 30.9.2019).

ubuntuusers.de (2019): ubuntuusers.de/AppArmor. Text online abrufbar unter: https://wiki.ubuntuusers.de/AppArmor/ (Zugriff am 11.9.2019).

Varghese, Blesson/Wang, Nan/Barbhuiya, Sakil/Kilpatrick, Peter/Nikolopoulos, Dimitrios S. (2016): Challenges and Opportunities in Edge Computing. Präsentiert auf: 2016 IEEE International Conference on Smart Cloud (SmartCloud), November 2016, *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, 20–26.

Velichko, Ivan (2019): A Journey from Containerization to Orchestration and Beyond. Text online abrufbar unter: https://iximiuz.com/en/posts/journey-from-containerization-to-orchestration-and-beyond/?z=1 (Zugriff am 26.9.2019).

Villari, M./Fazio, M./Dustdar, S./Rana, O./Ranjan, R. (2016): Osmotic Computing: A New Paradigm for Edge/Cloud Integration. In: *IEEE Cloud Computing*, 3 (6), 76–83.

Virtuozzo International GmbH (o. J.): Openvz.Org. Text online abrufbar unter: https://openvz.org/ (Zugriff am 27.10.2019).

VMware, Inc. (2014): Virtualization Essentials. Text online abrufbar unter: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/ebook/gated-vmw-ebook-virtualization-essentials.pdf (Zugriff am 13.9.2019).

Vohra, Deepak (2018): Amazon Fargate Quick Start Guide: Learn How to Use AWS Fargate to Run Containers with Ease. Packt Publishing Ltd. Google-Books-ID: c2dmDwAAQBAJ.

Wadia, Yohan (2016): AWS Administration – The Definitive Guide. Packt Publishing Ltd. Google-Books-ID: K7FKDAAAQBAJ.

Walsh, Dan (2018): Reintroduction of Podman. Text online abrufbar unter: http://www.projectatomic.io/blog/2018/02/reintroduction-podman/ (Zugriff am 27.10.2019).

Wikimedia Foundation Inc. (2018): OpenVZ. Text online abrufbar unter: https://de.wikipedia.org/w/index.php?title=OpenVZ&oldid=182315912 (Zugriff am 27.10.2019). Page Version ID: 182315912.

Wikimedia Foundation Inc. (2019a): Linux Security Modules. Text online abrufbar unter: https://en.wikipedia.org/wiki/Linux_Security_Modules (Zugriff am 11.9.2019). Page Version ID: 885718502.

Wikimedia Foundation Inc. (2019b): AppArmor. Text online abrufbar unter: https://en.wikipedia.org/wiki/AppArmor (Zugriff am 11.9.2019).

Wikimedia Foundation Inc. (2019c): FreeBSD Jail. Text online abrufbar unter: https://en.wikipedia.org/w/index.php?title=FreeBSD_jail&oldid=919426099 (Zugriff am 27.10.2019). Page Version ID: 919426099.

Wikimedia Foundation Inc. (2019d): Systemd. Text online abrufbar unter: https://en.wikipedia.org/wiki/Systemd (Zugriff am 4.11.2019). Page Version ID: 917581598.

Wikimedia Foundation Inc. (2019e): OS-Level Virtualization. Text online abrufbar unter: https://en.wikipedia.org/w/index.php?title=OS-level_virtualization&oldid=924094505 (Zugriff am 4.11.2019). Page Version ID: 924094505.

Xie, Xiao-Lan/Wang, Peng/Wang, Qi (2017): The performance analysis of Docker and rkt based on Kubernetes. Präsentiert auf: 2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), Juli 2017, *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, 2137–2141.

Yang, Xin-She/Sherratt, Simon/Dey, Nilanjan/Joshi, Amit (Hrsg.) (2019): Third International Congress on Information and Communication Technology: ICICT 2018, London, Bd. 797. Singapore: Springer Singapore.

Yong Li/Min Chen (2015): Software-Defined Network Function Virtualization: A Survey. In: *IEEE Access*, 3, 2542–2553.

6.2 Abbildungsverzeichnis

6.3 Tabellenverzeichnis

7 Anhang

7.1 weitere Angebote mit Container-Bezug

weitere Container Runtimes

- railcar
- runv
- clearcontainers
- virtcontainers
- gVisor/runsc
- Udica
- Firecracker
- lmctfy (veraltet)
- Singularity
- SmartOS
- unik
- Nabla Containers
- VMware vSphere Integrated Containers
- Linux VServer

weitere Container Engines

- Kurma
- JetPack
- Moby
- balena
- PouchContainer
- OpenStack ZUN

weitere Container-Orchestrierungen

- machinectl
- Eliot
- Nomad
- IBM – Cloud Kubernetes Service (vgl. IBM Deutschland, o. J.)

Windows Container-Software

Da ein vollständiges Windows installiert sein muss (WinServer oder Win10), für Edge Devices nicht brauchbar

- Windows Server Container
- Hyper-V-Container
- Microsoft Drawbridge
- WinDocks
- Sandboxie
- Turbo (ehemals Spoon)
- VMware ThinApp

weitere nicht-kategorisierte Container-Software

- container/image
- container/storage

7.2 weiterführende Literatur:

(Shah et al. 2019)

- weitere Container-Technologien beschrieben und gegenübergestellt

(Kratzke 2018)

- Darstellung, wie Plattformen wie GKE, Microsoft Azure und Amazon (EC2) für die Cloud-übergreifende Anwendung von Container-Orchestrierungen verwendet werden können

(Quint/Kratzke 2018)

- Vergleich der Migration einer Container-Anwendung zwischen verschiedenen Cloudorchestrierungen

(Puliafito 2018: 221–235)

- Überblick verschiedener Container-Software

(Velichko 2019)

- überblicksmäßige Zusammenfassung vieler Softwareangebote im Bereich Container-Technologie

(Qanbari et al. 2016)

- Ratschläge, wie die Nutzung von Edge Computing funktionieren kann

(Anand et al. 2017)

- Vorstellung einer neuen Art, wie Analysetools für Anwendungen auf Edge Devices eingesetzt werden

(Morabito et al. 2018)

- Vergleich von Containern und Unikernels zur Verwendung in IoT Clouds

(Cziva/Pezaros 2017; Montero et al. 2015; Yong Li/Min Chen 2015)

- Vorstellung von NFV (Network Function Virtualization), eine weitere Virtualisierungslösung, die insbesondere die Sicherung von Netzwerkzugängen und andere Netzwerk-lastige Probleme beseitigen kann