h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Hochschule Darmstadt

– Fachbereich Informatik–

Evaluation von Kubernetes Persistenzlösungen

Abschlussarbeit zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

vorgelegt von

Lukas Menzel

Referent	:	Prof. Dr. Lars-Olof Burchard
Korreferent	:	Prof. Dr. Alexander del Pino

ERKLÄRUNG

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, 16. Dezember 2020

Lukas Menzel

ABSTRACT

Many companies have gained experience with Docker in recent years. With the increasing complexity of applications, Kubernetes is often used as a solution for container orchestration[1] to further expand the container infrastructure.

With the use of such a solution and the associated change into an often fast moving dynamic environment, the persistent storage of data poses a challenge.

As with the use of containers, data is only available as long as the application exists. To prevent this, Docker and Kubernetes offer a variety of ways to persistently store data.

The Bachelor's thesis gives an overview of different persistence solutions in Kubernetes and the technology behind them. Based on this analysis, the solutions are evaluated according to a catalogue of requirements and a recommendation for action is derived for an existing Kubernetes cluster. The findings are then applied to this cluster and the persistence solution is set up.

1 Organizing containers

ZUSAMMENFASSUNG

Viele Unternehmen haben in den letzten Jahren Erfahrungen mit Docker gesammelt. Mit zunehmender Komplexität der Anwendungen wird zum weiteren Ausbau der Container Infrastruktur oft Kubernetes als Lösung für die Container Orchestration[2] verwendet.

Mit der Verwendung einer solchen Lösung und dem damit einhergehenden Wechseln in ein oft schnelllebiges dynamisches Umfeld stellt die persistente Speicherung von Daten eine Herausforderung dar.

Wie auch bei der Verwendung von Containern sind Daten grundsätzlich nur solange vorhanden wie die Anwendung existiert. Um dies zu verhindern, bietet Docker und auch Kubernetes eine Vielzahl an Möglichkeiten, die Daten persistent zu speichern.

Die Bachelorthesis gibt einen Überblick über verschiedene Persistenzlösungen in Kubernetes und die Technologie hinter ihnen. Basierend auf dieser Analyse werden die Lösungen nach einem Anforderungskatalog evaluiert und eine Handlungsempfehlung für ein existierendes Kubernetes Cluster abgeleitet. Anschließend werden die Erkenntnisse auf dieses Cluster angewandt und die Persistenzlösung eingerichtet.

2 Die Organisation von Containern

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGS

ABKÜRZUNGSVERZEICHNIS

ALM Application Lifecycle Management

API Application Programming Interface

CLI Command Line Interface

CNCF Cloud Native Computing Foundation

CSI Container Storage Interface

EHA Elastic Hash Algorithm

GCE Google Compute Engine

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

LTS Long Term Support

NFS Network File System

PV Persistent Volume

PVC Persistent Volume Claim

REST Representational State Transfer

ROX ReadOnlyMany

RWO ReadWriteOnce

RWX ReadWriteMany

SC Storage Class

SPoF Single Point of Failure

Teil I

THESIS

1 EINLEITUNG

Viele Unternehmen haben in den letzten Jahren Erfahrungen mit dem 2013 veröffentlichten Docker [Bar14] gesammelt. Mithilfe der Container Technologie können Entwickler Software vorkonfigurieren und Images der funktionalen Anwendungen, inklusive aller Abhängigkeiten, erzeugen. Dabei bieten containerisierte Anwendungen ähnliche Vorteile wie virtuelle Maschinen. So sind Anwendungen in Containern sowohl voneinander isoliert als auch plattformunabhängig. Allerdings bieten Container durch ihre Beschaffenheit auch weitere Vorteile wie das Schonen von Ressourcen oder das Beschleunigen der vorher zeitintensiven manuellen Konfiguration von System und Software.

Mit zunehmender Komplexität der Anwendungen wird zum weiteren Ausbau der Container Infrastruktur, wie eine aktuelle Studie der Cloud Native Computing Foundation (CNCF) zeigt [Bar18], weitgehend Kubernetes als Lösung für die Orchestration von Containern eingesetzt. Kubernetes automatisiert das Verwalten, Skalieren und Bereitstellen von Containern verteilt auf mehreren Systemen. Hier können die mit Docker erzeugten Images ohne weiteren Aufwand verwendet werden.

Im Gegensatz zu physischen Systemen oder virtuellen Maschinen, die darauf ausgelegt sind Daten von Anwendungen persistent zu speichern, müssen bei Containern dafür Vorkehrungen getroffen werden. Während Software zwar ohne nähere Konfiguration aus dem Docker Hub oder von einer anderen Registry gestartet werden kann, ist, um den Zustand einer Anwendung zu bewahren, ein Einbinden eines persistenten Speichers zur Sicherung der Daten nötig. Ohne diese Konfiguration sind die Daten, sobald ein Container abstürzt oder gestoppt wird, verloren.

Docker und Kubernetes bieten daher inzwischen eine Vielzahl an Lösungen, um die Daten einer Anwendung persistent zu speichern. Diese basieren auf unterschiedlichen Technologien und bieten oft eine Reihe von Zusatzfunktionen, wie beispielsweise das Erstellen von Snapshots der gespeicherten Daten oder das automatisierte Erzeugen von Redundanzen, sodass die Auswahl der besten Lösung für die eigenen Anforderungen eine Herausforderung darstellt.

1.1 ZIELSETZUNG

Ziel dieser Bachelorthesis ist die Evaluation einer Speicherlösung für die persistente Datenspeicherung in einem bestehenden Kubernetes Cluster. Dafür werden zunächst die Anforderungen anhand von beispielhaften Anwendungsfällen analysiert und ein Anforderungskatalog, bestehend aus funktionalen und nichtfunktionalen Anforderungen, entworfen. Die in Kubernetes existierenden Möglichkeiten für eine persistente Datenspeicherung werden untersucht und die Anwendungen sowie die Technologie dahinter anhand des zuvor aufgestellten Anforderungskatalogs evaluiert. Abschlie-

ßend wird, basierend auf den Ergebnissen, eine Handlungsempfehlung für das Cluster erstellt und umgesetzt.

1.2 AUFBAU DER ARBEIT

Die Bachelorthesis ist in sechs Kapitel gegliedert. Das erste Kapitel bietet eine Einleitung in das Thema und erläutert die Ziele dieser Arbeit. Im zweiten Kapitel werden für ein Verständnis der Arbeit relevante Grundlagen wie Docker, Kubernetes und die persistente Datenspeicherung behandelt. Das dritte Kapitel widmet sich der Analyse der Anforderungen an die persistente Speicherung von Daten. Dafür werden für das bestehende Cluster Anwendungsfälle erarbeitet und basierend darauf die Anforderungen formuliert. Anschließend werden im vierten Kapitel, um eine Handlungsempfehlung zu verfassen, die in Kubernetes verfügbaren Lösungen evaluiert und anhand der Anforderungen bewertet. Im fünften Kapitel wird aufbauend auf den vorherigen Ergebnissen ein Konzept für die Umsetzung der Empfehlung für das aktuelle Cluster entworfen und umgesetzt. Das sechste Kapitel fasst die Ergebnisse dieser Thesis zusammen.

2 GRUNDLAGEN

Dieses Kapitel beschäftigt sich mit den für das Thema essentiellen Grundlagen. Darunter fallen neben der allgemeinen Funktion von Docker und Kubernetes sowie der Unterschiede zu virtuellen Maschinen auch Grundlagen zur persistenten Speicherung von Daten.

2.1 PERSISTENTE DATENSPEICHERUNG

Für den Betrieb von Anwendungen ist es wichtig, dass der Zustand und die generierten Daten der Anwendung langfristig gespeichert werden. Neben der Hardware, welche physisch für das Speichern der Daten zuständig ist, agiert die Software als Schnittstelle zwischen der Anwendung und den Datenträgern. Dabei bieten verschiedene Konzepte dieser Software unterschiedliche Vor- und Nachteile, so dass sie für verschiedene Anwendungsfälle geeignet sind.

2.1.1 *Die Herausforderung der Datenspeicherung*

Das persistente Speichern ist eine Herausforderung, welche bereits mit der Sicherstellung der Datensicherheit, also der Vertraulichkeit, der Integrität und der Verfügbarkeit von Daten anfängt. Darüber hinaus benötigen Anwendungen in modernen Infrastrukturen für die Reaktion auf unvorhersehbare Ereignisse wie sich schnell verändernde Anforderungen oder starke Beanspruchung der Systeme für die Datenspeicherung ein skalierbares und flexibles Konzept. Unabhängig von der Menge und der Größe der Daten wird eine Lösung benötigt, um von der Anwendung oder dem Nutzer generierte Daten wie Logdateien, Bilder, Videos usw. zu speichern.

Während bei monolithischen Anwendungen das Nutzen eines einzelnen Servers für die Speicherung von Daten oft ausreicht, benötigen verteilte Systeme und Cluster-Lösungen für ihr schnelllebiges und dynamisches Umfeld oft komplexere Lösungen. Dadurch entwickelten sich für die Datenspeicherung in diesen modernen Infrastrukturen verschiedene Möglichkeiten, die oft auf einem von drei grundlegenden Konzepten aufbauen.

2.1.2 *Konzepte der Datenspeicherung*

2.1.2.1 *File-Storage*

Ein File-Storage System speichert Daten in einer hierarchischen Struktur. Die Struktur wird ausgehend von einem Root-Verzeichnis mithilfe von Verzeichnissen aufgebaut. Um auf Dateien zuzugreifen, wird dabei aus den Dateinamen und den Verzeichnissen ein Pfad gebaut. Auf die physische Ebene eines Datenspeichers wird eine virtuelle Ebene, ein Dateisystem, aufgesetzt. Dabei gibt es viele verschiedene Dateisys-

teme, welche unterschiedliche Spezifikationen haben, und dadurch für verschiedene Anwendungsfälle geeignet sind. Durch die zusätzliche Ebene bietet dieses Konzept eine höhere Latenz als zum Beispiel Block-Storage. Daten, welche gespeichert werden, enthalten einige Metadaten wie Zeit und Datum der Erstellung oder auch die Größe der Datei. Der Speicher wird auf der Ebene des Dateisystems eingebunden und benötigt dabei bei den gängigen Betriebssystemen keine zusätzliche Software. Ein File-Storage kann durch die Nutzung von zusätzlichen Anwendungen um Funktionen wie Kompression der Daten oder das Erstellen von Sicherungen erweitert werden.

2.1.2.2 *Block-Storage*

Bei einem Block-Storage werden Daten in gleichgroßen Blöcken gespeichert. Dabei enthält ein Block neben einer eindeutigen ID, um ihn zu identifizieren, keine zusätzlichen Metadaten. Werden Metadaten benötigt, müssen Daten auf Anwendungsebene mit diesen versehen werden. Um eine Datei zu modifizieren, müssen nur die veränderten Blöcke der Datei bearbeitet werden, ohne dabei den gesamten Block neu zu schreiben. Durch seine Struktur bietet ein Block-Storage eine niedrigere Latenz als ein File-Storage oder Objekt Storage. Dadurch eignet sich diese Art von Speicher gut für Transaktionsdaten sowie Daten die häufig geändert werden. Für die Verwaltung der Blöcke wird ein Controller benötigt, welcher Schreib- und Lesevorgänge auf die Blöcke ausführt.

2.1.2.3 *Object-Storage*

Ein Object-Storage System speichert Daten als sogenannte Objekte. Ein Objekt kann dabei eine variable Größe haben und jegliche Art von Daten enthalten [MGR03]. Dabei ist es möglich, die Datei, neben der für die Identifizierung genutzten ID, mit benutzerdefinierten Metadaten zu versehen. So lassen sich auch für jedes Objekt individuell Sicherheits- oder Zugriffsrichtlinien festlegen [MGR03]. Um eine Datei zu ändern, muss das komplette Objekt geladen und anschließend die veränderte Version neu gespeichert werden. Dadurch eignet sich diese Art von Speicher für statische Daten, welche zum Beispiel über eine Hypertext Transfer Protocol (HTTP) oder Hypertext Transfer Protocol Secure (HTTPS) Schnittstelle abgerufen werden können. Dadurch bietet Objektspeicher eine sichere und plattformunabhängige Schnittstelle [MGR03]. Um einen Objektspeicher mit einem herkömmlichen Protokoll einzubinden, wird eine zusätzliche Software benötigt. Ein weiterer Vorteil von Object-Storage ist seine Ausfallsicherheit und Robustheit. Daten können über mehrere Server redundant gehalten werden und stehen so auch nach einem Ausfall eines Servers zur Verfügung.

2.2 DOCKER

Docker ist eine Open-Source Anwendung, welche es ermöglicht, Container zu erzeugen, auszuliefern und bereitzustellen. Container enthalten Anwendungen sowie alle Abhängigkeiten und Ressourcen die zur Laufzeit benötigt werden. Dabei bietet Docker selbst auch alle notwendigen Tools, um die Container zu verwalten. Docker erlaubt es, Anwendungen in einem Container zu starten und isoliert voneinander

auf demselben Hostsystem ohne Hypervisor mit Hilfe der Docker-Engine (siehe 2.2.1) zu betreiben [Bui15; Sol+07]. Dabei wird beim Starten eines Containers kein Kernel geladen, sondern der Kernel des Hosts mitgenutzt. Um die Anwendungen zu isolieren, werden im Linux Kernel integrierte Funktionen wie Cgroups und Namespaces eingesetzt [And15]. Durch die Isolation wird garantiert, dass ein Prozess innerhalb eines Containers keinen Zugriff auf Informationen über Prozesse oder die Nutzung von Ressourcen des Systems hat. Durch die Auslieferung der Software mit allen Abhängigkeiten als Container ist sie plattform- und betriebssystemunabhängig. Die einzige Voraussetzung ist somit eine Plattform, welche Docker unterstützt. Entwicklern ermöglicht ein sogenanntes Dockerfile (siehe 2.2.2), den Container sowie seine Infrastruktur zu konfigurieren.

2.2.1 *Docker Engine*

Im Mittelpunkt steht die Docker Engine, welche für das Erzeugen und Bereitstellen der Container verantwortlich ist. Die Docker Engine ist eine, auf der Client-Server Architektur basierende, Applikation, welche aus drei Komponenten besteht [Doc].

2.2.1.1 *Docker Daemon*

Der Docker Daemon (dockerd) ist ein Prozess, der verantwortlich dafür ist, Anfragen über die API zu verarbeiten. Er läuft als root auf dem System. Eine weitere seiner Aufgaben ist die Verwaltung von Docker Objekten wie Containern und Images.

2.2.1.2 *Rest API*

Die Representational State Transfer (REST) Application Programming Interface (API), welches vom Docker Daemon zur Verfügung gestellt wird, empfängt Kommandos vom Docker Client für die weitere Verarbeitung.

2.2.1.3 *Docker Client*

Der Docker Client ermöglicht es über ein Command Line Interface (CLI) und den *docker* Befehl, mit dem Docker Daemon zu kommunizieren. Sobald ein Befehl über das CLI ausgeführt wird, wird dieser vom Docker Daemon verarbeitet. Der Docker Client und der Docker Daemon müssen nicht auf demselben Host laufen.

2.2.2 *Dockerfile*

Das Dockerfile ist eine Textdatei, welche die Instruktionen für den Bau eines Docker Images enthält. Jede neue Zeile stellt eine neue Instruktion dar und erzeugt, außer bei wenigen Ausnahmen, eine neue Schicht auf dem stapelbaren Dateisystem des Images. Die erstellten Schichten werden zwischengespeichert, so dass bei einer Änderung des Dockerfiles nur die geänderte Schicht neu erstellt wird [Doc]. Für das Interpretieren der Schritte und das Bauen des Images ist die Docker Engine zuständig. Typischerweise enthält ein Dockerfile Instruktionen wie FROM, um das Betriebssystem, welches als Basis dient, auszuwählen; RUN, um Befehle in dem

Image auszuführen; VOLUME, um Datenträger einzubinden; EXPOSE, um Ports freizugeben und CMD, um den initialen Befehl des Containers festzulegen [Gra16].

```
1 FROM python:2
2
3 WORKDIR /usr/src/app
4
5 COPY . .
6
7 RUN pip install requests requests-toolbelt psycopg2
8
9 CMD ["python", "./application.py"]
```

Listing 2.1: Beispiel: Dockerfile

Listing 2.1 zeigt ein beispielhaftes Dockerfile, um eine Python[1] Anwendung auszuführen. Als Basis wird hier das offizielle Python Image mit dem Tag *2* gewählt. Danach wird das Arbeitsverzeichnis festgelegt und alle benötigten Dateien kopiert. Anschließend wird ein Befehl ausgeführt, der die für die Anwendung nötigen Pakete installiert und die Python Anwendung startet.

2.2.3 *Docker-Container und Docker-Images*

Ein Docker-Image ist eine aus einem Dockerfile generierte, schichtweise aufgebaute Vorlage für Docker-Container. In dieser Read-Only Vorlage sind alle für das Erzeugen einer Container-Instanz benötigten Daten und Abhängigkeiten enthalten.
Ein Docker-Container stellt eine von einem Image ausgeführte Instanz dar. Aus einem Image können unbegrenzt viele Instanzen gestartet werden. Gestartete Container werden von dem Docker Client verwaltet und können so in unterschiedliche Zustände versetzt werden.

2.3 KUBERNETES

Kubernetes (griech. für Steuermann) ist eine von Google entworfene Plattform für das automatisierte Verwalten, Skalieren und Bereitstellen von Anwendungen in Containern. Google verwendet schon seit mehr als zehn Jahren Container in seinen Datencentern [Bur+16]. Für die Verwaltung dieser wurde vorher die eigens entwickelte Plattform Borg und Omega genutzt. Als auch viele andere Entwickler begannen, sich für Linux-Container zu interessieren, wurde Kubernetes entwickelt [Bur+16].

Seit 2014 ist das System Open-Source und wird aktuell von der CNCF gepflegt. Eine aktuelle Umfrage der CNCF zeigt, dass Kubernetes zurzeit die führende Plattform ist und seine Konkurrenten wie Docker Swarm überholt hat [Bar18]. Ein Unterschied gegenüber anderen Lösungen ist, dass Kubernetes neben Docker auch andere Software für Container einsetzen kann, auf die in dieser Arbeit allerdings nicht weiter eingegangen wird.

1 Universelle Programmiersprache

2.3.1 *Kubernetes Architektur*

Ein Kubernetes Cluster besteht aus zwei Komponenten, welche sich in einer sogenannten Master-Slave-Architektur befinden. Die Architektur besteht aus einem Master, welcher aus einem oder mehreren Servern bestehen kann, und den von ihm verwalteten Slaves, in Kubernetes Nodes genannt (Abbildung 2.1). Die Komponenten des Clusters können dabei sowohl lokal, auf physischen Servern oder virtuellen Maschinen als auch online auf Infrastruktur von Cloud-Computing Anbietern laufen.

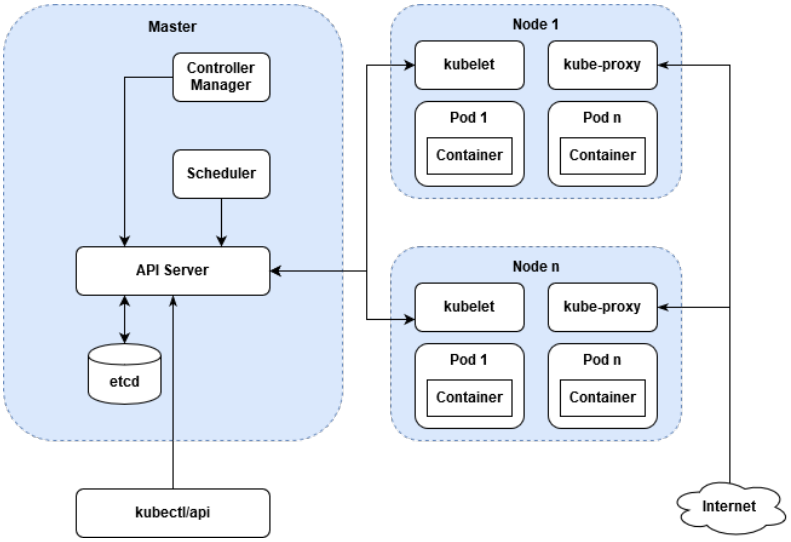

Abbildung 2.1: Kubernetes Architektur

2.3.2 *Master Node*

Der Master ist verantwortlich für die Verwaltung der Nodes im Cluster. Er leitet die Befehle vom Nutzer an die Nodes weiter. Er enthält mehrere für das Cluster wichtige Komponenten (vergleiche Abbildung 2.1). Die Verwendung von mehreren Mastern ermöglicht es, ein hochverfügbares Cluster zu planen [BW18].

2.3.2.1 *API*

Der *kube-apiserver* stellt die API zur Verfügung, über die jegliche Kommunikation abläuft. Befehle, die der API Server über die Rest-Schnittstelle oder das CLI erhält, verarbeitet er und reicht sie weiter [Ren15]. Über die API ist es möglich, Kubernetes Objekte zu erstellen, zu bearbeiten und zu löschen.

2.3.2.2 *Controller Manager*

Auf dem *kube-controller-manager* laufen die Controller. Auch wenn jeder Controller ein eigener Prozess sein könnte, werden sie, um die Komplexität zu reduzieren, in einem einzigen Prozess ausgeführt [Kuba]. Dieser läuft auf dem Master und überwacht den Status des Clusters über die API. Verändert sich der gewünschte Status des Clusters, führen sie die nötigen Operationen aus, um das Cluster in den gewollten

Status zu versetzen. Ein Beispiel hierfür wäre, wenn die Anzahl der geforderten Replicas nicht der Anzahl der existierenden entspricht. In diesem Fall würden neue Replicas gestartet werden. Der Controller Manager enthält unter anderem folgende Controller:

REPLICATION CONTROLLER Der *Replication Controller* ist dafür verantwortlich, die richtige Anzahl an Pods, welche eine logische Gruppe aus einem oder mehreren Containern darstellt (siehe 2.3.4), für jedes Replication Controller API Objekt beizubehalten.

ENDPOINT CONTROLLER Erzeugt Endpoint Objekte, welche für die Verbindung von Services und Pods genutzt werden.

NODE CONTROLLER Bei dem Node Controller handelt es sich um einen Controller, welcher die Verfügbarkeit der Nodes im Clusters überwacht. Er reagiert, sobald ein Node nicht mehr verfügbar ist.

SERVICE ACCOUNT & TOKEN CONTROLLER Dieser Controller erzeugt die Standard Accounts und verwaltet die Access Tokens [Kuba].

2.3.2.3 *Scheduler*

Um eine gleichmäßige Auslastung der Nodes zu gewährleisten, verteilt der Scheduler die Pods auf die Nodes. Dabei werden verschiedene Faktoren, wie die benötigten Ressourcen für die Anwendung und die Verfügbarkeit von Ressourcen auf dem Node bei der Auswahl der Nodes berücksichtigt. Abhängig vom Zustand des Clusters kann es auch vorkommen, dass Pods von einem Node auf einen anderen verschoben werden [Kuba].

2.3.2.4 *etcd*

Etcd[2] ist ein leichtgewichtiger, verteilter Key-Value-Store, der die Konfiguration des Clusters sowie Informationen zu dessen Zustand speichert. Andere Komponenten des Clusters können über REST auf ihn zugreifen [BW18].

2.3.2.5 *Add-ons*

Bei Add-ons handelt es sich um Pods oder Services, welche die Funktionalität von Kubernetes erweitern. Ein Beispiel für ein Add-on ist das Web UI (Dashboard) für das Verwalten des Clusters und der Anwendungen, die dort laufen.

2.3.3 *Worker Node*

Sobald eine Anwendung im Cluster gestartet wird, werden die benötigten Pods auf den Nodes verteilt. Die Nodes stellen die Laufzeitumgebung für die Pods zur Verfügung. Jeder Node nutzt dabei dieselben Komponenten [Kuba]. Die zwei wichtigsten Komponenten sind der Kube-Proxy und der kubelet [BW18].

2 https://coreos.com/etcd/

2.3.3.1 *kubelet*

Der kubelet verwaltet die Pods und Container auf den Nodes. Er antwortet auf die Befehle vom Master und setzt diese um. Typische Befehle sind zum Beispiel das Erstellen, Überwachen und Zerstören von Containern auf dem Node [Ren15]. Eine weitere Aufgabe ist, den Status der Nodes an den Master zu melden.

2.3.3.2 *Kube-Proxy*

Beim Kube-Proxy handelt es sich um einen Netzwerk Proxy, der Container und Service (siehe 2.3.5) verbindet. Zusätzlich bietet er eine Lastverteilung. Der Proxy selbst unterstützt kein http, aber beherrscht dafür TCP und UDP Weiterleitung oder Round Robin für eine Lastverteilung.

2.3.4 *Pod*

Ein Pod ist die kleinste ausführbare Einheit in einem Kubernetes Cluster und repräsentiert eine laufende Anwendung. Innerhalb eines Pods ist es möglich, einen oder mehrere Container zu starten, welche miteinander gekoppelt sind. Diese teilen sich die Ressourcen wie Speicher, die IP-Adresse im Cluster oder das Netzwerk. Daher ist es Containern in einem Pod möglich, über *localhost*[3] miteinander zu kommunizieren. Container in einem Pod werden beim Ausführen nicht über mehrere Nodes verteilt, sondern immer als eine Einheit auf einem Node ausgeführt. Der Node wird dabei durch den *Scheduler* (siehe 2.3.2.3) anhand der Auslastung der Nodes und der vom Pods angeforderten Ressourcen gewählt.

Wird eine Anwendung horizontal skaliert[4], startet ein zusätzlicher Pod, welcher eine eigenständige Instanz darstellt. Kubernetes empfiehlt, keine Pod Objekte von Hand zu erzeugen, und stattdessen ein Deployment (siehe 2.3.8.2) oder für zustandsbehaftete Anwendungen ein StatefulSet (siehe 2.3.8.3) einzusetzen [Kubb]. Ein manuell erzeugter Pod ist im Gegensatz zu von Controllern erzeugten und verwaltenden Pods nicht auf Beständigkeit ausgelegt und würde einen Absturz oder Fehler eines Nodes nicht überstehen.

2.3.5 *Services*

Ein Service sorgt dafür, dass Pods für den Nutzer erreichbar sind. Ein Service ist ein REST API Objekt und bekommt eine IP-Adresse zugeteilt. Auch jeder Pod im Cluster hat seine eigene IP-Adresse, doch kann es passieren, dass sich ein Pod zum Beispiel durch den Replica Controller ändert. Dadurch würde der Pod eine neue Adresse zugeteilt bekommen. Um dies übersichtlicher zu gestalten und für den Nutzer zu vereinfachen, fügen Services eine zusätzliche Abstraktionsebene hinzu. Ein Service beschreibt dabei eine Regel, wie ein Pod erreicht werden kann.

Auf welche Pods ein Service weiterleitet, wird meistens über einen Label Selector festgelegt. Ein Label ist ein Key-Value Paar, welches einem Kubernetes Objekt wie

3 Standardisierter Domainname des aktuellen Computers
4 Erweitern der Kapazität durch zusätzliche Hard- oder Software

einem Pod zugeordnet wird und selbst festgelegt werden kann. Über das Label ist so eine Identifikation des Objektes möglich.

```
kind: Service
apiVersion: v1
metadata:
  name: app-service
spec:
  selector:
    app: App
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

Listing 2.2: Beispiel: Kubernetes Service

In dem Beispiel in Listing 2.2 wird ein Kubernetes Service Objekt erzeugt, welches den Namen *app-service* hat. Der Service hat als Ziel Port 8080 bei jedem Pod mit dem Label *App*. Das Ergebnis der Suche nach Pods mit dem festgelegten Label wird in ein Endpoint Objekt gespeichert. Zusätzlich bekommt der Service auch noch eine IP-Adresse im Cluster. Neben Services, welche als Proxy oder Loadbalancer agieren, gibt es in Kubernetes auch noch *Headless Services*. Diese bieten die Möglichkeit, einen Pod direkt anzusprechen, und werden zum Beispiel für ein StatefulSet zwingend benötigt (siehe 2.3.8.3).

2.3.6 *Namespace*

Ein Namespace kann dafür genutzt werden, um verschiedene Bereiche auf dem Cluster, für verschiedene Projekte oder Teams abzugrenzen. Zur Identifizierung erhält der Namespace einen eindeutigen Namen. Durch Namespaces ist es ebenfalls möglich, Ressourcen zu limitieren.

2.3.7 *Ingresses*

Mit einem Ingress bietet Kubernetes eine einfache Möglichkeit, Zugriff auf Ressourcen des Clusters von außerhalb zu gewähren. Ein Kubernetes Cluster ist ein eigenes Netzwerk und isoliert grundsätzlich vor dem Zugriff von außen. In dem Ingress Objekt werden Regeln für das Routing über HTTP oder HTTPS von außerhalb sowie für die Konfiguration des Ingress gespeichert. Zusätzlich bietet er Optionen für Lastverteilung und SSL Termination.

Für den Ingress kann verschiedene Software benutzt werden. Im Kubernetes Projekt wird ein Nginx[5] Ingress Controller und ein Google Compute Engine (GCE) Ingress Controller unterstützt und gewartet.

5 Ein Open-Source Webserver, der auch als Reverse-Proxy genutzt werden kann

2.3.8 *Controller*

Die unterschiedlichen Controller bieten für das Verwalten der Pods verschiedene Funktionen, die sich für verschiedene Einsatzzwecke eignen. Ein Controller läuft dabei in einer Kontrollschleife, welche den Zustand der von ihm verwalteten Pods über die Kubernetes API überwacht. Sollte der gewünschte Zustand vom aktuellen abweichen, führt er die nötigen Schritte aus, um den Zustand zu ändern.

2.3.8.1 *ReplicaSet & ReplicationController*

Ein ReplicationController oder ReplicaSet sorgt dafür, dass mehrere Instanzen eines Pods existieren. Falls die Anzahl der angeforderten Replicas sich ändern sollte, sorgt der Controller dafür, dass weitere Replicas erzeugt oder entfernt werden. Durch diese Funktion lassen sich große Lasten verarbeiten oder eine höhere Verfügbarkeit erreichen. Eine Replica ist dabei eine Kopie eines Pods. Um die Anzahl der Replicas oder die Pods zu konfigurieren, wird ein sogenanntes Pod Template verwendet [Ren15]. Trotz des Namens ist dieser Controller auch für einzelne Pods ohne Replicas sinnvoll, da er die Anzahl der Pods überwacht und so auf mögliche Abstürze reagiert. Der bisher einzige Unterschied zwischen einem ReplicationController und dem neueren ReplicaSet ist die Art, wie sie Pods auswählen. Wird ein Controller entfernt bleiben die von ihm verwalteten Pods enthalten.

2.3.8.2 *Deployment*

Ein Deployment ist ein, in 2016 mit der Kubernetes Version 1.2 veröffentlichter Controller, der die Funktionalität eines ReplicaSets erweitert [BW18]. Für Pods die im Pod Template beschrieben sind, wird ein ReplicaSet erzeugt. Durch die zusätzliche Abstraktionsebene setzt er einige Funktionen wie zum Beispiel *Rolling Updates* um. Dabei wird ein zusätzliches ReplicaSet für die neue Version des Pods gestartet und die alte erst entfernt, wenn die neue einwandfrei läuft. Kommt es zu Fehlern, wird das Update nicht ausgeführt. Das Pod Template beschreibt, wie auch bei den anderen Controllern, einen gewünschten Zustand, den das Deployment überwacht und herstellt. Sollte ein Pod unerwartet abstürzen, wird er auch hier neu gestartet.

Das Listing 2.3 zeigt beispielhaft eine YAML-Datei, welche dafür genutzt wird, ein Deployment Objekt, welches einen Postgres Datenbank Pod enthält, zu erzeugen.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-deployment
spec:
  selector:
    matchLabels:
    app: postgres
  replicas: 2
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
      - name: postgres
        image: postgres:9.6
        ports:
        - containerPort: 5432
```

Listing 2.3: Beispiel: Kubernetes Deployment

2.3.8.3 *StatefulSet*

Das *StatefulSet* eignet sich besonders für Anwendungen, deren Zustand bewahrt werden soll. Im Gegensatz zu einem Deployment sorgt es dafür, dass Pods eine einzigartige Bezeichnung bekommen, welche über die Laufzeit hinweg bestehen bleibt. Anhand dieser Bezeichnungen werden dem StatefulSet auch die Ressourcen zugeteilt. Wird ein Pod mit denselben Spezifikationen erneut gestartet, bekommt er vom Controller dieselbe Bezeichnung und die gleichen Ressourcen zugeteilt. Darüber hinaus bieten StatefulSets die Möglichkeit, Pods geordnet zu starten. Auch der StatefulSet Controller, welcher seit Version 1.9 in einer stabilen Version verfügbar ist, versucht, wie die anderen Controller, einen gewünschten Zustand herzustellen. Sollte der Zustand nicht der gewünschte sein, werden die nötigen Schritte, um ihn zu ändern, eingeleitet. Zusätzlich benötigt ein *StatefulSet*, im Gegensatz zu einem ReplicaSet oder einem Deployment, noch einen *Headless Service* (siehe 2.3.5) für die Identifizierung im Netzwerk. Das Entfernen eines StatefulSet löscht nicht die mit den Pods verbunden Ressourcen. So kann für zustandsbehaftete Anwendungen die Datensicherheit garantiert werden.

2.3.8.4 *DaemonSet*

Der Daemonset Controller sorgt dafür, dass auf jedem Node oder einem Teil der Nodes des Clusters eine Instanz eines Pods läuft. Daher eignet sich dieser Controller besonders gut für Daemons oder Agents. Sollte ein Node vom Cluster entfernt oder

zum Cluster hinzugefügt werden, kümmert sich der Controller darum, die Anzahl der Instanzen an die Anzahl der Nodes anzupassen.

2.3.9 *Helm*

Helm ist ein Werkzeug zum Verwalten von Anwendungen in Kubernetes. In sogenannten Helm Charts wird der Zustand von Anwendungen definiert. Die Verwendung von Helm vereinfacht sowohl die Installation als auch die Wartung der Anwendungen. Dabei ist es möglich, Objekte dynamisch zu definieren, oder Werte, ohne direktes Bearbeiten der Dateien, zu überschreiben. Dadurch ist es zum Beispiel ohne Aufwand möglich, eine andere Version eines Docker Images einzubinden. Wie auch Kubernetes wird Helm von der CNCF verwaltet [Hela]. Helm bietet einen offiziellen Katalog, der Anwendungen in stabilen Charts zur Verfügung stellt und standardmäßig installiert wird [Helb]. Diese können durch den *helm install* Befehl installiert werden.

2.3.9.1 *Helm Charts*

Die sogenannten Helm Charts bestehen aus einem Verbund mehrerer Dateien, welche die Anwendung und ihren Zustand definieren. Dabei werden verschiedene Charts über die Verzeichnisstruktur abgegrenzt. Wichtige Dateien sind die *Charts.yaml*, welche allgemeine Informationen über den Chart enthält, die *requirements.yaml*, welche Informationen über Abhängigkeiten enthält und die *values.yaml*, welche für die Konfiguration der den Variablen zugeordneten Werten genutzt wird.

2.4 PERSISTENZ IN KUBERNETES

Für die persistente Speicherung von Daten kann in Kubernetes zwischen einer Vielzahl an Möglichkeiten gewählt werden. Durch die Verwendung von Volume Plugins ist es möglich, Volumes dynamisch erzeugen zu lassen oder sie bereits vorher für die Verwendung zu erzeugen.

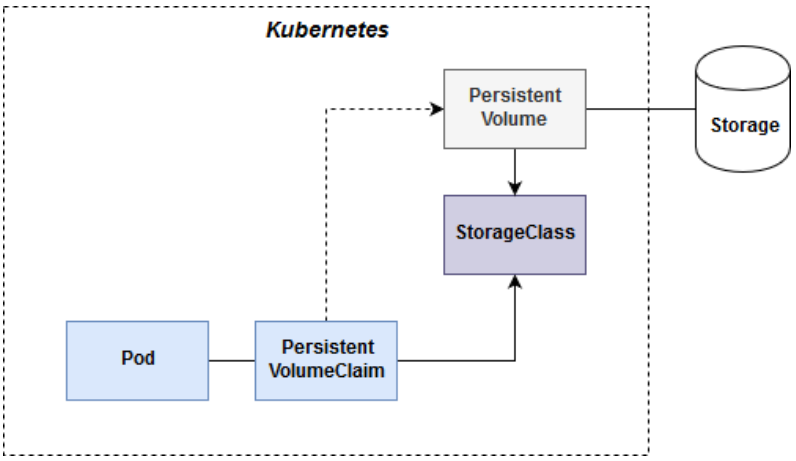

Abbildung 2.2: Zusammenhang Pod und Volume

Die Konfiguration des von Pods angeforderten Speichers geschieht direkt bei der Konfiguration des Pods über ein sogenanntes Persistent Volume Claim (PVC). Bei der statischen Bereitstellung von Volumes wird zunächst geprüft, ob ein Persistent Volume (PV) mit mindestens der geforderten Größe existiert, und das am besten passende eingebunden. Wird hier kein PV gefunden und ist die dynamische Erstellung nicht konfiguriert, wird auf die Erzeugung eines Speichers gewartet [Kubc]. Die dynamische Methode bietet die Möglichkeit, das benötigte Volume durch das in der Storage Class (SC) Provisioner genannte Volume Plugin erzeugen zu lassen.

Die Abbildung 2.2 zeigt den Ablauf der Erstellung eines PVs. Fordert ein Pod durch eine PVC Speicher an, so erstellt die SC dynamisch ein PV, welches der Konfiguration des PVC entspricht.

2.4.1 *Persistent Volumes*

Ein PersistentVolume bietet in Kubernetes die Möglichkeit, Daten persistent zu speichern. So sind Daten erneut verfügbar sollte ein Pod aufgrund eines Fehlers neu gestartet werden. Die Verwendung von PVs ermöglicht es, die Art und Weise wie der Speicher genutzt wird von der Bereitstellung abstrahiert zu betrachten. Für die Bereitstellung von PVs gibt es zwei Möglichkeiten. Bei der statischen Provisionierung ist es nötig, dass alle PVs manuell angelegt werden. Direkt nach dem Anlegen sind sie für die Nutzung verfügbar. Die zweite Möglichkeit ist das dynamische Provisionieren. Dabei wird ein neues Volume für die Anforderungen erzeugt. Dafür wird das PVC des Pods betrachtet, in dem der angeforderte Speicher konfiguriert ist. Dort ist neben der Größe des Speichers und weiteren Einstellungen auch eine SC definiert [BW18].

2.4.2 *Persistent Volume Claim*

Durch ein PVC ist es möglich, das gewünschte PV näher zu spezifizieren. Es lässt sich bereits während der Erstellung des Pods konfigurieren. Dabei enthält ein PVC Objekt einige Informationen wie einen Namen für die Identifikation, die Größe des benötigten Speichers und einige weitere Einstellungen für den Speicher (Listing 2.4).

Der Zugriffsmodus ist eine wichtige Einstellung, welche definiert, wie der Speicher eingebunden ist. Der Speicher kann nach Bedarf nur mit Leserechten oder mehrfach eingebunden werden. Dadurch lässt sich der Speicher unterschiedlichen Einsatzzwecken anpassen.

READWRITEMANY ReadWriteMany (RWX) erlaubt es, den Speicher in einem oder mehreren Pods einzubinden. Dadurch ist es möglich, dass Pods über den Speicher Daten austauschen. Schreib- sowie Lesezugriff sind möglich.

READWRITEONCE ReadWriteOnce (RWO) erlaubt es, einen Speicher in genau einem Pod einzubinden. Schreib- sowie Lesezugriff sind möglich.

READONLYMANY ReadOnlyMany (ROX) kann wie RWX mehrfach eingebunden werden, allerdings sind nur Lesezugriffe möglich. Daher eignet es sich vorallem für statische Daten, die in mehreren Pods verfügbar sein sollen.

Die nächste wichtige Option ist die Wahl der SC. Sie wird für die dynamische Erstellung von Speicher benötigt (siehe 2.4.3).

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: claim
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi
  storageClassName: sc-name
```

Listing 2.4: Beispiel: Kubernetes PersistentVolumeClaim (PVC)

Das Listing 2.4 zeigt eine beispielhafte Konfiguration eines PVC. Dabei wird von der SC *sc-name* ein Speicher mit einer Kapazität von acht GB und dem Zugriffmodus RWO angefordert.

2.4.3 *Storage Class*

Durch die Verwendung von mehreren SCs ist es möglich, verschiedene Arten von Speicher anzubieten. So kann durch den Einsatz unterschiedlicher Technologien oder der Einbindung von unterschiedlicher Hardware auf unterschiedliche Anforderungen reagiert werden.

Es ist möglich, eine SC als Standard zu benennen, welche immer wenn in einem PVC keine SC benannt wird genutzt wird. Das SC Objekt enthält Informationen wie das als Provisioner gewählte Volume-Plugin und Parameter, um ihn zu konfigurieren. Desweiteren lassen sich Regeln für eine erneute Beanspruchung von dynamisch erstellten Volumen oder zusätzliche Optionen für das Einbinden des Speichers konfigurieren. Mögliche Regeln für die erneute Beanspruchung sind Retain, Delete und Recyle.

RETAIN Retain bedeutet nach dem Löschen des PV bleiben die Daten bestehen und müssen manuell auf dem System gelöscht werden. Durch ein erneutes Erstellen des dazugehörigen PV lassen sich die Daten erneut nutzen.

DELETE Bei Delete werden durch das Löschen des dazugehörigen PV die Daten auf dem System automatisch entfernt. Bei dynamisch erstellten PVs werden die Regeln für eine erneute Beanspruchung der StorageClass verwendet. Falls in der StorageClass keine Regeln spezifiziert wurden, ist *Delete* der Standardwert.

RECYCLE Recycle ist eine veraltete Regel, welche durch das dynamische Provisionieren ersetzt wurde. Sie entfernt den Inhalt des eingebundenen PVs. Dadurch ist es bereit, erneut eingebunden zu werden.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: standard
provisioner: kubernetes.io/provisioner
reclaimPolicy: Retain
volumeBindingMode: Immediate
```

Listing 2.5: Beispiel: Kubernetes StorageClass

Das Listing 2.5 zeigt beispielhaft die Erstellung einer SC mit dem Namen *standard*. Neben der Wahl des Provisioners *kubernetes.io/provisioner* wird die Regel für die erneute Beanspruchung des Speichers auf *Retain* gesetzt.

2.4.4 *Volume Plugin System*

Ohne Hilfe durch Tools müssten von Dritten erstellte Plugins fest in Kubernetes integriert und mit Kubernetes ausgeliefert werden. Um auch ohne diese Integration die Möglichkeit zu haben eigene Volume Plugins erstellen zu können, existieren derzeit zwei Möglichkeiten. Das seit Version 1.2 existierende Flex Volume, welches eine API für externe Plugins bereitstellt, oder das seit Version 1.10 im Beta Status existierende Container Storage Interface (CSI).

2.4.4.1 *FlexVolume*

Bei FlexVolume handelt es sich um ein Tool, welches eine API für Volume Plugin von Dritten zur Verfügung stellt. Um die Speichertreiber zu nutzen, müssen die Nodes vorbereitet werden und die nötigen Daten auf das System kopiert werden. Dafür wird Zugriff auf alle Nodes des Clusters benötigt. Seit Version 1.8 erkennt FlexVolume, ohne einen Neustart der betroffenen Nodes, wenn neue Treiber hinzugefügt oder alte aktualisiert werden.

2.4.4.2 *Container Storage Interface*

Das CSI ist eine Schnittstelle für Plattformen wie Kubernetes, um die Persistenzlösungen den Containern zugänglich zu machen. Es befand sich bis Version 1.12 im Beta Status und ist seit der Version 1.13 in einer stabilen Version verfügbar.

Eine Konfiguration des CSI ist über Parameter möglich. Dort lassen sich der Treiber, der genutzt werden soll, die Art des Dateisystems und weitere Optionen konfigurieren.

3 ANFORDERUNGSANALYSE

Dieses Kapitel der Thesis untersucht die Anforderungen an die persistente Speicherung von Daten. Dafür wird zunächst das Projekt betrachtet und die zu erwartenden Anwendungsfälle analysiert. Anschließend werden darauf aufbauend die Anforderungen für die Persistenzlösung formuliert.

3.1 DAS PROJEKT

Die Idee der ALM v2.0 ist eine Application Lifecycle Management (ALM) Umgebung für die automatisierte Installation und Konfiguration eines Atlassian Stacks durch eine Container Orchestration Lösung. Ziel ist, den Prozess der Bereitstellung, welcher vorher mehrere Tage in Anspruch genommen hat, auf eine möglich kurze Zeit zu reduzieren. Der Atlassian Stack ist eine Tool-Sammlung des Softwareherstellers Atlassian, welche den Prozess der Produkt- und Softwareentwicklung begleitet.

Um dieses Ziel umzusetzen, wurden die Prozesse der Installation und Konfiguration automatisiert und die grundlegende Infrastruktur verändert. Während zuvor für jede Anwendung eine einzelne virtuelle Maschine erstellt wurde, werden in der prototypischen Umsetzung der ALM v2.0 die containerisierten Anwendungen gestartet. Dafür wurden zunächst Container erstellt, die über ein Dockerfile (siehe 2.2.2) die Anwendungen installieren und für den ersten Start konfigurieren. Basierend auf diesen Containern wurden anschließend Helm Charts (siehe 2.3.9.1) entworfen, welche neben den Anwendungen selbst noch einen vorgefertigten Datenbank-Container und einen weiteren Container für die Erstkonfiguration der Anwendung durch ein in Python entworfenes Script enthalten.

Um den Zustand der ausgeführten Anwendungen zu bewahren, werden die Anwendung und die Datenbank als StatefulSet gestartet (siehe 2.3.8.3). Neben einer eindeutigen Identifizierung der Anwendung ermöglicht dies die erneute Zuteilung des dynamisch erstellten Datenträgers. Um die Helm Charts anschließend auszuführen, steht ein Kubernetes Cluster bereit, welches verteilt auf mehreren virtuellen Maschinen läuft. Für die persistente Speicherung der Daten wird zurzeit ein ebenfalls auf einer virtuellen Maschine laufender einzelner Network File System (NFS) Server genutzt.

3.2 ANWENDUNGSFÄLLE

Bevor Anforderungen für das Projekt aufgestellt werden, muss betrachtet werden, in welchen Situationen die Lösung eingesetzt wird. Dies geschieht, indem die zu erwarteten Anwendungsfälle betrachtet werden. Für die Auswahl einer Lösung für die persistente Speicherung von Daten in einem Cluster werden dabei nicht nur Situationen während des Betriebs von Anwendungen sondern auch die Installation

und die Wartung betrachtet. Aus den beispielhaften Szenarien werden anschließend die Anforderungen extrahiert.

3.2.1 *Auswahl, Installation und Betrieb der Speicherlösung*

Für die ausgewählte Persistenzlösung steht bestehende Infrastruktur bereit. Da auf der Infrastruktur sowohl das Kubernetes Cluster als auch verschiedene andere Anwendungen betrieben werden, ist zu überprüfen, welche Ressourcen zur Verfügung stehen, und mit dem Bedarf abzugleichen. Um auf mögliche Veränderungen der Infrastruktur oder der Auslastung des Clusters zu reagieren, soll die Lösung skalierbar sein.

3.2.2 *Ausführen von Stateful Anwendungen*

Die Hauptaufgabe des Kubernetes Clusters ist das Ausführen von Anwendungen. Dabei handelt es sich aktuell größtenteils um Stateful Anwendungen. Im späteren Verlauf des Projekts sollen einige Anwendungen selbst in einem Cluster ausgeführt werden, um neue Funktionen wie das Zero-Downtime Upgrade zu nutzen. Dafür muss, um die Daten der Anwendungen synchron zu halten, zusätzlich in jeder Instanz der Anwendung der gleiche Speicher eingebunden werden. Neben den Anwendungen selbst wird darüber hinaus oft noch eine Datenbank benötigt. Beide Pods benötigen das persistente Speichern ihrer Daten, um einen stabilen Betrieb zu gewährleisten. Um die Chance auf Ausfälle gering zu halten, soll die Software bereits in einer stabilen Version vorliegen.

3.2.3 *Veränderung von Dateien auf dem Dateisystem der Pods*

Neben dem Betrieb der Anwendungen selbst, ist vor allem die Konfiguration dieser wichtig. Einige Einstellungen lassen sich nur durch das Verändern der Daten auf Dateisystemebene der Anwendung konfigurieren. Ein Nutzer, der administrativen Zugriff auf die Anwendung hat, soll, ohne administrativen Zugriff auf das Kubernetes Cluster zu benötigen, die Daten anpassen können.

3.2.4 *Sicherheit der Daten*

Die persistent gespeicherten Daten sollen auch bei Ausfällen oder Fehlern im System erhalten bleiben. Unabhängig davon durch welche Funktionen oder Technologien dieses Problem gelöst wird, muss die Reaktion des Systems zuverlässig sein.

3.3 ANFORDERUNGEN

Aufbauend auf den beispielhaften Anwendungsfällen lassen sich die Anforderungen ableiten. Diese werden näher beschrieben und in zwei Kategorien eingeteilt.

3.3.1 *Funktionale Anforderungen*

Funktionale Anforderungen sind Anforderungen, welche Funktionen oder Dienste der Persistenzlösung näher spezifizieren. Um die Anzahl an möglichen Speicherlösungen einzugrenzen, werden sie zu Beginn anhand der funktionalen Anforderungen verglichen.

3.3.1.1 *Mehrfachzugriff auf den eingebundenen Speicher*

Um dem Nutzer die Möglichkeit zu geben, die Anwendung ohne administrativen Zugriff auf das Cluster selbst zu konfigurieren, ist es nötig, ihm Zugriff zum Speicher zu gewähren. Dabei soll es sich um eine Schnittstelle handeln, auf die ohne großen Aufwand zugegriffen werden können. Eine Option wäre es, Speicher zu verwenden, die mehrfach eingebunden werden können. Dies hätte ebenfalls zur Folge, dass die Speicherlösung im späteren Verlauf des Projekts, wenn die Anwendungen im Cluster ausgeführt werden, verwendet werden kann. Die Persistenzlösung soll daher eine Lösung sein, welche mehrfache Schreib- und Lesezugriffe unterstützt, also den Zugriffsmodus RWX unterstützt (siehe 2.4.2).

3.3.1.2 *On-Premises*

Eine weitere Anforderung ist das Verwenden der bestehenden Systeme. Die Software soll daher On-Premise, also lokal, im internen Netzwerk betrieben werden. Auch wenn zurzeit eine Verbindung zum Internet besteht, soll sie auch ohne diese funktional sein.

3.3.2 *Nichtfunktionale Anforderungen*

Neben den funktionalen Anforderungen, welche sich oft aus einer Funktion oder einem Dienst ableiten, gibt es die nichtfunktionalen Anforderungen. Diese enthalten zusätzliche Voraussetzungen wie Qualitätsmerkmale oder Einschränkungen.

3.3.2.1 *Ausfallsicherheit*

Um einen fehlerfreien Betrieb zu gewährleisten, sollte die Software eine hohe Verfügbarkeit aufweisen. Durch eine hochverfügbare Architektur und das Verwenden von Redundanzen kann auf Ausfälle von einzelnen Festplatten oder ganzen Servern reagiert werden. Dabei ist darauf zu achten, dass die Redundanzen auf möglichst vielen verschiedenen Servern platziert werden, um einen Single Point of Failure (SPoF) zu vermeiden.

Die Ausfallsicherheit gilt als erfüllt, wenn die Software sowohl automatisch Redundanzen erzeugt als auch eine hochverfügbare Architektur aufweist oder andere technische Vorkehrungen für eine hohe Verfügbarkeit trifft.

3.3.2.2 *Datensicherung*

Für die Evaluation der Persistenzlösung werden hier die technischen Maßnahmen für Sicherungen der Daten betrachtet. Dafür ist zu prüfen, mit welchem Aufwand

automatische, regelmäßige Datensicherungen erstellt werden können, sowie um welche Art von Backup es sich handelt. Unterschieden werden kann hier zwischen vollständigen und inkrementellen Backups. Auch der Wiederherstellungsprozess soll einfach ausführbar sein. Neben Sicherungen können Funktionen wie Snapshots für Tests der im Cluster laufenden Anwendungen hilfreich sein, da sie ermöglichen zwischen verschiedenen Ständen der Daten zu wechseln.

Die Anforderung gilt daher als erfüllt, wenn die Speicherlösung automatisierte Sicherungen erstellen und wiederherstellen kann. Zusätzlich sollte sie ebenfalls eine Funktion zum Erstellen von Snapshots bieten.

3.3.2.3 *Datenintegrität*

Die Datenintegrität stellt die Unversehrtheit der Daten dar. Die Software soll automatisiert fehlerhafte Daten erkennen und entweder den Anwender benachrichtigen oder sie korrigieren. Dafür ist zu betrachten, welche Funktionen oder Algorithmen existieren, um korrupte Daten aufzuspüren.

3.3.2.4 *Skalierbarkeit*

Um auf Änderungen der vom Cluster benötigten Speicherkapazität oder Leistung reagieren zu können, sollte die Lösung skalierbar sein. Die Lösung sollte sich sowohl vertikal, durch das Hinzufügen von Ressourcen zu einem bestehenden Server, als auch horizontal, durch das Hinzufügen von weiteren Servern, erweitern lassen.

3.3.2.5 *Geringer Ressourcenbedarf*

Die Software soll auf einem bereits bestehenden System eingesetzt werden. Um dabei einen stabilen Betrieb des Kubernetes Clusters sowie auch anderer Anwendungen zu gewährleisten, muss auf den Ressourcenbedarf geachtet werden.

Die Speicherlösung soll, unabhängig von der Anzahl der Nodes und der Kapazität der Festplatte, vorerst nicht mehr als vier Gigabyte Arbeitsspeicher und vier Prozessorkerne benötigen.

3.3.2.6 *Benutzerfreundlichkeit*

Um die Benutzerfreundlichkeit näher zu bewerten, ist zu klären, aus welcher Perspektive diese bewertet wird. Den Nutzern der Anwendungen des Kubernetes Clusters bleibt die Speicherlösung bis auf das Einbinden des Speichers eines Pods, um Dateien über ein Protokoll wie NFS zu modifizieren, verborgen. Daher bewertet die Benutzerfreundlichkeit vor allem die Einfachheit der Installation und Wartung während des Betriebs.

In der folgenden Evaluation wird vor allem das Vorhandensein einer Dokumentation und von Scripts für eine automatische Installation und Wartung betrachtet.

3.3.2.7 *Wartbarkeit*

Neben der Einfachheit der Wartung gibt es vieles, um die Fehleranfälligkeit zu verringern und somit die Wartbarkeit zu verbessern. So bietet eine Persistenzlösung, welche

bereits lange am Markt etabliert ist, viele Erfahrungsberichte und oft auch eine gute Dokumentation. Auch kann die Nutzung einer als stabil markierten Version die Fehleranfälligkeit verringern und so für eine seltenere Wartung sorgen. Zusätzlich ist zu betrachten, ob die Lösung Open-Source ist oder von einem Unternehmen entwickelt wird. Ist eine Lösung Open-Source, so kann die Community den Quellcode einsehen und an der Entwicklung oder Korrektur von Fehlern teilhaben. Zusätzlich kann eine Schnittstelle für Monitoring die Fehlersuche erleichtern.

3.4 ZUSAMMENFASSUNG DER ANFORDERUNGEN

In der Tabelle 3.1 werden die in 3.3.1 und 3.3.2 genannten Anforderungen aufgelistet. Neben einer Nummer für die Identifizierung erhalten die nichtfunktionalen Anforderungen eine weitere Spalte mit technischen Anforderungen. Werden diese erfüllt, gilt die nichtfunktionale Anforderung ebenfalls als erfüllt.

ID	Beschreibung	
Funktionale Anforderungen		
01	On-Premises Hosting	
02	Mehrfaches Einbinden des Speichers	
Nichtfunktionale Anforderungen		Technische Anforderungen
03	Ausfallsicherheit	
		Redundanzen
		Hochverfügbarkeit
04	Datensicherung	
		Automatische Sicherung
		Fehlerfreie Wiederherstellung
		Snapshots
05	Datenintegrität	
		Fehlererkennung
		Fehlerkorrektur
06	Skalierbarkeit	
		Vertikale Skalierbarkeit
		Horizontale Skalierbarkeit
07	Geringer Ressourcenbedarf	
		Ressourcenbedarf maximal
		vier Cores und vier GB Arbeitsspeicher
08	Benutzerfreundlichkeit	
		Automatisierte Installation
		Automatisierte Wartung
09	Wartbarkeit	
		Stabile Version
		Etablierte Software
		Dokumentation
		Open-Source

Tabelle 3.1: Liste der Anforderungen

4 EVALUATION

Dieser Abschnitt widmet sich der Evaluation der in Kubernetes existierenden Persistenzlösungen. Dafür werden zuerst die in der Kubernetes Dokumentation genannten Volume Plugins näher betrachtet. Die Lösungen, welche die funktionalen Anforderungen aus 3.3.1 erfüllen, werden anschließend analysiert und danach anhand der in 3.3.2 genannten nicht-funktionalen Anforderungen bewertet. Abschließend wird eine Handlungsempfehlung basierend auf den Ergebnissen der Evaluation formuliert.

4.1 MARKTÜBERSICHT

Die Tabelle A.1 im Anhang zeigt die in der Kubernetes Dokumentation genannten Persistenzlösungen [Kubd]. Für Lösungen, welche die CSI oder Flexvolume Schnittstelle nutzen, wurde universal ein entsprechender Eintrag erstellt. Zusätzlich zu den Namen der Volume Plugins wurden Spalten für die funktionalen Anforderungen, also die Zugriffmodi und das On-Premise Hosting, erstellt. Dadurch ist bereits hier eine Eingrenzung möglich.

Volume Plugin	**On-Premises**	**RWO**	**ROX**	**RWX**
CephFS	x	x	x	x
CSI	Abhängig von Treiber			
Flexvolume	Abhängig von Treiber			
GlusterFS	x	x	x	x
Quobyte	x	x	x	x
NFS	x	x	x	x
PortworxVolume	x	x		x

Tabelle 4.1: Persistenzlösungen, welche die funktionalen Anforderungen erfüllen

Die Tabelle 4.1 zeigt die Lösungen, welche die funktionalen Anforderungen bereits erfüllen, und sich so für das existierende System eignen. Sowohl CSI als Flexvolume bieten durch das Einbinden einer eigens oder von Dritten entwickelten Lösung, die Möglichkeit, die Anforderungen zu erfüllen. Da es sich dabei allerdings um keine spezifische Speicherlösung handelt, werden diese in der Evaluation nicht näher betrachtet.

4.1.1 *NFS*

Das NFS ist ein UNIX-Protokoll, welches den Zugriff auf Dateien über das Netzwerk ermöglicht. Dabei kann der Server Teile seines eigenen Dateisystems freigeben. Auf Clientseite wird ein Pfad über einen Befehl eingebunden. Anschließend kann auf die im eingebundenen Verzeichnis existierenden Ordner und Dateien zugegriffen werden,

als wären sie eine lokal existierende Festplatte. Da es sich um ein Protokoll handelt, können die Daten von unterschiedlichen Programmen zur Verfügung gestellt werden, welche unterschiedliche Features ermöglichen.

4.1.1.1 *Einbinden in Kubernetes*

Um einen bereits existierenden NFS Server in Kubernetes einzubinden und ihn für die dynamische Erstellung von Speicher zu Nutzen, wird neben einer SC die Anwendung *nfs-client-provisioner* benötigt. Dieser kann unter Hilfe von einem gleichnamigen Helm Chart installiert werden.

```
$ helm install stable/nfs-client-provisioner --set nfs.
    server=172.168.10.100 --set nfs.path=/data/kubernetes
```

Listing 4.1: nfs-client-provisioner: Installation

Für die Installation ist nur ein Kommando notwendig (Listing 4.1). Über dieses Kommando werden zusätzliche Parameter wie die Netzwerkadresse des Servers, der Pfad der eingebunden werden soll und auch der Name der erstellten StorageClass (Listing 4.2) definiert.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-sc
provisioner: nfs-client-provisioner
parameters:
  archiveOnDelete: "false"
```

Listing 4.2: StorageClass nfs-client-provisioner

4.1.2 *GlusterFS*

Bei GlusterFS handelt es sich um eine Open-Source Persistenzlösung, welche ein skalierbares und verteiltes Dateisystem erzeugt, das es ermöglicht, Speicher von mehreren Servern als virtuelles Laufwerk zur Verfügung zu stellen [BS18]. Um ein vorhandenes System zu skalieren, kann die Speicherkapazität der einzelnen Nodes erweitert oder neue Nodes in das Cluster hinzugefügt werden. Im Gegensatz zu anderen Lösungen nutzt Gluster weder einen zentralen, noch einen verteilten Server für die Metadaten, sondern setzt den Elastic Hash Algorithm (EHA) ein [Inc11]. Über diesen Algorithmus wird der Speicherort dem Dateinamen zugeordnet [Ped+10]. Um den Zugriff auf verschiedene Redundanzen von Daten zu ermöglichen, wird ein Load-Balancing verwendet.

Gluster bietet für die virtuellen Laufwerke verschiedene Modi. Darunter befindet sich neben dem standardmäßig aktiven Verteilen über mehrere Server auch die Möglichkeit, Daten auf mehrere Server zu replizieren. Zusätzlich zu diesen Modis

bietet die Lösung auch viele weitere Funktionen, wie Snapshots, Replikationen oder Quotas, um den Speicher von virtuellen Laufwerken oder auch Ordnern zu limitieren.

4.1.2.1 *Einbinden in Kubernetes*

Für die einfache Integration von GlusterFS in Kubernetes stehen mit *gluster-kubernetes* einige Dateien und Skripte zur Verfügung [Gluc]. Dabei kann entweder ein bestehender GlusterFS-Verbund eingebunden oder in einem Kubernetes Clusters als Daemon-Set gestartet werden.

Das Listing 4.3 zeigt beispielhaft eine SC für GlusterFS. Hier wird über die Parameter der Zugriff auf das GlusterFS-Cluster konfiguriert. Dabei kann die Art der Authentifizierung sowie die Art des Speichers festgelegt werden.

```
1  kind: StorageClass
2  apiVersion: storage.k8s.io/v1
3  metadata:
4    name: gluster-sc
5  provisioner: kubernetes.io/glusterfs
6  parameters:
7    resturl: "http://172.168.10.100:8080"
8    restuser: ""
9    secretNamespace: ""
10   secretName: ""
11   restauthenabled: "false"
12   volumetype: "replicate:2"
13 allowVolumeExpansion: true
```

Listing 4.3: StorageClass GlusterFS

4.1.3 *Ceph*

Ceph ist eine skalierbare, verteilte Speicher-Lösung [Wei+06]. Sie ist Open-Source und bietet eine aktive Community und vielversprechende Entwicklung [Du+18]. Die Anwendung bietet Interfaces für drei verschiedene Arten von Speicher (siehe 2.1). Darunter mit dem Ceph Object Gateway, ein S3-Kompatibler Object-Storage, mit CephFS, ein File-Storage und mit dem RADOS Block Device (RBD), ein Block-Storage.

Ein Ceph Speicher-Cluster lässt sich in verschiedene Teile zerlegen. Für die Speicherung der Daten sind die intelligenten Object-Storage Devices (OSDs) zuständig [Wei+06]. Neben der Speicherung der Daten sind sie auch für die Migration, Replikation und Fehlerbehandlung von Daten sowie die Kommunikation untereinander zuständig. Die Metadaten werden auf den Metadata Servers (MDSs) gespeichert. Diese bearbeiten Zugriffe von Clients auf die Metadaten. Um eine Datei auf einem OSD zu finden, nutzen die MDSs, da sie den Ort der Datei nicht kennen, einen Algorithmus namens CRUSH. Dieser dient der pseudo-zufälligen Berechnung der Speicherorte von Objekten. Neben den OSDs und MDSs gibt es noch einen weiteren Daemon,

der für die Überwachung des Zustands der Nodes zuständige Monitoring Software (MON) [UHS17].

4.1.3.1 *Einbinden in Kubernetes*

Ceph bietet mit *ceph-helm* einen Helm-Chart, der das Erzeugen eines neuen Ceph Clusters auf den Nodes des Kubernetes-Clusters vereinfacht. Dabei wird über die Datei *ceph-overrides.yaml* das Ceph Cluster konfiguriert. Anschließend wird ein Namespace für Ceph erstellt und die bereits existierenden Nodes, abhängig von ihrer Aufgabe, mit den Labeln *ceph-mon=enabled*, *ceph-mgr=enabled*, *ceph-osd=enabled* und *ceph-osd-<device>=enabled* versehen.

Der Befehl in Listing 4.4 installiert schließlich basierend auf der vorherigen Konfiguration ein Ceph-Cluster.

```
$ git clone https://github.com/ceph/ceph-helm
$ helm install --name=ceph local/ceph --namespace=ceph
    -f ceph-overrides.yaml
```

Listing 4.4: Ceph: Installation

Das Listing 4.5 zeigt beispielhaft eine StorageClass für CephFS. Hier wird über die Parameter der Zugriff auf das Ceph-Cluster konfiguriert.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: cephfs-sc
provisioner: ceph.com/cephfs
parameters:
  monitors: 172.168.10.100:6789
  adminId: admin
  adminSecretName: ceph-secret-admin
  adminSecretNamespace: "kube-system"
  claimRoot: /volumes/kubernetes
```

Listing 4.5: StorageClass CephFS

4.1.4 *Portworx*

Ähnlich wie die Lösungen zuvor wird bei Portworx der vorhandene Speicher mehrerer Nodes gebündelt und zur Verfügung gestellt. Die Lösung steht dabei in zwei verschiedenen Versionen zur Verfügung. Darunter die frei verfügbare px-dev Version und die kostenpflichtige px-enterprise Version, wobei Erstere in ihrem Funktionsumfang reduziert wurde.

Portworx bietet die Möglichkeit, Funktionen wie Replikationen oder geplante regelmäßige Snapshots zu konfigurieren. Dadurch lässt sich die Persistenzlösung den Bedürfnissen anpassen und so zum Beispiel eine höhere Verfügbarkeit garantieren. Für

die Verwaltung der in Portworx erstellten Volumes gibt es das Tool pxctl, welches über die CLI sowohl die Volumes und deren Snapshots oder das Portworx Cluster managen kann. Auch ermöglicht es Zugriff auf Informationen des Speicherverbrauchs einzelner Nodes oder des gesamten Clusters.

4.1.4.1 *Einbinden in Kubernetes*

Neben den bereits aus den vorherigen Lösungen bekannten Helm Chart bietet Portworx zusätzlich die Möglichkeit, über die eigene Webseite die Konfiguration durchzuführen. Dieser Weg generiert einen Konsolen-Befehl, der für die Installation genutzt werden kann. Portworx benötigt bereits vor der Installation einen etcd Server, welcher während der Installation angegeben werden muss. Die Möglichkeit einen Built-In Server zu verwenden, befindet sich derzeit im Beta Status.

Um Portworx mit Helm zu installieren, müssen die in Listing 4.6 genannten Befehle ausgeführt werden.

```
$ git clone https://github.com/portworx/helm.git
$ helm install --debug --name portworx --set
    etcdEndPoint=etcd:http://172.168.10.100:2379,
    clusterName=$(uuidgen) ./helm/charts/portworx/
```

Listing 4.6: Portworx: Installation

Das Listing 4.7 zeigt beispielhaft eine StorageClass für Portworx. Hier lassen sich über die Parameter Einstellungen wie das Dateisystem, die Anzahl an Replikationen, eine Regelung für regelmäßige Snapshots oder auch ob ein mehrfacher Zugriff auf den Speicher möglich ist konfigurieren.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
    name: portworx-sc
provisioner: kubernetes.io/portworx-volume
parameters:
  repl: "2"
```

Listing 4.7: StorageClass Portworx

4.1.5 *Quobyte*

Quobyte ist eine Softwarelösung für ein verteiltes Speichersystem. Der Speicher mehrerer Server wird gebündelt und zusammen über verschiedene Protokolle wie NFS und S3 zur Verfügung gestellt. Auch ist es möglich, verschiedene Arten von Storage zu nutzen, wie einen Object-Storage, ein File-Storage oder auch Block-Storage zum Beispiel für Datenbanken. Das System bietet drei Services, welche zwar auf einem Node gleichzeitig laufen können, aber bei größeren Systemen oft getrennt werden. Die Registry Services, welche auf mindestens vier Nodes laufen

und die für die Registrierung von Geräten genutzt werden. Die Metadata Services, welche alle Informationen außer die Daten selbst enthalten. Dazu zählen Attribute, Berechtigungen und Speicherort der Datei. Und die Data Services, welche für das Lesen und Schreiben der Daten zuständig sind. Davon existiert ein Service pro Node.

Um auf eine Veränderung der Anforderungen an die Lösung zu reagieren, kann das System ohne Downtime horizontal über das Hinzufügen neuer Nodes skaliert werden. Durch eine Parallelisierung der Lese- und Schreibanfragen erhöht der neue Node so ebenfalls die Performance des Clusters. Über Richtlinien lassen sich das Dateilayout oder die Regeln für die Platzierung von Dateien festlegen und durch Namespace lassen sich Daten voneinander isolieren.

4.1.5.1 *Einbinden in Kubernetes*

Auch Quobyte stellt für die Verwendung in Kubernetes eine Anleitung sowie einige Dateien zur Unterstüzung zur Verfügung. Nach der Konfiguration über die Dateien kann ein Quobyte-Cluster mit wenigen Befehlen gestartet werden.

```
1  $ kubectl label node boostrap_node quobyte_registry="
       true"
2
3  $ kubectl create -f quobyte-ns.yaml
4  $ kubectl -n quobyte create -f config.yaml
5  $ kubectl -n quobyte create -f quobyte-services.yaml
6
7  $ kubectl -n quobyte create -f registry-ds.yaml
8  $ kubectl -n quobyte create -f data-ds.yaml
9  $ kubectl -n quobyte create -f metadata-ds.yaml
10
11 $ kubectl create -f webconsole-deployment.yaml
12 $ kubectl create -f qmgmt-pod.yaml
13
14 $ kubectl port-forward webconsole-pod 8080:8080
```

Listing 4.8: Quobyte: Installation

Das Listing 4.8 zeigt die Befehle. Dort wird zuerst der Node, der zu Beginn als Registry genutzt werden soll, mit einem Label versehen, und der Quobyte Namespace in Kubernetes erzeugt. Danach wird die Konfiguration geladen und die Services erzeugt. Nachdem die Services gestartet sind, werden die Webconsole und die API gestartet. Der Geräteinspektor der Webconsole ermöglicht nach dem initialen Aufsetzen das Hinzufügen von neuen Nodes. Dafür müssen diese lediglich mit einem Label entsprechend ihrerer Funktion versehen werden.

Das Listing 4.9 zeigt beispielhaft eine StorageClass für Quobyte. Über die Parameter werden Einstellungen, wie die URL der API und Registry, aber auch die Daten für den Zugriff auf das System konfiguriert.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: quobyte-sc
provisioner: kubernetes.io/quobyte
parameters:
  quobyteAPIServer: "http:/172.168.10.100:7860"
  registry: "172.168.10.100:7861"
  adminSecretName: "quobyte-admin-secret"
  adminSecretNamespace: "kube-system"
  user: "root"
  group: "root"
  quobyteConfig: "BASE"
  quobyteTenant: "DEFAULT"
  createQuota: "False"
```

Listing 4.9: StorageClass Quobyte

4.2 BEWERTUNG

In dem vorherigen Abschnitt wurden die Persistenzlösungen, welche die funktionalen Anforderungen erfüllen, näher betrachtet. Um eine Lösung zu finden, werden die Lösungen anhand der nicht-funktionalen Anforderungen verglichen und bewertet. Anschließend wird eine Handlungsempfehlung formuliert und in Kapitel 5 diese für das bestehende System praktisch umgesetzt und abschließend ein Fazit erarbeitet.

4.2.1 *NFS*

Im Gegensatz zu den anderen hier evaluierten Lösungen handelt es sich bei NFS um keine allumfassende Software, sondern lediglich um ein Protokoll, um Daten über das Netzwerk zur Verfügung zu stellen. Einige der anderen vorgestellten Lösungen können zusätzlich zu den eigenen Volume Plugins auch über NFS eingebunden werden. Standardmäßig denkt der Nutzer bei NFS an einen zentralen Server, der seinen Clients Daten zur Verfügung stellt. Allerdings lässt sich ein NFS Server durch weiterführende Konfiguration oder zusätzliche Software um viele Funktionen erweitern. Zusätzlich bietet er durch seinen Verbreitungsgrad die perfekte Grundlage für Weiterentwicklungen. Einige Forscher haben bereits das NFS Protokoll verbessert und erweitert. Ein paar Beispiele sind nfsp [LD02], IncFS [Yi+06] und FT-NFS [PM96].

4.2.1.1 *Ausfallsicherheit*

Um ein hochverfügbares NFS-Cluster zu planen, wird neben mehreren NFS-Servern zusätzliche Software benötigt. Die Software Pacemaker ist ein Open-Source Cluster-Manager für Linux, welche ein aus mehreren NFS-Servern ein hochverfügbares NFS-

Cluster machen kann. Für die Kommunikation zwischen den einzelnen Servern kann zum Beispiel die Software Corosync, welche ebenfalls eine Open-Source Anwendung ist, eingesetzt werden. Zusätzlich ermöglicht die Verwendung von der Software Distributed Replicated Block Device (DRBD) das Spiegeln eines Servers auf einen zweiten in Echtzeit. Auch wenn im NFS-Protokoll selbst die Funktionen, um die Anforderungen an die Ausfallsicherheit zu erfüllen, nicht standardmäßig integriert sind, lässt sich durch die Verwendung von freier Software diese Funktionalität ergänzen.

4.2.1.2 *Datensicherung*

Das Erstellen von Sicherungen ist bei einem NFS-Server oder NFS-Cluster möglich. Das Dateisystem oder Teile davon lassen sich durch gängige Tools, wie zum Beispiel rsync, synchronisieren. Dadurch lassen sich durch die Verwendung vorgefertigter oder selbst entworfener Skripte automatisierte Sicherungen anfertigen. Die Funktionen wie Snapshots oder auch inkrementelle Backups lassen sich durch das Verwenden geeigneter Software umsetzen. Wird ZFS als Dateisystem verwendet, ermöglicht es das Erstellen von Snapshots. Zusätzlich bietet ZFS die Möglichkeit, die Unterschiede zwischen Snapshots festzustellen, und diese nach der ersten Sicherung zu speichern.

4.2.1.3 *Datenintegrität*

Neben den Vorkehrungen, welche NFS für eine Sicherstellung der Integrität der Daten trifft, bietet sowohl das Protokoll TCP und die Verwendung von Ethernet zwei Stellen, welche Checksummen benutzen. Seit NFS v4 ist es möglich, durch die Verwendung von Kerberos im Modus krb5i jeder Transaktion, um die Integrität zu verbessern, einen Hash hinzuzufügen. Dieser Modus alleine bietet allerdings keine Ende-zu-Ende Integrität.

4.2.1.4 *Skalierbarkeit*

Auch wenn NFS oft mit einem zentralen Server verbunden wird, lässt sich dieser sowohl in der Performance als auch der Speicherkapazität skalieren. Während die Skalierung des zur Verfügung stehenden Speichers über das Vergrößern der vorhandenen Festplatten oder das Installieren neuer Festplatten gelöst werden kann, muss für eine Skalierung der Performance ein wenig mehr Aufwand betrieben werden. Eine Möglichkeit ist, dass mit NFS v4.1 veröffentlichte *pnfs*, welches NFS parallelisiert. Dabei ist ein zentraler Server nur noch für den Kontakt zwischen den Clients und den NFS-Speichern zuständig.

4.2.1.5 *Ressourcenbedarf*

Der Ressourcenbedarf eines einfachen NFS Servers ist in unseren Fall vernachlässigbar. Wird die Funktionalität durch die Konfiguration oder zusätzliche Software erweitert, so steigt der Bedarf an Ressourcen ebenfalls und muss im Einzelfall überprüft werden.

4.2.1.6 *Benutzerfreundlichkeit*

Da es sich bei NFS um ein im Linux-Betriebssystem integriertes Protokoll handelt, ist das Einbinden für Nutzer ohne großen Aufwand möglich. Auch das Entwerfen oder die Nutzung von Skripten für die automatische Installation ist möglich. Wird NFS durch Software für unsere Anforderungen erweitert, ist dies beim Entwurf der Skripte zu berücksichtigen.

4.2.1.7 *Wartbarkeit*

Von den evaluierten Speicherlösungen ist NFS die Software, welche am längsten am Markt ist. Bereits 1985 wurde NFS v2 veröffentlicht, wobei die erste Version nie öffentlich verfügbar war [Paw+94]. In der Zeit seit Veröffentlichung hat sich die Software fest am Markt etabliert, wodurch eine große Anzahl an Erfahrungsberichten und eine ausführliche Dokumentation existiert. Inzwischen liegt NFS in der Version 4 vor, welche die Software um neue Funktionen wie zum Beispiel pnfs erweitert.

4.2.1.8 *Zusammenfassung*

Mit NFS lassen sich durch die hohe Erweiterbarkeit und den vorhandenen Erfahrungen viele Anwendungsfälle abdecken. Vor allem, wenn bereits ein NFS-Server im Unternehmen zur Verfügung steht, bietet es sich an, diesen zu nutzen. Wird die Funktionalität durch mehrere externe Programme erweitert, steigt die Komplexität des Systems exponentiell. Durch eine hohe Komplexität wird unter anderem die Suche von Fehlern bei Ausfällen oder die Wartung aufwendig.

4.2.2 *GlusterFS*

GlusterFS ist eine Open-Source-Software, welche es ermöglicht ein Speichercluster bestehend aus mehreren Nodes aufzusetzen. Dabei wird der Speicher der Nodes als einheitliches Dateisystem präsentiert. Dabei können die in GlusterFS erstellten Volumes unter Linux per NFS und unter Windows über CIFS eingebunden werden.

4.2.2.1 *Ausfallsicherheit*

Der Speicher wird von GlusterFS über mehrere Nodes verteilt und abhängig von der Konfiguration als ein einzelner oder mehrere zusammenhängende Speicher zur Verfügung gestellt. Dadurch bietet das System durch den Verzicht zentralisierter Strukturen keinen SPoF. Beim Erstellen dieses Laufwerks ist es möglich, den Modus auszuwählen, und die Daten über mehrere Server zu replizieren (siehe 4.1.2). Zusätzlich bietet die Software die Möglichkeit, die Nodes in Zonen einzuteilen. Werden Redundanzen erzeugt, werden diese soweit möglich über die Zonen verteilt. So lässt sich durch die Verwendung mehrerer Replikationen ein hochverfügbares Cluster entwerfen.

4.2.2.2 *Datensicherung*

Für eine Datensicherung bietet GlusterFS ähnlich wie NFS keinen integrierten Weg. Durch die Verwendung von Skripten oder der direkten Verwendung von einem Tool

wie rsync ist es allerdings möglich, das Dateisystem vollständig zu sichern. Wird zusätzlich noch das in der Software integrierte Tool *glusterfind* benutzt, kann eine Liste der Dateien, welche sich geändert haben, erstellt werden. Mithilfe dieser Liste kann nach dem ersten vollständigen Backup eine inkrementelle Sicherung erstellt werden. Die Erstellung von Snapshot ist ohne weiteren Aufwand möglich und bereits in GlusterFS integriert.

4.2.2.3 *Datenintegrität*

Die Software bietet neben der bereits in den Protokollen integrierten Fehlererkennung Möglichkeiten, Fehler zu erkennen und zu beheben. Werden Redundanzen verwendet, sorgt der Self Heal Daemon (shd) dafür, dass sobald ein Node nach einem Fehler erneut gestartet wird, der aktuelle Stand der Daten synchronisiert wird. Zusätzlich kann manuell auch ein Full Heal ausgeführt werden, welcher den kompletten Datenbestand kopiert.

Eine weitere Möglichkeit ist die Bit Rot Detection, welche die Cheksumme jeder Datei und jedes Objektes abgleicht, um dadurch Fehler zu erkennen. Die Ergebnisse werden anschließend in einer Log-Datei gespeichert.

4.2.2.4 *Skalierbarkeit*

Um auf veränderte Anforderungen an die Persistenzlösung zu reagieren, ist es möglich, GlusterFS sowohl durch das Vergrößern oder das Hinzufügen von Festplatten als auch durch das Ergänzen um zusätzliche Nodes zu erweitern. Anschließend an das Hinzufügen muss über einen Befehl entweder ein bestehendes virtuelles Laufwerk erweitert oder ein neues erstellt werden.

4.2.2.5 *Ressourcenbedarf*

Um ein Speichercluster mit GlusterFS zu entwerfen, werden mindestens zwei Nodes benötigt. Neben dieser Anforderung wird empfohlen, bei Bare-Metal Servern zwei Gigabyte Arbeitsspeicher [Glue] und bei virtuellen Maschinen mindestens ein Gigabyte Arbeitsspeicher [Glud] zu installieren. Abhängig von der Last der Server und dem Einsatzzweck ist zu überprüfen, ob die minimalen Anforderungen ausreichend sind oder ob das System skaliert werden muss.

4.2.2.6 *Benutzerfreundlichkeit*

Gluster selbst bietet mit *gluster-kubernetes* eine Möglichkeit, GlusterFS einfach in einem bestehenden Kubernetes Cluster bereitzustellen. Dabei besteht die Möglichkeit, entweder GlusterFS innerhalb des Kubernetes Clusters aufzusetzen oder ein externes existierendes GlusterFS einzubinden. Entscheidet der Nutzer sich, GlusterFS extern auf virtuellen Maschinen oder Bare-Metal Server aufzusetzen, stehen auch hierfür Skripte zum Beispiel für Ansible zur Verfügung.

4.2.2.7 *Wartbarkeit*

Gluster wurde 2005 als Open-Source Projekt gestartet [Xia15], ist daher schon einige Zeit am Markt und hat sich inzwischen etabliert. Der Updatezyklus von GlusterFS

sieht ein Major Release alle vier Monate, sowie jeden Monat Updates für die bereits veröffentlichten Major Releases vor [Glub]. Durch die Dokumentation und die zahlreichen Erfahrungsberichte wird die Wartung vereinfacht.

4.2.2.8 *Zusammenfassung*

Im Vergleich mit NFS bietet GlusterFS ohne die Verwendung von externer Software bereits viele Funktionen, lediglich für das Erstellen von Sicherungen muss auf Skripte oder die manuelle Verwendung externer Tools ausgewichen werden. Ein weiterer Nachteil ist, dass in der Software kein WebUI für die einfache Verwaltung der Software vorhanden ist. Dadurch ist es nötig das System entweder durch Skripte oder das manuelle Ausführen von Befehlen zu verwalten oder auf eine externe Lösung wie oVirt[1] zu setzen.

Durch die verteilte Architektur und das Verwenden von Redundanzen besitzt die Speicherlösung keinen SPoF. Zudem bietet der Verzicht auf Server für Metadaten eine vereinfachte Struktur und erleichtert die Planung. Zusätzlich bietet die Open-Source-Software einen transparenten Updatezyklus und eine aktive Entwicklung.

4.2.3 *CephFS*

Ähnlich wie GlusterFS ist auch Ceph eine Open-Source-Software für das Bereitstellen von Speicher. Dabei ist die Software unabhängig von der Art der Hardware und kann auf nahezu jedem System genutzt werden [UHS17]. Ceph wird standardmäßig vom Linux-Kernel unterstützt und kann so ohne weitere Software über den mount Befehl eingebunden werden.

4.2.3.1 *Ausfallsicherheit*

Auch Ceph bietet durch seine verteilte Struktur eine hohe Ausfallsicherheit. So ist es durch Redundanzen möglich, auf Ausfälle von einzelnen Nodes ohne einen Ausfall des Systems zu reagieren. Um ein hochverfügbares System aufzubauen, wird empfohlen, eine ungerade Anzahl an MONs zu verwenden [UHS17]. Durch die ungerade Anzahl ist es möglich, dass die MONs im Quorum eine Mehrheit bilden können. Wie auch bei GlusterFS bietet sich durch die redundanten Systeme kein SPoF.

4.2.3.2 *Datensicherung*

Um ein CephFS Speicher zu erstellen, werden zwei sogenannte RADOS Pools benötigt, einen für die Metadaten und einen für die Daten. Während des Erstellens eines Pools lässt sich unter anderem die Anzahl an Redundanzen einstellen. Zudem ist es möglich, Snapshots dieser Pools zu erstellen. Snapshots vom CephFS direkt zu erstellen ist bisher nur in einer Version möglich, die sich aktuell noch in Entwicklung befindet. Ähnlich wie GlusterFS und NFS gibt es für CephFS keinen integrierten Weg um Sicherungen zu erstellen, allerdings ist es auch hier möglich manuell oder durch Skripte eine automatisierte Sicherung auf Dateisystemebene zu konfigurieren. Durch Überprüfen der Attribute der Verzeichnisse kann festgestellt werden, ob sich

1 Eine Open-Source Lösung für Virtualisierung

Daten in dem Verzeichnis geändert haben und so eine Liste der Veränderungen für ein inkrementelles Backup erstellt werden.

4.2.3.3 *Datenintegrität*

Auch Ceph integriert bereits in der Software einige Funktionen für die Erkennung von Fehlern. So wird die Datenintegrität durch sogenannte Scrubbing Placement Groups sichergestellt. Dabei erzeugt Ceph für jede Gruppe eine Liste von allen Objekten und vergleicht die primären Objekte mit ihren Redundanzen. Dadurch lassen sich fehlende oder fehlerhafte Objekte aufspüren. Ceph bietet zwei Arten von Scrubbing. Täglich wird einmal Light Scrubbing ausgeführt, welche lediglich die Attribute und Größe der Objekte überprüft und einmal wöchentlich mit Deep Scrubbing werden alle Daten anhand von Checksummen verglichen.

4.2.3.4 *Skalierbarkeit*

Verändern sich die Anforderungen an die Persistenzlösung, kann Ceph um zusätzlichen Speicher oder auch Nodes erweitert werden. So lassen sich alle Bestandteile des Clusters, also sowohl die OSDs, MDSs, als auch die MONs skalieren. Dadurch lässt sich das System linear skalieren und dem realen Bedarf anpassen. Das Hinzufügen neuer Nodes erhöht dabei nicht nur die Kapazität, sondern kann auch die Performance erhöhen.

4.2.3.5 *Ressourcenbedarf*

Für die Installation von Ceph wird lediglich eine gängige Linux Distribution benötigt. Auch wenn Ceph keine minimale Anzahl an Nodes benötigt und alle Daemons somit auf einem einzigen laufen könnten, wird empfohlen die Daemons auf unterschiedlichen Nodes zu verteilen. Um ein hochverfügbares Cluster aufzubauen, werden zudem mindestens zwei Nodes benötigt. Zusätzlich wird empfohlen mindestens ein Gigabyte Arbeitsspeicher für jeden Daemon, der auf dem Node läuft, bereitzustellen. Handelt es sich dabei um OSDs wird empfohlen, ein Gigabyte Arbeitsspeicher je Terabyte Speicher zu verwenden.

4.2.3.6 *Benutzerfreundlichkeit*

Ähnlich wie zuvor GlusterFS bietet Ceph mit ceph-helm eine Möglichkeit, ein neues Ceph Cluster innerhalb eines bestehenden Kubernetes Clusters aufzusetzen. Dieser Helm Chart befindet sich zurzeit allerdings noch in Entwicklung. Um ein Ceph auf virtuellen Maschinen oder Servern außerhalb eines Kubernetes Clusters aufzusetzen, steht mit ceph-ansible eine Sammlung an Skripten bereit, um das Cluster aufzusetzen, zu erweitern und zu warten.

4.2.3.7 *Wartbarkeit*

Die erste Long Term Support (LTS) Version der Open-Source-Software Ceph wurde am 03. Juli 2012 veröffentlicht [UHS17]. Seitdem hat die Software sich stetig weiterentwickelt und etablierte sich am Markt. Zudem bietet Ceph eine aktive Community [Du+18] und besitzt einen transparenten Zeitplan für Veröffentlichungen.

Der erste LTS Release der aktuellen Version 12.2 wurde bereits 2017 veröffentlicht. Neben einer Dokumentation bietet Ceph aufgrund der aktiven Community viele Erfahrungsberichte, was bei der Suche nach Fehlern und der Planung eines stabilen Systems hilfreich ist.

4.2.3.8 *Zusammenfassung*

Ceph ist eine Software, welche sich aufgrund ihres Funktionsumfangs für nahezu jeden Einsatzzweck eignet. Durch ihren hochverfügbaren Aufbau und verteilten Aufbau bietet die Speicherlösung eine gute Skalierbarkeit. Allerdings befinden sich einige Funktionen wie ceph-helm oder speziell bei CephFS Snapshots noch in Entwicklung. Anders als GlusterFS bietet Ceph durch das Dashboard Plugin ein WebUI.

4.2.4 *Portworx*

Bei dem 2014 veröffentlichten [Por] handelt es sich eine Speicherlösung, welche sowohl von der Infrastruktur als auch von der Container Orchestration Lösung unabhängig ist. Neben Kubernetes kann sie zum Beispiel auch direkt mit Docker oder mit Mesophere[2] verwendet werden. Dabei bietet die Software zwei unterschiedliche Versionen. Eine kostenlose Version, genannt px-dev, welche in ihrem Funktionsumfang reduziert ist und eine kostenpflichtige Version, genannt px-enterprise, welche neben dem vollen Funktionsumfang auch Unterstützung für größere Systeme bietet.

4.2.4.1 *Ausfallsicherheit*

Die Software Portworx bietet die Möglichkeit, durch eine verteilte Struktur und Redundanzen ein hochverfügbares Cluster aufzubauen. Redundanzen werden beim Erstellen von virtuellen Datenträgern über eine Replication Factor genannte Einstellung konfiguriert. Durch einen hochverfügbares Speicherclusters lässt sich mit Portworx auf Fehler von Hardware oder von den Nodes selbst ohne Downtime reagieren.

4.2.4.2 *Datensicherung*

Während beide Versionen das Erstellen von Snapshots ermöglichen, wird die Cloud Snap genannte Funktion, um Backups von Volumen auf einen zweiten S3 Kompatiblen System abzulegen, nur von der kostenpflichtigen px-enterprise Version unterstützt. Dabei ist das erste Backup ein vollständiges Backup und danach die nächsten sechs inkrementell. Das folgende Siebte ist wieder ein vollständiges Backup. Eine Wiederherstellung dieser Sicherungen ist ohne Aufwand direkt durch den *cloudsnap* Befehl möglich. Zusätzlich ist es möglich, Snapshots zu planen und regelmäßig automatisch zu erstellen.

4.2.4.3 *Datenintegrität*

Wird Portworx auf Bare-Metal Servern installiert kann es die Laufwerke auf Fehler überprüfen und bei gefunden Fehlern den Anwender benachrichtigen oder diese

2 Plattform für das Ausführen von Containern

reparieren. Neben den in den Protokollen integrierten Mechanismen, um die Integrität sicherzustellen, kann durch Verwendung von einem Dateisystem, welches eine Überprüfung der Datenintegrität bereits enthält, die Erkennung von Fehlern sowie auch die Korrektur dieser verbessert werden.

4.2.4.4 *Skalierbarkeit*

Ändern sich die Anforderungen an die persistente Speicherung kann Portworx ähnlich wie Gluster oder Ceph ohne großen Aufwand skaliert werden. Durch das Hinzufügen neuer Festplatten lässt sich die Gesamtspeicherkapazität erweitern und durch das Hinzufügen neuer Nodes die Performance steigern. Features wie eine für Container optimierte elastische Skalierung von Volumes ermöglichen eine zusätzliche Art Skalierung.

4.2.4.5 *Ressourcenbedarf*

Um ein Portworx Cluster zu erstellen, werden neben einer gängigen Linux Distribution mindestens drei Nodes benötigt. Empfohlen werden für jeden dieser Nodes mindestens vier Prozessorkerne und mindestens vier Gigabyte Arbeitsspeicher. Zusätzlich zu den Anforderungen für die Portworx Nodes wird noch eine Key-Value Datenbank benötigt. Für eine produktive Umgebung empfiehlt Portworx ein etcd Cluster mit mindestens drei Nodes. Dabei sollte jeder dieser Node acht Gigabyte Arbeitsspeicher und mindestens 100 Gigabyte Festplattenspeicher haben. Ab Version 2.0 kann automatisch ein Key-Value Datenbank bei der Installation von Portworx erzeugt werden. Die kostenlose Version px-dev ist auf ein Maximum von einem Terabyte Speicher und drei Nodes beschränkt.

4.2.4.6 *Benutzerfreundlichkeit*

Neben einem Helm Chart steht für die Installation über die Webseite von Portworx mit dem Kubernetes Spec Generator ein Tool bereit, um Portworx zu konfigurieren und ein Skript für die Installation in einem Kubernetes Cluster zu erstellen. Dadurch lässt sich Portworx einfach und ohne zusätzlichen Aufwand installieren. Im Gegensatz zu den vorherigen Lösungen wird Portworx typischerweise durch den Container Orchestrator bereitgestellt. Für die Verwaltung des Speicherclusters und der erzeugten Datenträger steht mit Lighthouse eine WebUI zur Verfügung.

4.2.4.7 *Wartbarkeit*

Portworx bietet eine Speicherlösung, welche im Vergleich mit den anderen Lösungen die kürzeste Zeit am Markt ist. Dennoch bietet sie eine Dokumentation und aktive Community. Allerdings bietet die Software daher auch weniger Erfahrungsberichte. Zusätzlich fehlt eine transparente Darstellung des Zeitplans für Veröffentlichungen.

4.2.4.8 *Zusammenfassung*

Im Gegensatz zu den vorherigen Lösungen bietet Portworx, welches für die persistente Datenspeicherung von Container entwickelt wurde, bereits von Anfang an eine gute Integration in Container Infrastruktur. Weitere Funktionen wie das elastische Ska-

lieren von erstellten Datenträgern oder das Verschlüsseln der Datenträger mit selbst bereitgestellten Schlüsseln machen diese Persistenzlösung zu einer umfangreichen Software für die Datenspeicherung. Allerdings benötigt Portworx im Vergleich mit den anderen Lösungen eine große Menge an Ressourcen und aufgrund der kürzeren Zeit am Markt existieren für diese Software weniger Erfahrungsberichte.

4.2.5 *Quobyte*

Die letzte evaluierte Persistenzlösung Quobyte ist ebenfalls eine reine Software Lösung, welche den Speicher mehrerer Nodes gebündelt zur Verfügung stellt. Dabei versucht Quobyte mit einem wartungsarmen für Datencenter geeignetem System, viele Anwendungsfälle abzudecken. Die Software bietet Schnittstellen für Protokolle wie NFS, S3 und SMB. Dadurch ermöglicht sie das Zusammenfassen verschiedener Speicheranwendungen in einem System.

4.2.5.1 *Ausfallsicherheit*

Quobyte bietet wie auch GlusterFS, CephFS und Portworx ein integriertes, automatisches Erstellen von Redundanzen. Dadurch kann ein hochverfügbares System konzipiert werden, bei dem sowohl Metadaten als auch die eigentlichen Daten redundant sind. Durch manuell festlegbare Richtlinien lässt sich die Platzierung der Daten zusätzlich beeinflussen und so sicherstellen, dass sie auf unterschiedlichen Systemen oder an unterschiedlichen Orten gespeichert werden. Diese Architektur ermöglicht es, einen SPoF zu vermeiden [Inc18]. Bei einem Fehler wie Festplattenversagen oder der Verschiebung der Daten ändert die Software automatisch das Routing der Dateien, sodass sie ohne Downtime weiter verwendet werden können. Funktionen wie Rolling Updates ermöglichen es, während des Betriebs ohne Downtime defekte Hardware auszutauschen oder Updates auszuführen.

4.2.5.2 *Datensicherung*

Das Erstellen von Datensicherungen ist unabhängig von der Art des verwendeten Speichers durch gängige Tools für Sicherungen von Dateisystemen wie rsync oder fsync möglich. Auch wenn es sich um Block- oder Object-Storage handelt ist es möglich, diese als Dateisystem einzubinden. Eine weitere Option ist das asynchrone Synchronisieren der Daten auf ein anders Cluster. Externe Tools wie die Open-Source Anwendung Duplicati[3] machen ein verschlüsseltes Backup über den S3-Service von Quobyte möglich. Für das Erstellen und Verwalten von Snapshots steht eine in Quobyte integrierte Funktion bereit. Erstellte Snapshots werden innerhalb eines *.snapshots* genannten Verzeichnisses gespeichert.

4.2.5.3 *Datenintegrität*

Für einen wartungsarmen und fehlerfreien Betrieb verwendet die Software, um die Datenintegrität zu gewährleisten, Checksummen, welche auf den Client generiert und mit jedem Datenblock gespeichert werden [Inc18]. Zusätzlich führt der Health

3 https://www.duplicati.com/

Manager in regelmäßigen Abständen automatisch Data Scrubbing, also die Prüfung von Dateien auf dem Laufwerk auf Fehler und die Korrektur dieser durch die Redundanzen durch. Um den produktiven Betrieb dabei nicht zu stören, ist es möglich, Zeitfenster für diese Wartungen einzustellen. Besteht Zweifel an der Konsistenz der Daten, wie zum Beispiel während eines Split Brain Zustandes, bei dem zwei unterschiedliche Stände an Daten zur Verfügung stehen, werden diese nicht ausgeliefert bis dieser Zustand geklärt ist [Inc18].

4.2.5.4 *Skalierbarkeit*

Auch diese verteilte Speicherlösung bietet eine gute Skalierbarkeit. Die Gesamtkapazität lässt sich durch das Hinzufügen neuer oder Vergrößern bestehender Festplatten skalieren. Das Hinzufügen neuer Nodes dagegen kann neben der Kapazität auch die Performance steigern.

4.2.5.5 *Ressourcenbedarf*

Wie auch die Speicherlösungen zuvor benötigt Quobyte für die Installation ein System mit einer unterstützen Linux Distribution [Inc18]. Allerdings benötigt die Software als minimale Systemanforderungen mehr Ressourcen als die vorherigen Persistenzlösungen. Es werden mindestens vier Server benötigt, welche jeweils mit mindestens vier Prozessorkerne und 16 Gigabyte Arbeitsspeicher ausgestattet sind. Zusätzlich werden mindestens zwei leere Datenträger mit einer Kapazität von mindestens 50 Gigabyte benötigt.

4.2.5.6 *Benutzerfreundlichkeit*

Im Gegensatz zu den vorherigen Lösungen existiert für Quobyte kein Helm Chart für die Installation in einem bestehenden Kubernetes Cluster. Für die Installation auf Servern oder virtuellen Maschinen wird ein Installer ausgeliefert, der durch die Installation begleitet und eine einfache Installation in weniger als einer Stunde ermöglichen soll [Inc18]. Nach der erfolgreichen Installation durch den Installer ist es möglich, über das WebUI die existierenden Datenträger zu formatieren. Dabei kann ausgewählt werden, ob der Datenträger Metadaten oder Daten enthält. Quobyte empfiehlt, auf jedem Server mindestens ein Laufwerk für Metadaten zu verwenden. Anschließend ist die Software für die Verwendung bereit, und es können sowohl manuell über das WebUI als auch über die dynamische Erstellung von Kubernetes Volumes erzeugt werden. Das WebUI kann neben der CLI auch für die Wartung und die Konfiguration der Software verwendet werden.

4.2.5.7 *Wartbarkeit*

Die Software Quobyte, welche von der gleichnamigen in 2013 gegründeten [Quo] Firma vertrieben wird, ist die einzige der evaluierten Persistenzlösungen, die keine frei verfügbare Version anbietet. Das Unternehmen entstand aus Entwicklern des objektbasierten verteilten Dateisystems XtremeFS[4]. Aufgrund des Fehlens einer frei verfügbaren Version, dem Verzicht auf eine Open-Source Veröffentlichung und der

4 http://www.xtreemfs.org/

ähnlich wie Portworx kurzen Zeit am Markt existieren wenig Erfahrungsberichte. Durch den Erwerb der Software erhält der Nutzer Zugriff auf die Dokumentation und Support durch das Unternehmen.

4.2.5.8 *Zusammenfassung*

Die Anwendung bietet wie die anderen Speicherlösungen eine umfangreiche Lösung, welche vor allem bei großen Systemen oder Datencentern seine Vorteile ausspielen kann. Durch die Möglichkeit, mehrere Anwendungen in eine einzelne wartungsarme Software zusammenzulegen, kann aus einem bestehenden System viel Komplexität genommen werden und somit der Betrieb erleichtert werden. Dabei ist es möglich, die erstellten virtuellen Datenträger durch unterschiedliche Namespaces voneinander zu trennen. Für eine einfache Wartung stehen neben automatischer Fehlerkennung und Korrektur auch Echtzeit- und Langzeitstatistiken bereit.

4.2.6 *Zusammenfassung der Ergebnisse*

Durch den Vergleich der fünf Persistenzlösungen zeigen sich die Vor- und Nachteile dieser. Die Tabelle 4.2 zeigt dabei eine Übersicht der Anwendungen und die einzelnen technischen Anforderungen (siehe Tabelle 3.1). Ein + wurde vergeben, wenn die Anforderung in der Software integriert ist und ohne die Verwendung von externen Tools erfüllt wurde. Ein *o* erhielt eine Speicherlösung, wenn die Anforderung nur durch die Verwendung externer Tools erfüllt wurden konnte und ein - , wenn die Anforderung sich weder durch die Software selbst, noch durch externe Anwendungen erfüllen ließ.

Durch ein näheres Betrachten der Tabelle lässt sich feststellen, dass ein großer Teil der Persistenzlösungen viele Funktionen bereits in der Software integriert. Eine Ausnahme bildet das Protokoll NFS, welches nur wenige Features direkt integriert und daher auf die Verwendung externer Tools angewiesen ist. Die größten Unterschiede zeigen sich im Ressourcenbedarf sowie der Zeit der Anwendungen am Markt. Basierend auf den Anforderungen ist GlusterFS, welches bis auf die Datensicherung jede Anforderung ohne das Verwenden externer Software erfüllt, die geeignetste Lösung.

4.3 HANDLUNGSEMPFEHLUNG

Basierend auf der Evaluation der Speicherlösungen zeigt sich, dass die im Kapitel 3 für das Projekt erarbeiteten Anforderungen durch die Verwendung von GlusterFS nahezu ohne das Verwenden zusätzlicher Software abgedeckt werden können.

Durch den Wechsel von einem zentralen NFS Server auf eine verteilte Persistenzlösung wird ein möglicher SPoF beseitigt. Die Open-Source-Software bietet durch ihre Zeit am Markt und einem transparenten Veröffentlichungszyklus gute Voraussetzungen für eine langfriste Nutzung. Durch den geringen Ressourcenbedarf ist es möglich, ohne den Kauf zusätzlicher Hardware ein System aufzusetzen, welches durch die Skalierbarkeit bei sich veränderten Anforderungen während der Weiterentwicklung des Projekts einfach erweitert werden kann.

ID	Feature	NFS	GlusterFS	CephFS	Portworx	Quobyte
03	Ausfallsicherheit					
	Redundanzen	o	+	+	+	+
	Hochverfügbarkeit	o	+	+	+	+
04	Datensicherung					
	Automatische Sicherung	o	o	o	+	o
	Wiederherstellung	o	o	o	+	o
	Snapshots	o	+	+	+	+
05	Datenintegrität					
	Fehlererkennung	o	+	+	+	+
	Fehlerkorrektur	o	+	+	+	+
06	Skalierbarkeit					
	Vertikal	o	+	+	+	+
	Horizontal	o	+	+	+	+
07	Ressourcenbedarf					
	Erfüllt Anforderung	+	+	-	-	-
08	Benutzerfreundlichkeit					
	Automatische Installation	o	+	+	+	+
	Automatische Wartung	o	+	+	+	+
09	Wartbarkeit					
	Stabile Version	+	+	+	+	+
	Etablierte Software	+	+	+	o	o
	Dokumentation	+	+	+	+	+
	Open-Source	+	+	+	o	-

Tabelle 4.2: Zusammenfassung: Bewertung der Persistenzlösungen

Lediglich für das Erstellen von Datensicherungen ist bei GlusterFS ein zusätzliches Konzept nötig. Allerdings bieten gängige Tools wie zum Beispiel rsync die nötige Funktionalität, um ein Backup auszuführen. Für ein automatisiertes Erstellen von Sicherungen empfiehlt es sich, ein Script für diesen Zweck zu entwerfen.

5 IMPLEMENTIERUNG

Das vorherige Kapitel widmete sich der Evaluation der in Kubernetes existierenden Persistenzlösungen. Dafür wurden die in der Kubernetes Dokumentation genannten Volume Plugins anhand der vorher aufgestellten Anforderungen bewertet und basierend auf dieser eine Handlungsempfehlung gegeben. In diesem Abschnitt wird die Umsetzung der Handlungsempfehlung für das aktuelle System geplant und ausgeführt. Abschließend wird die finale Umsetzung betrachtet und ein Fazit erarbeitet.

5.1 IST-ZUSTAND

Die Abbildung 5.1 zeigt den aktuellen Aufbau des Kubernetes Cluster der prototypischen Umsetzung der ALM v2.0 (siehe 3.1). Dieses besteht, um eine möglichst hohe Trennung der Anwendungen selbst von der Verwaltung zu erreichen, aus vier virtuellen Maschinen, wobei zwei davon als Worker Nodes (siehe 2.3.3) die eigentlichen Pods der Anwendungen ausführen. Die zwei Worker Nodes verfügen mit je 24 Gigabyte Arbeitsspeicher und sechs Prozessorkernen über eine wesentlich bessere Ausstattung als die zwei Master Nodes (siehe 2.3.2) mit vier Prozessorkernen und vier Gigabyte Arbeitsspeicher. Um den Zustand und die Daten der Anwendungen

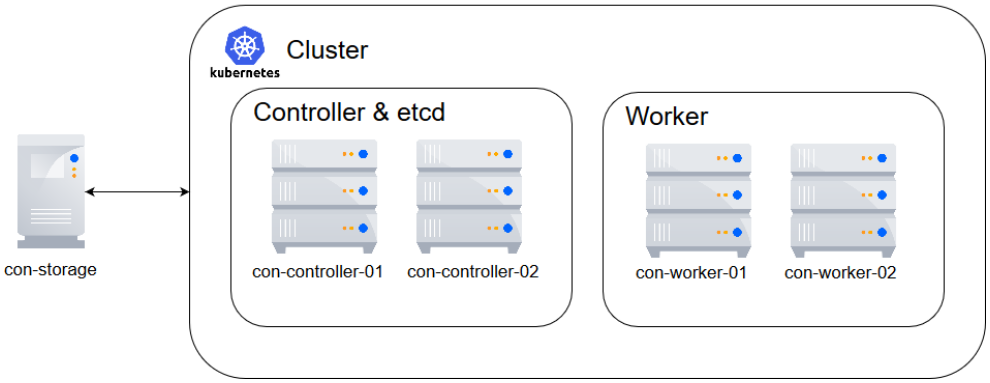

Abbildung 5.1: Cluster: Aktuelle Architektur

zu speichern, wurde zuerst eine auf einem Blockspeicher basierende Speicherlösung eingesetzt, welche direkt im Kubernetes Cluster lief. Im Laufe des Projekts wurde diese, da sie kein RWX unterstützt, durch einen zentralen NFS Server ersetzt. Diese virtuelle Maschine ist nicht redundant ausgelegt, befindet sich aber innerhalb einer Backup-Routine, welche täglich eine Sicherung des Systems erstellt. Die Datenträger für die Pods werden dynamisch durch den *nfs-client-provisioner* (siehe 4.1.1.1) erzeugt, wobei die Konfiguration dieser über die Parameter der Helm Charts erfolgt.

5.2 PLANUNG

Während der Planung der Integration von GlusterFS in das Kubernetes Cluster stellt sich zunächst die zentrale Frage, ob die Persistenzlösung innerhalb des Kubernetes Clusters oder als eigenständiges Speichercluster agieren soll. Während die Integration innerhalb des Kubernetes Clusters Vorteile wie eine schnelle Installation und eine gute Skalierbarkeit mit sich bringt, hilft das Aufsetzen außerhalb des Clusters dabei eine bessere Trennung zu schaffen. Da sich das Projekt im ständigen Wandel befindet und dies ebenfalls das Kubernetes Cluster betrifft, soll das GlusterFS-Cluster möglichst unabhängig agieren. Für die Installation stehen mehrere Hilfsmittel bereit. Neben der offiziellen Dokumentation bietet Gluster auch Skripte für die automatische Installation über Ansible, einem Tool für das automatische Aufsetzen und Konfigurieren von Systemen.

Durch das Aufbauen eines Speicherclusters verändert sich die Architektur des Clusters. Dabei wird der vorher für die Speicherung zuständige zentrale NFS-Server durch ein GlusterFS Cluster ersetzt. Für dieses werden drei virtuelle Maschinen erstellt, welche im Verbund für die Speicherung zuständig sind. Durch das Verwenden von drei Servern ist es möglich, durch eine dreifache Redundanz die Chance auf einen Split-Brain Zustand gering zu halten. Die Abbildung 5.2 zeigt den geplanten

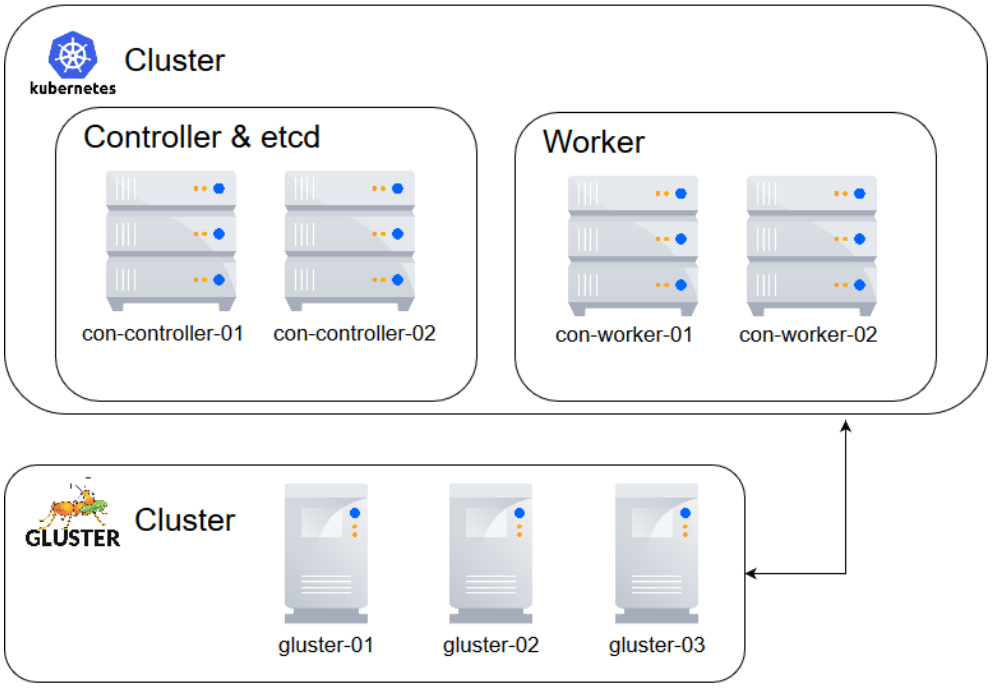

Abbildung 5.2: Cluster: Neue Architektur

zukünftigen Aufbau des Systems. Während am Kubernetes Cluster keine Änderung durchgeführt wurde, wurde der NFS Server durch ein Speichercluster aus drei Systemen ersetzt. Dieses agiert als eigenes Cluster und kann so auch unabhängig vom Kubernetes Cluster betrieben und auch in anderen Projekten verwendet werden.

5.3 UMSETZUNG

Die Integration von GlusterFS in das Kubernetes Cluster besteht aus zwei Schritten. Sie beginnt mit dem Aufsetzen der virtuellen Maschinen und der Installation und Konfiguration des Speicherclusters. Sobald dieses eigenständige System funktional ist, beginnt mit der Integration in das Kubernetes Cluster und dem Einrichten für die dynamische Datenträgererstellung der zweite Schritt.

5.3.1 *Installation und Konfiguration von GlusterFS*

Die Umsetzung beginnt mit dem Erzeugen drei neuer virtueller Maschinen. Diese erhalten jeweils ein Gigabyte Arbeitsspeicher, zwei Prozessorkerne sowie zwei Festplatten. Die Erste ist dabei für die Linux-Distribution Debian und die Daten des Betriebssystems zuständig während auf der Zweiten die Daten von GlusterFS abgelegt werden. Um die Installation und Konfiguration möglichst einfach zu halten, wird dies aus einer Kombination von mehreren Skripten für Ansible umgesetzt. Für die Installation unter Debian wird die von Jeff Geerling auf Github veröffentlichte Rolle für Ansible genutzt [Gee15].

Ist die Installation abgeschlossen, wird die Konfiguration über die von Gluster veröffentlichten Ansible Rollen [Glua] vorgenommen. Diese enthalten unterschiedliche Rollen für Wartung, die Aktivierung von Funktionen, das Aufsetzen und Verwalten des Clusters und das Erstellen und Verwalten von Datenträgern. Für die Konfiguration müssen unter Nutzung der Rolle *gluster.infra* zuerst die Datenträger erstellt und anschließend diese und das Speichercluster durch *gluster.cluster* konfiguriert werden. Werden hier bereits virtuelle Datenträger für die Verwendung im Kubernetes Cluster oder in einem anderen Projekt erstellt, bieten die Parameter die Möglichkeit, die Anzahl an Redundanzen und weitere Einstellungen zu konfigurieren.

Abschließend werden die neuen Systeme noch in eine Backup-Routine integriert. Dafür wird das Skript, welches bereits beim zentralen NFS Server genutzt wurde, erneut verwendet.

5.3.2 *Einbinden von GlusterFS in Kubernetes*

Nachdem das Speichercluster installiert wurde, kann es im Kubernetes Cluster für die dynamische Erstellung von Datenträgern eingerichtet werden. Eine Möglichkeit wäre das Einbinden eines zuvor erstellten virtuellen Datenträgers durch den bereits verwendeten *nfs-client-provisioner*. Dies würde bedeuten, dass alle gespeicherten Daten in getrennten Verzeichnissen auf diesem Datenträger erzeugt werden.

Um für jede Anwendung automatisch einen virtuellen Datenträger in GlusterFS erstellen zu lassen, welcher an die Nutzer der Anwendung ausgehändigt werden kann, wird Heketi benötigt. Heketi ist eine Software, welche eine REST Schnittstelle für das Erzeugen und Verwalten von GlusterFS-Datenträgern bietet.

Um diese Anwendung zu installieren, sind einige Schritte notwendig [Cre17]. Um diese Schritte zu automatisieren, hat Gluster ein Skript veröffentlicht [Gluf]. Durch dieses wird Heketi im Kubernetes Cluster bereitgestellt und die Kommunikation

mit dem Speichercluster sichergestellt. Damit Heketi einwandfrei funktioniert, wird noch eine passende *topology.json* benötigt. In dieser Datei werden, damit Heketi die Architektur des Speicherclusters kennt, die einzelnen Nodes des GlusterFS Systems, sowie ihr Hostname, ihre IP-Adresse und ihre Datenträger definiert.

5.4 FAZIT

Der Wechsel von einem zentralen NFS Server als Speicherlösung auf ein GlusterFS Cluster war durch die verfügbaren Skripte einfach möglich. Auch bieten diese Skripte Funktionen für die Verwaltung und Wartung des Clusters wie zum Beispiel die Skalierung und das manuelle Erstellen von Datenträgern.

Die Verwendung im Kubernetes Cluster gestaltet sich durch die Verwendung von Heketi ebenfalls einfach. Wie auch zuvor bei dem NFS Server ist es nach der Integration im Kubernetes Cluster möglich, ohne die unterliegende Architektur des Speichers zu bemerken, Datenträger dynamisch erstellen zu lassen. Diese können anschließend bei Bedarf durch Dritte eingebunden und modifiziert werden.

6 ZUSAMMENFASSUNG UND AUSBLICK

Dieses Kapitel bildet den Abschluss dieser Thesis und dient dazu, die Ergebnisse zusammenzufassen und einen Ausblick auf zukünftige Themen zu bieten.

6.1 ZUSAMMENFASSUNG

Diese Bachelorarbeit beschäftigte sich mit dem Finden einer geeigneten Kubernetes Persistenzlösung für das aktuelle Projekt, die prototypische Umsetzung der ALM v2. Dabei wurde zuerst die Idee des Projektes betrachtet und mögliche Anwendungsfälle der Software analysiert und basierend darauf Anforderungen erstellt, welche für die Evaluation der Speicherlösungen verwendet wurden.

Anschließend wurden die in der Kubernetes Dokumentation genannten Volume Plugins mit den funktionalen Anforderungen, wie den Zugriffsmodi oder die On-Premises Kompatibilität, abgeglichen, um die zu evaluierenden Speicherlösungen einzugrenzen. Dabei zeigte sich, dass fünf Lösungen diese Anforderungen erfüllen und für die weitere Analyse und Bewertung infrage kommen.

Anforderungen wie die Datensicherheit erfüllten nahezu alle Persistenzlösungen. Die größten Unterschiede zeigten sich im Ressourcenbedarf. Während NFS die wenigsten Ressourcen benötigt, muss um alle Anforderungen zu erfüllen auf verschiedene zusätzliche Software zurückgegriffen werden, was sowohl die Komplexität erhöht als auch die Wartbarkeit verschlechtert. Daher fiel die Entscheidung auf GlusterFS, welche bis auf die Datensicherung alle Anforderungen ohne zusätzliche Software erfüllt und auf virtuellen Maschinen mit einer moderaten Ausstattung lauffähig ist. Als Abschluss der Evaluation wurde eine Handlungsempfehlung formuliert.

Nach der Auswahl der geeignetsten Software wurde das bestehende System analysiert und ein Konzept für die Umsetzung der Handlungsempfehlung erarbeitet. Um die vorher existierende Lösung zu ersetzen, wurde ein eigenständiges GlusterFS Cluster erstellt, welches eigenständig lauffähig ist und nicht im Kubernetes Cluster läuft. Für diese Installation, sowie die Integration in das existierende Cluster stehen Skripte zur Verfügung, welche eine fast vollständig automatisierte Einrichtung ermöglichen. Für die Datensicherung wurde das neue System, wie auch zuvor das alte, in eine Backup-Routine integriert.

6.2 AUSBLICK

In dieser Bachelorarbeit wurde ein geeignetes Konzept für die Speicherung von Daten für die prototypische ALM v2.0 entworfen. Da das Projekt im ständigen Wandel ist, ist es nötig, diese Entscheidung bei wechselnden Anforderungen erneut zu hinterfragen.

Zusätzlich ist die Entwicklung der in Kubernetes verfügbaren Persistenzlösungen zu verfolgen. Durch die CSI und Flexvolume Schnittstelle ist für Dritte möglich, neue Volume Plugins zu entwickeln, welche für die Anforderungen gut geeignet sind. Neue Lösungen wie *Rook*[1], welches sich zurzeit in der Entwicklung befindet und in noch keiner stabilen Version vorliegt, vereinen mehrere Anwendungen wie zum Beispiel Ceph, NFS und den verteilten Object Speicher Minio in einer Lösung. Sie bieten daher die Möglichkeit, für unterschiedliche Einsatzzwecke universal genutzt zu werden.

Als letzter Punkt sind die möglichen Einsatzzwecke, um welche das GlusterFS System erweitert werden kann, zu betrachten. So können die derzeit noch im Unternehmen verwendenden NFS Servern in der Speicherlösung integriert werden oder ein Standortübergreifendes Speichersystem aufgebaut werden.

1 https://github.com/rook/rook

[Quo] *About Quobyte*. [Online; Stand 16. Dezember 2020]. URL: https://www.quobyte.com/about.

[And15] Charles Anderson. "Docker". In: *IEEE Software* 32.3 (2015), S. 102–105. ISSN: 07407459. DOI: 10.1109/MS.2015.62.

[BW18] Jonathan Baier und Jesse White. *Getting Started with Kubernetes: Extend your containerization strategy by orchestrating and managing large-scale container deployments, 3rd Edition*. Packt Publishing, 2018. ISBN: 1788994728, 9781788994729.

[Bar14] Julien Barbier. *It's Here: Docker 1.0*. [Online; Stand 16. Dezember 2020]. 2014. URL: https://blog.docker.com/2014/06/its-here-docker-1-0/.

[Bar18] Kaitlyn Barnard. *CNCF Survey: Use of Cloud Native Technologies in Production Has Grown Over 200%*. [Online; Stand 16. Dezember 2020]. 2018. URL: https://www.cncf.io/blog/2018/08/29/cncf-survey-use-of-cloud-native-technologies-in-production-has-grown-over-200-percent/.

[BS18] Ryan Bertsche und Dulce Smith. "Cluster File Systems : A Look at Implementing GlusterFS Supporting Dockerized Db2 ® , MongoDB , and PostgreSQL". In: (2018), S. 1–56.

[Bui15] Thanh Bui. "Analysis of Docker Security". In: *CoRR* abs/1501.02967 (2015). arXiv: 1501.02967. URL: http://arxiv.org/abs/1501.02967.

[Bur+16] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer und John Wilkes. "Borg, Omega, and Kubernetes". In: *Queue* 14.1 (Jan. 2016), S. 70–93. ISSN: 15427730. DOI: 10.1145/2898442.2898444. URL: http://dl.acm.org/citation.cfm?doid=2898442.2898444.

[Cre17] Scott Creeley. *Containerized Heketi with an dedicated GlusterFS cluster*. [Online; Stand 16. Dezember 2020]. 2017. URL: https://github.com/gluster/gluster-kubernetes/blob/master/docs/examples/containerized_heketi_dedicated_gluster/README.md.

[Doc] *Docker overview*. [Online; Stand 16. Dezember 2020]. URL: https://docs.docker.com/engine/docker-overview/#docker-objects.

[Du+18] Lian Du, Tianyu Wo, Renyu Yang und Chunming Hu. “Cider: A Rapid Docker Container Deployment System through Sharing Network Storage”. In: *Proceedings - 2017 IEEE 19th Intl Conference on High Performance Computing and Communications, HPCC 2017, 2017 IEEE 15th Intl Conference on Smart City, SmartCity 2017 and 2017 IEEE 3rd Intl Conference on Data Science and Systems, DSS 2017* 2018-Janua (2018), S. 332–339. DOI: 10.1109/HPCC-SmartCity-DSS.2017.44.

[Gee15] Jeff Geerling. *Ansible Role: GlusterFS*. [Online; Stand 16. Dezember 2020]. 2015. URL: https://github.com/geerlingguy/ansible-role-glusterfs.

[Glua] *Gluster Ansible Roles*. [Online; Stand 16. Dezember 2020]. URL: https://github.com/gluster/gluster-ansible.

[Glub] *Gluster Release Schedule*. [Online; Stand 16. Dezember 2020]. URL: https://www.gluster.org/release-schedule/.

[Gluc] *GlusterFS Native Storage Service for Kubernetes*. [Online; Stand 16. Dezember 2020]. URL: https://github.com/gluster/gluster-kubernetes/blob/master/docs/setup-guide.md.

[Gra16] Aaron Grattafiori. *Understanding and Hardening Linux Containers-Version 1.1*. Techn. Ber. NCC Group, 2016, S. 1–123. URL: https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/2016/april/ncc_group_understanding_hardening_linux_containers-1-1.pdf.

[Hela] *Helm - The Kubernetes Package Manager*. [Online; Stand 16. Dezember 2020]. URL: https://helm.sh/.

[Helb] *Helm Charts*. [Online; Stand 16. Dezember 2020]. URL: https://github.com/helm/charts.

[Inc11] Gluster Inc. *Cloud Storage for the Modern Data Center An Introduction to Gluster Architecture*. White Paper. 2011. URL: https://confluence.oceanobservatories.org/download/attachments/30998760/An_Introduction_To_Gluster_ArchitectureV7_110708.pdf.

[Inc18] Quobyte Inc. *Data Center File System*. White Paper. [Online; Stand 16. Dezember 2020]. Quobyte Inc., 2018. URL: https://www.quobyte.com/downloads/quobyte-tech_whitepaper.pdf.

[Kuba] *Kubernetes Components*. [Online; Stand 16. Dezember 2020]. URL: https://kubernetes.io/docs/concepts/overview/components/.

[LD02] P. Lombard und Y. Denneulin. “nfsp: A distributed NFS server for clusters of workstations”. In: *Proceedings - International Parallel and Distributed Processing Symposium, IPDPS 2002* (2002), S. 337–342. DOI: 10.1109/IPDPS.2002.1015512.

[MGR03] M. Mesnier, G.R. Ganger und E. Riedel. “Storage area networking - Object-based storage”. In: *IEEE Communications Magazine* 41.8 (2003), S. 84–90. ISSN: 0163-6804. DOI: 10.1109/MCOM.2003.1222722. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1222722.

[Por] *PX-Enterprise 1.4 Brings Application Consistent Snapshots to Kubernetes*. [Online; Stand 16. Dezember 2020]. URL: https://portworx.com/px-14-release/.

[Paw+94] Brian Pawlowski, Chet Juszczak, Peter Staubach, Carl Smith, Diane Lebel und David Hitz. “NFS Version 3 Design and Implementation”. In: *USENIX Summer Conference* (1994), S. 1–16. URL: papers2://publication/uuid/E5B9BC7A-BB8A-46F5-ADFF-5B3567849ABE.

[Ped+10] Arjan Peddemors, Christiaan Kuun, Paul Dekkers und Christiaan Den Besten. “Survey of Technologies for Wide Area Distributed Storage”. In: *Database* (2010).

[PM96] N. Peyrouze und G. Muller. “FT-NFS: an efficient fault-tolerant NFS server designed for off-the-shelf workstations”. In: *Proceedings of Annual Symposium on Fault Tolerant Computing* (1996), S. 64–73. ISSN: 07313071. DOI: 10.1109/FTCS.1996.534595. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=534595.

[Kubb] *Pods - Kubernetes*. [Online; Stand 16. Dezember 2020]. URL: https://kubernetes.io/docs/concepts/workloads/pods/pod/.

[Ren15] David K. Rensin. *Kubernetes - Scheduling the Future at Cloud Scale*. 1005 Gravenstein Highway North Sebastopol, CA 95472: O’Reilly Media, Inc, 2015, All. URL: http://www.oreilly.com/webops-perf/free/kubernetes.csp.

[Glud] *Setup, Method 1 – Setting up in virtual machines*. [Online; Stand 16. Dezember 2020]. URL: https://docs.gluster.org/en/v3/Install-Guide/Setup_virt/.

[Glue] *Setup, Method 2 – Setting up on physical serverss*. [Online; Stand 16. Dezember 2020]. URL: https://docs.gluster.org/en/v3/Install-Guide/Setup_Bare_metal/.

[Sol+07] Stephen Soltesz, Herbert Pötzl, Marc E Fiuczynski, Andy Bavier und Larry Peterson. “Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors”. In: *ACM SIGOPS Operating Systems Review* 41.3 (2007), S. 275. ISSN: 01635980. DOI: 10.1145/1272998.1273025.

[UHS17] Vikhyat Umrao, Michael Hackett und Karan Singh. *Ceph Cookbook - Second Edition: Practical Recipes to Design, Implement, Operate, and Manage Ceph Storage Systems*. 2nd. Packt Publishing, 2017. ISBN: 1788391063, 9781788391061.

[Kubc] *Volumes - Kubernete*. [Online; Stand 16. Dezember 2020]. URL: https://kubernetes.io/docs/concepts/storage/persistent-volumes/.

[Kubd] *Volumes - Kubernete*. [Online; Stand 16. Dezember 2020]. URL: https://kubernetes.io/docs/concepts/storage/volumes/.

[Wei+06] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long und Carlos Maltzahn. "Ceph: A Scalable, High-performance Distributed File System". In: *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*. OSDI '06. Seattle, Washington: USENIX Association, 2006, S. 307–320. ISBN: 1-931971-47-1. URL: http://dl.acm.org/citation.cfm?id=1298455.1298485.

[Xia15] Zhang C. & Li X. Xiao D. "The Performance Analysis of GlusterFS in Virtual Storage". In: Ameii (2015), S. 199–203.

[Yi+06] Zhao Yi, Tang Rongfeng, Xiong Jin und Ma Jie. "IncFS: An integrated high-performance distributed file system based on NFS". In: *Proceedings - 2006 International Workshop on Networking, Architecture, and Storages, NAS'06* 2006 (2006), S. 87–92. ISSN: 09258388. DOI: 10.1109/IWNAS.2006.31.

[Gluf] *gluster-kubernetes: Setup Guide*. [Online; Stand 16. Dezember 2020]. URL: https://github.com/gluster/gluster-kubernetes/blob/master/docs/setup-guide.md.

Teil II

APPENDIX

A

ANHANG

A.1 KUBERNETES PERSISTENZLÖSUNGEN

Volume Plugin	**On-Premises**	**RWO**	**ROX**	**RWX**
AWSElasticBlockStore		x		
AzureFile		x	x	x
AzureDisk		x		
CephFS	x	x	x	x
CSI	Abhängig von Treiber			
Cinder	x	x		
emptyDir	x	x		
FC	x	x	x	
Flexvolume	Abhängig von Treiber			
Flocker	x	x		
GCEPersistentDisk		x	x	
Glusterfs	x	x	x	x
HostPath	x	x		
iSCSI	x	x	x	
Quobyte	x	x	x	x
NFS	x	x	x	x
RBD	x	x	x	
VsphereVolume	x	x		
PortworxVolume	x	x		x
ScaleIO	x	x	x	
StorageOS	x	x		

Tabelle A.1: Volume Plugins in der Kubernetes Dokumentation