Hochschule Worms, Fachbereich Informatik

Studiengang: Angewandte Informatik

Bachelorarbeit

Leistungsbewertung von VM-basierten Containerlösungen

Vahel Hassan

Abgabe der Arbeit: 19. August 2020

Betreut durch:

Prof. Dr. Zdravko Bozakov Hochschule Worms

Zweitgutachter/in: Prof. Dr. Herbert Thielen, Hochschule Worms

Hochschule Worms
University of Applied Sciences

Kurzfassung

Es gibt in der heutigen Zeit viele Lösungen zu Container-basierte Virtualisierung, viele Unternehmen versuchen verschiedene Containerlösungen auszuprobieren, um schneller, günstiger und flexibler seine Anwendungen darauf auszuführen. Im Gegensatz zu klassischen virtuellen Maschinen, die heute noch sehr oft in Unternehmen verwendet werden, sind Containerlösungen in den letzten Jahren sehr bekannt geworden. Einer der bekanntesten Faktoren, warum Container so populär geworden sind, ist Docker Container. Seitdem Docker Container veröffentlicht wurde, interessieren sich immer mehr Unternehmen für Containerlösungen, um leistungsfähiger zu arbeiten. Seitdem sind zwei neue Interessante Projekte entstanden, zum einen der Kata-Container und Firecracker-Containerd. Die vorliegende Bachelorarbeit möchte einen Überblick über die 3 unterschiedlichen erwähnten Containerlösungen geben und die Leistung der zwei neuen Containerlösungen mit dem Docker Container vergleichen und bewerten. Es werden mehrere Testdurchläufe durchgeführt, um die einzelnen Container-Technologien zu testen und bewerten.

Hochschule Worms
University of Applied Sciences

Abstract

There are many solutions to container-based virtualization today, many companies try different container solutions to run their applications faster, cheaper and more flexible. In contrast to classic virtual machines, which are still very often used in companies today, container solutions have become very popular in recent years. One of the best known factors why containers have become so popular is Docker Container. Since Docker Container was released, more and more companies are interested in container solutions to work more efficiently. Since then, two new interesting projects have been created, the Kata Container and Firecracker Containererd. This bachelor thesis wants to give an overview of the 3 different mentioned container solutions and to compare and evaluate the performance of the two new container solutions with the Docker Container. Several test runs are carried out to test and evaluate the different container technologies.

Danksagung

Ich möchte mich bei Prof. Dr. Zdravko Bozakov herzlich bedanken, er hat trotz der Corona Phase sehr viel Zeit investiert um mir sowohl bei der Durchführung des Praktischen Teiles als auch bei der Verfassung der Arbeit mit Ratschlägen geholfen.

Ich möchte mich an letzte Stelle für die Korrektur Lesung an meinem Bruder Gerwen und meine Freundin herzlich dafür bedanken.

Hochschule Worms
University of Applied Sciences

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Hochschule Worms
University of Applied Sciences

Programmcodeverzeichnis

Abkürzungsverzeichnis

VM	Virtual Machin
AWS	Amazone Webservices
CPU	Central Processing Unit
RAM	Random-Access Memory
App	Application
API	Application Programming Interface
CLI	Command-line reference
RESTful	Representational State Transfer full
mircoVM	Micro Virtual Machine
vCPU	Virtual Central Processing Unit
VMM	Virtual Maschin Manager
UnionFS	Union Filesystem
OCI	Open Containers Initiative
TAP	Terminal Access Point
Rootfs	Root Dateisystem
Initrd	Initial ramdisk
Cpio	Copy files to and from archives
Tmpfs	Temporary File System
PID	Process identifier
IPC	Interprocess communication
GPG	GNU Privacy Guard

1 Einleitung

Die Digitalisierung hat sich in den letzten Jahren stark verändert. Früher hat man Telefone für nur einen Zweck benutzt, um mit anderen zu kommunizieren, diese wurden durch sogenannte Smarthone und andere smart Devices ersetzt, weil sie nicht nur für ein Zweck dienen sondern für viele andere Funktionen auch, wie Fotos schießen, viele verschiedene Apps die wiederum unterschiedliche Funktionen anbieten. Diese Möglichkeiten gibt es seitdem das Internet so populär geworden ist. Die Digitalisierung hat sich in den letzten Jahren ständig weiterentwickelt und einer der großen Technologien über fast jedes Unternehmen in der jetzigen Zeit interessiert ist, sind die Virtualisierungstechniken. Immer mehr Unternehmen möchten sich in diesem Bereich weiter entwickeln. Zu einer den weitverbreiteten Containern zählt der Docker Container (Witt et al. 2017). Der Docker Container wurde 2013 veröffentlicht und hat in diesen kurzen Jahren viele Firmen aufmerksam gemacht, sie haben ähnliche Vorteile wie herkömmliche VMs aber sind effizienter, und tragbarer (vgl. Docker Inc. 2020). 4 Jahre später wurde eine weitere Containerlösung veröffentlicht, dabei handelt es sich um den Kata-Container, der zurzeit sich noch in der Gründungsphase befindet, die Entwickler haben sich hierbei stark auf den Workload-Isolations und Sicherheitsvorteile von Containern fokussiert (vgl. Kata Container 2020a). Die neuste und interessanteste Container Veröffentlichung wurde von einer der größten Unternehmen der Welt AWS herausgebracht, hierbei handelt es sich um den Firecracker-Containerd der im Jahr 2018 von AWS selbst entwickelt wurde. Der Firecracker soll hohe Workload-Isolation bieten und gleichzeitig die Geschwindigkeit und Ressourceneffizient von Containern ermöglichen (vgl. Firecracker Microvm 2020). Diese Arbeit beschäftigt sich mit der Leistungsbewertung von Containerlösungen und deren Technologische Anwendungsarten.

1.1 Problemdarstellung

Viele Unternehmen die auf Schnelligkeit, Sicherheit und kostengünstige Virtualisierung Wert drauf liegen ist es wichtig zu wissen wie sich unterschiedliche Containerlösungen voneinander vergleichen. Diese Forschung wird durchgeführt, um die zwei neuen Containerlösungen Firecracker-Containerd und Kata-Container mit Docker-Container sowohl auf ihre Technologische Anwendungsarten und auch in der Perfomance miteinander zu vergleichen und zu bewerten.

1.2 Zielsetzung

Die Zielsetzung dieser Arbeit ist es dem Laien zu veranschaulichen wie sich die 3 verschiedenen Containerlösungen Firecracker-Containerd, Docker-Container und Kata-Container sowohl von Technischen Aspekten als auch von der Perfomance unterscheiden. Es soll mithilfe von Grafiken veranschaulicht werden wie die CPU-Geschwindigkeit, die RAM-Geschwindigkeit, die Network-Perfomance und Startzeiten sich voneinander unterschieden. Aber nicht nur die Unterscheidung dieser Aspekte soll verdeutlicht werden, sondern auch worin sich diese Containerlösungen von Technologische Aspekten unterscheiden. Welche Vorteile die verschiedenen Containerlösungen versprechen und mit welche Technologischen Wissen diese entwickelt wurde.

1.3 Aufgabenstellung

Die Aufgabe mit dem sich diese Arbeit beschäftigt ist die Leistungsbewertung der Container-Technologien: Firecracker-Containerd, Kata-Container und Docker-Container. Zunächst werden die einzelnen Container-Technologien auf einem Linux Betriebssystem installiert, sodass das sie in einer ausführbaren Umgebung mit einer Netzwerkverbindung laufen. Nachdem diese installiert sind werden die 3 Container auf den gleichen Systemstand gebracht. Dabei wird für alle 3 Containerarten dieselbe Docker Image verwendet. Der Docker Image wird mithilfe eines Dockerfiles erzeugt und dann für den jeweiligen Container ausgeführt, sodass alle 3 Technologien die gleichen Systemvoraussetzungen erfüllen.

Nachdem die die Container alle auf den gleichen stand sind, werden verschiedene shell-scripts verwendet, um die verschiedenen Testdurchläufe auszuführen und die Ergebnisse in einer Datei abzulegen. Diese Ergebnisse werden mithilfe von Python Programmiert eingelesen, und zum

Schluss dann geplottet, sodass man die Ergebnisse in Grafiken dargestellt wird und eine Bewertung der unterschiedlichen Perfomance stattfinden kann. Bestandteil der Arbeit ist es nicht die Gesamte Technologischen Hintergründe der Containerlösungen zu erklären, sondern die Leistungen der Container zu Bewertungen und wichtige Technische Prozesse, um Hintergrund zu verstehen. Es werden nur die dafür benötigten Technischen Hintergründe aufgeklärt, die zum Verständnis der Leistungsbewertung der Containerlösungen auch benötigt wird.

1.4 Aufbau der Arbeit

In Kapitel 2 werden die benötigten Theoretischen Grundladen dieser Arbeit behandelt. Zu Beginn wird erklärt der Virtualisierung bedeutet, danach was unter Container-basierte Virtualisierung zu verstehen ist und zum Schluss die Vorteile und Nachteile zwischen einer Virtuellen Maschine und Container-basierte Virtualisierung. Darauffolgend wird sowohl Docker-Container, Firecracker-Containerd als auf Kata-Container von ihre Funktionsweise beschrieben und aus welchen Schichten diese besteht.

Im Kapitel 3 wird erklärt wie man die einzelnen Containerlösungen installiert und welche Voraussetzungen benötigt wird, um die Container zu starten.

Im Kapitel 4 wird erklärt, wie die einzelnen Container auf demselben Systemstand gebracht wird und welche Tools für die Durchführung der Testdurchläufe benötigt wird.

Im Kapitel 5 werden die Testdurchläufe erklärt und wie die zu ausführenden shell-scripts zu verstehen sind.

Im Kapitel 6 wird beschrieben wie die Ergebnisse in einem Programm eingelesen und in Grafiken dargestellt wird. Danach erfolgt eine Bewertung der dargestellten Ergebnisse, sodass man auf ein Fazit kommt.

2 Theoretische Grundlagen

Im Theoretischen Teil werden Grundwissen über die Virtualisierung, Container, Docker, Firecracker und Kata Container aufgeklärt. Es werden die wichtigsten Teschnischen Hintergründe dieser Containerlösungen erklärt, um bei der Durchführung zu verstehen was im Hintergrund passiert.

2.1.1 Virtualisierung

Schone Ende 1960-Jahre wurde in einem Großrechner die Virtualisierungstechnik verwendet.

Mit der Virtualisierung war es möglich, dass mehrere Benutzer auf einem System parallel arbeiten konnte, dadurch hat man eine Menge Hardwarekosten erspart und das Interesse vieler Unternehmen enorm erhöht (Buhl, H. und Winter 2008).

Bei der Virtualisierung werden physischer Hardwareressourcen, Softwareressourcen, Speicheressourcen und Netzwerkkomponenten abstrahiert, um diese auf der virtuellen Ebene zur Verfügung zu stellen. Mithilfe dieser Bereitstellung soll der Verbrauch von IT-Ressourcen bei der Virtualisierung stark reduziert werden (vgl. IONOS 2020).

Es gibt eine sogenannte Software namens Hypervisors, diese Software ist für die Trennung der physischen Ressourcen von den virtuellen Maschinen zuständig. Im Prinzip sind virtuelle Maschinen, Systeme, die Ressourcen benötigen, um zu laufen. Diese IT-Ressourcen werden vom Hypervisor auf die jeweiligen virtuellen Maschinen partitioniert, sodass mehrere virtuelle Maschinen gleichzeitig laufen können (vgl. redhat Inc. 2020a).

2.1.2 Containerbasierte Virtualisierung

Containerbasierte Virtualisierung gilt als Leichtgewichte Alternative zu herkömmlichen virtuellen Maschinen, weil viel weniger IT-Ressourcen benötigt wird. Bei Virtuellen Maschinen werden die Ressourcen vollständig vom Hypervisor abgebildete und Container bilden nur ein Abbild der benötigten Betriebssysteme und deren Funktionen. Container bestehen aus 2 verschiedenen Teilen.

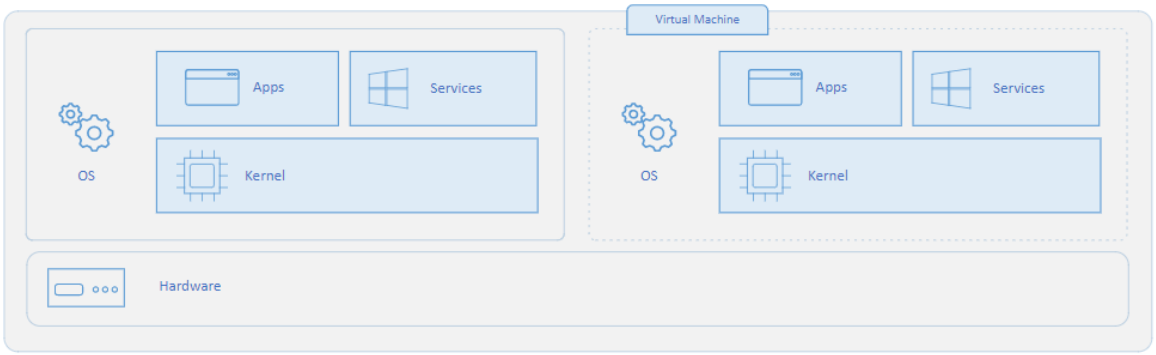
Damit ist ein Container im Prinzip nichts anderes als eine Software, die Code und alle seine Abhängigkeiten zusammenbringt, damit die Anwendung schneller und zuverlässiger ausgeführt werden kann. Ein Container wird mithilfe einer Image-Datei aufgebaut, diese beinhaltet

alle erforderlichen Anwendungen wie Code, Systemtools und Systembibliotheken, sodass der Container eigenständig laufen kann (vgl. Docker Inc. 2020).

Bei Containerbasierter Virtualisierung spielt es keine Rolle um welches Nutzen es sich handelt, es wird immer leicht und konsistent bereitgestellt (vgl. Cloud Google).

2.1.3 Vergleich von Containern und virtuellen Maschinen

Bei der Isolierung und Zuweisungen der IT-Ressourcen ähneln sich Container und virtuelle Maschinen. Woran sie sich unterscheiden, ist die Art der Virtualisierung. Bei einer virtuellen Maschine virtualisiert die Hardware das Betriebssystem und beim Container wird es vom Container selbst virtualisiert (vgl. Docker Inc. 2020).

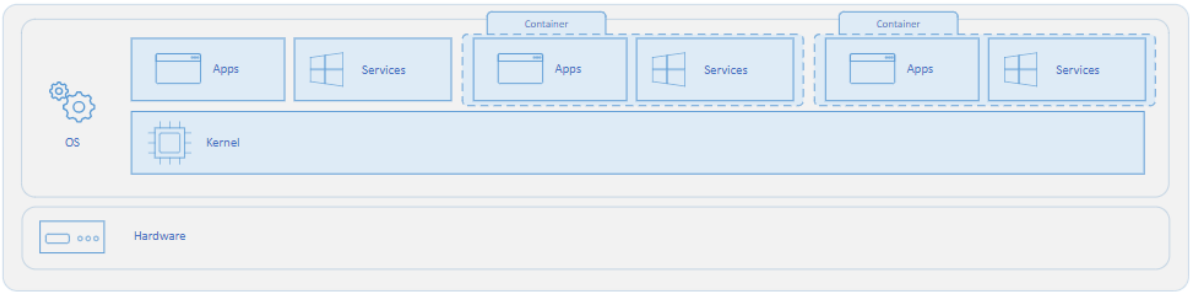


Quelle: (Docs Microsoft Inc. 2020)

Abbildung 1: Architektur von virtueller Maschine

Abbildung 1 zeigt das im Gegensatz zu einem Container eine Virtuelle Maschinen ein vollständiges Betriebssystem mit Samt seine Komponenten Abbildetet (vgl. Docs Microsoft 2019). Man unterscheidet bei der Virtualisierung 2 Arten, einmal die Aggregation von Ressourcen und das Aufsplitten von Ressourcen. Wenn man von aggregiert spricht wird eine virtuelle Umgebung auf mehrere Geräte abgebildet und wenn Ressourcen gesplittet werden, dann wird ein Gerät in mehreren virtuellen Umgebungen aufgeteilt. Diese beiden Faktoren werden dafür verwendet, um eine Optimale Nutzung der Ressourcen anzubieten (Berl et al. 2010). Beim Ausführen eine virtuelle Maschine wird nicht nur die Anwendung selbst, sondern auch die dafür benötigten Ressourcen benutzt, damit diese dann laufen kann. Dies verursacht ein enormer

Overhead und sehr große Abhängigkeit von notwendigen Bibliotheken, vollwertige Betriebssystem und andere mögliche Dienste, die zum Ausführen der Anwendung benötigt wird (vgl. crisp 2014).



Quelle: (Docs Microsofft Inc. 2020)

Abbildung 2:Architektur von Container

Abbildung 2 zeigt mehrere Containern, worauf eine Anwendung auf dem Hostbetriebssystem läuft. Der Container baut auf dem Kernel des Hostbetriebssystem auf und enthält verschiedene Apps, APIs, und verschiedene Prozesse für das Betriebssystem (vgl. Docs Microsoft 2019). Bei der Ausführung von Containern wird der Linux Kernel und seine Funktionen verwendet, um Prozesse voneinander isoliert, damit diese voneinander unabhängig laufen können (vgl. redhat Inc. 2020b). Mithilfe der Isolation kann der Zugriff von mehreren Containern auf dieselben Kernel Ressourcen erfolgen. Es lässt sich dadurch bestimmten, wie viele Prozessoren, Ram und Bandbreite für jede Container bereitgestellt wird (vgl. crisp 2014).

In der Folgende Tabelle werden wichtige Unterschiede und Gemeinsamkeiten der beiden Virtualisierungsarten erklärt.

	Virtuelle Maschine	**Container**
Isolierung	Die virtuellen Maschinen werden voneinander und vom Hostbetriebssystem isoliert. Es bietet dadurch eine sehr hohe Sicherheitsgrenze an.	Die Container bieten eine vereinfachte Isolierung des Host-Systems und anderen Container, dennoch ist die Sicherheitsgrenze nicht so stark wie bei virtuellen Maschinen.
Betriebssystem	Ein vollständiges Betriebssystem wird ausgeführt mit Kernel. Somit werden mehr Systemressourcen wie CPU, RAM und Speicherplatz benötigt.	Bei Containern ist die Systemressourcen Verteilung viel flexibler als virtuelle Maschinen. Nur die benötigten Ressourcen, die eine Anwendung für ihre Dienste benötigt wird, auch verwendet.
Bereitstellung	Es können mehrere VMs erstellt und verwaltet werden.	Es können auch hier mehrere Container erstellt und verwaltet werden.
Netzwerk	Für jede virtuellen Maschine werden virtuelle Netzwerkadapter erzeugt.	Genauso wie bei virtuellen Maschinen werden virtuelle Netzwerkadapter für jede einzelne Container erzeugt.

Tabelle 1:Unterschied von Container und virtuelle Maschine

Quelle: (Docs Microsoft Inc. 2020)

2.2 Docker-Container

Docker Container ist ein Open-Source Projekt, da es dem Anwender ermöglicht verschiedenen App in einem Container bereitzustellen und zu organisieren. Dabei wird der Linux Kernel verwendet, um IT-Ressourcen wie Prozessor, RAM und Netzwerk voneinander zu isolieren. Zudem lassen sich die Container mit der jeweiligen App vollständig voneinander isolieren, sodass sie unabhängig laufen können. Dabei werden in einen virtuellen Container sowohl Anwendungen als auch Bibliotheken bereitgestellt und auf mehrere verschiedene Linux Server ausgeführt. Dadurch wird die Portabilitätsgrad und Flexibilität stark erhöht (vgl. crisp 2014). Um die technische Seite von Docker zu verstehen gibt es drei wichtige verschiedene Bestandteile der ein Docker Container verwendet, um richtig zu funktionieren. Der erste wichtige Bestandteil ist der Docker Host, es handelt sich hierbei um die Laufzeitumgebungen, wo der Docker Container ausgeführt wird. Das Erstellen, Ausführen und Terminieren kann mithilfe des Docker Clients ausgeführt werden und sorgt zugleich dafür, dass ein Netzwerk definiert werden kann. Mit dem letzten Bestandteil, der Docker Registry können Images angelegt werden und mithilfe des Docker Client gestartet oder terminiert werden. Mit einem Snapshot wird ein Container gespeichert, sodass man den wiederverwenden kann (vgl. eos 2017). Da der Docker Image und daraus erstellte Docker Container kein eigenes Betriebssystem enthält, ist ein Docker Image viel kleiner als eine virtuelle Maschine und dadurch kann ein Container viel schneller gestartet und gestoppt werden, dies kann man wie ein Prozessor betrachten der gestoppt und gestartet wird (vgl. Entwickler 2019).

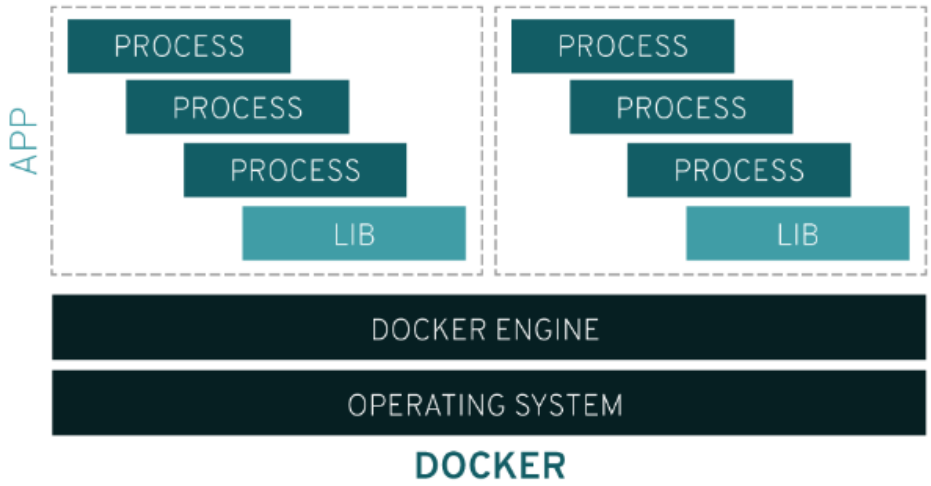


Abbildung 3:Architektur von Docker Container

Quelle: (redhat Inc. 2020)

Abbildung 3 zeigt die Architektur von Docker Container. Der Docker Engine ist dafür zuständig, dass ein Zugriff auf dem Kernel des Host-Betriebssysteme möglich ist und zusätzlich wird der Docker Engine benötigt, um ein Docker Container zu erstellen, zu starten und zu stoppen. Dabei kontaktiert der Docker Client den Docker Engine und der Container kann erstellt, gestartet oder gestoppt werden (vgl. Entwickler 2019). Ein Docker Container wird mithilfe von Dockerfile aufgebaut. Alle Abhängigkeiten lassen sich in einer Docker Image abbilden und draus lässt sich ein Container erstellen und ausführen (vgl. Entwickler 2019).

2.2.1 Docker CLI

Bei Docker Container gibt es das sogenannte Docker CLI. Es handelt sich hierbei um eine Dokumentation von Docker Befehlen, die man benötigt, um Beispielweise mithilfe einer Docker Image ein Container laufen zu lassen. In dieser Dokumentation werden sowohl Docker Befehle definiert und erklärt, wie sie funktionieren. Ein besonderer wichtiger Befehl ist beispielsweise in der Docker CLI der Befehl `docker run`. Dieser Befehl kann in isolierten Docker durchgeführt werden, um aus einer existierende Docker Image ein Docker Container zu erstellen und auszuführen. Dabei wird dem Docker Container eine Container-ID und Docker Name zugewiesen, um diese dann später stoppen, löschen oder starten zu können. In der Regel ist für jede Container Standardmäßig die Automatische Netzwerkbrücke aktiviert. Dabei wird für jede Container, der läuft vom Host-System zum Container eine Netzwerkbrücke zum Container aufgebaut, damit sich der Container mit dem Internet verbinden kann. Natürlich hat man auch die Möglichkeit Benutzerdefinierte Einstellungen für Netzwerkverbindungen und andere Sicherheitsaspekte zu konfigurieren. (vgl. Docs. Docker 2020a).

In der Folgende Tabelle werden wichtige Docker CLI beschrieben die wir später für die Leistungsbewertung benötigen.

Docker Befehl	**Bedeutung**
docker attach	Attach gibt die Standard Eingabe, Ausgabe, und Fehler-Datenströme eines laufenden Containers.
docker build	Baut mithilfe des Dockerfiles eine Image Datei.

docker commit	Erstellt aus den Änderungen eines Images eine neue Image Datei.
docker exec	Das ausführen eines Befehles innerhalb des Containers.
docker images	Die Docker Images auflisten.
docker kill	Eine oder mehrere laufende Container killen.
docker login	Einloggen in ein Docker Register.
docker logout	Ausloggen von einem Docker Register.
docker network	Verwalten des Netzwerkes.
docker pause	Pausieren von allem Prozesse innerhalb eines oder mehreren Containern.
docker ps	Die Docker Container auflisten.
docker pull	Ziehen eines Images oder Repositorys aus einem Register.
docker push	Hochziehen eines Images oder Repositorys in einem Register.
docker rename	Name eines Containers ändern.
docker restart	Neustarten von einem oder mehrere Container.
docker rm	Entfernen von einem oder mehrere Container.
docker rmi	Entfernen von einem oder mehrere Images.
docker run	Ausführen von Befehlen in einem neuen Container.
docker stop	Stoppen von einem oder mehreren laufenden Containern.

Tabelle 2: Docker CLI

Quelle: (Docs. Docker 2020b)

2.2.2 Dockerfile

Das Dockerfile ist eine Zusammenfassung von mehreren Befehlen der den Inhalt eines Images beschreibt und somit die Baueinleitung für Container ist (vgl. Entwickler 2019).

Das Format des Dockerfiles besteht aus `INSTRUSTION arguments`. Ein Kommentar kann mithilfe von `#` definiert werden. Diese Kommentarzeilen werden bevor der Dockerfile zu einem Image aufgebaut wird automatisch gelöscht (vgl. Docs. Docker 2020c).

In der Folgende Tabelle werden wichtige Anweisungen beschrieben die in einem Dockerfile benötigt wird, um daraus dann eine Docker Image aufzubauen.

INSTRCTION	**arguments**
MAINTAINER	Autor des Images.
RUN	Ein shell Befehl kann dadurch ausgeführt werden.
CMD	Ein CMD Befehl kann nach dem Starten eines Docker Containers ausgeführt werden.
EXPOSE	Mithilfe von EXPOSE können Ports angegeben werden, auf den der Container dann hört.
ADD	Es ermöglicht komprimierte Dateien automatisch zu öffnen und zu entpacken.
COPY	Dadurch kann der Inhalt vom Host-Betriebssysteme in einem Container kopiert werden.
ENV	Dadurch können Umgebungsvariablen innerhalb des Containers gesetzt werden.
USER	Dadurch kann der User festgelegt werden unter welchem die Skripte dann ausgeführt werden können.

ENTRYPOINT	Dadurch kann festgelegt werden, welcher Befehl beim Starten des Containers ausgeführt werden soll

Tabelle 3: Dockerfile-Anweisungen

Quelle: (anecon, 2018)

2.3 Firecracker Containerd

Firecracker ist ein sehr neues Projekt die vor knapp 2 Jahren von AWS veröffentlicht, die neue virtualiersungs- und Open-Source Technologie. Es soll ermöglicht werden, dass Serverbesitzer sicherer containerbasierte Services zu benutzen. Die Geschwindigkeit, Ressourceneffizient und Leistungen der Container sollen mit VMs kombiniert werden. MicroVMs sind gegenüber VMs eine verbesserte Form von VMs, sie sind Sichererer und bieten eine verbesserte Work-Isolation an. Gleichzeitig bieten sie wie normale Container eine hohe Geschwindigkeit an und gute Ressourceneffizienz (vgl. Firecracker Microvm 2020). Zudem wird eine reduzierte Speicherverbrauch und Sandboxing-Umgebung für jede MicroVM angeboten (vgl. Firecracker Microvm 2020). Sandbox ist eine Isolierte Umgebungsbereich, die von anderen Bereichen abgeschottet ist. Dadurch lassen sich beispielsweise Konflikte zwischen Betriebssysteme und Software durch die isolierte Umgebung vermeiden (vgl. Vogel 2018). Dadurch werden Anwendungen nicht direkt auf dem Hostbetriebssystem ausgeführt (vgl. Cloud Google). In dem nächsten Kapitel wird genauer auf die Funktionsweise des Firecracker eingegangen und bestimmte Komponente und Technische Hintergrundwissen aufgeklärt.

2.3.1 Firecracker funktionsweise

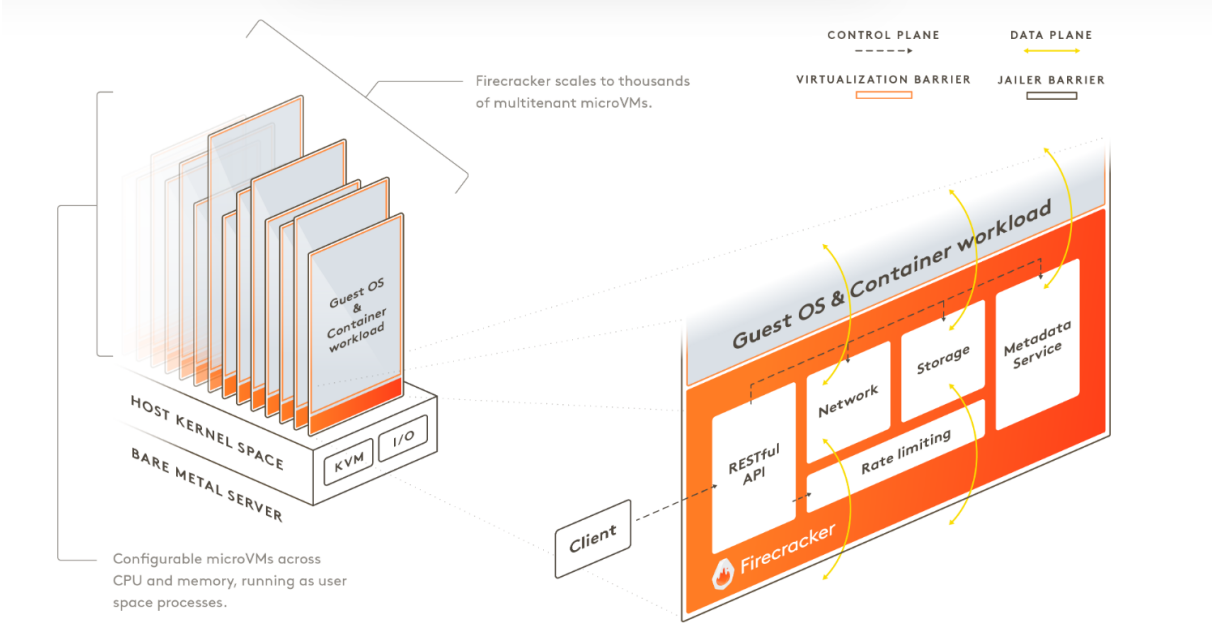


Abbildung 4: Firecracker in Funktionsweise

Quelle: (Firecracker Microvm 2020)

Abbildung 4 zeigt wie Firecracker funktionsweise arbeitet. Der Firecracker verwendet auf dem Kernel-bassierende virtuelle Maschine (KVM, um microVMs zu erstellen. Die Ausführung erfolgt im Benutzerbereich. Durch die schnelle Startzeit und geringe Speicheraufwand können Tausende microVMs erzeugt werden. Jede Funktion, jeder Container bzw. Containergruppe mit einer Barriere für virtuelle Maschinen eingekapselt werden. Mithilfe dieser Funktionen können Workloads auf derselben Maschine ausgeführt werden, ohne sich über die Sicherheit und Effizienz Gedanken zu machen. Es kann eine beliebige Anzahl von Gastbetriebssystemen gehostet werden (vgl. Firecracker MicroVm 2020). Firecracker soll einen VMM basierend auf dem virtuellen Maschinen Kernel implementieren und dadurch ein RESTful API anbieten, um microVMs zu erstellen und verwalten. Die vCPU und Speicher können unabhängig irrerer Anwendungsanforderungen in beliebigen Kombinationen mit microVMs erstellt werden (vgl. Firecracker Microvm 2020). Durch ein Rate limiting können Netzwerke und Speicherressourcen von mehreren microVMs gesteuert werden. Die Rate limiting können über die Firecracker-API erstellt und konfiguriert werden. Außerdem gibt es noch eine Metadaten Service, der für den Austausch von Informationen zwischen dem Host-Betriebssystem und Gast-Betriebssystem zuständig ist. Der Metdatendienst kann genauso wie der Rate limiting über die Firecracker-API eingerichtet und konfiguriert werden. Um die Sicherheit von microVM zu erhöhen hat man ein Begleitprogramm mit dem Namen Jailer implementiert, diese sorgt für eine Sicherheitsbarriere

im Linux user space. Diese Barriere soll eine doppelte Sicherheit gewähren, im Falle der Fälle, wenn die Virtualisierungbarriere durchbrochen wird (vgl. Firecracker Microvm 2020).

2.3.2 Firecracker Containerd Architektur

Es werden die wichtigsten Komponenten von Firecracker-Containerd erklärt, damit der Technische Hintergrund einigermaßen verständlich bzw. nachvollziehbar ist. Dazu gehören wichtige Komponente wie Orchestrator, VMM, Agent, Snapshotter, V2 runtime und der control plugin.

2.3.2.1 Orchestrator

Container Orchestration ermöglicht, dass mehrere verknüpfte Anwendungen in einem Container zusammenarbeiten können. Dadurch können bestimmte Anwendungen in einer festgelegten weiße funktionieren. Es können beispielweise durch den Benutzer Container gestoppt und gestartet werden (vgl. HPE 2020).

2.3.2.2 Virtual Maschine Manager

Die VMM soll die Ausführung von Containern mit einer Isolierung der virtuellen Maschine erleichtern. Dafür wurden 2 Schnittstellen in Firecracker-Containerd implementiert, einmal der snapshotter und zum anderen der V2 runtime (vgl. Github 2019c).

2.3.2.3 Agent

Der Agent wird innerhalb von Firecracker microVM benötigt, um eine Verbindung mit runc aufzubauen und dadurch dann Container zu erstellen. Er Kommuniziert außerhalb der VM mit dem runtime über die vsock (vgl. Github 2018a). Runc ist ein CLI-Tool mit denen man Container Spawnen und ausführen kann (vgl. Github 2020b). Ein Container runtime ist eine Software, die für das Ausführen und Verwalten eines Containers auf einem Knoten zuständig ist (Lou und Brown 2017). Der vsock ist für die Erleichterung der Kommunikation zwischen einer virtuellen Maschine und Host-Betriebssystem zuständig (vgl. Man7 2020).

2.3.2.4 Snapshotter

Snapshotters ist für die Bereitstellung von Layer Storage und UnionFS für Containerd Container zuständig (vgl. Github 2019c). Der Storage ist eine Art Speicherlösung, um Daten temporär bzw. dauerhaft aufzubewahren (vgl. Floyd und Dr. Bergler 2017). Mithilfe von UnionFS lassen für Schnittstellen vereinfacht stapelbare Dateisysteme bereitstellen (Linux Magazin 2005).

2.3.2.5 V2 runtime

Mithilfe von runtime können die Implementationen für Konfigurationen und Laufzeiten von Container Prozesse bereitgestellt werden. Der V2 runtime gilt als eine spezifische Schnittstelle, die nicht von OCI standardisiert ist (vgl. Github 2019c).

2.3.2.6 Control Plugin

Der Control Plugin ist für die Verwaltung des Lebenszyklus von runtime zuständig. Außerdem ist der Plugin für die Implementierung von API zuständig (vgl. Github 2019c).

2.4 Kata Container

Kata Container verhält sich wie ein Container, legt aber zusätzlich noch viel Wert auf die Workload-Isolations und Sicherheitsvorteile von VMs. Es ist sozusagen eine Kombination aus den Vorteilen eines Containers und VMs. Zurzeit befindet sich der Kata Container auch wie Firecracker am Anfang ihrer Phase und ist deswegen noch in der Entwicklung. Zurzeit kann der Kata-Container im Linux Betriebssystem installiert werden. Es ist ein Open Source Community Projekt und kann auch kostenlos unter der Lizenz von Apache 2.0 installiert und bei der Entwicklung im mitgewirkt werden. Da es sich um ein Community Open Source Entwicklung handelt, ist Kata-Container drauf angewiesen das freiwillige bei diesem Projekt auch mitwirken (vgl. Kata Container 2020a).

Der Kata-Container Projekt wurde aus 2 verschiedenen Projekten vereinigt, einmal der Intel Clear Container Projektund Hyper runV Projekt. Man hat die die Technologien dieser beiden Projekte zusammengefasst und daraus entstand dann das Kata-Container Projekt (vgl. Kata Container 2020a). Der Intel Clear Container Projekt hat Beispielweise dazu beigetragen, dass der dazu resultierende Kata-Container Bootzeiten von <100ms schafft dazu bietet es noch eine

höhere Sicherheit an. Der Hyper runV hat bei Kata-Container auf die Unterstützung von Technologische-Agnostische fokussiert (vgl. OpenStack Foundation 2017). Unter Agnostisch versteht man, dass etwas soweit verallgemeinert wurde, dass es auch unter verschiedene Umgebungen eingesetzt werden kann (vgl. Rouse M. 2020). Durch diese Zusammenführung bietet der Kata-Container eine Kompatible Leistungsfähige Technologie an (vgl. OpenStack Foundation 2017).

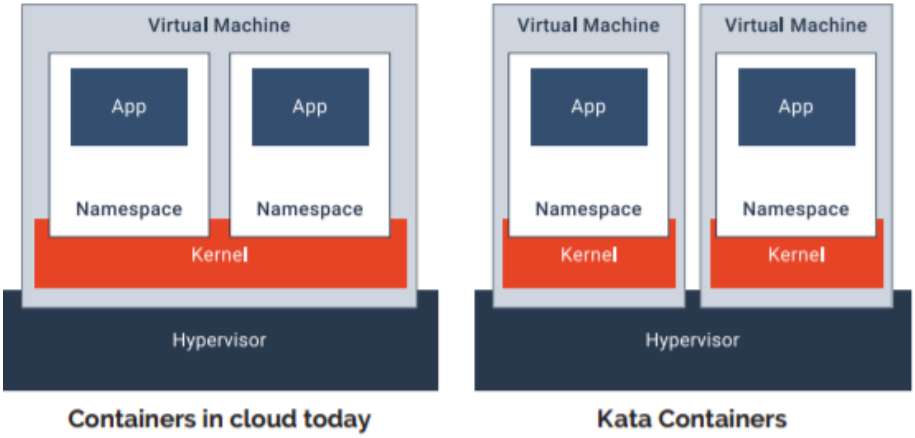


Abbildung 6: Unterschied zwischen Containern in Cloud und Kata Containern

Quelle: (OpenStack Foundation 2017)

Abbildung 6 zeigt das sowohl des Containers in einer Cloud als auch ein Kata Container beide Hypervisor verwenden. Man sieht auf der Abbildung das links 2 virtuelle Maschinen in einer gleichen Umgebung arbeiten, im Gegenzug ist bei der Kata Container beide virtuellen Maschinen voneinander isoliert, so sorgt man für eine höhere Sicherheit. Dadurch ist es nicht nur sicherer, sondern bietet auch eine bessere Skalierbarkeit, und höhere Ressourcenauslastung (vgl. OpenStack Foundation 2017).

2.4.1 Kata Container Architektur

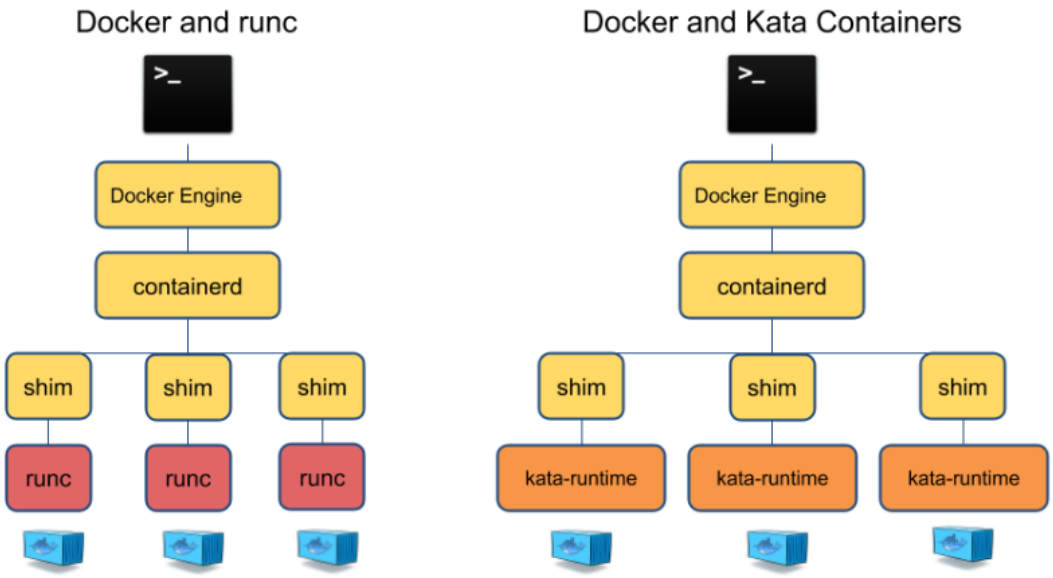


Abbildung 7: Docker und runc im vergleich mit Docker und Kata Containers

Quelle: (Github 2020d)

Auf diese Abbildungen werden 2 unterschiedliche Anwendungen von Docker Container aufgezeigt. Die Linke Abbildung zeigt Docker Container mit runc und die rechte Abbildung zeigt die Kombination von Docker und Kata Containers. Auf diese Abbildungen kann man sehen das bei der Kombination von Docker und Kata Container anstatt runc, kata-runtime verwendet wird. Die Kombination aus Docker Engine und kata-runtime läuft einwandfrei, der kata-runtime erstellt für jede Container ein QEMU KVM-Maschine, der dann innerhalb des Kata Containers läuft (vgl. Github 2020d). Mithilfe des kata-shims kann ein POD Sandbox für jeden Container erstellt werden (vgl. Github 2020d).

2.4.1.1 Guest Assests

Eine virtuelle Maschine wird mit eine minimale Gast-Kernel und Gast-Images, mithilfe von Hypervisor gestartet. Der Gast Kernel wird zum Hochfahren der virtuellen Maschine verwendet. Es ist auf minimalen Speicherbedarf ausgeprägt und stellt nur die Dienste, die auch für eine Container erforderlich sind zur Verfügung. Der Gast-Images unterstützt sowohl eine Initird als auch ein Rootfs Image. Bei der Rootfs image handelt es sich um eine optimierte bootstrap system, der für eine minimale Umgebung sorgt. Im Gegensetz zu dem Rootfs image besteht der

Initird aus eine komprimierte cpio-Archiv, das als Linux-Startprozess verwendet wird. Während des Startvorgangs, wird vom Kernel eine Speziale tmpfs entpackt und dadurch dann ein Root Dateisystem erzeugt (vgl. Github 2020d).

2.4.1.2 Agent

Der Agent läuft in einem Gast als Prozess und ist für die Verwaltung von Containern zuständig. Zur wichtigen Ausführungseinheit für sandbox gehört der kata-agent. Diese definiert verschiedene Namensräume wie Beispielsweise PID oder IPC (vgl. Github 2020d).

2.4.1.3 Runtime

Der kata-runtime nutzt das virtcontainers projekt, die eine Agnostische und hardware-virtualisierte Container Bibliotheken für Kata-Container bereitstellt (vgl. Github 2020d). Es gibt die Datei configration.toml die automatisch bei der Installation des Kata-Containers erzeugt wird, in dieser Datei werden verschiede Pfade festgelegt. Beispielweise muss man den Pfad angeben, wo sich genau der Hypervisor oder die Image Datei befindet (vgl. Github 2020d).

3 Installation

Ab diesem Abschnitt befassen wir uns nicht mehr mit den Theoretischen Dingen, sondern mit den Praktischen Teilen, um die Leistungen der Container zu bewerten.

In diesem Abschnitt wird erklärt, wie man die Unterschiedlichen Containerlösungen installiert zunächst installiert.

3.1 Docker Container installieren

Zunächst erfolgt die Installation von Docker auf einem Ubuntu-Betriebssystem. Danach wird ein Dockerfile konfiguriert und daraus dann der Docker Image erstellt. Nachdem ein Image erstellt wurde kann mithilfe des Docker Images ein Container ausgeführt. Der Dockerfile wird sowohl für Docker Container, Firecracker Containerd und Kata Container als Basis verwendet, um eine äquivalente Umgebung zu realisieren.

Wie schon in der Einführung angedeutet wird zunächst der Docker auf ein Ubuntu-Betriebssystem installiert. Die Anleitung wie man Docker installiert findet man auf der Offiziellen Docker Seite (vgl. Docs. Docker 2020d).

Die Folgenden schritte zeigen wie Docker installiert wird:

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

Docker installieren, Schritt 1: Ältere Docker Versionen entfernen

Zunächst werden ältere Docker Versionen, die schonmal installiert waren, entfernt.

```
sudo apt-get update

sudo apt-get install \
       apt-transport-https \
       ca-certificates \
       curl \
       gnupg-agent \
    software-properties-common
```

Docker installieren, Schritt 2: Aktualisierung des Systems und Tools installieren

Bei diesem Vorgang wird das Ubuntu-Betriebssystem zunächst aktualisiert und danach die benötigten Tools für die Installation von Docker installiert.

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo apt-key fingerprint 0EBFCD88

pub   rsa4096 2017-02-22 [SCEA]

      9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88

uid           [ unknown] Docker Release (CE deb) <docker@docker.com>

sub   rsa4096 2017-02-22 [S]
```

Docker installieren, Schritt 3: Das GPG Herunterladen und überprüfen

Im dritten Schritt wird der GPG von Docker heruntergeladen und dann mit dem fingerprint Befehl überprüft, ob GPG tatsächlich von Docker ist. Die `pup` Ausgabe muss mit dem folgenden Schlüssel Öffentliche Schlüssel von Docker `9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88` übereinstimmen.

```
sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"
```

Docker installieren, Schritt 4: Repository von Docker hinzufügen

Hier wird der Repository von Docker auf unser Linux Ubuntu System hinzugefügt. Je nach Linux System muss der Download link am Ende des Pfades verändert werden. Bei einem Debian System würde es die Download Adresse folgendermaßen aussehen: `https://download.docker.com/linux/debian`.

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Docker installieren, Schritt 5: System Aktualisieren und Docker Engine installieren

Nachdem der Repository heruntergeladen wurde, wird das System noch einmal aktualisieren und anschließend der Docker Engine installiert, um Container beispielsweise zu starten oder zu stoppen.

```
apt-cache madison docker-ce

        docker-ce | 5:18.09.1~3-0~ubuntu-xenial | https://download.docker.com/linux/ubuntu  xenial/stable amd64 Packages
        docker-ce | 5:18.09.0~3-0~ubuntu-xenial | https://download.docker.com/linux/ubuntu  xenial/stable amd64 Packages
        docker-ce | 18.06.1~ce~3-0~ubuntu       | https://download.docker.com/linux/ubuntu  xenial/stable amd64 Packages
        docker-ce | 18.06.0~ce~3-0~ubuntu       | https://download.docker.com/linux/ubuntu  xenial/stable amd64 Packages

sudo apt-get install docker-ce=<VERSION_STRING> docker-ce-cli=<VERSION_STRING>
containerd.io

sudo docker run hello-world
```

Docker installieren, Schritt 6: Docker Engine Version auswählen

Mit dem Ersten Befehl wird der Docker Cache für Docker Engine Versionen aufgelistet und eine Docker Engine Version für ein Linux Betriebssystem ausgewählt. In unserem Fall wird ein Ubuntu 18.06 verwendet, somit wird die Docker Engine Version `18.06.0~ce~3-0~ubuntu ausgewählt.` Dies Versionsnummer wird mit den Platzhaltern des unteren Befehles ersetzt und durchgeführt. Nachdem der Docker Engine erfolgreich installiert wurde, kann man den Daemon mit dem Befehl `sudo service Docker start/status/stop`, den starten, stoppen oder den Status abfragen. Wenn der Daemon keine Probleme beim, starten und Status Abfrage aufweist kann man mit dem Befehl `sudo docker run hello-word`, den ersten Docker Container starten.

3.1.1 Dockerfile

Bis hierhin wurde der Docker erfolgreich installiert, somit können wir den ersten Dockerfile konfigurieren, um die raus resultierende Docker Image dann für unsere Leistungsbewertung der drei Containerlösungen zu verwenden.

Zunächst müssen wir ein Dockerfile mit dem Befehl `sudo touch Dockerfile` erzeuge. Nachdem der Dockerfile erzeugt wurde müssen dem Befehl `sudo nano Dockerfile` konfigurieren. Der Inhalt des Dockerfiles sollte folgendermaßen aussehen.

```
# Dockerfile für Firecracker-Container, Kata-Container und Docker-Container.
# Installation von: Debian System, wichtige Tools und Webserver Apache2.
from debian # Zeile:1

MAINTAINER Vahel Hassan <Vahel.Hassan@outlook.de> # Zeile:2
RUN apt-get update && apt-get install -y # Zeile:3
# Tools die wir später für unsere Leistungsbewertung benötigen
RUN apt-get install sysstat -y # Zeile:4
RUN apt-get install nicstat -y # Zeile:5
RUN apt-get install iftop -y # Zeile:6
RUN apt-get -y install nano # Zeile:7
RUN apt-get -y install wget # Zeile:8
# Install Benchmark
RUN echo "deb http://deb.debian.org/debian  stretch main" >> /etc/apt/sources.list # Zeile:9
RUN echo "deb-src http://deb.debian.org/debian  stretch main" >> /etc/apt/sources.list # Zeile:10
RUN apt-get update # Zeile:11
```

```
RUN apt-get install -y libmariadbclient18 # Zeile:12
RUN apt-get install -y sysbench # Zeile:13
# Install Apache2
RUN apt-get install apache2 -y # Zeile:14
RUN mkdir /run/lock # Zeile:15
RUN mkdir -p /var/www/ # Zeile:16
RUN chown -R $USER:$USER /var/www/ # Zeile:17
RUN chmod -R 755 /var/www/ # Zeile:18
RUN service apache2 restart # Zeile:19
```

Code 1: Dockerfile

Der Dockerfile beinhaltet die Bauanleitung für das Image, dieses Image wird für die Leistungsbewertung der 3 verschiedenen Containerlösungen verwendet.

In Zeile 1 wird die Anweisung `from` aufgerufen. Mithilfe dieser Anweisung wird Debian auf unsere Image Datei installiert.

Danach wird in Zeile 2 der Autor und die dazugehörige E-Mail-Adresse mithilfe der Anweisung `MAINTAINER` definiert. Die E-Mail-Adresse wird benötigt, um den Autor bei Problemen oder Fragen kontaktieren zu können.

In Zeile 3 wird beim Aufbauen des Docker Images, der erstellte Debian System mit den Befehlen `update` aktualisiert und mit `install` können andere verschiedene Pakete auf das Debian System installiert werden.

In den weiteren Zeilen folgen `run` Anweisungen, mit dieser Anweisung kann auf das installierte Debian System verschiedene Pakete installiert werden und verschiedene Linux-Befehle ausgeführt werden.

In den Zeilen 4-6 werden wichtige Tools mit der Anweisung `run` installiert, diese Tools werden eventuell für die Leistungsbewertung der Containerlösungen benötigt. Mit dem Tool `sysstat` können verschiedene Systemleistungen überwacht werden (vgl. Packages Debian SPI Inc. 2020a). Das Netzwerk kann durch den Befehl `nicstat` überwacht werden (vgl. Packages Debian SPI Inc. 2020b). Mit dem letzten Tool in Zeile 6 kann das Netzwerkverkehr überwacht werden, um Schluss zu folgern wie die aktuelle Bandbreitnutzung erfolgt (vgl. Packages Debian SPI Inc. 2020c).

4 Leistungsbewertung

5 Zusammenfassung

6 Ausblick

7 Anhang

8 Literaturverzeichnis

Witt, M., Jansen, C., Breuer, S., Beier, S., Krefitng, D. (2017), Artefakterkennung über eine cloud-basierte Plattform, 19. September, https://link.springer.com/article/10.1007/s11818-017-0138-0#citeas, letzter Zugriff: 22.07.2020.

DOCKER.de Inc. (2020), What is a Container?, https://www.docker.com/resources/what-container, letzter Zugriff: 23.07.2020.

KATACONTAINERS.io (2020a), An overview of the Kata Containers project, https://katacontainers.io/learn/, letzter Zugriff: 23.07.2020.

AWS.AMAZONE.com (2020), Jetzt neu: Firecracker, ein neues Virtualisierungstechnologie- und Open-Source-Projekt zur Ausführung von Mehrmandanten-Container-Workloads, 26. November, https://aws.amazon.com/de/about-aws/whats-new/2018/11/firecracker-lightweight-virtualization-for-serverless-computing/, letzter Zugriff 23.07.2020.

FIRECRACKER-MICROVM.GITHUB.io (2018), Firecracker is an open source virtualization technology that is purpose-built for creating and managing secure, multi-tenant container and function-based services, https://firecracker-microvm.github.io/, letzter Zugriff: 23.07.2020.

Buhl, H., Winter, R. (2008), Vollvirtualisierung – Beitrag der Wirtschaftsinformatik zu einer Vision, 13. Dezember, https://link.springer.com/article/10.1007/s11576-008-0129-7, letzter Zugriff: 23.07.2020.

IONOS.de (2020), Konzepte der Virtualisierung im Überblick, 24. März, https://www.ionos.de/digitalguide/server/konfiguration/virtualisierung/, letzter Zugriff: 24.07.2020.

REDHAT.com Inc. (2020a), Was ist Virtualisierung?, https://www.redhat.com/de/topics/virtualization/what-is-virtualization, letzter Zugriff: 24.07.2020.

REDHAT.com Inc. (2020b), Docker – Funktionsweise, Vorteile, Einschränkungen, https://www.redhat.com/de/topics/containers/what-is-docker

, letzter Zugriff: 24.07.2020.

CLOUD.GOOGLE.com, CONTAINER BEI GOOGLE,
https://cloud.google.com/containers?hl=de, letzter Zugriff: 24.07.2020.

DOCS.MICROSOFT.com Inc. (2019), Container im Vergleich zu virtuellen Computern, 21. Oktober, https://docs.microsoft.com/de-de/virtualization/windowscontainers/about/containers-vs-vm, letzter Zugriff, 24.07.2020.

Berl. A., Fischer A., Meer H. (2010), Virtualisierung im Future Internet, 16. Februar, https://link.springer.com/article/10.1007/s00287-010-0420-z, letzter Zugriff: 24.07.2020.

DOCS.DOCKER.com (2020a), Docker run reference, https://docs.docker.com/engine/reference/run/, letzter Zugriff: 24.07.2020.

DOCS.DOCKER.com (2020b), The base command for the Docker CLI, https://docs.docker.com/engine/reference/commandline/docker/, letzter Zugriff: 24.07.2020.

DOCS.DOCKER.com (2020c), Build an image from a Dockerfile, https://docs.docker.com/engine/reference/commandline/build/, letzter Zugriff: 24.07.2020.

EOSGMBH.de (2017), Was sind eigentlich Container und Docker?, 13. September, https://www.eosgmbh.de/container-und-docker, letzter Zugriff: 24. 07.2020.

CRISP-RESEARCH.com (2014), Docker Container: Die Zukunft moderner Applikationen und Multi-Cloud Deployments?, 31. Juli, https://www.crisp-research.com/docker-container-die-zukunft-moderner-applikationen-und-multi-cloud-deployments/, letzter Zugriff: 24.07.2020.

ENTWICKLER.de (2019), Docker: Einstieg in die Welt der Container, 19. Februar,, https://entwickler.de/online/windowsdeveloper/docker-grundlagen-dotnet-container-579859289.html, letzter Zugriff: 24.07.2020.

ANECON.com (2018), Docker Basics – Befehle und Life Hacks, 26. Juni, http://www.anecon.com/blog/docker-basics-befehle-und-life-hacks/, letzter Zugriff: 25.07.2020.

GITHUB.com (2018a), agent, 19. November, https://github.com/firecracker-microvm/firecracker-containerd/blob/master/docs/agent.md, letzter Zugriff: 26.07.2020

Lius L., Brown M. (2017), Containerd Brings More Container Runtime Options for Kubernetes, 02. November, https://kubernetes.io/blog/2017/11/containerd-container-runtime-options-kubernetes/, letzter Zugriff: 26.07.2020.

GITHUB.com (2020b), runc, 23. Juli, https://github.com/opencontainers/runc, letzter Zugriff: 26.07.2020.

MAN7.org (2020), vsock(7) – Linux manual page, 2. September, https://man7.org/linux/man-pages/man7/vsock.7.html, letzter Zugriff: 26.07.2020.

GITHUB.com (2019c), firecracker-containerd architecture, 04. Juni, https://github.com/firecracker-microvm/firecracker-containerd/blob/master/docs/architecture.md, letzter Zugriff: 26.07.2020

Floyd B., Dr. Bergler A. (2017), Was ist Storage?, 19. Oktober, https://www.it-business.de/was-ist-storage-a-663183/, letzter Zugriff 26.07.2020.

LINUX-MAGAZIN.de (2005), Überlagerte Dateisysteme in der Praxis, https://www.linux-magazin.de/ausgaben/2005/10/hochstapler/, letzter Zugriff: 26.07.2020.

GITHUB.com (2020c), Firecracker Design, 01. Juli, https://github.com/firecracker-microvm/firecracker/blob/master/docs/design.md#host-integration, letzter Zugriff: 26.07.2020.

HPE.com (2020), WAS IST CONTAINER ORCHESTRATION?, https://www.hpe.com/de/de/what-is/container-orchestration.html, letzter Zugriff: 26.07.2020.

GITHUB.com (2020d), Kata Containers Architecture, 20. Juli, https://github.com/kata-containers/documentation/blob/master/design/architecture.md, letzte Zugriff: 26.07.2020.

OpenStack Foundation (2017), The speed of containers, the security of VMs, https://katacontainers.io/collateral/kata-containers-1pager.pdf, letzter Zugriff: 26.07.2020.

Rouse M. (2020), Agnostisch, https://whatis.techtarget.com/de/definition/Agnostisch, letzter Zugriff: 26.07.2020.

DOCS.DOCKER.com SPI Inc. (2020d), Build an image from a Dockerfile, https://docs.docker.com/get-docker/, letzter Zugriff: 30.07.2020.

PACKAGES.DEBIAN.org SPI Inc. (2020a), Paket: sysstat (11.0.1-1), https://packages.debian.org/de/jessie/sysstat, letzter Zugriff: 30.07.2020.

PACKAGES.DEBIAN.org (2020b), Paket: nicstat (1.95-1 und andere), https://packages.debian.org/de/sid/nicstat, letzter Zugriff: 30.07.2020.

PACKAGES.DEBIAN.org SPI Inc. (2020c), Paket: iftop (1.0~pre4-7 und andere), https://packages.debian.org/de/sid/iftop, letzter Zugriff: 30.07.2020.

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich meine **Bachelorarbeit** mit dem Titel

_____

_____

selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie nicht an anderer Stelle als Prüfungsarbeit vorgelegt habe.

_____
Ort

_____ _____
Datum Unterschrift