Dualer Bachelorstudiengang Wirtschaftsingenieurwesen

Studiengang: BWIW8-INF

Bachelor-Thesis

Vergleich von Virtualisierungstechnologien für den Betrieb von IT-Services

eingereicht von: Vincent Pelz

Praxisbetrieb: GISA GmbH

betrieblicher Betreuer: Herr M. Sc. Matthias Friedrich

Hochschulbetreuer: Herr Prof. Dr. Sven Karol

Abgabetermin: 02. März 2023

Unterschrift Matthias Friedrich

E-Mail: Matthias.Friedrich@gisa.de

Unterschrift Vincent Pelz

E-Mail: Vincent.Pelz@gisa.de

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen Bedanken, die mich bei der Erstellung dieser Bachelor-Thesis unterstützt haben.

Ich möchte mich insbesondere bei Herrn Matthias Friedrich bedanken, für die fachliche Unterstützung, das Ausfüllen der Umfrage für die Aufwandsschätzung und das wiederholte Korrekturlesen, sowie bei Prof. Dr. Sven Karol für die hilfsbereite Betreuung.

Ich möchte mich zudem bei Jonathan Stapf und Matthias Pabst bedanken, welche mich ebenfalls durch fachliche Unterstützung und das Ausfüllen der Umfrage unterstützt haben, sowie bei Johannes Bartsch und Kevin Wagner, für das Korrekturlesen dieser Arbeit.

Abschließend möchte ich mich bei meinen Eltern bedanken, die mich bei meinem Studium und meiner beruflichen Laufbahn stets unterstützt haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Quelltextverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ABAC	Attribute-based access control
API	Application Programming Interface
BPM	Business Process Management
cgroups	Control Group
CLI	Command Line Interface
CRD	Custom Resource Definitions
CSI	Container Storage Interface
CVE	Common Vulnerabilities and Exposures
DAC	Discretionary Access Control
DBMS	Datenbank-Managementsystem
DMC	Daten Management Center
DOS	Denial-of-Service
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KVM	Kernel-Based Virtual Machine
LSM	Linux Security Modules
MAC	Mandatory Access Control
PID	Process-ID
PV	PersistentVolume
PVC	PersistentVolumeClaim
RBAC	role-based access control
REST API	Restful API
Seccomp	Secure Computing Mode
SELinux	Security-Enhanced Linux
TPS	Transaktionen pro Sekunde
TLS	Transport Layer Security

TCP	Transmission Control Protocol
union FS	Union File Systems
VM	Virtuelle Maschine
YAML	YAML Ain't Markup Language

1 Einleitung

1.1 GISA GmbH

Die GISA GmbH ist ein IT-Komplettdienstleister mit Hauptsitz in Halle (Saale), zu dessen Leistungsportfolio die IT-Beratung, IT-Betreuung und der Betrieb von IT-Lösungen im hauseigenen Rechenzentrum gehört [1]. Sie entstand 1993 als Ausgründung der IT-Abteilungen der Energieversorgungsunternehmen Mitteldeutsche Energieversorgung AG, Stadtwerke Halle GmbH, Erdgasversorgung Westsachsen GmbH und der Gasversorgung Sachsen-Anhalt GmbH. Seit 2014 ist die NTT DATA Business Solutions AG, eine hundertprozentige Tochter der NTT DATA, neuer Haupteigentümer der GISA GmbH [2].

Bis heute ist die GISA GmbH besonders in der Energiewirtschaft und bei öffentlichen Auftraggebern erfolgreich, indem sie mit langjähriger Branchenerfahrung und zahlreichen Zertifizierungen überzeugt [1].

Die GISA GmbH entwickelt und betreibt unter anderem die Systemlandschaft *Daten Management Center* (DMC) [3], welche den Austausch elektronischer Daten zwischen Geschäftspartnern ermöglicht. Die Aufgaben des DMC beinhaltet neben dem Versand und dem Speichern der Daten im Zielsystem zum Beispiel auch das Validieren, Konvertieren und Archivieren dieser. Zusätzlich dazu existieren Anwendungen, welche das Durchsuchen und Verwalten der gespeicherten Nachrichten und deren Verarbeitung ermöglicht [3].

1.2 Ausgangssituation

Bisher wurden in der GISA GmbH im Umfeld des DMC für den Betrieb von IT-Services virtuelle Maschinen und die Containertechnologie Docker eingesetzt. Zusätzlich dazu wurde zuletzt ein Cluster der Container-Orchestrierungs-Lösung Kubernetes aufgebaut, auf dem bereits die ersten Services laufen.

Die Vielzahl an zur Verfügung stehenden Virtualisierungstechnologien bietet zwar ein hohes Maß an Flexibilität, wirft allerdings auch die Frage auf, welche Vor- und Nachteile diese einzelnen Technologien mit sich bringen. Da diese Frage bis jetzt nicht detailliert betrachtet wurde, herrscht keine Klarheit, wann welche dieser Technologien für den Betrieb eines IT-Service ausgewählt werden sollte.

1.3 Zielsetzung der Bachelor-Thesis

Ziel dieser Arbeit ist es, die zurzeit im DMC-Umfeld der GISA GmbH eingesetzten Virtualisierungstechnologien zunächst hinsichtlich Funktionsweise und Konfiguration voneinander abzugrenzen.

Anschließend sollen die Technologien anhand des Aufwands, der Ressourcennutzung und der Performance, sowie der IT-Sicherheit betrachtet werden. Zur Feststellung der Aufwände soll eine Umfrage erstellt werden, in der ein Entwickler oder Betreuer des jeweiligen Themenbereichs den Aufwand für ein Set an gegebenen Aufgaben schätzt. Für die Ermittlung der Performance sollen mehrere Leistungsparameter anhand von verschiedenen Performancetests betrachtet werden. Da die im DMC-Umfeld betriebenen Services, bis auf wenige Ausnahmen, Linux-basiert sind [Anhang W], soll sich der Vergleich auf diese fokussieren.

Basierend auf den im Vergleich erarbeiteten Erkenntnissen soll final eine Entscheidungshilfe für die Auswahl der Virtualisierungstechnologie von bestehenden und zukünftigen IT-Services erstellt werden. Die entstandene Entscheidungshilfe soll innerhalb dieser Arbeit beispielhaft auf eine Angular-basierte Webanwendung [4] und die Business Process Management Software inubit BPM [5] angewendet werden. Angular und inubit BPM werden in Abschnitt 3.3 noch genauer eingeführt.

2 Grundlagen

Bei der Bereitstellung von IT-Services ist die Virtualisierung allgegenwärtig. Unter Virtualisierung versteht man in der Informatik die Abstraktion entweder physikalischer Ressourcen, wie z. B. der Prozessor oder Arbeitsspeicher, oder einer Software, wie z. B. ein Betriebssystem, mithilfe von Software [6]. Die wesentlichen Ziele von Virtualisierung sind eine Verbesserung der Flexibilität, Verfügbarkeit und Auslastung der Ressourcen [6]. Die Unterschiede der beiden für das Projektumfeld relevanten Virtualisierungsarten, Hardwarevirtualisierung und Betriebssystemvirtualisierung, werden fort folgend erläutert.

2.1 Hardwarevirtualisierung

Die nachfolgende Einführung in die Hardwarevirtualisierung basierend auf einem technischen Bericht des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam [6].

Bei der Hardwarevirtualisierung werden physikalische Computer zu einer oder mehreren virtuellen Maschinen (VM) [6] abstrahiert. Zwischen den VMs und dem darunterliegenden System liegt eine Softwareschicht, welche als *Hypervisor* bezeichnet wird und die Aufgabe hat, die VMs zu koordinieren und die Systemressourcen zu verwalten.

Grundsätzlich unterscheidet man zwischen zwei Hypervisor Arten. Dem Type 1 Hypervisor, welcher auch als *Bare Metal* bezeichnet wird und direkt über der Hardware läuft und dem Type 2 Hypervisor, welcher auch als *Hosted* bezeichnet wird und zwischen einem auf der physikalischen Hardware installieren Host-Betriebssystem und den VMs läuft (siehe Abbildung 1).

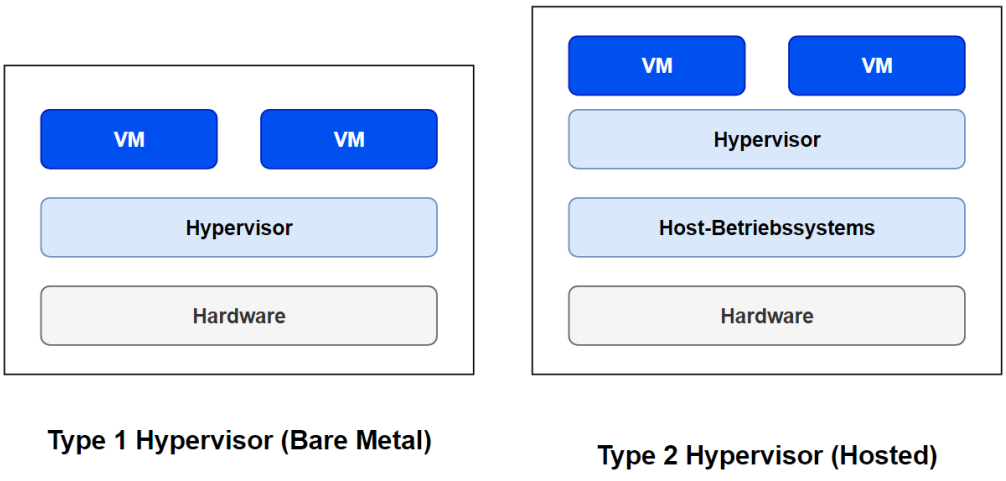


Abbildung 1: Type 1 (bare-metal) und Type 2 (hosted) Hypervisor; in Anlehnung an [6]

Die Bare-Metal-Variante zeichnet sich gegenüber der Hosted-Variante durch eine bessere Performance aus, da der Hypervisor direkt mit der Hardware kommunizieren kann und der Overhead des Host-Betriebssystems wegfällt. Allerdings muss die Hardware die Bare-Metal-Virtualisierung unterstützen, was bei der Hosted-Variante nicht vonnöten ist und nicht jede Hardware besitzt einen passenden Bare-Metal Treiber, sodass die Treiber stattdessen auf dem Host-Betriebssystem der Hosted-Variante installiert werden müssen. Die VMs können dann über standardisierte Systemaufrufe mit der Hardware kommunizieren. Der Hosted-Ansatz ist zudem während der Entwicklung von Vorteil, da die VMs direkt auf dem Entwicklungsrechner gestartet werden können und somit keine zusätzliche Hardware benötigt wird.

Da bei beiden Hypervisor Arten die Hardware virtualisiert wird, ist es möglich, in den VMs beliebige Betriebssysteme zu installieren.

Für die Isolation der VMs setzen beide Hypervisor-Typen auf das in Abbildung 2 dargestellte Ringkonzept der Intel x86 Architektur. So läuft der Hypervisor, bzw. der Kernel des Host-Betriebssystems im Ring 0, dem sogenannten Kernelmode, welcher uneingeschränkten Zugriff auf das ganze System hat. Dazu zählt insbesondere der Zugriff auf den gesamten Speicher und der direkte Zugriff auf die Hardware. Die übergeordneten Ringe haben hingegen nur eingeschränkten Zugriff auf das System. Die VM läuft üblicherweise im Ring 3, dem User-Space, während die Ringe 1 und 2 bis auf einige Ausnahmen unbenutzt bleiben. Für kritische Systemaufrufe besitzen moderne x86 Server-CPUs fast ausnahmslos eine Befehlssatzerweiterung, welche es ihnen ermöglicht, diese isoliert auszuführen. Man spricht von der hardwareunterstützten Virtualisierung. Sollte diese Befehlssatzerweiterung nicht vorliegen, müssen jegliche kritische Systemaufrufe der VMs durch den Hypervisor abgefangen, angepasst und stellvertretend ausgeführt werden. Dieser Vorgang wird als *binary translation* bezeichnet [6]. Die hardwareunterstützte Virtualisierung bietet gegenüber der binary translation u. a. eine verbessere CPU-Performance.

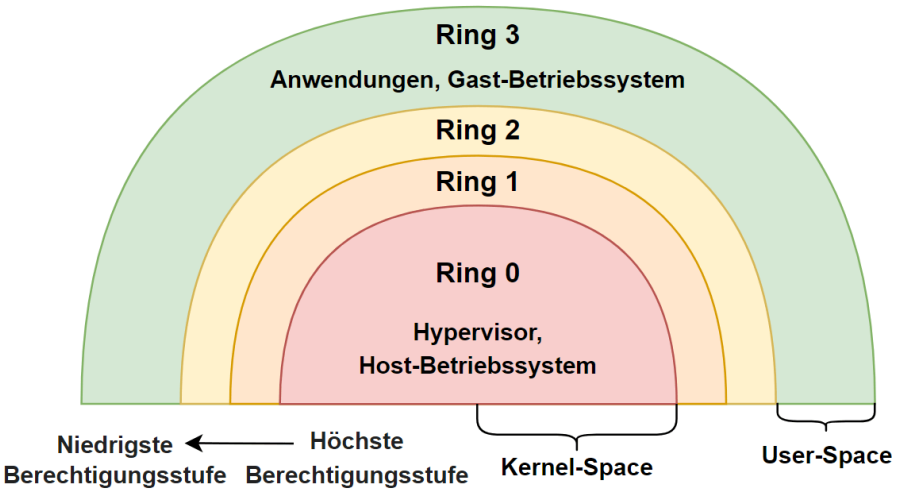


Abbildung 2: Ringkonzept für Zugriffsberechtigungen der Intel x86 Architektur

In der GISA GmbH kommen für den Betrieb von IT-Services in VMs die Virtualisierungslösungen VMWare vSphere und KVM/Ganeti zum Einsatz [Anhang W]. Die Verwaltung und der Betrieb dieser beiden Lösungen wird von spezialisierten Teams außerhalb des DMC-Umfelds übernommen. Nachfolgend werden beide Technologien kurz vorgestellt.

2.2 Virtualisierung mittels VMware vSphere/ESXi

VMware ESXi ist ein Type 1 Hypervisor der Firma VMware, Inc. und Teil der Softwarelösung vSphere [7], welche neben VMs noch weitere Komponenten bereitstellt und es z. B. ermöglicht, weite Teile der Netzwerkinfrastruktur zu virtualisieren [8].

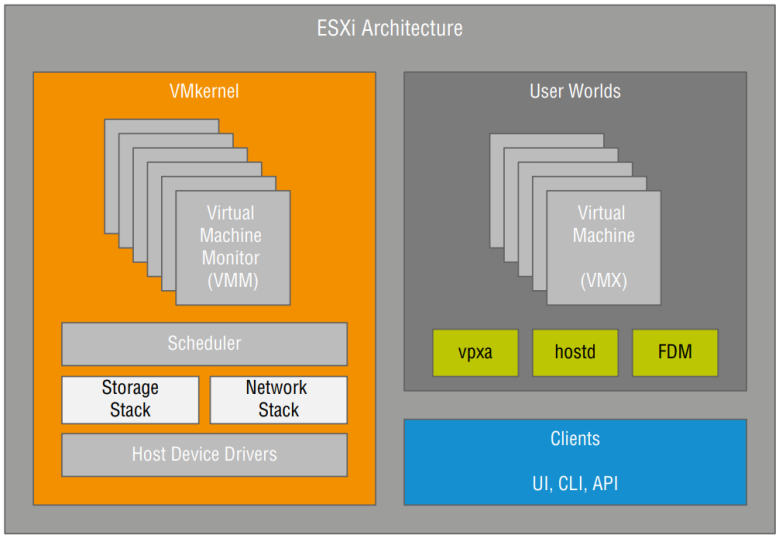


Abbildung 3: Die Architektur des VMware ESXi Hypervisors [9]

Der zentrale Bestandteil der in Abbildung 3 dargestellten Architektur des ESXi-Hypervisors ist der *Vmkernel* [9]. Dieser besteht unter anderem aus dem *Virtual Machine Monitor* (VMM) [9] und dem *Ressouren Manager*. Letzterer ist für die Aufteilung der Hardware an die VMs, unter Berücksichtigung von Reservierungen und Limits, zuständig. Der VMM ist für die Verwaltung der VMs und die Weiterleitung der Ressourcen zuständig und leitet dafür alle Anfragen entweder an den VMkernel oder den *Virtual Machine Executable Prozess* (VMX) [9] weiter. Der VMX-Prozess ist ein Hilfsprozess, der auf einem ESXi-Host ausgeführt wird und der für die Verwaltung bestimmter Aspekte des Betriebs einer VM verantwortlich ist, einschließlich der Eingabe- und Ausgabe von Tastatur, Maus und Bildschirm, der Konsole und einiger nicht kritischer Eingabe-/Ausgabeoperationen wie CD-ROM. Für jede virtuelle CPU einer VM existiert ein separater VMM-Prozess. Letztlich beinhaltet der VMkernel noch die Treiber für z. B. die Hardware, Speicher und Netzwerkkommunikation.

Neben dem privilegierten VMkernel beinhaltet der ESXi-Hypervisor auch einen unprivilegierten Teil, in dem unter anderem für jede VM ein VMX-Prozess und ein Dienst namens *hostd* läuft. Hostd ist ein Proxy-Dienst, welcher den Hauptkommunikationskanal zwischen den zentralen ESXi-Verwaltungsdiensten, wie dem *VirtualCenter Management Server*, und den VMkernels der einzelnen Hosts darstellt.

Neben dem grundlegenden ESXi-Hypervisor zur Bereitstellung von VMs bietet vSphere laut dem vSphere – Datenblatt [10] noch weitere Funktionen:

VMotion: Ermöglicht die Migration von VMs zwischen ESXi-Hosts ohne Ausfallzeit

High Availability: Startet VMs automatisch auf einem anderen ESXi-Host neu, sollte der ursprüngliche Host nicht mehr erreichbar sein oder ein Neustart der VM aus einem anderen Grund nötig sein.

Distributed Resource Scheduler: Sorgt für einen Lastausgleich, indem VMs unter Nutzung von VMotion von einem stark belasteten ESXi-Host auf einen anderen Host migriert werden, der über genügend Rechenressourcen verfügt.

Die vSphere Dokumentation für die VM Administration [11] beschreibt, dass Systemressourcen zu einem sogenannten Pool zusammengefasst und anschließend anhand bestimmter Regeln auf die einzelnen VMs verteilt werden. So kann zum einen eine Reservierung angelegt werden, die einer VM eine bestimmte Menge an Systemressourcen garantiert und zum anderen ein Limit, welches den maximalen Systemressourcenverbrauch einschränkt. Die Summe der Limits aller VMs kann dabei größer sein, als tatsächlich Ressourcen zur Verfügung stehen.

Sollten mehrere VMs um Ressourcen konkurrieren, d. h. mehr Ressourcen anfordern als zur Verfügung stehen, kommt eine weitere VM-Einstellung, die sogenannten *Shares* (Anteilswerte), zum Tragen, welche die Priorität einer App bei der Ressourcenzuordnung in Form eines relativen Anteils definieren. Die konkurrierenden VMs teilen sich die zur Verfügung stehenden Ressourcen entsprechend dem Verhältnis ihrer Shares zueinander. Eine VM mit doppelt so vielen Shares wie eine zweite VM darf somit im Konkurrenzfall doppelt so viele Systemressourcen beanspruchen.

Die Dokumentation beschreibt ebenfalls, dass für die Cluster und VM-Verwaltung der *vSphere Client* eingesetzt wird, welcher sich mit dem vCenter Server über ein Application Programming Interface (API) [12] verbindet und die administrativen Funktionen über eine Web-Oberfläche bereitstellt. Optional ist der Client dank seiner Plug-in-basierten Architektur erweiterbar. Um eine neue VM zu erstellen, kann ein entsprechender Wizard über den vSphere Client verwendet werden. Innerhalb dieses Wizzard können alle VM-Parameter, wie Name, Festplatten, aber auch CPU und Speicherbegrenzungen, festgelegt werden. Abschließend kann aus der VM ein Template

generiert werden, um eine VM mit ähnlichen Konfigurationsparametern mit geringerem Aufwand zu erstellen. Alternativ ist es auch möglich eine existierende VM zu kopieren.

2.3 Virtualisierung mittels KVM/Ganeti

KVM (Kernel-Based Virtual Machine) ist laut dem Dozentenhandbuch der Initiative "IT-Sicherheit in der Wirtschaft" des Bundesministeriums für Wirtschaft und Energie ein Type 2 Hypervisor, welcher als Modul fester Bestandteil des Linux Kernels ist [12]. Ergänzt wird KVM durch die Emulationssoftware QEMU [13] für die Bereitstellung virtueller Hardware wie z. B. Netzwerkadapter und dem Virtualisierungsframework VirtIO [14] für den Zugriff auf reale IO-Schnittstellen wie Festplatten und Netzwerkadapter.

Ganeti ist laut [15] eine Management-Software für VM-Cluster, welche auf Virtualisierungstechnologien wie z. B. KVM aufbaut. Ein Ganeti Cluster besteht aus mehreren physikalischen Rechnern, sogenannten Nodes. Pro Cluster existiert eine Node mit der Rolle *master* und beliebig viele mit den Rollen *master_candidate* und *regular*. Die Node mit der Rollen master hat die Autorität über die gespeicherten Clusterkonfiguration und kann Änderungen auf dem Cluster imitieren. Nodes der Rolle master_candidate halten ebenfalls die gesamte Konfiguration, sodass sie die master Node ersetzen können, sollte diese ausfallen. Zusätzlich gibt es noch die Rollen *drained*, für Nodes, die keine neuen VMs aufnehmen und *offline*, für Nodes, die zwar als Teil des Clusters konfiguriert, allerdings zurzeit nicht verfügbar sind.

Die Ressourcen des Ganeti Clusters werden, ähnlich wie bei VMware, durch Ganeti zusammengefasst und die VMs im Normalfall automatisch auf die Nodes verteilt. Für Arbeitsspeicher kann erneut sowohl ein Minimum festgelegt werden, als auch ein Maximum, welches von den VMs ausgenutzt werden kann, solange auf der Node genug Speicher frei ist. Die Summe des zugewiesenen maximalen Arbeitsspeichers kann also auch bei Ganeti größer sein, als tatsächlich Ressourcen zur Verfügung stehen. Für die Anzahl der virtuellen CPU-Kerne ist allerdings keine Definition eines Min-/Max-Wertepaares möglich.

Um eine neue VM mit Ganeti zu starten, kann der Befehl `gnt-instance add` verwendet werden. Der Befehl bietet verschiedene Optionen, um Einstellungen wie den Typ der Festplatte (z. B. einfach, redundant oder über einen Netzwerkspeicher), das zu installierende Betriebssystem und Einschränkungen des Ressourcenverbrauchs zu definieren. Die im Cluster verfügbaren Betriebssysteme können mit `gnt-os list` aufgelistet werden. Ein Beispielbefehl für eine Ubuntu-VM mit der IP 192.168.1.10, 2 vCPUs, 1 GB Arbeitsspeicher und einer redundanten 15 GB großen Festplatte lautet:

```
gnt-instance add -t drbd -disk 0:size=15g -B maxmem=1024 vcpus=2
 -net nic:mode=bridged,link=eth0,ip=192.168.1.10 -o ubuntu-20.04
-n node.example instance.example.
```

2.4 Betriebssystemvirtualisierung

Eine weitere, für diese Arbeit relevante, Virtualisierungstechnologie ist die Betriebssystemvirtualisierung, welche in [16] näher erläutert wird. Wie in Abbildung 4 dargestellt ist, werden bei der Betriebssystemvirtualisierung verschiedene Instanzen des gleichen Betriebssystems voneinander isoliert ausgeführt, welche sich einen Betriebssystem-Kernel teilen, anstatt ein vollständiges Betriebssystem auf virtueller Hardware auszuführen. Dadurch entfällt auch die Notwendigkeit für einen Hypervisor. Eine einzelne solche Instanz wird als *Container* bezeichnet, weshalb man statt Betriebssystemvirtualisierung auch von Containervirtualisierung spricht.

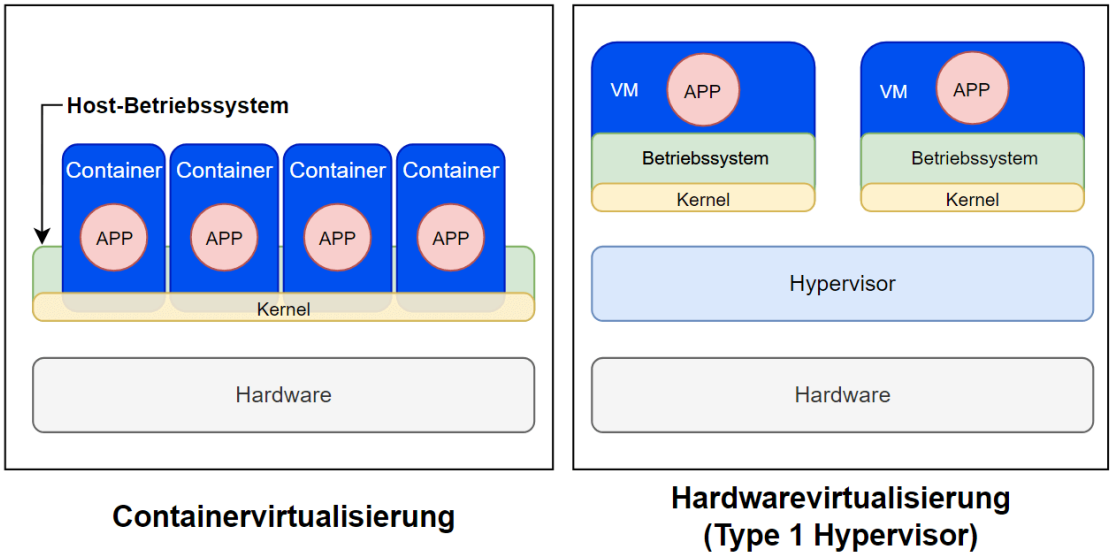


Abbildung 4: Containervirtualisierung und Hardwarevirtualisierung im Vergleich

Die Lösung Docker Desktop [17] ermöglichen es zwar, Linux-Container unter Windows zu nutzen, allerdings wird dafür im Hintergrund eine Linux VM erstellt bzw. das Windows-Subsystem für Linux genutzt, um drin dann die Linux-Container auszuführen. Linux-Container unter Windows sind somit das Resultat aus der Kombination von Hardware und Betriebssystemvirtualisierung. Docker Desktop ist allerdings explizit für die Entwicklung und nicht den Betrieb von Anwendungen gedacht und wird somit in dieser Arbeit nicht weiter beachtet.

2.4.1 Mehrschichtige Dateisysteme und Container Images

Mehrschichtige Dateisysteme bestehen, wie in Abbildung 5 grafisch verdeutlicht und in [18] beschrieben wird, aus mehreren übereinanderliegenden Dateisystemschichten. Die

Schreibzugriffe auf die unterliegenden Schichten sind jeweils per *copy-on-write* umgesetzt, d. h. die Originaldateien liegen real nur mit Lesezugriff vor und alle Schreibvorgänge werden auf eine Kopie angewendet und separat abgespeichert. So wurde in Abbildung 5 *Datei 2* und *Datei 5* bereits in der untersten Schicht (*Schicht 1*) erstmalig erstellt und in den Schichten *Schicht 3* bzw. *Schicht 2* und *Schicht 3* geändert. Dadurch ist die geänderte Dateiversion, zusammen mit den ggf. neu erstellten Dateien, wie *Datei 3* in *Schicht 2*, Bestandteil der jeweiligen Schicht. Das Ziel der mehrschichtigen Dateisysteme ist u. a., die Wiederverwendung der einzelnen Schichten durch mehrere verschiedenen Prozesse zu ermöglichen, dadurch unnötige Kopien zu vermeiden und somit speichereffizienter zu arbeiten [19].

Da mehrere virtuelle Dateisysteme übereinandergelegt werden, um aus Sicht eines Prozesses ein vereinigtes Dateisystem zu bilden (merged Layer), spricht man auch von Union File Systems (union FS) [19].

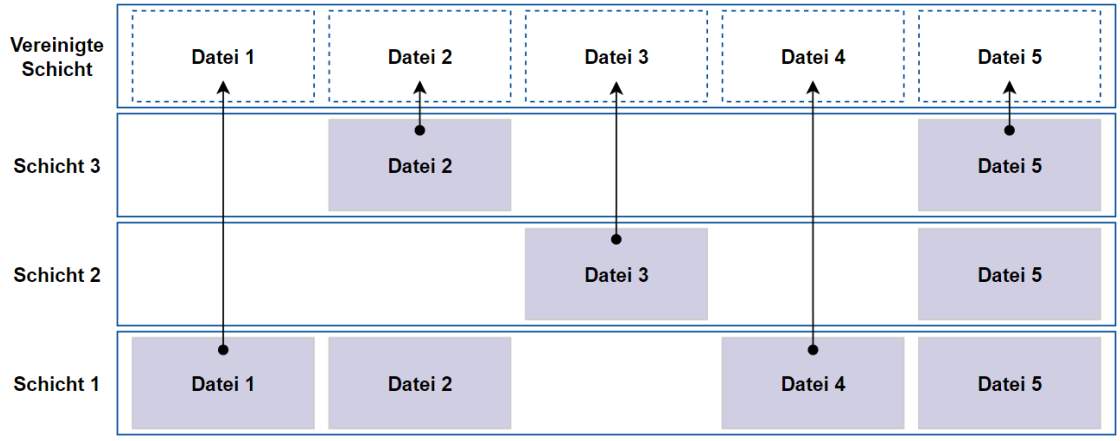


Abbildung 5: Funktionsweise von Union File Systems; in Anlehnung an [18]

Ein *Container Image* ist, laut der Dokumentation [20] der in Abschnitt 2.5 eingeführten Containertechnologie Docker, eine Datei, welche den Anwendungscode, Bibliotheken und alle anderen Software-Abhängigkeiten beinhaltet, die zum Ausführen einer Software in einem Container erforderlich sind. Ein Image wird über eine sogenannte *Dockerfile* definiert und kann auf einem anderen, sogenannten *Base Image*, aufbauen. Container Images bauen auf den mehrschichtigen Dateisystemen auf und sind die Basis einer jeden Container Instanz. Anhang A beinhaltet eine beispielhafte Dockerfile für die Generierung eines Images einer NodeJS [21] Anwendung auf Basis des NodeJS-Images [22]. Durch den Einsatz von mehrschichtigen Dateisystemen kann dasselbe Image von mehreren anderen Images oder Container Instanzen verwendet werden.

Der Zusammenhang von Container und Container Image ist in Abbildung 6 grafisch dargestellt. Zu sehen ist, dass jede Anweisung der Dockerfile eine neue Image-Schicht und damit Schicht im mehrschichtigen Dateisystem erzeugt. Ebenfalls zu erkennen ist eine nicht-persistente Container-Schicht, in diese werden während der Laufzeit eines

Containers alle Dateiänderungen nach der Copy-On-Write Methode geschrieben. Daraus folgt, dass bei einem Container-Neustart alle Dateiänderungen verloren gehen. Um dies zu verhindern, müssen persistente Speicher in den Container eingebunden werden. Bei der Vorstellung der einzelnen Containerbasierten Virtualisierungstechnologien in Abschnitt 2.5.3 und 2.6.1.2 wird darauf noch im Detail eingegangen.

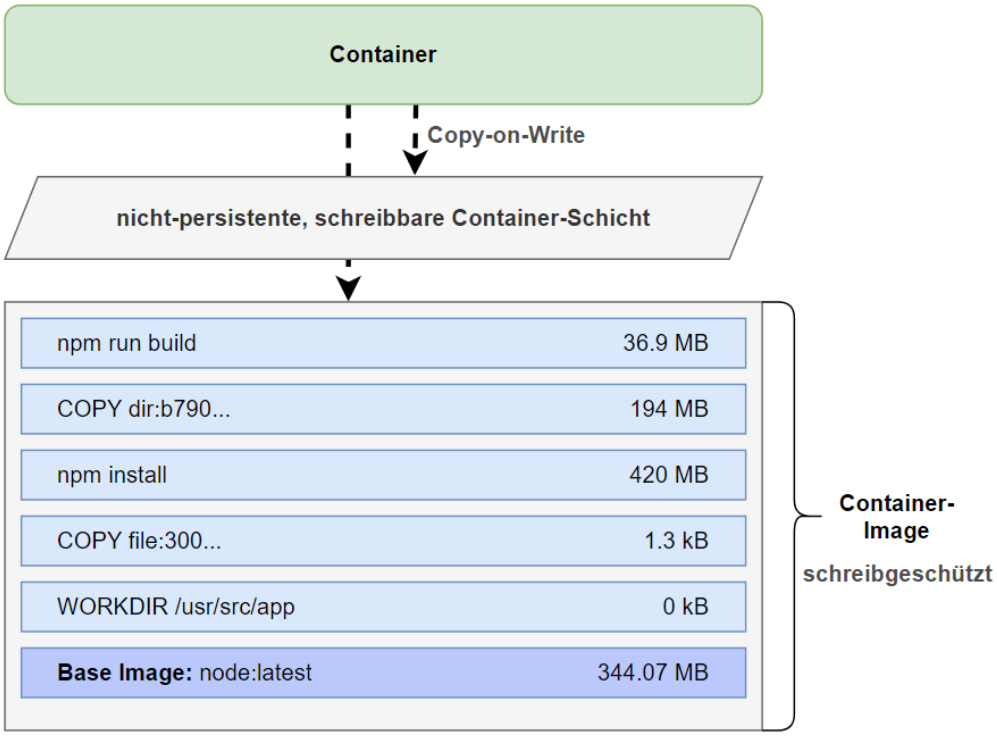


Abbildung 6: schreibgeschützte Schichten eines Container Images mit der obersten nicht-persistenten, schreibbaren Container-Schicht [20]

Erstellte Images können entweder lokal genutzt oder über eine *Container Registry* zur Verfügung gestellt werden, um sie von dort aus auf andere Server herunterzuladen. Mit hub.docker.com stellt die Docker Inc. eine eigene Registry zur Verfügung, in dem bereits zahlreiche Images z. B. für Ubuntu, NodeJS, Redis und PostgreSQL angeboten werden [23]. Verschiedene Versionen eines Images können über sogenannte Tags unterschieden werden, wie z. B. der Tag *latest* in *node:latest* in Anhang A.

2.4.2 Container Engine und Container Runtime

Zu den Aufgaben einer *Container Engine* gehören laut Scott McCarty [24] u. a. das Bereitstellen einer API, das Herunterladen und Verwalten von Container Images und das Erstellen und Übergeben der Daten und Metadaten[1], die zum Starten des Containers benötig werden, an die sogenannte *Container Runtime*. Die Container Runtime startet

[1] z. B. Container Image, Sicherheitsrichtlinien (vgl. Abschnitt 3.1.1.5), Namespace Informationen (vgl. Abschnitt 2.4.3)

und verwaltet die Container-Instanzen. Teilweise wird die Container Engine als *Container Manager* oder ebenfalls als Container Runtime bezeichnet und stattdessen in High- und Low-Level Runtime unterschieden bzw. keine terminologische Unterscheidung vorgenommen [25].

2.4.3 Linux-Container

Linux-Container bauen laut [19] auf den drei Kernel-Funktionen *Linux-Namespaces*, *Control Group* (cgroups) [26], und den mehrschichtigen Dateisystemen auf.

Linux-Namespaces sind hierarchisch organisiert und dienen der Isolierung von Prozessen, indem sie die Systemressourcen abstrahieren und es dadurch ermöglichen, die Sichtbarkeit dieser für die Prozesse innerhalb eines Namespaces aussteuern. Man unterscheidet zwischen verschiedenen Namespace-Typen.

So dient der *Process-ID* (PID) [27] Namespace dazu die Sichtbarkeit von Prozessen einzuschränken. Die PID ist laut [28] eine eindeutige Zahl, die jedem Prozess vom Betriebssystem bei Prozessstart zugewiesen wird. Sie dient dazu, den Prozess eindeutig zu identifizieren. Der Prozess mit der PID 1 wird als *Init-Prozess* bezeichnet. Dieser Prozess wird beim Systemstart gestartet und ist der "Vater" aller anderen Prozesse im System. Jeder Prozess kann seinerseits wiederum andere Prozesse starten und somit selbst zum Elternprozess werden. So entsteht ein Baum von Prozessen, in dem jeder Prozess einem anderen Prozess untergeordnet ist. Prozesse innerhalb eines PID Namespaces haben, wie in Abbildung 7 dargestellt, einen eigenen internen Prozessbaum, welches Teil des für diese Prozesse nicht einsehbaren Prozessbaumes des jeweils übergeordneten Namespaces ist. Der Prozess, der den neuen Namespace erzeugt hat, erhält innerhalb des Prozessbaumes die PID 1 und stellt somit die „Spitze“ des internen Baumes bzw. den Init-Prozess innerhalb des Containers dar.

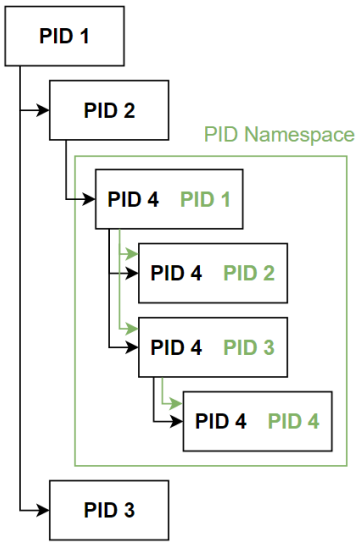


Abbildung 7: Isolierung von Prozessen durch den PID Namespace

Der *Mount* Namespace hingegen regelt, welche Einhängepunkte ein Prozess einsehen kann. Da statt für Geräte auch für Ordner Einhängepunkte erzeugt werden können, kann dadurch einem Namespace Zugriff auf einzelnen Ordner gewährt werden.

Zusätzlich existieren Namespace z. B. zum Isolieren des Netzwerkes oder der Zeit. Eine Übersicht über alle existierenden Linux-Namespaces und die von ihnen isolierten Ressourcen ist in Tabelle 1 zu sehen.

Namespace	Isoliert
Cgroup	cgroup-Wurzelverzeichnis
IPC	System V IPC, Nachrichtenwarteschlangen
Network	Netzwerkgeräte, Stacks, Ports, usw.
Mount	Einhängepunkte
PID	Prozesskennungen
Time	Startzeit und monotone Uhren
User	Benutzer- und Gruppenkennungen
UTS	Rechnername und NIS- Domain-Name

Tabelle 1: Übersicht über die Linux-Namespaces [29]

Die cgroups sind ebenfalls hierarchisch organisiert und ermöglichen die Auslastung von Ressourcen für eine definierte Gruppe von Prozessen zu beschränken und zu überwachen. Dazu zählt z. B. die CPU-, Netzwerk- und Arbeitsspeicherauslastung oder die Anzahl der Prozesse [30]. Die Eigenschaften der cgroups werden über ein virtuelles Dateisystem namens *cgroupfs* gesteuert. Für jede Gruppe wird ein neuer Ordner in diesem Dateisystem erzeugt. Anschließend können die Mitglieder, Systembegrenzungen und Statistiken über entsprechende Dateien eingesehen und modifiziert werden.

Ein Beispiel für ein mehrschichtiges Dateisystem ist OverlayFS, welches u. a. von Docker (Abschnitt 2.5) unterstützt wird [18] und Teil des Linux Kernels ist.

2.4.4 Windows-Container

Neben Linux, bietet auch Microsoft ab Windows 10 und Windows Server 2016 Container-Funktionalitäten in Form von Windows-Containern an [31].

Die grundlegende Funktionsweise von Windows-Containern ist laut Dokumentation [31] sehr ähnlich zu der von Linux-Containern. So verwenden sie auch Namespaces, mehrschichtige virtuelle Dateisysteme und ermöglichen eine Ressourcensteuerung. Allerdings bieten Windows-Container zwei unterschiedliche Isolierungsarten an, die

Prozess- und die *Hyper-V-Isolation*. Hyper-V ist eine auf einem Type 1 Hypervisor basierende Virtualisierungsplattform von Microsoft [32]. Bei der Prozessisolation teilen sich die Container, wie bei Linux, den Kernel des Host-Betriebssystems. Bei der Hyper-V-Isolierung wird hingegen für jeden Container eine stark dafür optimierte VM erstellt, innerhalb derer der Container ausgeführt wird. Dadurch soll die Sicherheit und Kompatibilität erhöht werden. So ist es durch die Verwendung der Hyper-V-Isolierung auch möglich, Container-Images zu nutzen, die auf einer älteren Windows-Version aufbauen, während bei der Prozessisolierung das Containerimage immer auf der Windows-Version des Hosts aufbauen muss. Windows 10/11 unterstützen die Prozessisolation zudem nur in eingeschränkten Maßen.

2.5 Virtualisierung mittels Docker

Docker ist laut der offiziellen Dokumentation [20] eine Open Source Plattform zum Erstellen und Verwalten von containerisierten Anwendungen mittels Linux und Windows-Containern.

Nachfolgend sollen die grundlegenden Bestandteile der Docker Plattform anhand der Dokumentation [20] erläutert werden.

2.5.1 Docker Engine

Docker setzt auf eine Client-Server-Architektur, d. h. ein *Docker Client* sendet die Anweisungen an den *Docker Daemon* Server. Docker Client und Docker Daemon können entweder auf demselben oder getrennten (virtuellen) Servern installiert werden und bilden zusammen die *Docker Engine*, die Container Engine von Docker. Standardmäßige wird das Docker Command Line Interface (CLI) [9] als Client verwendet. Es ist allerdings auch möglich, einen eigenen Client zu entwickeln oder Docker Compose (vgl. Abschnitt 2.5.4) zu verwenden.

2.5.2 Konfigurationsparameter

Um einen Container mit Docker zu starten, muss der Befehl `docker run <options> <container-image>` verwendet werden.

Über die Optionen können verschiedene Aspekte des Containers konfiguriert werden. So ermöglicht die Option `--name` beispielsweise den Namen des Containers zu definieren und mit der Option `-p` kann eine Port-Zuordnung angegeben werden (`-p Port_auf_Host:Port_innerhalb_von_Container`). Die Option `-d` wird verwendet, um einen Container im losgelösten Modus zu betreiben. In diesem Modus läuft der Container im Hintergrund und blockiert das Terminal nicht. Die Option `-ti` wird hingegen verwendet, um einen Container im interaktiven Modus, also im

Vordergrund auszuführen. Wenn diese Option verwendet wird, startet Docker den Container und verbindet dann das Terminal mit diesem, sodass Befehle innerhalb des Containers ausgeführt und deren Ausgabe in Echtzeit betrachtet werden kann. In Bezug auf die Sicherheit kann angegeben werden, als welcher Benutzer oder welche Gruppe der Prozess innerhalb des Containers laufen soll, indem die Optionen `--user` und `--group-add` verwendet werden. Um die CPU- und RAM-Nutzung eines Docker-Containers zu begrenzen, kann mit der Option `--cpus` die Anzahl der CPU-Kerne und mit der Option `--memory` die maximale Arbeitsspeichermenge festgelegt werden, die ein Docker-Container verwenden darf. Um eine Mindestarbeitsspeichermenge für einen Container zu reservieren, muss hingegen die Option `--memory-reservation` verwendet werden.

2.5.3 Persistente Datenspeicherung in Docker Containern

Wie bereits beschrieben, gehen die geschriebenen Daten eines Containers immer verloren, wenn dieser neu gestartet oder gelöscht wird. Sollte dieses Verhalten nicht gewünscht sein, weil z. B. innerhalb des Containers eine Datenbank betrieben wird, können entweder *benannte Volumes* oder *Bind Mounts* genutzt werden. Bei beiden Lösungen können Ordner oder einzelne Dateien definiert werden, die nicht, wie zuvor beschrieben, in der nicht persistenten Container-Schicht, sondern auf dem Host Dateisystem persistent gespeichert werden. Dafür werden die entsprechenden Ordner/Dateien im Host-Dateisystem als Ordner/Datei innerhalb des Container-Dateisystems einhängen. Bei der Verwendung von benannten Volumes werden die Daten in einem von Docker verwalteten Teil des Host-Dateisystems gespeichert. Beim Bind Mount kann hingegen ein beliebiger Pfad innerhalb des Host-Dateisystems als Speicherort angegeben werden. Sollte die entsprechende Datei bereits existieren oder sich in dem entsprechenden Ordner bereits Dateien befinden, sind diese durch das Einhängen innerhalb des Containers verfügbar.

Um einen persistenten Speicher in den Docker Container einzuhängen, kann beim Erstellen des Containers (`docker run …`) die Option `-v /host/dir1:/container/dir` für einen Bind Mound bzw. `-v volume-name:/container/dir` für ein benanntes Volume verwendet werden.

2.5.4 Docker Compose

Docker Compose ermöglicht es die Konfigurationsparameter, nicht wie bisher beschrieben, über die Kommandozeile zu übergeben, sondern in einer YAML Ain't Markup Language (YAML) [33] Datei zu definieren. Besonders vorteilhaft ist dies für IT-Services, die aus mehreren Containern bestehen, da es möglich ist, in einer Datei mehrere Container zu konfigurieren. Um Docker Compose zu nutzen, müssen die

benötigten Container, die Abhängigkeiten dieser zueinander, und alle weiteren Konfigurationsparameter, wie virtuelle Netzwerke, geöffnete Ports und Volumes (vgl. Abschnitt 2.5.3), in der Docker Compose Konfigurationsdatei hinterlegt werden. Anschließend kann die Anwendung über den Befehl `docker-compose up` zusammenhängend gestartet werden. Im Anhang D wurde beispielhaft ein Webservice konfiguriert, welcher auf den Datenbanken Redis [34] und PostgreSQL [35] aufbaut. Das Container Image für die Webseite wird bei jedem Start der Anwendung neu generiert, während die Images der Datenbanken bereits vorliegen. Zusätzlich wurden für die notwendigen Ports über das Schlüsselwort `ports` die Port-Zuordnung konfiguriert und für die persistente Datenspeicherung entsprechende Volumes angelegt und eingehängt.

2.6 Virtualisierung mittels Kubernetes

Kubernetes ist laut der Dokumentation [36] eine Plattform zum automatisierten Bereitstellen, Skalieren und Verwalten von Containern, welche ursprünglich von Google entwickelt und als Open-Source-Projekt veröffentlicht wurde. Mit der Veröffentlichung der Version 1.0 im Jahr 2015 wurde das Projekt jedoch an die zur *Linux Foundation* [37] gehörende *Cloud Native Computing Foundation* [38] übergeben [39]. Kubernetes unterstützt Linux- und prozessisolierte Windows-Container, Hyper-V-isolierte Windows-Container werden hingegen nicht unterstützt und können somit nicht über Kubernetes bereitgestellt werden.

Nachfolgend werden die Kubernetes Grundlagen basierend auf der Dokumentation [36] beschrieben.

2.6.1 Kubernetes Ressourcen

Eine Kubernetes Ressource ist ein durch das Kubernetes API verwaltetes Objekt, durch das z. B. Volumes oder Container-Eigenschaften definiert werden. Neben den von Kubernetes bereitgestellten Ressourcen ist es auch möglich, eigene Ressourcentypen, z. B. zur Konfiguration von Drittanbietererweiterungen, zu definieren. Typischerweise werden die Ressourcen in Form einer YAML-Datei definiert. Es ist allerdings auch möglich, die JavaScript Object Notation (JSON) [40] zu nutzen. Um als Entwickler mit der Kubernetes API zu interagieren und z. B. die Konfigurationen auf den Server zu übertragen, wird das CLI-Werkzeug *kubectl* genutzt. kubecrl ist somit Kubernetes Pendant zu Dockers *Docker Client*.

Um die Konfiguration zum Bereitstellen einer containerisierten Anwendung mittels Kubernetes zu verstehen, ist es wichtig, die Bedeutung und Funktionsweise einiger wesentlicher Ressourcen zu kennen, welche im Folgenden erläutert werden.

2.6.1.1 Workload Ressourcen

Workload Ressourcen definieren z. B. die zugeordneten Ressourcen eines oder einer Gruppe von zusammengehörigen Containern oder die Anzahl zu starteten Instanzen. Wesentliche Workload Ressourcen sind *Pods, ReplicaSet* und *Deployments*.

Pods stellen die grundlegendste Workload-Einheit innerhalb eines Kubernetes-Clusters dar und bestehen aus einem oder mehreren Containern, für die gemeinsame Speicher- und Netzwerkressourcen, Startbedingungen und weitere Metadaten definiert werden können.

Ein ReplicaSet erweitert die Pod-Konfiguration und ermöglicht es festzulegen, wie viele Pod-Instanzen bereitgestellt werden sollen. Neben dem initialen Starten mehrerer Pod-Instanzen, stellt Kubernetes auch sicher, dass die im ReplicaSet definierte Anzahl an Pods-Instanzen über den gesamten Bereitstellungszeitraum erreicht wird, indem z. B. neue Instanzen gestartet werden, sollte ein Pod ungeplant beendet werden. Die Lastenverteilung zwischen den Pod-Instanzen übernimmt der Ingress-Controller (siehe Abschnitt 2.6.1.3 und Abschnitt 2.6.2).

Deployments erstellen im Hintergrund automatisch ein ReplicaSet, bieten allerdings zusätzliche Funktionalitäten, wie *deklarative Updates*. Deklarative Updates ermöglichen es, über die Deployment-Konfiguration einen Ziel-Status zu definieren. Kubernetes übernimmt dann automatisch die notwendigen Schritte zum Erreichen dieses Status. So ist es z. B. möglich, die Container-Image-Version in der Deployment-Konfiguration zu ändern. Kubernetes wird daraufhin ein sogenanntes *Rolling Update* anstoßen. Dabei wird automatisch ein neues ReplicaSet für das neue Container-Image erzeugt und anschließend so lange jeweils ein neuer Pod im neuen ReplicaSet gestartet und ein Pod aus dem veralteten ReplicaSet gestoppt, bis alle Pods migriert wurden. Durch dieses Vorgehen gibt es während des Updatevorgangs keine Ausfallzeiten.

Der Zusammenhang dieser drei Ressourcen und der Ablauf eines Rolling Updates wird in Abbildung 8 noch einmal grafisch verdeutlicht. Im Anhang E ist zudem eine beispielhafte Deployment-Konfiguration der Anwendung *exampleApp* zu finden, welche das Container-Image *example:latest* dem Namespace *example-namespace* zuordnet, ein Secret und eine *ConfigMap* nutzt und ein *PersistentVolumeClaim* referenziert. Die Bedeutung dieser Begriffe wird fort folgend noch genauer erläutert.

Eine Alternative zum Deployment und dem ReplicaSet ist das *StatefullSet*. Die vom StatefullSet erstellten Pods erhalten, im Gegensatz zum ReplicaSet, einen Namen, welcher nach einem definierten Muster generiert wird und der sich bei Neustart des Pods nicht ändert. Jede Instanz des RepicaSets hat zudem seine eigenen PersistentVolumeClaims (vgl. Abschnitt 2.6.1.2), während sich bei einem Deployment alle Instanzen ein PersistentVolumeClaim teilen. Dieses Verhalten ist z. B. bei hochverfügbaren

Datenbanken notwendig, da sich mehrere Instanzen einer Datenbank nicht ein und denselben Speicherplatz teilen können.

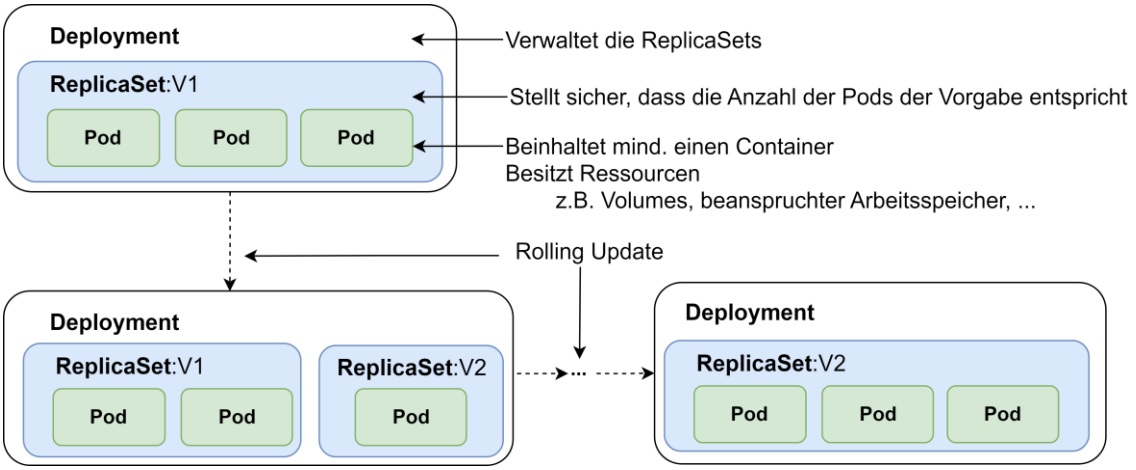


Abbildung 8: die Workload Ressourcen Pod, ReplicaSet und Deployment

2.6.1.2 Konfigurations- und Speicherressourcen

Die Konfigurations- und Speicherressourcen dienen der Verwaltung der persistenten Daten. Dazu zählen auch Konfigurationsdateien und sensible Daten wie Passwörter und Authentifizierungstoken. Wesentliche Ressourcen dieses Typs sind *PersistentVolumeClaims* (PVC) [36], *ConfigMaps* und *Secrets*.

Wie in Abbildung 9 dargestellt, beansprucht ein PersistentVolumeClaim den Speicherplatz für ein persistentes Speicher-Volume, welches von einem Pod genutzt werden kann und beschreibt mindestens Name, Größe und Zugriffsberechtigungen des entsprechenden Volumes. Der Speicherplatz muss zunächst über eine weitere Speicherressource, den *PersistentVolume* (PV) [36] bereitgestellt werden und kann optional über die Speicherressource *StorageClass* in verschiedene Klassen[2] unterteilt werden. So kann z. B. eine lokale Festplatte zunächst über ein PV verfügbar gemacht werden. Anschließend kann über ein PVC ein Teil dieser Cluster Ressource beansprucht werden. Innerhalb eines Deployments kann dieses PVC dann referenziert und eingebunden werden. In Anhang F ist eine beispielhafte Konfiguration eines PersistentVolume zu finden, welches dem Cluster 10 Gibibyte (GiB) [36] des lokalen Speicherplatzes der Node bereitstellt und das im *exampleApp*-Deployment referenzierte PersistentVolumeClaim, welcher 1 GiB an Speicherplatz anfordert.

[2] z. B. *fast* und *slow* oder *local* und *remote*

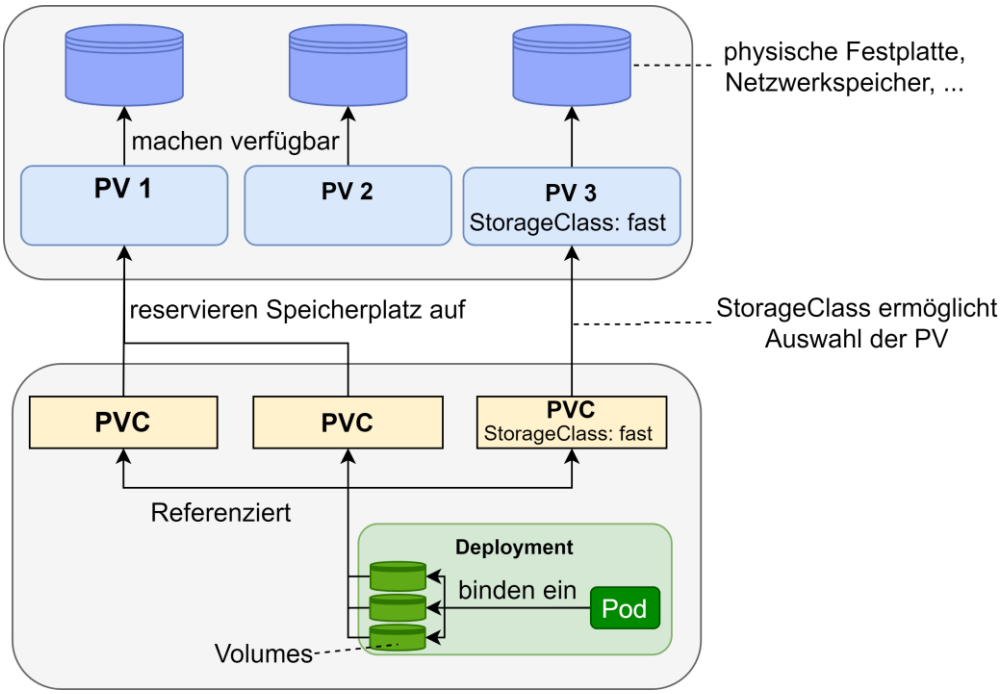


Abbildung 9: Zusammenhang von PersistentVolume und PersistentVolumeClaim zur Bereitstellung von persistentem Speicher für einen Pod.

In ConfigMaps können nicht-sensible Daten als Schlüssel-Wert-Paar gespeichert werden. Diese können anschließend innerhalb der Pod-Konfiguration als Umgebungsvariable genutzt, oder als Datei eingebunden werden. Secrets sind in Aufbau und Funktionsweise der ConfigMap sehr ähnlich, dienen aber speziell der Speicherung sensibler Daten. In Gegensatz zur ConfigMap bieten Secrets zusätzlich zur einfachen Zeichenkette noch spezialisierte Secret-Typen z. B. *Basic Authentification* oder Authentifizierung mittels Zertifikat. Die Inhalte der Secrets können entweder Base64-kodiert oder in Klartext gespeichert werden. Durch die Base64-Kodierung ist es möglich, binäre Daten, wie z. B. Zertifikate, als Secret zu speichern. In Anhang G ist eine beispielhafte Konfiguration der im *exampleApp*-Deployment verwendeten ConfigMap und des Secrets dargestellt.

2.6.1.3 Service Ressourcen

Service Ressorcen dienen der Verwaltung von Netzwerkdiensten, dazu zählt z. B. das Freigeben von Netzwerk-Ports und das clusterweite Routing und die Lastverteilung. Die relevantesten Service Ressourcen sind *Service* und *Ingress*.

Ein Service stellt einen Pod als Netzwerksdienst zur Verfügung. Die Servicedefinition besteht mindestens aus dem Servicenamen, über den der Service später erreichbar ist, einem Auswahl-Kriterium, welches bestimmt welche Pods als Service bereitgestellt werden sollen und mindestens einem Port-Mapping bestehend aus Eingangsport,

Zielport und Netzwerkprotokoll. Je nach Konfiguration werden die Services entweder nur innerhalb des Clusters verfügbar gemacht (Standard) oder auch für außerhalb.

Ein Ingress stellt Routen für das Hypertext Transfer Protocol (HTTP) [41] in Form von Domainnamen von außerhalb des Clusters zu *Services* innerhalb des Clusters zur Verfügung. Zusätzlich kann ein Ingress als Lastverteiler fungieren und Transport Layer Security (TLS) [42] -Verschlüsselung bereitstellen.

In Abbildung 10 wird der Zusammenhang von Service und Ingress noch einmal schematisch verdeutlicht und eine beispielhafte Konfiguration eines Service und eines Ingress ist in Anhang H zu finden. Der Service macht den internen Port 3000 des Pods des *exampleApp*-Deployment über den Port 80 im Cluster verfügbar und der Ingress stellt diesen Service über den Domainnamen `kubernetes.example.com` außerhalb des Clusters bereit.

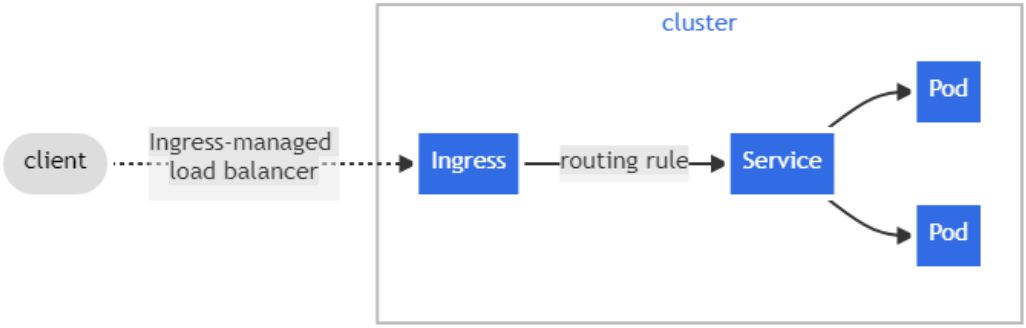


Abbildung 10: schematischer Aufbau der Routing-Architektur in Kubernetes [36]

2.6.1.4 Cluster Ressourcen

Als Cluster Ressourcen werden Ressourcen wie die Nodes und der API-Server des Clusters bezeichnet. Besonders hervorgehoben werden sollte an dieser Stelle noch die Cluster-Ressource *Namespace*. Namespaces ermöglichen es, den Sichtbarkeitsbereich von Ressourcen wie z. B. Deployments, Services und PersistentVolumeClaim zu beschränken. D. h., dass die Namen der einzelnen Ressourcen zwar innerhalb eines Namensraums eindeutig sein müssen, in verschiedenen Namensräumen ein Ressourcenname jedoch auch mehrfach vergeben werden kann. Clusterweite Ressourcen wie Nodes oder PV sind von den Namespaces nicht betroffen. Die Konfiguration des in den bisherigen Beispielen verwendeten Namespaces *example-namespace* befindet sich im Anhang I.

2.6.1.5 Policy Ressourcen

Policy Ressourcen legen verschiedene Richtlinien fest und ermöglichen dadurch z. B. den Verbrauch von Systemressourcen einzuschränken. Wesentliche Ressourcen dieses Typs sind die *LimitRange* und *ResourceQuota*, sowie die *NetworkPolicy*.

Mithilfe der LimitRange Ressource lassen sich minimal und maximal Grenzen für alle Pods eines Namespaces für verschiedene Systemressourcen wie den CPU[3], Arbeitsspeicher und Speicher setzen. Diese Grenzen können innerhalb der Pod-Konfiguration noch spezifiziert, jedoch nicht unter- (Minimum) oder überschritten (Maximum) werden. Um den Systemressourcenverbrauch des Namespaces als ganzen zu beschränken, kann hingegen das ResourceQuota genutzt werden. Beispielhaft wurde im Anhang J ein LimitRange und ResourceQuota konfiguriert.

2.6.1.6 Authentifizierungs- und Autorisierungsressourcen

Im Rahmen der Authentifizierung unterscheidet Kubernetes zwischen Nutzeraccounts für Administratoren, Entwickler usw. und Dienstkontos für Dienste, die innerhalb von Pods oder außerhalb des Clusters laufen. Während Dienstkontos nur innerhalb eines Namespaces gültig sind und als Kubernetes-Ressource (*ServiceAccount*) angelegt werden können, haben Nutzeraccounts eine clusterweite Gültigkeit und können nicht über die API angelegt werden. Stattdessen können sie z. B. über einen externen Service wie ein Active Directory oder ein Nutzerzertifikat authentifiziert werden.

Die Autorisierung der API-Zugriffe kann über verschiedene Module durchgeführt werden. Das relevanteste ist die Role-based access control (RBAC) [36], welche sich dynamisch über Ressourcen konfigurieren lässt. Es ist allerdings auch möglich, externe Web-Services über Webhooks einzusetzen, oder im Rahmen der Attribute-based access control (ABAC) [36] die Berechtigungen in einer lokalen Datei auf dem API-Server zu definieren.

Die für RBAC benötigten Kubernetes Ressourcen sind zum einen *Role* und *ClusterRole* für die Definition von Rollen, die jeweils bestimmte Berechtigungen besitzen und zum anderen *RoleBinding* und *ClusterRoleBinding* für die Zuordnung der Rollen aufgrund bestimmter Bedingungen. Dieser Zusammenhang wird in Abbildung 11 schematisch dargestellt. Role und ClusterRole unterscheiden sich lediglich dadurch, dass Role nur innerhalb eines bestimmten Namespaces gültig ist, während ClusterRole eine clusterweite Ressource ist und Clusterweit gilt. D. h., dass die über ein ClusterRoleBinding erhaltenen Berechtigungen für alle Namespaces gelten. Eine beispielhafte Konfiguration der Role *pod-reader*, welche es erlaubt, die Status der Pods zu lesen und des RoleBinding *read-pods*, welcher diese Rolle dem Nutzer *exampleUser* zuweist, befindet sich in Anhang K.

[3] Als eine CPU definiert Kubernetes einen physikalischen oder virtuellen CPU-Kern

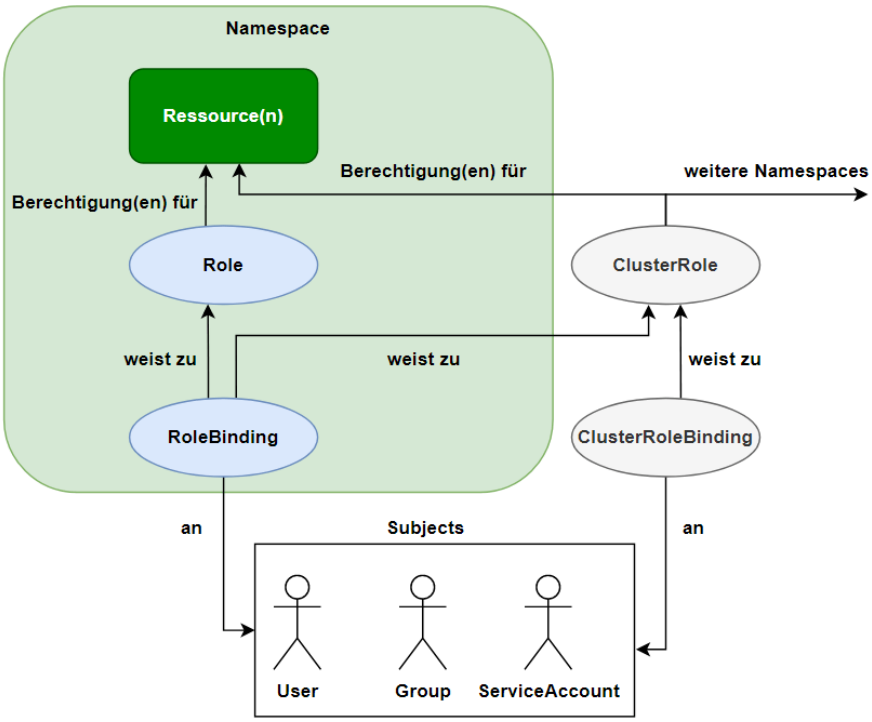


Abbildung 11: RBAC-Autorisierung in Kubernetes

2.6.1.7 Custom Resource Definition

Die Custom Resource Definitions (CRD) [36] bieten Drittanbietern die Möglichkeit, eigene Ressourcen zu definieren, um ihre Software-Konfiguration in Kubernetes zu integrieren.

2.6.2 Architektur

Eine bereitgestellte Kubernetes-Umgebung ist grundsätzlich als Cluster, also als Verbund mehrerer Rechner, aufgebaut. Dieser Cluster besteht aus mindestens einem *Control Plane* und einer *Worker Node*. Die einzelnen Komponenten des Control Planes können prinzipiell auf beliebigen Maschinen getrennt voneinander betrieben werden, typischerweise werden sie jedoch alle auf einer sogenannten *Control Plane Node* zusammengefasst. Eine Node ist dabei ein virtueller oder physikalischer Server.

Das Control Plane stellt die API bereit, verwaltet den gesamten Cluster und beinhaltet alle Daten bezüglich des momentanen Cluster Status (Nodes, Konfigurationen, laufende Container, …). Die Worker Nodes dienen hingegen der Ausführung der Container. In einer Produktivumgebung sollte es grundsätzlich mehrere Worker und Control Plane Nodes geben, um eine hohe Verfügbarkeit sicherzustellen.

Um die Funktionalität des Clusters zu erweitern, können zusätzlich Add-ons genutzt werden, welche über die Workload Ressourcen (Abschnitt 2.6.1.1) bereitgestellt werden. Die werden zum überwiegenden Teil nicht von Kubernetes selbst, sondern von

Drittanbietern entwickelt, z. B., um deren Software in Kubernetes zu integrieren. Populäre Anwendungsfälle für Add-ons umfassen Dashboards, zentrale Log-Verwaltung und Monitoring. Besonders relevant sind zudem Add-ons des Typs Ingress-Controller, welche die in den bereits erläuterten Ingress Ressourcen definierten Regeln interpretieren und das Domainnamen Routing, Load Balancing und die TLS-Ver-/Entschlüsselung übernehmen.

3 Vergleich

Um für einen bestimmten IT-Service die richtige Virtualisierungstechnologie auswählen zu können, sind verschiedene Aspekte relevant. Diese sollen in diesem Abschnitt detailliert beachtet werden.

Die zurzeit eingesetzten Virtualisierungstechnologien sind im Detail:

Virtuelle Maschinen:

Eine VM auf Basis von VMware ESXi oder KVM/Ganeti mit Linux basierenden Distributionen wie Ubuntu Server und SUSE Linux Enterprise Server oder Windows Server [Anhang X].

Docker:

Ein Mehrmandanten Docker Host, welcher auf einer durch KVM/Ganeti verwalteten VM installiert wurde und bei dem die Linux Distribution Ubuntu als Betriebssystem zum Einsatz kommt [Anhang W].

Kubernetes:

Ein Mehrmandanten Kubernetes Cluster bestehend aus einer Control Plane Node und zwei Worker Nodes [Anhang W]. Jede Node ist eine KVM/Ganeti VM, welche ebenfalls Ubuntu Server als Betriebssystem einsetzen [Anhang W].

Der Vergleich geht dabei stets von der Perspektive der für das DMC-Umfeld zuständigen Entwicklungs- und Betriebsteams aus.

3.1 IT-Sicherheit

Der erste zu beachtende Aspekt ist die IT-Sicherheit. Unter IT-Sicherheit versteht man den Schutz von IT-Systemen vor Schäden und Bedrohungen, wie z. B. dem Diebstahl oder der Beschädigung ihrer Hardware, Software oder Daten sowie vor der Unterbrechung der von ihnen bereitgestellten Diensten [43]. Derartige Bedrohungen können dabei unter anderem durch eigene Fehlkonfigurationen, Hardwaredefekte und externe Akteure, z. B. in Form von Schadsoftware, entstehen [44].

3.1.1 Schutz vor Schadsoftware und Cyberattacken

Schadsoftware und Cyberattacken können die IT-Sicherheit eines IT-Service bzw. der IT-Infrastruktur auf verschiedene Arten bedrohen. So können sie einen sogenannten *Denial-of-Service* (DOS) [45] hervorrufend, d. h. die Verfügbarkeit des Service einschränken oder diesen außer Betrieb setzen, indem sie z. B. die VM / den Container

oder den Host zum Absturz bringen oder die zur Verfügung stehenden Ressourcen auslasten. Des Weiteren können durch die Angreifer Daten gestohlen, manipuliert oder zerstört werden, um diese womöglich anschließend weiterzuverkaufen oder Lösegeld zu fordern.

Der Schutz vor Schadsoftware und Cyberattacken besteht zum einen darin, eine initiale Kompromittierung der IT-Infrastruktur bzw. der IT-Services zu verhindern und zum anderen im Falle einer Kompromittierung den betroffenen Bereich durch die Isolation der einzelnen VMs/Container zu minimieren.

Durch die in Abschnitt 2 erläuterten Unterschiede in der Art und Weise der Isolation und der allgemeinen Funktionsweise, existieren, wie fort folgend erläutert, verschiedene Angriffsvektoren, um Schadsoftware oder Cyberattacken innerhalb einer einzelnen Instanz auszuführen und ggf. die Isolation zu umgehen. Dabei wird auch auf *Common Vulnerabilities and Exposures* (CVE) [46] eingegangen, dabei handelt es sich um veröffentlichte Sicherheitslücken, welche katalogisiert wurden und eine ID erhalten haben [47]. Alle hier erwähnten CVEs haben exemplarischen Charakter und wurden in den neusten Softwareversionen bereits behoben.

3.1.1.1 Härten von Linux durch MAC und Seccomp

Linux bietet mit der *Mandatory Access Control* (MAC) [48] und *Secure Computing Mode* (Seccomp) [49] zusätzliche Möglichkeiten, die Sicherheit über die Standardfunktionalitäten hinaus zu erhöhen. Diese können sowohl bei der Hard- als auch bei der Betriebssystemvirtualisierung eingesetzt werden, worauf in den jeweiligen Kapiteln eingegangen wird. Anschließend soll bereits ein allgemeingültiger Überblick über diese Mechaniken gegeben werden.

Standardmäßig kontrolliert Linux den Zugriff auf Kernel-Objekte im Rahmen des Sicherheitsmodells *Discretionary Access Control* (DAC) [50] allein anhand der Besitzverhältnisse und der Gruppenzugehörigkeit und gibt dem Besitzer einer Ressource die Möglichkeit, die Berechtigungen zu ändern, Besitzverhältnisse zu übertragen oder ein Programm mit *root* Berechtigungen auszuführen, wenn der Nutzer dazu privilegiert ist [50]. Bei MAC werden die Berechtigungen hingegen durch den Administrator in Form von systemweiten Richtlinien festgelegt, die die Nutzer nicht mehr selbst ändern können, selbst wenn sie Eigentümer der Ressource sind [50]. MAC wird durch die *Linux Security Modules* (LSM) [51] umgesetzt, welche fester Bestandteil des Kernels sind und sogenannte *Hooks* implementieren [52]. Diese Hooks sind, wie in Abbildung 12 dargestellt, zwischengeschaltete Aufrufe, welche die Systemaufrufe direkt vor dem der Ausführung des Systemaufrufs unterbrechen, um die Berechtigungen anhand der hinterlegten Richtlinien zu überprüfen [52]. MAC bietet im Vergleich zum DAC-Modell zudem eine deutliche höhere Granularität über die Berechtigungen [53].

Wie in der Abbildung 12 ebenfalls zu sehen ist, wird DAC allerdings durch MAC nicht ersetzt, sondern lediglich ergänzt. *Security-Enhanced Linux* (SELinux) [53] und *AppArmor* [54] sind MAC-Frameworks, welche auf dem LSM aufbauen. Die Unterstützung für diese MAC-Frameworks variiert von Distribution zu Distribution. So liefert die Distribution Ubuntu AppArmor standardmäßig mit aus [55] und enthält keine offizielle Unterstützung für SELinux [56], die Distribution Red Hat Enterprise Linux setzt auf SELinux [57] und besitzt keine offizielle Unterstützung für AppArmor [58] und die Distribution SUSE Linux Enterprise Server unterstützt beide Frameworks [59].

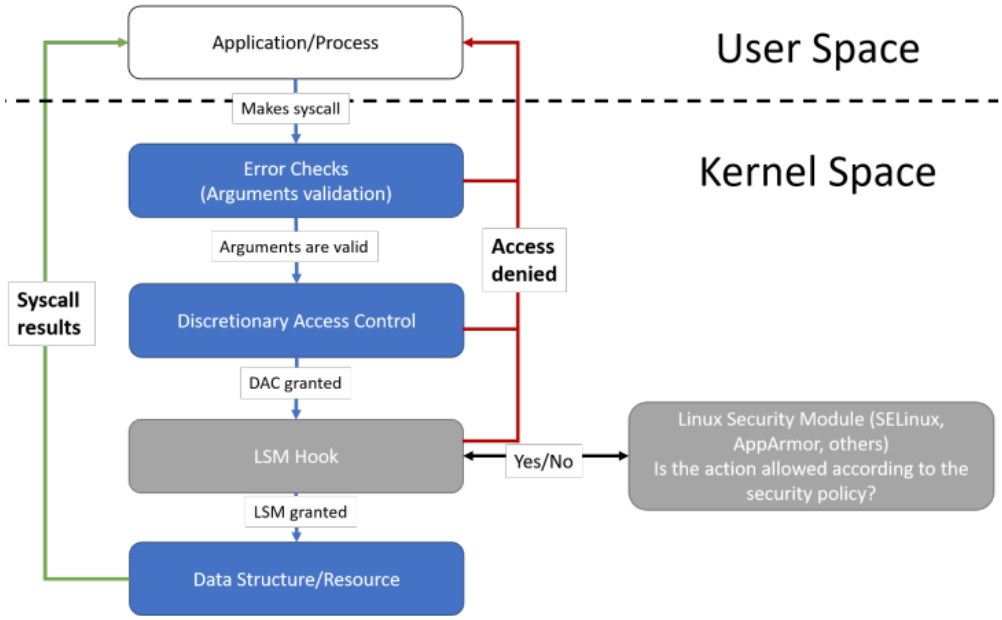


Abbildung 12: Ablauf eines Systemaufrufs bei Verwendung von LSM [60]

Seccomp ermöglicht laut Dokumentation [49] hingegen, dass Prozesse ihre eigenen Berechtigungen beschneiden, indem sie ihren Zugriff auf Systemaufrufe einschränken. Dafür muss das Programm den `prctl()` Systemaufruf ausführen und einen entsprechenden Filter (Black-/Whitlists) übergeben. Die gesetzten Filter werden bei Verwendung der Systemaufrufe `fork` und `clone`, and `execve` vererbt. Durch diese Minimierung der Systemaufrufe kann verhindert werden, dass nicht zwingend benötigte Systemaufrufe durch einen potenziellen Angreifer missbraucht werden.

Die Verwendung von MAC und Seccomp kann allerdings auch Nachteile mit sich bringen. Zum einen kann die Konfiguration eigener Regeln und Black-/Whitelists sehr komplex werden [61, 62] und zum anderen erzeugen sowohl LSM als auch Seccomp einen zusätzlichen Overhead, der bei Anwendungen mit einer hohen Anzahl an Zugriffen auf Kernelobjekte bzw. Systemaufrufen zu nennenswerten Performanceeinbußen führen kann [63, 64]

3.1.1.2 Sicherheitskritische Betrachtung der Hardwarevirtualisierung

Mögliche Bedrohungen ergeben sich im Rahmen der Hardwarevirtualisierung laut des IT-Grundschutz-Bausteins *Virtualisierung* des BSI [65] z. B. durch eine fehlerhafte Konfiguration und einen Angriff auf den Hypervisor, oder den Verwaltungsserver der jeweiligen Virtualisierungstechnologie.

Ein Beispiel für eine Konfiguration, die für eine Erhöhung des Angriffsrisikos sorgt, ist einer VM Internetzugriff zu gewähren bzw. sie aus dem Internet erreichbar zu machen.

Beispiele für mögliche Angriffe auf den ESXi-Hypervisor bieten z. B: die Schwachstellen CVE-2022-31681 [66], welche es einem Angreifer ermöglicht, innerhalb einer VM einen DOS des Host-Systems zu erzeugen und die Schwachstellen CVE-2021-22041 [67] und CVE-2022-31705 [68], welche es einen Angreifer mit Administratorrechten innerhalb einer VM ermöglichen, Code in Namen des VMX-Prozesses (siehe Abschnitt 2.2) auf dem Host auszuführen. Ähnliche Schwachstellen existieren auch bei der Virtualisierung mittels KVM mit den Schwachstellen CVE-2022-1158 [69] (KVM), zum Erreichen eines DOS des Host-Systems und CVE-2021-3713 [70] (QEMU) um Code in Namen des QEMU-Prozesses auf dem Host auszuführen.

3.1.1.3 Maßnahmen zur Mitigation der Gefahren der Hardwarevirtualisierung

Um die Gefahren, die von den genannten Bedrohungen ausgehen, zu minimieren, definiert das BSI im IT-Grundschutz-Baustein *Virtualisierung* [65] 28 Anforderungen, die erfüllt werden sollten bzw. müssen, um die Schutzanforderungen zu gewährleisten. Dazu zählt u. a. das regelmäßige Aktualisieren des Host-Betriebssystems, der Management-Software und Firmware, das Absichern administrativer Schnittstellen vor den Zugriff unbefugter und das Reduzieren der administrativen Berechtigungen der einzelnen Nutzer auf das tatsächlich notwendige Minimum, das Untersagen von Netzwerkverbindungen, solange diese nicht zwingend notwendig sind und die ständige Überwachung von Zustand und Auslastung der Hosts, VMs und Netzwerke. Bei der Nutzung von Linux-Servern empfiehlt das BSI MAC-Frameworks wie AppArmor oder SELinux zu aktivieren und existierende Standardprofile zu nutzen, für Anwendungen mit erhöhten Sicherheitsanforderungen sollten zudem eigene anwendungsspezifische Profile erstellt bzw. existierende Standardprofile überprüft und ggf. angepasst werden [71]. Im gleichen Maße kann Seccomp verwendet werden, um, wie bereits beschrieben, Systemaufrufe zu begrenzen. Sollte dies von der jeweiligen Anwendung nicht nativ unterstützt werden, bietet z. B. der System- und Sitzungs-Manager *systemd* die Möglichkeit die Anwendung mit einem gewünschten Profil zu starten [72].

Des Weiteren sollte die Anzahl der Dienste innerhalb einer VM, welche mit Administratorrechten ausgeführt werden, auf ein erforderliches Minimum reduziert werden, um Angreifern die Möglichkeit zu nehmen, Schwachstellen in diesen Diensten zu nutzen,

um selbst Administratorrechte zu erlangen und diese z. B im Rahmen der genannten Schwachstelle CVE-2021-22041 auszunutzen.

Der wesentliche Teil der Umsetzung und Überwachung dieser Sicherheitsaspekte wird von Teams außerhalb des DMC-Umfeld übernommen [Anhang W]. Zum Verantwortungsbereich der Mitarbeiter des DMC-Umfelds zählt jedoch z. B. die benötigten Netzwerkzugriffe einer VM zu hinterfragen und ggf. Einschränkungen vorzunehmen oder zu beantragen, die MAC-Frameworks in den Linux Gast-Betriebssystemen zu konfigurieren und sicherzustellen, dass die Dienste und Befehle, sofern nicht zwingend erforderlich, ohne Administratorrechte ausgeführt werden [Anhang W].

3.1.1.4 Sicherheitskritische Betrachtung der Containervirtualisierung

Da der Docker Host und die Kubernetes Nodes innerhalb von VMs laufen, gelten die genannten Bedrohungen und Maßnahmen der Hardwarevirtualisierung auch für die Hosts/Nodes dieser Virtualisierungstechnologien. Allerdings ergeben sich durch die Architektur und Funktionsweise von Containern zusätzliche Bedrohungen.

Die ersten beiden zusätzlichen Angriffsvektoren sind die verwendete Container-Runtime und der gemeinsam genutzte Betriebssystem-Kernel. Grundlage für einen erfolgreichen Angriff mittels dieser Vektoren ist eine fehlerhafte Konfiguration des Containers und/oder eine Schwachstelle, die ausgenutzt werden kann. Beispiele für entsprechende Schwachstellen, die einen *Container-Escape*, also das Umgehen der Isolation, ermöglicht sind, die Schwachstelle CVE-2022-0185 [73] des Linux Kernels, welche insbesondere Kubernetes betraf, da der Angriff hier unter Standardeinstellungen möglich war [74], und CVE-2019-5736 [75] der Container Runtime *runc*, die sowohl Kubernetes als auch Docker standardmäßig nutzen [76]. Da sowohl Docker als auch Kubernetes mit root-Berechtigungen ausgeführt werden, stellen die Container-Escapes auch eine erhöhte Gefahr für die zugrunde liegende VM-Infrastruktur dar (vgl. Abschnitt 3.1.1.2). Weitere Schwachstellen, wie CVE-2022-31030 [77] in der Container Engine *containerd*, ermöglichen zwar keinen Escape können aber einen DOS herbeiführen, in diesem Fall durch die Beanspruchung des gesamten zur Verfügung stehenden Arbeitsspeichers. Ein Beispiel für einen Konfigurationsfehler mit besonders hohem Bedrohungspotential, ist der Zugriff auf die API des Docker Daemon bzw. des Kubernetes Control Planes, da über diese Container und deren Zugriffsberechtigungen erstellt, geändert und gelöscht werden können.

Eine weitere Bedrohung können die verwendeten (Base) Images darstellen, insbesondere, wenn diese nicht selbst erstellt wurden. Zum einen laufen die Prozesse innerhalb des Containers, sofern dies in der Dockerfile des Images nicht explizit geändert wurde, als *root* Nutzer, wodurch Angriffe über die zuvor erwähnten Schwachstellen begünstigt werden. Des Weiteren können in den Images selbst Schwachstellen und Schadsoftware

enthalten sein. Diese Schwachstellen können z. B. durch eine fehlerhafte Konfiguration entstehen. So wird in CVE-2022-43679 [78] beschrieben, dass das Image der Software ownCloud eine Konfiguration enthielt, die es ermöglichte die accountspezifische URL zum Zurücksetzen des Passworts mitzuschneiden, oder, weil das entsprechende Image Abhängigkeiten beinhaltet, die Schwachstellen besitzen und die nicht durch das erneute Erstellen des Images bzw. durch das Anpassen der Dockerfile vor dem Erstellen aktualisiert wurden. Images, die Schadsoftware enthalten können z. B. ähnliche Namen wie offizielle Images tragen, um im Folge von Schreibfehlern oder Verwechselungen auf dem System durch das Opfer ausgeführt werden [79] oder, ggf. durch unautorisierten Zugriff auf das Image-Repository, in offizielle Images eingeschleust werden [80].

Bei Windows-Containern ist die Sicherheit laut der Dokumentation [81] stark vom verwendeten Typ abhängig. Während sich prozessisolierte Windows-Container einen gemeinsamen Kernel teilen und damit in ihren Angriffsvektoren sehr den Linux-Containern ähneln, setzt die Hyper-V-Isolierung auf einen eigenen Kernel je Container und Isolation auf Hardwareebene (vgl. Abschnitt 3.1.1.2). Bezüglich prozessisolierten Containern sagt Microsoft, dass sie den „Kernel nicht als ausreichende Sicherheitsgrenze für gefährdete Workloads mit mehreren Mandanten“ [81] betrachten. Laut Dokumentation werden Container-Escapes nur dann als sicherheitsrelevanter Programmfehler deklariert und in Folge eines dedizierten Sicherheitsupdates behoben, wenn sie die Hyper-V-Isolierung durchbrechen oder sie es ermöglichen, die Prozessisolierung ohne Administratorrechte zu umgehen. „Wenn ein Exploit einen Administratorprozess zum Ausbruch aus dem Container verwendet, betrachtet Microsoft dies [hingegen] als nicht sicherheitsrelevanten Programmfehler und patcht ihn gemäß dem normalen Wartungsvorgang“ [81].

3.1.1.5 Maßnahmen zur Mitigation der Gefahren der Containervirtualisierung

Um die zusätzlichen Risiken der Containervirtualisierung zu minimieren, sollten verschiedenen Maßnahmen angewandt und Techniken eingesetzt werden. So definiert das BSI im IT-Grundschutz-Baustein *Containerisierung* [48] insgesamt 26 Anforderungen, die eine Container-Infrastruktur im Rahmen eines Sicherheitskonzepts erfüllen muss bzw. sollte. Dazu zählt u. a., dass nur Images aus vertrauenswürdigen Quellen genutzt werden sollten, die zuvor einen Test- und Freigabeprozess durchlaufen sind, zudem sollten die Ressourcen, die ein Container konsumieren darf, begrenzt sein, erweiterte Sicherheitsrichtlinien, z. B. durch MAC, eingerichtet und das Verhalten, die Performance und der Status der Container überwacht werden.

Im Rahmen des Test- und Freigabeprozesses können Image-Scanner wie der in der GISA eingesetzte *Nexus IQ Server* [82] genutzt werden, um Images automatisiert auf Schwachstellen und Schadsoftware zu überprüfen [Anhang Y]. Docker und Kubernetes bieten zudem beide die Möglichkeit, die zu verwendeten Images über eine, beim

Erstellen des Images generierte, Prüfsumme zu referenzieren [83, 84]. Sollte ein Angreifer, die im Repository gespeicherten Images manipulieren, stimmen diese Prüfsummen nicht mehr überein, sodass Docker/Kubernetes diese Images nicht verwendet. Zusätzlich sollten Images so erstellt werden, dass sie als non-root User ausgeführt werden. Sollte dies nicht beim Erstellen des Images beachtet worden sein, ist es möglich, den Nutzer innerhalb der Docker/Kubernetes Konfiguration nachträglich festzulegen [85, 86]. Sollte die Verwendung eines root-users innerhalb des Containers zwingend erforderlich sein, gibt es die Möglichkeit, den User-Namespace (vgl. Tabelle 1) zu verwenden um den Container-internen root-user einen non-root User auf dem Host-Betriebssystem zuzuweisen. Allerdings ergeben sich dadurch einige schwerwiegende Einschränkungen, z. B. bei der Verwendung von Volumes [87, 88].

Zur Umsetzung der vom BSI empfohlenen erweiterten Sicherheitsrichtlinien unterstützt Docker laut Dokumentation [89] u. a. AppArmor, SELinux und Seccomp. Docker liefert bereits ein Seccomp und eine AppArmor Profil mit aus, welche auf alle Container standardmäßig angewendet werden, und stellt den Quellcode einer AppArmor Richtline für den Docker-Daemon zum Download bereit. Kubernetes unterstützt laut Dokumentation [85] zurzeit SELinux und Seccomp, die Implementation von AppArmor befindet sich in der Betaphase. Allerdings wendet Kubernetes standardmäßig kein Seccomp-, AppArmor oder SELinux-Profil an[4], sodass eine entsprechende Konfiguration durch den Administrator besonders zu empfehlen ist. Da Kubernetes selbst keine Profile zur Verfügung stellt, müssen entweder Profile aus anderen vertrauenswürdigen Quellen genutzt werden, wie das von Docker genutzte Seccomp-Profil, oder eigene erstellt werden. Nachdem die Profile auf die Nodes verteilt wurden, können diese für Workload Ressourcen genutzt werden. Die Möglichkeit, Profile als clusterweiten Standard zu definieren, befindet sich zurzeit ebenfalls noch in der Betaphase. Mit den *Pod Security Standards* bietet Kubernetes zudem die Möglichkeit, verschiedene Sicherheitsstandards, wie die Verwendung eines Seccomp Profil oder das Verwenden von non-root Nutzern innerhalb der Container, zu erzwingen, allerdings kann nur zwischen dem *Privileged* Profil, ohne jegliche Einschränkungen, dem *Baseline* Profil, welches den Kubernetes Standardeinstellungen entspricht, und *Restricted* Profil mit den erwähnten Einschränkungen, gewählt werden [90].

Wie bereits bei VMs birgt eine Erreichbarkeit des Services von innerhalb und insbesondere von außerhalb des Clusters/Hosts, sowie der Zugriff aus den Containern heraus auf andere Services und das Internet, ein Gefahrenpotential. Vorteilhaft für die Sicherheit ist, dass Container über den Network Namespace (vgl. Tabelle 1) grundsätz-

[4] Kubernetes war aus diesem Grund, im Gegensatz zu Docker, für die erwähnte Schwachstelle CVE-2022-0185 in der Standardkonfiguration anfällig.

lich alle Netzwerk-Ports blockieren und diese über die Konfiguration erst explizit geöffnet werden müssen. Allerdings sollte die Erreichbarkeit der Services durch eine weitere Netzwerksegmentierung auf ein notwendiges Minimum reduziert werden. Docker bietet dafür laut Dokumentation [91] die Möglichkeit, einem Container ein oder mehrere virtuelle Netzwerke zuzuordnen und diese Netzwerke als intern zu deklarieren. Container, die sich nur in internen Netzwerken befinden, können nur Services erreichen, mit denen sich ein Netzwerk teilen und können selbst auch nur von diesen erreicht werden. Bei Kubernetes ist ein Pod hingegen standardmäßig für alle Services, die innerhalb des Clusters laufen, erreichbar. Damit ein Service von außerhalb des Clusters erreicht werden kann, gibt es verschiedene Möglichkeiten. Die relevantesten sind zum einen die Definition eines externen erreichbaren Domainnamen über einen Ingress und zum anderen einen Service über den Port der jeweiligen Node erreichbar zu machen (vgl. Abschnitt 2.6.1.3). Für eine weitere Segmentierung des Netzwerkes müssen zusätzliche Plugins, wie *Calico* von Tigera [92], verwendet werden, welche es ermöglichen zusätzliche Netzwerkrichtlinien zu definieren, die z. B. den Zugriff für einzelnen Pods oder Namespaces erlauben.

Schlussendlich sollte bei der Vergabe von Berechtigungen auf die Docker Daemon und Kubernetes API nach dem Minimalprinzip gearbeitet werden. D. h. es sollten nur diejenigen Zugriff auf die API des Docker Daemon haben, die tatsächlich für den Betrieb der Services zuständig sind und innerhalb von Kubernetes sollten für verschiedenen Stakeholder verschiedene Rollen (vgl. Abschnitt 2.6.1.6) erstellt und angewendet werden, welche die Berechtigungen auf das tatsächlich erforderliche Minimum reduzieren.

3.1.2 Ausfallsicherung

Ausfallsicherung hat das Ziel, dass ein IT-Service trotz Ausfall einer virtuellen Instanz oder eines physikalischen Servers weiterhin betriebsbereit ist [9].

Zu wichtigen Maßnahmen der Ausfallsicherung zählt im Projektumfeld im Wesentlichen das zeitnahe Starten von neuen Instanzen, im Falle eines Ausfalls, ggf. auf einem neuen Rechner, bzw. das Bereitstellen von redundanten Service-Instanzen, sodass der Service trotz Ausfall unterbrechungsfrei bereitgestellt werden kann. Bei der Bereitstellung mehrere Instanzen unterscheidet man laut Luber [93] zwischen dem *Active/Passiv* und *Active/Active* Betrieb. Bei einem Active/Passiv Betrieb werden die redundanten Instanzen als Failover betrieben, d. h., dass sie im Normalbetrieb nur im Standby laufen und keine produktive Arbeit übernehmen. Bei Active/Active nehmen hingegen bereits im Normalbetrieb alle Instanzen Anfragen entgegen, um diese zu verarbeiten, sodass sich die Last auf mehrere Instanzen verteilt. Der Active/Active Betrieb bietet somit im Normalbetrieb eine bessere Leistungsfähigkeit als Active/Passiv und kommt in Falle

eines Ausfalls ohne jegliche Unterbrechung aus, allerdings reduziert sich die Gesamtleistung um die Leistung der ausgefallenen Instanz(en) und die entsprechenden Ressourcen/Anwendungen müssen den Active/Active Betrieb unterstützen.

3.1.2.1 Virtuelle Maschinen

Beim Einsatz von ESXi VMs ermöglicht laut Dokumentation [94] die bereits in Abschnitt 2.2 erwähnte Funktion *vSphere High Availability* die ESXi Host zu überwachen und im Falle eines Host-Ausfalls die betroffenen VMs auf einen anderen Host neu zu starten. Zusätzlich gibt es die Möglichkeit, das Softwarepaket VMware Tools auf dem Gast-Betriebssystem zu installieren und damit die VM bzw. einzelne Applikationen innerhalb der VM zu überwachen, um die VM im Falle eines schwerwiegenden Fehlers neu zu starten. Zur Überwachung von Performancemetriken wie Netzwerkdurchsatz, CPU- und Arbeitsspeicherauslastung kann das *Network Operation Center* eingesetzt werden [95]. Um eine höhere Verfügbarkeit zu gewährleisten, bietet VMware mit *vSphere Fault Tolerance* zudem die Option, den vollständigen Zustand einer VM, inkl. deren Arbeitsspeicher, kontinuierlich auf einen anderen ESXi Host zu kopieren, um in Falle eines Ausfalls die entsprechende VM mit nahezu keiner Unterbrechung zu ersetzen. Es handelt sich somit um einen Active/Passiv Betrieb.

KVM/Ganeti bietet laut dessen Dokumentation [96] ebenfalls die Funktion, Festplatten einer VM zwischen zwei Nodes zu replizieren und im Falle des Ausfalls einer Node die betroffenen VMs auf der anderen Node neu zu starten, allerdings keinen vollwertigen automatischen Active/Passiv Betrieb. Es ist lediglich möglich die VMs bei geplanten Ausfallzeiten einer Node ohne Betriebsunterbrechung zwischen zwei aktiven Nodes zu migrieren. Alternativ zu VMWare Network Operation Center kann z. B. das Open-Source-Netzwerk-Monitoringsystem Zabbix [97] für die Überwachung eingesetzt werden.

Für die Bereitstellung von redundanten Instanzen im Active/Active Betrieb müssen sowohl bei ESXi als auch Ganeti, durch die Entwickler oder Betreuer mehrere VMs getrennt voneinander konfiguriert und verwaltet werden. Die Lastverteilung wird durch einen externen Service, z. B. einen selbst konfigurierten nginx-Webserver [98], bereitgestellt.

3.1.2.2 Docker

Docker bietet laut Dokumentation in der jetzigen Verwendungsform nur eine sehr begrenzte Ausfallsicherheit. Es existiert zwar die Möglichkeit, die Docker-Container auf demselben Host automatisch neu zu starten. Sollte innerhalb des Containers ein schwerwiegender Fehler auftreten, ist das Neustarten der Container auf einem anderen Host, z. B. im Falle eines Host-Ausfalls, jedoch nicht möglich.

Das Betreiben von mehreren Instanzen wird zwar grundsätzlich in Form eines Active/Active Betriebs unterstützt, allerdings bietet Docker selbst keine Funktionalitäten für die Lastverteilung an, sodass entweder ein zusätzlicher Lastverteiler als Service auf dem Docker-Host betrieben und konfiguriert oder auf einen Lastverteiler, der außerhalb des Docker-Hosts betrieben wird, zurückgegriffen werden muss. Sollte der Docker-Host an sich ausfallen, muss auf die zugrunde liegende VMware Infrastruktur zurückgegriffen werden, um den Host neu zu starten. Alternativ dazu bietet Docker den sogenannten *Swarm Mode*, welcher, wie Kubernetes, Cluster-Funktionalitäten und Lastverteilung bereitstellt [99], im Gegensatz zu Kubernetes allerdings, auf Docker Compose Dateien für die Konfiguration setzt [100]. Da in der GISA GmbH jedoch zurzeit kein Docker Swarm eingesetzt wird und dies auch zukünftig nicht geplant ist, wird Docker Swarm, in Absprache mit der GISA GmbH, in dieser Arbeit nicht weiter beachtet.

3.1.2.3 Kubernetes

Der Kubernetes-Cluster bietet diese Funktionalitäten bereits nativ an. Mit Kubernetes ist es also möglich, Container nicht nur bei schwerwiegendem Fehler innerhalb des Containers automatisch neu zu starten, sondern auch auf den Ausfall einzelner Nodes automatisiert zu reagieren, indem die einzelnen Pods auf einer anderen Node neu gestartet werden. Die Möglichkeit, redundante Instanzen im Active/Active Betrieb auf verschiedenen Nodes zu betreiben, ermöglicht dabei nicht nur einen unterbrechungsfreien Betrieb bei Ausfall einzelner Instanzen, sondern auch bei geplanten Ausfallzeiten wie z. B. dem Durchführen von Updates (siehe Rolling Updates in Abschnitt 2.6.1.1) und den Ausfall einer ganzen Worker Node. Sollte der Cluster mindestens drei Control Plane Nodes, oder zwei Control Plane Nodes und eine externe Instanze der Control Plane Komponente *etcd*-Datenbank besitzen, ist es zudem möglich, dass eine Control Plane Node ausfällt, ohne dass der Betrieb beeinträchtigt wird.

3.1.3 Disaster Recovery

Disaster Recovery beschreibt Maßnahmen, die unternommen werden müssen, um ein beschädigtes System wieder funktionsbereit zu machen, dazu zählt hauptsächlich die Wiederherstellung von Daten [9]. Die wichtigste Maßnahme im Rahmen der Disaster Recovery ist das Erstellen und wieder einspielen von Backups. Dabei kann zwischen verschiedenen Vorgehensweisen unterschieden werden. So kann ein Backup z. B. eine gesamte virtuelle Festplatte auf Blockebene, einzelne Dateien auf dem Dateisystem oder einen Datenexport aus einzelnen Applikationen, wie einer Datenbank, enthalten. Einige Applikationen, insbesondere Datenbanken, haben speziellen Anforderungen, welche beachtet werden müssen, um eine konsistente Sicherung zu erstellen.

3.1.3.1 Virtuelle Maschinen

In der GISA wird für VMs die Backuplösung der Firma Commvault eingesetzt [Anhang W, Anhang X], welche u. a. das Erstellen von Backup der virtuellen Festplatten einer VM ermöglicht [101] und Datenbank-spezifische Lösungen anbietet [102]. Commvault wird allerdings von einem Team außerhalb des DMC-Umfelds betrieben und verwaltet [Anhang X]. Insbesondere bei kleineren Datenbanken können zudem die vom entsprechenden Datenbank-Managementsystem (DBMS) [103] bereitgestellten Backuplösungen wie *mysqldump* [104] genutzt werden, um einmalig oder durch die im DMC-Umfeld eingesetzte Business Process Management (BPM) [5] Software inubit BPM [5] automatisch periodische Datenbankbackups zu erzeugen [Anhang W]. mysqldump ist ein Backup Tool für die relationalen Datenbanken MySQL [104] und MariaDB [105].

Zusätzlich dazu bieten ESXi [10, 106] und Ganeti [107] eigene Lösungen an, um VMs und deren Konfiguration zu exportieren und diese anschließend zu sichern.

3.1.3.2 Docker

Docker bietet keine dedizierte Backupfunktionalität an, allerdings kann ein Befehl verwendet werden, um ein Backup aller Dateien eines Volumes eines Containers mithilfe des Programms *tar* [108] zu erstellen bzw. wiederherzustellen [109]. Für die Befehle existiert ein Beispiel im Anhang B.

Der erste Befehl erzeugt einen neuen Container, welcher dieselben Volumes an derselben Stelle einhängt, wie der Container für den ein Backup erstellt werden soll und zusätzlich dazu einen weiteren Einhängepunkt als Backup-Ziel. Der gestartete Container erzeugt mithilfe von tar ein Archiv aus allen zu sichernden Dateien, welches im Ordner des zweiten Einhängepunktes gespeichert wird. Abschließend wird der Container beendet. Hat ein Container mehrere Volumes, muss der Befehl entsprechend mehrfach ausgeführt werden. Zum Wiederherstellen des Backups muss der zweite Befehl verwendet werden, welcher den Vorgang umkehrt. Nach der Erstellung muss das Backup, sofern das Backup-Ziel ein lokaler Ordner ist, noch an einen Speicherort außerhalb des Host transferiert werden, damit es im Falle eines Host-Ausfalls für die Wiederherstellung genutzt werden kann und nicht ebenfalls verloren geht. Das regelmäßige Ausführen der Befehle kann z. B. über einen *Cronjob* [110] sichergestellt werden.

Für die Erstellung von Datenbankbackups kann, sofern der Container der Datenbank gestoppt wurde, ebenfalls die bereits beschriebene Methode genutzt werden. Alternativ können die DBMS-Tools wie mysqldump genutzt werden [111], die entweder erneut von inubit BPM oder durch einen eigens dafür erstellten Container aufrufen werden, um das Backup bei laufendem Datenbankserver zu erstellen [Anhang W].

Zusätzlich zu den Sicherungsmaßnahmen auf Container Volume bzw. Datenbank Ebene, wird die gesamte Festplatte der Dockerhost-VM einmal täglich gesichert [Anhang W]

Die Erstellung der benötigten Befehle, Cronjobs und Workflows zum Erstellen und ggf. Transferieren der Backups und das Überprüfen auf eventuelle Fehler liegt im Verantwortungsbereich der Entwickler und Service-Betreuern des DMC-Umfelds [Anhang W].

3.1.3.3 Kubernetes

Kubernetes hat ebenfalls keine dedizierte Backupfunktionalität, allerdings bietet es die Möglichkeit, Dateisystem-Snapshots eines PersistentVolumes zu erstellen. Die Konfiguration erfolgt über die Storage Ressource *VolumeSnapshot*. Allerdings gilt auch hier, dass der Snapshot auf einem anderen physischen Speicher transferiert werden muss, um für die Wiederherstellung im Falle eines Ausfalls genutzt werden zu können. Des Weiteren setzt die VolumeSnapshot Funktion voraus, dass der verwendete *Storage Driver* das *Container Storage Interface* (CSI) [112] unterstützt. Das CSI ist eine standardisierte Schnittstelle, um beliebige Speichersysteme für containerisierte Workloads zugänglich zu machen [112]. Beim betrachteten Kubernetes Cluster ist dies zurzeit nicht der Fall [Anhang W].

Als Alternative zu den nativen Kubernetes Funktionen kann Drittanbietersoftware, wie Velero eingesetzt werden. Velero ist ein von VMware unterstütztes Open-Source-Projekt für die Migration und das Erstellen von Backups von Kubernetes Ressourcen und Volumes [113]. Laut Dokumentation [114] wird Velero über Kubernetes Custom Resource Definitions konfiguriert und bietet die Möglichkeit einmalige oder periodische Backups zu erstellen. Dabei ist es möglich entweder alle Objekte des Clusters zu sichern oder nach Typ, Namensraum oder Bezeichnung zu filtern. Alle erstellten Backups werden in einen Cloud-Objektspeicher, wie z. B. MinIO [115], geladen. Um Backups mit Velero zu erstellen, muss zunächst die Server Komponente auf dem Cluster und ein Client auf dem lokalen Rechner installiert und konfiguriert werden. Anschließend kann z. B. ein Backup aller Objekte in einem Namespace erstellt bzw. die regelmäßige automatische Erstellung eines Backups geplant werden[5]. Im Katastrophenfall kann das Backup per Befehl wiederhergestellt werden. Beispielhafte Befehle befinden sich in Anhang C. Alternativ zu den Befehlen kann der Backup- und Wiederherstellungsprozess auch über YAML-Dateien

[5] Velero nutzt für regelmäßiger Backups in der --*schedule* Option die Cron-Syntax, vgl. [116].

konfiguriert werden. Velero besitzt keine offizielle Unterstützung für Windows-Container[6].

Datenbankbackups werden von Verlero nur auf Dateiebene unterstützt, sodass die Datenbank beim Erstellen des Backups gestoppt sein sollte, um inkonsistente Datenbestände zu verhindern. Alternativ kann abermals auf Tools wie mysqldump zurückgegriffen werden. Dafür wird ein Container erzeugt, welche durch Kubernetes im Rahmen eines Cronjobs periodisch gestartet wird, um den Datenexport durchzuführen und das Backup anschließend außerhalb des Clusters zu speichern. Einige andere Backuplösungen wie Commvault unterstützen das Sichern von populären Datenbanken wie MySQL bereits nativ, sodass die Verwendung von zusätzlichen Tools wie mysqldump nicht nötig ist [117].

Zusätzlich zur Sicherung der einzelnen Volumes und Datenbanken werden die zentrale etcd-Datenbank und die Festplatten aller Nodes, wie bei Docker, einmal täglich als Ganzes gesichert [Anhang W].

Die Konfiguration der Backuplösungen wie Velero und die Datenbankbackups über den Kubernetes Cronjob liegen im Verantwortungsbereich der Entwickler und Service-Betreuen des DMC-Umfelds [Anhang W].

3.1.4 Fazit des Aspekts IT-Sicherheit

Zusammenfassend lässt sich sagen, dass die Hardwarevirtualisierung eine höhere Sicherheit vor Schadsoftware und Cyber-Attacken bieten als die Containervirtualisierung und daher für IT-Services mit besonders hohem Sicherheitsbedarf zu bevorzugen ist. Allerdings bieten auch Container, sofern die genannten Mitigation-Maßnahmen beachtet werden, eine ausreichende Sicherheit für die meisten Anwendungen. Für Windows-Anwendungen sollte bei der Nutzung von Containern, auf die Hyper-V-Isolierung gesetzt werden.

Das Aufsetzen von hochverfügbaren IT-Services ist mit Kubernetes und VMs möglich, während dies für Docker nur eingeschränkt gilt, da es keinen Schutz vor dem Ausfall des einzelnen Docker-Hosts bietet. Der Aufwand für das Konfigurieren der Hochverfügbarkeit wird in Abschnitt 3.3.5 verglichen.

Das Erstellen von Backups ist grundsätzlich mit jeder Virtualisierungstechnologie möglich, der Unterschied liegt lediglich in den verwendeten Werkzeugen.

[6] Stand Velero v1.10

3.2 Ressourcenverbrauch und Performance

Da die unterschiedlichen Performance- und Verbrauchscharakteristiken im Wesentlichen durch die Unterschiede in der Hardware- und Betriebssystemsvirtualisierung begründet sind, wurden exemplarisch die Technologien Docker und das von Ganeti genutzte KVM miteinander verglichen.

Der Vergleich wurde auf der Cloud-Computing-Plattform Microsoft Azur [118] in einer VM des Typs *D4s_v3* mit dem Betriebssystem *Ubuntu 20.04 LTS* durchgeführt. Dieser VM-Type bietet 4 vCPUs und 16 GB Arbeitsspeicher [119]. Als Speicher kam eine *Premium Solid State Drive* der Leistungsstufe P30 mit maximal 5000 Input/Output Operations Per Second (IOPS) [119] und einem maximalen Durchsatz von 200 MBit/s zum Einsatz [120]. Diese Leistungsstufe wurde ausgewählt, da sie die geringste Leistungsstufe ist, bei der das sogenannte *bedarfsgesteuerte Bursting* das *guthabenbasierte Bursting* ersetzt. Die Bursting-Arten werden in der Azur-Dokumentation [121] beschrieben. Beim guthabenbasiertem Bursting hat jede Festplatte ein kostenloses Guthaben für das IOPS-Bursting und eins für das Durchsatzt-Bursting. Diese Guthaben sammeln sich bis zum Erreichen der Guthabengrenze an, wenn die angeforderte Leistung unter der Leistungsgrenze der Festplatte liegt und werden automatisch wieder verbraucht, wenn die Leistungsgrenze überschritten wird. Beim bedarfsgesteuerten Bursting existiert hingegen kein Guthaben, stattdessen muss das Bursting manuell aktiviert werden. Da für den Performancetest eine möglichst konstante Leistung benötigt wird, ist das bedarfsgesteuerte Bursting von klarem Vorteil.

Der Docker Container wurde, wie im DMC-Umfeld, zusätzlich in einer KVM VM ausgeführt, welche als Docker Host fungiert. Der Aufbau der Testlandschaft wurde in Abbildung 13 grafisch dargestellt und die Details in Tabelle 2 aufgelistet. Die verwendeten Befehle zum Erstellen der KVM VMs und des Docker Containers befindet sich im Anhang L.

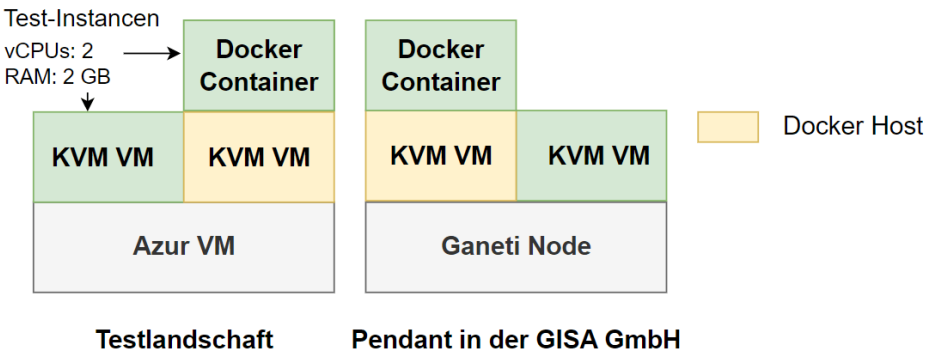


Abbildung 13: Aufbau der Testlandschaft für die Bestimmung des Ressourcenverbrauchs und der Durchführung der Performancetests

Ressource	KVM VM Docker-Host	KVM VM Test-Instanz	Docker Container Test-Instanz
Betriebssystem	Ubuntu 20.04	Ubuntu 20.04	Ubuntu 20.04 (Base-Image)
vCPUs	3	2	2
Arbeitsspeicher	4 GB	2 GB	2 GB

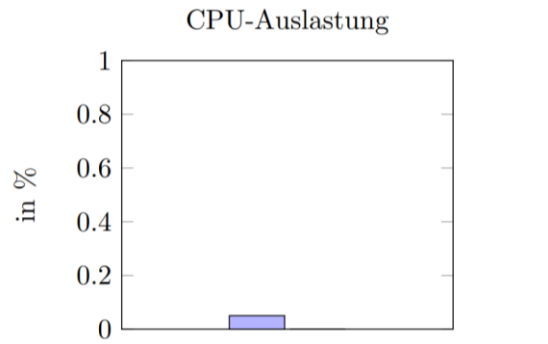
Tabelle 2: Übersicht über die Konfiguration der KVM VMs und Container

Zur Bestimmung des Ressourcenverbrauchs wird für die VM die Open Source Systemüberwachungssoftware *nmon* [122] und für die Docker Container der Docker Befehl `docker stats` eingesetzt. Während nmon innerhalb der Testinstanz läuft, wird der Befehl `docker stats` auf dem Docker Host ausgeführt. Für die Performancetests wurde hauptsächlich die *Phoronix Test Suite* verwendet, eine von Phoronix Media bereitgestellte Open Source Benchmark-Software, welche u. a. für Linux und Windows verfügbar ist und sowohl die Installation der Tests als auch deren Durchführung übernimmt [123]. Die Phoronix Test Suite bietet zurzeit 701 Tests und 85 Test-Suites, d. h. Sammlungen von Test zu einem Thema, und ermöglicht zudem das Erstellen eigene Tests/Test-Suites [124]. Für die Ausführung der Tests im Container wurde ein eigenes Image erstellt, welches auf dem im offiziellen GitHub-Repository hinterlegte Skript zur Erstellung eines Container Images basiert [125]. Die dazugehörige Dockerfile befindet sich im Anhang M. Die Phoronix Test Suite führt jeden Tests mindestens dreimal durch, erhöht die Testanzahl jedoch automatisch, sollte die Abweichung größer sein als für diesen Test üblich. Bei der Auswertung des Tests ist anzumerken, dass bei der Unterteilung in CPU-, Datenträger, Netzwerk- und weitere Kernel-Performancetests zwar jeweils der Fokus auf den jeweiligen Aspekt gelegt wird, sich diese Performance-parameter allerdings gegeneinander beeinflussen, die CPU-Performance also auch Einfluss auf die Netzwerk-Performance hat usw. Der vollständige Phoronix Testbericht kann in Anhang R und Anhang S eingesehen werden.

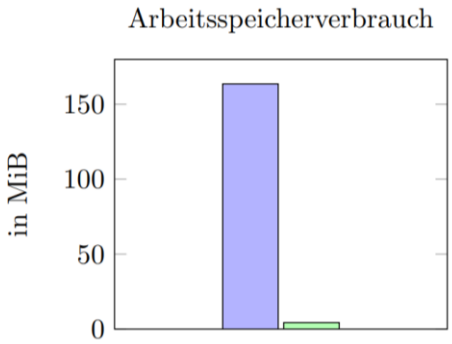
3.2.1 Ressourcenverbrauch im Leerlauf

Zunächst wurde der Ressourcenverbrauch im Leerlauf betrachtet, dafür wurde in der VM und im Container jeweils ein nginx-Server gestartet und die Arbeitsspeicher- und CPU-Auslastung über eine Zeit von 60 Sekunden mit dem genannten Tool aufgezeichnet. Die verwendeten Befehle befinden sich in Anhang N. Da es beim Starten der Aufzeichnung für einige Sekunden zu einer erhöhten CPU-Auslastung kam, welche voraussichtlich auf das Starten der Aufzeichnung zurückzuführen ist und somit nicht Teil der Auslastung im Leerlauf ist, wurden die ersten zehn Sekunden bei der Auswertung nicht beachtet. Bei dem Vergleich der Wertepaare muss zudem darauf geachtet werden, dass die Daten, wie eingangs erwähnt, auf verschiedene Art und Weise

erhoben wurden, sodass z. B. unterschiedliche Rundungsregeln zu gewissen Messungenauigkeiten führen können.



CPU-Auslastung in %	Testergebnis	σ
VM	0.05	0.15 %
Docker	0	0 %



Arbeitsspeicherauslastung	Testergebnis	σ
VM	163.5 MiB	0.74 %
Docker	4.3 MiB	0 %
Differenz	-97.36 %	

Abbildung 14: Ressourcenverbrauch im Leerlauf

Wie man an den Ergebnissen in Abbildung 14 erkennen kann, verbraucht der Docker Container im Durchschnitt mit 4.3 MB 97.36 % weniger Arbeitsspeicher als die VM (163.5 MB). Der niedrigere Arbeitsspeicherverbrauch lässt sich auf die Tatsache zurückführen, dass die VM das gesamte Betriebssystem inkl. Kernel in den Arbeitsspeicher laden muss, während der Docker Container auf den Kernel des Docker-Hosts zurückgreift. Die durchschnittliche CPU-Auslastung liegt hingegen bei beiden Virtualisierungstechnologien nahe 0 % (jeweils <0.1 %). Aufgrund der geringen Differenz zwischen den Werten und der erläuterten möglichen Messungenauigkeit muss geschlussfolgert werden, dass sich die CPU-Auslastung nicht signifikant unterscheidet.

3.2.2 Ressourcenverbrauch der Containervirtualisierungsinfrastruktur

Beim Betrieb eines IT-Service ist allerdings zu beachten, dass nicht nur die Software selbst, sondern auch die zugrunde liegende Infrastruktur einen gewissen Ressourcenverbrauch hat. Dies ist insbesondere bei der Containervirtualisierung relevant, da der Betrieb des Docker Hosts und der Kubernetes Nodes einen Ressourcenverbrauch verursachen, der beim Betrieb mittels VM entfällt. Um den Ressourcenverbrauch eines Containers mit der einer VM vergleichen zu können, muss dieser zusätzliche Ressourcenverbrauch auf alle im Cluster betriebenen Container verteilt werden.

Der Ressourcenverbrauch der Kubernetes Infrastruktur besteht zum einen aus den Control Plane Nodes, von denen der aktuelle DMC Kubernetes Cluster eine besitzt, und zu anderen aus dem Teil des Ressourcenverbrauchs der Worker Nodes, der durch das Betriebssystem und die Software, die für die Verwaltung des Clusters und die Bereitstellung der Cluster-Funktionen zuständig ist, verursacht wird. Der Control Plane Node wurden ca. 4 GB Arbeitsspeicher (3931.7 MiB) und 2 CPU-Kernen zugeordnet

[Anhang Q]. Der beschriebene Ressourcenverbrauch der Worker Nodes besteht zunächst aus der Differenz der Ressourcen, die Node insgesamt besitzt (*Capacity*) und den Ressourcen, die tatsächlich den Pods zur Verfügung stehen (*Allocatable*) [126][Anhang Q]. Dieser beträgt bei beiden Worker Nodes jeweils 100 MiB. Hinzu kommen Pods, die keine IT-Services, sondern Cluster-Funktionen, wie das Routing, bereitstellen. Deren Ressourcenverbrauch lässt sich über das Kubernetes Dashboard [127][Anhang P] und dem Befehl *kubectl top nodes* [Anhang O] auslesen und beträgt insgesamt 920.9 MiB Arbeitsspeicher und 149m[7] CPU-Kerne. Somit ergibt sich für die Kubernetes Infrastruktur ein Ressourcenverbrauch von insgesamt 4852.6 MiB Arbeitsspeicher und 2.1 CPU-Kernen.

Bei Docker besteht der Ressourcenverbrauch der Infrastruktur lediglich aus dem Ressourcenverbrauch des Betriebssystems des Docker Hosts, der Docker Engine und der zusätzlich auf dem Docker Host installierter Software, z. B. für das Monitoring. Deren Arbeitsspeicherauslastung beträgt insgesamt ca. 500 MiB Arbeitsspeicher [Anhang W]. Auf dem System der folgenden Performancetests konnte für die Docker Engine bei zehn laufenden Ubuntu Containern eine CPU-Auslastung von < 1 % festgestellt werden, mit vereinzelten Sprüngen auf Werte > 1 %.

Je nachdem, wie viele IT-Services auf dem Docker Host bzw. im Kubernetes Cluster betrieben werden, desto kleiner wird der Infrastruktur-Overhead pro IT-Service. Die resultierende Summe aus dem Ressourcenverbrauch im Leerlauf (vgl. Abschnitt 3.2.1) und der Infrastruktur wurde in Tabelle 3 an Beispiele des Arbeitsspeichers für 10 und 25 Container dargestellt.

Arbeitsspeicherauslastung bei	10 IT-Services	25 IT-Services
VM	163.5 MiB	163.5 MiB
Docker	50.4 MiB	20.2 MiB
Differenz zu VM	-69.2 %	-87.7 %
Kubernetes	485.7 MiB	194.3 MiB
Differenz zu VM	+197.1 %	+18.8 %

Tabelle 3: Summe des Ressourcenverbrauchs (Arbeitsspeicher) im Leerlauf und der Infrastruktur für 10 und 25 IT-Services

Wie man sieht, schneidet Docker, trotz des Infrastruktur-Overheads, in beiden Beispielszenarien besser ab als die VM, während Kubernetes in beiden Fällen mehr Ressourcen benötigt. Grund dafür ist die Control Plane Node, welche dem Kubernetes Cluster insbesondere die Verwaltung der Nodes und die Bereitstellung der Hochverfügbarkeit ermöglicht. Der komplette Funktionsverlauf für bis zu 38 IT-Services ist in

[7] Kubernetes Einheit: 1 CPU-Kern = 1000m [128].

Abbildung 15 dargestellt. Anhand der Schnittpunkte lässt sich ablesen, dass Docker ab 4 und Kubernetes ab 31 betriebenen IT- Services im Vorteil ist.

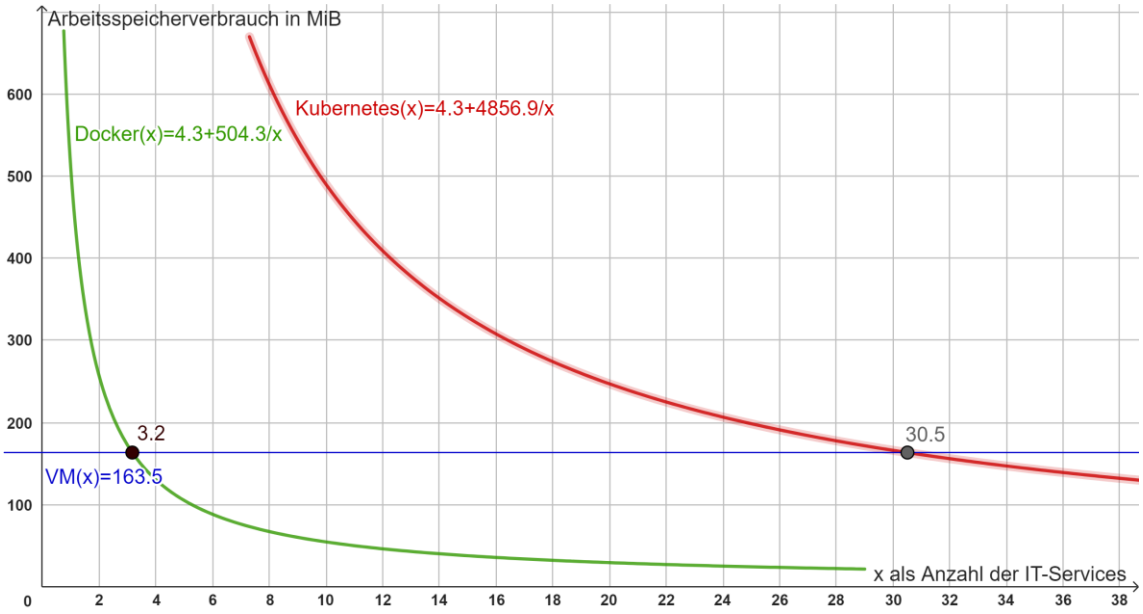


Abbildung 15: Summe des Ressourcenverbrauchs im Leerlauf und der Infrastruktur (Arbeitsspeicher) in Abhängigkeit der Anzahl der IT-Services mit gekennzeichneten Schnittpunkten

3.2.3 CPU – Performancetests

Zur Bewertung des durch die Containerisierung hervorgerufenen CPU-Overheads wurden vier Tests aus den Test Suites *CPU Massive* [129] und *Single-Threaded* [130] ausgewählt, um eine möglichst große Bandbreite an verschiedenen Arbeitslasten abzudecken, die hohe CPU-Anforderungen haben. Folgende Tests wurden ausgeführt und anschließend ausgewertet:

GnuPG:

Dieser Test misst die Zeit, die für das Verschlüsseln mittels der Kryptografie-Software GnuPG benötigt wird [131].

7-Zip Compression:

Dieser Test überprüft die Kompressions- und Dekompressionsleistung des Packprogramms 7-Zip unter Verwendung der integrierten Benchmarking-Funktion [132].

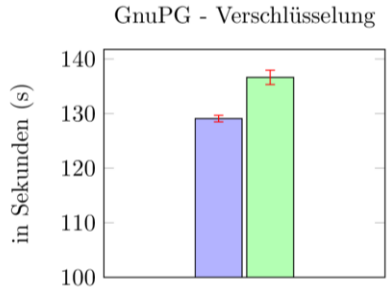
nginx:

Dieser Test verwendet das HTTP benchmarking Programm *wrk*, um die Anzahl der Anfragen pro Sekunde zu messen, die der nginx-Webserver mit 100 offenen Verbindungen verarbeiten kann [133]. nginx ist der im DMC-Umfeld standardmäßig eingesetzte Webserver und Reverse Proxy.

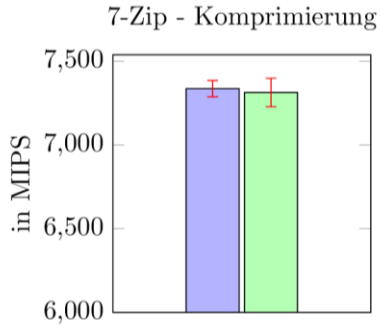
MariaDB:

Dieser Benchmark testet die Leistung eines MariaDB Datenbankservers unter Verwendung der benchmarking Software *mysqlslap*. Der Test misst die Anzahl der Abfragen pro Sekunde eines einzelnen Clients, die der Server verarbeiten kann [134]. Die MariaDB Datenbank ist im DMC-Umfeld weit verbreitet [Anhang W]

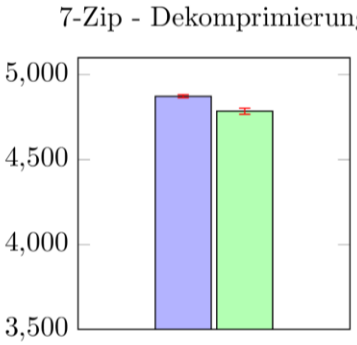
Aus den Testergebnissen in Abbildung 16 ist erkennbar, dass der Docker Container meist eine leicht schlechtere CPU-Performance bietet als die VM. Vergleichsweise stark fällt der Unterschied bei der Verschlüsselung mittels GnuPG (5.85 % langsamer) und der Anzahl der verarbeiteten nginx Anfragen (3.87 % weniger) aus. Auch bei der Dekomprimierung mittels 7-Zip schneidet Docker leicht schlechter ab (-1.81 %). Bei der Komprimierung mittels 7-Zip und der Anzahl der MariaDB Datenbankabfragen sind die Performanceunterschiede hingegen so gering (jeweils <0.5 %), dass sie sich innerhalb des Standardfehlers (im Balkendiagramm rot dargestellt) und der Standardabweichung (σ) befinden. Sollte die Differenz innerhalb dieser Werte liegen, unterscheiden sich die Werte nicht signifikant genug, um eine sinnvolle Aussage treffen zu können. Bei den zuvor ausgewerteten Performancetests liegt die Differenz deutlich außerhalb der Standardabweichung von unter 1 %, bzw. 1.7 % bei dem Docker Verschlüsselungstest.



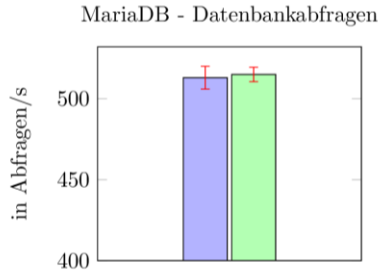
Verschlüsselung	Testergebnis	σ
VM	129.09 s	0.8 %
Docker	136.64 s	1.7 %
Differenz	+5.85 %	



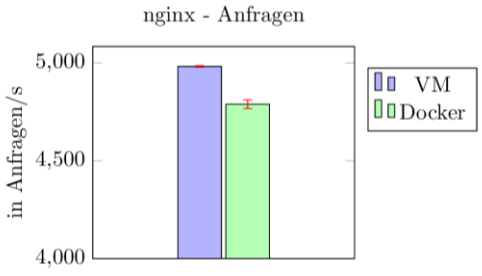
Komprimierung	Testergebnis	σ
VM	7336 MIPS	2.6 %
Docker	7313 MIPS	2 %
Differenz	-0.31 %	



Dekomprimierung	Testergebnis	σ
VM	4873 MIPS	0.7 %
Docker	4785 MIPS	0.6 %
Differenz	-1.84 %	



Datenbankabfragen	Testergebnis	σ
VM	513	2.4 %
Docker	515	1.5 %
Differenz	+0.39 %	



Anfragen	Testergebnis	σ
VM	4982	0.1 %
Docker	4789	0.8 %
Differenz	-3.87 %	

Abbildung 16: Testergebnisse der CPU – Performancetests

3.2.4 Datenträger – Performancetests

Die Performance des Datenträgers wurde mit zwei verschiedenen Tests überprüft, zum einen mit dem Phoronix *PostMark* Benchmark aus der *Disk Test Suite* [135] und zum mit der benchmarking Software *sysbench*:

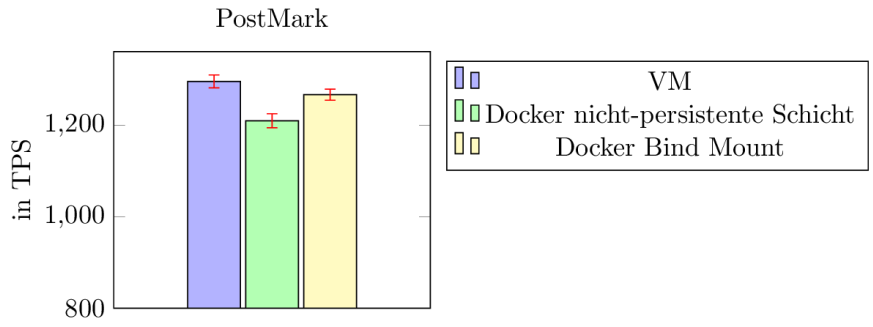
PostMark:

Laut Testbeschreibung [136] bestimmt PostMark die Leistung des Datenträgers bei kleinen Dateigrößen zwischen 5 und 512 Kilobyte. Der Test besteht aus 500 Dateien und 25.000 Transaktionen. Dafür werden die durchschnittlich erreichten Transaktionen pro Sekunde (kurz TPS) ermittelt.

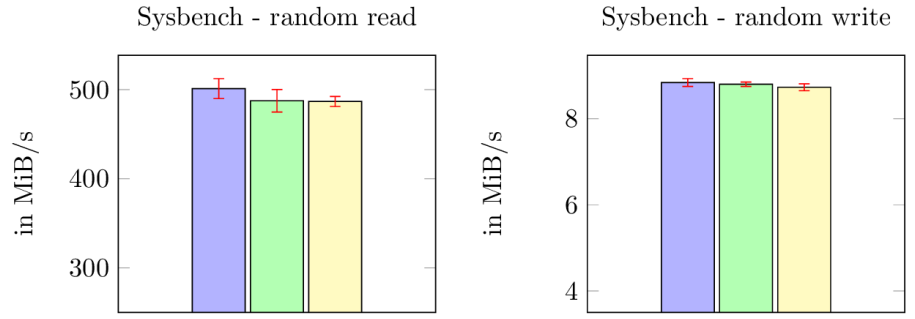
Sysbench:

Sysbench ist eine Sammlung von verschiedenen Performancetests, darunter auch das Modul *fileio* zur Bestimmung der Lese- und Schreibgeschwindigkeit bei zufälligen und seriellen Dateizugriffen [137]. Der Performancetest wurde in der Standardkonfiguration durchgeführt, bei der 128 Dateien mit einer Gesamtgröße von 2 GB verwendet werden [137]. Es wurden für jede Testvariation fünf Testdurchläufe durchgeführt.

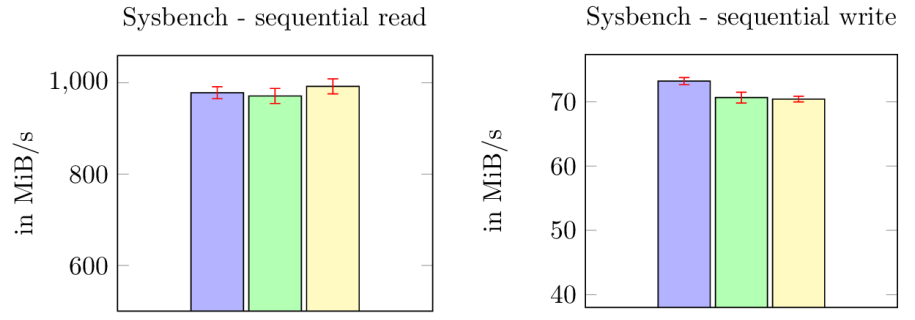
Da sich bei Containern die nicht persistente Schicht in ihrer Funktionsweise stark von den eingehängten persistenten Speichern unterscheidet, wurde für Docker die Datenträgerperformance für beide Speicherortarten getestet. Bei den eingehängten persistenten Speichern wurde ein Bind Mount (Vgl. Abschnitt 2.5.3) gewählt.



Postmark	Testergebnis	σ
VM	1296 TPS	2.4 %
Docker nicht-persistente Schicht	1210 TPS	2.2 %
→ Differenz zu VM	-6.64 %	
Docker Bind Mount	1267 TPS	3.3 %
→ Differenz zu VM	-2.24 %	



Sysbench	random read	σ	random write	σ
VM	501.17 MiB/s	4.99 %	8.84 MiB/s	2.18 %
Docker nicht-persistente Schicht	487.53 MiB/s	5.82 %	8.80 MiB/s	1.35 %
→ Differenz zu VM	-2.72 %		-0.36 %	
Docker Bind Mount	486.74 MiB/s	2.61 %	8.73 MiB/s	2.06 %
→ Differenz zu VM	-4.16 %		-1.20 %	



Sysbench	sequential read	σ	sequential write	σ
VM	978.36 MiB/s	3.02 %	73.23 MiB/s	1.67 %
Docker nicht-persistente Schicht	971.24 MiB/s	3.79 %	70.66 MiB/s	2.65 %
→ Differenz zu VM	-0.73 %		-3.52 %	
Docker Bind Mount	992.31 MiB/s	3.69 %	70.43 MiB/s	1.39 %
→ Differenz zu VM	1.43 %		-3.82 %	

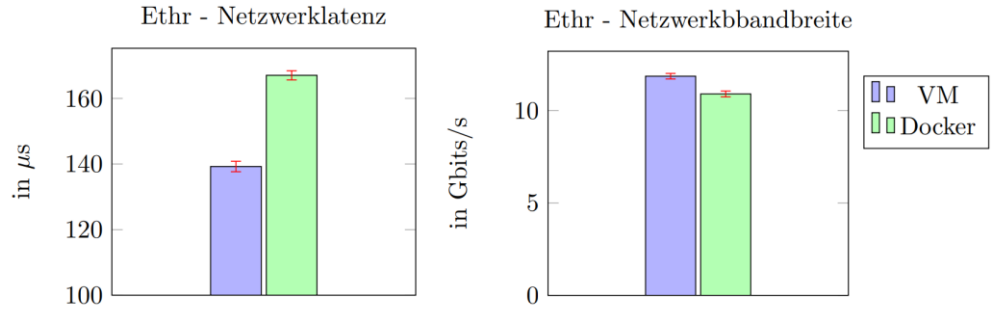
Abbildung 17: Testergebnisse der Datenträger – Performancetests

Die in Abbildung 17 aufgezeigten Ergebnisse zeigen, dass beim PostMark Benchmark die nicht persistente Schicht eine schlechtere Performance bietet als die VM (-6.64 %), während die Performanceeinbußen des Bind Mounts mit -2.24 % so gering sind, dass

sie innerhalb der Bind Mount Standardabweichung von 3.3 % liegen. Bei den Sysbench Performancetest liegen hingegen sowohl die nicht persistente Schicht, also auch der Bind Mount bei fast allen Testvariationen innerhalb der Standardabweichung. Lediglich bei den *sequential write* Docker Tests liegen die Performanceeinbußen außerhalb der Standardabweichung von bis zu 2.65 %, mit einer minimal niedrigeren Schreibgeschwindigkeit von -3.52 % bzw. -3.82 % im Vergleich zur VM. Anzumerken ist allerdings, dass für die Sysbench Performancetests eine im Vergleich zu den anderen Performancetests höhere Standardabweichung von bis zu 5.82 % ermittelt wurden.

3.2.5 Netzwerk – Performancetests

Ergänzend zu dem in Abschnitt 3.2.3 durchgeführten nginx Performancetest, wurde mithilfe des Phoronix Benchmarks Ethr aus der Networking Test Suite [138] die Netzwerkbandbreite und -latenz ermittelt. In dem durchgeführten Tests wurde der dazugehörige Ethr-Server durch die Phoronix Benchmark Suite auf demselben Host ausgeführt (localhost) [139] und das Transmission Control Protocol (TCP) [140] verwendet.



Latenz	Testergebnis	σ
VM	139 μs	4.4 %
Docker	167 μs	1.4 %
Differenz	+19.97 %	

Bandbreite	Testergebnis	σ
VM	11.86 Gbits/s	2.2 %
Docker	10.90 Gbits/s	5.8 %
Differenz	-8.09 %	

Abbildung 18: Testergebnisse der Netzwerk - Performancetests

Wie in den Testergebnissen in Abbildung 18 zu sehen ist, ergeben sich durch die Verwendung von Docker deutliche Einbußen bei der Netzwerkperformance. Bei der Netzwerkbandbreite erreicht Docker mit 10.90 Gbits/s ein um ca. 8 % geringeres Ergebnis als die VM mit 11.86 Gbits/s und die Netzwerklatenz ist mit 167 µs bei Docker fast 20 % höher als bei der VM. Die ermittelten Differenzen liegen deutlich außerhalb der Standardabweichung von unter 6 %.

Anzumerken ist allerdings auch, dass sowohl 10.90 Gbits/s Netzwerkbandbreite als auch 167 µs Netzwerklatenz für die allermeisten IT-Services mehr als ausreichend sein dürften und andere Faktoren, wie die physische Netzwerkinfrastruktur und die

Entfernung zwischen dem Server und dem Nutzer des IT-Services eine deutlich höhere Auswirkung auf die Performance haben können [141].

3.2.6 Weitere Kernel – Performancetests

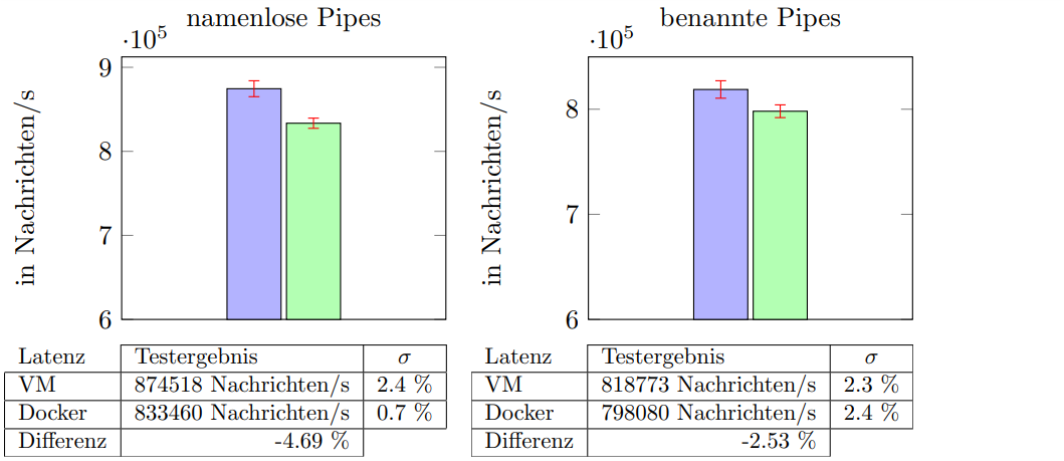
Ergänzend zu dem bereits durchgeführten Tests, deren Fokus auf der CPU, Datenträger, und Netzwerkperformance lag, wurden mit MBW und IPC_benchmark weitere Performancetests aus der *Common Kernel Benchmarks* [142] Test Suite durchgeführt, um den Einfluss der Containerisierung auf diese Kernel-Funktionen zu bestimmen. Die bereits durchgeführten Tests PostMark und Ethr, sowie Test, welche mit dem nginx- und MariaDB-Test vergleichbar sind, finden sich ebenfalls in dieser Test Suite wieder.

MBW:

MBW ist ein Benchmark für die Bandbreite von Kopieroperationen im Arbeitsspeicher [143]. In der gewählten Testkonfiguration betrug die Array-Größe 128 MiB.

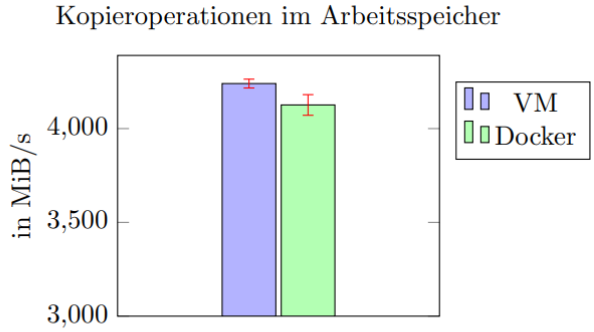
IPC_benchmark:

IPC_benchmark ist ein Linux-Benchmark für die Interprozesskommunikation [144]. Im Rahmen der Testdurchführung wurden sowohl Namenlose als auch Benannte Pipes betrachtet.



Latenz	Testergebnis	σ
VM	874518 Nachrichten/s	2.4 %
Docker	833460 Nachrichten/s	0.7 %
Differenz	-4.69 %	

Latenz	Testergebnis	σ
VM	818773 Nachrichten/s	2.3 %
Docker	798080 Nachrichten/s	2.4 %
Differenz	-2.53 %	



Kopieroperationen	Testergebnis	σ
VM	4240 MiB/s	1 %
Docker	4126 MiB/s	2.3 %
Differenz	-2.69 %	

Abbildung 19: Testergebnisse der weiteren Kernel - Performancetests

Die in Abbildung 19 dargestellten Ergebnisse zeigen, dass Docker bei der Interprozesskommunikation mit -4.69 % bei namenlosen und -2.53 % bei benannten Pipes erneut leicht schlechter abschneidend als die VM, diese Differenzen ist allerdings nur leicht größer sind als die Standardabweichung von bis zu 2.4 %. Ähnliches gilt für die Arbeitsspeicher Kopieroperationen, bei denen Docker mit -2.69 % erneut nur leicht schlechter als die VM abschneidet. Diese Differenz allerdings nur minimal größer als die Standardabweichung von bis zu 2.3 %.

3.2.7 Fazit des Aspekts Ressourcenverbrauch und Performance

Zusammenfassend lässt sich sagen, dass ein Container für sich zwar deutlich weniger Arbeitsspeicher benötigt als eine VM (-160 MB/-97 %), aber dafür bei der Containervirtualisierung noch ein zusätzlicher Ressourcenverbrauch durch die Infrastruktur entsteht. Je nachdem, wie viele IT-Services betrieben werden, sind so entweder die VM oder die Containervirtualisierung im Vorteil. Im Hinblick auf die Kosten pro CPU-Kern bzw. GiB Arbeitsspeicher [145] im Vergleich zu den Kosten für Personalaufwände [146] ist der Unterschied im Normalfall jedoch nicht relevant.

Bei den Performancetests variieren die Ergebnisse von Test zu Test stark. Während bei den meisten CPU-, Datenträger- und weiteren Kernel-Performancetests entweder keine signifikanten oder nur sehr geringe Performanceverluste durch die Verwendung von Docker ermittelt werden konnten, gibt es auch einzelne Tests, welche sich deutlicher von der Standardabweichung absetzen und Einbußen von bis zu 6.64 % aufweisen. Des Weiteren zeichnen sich bei den Netzwerk-Performancetests deutliche Einbußen von bis zu 20 % ab. Welchen Einfluss die Containerisierung auf die Performance eines IT-Services hat, ist somit stark von dessen konkreten Anforderungen abhängig.

3.3 Aufwand

Um zu vergleichen, wie groß der Aufwand für den Betrieb eines IT-Services in Abhängigkeiten zur verwendeten Virtualisierungstechnologie ist, wurden, mit einer *Angular*-basierten Webapplikation und der *inubit BPM* Software, zwei fürs DMC-Umfeld typische IT-Services ausgewählt und deren Aufwände in verschiedenen Situationen im Rahmen einer Umfrage von Mitarbeitern des DMC-Umfelds geschätzt. Da es im realen Geschäftsbetrieb auch möglich ist bereits vorhandene Konfigurationen als Vorlage zu verwendet, ist dies, solange nicht anders angegeben, den Umfrageteilnehmern dieser Umfrage ebenfalls gestattet.

Angular ist laut dessen Dokumentation [4] ein durch Google entwickeltes Front-End-Webapplikationsframework für Single Page Applications. Diese zeichnen sich dadurch aus, dass alle Funktionen auf ein und derselben HTML Seite ausgeführt werden und, anstatt die gesamte Seite neu zu laden, immer nur der Teil des HTML DOMs neu gerendert wird, der sich auch tatsächlich ändert. Um dynamische Daten darzustellen, werden meist REST-Services angesprochen.

Die inubit BPM ist eine Java basierende BPM Software der Virtimo AG, welche die grafische Konfiguration ausführbarer Prozesse und die Dokumentation dieser mittels BPMN 2.0 oder Prozesslandkarte ermöglicht [147]. Durch die inubit BPM werden u. a. die REST APIs für die Angular Applikationen bereitgestellt.

Alle Umfragen, die die Angular-basierten Webapplikation betreffen, befinden sich im Anhang T und die Umfragen bezüglich der inubit BPM befinden sich Anhang U. Unabhängig von diesen IT-Services, wurde der Aufwand für die Betreuung des Kubernetes Clusters und des Docker Hosts geschätzt, diese Umfragen befinden sich im Anhang V. Teilweise wurde in den Umfragen für den Aufwand eine Zeitspanne geschätzt. In diesen Fällen wurde für den Vergleich der Mittelwert gebildet und in den Diagrammen die Spannweite durch die roten Fehlerindikatoren dargestellt.

3.3.1 Verwendete Softwarelösungen

Neben den bereits beschriebenen Virtualisierungstechnologien werden für die Bereitstellung von IT-Services in der GISA GmbH noch weitere Softwarelösungen eingesetzt [Anhang Z, Anhang AA, Anhang BB]. Diese werden anschließend kurz erläutert.

3.3.1.1 Jenkins

Jenkins ist laut dessen Dokumentation [148] ein Automatisierungsserver, welcher sich durch zahlreiche Plugins erweitern lässt. So ist es unter anderem möglich, Container

Images basierend auf einer, in einem git-Repository hinterlegtem, Dockerfile zu bauen und Bash Skripte auszuführen. Die Konfiguration erfolgt über eine *Jenkins Pipeline*, welche in einer sogenannten *Jenkinsfile* in der Skriptsprache *Groovy* [149] definiert wird. Eine Pipeline besteht aus einer oder mehrere sequenziell ausgeführten Automationen und unterstützt u. a. Umgebungsvariablen und bedingte Anweisungen.

3.3.1.2 Nexus Repository Manager

Der Nexus Repository Manager von Sonatype ist laut dessen Webseite [150] ein Repository Verwaltungssystem für Pakete, Binärdateien und Build-Artefakte. Zu den unterstützten Ökosystemen gehören JavaScript mit dem Paketmanager npm, Java mit dem Build Tool Apache Maven, Linux mit Paketmanagern APT und YUM und Docker und Kubernetes mit der Bereitstellung von Container Images und Helm Charts (siehe Abschnitt 3.3.1.3) über deren respektive Repositories. Das Nexus Repository ermöglicht neben dem Caching externe Repositories auch das Hosten privater Pakete, etc. und bietet mit dem Nexus IQ Server einen automatischen Sicherheitsscanner z. B. für Container Images. Der Zugriff auf die Repositories erfolgt über eine RBAC.

3.3.1.3 Helm

Helm ist laut dessen Dokumentation [151] ein zur CNCF gehörendes Softwarewerkzeug, um Vorlagen, sogenannte Templates, für Kubernetes Ressourcen zu erstellen, welche für die Bereitstellung eines speziellen IT-Services benötigt werden, und diese in einem sogenannten Chart zu bündeln. Wie in Abbildung 20 zusehen, übergibt der Entwickler lediglich Parameter (*Values*), die den einzelnen IT-Service von den Chart-Standardeinstellungen unterscheidet. Helm generiert, basierend auf den Values, alle benötigten Kubernetes Ressourcen und stellt diese über die Kubernetes API im Cluster bereit. Die Helm Software läuft auf dem Rechner des Entwicklers und die Charts können entweder lokal gespeichert oder über ein Repository bereitgestellt werden.

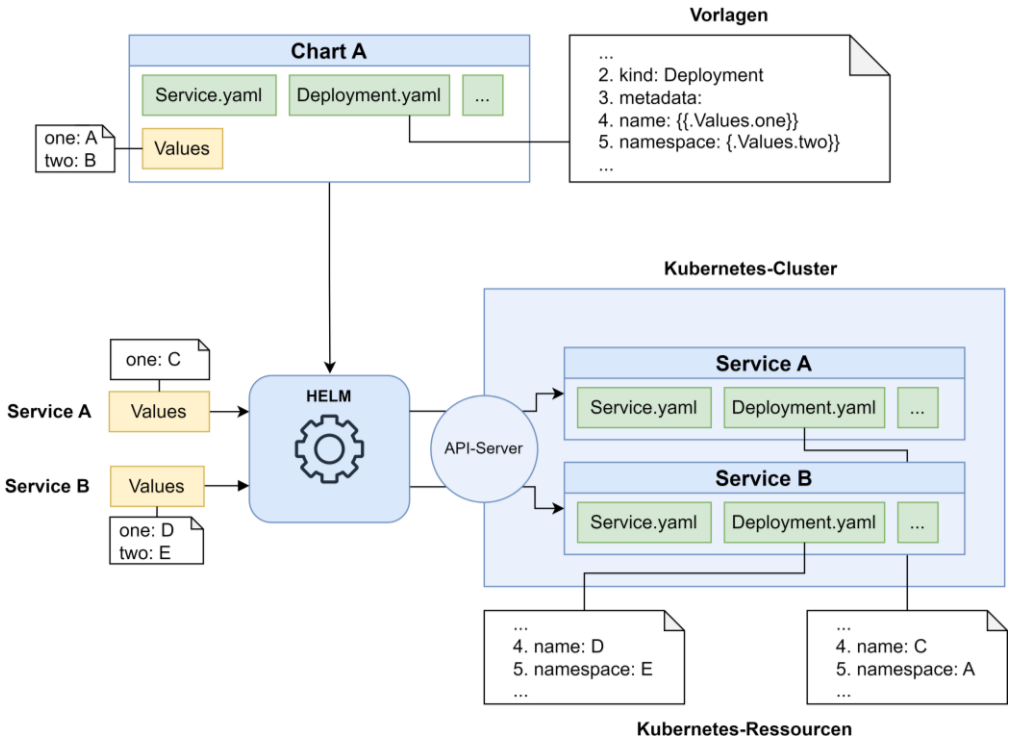


Abbildung 20: Funktionsweise der Software Helm

3.3.1.4 Kompose

Kompose ist ein zu Kubernetes gehöriges Softwarewerkzeug, um die Migration von Docker Compose zu Kubernetes zu erleichtern [152]. Kompose übernimmt dafür die Konvertierung zahlreicher Docker Compose Konfigurationsparameter in die entsprechenden Kubernetes Ressourcen, eine vollständige Übersicht über die unterstützten Werte findet sich in der *Conversion Matrix* [153].

3.3.2 Initiale Installation und Konfiguration

In einem ersten Schritt soll der Aufwand für die initiale Installation und Konfiguration der IT-Services beachtet werden.

3.3.2.1 Angular-basierte Webapplikation

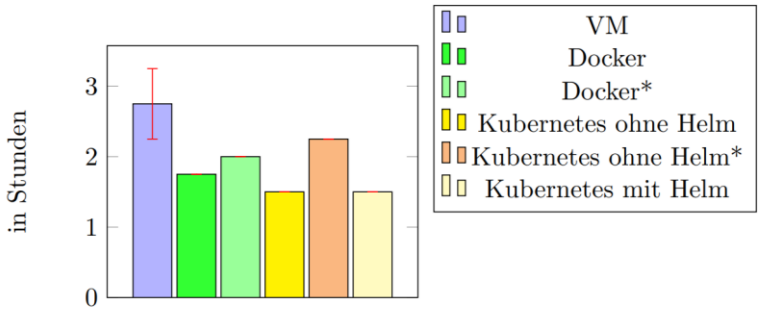
Aus der Umfrage ergibt sich, dass zunächst ein nginx-Server und eine dazugehörige Konfiguration benötigt wird, um die Angular-basierte Webapplikation bereitzustellen. Des Weiteren muss zum einen aus dem Angular Quellcode ein *Build* erstellt werden, d. h. der vom Entwickler erstellte Quellcode und dessen Abhängigkeiten werden in einige wenige JavaScript-Dateien zusammengefasst und komprimiert, und zum anderen müssen ggf. automatisierte Softwaretests durchgeführt werden. Für das Erstellen des Builds und das Durchführen der Softwaretest wird eine Jenkins Pipeline erstellt und ausgeführt. Abschließend muss sichergestellt werden, dass die Webapplikation über die gewünschte URL durch den Endanwender aufrufbar ist.

Für die Bereitstellung über eine VM muss zunächst eine VM beantragt werden. Nachdem die VM erstellt wurde, muss der nginx Server auf der VM installiert und die nginx-Konfiguration erstellt und auf dem Server hinterlegt werden. Anschließend muss die erwähnte Jenkins Pipeline, über die das Angular Build generiert und auf den Server übertragen wird, erstellt und ausgeführt werden. Abschließend erfolgt die Freischaltung der in der Firewall und Konfiguration des zentralen Reverse Proxys, um die Anwendung über die URL erreichbar zu machen.

Für die Nutzung des Docker Host müssen die Anzahl der Container (hier einer) und der geschätzte Ressourcenverbrauch für die Abrechnung und die Verwaltung des Hosts gemeldet werden. Für die nginx-Server kann das offizielle nginx Container Image [154] verwendet werden, sodass lediglich die nginx-Konfiguration erstellt werden muss. In der Jenkins Pipeline, welche erstellt und ausgeführt werden muss, wird zusätzlich zum erwähnten Build-Prozess, ein neues Container Image erstellt und in das Nexus Repository geladen. Dieses Container Image stellt einen einsatzbereiten nginx Server dar, welcher das erstellte Angular Build und die nginx-Konfiguration beinhaltet und auf dem erwähnten offizielle nginx Image aufbaut. Die benötige Dockerfile muss ebenfalls durch den Entwickler erstellt werden. Anschließend muss die Docker Compose Datei erstellt und die Applikation auf dem Docker Host bereitgestellt werden. Abschließend muss, wie bei der Verwendung einer VM, sichergestellt werden, dass die Webapplikation über die gewünschte URL durch den Endanwender aufrufbar ist.

Bei der Bereitstellung über Kubernetes wurde zwischen der Bereitstellung mit und ohne Helm Chart unterschieden. In beiden Fällen müssen zunächst erneut die Anzahl der benötigten Container und deren Ressourcenverbrauch gemeldet werden. Anschließend müssen dieselbe Dockerfile und Jenkins Pipeline erstellt werden, die bereits bei der Docker-Bereitstellung eingesetzt wurden. Bei der Bereitstellung ohne Helm müssen anschließend die Konfigurationsdateien für die Kubernetes Ressourcen (vgl. Abschnitt 2.6.1) erstellt und abschließend die Ressourcen im Kubernetes Cluster bereitgestellt werden. Bei der Verwendung eines Helm Charts müssen lediglich die Values angepasst werden. Die Bereitstellung der Applikation über die gewünschte URL wurde bereits über die Kubernetes Ressourcen bzw. die Values konfiguriert und wird über den Kubernetes eigenen Ingress Controller sichergestellt.

Neben einer Aufwandsschätzung, bei der alle bereits existierenden Konfigurationen als Vorlage genutzt werden durften, sollte in diesem Beispiel auch der Aufwand geschätzt werden, wenn die bereits existierende Docker Compose bzw. Kubernetes Konfiguration nicht verwendet werden kann, um so einen Vergleich zur Nutzung von Helm Charts in diesem Szenario zu ermöglichen.



Installation Angular	Aufwand	Spannweite
VM	2.75 h	± 0.5 h
Docker	1.75 h	± 0 h
Docker*	2 h	± 0 h
Kubernetes ohne Helm	1.5 h	± 0 h
Kubernetes ohne Helm*	2.25 h	± 0 h
Kubernetes mit Helm	1.5 h	± 0 h

*: Ohne die Verwendung von bereits existierenden Docker Compose/Kubernetes Konfigurationen als Vorlage

Abbildung 21: Aufwand für das initiale Bereitstellen einer Angular-basierte Webapplikation

Aus der Auswertung der Umfrage in Abbildung 21 ist zu sehen, dass die Kubernetes Installationen mit Helm Chart und mit Wiederverwendung bereits existierender Kubernetes Konfigurationen den gleichen und im Vergleich zu den anderen Szenarien am wenigsten Aufwand benötigen (jeweils 1.5 h). Da das Anpassen der existierenden Kubernetes Konfigurationsdateien mit dem gleichen Aufwand geschätzt wurde (0,25 h), wie das Erstellen der Helm Values Datei, konnte das Helm Chart in diesem Szenario keine Zeitersparnis erbringen. Die Bereitstellung über Docker benötig mit Wiederverwendung aller existierender Konfigurationsdateien mit 1.75 h etwas mehr Aufwand, was auf die Konfiguration des Reverse Proxys zurückzuführen ist, der bei Kubernetes bereits innerhalb der Ingress Ressource konfiguriert wird. Wenn die Kubernetes bzw. Docker Compose Konfiguration neu erstellt werden muss, benötigt Docker mit 2h etwas weniger Aufwand als Kubernetes mit 2,25 h. Im Vergleich zur Installation mit Helm haben diese Szenarien einen um 0.5 bzw. 0.75 h größeren Aufwand. Das zeigt, dass die Verwendung eines Helm Charts in diesen Fällen vorteilhaft hinsichtlich des Aufwands sein kann, zumal davon ausgegangen werden kann, dass die Differenz zwischen mit und ohne Helm Chart mit steigender Komplexität der Anwendung größer wird. Es muss allerdings auch beachtet werden, dass zusätzlicher Aufwand anfallen kann, wenn die Anforderungen an den IT-Service die Möglichkeiten des genutzten Helm-Charts im Laufe dessen Lebenszyklus übersteigen sollten. Die Bereitstellung mittels VM benötigt mit 2.75 h am meisten Aufwand. Das liegt u. a. daran, dass der Aufwand zum Anmelden des Containers, im Vergleich zur Beantragung der VM deutlich geringer ausfällt und für die Container keine separate Firewall Freischaltung benötigt wird.

3.3.2.2 inubit BPM

Die Umfrage zeigt, dass bei einer komplexeren Software wie die inubit BPM in ersten Schritt eine Anforderungs- und Abhängigkeitsanalyse erfolgt, sowie daraus folgend die Festlegung der benötigten Systemressourcen. Aus der Abhängigkeitsanalyse gibt sich, dass für die Bereitstellung, neben dem inubit-Server, noch ein relationaler Datenbankserver benötigt wird, zudem wird ein Berechtigungskonzept angelegt und die Umgebungsvariablen und zu ändernde Systemeinstellungen definiert. Für die Installation des inubit-Servers wird von der Virtimo AG ein Installer bereitgestellt. Nach der Installation müssen die Verbindungsinformationen des Datenbankservers in den entsprechenden inubit Konfigurationsdateien hinterlegt, der Datenbanktreiber heruntergeladen und weitere Einstellungen z. B: an der Java Virtual Machine (JVM) [155] getätigt werden.

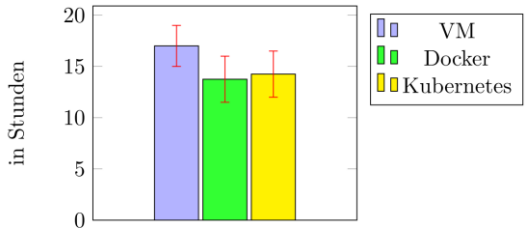
Bei der Bereitstellung mittels VM muss nach Abschluss der Anforderungs- und Abhängigkeitsanalyse, ähnlich wie bei Angular, zunächst eine VM beantragt und die Änderungen an der Firewall und dem Reverse Proxy vorgenommen werden. Wenn die VM zur Verfügung steht, müssen die Betriebssystemnutzer des Berechtigungskonzepts angelegt und die Umgebungsvariablen und Systemeinstellungen angewendet werden. Anschließt kann der Datenbank- und der inubit-Server in der VM installiert werden. Abschließend erfolgt das Hinterlegen der Datenbanktreiber und die Konfiguration des inubit-Servers hinsichtlich Datenbankverbindung, JVM und weitere Einstellungen.

Bei Docker und Kubernetes muss zunächst eine eigene Dockerfile für inubit erstellt werden, da von der Virtimo AG kein offizielles inubit Container Image angeboten wird. Innerhalb der Dockerfile wird der inubit-Installer im nicht interaktiven Modus ausgeführt, ein Non-root User erstellt, die Systemeinstellungen und Umgebungsvariablen gesetzt und ein Skript definiert, welches beim Start des Containers den inubit-Server startet. Für den nicht interaktiven Modus wird allerdings eine sogenannte *Antwortdatei* benötigt, in der die Installer-Einstellungen hinterlegt sind und für dessen Generierung der inubit-Installer im interaktiven Modus ausgeführt werden muss [147]. Dafür kann z. B. ein Ubuntu Container gestartet werden, welcher nach der Generierung der Antwortdatei wieder entfernt werden kann. Bevor der Container allerdings gelöscht wird, sollten neben der Antwortdatei auch die relevanten inubit-Konfigurationsdateien aus dem Container kopiert werden. Aus der Dockerfile wird das inubit Container Image generiert, welches anschließend in das Nexus Repository geladen wird. Für die meisten Datenbankserver, wie die MariaDB, stehen offizielle Container Images zur Verfügung, welche verwendet werden können.

Um inubit schließlich per Docker oder Kubernetes bereitzustellen, muss die Docker Compose bzw. die Kubernetes Konfiguration erstellt werden. Da die inubit-Lizenzierung auf der Media Access Control (MAC) [156] basiert [147], muss zudem

eine feste MAC-Adresse vergeben werden. Für Kubernetes muss dafür ein Netzwerkplugin genutzt werden, welche diese Funktionalität unterstützt, wie die bereits in Abschnitt 3.1.1.5 erwähnte Lösung Calico [157]. Die zuvor aus dem Container kopierten inubit-Konfigurationsdateien müssen angepasst und in den inubit-Container/Pod eingebunden werden. Abschließend muss zunächst der Datenbankserver bereitgestellt und eingerichtet werden, bevor schließlich der inubit-Container gestartet werden kann.

Anzumerken sei an dieser Stelle, dass die Virtimo AG nicht nur kein offizielles Container Image bereitstellt, sondern laut Dokumentation den Betrieb in einem Container nicht explizit unterstützt [147], sodass die Virtimo AG den Support für Fehler, die bei der Installation oder während des Betriebes auftreten, versagen könnte.



Installation Inubit	Aufwand	Spannweite
VM	17 h	± 2 h
Docker	13.75 h	± 2.25 h
Kubernetes	14.25 h	± 2.25 h

Abbildung 22: Aufwand für das initiale Bereitstellen der inubit BPM

Die in Abbildung 22 dargestellte Auswertung zeigt, dass die Bereitstellung mittels Docker und Kubernetes erneut weniger Zeit benötigt als mittels VM. Der Grund dafür liegt insbesondere darin, dass die Bereitstellung und Konfiguration des Datenbankservers über Docker und Kubernetes deutlich weniger Aufwand benötigt als bei der VM. Der Aufwand für Docker/Kubernetes ist im Vergleich zur Bereitstellung der Webapplikation u. a. deutlich höher, weil kein offizielles inubit Container Image existiert.

3.3.3 Softwareupdates

Neben der initialen Installation und Konfiguration verursacht das Installieren von Softwareupdates einen Aufwand beim Betrieb des IT-Services, welcher anschließend betrachtet werden soll. Da die Services, u. a. aus Sicherheitsgründen, regelmäßig mit Softwareupdates versorgt werden müssen, sind diese Aufwände stets wiederkehrend. Mehraufwände bzw. Zeiteinsparungen summieren sich somit über die gesamte Lebenszeit der Service auf und haben dadurch einen besonders großen Einfluss auf den Gesamtaufwand.

In produktiven Umgebungen werden Softwareupdates in der GISA laut Anhang W meist im Rahmen eines der vier sogenannten Wartungswochenenden durchgeführt. Ein Wartungswochenende ist ein Wochenende, an dem für eine möglichst große Anzahl an IT-Servicen innerhalb eines möglichst kleinen Zeitraums gleichzeitig Updates installiert und ggf. weitere Wartungsaufgaben durchgeführt werden. Ziel ist es, dadurch die Gesamtausfallzeit zu minimieren und die Zeiträume der Nichtverfügbarkeit für alle Stakeholder planbar zu machen. Gründe, ein Update außerhalb eines Wartungswochenendes zu installieren, sind z. B. wichtige neue Funktionen oder das Schließen akuter Sicherheitslücken.

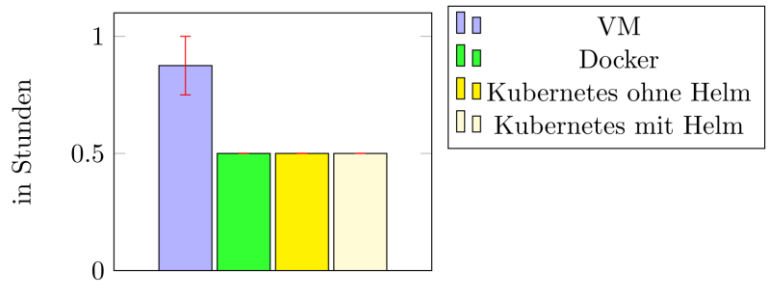
3.3.3.1 Angular-basierte Webapplikation

Je nachdem, wie aktiv eine Webapplikation weiterentwickelt wird, müssen monatlich oder wöchentlich neue Versionen bereitgestellt werden. Insbesondere in Test- bzw. Entwicklungsumgebungen werden die Applikationen teilweise sogar mehrmals wöchentlich/täglich aktualisiert. Jede Angular-basierte Webapplikation sollte jedoch auch ohne aktive Weiterentwicklung, mindestens zweimal jährlich auf die aktuelle Angular Hauptversion aktualisiert werden, welche alle sechs Monate erscheinen [158].

Um eine Angular-basierte Webapplikation zu aktualisieren, muss grundsätzlich ein neues Angular Build erstellt und auf den Server bzw. in das Image kopiert werden. Sollte die nginx-Konfiguration geändert wurden sein, muss zudem der nginx-Server neu gestartet werden. Zusätzlich dazu sollte z. B. im Rahmen des Wartungswochenendes das zugrunde liegende Betriebssystem und der nginx-Server aktualisiert werden.

Für das Aktualisieren der über eine VM bereitgestellten Webapplikation, muss die bei der initialen Installation erstellte Jenkins Pipeline erneut ausgeführt werden, um das neue Build zu erstellen und auf die VM zu kopieren. Um den nginx-Server neu zu starten, muss sich ein Entwickler per Secure Shell (SSH) [159] auf die VM verbinden. Für die Aktualisierung des Betriebssystems und des Nginx-Servers wird der Paketmanager *apt* [160] genutzt.

Bei der Bereitstellung über Docker und Kubernetes muss ggf. die jeweils zugrunde liegende Dockerfile angepasst werden, um die neuste Version des nginx-Images zu nutzen. Anschließend muss jeweils die Jenkins Pipline erneut ausgeführt werden, um eine neue Version des zum Service gehörenden Container Images zu erstellen. Sollte in der Docker/Kubernetes Konfiguration die Version des Images über den Image Tag definiert wurden sein, muss dieser angepasst werden. Abschließend muss der zum IT-Service gehörende Container/Pod neu gestartet werden, sodass das neu erstelle Image genutzt wird.



Softwareupdates Angular	Aufwand	Spannweite
VM	0.875 h	± 0.125 h
Docker	0.5 h	± 0 h
Kubernetes ohne Helm	0.5 h	± 0 h
Kubernetes mit Helm	0.5 h	± 0 h

Abbildung 23: Aufwand für das Durchführen von Softwareupdates bei einer Angular-basierte Webapplikation

Die Auswertung der Umfrage in Abbildung 23 zeigt, dass das Durchführen von Softwareupdates bei der Bereitstellung mittels VM mehr Aufwand benötigt als bei Docker und Kubernetes. Die Ursache für den geringeren Aufwand liegt sowohl in der schnelleren Aktualisierung der zugrunde liegenden Software und des Betriebssystems durch den Wechsel des Base-Images, als auch in der Tatsache, dass der nginx-Server in der VM manuell neu gestartet werden, wenn sich dessen Konfiguration ändert, während dies bei Docker/Kubernetes durch den Neustart des Containers automatisch geschieht.

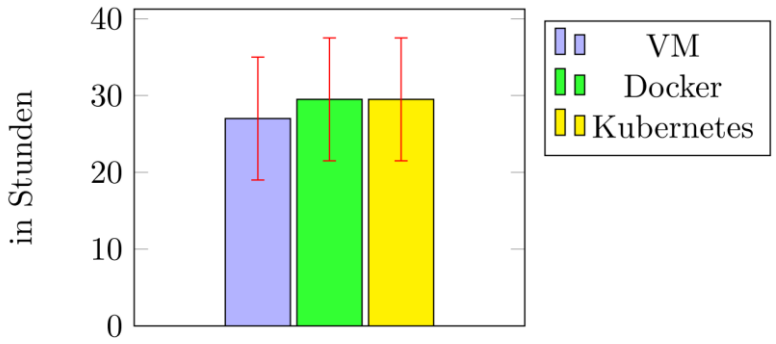
3.3.3.2 inubit BPM

Laut der Umfrage geht einem inubit-Update eine Evaluation voraus, um abzuschätzen, ob die Durchführung des Updates sinnvoll ist. Sollte dies zutreffen, wird die momentane Installation zunächst kopiert und das Update innerhalb der Kopie testweise durchgeführt und dokumentiert. Bevor das Update auf dem eigentlichen Server installiert werden kann, muss zudem eine Abstimmung mit allen Nutzern erfolgen. Um das Update durchzuführen, muss der inubit-Server zunächst heruntergefahren werden. Anschließend muss ein Patch-Installer ausgeführt werden, welcher die existierende Installation aktualisiert. Zusätzlich sind ggf. weitere Patch-Schritte, wie das Aktualisieren der Java Version oder das Anpassen von Konfigurationen vonnöten [147]. Anschiebend kann der Server wieder gestartet werden.

Bei der Installation in einer VM muss der Patch-Installer zunächst in die VM kopiert werden. Anschließend wird der Server heruntergefahren und der Patsch-Installer ausgeführt. Schließlich werden die zusätzlichen Patch-Schritte durchgeführt und der Server gestartet.

Bei der Bereitstellung über Docker und Kubernetes ist es, anders als bei der Angular Webanwendungen, nicht ausreichend lediglich ein neues Container Image zu erstellen, da der Patch-Installer während der Ausführung z. B. Zugriff auf die Datenbank braucht,

was während der Image-Erstellung nicht gegeben ist. Des Weiteren benötigt der Patch-Installer erneut eine Antwortdatei, welche auf dieselbe Weise, wie bei der Installation in Abschnitt 3.3.2.2 generiert werden muss. Anschließend muss der Patch-Installer zunächst in den laufenden Container/Pod kopiert und dort ausgeführt werden. Für die Erstellung des neues Image mit dem Installer der aktuellen Version, wird ebenfalls eine aktuelle Antwortdatei benötigt. Abschließend müssen die ggf. notwendigen zusätzlichen Patch-Schritte ausgeführt und der Container/Pod neu gestartet werden.



Softwareupdates Inubit	Aufwand	Spannweite
VM	27 h	± 8 h
Docker	29.5 h	± 8 h
Kubernetes	29.5 h	± 8 h

Abbildung 24: Aufwand für das Durchführen von Softwareupdates bei der inubit BPM

Wie in der Auswertung in Abbildung 24 zu sehen ist, ist das Durchführen von Softwareupdates bei der Bereitstellung mittels Docker und Kubernetes aufwändiger als bei der VM (+2,5 h). Grund dafür ist zum einen die Funktionsweise des Patch-Installer, welche es erfordert diesen innerhalb des Containers/Pod auszuführen, bevor das neu generierte Image genutzt werden kann und zum anderen die Tatsache, dass erneut Antwortdateien generiert werden müssen. Auffällig ist zudem eine sehr große Spannweite der Aufwände. Die Ursachen liegen jedoch insbesondere im unterschiedlichen Umfang der Updates und fällt somit bei der Betrachtung eines konkreten Updates für alle Virtualisierungstechnologien gleich aus.

3.3.4 Migration zwischen den Virtualisierungstechnologien

Neben den zukünftigen neuen IT-Services werden im DMC-Umfeld bereits eine Vielzahl an existierende Services betrieben, deren zurzeit verwendete Virtualisierungstechnologie ebenfalls überprüft werden können soll. Anderes als bei einer initialen Installation und Konfiguration eines neuen IT-Services, umfasst eine Migration auf eine andere Virtualisierungstechnologie aber auf der einen Seite zusätzliche Schritte, wie eine möglicherweise notwendige Datenmigration, auf der anderen Seite können aber auch Arbeitsschritte entfallen oder sich vereinfachen, weil z. B. bestehenden

Konfigurationsdateien wiederverwendet oder ggf. nur angepasst werden müssen. Daher wurde anschließend der Aufwand für verschiedene Migrationen geschätzt.

3.3.4.1 Migration einer Angular-basierten Webapplikation von Docker zu Kubernetes

Zunächst soll die Migration eines IT-Services vom Docker-Host in den Kubernetes Cluster betrachtet werden. Da auf dem Docker Host bereits eine große Anzahl an Angular basierten Webanwendungen betrieben werden, wird diese Migration anhand einer solchen geschätzt. Wie bei der initialen Installation und Konfiguration wird erneut zwischen Kubernetes mit und ohne Helm unterschieden.

Um die Migration ohne die Nutzung eines Helm Charts durchzuführen, muss die Docker Compose Konfigurationsdatei in die verschiedenen Konfigurationsdateien der Kubernetes Ressourcen transformiert werden. Da sich die Konfiguration der Angular Anwendungen untereinander nur minimal unterscheidet, kann wieder auf eine bereits existierende Konfiguration zurückgegriffen werden, um diese nur noch anzupassen. Bei komplexeren IT-Services bzw. wen noch keine Konfiguration existiert, die als Vorlage genutzt werden kann, kann das eingangs erwähnte Software-Werkzeug Kompose genutzt werden. Sollte ein Helm Chat für die Bereitstellung verwendet werden, müssen die Helm Values entsprechend von der Docker Compose Konfiguration übertragen und ggf. angepasst werden. Abschließend muss der Docker Container vom Docker Host gelöscht und abgemeldet werden.

Für die Migration mit bzw. ohne Helm wurde jeweils ein Aufwand von 1.5 h geschätzt.

3.3.4.2 Migration der inubit BPM von einer VM zu Kubernetes

Da u. a. die inubit BPM zurzeit in einer VM betrieben wird, soll auch die Migration von einer VM in den Kubernetes Cluster am Beispiel inubit BPM betrachtet werden.

Für die Migration muss zunächst ein neuer inubit-Server im Kubernetes Cluster nach dem in Abschnitt 3.3.2.2 beschriebenen Vorgehen erstellt werden, der Unterschied besteht lediglich darin, dass die Konfigurationsdateien z. B. für die Datenbankverbindung in einem späteren Schritt von der existierenden VM kopiert und ggf. angepasst werden müssen. Bei der Anforderung- und Abhängigkeitsanalyse können die meisten Erkenntnisse von der VM übernommen werden, allerdings sollte jeweils nochmal eine Kubernetes spezifische Überprüfung erfolgen. Des Weiteren muss untersucht werden, wie sich die Migration auf externe Systeme auswirken wird, die auf die inubit BPM zugreifen.

Bevor die Migration durchgeführt wird, wird zunächst ein Migrationsplan erstellt. Dafür wird die Migration testweise durchgeführt. Sollten Fehler auftreten, werden diese analysiert und die Testmigration wiederholt. Sollte für einen Fehler keine Lösung

gefunden werden können, muss auf den Virtimo Support zurückgegriffen werden. Zudem ist vor der eigentlichen Migration erneut eine Abstimmung mit allen Nutzern des inubit BPM-Servers notwendig.

Um die Daten von der VM in die Kubernetes Umgebung zu übertragen, müssen die Daten auf der VM zunächst aufs wesentliche reduziert und anschließend gesichert werden. Ein Skript für die Sicherung ist bereits Teil der inubit Installation [147]. Die Erstellte Sicherung muss anschließend in den Kubernetes Pod kopiert und durch das offizielle Migrationsskript wiederhergestellt werden. Zusätzlich zu der beschriebenen automatischen Migration mittels Skripts müssen einige Daten und Einstellungen manuell übertragen werden, dazu gehören z. B. die Konfigurationsdateien der Datenbankverbindungen und zusätzlich verwendete Java-Bibliotheken [x]. Nach der vollständigen Migration kann der neue inubit-Server gestartet und der sogenannte Wartungsmodus deaktiviert werden, welcher die Ausführung der Workflows für die Zeit der Migration verhindert hat. Abschließend müssen diverse Funktionstests durchgeführt und die VM zurückgebaut werden. Dazu gehört das Beantragen der Löschung der VM, sowie das Entfernen der alten Firewall und Proxy Einstellungen.

Aus der Umfrage geht ein geschätzter Migrationsaufwand von 39 h ± 4.5 h hervor. Die Migration ist somit noch einmal deutlich aufwendiger als die initiale Installation.

3.3.4.3 Migration der inubit BPM von Kubernetes zu einer VM

Sollte inubit über Kubernetes bereitgestellt werden und sich Kubernetes, z. B. durch die in Abschnitt 3.3.3.2 angesprochenen Komplikationen als ungeeignet erweisen, muss eine Migration in eine VM erfolgen. Der Aufwand für eine Migration von Kubernetes in eine VM soll daher beispielhaft an der inubit BPM geschätzt werden.

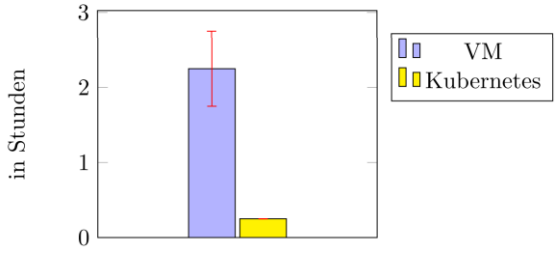
Das Vorgehen bei der Migration ähnelt sehr dem Vorgehen bei der Migration von einer VM zu Kubernetes. Der Unterschied liegt darin, dass der neue inubit-Server in einer neuen VM aufgebaut wird und abschließend das Kubernetes mit den entsprechenden Konfigurationen gelöscht werden muss.

Der Aufwand wurde in der Umfrage auf 41 h ± 4 h geschätzt und somit noch einmal leicht höher als die Migration von der VM in den Kubernetes Cluster,

3.3.5 Hochverfügbarkeit

Unter Hochverfügbarkeit versteht man, wie in Abschnitt 3.1.2 bereits erläutert, die Möglichkeit eines IT-Services trotz Ausfall einer virtuellen Instanz oder eines physikalischen Servers weiterhin betriebsbereit zu sein. Während das grundsätzliche Vorgehen zur Erziehung dieser Hochverfügbarkeit bereits in Abschnitt 3.1.2 erläutert wurde, soll nun der Aufwand dafür am Beispiel der Angular Webapplikation geschätzt werden.

Um eine Angular Anwendung, die in einer VM installierten wurde, hochverfügbar bereitzustellen, muss zunächst eine Kopie der bereits existierenden VM erstellt werden. Anschließend muss die Jenkins Pipline angepasst und erneut ausgeführt werden. Abschließend muss der Reverse Proxy so konfiguriert werden, dass er als Lastverteiler fungiert. Für Kubernetes muss hingegen lediglich die Konfigurationsdatei des Deployments angepasst und angewendet werden. Für die Bereitstellung einer hochverfügbaren Anwendung mittels Docker, ist dessen Cluster-Lösung *Docker Swarm* erforderlich. Wie aber bereits in Abschnitt 3.1.2.2 erläutert, wird diese Lösung innerhalb dieser Arbeit nicht beachtet, sodass der Aufwand nicht geschätzt wurde.



Hochverfügbarkeit	Aufwand	Spannweite
VM	2.25 h	± 0.5 h
Kubernetes	0.25 h	± 0 h

Abbildung 25: Aufwand für das Erreichen der Hochverfügbarkeit

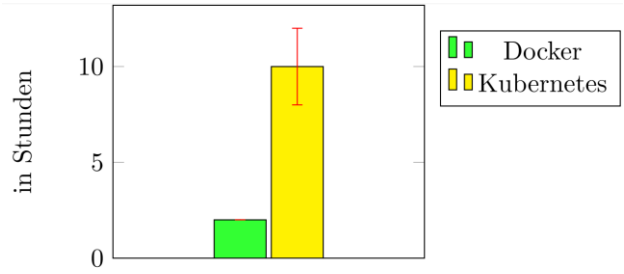
Die Auswertung in Abbildung 25 zeigt, dass die Hochverfügbarkeit mithilfe von Kubernetes aufgrund der nativen Unterstützung mit deutlich weniger Aufwand erreicht werden kann als mithilfe einer VM.

3.3.6 Wartung und Verwaltung der Infrastruktur

Während die Betreuung und Verwaltung der VM Infrastruktur im Wesentlichen von den dafür zuständigen Fachbereichen übernommen wird, muss sowohl der Dockerhost als auch der Kubernetes Cluster vom DMC-Umfeld selbst getragen werden. Der dadurch entstehende zusätzliche Aufwand soll ebenfalls betrachtet werden. Dabei soll jeweils auf den allgemeinen jährlichen Wartungsaufwand, sowie das Bereitstellen zusätzlicher Systemressourcen, wie Arbeitsspeicher oder CPUs, eingegangen werden.

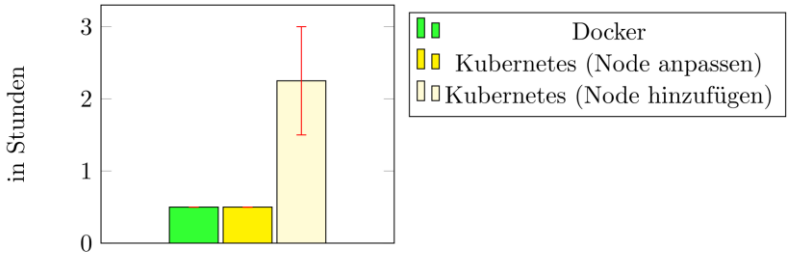
Der jährliche Wartungsaufwand verteilt sich im Normalfall auf die vier Wartungswochenenden, an denen das Betriebssystem und ggf. weitere auf dem System verwendete Software aktualisiert werden. Während der Docker Host allerdings lediglich aus einer einzelne VM und der Docker Engine besteht, umfasst der Kubernetes Cluster mehreren VMs, welche alle aktualisiert werden müssen, und besitzt des Weiteren „zusätzliche Komponenten wie Ingress-Controller, Storage-Provisioner, Monitoring, etc. [die im Verantwortungsbereich des DMC-Umfelds] liegen“ [Anhang V].

Da es sich bei dem Docker Host und den Kubernetes Nodes ebenfalls um VMs handelt, muss für die Bereitstellung zusätzlicher Systemressourcen bei allen drei Virtualisierungstechnologien die Konfiguration der zugrunde liegenden VM angepasst werden. Bei Kubernetes können zusätzliche Systemressourcen allerdings nicht nur über die Anpassung einer bereits existierenden Node bereitgestellt werden, sondern auch eine neue Node in dem Cluster aufgenommen werden. Dafür muss eine neue VM bereitgestellt und in den Kubernetes Cluster integriert werden.



Wartungsaufwand	Aufwand	Spannweite
Docker	2 h	± 0 h
Kubernetes	10 h	± 2 h

Abbildung 26: jährlicher Wartungsaufwand



Ressourcenerweiterung	Aufwand	Spannweite
Docker	0.5 h	± 0 h
Kubernetes (Node erweitern)	0.5 h	± 0 h
Kubernetes (Node hinzufügen)	2.25 h	± 0.75 h

Abbildung 27: Aufwand für die Bereitstellung zusätzlicher Systemressourcen

Wie in Abbildung 27 zu sehen ist, ergeben sich aus der Umfrage zunächst bei Docker, Kubernetes und VM identische Werte für die Anpassung der Systemressourcen. Während der Docker Host allerdings pro Wartungswochenende lediglich einen Aufwand von 0.5 h verursacht, weist Kubernetes mit 2 – 3 h pro Wartungswochenende einen deutlich höheren jährlichen Verwaltungsaufwand auf. Des Weiteren zeigt sich, dass das Erstellen und Einbinden einer neuen Node deutlich zeitaufwendiger ist als das Anpassen einer existierenden Node (0.5 h vs. 2.25 h). Zusätzlich erhöht die neue Node den Wartungsaufwand, da es sich z. B. um eine neue VM handelt, die am Wartungswochenenden aktualisiert werden muss. Im Gegenzug erhöht die Node jedoch die Ausfallsicherheit der IT-Services im Cluster, da Kubernetes eine weitere Node zur

Verfügung hat, auf der die Workloads, im Falle eines Node-Ausfalls, neu gestartet werden können.

Bei der Auswertung muss allerdings beachtet werden, dass in einer VM im Normalfall nur ein einzelner IT-Service läuft, während auf einem Docker Host bzw. im Kubernetes Cluster eine Vielzahl an Services betrieben werden können, auf welche sich der (zusätzliche) Aufwand verteilt. So entfallen beispielsweise für den jährlichen Verwaltungsaufwand bei 25 auf dem Docker Host bzw. im Kubernetes Cluster betriebenen IT-Servicen je pro Service 0.08 h (5 min.) bei Docker und ~0.4 h bei Kubernetes.

3.3.7 Fazit des Aspekts Aufwand

Zusammenfassen kann beim Aspekt Aufwand festgestellt werden, dass die initiale Bereitstellung und das Installieren von Softwareupdates mittels Kubernetes und Docker einen geringeren Aufwand benötigt als bei der VM, solange ein offizielles Container Image zur Verfügung gestellt wird, wie bei nginx und MariaDB. Sollte dies nicht der Fall sein, kann eine Bereitstellung über eine VM jedoch zeitsparender sein, wie es bei inubit der Fall ist. Grund dafür sind zum einen ein ggf. komplexer Image-Erstellungsprozess, sowie eine allgemein schlechte Unterstützung der Containerisierung, die auch bei Softwareupdates für Komplikationen sorgen kann. So waren die Container-basierten Lösungen bei der Installation von inubit hinsichtlich des Aufwandes noch im Vorteil, allerdings ist der wiederkehrende Aufwand für das Durchführen von Softwareupdates bei der VM deutlich kleiner.

Der Aufwand für die Bereitstellung mittels Docker war vergleichbar mit der Bereitstellung über Kubernetes. Bei neu zu erstellenden Konfigurationen ist die Bereitstellung über Kubernetes etwas zeitaufwendiger, während die Konfiguration des Reverse Proxys für einen etwas größeren Aufwand seitens Docker gesorgt hat, wenn bereits Konfigurationen als Vorlage zur Verfügung stehen. Sollten keine Konfigurationen existieren und ein Helm-Chart aus vertrauenswürdiger Quelle zur Verfügung stehen, hat sich zudem die Nutzung dieser als sinnvoll dargestellt.

Bei der Migration wurden drei verschiedene Szenarien beispielhaft an realistischen Beispielen betrachtet. Während die Migration der Angular Applikation nur einen geringen Aufwand benötigt hat, war die Migration von inubit deutlich aufwendiger. Das liegt zum einen an der komplexeren Migration der Daten und Einstellungen und zum anderen an der aufwendigen Bereitstellung im Kubernetes Cluster. Während sich das grundsätzliche Vorgehen der Migrationen von neue Instanz aufbauen, Daten übertragen, altes System zurückbauen, auf anderer IT-Services übertragen lässt, ist der Aufwand z. B. für die Migration der Daten von Service zu Service unterschiedlich. Die Umfrage zeigt aber eindeutig, dass eine Migration, zumindest bei komplexen IT-Services wie die

inubit BPM, mit einem hohen Aufwand verbunden ist und daher nur in Betracht gezogen werden sollte, wenn sich eine Anforderung nicht oder nur mit hohem Aufwand mit der jetzigen Virtualisierungstechnologie umsetzen lässt.

Es wurde zudem aufgezeigt, dass das Erreichen der Hochverfügbarkeit mittels Kubernetes im Vergleich zur VM mit deutlich weniger Aufwand möglich ist und das der zusätzliche Aufwand durch die Wartung der Kubernetes/Docker Infrastruktur mit 5 min/0.5 h pro IT-Service pro Wartungswochenende im Vergleich zum Einsparpotenzial (Vgl. Abschnitt 3.3.2.1, Abschnitt 3.3.3.1 und Abschnitt 3.3.5) gering ausfällt.

4 Entscheidungshilfe

Um für einen IT-Service die am besten geeignete Virtualisierungstechnologie auswählen zu können, müssen alle untersuchten Aspekte beachtet werden, deren Gewichtung von IT-Service zu IT-Service unterschiedlich ist. Es ist daher nicht möglich, eine allgemeingültige Aussage über die ideale Virtualisierungstechnologie zu treffen. Stattdessen sollen anschließend, auf Basis der Ergebnisse des in dieser Arbeit durchgeführten Vergleichs, Entscheidungshilfen aufgestellt werden, welche es den Entwicklern und Service-Betreuern des DMC-Umfelds erleichtert, die richtige Entscheidung zu treffen. Aufgrund der unterschiedlichen Ausgangssituation soll die Bereitstellung neue IT-Services und die Migration von bereits bestehenden IT-Services getrennt voneinander betrachtet werden.

4.1 Aufstellen der Entscheidungshilfe

Bei der Entscheidungshilfe für den Betrieb eines neuen IT-Services sein zunächst festzuhalten, dass für Linux lediglich zwischen Kubernetes und VM unterschieden wird, während bei Windows-basierten Services die Entscheidung zwischen VM und Docker fällt. Grund dafür ist, dass im Rahmen dieser Arbeit kein überzeugender Grund gefunden werden konnte, den Docker Host dem parallel existierenden Kubernetes Cluster für Linux-basierte Services vorzuziehen. Zwar hat Kubernetes einen leicht höheren Ressourcenverbrauch, Wartungsaufwand für die Infrastruktur und, in einigen Fällen, Aufwand für die Bereitstellung, aber dafür bietet es eine deutlich bessere Ausfallsicherung durch die Möglichkeit der hochverfügbaren Bereitstellung der IT-Services. Auf der anderen Seite sollten Windows-basierte Services, aus den in Abschnitt 3.1.1.4 ausführlicher beschriebenen Sicherheitsgründen, lediglich durch Hyper-V basierten Containern betrieben werden. Diese werden allerdings, wie in Abschnitt 2.5 und Abschnitt 2.6 bereits erwähnt, lediglich durch Docker unterstützt.

Die Entscheidungshilfe für neue IT-Service wurde in Abbildung 28 schematisch dargestellt. Sollte ein neuer IT-Service betrieben werden, muss zunächst entschieden werden, wie gut die Containerisierung von der zum Service gehörigen Software unterstützt wird. Dazu gehört z. B. die offizielle Unterstützung durch den Softwarehersteller, insbesondere bei Software mit Supportverträgen, und das Vorhandensein von Container-Images, die im besten Fall durch den Softwarehersteller bereitgestellt werden, bzw. die Möglichkeit mit geringem Aufwand eigene Images zu erstellen. Sollte ein IT-Service deutliche Mängel bei der Unterstützung vom Container-basierten Betrieb aufweisen, sollte der Betrieb in eine VM vorgezogen werden. Ansonsten sollten die Container-basierten Lösungen vorgezogen werden, da die Ergebnisse aus Abschnitt 3.3

zeigen, dass der Aufwand für den Betrieb eines IT-Services im Falle einer Unterstützung der Containerisierung deutlich geringer ausfällt.

Anschließend muss der Sicherheitsbedarf des IT-Service bewertet werden. Wie in Abschnitt 3.1.1 beschrieben, bieten sowohl die Hardware- als auch die Containervirtualisierung einen ausreichenden Schutz für die meisten IT-Services und deren Daten, sofern die Sicherheitsrichtlinien ordnungsgemäß definiert, aktualisiert und umgesetzt werden. Für die zusätzliche Härtung können zudem Seccomp- oder AppArmor/SELinux-Profil eingesetzt werden. Aufgrund der zusätzlichen Angriffsvektoren bei der Containervirtualisierung (vgl. Abschnitt 3.1.1.4) sollte jedoch bei IT-Services, in denen Daten mit besonders hohem Schutzbedarf verarbeitet werden, die zusätzliche Sicherheit der VM in Anspruch genommen werden. Im Bezug zur Disaster Recovery konnte hingegen bei VMs, Docker und Kubernetes, abgegeben von Zuständigkeitsbereich, kein signifikanter Unterschied festgestellt werden (vgl. Abschnitt 3.1.2). Bei der Ausfallsicherung lässt sich mit einer VM und Kubernetes eine Hochverfügbarkeit erreichen (vgl. Abschnitt 3.1.3), allerdings ist der dafür benötigte Aufwand bei Kubernetes deutlich geringer (vgl. Abschnitt 3.3.5).

Ein weiterer Aspekt, der berücksichtigt werden sollte, ist die in Abschnitt 3.2 untersuchte Performance und Ressourcennutzung. Die Containerisierung weist bei den meisten der durchgeführten Performancetests keine oder nur geringe Performanceeinbußen auf, ausgenommen der Netzwerk–Performancetests. Sollte ein IT-Service demnach besonders hohe Anforderungen an die Performance haben, insbesondere in Bezug auf die Netzwerk-Performance, kann der Einsatz einer VM sinnvoller bzw. notwendig sein. Dabei ist allerdings zu beachten, dass den Performanceeinbußen bei einigen Performanceparametern durch das Reservieren von zusätzlichen Ressourcen für den Service (vgl. Abschnitt 2.6.1.5), wie CPU-Kernen, entgegengewirkt werden kann. Sollte die Performance für den IT-Service eine hohe Bedeutung haben, ist eine individuelle Analyse der Performanceanforderungen des IT-Services nötig. Dabei muss auch eine Abwägung zwischen der durch die Verwendung einer VM gewonnenen Performance und dem dadurch entstehenden größeren Aufwand, z. B. für Softwareupdates oder die Hochverfügbarkeit, durchgeführt werden.

Schlussendlich muss bei Windows-basierten IT-Services beachtet werden, dass zurzeit noch kein Windows Docker-Host existiert. Der Aufwand, der für den Aufbau des Windows Docker-Hosts und das Erlangen des Fachwissens benötigt wird, wurde im Rahmen dieser Arbeit nicht geschätzt. Basierend auf dem durchgeführten Vergleich ist allerdings davon auszugehen, dass sich dieser Aufwand bei einer sehr kleinen Anzahl an Windows IT-Services nicht rentiert. Solange noch kein Windows Docker Host existiert, müssen Windows-basierte Services weiterhin in VMs betrieben werden. Sollte eine größere Anzahl an potenziellen Windows-basierten Docker IT-Services existieren oder

in naher Zukunft absehbar sein, sollte der Aufbau eines Windows Dockers Host evaluiert werden. Zu beachten ist zudem, dass, wie Eingangs bereits erwähnt, Docker keine Möglichkeit der hochverfügbare Bereitstellung von IT-Service bietet, sofern nicht die Einführung von Docker Swarm beschlossen wird. Windows-basierte Services, die eine hochverfügbare Bereitstellung voraussetzen, müssen somit weiterhin über eine VM bereitgestellt werden.

Bei der Bereitstellung über Kubernetes sollten die in Abschnitt 3.3 erlangten Erkenntnisse bezüglich Helm Charts beachtet werten. Helm Charts können den Aufwand für die Bereitstellung von IT-Services senken, allerdings ist der Einsatz nur sinnvoll, wenn nicht stattdessen Konfigurationsdateien eines artgleichen IT-Services als Vorlage genutzt werden können und man davon ausgehen kann, dass die Anforderungen an den IT-Service auch in Zukunft nicht die Möglichkeiten des jeweiligen Helm Charts überschreiten.

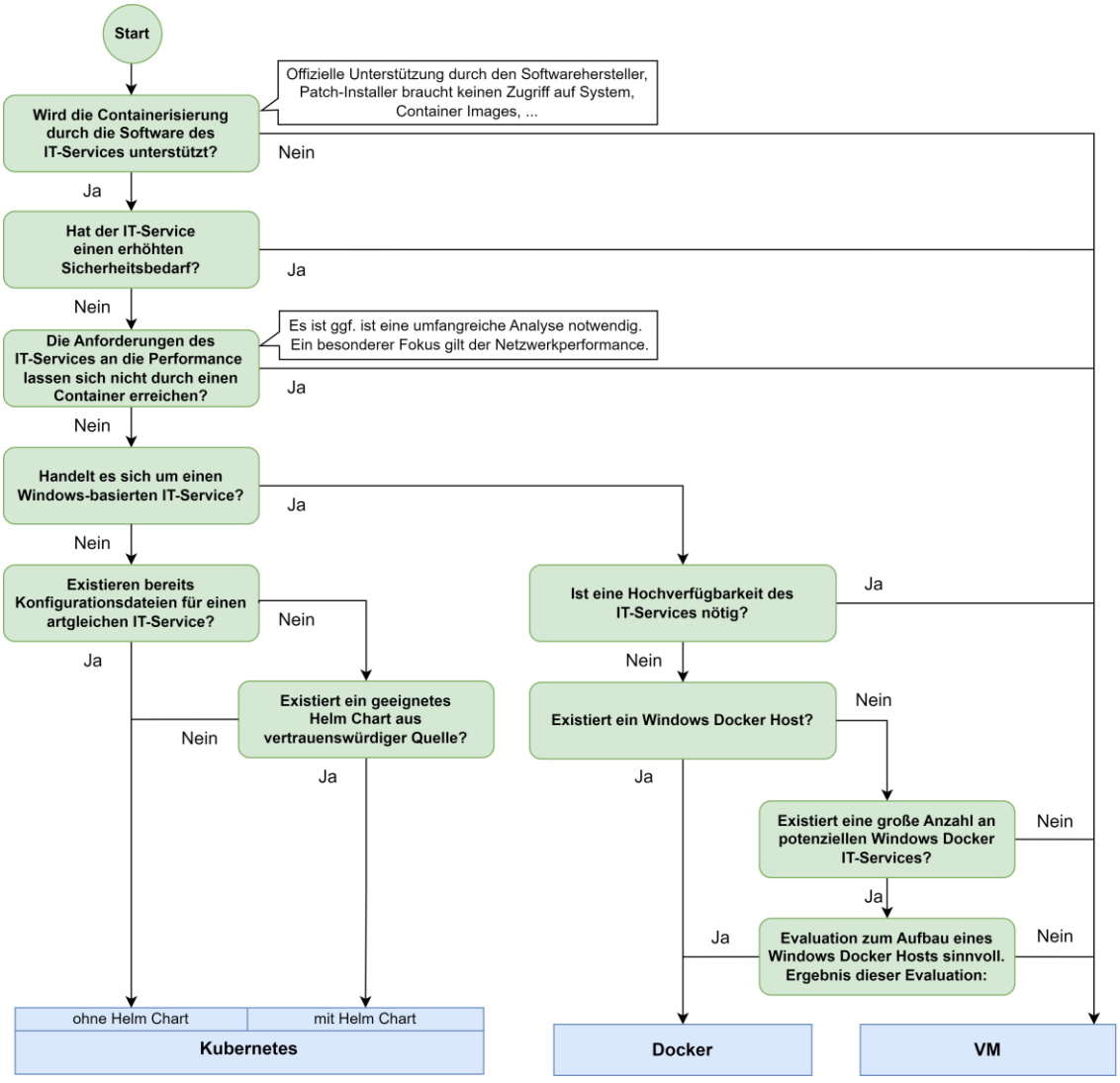


Abbildung 28: Entscheidungshilfe für Wahl der Virtualisierungstechnologie für den Betrieb von IT-Services

Bei der Migration sollte zwischen Docker zu Kubernetes, Kubernetes zu VM und VM zu Kubernetes unterschieden werden. Die Migration von Docker zu Kubernetes ist insbesondere für Produktivsysteme, aber auch für Test-/Entwicklungssysteme, die für die tägliche Entwicklung zwingend benötigt werden, aufgrund der bereits erläuterten Vorteile im Bereich Ausfallsicherung zu empfehlen. Bei einer Migration aller Services von dem Docker-Host könnte dieser abgebaut werden und der jährliche Wartungsaufwand könnte eingespart werden. Dafür würde allerdings ein einmaliger Aufwand für die Migration der Services und eine ggf. benötigte Anpassung der Kubernetes zur Verfügung stehenden Systemressourcen anfallen. Inwiefern auch die Migration der Test- und Entwicklungssystemen von niedrigerer Priorität sinnvoll ist, ist somit stark von der Anzahl und Komplexität der Services dieser Art abhängig.

Für die Migration von Kubernetes (bzw. Docker) zur VM ist die Entscheidungshilfe für neue Services aus Abbildung 28 ebenfalls aussagekräftig. D. h. sollten sich z. B. die Sicherheitsanforderungen der Daten eines Service im Laufe seines Bestehens so erhöhen, dass man sich bei einer initialen Installation nun für eine VM entscheiden sollte, sollte auch eine Migration vorgenommen werden.

Für die Migration von einer VM zu Kubernetes muss die Entscheidungshilfe allerdings erweitert werden, da die potenziellen Vorteile von Kubernetes und Docker in der Ressourceneinsparung und dem geringeren Aufwand liegen, bzw. in dem Kostenvorteil, der sich dadurch ergibt. Dieser Kostenvorteil muss dem für die Migration benötigten Aufwand entgegengesetzt werden. Nur wenn Kubernetes/Docker nach der Entscheidungshilfe in Abbildung 28 geeignet ist und die durch die Migration erreichbaren Einsparungen größer sind als die durch den Aufwand für die Migration verursachten Kosten, sollte eine Migration erfolgen.

4.2 Beispielhafte Anwendung der Entscheidungshilfe

Um die Funktionsweise der Entscheidungshilfe zu demonstrieren, wird diese im Folgenden exemplarisch auf den Betrieb einer Angular-basierte Webapplikation und die inubit BPM angewendet.

Für die Angular Webapplikation hat die Aufwandsbetrachtung gezeigt, dass die Containerisierung sehr gut unterstützt wird. So kann das offizielle nginx Container Image als Basis genutzt werden, um durch Jenkins vollautomatischer ein Service-spezifisches Image erzeugt werden kann und für den Updateprozess genügt eine erneute Generierung des Images und ein Pod Neustart. Die Webapplikation haben gewöhnlicherweise auch keinen erhöhten Sicherheitsbedarf, da die Daten des dazugehörigen IT-Services nicht vom Webserver, sondern von externen Systemen wie die inubit BPM verarbeitet werden. Im Container liegen somit nur die Webseite, die beim Aufruf der entsprechenden URL an den Client gesendet wird. Bei der Betrachtung der Perfor-

manceansprüche lässt sich festhalten, dass insbesondere die Netzwerkperformance für eine Webapplikation zwar grundsätzlich wichtig ist, bis jetzt aber keine IT-Services bereitgestellt wurden, die tatsächlich diese besonders hohen Anforderungen z. B. an die Netzwerkbandbreite des Webservers oder einen anderen Performanceparameter haben. Da es sich bei dem verwendeten nginx-Image um ein Linux basiertes Container Image handelt und bereits Kubernetes-Konfigurationen für artgleiche IT-Services existieren, sollte die Angular Webapplikation somit über den Kubernetes Cluster ohne die Verwendung eines Helm Charts bereitgestellt werden.

Bei der inubit BPM muss hingegen bereits bei der ersten Frage der Entscheidungshilfe festgestellt werden, dass dieser IT-Service, basierend auf den Erkenntnissen der Aufwandsschätzung, eher in einer VM betrieben werden sollte, da die Containerisierung nicht nur keine offizielle Unterstützung vom Softwarehersteller erhält, sondern auch der Aufwand für Softwareupdates bei der containerisierten Bereitstellung größer ist als bei der VM. Aber auch der Sicherheitsbedarf der inubit BPM ist deutlich höher als z. B. der einer Angular Applikation, da inubit eine große Bandbreite an Daten verarbeitet, darunter auch hochsensible Informationen, und direkten Zugriff auf viele weitere externe Systeme besitzt. Des Weiteren wird eine inubit Instanz von einer großen Anzahl an IT-Services genutzt wird, die bei einem Sicherheitsvorfall alle potenziell betroffen wären, sodass sich die Empfehlung, inubit in einer VM zu betreiben, bestärkt.

Die Angular-basierte Webapplikationen, die zurzeit auf dem Docker Host betrieben werden, sollten, wie bereits beim Aufstellen der Entscheidungshilfe erläutert, auf den Kubernetes Cluster migriert werden, da Kubernetes eine deutliche Verbesserung der Ausfallsicherheit bei geringem Migrationsaufwand bietet.

Da der Betrieb der inubit BPM durch Kubernetes bereits als ungeeignet bestimmt wurde, ist eine detaillierte Betrachtung der Sinnhaftigkeit einer Migration der bestehenden VM Instanzen nicht nötig. Es sei allerdings festzuhalten, dass der in Abschnitt 3.3.4.2 geschätzte hohe Migrationsaufwand diese Feststellung erneut bestätigt. Für eine verbesserte Ausfallsicherheit bietet inubit zudem seine eigene Clusterunterstützung an [147].

5 Zusammenfassung und Ausblick

Zusammenfassend lässt sich zunächst sagen, dass die in der Zielsetzung definierten Ziele für diese Arbeit erreicht wurden. So wurden in Abschnitt 2 zunächst die zurzeit im DMC-Umfeld der GISA GmbH eingesetzten Virtualisierungstechnologien hinsichtlich ihrer Funktionsweise und Konfiguration abgegrenzt.

Anschließend wurden die Technologien in Abschnitt 3.1 hinsichtlich der IT-Sicherheitsaspekte Schutz vor Schadsoftware und Cyberattacken, Ausfallsicherung und Disaster Recovery untersucht. In Abschnitt 3.2 wurde der Ressourcenverbrauch im Leerlauf und der Infrastruktur verglichen, sowie die Performanceeinbußen durch die Containerisierung bei CPU, Datenträgern, Netzwerk und weiteren Kernel-Funktionen anhand mehrere Performancetests ermittelt. Schließlich wurde in Abschnitt 3.3 der Aufwand für das initiale Installieren eines IT-Services, sowie das Installieren von Softwareupdates, das Erreichen der Hochverfügbarkeit und die Migration zwischen den einzelnen Virtualisierungstechnologien ermittelt. Da bei Docker und Kubernetes durch die Betreuung und Verwaltung der jeweiligen Infrastruktur ebenfalls ein Aufwand entsteht, wurde dieser ebenfalls ermittelt. Für die Ermittlung der Aufwände wurde eine Umfrage erstellt.

Basierend auf den im Vergleich erarbeiteten Erkenntnissen wurde in Abschnitt 4.1 eine Entscheidungshilfe erstellt, welche in Abschnitt 4.2 auf eine Angular-basierte Webanwendung und die Business Process Management Software inubit BPM angewendet wurde. Aus der erstellten Entscheidungshilfe lässt sich ablesen, dass, solange die zugrunde liegende Software die Bereitstellung über Container unterstützt, für die Mehrheit der IT-Services die Containerisierung zu empfehlen ist, die Migration von VM zu Kubernetes/Docker aber dennoch aufgrund des zusätzlichen Migrations-aufwandes nicht zwingend sinnvoll ist. Dieses Ergebnis deckt sich mit einem von *451 Research* im Jahre 2019 veröffentlichen Report, in dem es anlässlich des fünften Jahrestages von Kubernetes heißt „It's hardly scientific but one way to think about this market - and the opportunity - is that while 90% of applications don't use containers, 95% of new applications do“ [161].

In Zukunft sollten alle IT-Services des DMC-Umfelds mithilfe der Entscheidungshilfe bewertet werden. Zusätzlich sollte in Erwägung gezogen werden, die Anzahl der Control Plane Nodes des DMC Kubernetes Cluster zu erhöhen, um so den Cluster resilient gegen einen Ausfall einer dieser Nodes zu machen (vgl. Abschnitt 3.1.2.3). Sollte in Zukunft eine Steigerung der Anzahl der Windows-basierten IT-Services abzusehen sein, sollte, wie bereits bei der Aufstellung der Entscheidungshilfe erwähnt,

der durchgeführte Vergleich mit einem Fokus auf Windows-Container wiederholt und die Entscheidungshilfe entsprechend angepasst werden.

Anhang

Anhang A : Beispielhafte Dockerfile zum Erstellen eines Container Images

```
1.  FROM node:latest             # Auswahl des NodeJS Images als Base Image
2.  WORKDIR /usr/src/app         # Setzen des Arbeitsverzeichnisses
3.  COPY package.json ./         # Kopiert Dateien die u. a. eine Liste der
                                   Abhängigkeiten beinhaltet in das Image
4.  RUN npm install              # Führt den Befehl zum Installieren der
                                   Abhängigkeiten aus.
5.  COPY . .                     # Kopiert Quelltext in das Image kopiert
6.  CMD [ "node", "example.js" ] # Startet die NodeJS Applikation
```

Quelltext 1: Beispielhafte Dockerfile zum Erstellen eines Images für eine NodeJS Anwendung auf Basis des NodeJS Images

Anhang B : Befehle zum Sichern eines Docker Volumes

```
1.  # Backup des Ordners /locationOfDataToBackup (innerhalb des Containers) in den
    Ordner /destinatinOfBackup (außerhalb des Containers)
2.  docker run --rm --volumes-from containerToBackUp \
    -v /destinatinOfBackup: /backup ubuntu \
    bash -c "tar -cvf /backup/backup.tar /locationOfDataToBackup"
3.  # obiges Backup wiederherstellen
4.  docker run --rm --volumes-from containerToBackUp \
    -v /destinatinOfBackup:/backup ubuntu \
    bash -c "cd /locationOfDataToRestore && rm -rf ./* && tar -xvf /backup/backup.tar
    --strip 1"
```

Quelltext 2: Befehle zum Sichern und Wiederherstellen eines Docker Volumes mithilfe des Programms tar; Befehle basieren auf [162]

Anhang C : Befehle zum Sichern von Kubernetes Objekten durch Velero

```
1.  # Erstellt ein Backup aller Kubernetes-Objekte mit dem Label app=nginx
2.  velero backup create nginx-backup --selector app=nginx
3.  # Erstellt ein Job, welcher täglich um 01:00 ein Backup generiert.
4.  velero schedule create nginx-daily --schedule="0 1 * * *" --selector app=nginx
5.  # Stellt ein zuvor erstelltes Backup wieder her.
6.  velero restore create --from-backup nginx-backup
```

Quelltext 3: Befehle zum Sichern und Wiederherstellen von Kubernetes Objekten durch das Softwarewerkzeug Velero; Befehle entnommen aus [163]

Anhang D : Docker Compose Datei für einen Webservice

```
1.  version: "3.9"
2.  services:
3.    webpage:
4.      build: .
5.      ports:
6.        - "8080:80"
7.      depends_on:
8.        - db
9.        - redis
10.   redis:
11.     image: "redis"
12.     ports:
13.       - "6379:6379"
14.     volumes:
15.       - "redisdata:/data"
16.   postgres:
17.     image: "postgres"
18.     ports:
19.       - "5432:5432"
20.     volumes:
21.       - "pgdata:/var/lib/postgresql/data/"
22. volumes:
23.   pgdata:
24.   redisdata:
```

Quelltext 4: eine beispielhafte Docker Compose Datei für einen Webservice; Basiert auf [86]

Anhang E : Konfiguration eines Kubernetes-Deployment

```
1.  apiVersion: apps/v1
2.  kind: Deployment
3.  metadata:
4.    name: example-deployment
5.    namespace: example-namespace # Abschnitt 2.6.1.4 und Quelltext 9
6.  spec:
7.    replicas: 3 # Zielvorgabe für die Anzahl der Pods
8.    selector:
9.      matchLabels:
10.       app: exampleApp # Deploment definiert Template für alle Pods mit dem
                            Label "app: exampleApp"
11.   template: # Definiton des Pod-Templates
12.     metadata:
13.       labels: # Labels sind Key-Value-Pairs und dienen der Identifizierung
                    von Ressourcen (siehe Zeile 8 - 10)
14.         app: exampleApp
15.     spec:
16.       containers:
17.       - name: example
18.         image: example:latest
19.         env:
20.         - name: example_enviroment # Umgebungsvariablen
21.           valueFrom:
22.             secretKeyRef: # z. B. Secrets (Abschnitt 2.6.1.2, Quelltext 7)
23.               name: example_secret
24.               key: EXAMPLE_ENVIROMENT
25.         volumeMounts: # mounten der Volumes
26.         - name: configuration
27.           mountPath: /example/configuration.yml
28.           subPath: configuration.yml
29.         - mountPath: /example/volume
30.           name: example-volume
31.       volumes: # Referenzierung der Volumes (Abschnitt 2.6.1.2, Quelltext 6)
32.       - name: configuration
33.         configMap:
34.           name: example-config
35.       - name: example-volume
36.         persistentVolumeClaim:
37.          claimName: example-volume
```

Quelltext 5: beispielhaften Konfiguration des Kubernetes-Deployment der Anwendung „exampleApp“

Anhang F : Konfiguration eines Kubernetes PV und ein PVCs

```
1.  apiVersion: v1
2.  kind: PersistentVolume
3.  metadata:
4.    name: example-pv-volume
5.    labels:
6.      type: local
7.  spec:
8.    capacity:
9.      storage: 10Gi
10.   accessModes:
11.     - ReadWriteOnce
12.   hostPath:
13.     path: "/mnt/data"
14. ---
15. apiVersion: v1
16. kind: PersistentVolumeClaim
17. metadata:
18.   name: example-volume
19.   namespace: example-namespace
20. spec:
21.   accessModes:
22.   - ReadWriteOnce # Eine Node hat ReadWrite-Berechtigungen, anderen Nodes keine
23.   resources:
24.     requests:
25.       storage: 1Gi
```

Quelltext 6: beispielhafte Konfiguration eines Kubernetes PV und ein PVCs

Anhang G : Konfiguration einer Kubernetes ConfigMap und eines Secrets

```
1.  apiVersion: v1
2.  kind: ConfigMap
3.  metadata:
4.    name: example-config
5.    namespace: example-namespace
6.  data:
7.    configuration.yml: |
8.      config:
9.          example: true
10. ---
11. apiVersion: v1
12. kind: Secret
13. metadata:
14.   name: example_secret
15.   namespace: example-namespace
16. data: # Secret ist base64 encoded, stringData: für Plaintext-Secret
17.   EXAMPLE_ENVIROMENT: ZXhhbXBsZQ== # example
```

Quelltext 7: beispielhafte Konfiguration der Kubernetes ConfigMap „example-config“ und des Secrets „example_secret“

Anhang H : Konfiguration eines Kubernetes Service und eines Ingress

```
1.  apiVersion: networking.k8s.io/v1
2.  kind: Ingress
3.  metadata:
4.    annotations:
5.      kubernetes.io/ingress.class: nginx
6.    name: example-ingress
7.    namespace: example-namespace
8.  spec:
9.    rules:
10.   - host: kubernetes.example.com # Von außerhalb ist der Service über diese URL
                                      ansprechbar
11.     http:
12.       paths:
13.       - path: /
14.         pathType: Prefix
15.         backend:
16.           service:
17.             name: example-service # Ingress für den Service "example-service"
18.             port:
19.               number: 80
20.   tls:
21.   - hosts:
22.     - kubernetes.example.com
23. ---
24. apiVersion: v1
25. kind: Service
26. metadata:
27.   name: example-service
28.   namespace: example-namespace
29. spec:
30.   ports: # Port 3000 der Pods wird über den Service-Port 80 bereitgestellt
31.   - port: 80
32.     protocol: TCP
33.     targetPort: 3000
34.   selector:
35.     app: exampleApp # Service für alle Pods mit dem Label "app: exampleApp"
```

Quelltext 8: beispielhafte Konfiguration des Service „example-service“ und des Ingress „example-ingress“

Anhang I : Konfiguration eines Kubernetes Namespaces

```
1.  apiVersion: v1
2.  kind: Namespace
3.  metadata:
4.    name: example-namespace
```

Quelltext 9: beispielhafte Konfiguration des Kubernetes Namespaces „example-namespace“

Anhang J : Konfiguration von Ressourcenbeschränkungen in Kubernetes

```
1.  apiVersion: "v1"
2.  kind: "LimitRange"
3.  metadata:
4.    name: "example-limits"
5.  spec:
6.    limits:
7.      - type: "Pod"
8.        max: # Maximaler Systemressourcenverbrauch je Pod
9.          cpu: "1"
10.         memory: "1Gi"
11.       min: # Minimum, das für jeden Pod reserviert wird
12.         cpu: "200m" # (1 CPU = 1000m)
13.         memory: "6Mi"
14. ---
15. apiVersion: v1
16. kind: ResourceQuota
17. metadata:
18.   name: example-quota
19.   namespace: example-namespace
20. spec:
21.   hard: # Die Summe des Systemressourcenverbrauchs aller Pods darf folgende Werte
    nicht überschreiten
22.     requests.cpu: "2" # request = reserviren von Systemressourcen in Pod-
    Konfiguration
23.     limits.cpu: "3"   # maximaler Systemressourcenverbrauch
```

Abbildung 29: beispielhafte Konfiguration für eine Ressourcenbeschränkungen für einzelne Pods und den gesamten Namespace in Kubernetes

Anhang K : Konfiguration einer Kubernetes Role und eines RoleBindings

```
1.  # Diese Rolle erlaubt es den Rolleninhabern alle Pods des
2.   Namespaces example-namespace zu lesen.
3.  apiVersion: rbac.authorization.k8s.io/v1
4.  kind: Role
5.  metadata:
6.    namespace: example-namespace
7.    name: pod-reader
8.  rules: # Berechtigungen
9.  - apiGroups: [""]
10.   resources: ["pods"]
11.   verbs: ["get", "watch", "list"]
12. ---
13. apiVersion: rbac.authorization.k8s.io/v1
14. # Dieses RoleBinding gibt dem Nutzer exampleUser die Rolle pod-reader
15. kind: RoleBinding
16. metadata:
17.   name: read-pods
18.   namespace: example-namespace
19. subjects: # users, groups, or service accounts
20. - kind: User
21.   name: exampleUser # "name" is case sensitive
22. roleRef: # welcher Rolle sollen sie zugeordnet werden
23.   kind: Role # Role oder ClusterRole
24.   name: pod-reader # Name der Role/ClusterRole
```

Quelltext 10: beispielhafte Konfiguration der Rolle „pod-reader“ und zuweisung an User „exampleUser“

Anhang L : Befehle zum Erstellen der VMs mittels KVM

```
1.  # VM für Performancetest
2.  sudo virt-install \
3.   --name ubuntu-vm-perfTest \
4.   --os-variant ubuntu20.04 \
5.   --vcpus 2 \
6.   --ram 2048 \
7.   --location http://ftp.ubuntu.com/ubuntu/dists/focal/main/installer-amd64/ \
8.   --network bridge=virbr0,model=virtio \
9.   --graphics none \
10.  --extra-args='console=ttyS0,115200n8 serial' \
11.  --disk pool=default,size=64,bus=virtio,format=qcow2 \
12.  --console pty,target_type=serial
13. # Docker-Host VM
14.  sudo virt-install \
15.  --name ubuntu-dockerHost \
16.  --os-variant ubuntu20.04 \
17.  --vcpus 3 \
18.  --ram 4096 \
19.  --location http://ftp.ubuntu.com/ubuntu/dists/focal/main/installer-amd64/ \
20.  --network bridge=virbr0,model=virtio \
21.  --graphics none \
22.  --extra-args='console=ttyS0,115200n8 serial' \
23.  --disk pool=default,size=15,bus=virtio,format=qcow2 \
24.  --console pty,target_type=serial
25. # Docker Container für Performancetest
26. sudo docker run -it \
27. --memory="2g" \
28. --cpus="2" \
29. --cpuset-cpus="0-1" \
30. -v /root/testresults:/var/lib/phoronix-test-suite/test-results \
31. -v /mountPoint:/mountPoint \ # für Datenträger - Performancetest
32. ph /bin/bash
```

Quelltext 11: Befehle zum Erstellen der KVM basierten VMs und des Docker Containers für die Bestimmung des Ressourcenverbrauch und der Performance.

Anhang M : Dockerfile für die Performancetest

```
1.  FROM ubuntu:latest
2.  RUN apt update && DEBIAN_FRONTEND="noninteractive" apt install -y tzdata
3.  RUN apt install -y zip unzip php-zip iperf3 gdebi-core cmake cmake-data libsctp-
    dev php-cli apt-utils mesa-utils php-xml git-core apt-file sudo git build-
    essential libcurl4-openssl-dev libssl-dev zlib1g-dev automake libtool libncurses5-
    dev
4.  RUN git clone https://github.com/phoronix-test-suite/phoronix-test-suite.git
5.  RUN cd phoronix-test-suite && chmod +x install-sh && ./install-sh
```

Quelltext 12: Dockerfile für das *Phoronix Test Suite* Conatiner Image, dass im Performancetest genutzt wurde. Basiert auf [164]

Anhang N : Befehle zur Bestimmung des Ressourcenverbrauchs im Leerlauf

```
1. # Misst die Performance aller einer Sekunde für 60 Sekunden
2. nmon -f -s 1 -c 60
3. # Führt den docker stats Befehl für 60 Sekunden aus und schreibt den Stream Output
   mit Zeitangabe in eine Datei. Im zweiten Schritt wird die Ausgabe noch gefiltert.
4. timeout 60 docker stats | ts '[%Y-%m-%d %H:%M:%S]' >> ./idleDocker.txt
5. cat ./idleDocker.txt | grep -v CONTAINER >> idleDocker.csv
```

Quelltext 13: Befehle zur Bestimmung des Ressourcenverbrauchs im Leerlauf durch die Software *nmon* für die VMs bzw. *docker stats* für die Container

Anhang O : Abfrage der Ressourcenauslastung der Kubernetes Nodes mittels kubecrtl

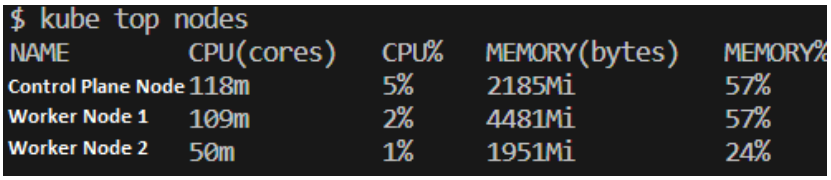


Abbildung 30: Abfrage der Ressourcenauslastung der Control Plane Node und der Worker Nodes.

Anhang P Ergebnisse des Kubernetes Dashboards

nicht-IT-Service Pods
(stellen stattdessen Cluster-Funktionen bereit oder dienen ader Cluster Verwaltung)

Name	Namespace	Node	CPU-Nutzung	Speichernutzung
nfs-provisioner-nfs-subdir-	nfs-provisioner	Worker Node 1	1,00m	9,59MiB
metallb-controller-7ffbf6c!	metallb	Worker Node 1	1,00m	22,42MiB
dashboard-metrics-scrape	kubernetes-dashboard	Worker Node 1	9,00m	14,39MiB
tigera-operator-64cf9f98d	tigera-operator	Worker Node 1	2,00m	32,68MiB
kubernetes-dashboard-59	kubernetes-dashboard	Worker Node 1	9,00m	26,89MiB
coredns-7cb5d66c8-5h9dp	kube-system	Worker Node 1	2,00m	15,86MiB
calico-kube-controllers-55	calico-system	Worker Node 1	2,00m	34,24MiB
calico-apiserver-5f99f7887	calico-apiserver	Worker Node 1	2,00m	71,62MiB
calico-typha-78cf65544f-9	calico-system	Worker Node 1	2,00m	24,16MiB
calico-node-vthpv	calico-system	Worker Node 1	12,00m	148,35MiB
kube-proxy-qwjhs	kube-system	Worker Node 1	1,00m	48,23MiB
metallb-speaker-4nnrc	metallb	Worker Node 1	2,00m	21,06MiB
csi-node-driver-bbz8z	calico-system	Worker Node 1	1,00m	11,91MiB
metrics-server-8bc46949b	metrics-server	Worker Node 2	2,00m	18,82MiB
ingress-nginx-controller-7	ingress-nginx	Worker Node 2	2,00m	142,17MiB
calico-typha-78cf65544f-q	calico-system	Worker Node 2	1,00m	51,34MiB
calico-node-49g9f	calico-system	Worker Node 2	13,00m	147,79MiB
kube-proxy-j7xh7	kube-system	Worker Node 2	1,00m	48,29MiB
metallb-speaker-87rwk	metallb	Worker Node 2	3,00m	19,62MiB
csi-node-driver-4wzj6	calico-system	Worker Node 2	2,00m	11,46MiB
Summe			**70,00m**	**920,89MiB**

Verursacher	Arbeitsspeicherverbrauch	CPU-Kerne
Worker Node Pods	920,89 MiB	70m
Controll Plane Node	3931,69 MiB	2000m
Summe	**4852,58 MiB**	**2070m**

IT-Service Pods

Name	Namespace	Node	CPU-Nutzung	Speichernutzung
kbv-erechnungskorb-test-7f75cd	kbv-erechnungskorb	Worker Node 2	0,00m	7,46MiB
mariadb-7f68d8f8b-8cr9f	redmine	Worker Node 1	1,00m	111,82MiB
redmine-6b78595f6-twv6d	redmine	Worker Node 1	1,00m	234,42MiB
webpdf-588cc48498-zj7fh	webpdf	Worker Node 1	1,00m	808,72MiB
keycloak-7d68684f9b-pp4sh	keycloak	Worker Node 1	1,00m	559,43MiB
gkv-erechnungskorb-6bd64b5c8	gkv-portal	Worker Node 1	1,00m	12,69MiB
kbv-erechnungskorb-b7c5cc577-	kbv-erechnungskorb	Worker Node 1	0,00m	13,88MiB
mariadb-57b8f65d56-4xcvk	keycloak	Worker Node 1	1,00m	152,17MiB
gisa-erechnungsportal-test-8d58	gisa-erechnungsportal	Worker Node 1	1,00m	3,48MiB
gisa-erechnungsportal-7685c497	gisa-erechnungsportal	Worker Node 1	1,00m	9,81MiB
redmine-77c845d679-jssf2	dbe-kfm-redmine-prod	Worker Node 1	1,00m	275,80MiB
mariadb-57b8f65d56-z4m4x	dbe-kfm-redmine-prod	Worker Node 1	1,00m	123,65MiB

Verursacher	Arbeitsspeicherverbrauch	CPU-Kerne
Worker Node Pods	920,89 MiB	10m
Controll Plane Node	0 MiB	0m
Summe	**920,89 MiB**	**10m**

Abbildung 31: Ressourcenverbrauch der Pods im Kubernetes Cluster, getrennt nach IT-Service und nicht IT-Service Pods; ausgelesen mithilfe des Kubernetes Dashboards

Anhang Q : weiterführende Informationen über die Kubernetes Nodes

```
# Control Plane Node
[...]
Capacity:
  cpu:                2
  ephemeral-storage:  58709636Ki
  hugepages-2Mi:      0
  memory:             4026048Ki
  pods:               110
Allocatable:
  cpu:                2
  ephemeral-storage:  54106800449
  hugepages-2Mi:      0
  memory:             3923648Ki
  pods:               110
System Info:
[...]
  OS Image:                   Ubuntu 20.04.5 LTS
  Operating System:           linux
  Architecture:               amd64
  Container Runtime Version:  containerd://1.5.9
[...]
##Worker 1
[...]
Capacity:
  cpu:                4
  ephemeral-storage:  91736696Ki
  hugepages-2Mi:      0
  memory:             8148188Ki
  pods:               110
Allocatable:
  cpu:                4
  ephemeral-storage:  84544538894
  hugepages-2Mi:      0
  memory:             8045788Ki
  pods:               110
System Info:
[...]
  OS Image:                   Ubuntu 20.04.5 LTS
  Operating System:           linux
  Architecture:               amd64
  Container Runtime Version:  containerd://1.5.9
[...]
### Worker Node 2
[...]
Capacity:
  cpu:                4
  ephemeral-storage:  91736696Ki
  hugepages-2Mi:      0
  memory:             8148192Ki
  pods:               110
Allocatable:
  cpu:                4
  ephemeral-storage:  84544538894
  hugepages-2Mi:      0
  memory:             8045792Ki
  pods:               110
System Info:
[...]
  OS Image:                   Ubuntu 20.04.5 LTS
  Operating System:           linux
  Architecture:               amd64
  Container Runtime Version:  containerd://1.5.9
[...]
```

Quelltext 14: Auszug aus dem Befehl *kubctl describe nodes*, welcher Umfangreiche Informationen über die Konfiguration einer Node bietet.

Anhang R : Phoronix Test Suite Testbericht VM

www.phoronix-test-suite.com

Performancetest VM

KVM testing on Ubuntu 20.04 via the Phoronix Test Suite.

Test Systems:

VM

Processor: 2 x Intel Xeon (Skylake) (2 Cores), Motherboard: QEMU Standard PC (Q35 + ICH9 2009) (1.13.0-1ubuntu1.1 BIOS), Chipset: Intel 82G33/G31/P35/P31 + ICH9, Memory: 1 x 2048 MB RAM QEMU, Disk: 63GB, Network: Red Hat Virtio device

OS: Ubuntu 20.04, Kernel: 5.4.0-135-generic (x86_64), Compiler: GCC 9.4.0, File-System: ext4, System Layer: KVM

Kernel Notes: Transparent Huge Pages: madvise
Compiler Notes: --build=x86_64-linux-gnu --disable-vtable-verify --disable-werror --enable-checking=release --enable-clocale=gnu --enable-default-pie --enable-gnu-unique-object --enable-languages=c,ada,c++,go,brig,d,fortran,objc,obj-c++,gm2 --enable-libstdcxx-debug --enable-libstdcxx-time=yes --enable-multiarch --enable-multilib --enable-nls --enable-objc-gc=auto --enable-offload-targets=nvptx-none=/build/gcc-9-Av3uEd/gcc-9-9.4.0/debian/tmp-nvptx/usr,hsa --enable-plugin --enable-shared --enable-threads=posix --host=x86_64-linux-gnu --program-prefix=x86_64-linux-gnu- --target=x86_64-linux-gnu --with-abi=m64 --with-arch-32=i686 --with-default-libstdcxx-abi=new --with-gcc-major-version-only --with-multilib-list=m32,m64,mx32 --with-target-system-zlib=auto --with-tune=generic --without-cuda-driver -v
Disk Notes: MQ-DEADLINE / errors=remount-ro,relatime,rw / Block Size: 4096
Processor Notes: CPU Microcode: 0x1

Phoronix Test Suite

Performancetest VM

Security Notes: itlb_multihit: Not affected + l1tf: Mitigation of PTE Inversion; VMX: flush not necessary SMT disabled + mds: Mitigation of Clear buffers; SMT Host state unknown + meltdown: Mitigation of PTI + mmio_stale_data: Vulnerable: Clear buffers attempted no microcode; SMT Host state unknown + retbleed: Vulnerable + spec_store_bypass: Vulnerable + spectre_v1: Mitigation of usercopy/swapgs barriers and __user pointer sanitization + spectre_v2: Mitigation of Retpolines STIBP: disabled RSB filling PBRSB-eIBRS: Not affected + srbds: Not affected + tsx_async_abort: Mitigation of Clear buffers; SMT Host state unknown

	VM
GnuPG - 2.7.S.F.E (sec)	129.087
Standard Deviation	0.8%
7-Zip Compression - Compression Rating (MIPS)	7336
Standard Deviation	2.6%
7-Zip Compression - D.R (MIPS)	4785
Standard Deviation	0.7%
MariaDB - 1 (Queries/sec)	513
Standard Deviation	2.4%
nginx - 100 (Reqs/sec)	4982
Standard Deviation	0.1%
PostMark - D.T.P (TPS)	1296
Standard Deviation	2.4%
Ethr - TCP - Latency - 1 (us)	139.249
Standard Deviation	4.4%
Ethr - TCP - Bandwidth - 1 (Gbits/sec)	11.86
Standard Deviation	2.2%
MBW - Memory Copy - 128 MiB (MiB/s)	4240
Standard Deviation	1%
IPC_benchmark - Unnamed Pipe - 1024 (Messages/sec)	874518
Standard Deviation	2.4%
IPC_benchmark - FIFO Named Pipe - 1024 (Messages/sec)	818773
Standard Deviation	2.3%

Phoronix Test Suite

Performancetest VM

GnuPG 2.2.27

2.7GB Sample File Encryption

1. (CC) gcc options: -O2

7-Zip Compression 22.01

Test: Compression Rating

1. (CXX) g++ options: -lpthread -ldl -O2 -fPIC

7-Zip Compression 22.01

Test: Decompression Rating

1. (CXX) g++ options: -lpthread -ldl -O2 -fPIC

MariaDB 10.8.2

Clients: 1

1. (CXX) g++ options: -pie -fPIC -fstack-protector -O3 -pthread -lcrypt -lz -lm -lssl -lcrypto -lpthread -ldl

nginx 1.23.2

Connections: 100

1. (CC) gcc options: -lluajit-5.1 -lm -lssl -lcrypto -lpthread -ldl -std=c99 -O2

Phoronix Test Suite

Performancetest VM

PostMark 1.51

Disk Transaction Performance

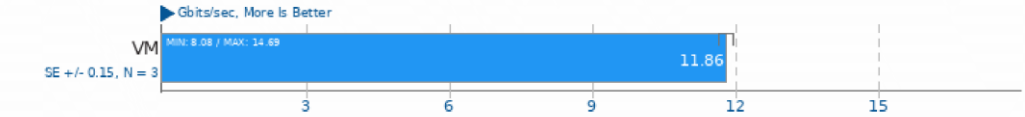
1. (CC) gcc options: -O3

Ethr 1.0

Server Address: localhost - Protocol: TCP - Test: Latency - Threads: 1

Ethr 1.0

Server Address: localhost - Protocol: TCP - Test: Bandwidth - Threads: 1



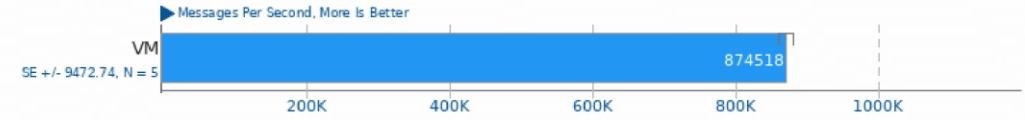
MBW 2018-09-08

Test: Memory Copy - Array Size: 128 MiB

1. (CC) gcc options: -O3 -march=native

IPC_benchmark

Type: Unnamed Pipe - Message Bytes: 1024

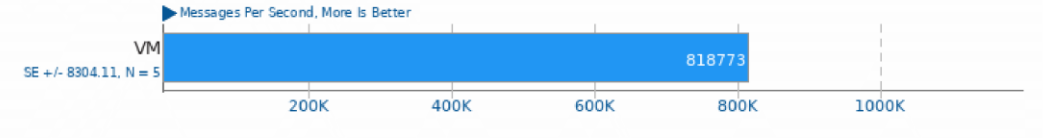


Phoronix Test Suite

Performancetest VM

IPC_benchmark

Type: FIFO Named Pipe - Message Bytes: 1024



This file was automatically generated via the Phoronix Test Suite benchmarking software on Thursday, 5 January 2023 13:21.

Anhang S : Phoronix Test Suite Testbericht Docker

www.phoronix-test-suite.com

Performancetest Docker

Docker testing on Ubuntu 22.04 via the Phoronix Test Suite.

Test Systems:

Docker

Processor: 2 x Intel Xeon (Skylake) (2 Cores), Motherboard: QEMU Standard PC (Q35 + ICH9 2009) (1.13.0-1ubuntu1.1 BIOS), Memory: 2048MB, Disk: 63GB

OS: Ubuntu 22.04, Kernel: 5.4.0-135-generic (x86_64), Compiler: GCC 11.3.0, File-System: overlayfs, System Layer: Docker

Kernel Notes: Transparent Huge Pages: madvise
Compiler Notes: --build=x86_64-linux-gnu --disable-vtable-verify --disable-werror --enable-bootstrap --enable-cet --enable-checking=release --enable-clocale=gnu --enable-default-pie --enable-gnu-unique-object --enable-languages=c,ada,c++,go,brig,d,fortran,objc,obj-c++,m2 --enable-libphobos-checking=release --enable-libstdcxx-debug --enable-libstdcxx-time=yes --enable-link-serialization=2 --enable-multiarch --enable-multilib --enable-nls --enable-objc-gc=auto --enable-offload-targets=nvptx-none=/build/gcc-11-xKiWfi/gcc-11-11.3.0/debian/tmp-nvptx/usr,amdgcn-amdhsa=/build/gcc-11-xKiWfi/gcc-11-11.3.0/debian/tmp-gcn/usr --enable-plugin --enable-shared --enable-threads=posix --host=x86_64-linux-gnu --program-prefix=x86_64-linux-gnu- --target=x86_64-linux-gnu --with-abi=m64 --with-arch-32=i686 --with-build-config=bootstrap-lto-lean --with-default-libstdcxx-abi=new --with-gcc-major-version-only --with-multilib-list=m32,m64,mx32 --with-target-system-zlib=auto --with-tune=generic --without-cuda-driver -v

Phoronix Test Suite

Performancetest Docker

Processor Notes: CPU Microcode: 0x1
Security Notes: itlb_multihit: Not affected + l1tf: Mitigation of PTE Inversion; VMX: flush not necessary SMT disabled + mds: Mitigation of Clear buffers; SMT Host state unknown + meltdown: Mitigation of PTI + mmio_stale_data: Vulnerable: Clear buffers attempted no microcode; SMT Host state unknown + retbleed: Vulnerable + spec_store_bypass: Vulnerable + spectre_v1: Mitigation of usercopy/swapgs barriers and __user pointer sanitization + spectre_v2: Mitigation of Retpolines STIBP: disabled RSB filling PBRSB-eIBRS: Not affected + srbds: Not affected + tsx_async_abort: Mitigation of Clear buffers; SMT Host state unknown

	Docker
GnuPG - 2.7.S.F.E (sec)	136.639
Standard Deviation	1.7%
7-Zip Compression - Compression Rating (MIPS)	7313
Standard Deviation	2%
7-Zip Compression - D.R (MIPS)	4873
Standard Deviation	0.6%
MariaDB - 1 (Queries/sec)	515
Standard Deviation	1.5%
nginx - 100 (Reqs/sec)	4789
Standard Deviation	0.8%
PostMark nicht-persistente Schicht - D.T.P (TPS)	1210
Standard Deviation	2.2%
PostMark Docker Bind Mount - D.T.P (TPS)	1267
Standard Deviation	3.3%
Ethr - TCP - Latency - 1 (us)	167.057
Standard Deviation	1.4%
Ethr - TCP - Bandwidth - 1 (Gbits/sec)	10.90
Standard Deviation	5.8%
MBW - Memory Copy - 128 MiB (MiB/s)	4126
Standard Deviation	2.3%
IPC_benchmark - Unnamed Pipe - 1024 (Messages/sec)	833460
Standard Deviation	0.7%
IPC_benchmark - FIFO Named Pipe - 1024 (Messages/sec)	798080
Standard Deviation	2.4%

Phoronix Test Suite

Performancetest Docker

GnuPG 2.2.27

2.7GB Sample File Encryption

1. (CC) gcc options: -O2

7-Zip Compression 22.01

Test: Compression Rating

1. (CXX) g++ options: -lpthread -ldl -O2 -fPIC

7-Zip Compression 22.01

Test: Decompression Rating

1. (CXX) g++ options: -lpthread -ldl -O2 -fPIC

MariaDB 10.8.2

Clients: 1

1. (CXX) g++ options: -pie -fPIC -fstack-protector -O3 -lcrypt -lz -lm -lssl -lcrypto -lpthread -ldl

nginx 1.23.2

Connections: 100

1. (CC) gcc options: -lluajit-5.1 -lm -lssl -lcrypto -lpthread -ldl -std=c99 -O2

Phoronix Test Suite

Performancetest Docker

PostMark nicht-persistente Schicht 1.51

Disk Transaction Performance

1. (CC) gcc options: -O3

PostMark Docker Bind Mount 1.51

Disk Transaction Performance

1. (CC) gcc options: -O3

Ethr 1.0

Server Address: localhost - Protocol: TCP - Test: Latency - Threads: 1

Ethr 1.0

Server Address: localhost - Protocol: TCP - Test: Bandwidth - Threads: 1

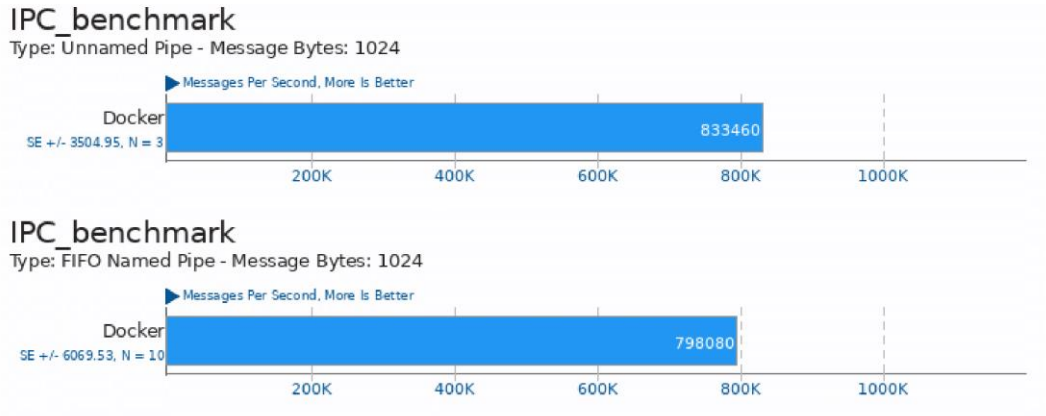
MBW 2018-09-08

Test: Memory Copy - Array Size: 128 MiB

1. (CC) gcc options: -O3 -march=native

Phoronix Test Suite

Performancetest Docker



This file was automatically generated via the Phoronix Test Suite benchmarking software on Thursday, 5 January 2023 13:17.

Anhang T : Umfrage Angular

Angular-basierte Webapplikation

Initiale Installation und Konfiguration des IT-Services

Ziel:

Es soll eine Angular-basierte Webanwendung bereitgestellt werden, welche über eine URL aufrufbar ist. Es existiert bereits ein Bitbucket-Repository für das Projekt.

Aufwandsschätzung für VM

Tätigkeiten	Zeitaufwand
Bereitstellung neuer VM	1-2 h
Installation von nginx auf VM und Erstellen der nginx-Konfiguration	0,5 h
Jenkins Pipeline erstellen und ausführen	0,25 h
VM im Netz erreichbar machen (Firewall-Regeln und Nginx-Proxy)	0,5 h

Aufwandsschätzung für Docker

Tätigkeiten	Zeitaufwand
Docker-Container anmelden	0,5 h
Erstellen der Jenkins Pipeline, nginx-Konfiguration und der Dockerfile	0,5 h
Erstellen der Docker-Compose Datei	0,25 h
Jenkins Pipeline ausführen und auf Docker-Host bereitstellen	0,25 h
Container über Domain Name erreichbar machen (Nginx-Proxy)	0,25 h

Aufwandsschätzung für Docker ohne bereits existierende Docker Compose Datei als Vorlage

Tätigkeiten	Zeitaufwand
Docker-Container anmelden	0,5 h
Erstellen der Jenkins Pipeline, nginx-Konfiguration und der Dockerfile	0,5 h
Erstellen der Docker Compose Datei	0,5 h
Jenkins Pipeline ausführen und auf Docker-Host bereitstellen	0,25 h
Container über Domain Name erreichbar machen (Nginx-Proxy)	0,25 h

Aufwandsschätzung für Kubernetes ohne Helm Chart

Tätigkeiten	Zeitaufwand
Kubernetes Pods anmelden	0,5 h
Erstellen der Jenkins Pipeline, nginx-Konfiguration und der Dockerfile	=
Kubernetes Konfigurationsdateien erstellen	0,25 h
Jenkins Pipeline ausführen und im Kubernetes Cluster bereitstellen	0,25 h

Aufwandsschätzung für Kubernetes mit Helm Chart

Tätigkeiten	Zeitaufwand
Kubernetes Pods anmelden	=
Erstellen der Jenkins Pipeline, nginx-Konfiguration und der Dockerfile	=
Values für Angular Helm Chart erstellen	0,25 h
Jenkins Pipeline ausführen und im Kubernetes Cluster bereitstellen	0,25 h

Aufwandsschätzung für Kubernetes ohne Helm Chart und ohne bereits existierende Kubernetes Konfigurationen als Vorlage

Tätigkeiten	Zeitaufwand
Kubernetes Pods anmelden	=
Erstellen der Jenkins Pipeline, nginx-Konfiguration und der Dockerfile	=
Kubernetes Konfigurationsdateien erstellen	0.75 h
Jenkins Pipeline ausführen und im Kubernetes Cluster bereitstellen	0,25 h

Zusätzliche Anmerkungen:

Die Container Anmeldung dient der Abrechnung und der Verwaltung des Hosts/Clusters.

Da bereits Angular Anwendungen im Kubernetes bereitgestellt werden, können deren Konfigurationen als Grundlage genutzt werden.

Durchführen von Softwareupdates

Ziel:

Für die zuvor installierte Angular-basierte Webanwendung steht eine neue Version bereit, welche nun bereitgestellt werden soll. Zudem soll das Betriebssystem und der nginx-Server aktualisiert werden.

Aufwandsschätzung für VM

Tätigkeiten	Zeitaufwand
Aktualisieren des Gast-Betriebssystems und des nginx Servers	0,5 h
Jenkins Pipeline ausführen	0,25 h
ggf. nginx-Server neustarten (für Angular-Update)	0-0,25 h

Aufwandsschätzung für Docker

Tätigkeiten	Zeitaufwand in PT
Anpassen der Dockerfile für neues nginx Base Image	0,25 h
Jenkins Pipeline ausführen, ggf. Image Tag in Konfiguration ändern und Container neustarten	0,25 h

Aufwandsschätzung für Kubernetes

Tätigkeiten	Zeitaufwand in PT
Anpassen der Dockerfile für neues nginx Base Image	=
Jenkins Pipeline ausführen, ggf. Image Tag in Konfiguration ändern und Pods neustarten	0,25 h

Zusätzliche Anmerkungen:

Sollte bei der Aktualisierung der Angular Applikation die nginx-Konfiguration geändert wurden sein, muss der nginx-Server in der VM neugestartet werden. Bei Docker und Kubernetes erfolgt das durch den Container- bzw. Pod-Neustart automatisch.

Hochverfügbarkeit eines IT-Services

Für eine Angular-basierte Webapplikation sollen mehrere Instanzen bereitgestellt werden, um die Hochverfügbarkeit zu gewährleisten.

Aufwandsschätzung für VM

Tätigkeiten	Zeitaufwand in PT
Kopie der ersten VM beantragen	1-2 h
Anpassen und Ausführen der Jenkins Pipeline	0,25 h
Konfiguration von zentralem nginx-Servers als Load-Balancers	0,5 h

Aufwandsschätzung für Kubernetes

Tätigkeiten	Zeitaufwand in PT
Kubernetes Konfigurationsdatei anpassen	0,25 h

Zusätzliche Anmerkungen:

Migration

Migration von Docker zu Kubernetes ohne Helm Chart

Ziel: Die zuvor per Docker bereitgestellte Angular-Anwendung soll zu Kubernetes migriert werden.

Tätigkeiten	Zeitaufwand
Kubernetes Konfigurationsdateien erstellen	0,25 h
Im Kubernetes Cluster bereitstellen und Docker Container entfernen	0,25 h
Docker Container ab- und Kubernetes Pods anmelden	0,75 h
Reverse Proxy Einstellungen für Docker Container entfernen	0,25 h

Migration von Docker zu Kubernetes mit Helm Chart

Ziel: Die zuvor per Docker bereitgestellte Angular-Anwendung soll zu Kubernetes migriert werden.

Tätigkeiten	Zeitaufwand
Values für Angular Helm Chart erstellen	0,25 h
Im Kubernetes Cluster bereitstellen und Docker Container entfernen	=
Docker Container ab- und Kubernetes Pods anmelden	=
Reverse Proxy Einstellungen für Docker Container entfernen	=

Zusätzliche Anmerkungen:

Anhang U : Umfrage inubit

Business Process Management Software Inubit

Initiale Installation und Konfiguration des IT-Services

Die BPM Software Inubit und der dazugehörige Datenbankserver soll bereitgestellt werden. In der VM können sowohl inubit als auch der Datenbankserver installiert werden.

Aufwandsschätzung für VM

Tätigkeiten	Zeitaufwand
Anforderungsanalyse: Erfassung möglichst aller relevanter Anforderungen. Das können Verarbeitungsintervalle, Größe von Daten, Anzahl Userinteraktionen sowie Anforderungen an bestimmte Funktionalitäten sein (http/https, woher erreichbar, …), welche Verfügbarkeit des Systems ...	2-6 h
Prüfung Abhängigkeiten: I.d.R. gibt es in den Installationsanleitungen Hard- und Softwarevoraussetzungen. Diese müssen geprüft und bewertet werden	1 h
Ableitung für Beschaffung und Erstellung entsprechender Changes: Anhand der Anforderungen wird eine konkrete Liste an benötigten Infrastrukturkomponenten erstellt und beantragt. Weiterhin muss ein Sizing der Komponenten erfolgen (CPU, RAM, HDD,…). Zur Kommunikation mit Umsystemen ist die Einrichtung von FW-Rules usw. nötig	3 h
Einrichtung der Systeme I: Erstellung eines Userkonzeptes (DB, App1, App2, …) für Betriebssystem-User	1 h
Einrichtung der Systeme II: Einrichtung Environment, Pfade, Sprachen, Zeichencodierung, Aliases, weitere Umgebungsvariablen oder BS-Parameter (z.B. Limits)	2 h
Installation der Software I: Zuerst Basis-Software wie DB. Maria-DB wird auf der VM durch DBAs installiert. Bestimmte Parameter werden mit den DBAs abgestimmt.	4 h
Installation der Software II: Inubit installieren Bereitstellung des Installers auf dem System (per wget oder per sftp). Installer mit dem entsprechenden User ausführen. Auswahl inubit Process Engine. Danach Bereitstellung MariaDB-JDBC-Treiber. Anpassung der Config zur Anbindung der DB. Weitere Anpassungen für JVM. Automatischen Start einrichten (systemd). Inubit starten.	2 h

Aufwandsschätzung für Docker

Tätigkeiten	Zeitaufwand
Anforderungsanalyse: **(Wie VM)**	(Wie VM)
Prüfung Abhängigkeiten: **(Wie VM)**	(Wie VM)
Ableitung für Beschaffung und Erstellung entsprechender Changes: Anhand der Anforderungen wird eine konkrete Liste an benötigten Containern usw. erstellt und angemeldet. Weiterhin muss ein Sizing der Komponenten erfolgen (CPU, RAM, HDD,…). Zur Kommunikation mit Umsystemen ist die Einrichtung von FW-Rules usw. nötig	1,5 h
Container Image erstellen I: Antwortdatei für den inubit Installer erstellen. Container erstellen. Inubit-Installer in Container Kopieren und ausführen. Dateien aus Container kopieren.	0,5 h
Container Image erstellen II: Erstellen der Dockerfile. Die Dockerfile entspricht inhaltlich „Installation der Software II“ bei der VM, automatisches Starten des inubit Servers über Skript bei Containerstart. Image wird ins Repository geladen.	2 h
Maria-DB bereitstellen: Wird nicht durch DBAs übernommen. Erstellen der DBs und setzen der Parameter. Backup Job einrichten.	2 h
Erstellen der Docker-Compose Datei und Container starten Definition einer Festen MAC-Adresse für Lizenzierung. Kopieren der Konfigurationsdateien z.B. für DB-Verbindungen aus Container, anpassen und einbinden dieser.	2-3 h

Aufwandsschätzung für Kubernetes

Tätigkeiten	Zeitaufwand
Anforderungsanalyse: **(Wie VM)**	(Wie VM)
Prüfung Abhängigkeiten: **(Wie VM)**	(Wie VM)
Ableitung für Beschaffung und Erstellung entsprechender Changes: (Wie Docker)	(Wie Docker)
Container Image erstellen I: (Wie Docker)	(Wie Docker)
Container Image erstellen II: (Wie Docker)	(Wie Docker)
Maria-DB bereitstellen: (Wie Docker)	(Wie Docker)
Kubernetes Konfigurationsdateien erstellen und im Kubernetes Cluster bereitstellen Kopieren der Konfigurationsdateien z.B. für DB-Verbindungen aus Container, anpassen und einbinden dieser. Definition einer Festen MAC-Adresse für Lizenzierung → Calico	2,5-3,5 h

Zusätzliche Anmerkungen:

Durchführen von Softwareupdates

Für die zuvor installierte Software Inubit steht eine neue Version bereit, welche nun bereitgestellt werden soll. Zusätzlich dazu soll das Gast-Betriebssystem/Base-Image aktualisiert werden.

Aufwandsschätzung für VM

Tätigkeiten	Zeitaufwand
Evaluation: Es wird überprüft, ob das Update durchgeführt werden soll. Das basiert z.B. auf wichtigen Sicherheitsupdates, relevanten Fehlerbehebungen oder wichtigen neuen Features. Breaking Changes/Deprecations müssen beachtet werden. Aufwand stark von angekündigten Änderungen abhängig da ggfs. Untersuchungen zur Relevanz und Auswirkung und möglicher Alternativen nötig ist. Weiterhin bestimmt die Anzahl von Patchversionen von der alten zur neuen Version den Aufwand da potenziell mehr zu untersuchen ist je größer dieser Abstand ist.	4 -16 h
Patchplan erstellen idealerweise anhand einer Systemkopie werden der Patch und alle nötigen Patchschritte durchgeführt und dokumentiert	4 -8 h
Vorbereitung Terminabstimmung mit Entwicklern/Betrieb/ internen Kunden nötig.	4 h
Inubit Update durchführen: Server herunterfahren. Patch-Installer auf Server bereitstellen und Patch durchführen.	1 h
zusätzliche Patch-Schritte: Bei einigen Patches sind zusätzliche Patch-Schritte notwendig, welche manuell durchgeführt werden müssen.	2 h
Abschluss: Funktionstests durch Admins und Fachbereiche. Je nach Menge der Änderungen durch den Patch ist ein mehr oder weniger großer Aufwand zum Testen einzuplanen	4 h

Aufwandsschätzung für Docker

Tätigkeiten	Zeitaufwand
Evaluation: (Wie VM)	(Wie VM)
Vorbereitung: Terminabstimmung mit Entwicklern/Betrieb/ internen Kunden nötig. Neue Antwortdateien erzeugen (wie bei Installation).	4,5 h
Patchplan erstellen: Kopie des inubit und Datenbank Container erstellen, Patch-Installer dort bereitstellen und Patch mit Antwortdatei durchführen. Ergebnisse dokumentieren.	4 -8 h
Inubit Update in Container durchführen: Patch-Installer und Antwortdatei innerhalb des Containers bereitstellen und Patch mit Antwortdatei durchführen.	1 h
zusätzliche Patch-Schritte: Bei einigen Patches sind zusätzliche Patch-Schritte notwendig, welche manuell durchgeführt werden müssen. Im Durchschnitt ein Aufwand von ca.:	(Wie VM)
Docker Compose Konfiguration und Image aktualisieren: Dockerfile aktualisieren, Image neu bauen mit neuem Installer und Antwortdatei und in Repository laden. Ggf. Docker Compose Config anpassen.	2 h
Abschluss: (Wie VM)	(Wie VM)

Aufwandsschätzung für Kubernetes

Tätigkeiten	Zeitaufwand
Evaluation: (Wie VM)	(Wie VM)
Vorbereitung:	(Wie Docker)
Patchplan erstellen:	(Wie Docker)
Inubit Update in Container durchführen:	(Wie Docker)
zusätzliche Patch-Schritte:	(Wie VM)
Docker Compose Konfiguration und Image aktualisieren:	(Wie Docker)

Zusätzliche Anmerkungen:

Der inubit Patch Installer benötig bei der Ausführung Zugriff auf die Datenbanken, da z.B. Schemata aktualisiert werden müssen etc. Das macht den Update Prozess bei Kubernetes und Docker um einiges Komplexer.
Docker und Kubernetes besitzen hier im Ablauf und Aufwand keine relevanten Unterschiede.

Migration

VM zu Kubernetes

Ziel: Die bereits in einer VM existierende Inubit Instance soll in den Kubernetes Cluster migriert werden. Dabei muss auch der Datenbankserver, der in derselben VM läuft, migriert werden.

Tätigkeiten	Zeitaufwand
Anforderungsanalyse: Die meisten Informationen können vom Quell System übernommen werden. Einige Anforderungen müssen aber erneut überprüft werden z.B. Firewall Einstellungen etc.	1 h
Prüfung Abhängigkeiten: Die meisten Informationen können auch hier vom Quell System übernommen werden. Es sollte aber nochmal eine Kubernetes spezifische Überprüfung erfolgen. Zudem: Überprüfen, welche Auswirkungen die Migration auf externe Systeme hat.	1 h
Ableitung für Beschaffung und Erstellung entsprechender Changes: Anhand der Anforderungen wird eine konkrete Liste an benötigten Infrastrukturkomponenten erstellt. Überprüfen, ob Sizing des Quellsystem übernommen oder angepasst werden muss. Anmelden der Kubernetes Pods	3 h
Container Image erstellen I: (Wie bei Installation)	(Wie bei Installation)
Container Image erstellen II: (Wie bei Installation)	(Wie bei Installation)
Maria-DB bereitstellen: (Wie bei Installation)	(Wie bei Installation)
Kubernetes Konfigurationsdateien erstellen und im Kubernetes Cluster bereitstellen	(Wie bei Installation)
Migrationsplan erstellen: Das leere frisch installierte Inubit wird gesichert. Mit einem Abzug des Quellsystems (Offlinebackup) wird die Migration durchgeführt und dokumentiert. Bei Fehlern werden diese analysiert und behoben (das kann dauern, da manchmal auch Tickets bei Virtimo nötig sind). Das leere Inubit wird aus der Sicherung wieder hergestellt und die Migration nochmals wiederholt, bis sie fehlerfrei durchgelaufen ist. Das Ergebnis ist eine Migrationsplan, der dann bei der scharfen Migration abgearbeitet wird.	8-16 h
Vorbereitung + Sicherung in VM: Terminabstimmung mit Entwicklern/Betrieb/(internen Kunden) nötig. Optimierung von Daten, d.h. Reduktion von Daten (v.a. Logging-DB) um den Migrationslauf zu verkürzen. Server auf Quell und Zielsystem ausschalten. Automatische Sicherung mithilfe des Skripts erstellen.	5 h
manuelle Migration: Einige Dateien und Einstellungen müssen manuell zu Kubernetes übertragen und ggf. angepasst werden.	1 h

Wiederherstellung der Sicherung in Kubernetes Cluster: Erstellte Sicherung in Pod übertragen. Ausführen des Migrationsskriptes zur Wiederherstellung der Sicherung.	3 h
neuen Server starten und testen Kubernetes Instanz wieder starten und Funktionstest durchführen.	4.5 h
alten Server zurückbauen: Löschung der alten VM beantragen. FW-Einstellungen usw. für altes System entfernen	1 h

Kubernetes zu VM

Ziel: Die zuvor in Kubernetes aufgesetzte Inubit Instance soll in eine VM migriert werden. Der Datenbankserver soll ebenfalls in die VM verlagert werden.

Tätigkeiten	Zeitaufwand
Anforderungsanalyse: Wie „VM zu Kubernetes“	Wie „VM zu Kubernetes“
Prüfung Abhängigkeiten: Die meisten Informationen können auch hier vom Quell System übernommen werden. Es sollte aber nochmal eine VM spezifische Überprüfung erfolgen. Zudem: Überprüfen, welche Auswirkungen die Migration auf externe Systeme hat.	1 h
Ableitung für Beschaffung und Erstellung entsprechender Changes: Anhand der Anforderungen wird eine konkrete Liste an benötigten Infrastrukturkomponenten erstellt. Überprüfen, ob Sizing des Quellsystem übernommen oder angepasst werden muss. Anmelden der VM	3,5 h
Einrichtung der Systeme I: (Wie bei Installation)	(Wie bei Installation)
Einrichtung der Systeme II: (Wie bei Installation)	(Wie bei Installation)
Installation der Software I: (Wie bei Installation)	(Wie bei Installation)
Installation der Software II: (Wie bei Installation)	(Wie bei Installation)
Vorbereitung + Sicherung in Kubernetes: Vergleichbar „VM zu Kubernetes“	Wie „VM zu Kubernetes“
manuelle Migration: Vergleichbar „VM zu Kubernetes“	Wie „VM zu Kubernetes“
Wiederherstellung der Sicherung in Kubernetes Cluster: Vergleichbar „VM zu Kubernetes“	Wie „VM zu Kubernetes“
Inubit-Server starten: Server in VM wieder starten und Erfolg überprüfen	Wie „VM zu Kubernetes“
Externe Systeme anpassen: Systeme, die z.B. zuvor auf Cluster-interne Kommunikation gesetzt haben müssen angepasst werden.	Wie „VM zu Kubernetes“
alten Server zurückbauen: Kubernetes Pods abmelden und aus Cluster entfernen	0,25 h

Zusätzliche Anmerkungen:

Anhang V : Umfrage Infrastruktur

Infrastruktur

Erweiterung des Kubernetes Clusters um eine Node

Um mehr Workloads auf zu Cluster bereitstellen zu können soll dem Cluster eine neue Worker Node hinzugefügt werden.

Tätigkeiten	Zeitaufwand
Bereitstellung neuer VM	1-2 h
Integration Kubernetes-Cluster	0,5-1 h

Erweiterung einer Kubernetes Node um Arbeitsspeicher und CPU

Um mehr Workloads auf zu Cluster bereitstellen zu können soll einer bereits existierenden Worker Node zusätzlicher Arbeitsspeicher und CPU-Kerne zugeordnet werden.

Tätigkeiten	Zeitaufwand
Anpassung der VM	0,5 h

Allgemeiner jährlicher Verwaltungsaufwand für den Kubernetes Cluster

Neben der Verwaltung der Services auf dem Cluster fällt auch Aufwand für die Verwaltung des Clusters an sich an. z. B. für das Updaten der Kubernetes Versionen usw.

Tätigkeiten	Zeitaufwand
Wartungswochenende (4 x im Jahr) = Cluster Patches und Updates von „Kernkomponenten“ etc.	4 x 1-2 h
Sonstige anfallenden Problembehebungsmaßnahmen Kubernetes-Cluster	4 h

Erweiterung des Docker Hosts um Arbeitsspeicher und CPU

Um mehr Workloads auf dem Host bereitstellen zu können soll zusätzlicher Arbeitsspeicher und CPU-Kerne zugeordnet werden.

Tätigkeiten	Zeitaufwand
Anpassung der VM	0,5 h

Allgemeiner jährlicher Verwaltungsaufwand für den Docker Host

Neben der Verwaltung der Services auf dem Docker Host fällt auch Aufwand für die Verwaltung des Clusters an sich an. z. B. für das Updaten der Kubernetes Versionen usw.

Tätigkeiten	Zeitaufwand
Wartungswochenende (4 x im Jahr) = Patches und Updates etc. (hier bisher nur Host ohne Anwendungen / Container, diese liegen in Verantwortung der Nutzer)	4 x 0,5 h

Zusätzliche Anmerkungen:

Der von uns bereitgestellte Docker-Host ist im Grunde nur eine VM mit Docker-Installation. Der Wartungsaufwand ist hierbei für uns sehr gering, weil sich das Patchen nicht von anderen VMs unterscheidet. Bei Kubernetes-Clustern ist der Aufwand von unserer Seite größer, weil neben dem Patchen des Clusters (im Clusterverbund) noch zusätzliche Komponenten wie Ingress-Controller, Storage-Provisioner, Monitoring, etc. in unserer Verantwortung liegen.

Anhang W: Fragenkatalog

Fragenkatalog

Fragen 28.11.2022

1. **Frage:**
   Welche Virtualisierungslösungen werden für den Betrieb von IT-Services bei VMs genutzt?

   Antwort:
   Neben ESXi wird z. B. für den Kubernetes Cluster und die Docker Hosts, die auf Ubuntu laufen, Ganeti/KVM genutzt.

2. **Frage:**
   Welche Tools werden für das Erstellen von Backups der Festplatte bei virtuellen Maschinen eingesetzt?

   Antwort:
   Soweit ich weiß, kommt bei VMs ausschließlich Commvault zum Einsatz.

3. **Frage:**
   Welche Tools werden für das Erstellen von Backups beim Docker Host genutzt?

   Antwort:
   Auch beim Docker Host gibt es eine tägliche Sicherung der verwendeten virtuellen Festplatte. Bei Datenbanken ist das Sichern den Betreibern der Datenbanken überlassen. GISA bietet derzeit keine „gemanagte“ Datenbank auf Docker oder Kubernetes an.

4. **Frage:**
   Wie viele Control Plane (Master) und Worker Nodes besitzt der Kubernetes Cluster zurzeit?

   Antwort:
   Der DMC Kubernetes Cluster besitzt, wie man mit der Konsolenanwendung auch auslesen kann, zurzeit zwei Worker und eine Control Plane Node. Aufgrund der einzelnen Control Plane Node ist es also kein wirklicher hochverfügbarer Cluster. Es existieren allerding auch Multi-Master-Cluster in der GISA.

5. **Frage:**
   Welche Tools werden im Kubernetes Umfeld für das Erstellen von Backups genutzt?

   Antwort:
   Wir bieten derzeit keine Form von Backups einzelner PersistentVolumes oder Datenbanken im Kubernetes-Cluster an. Es erfolgt allerdings eine tägliche Sicherung der von Kubernetes verwendeten virtuellen Festplatte, vFiler und der etcd-Datenbank. Derzeit evaluieren wir jedoch den Einsatz von Commvault oder Velero. Die Kubernetes Funktion VolumeSnapshot können wir nicht einsetzen, da der von uns eingesetzte Storage-Driver das CSI nicht unterstützt.

   Nachtrag vom 27.01.2023: Die Einführung von Commvault scheint mittlerweile am wahrscheinlichsten. Des Weiteren haben wir derzeit den Storage-Driver Trident auf einem Test-Cluster im Einsatz, weil hiermit unter anderem Netapp mit iSCSI genutzt werden kann und auch das CSI bedient wird, wodurch auch Snapshots von Volumes möglich sind. Auf dem DMC-System haben wir es nicht installiert.

6. **Frage:**
   Für Netzwerksegmentierung werden bei Kubernetes zusätzliche Plugins eingesetzt, welches soll im DMC-Cluster eingesetzt werden?

   Antwort:
   Für die Netzwerksegmentierung soll das Netzwerkplugin Calico genutzt werden, welches es ermöglicht Namespaces durch entsprechende Network Policies zu trennen.

Fragen 12.12.2022

1. **Frage:**
   Welche Rolle spielen die Entwickler beim Thema Sicherheit und Backups bei IT-Services, die in einer VMs betrieben werden?

   Antwort:
   Das meiste hinsichtlich der VM Verwaltung wird von den Server Teams übernommen und wir müssen dann entsprechende Change-Request stellen. Als Entwickler und Betreuer der IT-Services sind wir aber dafür verantwortlich, welche Software mit welchen Berechtigungen in den VMs laufen und wie aktuell diese Software ist bzw. allgemein, die Einstellungen, die direkt im Betriebssystem vorgenommen werden. Bei den Datenbanken kümmert sich das DB-Team um das Aufsetzen, Betreuen und regelmäßige Sichern der Datenbanken. Für zusätzliche Updates z. B., bevor etwas getestet oder umgestellt wird, erstellen wir aber auch selbst Backups mit Tools wie mysqldump. Bei einigen Datenbanken kommt zudem inubit zum Einsatz, welches per Scheduler regelmäßig Updates mit mysqldump erzeugt.

2. **Frage:**
   Welches Monitoring wird eingesetzt, um VMs zu überwachen?

   Antwort:
   Die Server Teams nutzen unter anderem VMware NOC zum Überwachen der VMs

3. **Frage:**
   Welche Datenbanken setzen wir im DMC-Umfeld für unsere IT-Services hauptsächlich ein?

   Antwort:
   Üblicherweise nutzen wir eine Maria DB oder Oracle Database

4. **Frage:**
   Wie erfolgt das Backup von Datenbanken in Docker und Kubernetes?

   Antwort:
   Anders als bei den VMs bietet das DB-Team keine gemanagten Datenbanken an. D. h., dass wir selbst für den Betrieb und das Erstellen der Backups verantwortlich sind. Meist erfolgt das erneut über inubit, es gibt aber auch Container Images z. B. für mysql dump, oder eine Kombination diesen beiden.

5. **Frage:**
   Für einen hochverfügbaren Cluster bietet Docker die Lösung Docker Swarm, soll diese auch in der Arbeit betrachtet werden?

   Antwort:
   Nein, in der GISA existiert zurzeit keine Docker Swarm Instanz und es ist auch nicht geplant eine solche Instanz in Zukunft aufzubauen.

Fragen 07.02.2023

1. **Frage:**
   Bei Docker und Kubernetes fallen zusätzliche Kosten für die Ressourcen und die Verwaltung der Infrastruktur an. Diese verteilen sich allerdings auf alle IT-Services, die auf dem Docker Host bzw. im Kuberntes Cluster betrieben werden.
   Mit wie vielen IT-Services kann je Docker Host bzw. Kubernetes Cluster gerechnet werden.

   Antwort:
   Wie viele IT-Services langfristig auf dem Docker Host bzw. in Kubernetes laufen könnten, lässt sich nicht genau sagen, und ist z. B. auch vom Ergebnis der Arbeit abhängig. Aber 25 Services ist ein realistischer Richtwert.

2. **Frage:**
   Was ist das Wartungswochenende in Bezug auf IT-Services?

   Antwort:
   Wenn Systeme aktualisiert werden, müssen sie üblicherweise heruntergefahren werden oder es besteht die Gefahr, dass nach einem Update Fehler auftreten. Das alles kann dazu führen, dass ein IT-Service kurzzeitig nicht verfügbar ist. Deshalb gibt es in der GISA viermal im Jahr ein sogenanntes Wartungswochenende, an dem die meisten Systeme gepatscht werden. Dadurch werden die Ausfälle planbar, sie fallen alle aufeinander, d.h. sie sind insgesamt möglichst kurz und die Nichtverfügbarkeit fällt in einen Zeitraum, in dem die Services möglichst wenig gefragt sind.

Fragen 17.02.2023

1. **Frage:**
   Wie hoch ist der Arbeitsspeicher- und CPU-Auslastung des Docker Hosts, ohne die betriebenen IT-Services?

   Antwort:
   Der Docker Host an sich, so wie wir ihn in der GISA aufbauen, verbraucht ca. 500 MiB Arbeitsspeicher und minimal CPU.

Anhang X : Datenbankprodukte – GISA Wiki

Seiten / DB@GISA Startseite

Produkte

Erstellt von [redacted]

Vorwort

Die hier aufgeführten Eckpunkte sind unsere gesetzten Standards.
Abweichungen davon (z. B. andere Betriebssysteme, zusätzliche Features wie HV, spezielle Betreuungsleistungen) sind möglich, müssen aber in jeden Fall mit uns (B-[redacted]) abgesprochen werden.

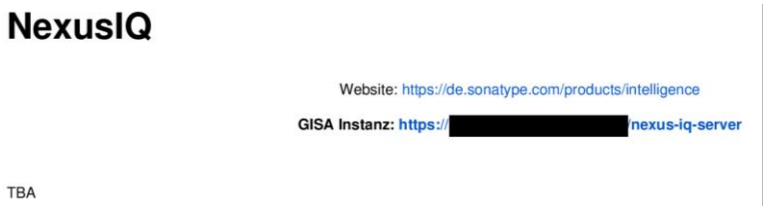
* KEF = Kundeneinzelfertigung (kein Standardprodukt)

MariaDB / MySQL / PostgreSQL

	MariaDB	MySQL	PostgreSQL
Lokation	RZ der GISA	RZ der GISA	RZ der GISA
Betriebssystem	SUSE Linux Enterprise Server (SLES) oder Ubuntu (aktuell unterstützte Version)	SUSE Linux Enterprise Server (SLES) oder Ubuntu (aktuell unterstützte Version)	Ubuntu (aktuell unterstützte Version)
Backup-Verfahren	Backups über Commvault/Simpana (1x inkrementelle Sicherung am Tag und 1x volle Sicherung in der Woche)	Backups über Commvault/Simpana (1x inkrementelle Sicherung am Tag und 1x volle Sicherung in der Woche)	Backups über Commvault/Simpana (2x inkrementelle Sicherungen am Tag und 1x volle Sicherung in der Woche)
Restore-Verfahren	Point in Time Recovery (Commvault/Simpana)	Point in Time Recovery (Commvault/Simpana)	Point in Time Recovery (Commvault/Simpana)
Monitoring	Monitoring über Icinga (NOC). Zusätzlich internes Monitoring über Prometheus (Erreichbarkeit, Anzahl Datenbankverbindungen und Auslastung Speicherplatz).	Monitoring über Icinga (NOC). Zusätzlich internes Monitoring über Prometheus (Erreichbarkeit, Anzahl Datenbankverbindungen und Auslastung Speicherplatz).	Monitoring über Icinga (NOC). Zusätzlich internes Monitoring über Prometheus (Erreichbarkeit, Anzahl Datenbankverbindungen und Auslastung Speicherplatz).
Vorraussetzungen	• eine Datenbank pro Server (VMWare / Ganeti) • Betrieb der DB nur in Binary Log Modus (Transaktionslog)	• eine Datenbank pro Server (VMWare) • Betrieb der DB nur in Binary Log Modus (Transaktionslog)	• eine Datenbank pro Server (VMWare / Ganeti)
Patchzyklus	• 4x im Jahr im Rahmen der Patchdays (aktuelle Patch-Sets) • individuelle Patchfenster möglich (4h Downtime)	• 4x im Jahr im Rahmen der Patchdays (aktuelle Patch-Sets) • individuelle Patchfenster möglich (4h Downtime)	• 4x im Jahr im Rahmen der Patchdays (aktuelle Patches) • individuelle Patchfenster möglich (4h Downtime)
enthaltene Betreuungsleistungen	• Aufbau neuer Datenbanksysteme nach Kundenwunsch • Monitoring und Alerting in Fehlerfällen • Selbstständige Behebung von betriebsbeeinträchtigen Störungen • Sicherstellen der physikalischen Datenbankkonsistenz • Nutzeradministration (Anlegen, Sperren, Passwort zurücksetzen, Rollen und System-Berechtigungen vergeben) • Technische Prüfung, Freigabe und Einspielung von Patches und Hotfixes • Restore bei Service-Ausfall durch Hardware oder Dienstleister • Einhaltung von Sicherheitsrichtlinien		
optionale Betreuungsleistungen	• Verwalten der DB-Inhalte • Planen und Durchführen von Upgrades (neues Major-Release) • Restore auf Kundenwunsch • Performance Analyse und Tuning • Fehleranalyse auf Anwendungsebene • Beratung, Planung und Aufbau von HA- und HV-Lösungen • Mitwirkung in Projekten		
Einsatzgebiet	***Standard DB, welche für die meisten Anwendungen geeignet bzw. ausreichend ist.***	***Standard DB, welche für die meisten Anwendungen geeignet bzw. ausreichend ist.***	**OpenSource DB, welche den Fokus auf Transaktionssicherheit legt. Für die meisten Anwendungen geeignet.** **Als Ersatz für eine Oracle Datenbank denkbar.**

Abbildung 32: Ausschnitt des Eintrag im GISA Wiki über die angebotenen Datenbank-produkte

Anhang Y : NexusIQ-Server – GISA Wiki


NexusIQ

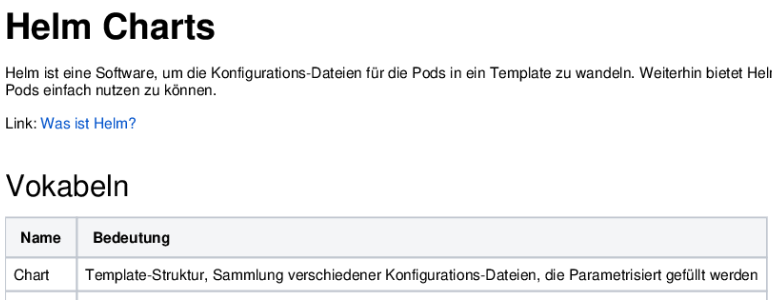
Website: https://de.sonatype.com/products/intelligence

GISA Instanz: https://[redacted]nexus-iq-server

TBA

Abbildung 33: Eintrag im GISA Wiki mit einem Link zum GISA eigenen NexusIQ Server

Anhang Z : Helm Charts– GISA Wiki


Helm Charts

Helm ist eine Software, um die Konfigurations-Dateien für die Pods in ein Template zu wandeln. Weiterhin bietet Helm ein eigenes Repository um fertige Pods einfach nutzen zu können.

Link: Was ist Helm?

Vokabeln

Name	Bedeutung
Chart	Template-Struktur, Sammlung verschiedener Konfigurations-Dateien, die Parametrisiert gefüllt werden

Abbildung 34: Ausschnitt aus einem Eintrag im GISA Wiki über Helm Chats

Anhang AA : Jenkins – GISA Wiki


Jenkins

Website: https://www.jenkins.io/

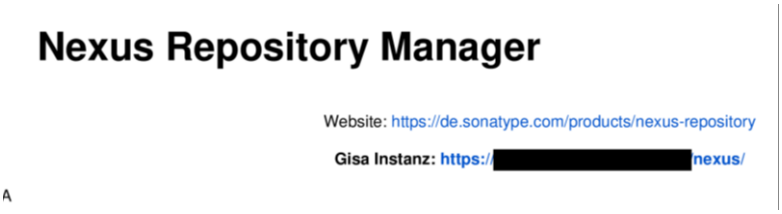
Github: https://github.com/jenkinsci/jenkins

GISA Instanz: https://[redacted]/jenkins/

TBA

Abbildung 35: Eintrag im GISA Wiki mit einem Link zum GISA eigenen Jenkins Server

Anhang BB : Nexus Repository Manager – GISA Wiki


Nexus Repository Manager

Website: https://de.sonatype.com/products/nexus-repository

Gisa Instanz: https://[redacted]nexus/

A

Abbildung 36: Eintrag im GISA Wiki mit einem Link zum GISA eigenen Nexus Repository Manager

Glossar

Cluster	Ein Cluster ist eine Gruppe von Servern, die zusammengefasst werden, um gemeinsam eine Aufgabe oder eine Anwendung auszuführen. Clustering wird verwendet, um die Verfügbarkeit und Leistung von Systemen zu erhöhen, indem Ausfälle von einzelnen Geräten kompensiert werden und die Last auf mehrere Geräte verteilt wird [165].
Denial-of-Service (DOS)	Eine DoS-Attacke ist ein Cyberangriff, der darauf abzielt, die Verfügbarkeit eines IT-Services oder einer Ressource zu stören. Um dies zu erreichen, kann ein Angreifer das Ziel entweder durch eine große Anzahl an Anfragen überlasten oder einen Exploit nutzen [45].
Docker-Host	Ein Docker-Host ist eine physische oder virtuelle Maschine auf dem der Docker Daemon Server ausgeführt wird. Auf dem Docker-Host werden die Docker Container auszuführt [20].
Exploit	Ein Exploit ist ein Programm, welches eine Sicherheitslücke in einem IT-Service oder einem IT-System ausnutzt, um z. B. weitern Code auf dem System auszuführen oder einen Denial-of-Service herbeizuführen [167].
Lastverteilung	Beim Lastverteilung werden die Anfragen, die an einen IT-Service gestellt werden, auf mehrere Server verteilt. Dadurch kann die die Leistung und die Ausfallsicherheit erhöht werden [165].
Node	Eine Node ist ein einzelner (virtueller) Server in einem Cluster [15, 36].
Portweiterleitung	Bei der Portweiterleitung werden die Netzwerkanfragen aus einem Netzwerk N_A, die an einer Netzwerkschnittstelle an einem bestimmten Port P_A eintreffen an einen definierten Port P_B einer anderen Netzwerkschnittstelle im Netzwerk N_B weitergeleitet [166].
Repository	Ein Repository ist ein Verzeichnis, dass zur Verwaltung verschiedenster Daten verwendet wird und meist eine Versionsverwaltung beinhaltet [168].

Literaturverzeichnis

[1] GISA GmbH, *Über GISA. IT komplett aus einer Hand. Beratung - Betreuung - Betrieb.* [Online]. Available: https://www.gisa.de/unternehmen/ (accessed: Nov. 23 2022).

[2] GISA GmbH, *Geschichte der GISA GmbH im Überblick.* [Online]. Available: https://www.gisa.de/unternehmen/geschichte/ (accessed: Nov. 23 2022).

[3] GISA GmbH, *Enterprise Information Management für Ihren Unternehmenserfolg.* [Online]. Available: https://www.gisa.de/it-loesungen-services/eim/ (accessed: Nov. 23 2022).

[4] Google Inc., *Angular - Introduction to the Angular Docs.* [Online]. Available: https://angular.io/docs (accessed: Dec. 3 2022).

[5] Virtimo AG, *INUBIT - POWERED BY VIRTIMO! - Virtimo.* [Online]. Available: https://www.virtimo.de/inubit-bpm/ (accessed: Nov. 15 2022).

[6] C. Meinel, C. Willems, S. Roschke, and M. Schnjakin, *Virtualisierung und Cloud Computing: Konzepte, Technologiestudie, Marktübersicht*. Potsdam: Univ.-Verl., 2011. Accessed: Nov. 15 2022.

[7] VMware Inc., *Was ist ESXi? | Bare-Metal-Hypervisor | ESX | VMware.* [Online]. Available: https://www.vmware.com/de/products/esxi-and-esx.html (accessed: Nov. 15 2022).

[8] VMware Inc., *vSphere Landing Page | VMware.* [Online]. Available: https://store-us.vmware.com/products/data-center-virtualization-cloud-infrastructure.html (accessed: Nov. 16 2022).

[9] N. Marshall, *Mastering VMware VSphere 6. 7*. Newark: John Wiley & Sons Incorporated, 2019. Accessed: Nov. 15 2022. [Online]. Available: https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=5553535

[10] VMware Inc., “VMware vSphere Datasheet,” 2022. [Online]. Available: https://www.vmware.com/content/dam/digitalmarketing/vmware/de/pdf/vsphere/vmw-vsphere-datasheet.pdf

[11] VMware Inc., “vSphere Virtual Machine Administration - VMware vSphere 7.0,” [Online]. Available: https://docs.vmware.com/en/VMware-vSphere/7.0/vsphere-esxi-vcenter-server-70-virtual-machine-admin-guide.pdf

[12] Falk Gaentzsch, Prof. Norbert Pohlmann, “IT-Sicherheitsbotschafter im Handwerk - Virtualisierung,” p. 25, 2015. [Online]. Available: https://www.internet-sicherheit.de/fileadmin/docs/mitarbeiter/Gaentzsch__Falk/D-Virtualisierung.pdf

[13] The QEMU Project Developers, *QEMU.* [Online]. Available: https://www.qemu.org/ (accessed: Nov. 17 2022).

[14] C. H. Michael S. Tsirkin, *Virtual I/O Device (VIRTIO) Version 1.1.* [Online]. Available: https://docs.oasis-open.org/virtio/virtio/v1.1/virtio-v1.1.html (accessed: Nov. 18 2022).

[15] Google Inc., *Ganeti 3.0 Manpages.* [Online]. Available: https://docs.ganeti.org/docs/ganeti/3.0/man/index.html (accessed: Nov. 18 2022).

[16] P. D. H. P. Reiser, N. Rakotondravony, and J. Köstler, “Mikromodul 8001: Grundlagen von Virtualisierungstechnik und Cloud Computing,” pp. 11–13. [Online]. Available: https://www.fim.uni-passau.de/fileadmin/dokumente/fakultaeten/fim/lehrstuhl/reiser/openc3s/CloudSecFor-MM-8001.pdf

[17] Docker Inc., “Docker Alternatives: Comprehensive Overview Guide | Docker,” *Docker*, 25 Jan., 2022. https://www.docker.com/products/docker-desktop/alternatives/ (accessed: Feb. 24 2023).

[18] Docker Inc., *Use the OverlayFS storage driver.* [Online]. Available: https://docs.docker.com/storage/storagedriver/overlayfs-driver/ (accessed: Nov. 21 2022).

[19] S. M. Jain, Ed., *Linux Containers and Virtualization*. Berkeley, CA: Apress, 2020. Accessed: Nov. 21 2022.

[20] Docker Inc., *Docker Documentation.* [Online]. Available: https://docs.docker.com/ (accessed: Nov. 24 2022).

[21] The OpenJS Foundation, *Node.js.* [Online]. Available: https://nodejs.org/ (accessed: Nov. 15 2022).

[22] The Node.js Docker Team, *node - Official Image | Docker Hub.* [Online]. Available: https://hub.docker.com/_/node (accessed: Nov. 23 2022).

[23] Docker Inc., *Docker Hub Container Image Library.* [Online]. Available: https://hub.docker.com/ (accessed: Nov. 24 2022).

[24] Scott Mccarty, “A Practical Introduction to Container Terminology,” 22 Feb., 2018. https://developers.redhat.com/blog/2018/02/22/container-terminology-practical-introduction (accessed: Nov. 25 2022).

[25] Daniel J Walsh, *A history of low-level Linux container runtimes.* [Online]. Available: https://opensource.com/article/18/1/history-low-level-container-runtimes (accessed: Nov. 25 2022).

[26] T. Heo, *Control Group v2 — The Linux Kernel documentation.* [Online]. Available: https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v2.html (accessed: Nov. 21 2022).

[27] R. E. Faith, *getpid(2) - Linux manual page.* [Online]. Available: https://man7.org/linux/man-pages/man2/getpid.2.html (accessed: Nov. 21 2022).

[28] S. Wendzel, "Einstieg in Linux: Linux verstehen und einsetzen, 4. Auflage," [Online]. Available: https://ipfs.io/ipfs/bafykbzacecxk5vz7xtdayahbbiejp6rrz6ns2n36p6jj5rehlpkwxkvt2mei2?filename=Steffen%20Wendzel%2C%20Johannes%20Pl%C3%B6tner%20-%20Einstieg%20in%20Linux_%20Linux%20verstehen%20und%20einsetzen%2C%204.%20Auflage-Galileo%20Press%20%282010%29.pdf

[29] M. Kerrisk and E. W. Biederman, *namespaces(7) - Linux manual page.* [Online]. Available: https://man7.org/linux/man-pages/man7/namespaces.7.html (accessed: Nov. 21 2022).

[30] S. Hallyn and M. Kerrisk, *cgroups(7) - Linux manual page.* [Online]. Available: https://man7.org/linux/man-pages/man7/cgroups.7.html (accessed: Nov. 21 2022).

[31] Microsoft Corporation, Inc., *Dokumentation zu Containern unter Windows.* [Online]. Available: https://learn.microsoft.com/de-de/virtualization/windowscontainers/ (accessed: Nov. 22 2022).

[32] Microsoft Corporation, Inc., *Hyper-V-Architektur.* [Online]. Available: https://learn.microsoft.com/de-de/windows-server/administration/performance-tuning/role/hyper-v-server/architecture (accessed: Nov. 15 2022).

[33] The YAML Project, *The Official YAML Web Site.* [Online]. Available: https://yaml.org/ (accessed: Nov. 15 2022).

[34] Redis Ltd., *Redis.* [Online]. Available: https://redis.io/ (accessed: Nov. 15 2022).

[35] The PostgreSQL Global Development Group, *PostgreSQL.* [Online]. Available: https://www.postgresql.org/ (accessed: Nov. 15 2022).

[36] The Linux Foundation, *Kubernetes Documentation.* [Online]. Available: https://kubernetes.io/docs/home/ (accessed: Nov. 24 2022).

[37] The Linux Foundation, *Linux Foundation - Decentralized innovation, built with trust.* [Online]. Available: https://www.linuxfoundation.org/ (accessed: Nov. 15 2022).

[38] The Linux Foundation, *Cloud Native Computing Foundation.* [Online]. Available: https://www.cncf.io/ (accessed: Nov. 15 2022).

[39] B. Burns, *The History of Kubernetes & the Community Behind It.* [Online]. Available: https://kubernetes.io/blog/2018/07/20/the-history-of-kubernetes-the-community-behind-it/ (accessed: Nov. 24 2022).

[40] T. Bray, “The JavaScript Object Notation (JSON) Data Interchange Format,” rfc8259, 2017. Accessed: Nov. 15 2022. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8259

[41] Fielding, et al., *Hypertext Transfer Protocol -- HTTP/1.1.* [Online]. Available: https://www.ietf.org/rfc/rfc2616.txt (accessed: Nov. 15 2022).

[42] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3,” rfc8446, 2018. Accessed: Nov. 15 2022. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8446

[43] Bundesamt für Sicherheit in der Informationstechnik, *Glossar der Cyber-Sicherheit - IT-Sicherheit.* [Online]. Available: https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Glossar-der-Cyber-Sicherheit/Functions/glossar.html?nn=522504&cms_lv2=132764 (accessed: Dec. 1 2022).

[44] Bundesamt für Sicherheit in der Informationstechnik, *Glossar der Cyber-Sicherheit - Bedrohungen.* [Online]. Available: https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Informationen-und-Empfehlungen/Glossar-der-Cyber-Sicherheit/Functions/glossar.html?nn=522504&cms_lv2=132796 (accessed: Dec. 1 2022).

[45] Bundesamt für Sicherheit in der Informationstechnik, *Denial-of-Service (DoS) und Distributed Denial-of-Service (DDoS).* [Online]. Available: https://www.bsi.bund.de/DE/Themen/Verbraucherinnen-und-Verbraucher/Cyber-Sicherheitslage/Methoden-der-Cyber-Kriminalitaet/DoS-Denial-of-Service/dos-denial-of-service_node.html (accessed: Dec. 1 2022).

[46] The MITRE Corporation, *Glossary | CVE.* [Online]. Available: https://www.cve.org/ResourcesSupport/Glossary?activeTerm=glossaryCVEID# (accessed: Dec. 2 2022).

[47] The MITRE Corporation, *Overview | CVE.* [Online]. Available: https://www.cve.org/About/Overview (accessed: Dec. 2 2022).

[48] Bundesamt für Sicherheit in der Informationstechnik, “SYS.1.6 Containerisierung,” [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/IT-GS-Kompendium_Einzel_PDFs_2022/07_SYS_IT_Systeme/SYS_1_6_Containerisierung_Edition_2022.pdf?__blob=publicationFile&v=3#download=1

[49] The kernel development community, *Seccomp BPF (SECure COMPuting with filters) — The Linux Kernel documentation.* [Online]. Available: https://www.kernel.org/doc/html/v4.18/userspace-api/seccomp_filter.html (accessed: Dec. 4 2022).

[50] Preston St. Pierre, *Securing Linux with Mandatory Access Controls - Linux.com.* [Online]. Available: https://www.linux.com/news/securing-linux-mandatory-access-controls/ (accessed: Dec. 3 2022).

[51] The kernel development community, *Linux Security Module Usage — The Linux Kernel documentation.* [Online]. Available: https://www.kernel.org/doc/html/v4.16/admin-guide/LSM/index.html (accessed: Dec. 4 2022).

[52] Chris Wright, Crispin Cowan, James Morris, Stephen Smalley, and Greg Kroah-Hartman, "Linux Security Module Framework," [Online]. Available: https://www.kernel.org/doc/ols/2002/ols2002-pages-604-617.pdf

[53] SELinuxProject, *SELinuxProject/selinux: This is the upstream repository for the Security Enhanced Linux (SELinux) userland libraries and tools. The software provided by this project complements the SELinux features integrated into the Linux kernel and is used by Linux distributions. All bugs and patches should be submitted to selinux@vger.kernel.org.* [Online]. Available: https://github.com/SELinuxProject/selinux (accessed: Dec. 3 2022).

[54] The kernel development community, *AppArmor — The Linux Kernel documentation.* [Online]. Available: https://www.kernel.org/doc/html/v4.16/admin-guide/LSM/apparmor.html (accessed: Dec. 3 2022).

[55] Canonical Ltd., *AppArmor - Ubuntu Wiki.* [Online]. Available: https://wiki.ubuntu.com/AppArmor (accessed: Dec. 4 2022).

[56] Canonical Ltd., *SELinux - Ubuntu Wiki.* [Online]. Available: https://wiki.ubuntu.com/SELinux (accessed: Dec. 4 2022).

[57] I. Red Hat, *Using SELinux Red Hat Enterprise Linux 8 | Red Hat Customer Portal.* [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html-single/using_selinux/index (accessed: Dec. 4 2022).

[58] Red Hat Customer Portal, *Does Red Hat Enterprise Linux support AppArmor? - Red Hat Customer Portal.* [Online]. Available: https://access.redhat.com/solutions/143723 (accessed: Dec. 8 2022).

[59] SUSE Software Solutions Germany GmbH, *Security Guide | About This Guide | SLES 12 SP4.* [Online]. Available: https://documentation.suse.com/sles/12-SP4/html/SLES-all/preface-security.html (accessed: Dec. 3 2022).

[60] Denis Efremov and Ilya Shchepetkov, *Runtime Verification of Linux Kernel Security Module*, 2020. [Online]. Available: https://www.researchgate.net/publication/338420458_Runtime_Verification_of_Linux_Kernel_Security_Module

[61] SUSE Software Solutions Germany GmbH, *Configuring SELinux | SLES 12 SP4.* [Online]. Available: https://documentation.suse.com/sles/12-SP4/html/SLES-all/cha-selinux.html (accessed: Dec. 3 2022).

[62] D. W. Valentin Rothberg, "SECCOMP: Effektiv und komplex," *heise online*, 24 May., 2019. https://www.heise.de/hintergrund/Podman-Linux-Container-einfach-gemacht-Teil-2-4429630.html?seite=4 (accessed: Dec. 3 2022).

[63] Zabramowska, *Security:Seccomp - Tizen Wiki.* [Online]. Available: https://wiki.tizen.org/Security:Seccomp (accessed: Dec. 3 2022).

[64] W. Zhang, P. Liu, and T. Jaeger, "Analyzing the Overhead of File Protection by Linux Security Modules," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, Virtual Event Hong Kong, 2021, pp. 393–406. Accessed: Nov. 16 2022. [Online]. Available: https://www.cse.psu.edu/~trj1/papers/asiaccs21.pdf

[65] Bundesamt für Sicherheit in der Informationstechnik, "SYS.1.5 Virtualisierung," [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Kompendium_Einzel_PDFs/07_SYS_IT_Systeme/SYS_1_5_Virtualisierung_Edition_2020.pdf?__blob=publicationFile&v=1

[66] The MITRE Corporation, *CVE-2022-31681.* [Online]. Available: https://www.cve.org/CVERecord?id=CVE-2022-31681 (accessed: Dec. 4 2022).

[67] The MITRE Corporation, *CVE-2021-22041.* [Online]. Available: https://www.cve.org/CVERecord?id=CVE-2021-22041 (accessed: Dec. 4 2022).

[68] The MITRE Corporation, *CVE-2022-31705.* [Online]. Available: https://www.cve.org/CVERecord?id=CVE-2022-31705 (accessed: Dec. 20 2022).

[69] The MITRE Corporation, *CVE-2022-1158.* [Online]. Available: https://www.cve.org/CVERecord?id=CVE-2022-1158 (accessed: Dec. 4 2022).

[70] The MITRE Corporation, *CVE-2021-3713.* [Online]. Available: https://www.cve.org/CVERecord?id=CVE-2021-3713 (accessed: Dec. 4 2022).

[71] Bundesamt für Sicherheit in der Informationstechnik, "SYS.1.3: Server unter Linux und Unix," [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Kompendium_Einzel_PDFs_2021/07_SYS_IT_Systeme/SYS_1_3_Server_unter_Linux_und_Unix_Edition_2021.pdf?__blob=publicationFile&v=3

[72] manpages.debian.org, *systemd.exec(5) — manpages-de — Debian unstable — Debian Manpages.* [Online]. Available: https://manpages.debian.org/unstable/manpages-de/systemd.exec.5.de.html (accessed: Nov. 19 2022).

[73] The MITRE Corporation, *CVE-2022-0185.* [Online]. Available: https://www.cve.org/CVERecord?id=CVE-2022-0185 (accessed: Dec. 4 2022).

[74] M. Ahuje, "CVE-2022-0185: Kubernetes Container Escape Using Linux Kernel Exploit," *CrowdStrike*, 01 Feb., 2022. https://www.crowdstrike.com/blog/cve-2022-0185-kubernetes-container-escape-using-linux-kernel-exploit/ (accessed: Dec. 4 2022).

[75] The MITRE Corporation, *CVE-2019-5736.* [Online]. Available: https://www.cve.org/CVERecord?id=CVE-2019-5736 (accessed: Dec. 4 2022).

[76] The Linux Foundation, *Runc and CVE-2019-5736.* [Online]. Available: https://kubernetes.io/blog/2019/02/11/runc-and-cve-2019-5736/ (accessed: Dec. 8 2022).

[77] The MITRE Corporation, *CVE-2022-31030.* [Online]. Available: https://www.cve.org/CVERecord?id=CVE-2022-31030 (accessed: Dec. 4 2022).

[78] The MITRE Corporation, *CVE-2022-43679.* [Online]. Available: https://www.cve.org/CVERecord?id=CVE-2022-43679 (accessed: Dec. 4 2022).

[79] Chierici, Stefano (Sysdig, Inc.), "Analysis on Docker Hub malicious images: Attacks through public container images," *Sysdig*, 23 Nov., 2022. https://sysdig.com/blog/analysis-of-supply-chain-attacks-through-public-docker-images/ (accessed: Dec. 20 2022).

[80] A. Jerbi, *Docker Hub Unauthorized Access Incident: What You Should Know.* [Online]. Available: https://blog.aquasec.com/docker-hub-incident-container-encryption (accessed: Dec. 8 2022).

[81] MelanieHom, Brasmith, v-kents, vrapolinario, and v-susbo, *Sichere Windows-Container.* [Online]. Available: https://learn.microsoft.com/de-de/virtualization/windowscontainers/manage-containers/container-security (accessed: Dec. 10 2022).

[82] Sonatype Inc., *Docker Container Analysis.* [Online]. Available: https://help.sonatype.com/iqserver/analysis/docker-container-analysis (accessed: Dec. 5 2022).

[83] The Linux Foundation, *Images.* [Online]. Available: https://kubernetes.io/docs/concepts/containers/images/#image-pull-policy (accessed: Dec. 5 2022).

[84] Docker Inc., *docker pull.* [Online]. Available: https://docs.docker.com/engine/reference/commandline/pull/ (accessed: Dec. 5 2022).

[85] The Linux Foundation, *Configure a Security Context for a Pod or Container.* [Online]. Available: https://kubernetes.io/docs/tasks/configure-pod-container/security-context/ (accessed: Dec. 5 2022).

[86] Docker Inc., *Docker Documentation: docker run.* [Online]. Available: https://docs.docker.com/engine/reference/commandline/run/ (accessed: Dec. 5 2022).

[87] Docker Inc., *Isolate containers with a user namespace.* [Online]. Available: https://docs.docker.com/engine/security/userns-remap/ (accessed: Dec. 1 2022).

[88] The Linux Foundation, *User Namespaces.* [Online]. Available: https://kubernetes.io/docs/concepts/workloads/pods/user-namespaces/ (accessed: Dec. 5 2022).

[89] Docker Inc., *Docker security.* [Online]. Available: https://docs.docker.com/engine/security/ (accessed: Dec. 8 2022).

[90] The Linux Foundation, *Pod Security Standards.* [Online]. Available: https://kubernetes.io/docs/concepts/security/pod-security-standards/ (accessed: Dec. 9 2022).

[91] Docker Inc., *docker network create.* [Online]. Available: https://docs.docker.com/engine/reference/commandline/network_create/ (accessed: Dec. 1 2022).

[92] Tigera, Inc., *About Network Policy.* [Online]. Available: https://projectcalico.docs.tigera.io/about/about-network-policy (accessed: Dec. 10 2022).

[93] Dipl.-Ing. (FH) Stefan Luber, "Was sind Active/Active und Active/Passive?," *Storage-Insider*, 12 Jun., 2020. https://www.storage-insider.de/was-sind-activeactive-und-activepassive-a-926889/ (accessed: Dec. 10 2022).

[94] VMware Inc., *About vSphere Availability.* [Online]. Available: https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.avail.doc/GUID-63F459B7-8884-4818-8872-C9753B2E0215.html (accessed: Dec. 12 2022).

[95] VMware Inc., *Network Operation Center.* [Online]. Available: https://docs.vmware.com/en/vRealize-Operations/Cloud/com.vmware.vcom.config.doc/GUID-A4DE463A-5582-4159-9FF8-33C00C2D4572.html (accessed: Dec. 19 2022).

[96] Google Inc., *Instance auto-repair — Ganeti 3.0.2 documentation.* [Online]. Available: https://docs.ganeti.org/docs/ganeti/3.0/html/design-autorepair.html (accessed: Dec. 15 2022).

[97] Zabbix LLC., *Zabbix :: The Enterprise-Class Open Source Network Monitoring Solution.* [Online]. Available: https://www.zabbix.com/ (accessed: Dec. 18 2022).

[98] F5 Inc., *Advanced Load Balancer, Web Server, & Reverse Proxy - NGINX.* [Online]. Available: https://www.nginx.com/ (accessed: Nov. 23 2022).

[99] Docker Inc., *Swarm mode overview.* [Online]. Available: https://docs.docker.com/engine/swarm/ (accessed: Dec. 13 2022).

[100] Docker Inc., *Deploy to Swarm.* [Online]. Available: https://docs.docker.com/get-started/swarm-deploy/ (accessed: Dec. 13 2022).

[101] Commvault Systems Inc., *Protecting Virtualized Workloads in the Command Center.* [Online]. Available: https://documentation.commvault.com/v11/essential/119353_protecting_virtualized_workloads_in_command_center.html (accessed: Dec. 25 2022).

[102] Commvault Systems Inc., *Databases.* [Online]. Available: https://documentation.commvault.com/v11/essential/146028_databases.html (accessed: Dec. 15 2022).

[103] MariaDB Corporation Ab, *About MariaDB Software.* [Online]. Available: https://mariadb.com/kb/en/about-mariadb-software/ (accessed: Nov. 15 2022).

[104] Oracle Corporation, *MySQL :: MySQL 8.0 Reference Manual :: 4.5.4 mysqldump — A Database Backup Program.* [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html (accessed: Dec. 15 2022).

[105] MariaDB KnowledgeBase, *mariadb-dump/mysqldump.* [Online]. Available: https://mariadb.com/kb/en/mariadb-dumpmysqldump/ (accessed: Dec. 25 2022).

[106] VMware, Inc., "Virtual Disk Development Kit Programming Guide - VMware vSphere 7.0 U1," [Online]. Available: https://vdc-download.vmware.com/vmwb-repository/dcr-public/2d04136c-09ff-4975-8477-f55168deb73e/8c5d57af-38ef-4351-a80b-18f5290024fe/vsphere-vddk-701-programming-guide.pdf

[107] Google Inc., *gnt-backup — Ganeti 3.0.2 documentation.* [Online]. Available: https://docs.ganeti.org/docs/ganeti/3.0/html/man-gnt-backup.html (accessed: Dec. 23 2022).

[108] Free Software Foundation, Inc., *tar(1) - Linux manual page.* [Online]. Available: https://man7.org/linux/man-pages/man1/tar.1.html (accessed: Nov. 15 2022).

[109] Docker Inc., *Backup, restore, or migrate data volumes.* [Online]. Available: https://docs.docker.com/storage/volumes/#backup-restore-or-migrate-data-volumes (accessed: Dec. 25 2022).

[110] Paul Vixie, Marcela Mašláňová, Colin Dean, and Tomáš Mráz, *cron(8) - Linux manual page.* [Online]. Available: https://man7.org/linux/man-pages/man8/cron.8.html (accessed: Nov. 15 2022).

[111] Databack, *databack/mysql-backup - Docker Image | Docker Hub.* [Online]. Available: https://hub.docker.com/r/databack/mysql-backup (accessed: Dec. 26 2022).

[112] Saad Ali, *Container Storage Interface (CSI) for Kubernetes GA.* [Online]. Available: https://kubernetes.io/blog/2019/01/15/container-storage-interface-ga/ (accessed: Nov. 23 2022).

[113] Velero Authors, *Velero.* [Online]. Available: https://velero.io/ (accessed: Dec. 25 2022).

[114] Velero Authors, *Velero Docs.* [Online]. Available: https://velero.io/docs/v1.9/ (accessed: Dec. 25 2022).

[115] MinIO, Inc., *MinIO | High Performance, Kubernetes Native Object Storage.* [Online]. Available: https://min.io/ (accessed: Dec. 25 2022).

[116] Paul Vixie, *crontab(5) - Linux manual page.* [Online]. Available: https://man7.org/linux/man-pages/man5/crontab.5.html (accessed: Nov. 15 2022).

[117] Commvault Systems Inc., *Protecting Kubernetes with Commvault.* [Online]. Available: https://documentation.commvault.com/v11/essential/123634_protecting_kubernetes_with_commvault.html (accessed: Dec. 21 2022).

[118] Microsoft Corporation, Inc., *Cloud-Computing-Dienste | Microsoft Azure.* [Online]. Available: https://azure.microsoft.com/de-de/ (accessed: Dec. 26 2022).

[119] Microsoft Corporation, Inc., *Dv3- und Dsv3-Serie - Azure Virtual Machines.* [Online]. Available: https://learn.microsoft.com/de-de/azure/virtual-machines/dv3-dsv3-series (accessed: Dec. 26 2022).

[120] Microsoft Corporation, Inc., *Auswählen eines Datenträgertyps für virtuelle Azure IaaS-Computer: verwaltete Datenträger - Azure Virtual Machines.* [Online]. Available: https://learn.microsoft.com/de-de/azure/virtual-machines/disks-types (accessed: Dec. 28 2022).

[121] Microsoft Corporation, Inc., *Verwaltetes Datenträgerbursting - Azure Virtual Machines.* [Online]. Available: https://learn.microsoft.com/de-de/azure/virtual-machines/disk-bursting#disk-level-bursting (accessed: Dec. 27 2022).

[122] International Business Machines Corp., *nmon for Linux | Main / HomePage.* [Online]. Available: https://nmon.sourceforge.net/pmwiki.php (accessed: Dec. 27 2022).

[123] Phoronix Media, *Phoronix Test Suite - Linux Testing & Benchmarking Platform, Automated Testing, Open-Source Benchmarking.* [Online]. Available: https://www.phoronix-test-suite.com/ (accessed: Dec. 26 2022).

[124] Phoronix Media, *OpenBenchmarking.org - Cross-Platform, Open-Source Automated Benchmarking Platform.* [Online]. Available: https://openbenchmarking.org/ (accessed: Dec. 25 2022).

[125] Phoronix Media, *phoronix-test-suite/ubuntu-pts-docker-build.sh.* [Online]. Available: https://github.com/phoronix-test-suite/phoronix-test-suite/blob/master/deploy/docker/ubuntu-pts-docker-build.sh (accessed: Dec. 25 2022).

[126] The Linux Foundation, *Reserve Compute Resources for System Daemons.* [Online]. Available: https://kubernetes.io/docs/tasks/administer-cluster/reserve-compute-resources/ (accessed: Feb. 13 2023).

[127] Kubernetes, *Deploy and Access the Kubernetes Dashboard.* [Online]. Available: https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/ (accessed: Feb. 13 2023).

[128] The Linux Foundation, *Assign CPU Resources to Containers and Pods.* [Online]. Available: https://kubernetes.io/docs/tasks/configure-pod-container/assign-cpu-resource/#cpu-units (accessed: Feb. 13 2023).

[129] Phoronix Media, *CPU Massive Test Suite Collection - OpenBenchmarking.org.* [Online]. Available: https://openbenchmarking.org/suite/pts/cpu-massive (accessed: Dec. 26 2022).

[130] Phoronix Media, *Single-Threaded Test Suite Collection - OpenBenchmarking.org.* [Online]. Available: https://openbenchmarking.org/suite/pts/single-threaded (accessed: Dec. 26 2022).

[131] Phoronix Media, *GnuPG Benchmark - OpenBenchmarking.org.* [Online]. Available: https://openbenchmarking.org/test/pts/gnupg (accessed: Dec. 26 2022).

[132] Phoronix Media, *7-Zip Compression Benchmark - OpenBenchmarking.org.* [Online]. Available: https://openbenchmarking.org/test/pts/compress-7zip (accessed: Dec. 26 2022).

[133] Phoronix Media, *NGINX Benchmark Benchmark - OpenBenchmarking.org.* [Online]. Available: https://openbenchmarking.org/test/pts/nginx (accessed: Dec. 26 2022).

[134] Phoronix Media, *MariaDB Benchmark - OpenBenchmarking.org.* [Online]. Available: https://openbenchmarking.org/test/pts/mysqlslap (accessed: Dec. 26 2022).

[135] Phoronix Media, *Disk Test Suite Test Suite Collection - OpenBenchmarking.org.* [Online]. Available: https://openbenchmarking.org/suite/pts/disk (accessed: Dec. 28 2022).

[136] Phoronix Media, *PostMark Benchmark - OpenBenchmarking.org.* [Online]. Available: https://openbenchmarking.org/test/pts/postmark (accessed: Dec. 25 2022).

[137] Alexey Kopytov, *akopytov/sysbench: Scriptable database and system performance benchmark.* [Online]. Available: https://github.com/akopytov/sysbench (accessed: Dec. 28 2022).

[138] Phoronix Media, *Networking Test Suite Test Suite Collection - OpenBenchmarking.org.* [Online]. Available: https://openbenchmarking.org/suite/pts/network (accessed: Dec. 26 2022).

[139] Phoronix Media, *Ethr Benchmark - OpenBenchmarking.org.* [Online]. Available: https://openbenchmarking.org/test/pts/ethr (accessed: Dec. 27 2022).

[140] Jon Postel, *RFC: 793 TRANSMISSION CONTROL PROTOCOL.* [Online]. Available: https://www.ietf.org/rfc/rfc0793.txt (accessed: Dec. 9 2022).

[141] Amazon Web Services, Inc., *Was ist Latenz? – Erläuterung zu Latenz – AWS.* [Online]. Available: https://aws.amazon.com/de/what-is/latency/ (accessed: Dec. 21 2022).

[142] Phoronix Media, *Common Kernel Benchmarks Test Suite Collection - OpenBenchmarking.org.* [Online]. Available: https://openbenchmarking.org/suite/pts/kernel (accessed: Dec. 27 2022).

[143] Phoronix Media, *MBW Benchmark - OpenBenchmarking.org.* [Online]. Available: https://openbenchmarking.org/test/pts/mbw (accessed: Dec. 26 2022).

[144] Phoronix Media, *IPC_benchmark Benchmark - OpenBenchmarking.org.* [Online]. Available: https://openbenchmarking.org/test/pts/ipc-benchmark (accessed: Dec. 26 2022).

[145] Microsoft Corporation, Inc., *Pricing - Linux Virtual Machines | Microsoft Azure.* [Online]. Available: https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/#a-series (accessed: Feb. 13 2023).

[146] H. Schuster, “Die aktuellen IT-Servicepreise sind da,” *IT-BUSINESS*, 02 Feb., 2022. https://www.it-business.de/die-aktuellen-it-servicepreise-sind-da-a-1090058/ (accessed: Feb. 13 2023).

[147] Virtimo AG, “inubit BPM Dokumentation Version 7.4,” [Online]. Available: https://files.virtimo.net/Userportal/Inubit/7.4/7.4.0.42/inubit-documentation-7.4.0.42-de.pdf

[148] jenkins.io, *Jenkins Handbook.* [Online]. Available: https://www.jenkins.io/doc/book/ (accessed: Dec. 3 2022).

[149] The Apache Groovy project, *The Apache Groovy programming language.* [Online]. Available: https://groovy-lang.org/ (accessed: Dec. 3 2022).

[150] Sonatype Inc., *Nexus Repository Manager – Binär- und Artefaktverwaltung | Sonatype.* [Online]. Available: https://de.sonatype.com/products/nexus-repository (accessed: Dec. 3 2022).

[151] The Linux Foundation, *Helm | Docs Home.* [Online]. Available: https://helm.sh/docs/ (accessed: Nov. 24 2022).

[152] The Linux Foundation, *Kompose - Convert your Docker Compose file to Kubernetes or OpenShift.* [Online]. Available: https://kompose.io/ (accessed: Dec. 3 2022).

[153] The Linux Foundation, *Kompose - Conversion.* [Online]. Available: https://kompose.io/conversion/ (accessed: Dec. 3 2022).

[154] the NGINX Docker Maintainers, *nginx - Official Image | Docker Hub.* [Online]. Available: https://hub.docker.com/_/nginx (accessed: Dec. 21 2022).

[155] Oracle Corporation, *Konfigurieren der Java Virtual Machine (JVM) (Sun Java Enterprise System 5 Installationshandbuch für UNIX).* [Online]. Available: https://docs.oracle.com/cd/E19528-01/820-0098/gazce/index.html (accessed: Jan. 25 2023).

[156] Institute of Electrical and Electronics Engineers, Inc, *IEEE 802.1: 802.1AC - Media Access Control (MAC) Services Definition.* [Online]. Available: https://www.ieee802.org/1/pages/802.1ac.html (accessed: Jan. 21 2023).

[157] Tigera, Inc., *Use a specific MAC address for a pod.* [Online]. Available: https://projectcalico.docs.tigera.io/networking/pod-mac-address (accessed: Jan. 29 2023).

[158] Google Inc., *Angular - Angular versioning and releases.* [Online]. Available: https://angular.io/guide/releases (accessed: Dec. 16 2022).

[159] T. Ylonen and C. Lonvick, “The Secure Shell (SSH) Transport Layer Protocol,” rfc4253, 2006. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4253

[160] Canonical, *Ubuntu Manpage: apt - Befehlszeilenschnittstelle.* [Online]. Available: https://manpages.ubuntu.com/manpages/bionic/de/man8/apt.8.html (accessed: Jan. 26 2023).

[161] *Kubernetes Turns Five: Cloud Native Goes Mainstream.* [Online]. Available: https://tanzu.vmware.com/content/analyst-reports/kubernetes-turns-five (accessed: Feb. 16 2023).

[162] Docker Inc., *Back up, restore, or migrate data volumes.* [Online]. Available: https://docs.docker.com/storage/volumes/#back-up-restore-or-migrate-data-volumes (accessed: Jan. 29 2023).

[163] Velero Authors, *Velero Docs - Examples.* [Online]. Available: https://velero.io/docs/v1.10/examples/ (accessed: Jan. 29 2023).

[164] Phoronix Media, *phoronix-test-suite/ubuntu-pts-docker-build.sh at master · phoronix-test-suite/phoronix-test-suite.* [Online]. Available: https://github.com/phoronix-test-suite/phoronix-test-suite/blob/master/deploy/docker/ubuntu-pts-docker-build.sh (accessed: Jan. 29 2023).

[165] P. H. Peter Fischer, *Lexikon der Informatik*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[166] Alexandre Becoulet, *libassh reference manual.* [Online]. Available: https://www.nongnu.org/libassh/manual/SSH_Lexicon.html (accessed: Feb. 2 2023).

[167] "Definition: Exploit," *Springer Fachmedien Wiesbaden GmbH*, 19 Feb., 2018. https://wirtschaftslexikon.gabler.de/definition/exploit-53419/version-276511 (accessed: Feb. 2 2023).

[168] Professor Dr. Frank Leymann, "Definition: Repository," *Springer Fachmedien Wiesbaden GmbH*, 19 Feb., 2018. https://wirtschaftslexikon.gabler.de/definition/repository-52691/version-275809 (accessed: Feb. 2 2023).

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, vorliegende Arbeit selbstständig und ohne Zuhilfenahme unzulässiger Hilfsmittel angefertigt zu haben. Wörtliche oder dem Sinne nach übernommene Ausführungen sind gekennzeichnet, sodass Missverständnisse über die geistige Urheberschaft ausgeschlossen sind. Diese Arbeit war bisher noch nicht Bestandteil einer Studien- oder Prüfungsleistung in gleicher oder ähnlicher Fassung.

Merseburg, den 02. März 2022

Unterschrift Vincent Pelz