Hochschule der Medien

Bachelorarbeit

Sicherheitsbetrachtungen von Applikations-Containersystemen in Cloud-Infrastukturen am Beispiel Docker

Moritz Hoffmann

Studiengang: Mobile Medien
Matrikelnummer: 26135
E-Mail: mh203@hdm-stuttgart.de

Dezember 2015

Erstbetreuer:
Prof. Dr. Joachim Charzinski
Hochschule der Medien

Zweitbetreuer:
Patrick Fröger
ITI/GN, Daimler AG

Sicherheitsbetrachtungen von Applikations-Containersystemen in Cloud-Infrastukturen am Beispiel Docker

Moritz Hoffmann
Studiengang Mobile Medien,
Hochschule der Medien
mh203@hdm-stuttgart.de

Dezember 2015

Eidesstattliche Erklärung

„Hiermit versichere ich, Moritz Hoffmann, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit mit dem Titel: „Sicherheitsbetrachtungen von Applikations-Containersystemen in Cloud-Infrastukturen am Beispiel Docker“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden. Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§26 Abs. 2 Bachelor-SPO (6 Semester), § 24 Abs. 2 Bachelor-SPO (7 Semester), § 23 Abs. 2 Master-SPO (3 Semester) bzw. § 19 Abs. 2 Master-SPO (4 Semester und berufsbegleitend) der HdM) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.“

Unterschrift	Datum

Abstract

English version:

....
..

Deutsche Version:

....
...
.
..

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abbildung 1: Awesome Image

lkasjdflkj asldkjf lasjkdflkadsjf ladksjflkjslkdjf dslfjklaks df a sdfjaldsfj ladksjf lkjlakjsd f asdf aljsdflkjasldfjalsdfj l adskjflj d f dslkfjalksdjf sd fljsdfkjsld f

dieser text ist kursiv

asdfasdfasdfasdlkvalrkgjval asdkfj sldkfjlsdjfa adaher is kes ji lkaskdj ladskj a ldksfjll aldkfj lkj afsdlfkjl alsdkf jaldskfj la sdflaldsflas df sadfl sf

```
das hier ist monotype
```

Kapitel 1

Überblick

Trotz ein paar intrinsischen Schwächen von Containerlösungen, werden Container bereits in einer Vielzahl von Szenarien eingesetzt [5, S.6].

Container sind in Infrastrukturen, in denen es auf Skalierbarkeit ankommt, trotz Sicherheitsbedenken beliebt. Vor allem "Multi-TenantSServices werden gerne mit Docker eingesetzt.

1.1 Arten von Virtualisierungen

1.1.1 Hypervisor-basierte Virtualisierung

Bei Hypervisor-basierten Systemen laufen unabhängig voneinander eine oder mehrere Maschinen virtuell auf physischer Hardware (in der englischen Literatur auch "bare metal"genannt). Der Hypervisor nimmt dabei die Rolle eines Vermittlers zwischen Host-OS und Gast-OS ein [5, S.6].

1.1.2 Container-basierte Virtualisierung

(Benötigt keine Emulations- oder Hypervisorschicht [5, S.7].)

Container-Virtualisierung wird oftmals Virtualisierung auf Betriebsystemebene genannt [5, S.6].

Containerlösungen umfassen die Technologien *OpenVZ*, *Solaris Zones*, sowie Linux-Container wie *LXC* [5, S.7] und *Docker*, welches im Fokus dieser Arbeit steht.

Moderne Container können als vollwertige Systeme betrachtet werden, nicht mehr als ursprünglich vorgesehen, reine Ausführungsumgebungen [5, S.7].

In Container-basierten Systemen hingegen, laufen die Container im Üser Space"direkt auf dem Kernel des Host-OS und nutzen dessen *System Call*-Interface [5, S.6+7]. Dadurch kommt es im Vergleich zu Hypervisor-Virtualisierungen zu einem erheblichen Performancegewinn, da der Virtualisierungs-Overhead

des Hypervisors wegfällt. In der Praxis macht das eine hohe Dichte an Containern auf einem Host und dadurch indirekt eine bessere Resourcenausnutzung möglich [5, S.7+8].

Container-Lösungen erlauben es, mehrere voneinander isolierte User Space-Instanzen parallel auf einem einzigen physischen Host zu betreiben [5, S.6]. Dadurch, dass ein Hypervisor in einer solchen Konfiguration nicht existiert und die Container direkt Hostkernel-Features nutzen, gibt es einen entscheidenden Nachteil für Containerlösungen - und damit auch Docker - gegenüber Hypervisor-basierter Virtualisierung: Das Container-Betriebssystem muss wie das Host-Betriebsystem Linux-basiert sein. In einem Host auf dem Ubuntu Server installiert ist, können nur weitere Linux-Distributionen als Container laufen. Ein Microsoft Windows kann also nicht als Container auf genannten Host gestartet werden, da die Kernel miteinander nicht kompatibel sind [5, S.6]. Diese Inflexibilität im Spektrum der einsetzbaren Betriebssysteme liegt den Containerlösungen zugrunde.

Außerdem werden Container als weniger sicher im Vergleich zur Hypervisorgestützen Virtualisierung gesehen [5, S.6].

Hingegen muss in containerbasierten Systemen nicht das gesamte Betriebssystem virtualisiert werden, da von den Containern direkt auf den Host-Kernel zugegriffen wird. Zum Einen schrumpft dadurch die Angriffsfläche des Hosts [5, S.6], da, wie später noch zu sehen ist, die Zugriffsrechte der Container auf den Host sehr feingranular festgelegt werden müssen. Zum Anderen entsteht durch diese Tatsache ein Risko im Design, weil Host-Features ohne Hypervisor direkt genutzt werden.

1.1.3 Einordnung Docker

Docker nutzt moderne Linux-Kernelfeatures, wie z.b. *control groups* und *namespaces*, um ein Resourcenmanagement zwischen Containern und eine effektive Isolierung der Container vom Hostsystem zu realisieren [5, S.7].

Docker ist wie in KAPITEL-ZUVOR angedeutet, nicht die erste containerbasierte Virtualisierungslösung. Einige ältere Containersysteme, wie z.B. *Solaris Zones*, gibt es schon viel länger als Docker, aber wurden allerdings nie von der Industrie als Lösung akzeptiert. Worauf beruht also der Erfolg von Docker in den letzten Jahren? Dieser Frage werde ich im folgenden Kapitel nachgehen.

1.2 Einführung in Docker

Docker ist eine unter der Apache 2.0 Lizenz veröffentlichte Open-Source Engine, die den Einsatz von Anwendungen in Containern automatisiert. Es ist überwiegend in der Programmiersprache *Golang* von dem Unternehmen *Docker, Inc.* (vormals *dotCloud Inc.*), implementiert [1][5, S.7].

Der große Vorteil von Docker gegenüber älteren Containerlösungen ist das Level an Abstraktion und die Bedienungsfreundlichkeit, die Nutzern ermöglicht wird. Während sich Lösungen vor Docker auf dem Markt durch deren schwierige Installation und Management sowie schwachen Automatisierungsfunktionen nicht etablieren konnten, addressiert Docker genau diese Schwachpunkte [5, S.7] und bietet neben Containern viele Tools, die die Arbeit mit Containern und weiteren Modulen, erleichtern.

Quellcode kann samt virtualisierter Ausführungsumgebung flexibel von einem Laptop auf einen Testserver und später auf einen Produktionsserver geschoben werden. Dieser kurzlebige Zyklus zwischen Entwicklung, Testen und Deployment erlaubt einen effizienten Workflow [5, S.8].

Eine weitere wichtige Eigenschaft von Docker ist Konsistenz: Die Umgebungen, in denen Softwareentwickler Code schreiben, sind identisch mit den Umgebungen, die später auf Servern laufen. Die Wahrscheinlichkeit, dass ein Fehler erst im Betrieb auftritt, nicht aber in der Entwicklung, wird dadurch sehr klein gehalten [5, S.8].

Wenn wie von Docker empfohlen in jedem Container nur eine Anwendung läuft, begünstigt das eine moderne Service-orientierte Architektur mit *Microservices*. Nach dieser Architektur werden Anwendungen oder Services verteilt zur Verfügung gestellt und durch eine Serie an miteinander kommunizierenden Containern umgesetzt. Der Grad an Modularisierung der dadurch ensteht, kann für die Verteilung, die Skalierung und das Debugging von Service- oder Anwedungskomponeten (Container) gewinnbringend eingesetzt werden [5, S.9].

Die Eigenheiten von Docker sowie die gängige Begrifflichkeiten im Docker-Ökosystem, werden in den folgenden Unterkapiteln genauer beleuchtet.

1.2.1 Container

Der Begriff „Container“ ist bisher schon oft gefallen, deswegen will ich auf ihn zuerst eingehen.

Docker-Container beinhalten eine idealerweise minimale Laufzeitumgebung, in der eine oder mehrere Anwendungen laufen.

Basiert auf einem *Copy-on-write*-Modell, das Änderungen der Anwendung schnell handhabt [5, S.8].

1.2.2 Images

Images liegen Containern als statische Files zugrunde. Container werden auf der Basis von Images gestartet. Images sind durch das eingesetzte *Union-Filesystem* in Schichten gegliedert, die zusammen ein Image ergeben, das als Container gestartet werden kann [5, S.11].

Images werden Schritt für Schritt erstellt, z.B. mit den folgenden Aktionen [5, S.11]:

- Eine Datei hinzufügen
- Ein Kommando ausführen, z.B. ein Tool mittels des Paketmanagers `apt` installieren
- Einen Port öffnen, z.B. den Port 80 für einen Webserver

Images sind einfach portierbar und können geteilt, gespeichert und aktualisiert werden [5, S.11].

Auf der Basis von existierenden Images können durch das Hinzufügen neuer Schichten durch oben beschriebene Aktionen, neue Images erstellt werden (EIGENES STATEMENT).

Über die Kommandozeile kann z.B. das Image eines *Nginx*-Webservers von der öffentlichen Docker-Registry mit dem Befehl `docker pull nginx` auf die lokale Maschine gespeichert werden.

1.2.3 Registries

Docker stellt eine Vielzahl an Images öffentlich in einer eigenen Registry, dem Docker Hub [5, S.11], zur Verfügung. Für dieses System kann jeder Nutzer einen Account anlegen und eigenständig Images hochladen. Das Docker-Hub bietet bereits mehr als 150.000 Repositories, die etwa 240.000 Nutzer zusammenstellten und hochluden, zur freien Verwendung an (Stand Juni 2015) [3, S.16]. Die Einträge im Hub können von Nutzern bewertet werden. Außerdem wird angezeigt, wie oft ein Image bereits über das Hub bezogen wurde.

Ein Repository besteht aus mindestens einem Image. Um Images in einem Repository voneinander zu unterscheiden, werden Images Tags zugewiesen, um beispielweise mehrere Versionen eines Images in einem Repository zu kennzeichnen. Die Images werden nach dem Schema `<repository>:<tag>` identifiziert. So gibt es z.B. im offiziellen Repository des Webservers *Nginx* Images mit den Tags `latest`, `1`, `1.9` und `1.9.9` [2]. Wenn bei dem Download kein Tag angegeben ist, wie in Kapitel wird automatisch das aktuellste Image `latest` bezogen, wie es im letzten Kapitel „Images“ praktiziert wurde.

1.2.4 Dockerfile

1.2.5 Client und Server

Docker selbst ist nach einem Client-Server-Modell aufgebaut: Ein Docker-Client kommuniziert mit einem Docker-Daemon, also ein Prozess der den Server abbildet. Beide Teile können auf einer Maschine oder einzeln auf unterschiedlichen Hosts laufen. Wie in Abb.2 zu sehen ist, sind auch mehrere Docker-Clients möglich.

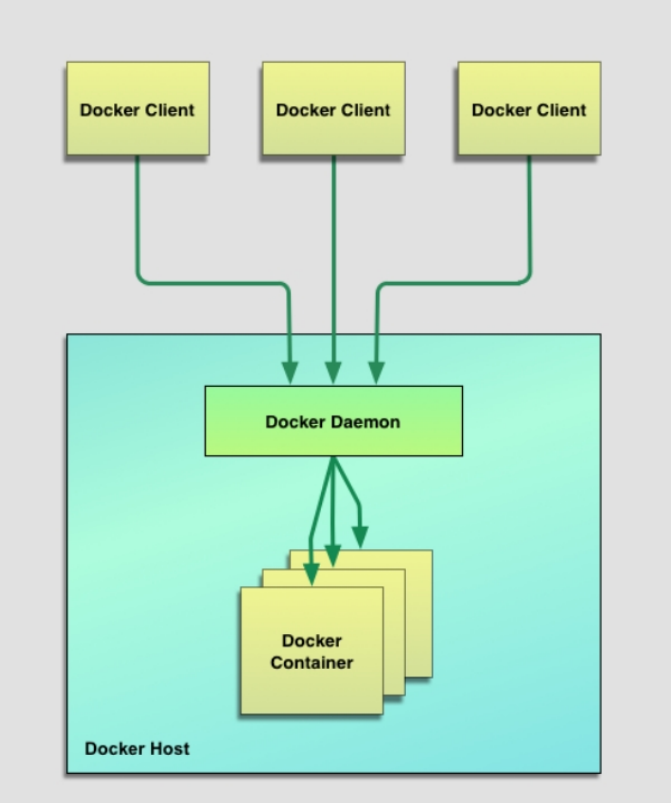

Abbildung 2: Die Client-Server-Architektur von Docker [5, S.10].

Kapitel 2

Ziel der Arbeit/Forschungsfrage

Kapitel 3

Security aus Linux Kernel-Features

3.1 Isolierung

3.1.1 `namespaces`

3.1.1.1 `user namespaces`

3.1.2 `capabilities`

3.1.2.1 Beispiele, `/proc`-Verzeichnis, (Un-)Mounten des Host-Filesystems

3.1.3 Mandatory Access Control (MAC)

3.1.3.1 Beispiel SELinux

3.1.3.2 AppArmor

3.2 Ressourcenverwaltung

3.2.1 `cgroups`

3.3 Docker im Vergleich zu anderen Containerlösungen

Kapitel 4

Security im Docker-Ökosystem

4.1 Docker Images und Registries

4.1.1 neues Signierungs-Feature

4.2 Docker Daemon

4.2.1 REST-API

4.2.2 Support von Zertifikaten

4.3 Containerprozesse

4.4 Docker Cache

4.5 `privileged` Container

4.6 Networking

4.6.1 `bridge` Netzwerk

4.6.2 `overlay` Netzwerk

4.6.3 DNS

4.6.4 Portmapping

4.7 Daten-Container

4.8 Docker mit VMs

4.9 Sicherheitskontrollen für Docker

4.10 Tools rund um Docker

4.10.1 Docker Swarm

4.10.2 Docker Compose

4.10.3 Nautilus Project

4.10.4 Vagrant

Kapitel 5

Docker in Unternehmen/Cloud-Infrastrukturen

Kapitel 6

Fazit/Ausblick

Literaturverzeichnis

[1] Github repository der docker engine. über Website `https://github.com/docker/docker` , aufgerufen am 11.01.2016.

[2] Offizielles repository des webservers nginx. über Website `https://hub.docker.com/_/nginx/` , aufgerufen am 11.01.2016.

[3] Slides of keynote at dockercon in san francisco - day 2. über Website `de.slideshare.net/Docker/dockercon-15-keynote-day-2/16` , aufgerufen am 11.01.2016.

[4] Jérôme Petazzoni. Containers, docker, and security: State of the union. über Website http://de.slideshare.net/jpetazzo/containers-docker-and-security-state-of-the-union-bay-area-infracoders-meetup , aufgerufen am 22.12.2015, October 2015.

[5] James Turnbull. *The Docker Book*. 1.2.0 edition, September 2014.