

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

Kurzfassung

Es gibt in der heutigen Zeit viele Methoden um Systeme zur 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 der Docker Container. Seitdem Docker Container veröffentlicht wurde, interessieren sich immer mehr Unternehmen für Containerlösungen, um leistungsfähiger zu virtualisieren. Seitdem sind zwei neue interessante Projekte entstanden, zum einen der Kata-Container und zum anderen der Firecracker Containerd. Die vorliegende Bachelorarbeit möchte einen Überblick über Technologischen Aspekte für die erwähnten Containerlösungen geben und die Leistung der zwei neuen Containerlösungen mit dem Docker Container vergleichen. Mithilfe von verschiedenen Tests werden die Leistungen der Containerlösungen miteinander verglichen.

Abstract

There are many methods around virtualization systems 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 the Docker Container. Since Docker Container was released, more and more companies are interested in container solutions to virtualize more efficiently. Since then two new interesting projects have been created, the Kata Container and the Firecracker Container. This bachelor thesis wants to give an overview of technological aspects for the mentioned container solutions and compare the performance of the two new container solutions with the Docker Container. By means of different tests the performance of the container solutions will be compared.

Danksagung

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

Außerdem geht auch ein herzlicher Dank an meine Freundin Irem und meinem Bruder Grewan raus, die haben mich während dieser Phase täglich motiviert und am Ende der Abschlussarbeit mit der Korrektur der Arbeit sehr geholfen.

Abbildungsverzeichnis

Abbildung 1-Architektur von virtueller Maschine	17
Abbildung 2-Architektur von Container	18
Abbildung 3-Architektur von Docker Container	20
Abbildung 4-Firecracker Architektur	24
Abbildung 5-Unterschied zwischen Container und Kata Container	27
Abbildung 6-Docker und Kata Containers	28
Abbildung 7-Box-Plot	41
Abbildung 8-Grafik: Primzahlen-Test	43
Abbildung 9-Grafik: Rechenoperationen-Test	46
Abbildung 10-Grafik: Hauptspeicher-Test	49
Abbildung 11-Grafik: Network-Performance-Test	52
Abbildung 12-Grafik: Network-Performance-Test Kata Container	53
Abbildung 13-Grafik: Network-Performance-Test Docker Container	53
Abbildung 14-Grafik: Startzeit-Test	56

Tabellenverzeichnis

Tabelle 1-Unterschied von Container und virtuelle Maschine.....	19
Tabelle 2-Docker CLI	22
Tabelle 3-Dockerfile-Anweisungen	23
Tabelle 4-CTR-Options.....	35
Tabelle 5-Ergebnisse: Primzahlen-Test	44
Tabelle 6-Ergebnisse: Primzahlen-Test	45
Tabelle 7-Ergebnisse: Rechenoperationen-Test.....	46
Tabelle 8-Ergebnisse: Primzahlen-Test	47
Tabelle 9-Ergebnisse: Hauptspeicher-Test.....	50
Tabelle 10-Ergebnisse: Primzahlen-Test	51
Tabelle 11-Network-Perfomance-Test	53
Tabelle 12-Ergebnisse: Primzahlen-Test	54
Tabelle 13-Ergebnisse: Startzeit-Test	57
Tabelle 14-Ergebnisse: Primzahlen-Test	57

Codeverzeichnis

Code 1-Container ausführen.....	32
Code 2-Docker API-Socket aktivieren.....	33
Code 3-config.toml.....	33
Code 4-Terminal 1 starten.....	34
Code 5-Terminal 2 starten.....	34
Code 6-configuration.toml	36
Code 7-Docker Daemon einrichten.....	37
Code 8-Docker daemon.json	37
Code 9-Dockerfile	38
Code 10-Dockerfile aufbauen und Images auflisten	39
Code 11-Benchmark: CPU Bash-Script	42
Code 12-Benchmark :RAM Bash-Script.....	48
Code 13-Network-Performance Bash-Script	51
Code 14-Startzeit Bash-Script.....	55

Abkürzungsverzeichnis

VM	Virtual Machine
AWS	Amazon Web Services
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 Machine Manager
UnionFS	Union Filesystem
OSCP	Open Source Community Project
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
MB/sec	Megabyte pro Sekunde

Glossar

Begriffe	Definition/Erklärung
Sandbox	Sandbox ist ein isolierter Umgebungsbereich, die von anderen Bereichen abgeschottet ist. Dadurch lassen sich beispielsweise Konflikte zwischen Betriebssysteme und Software durch die isolierte Umgebung vermeiden (vgl. Vogel 2018).
runc	Runc ist ein CLI Tool mit dem man Container Spawnen ausführen kann (vgl. Github 2020a).
runtime	Der Container Runtime ist eine Software, die für das Ausführen und Verwalten eines Containers auf einem Knoten zuständig ist (Lou und Brown 2017).
vSock	Der VSock ist für die Erleichterung der Kommunikation zwischen einer virtuellen Maschine und Host-Betriebssystem zuständig (vgl. Man7 2020).
Storage	Der Storage ist eine Art Speicherlösung, um Daten temporär bzw. dauerhaft aufzubewahren (vgl. Floyd und Dr. Bergler 2017).
UnionsFS	Mithilfe von UnionFS lassen sich für Schnittstellen mehrere gestapelte Dateisysteme bereitstellen (vgl. Linux Magazin 2005).
Agnostisch	„Unter Agnostisch versteht man, dass etwas soweit verallgemeinert wurde, dass es auch unter verschiedenen Umgebungen eingesetzt werden kann“ (Rouse M. 2020).

POD	„eine Gruppe von einem oder mehreren Containern, die sich Storage und Netzwerk teilen“ (Datacenter Insider 2019).
QEMU	„QEMU is a generic and open source machine & userspace emulator and virtualizer“ (GitHub 2020g).
sysstat	Mit dem Tool Sysstat können verschiedene Systemleistungen überwacht werden (vgl. Packages Debian SPI Inc. 2020a).
nicstat	Das Netzwerk kann durch den Befehl nicstat überwacht werden (vgl. Packages Debian SPI Inc. 2020b).
iftop	Mit dem Tool iftop kann das Netzwerkverkehr überwacht werden, um Schluss zu folgern wie die aktuelle Bandbreitnutzung erfolgt (vgl. Packages Debian SPI Inc. 2020c).
wget	Mit wget ist es innerhalb eines Terminals möglich HTTP oder HTTPS Server Dateien Herunterzuladen (vgl. Wiki Ubuntuusers 2020a).

Inhaltsverzeichnis

Abbildungsverzeichnis	13
Tabellenverzeichnis	14
Codeverzeichnis	15
Abkürzungsverzeichnis.....	16
Glossar.....	17
1 Einleitung	13
1.1 Problemendarstellung	14
1.2 Zielsetzung.....	14
1.3 Aufgabenstellung	14
1.4 Aufbau der Arbeit.....	15
2 Theoretische Grundlagen	16
2.1.1 Virtualisierung.....	16
2.1.2 Containerbasierte Virtualisierung	16
2.1.3 Vergleich von Containern und virtuellen Maschinen	17
2.2 Docker-Container	19
2.2.1 Docker CLI.....	21
2.2.2 Dockerfile.....	22
2.3 Firecracker Containerd	23
2.3.1 Firecracker funktionsweise	24
2.3.2 Firecracker Containerd Architektur	25
2.4 Kata Container	26
2.4.1 Kata Container Architektur	28
3 Test-Setup	30
3.1 Übersicht der durchzuführenden Teste.....	30
4 Installation der Containerlösungen	31
4.1 Docker installieren.....	31
4.1.1 Docker starten	32

4.2	Firecracker Containerd installieren	32
4.2.1	Firecracker starten	34
4.3	Kata Container installieren	35
4.3.1	Kata Container starten	37
4.4	Vorbereitung von Container Image	37
5	Evaluation der Container	40
5.1	Durchführung von CPU-Test.....	42
5.1.1	Ergebnisse von CPU Test.....	43
5.2	Durchführung von RAM-Test	48
5.2.1	Ergebnisse RAM Test	49
5.3	Durchführung von Network-Performance Test	51
5.3.1	Ergebnisse von Network-Performance Test.....	52
5.4	Durchführung von Startzeit-Test	55
5.4.1	Ergebnisse von Startzeit-Test.....	56
6	Zusammenfassung.....	59
7	Ausblick.....	61
8	Anhang	68
9	Ehrenwörtliche Erklärung	78

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 heute 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 die fast jedes Unternehmen in der jetzigen Zeit redet, sind die Virtualisierungstechniken. Immer mehr Unternehmen möchten sich in diesem Bereich weiter entwickeln, weil das Interesse immer mehr ansteigt. 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 auf die neue Virtualisierungsmethode 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 sich zurzeit noch in der Gründungsphase befindet, die Entwickler haben sich hierbei stark auf der Workload-Isolation und Sicherheitsvorteile von Containern fokussiert (vgl. Kata Container 2020a). Einer der neuesten und interessantesten Container wurde von einer der größten Unternehmen der Welt Amazone Webservices Veröffentlichung, hierbei handelt es sich um den Firecracker Containerd der im Jahr 2018 von AWS selbst entwickelt wurde. Der Firecracker soll wie auch der Kata Container hohe Workload-Isolation anbieten und gleichzeitig die Geschwindigkeit und Ressourceneffizienz von Containern verbessern (vgl. Firecracker Microvm 2020). Diese Arbeit beschäftigt sich mit der Leistungsbewertung von Containerlösungen und deren Technologische Anwendungsarten.

1.1 Problemdarstellung

Für viele Unternehmen die auf Schnelligkeit, Sicherheit und kostengünstige Virtualisierung Wert legen ist es besonders wichtig, wie die Leistungen der unterschiedlichen Container zu bewerten ist. Diese Arbeit befasst sich mit der Installation und Durchführung von verschiedenen Tests, um die Leistungen von Docker-Container mit den 2 neuartigen Containern, Firecracker Containerd und Kata Container zu vergleichen und anschließend zu bewerten. Zu der Durchführung werden noch wichtige Technologische Kenntnisse, die man benötigt erklärt.

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 Performance unterscheiden. Es soll mithilfe von Grafiken veranschaulicht werden wie die CPU-Geschwindigkeit, die RAM Geschwindigkeit, die Network Performance und Startzeiten sich voneinander unterscheiden. Aber nicht nur die Leistungen der Containerlösungen sollen untersucht werden, sondern auch die wichtigen Technologischen Aspekte. Darin geht es unter anderem um die verschiedenen Architekturen und deren Technologischen Vorteilen.

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 sie in einer ausführbaren Umgebung mit einer Netzwerkverbindung laufen können. Nachdem diese installiert sind werden die 3 Container auf den gleichen Systemstand versetzt. Diese wird mithilfe einer Docker Image erreicht. Der Docker Image wird mithilfe eines Dockerfiles erzeugt und dann für den jeweiligen Container ausgeführt, damit alle 3 Technologien die gleichen Systemvoraussetzungen erfüllen. In den späteren Kapiteln wird diese näher erklärt.

Nachdem die Container alle auf den gleichen Stand sind, werden verschiedene Bash-Skripts verwendet, um den Test durchzuführen und die Ergebnisse anschließend in einer Datei zu transferieren. Diese Ergebnisse werden mithilfe von Python eingelesen, und zum Schluss dann geplottet, sodass die Ergebnisse in Grafiken dargestellt werden. Die Unterschiedlichen

Ergebnisse werden anschließend zusammengefasst und bewertet. Bestandteil der Arbeit ist es nicht die Gesamte Technologie der Containerlösungen zu erklären, sondern die Leistungen der Container zu Bewertungen und die benötigten wichtigen Technische Prozesse, die im Hintergrund passieren zu erklären.

1.4 Aufbau der Arbeit

In Kapitel 2 werden die benötigten Theoretischen Grundlagen dieser Arbeit behandelt.

In diesem Kapitel wird zunächst der Unterschied zwischen Virtualisierung und Container erklärt. Danach werden die Gemeinsamkeiten und die Unterschiedlichen Architekturen dargestellt und wichtige Komponenten erläutert.

In Kapitel 3 wird eine Übersicht der durchzuführenden Tests dargestellt.

In Kapitel 4 wird erklärt, wie man die unterschiedlichen Containerlösungen installiert.

In Kapitel 5 werden verschiedene Hardwarekomponenten der 3 Containerlösungen auf ihre Leistungsfähigkeiten überprüft und in Grafiken dargestellt und bewertet.

In Kapitel 6 erfolgt die Zusammenfassung der Ergebnisse und in Kapitel 6 der Ausblick.

2 Theoretische Grundlagen

Im Theoretischen Teil werden Grundwissen über die Virtualisierung und Container erklärt, sowie die Unterschiede als auch die Gemeinsamkeiten. Danach erfolgt für Docker Container, Firecracker und Kata Container ein Überblick über die Architektur und wichtige Technologische Prozesse, die im Hintergrund der Container passieren. Als eine Überleitung erfolgt im nächsten Kapitel die Installation der Containerlösungen.

2.1.1 Virtualisierung

Schon am Ende der 1960er-Jahre wurde in einem Großrechner Virtualisierungstechniken verwendet, damit mehrere Benutzer auf einem System gleichzeitig arbeiten können. 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, Speicherressourcen 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 der virtuellen Maschinen zuständig. Im Prinzip sind virtuelle Maschinen, Systeme, die Hardware-Ressourcen benötigen, um zu laufen. Diese IT-Ressourcen werden vom Hypervisor auf die jeweiligen virtuellen Maschinen partitioniert, sodass mehrere virtuelle Maschinen gleichzeitig auf einem System laufen können (vgl. redhat Inc. 2020a).

2.1.2 Containerbasierte Virtualisierung

Containerbasierte Virtualisierung gilt als Leichtgewichte Alternative zu herkömmlichen virtuellen Maschinen, weil bei der Erstellung und Ausführung viel weniger IT-Ressourcen benötigt wird. Bei Virtuellen Maschinen werden die Ressourcen vollständig vom Hypervisor abgebildet wobei die Container im Gegensatz zu virtuellen Maschinen nur ein Abbild der benötigten Betriebssysteme und deren Funktionen darstellt (vgl. Docker Inc. 2020).

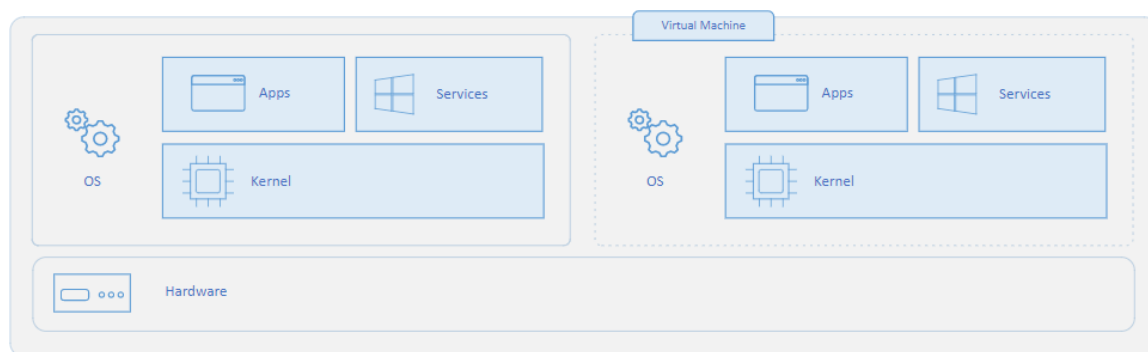
Damit ist ein Container im Prinzip nichts anderes als eine Software die Quellcode und all seine Abhängigkeiten zusammenbringt, damit die Anwendung schneller und zuverlässiger ausgeführt werden kann. Ein Container wird mithilfe einer Image-Datei aufgebaut, diese beinhaltet

alle erforderlichen Anwendungen wie Code, Systemtools und Systembibliotheken, sodass der Container eigenständig laufen kann (vgl. Docker Inc. 2020). Im Späteren Verlauf werden wir nochmal detaillierter drauf eingehen was die Image Datei genau beinhaltet und wie die Syntax zu verstehen ist.

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).

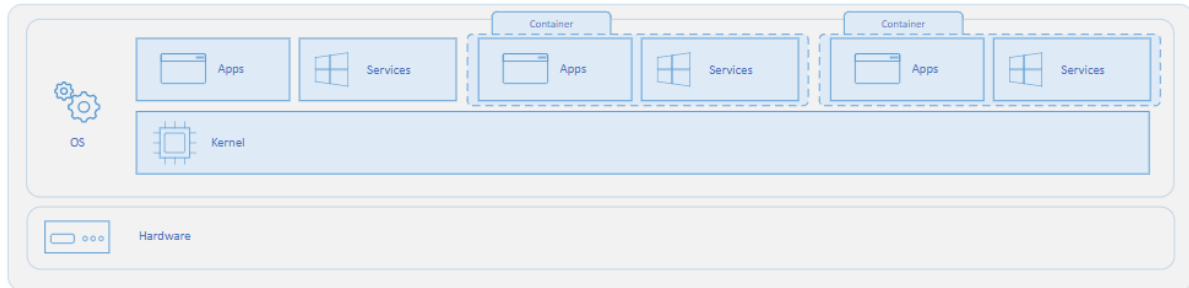


Quelle: (Docs Microsoft Inc. 2020)

Abbildung 1-Architektur von virtueller Maschine

Abbildung 1 zeigt das im Gegensatz zu einem Container eine Virtuelle Maschine die ein vollständiges Betriebssystem mit samt seine Komponenten abbildet (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 einer virtuellen 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 Microsoft Inc. 2020)

Abbildung 2-Architektur von Container

Abbildung 2 zeigt mehreren Containern, worauf eine Anwendung auf dem Hostbetriebssystem ausgeführt wird. Der Container baut auf dem Kernel des Hostbetriebssystems auf und enthält verschiedene Apps, APIs, und verschiedene andere Prozesse für das Betriebssystem (vgl. Docs Microsoft 2019). Bei der Ausführung von Containern wird der Linux Kernel und seine Funktionen verwendet, um Prozesse voneinander zu isolieren, damit diese voneinander unabhängig laufen können (vgl. redhat Inc. 2020b). Mithilfe der Isolation kann der Zugriff von mehreren Containern auf dasselbe Kernel erfolgen. Es lässt sich dadurch bestimmen, 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 an, 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 virtuelle Maschine werden virtuelle Netzwerkdapter erzeugt.	Genauso wie bei virtuellen Maschinen werden virtuelle Netzwerkdapter für jede einzelne Container erzeugt.

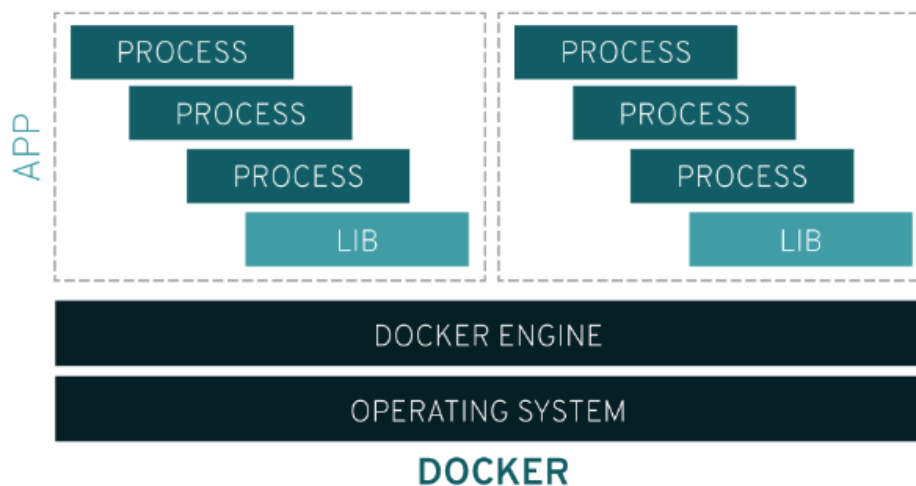
Quelle: (vgl. Docs Microsoft Inc. 2020)

Tabelle 1-Unterschied von Container und virtuelle Maschine

2.2 Docker-Container

Docker Container ist ein Open-Source-Project, somit können alle egal ob es sich um einen privaten Anwender oder ein Unternehmen handelt diese kostenlos installieren und verwenden. Docker Container können verschiedene Apps und APIs beinhalten und ausgeführt werden. Beim Ausführen eines Docker Containers 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 wird in einem 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 den Technischen Hintergrund von Docker zu verstehen gibt es drei wichtige Bestandteile, die von Docker Container verwendet wird. 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 Container 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 vollständiges Betriebssystem enthält, ist ein Docker Image viel kleiner als eine virtuelle Maschine und ist dadurch beim Starten viel schneller als eine virtuelle Maschine (vgl. Entwickler 2019).



Quelle: (redhat Inc. 2020)

Abbildung 3-Architektur von Docker Container

Abbildung 3 zeigt die Architektur von Docker Container. Der Docker Engine ist dafür zuständig, dass ein Zugriff auf dem Kernel des Host-Betriebssystems möglich ist, zusätzlich wird der Docker Engine benötigt, um ein Docker Container zu erstellen, zu starten oder 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 durch den Dockerfile abbilden (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 Beispielsweise mithilfe einer Docker Image ein Container laufen zu lassen. In dieser Dokumentation werden sowohl Docker Befehle erklärt als auch die praktische Umsetzung. Ein besonderer wichtiger Befehl ist beispielsweise 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. Für eine Internetverbindung wird in der Regel für jede Container Standardmäßig die Automatische Netzwerkbrücke aktiviert. Bei der Erstellung der Netzwerkbrücke wird vom Host-System zum Container eine Netzwerkbrücke aufgebaut, sodass man über das Host Netzwerk eine Internetverbindung herstellen 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 später für die Leistungsbewertung benötigt wird.

Docker Befehl	Bedeutung
<code>docker attach</code>	Attach gibt die Standard Eingabe, Ausgabe, und Fehler-Datenströme eines laufenden Containers.
<code>docker build</code>	Baut mithilfe des Dockerfiles eine Image Datei auf.
<code>docker commit</code>	Erstellt aus den Änderungen eines Containers eine neue Image Datei.
<code>docker exec</code>	Das Ausführen eines Befehls innerhalb des Containers.
<code>docker images</code>	Die Docker Images auflisten.
<code>docker kill</code>	Eine oder mehrere laufende Container killen.
<code>docker login</code>	Einloggen in ein Docker Register.
<code>docker logout</code>	Ausloggen von einem Docker Register.

docker network	Verwalten des Netzwerkes.
docker pause	Pausieren von allen Prozessen 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 mehreren Containern.
docker rm	Entfernen von einem oder mehreren Containern.
docker rmi	Entfernen von einem oder mehreren Images.
docker run	Ausführen von Befehlen in einem neuen Container.
docker stop	Stoppen von einem oder mehreren laufenden Containern.

Quelle: (vgl Docs. Docker 2020b)

Tabelle 2-Docker CLI

2.2.2 Dockerfile

Das Dockerfile ist nichts anderes als eine Bauanleitung der benötigt wird, um den Docker Image aufzubauen. Der Dockerfile beinhaltet für den Aufbau der Image Datei verschiedene Linux Befehle, die dann beim Erstellen des Docker Images durchgeführt werden (vgl. Entwickler 2019). Das Format des Dockerfiles besteht aus INSTRUCTION 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 folgenden Tabelle werden wichtige Anweisungen beschrieben die in einem Dockerfile benötigt wird, um daraus dann eine Docker Image aufzubauen.

INSTRUCTION	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.

Quelle: (vgl. anecon, 2018)

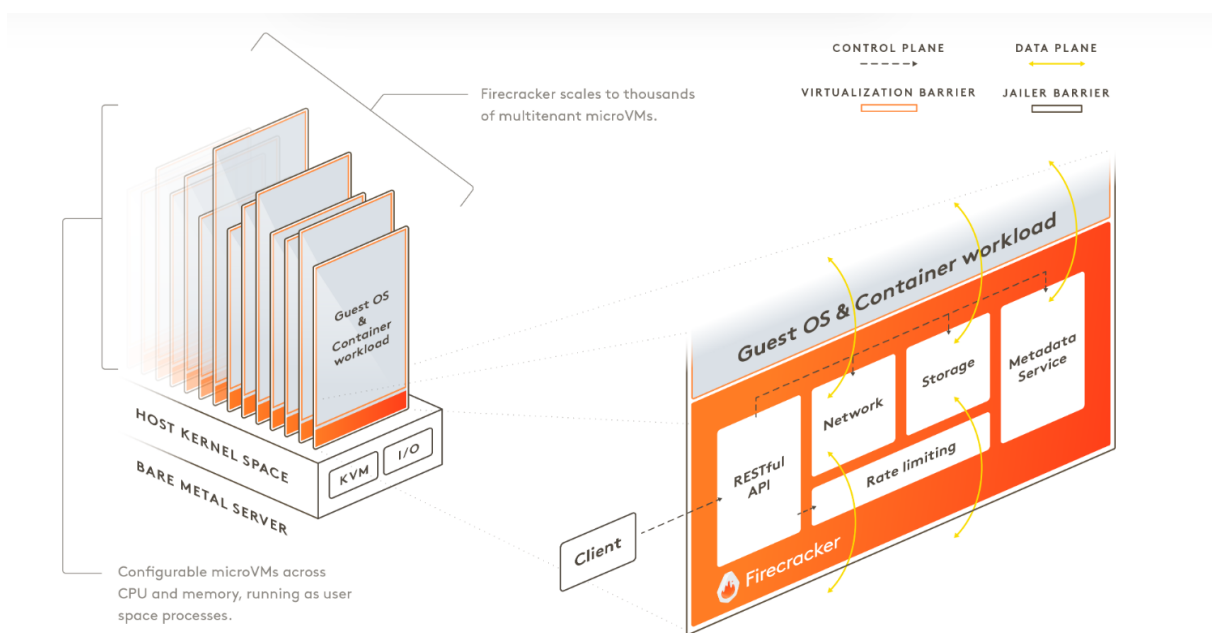
Tabelle 3-Dockerfile-Anweisungen

2.3 Firecracker Containerd

Firecracker ist ein sehr neues Projekt, dass vor knapp 2 Jahren von AWS veröffentlicht worden ist. Die Virtualisierungstechnik wird Open-Source angeboten. Die Geschwindigkeit, Ressourceneffizienz und Leistungen der Container sollen mit VMs kombiniert werden und dadurch die Sicherheit und Performance der Container erhöht werden. Firecracker benutzen sogenannte MicroVMs. Sie bieten im Gegensatz zu VMs eine höhere Sicherheit und Workload-

Isolation an. Zudem sollen MicroVMs gleichzeitig, wie normale Container eine hohe Geschwindigkeit und Ressourceneffizienz erzielen (vgl. Firecracker Microvm 2020). Außerdem wird eine reduzierter Speicherverbrauch und Sandboxing-Umgebung für jede MicroVM eingerichtet (vgl. Firecracker Microvm 2020). Dadurch werden Anwendungen nicht direkt auf dem Hostbetriebssystem ausgeführt, sondern nur in der eigenen Umgebung (vgl. Cloud Google). In dem nächsten Kapitel wird die Funktionsweise des Firecrackers aufgeklärt und wichtige Komponente werden erläutert.

2.3.1 Firecracker funktionsweise



Quelle: (Firecracker Microvm 2020)

Abbildung 4-Firecracker Architektur

Abbildung 4 zeigt wie Firecracker funktionsweise arbeitet. Es wird die KVM verwendet, um ein MicroVMs zu erstellen, der erstellte MicroVMs wird im Benutzerbereich ausgeführt. Da ein MicroVMs beim Starten ein geringer Speicheraufwand benötigt, können mehrere bzw. hunderte oder Tausende MicroVMs erstellt und ausgeführt werden. Die erstellten MicroVMs werden in Containergruppen innerhalb einer virtuellen Maschine mit einer Barriere eingekapselt. Mithilfe dieser Funktionen können Workloads auf derselben Maschine ausgeführt werden, ohne sich über die Sicherheit und Effizienz Gedanken zu machen und beliebige Gastbetriebssystem gehostet werden (vgl. Firecracker MicroVm 2020). Firecracker soll einen VMM basierend auf dem KVM implementieren und dadurch ein RESTful API anbieten, um MicroVMs zu erstellen

und zu verwalten. Die vCPU kann unabhängig ihrer Anwendungsanforderungen in beliebigen Kombinationen mit MicroVMs erstellt werden (vgl. Firecracker Microvm 2020).

Um verschiedene Netzwerke und Ressourcen, wie beispielsweise Speicher zu steuern gibt es sogenannte Rate Limiting, diese können über die Firecracker API erstellt und konfiguriert werden. Außerdem gibt es noch eine Metadata 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 erstellt und konfiguriert werden. Um die Sicherheit von MicroVM zu erhöhen hat man ein Supporting Programm 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 Virtualisierungsbarrriere durchbrochen wird (vgl. Firecracker Microvm 2020).

2.3.2 Firecracker Containerd Architektur

In diesem Kapitel 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 laufen, sodass die Anwendungen in einer festgelegten Weise zusammenarbeiten können. So kann der Benutzer beispielweise mehrere Anwendungen gleichzeitig starten und stoppen (vgl. HPE 2020).

2.3.2.2 Agent

Der Agent wird innerhalb von Firecracker MicroVM benötigt, um eine Verbindung mit runc aufzubauen und dadurch dann ein Container zu erstellen. Er kommuniziert außerhalb der VM mit dem Runtime über die vsock (vgl. GitHub 2018).

2.3.2.3 Virtual Maschine Manager

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

2.3.2.4 Snapshotter

Snapshotter ist für die Bereitstellung von Layer Storage und UnionFS für Containerd Container zuständig (vgl. GitHub 2019).

2.3.2.5 V2 runtime

Der V2-runtime ist für die Ausführung von Container Prozessen zuständig, indem sie die benötigten Konfigurationen und Implementationen des Containers bereitstellt (vgl. GitHub 2019).

2.3.2.6 Control Plugin

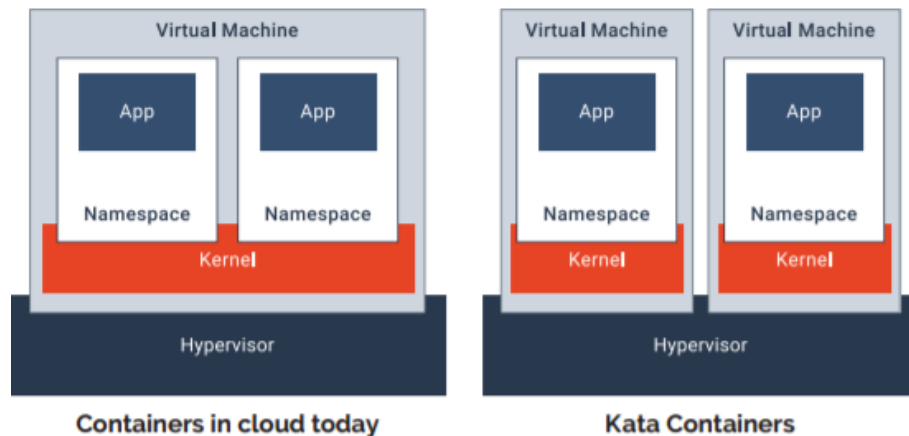
Der Control Plugin ist für die Implementierung von API und Verwaltung des Lebenszyklus von Runtime zuständig (vgl. GitHub 2019).

2.4 Kata Container

Kata Container verhält sich wie ein herkömmlicher Container, mit dem Unterschied, dass besonders viel Wert auf die Sicherheit und Workload-Isolation geachtet wird. Gleichzeitig sollen auch die Sicherheitsvorteile von VMs erfüllt werden. 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 werden und bei der Entwicklung mitwirken. Da es sich um ein OSCP Entwicklung handelt, ist Kata-Container drauf angewiesen das freiwillige bei diesem Projekt auch mitarbeiten (vgl. Kata Container 2020a).

Der Kata Container Projekt wird aus 2 verschiedenen Projekten vereinigt, einmal der Intel Clear Container Project und Hyper runV Project. Man hat die Technologien dieser beiden Projekte

kombiniert und daraus entstand das Kata-Container Project (vgl. Kata Container 2020a). Der Intel Clear Container Project 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 sich bei Kata-Container auf die Unterstützung von Technologische-Agnostik fokussiert. Durch diese Zusammenführung bietet der Kata-Container eine Kompatible Leistungsfähige Technologie an (vgl. OpenStack Foundation 2017).

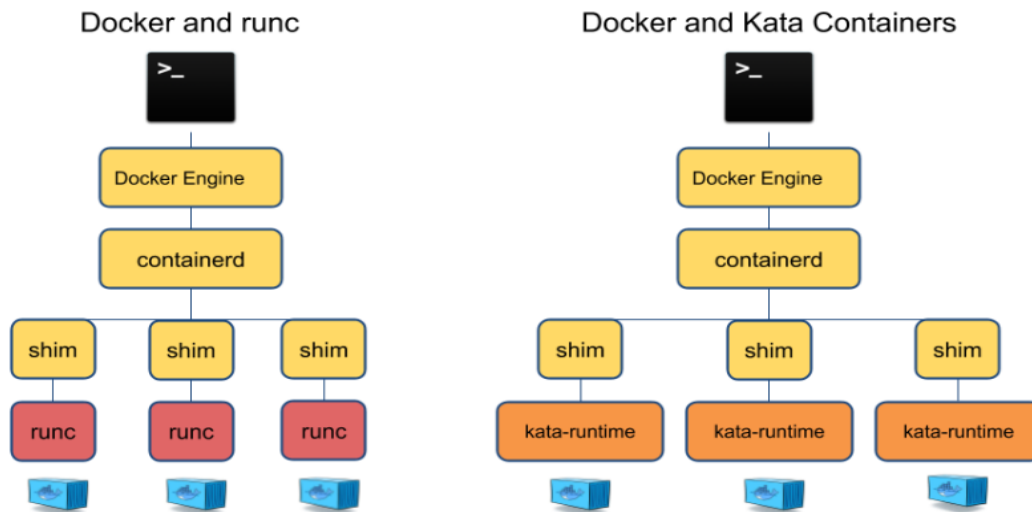


Quelle: (OpenStack Foundation 2017)

Abbildung 5-Unterschied zwischen Container und Kata Container

Man sieht bei der Abbildung 6, dass sowohl der Container in der Cloud als auch der Kata Container mit dem Hypervisor laufen. Der einzige Unterschied besteht darin, dass bei Kata Container die verschiedenen Apps nicht auf derselben virtuellen Maschine ausgeführt werden, sondern jede App auf seine eigene virtuelle Maschine läuft. So sind die verschiedenen Anwendungen voneinander isoliert. Bei normalen Containern laufen alle Apps auf derselben virtuellen Maschine. Diese Lösung soll die Sicherheit, Skalierbarkeit und Ressourcenauslastung von Kata Container im Gegenzug zu normalem Container enorm steigern. (vgl. OpenStack Foundation 2017).

2.4.1 Kata Container Architektur



Quelle: (GitHub 2020b)

Abbildung 6-Docker und Kata Containers

Die Abbildung 7 zeigt den Aufbau von Docker Container und Kata Container. Der Unterschied dieser zwei Strukturen ist die Laufzeitumgebung. Bei Docker Container wird dafür runc verwendet und für Kata Container der kata-runtime. Der kata-runtime sollte genauso wie runc mit Docker Engine kompatibel sein und einwandfrei laufen. Der kata-runtime erstellt für jede Container ein QEMU KVM Maschine, der dann für das Kata Container läuft. Der Kata-Shim ist für die Erstellung von POD Sandbox zuständig, diese wird für jede Container erstellt (vgl. GitHub 2020b). In den Nächsten Kapiteln wird der Kata-Shim und andere wichtige Komponente erklärt.

2.4.1.1 Guest Assests

Eine virtuelle Maschine wird mit einem minimalen Gast-Kernel und Gast-Image, 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-Image 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 Gegensatz zu dem Rootfs Image besteht der Initird aus einem komprimierten cpio-Archiv, das als Linux-Startprozess verwendet wird.

Während des Startvorgangs, wird vom Kernel eine Spezielle tmpfs entpackt und dadurch dann ein Root Dateisystem erzeugt (vgl. GitHub 2020b).

2.4.1.2 Kata-Agent

Der Kata-Agent läuft in einem Gast als Prozess und ist für die Verwaltung von Containern zuständig. Zur wichtigen Ausführungseinheit für Sandboxing gehört der Kata-Agent dazu. Diese definiert verschiedene Namespaces wie Beispielsweise PID oder IPC (vgl. GitHub 2020b).

2.4.1.3 Kata-Runtime

Der Kata-Runtime nutzt das Virtcontainers Project, die eine Agnostische und Hardware-Virtualisierte Container Bibliotheken für Kata-Container bereitstellt (vgl. GitHub 2020b). Es gibt die Datei `configuration.toml` die automatisch bei der Installation des Kata Containers erzeugt wird, in dieser Datei werden verschiedene Pfade festgelegt. Beispielsweise muss man den Pfad angeben, wo sich genau der Hypervisor oder die Image Datei befindet (vgl. GitHub 2020b).

2.4.1.4 Kata-Proxy

Der Kata-Proxy hat zwei wesentliche Aufgaben zu erledigen, zum einen ist es dafür da um Kata-Runtime und Kata-Shim die mit virtuellen Maschinen fungieren Zugriff auf den Kata-Agent zu geben und zum anderen hat es die Hauptaufgabe zwischen Kata-Shim und Kata-Runtime Instanzen E/A-ströme und Signale weiterzuleiten (vgl. GitHub 2020b)

2.4.1.5 Shim

Der Container Prozess Reaper ist eigentlich für die Überwachung von Containerprozesse zuständig. Das Problem ist, das dieser Prozess auf dem Host läuft und somit die Überwachung eines Prozesses die in einer VM nicht möglich ist. Deswegen gibt es als Ersatz das Kata-Shim, mithilfe dieses Prozesses ist es mögliche Containerprozesse zu überwachen. Es werden alle E/A-Ströme verarbeitet und Signale an Reaper weitergeleitet, dadurch ist die Überwachung von Containerprozessen wieder möglich (vgl. GitHub 2020b).

3 Test-Setup

In diesem Kapitel wird ein Überblick über die durchzuführenden Tests dargestellt.

3.1 Übersicht der durchzuführenden Teste

Es gibt unzählige Tests, die man durchführen kann. Bei Containern liegt man wie schon davor erwähnt sehr viel Wert auf Leichtgewichtigkeit. Da sowohl Docker Container, Firecracker und Kata Container auf die Leichtgewichtigkeit sehr viel Wert drauflegen, ist es interessant sich die CPU, RAM, Startzeiten und Network Performance dieser Containerlösungen näher anzuschauen.

Alle Unternehmen möchten flexibel und kostengünstig arbeiten, deswegen spielt gerade diese Komponenten eine sehr große Rolle, wenn es um Zeit und Geld geht. Die durchgeführten Tests sollen folgende Ergebnisse liefern:

- CPU: Wie viele Sekunden benötigt ein Container, um die Primzahlen zwischen 1 und 20.000 auszurechnen?
- CPU: Wie schnell können die Container Rechenoperationen pro Sekunde ausführen?
- RAM: Wie viel Megabyte kann pro Sekunde im Hauptspeicher geschrieben werden?
- Network-Performance: Wie viel Megabyte kann ein Container pro Sekunde Herunterladen?
- Startzeiten: Wie lange braucht ein Container, um ein Befehl auszuführen?

Diese Tests werden mithilfe von verschiedenen Bash Skripte durchgeführt, dann in einem Programm eingelesen und in Grafiken dargestellt. Mithilfe dieser Grafiken können die Ergebnisse interpretiert werden.

4 Installation der Containerlösungen

In diesem Kapitel wird zunächst erklärt, wie der Docker Container installiert wird, danach der Firecracker Containerd und anschließend dann der Kata Container.

4.1 Docker installieren

Die Anleitung wie man Docker installiert findet man auf der Offiziellen Docker Seite (siehe. Docs. Docker 2020d). In diesem Kapitel wird die Installation von Docker Container abstrakt erklärt. Die genaue Installation findet man entweder auf der Offiziellen Docker Seite oder im Anhang A1.

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

Um Docker zu installieren werden wie schon im Anhang A1 zusehen die älteren Versionen von Docker entfernt und danach die neuste Version installiert. Aus Sicherheitsgründen wird von Docker ein GPG heruntergeladen und diese dann mit einem Fingerprint Befehl ausgegeben. Die ausgegebene Key wird mit dem Key der Docker Seite verglichen. Wenn diese übereinstimmt kann danach der Repository von Docker heruntergeladen werden. Nachdem der Docker Repository auf unser System hinzugefügt wird kann man Docker installieren. Bei der Installation von Docker ist die Auswahl von Docker Engine Version besonders wichtig. Der Docker Engine wird im Kapitel 2.2 ausführlich erklärt. Wenn die richtige Version nicht ausgewählt wird, dann kann auch der Docker Container auf dem System nicht problemlos laufen. Beispielsweise wird im Anhang A1 die Version 18.06.0~ce~3-0~ubuntu ausgewählt. Nachdem der Docker Engine erfolgreich installiert wird kann man mit den Befehlen aus der Tabelle 2 beispielsweise ein Container starten, stoppen oder eliminieren.

Wenn beim Starten eines Containers Probleme auftauchen, dann liegt es meistens an dem Docker Engine Daemon. Diese kann man mit den Befehlen `sudo service docker status/start/stop` starten, stoppen oder den Status abfragen, um das Problem zu analysieren.

4.1.1 Docker starten

Um ein Container zu starten wird der Docker Befehl `run` verwendet.

```
$ docker run -it -d debian-test
b78f6ce04028a3db8ab5c0a87d02388c3272bbc2e1e6d9725196703131a7ea16
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b78f6ce04028	debian-test	"bash"	7 seconds ago	Up 4 seconds		bold_poincare

Quelle: (Docs. Docker 2020d)

Code 1-Container ausführen

Der Docker Befehl `docker run -it -d debian-test`, macht die Ausführung eines Containers im Hintergrund möglich. Nachdem ein Container ausgeführt wird erhält es eine Container ID, diese Container ID wird verwendet, um beispielsweise mit dem Docker Befehl ein Container zu `stoppen` oder zu `starten`. Die Container IDs können mit dem Befehl `docker ps` aufgelistet werden. Wenn irgendwelche Änderungen innerhalb des erstellten Docker Container stattfinden, kann man mit dem `docker commit` Befehl eine neue Image Datei aus dem Docker Container erstellen. Der Befehl beinhaltet die ID des Containers und Name des neuen Images.

Zusammenfassend ist die Installation von Docker Container hiermit erfolgreich absolviert.

4.2 Firecracker Containerd installieren

Für die Installation von Firecracker-Containerd wird der offizielle Firecracker GitHub Repositories verwendet (siehe. GitHub 2020c und GitHub 2020d). Für Firecracker wird auch die Installation von Docker Container vorausgesetzt, weil eine Image Datei verwendet wird, um den Firecracker Containerd auszuführen. Im Späteren Verlauf wird erklärt was ein Docker Image ist und wie diese erstellt wird. Wie auch bei Docker Container wird die Installation nur abstrakt erklärt im Anhang A2 wird die genaue Installation durchgeführt.

Bevor die richtige Installation von Firecracker beginnt, muss die Hardware überprüft werden, damit man weiß ob die Systemvoraussetzungen für die Virtualisierung von Firecracker erfüllt ist. Als Ergebnis sollten folgende Zeilen im Terminal ausgegeben werden: `Your system looks ready for Firecracker!`

Danach kann der Firecracker Repository von der GitHub Seite mit dem Befehl `git clone` heruntergeladen werden. Damit später eine Internetverbindung aufgebaut werden kann wird auch wie im Anhang A2 zusehen ist ein Demo Netzwerk installiert. Im Heruntergeladenen

Repository sollten folgende Binärdateien beinhaltet sein, um den Firecracker Containerd zu starten:

- runtime/containerd-shim-aws-firecracker
- firecracker-control/cmd/containerd/firecracker-containerd
- firecracker-control/cmd/containerd/firecracker-ctr

Jedes Betriebssystem benötigt ein Kernel, bei Firecracker Containerd wird ein sogenannte KVM wie auch im Kapitel 2.3.1 erklärt benötigt. Dies KVM wird von Amazone selbst bereitgestellt und kann kostenlos verwendet werden, diese wird auch wie im Anhang A2 zusehen ist installiert.

Damit auch der Firecracker Containerd mithilfe von Docker ausgeführt werden kann muss die API-Socket von Docker aktiviert werden.

Bevor man überhaupt ein Firecracker Containerd startet muss beim Docker die API-Socket mit den folgenden Zeilen aktiviert werden. Wenn der API-Socket nicht aktiviert ist, dann kann man auch den Firecracker Containerd nicht richtig starten.

```
[Service]
ExecStart=
ExecStart=/usr/bin/dockerd -H fd:// -H tcp://0.0.0.0:2376
```

Quelle: (Unix Stackexchange 2019)

Code 2-Docker API-Socket aktivieren

Die Zeilen aus dem Code 2 wird in der `startup_options.conf` hinzugefügt und anschließend der Docker Engine aktualisiert. Wie man den Docker Engine aktualisiert wird im Anhang A2 dargestellt.

```
disabled_plugins = ["cri"]
root = "/var/lib/firecracker-containerd/containerd"
state = "/run/firecracker-containerd"
[grpc]
  address = "/run/firecracker-containerd/containerd.sock"
[plugins]
  [plugins.devmapper]
    pool_name = "fc-dev-thinpool"
    base_image_size = "10GB"
    root_path = "/var/lib/firecracker-containerd/snapshotter/devmapper"
[debug]
  level = "debug"
```

Quelle: (GitHub 2020c)

Code 3-config.toml

Besonders wichtig bei Firecracker ist die Datei `config.toml`. Mithilfe dieser Datei werden die verschiedenen Speicherorte für wichtige Dateien festgelegt. Diese Dateien werden beim Ausführen der Firecracker Umgebung benötigt, damit diese einwandfrei laufen kann.

Diese Informationen kann man wie eine Art Datenbank für erstellte Containernd betrachten. Im State Anweisung wird beispielsweise der `containerd.sock` abgelegt, der Socket wird für das Starten eines Images benötigt. Die Adresse gibt den Pfad an, wo der `containerd.sock` abgelegt ist, damit diese dann auch verwendet werden kann. Zum Schluss werden Plugins für den Devmapper-Snapshotter angegeben, wie zum Beispiel der Name, die Größe des Images und wo diese Informationen abgelegt werden sollen.

4.2.1 Firecracker starten

Zum Starten von Firecracker Containerd werden 2 Terminals benötigt.

```
$ sudo PATH=$PATH ~/go/src/github.com/firecracker-containerd/firecracker-  
control/cmd/containerd/firecracker-containerd \  
--config /etc/containerd/config.toml
```

Quelle: (GitHub 2020d)

Code 4-Terminal 1 starten

In Kapitel 4.2 werden Binärdateien wie `firecracker-containerd` erwähnt. Diese Datei wird benötigt, um den Firecracker Umgebung zu starten. Diese muss parallel im Terminal 1 ausgeführt werden. Die Umgebung wird mithilfe der `config.toml` Datei im Code 3 durchgeführt.

```
$ sudo firecracker-ctr --address /run/firecracker-containerd/containerd.sock \  
run \  
--snapshotter devmapper \  
--runtime aws.firecracker \  
--rm --tty --net-host \  
docker.io/library/busybox:latest busybox-test
```

Quelle: (GitHub 2020d)

Code 5-Terminal 2 starten

Während im Terminal 1 die Umgebung von Firecracker Containerd realisiert wird kann im Terminal 2 ein oder mehrere Container mithilfe der Binärdatei `firecracker.ctr` gestartet werden. Firecracker-ctr stellt dem Benutzer wie die Docker Umgebung verschiedene Befehle zur Verfügung. Diese CTR Befehle sind zu Docker Befehle fast Äquivalent zu verstehen. Die Befehle von Docker und CTR sind Äquivalent zu verstehen, beispielsweise wird bei allen beiden

Containern der Befehl `run` verwendet, um ein Docker Image zu starten. Beim Starten des Containers werden noch die folgenden Optionen in der Tabelle ausgeführt.

CTR Options	Erklärung
<code>--snapshotter</code>	Hier wird der Name des Snapshots angegeben.
<code>--runtime</code>	Hier wird der Laufzeitname angegeben.
<code>--rm</code>	Nachdem Ausführen des Containers wird diese wieder gelöscht.
<code>--tty</code>	Der Container wird einem tty zugeordnet.
<code>--net-host</code>	Dadurch wird der Host-Netzwerk für den Container aktiviert.

Quelle: (vgl. Systutorials)

Tabelle 4-CTR-Options

In Code 5 ist der Name des Snapshots und der Name der Laufzeitumgebung benannt. Zusätzlich sind noch Optionen wie `--rm`, `--tty` und `--net-host` eingestellt.

Nachdem dieser Vorgang erfolgreich durchgeführt wird, ist man im Container eingeloggt.

Im Anhang A4 wird eine zweite Methode gezeigt, wie man Firecracker installiert und ausführt (siehe. GitHub 2020e). Diese Methode ist aber nicht der Bestandteil dieser Arbeit.

4.3 Kata Container installieren

Für die Installation von Kata Container wird die offizielle Kata Container GitHub Repository verwendet (siehe. GitHub 2020f). Der Kata Container läuft mithilfe von Docker Container Umgebung, deswegen ist die Installation von Docker Container einer der wichtigsten Voraussetzungen von Kata Container. Der Kapitel beschäftigt sich nicht detailliert mit der Installation von Kata Container, sondern sehr abstrakt. Die genaue Installation von Docker Container wird im Anhang A3 erklärt.

Ein besonders wichtiges Tool die zu erwähnen ist, ist der Goolang. Diese wird bei der Installation von Kata Container benötigt, um verschiedene Repositories herunterzuladen. Neben Goolang werden noch andere wichtige Tools wie im Anhang A5 zusehen sind installiert.

Zunächst wird mithilfe von Goolang das Kata Container Repository heruntergeladen. Danach wird im Ordner `runtime` die `Makefile` Datei aufgebaut. Mit diesem Vorgang wird der Repository Architektur realisiert und der Kata Container Repository wird erfolgreich installiert.

Nachdem der Kata Container Repository installiert wurde, ist zu überprüfen ob auch das System die benötigten Hardware Ressourcen verfügt, um ein Kata Container auszuführen. Wenn dies überprüft wird, können Container ausgeführt werden.

Beim Installieren des Repositories wird auch eine Datei mit dem Namen `configuration.toml` installiert. Diese Datei muss konfiguriert werden, damit der Hypervisor die benötigten Informationen suchen kann, um den Kata Container zu starten. Wie diese zu konfigurieren ist wird im nächsten Schritt erläutert.

```
[hypervisor.qemu]
path = "/usr/bin/qemu-system-x86_64"
kernel = "/usr/share/kata-containers/vmlinuz.container"
# initrd = "/usr/share/kata-containers/kata-containers-initrd.img"
image = "/usr/share/kata-containers/kata-containers.img"
machine_type = "pc"
```

Quelle: (Github 2020f)

Code 6-configuration.toml

In der Datei `configuration.toml` werden wichtige Speicherorte festgelegt, wie zum Beispiel der Pfad zum QEMU Datei oder das Kernel System und der Speicherort des verwendeten Images. Wichtig hier zu beachten ist, ob das Image oder Initrd für das Ausführen des Kata Container verwendet werden soll. In diesem Code Beispiel wird Beispielsweise die Image Datei ausgewählt und deswegen die Zeile Initrd mit `#` ausgeklammert.

Was besonders wichtig ist, ist die Aktivierung von Debug für Kata-Shim, diese kann man in der `config.toml` wie auch im Anhang A3 zusehen ist konfigurieren.

Nachdem diese beiden Funktionen erstellt sind, muss ein Rootfs wie auch bei Firecracker installiert werden.

Wie auch bei Firecracker benötigt der Kata Container ein Kernel. Das Kernel kann wie auch im Anhang A3 zusehen ist mithilfe des Repository `packaging` heruntergeladen werden.

Der Hypervisor ist einer der wichtigsten Bestandteile von Kata Container und wird auch wie im Anhang A3 zusehen ist installiert. Wie der Hypervisor genau funktioniert wird im Kapitel 2.1.1 detailliert erklärt.

4.3.1 Kata Container starten

```
$ sudo mkdir -p /etc/systemd/system/docker.service.d/  
$ cat <<EOF | sudo tee /etc/systemd/system/docker.service.d/kata-containers.conf  
[Service]  
ExecStart=  
ExecStart=/usr/bin/dockerd -D --add-runtime kata-runtime=/usr/bin/kata-runtime --  
default-runtime=kata-runtime  
EOF
```

Quelle: (GitHub 2020h)

Code 7-Docker Daemon einrichten

Damit der Kata Container über den Docker Daemon auch ordnungsgemäß läuft muss der Kata-
Runtime hinzugefügt werden. Diese wird mithilfe von Code 7 realisiert.

```
{  
  "default-runtime": "kata-runtime",  
  "runtimes": {  
    "kata-runtime": {  
      "path": "/usr/bin/kata-runtime"  
    }  
  }  
}
```

Quelle: (GitHub 2020h)

Code 8-Docker daemon.json

Anschließend müssen die Zeilen im Code 8 in der Datei `/etc/docker/daemon.json` hinzugefügt werden. Danach wird, wie auch im Anhang A3 zusehen ist der Docker Daemon aktualisiert. Nachdem der Daemon richtig eingerichtet wird kann der Kata Container gestartet werden.

Jetzt kann man mit dem einfachen Befehl `sudo docker run -ti --runtime kata-runtime debian-test sh` ein Kata Container ausführen.

4.4 Vorbereitung von Container Image

Bis zu diesem Kapitel sind die Containerlösungen erfolgreich installiert. Damit die Containerlösungen gleich fair bewertet werden können, wird ein Dockerfile Image erstellt, damit alle drei Containerlösungen auf einem gleichen Betriebssystem laufen können. Im nächsten Schritt wird ein Dockerfile erstellt und konfiguriert, um ein Docker Image für die Leistungsbewertung der drei Containerlösungen aufzubauen.

Zunächst muss der Dockerfile mit dem Befehl `sudo touch Dockerfile` erzeugt werden wird diese konfiguriert.

```
# 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
```

Quelle: (vgl. Docs. Docker 2020d)

Code 9-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 das Linux System Debian auf die 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, das erstellte Debian System mit den Befehlen `update` aktualisiert. Das installierte Paket `install`, wird benötigt, um andere verschiedene Linux Pakete zu installieren wie auch Beispielweise in Zeile 4 zusehen ist.

In den Zeilen 4-6 werden wichtige Tools mit der Anweisung `run` installiert, diese Tools werden eventuell für die Leistungsbewertung der Containerlösungen benötigt.

Von Zeile 9 -13 wird das Tool Benchmark installiert. Mit Benchmark hat man die Möglichkeit die Leistungsfähigkeit eines Systems zu überprüfen (vgl. Wiki Ubuntuusers 2020a). In der Arbeit wird Benchmark benötigt, um einzelne System Komponente wie Beispielsweise CPU auf ihre Leistungen mit verschiedenen Testdurchläufen zu bewerten.

Die Zeile 14-19 beschreibt wie Apache 2.4 installiert wird. Bei Apache 2.4 handelt es sich um einer der meistverwendeten Webserver im Internet (vgl. Wiki Ubuntuusers 2020a). Apache 2.4 wird später für die Bewertungen der Container Startseiten benötigt.

Nachdem der Dockerfile erstellt wird, kann man diese mit dem Befehl `docker build` aufbauen. Jetzt wird ein Docker Image erstellt, die Erstellung des Images ermöglicht es mit dem Docker Befehl `docker run` ein Container auszuführen.

```
$ docker build -t debian-test .
```

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
debian-test	latest	3417c1c607bc	5 seconds ago	352MB

Quelle: (vgl. Docs. Docker 2020d)

Code 10-Dockerfile aufbauen und Images auflisten

Mit dem Code 10 wird ausgedrückt, dass der Dockerfile im aktuellen Pfad verwendet werden soll, um ein Docker Image mit dem Namen `debian-test` aufzubauen. Der nächste Befehl `docker images` listet alle erstellten Images auf.

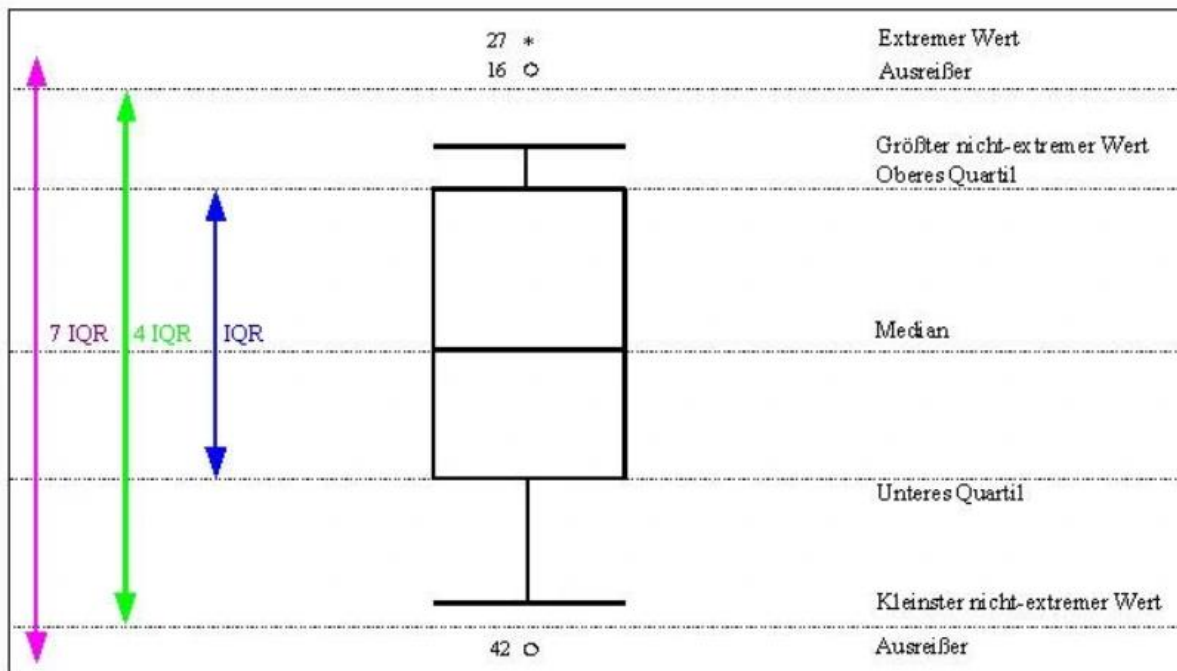
5 Evaluation der Container

In dem letzten Kapitel wird erläutert, welche verschiedenen Architekturen die Containerlösungen haben, welche Tests durchgeführt werden und wie man die verschiedenen Containerlösungen installiert. Für die Leistungsbewertung wird bei allen drei Containerarten die gleiche Docker Image aus Kapitel 4.4 verwendet. Diese Voraussetzung macht es uns möglich die Container in einem Äquivalenten Systemzustand zu bewerten. Für die Leistungsbewertung wird Benchmark als Haupt Werkzeug verwendet, um die verschiedenen Hardware Komponenten zu testen. Danach werden die einzelnen Startzeiten der Container und Network Performance auf ihre Leistungen bewertet. Die Tests erfolgen mithilfe von verschiedenen Bash Skripte. Die Ergebnisse der durchgeführten Tests werden anschließend mit Python Programme eingelesen und geplottet, damit diese in Grafiken dargestellt werden. Die Grafiken werden danach miteinander verglichen, analysiert und bewertet. Zudem werden noch die Standardabweichung, der Mittelwert, der Median und der Interquartilsabstand in einer Tabelle dargestellt und interpretiert.

Die ersten beiden Test Durchführungen beziehen sich auf CPU und RAM Leistung, um diese zu testen wird Benchmark verwendet. Benchmark ermöglicht es uns verschiedene Hardware Komponenten auf ihre Performance zu messen. In Benchmark gibt eine sogenannte Software namens `sysbench`, mit dieser Software kann man beispielsweise verschiedene CPU und RAM Tests durchführen (vgl. Webhosterwissen 2018).

Der dritte und der vierte Test beziehen sich auf die Leistung der Startzeiten und Network-Performance von Containern.

Der Startzeit-Test wird mithilfe von `time` und Benchmark durchgeführt und der Network-Performance Test wird mithilfe eines Tools namens `wget` realisiert. Alle 4 Test Durchführungen werden mit Bash Skripte ausgeführt. Die Ergebnisse werden wie schon vorhin erwähnt eingelesen und geplottet. Dies erfolgt mit der Python Bibliothek Matplotlib. Für die Mathematische Berechnungen wird die Bibliothek Numpy verwendet. Mithilfe dieser Bibliothek werden die Mittelwerte und Standardabweichungen der Ergebnisse berechnet. Für die Darstellung der Ergebnisse wird Box-Plot verwendet (siehe. Matplotlib 2020).



Quelle: (Marktforschung)

Abbildung 7-Box-Plot

Die Abbildung 8 zeigt eine verteilte Übersicht von Ergebnissen. Das obere Quartil liegt bei 75%, der Median bei 50% und das untere Quartil bei 25%. Außerhalb des kleinsten und größten Quartils liegen die Ausreißer Werte, das sind die Werte, die zu den anderen Daten sehr weit entfernt liegen. Meistens sind die Ausreißer 1,5-3 fach größer als die Box selbst (vgl. Marktforschung). An einem Beispiel wird die Interpretation der Box erklärt. Diese Interpretation gilt fortführend für alle Box-Plots in den nächsten Kapiteln. Es wird angenommen das der Median bei 50 liegt, das erste Quartil bei 10 und das dritte Quartil bei 100. Dies würde man dann folgendermaßen interpretieren:

Der Median liegt bei 50, dies bedeutet das 50% der Werte oberhalb der Linie und 50% der Werte unterhalb der Linie liegen. Das erste Quartil liegt bei 10, dies bedeutet 25% der Werte liegen oberhalb der Linie und 75% unterhalb dieser Linie. Das dritte Quartil liegt bei 100, dies bedeutet das 75% der Werte unterhalb von 100 liegen und 25% der Werte oberhalb dieser Linie. Der Interquartilsabstand liegt ca. bei 90 ($Q_3 - Q_1$), dies gibt den Abstand zwischen der Werter zwischen dem ersten und dritten Quartil.

5.1 Durchführung von CPU-Test

Der CPU Test wird bei allen drei Container innerhalb des Containers mit dem gleichen Docker Image durchgeführt. Für jede Containerart wird aus dem Docker Image ein Container gestartet und nacheinander der CPU Test durchgeführt.

```
#!/bash/bin!  
# CPU: Primzahlen-Test  
  
i=1  
while [ $i -le 25 ]  
do  
    sysbench --num-threads=1 --test=cpu --cpu-max-prime=20000 run | head -n15 | tail -  
n1 >> CPU_Test_fuer_Docker_Container.csv  
    sleep 30s  
    i=`expr $i + 1`  
done
```

Quelle: (Webhosterwissen 2018)

Code 11-Benchmark: CPU Bash-Script

In Code 11 wird ein `sysbench` Test durchgeführt, dabei wird die Anzahl der Cores mit der Option `--num-threads=` festgelegt. Da in allen Container nur ein Core verwendet wird, kann man auch maximal nur ein Core festlegen. Der `--test=` legt fest welche Art von Test es sein soll. In diesem Fall handelt es sich um ein CPU Test. Mit der Option `--cpu-max-prime=` legt man die Art des Testes fest, in diesem Fall handelt es sich um ein Primzahlen-Test und der maximale Wert wird auf 20.000 festgelegt. Dies bedeutet, dass der Container die Primzahlen zwischen 1 und 20.000 sucht und als Ergebnis wird in Zeile 15 ein totale: time Wert ausgegeben, dieser Wert sagt aus, wie lange der Container benötigt hat, um die Primzahlen zwischen 1 und 20.000 zu suchen. Der Wert 20.000 entspricht die Ausführung von insgesamt 321.238 Rechenoperationen, d.h. es werden beim Suchen der 20.000 Primzahlen 32.238 Rechenoperationen auf die CPU ausgeführt (vgl. Webhosterwissen 2018). Dieser Vorgang wird mithilfe der `while` Schleife 25-mal wiederholt und nach jedem Durchlauf mit dem `sleep` Befehl 30 Sekunden pausiert. Dadurch kann sich der Container nach jedem Durchlauf kurz erholen und genauere Ergebnisse liefern. Nach jedem Testdurchlauf wird das Ergebnis in ein CSV Datei hinterlegt, sodass man später diese Ergebnisse mithilfe von Python einlesen und plotten kann. Im Anhang A6 kann man die Resultate der Primzahlen Test für Docker Container, Firecracker Containerd und Kata Container einsehen.

5.1.1 Ergebnisse von CPU Test

Die CSV Datei wird im Anhang A7 mit der Funktion `open` geöffnet werden. Danach werden die Zeilen der CSV Datei gelesen und anschließend mit der Funktion `plt.boxplot` geplottet. Als Ausgabe werden die Ergebnisse geplottet und in einer Grafik dargestellt. Sowohl für den CPU-Test, RAM-Test, Network-Performance Test und Startzeit Test, werden diese Optionen verwendet. Die Programme sind Äquivalent aufgebaut nur mit dem Unterschied, dass die Ergebnisse anders sind.

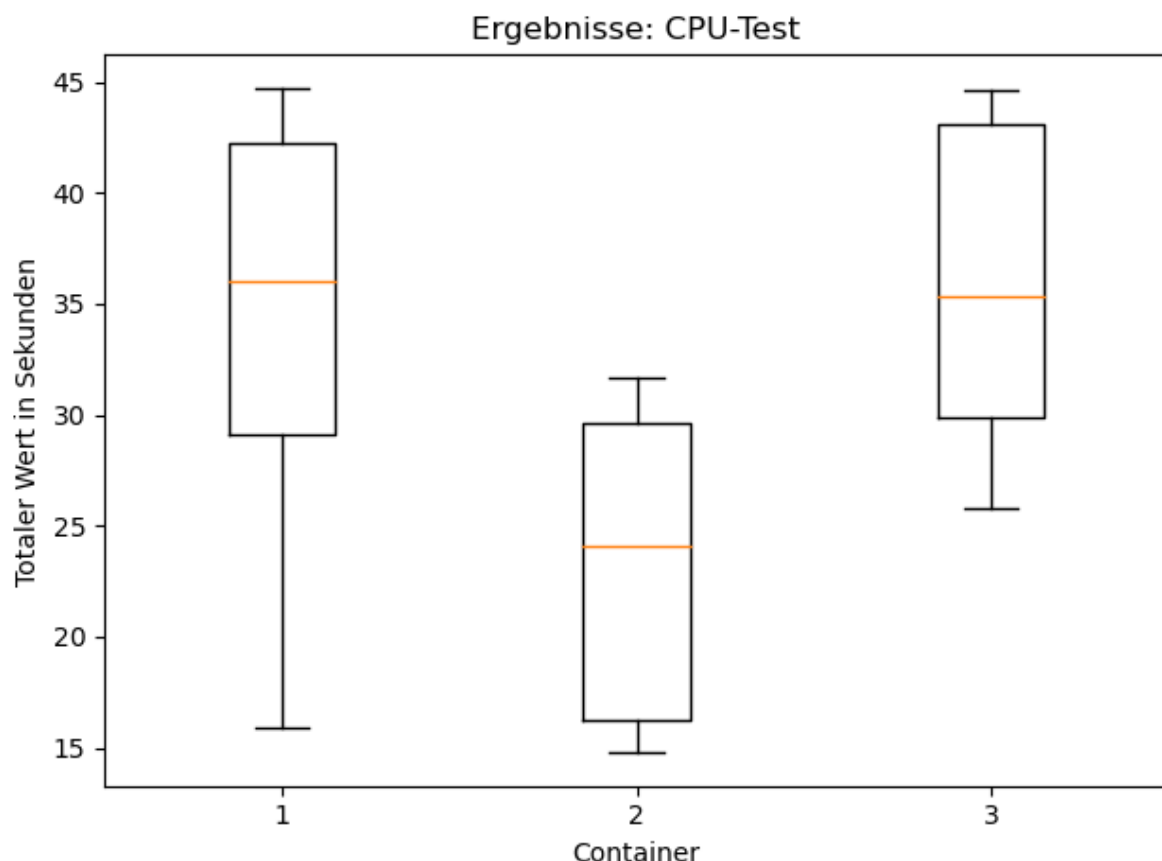


Abbildung 8-Grafik: Primzahlen-Test

Die Abbildung 9 zeigt die Ergebnisse von Primzahlen-Test.

Die x-Achse beschreibt die drei Container, x=1 soll Kata Container darstellen, x=2 Firecracker Container und x=3 Docker Container. Auf der y-Achse werden die Ergebnisse im Anhang A6 in Sekunden dargestellt. Wie schon in Abbildung 8 erklärt ist, ist der Box-Plot in unterschiedlichen Bereichen aufgeteilt und geben wichtige Informationen über die Ergebnisse der durchgeführten Tests. Die Innere Darstellung sind bei allen Box-Plot Ergebnissen im

späteren Kapitel gleich. Nur die y-Achse sind unterschiedlich beschriftet. Mithilfe der Abbildung 8 lassen sich folgende Ergebnisse in der Tabelle zusammenfassen:

Ergebnisse	Median (50%)	Quartil 1 (25%)	Quartil 3 (75%)	Interquartilsabstand
Kata Container	~35.94s	~28.99s	~42.35s	~13.36s
Firecracker Containerd	~24.08s	~16.33s	~26.61s	~10.28s
Docker Container	~35.31s	~29.97s	~43.24s	~13.27s

Tabelle 5-Ergebnisse: Primzahlen-Test

Der Median von Kata und Docker Container liegen bei beiden zwischen ~35.31s und ~35.84s, dies bedeutet, dass die Werte unterhalb und oberhalb der 50% sehr ähnlich aufgeteilt sind. Der Median von Firecracker liegt bei ~24.08s, das bedeutet, dass die Werte oberhalb und unterhalb der 50% sich sehr stark von Kata und Docker Container unterscheiden. Das erste Quartil und dritte Quartil bei Kata und Docker Container unterscheiden sich auch hier sehr minimal. Beim Kata Container beträgt das erste Quartil ~28.99s und bei Docker Container ~29.97s, dies zeigt das sich sowohl oberhalb als auch unterhalb des erstens Quartils minimal unterscheiden. Beim zweiten Quartil sind die Ergebnisse bei Kata Container ~42.35s und bei Docker Container ~43.24s, auch hier unterschieden sich die Ergebnisse kaum. Das erste und dritte Quartil von Firecracker unterscheidet sich auch hier sehr stark von Kata und Docker Container. Das untere Quartil liegt bei ca. ~16.33s und das obere bei 26.61s, wenn man diese mit den anderen Werten von Kata und Docker vergleicht, sieht man direkt, dass die Werte mit starkem Unterschied unterhalb liegen. Die Differenz von Q1 und Q3 ist der Interquartilsabstand, diese Ergebnisse geben Rückschlüsse wie nah die Daten innerhalb des Q1 und Q3 voneinander liegen.

Der Interquartilsabstand von Firecracker ist niedrig, deswegen liegen die Werte sehr wahrscheinlich nah beieinander. Genauso ein Verhältnis sieht man auch bei Docker oder Kata Container, deswegen liegen auch dort die Werte sehr nah beieinander bzw. am Median. Beim Kata und Docker Container sind die Werte höher als bei Firecracker, deswegen wird angenommen, dass die Werte mehr verstreut sind als bei Firecracker.

Es wäre noch interessant den Mittelwert und die Standardabweichung zu berechnen. Diese werden wie im Anhang A7 zusehen ist mit der Bibliothek Numby berechnet. Das Programm liefert die folgenden Ergebnisse:

Ergebnisse	Mittelwert	Standardabweichung	Varianz
Kata Container	35.92s	6.30s	39.69s ²
Firecracker Containerd	22.90s	6.55s	42.90s ²
Docker Container	34.83s	8.48s	71.91s ²

Tabelle 6-Ergebnisse: Primzahlen-Test

Der Mittelwert gibt Auskunft darüber, wie lange ein Container durchschnittlich benötigt, um die Primzahlen zwischen 1 und 20.000 zu berechnen. Der Firecracker Containerd benötigt die wenigste Zeit, gefolgt von Kata Container und anschließend dann der Docker Container. Der Docker Container ist um 13.09s langsamer als Firecracker und um 1.09s langsamer als der Kata Container. Mithilfe der Standardabweichung kann die Varianz berechnet werden daraus kann man schließen, wie stark die Daten der Container verstreut sind. Die Varianz liegt bei Docker Container bei 39.69, bei Firecracker Containerd 42.90 und bei Kata Container bei 71.91. Dies bedeutet das die Verstreuung der Daten von Docker Container und Firecracker sehr ähnlich verstreut sind. Vergleicht man jedoch diese beiden Werte jeweils mit der vom Firecracker, so sind diese Werte halb so stark verstreut.

Die Ergebnisse auf Abbildung 8 sagt aus wie schnell die Container die Primzahlen zwischen 1 und 20.000 ausrechnen, aber interessant wäre noch wie viele Rechenoperationen während des Vorgangs durchgeführt werden. Dieser Wert lässt sich berechnen indem man die Gesamtanzahl der Rechenoperationen von 321.238 durch die Ergebnisse von Anhang A6 dividiert. Als Ergebnis entsteht der folgende Box-Plot.

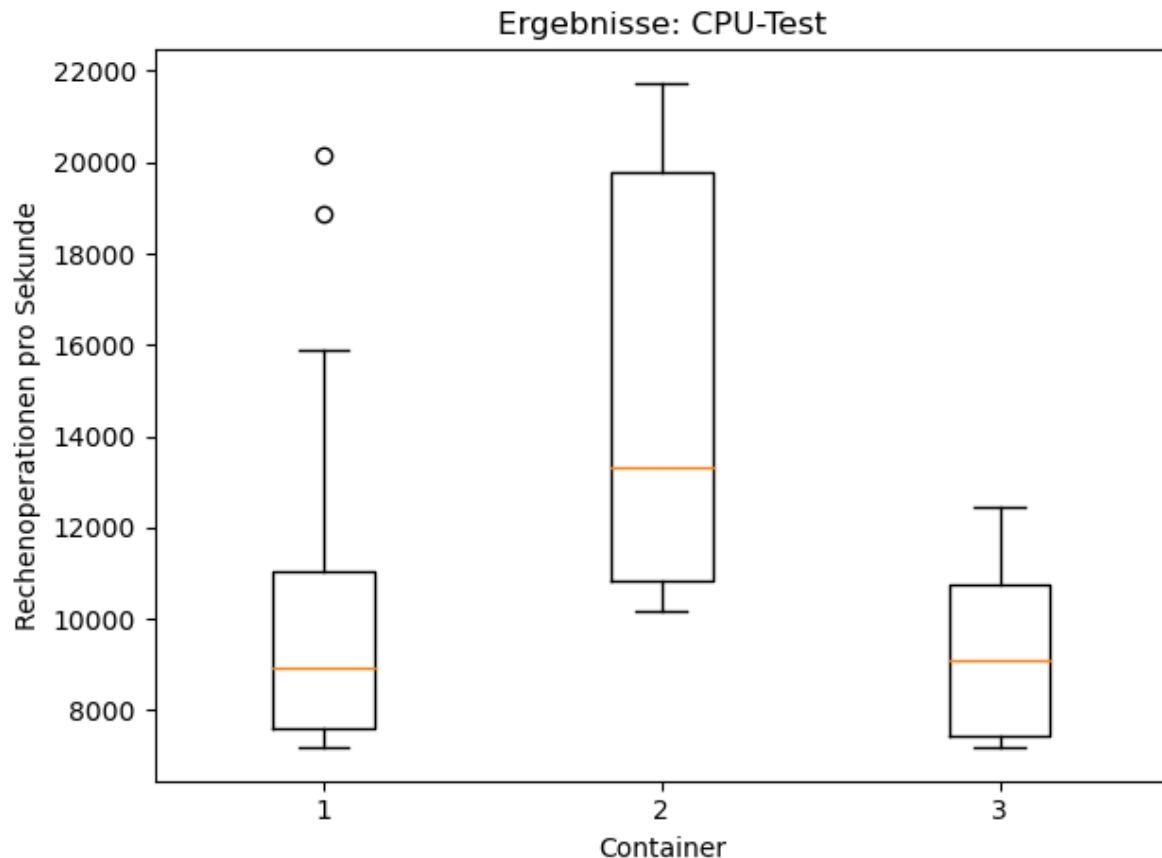


Abbildung 9-Grafik: Rechenoperationen-Test

Die Abbildung 10 zeigt die Ergebnisse von Rechenoperationen pro Sekunde.

Die x-Achse beschreibt die drei Container wie auch in der Abbildung 9 erklärt. Die y-Achse zeigt die unterschiedlichen Werte der Rechenoperationen pro Sekunden. Die inneren Darstellungen sind bei allen Box-Plot Ergebnissen wie auch bei Abbildung 9 erklärt, gleich aufgebaut. Mithilfe der Abbildung 10 lassen sich folgende Ergebnisse in der Tabelle zusammenfassen:

Ergebnisse	Median (50%)	Quartil 1 (25%)	Quartil 3 (75%)	Interquartilsabstand
Kata Container	~8.890s	~7.590s	~11.050s	~3.460s
Firecracker Containerd	~13.310s	~10.840s	~19.800s	~8.960s
Docker Container	~9.110s	~7.460s	~10.790s	~3.330s

Tabelle 7-Ergebnisse: Rechenoperationen-Test

Der Median von Kata und Docker Container liegen bei beiden zwischen ~8.890s und ~9.110s, dies bedeutet, dass die Werte unterhalb und oberhalb der 50% sehr ähnlich aufgeteilt sind. Der Median von Firecracker liegt bei ~13.310s, dass bedeutet das die Werte oberhalb und unterhalb der 50% sich sehr stark von Kata und Docker Container unterscheiden. Das erste Quartil und dritte Quartil bei Kata und Docker Container unterscheiden sich auch hier sehr minimal. Beim Kata Container beträgt das erste Quartil ~7.590s und bei Docker Container ~7.460s. Das zeigt, dass sie sich sowohl oberhalb als auch unterhalb des erstens Quartils minimal unterscheiden. Beim zweiten Quartil sind die Ergebnisse bei Kata Container ~11.050s und bei Docker Container ~10.790s. Auch hier unterscheiden sich die Ergebnisse kaum. Das erste und dritte Quartil von Firecracker unterscheidet sich auch hier sehr stark von Kata und Docker Container. Das untere Quartil liegt bei ca. ~10.840s und das obere bei ~19.800s, wenn man diese mit den anderen Werten von Kata und Docker vergleicht, sieht man direkt, dass die Werte mit starkem Unterschied unterhalb liegen. Die Differenz von Q1 und Q3 ist der Interquartilsabstand, diese Ergebnisse geben die Information wie nah die Werte zwischen Q1 und Q3 liegen. Die Standardabweichungen von Kata und Docker Container sind sehr niedrig, daraus kann man schlussfolgern, dass die Werte sehr schwach voneinander verstreut sind. Im Gegensatz dazu ist der Interquartilsabstand von Firecracker viel höher, d.h. die Werte sind stärker voneinander verstreut als Docker oder Kata Container. Auf der Abbildung sind 2 Ausreißer von Kata Container erkennbar, einmal ist das der Wert ~18.810s und zum anderen der Wert ~2.0150s. Diese 2 Werte zeigen, dass auch Kata Container sehr abweichende Werte aufweisen kann.

Wie bei Tabelle 2 wäre es sinnvoll den Mittelwert und die Standardabweichung zu berechnen. Die Berechnung wird im Anhang A7 durchgeführt, daraus lassen sich folgende Ergebnisse liefern:

Ergebnisse	Mittelwert	Standardabweichung	Varianz
Kata Container	9994.63	3477.98	12.096.344.88
Firecracker Containerd	15252.32	4506.29	20.306.649.56
Docker Container	9221.32	1667.97	2.782.123.92

Tabelle 8-Ergebnisse: Primzahlen-Test

Der Mittelwert gibt Auskunft darüber, wie viele Rechenoperationen durchschnittlich pro Sekunde in den jeweiligen Container ausgeführt werden kann. Mit einem Wert von 15252.32 MB/sec Rechenoperationen pro liefert Firecracker von den drei Containern das Beste Ergebnis. Der Docker Container kann 9994.63 MB/sec Rechenoperationen pro Sekunde ausführen der Kata Container mit 9221.32 Rechenoperationen pro. Der Docker Container kann somit 5257.69 MB/sec Rechenoperationen pro Sekunde weniger als Firecracker ausführen aber ist um 773.31 MB/sec Rechenoperationen pro Sekunde schneller als der Kata Container. Die Varianz liegt bei Docker Container bei 12.096.344,88 Rechenoperationen, bei Firecracker Containerd 20.306.649,56 Rechenoperationen und bei Kata Container bei 2.782.123,92 Rechenoperationen. Dies bedeutet das die Daten von Firecracker fast doppelt so groß wie Docker Container und 10-mal so groß wie Kata Container verstreut sind. Die Daten von Kata Container sind im vergleich zu Firecracker Containerd und Docker Container am wenigsten verstreut.

5.2 Durchführung von RAM-Test

Die Leistungsbewertung von RAM wird auch wie bei dem Testverlauf von CPU innerhalb des Containers ausgeführt und als Tool wird auch hier von Benchmark das sysbench benutzt, um den RAM zu testen. Beim Ram Test geht es darum, wie schnell ein Container MB pro Sekunde im Hauptspeicher abspeichern kann (vgl. Webhosterwissen 2018).

```
#!/bash/bin!  
# CPU: Primzahlen-Test  
  
i=1  
while [ $i -le 25 ]  
do  
    sysbench --num-threads=1 --test=memory --memory-block-size=1M --memory-total-size=100G run | head -n18 | tail -n1 >> RAM_Test_fuer_Docker_Container.csv  
    sleep 30s  
    i=`expr $i + 1`  
done
```

Quelle: (Webhosterwissen 2018)

Code 12-Benchmark :RAM Bash-Script

In Code 12 wird ein sysbench Befehl wie auch beim Primzahlen Test in Kapitel 5.1 durchgeführt und Optionen wie die Anzahl der Cores mit der Option --num-threads= festgelegt. Da in allen Container nur ein Core verwendet wird, kann man auch maximal nur ein Core festlegen. Beim Primzahlen Test wird für die Option --test= die CPU festgelegt, beim Rest Test wir nicht die CPU festgelegt, sondern memory wie auch im RAM Code 12 zu sehen

ist. Beim Memory Test werden GB in Blocken von je MB in dem RAM geschrieben, dadurch wird gezeigt, wie viel MB pro Sekunde in dem RAM geschrieben wird. Den MB Wert legt man wie oben schon im Code gesehen mit der Option `--memory-block-size=`. In dem Fall wird der MB Wert 1 festgelegt. Der Gigabyte wird mithilfe der Option `--memory-total-size=` festgelegt. Dafür wird in Code 12 der Wert 100G festgesetzt, d.h. im Test werden 100GB in Blocken von je 1MB in dem Ram geschrieben. Die Durchläufe der Tests werden mithilfe der `while` und `sleep` Option wie auch im Kapitel 5.1 erklärt und durchgeführt. Dabei wird der Test 25-mal wiederholt und nach jedem Durchlauf 30 Sekunden pausiert, sodass das System wie auch die anderen Tests sich ausruhen können. Die Ergebnisse im Anhang A8 werden mit dem Python Programm im Anhang A9 eingelesen und geplottet.

5.2.1 Ergebnisse RAM Test

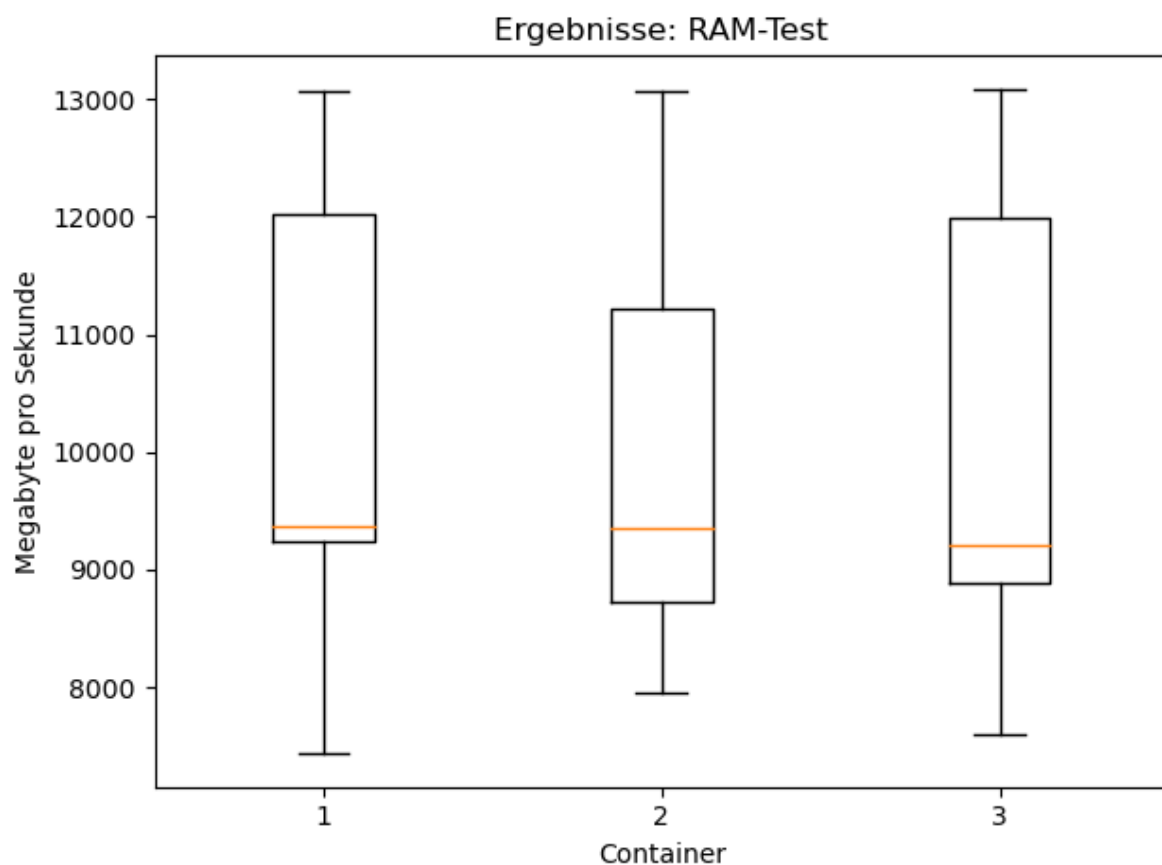


Abbildung 10-Grafik: Hauptspeicher-Test

Die Abbildung 11 zeigt, wie viel Megabyte pro Sekunde im Hauptspeicher geschrieben werden.

Die x-Achse beschreibt die drei Container wie auch in der Abbildung 10 erklärt. Die y-Achse zeigt wie viel Megabyte pro Sekunde ein Container im Hauptspeicher schreiben kann. Die innere Darstellung ist bei allen Box-Plot Ergebnissen wie auch bei Abbildung 9 und 10 erklärt, gleich aufgebaut. Mithilfe der Abbildung 1 lassen sich folgende Ergebnisse in der Tabelle zusammenfassen:

Ergebnisse	Median (50%)	Quartil 1 (25%)	Quartil 3 (75%)	Interquartilsabstand
Kata Container	~9.340 MB/sec	~9.240 MB/sec	~12.200 MB/sec	~3.460 MB/sec
Firecracker Containerd	~9.330 MB/sec	~8.720 MB/sec	~11.190 MB/sec	~2.470 MB/sec
Docker Container	~9.190 MB/sec	~8.870 MB/sec	~12.200 MB/sec	~3.330 MB/sec

Tabelle 9-Ergebnisse: Hauptspeicher-Test

Der Median von Kata Container, Docker Container und Firecracker Containerd unterscheiden sich minimal, wie in der Tabelle 5 zu sehen ist. Das bedeutet, dass die Werte der 3 Containerarten oberhalb und unterhalb der 50% sehr ähnlich aufgebaut sind. Auch beim Quartil ist ein sehr ähnliches Ergebnis dargestellt, sie unterscheiden sich im unteren Quartil und im oberen Quartil mit nur minimalen Werten. Die einzigen Werte, die sich unterscheiden ist der Interquartilsabstand, wie diese zu verstehen ist wird in Kapitel 5.1.1 erklärt. Da der Interquartilsabstand von Kata Container und Docker Container niedrig sind, ist anzunehmen, dass die Werte eine sehr schwache Abweichung aufweisen. Bei Firecracker ist der Interquartilsabstand ein wenig niedriger als bei Docker oder Kata Container, daher kann man sagen das die Abweichungen der Werte hier noch niedriger ausfallen. Bei Firecracker ist somit eine niedrigere Abweichung der Werte als bei Docker oder Kata Container zu beobachten.

Wie auch beim CPU-Test wird auch hier der Mittelwert und die Standardabweichung mit dem Python Programm im Anhang A9 berechnet, das Programm liefert folgende Ergebnisse:

Ergebnisse	Mittelwert	Standardabweichung	Varianz
Kata Container	10257.88 MB/sec	1901.31 MB/sec	3.614.979,72 (MB/sec) ²
Firecracker Containerd	10160.86 MB/sec	1762.58 MB/sec	3.106.688,25 (MB/sec) ²
Docker Container	10296.45 MB/sec	1689.06 MB/sec	2.853.599,35 (MB/sec) ²

Tabelle 10-Ergebnisse: Primzahlen-Test

Der Mittelwert gibt Auskunft darüber, wie viel Megabyte pro Sekunde durchschnittlich im Hauptspeicher gespeichert werden kann. Der höchste Wert erreicht der Kata Container mit 10296.45 MB/sec, danach erfolgt der Docker Container mit einem Wert von 10257.88 MB/sec und zum Schluss der Firecracker mit 10160.86 MB/sec. Der Docker Container schreibt somit 38.57 MB/sec langsamer als der Kata Container ist aber um 97.02 MB/sec schneller als der Firecracker. Die Varianz liegt bei Docker Container bei 3.614.979,72 (MB/sec)², bei Firecracker Containerd 3.106.688,25 (MB/sec)² und bei Kata Container bei 2.853.599,35 (MB/sec)². Dies bedeutet das die verstreueung der Daten von Docker Container und Firecracker Containerd sich sehr stark ähneln. Im Gegensatz dazu ist die Verstreuung von Kata Container viel geringer ausgeprägt also die von Docker Container oder Firecracker Containerd.

5.3 Durchführung von Network-Perfomance Test

In diesem Test geht es darum die verschiedenen Container zu testen, wie lange sie brauchen eine Webseite mit all samt der HTTP und CSS-Dateien herunterzuladen.

Für die Durchführung des Testes wird das folgende Bash Skript verwendet.

```
#!/bash/bin!  
i=1  
while [ $i -le 25 ]  
do  
  wget -nv -r -k -E -l 8 http://192.168.1.109/ -o networktestganz  
  cat networktestganz | head -n16 | tail -n1 >> networktestvoll  
  rm -rf 192.168.1.109  
  sleep 10s  
  i=`expr $i + 1`  
done
```

Quelle: (Webhosterwissen 2018)

Code 13-Network-Performance Bash-Script

In Code 13 wird mithilfe der `while` Schleife wie auch bei den anderen Skripten der Test 25-mal durchgeführt und nach jedem Durchlauf findet eine 10 Sekunden Pause statt. Detaillierter wird die `while` und `sleep` Option in Kapitel 5.1 erklärt. Was hier wichtig zu betrachten ist, ist der Befehl `wget`, `cat` und `rm`. Mithilfe von `wget` kann man von einer Domäne eine ganze Webseite herunterladen. In diesem Skript wird die Domäne `http://192.168.1.109/` heruntergeladen und die Dateien im Ordner `192.168.1.109` abgespeichert. Wie lange der Download gedauert hat wird in der Datei `networktestganz` abgespeichert und mit den Befehlen, `cat`, `head` und `tail` die wichtige Zeile in der Datei `networktestvoll` abgelegt. Die CSV

Die Datei ist im Anhang A10 zu sehen, diese wird mit dem Programm im Anhang A11 eingelesen und geplottet. Der Test wird durchgeführt, um herauszufinden, wie viel Megabyte pro Sekunde den jeweiligen Containern benötigt wird, um eine 432k große Domäne herunterzuladen. Die 432k Domäne beinhaltet HTML-Dateien, CSS-Dateien und Bilder.

5.3.1 Ergebnisse von Network-Performance Test

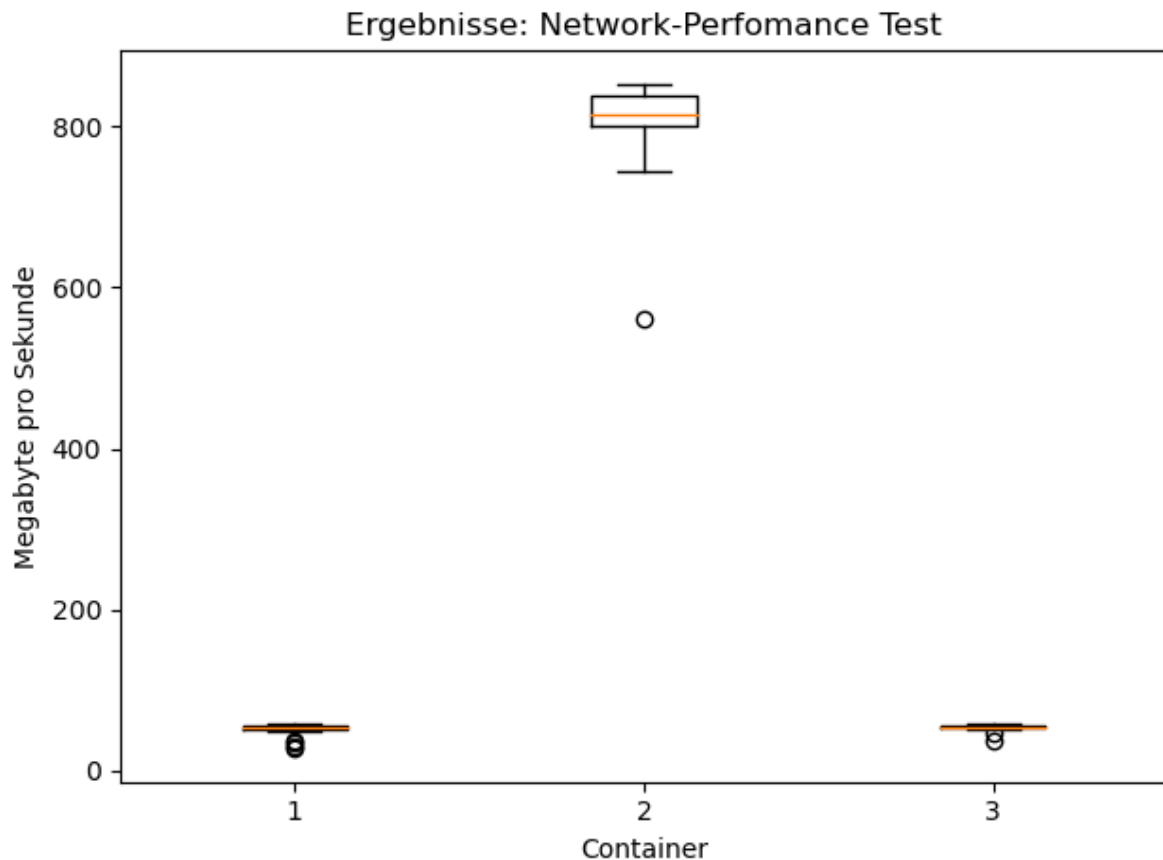


Abbildung 11-Grafik: Network-Performance-Test

Die Abbildung 12 zeigt, wie viel Megabyte pro Sekunde die Container benötigt haben, um eine 432k große Datei herunterzuladen.

Damit man die Werte von Kata Container und Docker Container besser erkennt, wird die Abbildung 12 nochmal vergrößert dargestellt.

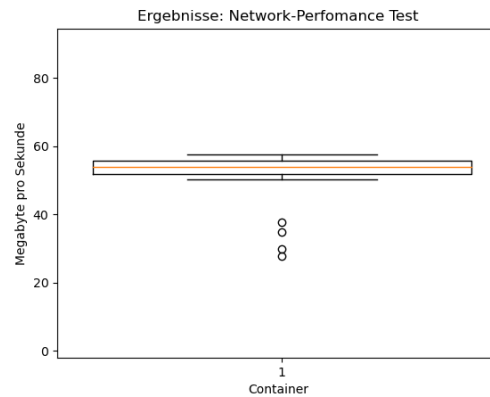


Abbildung 12-Grafik: Network-Performance-Test Kata Container

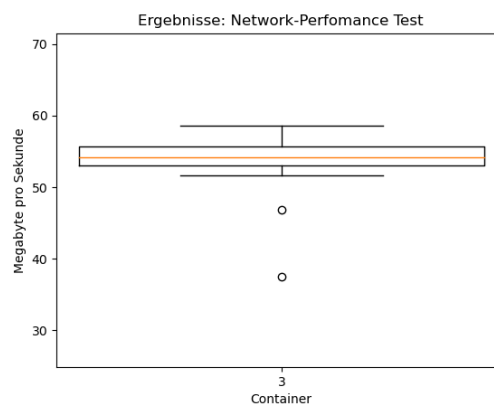


Abbildung 13-Grafik: Network-Performance-Test Docker Container

Die x-Achse und y-Achse ist Äquivalent zu der Abbildung 11 in Kapitel 5.2 Mithilfe der Abbildung 12 lassen sich folgende Ergebnisse in der Tabelle zusammenfassen:

Ergebnisse	Median (50%)	Quartil 1 (25%)	Quartil 3 (75%)	Interquartilsabstand
Kata Container	~53.8 MB/sec	~51.6 MB/sec	~55.8 MB/sec	~4.2 MB/sec
Firecracker Containerd	~813 MB/sec	~799 MB/sec	~838 MB/sec	~39 MB/sec
Docker Container	~54.3 MB/sec	~53.1 MB/sec	~55.5 MB/sec	~2.4 MB/sec

Tabelle 11-Network-Perfomance-Test

Wie man bei Tabelle 6 sehen kann, ist der Median von Kata Container und Docker Container minimal unterschiedlich, das bedeutet, dass die Werte unterhalb und oberhalb der 50% ähnlich aufgeteilt sind. Der Median bei Firecracker liegt mit einem Riesen Unterschied zu Docker oder Kata Container bei ~813 MB/sec, dies heißt das 50% der Werte oberhalb von ~813 MB/sec

liegen und 50% unterhalb dieses Wertes. Das erste Quartil und dritte Quartil bei Kata und Docker Container unterscheiden sich auch hier sehr minimal. Beim Kata Container beträgt das erste Quartil ~51.6 MB/sec und bei Docker Container 53.1 MB/sec, dies zeigt das sich sowohl oberhalb als auch unterhalb des erstens Quartils minimal unterscheiden. Beim dritten Quartil sind die Ergebnisse bei Kata Container ~55.8 MB/sec und bei Docker Container 55.5 MB/sec, auch hier unterscheiden sich die Ergebnisse kaum. Das erste und dritte Quartil von Firecracker Containerd unterscheidet sich auch hier sehr stark von Kata und Docker Container. Das untere Quartil liegt bei ca. ~799 MB/sec und das obere bei ~838 MB/sec, wenn man diese mit den anderen Werten von Kata und Docker vergleicht, dann sieht man, dass die Werte mit starkem Unterschied unterhalb liegen. Der Interquartilsabstand von allen drei Containern unterscheiden sich sehr. Der Firecracker hat im Gegensatz zu Docker und Kata Container ein sehr hoher Wert, dennoch ist es im Verhältnis zu dem Median nicht hoch, deswegen kann man schlussfolgern, dass die Werte nicht stark verstreut sind, sondern sehr nah beieinander liegen. Der Kata Container hat genauso wie der Docker Container ein sehr niedrige Interquartilsabstand daher kann man sagen, dass die Werte sehr schwach voneinander abweichen und die Werte kaum verstreut sind. Ausreißer sind sowohl bei Docker Container, Kata Container und Firecracker Containerd zusehen. Der Firecracker hat wie man bei der Abbildung 12 sehen kann einen Ausreißer. Die Abbildung 13 zeigt, dass der Kata Container 4 Ausreißer hat. Als letztes zeigt die Abbildung 14 das auch der Docker Container 2 Ausreißer hat.

Der Mittelwert und die Standardabweichung werden im Anhang A11 berechnet, daraus lassen sich die folgenden Ergebnisse realisieren:

Ergebnisse	Mittelwert	Standardabweichung	Varianz
Kata Container	53.55 MB/sec	3.95 MB/sec	15.60 (MB/sec) ²
Firecracker Containerd	803.36 MB/sec	57.10 MB/sec	3.260,41 MB/sec) ²
Docker Container	50.78 MB/sec	8.33 MB/sec	69.39 (MB/sec) ²

Tabelle 12-Ergebnisse: Primzahlen-Test

Der Mittelwert gibt Auskunft darüber, wie viel Megabyte pro Sekunde durchschnittlich die Container benötigen, um eine 432k große Datei herunterzuladen. Der Firecracker erzielt mit dem Wert 803,36 MB/sec den höchsten Wert. Der Docker Container erzielt den zweit höchsten

mit 53,55 MB/sec und der Kata Container den dritt höchsten Wert mit 50,78 MB/sec. Der Docker Container ist somit durchschnittlich um 2,77 MB/sec schneller als der Kata Container aber 749,81 MB/sec langsamer als der Firecracker Containerd. Die Varianz liegt bei Docker Container bei 15.60 (MB/sec)², bei Firecracker Containerd 3.260.41 (MB/sec)² und bei Kata Container bei 69,39 (MB/sec)². Dies bedeutet das die verstreueung der Daten von Docker Container und Firecracker Containerd und Kata Container sich sehr stark unterscheiden. Der Wert von Docker Container ist viermal geringer als der von Kata Container und 240 Fach geringer als der von Firecracker Containerd. Dies bedeutet das die Verstreueung der Daten von Docker Container viel geringer ist als Kata Container und sehr stark geringer ist als Firecracker Containerd.

5.4 Durchführung von Startzeit-Test

Bei diesem Test wird die Startzeit der unterschiedlichen Containerlösungen überprüft. Der Test wird mithilfe des `sysbench` Test in Kapitel 5.1 durchgeführt.

```
#!/bash/bin!  
i=1  
while [ $i -le 25 ]  
do  
    { time docker exec -it ba142a1ce87f sysbench --num-threads=1 --test=cpu --cpu-  
max-prime=20000 run ; } 2> test  
    cat test | head -n2 | tail -n1 >> Start_Test_fuer_Docker_Container.csv  
    sleep 30s  
    i=`expr $i + 1`  
done
```

Quelle: (Webhosterwissen 2018)

Code 14-Startzeit Bash-Script

In Code 15 wird der Befehl `sysbench` wie auch im Kapitel 5.1 verwendet, aber nicht für den CPU Test, sondern um die Startzeiten der Container zu testen. Es gibt insgesamt 25 Durchläufe und nach jedem Durchlauf gibt es eine 30 Sekunden Pause. Die Ergebnisse werden in einer CSV Datei abgespeichert. Die CSV-Datei ist im Anhang A12 zusehen. Der Unterschied zwischen dem Befehl in Kapitel 5.1 und dieser ist es, das zum einen der Befehl `time` verwendet wird und zum anderen wird dieser Test nicht innerhalb der Container durchgeführt, sondern außerhalb. Mit `time` ist es möglich von verschiedenen Befehlen oder Programmen die Laufzeit berechnen zu lassen. Diese Ausgabe wird dann im Terminal ausgegeben und der Benutzer weiß wie viel Zeit für die Durchführung des Befehls benötigt wird. In diesem Test wird überprüft, wie lange die verschiedenen Container benötigen, um den `sysbench` Befehl aus Kapitel 5.1 auszuführen. Diese Ergebnisse werden dann mithilfe des Programmes im Anhang A13 eingelesen und anschließend geplottet.

5.4.1 Ergebnisse von Startzeit-Test

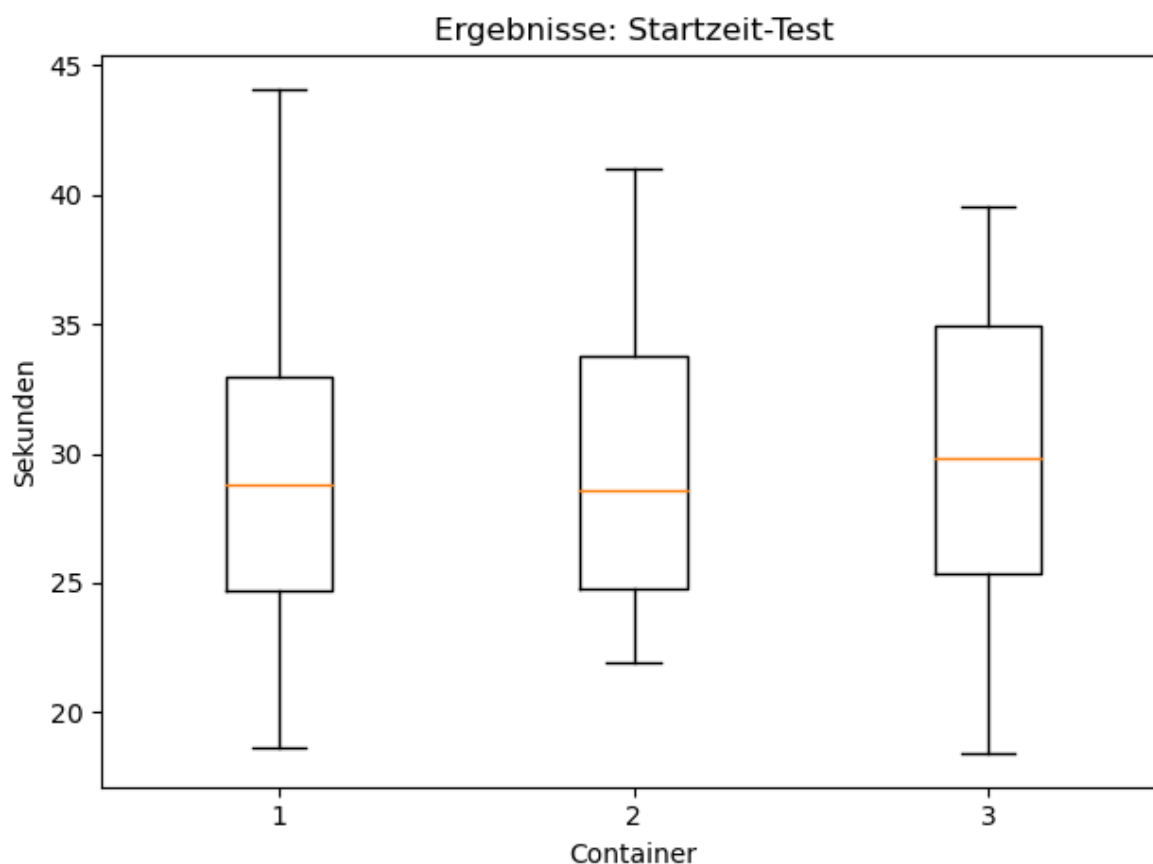


Abbildung 14-Grafik: Startzeit-Test

Die Abbildung 15 zeigt, wie lange die Container benötigt haben, um den erwähnten Sysbench Befehl auszuführen.

Die x-Achse beschreibt die drei Container wie auch in der Abbildung 14 erklärt. Die y-Achse zeigt die Sekunden Angabe der ausgeführten Testdurchläufe. Die innere Darstellung ist bei allen Box-Plot Ergebnissen wie auch bei Abbildung 13 erklärt, gleich aufgebaut. Mithilfe der Abbildung 15 lassen sich folgende Ergebnisse in der Tabelle zusammenfassen:

Ergebnisse	Median (50%)	Quartil 1 (25%)	Quartil 3 (75%)	Interquartilsabstand
Kata Container	~28.68s	~24.70s	~33.03s	~8.33s
Firecracker Containerd	~28.52s	~24.78s	~33.79s	~9.01s
Docker Container	~29.82s	~25.39s	~34.94s	~9.55s

Tabelle 13-Ergebnisse: Startzeit-Test

Wie man in der Tabelle 7 sehen kann ist der Median bei allen drei Containerlösungen sehr ähnlich, das hat zu bedeuten, dass die Werte unterhalb und oberhalb der 50% sich minimal voneinander unterscheiden. Der Median von Kata Container beträgt ~28,68s, von Firecracker Containerd 28,52s und zuletzt bei Docker Container um 29,82s. Der größte Wert, der sich unterscheidet, ist Docker Container. Dort beträgt der Wert 29,81, das macht ein Unterschied von ca. 1 Sekunde, wenn man diese mit den Werten von Docker und Kata Container vergleicht. Aber nicht nur bei Median sind minimale Unterschiede zu erkennen, sondern auch beim ersten und zweiten Quartil. Beim Kata Container beträgt das erste Quartil ~28.68s und der zweite ~33.03s, genauso ähnliche Werte hat auch der Firecracker mit den Werten von ~24.78s und ~33.79s. Der Docker Container hat einen leicht erhöhten Wert, wenn man diese mit den anderen beiden Containern vergleicht. Bei Docker beträgt diese ~25.39s und ~34.94s. Wenn man diese Werte mit den von Firecracker und Kata Container vergleicht dann ist ein Unterschied von ca. 1 Sekunde zu erkennen. Der Interquartilsabstand der Werte unterscheiden sich auch hier sehr minimal. Der Interquartilsabstand von Kata Container beträgt ~8,33s, von Firecracker Containerd ~9.01s und zuletzt von Docker Container ~9,55s. Dies bedeutet das die Werte von allen drei Container nah aneinander liegen und keine starken Abweichungen vorhanden sind. Ausreißer sind bei keinem der Container zu sehen.

Der Mittelwert und die Standardabweichungen werden im Anhang A13 berechnet, die Ergebnisse sind folgendermaßen zusammengefasst:

Ergebnisse	Mittelwert	Standardabweichung	Varianz
Kata Container	29.67s	6.16s	37.95s ²
Firecracker Containerd	29.19s	5.67s	32.15s ²
Docker Container	29.22s	5.97s	35.64s ²

Tabelle 14-Ergebnisse: Primzahlen-Test

Der Mittelwert gibt Auskunft darüber, wie viele Sekunden durchschnittlich ein Container benötigt, um den Sysbench Befehl aus Kapitel 5.4 auszuführen. Die Mittelwerte sind sehr identisch und zeigen, dass sie durchschnittlich gleich lange benötigen, um den Befehl auszuführen. Docker Container benötigt 29.67s, Firecracker Containerd 29.19s und der Kata Container 29.22s.

Die Varianz liegt bei Docker Container bei $37,95s^2$, bei Firecracker Containerd $32,15s^2$ und bei Kata Container bei $35,64s^2$. Dies bedeutet das die verstreung der Daten von Docker Container, Firecracker Containerd sich sehr stark ähneln. Die Verstreuung der Daten von Docker Container sind fast so gleich wie die von Kata Container verstreut. Der Wert von Firecracker Containerd ist mit minimal Abstand geringer als die von Docker Container und Kata Container, deswegen kann man Aussagen das die Verstreuung der Daten sich minimal unterscheiden.

6 Zusammenfassung

Die Werte von Docker Container und Kata Container haben sich im CPU-Test (siehe. Kapitel 5.1.1) kaum unterschieden, die Werte unterscheiden sich nur minimal, deswegen kann man sagen das sich die Leistungen sehr ähnlich verhalten. Der Firecracker Containerd hat dagegen mit großem Abstand viel bessere Ergebnisse als Kata Container und Docker Container geliefert, somit ist hier ganz klar eine bessere CPU-Leistung zu erkennen. Die Daten sind bei Firecracker verstreuter als bei Docker Container oder Kata Container, deswegen ist hier anzunehmen das der Firecracker Containerd sowohl ganz hohe als auch niedrige Ergebnisse liefern kann und die anderen beiden ähnliche hohe und niedrige Ergebnisse liefern.

Im RAM-Test (siehe. Kapitel 5.2.1) haben sich die Werte von Firecracker Containerd, Docker Container und Kata Container sehr ähnliche Werte aufgewiesen. Deswegen kann man sagen, dass die RAM Leistungen sich bei allen drei Containern sehr ähnlich verhalten. Die Daten sind bei Docker Container viel mehr verstreut als die von Firecracker Containerd oder Kata Container und die Verstreuung von Firecracker Containerd und Kata Container sind sehr ähnlich. Hier kann man sagen, dass Docker Container minimal starke hohe und niedrige Ergebnisse liefert. Im Vergleich sind die Ergebnisse von Firecracker Containerd und Kata Containerd ähnlich hoch und niedrig.

Der Network-Performance Test hat der Kata Container und Docker Container sehr ähnliche Ergebnisse geliefert, deswegen kann man sagen das diese beiden Container eine ähnliche Network-Performance aufweisen. Im Gegenzug hat der Firecracker mit großem Abstand viel bessere Ergebnisse geliefert als der Docker Container oder Kata Container, deswegen kann man nach den Ergebnissen sagen, dass der Firecracker eine viel bessere Network-Performance aufweist als den anderen beiden Containern. Die Verstreuung von Docker Container ist am niedrigsten gefolgt von Kata Container und Firecracker Containerd. Die Ergebnisse von Firecracker Containerd ist um ein Vielfaches stärker verteilt als die von Docker Container oder Kata Container, deswegen kann man sagen das Firecracker sowohl ganze hohe und tiefer Ergebnisse liefern kann.

Im Startzeit-Test (Kapitel 5.3.1) gab es bei allen drei Container sehr ähnliche Ergebnisse, deswegen kann man sagen das sich die Startzeit Leistung nur minimal unterscheiden und sehr ähnlich verhalten. Das Verhalten der Verstreuung unterschieden sich bei allen Containern minimal, deswegen kann man sagen das die Verstreuung der Daten sich kaum unterscheiden.

Zusammenfassend kann man sagen das sich der Docker Container und Kata Container von den Ergebnissen nicht stark unterscheiden. Der Firecracker hat im CPU-Test und Network-Perfomance bessere Ergebnisse geliefert als der Docker Container und Kata Container. Beim RAM und Startzeiten Test haben alle drei Container sehr ähnliche Ergebnisse geliefert. Der Firecracker kann sowohl ganz hohe Werte liefern als auch ganz niedrige, weil die Daten meistens sehr verstreut sind. Im Gegensatz dazu liefert der Kata Container und Docker Container gleich Konstante Ergebnisse, weil die Daten eher weniger verstreut sind als der Firecracker Containerd.

7 Ausblick

Wie schon in Kapitel 6 erläutert, liefern die Ergebnisse von Docker Container und Kata Container sehr ähnliche Ergebnisse aber der Firecracker hat im CPU-Test und Network-Performance Test viel bessere Ergebnisse abgeliefert. Es wäre interessant zu wissen warum der Firecracker in diesen 2 Punkten viel besser abgeschnitten hat als der Docker Container oder Kata Container. Interessant wäre es auch zu Testen ob die anderen Varianten von Firecracker so gute Leistungen erbringen wie der Firecracker Containerd. Man kann auch Firecracker mit Kata Container, Open Nebula, UniK oder Weave FireKube kombinieren und die unterschiedlichen Leistungen miteinander vergleichen. Es könnte beispielsweise sein das Firecracker Containerd eine viel bessere CPU-Leistung abliefert aber die RAM-Leistung aus der Kombination von Kata Container und Firecracker bessere Ergebnisse liefert. Anschließend könnte man auch hier den Grund untersuchen, warum die Leistungen von Firecracker Containerd und die von Firecracker mit Kata Container sich unterscheiden. Da Firecracker Containerd und Kata Container eine viel bessere Sicherheit anbieten, wie auch in Kapitel 2,3 und Kapitel 2,4 erläutert, könnte man schauen welcher der beiden Container die bessere Sicherheit anbietet und warum das so ist. Interessant wäre es auch zu wissen ob man nicht die Sicherheit von Docker Container erhöhen könnte bzw. warum diese nicht möglich ist. Es würden sich aus diesen Aspekten folgende weiterführende Forschungsfragen anschließen:

- Warum ist die Network-Performance und CPU-Leistung von Firecracker leistungsfähiger als von Kata Container und Docker Container?
- Wie sind die Leistungen der unterschiedlichen Kombinationen von Firecracker mit Open Nebula, Kata Container, UniK, Weave und FireKube zu unterscheiden?
- In welchen Sicherheitsaspekten unterscheidet sich der Firecracker Containerd von Kata Container?
- Wie kann man die Sicherheitsaspekte von Docker Container verbessern?

Literaturverzeichnis

Anecon (2018): Docker Basics – Befehle und Life Hacks. Text online abrufbar unter:
<http://www.anecon.com/blog/docker-basics-befehle-und-life-hacks/> (Zugriff am: 25.07.2020).

AWS Amazone (2020): Jetzt neu: Firecracker, ein neues Virtualisierungstechnologie- und Open-Source-Projekt zur Ausführung von Mehrmandanten-Container-Workloads. Text online abrufbar unter: <https://aws.amazon.com/de/about-aws/whats-new/2018/11/firecracker-lightweight-virtualization-for-serverless-computing/> (Zugriff am 23.07.2020).

Berl. A., Fischer A., Meer H. (2010): Virtualisierung im Future Internet. Text online abrufbar unter: <https://link.springer.com/article/10.1007/s00287-010-0420-z> (Zugriff am: 24.07.2020).

Buhl, H., Winter, R. (2008): Vollvirtualisierung – Beitrag der Wirtschaftsinformatik zu einer Vision. Text online abrufbar unter: <https://link.springer.com/article/10.1007/s11576-008-0129-7> (Zugriff am: 23.07.2020).

Cloud Google: CONTAINER BEI GOOGLE, Text online abrufbar unter:
<https://cloud.google.com/containers?hl=de> (Zugriff am: 24.07.2020).

Crisp Research (2014): Docker Container: Die Zukunft moderner Applikationen und Multi-Cloud Deployments?. Text online abrufbar unter: <https://www.crisp-research.com/docker-container-die-zukunft-moderner-applikationen-und-multi-cloud-deployments/> (Zugriff am: 24.07.2020).

Datacenter Insider (2019): Was ist ein Pod in der IT?. Text online abrufbar unter:
<https://www.datacenter-insider.de/was-ist-ein-pod-in-der-it-a-841195/#:~:text=Kubernetes%3A%20Der%20Software%2DPod&text=Hier%20ist%20ein%20Pod%20laut,sich%20Storage%20und%20Netzwerk%20teilen%E2%80%9C.&text=Die%20Container%20eines%20Kubernetes%2DPods,Umgebung%20und%20unter%20gemeinsamer%20Orchestrierung> (Zugriff am: 01.08.2020).

Docker Inc. (2020): What is a Container?. Text online abrufbar unter:
<https://www.docker.com/resources/what-container> (Zugriff am: 23.07.2020).

Docs Microsoft Inc. (2019): Container im Vergleich zu virtuellen Computern. Text online abrufbar unter: <https://docs.microsoft.com/de-de/virtualization/windowscontainers/about/containers-vs-vm> (Zugriff am: 24.07.2020).

Docs Docker (2020a): Docker run reference. Text online abrufbar unter: <https://docs.docker.com/engine/reference/run/> (Zugriff am 24.07.2020).

Docs Docker (2020b): The base command for the Docker CLI. Text online abrufbar unter: <https://docs.docker.com/engine/reference/commandline/docker/> (Zugriff am: 24.07.2020).

Docs Docker (2020c): Build an image from a Dockerfile. Text online abrufbar unter: <https://docs.docker.com/engine/reference/commandline/build/> (Zugriff am: 24.07.2020).

Docs Docker SPI Inc. (2020d): Build an image from a Dockerfile. Text online abrufbar unter: <https://docs.docker.com/get-docker/> (Zugriff am: 30.07.2020).

Entwickler (2019): Docker: Einstieg in die Welt der Container. Text online abrufbar unter: <https://entwickler.de/online/windowsdeveloper/docker-grundlagen-dotnet-container-579859289.html> (Zugriff am: 24.07.2020).

Eosgmbh (2017): Was sind eigentlich Container und Docker?. Text online abrufbar unter: <https://www.eosgmbh.de/container-und-docker> (Zugriff am: 24.07.2020).

Firecracker Microvm (2018): Firecracker is an open source virtualization technology that is purpose-built for creating and managing secure, multi-tenant container and function-based services. Text online abrufbar unter: <https://firecracker-microvm.github.io/> (Zugriff am: 23.07.2020).

GitHub (2018): agent, 19. November. Text online abrufbar unter: <https://github.com/firecracker-microvm/firecracker-containerd/blob/master/docs/agent.md> (Zugriff am: 26.07.2020).

GitHub (2019): firecracker-containerd architecture Text online abrufbar unter:

<https://github.com/firecracker-microvm/firecracker-containerd/blob/master/docs/architecture.md>. (Zugriff am: 26.07.2020).

GitHub (2020a): runc. Text online abrufbar unter: <https://github.com/opencontainers/runc>. (Zugriff am: 26.07.2020).

GitHub (2020b): Kata Containers Architecture, Text online abrufbar unter:

<https://github.com/kata-containers/documentation/blob/master/design/architecture.md> (Zugriff am: 26.07.2020).

GitHub (2020c): Quickstart with firecracker-containerd. Text online abrufbar unter:

<https://github.com/firecracker-microvm/firecracker-containerd/blob/master/docs/quickstart.md>. (Zugriff am: 01.08.2020).

GitHub (2020d): Getting started with Firecracker and containerd. Text online abrufbar unter:

<https://github.com/firecracker-microvm/firecracker-containerd/blob/master/docs/getting-started.md>. (Zugriff am: 01.08.2020).

GitHub (2020e): Getting Started with Firecracker. Text online abrufbar unter:

<https://github.com/firecracker-microvm/firecracker/blob/master/docs/getting-started.md>. (Zugriff am: 01.08.2020).

GitHub (2020f): documentation. Text online abrufbar unter: <https://github.com/kata-containers/documentation/blob/master/Developer-Guide.md>. (Zugriff am: 03.08.2020).

GitHub (2020g): Building. Text online abrufbar unter: <https://github.com/qemu/qemu>.

(Zugriff am: 04.08.2020).

GitHub (2020h): Install Docker for Kata Containers on Ubuntu. Text online abrufbar unter:

<https://github.com/kata-containers/documentation/blob/master/install/docker/ubuntu-docker-install.md>. (Zugriff am: 04.08.2020).

HPE (2020): WAS IST CONTAINER ORCHESTRATION?. Text online abrufbar unter:
<https://www.hpe.com/de/de/what-is/container-orchestration.html>. (Zugriff am: 26.07.2020).

IONOS (2020): Konzepte der Virtualisierung im Überblick. Text online abrufbar unter:
<https://www.ionos.de/digitalguide/server/konfiguration/virtualisierung/>. (Zugriff am: 24.07.2020).

katacontainers (2020a): An overview of the Kata Containers project. Text online abrufbar unter: <https://katacontainers.io/learn/>. (Zugriff am: 23.07.2020).

Linux Magazin (2005): Überlagerte Dateisysteme in der Praxis. Text online abrufbar unter:
<https://www.linux-magazin.de/ausgaben/2005/10/hochstapler/>. (Zugriff am: 26.07.2020).

Lius L., Brown M. (2017): Containerd Brings More Container Runtime Options for Kubernetes. Text online abrufbar unter: <https://kubernetes.io/blog/2017/11/containerd-container-runtime-options-kubernetes/>. (Zugriff am: 26.07.2020).

Man7 (2020): vsock(7) – Linux manual page. Text online abrufbar unter:
<https://man7.org/linux/man-pages/man7/vsock.7.html>. (Zugriff am: 26.07.2020).

Marktforschung: Box-Plot. Text online abrufbar unter: <https://marktforschung.fandom.com/de/wiki/Box-Plot>. (Zugriff am: 11.08.2020).

Matplotlib (2018): documentation. Text online abrufbar unter: <https://matplotlib.org/users/index.html>, (Zugriff am: 05.08.2020).

OpenStack Foundation (2017): The speed of containers, the security of VMs. Text online abrufbar unter: <https://katacontainers.io/collateral/kata-containers-1pager.pdf>. (Zugriff am: 26.07.2020).

Packages Debian SPI Inc. (2020a): Paket: sysstat (11.0.1-1). Text online abrufbar unter:
<https://packages.debian.org/de/jessie/sysstat>. (Zugriff am: 30.07.2020).

Packages Debian SPI Inc. (2020b): Paket: nicstat (1.95-1 und andere). Text online abrufbar unter: <https://packages.debian.org/de/sid/nicstat>. (Zugriff am: 30.07.2020).

Packages Debian SPI Inc. (2020c): Paket: iftop (1.0~pre4-7 und andere). Text online abrufbar unter: <https://packages.debian.org/de/sid/iftop>, (Zugriff am: 30.07.2020).

Redhat Inc. (2020a): Was ist Virtualisierung?. Text online abrufbar unter: <https://www.redhat.com/de/topics/virtualization/what-is-virtualization>. (Zugriff am: 24.07.2020).

Redhat Inc. (2020b): Docker – Funktionsweise, Vorteile, Einschränkungen. Text online abrufbar unter: <https://www.redhat.com/de/topics/containers/what-is-docker>. (Zugriff am: 24.07.2020).

Rouse M. (2020): Agnostisch. Text online abrufbar unter: <https://whatis.techtarget.com/de/definition/Agnostisch>. (Zugriff am: 26.07.2020).

Systutorials, ctr (1) – Linux Man Pages: Text online abrufbar unter: <https://www.systutorials.com/docs/linux/man/1-ctr/>. Zugriff am: 03.08.2020).

Unix Stackxchange (2019): docker.service - How to edit systemd service file?. Text online abrufbar unter: <https://unix.stackexchange.com/questions/542343/docker-service-how-to-edit-systemd-service-file>. Zugriff am: 01.08.2020).

Webhosterwissen (2018): Server-Benchmark mittels sysbench. Text online abrufbar unter: <https://www.webhosterwissen.de/know-how/server/server-benchmark/>. (Zugriff am: 05.08.2020).

Wiki Ubuntuusers (2020a): wget. Text online abrufbar unter: <https://wiki.ubuntuusers.de/wget/>. (Zugriff am: 01.08.2020).

Wiki Ubuntuusers (2020b): Benchmarks. Text online abrufbar unter: <https://wiki.ubuntuusers.de/Benchmarks/>. (Zugriff am: 01.08.2020).

Wiki Ubuntuusers (2020c): Apache 2.4. Text online abrufbar unter:
https://wiki.ubuntuusers.de/Apache_2.4/. (Zugriff am: 01.08.2020).

Witt, M., Jansen, C., Breuer, S., Beier, S., Krefitng, D. (2017): Artefakterkennung über eine cloud-basierte Plattform. Text online abrufbar unter:
<https://link.springer.com/article/10.1007/s11818-017-0138-0#citeas> (Zugriff am: 22.07.2020).

Floyd B., Dr. Bergler A. (2017): Was ist Storage?. Text online abrufbar unter: <https://www.it-business.de/was-ist-storage-a-663183/>. (Zugriff am: 26.07.2020).

8 Anhang

Anhang A 1. Docker Container installieren

```
# Alte Versionen von Docker Container entfernen
$ sudo apt-get remove docker docker-engine docker.io containerd runc

# System aktualisieren
$ sudo apt-get update

# Benötigte Tools installieren
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common

# GPG überprüfen
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

# Key ausgeben
$ 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 Engine installieren
$ sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io

# Docker Engine Version auswählen
$ 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

# Container starten
$ sudo docker run hello-world
```

Anhang A 2. Firecracker Containerd installieren

```
# Hardware überprüfen
err=""; \
[ "$(uname -m)" = "Linux x86_64" ] \
|| err="ERROR: your system is not Linux x86_64."; \
[ -r /dev/kvm ] && [ -w /dev/kvm ] \
|| err="$err\nERROR: /dev/kvm is innaccessible."; \
(( $(uname -r | cut -d. -f1)*1000 + $(uname -r | cut -d. -f2) >= 4014 )) \
|| err="$err\nERROR: your kernel version ($(uname -r)) is too old."; \
dmesg | grep -i "hypervisor detected" \
&& echo "WARNING: you are running in a virtual machine. Firecracker is not well
tested under nested virtualization."; \
[ -z "$err" ] && echo "Your system looks ready for Firecracker!" || echo -e "$err"

# Firecracker Repistory Herunterladen
$ git clone https://github.com/firecracker-microvm/firecracker-containerd.git
$ cd firecracker-containerd
$ sudo make install install-firecracker demo-network

# Kernel Herunterladen
$ curl -fsSL -o hello-vmlinux.bin https://s3.amazo-
naws.com/spec.ccfc.min/img/hello/kernel/hello-vmlinux.bin
# Image installieren
$ make image
$ sudo mkdir -p /var/lib/firecracker-containerd/runtime
$ sudo cp tools/image-builder/rootfs.img /var/lib/firecracker-sudo
containerd/runtime/default-rootfs.img

# config.toml konfigurieren
{
  "firecracker_binary_path": "/usr/local/bin/firecracker",
  "kernel_image_path": "/var/lib/firecracker-containerd/runtime/hello-vmlinux.bin",
  "kernel_args": "console=ttyS0 noapic reboot=k panic=1 pci=off nomodules ro sys-
temd.journald.forward_to_console systemd.unit=firecracker.target init=/sbin/over-
lay-init",
  "root_drive": "/var/lib/firecracker-containerd/runtime/default-rootfs.img",
  "cpu_template": "T2",
  "log_fifo": "fc-logs.fifo",
  "log_level": "Debug",
  "metrics_fifo": "fc-metrics.fifo",
  "shim_base_dir": "/root/go/src/github.com/firecracker-containerd/runtime/contai-
nerd-shim-aws-firecracker",
  "default_network_interfaces": [{
    "CNICfg": {
      "NetworkName": "fcnet",
      "InterfaceName": "veth0"
    }
  }]
}
```

Anhang A 3. Kata Container installieren

```
# Golang installieren
$ sudo apt-get update
$ sudo apt-get -y upgrade
$ cd /tmp
$ wget https://dl.google.com/go/go1.13.3.linux-amd64.tar.gz
$ sudo tar -xvf go1.13.3.linux-amd64.tar.gz
$ sudo mv go /usr/local
$ export GOROOT=/usr/local/go
$ export GOPATH=~/.go
$ export PATH=$GOPATH/bin:$GOROOT/bin:$PATH
```

```
$ go env // Pfade überprüfen

# Grundlegende Architektur installieren
$ go get -d -u github.com/kata-containers/runtime
$ cd $GOPATH/src/github.com/kata-containers/runtime
$ make && sudo -E PATH=$PATH make install

# Hardware überprüfen
$ sudo kata-runtime kata-check

# Full Debug aktivieren
$ sudo mkdir -p /etc/kata-containers/
$ sudo install -o root -g root -m 0640 /usr/share/defaults/kata-containers/configuration.toml /etc/kata-containers
$ sudo sed -i -e 's/^# *\(\enable_debug\).*=.*$/\1 = true/g' /etc/kata-containers/configuration.toml
$ sudo sed -i -e 's/^kernel_params = "\(.*)"/kernel_params = "\1 agent.log=debug initcall_debug"/g' /etc/kata-containers/configuration.toml

# Kata Shim installieren
$ go get -d -u github.com/kata-containers/shim
$ cd $GOPATH/src/github.com/kata-containers/shim
$ make && sudo make install

# Kata Proxy installieren
$ go get -d -u github.com/kata-containers/proxy
$ cd $GOPATH/src/github.com/kata-containers/proxy
$ make && sudo make install

# Osbuilder Herunterladen
$ go get -d -u github.com/kata-containers/osbuilder

# Rootfs Datei installieren und aufbauen
$ export ROOTFS_DIR=${GOPATH}/src/github.com/kata-containers/osbuilder/rootfs-builder/rootfs
$ sudo rm -rf ${ROOTFS_DIR}
$ cd $GOPATH/src/github.com/kata-containers/osbuilder/rootfs-builder
$ export distro=debian
$ script -fec 'sudo -E GOPATH=$GOPATH USE_DOCKER=true SECCOMP=no ./rootfs.sh ${distro}'
$ commit=$(git log --format=%h -1 HEAD)
$ date=$(date +%Y-%m-%d-%T.%N%z)
$ image="kata-containers-${date}-${commit}"
$ cd /usr/share/kata-containers
$ sudo install -o root -g root -m 0640 -D kata-containers.img "/usr/share/kata-containers/${image}"
$ (cd /usr/share/kata-containers && sudo ln -sf "$image" kata-containers.img)

# Kernel installieren
$ go get -d -u github.com/kata-containers/packaging
$ cd $GOPATH/src/github.com/kata-containers/packaging/kernel
$ ./build-kernel.sh setup
$ ./build-kernel.sh install

# Hypervisor installieren
$ go get -d github.com/qemu/qemu
$ mv ${GOPATH}/root/go/src/github.com kata-containers
$ cd ${GOPATH}/src/github.com/kata-containers/qemu/qemu
$ mkdir build
$ cd build
$ ../configure
$ make -j $(nproc)
$ sudo -E make install
```

Anhang A 4. Firecracker ohne Containerd installieren

```
#!/bin/bash
#Firecracker installieren
sudo setfacl -m u:${USER}:rw /dev/kvm
curl -Lo firecracker https://github.com/firecracker-microvm/firecracker/releases/download/v0.16.0/firecracker-v0.16.0
chmod +x firecracker
sudo mv firecracker /usr/local/bin/firecracker

# Rootfs Kernel für Firecracker installieren
curl -fsSL -o hello-vmlinux.bin https://s3.amazonaws.com/spec.ccfc.min/img/hello/kernel/hello-vmlinux.bin
curl -fsSL -o hello-rootfs.ext4 https://s3.amazonaws.com/spec.ccfc.min/img/hello/fsfiles/hello-rootfs.ext4

#Netzwerkverbindung konfigurieren
#!/bin/bash
sudo ip tuntap add tap0 mode tap
sudo ip addr add 172.20.0.1/24 dev tap0
sudo ip link set tap0 up

DEVICE_NAME=enp0s3

sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
sudo iptables -t nat -A POSTROUTING -o enp0s3 -j MASQUERADE
sudo iptables -A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
sudo iptables -A FORWARD -i tap0 -o enp0s3 -j ACCEPT
sudo ip addr add 172.20.0.1/24 dev tap0

# Firecracker in Terminal 1 starten
#!/bin/bash
rm -f /tmp/firecracker.socket

firecracker \
    --api-sock /tmp/firecracker.socket \

#Firecracker in Terminal 2
#!/bin/bash
curl --unix-socket /tmp/firecracker.socket -i \
-X PUT 'http://localhost/boot-source' \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
"kernel_image_path": "./hello-vmlinux.bin",
"boot_args": "console=ttyS0 reboot=k panic=1 pci=off"
}'

curl --unix-socket /tmp/firecracker.socket -i \
-X PUT 'http://localhost/drives/rootfs' \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
"drive_id": "rootfs",
"path_on_host": "./hello-rootfs.ext4",
"is_root_device": true,
"is_read_only": false
}'

curl --unix-socket /tmp/firecracker.socket -i \
-X PUT 'http://localhost/machine-config' \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
"vcpu_count": 1,
"mem_size_mib": 512
}'
```

```
curl --unix-socket /tmp/firecracker.socket -i \
-X PUT 'http://localhost/network-interfaces/eth0' \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
  "iface_id": "eth0",
  "guest_mac": "a2:96:04:dc:75:a1",
  "host_dev_name": "tap0"
}'

curl --unix-socket /tmp/firecracker.socket -i \
-X PUT 'http://localhost/actions' \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
  "action_type": "InstanceStart"
}'

#Im Gast System MAC-Adresse erfassen
#!/bin/bash
ifconfig eth0 up && ip addr add dev eth0 172.20.0.2/24
ip route add default via 172.20.0.1 && echo "nameserver 8.8.8.8" > /etc/resolv.conf
```

Anhang A 5. Wichtige Tools für die Installation von Kata Container

```
#!/bin/bash
# Wichtige Tools für die Installation von Kata Container

sudo apt-get update && pte-get upgrade
sudo apt-get install gcc
sudo apt install curl
sudo apt-get install flex
sudo apt-get install -y bison
sudo apt-get install libelf-dev
sudo apt-get install -y python3-simpleeval
sudo apt-get install python3
sudo apt-get install -y pkg-config
sudo apt-get install libglib2.0-dev
sudo apt-get install libpixmap-1-dev
```

Anhang A 6. CSV Datei für CPU

```
Durchlauf | KataContainer | Firecracker | DockerContainer (wird beim plotten wegge-
lassen)
1,totaltime:,44.7216,15.7989,26.9217
2,totaltime:,39.7975,16.0287,29.9154
3,totaltime:,32.0916,16.2886,25.8181
4,totaltime:,44.1692,16.2437,44.6201
5,totaltime:,44.0947,29.1637,38.8803
6,totaltime:,29.1214,29.8874,32.6317
7,totaltime:,42.2696,27.2782,44.1203
8,totaltime:,43.2580,31.6355,43.3279
9,totaltime:,34.5016,25.9607,29.3796
10,totaltime:,37.3852,14.7812,43.1258
11,totaltime:,43.8078,18.5145,43.1427
```



```
12, totaltime: 39.0521, 30.7640, 33.4642
13, totaltime: 32.7664, 29.6306, 38.1140
14, totaltime: 43.4334, 14.8777, 43.7146
15, totaltime: 28.5933, 16.8969, 38.2142
16, totaltime: 28.4297, 28.5868, 33.4431
17, totaltime: 28.6037, 30.2606, 43.4092
18, totaltime: 31.8891, 20.2853, 28.5042
19, totaltime: 33.7666, 14.8412, 28.3797
20, totaltime: 41.9213, 24.0811, 28.5179
21, totaltime: 36.0006, 31.0873, 31.7246
22, totaltime: 38.0205, 24.8814, 33.5470
23, totaltime: 15.9383, 15.2115, 41.8722
24, totaltime: 17.0055, 19.1876, 35.3179
25, totaltime: 20.2145, 30.2509, 37.8857
```

Anhang A 7. CPU Python Programm

```
# Autor: Vahel Hassan
# Abschlussarbeit: Leistungsbewertung von Container
# Benchmark Daten Plotten und berechnen
import matplotlib.pyplot as plt
import numpy as np

dateihandler = open('cpu_plotten.csv')

inhalt = dateihandler.read()

zeilen = inhalt.split('\n')

tabelle = []

for zeile in range(len(zeilen)):
    spalten = zeilen[zeile].split(',')
    tabelle.append(spalten)
    tabelle[zeile][2:] = [float(zahl) for zahl in tabelle[zeile][2:]]

kata_container = [zeile[2] for zeile in tabelle]
firecracker_container = [zeile[3] for zeile in tabelle]
docker_container = [zeile[4] for zeile in tabelle]

#Mittelwert
print("Der Mittelwert von Docker Container (Totaler Wert) liegt bei: %.2f" % np.mean(docker_container))
print("Der Mittelwert von Firecracker Container (Totaler Wert) liegt bei: %.2f" % np.mean(firecracker_container))
print("Der Mittelwert von Kata Container (Totaler Wert) liegt bei: %.2f" % np.mean(kata_container))

#Standardabweichung
print("Die Standardabweichung von Docker Container (Totaler Wert) liegt bei: %.2f" % np.std(docker_container, ddof=1))
print("Die Standardabweichung von Firecracker Container (Totaler Wert) liegt bei: %.2f" % np.std(firecracker_container, ddof=1))
print("Die Standardabweichung von Kata Container (Totaler Wert) liegt bei: %.2f" % np.std(kata_container, ddof=1))

data = [kata_container, firecracker_container, docker_container]

plt.title('Ergebnisse: CPU-Test')
plt.xlabel("Container")
plt.ylabel("Totaler Wert in Sekunden")

plt.boxplot(data)
plt.show()

#-----Auswertung von Firecracker-Container-----
#-----
ergebnisse_docker = \
    [321238/(docker_container[0]), 321238/(docker_container[1]), 321238/(docker_container[2]),
     321238/(docker_container[3]), 321238/(docker_container[4]), 321238/(docker_container[5]),
     321238/(docker_container[6]), 321238/(docker_container[7]), 321238/(docker_container[8]),
     321238/(docker_container[9]), 321238/(docker_container[10]), 321238/(docker_container[11]),
     321238/(docker_container[12]), 321238/(docker_container[13]), 321238/(docker_container[14]),
     321238/(docker_container[15]), 321238/(docker_container[16]), 321238/(docker_container[17]),
     321238/(docker_container[18]), 321238/(docker_container[19]), 321238/(docker_container[20]),
```

```

ner[23]), 321238/(docker_container[21]), 321238/(docker_container[22]), 321238/(docker_contai-
ner[23]),
321238/(docker_container[24])]

#-----Auswertung von Firecracker-Container-----
ergebnisse_firecracker = \
[321238/(firecracker_container[0]), 321238/(firecracker_container[1]), 321238/(firecra-
cker_container[2]),
321238/(firecracker_container[3]), 321238/(firecracker_container[4]), 321238/(firecra-
cker_container[5]),
321238/(firecracker_container[6]), 321238/(firecracker_container[7]), 321238/(firecra-
cker_container[8]),
321238/(firecracker_container[9]), 321238/(firecracker_container[10]), 321238/(firecra-
cker_container[11]),
321238/(firecracker_container[12]), 321238/(firecracker_container[13]), 321238/(firecra-
cker_container[14]),
321238/(firecracker_container[15]), 321238/(firecracker_container[16]), 321238/(firecra-
cker_container[17]),
321238/(firecracker_container[18]), 321238/(firecracker_container[19]), 321238/(firecra-
cker_container[20]),
321238/(firecracker_container[21]), 321238/(firecracker_container[22]), 321238/(firecra-
cker_container[23]),
321238/(firecracker_container[24])]

#-----Auswertung von Kata-Container-----
ergebnisse_kata = \
[321238/(kata_container[0]), 321238/(kata_container[1]), 321238/(kata_container[2]),
321238/(kata_container[3]), 321238/(kata_container[4]), 321238/(kata_container[5]),
321238/(kata_container[6]), 321238/(kata_container[7]), 321238/(kata_container[8]),
321238/(kata_container[9]), 321238/(kata_container[10]), 321238/(kata_container[11]),
321238/(kata_container[12]), 321238/(kata_container[13]), 321238/(kata_container[14]),
321238/(kata_container[15]), 321238/(kata_container[16]), 321238/(kata_container[17]),
321238/(kata_container[18]), 321238/(kata_container[19]), 321238/(kata_container[20]),
321238/(kata_container[21]), 321238/(kata_container[22]), 321238/(kata_container[23]),
321238/(kata_container[24])]

#Mittelwert
print("Der Mittelwert von Docker Container (Rechenoperationen) liegt bei: %.2f"% np.mean(ergebnisse_kata))
print("Die Mittelwert von Firecracker Container (Rechenoperationen) liegt bei: %.2f"% np.mean(ergeb-
nisse_firecracker))
print("Die Mittelwert von Kata Container (Rechenoperationen) liegt bei: %.2f"% np.mean(ergebnisse_docker))

#Standardabweichung
print("Die Standardabweichung von Docker Container (Rechenoperationen) liegt bei: %.2f"% np.std(ergeb-
nisse_kata, ddof=1))
print("Die Standardabweichung von Firecracker Container (Rechenoperationen) liegt bei: %.2f"% np.std(ergeb-
nisse_firecracker, ddof=1))
print("Die Standardabweichung von Kata Container (Rechenoperationen) liegt bei: %.2f"% np.std(ergebnisse_do-
cker, ddof=1))

data1 = [ergebnisse_kata, ergebnisse_firecracker, ergebnisse_docker]

plt.title('Ergebnisse: CPU-Test')
plt.xlabel("Container")
plt.ylabel("Rechenoperationen pro Sekunde")

plt.boxplot(data1)
plt.show()

```

Anhang A 8. CSV Datei für RAM

```

Durchlauf | KataContainer | Firecracker | DockerContainer (wird beim plotten wegge-
lassen)
1, 8811.76, 12918.40, 12195.62
2, 7437.72, 10921.80, 11991.45
3, 8550.27, 7953.22, 11571.40
4, 9239.54, 10864.56, 8024.11
5, 9287.28, 11219.07, 9004.17
6, 9443.67, 11209.14, 11693.18
7, 9371.69, 11703.00, 13084.89
8, 9592.20, 9798.41, 12913.66
9, 10546.95, 9111.94, 13066.23
10, 10518.15, 9140.12, 13011.20
11, 12424.42, 9122.03, 12966.20
12, 12015.21, 9350.39, 11815.09

```

```
13,12437.20,9270.56,9807.75
14,12887.03,8727.21,8593.12
15,11678.49,8167.83,8187.01
16,9264.62,8157.82,8659.22
17,9206.69,8287.49,8987.97
18,9256.39,8684.63,8451.08
19,9285.30,8505.59,7599.87
20,9184.25,9033.94,9161.15
21,8863.75,9951.28,9202.97
22,9237.74,13067.99,9242.92
23,12821.67,12955.79,9156.25
24,12984.14,13017.93,8879.65
25,13065.08,12881.44,9180.80
```

Anhang A 9. RAM Python Programm

```
# Autor: Vahel Hassan
# Abschlussarbeit: Leistungsbewertung von Container
# Matplotlib Daten Plotten und berechnen
import matplotlib.pyplot as plt
import numpy as np

dateihandler = open('ram_plotten.csv')

inhalt = dateihandler.read()

zeilen = inhalt.split('\n')

tabelle = []

for zeile in range(len(zeilen)):
    spalten = zeilen[zeile].split(',')
    tabelle.append(spalten)
    tabelle[zeile][1:] = [float(zahl) for zahl in tabelle[zeile][1:]]

kata_container = [zeile[1] for zeile in tabelle]
firecracker_container = [zeile[2] for zeile in tabelle]
docker_container = [zeile[3] for zeile in tabelle]

#Mittelwert
print("Der Mittelwert von Docker Container liegt bei: %.2f" % np.mean(docker_container))
print("Der Mittelwert von Firecracker Container liegt bei: %.2f" % np.mean(firecracker_container))
print("Der Mittelwert von Kata Container liegt bei: %.2f" % np.mean(kata_container))

#Standardabweichung
print("Die Standardabweichung von Docker Container liegt bei: %.2f" % np.std(docker_container))
print("Die Standardabweichung von Firecracker Container liegt bei: %.2f" % np.std(firecracker_container))
print("Die Standardabweichung von Kata Container liegt bei: %.2f" % np.std(kata_container))

data = [kata_container, firecracker_container, docker_container]

plt.title('Ergebnisse: RAM-Test')
plt.xlabel("Container")
plt.ylabel("Megabyte pro Sekunde")

plt.boxplot(data)
plt.show()
```

Anhang A 10. CSV Datei für Network

```
Durchlauf | KataContainer | Firecracker | DockerContainer (wird beim plotten wegge-
lassen)
1,52.8,806,53.2
2,57.5,799,53.3
3,37.6,800,55.1
4,53.8,743,52.8
5,34.8,833,56.1
6,56.1,838,52.9
7,29.7,834,58.6
8,53.8,561,56.3
```

```
9,53.0,807,53.7
10,53.7,752,54.8
11,27.7,766,55.7
12,55.3,838,52.0
13,54.1,815,53.2
14,56.7,834,53.2
15,51.7,814,53.1
16,52.6,840,54.2
17,55.9,791,55.7
18,51.5,790,46.9
19,50.3,846,55.8
20,53.3,805,56.2
21,54.7,807,51.7
22,55.6,848,54.4
23,54.0,839,37.5
24,57.2,852,57.0
25,56.0,826,55.4
```

Anhang A 11. Network Python Programm

```
# Autor: Vahel Hassan
# Abschlussarbeit: Leistungsbewertung von Container
# Matplotlib Daten Plotten und berechnen
import matplotlib.pyplot as plt
import numpy as np

dateihandler = open('network_plotten.csv')

inhalt = dateihandler.read()

zeilen = inhalt.split('\n')

tabelle = []

for zeile in range(len(zeilen)):
    spalten = zeilen[zeile].split(',')
    tabelle.append(spalten)
    tabelle[zeile][1:] = [float(zahl) for zahl in tabelle[zeile][1:]]

kata_container = [zeile[1] for zeile in tabelle]
firecracker_container = [zeile[2] for zeile in tabelle]
docker_container = [zeile[3] for zeile in tabelle]

#Mittelwert
print("Der Mittelwert von Docker Container liegt bei: %.2f" % np.mean(docker_container))
print("Der Mittelwert von Firecracker Container liegt bei: %.2f" % np.mean(firecracker_container))
print("Der Mittelwert von Kata Container liegt bei: %.2f" % np.mean(kata_container))

#Standardabweichung
print("Die Standardabweichung von Docker Container liegt bei: %.2f" % np.std(docker_container))
print("Die Standardabweichung von Firecracker Container liegt bei: %.2f" % np.std(firecracker_container))
print("Die Standardabweichung von Kata Container liegt bei: %.2f" % np.std(kata_container))

data = [kata_container, firecracker_container, docker_container]

plt.title('Ergebnisse: Network-Performance Test')
plt.xlabel("Container")
plt.ylabel("Megabyte pro Sekunde")

plt.boxplot(data)
plt.show()
```

Anhang A 12. CSV Datei für Startzeit

```
Durchlauf | KataContainer | Firecracker | DockerContainer (wird beim plotten wegge-
lassen)
1,026.625,029.180,028.475
2,023.902,027.296,026.240
3,024.689,023.060,025.649
4,027.086,021.903,025.377
```

```
5,025.397,024.759,023.456
6,022.446,024.848,021.562
7,022.142,024.790,022.825
8,026.361,032.097,031.314
9,028.785,025.022,031.028
10,032.139,022.172,029.043
11,032.056,026.633,026.831
12,029.034,029.333,020.997
13,023.105,041.037,029.810
14,030.098,035.005,036.508
15,034.728,032.314,038.679
16,032.933,025.767,034.933
17,038.824,022.689,036.217
18,044.079,030.655,039.560
19,036.488,039.071,029.985
20,023.684,037.298,018.406
21,018.607,036.084,023.264
22,026.534,033.763,034.068
23,034.586,033.857,038.775
24,034.341,028.603,036.880
25,031.839,022.619,031.933
```

Anhang A 13. Startzeit Python Programm

```
# Autor: Vahel Hassan
# Abschlussarbeit: Leistungsbewertung von Container
# Matplotlib Daten Plotten und berechnen
import matplotlib.pyplot as plt
import numpy as np

dateihandler = open('startzeit_plotten.csv')

inhalt = dateihandler.read()

zeilen = inhalt.split('\n')

tabelle = []

for zeile in range(len(zeilen)):
    spalten = zeilen[zeile].split(',')
    tabelle.append(spalten)
    tabelle[zeile][1:] = [float(zahl) for zahl in tabelle[zeile][1:]]

kata_container = [zeile[1] for zeile in tabelle]
firecracker_container = [zeile[2] for zeile in tabelle]
docker_container = [zeile[3] for zeile in tabelle]

#Mittelwert
print("Der Mittelwert von Docker Container liegt bei: %.2f" % np.mean(docker_container))
print("Der Mittelwert von Firecracker Container liegt bei: %.2f" % np.mean(firecracker_container))
print("Der Mittelwert von Kata Container liegt bei: %.2f" % np.mean(kata_container))

#Standardabweichung
print("Die Standardabweichung von Docker Container liegt bei: %.2f" % np.std(docker_container, ddof=1))
print("Die Standardabweichung von Firecracker Container liegt bei: %.2f" % np.std(firecracker_container, ddof=1))
print("Die Standardabweichung von Kata Container liegt bei: %.2f" % np.std(kata_container, ddof=1))

data = [kata_container, firecracker_container, docker_container]

plt.title('Ergebnisse: Startzeit-Test')
plt.xlabel("Container")
plt.ylabel("Sekunden")

plt.boxplot(data)
plt.show()
```

9 Ehrenwörtliche Erklärung

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

Leistungsbewertung von VM-basierten Containerlösungen


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

Ludwigshafen am Rhein

Ort

18.08.2020

Datum



Unterschrift