Fachbereich Elektrotechnik und Informatik
Dekanat - Prüfungsausschuss

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Thema der Masterarbeit

für

Herrn Thomas Finnern

Analyse und Integration von Storage-Clustern in elastische Container Plattformen

Professor Dr. Kratzke

gefördert durch:

Ausgabedatum: 02. Januar 2017
Abgabedatum: 03. Juli 2017

(Professor Dr. Andreas Hanemann)
Vorsitzender des Prüfungsausschusses

Aufgabenstellung:

Elastische Container Plattformen können auf Maschinen von verschiedenen Cloud Service Providern (z.B. AWS, GCE, Azure) ausgebracht werden und ermöglichen eine Migration von Workloads zwischen Cloud-Anbietern. Eine Realisierung dieses Ansatzes wurde im Forschungsprojekt Cloud TRANSIT bereits teilweise durch die Entwicklung eines Cluster-Deployers für Kubernetes auf Amazon AWS und OpenStack umgesetzt. Aktuell können zustandslose Applikationen providerunabhängig bereitgestellt werden. Mit einem ergänzenden Storage-Cluster sollen nun auch zustandsbehaftete Applikationen unterstützt werden.

In dieser Masterarbeit sollen dafür existierende Open Source Storage-Cluster Lösungen analysiert, integriert und hinsichtlich Migrierbarkeit evaluiert werden. Hierzu sind im Detail folgende Teilaufgaben zu bearbeiten und zu dokumentieren (Masterarbeit):

- Analyse von bestehenden Storage-Cluster Lösungen
- Automatisches Bereitstellen von Storage-Cluster Lösungen (Integration)
- Evaluierung der Migrierbarkeit von Storage-Clustern

Professor Dr. Kratzke

Fachbereich Elektrotechnik und Informatik
Dekanat - Prüfungsausschuss

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Erklärung zur Abschlussarbeit

Ich versichere, dass ich die Arbeit selbständig, ohne fremde Hilfe verfasst habe.

Bei der Abfassung der Arbeit sind nur die angegebenen Quellen benutzt worden. Wörtlich oder dem Sinne nach entnommene Stellen sind als solche gekennzeichnet.

Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, insbesondere dass die Arbeit Dritten zur Einsichtnahme vorgelegt oder Kopien der Arbeit zur Weitergabe an Dritte angefertigt werden.

_____ _____

(Datum) Unterschrift

Zusammenfassung der Masterarbeit

Fachbereich:	Elektrotechnik und Informatik
Thema:	Analyse und Integration von Storage-Clustern in elastische Container Plattformen
Zusammenfassung:	Flexibilität und Skalierbarkeit von Cloud Computing sind zwei ausreichend starke Argumente, um in Unternehmen einen Wechsel der herkömmlichen IT-Infrastruktur zu befürworten. Experten von Cloud-nativen Applikationen betrachten neben den Vorteilen auch die Tatsache, dass sich die Unternehmen durch nicht-standardisierte, herstellerspezifische Schnittstellen unweigerlich in die Abhängigkeit von Cloud Anbietern begeben (Vendor Lock-in). Ein Wechsel zwischen Cloud-Anbietern wird dadurch schwer kalkulierbar und kann unter Umständen nur mit einem hohen personellen und finanziellen Aufwand durchgeführt werden. Elastische Container Plattformen schaffen Abhilfe, da diese unabhängig von nicht-standardisierten, herstellerspezifischen Schnittstellen bei verschiedenen Cloud-Anbietern eingesetzt werden können. Ein Cluster-Deployer, der nach den Vorgaben des Anwenders automatisch ein Compute-Cluster erstellt, wurde bereits entwickelt. Da mit dem Cluster-Deployer bislang nur zustandslose Applikationen providerunabhängig bereitgestellt werden konnten, wurde im Rahmen der Arbeit eine Storage-Cluster Erweiterung, die zustandsbehaftete Applikationen unterstützt, implementiert. Dazu wurden bestehenden Storage-Cluster Lösungen auf ihre Besonderheiten hin untersucht und miteinander verglichen. Die Arbeit beschreibt die Implementierung von CephFS, Flocker und GlusterFS. Des Weiteren wurde die Migrierbarkeit zwischen zwei Cloud-Anbietern evaluiert.
Abstract:	Flexibility and scalability of cloud computing are two strong arguments to approve a change in the traditional IT infrastructure. Cloud-native application experts also consider the fact that companies are inevitably entering the dependency on cloud providers through non-standardized, manufacturer-specific interfaces (vendor lock-in). A change between cloud providers is difficult to calculate and can only be carried out with high personnel and financial expenses. Elastic container platforms provide a solution because they can be used by different cloud providers independently of non-standardized, manufacturer-specific interfaces. A cluster deployer that automatically creates a compute cluster according to the user's requirements has already been developed. As the Cluster Deployer has been able to provide only stateless applications with a provider-independent application, a storage cluster extension that supports stateful applications has been implemented as part of the work. To this end, existing storage cluster solutions were examined and compared with each other. The work describes the implementation of CephFS, Flocker and GlusterFS. In addition, the migration capability between two cloud providers was evaluated .
Autor:	Thomas Finnern
Betreuender Professor:	Prof. Dr. Nane Kratzke
WS / SS:	SS 2017

Inhaltsverzeichnis

1 Einleitung **9**

1.1 Motivation . . . 9

1.2 Ziel der Arbeit . . . 9

1.2.1 Analyse . . . 10

1.2.2 Integration . . . 10

1.2.3 Evaluation . . . 10

2 Grundlagen **11**

2.1 Cloud Computing . . . 11

2.1.1 Cloud Computing Servicemodelle . . . 12

2.1.2 Cloud Computing Entwicklungsmodelle . . . 13

2.1.3 Cloud-natives Referenzmodell . . . 13

2.2 ECP-Deploy . . . 14

2.3 Storage-Cluster . . . 17

2.3.1 Dateispeicher . . . 18

2.3.2 Blockspeicher . . . 18

2.3.3 Objektspeicher . . . 18

3 Identifikation von aktuell verfügbaren Storage-Cluster Lösungen **20**

3.1 Suchwortanalyse . . . 20

3.2 Ergebnis . . . 21

3.2.1 BeeGFS . . . 21

3.2.2 CephFS . . . 21

3.2.3 Flocker . . . 22

3.2.4 Gfarm . . . 23

3.2.5 GlusterFS . . . 23

3.2.6 Hadoop Distributed File System . . . 25

3.2.7 Infinit . . . 26

3.2.8 InterPlanetary File System . . . 26

3.2.9 Integrated Rule Orientated Data System . . . 26
3.2.10 LeoFS . . . 26
3.2.11 LizardFS . . . 27
3.2.12 Lustre . . . 27
3.2.13 MooseFS . . . 28
3.2.14 ObjectiveFS . . . 28
3.2.15 OpenAFS . . . 28
3.2.16 OrangeFS . . . 29
3.2.17 Quantcast FS . . . 29
3.2.18 RozoFS . . . 30
3.2.19 S3QL . . . 30
3.2.20 SeaweedFS . . . 30
3.2.21 Spectrum Scale . . . 30
3.2.22 Torus . . . 30
3.2.23 XtreemFS . . . 31

4 Eigenschaften der Storage-Cluster Lösungen 32

4.1 Analyse der Eigenschaften der Storage-Cluster Eigenschaften . . . 32
4.1.1 Entwicklungsstand . . . 32
4.1.2 Architektur . . . 33
4.1.3 Skalierbarkeit . . . 34
4.1.4 Transparenz . . . 35
4.1.5 Lizenzmodell . . . 35
4.1.6 CAP-Theorem . . . 36
4.1.7 Fehlertoleranz . . . 38
4.1.8 Datenkomprimierung . . . 39
4.1.9 Datendeduplizierung . . . 40
4.1.10 Geo-Replikation . . . 41
4.1.11 Objektspeicher . . . 42
4.1.12 Verschlüsselte Datenübertragung . . . 42
4.1.13 Verbindung zum Container-Cluster . . . 43
4.2 Zusammenfassung der Anforderungen . . . 45
4.3 Auswertung der Storage-Cluster Eigenschaften . . . 45
4.3.1 Storage-Cluster mit Client-Server- und Peer-to-Peer-Architektur . . . 45
4.3.2 Storage-Cluster mit Storage-Backend-Architektur . . . 46
4.3.3 Fazit . . . 47

5 Integration der Storage-Cluster Lösung 48

5.1 Umsetzungsmethoden . . . 48

5.1.1 Umsetzungsmethode 1: Gemeinsam genutzte Container-Cluster Plattform . . . 48

5.1.2 Umsetzungsmethode 2: Getrennte Cluster mit zwei Container-Cluster Plattformen 48

5.1.3 Umsetzungsmethode 3: Storage-Cluster außerhalb einer Container-Cluster Plattform . . . 49

5.1.4 Auswahl der Umsetzungsmethode . . . 49

5.2 Erweiterung des Deployers . . . 50

6 Evaluierung der Migrierbarkeit von Storage-Clustern 53

6.1 Migrationsszenario . . . 53

6.2 Migrierbarkeit von CephFS . . . 54

6.3 Migrierbarkeit von GlusterFS . . . 55

6.4 Netzwerklatenz . . . 55

6.5 Netzwerkbandbreite . . . 55

6.6 Schreib- und Lesegeschwindigkeit . . . 56

6.6.1 Vergleich der Schreibgeschwindigkeit . . . 58

6.6.2 Vergleich der Lesegeschwindigkeit . . . 59

6.7 Datenkonsistenz . . . 60

6.8 Auswahl der Storage-Cluster Konfiguration für die Migration . . . 62

7 Fazit und Ausblick 63

7.1 Fazit . . . 63

7.2 Ausblick . . . 64

7.2.1 Modularisierung des Deployers . . . 64

7.2.2 Skalierung des Speicherplatzes . . . 64

7.2.3 Storage-Multicluster Konfiguration . . . 64

A Suchmaschinen 65

B Docker-Images 66

C Instanz-Typen 67

D Anpassungen am Deployer 68

D.1 Programmiertechnische Anpassungen . . . 68

D.1.1 Anpassung der Klasse Parser . . . 68

D.1.2 Anpassung der Klasse Cluster . . . 69

D.1.3 Anpassung der Klasse Machine . . . 69

D.1.4 Anpassung der Klasse Provider . . . 69
D.1.5 Anpassung der Klassen AwsProvider und OpenStack . . . 70
D.1.6 Neue Klasse Storage . . . 72
D.1.7 Neue Klasse CephfsStorage . . . 73
D.1.8 Neue Klasse GlusterfsStorage . . . 74
D.1.9 Neue Klasse FlockerStorage . . . 76
D.2 Klassendiagramm . . . 78
D.3 Flussdiagramm . . . 80
D.4 Test der Deployer Erweiterung . . . 82
D.4.1 Test von CephFS und GlusterFS . . . 82
D.4.2 Test von Flocker . . . 84

E Testergebnisse **86**
E.1 Testergebnisse Netzwerkbandbreite . . . 86
E.1.1 TCP Netzwerkbandbreite Amazon (AWS) in Richtung OpenStack . . . 86
E.1.2 UDP Netzwerkbandbreite Amazon (AWS) in Richtung OpenStack . . . 87
E.1.3 TCP Netzwerkbandbreite OpenStack in Richtung Amazon (AWS) . . . 87
E.1.4 UDP Netzwerkbandbreite OpenStack in Richtung Amazon (AWS) . . . 88
E.2 Testergebnisse Schreib- und Lesegeschwindigkeit der Storage-Cluster . . . 89
E.2.1 Storage-Cluster basierend auf OpenStack . . . 90
E.2.2 Storage-Cluster basierend auf Amazon (AWS) . . . 93
E.2.3 Plattformübergreifendes Storage-Cluster mit Amazon (AWS) und OpenStack mit jeweils einem GlusterFS Node . . . 98
E.2.4 Plattformübergreifendes Storage-Cluster mit Amazon (AWS) und OpenStack mit jeweils zwei GlusterFS Nodes . . . 101
E.2.5 Plattformübergreifendes Storage-Cluster mit Amazon (AWS) und OpenStack mit jeweils drei GlusterFS Nodes . . . 104

Literaturverzeichnis **107**

Abbildungsverzeichnis

2.1 3-Ebenen-Modell der Cloud Computing Servicemodelle . . . 12
2.2 Cloud-natives Referenz Modell [KP16] . . . 14
2.3 Prozess „Erstellen eines Container-Clusters“ . . . 15
2.4 Virtuelle Maschine und Container . . . 16

3.1 BeeGFS Architektur [Hei] . . . 21
3.2 CephFS Architektur [Wei07] . . . 22
3.3 Flocker Architektur [flo] . . . 23
3.4 GlusterFS Varianten [glu] . . . 25
3.5 LizardFS Architektur [liz] . . . 27
3.6 Lustre Architektur [YVCJ07] . . . 28
3.7 Torus Architektur [tor] . . . 31
3.8 XtreemFS Architektur [HCK+08] . . . 31

4.1 CAP-Theorem . . . 37
4.2 CAP-Theorem Optionen . . . 37

5.1 Gemeinsam genutzte Cluster Plattform . . . 48
5.2 Getrennte Cluster mit zwei Cluster Plattformen . . . 49
5.3 Compute Cluster und Storage Cluster getrennt . . . 49
5.4 Angepasster Prozess *Erstellen eines Container-Clusters* . . . 50
5.5 Deployer Klassendiagramm . . . 51
5.6 Startzeiten der Storage-Cluster Konfigurationen . . . 52

6.1 Migrationsschritte . . . 54
6.2 Ceph-Cluster Migration . . . 54
6.3 GlusterFS Migration . . . 55
6.4 Messung der Latenz Amazon (AWS) in Richtung OpenStack . . . 56
6.5 Vergleich der Schreibgeschwindigkeit OpenStack und Amazon (AWS) . . . 59
6.6 Vergleich der Schreibgeschwindigkeit der plattformübergreifenden Storage-Cluster . . . 59

6.7 Vergleich der Lesegeschwindigkeiten der Storage-Cluster Varianten 60
6.8 Test der Datenkonsistenz - Schreiboperationen 62

D.1 Klassendiagramm 78
D.2 Klassendiagramm (Fortsetzung) 79
D.3 Flussdiagramm 80
D.4 Flussdiagramm (Fortsetzung) 81
D.5 Flussdiagramm (Fortsetzung) 82

E.1 Schreibgeschwindigkeit OpenStack mit zwei GlusterFS Nodes und Einkern-Prozessoren . 90
E.2 Lesegeschwindigkeit OpenStack mit zwei GlusterFS Nodes und Einkern-Prozessoren . . . 90
E.3 Schreibgeschwindigkeit OpenStack mit vier GlusterFS Nodes und Einkern-Prozessoren . 91
E.4 Lesegeschwindigkeit OpenStack mit vier GlusterFS Nodes und Einkern-Prozessoren . . . 91
E.5 Schreibgeschwindigkeit OpenStack mit sechs GlusterFS Nodes und Einkern-Prozessoren . 91
E.6 Lesegeschwindigkeit OpenStack mit sechs GlusterFS Nodes und Einkern-Prozessoren . . 92
E.7 Schreibgeschwindigkeit OpenStack mit zwei GlusterFS Nodes und Zwei-Prozessoren . . . 92
E.8 Lesegeschwindigkeit OpenStack mit zwei GlusterFS Nodes und Zwei-Prozessoren 92
E.9 Schreibgeschwindigkeit OpenStack mit vier GlusterFS Nodes und Zwei-Prozessoren . . . 93
E.10 Lesegeschwindigkeit OpenStack mit vier GlusterFS Nodes und Zwei-Prozessoren 93
E.13 Schreibgeschwindigkeit AWS mit zwei GlusterFS Nodes und Einkern-Prozessoren 93
E.11 Schreibgeschwindigkeit OpenStack mit sechs GlusterFS Nodes und Zwei-Prozessoren . . 94
E.12 Lesegeschwindigkeit OpenStack mit sechs GlusterFS Nodes und Zwei-Prozessoren 94
E.14 Lesegeschwindigkeit AWS mit zwei GlusterFS Node und Einkern-Prozessorens 94
E.15 Schreibgeschwindigkeit AWS mit vier GlusterFS Nodes und Einkern-Prozessoren 95
E.16 Lesegeschwindigkeit AWS mit vier GlusterFS Nodes und Einkern-Prozessoren 95
E.17 Schreibgeschwindigkeit AWS mit sechs GlusterFS Nodes und Einkern-Prozessoren 95
E.18 Lesegeschwindigkeit AWS mit sechs GlusterFS Nodes und Einkern-Prozessoren 96
E.19 Schreibgeschwindigkeit AWS mit zwei GlusterFS Nodes und Zweikern-Prozessoren 96
E.20 Lesegeschwindigkeit AWS mit zwei GlusterFS Nodes und Zweikern-Prozessoren 96
E.21 Schreibgeschwindigkeit AWS mit vier GlusterFS Nodes und Zweikern-Prozessoren 97
E.22 Lesegeschwindigkeit AWS mit vier GlusterFS Nodes und Zweikern-Prozessoren 97
E.23 Schreibgeschwindigkeit AWS mit sechs GlusterFS Nodes und Zweikern-Prozessoren . . . 97
E.24 Lesegeschwindigkeit AWS mit sechs GlusterFS Nodes und Zweikern-Prozessoren 98
E.25 Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils einem GlusterFS Node und Einkern-Prozessoren (Messung auf der OpenStack Plattform) 98
E.26 Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils einem GlusterFS Node und Einkern-Prozessoren (Messung auf der AWS Plattform) 99
E.27 Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils einem GlusterFS Node und Zweikern-Prozessoren (Messung auf der OpenStack Plattform) 99

E.28 Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils einem GlusterFS Node und Zweikern-Prozessoren (Messung auf der AWS Plattform) 99
E.29 Lesegeschwindigkeit zwischen AWS und OpenStack mit jeweils einem GlusterFS Node und Einkern-Prozessoren (Messung auf der OpenStack Plattform) 100
E.30 Lesegeschwindigkeit zwischen AWS und OpenStack mit jeweils einem GlusterFS Node und Einkern-Prozessoren (Messung auf der AWS Plattform) 100
E.31 Lesegeschwindigkeit zwischen AWS und OpenStack mit jeweils einem GlusterFS Node und Zweikern-Prozessoren (Messung auf der OpenStack Plattform) 100
E.32 Lesegeschwindigkeit zwischen AWS und OpenStack mit jeweils einem GlusterFS Node und Zweikern-Prozessoren (Messung auf der AWS Plattform) 101
E.33 Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils zwei GlusterFS Nodes und Einkern-Prozessoren (Messung auf der OpenStack Plattform) 101
E.34 Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils zwei GlusterFS Nodes und Einkern-Prozessoren (Messung auf der AWS Plattform) 102
E.35 Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils zwei GlusterFS Nodes und Zweikern-Prozessoren (Messung auf der OpenStack Plattform) 102
E.36 Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils zwei GlusterFS Nodes und Zweikern-Prozessoren (Messung auf der AWS Plattform) 102
E.37 Lesegeschwindigkeit zwischen AWS und OpenStack mit jeweils zwei GlusterFS Nodes und Einkern-Prozessoren (Messung auf der OpenStack Plattform) 103
E.38 Lesegeschwindigkeit zwischen AWS und OpenStack mit jeweils zwei GlusterFS Nodes und Einkern-Prozessoren (Messung auf der AWS Plattform) 103
E.39 Lesegeschwindigkeitzwischen AWS und OpenStack mit jeweils zwei GlusterFS Nodes und Zweikern-Prozessoren (Messung auf der OpenStack Plattform) 103
E.40 Lesegeschwindigkeitzwischen AWS und OpenStack mit jeweils zwei GlusterFS Nodes und Zweikern-Prozessoren (Messung auf der AWS Plattform) 104
E.41 Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils drei GlusterFS Nodes und Einkern-Prozessoren (Messung auf der OpenStack Plattform) 104
E.42 Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils drei GlusterFS Nodes und Einkern-Prozessoren (Messung auf der AWS Plattform) 105
E.43 Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils drei GlusterFS Nodes und Zweikern-Prozessoren (Messung auf der OpenStack Plattform) 105
E.44 Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils drei GlusterFS Nodes und Zweikern-Prozessoren (Messung auf der AWS Plattform) 105
E.45 Lesegeschwindigkeitzwischen AWS und OpenStack mit jeweils drei GlusterFS Nodes und Einkern-Prozessoren (Messung auf der OpenStack Plattform) 106
E.46 Lesegeschwindigkeitzwischen AWS und OpenStack mit jeweils drei GlusterFS Nodes und Einkern-Prozessoren (Messung auf der AWS Plattform) 106
E.47 Lesegeschwindigkeit zwischen AWS und OpenStack mit jeweils drei GlusterFS Nodes und Zweikern-Prozessoren (Messung auf der OpenStack Plattform) 106
E.48 Lesegeschwindigkeit zwischen AWS und OpenStack mit jeweils drei GlusterFS Nodes und Zweikern-Prozessoren (Messung auf der AWS Plattform) 107

Tabellenverzeichnis

2.1 Cloud Computing Eigenschaften [MG11a] . . . 12
2.2 Vergleich der Datenspeicherarchitekturen[JGG15] . . . 19

3.1 Schlüsselwörterliste . . . 20

4.1 Auswertung: Entwicklungsstand . . . 33
4.2 Auswertung: Architektur . . . 34
4.3 Arten der Transparenz in verteilten Systemen [FLdM95] . . . 35
4.4 Auswertung: Lizenzmodelle . . . 36
4.5 Auswertung: Konsistenzmodell . . . 38
4.6 Auswertung: Fehlertoleranz . . . 39
4.7 Analyse: Datenkomprimierung . . . 40
4.8 Analyse: Datendeduplizierung . . . 40
4.9 Auswertung: Geo-Replikation . . . 41
4.10 Auswertung: Objektspeicher . . . 42
4.11 Auswertung: Verschlüsselte Datenübertragung . . . 42
4.12 Auswertung: Verbindung zum Container-Cluster . . . 44
4.13 Auswertung der Storage-Cluster mit Client-Server- und Peer-to-Peer-Architektur . . . 46
4.14 Auswertung der Storage-Cluster mit Storage-Backend-Architektur . . . 46

6.1 Messung der TCP- und UDP-Netzwerkbandbreite . . . 56
6.2 Dateisystem Benchmark Tools . . . 57
6.3 Messkonstellationen Schreib- und Lesegeschwindigkeit der Storage-Cluster . . . 58

A.1 Suchmaschinen . . . 65

B.1 Docker-Images für ausgewählte Storage-Cluster Lösungen . . . 66

C.1 Instanz-Typen . . . 67

Kapitel 1

Einleitung

1.1 Motivation

Cloud Computing hat die IT-Landschaft von Unternehmen in Windeseile revolutioniert. Gegenüber klassischen IT-Landschaften, bei denen anfangs große Investitionen getätigt werden müssen, um eventuell auftretende Lastspitzen abzufangen, werden beim Cloud Computing nur die tatsächlich genutzten Ressourcen abgerechnet. Unternehmen können auf diese Weise die Kapitalausgaben durch Betriebsausgaben ersetzen und so das Investitionsrisiko verringern [MPR15]. Die Nutzung von Cloud-Dienstleistungen birgt aber das Risiko, dass sich die Unternehmen in die Abhängigkeit eines einzelnen Cloud Anbieters begeben. Diese Abhängigkeit wird den Unternehmen meist erst bewusst, wenn der Wechsel eines Cloud Anbieters notwendig ist. Geeignete Schnittstellen, die einen Wechsel begünstigen, werden von den Cloud Anbietern üblicherweise nicht bereitgestellt. Wenn die beschriebene Situation eintritt, stehen betroffene Unternehmen vor der Herausforderung, den Wechsel selbst zu organisieren, wobei der finanzielle und personelle Aufwand unüberschaubar hoch werden kann.

Lösungen hierzu werden im Rahmen des Forschungsprojektes Cloud TRANSIT, welches vom Kompetenzzentrum CoSA des Fachbereiches Elektrotechnik und Informatik der Fachhochschule Lübeck durchgeführt wird, erarbeitet. Das Projektteam untersucht Cloud-native Applikationen und deren Umfeld, um technologische Abhängigkeiten von Cloud Infrastrukturen zu minimieren. Ein Ziel ist, die Transferierbarkeit von Cloud-nativen Applikationen durch die Transferierbarkeit von elastischen Container Plattformen zu realisieren. Zusammen mit der Hochschule Hof entwickelte das Forschungsteam bereits ein Referenzmodell für Cloud-native Systeme (vgl. Abschnitt 2.1.3). Dieses Referenzmodell zeigt, dass eine Applikation sowohl auf funktionale Dienste als auch auf Storage Dienste zugreift. Die Storage Dienste werden benötigt, um Systemzustände von Cloud-nativen Applikationen providerübergreifend speichern zu können, damit eine Transferierbarkeit zwischen Cloud Anbietern ermöglicht wird. Für die funktionalen Dienste wurde bereits ein Cluster-Deployer (vgl. Abschnitt 2.2) entwickelt, der in der Lage ist, zustandslose Applikationen in einem Compute-Cluster providerunabhängig bereitzustellen. Durch die Erweiterung des Compute-Clusters mit einem Storage-Cluster soll eine Möglichkeit geschaffen werden, die Systemzustände von Cloud-nativen Applikationen zu speichern.

1.2 Ziel der Arbeit

Die Ziele dieser Arbeit sind das Analysieren von existierenden Open Source Storage-Clustern Lösungen, das Integrieren der Storage-Cluster Lösung in die vorhandene elastische Container Plattform und das Evaluieren der Migrierbarkeit der Storage-Cluster Lösung. Die Arbeit ist in folgende Teilaufgaben gegliedert.

1.2.1 Analyse

Vorhandene Storage-Cluster Lösungen sollen bezüglich ihrer speziellen Besonderheiten untersucht werden. Anschließend ist eine vergleichende Analyse mit den drei am geeignetsten Storage-Cluster Lösungen durchzuführen. Die Kriterien für die vergleichende Analyse sind im Rahmen der Arbeit zu entwickeln. Die Ergebnisse der Analyse sollen in Bezug auf die Integrationseignung in elastische Container Plattformen beurteilt werden. Ein wichtiger Punkt ist die Auswahl der Umsetzungsmethode, welche für die Integration sorgfältig ausgewählt werden muss. Es muss untersucht werden, ob die Storage-Cluster Lösung containerbasiert bereitgestellt werden soll oder ob eine separate, native Storage-Cluster Lösung in Kombination mit entsprechenden Plugins realisiert werden soll. Es ist zu erwarten, dass containerisierte Storage-Cluster Lösungen gegenüber nativen Storage-Cluster Lösungen flexibler einsetzbar sind. Meist sind die containerisierten Storage-Cluster Lösungen aber noch nicht für den produktiven Einsatz verfügbar, so dass die ausgereiften nativen Storage-Cluster weiterhin ihre Daseinsberechtigung besitzen.

1.2.2 Integration

Der bereits bestehende Deployer soll um eine Storage-Cluster Komponente erweitert werden. Das Storage-Cluster muss so implementiert werden, dass es skalierbar ist und die Architektur des Storage-Cluster-Deployer soll so gestaltet und dokumentiert werden, dass andere Storage-Systeme nachträglich ergänzt werden können.

1.2.3 Evaluation

Die Evaluierung der Migrierbarkeit zwischen Cloud-Dienstleistern muss durchgeführt werden. Hierzu müssen geeignete Test-Tools recherchiert oder entwickelt werden. Des Weiteren sollen sinnvolle Testfälle mit Applikationen sinnvoller Komplexität definiert und durchgeführt werden. Auftretende Einschränkungen insbesondere hinsichtlich Datenvollständigkeit und –konsistenz sind aufzuführen.

Kapitel 2

Grundlagen

2.1 Cloud Computing

Cloud Computing ist ein weltweiter Trend der IT, bei dem der Nutzer über ein Netzwerk üblicherweise über das Internet auf die Ressourcen entfernter Server, der sogenannten Cloud, zugreift. Die Wörter *Cloud Computing* können mit *Datenverarbeitung in der Wolke* ins Deutsche übersetzt werden. Der Begriff *Cloud* wird metaphorisch für einen Platz im Internet verwendet, an dem die Datenverarbeitung verfügbar ist und als Service angeboten wird [SMM14]. Ein großer Vorteil des Cloud Computings ist die flexible Nutzung der Ressourcen wie Prozessorleistung, Arbeitsspeicher, Speicherplatz und Netzwerkanbindung. Diese Ressourcen können mit einer großen Anzahl an Nutzern geteilt werden. Zum Beispiel stehen die flexiblen Ressourcen zum Bearbeiten von Lastspitzen in der Cloud zur Verfügung. Müsste ein Nutzer die Ressourcen für den Fall von kurzzeitigen Lastspitzen dauerhaft in einem lokalen Rechenzentrum bereithalten, so wäre dies nicht effizient und auch sehr kostenintensiv. Böhm et al. definieren 2009 das Cloud Computing als ein *"[...] auf Virtualisierung basierendes IT-Bereitstellungsmodell, bei dem Ressourcen sowohl in Form von Infrastruktur als auch Anwendungen und Daten als verteilter Dienst über das Internet durch einen oder mehrere Leistungserbringer bereitgestellt wird. Diese Dienste sind nach Bedarf flexibel skalierbar und können verbrauchsabhängig abgerechnet werden"* [BLRK03]. Das National Institute of Standards and Technology, kurz NIST, erstellte im Jahr 2011 eine Definition, nach der Cloud Computing ein serviceorientiertes Modell ist, welches einen einfachen Netzwerkzugriff auf einen Pool mit konfigurierbaren Rechenressourcen bietet. Diese Rechenressourcen können mit einem minimalen Managementaufwand oder durch einen Diensteanbieter schnell bereitgestellt und auch wieder freigegeben werden [MG11a]. Die Charakteristik einer Cloud Computing Umgebung bilden die Eigenschaften *On-demand self-service*, *Broad network access*, *Ressource pooing*, *Rapid elasticity* und *Measured service* (vgl. Tabelle 2.1).

Eigenschaft	Beschreibung
On-demand self-service	Ein Nutzer kann selbst Ressourcen wie z.B. Speicherplatz freigeben, ohne dass dieser Kontakt mit dem Cloud Computing Anbieter aufnehmen muss.
Broad network access	Der Zugang zu der Plattform ist mit Clients wie Smartphones, Tablets, PCs und Notebooks standardmäßig möglich.
Resource pooling	Der Cloud Computing Anbieter stellt einen Pool von Ressourcen (z.B. Prozessorleistung, Arbeitsspeicher, Speicherplatz und Netzwerkanbindung) zur Verfügung. Je nach Anfrage des Nutzers werden die physikalischen und virtuellen Ressourcen dynamisch zugewiesen und je nach Anwendungsfall auch wieder freigegeben und erneut zugewiesen. Aus Sicht des Nutzers ist das System transparent.
Rapid elasticity	Ressourcen werden elastisch bereitgestellt und je nach Anwendungsfall nach der Nutzung wieder freigegeben, um entsprechend der Nachfrage schnell skalieren zu können-
Measured service	Die Nutzung der Ressourcen steuert und optimiert das Cloud Computing System automatisch. Dies wird durch ein ständiges Überwachen der Ressourcen (z.B. Prozessorleistung, Arbeitsspeicher, Speicherplatz und aktive Nutzer) möglich. Sowohl der Nutzer als auch der Dienstleister erhalten anhand von Reports eine Transparenz über die eingesetzten Ressourcen.

Tabelle 2.1: Cloud Computing Eigenschaften [MG11a]

2.1.1 Cloud Computing Servicemodelle

Neben den Eigenschaften beschreibt das National Institute of Standards and Technology in einem Cloud-Modell die drei Cloud Computing Servicemodelle *Software as a Service*, *Platform as a Service* und *Infrastructure as a Service*, dessen Abhängigkeiten in dem 3-Ebenen-Modell in Abbildung 2.1 visualisiert sind [MG11a].

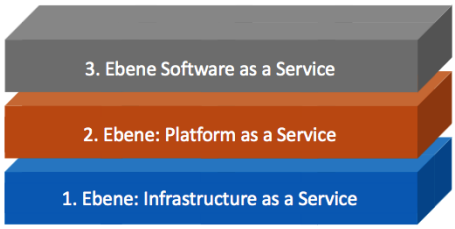

Abbildung 2.1: 3-Ebenen-Modell der Cloud Computing Servicemodelle

Nach dem 3-Ebenenmodell bildet **Infrastructure as a Service (IaaS)** die erste Ebene. Auf dieser Ebene werden Prozessorleistung, Speicherplatz, Netzwerke und andere grundlegende Rechenressourcen bereitgestellt. Der Nutzer ist somit in der Lage, beliebige Software wie zum Beispiel Betriebssysteme und Anwendungen zu installieren und auszuführen. Die Verwaltung und Kontrolle der darunterliegenden Cloud Infrastruktur ist die Aufgabe des Cloud Anbieters und nicht die des Nutzers. Der Nutzer entscheidet, zu welchem Zeitpunkt Systeme erstellt, pausiert, beendet oder gelöscht werden. Er har die Kontrolle über die Betriebssysteme, den Speicherplatz und die Anwendungen. Des Weiteren kann der Nutzer eingeschränkten Zugriff auf Netzwerkkomponenten erhalten (z.B. Netzwerkkarten, Host-Firewalls).

Die zweite Ebene ist die **Platform as a Service (PaaS)**-Ebene, bei der dem Nutzer die Möglichkeit erhält. Anwendungen basierend auf der Cloud Plattform zu entwickeln. Der Cloud Anbieter stellt eine

Ausführungsumgebung mit der jeweils benötigten Programmiersprache, den Bibliotheken, den Services und Tools zur Verfügung. Der Nutzer muss sich nicht um die darunterliegende Cloud Infrastruktur und somit auch nicht um den Betrieb des Netzwerkes, der Serverhardware oder des Betriebssystems kümmern. Die entwickelten Anwendungen und die Entwicklungsumgebung sind die Teile des Systems, die der Nutzer kontrolliert.

Bei **Software as a Service (SaaS)** handelt es sich um die dritte Ebene, bei der ein Cloud Anbieter eine Anwendung zur Verfügung stellt. Der Zugriff auf diese Anwendung ist über unterschiedliche Schnittstellen der Cloud Plattform möglich. Für die Verwaltung der darunterliegenden Cloud Infrastruktur ist der Cloud Anbieter zuständig, so dass der Nutzer sich nicht um das Netzwerk, Server. Betriebssysteme, Speicherplatz kümmern muss. Zudem kümmert sich der Cloud Anbieter auch um die Softwarepflege, so dass der Nutzer selbst nur sehr eingeschränkt Konfigurationsänderungen an der Software durchführen kann.

2.1.2 Cloud Computing Entwicklungsmodelle

Die Cloud Computing Dienstleistungen werden auf verschiedene Arten bereitgestellt. Je nach dem Umfeld und den Nutzern der Cloud Computing Dienstleistung kommt eines der vom National Institute of Standards and Technology definierten Cloud Computing Entwicklungsmodelle *Private Cloud*, *Community Cloud*, *Public Cloud* oder *Hybrid Cloud* zum Einsatz [MG11a].

Eine **private Cloud** wird innerhalb eines Unternehmens nur den eigenen Mitarbeitern und teilweise externen Organisationseinheiten zur Verfügung gestellt. Der Zugriff ist dabei nur direkt aus dem internen Netz oder mit entsprechenden VPN-Verbindungen möglich. Private Clouds werden meist von den unternehmenseigenen IT-Abteilungen betrieben, um Datenschutz und IT-Sicherheit garantieren zu können.

Der Zugriff auf eine **Community Cloud** ist nicht öffentlich für jeden verfügbar, sondern richtet sich nur an einen geschlossenen Nutzerkreis. Dies können zum Beispiel Unternehmen sein, die sich mit Hilfe der Cloud Daten zu einem gemeinsamen Thema (z.B. Projekte, Security Management, Compliance Management) austauschen.

Eine **Public Cloud** stellt einen Zugriff per Internet einer breiten Öffentlichkeit zur Verfügung. Sie wird von einem Unternehmen, einer akademischen oder staatlichen Organisation zur Verfügung gestellt und betrieben. Der Hardware der Public Cloud befindet sich in dem Rechenzentrum des jeweiligen Cloud Anbieters.

Die **Hybrid Cloud** besteht aus einer Kombination von zwei oder mehr Cloud Formen (Private Cloud, Community Cloud oder Public Cloud). Hierbei können zum Beispiel Konzepte umgesetzt werden, die datenschutzkritische Daten in der privaten Cloud und andere Daten in der Public Cloud speichern. Bei der Bearbeitung muss besonders gut auf die Klassifizierung der jeweiligen Daten geachtet werden. Die Kombination mehrerer Clouds kann ebenfalls dazu dienen, im Falle von Lastspitzen eine dynamische Verteilung der Cloud Ressourcen durchführen zu können.

2.1.3 Cloud-natives Referenzmodell

Beim Cloud Computing geht es nicht nur darum, vorhandene Serverhardware zu virtualisieren, um die in Abschnitt 2.1 genannten Vorteile zu nutzen. Vielmehr geht es um die Entwicklung neuer innovativer Applikationen, welche speziell in der Cloud verwendet werden. Kratzke und Peinl untersuchten die verschiedenen Ansätze Cloud-nativer Applikationen und entwickelten die Definition: "*[...] A cloud-native application is a distributed, elastic and horizontal scalable system composed of (micro)services which isolates state in a minimum of stateful components. The application and each self-contained deployment unit of that application is designed according to cloud-focused design patterns and operated on a*

selfservice elastic platform“ [KP16]. Eine Cloud-native Applikation, die gemäß einer Microservice Architektur aufgebaut ist, besteht aus mehreren unabhängig voneinander arbeitenden Prozessen, von denen jeder eine kleine Aufgabe erfüllt und dessen Kommunikation untereinander über eine standardisierte Kommunikationsschnittstelle (z.B. REST[1]) funktioniert. Bei Bedarf wird jeder einzelne Microservice in der für den Einsatzzweck am besten geeigneten Programmiersprache programmiert. Es muss dabei nur darauf geachtet werden, dass die Kommunikationsschnittstelle einheitlich implementiert wird. Der Name *Microservice* entstand 2011 während eines Workshops mit Softwarearchitekten [FL14]. Im Gegensatz zu einer Komponente eines Monoliths, bei dem die Software aus einer einzigen großen Komponente besteht, sind Microservices unabhängig und werden getrennt voneinander entwickelt. Die Erhöhung der Ressourcen eines Microservices geschieht durch das Hinzufügen weiterer Microservice, die die gleiche Funktionalität erfüllen. Eigenschaften wie Hochverfügbarkeit und Skalierbarkeit können mit der Microservice Architektur realisiert werden [Qui16].

Um eine Klassifizierung von den Konzepten zur Erstellung von Cloud-nativen Applikationen zu ermöglichen, entwickelten Kratzke und Peinl ein Cloud-natives Referenzmodell (vgl. Abbildung 2.2) [KP16].

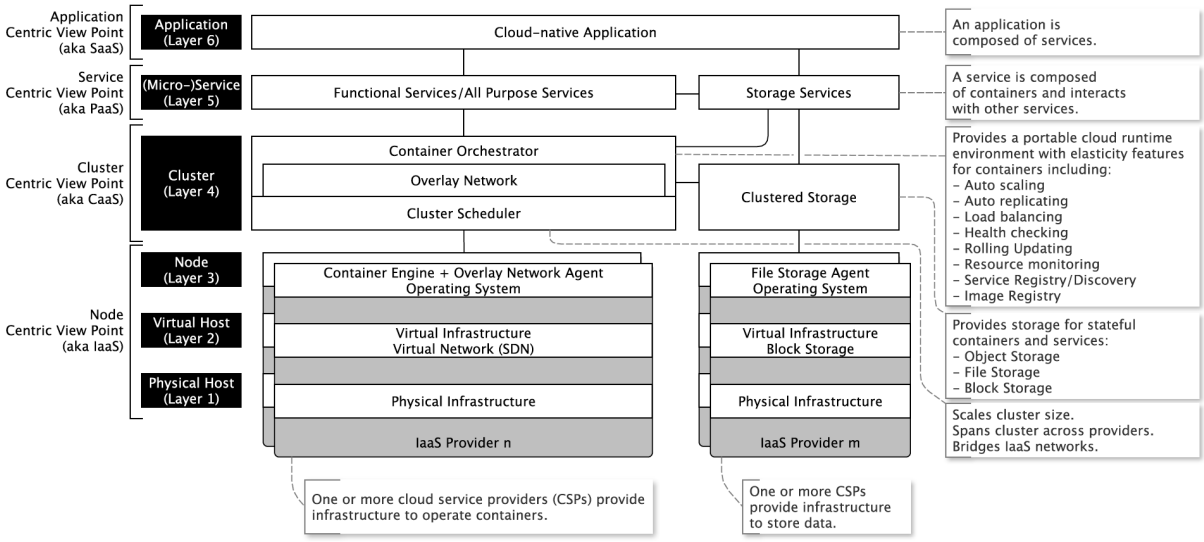

Abbildung 2.2: Cloud-natives Referenz Modell [KP16]

Neben der Klassifizierung zeigt das Referenzmodell, welche Komponenten sinnvoll kombiniert werden können, so dass Vendor Lock-in vermieden wird. Vendor Lock-in beschreibt die Situation, in der sich ein Kunde befindet, wenn dieser von den Produkten und den Dienstleistungen eines Anbieter abhängig ist und ein Wechsel zu einem anderen Anbieter nur mit erheblichen Kosten möglich ist [Kra14]. Mit Hilfe des Referenzmodells können bestehende Architekturen für Cloud-native Applikationen überarbeitet oder neue Architekturen für Cloud-Native Applikationen entwickelt werden.

2.2 ECP-Deploy

ECP-Deploy ist der Name der Softwarekomponente eines Forschungsprototyps, welcher an der FH Lübeck im Kompetenzzentrum CoSA (Kommunikation - Systeme - Anwendungen)[2] entwickelt wird. ECP-Deploy dient dazu, Container-Cluster Applikationen ohne Vendor Lock-in (automatisch) zu erstellen und zu betreiben [QK16]. Darüber hinaus soll das Gesamtsystem des Prototyps in der zukünftigen Entwicklung in der Lage sein, die Cloud Plattform, das Container-Cluster und die Container selbst zu überwachen.

[1] Representational State Transfer
[2] https://cosa.fh-luebeck.de

Neben dem Reporting sollen Methoden für die Fehlertoleranz enthalten sein. Das Gesamtsystem soll kleinen und mittleren Unternehmen die Nutzung von Cloud-nativen Applikationen ermöglichen, ohne dass die Unternehmen große, hoch spezialisierte IT-Abteilungen bereitstellen müssen.

Die aktuelle ECP-Deploy Version 0.0.5 ist nicht für den produktiven Einsatz freigegeben. Der Deployer ist bislang testweise für die Public Cloud Amazon Elastic Compute Cloud (AWS-EC2)[3] und Google Compute Engine (GCE)[4] einsetzbar. Auf der Cloud Plattform OpenStack, welche private Clouds bereitstellt, ist der Deployer ebenfalls nutzbar. Als Cluster-Software kommt in der aktuellen Deployer Version Kubernetes zum Einsatz. Zukünftig sollen weitere Cluster-Software Versionen unterstützt werden.

Funktionsweise: Beim Erstellen eines Cluster werden der Anwendung mit der Ausführung des Befehls *ecp_deploy create <credentials.json> <simpleCluster.json> <statusFile.yaml>* drei Parameter übergeben.

1. Die Datei *credentials.json* enthält die Anmeldedaten der jeweiligen Cloud Computing Plattform, bei der sich der Deployer authentisiert.
2. Die Datei *simpleCluster.json* enthält Informationen zu dem zu erstellenden Cluster. Dazu gehören der Name des Cloud Computing Anbieters, die Leistungsklasse der virtuellen Instanz (vgl. Tabelle C.1), das Betriebssystem der Instanzen und die Anzahl der Instanzen.
3. Die Informationen zu dem erzeugten Container-Cluster werden in einer Status-Datei mit dem Namen *status.yaml* abgespeichert.

Beim Start des Deployers authentisiert sich dieser mit den in der credentials.json angegebenen Daten bei dem jeweiligen Cloud Anbieter. Anschließend wird eine Security Group für die Instanzen erstellt. Eine Security Group ist die virtuelle Firewall einer Instanz. Sie besteht aus einem Regelwerk, welches die eingehende und ausgehende Datenkommunikation für bestimmte Ports und bestimmte IP-Adressbereiche blockiert oder zulässt. Nach dem Erstellen der Security Group wird aus der simpleCluster.json die Cluster-Konfiguration gebildet, welche Informationen zur Verfügung stellt, um anschließend den Master-Server und die Node-Server zu erstellen (vgl. Abbildung 2.3).

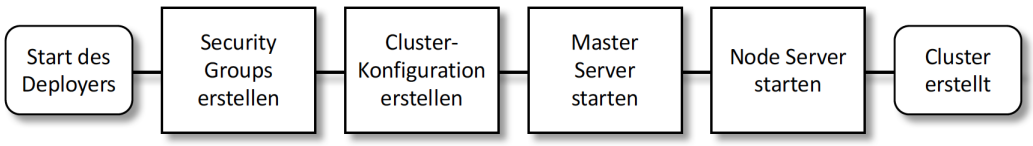

Abbildung 2.3: Prozess „Erstellen eines Container-Clusters“

Mit dem Abschluss des Startvorgangs des Deployers wird die Basis für eine containerbasierte Virtualisierung geschaffen. Mit der Virtualisierung von Ressourcen ist es möglich, mehrere virtuelle Ressourcen (z.B. Server, Speicher oder Netzwerk) auf einer entsprechenden physikalischen Ressource zu betreiben [SPEW11]. Vorteile einer Virtualisierung gegenüber physikalischen Servern sind unter anderem die effizientere Auslastung der Hardware, geringerer Stromverbrauch, die besseren Migrationsmöglichkeiten und die Kostenreduzierung. Die Virtualisierung im Bereich des Cloud Computings ermöglicht den flexiblen Einsatz von Ressourcen. Bei der containerbasierten Virtualisierung, auch bekannt als Betriebssystemvirtualisierung mit Containern, besitzen die Container kein eigenes Betriebssystem, sondern nutzen gemeinsam den Kernel des Host Betriebsystems, welches außerhalb der Container implementiert ist. Innerhalb eines Containers kann entweder ein einzelner Prozess oder mehrere parallele Prozesse laufen. Für die containerbasierte Virtualisierung wird das Betriebssystem Linux bevorzugt, da es über folgende positive Eigenschaften verfügt [FFRR14]:

[3] https://aws.amazon.com
[4] https://cloud.google.com/compute

- umfangreiche Hardwareunterstützung
- gute Leistung & Zuverlässigkeit
- weltweite Entwickler- und Nutzergemeinde
- Open Source Lizenzmodell

In einem Standard Linux Betriebssystem sind die Ressourcen (z.B. Dateisystem, Netzwerkschnittstellen, Festplatten) für alle Nutzer sichtbar. Bei der containerbasierten Virtualisierung sieht es für den Nutzer so aus, als würde zum Beispiel eine Netzwerkschnittstelle nur einmalig einem Nutzer zur Verfügung stehen. Die anderen Ressourcen werden ebenso behandelt, so dass für den Nutzer der Eindruck entsteht, dass ein Container ein eigenständiger Rechner sei. Dies wird durch die Nutzung der Kernel-Namensräume ermöglicht. welche einmalig vorhandene globale Ressourcen so zur Verfügung stellen, so dass der Nutzer denkt, dass diese ausschließlich für den Container zur Verfügung stehen. Mit Hilfe der Linux ControlGroups-Funktion (cgroups) lassen sich die CPU- und Speicherressourcen auf die einzelnen Container aufteilen und bei Bedarf administrativ beschränken oder erweitern [FFRR14]. Das ursprüngliche Konzept basiert auf Linuxcontainern (LXCs), welche entwickelt wurden, um Probleme in High-Performance-Computing-Clustern (HPCC) zu lösen [Bie06].

Die containerbasierte Virtualisierung wird gegenüber virtuellen Maschinen als *leichtgewichtig* bezeichnet, da das Betriebssystem nur einmal vorhanden sein muss [SPF+07]. Bei dieser Variante befindet sich in einerm Container nur die Anwendung bzw. der Microservice, die notwendigen Binärdateien und die Bibliotheken. Die Container führen jeweils einen voneinander isolierten Prozess im Benutzernamensraum des Host Betriebssystems aus. Bei den Namensräumen handelt es sich neben den ControlGroups um die zweite wichtige Kernelfunktion. Im Gegensatz zu der Betriebssystemvirtualisierung mit Containern besitzen virtuelle Maschinen neben dem Microservice, den Binärdateien und den Bibliotheken jeweils ein vollständiges Gast Betriebssystem mit allen notwendigen Komponenten (Speicher-, Prozess-, Geräte- und Dateiverwaltung) und somit eine weiteren Abstraktionsebene (vgl. Abbildung 2.4). Der Austausch von Daten zwischen den Gast Betriebssystemen und zwischen einem Gast Betriebssystem und dem Hypervisor erzeugt einen Overhead und minimiert die Leistung der gesamten Virtualisierungsumgebung. Der Einsatz von virtuellen Instanzen eignet sich eher, wenn in einer Virtualisierungsumgebung viele verschiedene Betriebssysteme (z.B. Windows Server, Redhat Linux, Debian Linux, Ubuntu Linux) eingesetzt werden.

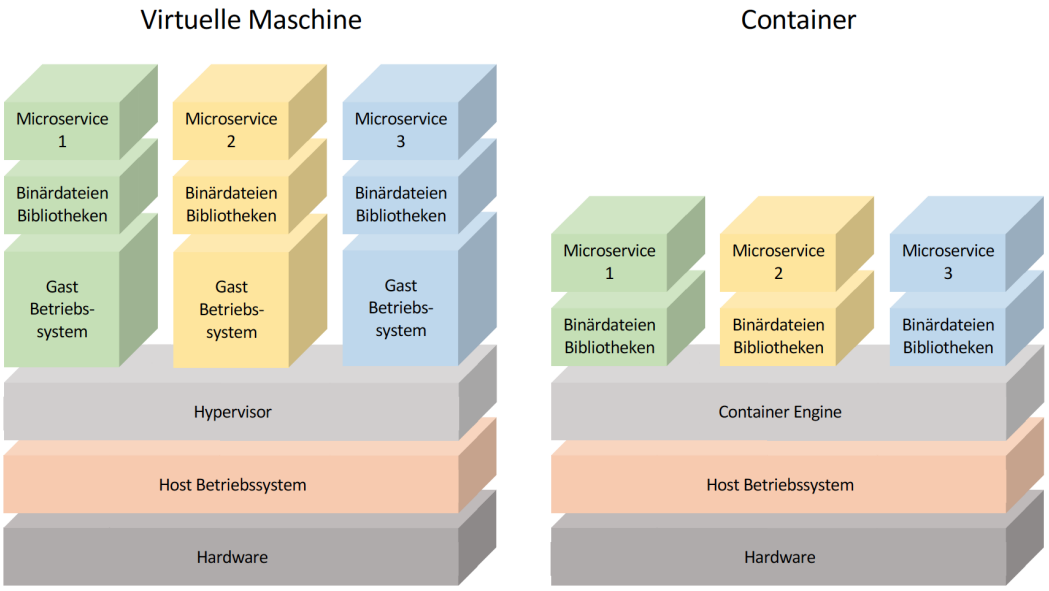

Abbildung 2.4: Virtuelle Maschine und Container

Das Programm ECP-Deploy erstellt virtuelle Instanzen mit unterschiedlichen Kombinationen aus CPU, Arbeitsspeicher und Festplattenspeicherplatz (vgl. Tabelle C.1). Die Instanzen mit den in der Konfiguration zu definierenden Leitungsklassen bilden ein Kubernetes Cluster. Kubernetes ist ein Open Source Cluster Manager, welcher von Google seit 2014 entwickelt wird. Die Entwickler verfolgen das Zeil, Kubernetes leichtgewichtig, einfach und zugänglich zu gestalten und es dennoch für komplexe Cloudumgebungen nutzbar zu machen. Kubernetes ermöglicht das Bilden von Clustern über mehrere verschiedene Server und stellt Werkzeuge zur Verfügung, mit denen Container im Cluster entwickelt, skaliert und betrieben werden können. Die Eigenschaften von Kubernetes werden auf der Internetpräsenz[5] wie folgt angegeben:

- Selbstheilend durch das automatische Ersetzen, Neustarten, Replizieren und Skalieren von Containern.
- Einfache Erweiterbarkeit und Austauschbarkeit der modularen Komponenten.
- Portierbarkeit zwischen Public-, Private-, Hybrid- und Multi-Clouds.

Eine Stärke von Kubernetes ist dessen Fehlertoleranz, welche dafür sorgt, dass das System auch bei auftretenden Fehlern stabil weiter funktioniert. Ein oder mehrere Container werden in sogenannten Pods ausgeführt. In dem Kubernetes Cluster können Docker Container ausgeführt werden. Docker ist ein Open Source-Projekt, welches das LXC-Konzept aufgegriffen und dies mit einer eigenen API (Application Programming Interface) erweitert hat [Ber14]. Hykes stellte Docker zum ersten Mal während einer Python Entwickler Konferenz in Santa Clara im März 2013 vor [MK15]. Das Docker-Container Format ermöglicht die Portierbarkeit von Containern zwischen Servern. Einzelne Docker-Container bestehen häufig aus nur einem Microservice, zum Beispiel einem Webserver oder einer Datenbank. Um ein Komplettsystem zu erstellen, können mehrere Microservices erstellt und untereinander verlinkt werden, um so gegenseitig auf die Dienste zuzugreifen, Sogenannte Docker Images werden genutzt, um Container mit einfachen Mitteln lauffähig zu erstellen. Die Images können entweder fertig vom Docker Store[6] heruntergeladen oder mit einem Dockerfile selbst erstellt werden. Fertige Images gibt es von vielen bekannten Diensten, wie zum Beispiel dem MySQL-Server[7] oder dem Apache Webserver[8]. Die fertigen Images können bei Bedarf mit dem bereitgestellten Dockerfile angepasst und anschließend selbst gebaut werden. Docker basiert auf einem Client-Server Modell, bei dem der Client mit einerm Dienstprogramm, dem sogenannten Daemon, spricht. Der Docker erstellt, startet und verteilt Docker Container. Die beiden Komponenten (Client und Daemon) befinden sich meistens auf dem selben Server, aber es wäre auch möglich den Daemon auf einem entfernten Server zu betreiben und eine Verbindung mit dem Client herzustellen. Der Docker Daemon und der Client unterhalten sich über Sockets oder per RESTful API.

Eine Stärke sind Container-Cluster, die Container ausführen. Das automatische Ausführen von Containern in einem Cluster und deren Skalierung wird von Cluster-Managern übernommen. Zu den interessantesten Cluster-Managern zählen Kubernetes [9] von Google, Docker Swarm [10], Apache Mesos [11] und Fleet von CoreOS [12]. Die genannten Cluster-Manager unterscheiden sich anhand deren Funktionsumfang.

2.3 Storage-Cluster

Ein weit verbreiteter Anwendungsfall von Cloud Computing ist das Storage-Cluster. Es ermöglicht die Speicherung von Daten auf verteilten entfernten Servern. Dem Nutzer wird eine Schnittstelle zur Spei-

[5] https://kubernetes.io/docs/concepts/overview/what-is-kubernetes
[6] https://store.docker.com
[7] https://hub.docker.com/r/mysql/mysql-server
[8] https://hub.docker.com/_/httpd
[9] https://kubernetes.io/docs/whatisk8s
[10] https://docs.docker.com/engine/swarm
[11] http://mesos.apache.org/documentation/latest/docker-containerizer
[12] https://coreos.com/fleet

cherung der Daten zur Verfügung gestellt, so dass dieser bestenfalls weltweit auf die Daten zugreifen kann. Der Nutzer sieht nicht, auf welchem der Server sich seine Daten gerade befinden. Bei dynamisch organisierten Storage-Clustern ändert sich der Speicherort von Tag zu Tag oder von Minute zu Minute [WPG+10]. Vorteile von Storage-Clustern sind Zuverlässigkeit, Konsistenz, Ausfalltoleranz und hohe Verfügbarkeit, wobei immer ein Kompromiss zwischen Konsistenz, Ausfalltoleranz und Verfügbarkeit getroffen werden muss (vgl. Abschnitt 4.1.6). Durch das Verteilen der Daten auf mehrere Storage-Nodes kommt es bei einem Hardwareausfall zu keinem Datenverlust. Wenn ein Nutzer kurzfristig Speicherplatz benötigt, muss er sich keine Gedanken, um die Serverhardware machen, sondern er kann direkt beim einem Storage-Cluster Dienstleister einen Storage-Cluster Dienst bestellen und anschließend mit dem Speichern von Daten beginnen. Spezielle Kostenmodelle ermöglichen es dem Nutzer nur den Speicher zu bezahlen, der auch genutzt wird. Für Storage-Cluster gibt es ebenfalls Servicenamen: Man spricht von *Storage as a Service*, kurz StaaS [WPG+10] oder *Data Storage as a Service*, kurz DSaaS [DIN16].

Storage-Cluster Lösungen müssen ebenso wie Container-Cluster dem Nutzer das Gefühl geben, dass die Kapazitäten des jeweiligen Systems unendlich zur Verfügung stehen. Die Elastizität eines Cloud-Dienstes wurde ebenfalls vom National Institute of Standards and Technology definiert und beschreibt die Eigenschaft, Kapazitäten eines Cloud-Dienstes elastisch zur Verfügung zu stellen [MG11b]. Einen plötzlichen Bedarf an zusätzlichem Speicherplatz muss die Storage-Cluster Lösung automatisch und unterbrechungsfrei zur Verfügung stellen.

Eine Storage-Cluster Lösung stellt Daten mittels File-, Block- oder Objektspeichern bereit. Die eingesetzte Speicherarchitektur hat großen Einfluss auf die Leistung und auf Sicherheit der Storage-Cluster Lösung. Die Tabelle Tabelle 2.2 zeigt die drei Datenspeicherarchitekturen im direkten Vergleich.

2.3.1 Dateispeicher

In einem Dateispeicher werden die Daten in der hierarchischen Struktur des Dateisystems abgespeichert. Zur Strukturierung der Dateien werden Verzeichnisse genutzt. Dem Nutzer wird die Klassifizierung anhand von Dateinamen ermöglicht [Leu03]. Zusätzliche Metadaten wie Dateigröße, Beschreibung, Besitzer und Zugriffsrechte stehen ebenfalls zur Verfügung. Der Zugriff auf die Daten erfolgt über eine Schnittstelle, welche den Dateinamen und ein Verzeichnis oder einen URL[13] beinhaltet. Damit der Dateispeicher die Daten auf einem Datenträger ablegen kann, greift dieser auf die nächste darunterliegende Blockebene zu.

2.3.2 Blockspeicher

Beim Einsatz eines Blockspeichers greift die Storage-Cluster Lösung direkt auf die Blockebene zu. Die Blockebene bildet die Schnittstelle zu den logischen und physischen Datenträgern [Leu03]. Die Daten werden erst in Blöcke aufgeteilt und anschließend mit einer eindeutigen Adressierung gespeichert. Durch die gleich große Aufteilung der Datenblöcke bietet sich die Möglichkeit an, Redundanzmechanismen auf der Ebene zu implementieren. Ein weiterer Vorteil ist die hohe Schreib- und Lesegeschwindigkeit beim direkten Zugriff auf die Blockebene. Der direkte Zugriff hat den Nachteil, dass die Themen Sicherheit und gemeinsame Datennutzung auf der Blockebene nicht ausreichend implementiert werden können [MGR03].

2.3.3 Objektspeicher

In einem Objektspeicher werden Objekte innerhalb eines objektbasierten Speichergerätes (englisch: Object-based storage device, kurz: OSD) verwaltet [FMN+05]. Das objektbasierte Speichergerät organisiert die Objekte auf der physikalischen Ebene selbständig. Ein Objekt stellt einen Container mit

[13] Uniform Resource Locator

flexibler Größe dar und besteht aus den zu speichernden Daten, den Metadaten und der eindeutigen Kennzeichnung, welche den Container im objektbasierten Speichergerät referenziert. Die Metadaten enthalten Informationen zu einem Objekt und ermöglichen eine zielgerichtete Suche nach Daten. Objektspeicher sind für große unstrukturierte Datenmengen ausgelegt und verwenden im Gegensatz zum Dateispeicher keine hierarchische Struktur.

Eigenschaft	**Dateispeicher**	**Blockspeicher**	**Objektspeicher**
Abstraktion	Dateisystemebene	Geräteebene	Benutzerebene
Zugriffsschnittstelle	NFS/ CIFS Protokoll	Fibre Channel/ iSCSI Protokoll	HTTP (REST/ SOAP API)
Organisation der Daten	Hierarchisches Dateisystem	Blöcke fester Größe	Container mit variabler Größe
Metadaten	Weniger beschreibend	nicht vorhanden	Sehr beschreibend
Kennung	Verzeichnis-, Dateiname	Benutzerdefinierter Datenträgername	Objekt ID
Leistung	Niedrig	Hoch	Hoch
Sicherheit	sicher	relativ wenig Sicherheit	sicher
Verteilt	ja	nein	ja
Anwendung	Strukturierte Daten, die nach einem bestimmten Muster organisiert werden können.	Anwendungen, die einen schnellen Zugriff und Datenbanken erfordern. Häufige Lese- und Schreibvorgänge.	Unstrukturierte Daten, die relativ statisch sind und auf die selten zugegriffen wird.

Tabelle 2.2: Vergleich der Datenspeicherarchitekturen[JGG15]

Kapitel 3

Identifikation von aktuell verfügbaren Storage-Cluster Lösungen

Um die Analyse der Eigenschaften von Storage-Cluster Lösungen durchführen zu können, wurden aktuell verfügbare Storage-Cluster recherchiert. Aufgrund der Aktualität des Themas wurde eine Online-Recherche mit Suchmaschinen, wie zum Beispiel die deutsche digitale Bibliothek[1] oder die Berkely Library[2] genutzt. Die weitere Recherche wurde entweder direkt oder explorativ durchgeführt, wodurch eine Tabelle mit den relevanten Suchmaschinen entstand (vgl. Tabelle A.1). Die Ergebnisse der Suchwortanalyse (vgl. Tabelle 3.1) wurden in die Suchmaschinen eingegeben. Dies führte zu einem repräsentativen Suchergebnis, welches in der Anforderungsanalyse in Kapitel 4 näher betrachtet wird.

3.1 Suchwortanalyse

Das Suchwort *Storage-Cluster* lässt sich für eine Internetrecherche nicht weiter in einzelne Wörter zerlegen, da die Bedeutung des Suchbegriffes verloren gehen würde. Bei der Recherche im Internet sucht die jeweilige Suchmaschine nicht nur nach der Bedeutung, sondern auch nach der Anordnung der Buchstaben. Aus diesem Grund wurde der Suchbegriff um übergeordnete Begriffe und verwandte Begriffe ergänzt und für die Suche jeweils ins Englische oder ins Deutsche übersetzt (Tabelle 3.1).

Fachbegriff: Storage-Cluster

Übergeordnete Begriffe: Cloud Computing

Verwandte Begriffe: Cluster Dateisystem, Verteiltes Dateisystem, Netzwerk Dateisystem, Paralleles Dateisystem

Nr.	Deutsch	Englisch
1	Speicher-Cluster	storage cluster
2	Cloud Datenverarbeitung	cloud computing
3	Cluster Dateisystem	cluster file system
4	Verteiltes Dateisystem	distributed file system
5	Netzwerk Dateisystem	network file system
6	Paralleles Dateisystem	parallel file system

Tabelle 3.1: Schlüsselwörterliste

[1] http://www.ddb.de
[2] http://lib.berkeley.edu

3.2 Ergebnis

Bei der Recherche wurden 23 Storage-Cluster Lösungen ermittelt, welche im folgenden vorgestellt werden.

3.2.1 BeeGFS

BeeGFS (ursprünglicher Name: FhGFS) wurde im Rahmen eines Forschungsprojektes seit dem Jahr 2005 vom Fraunhofer-Institut für Techno- und Wirtschaftsmathematik (ITWM) entwickelt [Hei]. In erster Linie ist es ein Netzwerk-Dateisystem, welches Clients die Kommunikation mit Storage-Server per Netzwerk ermöglicht. Daher wird es auch als Network Attached Storage bezeichnet. Des Weiteren ist BeeGFS ein paralleles Dateisystem. Fügt man weitere Server zu dem BeeGFS System dazu, so wird der zusätzliche Speicherplatz und die zusätzliche Leistung in einer Umgebung zusammengefasst. Bei der Speicherung von Daten werden Objekt- und Metadaten auf unterschiedlichen Servern gespeichert. Objektdaten sind die Daten, die vom Nutzer gespeichert werden. Metadaten beschreiben die Eigenschaften der Objekte, wie zum Beispiel die Dateigröße oder die Zugriffsrechte. Die Clientsoftware ist unter der GPL lizenziert und für den Server muss einer Endbenutzer-Lizenzvereinbarung zugestimmt werden, die unter anderem die Modifizierung des Quellcodes nur für interne Zwecke erlaubt. BeeGFS lässt sich auf einem Linux Betriebssystem installieren und benötigt für den Betrieb einen Managementserver, Objektspeicherserver, Metadatenserver und Dateisystem-Clients (vgl. Abbildung 3.1).

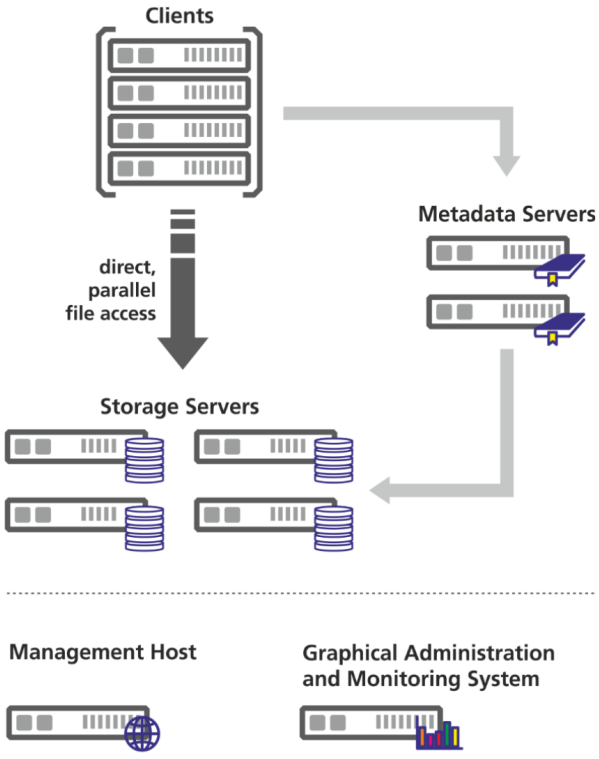

Abbildung 3.1: BeeGFS Architektur [Hei]

3.2.2 CephFS

CephFS wurde 2007 von Sage A. Weil im Rahmen seiner Dissertation an der Universität von Kalifornien als skalierbares, hochperformantes verteiltes Dateisystem vorgestellt [Wei07]. Zu den drei Hauptkomponenten eines Ceph Dateisystems gehören ein Cluster mit Object-Storage-Devices (OSDs), ein Cluster mit

Metadatenservern und die Clients, welche Daten speichern und lesen (vgl. Abbildung 3.2). CephFS stellt Objekt-, Block- und Dateispeicher in einem einheitlichen System zur Verfügung und unterscheidet sich von anderen Systemen durch das Entkoppeln der Daten-Operationen von den Metadaten-Operationen. Die Entkopplung wird möglich, in dem die Dateizuordnungstabellen durch spezielle Funktionen ersetzt werden. Somit können die Clients direkt mit den OSDs kommunizieren. Eine Kernkomponente von Ceph ist RADOS (Reliable Autonomic Distributed Object Storage). RADOS organisiert die dynamische Verteilung von Daten auf mehrere OSDs und reorganisiert diese bei Bedarf ohne den Zugriff auf die Daten einzuschränken. In einem RADOS Objekt werden eine beliebige Menge an Daten mit einem eindeutigen Schlüssel abgelegt. Die RADOS Objekte werden in RADOS Block Device Images (RBD Images) gespeichert. CephFS gehört mittlerweile zu dem US-amerikanischer Softwarehersteller Red Hat[3] und ist ein Open Source Produkt, welches mit LGPL lizenziert ist.

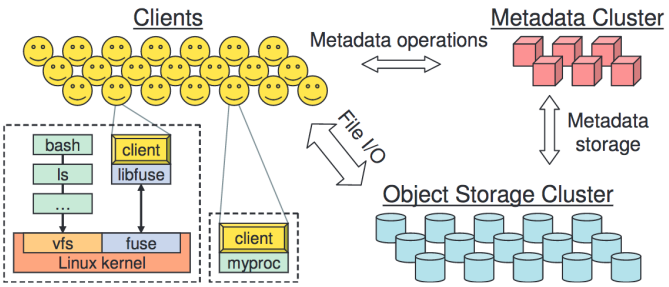

Abbildung 3.2: CephFS Architektur [Wei07]

3.2.3 Flocker

Flocker ist ein Container Orchestrator. Beim Verschieben von einem Container von einem Host zu einem anderen, organisiert Flocker die Migration der gespeicherten Daten [flo]. Flocker erstellt Data Volumes, die mit jedem Containern verknüpft werden können. Die Data Volumes nutzen wiederum Storage-Backends (vgl. Abschnitt 4.1.2), um die Daten zu speichern. Flocker ist in der Lage Docker Container und Data Volumes gemeinsam zu organisieren (vgl. Abbildung 3.3). Flocker kann mit Tools wie Docker[4], Kubernetes[5] oder Mesos[6] genutzt werden. Flocker ist Open Source und unter der Apache Lizenz 2.0 lizenziert.

[3] https://www.redhat.com/de
[4] https://www.docker.com/
[5] https://kubernetes.io/
[6] http://mesos.apache.org/

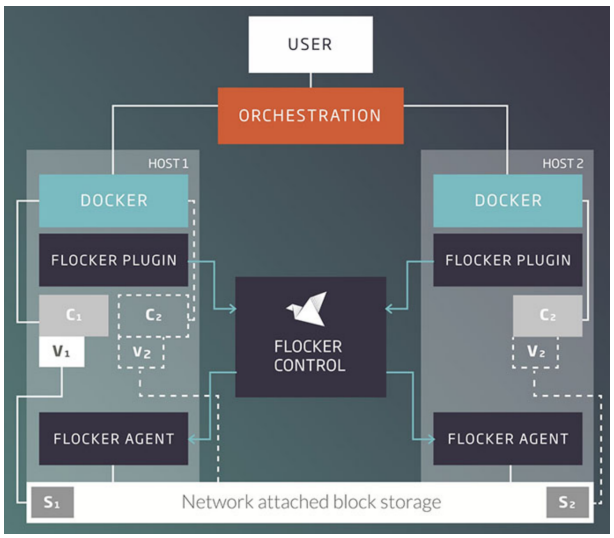

Abbildung 3.3: Flocker Architektur [flo]

3.2.4 Gfarm

Das Gfarm Dateisystem wird vom Grid Technology Research Center des National Institute of Advanced Industrial Science and Technology (AIST) entwickelt. Es ist ein paralleles Dateisystem für globale daten-intensive Computer-Cluster mit mehreren tausend Computing-Nodes [TMM+02]. Dabei wird das lokale Dateisystem tausender Nodes genutzt, um aus Gfarm Dateisystem-Nodes und Gfarm Metadatenservern Cluster zu bilden, welche Speicherplatz im Petabyte Bereich zur Verfügung stellen. Gfarm ist Open Source und unter der BSD Lizenz lizenziert.

3.2.5 GlusterFS

GlusterFS ist ein verteiltes Dateisystem, welches Speicherplatz von mehreren Petabytes tausenden Clients zur Verfügung stellt [SL16]. Es ist Open Source und unter der GPL lizenziert. Im Unterschied zu anderen Lösungen nutzt GlusterFS keine Metadatenserver. Mit der Hilfe von GlusterFS können mehrere Server über eine Netzwerkverbindung ein paralleles Netzwerk-Dateisystem aufbauen. Das Dateisystem basiert auf sogenannten Bricks. Die Bricks sind Verzeichnisse, die ein Linux Server GlusterFS zur Verfügung stellt. Mehrere Bricks bilden ein Volume. GlusterFS unterstützt abhängig von den Anforderungen fünf verschiedene Volume-Typen (vgl. Abbildung 3.4).

1. **Verteiltes GlusterFS Volume:** Dieser Speichertyp wird standardmäßig ausgewählt, wenn bei der Konfiguration keine eindeutige Auswahl getroffen wird. In diesem Fall werden die Dateien über mehrere Bricks verteilt gespeichert. Dies führt dazu, dass eine Datei entweder in dem einen Brick oder dem anderen Brick gespeichert wird, aber nie in beiden gleichzeitig. Das Ziel dieses Speichertyps ist nicht die Datenredundanz, sondern das günstige und einfache Bereitstellen von Speicherplatz. Kommt es zu einem Ausfall eines Bricks, so stehen auch dessen Daten nicht mehr zur Verfügung.

2. **Repliziertes GlusterFS Volume:** Bei diesem Speichertyp wird auf die fehlende Datenredundanz bei dem verteilten GlusterFS Volume eingegangen. Jedes Brick enthält eine exakte Kopie der Daten. Wie viele Kopien vorgehalten werden, kann beim Erstellen des Volumes festgelegt werden. Es werden mindestens zwei Bricks benötigt, um zwei Kopien der Daten zu erstellen. Werden drei Kopien der Daten benötigt, muss das Volume mit drei Bricks erstellt werden. Der Vorteil dieses Volumes ist, dass beim Ausfall eines Bricks weiterhin auf die Daten zugegriffen werden kann.

3. **Verteiltes repliziertes GlusterFS Volume:** Bei diesem Speichertyp werden die Daten über mehrere replizierte Bricks verteilt. Die Anzahl der Bricks muss immer ein Vielfaches der Anzahl der erstellten Kopien ergeben. Die Anordnung der Bricks spielt ebenfalls eine Rolle, da benachbarte Bricks Kopien von sich selbst darstellen. Das verteilte replizierte GlusterFS Volume wird verwendet, wenn ein redundantes, hoch verfügbares Speichermedium benötigt wird.

4. **Striped GlusterFS Volume:** Wenn eine große Anzahl an Clients gleichzeitig auf eine große Datei zugreift, die in einem einzelnen Brick gespeichert ist, wird dieser unter Umständen so stark überlastet, dass die Systemleistung verringert wird. In einem Striped Volume werden die Daten in kleine Chunks zerteilt und anschließend verteilt in den Bricks gespeichert. Die Anzahl der Chunks ist immer ein Vielfaches der Anzahl der Bricks. Die Last ist nun so verteilt, dass schneller auf die Daten zugegriffen werden kann. Die Datenredundanz ist bei dieser Art der Speicherung nicht gegeben.

5. **Distributed Striped Glusterfs Volume:** Bei diesem Volume wird das Striped GlusterFS Volume erweitert, so dass die Chunks in einem Striped Volume über mehrere Bricks verteilt werden.

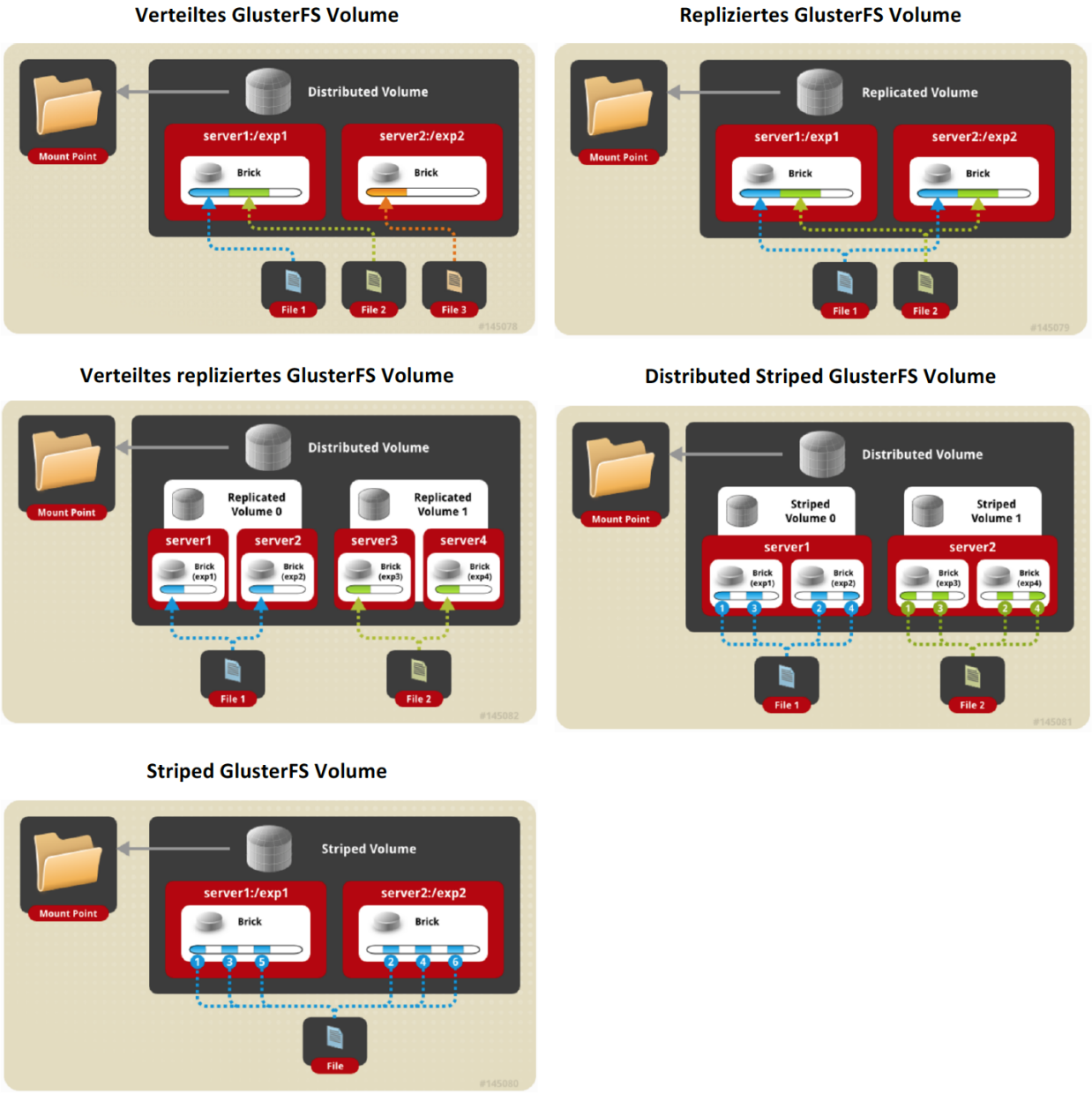

Abbildung 3.4: GlusterFS Varianten [glu]

3.2.6 Hadoop Distributed File System

Das Hadoop Distributed File System (HDFS) ist ein Apache[7] Projekt und wurde entwickelt, um große Datensätze zuverlässig zu speichern und die gespeicherten Datensätze mit einer hohen Datenrate zu einer Nutzeranwendung zu streamen [SKRC10]. Große HDFS Cluster bestehen aus tausenden Servern, welche Daten speichern und gleichzeitig Nutzeranwendungen ausführen. Die Architektur von HDFS besteht aus NameNodes, DataNodes und den HDFS Clients. Die NameNodes wissen, wo die Daten gespeichert sind. Möchte ein HDFS Client eine Datei vom HDFS Cluster lesen, so fragt er beim NameNode an, auf welchen DataNodes die Datei gespeichert ist und welcher DataNode die geringste Entfernung zum Client hat, um die Daten blockweise auf dem kürzesten Weg auszulesen. Beim Speichern von Daten beauftragt der Client den NameNode damit, drei DataNodes zu benennen, auf denen die Daten repliziert werden sollen. HDFS ist Open Source und unter der Apache Lizenz 2.0 lizenziert.

[7] https://www.apache.org/

3.2.7 Infinit

Infinit ist eine dezentralisierte Open Source Speicher-Plattform, welche während der Entwicklung die Ziele Skalierbarkeit und Elastizität verfolgt [inf]. Die Plattform unterstützt Datei-, Block- und Objektspeicher und sie kann auf Standard Serverhardware, virtuellen Maschinen oder Containern installiert werden. Mit der Peer-to-Peer Architektur möchte Infinit einen Single-Point auf Failure vermeiden. Um dies zu verhindern, nutzt Infinit Selbstheilungsmechanismen, die bei dem Ausfall eines Nodes die Daten wiederherstellen. Infinit ist zwar Open Source, die Software darf aber nur für den privaten Zweck verändert werden.

3.2.8 InterPlanetary File System

Das InterPlanetary File System (IPFS) ist ein verteiltes Peer-to-Peer Dateisystem [Ben14]. Es hat das Ziel die Ideen von erfolgreichen Peer-to-Peer Anwendungen zu nutzen und in einer neuen Anwendung zusammenzuführen. Hierzu bedienen die IPFS Entwickler sich an den Konzepten von verteilten Hash Tabellen (englisch: Distributed Hash Tables; kurz: DHTs), BitTorrent[8], Git[9] und dem selbst-zertifierenden Dateisystem[10]. Da es sich um ein Peer-to-Peer Dateisystem handelt, werden keine IPFS Nodes privilegiert. IPFS Nodes speichern die Daten in deren lokalem Speicher. IPFS ist Open Source und unter der MIT Lizenz lizenziert.

3.2.9 Integrated Rule Orientated Data System

Das integrierte regelorientierte Datensystem (englisch: integrated Rule Orientated Data System; kurz: iRODS) organisiert verteilte Daten inklusive deren Metadaten [HMP10]. Eine sogenannte *Rule Engine* erlaubt die flexible Definition von Datenspeicher, Datenzugriff und Datenverarbeitung. Es gibt drei Haupteigenschaften der iRODS Architektur [RMH+10]:

1. Eine auf einem Client-Server-Modell basierende Data Grid Architektur, welche die Interaktionen mit dem verteilten Speicher und den Rechenressourcen kontrolliert.
2. Ein Metadatenkatalog, welcher in einem Datenbanksystem abgelegt ist, um die Attribute der Daten und die Status Informationen vorzuhalten.
3. Ein Regelsystem zur Durchsetzung und Ausführung von adaptiven Regeln.

iRODS ist Open Source und unter der BSD Lizenz lizenziert.

3.2.10 LeoFS

LeoFS ist ein hoch verfügbarer, verteilter Objektspeicher und dient dazu, große Datenmengen zu speichern [leo]. Es besteht aus den Anwendungen LeoFS Storage, LeoFS Gateway und LeoFS Manager und wird mit der Programmiersprache Erlang entwickelt. LeoFS ist Open Source und unter der Apache 2.0 Lizenz lizenziert.

[8] http://www.bittorrent.com
[9] https://git-scm.com/
[10] https://pdfs.semanticscholar.org/106d/53e4b2b3d78113a53ed6ede5cf97a7006efe.pdf

3.2.11 LizardFS

LizardFS ist ein verteiltes, skalierbares, fehlertolerantes und hochverfügbares Dateisystem [liz]. Es ermöglicht das Zusammenführen des Speicherplatzes von mehreren Servern zu einem großen Speicherplatz, auf den per Linux- oder Windows-Betriebssystem zugegriffen werden kann. LizardFS legt die Metadaten und die zu speichernden Daten auf unterschiedlichen Systemen ab. Metadaten werden auf Metadatenservern gespeichert und die eigentlichen Daten werden auf sogenannten *Chunkservern* gespeichert. Eine übliche Installation besteht aus mindestens zwei Metadatenservern, welche im Master-Slave-Modus funktionieren. Eine Reihe von Chunkservern speichern die Daten. Vor dem Speichern wird jede Datei in Blöcke bzw. Chunks zerteilt und anschließend auf den Chunkservern gespeichert. Die Clients greifen direkt auf die gespeicherten Daten zu. Die LizardFS Architektur ist in Abbildung 3.5 dargestellt. LizardFS ist Open Source und unter der GPL lizenziert.

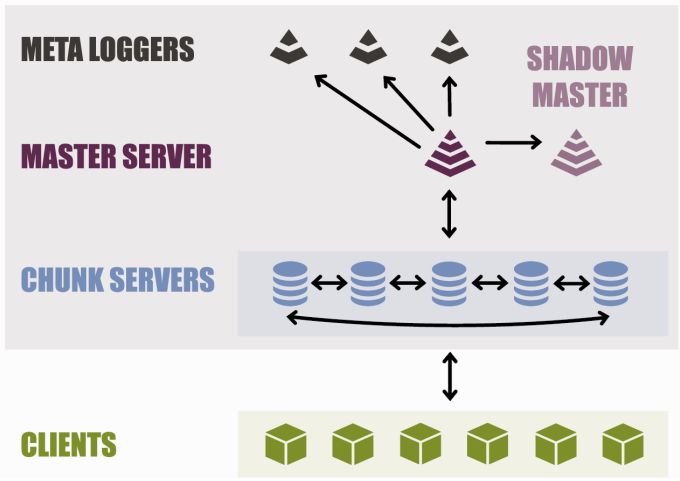

Abbildung 3.5: LizardFS Architektur [liz]

3.2.12 Lustre

Lustre ist ein Cluster-Dateisystem für Linux, welches auf Standard Serverhardware funktioniert [DHH+03]. Das Ziel der Entwicklung ist ein effizientes, skalierbares und redundantes System. Lustre besteht aus drei Komponenten - Object Storage Client (OSC), Meta-Data Server (MDS) und Object Storage Target (OST) [YVCJ07]. Abbildung 3.6 zeigt die Lustre Systemarchitektur. Ein OSC öffnet und erstellt eine Datei über den MDS (Schritt 1), welcher in jedem OST ein Objekt erstellt (Schritt 2). Im dritten Schritt, in der Abbildung 3.6 *IO* genannt, verteilt der OSC die Daten per *striping* an die drei OSTs. Dies ermöglicht Lustre, die Bandbreite von mehreren OSTs zu aggregieren. Lustre ist Open Source und unter der GPL lizenziert.

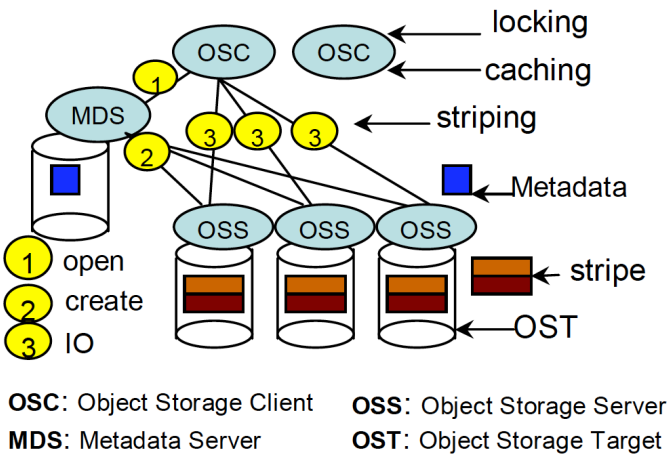

Abbildung 3.6: Lustre Architektur [YVCJ07]

3.2.13 MooseFS

MooseFS ist ein verteiltes Dateisystem, welches von der Firma Gemius SA[11] entwickelt wurde [BRF14]. Das Ziel der Entwicklung war es, ein fehlertolerantes, skalierbares Allzweck-Dateisystem für Rechenzentren zu erschaffen. Die MooseFS Systemstruktur besteht aus vier Teilen - dem Master-Server, dem Metalogger-Server, den Chunk-Servern und den Clients.

1. Der **Master-Server** verwaltet die Metadaten des Dateisystems. Unter anderem hält der Master-Server die Information vor, aus welchen Chunks sich eine Datei zusammensetzt und auf welchem Chunk-Server sich welcher Chunk befindet. Clients unterhalten sich nur mit dem Master-Server, wenn Attribute geändert werden. Die Daten werden direkt zwischen den Clients und den Chunk-Servern ausgetauscht.
2. Sollte es zum Ausfall des Master-Servers kommen, so übernimmt der **Metalogger-Server** diese Aufgabe. Im normalen Betrieb repliziert er die Daten des Master-Servers.
3. Die **Chunk-Server** speichern die Daten und replizieren diese bei Bedarf.
4. Die **Clients** kommunizieren mit dem Master-Server und den Chunk-Servern. Während der Speicherung werden die Daten in Chunks gleicher Größe zerteilt und auf verschiedene Chunk-Server verteilt.

3.2.14 ObjectiveFS

ObjectiveFS ist ein Dateisystem, welches auf den Objektspeicher von Amazon S3, Google Cloud Storage oder einen Objektspeicher mit S3-kompatibler API zurückgreift [obj]. Sollte mehr Speicherplatz benötigt werden, so teilt ObjectiveFS automatisch zusätzlichen Speicher zu. Auf dem lokalen System wird kein Speicher belegt. ObjectiveFS stellt eine Ende-zu-Ende-Verschlüsselung zur Verfügung. Es ist eine proprietäre Software.

3.2.15 OpenAFS

Bereits 1983 startete an der Carnegie Mellon University[12] zusammen mit IBM das Andrew Projekt, welches das Ziel hatte, ein verteiltes Computing System zu erstellen [MG07]. Zu diesem Projekt gehörte

[11] https://www.gemius.com/
[12] http://www.cmu.edu/

unter anderem das Andrew File System (AFS), ein skalierbares verteiltes Netzwerk-Dateisystem. AFS wurde 1998 mit der Transarc Corporation aus der Carnegie Mellon University ausgegründet und kurze Zeit später von IBM erworben. Im Jahr 2000 wurde der Quellcode durch IBM als IBM Public Lizenz veröffentlicht und das Open Source Projekt OpenAFS wurde gestartet. Die OpenAFS Architektur besteht aus Fileservern und Datenbankservern. Auf den Fileservern laufen folgende Prozesse:

- Der **Fileserver** ist ein Prozess, der Dateien und Verzeichnisse behandelt, die sich auf einem bestimmten Datenträger befinden.
- Der **Salvager Server** stellt nach einem Fehler korrupte Daten auf dem Datenträger wieder her.
- Der **Volume Server** verwaltet die lokalen Datenträger. Dazu gehört das Erstellen, Verschieben, Replizieren und Löschen der Datenträger auf einem Fileserver.

Auf dem Datenbankserver werden ebenfalls Prozesse ausgeführt und überwacht:

- Der **Backupserver** verwaltet die Backups auf dem Datenbankserver und verwaltet die Backup Datenbank.
- **Volume Location** kennt alle Informationen zu den Datenträgern und verwaltet diese in der Volume Location Datenbank.
- Der **Protection Server** verwaltet die Protection Datenbank, welche die Informationen für die Nutzerzugriffe speichert.

3.2.16 OrangeFS

OrangeFS ist ein Netzwerk-Dateisystem. Es organisiert Dateien und Verzeichnisse in Form von Speicherobjekten [YLIQ11]. Zu den wichtigsten Objekten gehören:

- **Metadatenobjekte** speichern Informationen über Dateien und Verzeichnisse wie zum Beispiel Eigentümer, Gruppe oder Zugriffsrechte.
- **Datendateiobjekte** sind Blöcke der Dateiinhalte. OrangeFS verteilt Dateien über mehrere Datendateien auf mehreren Servern, um den parallelen Zugriff zu erleichtern.
- **Dirdata-Objekte** enthalten Inhalte von Verzeichnissen oder Verzeichniseinträgen. Jeder Verzeichniseintrag ist ein Paar von Einträgen und der Bezeichner zu seinem Metadatenobjekt, entweder ein Dateimetadatenobjekt oder ein Verzeichnismetadatenobjekt.

OrangeFS ist Open Source und unter der LGPL lizenziert.

3.2.17 Quantcast FS

Quantcast FS (QFS) ist ein leistungsstarkes, fehlertolerantes, verteiltes Dateisystem. QFS baut auf dem Kosmos Dateisystem auf, welches als Open Source Dateisystem entwickelt wurde und der Architektur von HDFS ähnelt [ORR+13]. Die Daten werden in 64 MB großen Chunks lokal auf den Chunkservern gespeichert. Ein Metaserver kennt die Zuordnung der Chunks zu den Dateien. Die Clients kommunizieren mit dem Metaserver und erhalten die Informationen auf welchen Servern die Chunks gespeichert oder gelesen werden sollen. QFS ist Open Source und unter der Apache Lizenz 2.0 lizenziert.

3.2.18 RozoFS

RozoFS ist eine Open Source Software, welche ein skalierbares verteiltes Dateisystem zur Verfügung stellt [PDE+14]. Es besteht aus drei Komponenten:

- **exportd** ist ein Prozess, der auf dem Metadatenserver ausgeführt wird. Er verwaltet die Metadaten und teilt den Clients mit, welche Storage-Nodes zur Verfügung stehen und welche Storage-Nodes mit welcher Datei verknüpft sind.
- Der **storaged** Prozess läuft auf den Storage-Nodes, welche die Daten speichern.
- Der Prozess **rozofsmount** wird auf den Clients ausgeführt. Er kommuniziert mit dem Metadatenserver und ermöglicht den Schreib- und Lesezugriff auf die Storage-Nodes.

RozoFS ist unter der GPL lizenziert.

3.2.19 S3QL

S3QL ist ein Dateisystem, welches zur Speicherung der Daten auf die Cloud-Speicherdienste Google Storage, Amazon S3 oder OpenStack zugreift [s3q]. Es bietet weitere Eigenschaften wie zum Beispiel Datenkomprimierung, Verschlüsselung und Deduplizierung. S3QL ist Open Source und unter der GPL lizenziert.

3.2.20 SeaweedFS

SeaweedFS ist ein einfaches, hochskalierbares, verteiltes Dateisystem [sea]. Es wird entwickelt, um sehr viele Dateien zu speichern und um diese sehr schnell zur Verfügung zu stellen. Die SeaweedFS Architektur besteht aus einem Masterserver, Volume-Servern und den Clients. Der Masterserver verwaltet nur Datei-Volumes und überlässt den Volume-Servern die Verwaltung der Dateien und der Metadaten. Somit wird der Masterserver weniger belastet und schnellere Dateizugriffe werden ermöglicht. SeaweedFS ist Open Source und unter der Apache Lizenz 2.0 lizenziert.

3.2.21 Spectrum Scale

Spectrum Scale war bislang unter dem Namen General Parallel File System (GPFS) bekannt und ist ein paralleles Dateisystem für Cluster Computer, welches von dem US-amerikanisches IT- und Beratungsunternehmen IBM entwickelt wird [SH02]. Die Skalierbarkeit von GPFS wird durch eine Shared-Disk Architektur ermöglicht. Ein GPFS Cluster besteht aus Cluster-Nodes, auf dem die GPFS Applikationen und das GPFS Dateisystem laufen. Die Cluster-Nodes sind wiederum mittels einer Switching Fabric mit den Festplatten verbunden. Der Zugriff auf die Festplatten ist für alle Cluster-Nodes gleich. Die gespeicherten Dateien sind über alle Festplatten verteilt. Große GPFS Installationen besitzen mehrere tausend Festplatten. GPFS ist eine proprietäre Software und ist unter der IBM Lizenz lizenziert

3.2.22 Torus

Torus ist ein Open Source Projekt mit dem Ziel zuverlässigen, skalierbaren Speicher für Container-Cluster, die mit Kubernetes orchestriert werden, zur Verfügung zu stellen [tor]. Im Kern ist Torus eine Bibliothek mit einer Schnittstelle, die wie eine herkömmliche Datei aussieht und so eine Speichermanipulation durch grundlegende Dateioperationen ermöglicht. Die Dateioperationen werden von etcd[13] koordiniert (vgl. Abbildung 3.7).Torus ist unter der Apache Lizenz 2.0 lizenziert.

[13] https://github.com/coreos/etcd

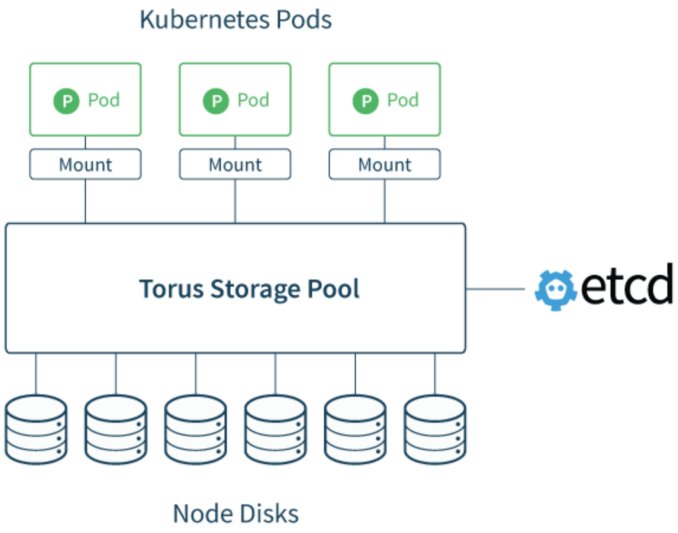

Abbildung 3.7: Torus Architektur [tor]

3.2.23 XtreemFS

Das verteilte Dateisystem XtreemFS wird am Zuse Institute Berlin[14] entwickelt. Wie auch andere objektbasierte Dateisysteme, gehören zu den Komponenten von XtreemFS Metadatenserver, Objektspeichergeräte (englisch: Object Storage Devices; kurz: OSD's) und Clients [HCK+08]. Zusätzlich merken sich die Metadatenserver den Speicherort der Replikate und werden Metadaten- und Replikat Katalog (englisch: Metadata and Replica Catalog; kurz: MRC) genannt (vgl. Abbildung 3.8).

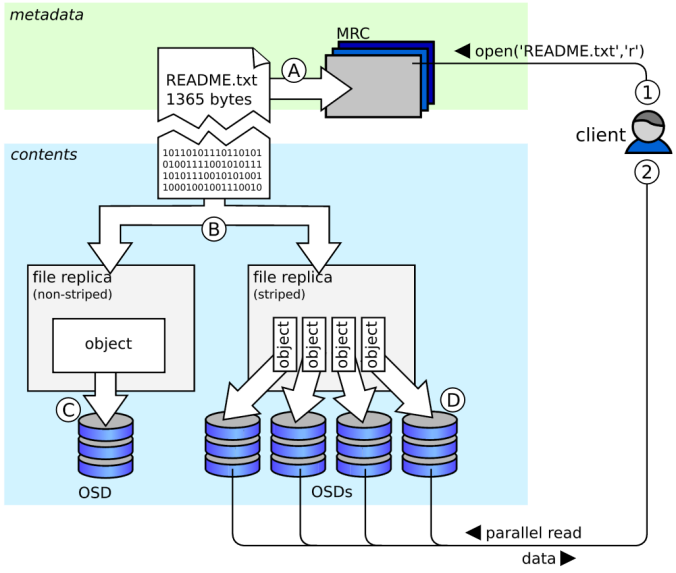

Abbildung 3.8: XtreemFS Architektur [HCK+08]

XtreemFS ist Open Source und unter der BSD Lizenz lizenziert

[14] http://www.zib.de/das

Kapitel 4

Eigenschaften der Storage-Cluster Lösungen

4.1 Analyse der Eigenschaften der Storage-Cluster Eigenschaften

Für die Eignungsbewertung wurden die einzelnen Storage-Cluster entsprechend der folgenden Eigenschaften untersucht:

1. Entwicklungsstand
2. Architektur
3. Skalierbarkeit
4. Transparenz
5. Lizenzmodell
6. CAP-Theorem
7. Fehlertoleranz
8. Datenkomprimierung
9. Datendeduplizierung
10. Geo-Replikation
11. Objektspeicher
12. Verschlüsselte Datenübertragung
13. Verbindung zum Container-Cluster

4.1.1 Entwicklungsstand

Bei modernen Softwareentwicklungsmodellen werden die einzelnen Phasen so oft wiederholt, bis die entwickelte Software einen geforderten Entwicklungsstand erreicht hat. Zur Bewertung des Entwicklungsstandes der Storage-Cluster Lösungen wurde die Semantic Versioning Spezifikation (SemVer) als Grundlage genutzt [PW]. Nach der Spezifikation ist eine marktreife Version durch eine Versionsnummer $>=$1.0.0 gekennzeichnet. Da aber nicht jede Storage-Custer Lösung die Semantic Versioning Spezifikation nutzt, wurde neben der Auswertung der Versionsnummer der Text in den Release Notes analysiert. Die Release Notes geben Aufschluss darüber, ob es sich um eine stabile Softwareversion handelt.

Entwicklungsstand	Storage-Cluster Lösung
einsetzbar	BeeGFS, CephFS, Flocker, Gfarm, GlusterFS, HDFS, iRODS, LeoFS, LizardFS, Lustre, MooseFS, ObjectiveFS, OpenAFS, OrangeFS, QuantcastFS, RozoFS, S3QL, Spectrum Scale, XtreemFS
nicht einsetzbar	Infinit, IPFS, SeaweedFS, Torus

Tabelle 4.1: Auswertung: Entwicklungsstand

Die Tabelle 4.1 zeigt die Ergebnisse der Untersuchung des Entwicklungsstandes der Storage-Cluster Lösungen. 19 der 23 Storage-Cluster Lösungen sind vom Entwicklungsstand her stabil und für eine Produktivumgebung einsetzbar. Folgende Storage-Cluster Lösungen sind nicht einsetzbar:

- **Infinit** ist in der Version 0.8.0 verfügbar. Die Software kann für die Integration zum Zeitpunkt der Arbeit nicht eingesetzt werden, da sich die AWS-S3 Schnittstelle und die NFS-Schnittstelle sich noch in der Entwicklung befinden.
- **IPFS** hat noch keine produktiv nutzbare Version veröffentlicht. Die Entwicklung befindet sich im Status *Alpha*,
- **SeaweedFS** befindet sich noch in der Entwicklung und hat die aktuelle Versionsnummer 0.69.
- **Torus** ist über den Status eines Prototyps nicht hinausgekommen, da die Entwicklung im Februar 2017 beendet wurde.

Da die Integration der Storage-Cluster Lösung in die elastische Container Plattform im Rahmen dieser Arbeit nicht nur theoretisch, sondern prototypisch implementiert wurde, musste eine Software identifiziert werden, die bereits als Storage-Cluster produktiv einsetzbar ist. Eine Software, welche sich in einer Konzeptionierungs- oder Designphase befindet, ist nicht verwendbar. Hieraus ergibt sich die Anforderung ***[M1]*: „Die Software muss produktiv einsetzbar sein“.**

4.1.2 Architektur

Bei der Untersuchung der Storage-Cluster Lösungen wurden drei Arten von Architekturen ermittelt. Dazu gehören die Client-Server-Architektur, die Peer-to-Peer-Architektur und die Storage-Backend-Architektur.

- Bei einer **Client-Server-Architektur** übernimmt ein Knoten entweder die Rolle des Servers oder des Clients, aber er kann nicht beide Rollen übernehmen [Sch01]. Hingegen kann bei einer Peer-to-Peer-Architekur ein Knoten gleichzeitig Server und Client sein. Die Zentralisierung der Kontrollfunktionen bei der Client-Server-Architektur bietet optimale Sicherheit mit guter Leistung [TLM15]. Bezüglich der Fehlertoleranz und Skalierbarkeit ergibt sich ein Single Point of Failure (kurz: SPOF), weil ein Serverausfall zum Ausfall des Komplettsystems führen kann. Es gibt zwei Arten der Client-Server-Infrastruktur: global-zentralisiert und lokal-zentralisiert. In dem ersten Fall ist nur ein Server für die Bereitstellung des Dienstes verantwortlich. Im zweiten Fall kann eine verbesserte Skalierbarkeit und Fehlertoleranz erreicht werden, indem die Steuerfunktionen auf mehrere Server verteilt werden und die dafür benötigten Daten zwischen den Servern repliziert werden. Trotzdem ist die Skalierbarkeit bei Client-Server-Architekturen nicht unbegrenzt.
- Im Gegensatz zur Client-Server-Architektur ist die **Peer-to-Peer-Architektur** für nicht vertrauenswürdige Anwendungsbereiche besser geeignet, da die Kontroll- und Speicherfunktionen der

Peer-to-Peer-Architektur auf alle Knoten verteilt werden. Des Weiteren ist die Steuerung einer Peer-to-Peer-Architektur komplexer als die einer Client-Server-Architektur. Die Peer-to-Peer-Architektur unterteilt sich in die Arten global-zentralisiert, lokal-zentralisiert und reines Peer-to-Peer: In einer global-zentralisierten Architektur dient einer der Knoten als zentraler Server, der die Informationen zu anderen Knoten speichert. Dadurch ergeben sich bei der global-zentrierten Peer-to-Peer Architektur bezüglich der System-Skalierbarkeit und der Fehlertoleranz die gleichen Schwachstellen wie bei der Client-Server-Architektur. Bei einer lokal-zentralisierten Architektur sind die Funktionen des zentralen Servers auf mehrere Knoten verteilt. Diese sogenannten Super-Knoten liefern den Clients Informationen zu dem Standort der Daten und dem Datenzugang. Schließlich enthält die reine Peer-to-Peer-Architektur keine spezifischen Knoten, sondern alle Knoten sind bezüglich ihrer Funktionen gleichwertig. Ein reine Peer-to-Peer Architektur besteht aus Knoten, die alle eine gleiche Funktion ausführen und bei denen kein Knoten eine spezielle Funktion übernimmt.

- Bei der **Storage-Backend-Architektur** stellt die Storage-Cluster Lösung ein Dateisystem und erweiterte Funktionen wie zum Beispiel eine Verschlüsselung der Daten zur Verfügung. Zum Speichern der Daten nutzt die Storage-Cluster Lösung ein Storage-Backend. Ein Storage-Backend ist ein Speicherdienst, welcher mit Hilfe einer Schnittstelle das Einbinden eines entfernten Speichers ermöglicht. Zu den bekanntesten Speicherdiensten zählen Amazon S3, Google Cloud Storage oder OpenStack Swift .

Bei den Storage-Cluster Lösungen wurde untersucht, ob es sich um eine Client-Server-Architektur, eine Peer-to-Peer-Architektur oder eine Storage-Backend-Architektur handelt. Das Ergebnis der Untersuchung zeigt, dass zwei Storage-Cluster Lösungen eine Peer-to-Peer Architektur, 17 Storage-Cluster Lösungen eine Client-Server Architektur und vier Storage-Cluster Lösungen eine Storage-Backend-Architektur verwenden (Tabelle 4.2).

Architektur	**Storage-Cluster Lösung**
Client-Server	BeeGFS, CephFS, Gfarm, GlusterFS, HDFS, LeoFS, LizardFS, Lustre, MooseFS, OpenAFS, OrangeFS, QuantcastFS, RozoFS, SeaweedFS, Spectrum Scale, Torus, XtreemFS
Peer-to-Peer	Infinit, IPFS
Storage-Backend	Flocker, iRODS, ObjectiveFS, S3QL

Tabelle 4.2: Auswertung: Architektur

Da alle drei untersuchten Architekturen für die Integration des Storage-Clusters genutzt werden können, ergibt sich daraus keine Anforderung an die Storage-Cluster Lösung.

4.1.3 Skalierbarkeit

Eine Storage-Cluster Lösung wird als skalierbar bezeichnet, wenn sie sowohl in kleinen und in großen Konfigurationen wirtschaftlich einsetzbar ist [JW00]. Es gibt zwei Formen der Skalierbarkeit [MCT08], wobei die horizontale Skalierbarkeit bei elastischen Container Plattformen im Vordergrund steht:

- Bei der vertikalen Skalierbarkeit werden die Ressourcen (z.B. Speicherplatz, CPU-Leistung) oder die Datenrate der Netzwerkschnittstelle eines einzelnen Storage-Nodes verändert. Die Skalierbarkeit fängt bei einem Storage-Node mit Einkern-Prozessor an und kann in Abhängigkeit von aktueller Hardware beliebig hoch skalieren.
- Bei der horizontalen Skalierbarkeit wird die Anzahl der Server eines Systems verändert. Mit dem Einsatz beliebig vieler Server sind theoretisch keine Leistungsgrenzen vorhanden. Bei der horizontalen Skalierbarkeit wird die obere Leistungsgrenze von der Software und der Möglichkeit zur Parallelisierung vorgegeben.

Die vorhergehende Analyse hat gezeigt, dass die Skalierbarkeit eine wesentlicher Bestandteil einer Storage-Cluster Lösung ist und alle 23 Storage-Cluster Lösungen geben allgemein an, dass diese horizontal skalierbar sind.

4.1.4 Transparenz

Storage-Cluster Lösungen bestehen aus verschiedenen Anwendungen, die auf miteinander vernetzten Servern ausgeführt werden und somit ein verteiltes System bilden. Coulouris et al. definieren ein verteiltes System als ein System, "*[...] bei dem sich die Hardware- und Softwarekomponenten auf vernetzten Rechnern befinden und nur über den Austausch von Nachrichten kommunizieren und ihre Aktionen koordinieren*" [CDK02]. Der Nutzer eines Storage-Clusters sieht die einzelnen vernetzten Server des verteilten Systems nicht, sondern für ihn ist nur der bereitgestellte Speicherplatz als eine einzelne Systemkomponente sichtbar. Diese Systeme werden auch als *transparente Systeme* bezeichnet [TVS07]. Die Transparenz-Konzepte können auf verschiedene Arten auf ein verteiltes System angewendet werden. In der Tabelle 4.3 sind die wichtigsten Transparenz-Konzepte aufgeführt.

Transparenz	Beschreibung
Zugriff	Der Nutzer merkt nicht, wenn ein verteiltes System aus verschiedenen Servern besteht und wenn unterschiedlich auf dessen Daten zugegriffen wird.
Standort	Der Standort eines Servers wird verborgen.
Migration	Es ist nicht erkennbar, wenn ein Server den Standort wechselt.
Umzug	Es ist nicht erkennbar, wenn ein Server den Standort während des Betriebs wechselt.
Replikation	Die Replikation eines Servers wird verborgen.
Gleichzeitigkeit	Der Nutzer merkt nicht, wenn ein weiterer Nutzer gleichzeitig auf den Server zugreift.
Fehler	Fehler und die Fehlerbeseitigung sind für den Nutzer nicht erkennbar.

Tabelle 4.3: Arten der Transparenz in verteilten Systemen [FLdM95]

Die Analyse der Eigenschaften der Storage-Cluster Lösungen hat gezeigt, dass die Transparenz eine notwendige Eigenschaft eines verteilten Systems ist. Aus diesem Grund muss auch die gesuchte Storage-Cluster Lösung die genannten Transparenz-Konzepte beinhalten. Da alle untersuchten Storage-Cluster Lösungen die notwendige Transparenz bereitstellen, ergibt sich daraus keine Anforderung für die Bewertung der Storage-Cluster Lösungen.

4.1.5 Lizenzmodell

Bei quelloffener Software (Open Source Software) wird zwischen restriktiven und permissiven Lizenzen unterschieden [DG16].

- **Restriktive Lizenzen** erlauben dem Nutzer die Software weiter zu verteilen und Änderungen am Quellcode durchzuführen. Dabei darf die geänderte Software nur mit der gleichen restriktiven Lizenz weiterverbreitet werden. Zu den restriktiven Lizenzen zählen unter anderem die GNU General Public License (GPL) und die GNU Lesser General Public License (LGPL), wobei die Anforderungen der LGPL weniger restriktiv sind als die der GPL.
- **Permissive Lizenzen** überlassen dem Nutzer die Entscheidung, was mit dem Quellcode getan werden soll. Somit darf der Nutzer den Quellcode beliebig ändern und gegebenenfalls auch verkaufen. Damit der Nutzer die Software verwenden kann, muss lediglich ein Urheberrechtshinweis im Quellcode bestehen bleiben. Zu den permissiven Lizenzen zählen unter anderem die Apache Lizenz 2.0, die BSD Lizenz und die MIT Lizenz.

Ein Endbenutzer-Lizenzvertrag (EULA, kurz für: End User License Agreement) wird speziell zwischen der Entwicklerfirma und dem Endnutzer abgestimmt [Sch13]. Der Nutzer erhält keine Rechte zur Weiterverbreitung oder Veränderung der Software.

Die Tabelle 4.4 zeigt, dass 20 der untersuchten Storage-Cluster Lösungen quelloffenen Code verwenden. Des Weiteren wurde bei drei Storage-Cluster Lösungen von den Entwicklern oder den Softwareherstellern eine eigene Lizenzbestimmung veröffentlicht:

- **Infinit** ist zwar als Open Source verfügbar, dennoch schreibt Infinit den Nutzern vor, dass die kostenlose Version der Software nur für den persönlichen Gebrauch und nicht für kommerzielle Zwecke genutzt werden darf. Sofern es nicht erlaubt wurde, darf die Software nicht zu irgendeinem Zweck kopiert, geändert oder verteilt werden.
- **ObjectiveFS** ist ebenfalls eine proprietäre Software. Bei einem bestehenden Abonnement darf die Software auf einer unbegrenzten Zahl von Computer installiert und kopiert werden. Eine Kopie in maschinenlesbaren Form darf nur für Sicherungszwecke erstellt werden. Die Sicherungskopie muss alle Urheber- oder sonstigen Eigentumsvermerke enthalten.
- **Spectrum Scale** ist eine proprietäre Software und nutzt die IBM Lizenz

Lizenzmodell	Storage-Cluster Lösung
GPL	GlusterFS, LizardFS, Lustre, MooseFS, RozoFS, S3QL
Apache Lizenz 2.0	Flocker, HDFS, LeoFS, QuantcastFS, SeaweedFS, Torus
eigene Lizenzbestimmungen	Infinit, ObjectiveFS, Spectrum Scale
BSD Lizenz	Gfarm, iRODS, XtreemFS
LGPL	CephFS, OrangeFS
MIT Lizenz	IPFS
IBM Public Lizenz	OpenAFS
Client: GPL; Server: EULA	BeeGFS

Tabelle 4.4: Auswertung: Lizenzmodelle

Die Integration der Storage-Cluster Lösung in das bestehende Container-Cluster setzt voraus, dass teilweise die Voraussetzungen für das Container-Cluster übernommen werden. Entsprechend spielt das Lizenzmodell der Storage-Cluster Lösung eine entscheidende Rolle bei dessen Auswahl. Quint und Kratzke beschreiben, dass die Nutzung von Open Source Cluster-Managern einen IaaS-Plattform-Wechsel begünstigt, die Abhängigkeit von einem Anbieter vermeidet und den Nutzern ermöglicht, die Software nach ihren Ansprüchen zu modifizieren [QK16]. Aus diesem Grund muss die Storage-Cluster Lösung ebenfalls quelloffen sein, damit ein IaaS-Plattform-Wechsel durchgeführt werden kann. Hieraus ergibt sich die Anforderung ***[M2]*: „Die Software muss quelloffen sein“.**

4.1.6 CAP-Theorem

Bereits im Jahr 2000 teilte Brewer mit, dass es drei systembedingte Anforderungen in jedem verteiltem System gibt, die miteinander direkt in Beziehung stehen [Bre00a]. Nach seiner Aussage kann ein verteiltes System nur zwei der drei Eigenschaften Datenkonsistenz, Verfügbarkeit und Ausfalltoleranz zufriedenstellend zur Verfügung stellen. Gilbert und Lynch entwickelten auf der Basis der Aussage von Brewer das CAP-Theorem [GL02], welches unter anderem die drei Varianten, die ein verteiltes System besitzen kann, beschreibt. Das CAP-Theorem wird häufig als Dreieck dargestellt, um die Abhängigkeiten von der Konsistenz (C), Verfügbarkeit (A) und Ausfalltoleranz (P) zu verdeutlichen (vgl. Abbildung 4.1).

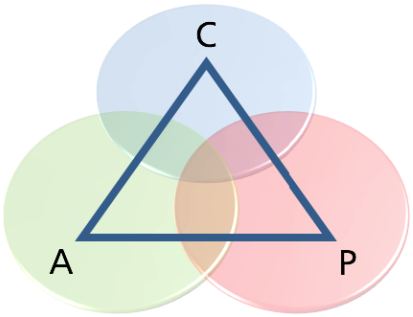

Abbildung 4.1: CAP-Theorem

Es lassen sich in einem massiv verteilten System entweder Konsistenz und Verfügbarkeit (CA), Konsistenz und Ausfalltoleranz (CP) oder Verfügbarkeit und Ausfalltoleranz (AP) kombinieren, aber alle drei sind nicht gleichzeitig kombinierbar. (vgl. Abbildung 4.2).

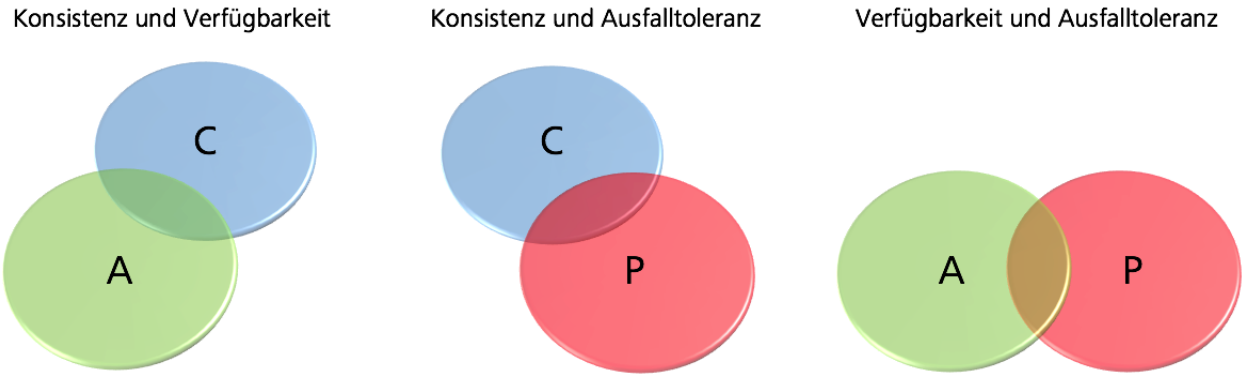

Abbildung 4.2: CAP-Theorem Optionen

Für das Storage-Cluster spielen die jeweiligen Eigenschaften eine entscheidende Rolle.

- Konsistenz (Consistency): Werden Daten im Storage-Cluster gespeichert oder verändert, so steht die Änderung der Daten allen Clients aktuell zur Verfügung.
- Verfügbarkeit (Availibility): Das Storage-Cluster steht ohne Unterbrechung mit tolerablen Zugriffszeiten zur Verfügung.
- Ausfalltoleranz (Partition Toleranz): Kommt es zum Ausfall eines Servers innerhalb der Storage-Clusters zum Beispiel durch einen Hardwaredefekt oder einen Netzwerkausfall, so wird das Storage-Cluster als Gesamtsystem nicht negativ beeinflusst.

Speziell bei Storage-Clustern muss man bedenken, dass mindestens zwei Knoten ihren Status gegenseitig aktualisieren müssen. Da der Vorgang der Aktualisierung Zeit in Anspruch nimmt, sind die Daten der Knoten bis zum Abschluss der Aktualisierung inkonsistent [Bre12]. Wenn die Konsistenz erhalten werden soll, so muss einer der beiden Knoten sich so verhalten, als wenn er nicht verfügbar wäre. In diesem Fall wäre aber die Verfügbarkeit nicht mehr gegeben. Da bei weitverteilten Storage-Clustern die Implementierung von Ausfalltoleranz eine übergeordnete Rolle spielt, müssen sich Entwickler dieser Lösungen häufig zwischen der Konsistenz und Verfügbarkeit entscheiden.

Im Jahr 2000 stellte Brewer das Akronym BASE vor, welches Brewer zusammen mit seinen Kollegen bereits imJahr 1990 entwickelte, um die zu der Zeit aufkommenden Designansätze für Hochverfügbarkeit erfassen zu können [Bre00b]:

- **B**asic **A**vailability bedeutet, dass der Dienst generell hochverfügbar ist, so dass er jederzeit auf Anfragen reagieren kann.
- **S**oft-state ist ein Zwischenzustand bei dem noch nicht alle Knoten alle Änderungen erhalten haben, aber das System trotzdem auf Anfragen reagieren kann.
- **E**ventual Consistency (Eventuelle Konsistenz) bedeutet, dass die Konsistenz nicht nach jeder Transaktion vollständig wieder hergestellt sein muss, sondern es genügt, wenn die Knoten erst nach einer gewissen Zeit einen konsistenten Zustand aufweisen.

Neben der *eventuellen Konsistenz* wurden die beiden Konsistenzmodelle *strenge Konsistenz* und *schwache Konsistenz* betrachtet.

- **Strenge Konsistenz** bedeutet, dass zur Vermeidung von Inkonsistenzen ein transaktionaler Ansatz genutzt wird, der bei einer Änderung entweder keinen oder alle Knoten aktualisiert [KM12]. Bei der Variante sind alle Schreibzugriffe für alle Prozesse sichtbar. Wenn ein lesender Zugriff stattfindet, wird auch immer der Wert des letzten Schreibzugriffes übermittelt. Lösungen mit diesem Modell sind langsamer, da eine ständige Synchronisation stattfinden muss.
- Be der **schwachen Konsistenz** sind inkonsistente Zustände zulässig. Die Reihenfolge, wann die Änderungen an die Knoten weitergegeben werden, ist nicht spezifiziert. Der Einsatz von schwachen Konsistenzmodellen hat gegenüber dem Einsatz von strengen Konsistenzmodellen den Vorteil einer besseren Skalierbarkeit.

Für das Ermitteln des Konsistenzmodells der jeweiligen Storage-Cluster Lösung wurden unter anderem die Erkenntnisse von Venugopal, et al., Pacheco et al. und Tormasov et al. genutzt [VBR06, PHS+16, TLM15]. Neun der untersuchten Storage-Cluster Lösungen nutzen ein strenges Konsistenzmodell. Bei vier Storage-Cluster Lösungen ist die Konsistenz von dem extern genutzten Speicher abhängig und für drei Storage-Cluster Lösungen wurden keine Informationen zur Konsistenz gefunden (vgl. Tabelle 4.5).

Strenge Konsistenz	Storage-Cluster Lösung
ja	BeeGFS, CephFS, GlusterFS, HDFS, Lustre, MooseFS, QuantcastFS, Spectrum Scale, XtreemFS
nein	Gfarm, Infinit, IPFS, LeoFS, OpenAFS, OrangeFS, SeaweedFS
vom externen Speicher abhängig	Flocker, iRODS, ObjectiveFS, S3QL
konnte nicht ermittelt werden	LizardFS, RozoFS, Torus

Tabelle 4.5: Auswertung: Konsistenzmodell

Bei einem Multi-Compute-Cluster werden Container wahlweise gestoppt und gestartet. Die gespeicherten Daten müssen zwingend konsistent sein. Zur Vermeidung von Inkonsistenzen muss ein Storage-Cluster mit einem strengen Konsistenzmodell ausgewählt werden. Hieraus ergibt sich die Anforderung ***[M3]*:** „Die Software muss ein strenges Konsistenzmodell besitzen“.

4.1.7 Fehlertoleranz

Bei dem Betrieb eines verteilten Systems sind Hard- und Softwarefehler unvermeidbar. Damit das System bei auftretenden Fehlern weiterhin funktioniert, müssen sich die Entwickler bereits in einer frühen Entwicklungsphase damit auseinandersetzen, mit welchen Verfahren die Fehlertoleranz realisiert werden kann. Aviziens definiert ein fehlertolerantes System, als ein System *“[...] which has the built-in capability*

(without external assistance) to preserve the continued correct execution of its programs and input/output (I/O) functions in the presence of a certain set of operational faults“ [Avi76]. Über diese eingebauten Funktionalitäten muss auch eine Storage-Cluster Lösung verfügen, damit der Storage-Dienst auch dann angeboten werden kann, während Hardware- oder Softwarefehler im System vorhanden sind. Im besten Fall reagiert die Storage-Cluster Lösung selbständig auf eine Vielzahl an möglichen Fehlerszenarien. Umso höher die Fehlertoleranz ist, umso besser eignet sich die Storage-Cluster Lösung für die Integration in die elastische Container Plattform. Würde eine Storage-Cluster Lösung bereits bei jedem einfachen Fehler (z.B. Ausfall einer Festplatte) den Dienst einstellen, so müsste ein Administrator jedes Mal direkt reagieren und den Dienst wieder herstellen. um die Ausfallzeiten gering zu halten. Dies würde dazu führen, dass die Storage-Cluster Lösung nicht kostengünstig und zuverlässig skalieren würde [Ham07].

Nachdem eine fehlertolerante Storage-Cluster Lösung in Betrieb genommen wurde, reicht es nicht aus, sich auf dessen Fehlertoleranz-Mechanismen zu verlassen, sondern die Fehlertoleranz muss regelmäßig getestet werden. Hierfür muss neben dem sanften Ausschalten von Teilsystemen auch das harte Abschalten (z.B. Unterbrechung der Stromversorgung) getestet werden. Während der Tests darf es idealerweise zu keinen Leistungseinbußen kommen.

Die vier Storage-Cluster Lösungen Flocker, iRODS, ObjectiveFS und SQ3L sind in Bezug auf die Fehlertoleranz von den extern eingebundenen Speichern abhängig, so dass zu dessen Fehlertoleranz keine Aussage getroffen werden kann. Des Weiteren wurde ermittelt, dass die Storage-Cluster Lösungen OrangeFS und SeaweedFS keine Fehlertoleranz-Mechanismen besitzen. 17 der untersuchten Storage-Cluster Lösungen besitzen Fehlertoleranz-Mechanismen. (Tabelle 4.6).

Fehlertoleranz	**Storage-Cluster Lösung**
ja	BeeGFS, CephFS, Gfarm, GlusterFS, HDFS, Infinit, IPFS, LeoFS, LizardFS, Lustre, MooseFS, OpenAFS, Spectrum Scale, QuantcastFS, RozoFS, Torus, XtreemFS
nein	OrangeFS, SeaweedFS
vom externen Speicher abhängig	Flocker, iRODS, ObjectiveFS, S3QL

Tabelle 4.6: Auswertung: Fehlertoleranz

Beim Betrieb des Storage-Clusters zusammen mit dem Container-Cluster sind Hard- und Softwarefehler nicht auszuschließen. Das Storage-Cluster muss in der Lage sein entsprechend auf die Fehler zu reagieren, um den Verlust von Daten einzelner Container zu vermeiden. Aus diesem Grund muss die Storage-Cluster Lösung fehlertolerant sein. Hieraus ergibt sich die Anforderung ***[M4]*****:** „Die Software muss fehlertolerant sein“.

4.1.8 Datenkomprimierung

Bei der Datenkomprimierung findet eine Umwandlung der Daten mit Hilfe von Algorithmen statt [HHDL15]. Das Ziel der Datenkomprimierung ist die Menge des benötigten Speicherplatzes von Daten zu reduzieren, Ein weiterer positiver Effekt der Datenkomprimierung ist die höhere Übertragungsgeschwindigkeit, da die Daten weniger Speicherplatz benötigen. Bei der Datenkomprimierung werden zusätzliche CPU-Ressourcen benötigt. Stehen diese nicht ausreichend zur Verfügung stehen, kann es bei einer Datenkomprimierung zur Verlangsamung des Gesamtsystems führen.

Von den untersuchten Storage-Cluster Lösungen ermöglichen neun eine Datenkomprimierung (vgl. Tabelle 4.7). Bei drei Storage-Cluster Lösungen ist die Datenkomprimierung von dem externen Speicher abhängig.

Datenkomprimierung	Storage-Cluster Lösung
ja	GlusterFS, HDFS, Infinit, IPFS, Lustre, OrangeFS, SeaweedFS, S3QL, Spectrum Scale
nein	BeeGFS, CephFS, Gfarm, LeoFS, LizardFS, MooseFS, OpenAFS, QuantcastFS, RozoFS, Torus, XtreemFS
Vom externen Speicher abhängig	Flocker, iRODS, ObjectiveFS

Tabelle 4.7: Analyse: Datenkomprimierung

Die Datenkomprimerung ist keine Eigenschaft, die das Storage-Cluster zwingend zur Verfügung stellen muss, da das Container-Cluster nicht das Ziel verfolgt, möglichst viele Daten in einem möglichst geringen Speicherplatz unterzubringen. Somit ergibt sich aus der Eigenschaft Datenkomprimierung keime Anforderung für die Bewertung der Storage-Cluster Lösungen.

4.1.9 Datendeduplizierung

Die Datendeduplizierung ist eine Funktion, um Datenduplikate zu identifizieren und zu löschen [SGLM08]. Die Funktion verfolgt das Ziel, den Bedarf an Speicherplatz und der notwendigen Datenrate zur Übertragung der Daten zu reduzieren. Speichersysteme basieren häufig auf einem der drei folgenden Deduplizierungsstrategien:

- Die **Deduplizierung auf der Dateiebene** verwendet den Hash-Wert der Datei als ihren Bezeichner. Wenn zwei oder mehrere Dateien denselben Wert haben, wird davon ausgegangen, dass sie identische Inhalte haben und nur einmal gespeichert werden (ohne redundante Kopien).
- Bei der **Deduplizierung auf der Blockebene mit fester Länge** werden die Dateien vor der Deduplizierung in feste Blöcke unterteilt, sodass Dateien, die einen identischen Inhalt teilen, ressourcenschonend gespeichert werden können.
- Die flexibelste Form ist die **Deduplizierung auf der Blockebene mit variabler Länge**. Bei dieser Variante werden die Dateien in Chunks mit variabler Länge zerlegt.

Die Untersuchung hat ergeben, dass fünf Storage-Cluster Lösungen die Datendeduplizierung unterstützen (vgl. Tabelle 4.8). Bei vier Storage-Cluster Lösungen ist die Datendeduplizierung von dem genutzten externen Speicher abhängig.

Datendeduplizierung	Storage-Cluster Lösung
ja	HDFS, IPFS, Lustre, S3QL, Spectrum Scale
nein	BeeGFS, CephFS, Gfarm, GlusterFS, Infinit, LeoFS, LizardFS, MooseFS, OpenAFS, OrangeFS, QuantcastFS, RozoFS, SeaweedFS, Torus, XtreemFS
Vom externen Speicher abhängig	Flocker, iRODS, ObjectiveFS

Tabelle 4.8: Analyse: Datendeduplizierung

Die Datendeduplizierung ist für die Integration des Storage-Clusters in das elastische Container-Cluster nicht zwingend erforderlich, da nicht das Ziel verfolgt wird, möglichst viele Daten in dem zur Verfügung stehenden Speicherplatz unterzubringen. Somit ergibt sich aus der Eigenschaft Datendeduplizierung keime Anforderung für die Bewertung der Storage-Cluster Lösungen.

4.1.10 Geo-Replikation

Bei der Nutzung von Replikationsalgorithmen innerhalb einer Storage-Cluster Lösung erhöhen sich nicht nur dessen Verfügbarkeit und Zuverlässigkeit, sondern die Leistung des Systems wird durch eine bessere Lastverteilung gesteigert und die Netzwerkdatenrate innerhalb des verteilten Systems wird gesenkt [AFS+10]. Besonders bei Storage-Cluster Lösungen, die aus mehreren geographisch weit verteilten Servern besteht, sorgt eine lokal verfügbare Replikation der Daten für schnellere Zugriffszeiten [TVS07].

Es gibt zwei Arten von Replikationsalgorithmen [SCG+12]:

- Bei einem statischen Replikationsalgorithmus steht die Replikationsstrategie im Vorwege fest und diese lässt sich im Betrieb nicht mehr ändern.
- Ein dynamischer Replikationsalgorithmus erstellt und löscht abhängig von den sich ändernden Zugriffsmustern automatisch Replikate.

Caching ist eine Form der Replikation von Daten [TVS07]. Beim Caching wird wie bei der Replikation eine Kopie der Daten erstellt. Der Unterschied ist aber, dass das Caching meist clientseitig bei Bedarf genutzt wird. Dagegen wird die Replikation von Daten häufig planmäßig durchgeführt. Umso höher die Anzahl von Replikations- und Cachingalgorithmen in einer Storage-Cluster Lösung ist, umso höher ist die Wahrscheinlichkeit, dass Dateninkonsistenzen entstehen können. Wenn zum Beispiel eine Datei in der Storage-Cluster Lösung aktualisiert wird, muss anschließend sichergestellt sein, dass alle Replikate ebenfalls die aktualisierte Datei erhalten. Ansonsten käme es zu unterschiedlichen Datenbeständen. Damit keine Dateninkonsistenzen entstehen, müssen alle Replikate zeitnah synchronisiert werden. Die Synchronisation der Replikate erfordert eine häufige Kommunikation der einzelnen Server untereinander, welches wiederum zu einer höheren Auslastung der Netzwerkbandbreite führt und die Gesamtleistung des Systems verringert. Ein Ansatz besteht darin, die häufige Synchronisation der Server untereinander zu minimieren, um die Leistungsfähigkeit des Systems zu steigern. Hierbei muss in Kauf genommen werden, dass die Kopien der Daten nicht auf allen Servern zur gleichen Zeit identisch sind.

Geo-Replikation ist ein Mechanismus um Daten zwischen geographisch entfernten Rechenzentren zu synchronisieren [LPC+12]. Bei der Analyse wurde untersucht, ob eine Storage-Cluster Lösung Geo-Replikation unterstützt oder ob sie dies nicht unterstützt. Das Ergebnis zeigt, dass mit 13 der untersuchten Storage-Cluster Lösungen eine Geo-Replikation möglich ist (vgl. Tabelle 4.9). Bei vier der untersuchten Storage-Cluster Lösungen ist die Geo-Replikation vom externen Speicher abhängig und nur möglich, wenn dieser die Funktion auch unterstützt.

Geo-Replikation	Storage-Cluster Lösung
ja	CephFS, Gfarm, GlusterFS, Spectrum Scale, IPFS, LeoFS, LizardFS, Lustre, QuantcastFS, RozoFS, SeaweedFS, Torus, XtreemFS
nein	BeeGFS, HDFS, Infinit, MooseFS, OpenAFS, OrangeFS
vom externen Speicher abhängig	Flocker, iRODS, ObjectiveFS, S3QL

Tabelle 4.9: Auswertung: Geo-Replikation

Geo-Replikation ermöglicht nicht nur Hochverfügbarkeit und kann Performance-Vorteile erzeugen. Sie wird zudem zwingend für eine Migration der Cloud Plattform benötigt. Da die Migrierbarkeit ein Hauptanforderung an das Storage-Cluster ist, ergibt sich die Anforderung ***[M5]*****:** „Die Software muss Geo-Replikation unterstützen“.

4.1.11 Objektspeicher

Ein Objektspeicher ist eine Datenspeicherarchitektur, die im Gegensatz zum Dateispeicher die Datenverwaltung auf Objektebene ausführt (vgl. Abschnitt 2.3.3). Gegenüber dem Dateispeicher besitzt der Objektspeicher Vorteile beim Umgang mit großen unstrukturierten Datenmengen. Die Objektspeicher-Architektur ermöglicht die Handhabung von nahezu unbegrenzten Speicherplatz, einen hochskalierbaren Lese- / Schreibzugriff und die Möglichkeit, den Inhalt direkt aus dem Storage-Cluster an die Clients zu liefern [JGG15].

Das Ergebnis der Untersuchung zeigt, dass die wenigsten der untersuchten Storage-Cluster Lösungen einen Objektspeicher zur Verfügung stellen (vgl. Tabelle 4.10). Nur neun Storage-Cluster Lösungen bieten einen Objektspeicher an.

Objektspeicher	Storage-Cluster Lösung
ja	CephFS, Flocker, Infinit, iRODS, LeoFS, Lustre, ObjectiveFS, S3QL, XtreemFS
nein	BeeGFS, Gfarm, GlusterFS, HDFS, IPFS, LizardFS, MooseFS, OpenAFS, OrangeFS, QuantcastFS, RozoFS, SeaweedFS, Torus, Spectrum Scale

Tabelle 4.10: Auswertung: Objektspeicher

Die in der Analyse festgestellten Vorteile des Objektspeichers sollen auch von der gesuchten Storage-Cluster Lösung zur Verfügung gestellt werden. Hieraus ergibt sich die Anforderung ***[S1]*: „Die Software soll Objektspeicher unterstützen“.**

4.1.12 Verschlüsselte Datenübertragung

Die Analyse der Storage-Cluster Lösungen hat ergeben, dass Lösungen vorhanden sind, die eine Verschlüsselung der übertragenen Daten zur Verfügung stellen. Die Ziele der Verschlüsselung der Daten sind folgende [Mül11].

- **Vertraulichkeit:** Der Inhalt einer Datei soll vor unberechtigten Zugriff geschützt werden.
- **Integrität:** Der Inhalt einer Datei soll nicht durch Angreifer verändert werden können.
- **Authentizität:** Die Authentizität der übertragenden Daten ist die sichere Zuordnung zum Sender und der Nachweis, dass die Information nicht nachträglich verändert wurde.

In acht der 23 untersuchten Storage-Cluster Lösungen ist eine verschlüsselte Datenübertragung implementiert (vgl. Tabelle 4.11). Bei zwei Storage-Cluster Lösungen ist die verschlüsselte Datenübertragung von dem verwendeten externen Speicher abhängig.

Verschlüsselte Datenübertragung	Storage-Cluster Lösung
ja	GlusterFS, HDFS, Infinit, Lustre, ObjectiveFS, OpenAFS, S3QL, Spectrum Scale
nein	BeeGFS, CephFS, Gfarm, IPFS, LeoFS, LizardFS, MooseFS, OrangeFS, QuantcastFS, RozoFS, SeaweedFS, Torus, XtreemFS
vom externen Speicher abhängig	Flocker, iRODS

Tabelle 4.11: Auswertung: Verschlüsselte Datenübertragung

Der Austausch von Daten zwischen den Nodes eines Storage-Clusters findet bei der Migration über ein ungesichertes Netzwerk statt. Da über dieses Netzwerk unter Umständen sensible Daten übertragen werden, müssen entsprechende Sicherheitsmechanismen implementiert werden. Der Einsatz von hardwarebasierten Layer-2 Verschlüsselungsmethoden kann nicht eingesetzt werden, da die Übertragung in Rechenzentren von Dritten zur Verfügung gestellt wird und so eine Ende-zu-Ende Verschlüsselung nicht ohne einen zusätzlichen Aufwand zustande kommt. Des Weiteren würden VPN-Lösungen meist nur Teilstrecken abbilden können, aber nicht durchgehend bis zum letzten Netzwerkport verfügbar sein. Eine Alternative stellt die Verschlüsselung des Datenverkehrs durch die Storage-Cluster Lösung dar. Hier sind die Möglichkeiten, den Datenverkehr unverschlüsselt abzugreifen, sehr gering. Daraus ergibt sich die Anforderung ***[S2]*: „Die Software soll eine verschlüsselte Datenübertragung ermöglichen“.**

4.1.13 Verbindung zum Container-Cluster

In der aktuellen Version des Programms ECP-Deploy basieren die Nodes der Kubernetes Container-Cluster Plattform auf der Container-Betriebssystem CoreOS[1]. Somit ergibt sich für die Auswahl des Storage-Clusters die Anforderung, dass dieses entweder in einem Docker Container unter CoreOS lauffähig ist oder dass dieses als eigenständiges Storage-Cluster funktioniert und eine Verbindung per Schnittstelle zur Kubernetes Container-Cluster Plattform oder zu CoreOS herstellt. Um Daten austauschen zu können, muss das Storage-Cluster eine Möglichkeit zur Verfügung stellen, mit der eine Verbindung zum Container-Cluster hergestellt werden kann. Für die Verbindung zum Container-Cluster wurden folgende Möglichkeiten recherchiert:

- Ein **Kubernetes Volume Plugin**[2] stellt die Verbindung von einem Pod zu einem Volume her. Solange nur die Container in einem Pod neugestartet werden, bleibt auch das Volume bestehen. Wenn ein Pod aufhört zu existieren, wird auch das Volume zerstört. Anders ist es, wenn das Volume in einem Storage-Cluster liegt. In dem Fall bleibt das Volume auch beim Beenden eines Pods bestehen.
- Ein **Docker Volume Plugin**[3] ermöglicht das Mounten eines Volumes in einen Container. Verglichen zu den Pods wird das *normale* Volume beim Beenden eines Containers ebenfalls beendet. Der Vorteil des Storage-Clusters besteht darin, dass die Daten beim Beenden des Containers erhalten bleiben.
- CoreOS unterstützt das Einbinden eines **Network File Systems (NFS)**. Für die Konfiguration muss die beim Starten genutzte CoreOS Cloud-Config Datei angepasst werden (vgl. Listing 4.1).

```
1  systemd:
2    units:
3     - name: rpc-statd.service
4       enable: true
5     - name: var-www.mount
6       enable: true
7       contents: |
8         [Mount]
9         What=nfs.example.com:/var/www
10        Where=/var/www
11        Type=nfs
```

Listing 4.1: CoreOS NFS Mount

[1] https://coreos.com/
[2] https://kubernetes.io/docs/concepts/storage/volumes
[3] https://docs.docker.com/engine/extend/legacy_plugins/#volume-plugins

- CoreOS ermöglicht das Einbinden von Ceph RADOS Block Device Images (vgl. Abschnitt 3.2.2). Für das Einbinden ist eine Anpassung der CoreOS Cloud-Config Datei notwendig (vgl. Listing 4.2).

```
1  - path: /opt/bin/rbd
2    permissions: '0775'
3    content: |
4      #!/bin/sh
5      exec docker run -v /dev:/dev -v /sys:/sys --net=host --privileged=true -v /etc/ceph:/etc/ceph
           ceph/rbd $@
```

Listing 4.2: CoreOS RBD Mount

Anhand der Dokumentation der jeweiligen Storage-Cluster Lösung wurde ermittelt, ob diese parallele Verbindungen zulässt. Die Untersuchung ergab, dass 15 der 23 Storage-Cluster Lösungen die Möglichkeit bieten, mehrere parallele Verbindungen zum vorhandenen Container-Cluster herzustellen.

	Parallele Verbindung möglich	**Kubernetes Volume Plugin**	**Docker Volume Plugin**	**NFS Mount**	**RBD Mount**
BeeGFS	x		x	x	
CephFS	x	x			x
Flocker	x		x		
Gfarm	x			x	
GlusterFS	x	x	x	x	
HDFS	x			x	
Infinit	x		x		
iRODS					
IPFS	x		x		
LeoFS	x			x	
LizardFS					
Lustre					
MooseFS					
ObjectiveFS					
OpenAFS	x			x	
OrangeFS					
QuantcastFS					
RozoFS	x			x	
S3QL					
SeaweedFS					
Spectrum Scale	x	x	x	x	
Torus	x	x	x		
XtreemFS	x			x	

Tabelle 4.12: Auswertung: Verbindung zum Container-Cluster

Die Pods, Container oder CoreOS-Nodes müssen auf den Datenbestand des Storage-Clusters gemeinsam zugreifen können. Damit dies ermöglicht wird, müssen beim Start des Programms ECP-Deploy mehrere Verbindungen zum Storage-Cluster hergestellt werden können. Hieraus ergibt sich die Anforderung ***[M6]***: „Die Software muss mehrere parallele Verbindungen zum Container-Cluster herstellen können“.

4.2 Zusammenfassung der Anforderungen

Die in dem vorherigen Abschnitten analysierten und begründeten Anforderungen werden an dieser Stelle zusammenfasend dargestellt:

- ***[M1]*** Die Software muss produktiv einsetzbar sein.
- ***[M2]*** Die Software muss quelloffen sein.
- ***[M3]*** Die Software muss ein strenges Konsistenzmodell besitzen.
- ***[M4]*** Die Software muss fehlertolerant sein.
- ***[M5]*** Die Software muss Geo-Replikation unterstützen.
- ***[M6]*** Die Software muss mehrere parallele Verbindungen zum Container-Cluster herstellen können.
- ***[S1]*** Die Software soll Objektspeicher unterstützen.
- ***[S2]*** Die Software soll eine verschlüsselte Datenübertragung ermöglichen.

Bei der Anforderungen wird zwischen „**M**uss“- und „**S**oll“-Anforderungen unterschieden. Wenn eine Storage-Cluster Lösung nur eine der „Muss“-Anforderungen nicht erfüllt, so ist diese nicht für die Integration in die elastische Container Plattform einsetzbar. „Muss“-Anfoderungen sind daher zwingend notwendig, damit die Integration möglich ist. Erfüllt eine Storage-Cluster Lösung eine „Soll“-Anfoderung nicht, so ist es dennoch weiterhin für die Integration nutzbar.

4.3 Auswertung der Storage-Cluster Eigenschaften

Die Analyse hat gezeigt, dass die Storage-Cluster Lösungen, welche auf einer Storage-Backend-Architektur basieren, in vielen Fällen eine Bewertung nicht zulassen, da ein Teil der Eigenschaften von dem extern genutzten Speicher abhängig ist. Aus diesem Grund wurden zwei voneinander getrennte Auswertungen durchgeführt.

4.3.1 Storage-Cluster mit Client-Server- und Peer-to-Peer-Architektur

Das Ergebnis der Auswertung der Storage-Cluster Lösungen ist in Tabelle 4.13 dargestellt. Da für LizardFS, RozoFS und Torus keine strengen Konsistenzmodelle ermittelt werden konnten, wurde in der Auswertung davon ausgegangen, dass diese zum Zeitpunkt der Auswertung nicht vorhanden waren (vgl. „Muss“-Anforderung M3). Die Storage-Cluster Lösungen CephFS, GlusterFS, LeoFS, und XtreemFS erfüllen die „Muss“-Anforderungen und zusätzlich die „Soll“-Anforderung S1. Sie sind bezüglich der Auswertung als gleichwertig zu betrachten.

	M1	M2	M3	M4	M5	M6	„Muss“-Anforderungen erfüllt	S1	S2	Erfüllungsgrad
BeeGFS	x	x		x		x	nein			50,0 %
CephFS	x	x	x	x	x	x	ja	x		87,5 %
Gfarm	x	x		x	x	x	nein			62,5 %
GlusterFS	x	x	x	x	x	x	ja		x	87,5 %
HDFS	x	x		x		x	nein		x	62,5 %
Infinit			x	x		x	nein	x	x	62,5 %
IPFS		x		x	x	x	nein			62,5 %
LeoFS	x	x	x	x	x	x	ja	x		87,5 %
LizardFS	x	x	o	x	x		nein			50,0 %
Lustre	x	x	x	x	x		nein	x	x	87,5 %
MooseFS	x	x		x			nein			37,5 %
OpenAFS	x	x		x		x	nein		x	62,5 %
OrangeFS	x	x					nein			25,0 %
QuantcastFS	x	x		x	x		nein			50,0 %
RozoFS	x	x	o	x	x	x	nein			62,5 %
SeaweedFS		x			x		nein			25,0 %
Spectrum Scale	x			x	x	x	nein		x	62,5 %
Torus		x	o	x	x	x	nein			50,0 %
XtreemFS	x	x	x	x	x	x	ja	x		87,5 %

o = konnte nicht ermittelt werden

Tabelle 4.13: Auswertung der Storage-Cluster mit Client-Server- und Peer-to-Peer-Architektur

4.3.2 Storage-Cluster mit Storage-Backend-Architektur

Für die Storage-Cluster Lösungen mit Storage-Backend-Architektur wurde eine separate Auswertung durchgeführt, da die folgenden drei der acht Anforderungen aufgrund der Nutzung des externen Storage-Backends nicht ausgewertet werden konnten:

- ***[M3]*** Die Software muss ein strenges Konsistenzmodell besitzen.
- ***[M4]*** Die Software muss fehlertolerant sein.
- ***[M5]*** Die Software muss Geo-Replikation unterstützen.

Das Ergebnis in Tabelle 4.14 zeigt, dass von den vier ausgewerteten Storage-Cluster Lösungen iRODS, ObjectiveFS und S3QL nicht genutzt werden können, da sie keine parallelen Verbindungen zum Container-Cluster aufbauen können. Zudem erfüllt ObjectiveFS nicht die „Muss“-Anforderung *Die Software muss quelloffen sein*. Für die prototypische Integration der Storage-Cluster Lösung steht nur Flocker zur Verfügung.

	M1	M2	M6	„Muss“-Anforderungen erfüllt	S1	S2	Erfüllungsgrad
Flocker	x	x	x	ja	x		80,0 %
iRODS	x	x		nein	x		60,0 %
ObjectiveFS	x			nein	x	x	60,0 %
S3QL	x	x		nein	x	x	80,0 %

Tabelle 4.14: Auswertung der Storage-Cluster mit Storage-Backend-Architektur

4.3.3 Fazit

Das Ergebnis der Untersuchung der Eigenschaften der Storage-Cluster zeigt, dass fünf Storage-Cluster für die Integration geeignet sind. Da im Rahmen dieser Arbeit drei Storage-Cluster Lösungen integriert werden sollten, musste eine Auswahl zwischen den fünf möglichen Storage-Cluster Lösungen stattfinden. Es wurden die Storage-Cluster Lösungen CephFS, Flocker und GlusterFS für die prototypische Integration ausgewählt.

Kapitel 5

Integration der Storage-Cluster Lösung

5.1 Umsetzungsmethoden

Die drei Umsetzungsmethoden beschreiben, wie eine Storage-Cluster Lösung in elastische Container Plattformen integriert werden kann. Grundlage für die Betrachtung ist die vorherige Analyse des Programms ECP-Deploy (vgl. Abschnitt 2.2) und die Möglichkeit eine Verbindung zum Container-Cluster herstellen zu können (vgl. Abschnitt 4.1.13). Für Implementierung von Flocker konnte nicht zwischen den Umsetzungsmethoden gewählt werden, da die Umsetzung durch die Architektur von Flocker bereits vorgegeben ist (vgl. Abschnitt 3.2.3).

5.1.1 Umsetzungsmethode 1: Gemeinsam genutzte Container-Cluster Plattform

Die Compute Container und die Storage Container nutzen eine gemeinsame Cluster Plattform (vgl. Abbildung 5.1). Das Storage-Cluster wird als Container-Anwendung innerhalb der Cluster Plattform betrieben. Die Verbindung untereinander wird entweder per Kubernetes Volume Plugin oder per Docker Volume Plugin hergestellt. Für das Erstellen der Container-Cluster Plattform wird in der aktuellen

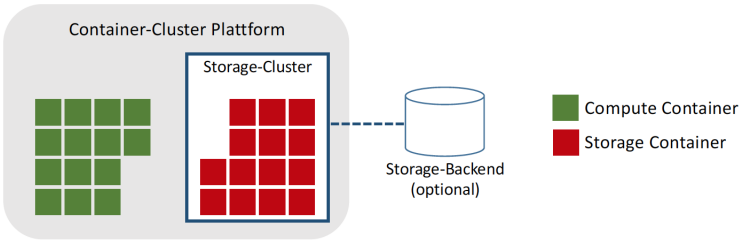

Abbildung 5.1: Gemeinsam genutzte Cluster Plattform

Deployer Version Kubernetes genutzt. Bei dieser Umsetzungsmethode können die Nodes des Compute-Cluster und des Storage-Clusters auf den Instanzen koexistieren. Die vorhandene Container-Cluster Plattform des Deployers basiert auf dem Betriebssystem CoreOS. Eine Verbindung von Storage-Cluster zu einem Storage-Backend ist optional möglich..

5.1.2 Umsetzungsmethode 2: Getrennte Cluster mit zwei Container-Cluster Plattformen

Bei dieser Variante wird das Compute-Cluster und Storage-Cluster in zwei voneinander getrennten Container-Cluster Plattformen zur Verfügung gestellt (vgl. Abbildung 5.2). Die Daten des Compute-

Clusters werden dabei in dem Storage-Cluster gespeichert. Das Compute-Cluster greift auf das Storage-Cluster per Volume Plugin zu. In der aktuellen Anwendung kann für den Datenaustausch zwischen Containern und dem Storage-Cluster entweder ein Docker Volume Plugin oder ein Kubernetes Volume Plugin genutzt werden. Wie in Abschnitt 2.2 beschrieben, startet der Deployer zu Beginn virtuelle

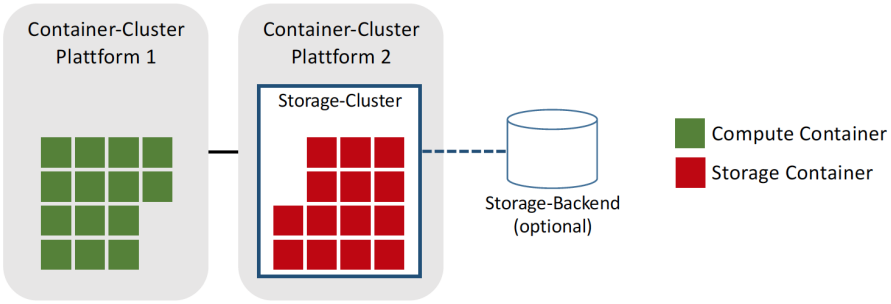

Abbildung 5.2: Getrennte Cluster mit zwei Cluster Plattformen

Instanzen mit dem Container-Betriebssystem CoreOS. Basierend auf den CoreOS Instanzen wird eine Kubernetes Container-Cluster Plattform für das Compute-Cluster erstellt. Mit dem Deployer werden zwei separate Container-Cluster erstellt. Das eine wird als Compute-Cluster verwendet. Auf dem zweiten wird das Storage-Cluster erstellt. Eine Verbindung von Storage-Cluster zu einem Storage-Backend ist optional möglich..

5.1.3 Umsetzungsmethode 3: Storage-Cluster außerhalb einer Container-Cluster Plattform

Das Compute-Cluster und das native Storage-Cluster sind in dieser Variante zwei voneinander getrennte Systeme (vgl. Abbildung 5.3). Der Datenzugriff erfolgt per Docker Volume Plugin oder Kubernetes Volume Plugin. Gegenüber den beiden anderen Umsetzungsmethoden ermöglicht diese das Erstellen

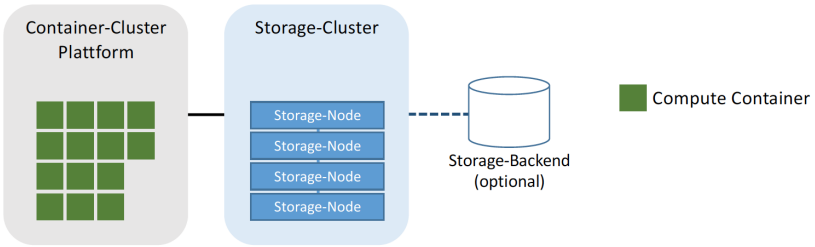

Abbildung 5.3: Compute Cluster und Storage Cluster getrennt

eines nativen Storage-Clusters auf separaten Storage-Nodes (virtuellen Instanzen). Eine Verbindung von Storage-Cluster zu einem Storage-Backend ist optional möglich..

5.1.4 Auswahl der Umsetzungsmethode

Grundsätzlich besteht die Möglichkeit, mit Hilfe eines Kubernetes Volume Plugins oder ein Docker Volume Plugins eine Verbindung vom Compute-Cluster zum Storage-Cluster mit Client-Server- und Peer-to-Peer-Architektur herzustellen. Die Recherche ergab, dass für CephFS und GlusterFS jeweils ein Kubernetes Volume Plugin zur Verfügung steht. Bei allen drei Umsetzungsmethoden können die Kubernetes Volume Plugins nicht genutzt werden, da die Kubernetes Nodes bzw. die CoreOS Nodes jeweils die Clients der Storage-Cluster Lösungen nicht als Basispaket (Client Package) installiert haben.

Diese werden benötigt, damit kubelet auf der Kommandozeile des Host-Betriebssystems die Storage-Cluster Befehle aufrufen kann. Als Container-Betriebssystem unterstützt CoreOS nur die Installation der Pakete als Docker Container, wodurch die Befehle des jeweiligen Storage-Clusters nicht direkt in der Kommandozeile des CoreOS Betriebssystems aufgerufen werden können. Ein Docker Volume Plugin steht nur für GlusterFS zur Verfügung. Dies wurde nicht genutzt, da die Entwicklung des Plugins gestoppt wurde.

Um für die erste und die zweite Umsetzungsmethode geeignete Docker-Images identifizieren zu können, wurde eine Recherche bei dem Online-Dienst Docker Hub durchgeführt. Der Dienst stellt eine Registry für Docker-Images zur Verfügung, die von jedem genutzt werden kann um eigene selbst erstellte Docker-Images hochzuladen und mit anderen Nutzern zu teilen. Die recherchierten Docker-Images (vgl. Tabelle B.1) wurden mit dem CoreOS Betriebssystem getestet. Durch die Tests stellte sich heraus, dass sich die Docker-Images für Storage-Cluster noch in der Entwicklung befinden und im Zusammenhang mit dem Betriebssystem CoreOS nicht für produktive Umgebungen einsetzbar sind. Zum Beispiel wurde beim Test von dem Docker-Image *ceph/daemon* [1] festgestellt, dass nach dem Neustart eines ceph-osd Containers die Funktion des Storage-Clusters stark eingeschränkt ist. Durch die Weiterentwicklung der containerisierten Storage-Cluster wird es in Zukunft produktiv einsetzbare Lösungen geben.

Für die Umsetzung im Rahmen diese Arbeit wurde die dritte Umsetzungsmethode *Storage-Cluster außerhalb einer Container-Cluster Plattform* gewählt. Sie bietet die Möglichkeit, die Storage-Cluster CephFS und GlusterFS einzubinden:

- GlusterfFS stellt per NFS eine Verbindung zum Container-Betriebssystem CoreOS her.
- Bei CephFS wird ein RBD-Image in das Container-Betriebssystem CoreOS eingebunden.

5.2 Erweiterung des Deployers

Der Deployer ist ein in der Entwicklung befindliches Proof-Of-Concept-System. Das Nachfolgesystem soll in der Microservice-Architektur erstellt werden, in der die Storage-Engine entsprechend separiert ist. Im Rahmen dieser Arbeit wurde aus Gründen der Effektivität die Storage-Engine in das Proof-of-Concept-System integriert. Beim Starten des Deployers werden mehrere Befehle sequentiell abgearbeitet (vgl. Abbildung 2.3). Das Erstellen des Storage-Cluster kann nicht an einer beliebigen Stelle des Startvorgangs eingefügt werden, da bei den Storage-Cluster Lösungen CephFS und GlusterFS eine Verbindung vom Container-Cluster zum Storage-Cluster nur möglich ist, wenn dieses auch vorab gestartet wird. Zudem müssen die IP-Adressen des Storage-Clusters in die Cloud-Config der CoreOS Instanzen eingefügt werden. Dazu wurde der identifizierte Prozess *Erstellen eines Container-Clusters* angepasst, so dass das Erstellen des Storage-Clusters zwischen dem Erstellen der Security Groups und dem Erstellen der Cluster-Konfiguration eingefügt wurde (vgl. Abbildung 5.4).

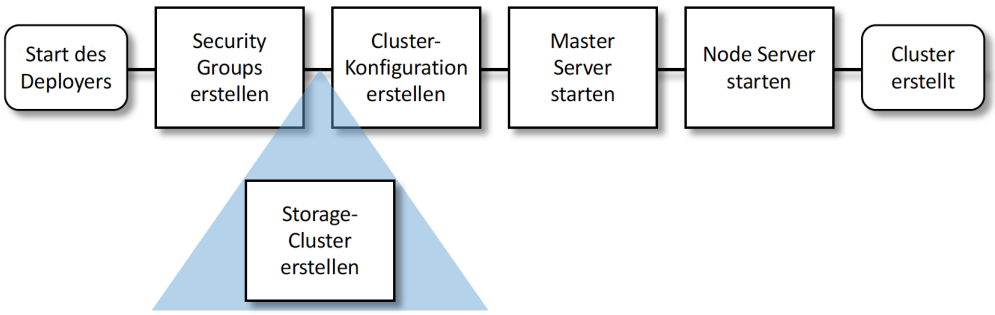

Abbildung 5.4: Angepasster Prozess *Erstellen eines Container-Clusters*

[1] https://hub.docker.com/r/ceph/daemon/

Ein ausführliches Flussdiagramm ist in der Abbildung D.3 dargestellt.

Eine weitere Herausforderung bestand darin, eine Möglichkeit zu finden, mit der es dem Deployer ermöglicht wird, Befehle an die Storage-Cluster Server zu senden. Hierfür wurde das Netzwerkprotokoll Secure Shell (SSH) gewählt.

Die Umsetzung mit den angepassten und den neu hinzugefügten Klassen im Anhang D beschrieben. Die Abbildung 5.5 zeigt ein Klassendiagramm, in dem neu hinzugefügte Klassen grün, angepasste Klassen orange und unveränderte Klassen weiß dargestellt sind. Ein ausführliches Klassendiagramm zeigt die Abbildung D.1.

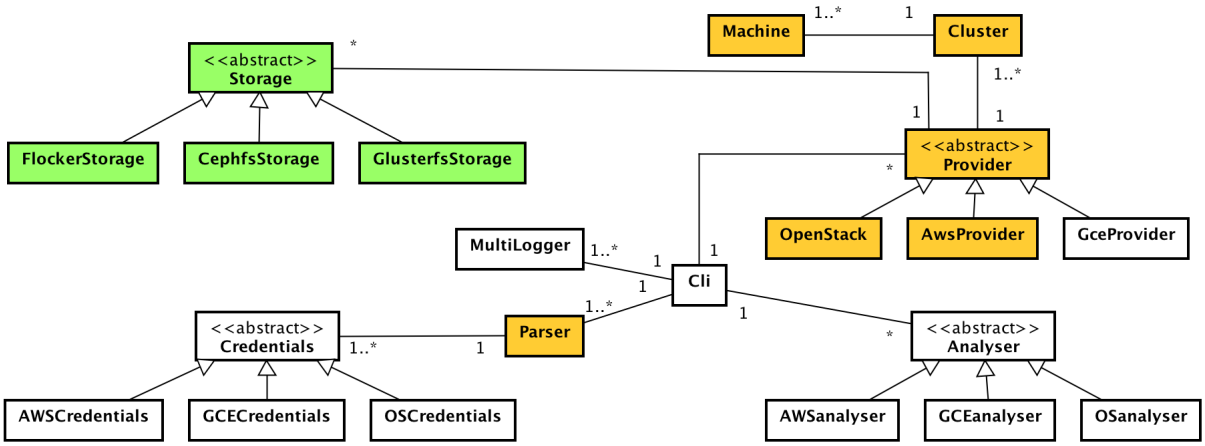

Abbildung 5.5: Deployer Klassendiagramm

Durch die ständige Weiterentwicklung des Deployers wächst dessen Quellcode und es wird zunehmend aufwendiger, neue Funktionalitäten zu ergänzen. Aus diesem Grund wurde darauf geachtet, dass die Funktionen für die Storage-Cluster so implementiert werden, so dass nachträglich weitere Storage-Cluster Lösungen ergänzt werden können. Für die Erweiterbarkeit sind die folgenden zwei von Meyer genannten Prinzipien unerlässlich [Mey97]:

- **Einfaches Design:** Mit einer einfachen Architektur ist ist es leichter Änderungen durchzuführen als mit einer komplexen Architektur.
- **Dezentralisierung:** Je autarker die einzelnen Module sind, desto höher ist die Wahrscheinlichkeit, dass eine einfache Änderung nur ein Modul oder eine kleine Anzahl von Modulen beeinflusst, anstatt eine Kettenreaktion von Änderungen über das gesamte System auszulösen.

Die objektorientierte Programmierung hilft dabei, die Struktur der Software einfach und dezentralisiert aufzubauen. Für das Storage-Cluster wurde eine neue abstrakte Klasse *Storage* erstellt (vgl. Abschnitt D.1.6). Zusätzlich wurde für die Storage-Cluster Lösungen CephFS, GlusterFS und Flocker jeweils eine Subklasse erstellt (vgl. Abschnitt D.1.7, Abschnitt D.1.8, Abschnitt D.1.9). Das Bilden eines Storage-Clusters wurde in die drei Phasen *Vorbereitung*, *Erzeugung* und *Finalisieren* eingeteilt. Dementsprechend werden zum Erstellen eines Storage-Cluster die Methoden *prepare*, *generate* und *finalize* von der Klasse *Provider* aufgerufen:

- Die Funktion **prepare** führt alle vorbereitenden Tätigkeiten, die nach dem Start der Instanzen und vor dem Start des Storage-Clusters notwendig sind, aus. Ein Beispiel hierfür ist das Erstellen einer SSH Konfiguration, um die Kommunikation zu den Storage-Nodes zu ermöglichen. Sollte keine Vorbereitung notwendig sein, so kann der Bereich in der jeweiligen Subklasse auskommentiert werden.

- Beim Aufruf der Funkton **generate** wird das Storage-Cluster gebildet. Hierzu werden die für das jeweilige Storage-Cluster notwendigen Kommandozeilenbefehle aufgerufen.
- Mit der Funktion **finalize** wird das Erstellen des Storage-Clusters abgeschlossen. Diese wird zum Beispiel genutzt, damit auf den CoreOS Instanzen nach dem Erstellen des Storage-Clusters Befehle ausgeführt werden können, die eine Verbindung zu dem Storage-Cluster ermöglichen.

Sollen nachträglich Storage-Cluster zu dem Deployer hinzugefügt werden, so ist eine zusätzliche Subklasse mit dem Namen des Storage-Clusters und den Funktionen *prepare*, *generate* und *finalize* zu erstellen. Neben der Erweiterbarkeit sind die Startzeiten der jeweiligen Storage-Cluster Lösungen wichtig. Umso schneller der Deployer mit dem Storage-Cluster startet, umso schneller ist dieses einsetzbar. Die Startzeiten des Deployer werden durch den Cloud Anbieter und durch das in der Konfiguration gewählte Storage-Cluster bestimmt. Für den Vergleich der Startzeiten der Storage-Cluster mit Client-Server-Architektur wurde jeweils die minimalste Konfiguration von CephFS und GlusterFS bei Amazon (AWS) und bei OpenStack getestet (vgl. Abbildung 5.6).

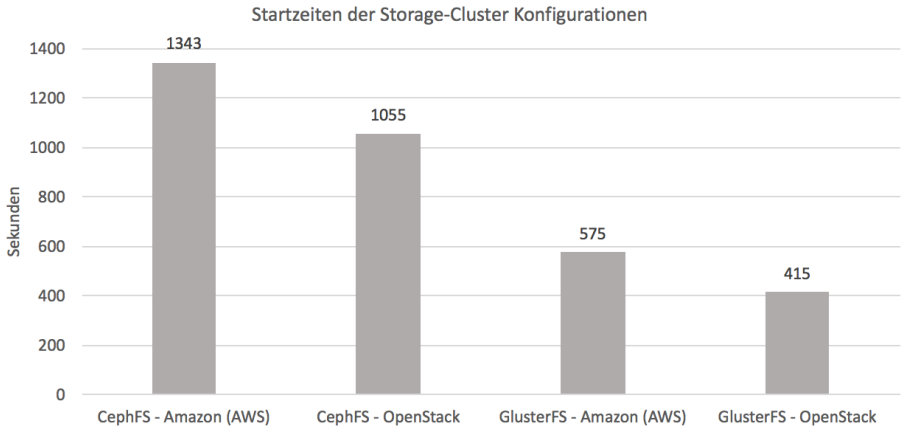

Abbildung 5.6: Startzeiten der Storage-Cluster Konfigurationen

Das auf OpenStack erstellte GlusterFS Storage-Cluster besitzt mit 415 Sekunden die schnellste Startzeit. Das Starten von einem auf OpenStack erstellten CephFS Storage-Cluster dauert dagegen 1055 Sekunden. Im Gegensatz zu den auf OpenStack basierenden Storage-Clustern benötigen die auf Amazon (AWS) erstellten Storage-Cluster mehr Zeit beim Erstellen einer identischen Storage-Cluster Konfiguration. Die unterschiedlichen Startzeiten von Amazon (AWS) und OpenStack sind auf die unterschiedlichen Startvorgänge zurückzuführen.

Kapitel 6

Evaluierung der Migrierbarkeit von Storage-Clustern

Das Ziel ist es, die Migration des Storage-Clusters zwischen verschiedenen Cloud Anbietern zu ermöglichen. Als größte Typenvertreter wurden für diese Arbeit die IaaS-Plattformen von Amazon (AWS) und OpenStack ausgewählt. Die vorhergehende Entwicklung hat gezeigt, dass es generell möglich ist, die Storage-Cluster Lösungen CephFS und GlusterFS einzeln bei dem Cloud Anbieter Amazon (AWS) und der privaten Cloud OpenStack zu betreiben. Lediglich die Storage-Cluster Lösung Flocker ist in der prototypischen Variante nur für Amazon (AWS) nutzbar und daher im aktuellen Entwicklungsstand nicht für eine plattformübergreifende Migration einsetzbar. Wie eine Migration mit CephFS und GlusterFS umgesetzt werden kann, wird in den folgenden Abschnitten beschrieben.

6.1 Migrationsszenario

Um die Migrierbarkeit der Storage-Cluster Lösungen zu evaluieren, wurde ein realistisches Migrationsszenario erstellt. Das Szenario sieht eine Migration eines Webserver-Dienstes inklusive der Webserver-Konfigurationsdaten von dem Cloud Anbieter Amazon (AWS) zu der privaten OpenStack Cloud Plattform vor. Die Abbildung 6.1 zeigt das Migrationsszenario in drei Schritten. Um einen Wechsel zwischen Amazon (AWS) und OpenStack durchführen zu können, wird eine bestehende Cluster Plattform mit einem Compute-Cluster und einem nativen Storage-Cluster bei Amazon (AWS) vorausgesetzt. Der Webserver-Dienst speichert die Webserver-Konfiguratoinsdaten in dem Storage-Cluster bei Amazon (AWS). Beim Start der Migration wird ein identisches natives Storage-Cluster bei OpenStack erstellt und mit dem Storage-Cluster von Amazon (AWS) synchronisiert (Schritt 1). Nach der Synchronisation stehen die Webserver-Konfigurationsdaten sowohl bei Amazon (AWS) als auch bei OpenStack zur Verfügung. Anschließend wird eine zweite Cluster Plattform bei der OpenStack Cloud Plattform erstellt (Schritt 2). Nach dem Erstellen der zweiten Cluster Plattform wird der Container mit dem Webserver bei Amazon (AWS) gestoppt und bei OpenStack wieder gestartet werden. Nach und nach werden die verbliebenen Container bei Amazon (AWS) ebenfalls gestoppt und bei OpenStack wieder gestartet, so dass das Container-Cluster bei OpenStack vollständig zur Verfügung steht. Das Ziel ist es, die Dienste durch das Container-Cluster unterbrechungsfrei zu migrieren. Nach dem Abschluss der Migration wird das nicht mehr benötigte Storage-Cluster bei Amazon (AWS) gestoppt (Schritt 3).

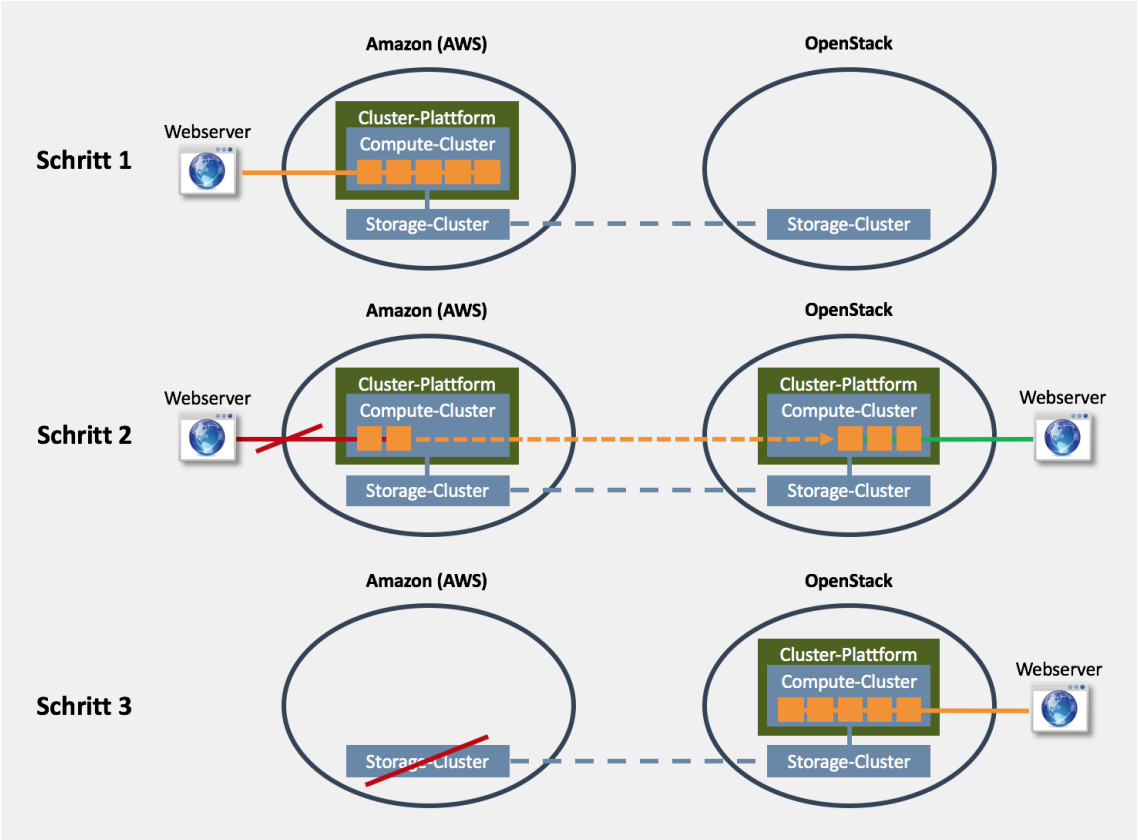

Abbildung 6.1: Migrationsschritte

6.2 Migrierbarkeit von CephFS

Die asynchrone Replikation der Daten von zwei geographisch getrennten Ceph-Clustern ist zwar möglich, sollte aber nur für die Durchführung der Migration genutzt werden, da dessen Konsistenzmodell vorsieht, dass die Abarbeitung der Schreibbefehle nur erfolgt, wenn auch die vorherigen Schreibvorgänge aller Replikate abgeschlossen sind. Dies hätte zur Folge, dass eine höhere Latenz im Netzwerk zwischen weit entfernten Cloud Anbietern die Leistung des Storage-Clusters negativ beeinflussen würde. CephFS setzt für die Kommunikation zwischen zwei Ceph-Cluster eine Layer-2 Netzwerk voraus. Da es zwischen Amazon (AWS) und OpenStack zurzeit nur eine Layer-3 Verbindung und keine Layer-2 Verbindung gibt, müsste diese vor der Durchführung einer Testmigration geschaffen werden. Ein Lösungsansatz stellt die Verwendung des Amazon Produktes *AWS Direct Connect*[1] dar. Bei einer bestehenden Layer-2 Verbindung wird die Migration durch den Einsatz des Ceph Programms *rbd-mirror* ermöglicht. Das Programm wird im Hintergrund auf beiden Ceph-Clustern ausgeführt und baut jeweils eine Verbindung zu dem lokalen und dem entfernten Ceph-Cluster mit dem Befehl *rbd mirror pool peer add* auf (vgl. Abbildung 6.2).

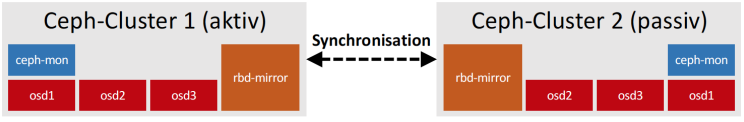

Abbildung 6.2: Ceph-Cluster Migration

[1] https://aws.amazon.com/de/directconnect/

6.3 Migrierbarkeit von GlusterFS

GlusterFS unterstützt die Synchronisation von Daten sowohl über ein Layer-2 Verbindung als auch über eine Layer-3 Verbindung (vgl. Abbildung 6.3). Die Latenz der Netzwerkverbindung spielt eine entscheidende Rolle, da GlusterFS nach dem strengen Konsistenzmodell konzipiert ist und ein ständiger Abgleich mit den einzelnen Storage-Knoten das Netzwerk belastet. Bei der Konfiguration muss darauf geachtet werden, dass alle Bricks (vgl. Abbildung 3.4) zeitsynchron sind. Hierzu muss die Systemzeit jedes Storage-Nodes mit einem validen Zeitserver mit dem Network Time Protocol (NTP) synchronisiert sein.

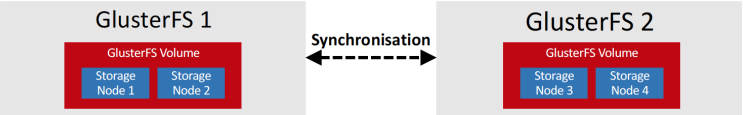

Abbildung 6.3: GlusterFS Migration

6.4 Netzwerklatenz

Da zum Zeitpunkt dieser Arbeit die Multicluster Konfiguration des Deployers noch nicht zur Verfügung stand, wurde für die Migrationstests ein plattformübergreifendes Storage-Cluster zwischen Amazon (AWS) und OpenStack manuell implementiert. Für die testweise Installation wurde das Storage-Cluster GlusterFS ausgewählt, da zwischen Amazon (AWS) und OpenStack zurzeit nur eine Internet Layer-3 Verbindung besteht. CephFS benötigt eine Layer-2 Verbindung zwischen Amazon (AWS) und OpenStack. Für das Erstellen der Layer-2 Verbindung zwischen Amazon (AWS) und OpenStack stehen entsprechende Technologien zur Verfügung. Da der Aufbau der Layer-2 Verbindung nicht Teil dieser Arbeit war, wurde der Migrationstest mit GlusterFS und nicht mit CephFS durchgeführt.

Für den Test der Netzwerklatenz wurde bei Amazon (AWS) eine t2.micro Instanz und bei OpenStack eine m1.micro Instanz mit dem Linux Betriebssystem Ubuntu 16.04.2 LTS gestartet (vgl. Tabelle C.1). Amazon (AWS) gibt beim Einsatz der t2.micro Instanz keine detallierte Information zu der zur Verfügung stehenden Netzwerkdatenrate an. In der Amazon (AWS) Produktbeschreibung werden ausbalancierte Netzwerkressourcen[2] genannt. Nur bei höherwertigen Produkten wird bei Amazon (AWS) eine Netzwerkleistung angegeben.

Die Netzwerklatenz gibt die Zeit an, die ein Datenpaket benötigt, um von der Quelle zum Ziel zu kommen. Zum Ermitteln der Latenz zwischen Amazon (AWS) und OpenStack wurde das Ping Programm genutzt. Es wird von der Quelle ausgeführt und sendet ein Internet Control Message Protocol Datenpaket (ICMP-Paket) zu dem Ziel. Das Ziel schickt ein Antwortpaket mit der ermittelten Latenzzeit an die Quelle zurück. Da das Ping Programm bereits in dem verwendeten Betriebssystem Ubuntu 16.04.2 LTS vorhanden ist, war keine zusätzliche Installation notwendig. Die Messung der Latenz von Amazon (AWS) in Richtung OpenStack mit einer Paketgröße von 64 Bytes ergab einen durchschnittlichen Wert von 22,76 ms (vgl. Abbildung 6.4).

6.5 Netzwerkbandbreite

Umso höher die zur Verfügung stehende Netzwerkbandbreite ist, umso mehr Daten können in einem vorgegebenen Zeitraum zwischen den Amazon (AWS) und OpenStack übertragen werden. Für die Messung der TCP- und UDP-Netzwerkbandbreite wurde das Open Source Programm iperf[3] eingesetzt. Für

[2] https://aws.amazon.com/de/ec2/instance-types/
[3] https://iperf.fr/

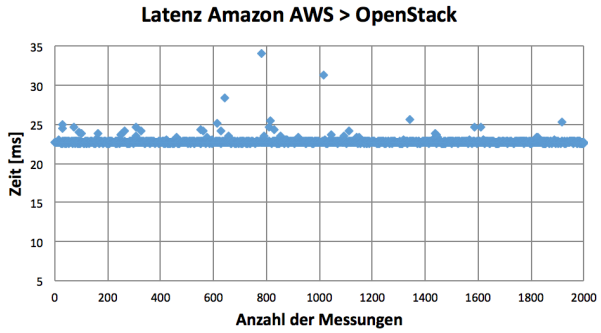

Abbildung 6.4: Messung der Latenz Amazon (AWS) in Richtung OpenStack

den Test wurde bei Amazon (AWS) eine t2.micro Instanz und bei OpenStack eine m1.micro Instanz mit dem Linux Betriebssystem Ubuntu 16.04.2 LTS gestartet (vgl. Tabelle C.1). Die Installation von iperf wurde auf den beiden Servern mit dem Befehl *apt-get install iperf* durchgeführt.

Für die Messung der **TCP Bandbreite** wurde auf dem ersten Server der iperf Daemon mit dem Befehl *iperf -s* ausgeführt. Auf dem zweiten Server wurde der Befehl *iperf -c <IP-Adresse> -i 1 -r* ausgeführt, um eine Verbindung zu dem iperf Daemon des ersten Servers herstellen zu können. Das Attribut *-i 1* sorgt dafür, dass jede Sekunde die Messergebnisse auf der Konsole ausgegeben werden. Die *<IP-Adresse>* ist die öffentliche IP-Adresse des ersten Servers. Die Ausgabe der Down- und der Uploadbandbreite wird durch das Attribut *-r* ermöglicht.

Bei der Messung der **UDP Bandbreite** wurden im Gegensatz zur Messung der TCP Bandbreite lediglich die verwendeten Attribute geändert. Der iperf Daemon wurde auf dem ersten Server mit *iperf -s -u* ausgeführt. Auf dem zweiten Server wurde wieder der Befehl *iperf -c <IP-Adresse> -i 1 -r* ausgeführt, um eine Verbindung zu dem iperf Daemon des ersten Servers herstellen zu können.

Für das endgültige Messergebnis wurde jeweils der Mittelwert der zehn Einzelmessungen genutzt.Die TCP- als auch die UDP-Netzwerkbandbreitenmessung wurde jeweils von OpenStack in Richtung Amazon (AWS) und von Amazon (AWS) in Richtung OpenStack durchgeführt. Das Ergebnis in Tabelle 6.1 zeigt

	Amazon (AWS) zu OpenStack	**OpenStack zu Amazon (AWS)**
TCP-Bandbreite Download	156 Mbit/s	201 Mbit/s
TCP-Bandbreite Upload	343 Mbit/s	76 Mbit/s
UDP-Bandbreite Download	511 Mbit/s	806 Mbit/s
UDP-Bandbreite Upload	151 Mbit/s	283 Mbit/s

Tabelle 6.1: Messung der TCP- und UDP-Netzwerkbandbreite

sehr unterschiedliche Werte. Da Amazon (AWS) und OpenStack nicht exklusiv Netzwerkbandbreite für das Storage-Cluster vorhält, sondern mehrere Nutzer sich jeweils die Netzwerkbandbreite teilen, kann es bei der Messung je nach Auslastung des jeweiligen Netzwerks zu unterschiedlichen Ergebnissen kommen. Des weiteren spielt das asymmetrische Routing ebenfalls eine entscheidende Rolle, da es dazu kommen kann, dass der Netzwerkpfad von Amazon (AWS) zu OpenStack nicht der gleiche Netzwerkpfad sein muss, wie von OpenStack zu Amazon (AWS).

6.6 Schreib- und Lesegeschwindigkeit

Umso höher die Schreib- und Lesegeschwindigkeit des Storage-Clusters ist, umso schneller können die Daten während oder vor der Migration übertragen werden. Das Ermitteln der Schreib- und Lesegeschwin-

digkeit ist nicht trivial, da zum Bestimmen der Schreib- und Lesegeschwindigkeit zwischen verschiedenen Datei- und Datenblockgrößen (Record Sizes) unterschieden werden muss. Zur Durchführung der Tests auf der Dateisystemebene können entsprechende Dateisystem Benchmark Tools genutzt werden, die diese Informationen ermitteln. Um eine Übersicht über vorhandene linux-taugliche Benchmark Tools zu bekommen, wurde eine Internetrecherche mit den Suchwörtern *File System, Dateisystem* und *Benchmark* durchgeführt. Die Übersicht der recherchierten Dateisystem Benchmark Tools ist in Tabelle 6.2 zu sehen.

Nr.	Name	Quelle
1	Bonnie++	http://www.coker.com.au/bonnie++
2	Filebench	https://github.com/filebench
3	Flexible File System Benchmark	https://sourceforge.net/projects/ffsb/
4	Flexible I/O Tester	https://github.com/axboe/fio
5	IOR HPC Benchmark	https://github.com/LLNL/ior
6	IOzone	http://iozone.org/

Tabelle 6.2: Dateisystem Benchmark Tools

Für die Messung wurde das Dateisystem Benchmark Tool IOzone ausgewählt, da es gegenüber den anderen Dateisystem Benchmark Tools einen automatisierten Test mit verschiedenen Datei- und Datenblockgrößen (Record Sizes) durchführt und die Testergebnisse in einer Exceldatei zur weiteren Verarbeitung speichert.

Für die Messung wurde der Befehl *iozone -a -c -e -i 0 -i 1 -+n -g 2g -R > /tmp/Testergebnisse.xls* verwendet:

- -a – Es wird ein automatisierter Test mit unterschiedlichen Dateigrößen und Datenblockgrößen (Record Sizes) ausgeführt.
- -c -e – Die Zeit zum endgültigen Schreiben der Datei wird mit kalkuliert (close und fsync).
- -i 0 -i 1 – Die Schreib- und die Lesegeschwindigkeit wird gemessen.
- -+n – Das erneute Schreiben und erneute Lesen von Daten wird nicht getestet.
- -g 2g – Es wird bis zu einer Dateigröße von 2 GB gemessen.
- -R > /tmp/Testergebnisse.xls – Die Testergebnisse werden direkt als Exceldatei ausgegeben.

Mit dem Dateisystem Benchmark Tool IOzone wurden 24 Tests zur Messung der Schreib- und Lesegeschwindigkeit der Storage-Cluster durchgeführt (vgl. 6.3).

Nr.	Gemessene Plattform	Plattformübergreifend	GlusterFS Nodes	virtuelle Prozessoren	Verweis
1.	OpenStack	nein	2	1	Abbildung E.1, Abbildung E.2
2.	OpenStack	nein	4	1	Abbildung E.3, Abbildung E.4
3.	OpenStack	nein	6	1	Abbildung E.5, Abbildung E.6
4.	OpenStack	nein	2	2	Abbildung E.7, Abbildung E.8
5.	OpenStack	nein	4	2	Abbildung E.9, Abbildung E.10
6.	OpenStack	nein	6	2	Abbildung E.11, Abbildung E.12
7.	Amazon (AWS)	nein	2	1	Abbildung E.13, Abbildung E.14
8.	Amazon (AWS)	nein	4	1	Abbildung E.15, Abbildung E.16
9.	Amazon (AWS)	nein	6	1	Abbildung E.17, Abbildung E.18
10.	Amazon (AWS)	nein	2	2	Abbildung E.19, Abbildung E.20
11.	Amazon (AWS)	nein	4	2	Abbildung E.21, Abbildung E.22
12.	Amazon (AWS)	nein	6	2	Abbildung E.23, Abbildung E.24
13.	OpenStack	ja	2	1	Abbildung E.25, Abbildung E.29
14.	OpenStack	ja	4	1	Abbildung E.33, Abbildung E.37
15.	OpenStack	ja	6	1	Abbildung E.41, Abbildung E.45
16.	OpenStack	ja	2	2	Abbildung E.27, Abbildung E.31
17.	OpenStack	ja	4	2	Abbildung E.35, Abbildung E.39
18.	OpenStack	ja	6	2	Abbildung E.43, Abbildung E.47
19.	Amazon (AWS)	ja	2	1	Abbildung E.26, Abbildung E.30
20.	Amazon (AWS)	ja	4	1	Abbildung E.34, Abbildung E.38
21.	Amazon (AWS)	ja	6	1	Abbildung E.42, Abbildung E.46
22.	Amazon (AWS)	ja	2	2	Abbildung E.28, Abbildung E.32
23.	Amazon (AWS)	ja	4	2	Abbildung E.36, Abbildung E.40
24.	Amazon (AWS)	ja	6	2	Abbildung E.44, Abbildung E.48

Tabelle 6.3: Messkonstellationen Schreib- und Lesegeschwindigkeit der Storage-Cluster

Für den Vergleich der Schreib- und Lesegeschwindigkeiten wurde die Datenblockgröße 4096 kByte gewählt, da bei den verwendeten Dateisystemen ebenfalls eine Datenblockgröße von 4096 kByte vorhanden war. Dies wurde mit dem Linux Befehl *blockdev –getbsz /dev/<Volume>* gegengeprüft.

6.6.1 Vergleich der Schreibgeschwindigkeit

Der Vergleich der Schreibgeschwindigkeit zeigt, dass ein auf OpenStack basierendes Storage-Cluster beim Einsatz einer unterschiedlichen Anzahl an GlusterFS Nodes mit Zweikern-Prozessoren eine nahezu konstante Schreibgeschwindigkeit besitzt, die zwischen 71,27 MByte/s und 71,63 MByte/s liegt (vgl. Abbildung 6.5). Es wurde ursprünglich vermutet, dass bei gemessenen Storage-Clustern die Schreibgeschwindigkeit mit der Anzahl der GlusterFS Nodes steigt. Diese Vermutung konnte anhand der Messergebnisse nicht bestätigt werden, da die Schreibgeschwindigkeiten bei drei Storage-Clustern mit zwei GlusterFS Nodes höher sind als bei den selben Storage-Clustern mit vier GlusterFS Nodes. Bei der Verwendung von sechs GlusterFS Nodes bewegen sich die Schreibgechwindigkeiten der gemessenen Storage-Cluster in einem Bereich zwischen 63,03 MByte/s und 71,63 MByte/s.

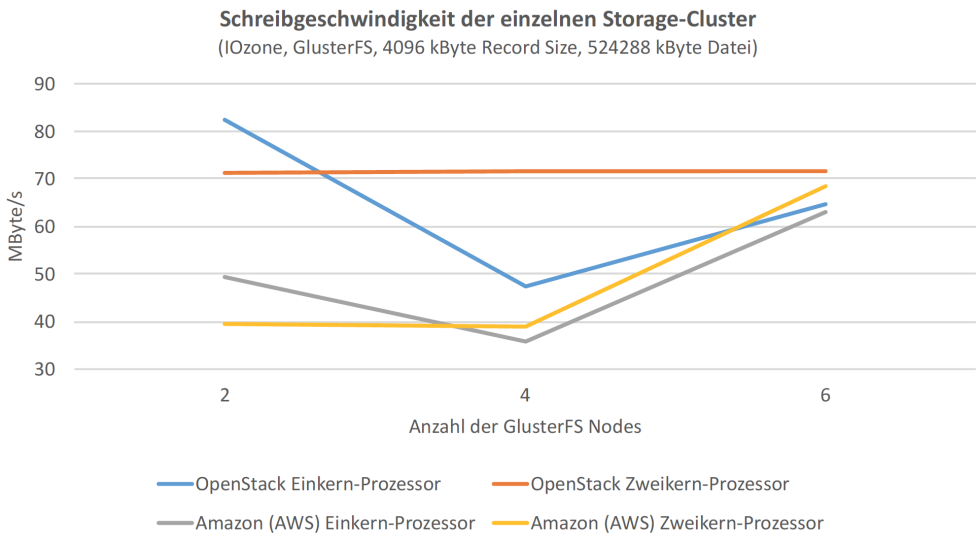

Abbildung 6.5: Vergleich der Schreibgeschwindigkeit OpenStack und Amazon (AWS)

Um eine Vergleichbarkeit zwischen den exclusiv genutzten Storage-Clustern und den plattformübergreifenden Storage-Clustern zu ermöglichen, wurden die Schreibgeschwindigkeiten der plattformübergreifenden Storage-Cluster ebenfalls mit einer Dateigröße von 524 MByte und einer Datenblockgröße von 4086 kByte gemessen. In der Abbildung 6.6 ist zu sehen, dass die Schreibgeschwindigkeiten beider von OpenStack aus durchgeführten Tests höhere Schreibgeschwindigkeiten erzielen als die von Amazon (AWS) aus durchgeführten Tests. Bei den plattformübergreifenden Storage-Clustern führt ein Wechsel

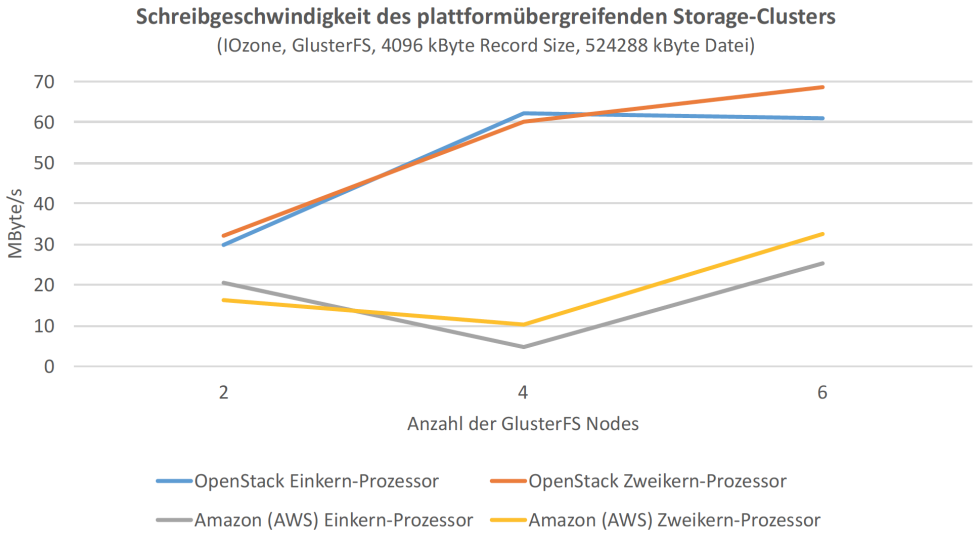

Abbildung 6.6: Vergleich der Schreibgeschwindigkeit der plattformübergreifenden Storage-Cluster

von Einkern- auf Zweikern-Prozessoren bei einer Dateigröße von 524 MB und einer Datenblockgröße von 4086 KB zu keiner überdurchschnittlichen Erhöhung der Schreibgeschwindigkeit.

6.6.2 Vergleich der Lesegeschwindigkeit

In der Abbildung 6.7 sind die Lesegeschwindigkeiten der exclusiv bei Amazon (AWS) und OpenStack betriebenen Storage-Cluster und die Lesegeschwindigkeiten der plattformübergreifenden Storage-Cluster dargestellt.

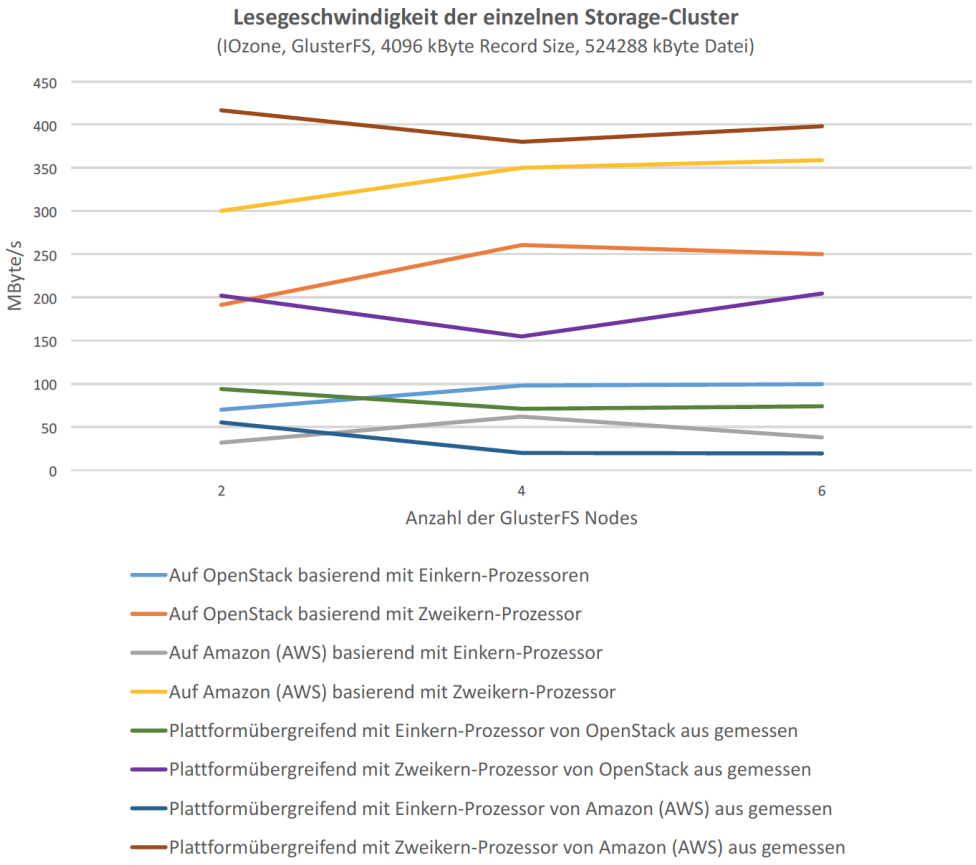

Abbildung 6.7: Vergleich der Lesegeschwindigkeiten der Storage-Cluster Varianten

Es ist zu deutlich erkennen, dass die Verwendung von Zweikern-Prozessoren anstatt Einkern-Prozessoren die Lesegeschwindigkeit steigert. Das auf Amazon (AWS) basierende Storage-Cluster mit Zweikern-Prozessoren und das plattformübergreifende von Amazon (AWS) aus gemessene Storage-Cluster mit Zweikern-Prozessoren erzielen höhere Lesegeschwindigkeiten als das auf OpenStack basierende Storage-Cluster mit Zweikern-Prozessoren und das plattformübergreifende von OpenStack aus gemessene Storage-Cluster mit Zweikern-Prozessoren. Dagegen erzielen das auf OpenStack basierende Storage-Cluster mit Einkern-Prozessoren und das plattformübergreifende von OpenStack aus gemessene Storage-Cluster mit Einkern-Prozessoren höhere Lesegeschwindigkeiten als das auf Amazon (AWS) basierende Storage-Cluster mit Einkern-Prozessoren und das plattformübergreifende von Amazon (AWS) aus gemessene Storage-Cluster mit Einkern-Prozessoren.

Im Vergleich zu den gemessenen Schreibgeschwindigkeiten von Storage-Clustern mit Zweikern-Prozessoren sind die gemessenen Lesegeschwindigkeiten von Storage-Clustern mit Zweikern-Prozessoren deutlich höher.

6.7 Datenkonsistenz

Für eine erfolgreiche Migration zwischen zwei Cloud Anbietern muss die Datenkonsistenz sichergestellt sein, um Datenverlust und den Ausfall von Diensten zu vermeiden. Bei der Auswertung der Storage-Cluster Eigenschaften wurden Storage-Cluster Lösungen ausgewählt, die ein strenges Konsistenzmodell besitzen (vgl. Abbildung 4.1). Eine Storage-Cluster Lösung mit einem strengen Konsistenzmodell hat die Aufgabe, die Storage-Nodes stets in einem konsistenten Zustand zu halten. Für die Evaluierung der Datenkonsistenz eines GlusterFS Storage-Clusters wurde neben der Migration eines vorhandenen Datenbestandes eine zusätzliche Auslastung mit 100 parallelen Schreiboperationen erzeugt. Für die

Migration wurden 6200 Testdateien mit unterschiedlicher Dateigrößen (1 Byte bis 10 MByte) mit Hilfe eines Bash-Skriptes erstellt (vgl. Listing 6.1).

```
#! /bin/bash
for n in {1..1000}; do
  dd if=/dev/urandom of=file$( printf %03d "$n" ).1 bs=1 count=$((1))
  dd if=/dev/urandom of=file$( printf %03d "$n" ).2 bs=1 count=$((10))
  dd if=/dev/urandom of=file$( printf %03d "$n" ).3 bs=1 count=$((100))
  dd if=/dev/urandom of=file$( printf %03d "$n" ).4 bs=1 count=$((1000))
  dd if=/dev/urandom of=file$( printf %03d "$n" ).5 bs=1 count=$((10000))
  dd if=/dev/urandom of=file$( printf %03d "$n" ).6 bs=1 count=$((100000))
done
for n in {1..100}; do
  dd if=/dev/urandom of=file$( printf %03d "$n" ).7 bs=1 count=$((1000000))
  dd if=/dev/urandom of=file$( printf %03d "$n" ).8 bs=1 count=$((10000000))
done
```

Listing 6.1: Erstellen der Testdateien

Zu Beginn des Migrationstests wurde mit dem Deployer ein Storage-Cluster mit zwei Storage-Nodes bei Amazon (AWS) erstellt. Anschließend wurden die 6200 Testdateien in das Storage-Cluster kopiert. Mit Hilfe des Hashwert-Algorithmus „Message-Digest Algorithm 5“ (MD5) kann die Integrität der Daten nach dem Kopieren von den Amazon (AWS) Storage-Nodes zu den OpenStack Storage-Nodes kontrolliert werden. Der Message-Digest Algorithm 5 (MD5) ist ein Algorithmus, der aus einer Nachricht mit beliebiger Länge einen 128-Bit-Hashwert erzeugt [Riv92]. Auf dem Amazon (AWS) Storage-Cluster wurde mit dem Befehl *md5sum * > checksum* eine Datei erzeugt, die zeilenweise den bearbeiteten Dateinamen und den dazugehörigen MD5-Hashwert speichert. Nach dem Bilden der MD5-Hashwerte wurde seitens OpenStack ein weiteres Storage-Cluster mit dem Deployer erstellt. Da zu dem Zeitpunkt zwei Storage-Cluster mit unterschiedlichen Konfigurationen parallel liefen, wurde die Konfiguration des auf OpenStack basierenden Storage-Clusters an die Konfiguration des auf Amazon (AWS) basierenden Storage-Clusters mit folgendem Ablauf manuell angepasst.

1. OpenStack - Stoppen des GlusterFS Data Volumes: *sudo gluster volume stop <Volume Name>*
2. OpenStack - Löschen des GlusterFS Data Volumes: *sudo gluster volume delete <Volume Name>*
3. OpenStack - Auflösen beider GlusterFS Peers: *sudo gluster peer detach <Hostname>*
4. Hinzufügen der GlusterFS Peers zu dem auf Amazon (AWS) basierenden Storage-Cluster : *sudo gluster peer probe <Hostname>*
5. Hinzufügen des Storage-Nodes 3 zu dem auf Amazon (AWS) basierenden Storage-Cluster: *sudo gluster volume add-brick <Data Volume> replica 3 <Hostname >:/mnt/gluster force*
6. Hinzufügen des Storage-Nodes 4 zu dem auf Amazon (AWS) basierenden Storage-Cluster: *sudo gluster volume add-brick <Data Volume> replica 4 <Hostname >:/mnt/gluster force*

Nach dem Eingeben der Befehle beginnt die Synchronisation zwischen den auf Amazon (AWS) basierenden Storage-Nodes und den auf OpenStack basierenden Storage-Nodes. Parallel zu der laufenden Synchronisation wurde seitens Amazon (AWS) eine zusätzliche Auslastung des Systems erzeugt, um das Verhalten des Storage-Clusters bei einer höheren I/O Auslastung zu analysieren. Zum Erzeugen der zusätzlichen Auslastung wurde das Benchmark Tool IOzone mit dem Befehl *iozone -i 0 -c -e -+n -I -w -s 100M -t 100 -r 4096k* bei dem Storage-Node 2 bei Amazon (AWS) verwendet:

- -i 0 – Das Schreiben einer Datei wird ausgeführt.

- -c -e – Die Zeit zum endgültigen Schreiben der Datei wird mit kalkuliert (close und fsync).
- -+n – Das erneute Schreiben und erneute Lesen von Daten wird nicht getestet.
- -I – Die Dateien werden nicht in den Zwischenspeicher, sondern direkt auf die virtuelle Festplatte geschrieben.
- -s 100M – Die Dateigröße beträgt 100 MByte.
- -t 100 – Es werden 100 parallele Schreiboperationen durchgeführt.
- -r 4096 – Die Datenblockgröße (Record Size) der Dateien beträgt 4096 kByte..

IOzone erzeugt mit dem Befehl 100 parallele Schreibvorgänge mit je 100 MByte Dateigröße. Bei dem Test wurden die IOzone Schreiboperationen auf den Storage-Node 2 bei Amazon (AWS) ausgeführt. Anschließend wurden die IOzone Dateien von GlusterFS auf den Storage-Node 1, den Storage-Node 3 und den Storage-Node 4 geschrieben. Da beim Beginn des Tests der Storage-Node 3 und der Storage-Node 4 zu dem auf Amazon (AWS) basierenden Storage-Cluster hinzugefügt wurden, wurde der bereits vorhandene Datenbestand des auf Amazon (AWS) basierenden Storage-Clusters auf die auf OpenStack basierenden Storage-Nodes 3 und 4 von GlusterFS automatisch geschrieben (vgl. Abbildung 6.8).

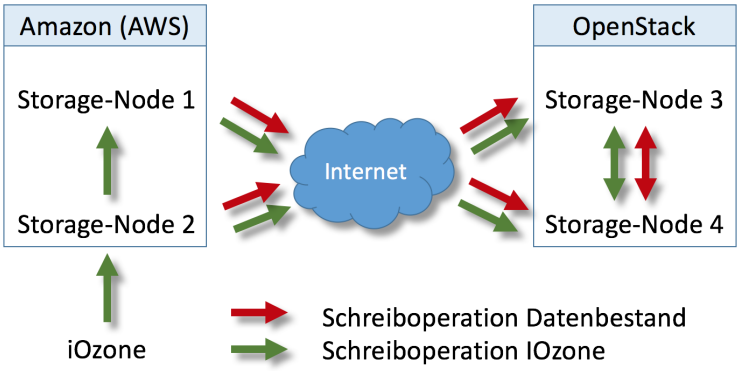

Abbildung 6.8: Test der Datenkonsistenz - Schreiboperationen

Nach Abschluss der Synchronisation wurde die Datenintegrität mit dem Befehl *md5sum -c checksum* geprüft. Der Befehl vergleicht die MD5-Hashwerte in der Datei checksum mit den MD5-Hashwerten der auf dem OpenStack Storage-Nodes gespeicherten Dateien und gibt bei gleichen MD5-Hashwerten ein *ok* zurück. Ein *ok* bedeutet, dass eine Datei nach der Übertragung unverändert vorhanden ist. Das Ergebnis der Überprüfung ergab, dass 100 % der 6200 Testdateien unverändert auf die OpenStack Storage-Nodes übertragen wurden, so dass diese konsistent vorhanden waren. Da es sich bei der Evaluierung um automatisch generierte Testdateien handelte, sollte in folgenden Untersuchungen eine umfangreichere Überprüfung mit Dateien einer produktiv eingesetzten Container-Cluster Plattform durchgeführt werden, um die Ergebnisse zu verifizieren.

6.8 Auswahl der Storage-Cluster Konfiguration für die Migration

Es ist davon auszugehen, dass bei der Migration der Storage-Cluster viele Dateien unterschiedlicher Größe geschrieben und gelesen werden müssen. Bei den Analysen wurden bei den Storage-Clustern unterschiedliche Scheib- und Lesegeschwindigkeiten bei einer Dateigröße von 524 MByte und einer Datenblockgröße (Record Size) von 4096 kByte festgestellt. Für die Migration sollte mindestens ein plattformübergreifendes Storage-Cluster bestehend aus sechs Storage-Nodes mit Zweikern-Prozessoren gewählt werden, da dieses die höchsten Schreib- und Lesegeschwindigkeiten bei einer Dateigröße von 524 MByte und einer Datenblockgröße (Record Size) von 4096 kByte erzielt.

Kapitel 7

Fazit und Ausblick

7.1 Fazit

Diese Arbeit hat gezeigt, dass eine Integration von Storage-Clustern in elastische Container Plattformen möglich ist. Um für die Integration verfügbare Storage-Cluster Lösungen zu identifizieren, wurden bei einer Online-Recherche 23 geeignete Storage-Cluster Lösungen ermittelt. Anschließend wurden die Storage-Cluster Lösungen hinsichtlich ihrer Besonderheiten untersucht. Für die vergleichende Analyse wurden sechs Eigenschaften als „Muss"-Krtierium und zwei Eigenschaften als „Soll"-Kriterium festgelegt. Die Storage-Cluster wurden hinsichtlich der Kriterien ausgewertet, um von den 23 verfügbaren Storage-Clustern Lösungen, die für die Integration am besten geeignetsten ausfindig zu machen. Während der Untersuchung wurde erkannt, dass es zwei verschiedene Architekturmodelle gibt, die nicht direkt miteinander verglichen werden können. Dies führte dazu, dass eine Auswertung für Storage-Cluster Lösungen mit Client-Server- und Peer-to-Peer-Architektur und eine weitere Auswertung für Storage-Cluster Lösungen mit Storage-Backend-Architektur erstellt wurde. Fünf von den 23 untersuchten Storage-Cluster Lösungen boten die besten Eigenschaften für die Integration in die elastische Container Plattform. Da nicht alle fünf Storage-Cluster Lösungen im Rahmen dieser Arbeit integriert werden sollten, sondern nur drei, fiel die Wahl auf CephFS, Flocker und GlusterFS.

Anschließend wurden drei Umsetzungsmethoden für die Integration des Storage-Clusters in die elastische Container Plattform ermittelt und miteinander verglichen. Als Umsetzungsmethode wurde die Integration eines nativen Storage-Cluster ausgewählt und programmiertechnisch als Erweiterung des bestehenden Deployers in der Programmiersprache Ruby umgesetzt. Dabei wurde von den Storage-Clustern CephFS, Flocker und GlusterFS eine lauffähige prototypische Erweiterung des Deployers erstellt, getestet und dokumentiert.

Im Anschluss an die programmiertechnische Umsetzung wurde die Migration des Storage-Clusters zwischen dem Cloud-Anbieter Amazon (AWS) und einer OpenStack Plattform evaluiert. Dabei wurden speziell die Migrationsszenarien für die Storage-Cluster CephFS und GlusterFS untersucht. Für die Evaluierung wurde ein Migrationsszenario entwickelt und ein plattformübergreifendes Storage-Cluster basierend auf GlusterFS erstellt. Bei durchgeführten Migrationstests wurde nicht nur die Netzwerklatenz und die Netzwerkbandbreite zwischen Amazon (AWS) und OpenStack gemessen, sondern es fanden insgesamt 24 Tests der Schreib- und Lesegeschwindigkeit statt. Hierfür wurden jeweils drei verschiedene Storage-Cluster Ausprägungen, welche exklusiv jeweils auf Amazon (AWS) und OpenStack installiert wurden, getestet. Zusätzlich wurde die Schreib- und Lesegeschwindigkeit in einem Migrationsszenario getestet und ausgewertet. Die Testumgebung bestand aus einem plattformübergreifenden Storage-Cluster mit ingesamt zwei, vier und sechs GlusterFS Nodes. Die Tests der Schreib- und Lesegeschwindigkeit wurde bei allen plattformübergreifenden Konstellationen jeweils auf den Amazon (AWS) Instanzen und auf den OpenStack Instanzen ausgeführt. Anhand der Messergebnisse wurden Unterschiede bei beiden Plattformen identifiziert. Der Einsatz ein höheren Anzahl an GlusterFS Nodes zeigt nur bei den Messungen,

die auf den OpenStack Instanzen durchgeführt wurden, höhere Schreib- und Lesegeschwindigkeiten. Entgegen der Erwartungen wurden bei den Messungen auf den Amazon (AWS) Instanzen bei einer höheren Anzahl an GlusterFS Nodes auch niedrigere Schreib- und Lesegeschwindigkeiten bei vorab fest definierten Datei- und Datenblockgrößen (Record Sizes) festgestellt. Hinsichtlich der Datenkonsistenz bei einer hohen I/O Auslastung des Storage-Cluster konnten während einer Migration keine Einschränkungen festgestellt werden. Bei der Evaluierung. konnte eine für die Migration geeignete Konfiguration ermittelt werden.

7.2 Ausblick

Die derzeitige Implementierung ist ein Prototyp zum Nachweis, das ein Storage-Cluster in eine elastischen Container Plattform implementiert werden kann. Die Weiterentwicklung des Systems zur produktiven Nutzung sollte folgende Funktionalitäten enthalten.

7.2.1 Modularisierung des Deployers

Die containerisierten Versionen der Storage-Cluster Lösungen werden ständig weiterentwickelt, so dass in naher Zukunft mit produktiv einsetzbaren containerisierten Versionen zu rechnen ist. Für weitere Arbeiten ist die Weiterentwicklung der Modularisierung der Core-, Monitor- und Storage-Komponente sinnvoll.

7.2.2 Skalierung des Speicherplatzes

Bei der prototypischen Implementierung des Storage-Clusters wurde der Speicherplatz pro Instanz für den Nutzer unveränderbar im Quellcode festgelegt. Bei der weiteren Entwicklung der Storage-Cluster Erweiterung sollte es ein Monitoring für den frei verfügbaren Speicherplatz geben. Je nachdem ob zu wenig oder zu viel Speicherplatz vorhanden ist, sollte das System automatisch skalieren können.

7.2.3 Storage-Multicluster Konfiguration

Zum Zeitpunkt der Arbeit befindet sich eine plattformübergreifende Multicluster Konfiguration für das Kubernetes Compute-Cluster in der Entwicklung. Darauf aufbauend sollte auch das automatische Erstellen eines plattformübergreifenden Storage-Clusters basierend auf einer Storage-Multicluster Konfiguration entwickelt werden. Sowohl die Compute-Multicluster Funktionalität als auch die Storage-Multicluster Funktionalität sind Voraussetzungen für eine erfolgreiche Migration zwischen zwei Cloud Anbietern.

Anhang A

Suchmaschinen

Nr.	Titel	Beschreibung	Adresse
1	ACM Digital Library	ACM Publikationen	http://dl.acm.org
2	Berkely Library	Bibliothek der University of California	http://lib.berkeley.edu
3	DDB	Deutsche digitale Bibliothek	http://www.ddb.de
4	DeepDyve	Stellt wissenschaftliche Inhalte von 100 führenden Verlagen zur Verfügung.	http://www.deepdyve.com
5	Emerald Insight	Suchmaschine des Emerald Verlags	http://www.emeraldinsight.com
6	Google Scholar	Dient der allgemeinen Literaturrecherche wissenschaftlicher Inhalte.	http://scholar.google.com
7	IEEE Xplore Digital Library	Bibliothek des IEEE	http://ieeexplore.ieee.org
8	KIT-Bibliothek	Bibliothek des Karlsruher Instituts für Technologie	http://bibliothek.kit.edu
9	Marriott Library	Bibliothek der University of Utah	http://databases.tools.lib.utah.edu
10	MOnAMi	Hochschulschriftenserver der Hochschule Mittweida	http://hsmw.bsz-bw.de
11	Oxford Handbooks	Oxford Verlag	http://www.oxfordhandbooks.com
12	ResearchGate	Soziales Netzwerk für Forscher	http://www.researchgate.net
13	ScienceDirect	Wissenschaftliche Online-Datenbank.	http://www.sciencedirect.com
14	Springer Link	Suchmaschine des Springer Verlags	http://link.springer.com
15	The European Library	Europäische Bibliothek	http://www.theeuropeanlibrary.org
16	Vanderbilt Bibliothek	Bibliothek der Vanderbilt University	http://library.vanderbilt.edu
17	Wiley Online Library	Suchmaschine des Wiley Verlags	http://onlinelibrary.wiley.com
18	ZHB Lübeck	Zentrale Hochschulbibliothek Lübeck	http://zhb.uni-luebeck.de

Tabelle A.1: Suchmaschinen

Anhang B

Docker-Images

Docker-Image	Storage-Cluster Lösung	Quelle
ceph/daemon	CephFS	https://hub.docker.com/r/ceph/daemon/
heketi/gluster	GlusterFS	https://hub.docker.com/r/heketi/gluster/
gluster/gluster-centos	GlusterFS	https://hub.docker.com/r/gluster/gluster-centos/
gluster/gluster-fedora	GlusterFS	https://hub.docker.com/r/gluster/gluster-fedora/

Tabelle B.1: Docker-Images für ausgewählte Storage-Cluster Lösungen

Anhang C

Instanz-Typen

Amazon (AWS) stellt eine Vielzahl an Instanzen für unterschiedliche Einsatzzwecke zur Verfügung. Bei OpenStack hat der Administrator der Cloud-Umgebung die Möglichkeit, Instanzen selbst zu definieren. Instanz-Typen stellen verschiedene Kombinationen aus CPU, Arbeitsspeicher und Festplattenspeicherplatz dar. Bei Amazon (AWS) werden vom Deployer sogenannte T2-Instanzen genutzt. Diese stellen eine Basisleistung zur Verfügung und können darüber hinaus Spitzenlasten bearbeiten. Die Abrechnung basiert auf der genutzten CPU-Leistung. Bei OpenStack sind die Kapazitäten durch die Hardware eingeschränkt Eine Abrechnung gegenüber einem Dienstleister findet bei der OpenStack Plattform nicht statt. Die zur Verfügung stehenden Instanz-Typen sind in der Tabelle C.1 zu sehen.

Cloud Anbieter	**Instanz-Typ**	**vCPUs**	**Arbeitsspeicher**	**Festplatte**
Amazon (AWS)	t2.nano	1	0.5 GB	8 GB
Amazon (AWS)	t2.micro	1	1 GB	8 GB
Amazon (AWS)	t2.small	1	2 GB	8 GB
Amazon (AWS)	t2.medium	2	4 GB	8 GB
Amazon (AWS)	t2.large	2	8 GB	8 GB
Amazon (AWS)	t2.xlarge	4	16 GB	8 GB
Amazon (AWS)	t2.2xlarge	8	32 GB	8 GB
OpenStack	m1.tiny	1	0.5 GB	1 GB
OpenStack	m1.micro	1	1 GB	8 GB
OpenStack	m1.small	1	2 GB	20 GB
OpenStack	m1.medium	2	4 GB	40 GB
OpenStack	m1.large.storage	2	8 GB	8 GB
OpenStack	m1.large	4	8 GB	80 GB
OpenStack	m1.xlarge	8	16 GB	160 GB

Tabelle C.1: Instanz-Typen

Anhang D

Anpassungen am Deployer

D.1 Programmiertechnische Anpassungen

D.1.1 Anpassung der Klasse Parser

Die Aufgabe der Klasse Parser ist es, die Dateien <credentials.json> und <simpleCluster.json> einzulesen und deren Inhalt an die entsprechenden Objekte zu übergeben. Das Objekt Cluster speichert die Informationen der Datei <simpleCluster.json>. Für die Integration der Storage-Cluster Lösungen wurde die Datei <simpleCluster.json> erweitert, so dass zu den bereits vorhandenen Informationen zusätzlich der Name des Storage-Clusters angegeben wird. In Listing D.1 ist eine Erweiterung der Datei <simpleCluster.json> mit einer CephFS Storage-Cluster Lösung mit einem Monitor-Server und drei OSD-Servern zu sehen.

```
"storage":[{
  "role" : "mon",
  "provider": "aws",
  "storage_cluster": "cephfs",
  "flavor": "t2.micro",
  "image": "ami-3f1bd150",
  "number": 1
  },{
  "role" : "osd",
  "provider": "aws",
  "storage_cluster": "cephfs",
  "flavor": "t2.micro",
  "image": "ami-3f1bd150",
  "number": 3
  }]
```

Listing D.1: Erweiterung der Datei <simpleCluster.json>

Um die zusätzlichen Angaben entsprechend verarbeiten zu können, wurde die Funktion readCluster für das Einlesen der Storage-Cluster Attribute erweitert. Da die Storage-Cluster Lösungen mit Hostnamen arbeiten, wurde in der Funktion das Speichern eines Hostnamens bestehend aus der Rolle und der Nummer der jeweiligen Instanz implementiert (vgl. Listing D.2).

```
storagehash.each do |hash|
  count = 1
  hash["number"].times do
   cluster.addMachine(Machine.new(hash["provider"],hash["storage_cluster"], hash["image"],hash["flavor"],
       hash["role"], "#{hash['role']}#{count}"))
   cluster.storage_cluster = hash["storage_cluster"]
```

```
    cluster.storage_cluster_network = hash["network"]
  count += 1
end
```

Listing D.2: Erweiterung der Funktion "readCluster"

D.1.2 Anpassung der Klasse Cluster

Die Klasse Cluster stellt Objekte zum Speichern der Informationen eines Clusters bereit. Für die Erweiterung des Deployers wurden folgende Attribute hinzugefügt:

- **storage_cluster** speichert den Namen des in der <simpleCluster.json> angegeben Storage-Clusters. Bei der Integration wurden die Storage-Cluster CephFS, GlusterFS und Flocker verwendet.
- Das Attribut **storage_cluster_network** übernimmt den in der <simpleCluster.json> angegeben Netzwerkbereich.
- **volume_name** ist der Verzeichnisname des eingebundenen Datenträgers. Für die Speicherung der Daten wird bei den Storage-Cluster Lösungen CephFS und GlusterFS der lokale Speicher der Instanzen benutzt. Der Pfad zu dem Datenträger ist bei Amazon (AWS) und OpenStack unterschiedlich. In der Datei <credentials.json> wird dem Cluster der jeweilige Name *xvdf* oder *sdb* übergeben.
- Das Attribut **keypair** enthält den Namen der Schlüsseldatei, mit der sich ein Nutzer an den Instanzen per SSH authentisiert. Der Name wird dem Deployer in der Datei <credentialis.json> beim Start mitgeteilt.

D.1.3 Anpassung der Klasse Machine

In einem Objekt der Klasse Machine werden die Attribute einer Instanz gespeichert. Für das Storage-Cluster wurden die Attribute der Klasse Machine erweitert, da das Storage-Cluster beim Erstellen zusätzliche Informationen der einzelnen Instanzen benötigt. Zu den benötigten Informationen zählen die IP-Adressen (öffentlich und privat) und der Hostname. Wird Amazon (AWS) mit einem zusätzlichen virtuellen Datenträger verwendet, so muss auch die Identifikationsnummer des virtuellen Datenträgers und die Datenträgergröße in einem Objekt der Klasse Machine gespeichert werden.

D.1.4 Anpassung der Klasse Provider

Die Klasse Provider stellt unter anderem die Funktion createCluster zur Verfügung. Diese Funktion entscheidet, welche Schritte zum Erstellen des Container-Clusters durchgeführt werden müssen. Zudem wird an dieser Stelle die Entscheidung zum Erstellen der Storage-Server getroffen. Ist eine Instanz mit der Rolle mon, osd, storage oder flocker in der Datei <simpleCluster.json> angegeben, so werden mit der Funktion generateStorageUserdata Konfigurationsdaten erstellt, welche anschließend von den Funktion generateStorage zum Erstellen der Storage-Nodes genutzt werden (vgl. D.3).

```
cluster.machines.each do |machine|
  clusterTyp = ['mon', 'osd', 'storage', 'flocker']
  role = machine.role.downcase
  if clusterTyp.include? role
    generateStorageUserdata( cluster )
    generateStorage(machine, cluster.secgrp_name)
```

```
  end
end
```

Listing D.3: Anpassung der Klasse Provider

Die Konfigurationsdaten enthalten Befehle, mit denen unter anderem zusätzliche Software auf den Servern installiert wird oder auch auch Anweisungen, um zum Beispiel Verzeichnisse zu erstellen. Die Konfigurationsdaten werden für jede Storage-Cluster Lösung anhand von vordefinierten Template-Dateien erstellt.

D.1.5 Anpassung der Klassen AwsProvider und OpenStack

Die Klassen AwsProvider und OpenStack sind spezielle Provider-Klassen, welche für die Interaktion mit der Cloud Computing Umgebung Amazon (AWS) und OpenStack genutzt werden. Diese Klassen stellen ursprünglich Funktionen zur Verfügung, um den Master-Server und die Nodes zu erstellen. In Listing D.4 ist neue Funktion generateStorage der Klasse AwsProvider zu sehen. Sie erstellt eine Instanz für das Storage-Cluster und ruft die Funktion generateVolume zum Erstellen einer zusätzlichen virtuellen Festplatte auf. Die Funktion generateVolume wird nur bei Amazon (AWS) genutzt.

```
def generateStorage(machine, sec_group)
  tmp_exe = "aws ec2 run-instances --image-id #{machine.image} --output text --region eu-central
      -1 --key-name #{@credentials.key_pair} --instance-type #{machine.flavor} --security-groups
      #{sec_group} --iam-instance-profile Name=allowSendingSsmCommands --user-data file://#{
      @storageUserDataFile}"
  Cli::LOG.debug(self){"Running command: #{tmp_exe}"}
  tmp_out = `#{tmp_exe}`
  machine.uid = tmp_out.split(" ")[8]
  machine.private_ipv4 = tmp_out.split(" ")[13]
  Cli::LOG.info(self){"Generated instance #{machine.uid} with private ip #{machine.private_ipv4}"}
  generateVolume(machine)
end
```

Listing D.4: Klasse AwsProvider - Funktion generateStorage

Für OpenStack wurde ebenfalls die Funktion generateStorage implementiert (vgl. Listing D.5). Eine virtuelle Festplatte muss nicht erstellt werden, da OpenStack über Servervarianten mit einer zusätzlichen Festplatte verfügt .

```
def generateStorage(machine,secgrp_name)
  generate_server(machine,secgrp_name,@storageUserDataFile)
  @master_uid = machine.uid
  Cli::LOG.info("Generated storage with uid: #{machine.uid}")
end
```

Listing D.5: Klasse OpenStack - Funktion generateStorage

Nachdem die für das Storage-Cluster zuständigen Server erfolgreich gestartet wurden, muss gegenüber der vorherigen Deployer Version festgestellt werden, ob der Startvorgang bei den Storage-Cluster Servern vollständig abgeschlossen ist. Dieser Prüfschritt ist notwendig, da die öffentlichen IP-Adressen der Storage-Cluster Server erst nach dem vollständigen Systemstart ausgelesen und dem Container-Cluster mitgeteilt werden können. Für das Überprüfen des Systemstarts wurde für die Klasse AwsProvider die Funktion checkMachineSystemState entwickelt (vgl. Listing D.6).

```
def checkMachineSystemState(cluster,uid, role)
  seconds_per_check = 30
  max_checks = 15
  Cli::LOG.info("Checking machine system state...")
```

```
5    cluster.machines.each do |machine|
6      if machine.uid == uid
7        checks = 1
8        while machine.last_state != "ok" do
9          tmp_exe="aws ec2 describe-instance-status --query \"InstanceStatuses[?InstanceId=='#{machine
               .uid}'].InstanceStatus.Status\" --output text"
10         Cli::LOG.debug(self){"Running command: #{tmp_exe}"}
11         tmp_out = `#{tmp_exe}`
12         machine.last_state = tmp_out.strip
13         if machine.last_state != "ok"
14           Cli::LOG.info("Machine with uid #{machine.uid} has not completely started. Checking state in
                 #{seconds_per_check} seconds...")
15           sleep(seconds_per_check)
16         end
17         if checks > max_checks
18           Cli::LOG.debug(self){"Machine could not be started in the given time. Aborting..."}
19         return false
20         end
21         checks += 1
22       end
23       Cli::LOG.info("Machine with uid #{machine.uid} has completely started")
24       tmp_exe="aws ec2 describe-instances --query \"Reservations[*].Instances[?InstanceId=='#{uid}'].
             PublicIpAddress\" --output=text"
25       Cli::LOG.debug(self){"Running command: #{tmp_exe}"}
26       tmp_out = `#{tmp_exe}`
27       machine.public_ipv4 = tmp_out.strip
28       tmp_exe="aws ec2 describe-instances --query \"Reservations[*].Instances[?InstanceId=='#{uid}'].
             PublicDnsName\" --output=text"
29       Cli::LOG.debug(self){"Running command: #{tmp_exe}"}
30       tmp_out = `#{tmp_exe}`
31       machine.public_dns_name = tmp_out.strip
32     end
33   end
34 end
```

Listing D.6: Klasse AwsProvider - Funktion checkMachineSystemState

Bei Amazon (AWS) lässt sich der Status einer Instanz eindeutig mit Hilfe der AWS-Befehlszeilen-Schnittstelle (CLI) abrufen, sodass die weitere Verarbeitung durchgeführt werden kann, sobald die Server den Status *ok* zurückgeben. Bei der OpenStack Plattform wartet der Deployer auf die Fertigstellung des Startvorgangs der Server. Nach der prototypischen Implementierung muss auch hier eine Möglichkeit gefunden werden, den Status des Servers zu identifizieren (vgl. Listing D.7).

```
1  def checkMachineSystemState(cluster,uid,role)
2    if role.downcase =="storage" || role.downcase =="mon" || role.downcase =="osd"
3      seconds = 45
4      Cli::LOG.info("Waiting for #{seconds} seconds for the machine with uid #{uid} to start ...")
5      sleep(seconds)
6    elsif role.downcase =="master" || role.downcase =="lowperformer"
7      seconds = 40
8      Cli::LOG.info("Waiting for #{seconds} seconds for the machine with uid #{uid} to start ...")
9      sleep(seconds)
10   end
11 end
```

Listing D.7: Klasse OpenStack - Funktion checkMachineSystemState

D.1.6 Neue Klasse Storage

Die Klasse Storage stellt Funktionen zur Verfügung, damit das Storage-Cluster erstellt werden kann. Eine Grundvoraussetzung für das Erstellen des Storage-Clusters ist die SSH Kommunikation des Deployers mit den einzelnen Storage-Cluster Servern. Damit nicht bei jedem SSH Befehl ein Passwort eingegeben werden muss, wird die SSH Konfiguration mit der Funktion generateSSHconfig angepasst. Um dies zu ermöglichen, wurden der Funktion generateSSHconfig der Hostname der Server und der Name der SSH Schlüsseldatei zur Verfügung gestellt (vgl. Listing D.8).

```
1  def generateSSHconfig(cluster)
2    Cli::LOG.info(self){"Writing the hostnames into the ssh config file ... "}
3    entries = "Host deployer \n Hostname deployer \n User ubuntu \n IdentityFile ~/.ssh/#{cluster.keypair
         }.pem \n StrictHostKeyChecking no \n UserKnownHostsFile=/dev/null \n\n"
4    cluster.machines.each do |machine|
5      if machine.role == "mon" || machine.role == "osd" || machine.role == "storage"
6        entry = "Host #{machine.hostname} \n Hostname #{machine.hostname} \n User ubuntu \n
             IdentityFile ~/.ssh/#{cluster.keypair}.pem \n StrictHostKeyChecking no \n UserKnownHostsFile=/
             dev/null \n\n"
7        entries << entry
8      else
9        entry = "Host #{machine.hostname} \n Hostname #{machine.hostname} \n User core \n IdentityFile
             ~/.ssh/#{cluster.keypair}.pem \n StrictHostKeyChecking no \n UserKnownHostsFile=/dev/null \n
             \n"
10       entries << entry
11     end
12   end
13   tmp_exe = "sudo truncate -s0 ~/.ssh/config"
14   Cli::LOG.debug(self){"Running command: #{tmp_exe}"}
15   tmp_out = `#{tmp_exe}`
16   tmp_exe = "echo \"#{entries}\" >> ~/.ssh/config"
17   Cli::LOG.debug(self){"Running command: #{tmp_exe}"}
18   tmp_out = `#{tmp_exe}`
19 end
```

Listing D.8: Klasse Storage - Funktion generateSSHconfig

Der Deployer nutzt für den Zugriff auf die Storage-Nodes nicht nur die IP Adresse, sondern auch die Hostnamen. Damit dies funktioniert, wurde die Funktion setHosts erstellt, die die Host-Datei des Deployers entsprechend anpasst (vgl. Listing D.9).

```
1  def setHosts(cluster)
2    Cli::LOG.info(self){"Writing host ip and hostname into /etc/hosts ... "}
3    entries = '127.0.0.1 \t deployer \n'
4    cluster.machines.each do |machine|
5      if machine.role == "mon" || machine.role == "osd" || machine.role == "storage"
6        entries << getHostFileEntry(cluster,machine.uid)
7      end
8    end
9    tmp_exe = "sudo truncate -s0 /etc/hosts"
10   Cli::LOG.debug(self){"Running command: #{tmp_exe}"}
11   tmp_out = `#{tmp_exe}`
12   tmp_exe = "echo \"#{entries}\" >> /etc/hosts"
13   Cli::LOG.debug(self){"Running command: #{tmp_exe}"}
14   tmp_out = `#{tmp_exe}`
15   entries = ''
16 end
```

Listing D.9: Klasse Storage - Funktion setHosts

D.1.7 Neue Klasse CephfsStorage

Die Klasse CephfsStorage enthält die Funktionen, die benötigt werden, um ein Storage-Cluster basierend auf CephFS zu implementieren. Für das Erstellen des Storage-Clusters wurde das offizielle Tool ceph-deploy[1] genutzt. ceph-deploy ermöglicht das einfache und schnelle Bereitstellen eines Ceph-Clusters. Es wurde ausgewählt, da es für die Konfiguration der einzelnen Server ebenfalls SSH nutzt.

Die folgende Befehlsfolge wurde für das Erstellen des Ceph Storage-Clusters entwickelt und als neue Funktion generate implementiert:

1. Ceph-Cluster Konfigurationsdatei *ceph.conf* erstellen. Befehl: *ceph-deploy mon1*.
2. Zusätzliche Informationen in die Ceph-Cluster Konfigurationsdatei schreiben (Netzwerkbereich des Clusters, Größe des Ceph Pools).
3. Ceph auf allen Servern installieren. Befehl: *ceph-deploy install <hostname>*.
4. Ceph Monitor Server erstellen. Befehl: *ceph-deploy mon create-initial*.
5. Monitor-Keyring erstellen, um zukünftig weitere Monitor-Server hinzufügen zu können. Befehl: *ceph-deploy gatherkeys mon1*.
6. OSD-Server vorbereiten. Befehl: *ceph-deploy osd prepare <osd-hostname:/directory/>*.
7. OSD-Server aktivieren. Befehl: *ceph-deploy osd activate <osd-hostname:/directory/>*.
8. Administrator-Keyring auf alle Storage-Cluster Server verteilen. Befehl: *ceph-deploy admin <hostname>*

Nachdem das Storage-Cluster erstellt wurde, müssen der Master und die Nodes in das Storage-Cluster integriert werden. Um dies zu ermöglichen, wurde eine weitere Funktion addClusterNodes entwickelt (vgl. Listing D.10).

```
1  def addClusterNodes(cluster)
2    entries = ''
3    disk = 'missing'
4    cluster.machines.each do |machine|
5      if !(Storage.getClusterRoles.include? machine.role)
6        Cli::LOG.info("Adding cluster nodes to the storage cluster ...")
7        command = "sudo modprobe rbd"
8        sshSend(machine.hostname,command)
9        command = "sudo rbd"
10       sshSend(machine.hostname,command)
11       command = "sudo chown core /etc/ceph"
12       sshSend(machine.hostname,command)
13       tmp_exe = "scp ceph.conf core@#{machine.hostname}:/etc/ceph"
14       Cli::LOG.debug(self){"Running command: #{tmp_exe}"}
15       tmp_out = `#{tmp_exe}`
16       wait(10)
17       tmp_exe = "scp ceph.client.admin.keyring core@#{machine.hostname}:/etc/ceph"
18       Cli::LOG.debug(self){"Running command: #{tmp_exe}"}
19       tmp_out = `#{tmp_exe}`
20       wait(10)
21       if disk == 'missing'
22         command = "sudo rbd create disk --size 1000"
23         sshSend(machine.hostname,command)
```

[1] http://docs.ceph.com/docs/hammer/man/8/ceph-deploy/

```
24        end
25        command = "sudo rbd feature disable disk  exclusive -lock object-map fast-diff deep-flatten"
26        sshSend(machine.hostname,command)
27        wait(10)
28        command = "sudo rbd map disk"
29        sshSend(machine.hostname,command)
30        command = "sudo mkdir -p /mnt/disk"
31        sshSend(machine.hostname,command)
32        if disk == 'missing'
33          command = "sudo mkfs.xfs /dev/rbd0"
34          sshSend(machine.hostname,command)
35          disk = 'build'
36        end
37        command = "sudo mount /dev/rbd0 /mnt/disk"
38        sshSend(machine.hostname,command)
39      end
40    end
41  end
```

Listing D.10: Klasse CephfsStorage - Funktion addClusterNodes

Von der Funktion addClusterNodes wird das Docker Image *ceph/rbd* genutzt, um in CoreOS Befehle für ein RADOS Block Device (RBD) ausführen zu können. Mit Hilfe von RADOS (Reliable Autonomic Distributed Object Store) lässt sich ein verteilter Blockspeicher auf der Basis von Ceph realisieren. In diesem RADOS Objektspeicher werden die Daten in RBD Images gespeichert. Zum Manipulieren der RBD Images wird die gleichnamige Anwendung rbd genutzt. Damit beim Aufrufen des Befehls rbd das richtige Docker Image angesprochen wird, wurde in der CoreOS Cloud-Config Datei das Erstellen der Datei */opt/bin/rbd* hinzugefügt (vgl. Listing D.11).

```
- path: /opt/bin/rbd
  permissions: '0775'
  content: |
    #!/bin/sh
    exec docker run -v /dev:/dev -v /sys:/sys --net=host --privileged=true -v /etc/ceph:/etc/ceph
        ceph/rbd $@
```

Listing D.11: RBD-Funktionalität in der CoreOS Cloud-Config Datei

Die wichtigsten Schritte der Funktion *addClusterNodes* sind folgende:

1. Das RBD Modul laden. Befehl: *sudo modprobe rbd*
2. Die Ceph Konfiguration und den Administrator-Keyring auf den Master-Server und die Nodes kopieren.
3. Ceph-Speicherplatz per rbd Kommando erstellen. Befehl: *sudo rbd create disk –size <Speicherplatzgröße>*
4. Mappen, Formatieren und Mounten des RBD Images.

D.1.8 Neue Klasse GlusterfsStorage

Die Klasse GlusterfsStorage wurde entwickelt, um eine Storage-Cluster basierend auf GlusterFS zu erstellen. Damit GlusterFS konfiguriert werden kann, wird im Start-Template das Installationspaket glusterfs-server angegeben. GlusterFS wird so konfiguriert, dass es die Replikation von Daten zwischen mehreren Servern zulässt. Für das Erstellen des GlusterFS Storage-Clusters wurde die Funktion generate entwickelt. Die Funktion übernimmt vorbereitende Maßnahmen wie das Formatieren der Laufwerke, das Anlegen von Verzeichnissen und das Mounten der Laufwerke und der Verzeichnisse (vgl. Listing D.12).

```
def generate( cluster )
  Cli :: LOG.info( self ){"Building the storage cluster "}
  create_volume = ''
  hostname = ''
  sleep_gluster_volume_start = 10
  mount_path = "/mnt/gluster"
  volume_name = "vol1"
  cluster .machines.each do |machine|
    if machine.role .downcase =="storage"
      command = "sudo umount /dev/#{cluster.volume_name}"
      sshSend(machine.hostname,command,cluster.keypair)
      wait
      command = "sudo mkfs.ext4 /dev/#{cluster.volume_name}"
      sshSend(machine.hostname,command,cluster.keypair)
      wait
      command = "sudo mkdir #{mount_path} -p"
      sshSend(machine.hostname,command,cluster.keypair)
      wait
      command = "sudo mount /dev/#{cluster.volume_name} #{mount_path}"
      sshSend(machine.hostname,command,cluster.keypair)
      wait
      hostname = machine.hostname
      cluster .machines.each do |machine|
        if machine.role .downcase =="storage"
          if hostname != machine.hostname
            Cli :: LOG.info( self ){"Adding storage machine with private ip #{machine.private_ipv4} to the
                cluster ... "}
            command = "sudo gluster peer probe #{machine.private_ipv4}"
            sshSend(hostname,command,cluster.keypair)
          end
        end
      end
    end
  end
  cluster .machines.each do |machine|
    if machine.role .downcase =="storage"
      create_volume_entry = machine.private_ipv4 + ":#{mount_path} "
      create_volume << create_volume_entry
    end
  end
  command = "sudo gluster volume create #{volume_name} replica 2 transport tcp #{create_volume}force"
  sshSend(hostname,command,cluster.keypair)
  command = "sudo gluster volume start #{volume_name}"
  sshSend(hostname,command,cluster.keypair)
  Cli :: LOG.info( self ){"Started storage volume. Waiting #{sleep_gluster_volume_start} seconds before
      mounting a directory ... "}
  sleep (sleep_gluster_volume_start)
end
```

Listing D.12: Klasse GlusterfsStorage - Funktion generate(cluster)

Anschließend werden die Storage-Nodes untereinander mit dem Befehl *sudo gluster peer probe <IP-Adresse>* zu einem gemeinsamen GlusterFS Peer Pool verknüpft. Darauf basierend wird ein Volume erstellt.

Die Klasse GlusterfsStorage benötigt keine weitere Funktion, um die Clients zu dem Storage-Cluster hinzuzufügen. Für den Zugriff auf das GlusterFS Storage-Cluster kann unter anderem ein NFS Client genutzt werden. Da seitens CoreOS ein NFS Client unterstützt wird, ergibt sich ein entscheidender

Vorteil bei der Verknüpfung der Clients mit dem Storage-Cluster, da keine weitere Software auf den CoreOS Instanzen installiert werden muss. Um die Verknüpfung per NFS beim Start von einer CoreOS Instanz herstellen zu können, müssen weitere Konfigurationseinträge in der CoreOS Cloud-Config Datei hinzugefügt werden (vgl. Listing D.13).

```
units:
  - name: rpc-statd.service
    command: start
    enable: true
  - name: mnt.mount
    command: start
    content: |
      [Mount]
      What=<storage-ip>:/vol1
      Where=/mnt
      Type=nfs
```

Listing D.13: NFS-Funktionalität in der CoreOS Cloud-Config Datei

D.1.9 Neue Klasse FlockerStorage

Da Flocker kein eigenes Storage-Cluster zur Verfügung stellt, sondern auf Storage-Backends zugreift, wurde keine Funktion generateStorage in der Klasse FlockerStorage implementiert. Das offizielle Flocker Tool ist für ein CoreOS Installation nicht vorgesehen. Zu den unterstützen Linux Distributionen gehören Red Hat Enterprise Linux 7, Centos 7, Ubuntu 14.04 und Ubuntu 16.04. Darüberhinaus gibt es eine inoffizielle Flocker Version, mit der das Zusammenspiel von Flocker und CoreOS erfolgreich getestet wurde. Die prototypische Implementierung von Flocker in den Deployer basiert in einigen Teilen auf einer Demonstration von Marsden [Mar15].

Bei dieser Umsetzung wird zuerst das Container-Cluster erstellt. Anschließend werden die IP-Adressen, DNS-Namen und AWS Credentials dafür genutzt, eine Konfigurationsdatei zu erstellen, die wiederum von den inoffiziellen Flocker Tools verarbeitet wird (vgl. Listing D.14).

```
cluster_name: "<name>"
agent_nodes:
- {public: "<node1_public_ipv4>", private: "<node1_private_ipv4>"}
- {public: "<node2_public_ipv4>", private: "<node2_private_ipv4>"}
- {public: "<node3_public_ipv4>", private: "<node3_private_ipv4>"}
control_node: "<dns_name_master>"
users:
- coreuser
os: coreos
private_key_path: "/home/ubuntu/.ssh/.pem"
agent_config:
  version: 1
  control-service:
    hostname: "<dns_name_master>"
    port: 4524
  dataset:
    backend: aws
    region: "eu-central-1"
    zone: "eu-central-1b"
    access_key_id:
    secret_access_key:
```

Listing D.14: Flocker Template "flocker$_{s}torage_{t}emplate.yaml$"

Die Funktion generateFlockerUserdata liest die Datei flocker_storage_template.yaml ein und ergänzt die fehlenden Variablen. Die Funktion finalize ruft die Funktion generateFlockerUserdata auf und führt anschließend die Befehle *uft-flocker-install <FlockerUserDataFile>*, *uft-flocker-config <FlockerUserDataFile>* und *uft-flocker-plugin-install <FlockerUserDataFile>* aus.

Diese prototypische Umsetzung ist zum Zeitpunkt der Arbeit nur für Amazon (AWS) und nicht für OpenStack vorgesehen.

D.2 Klassendiagramm

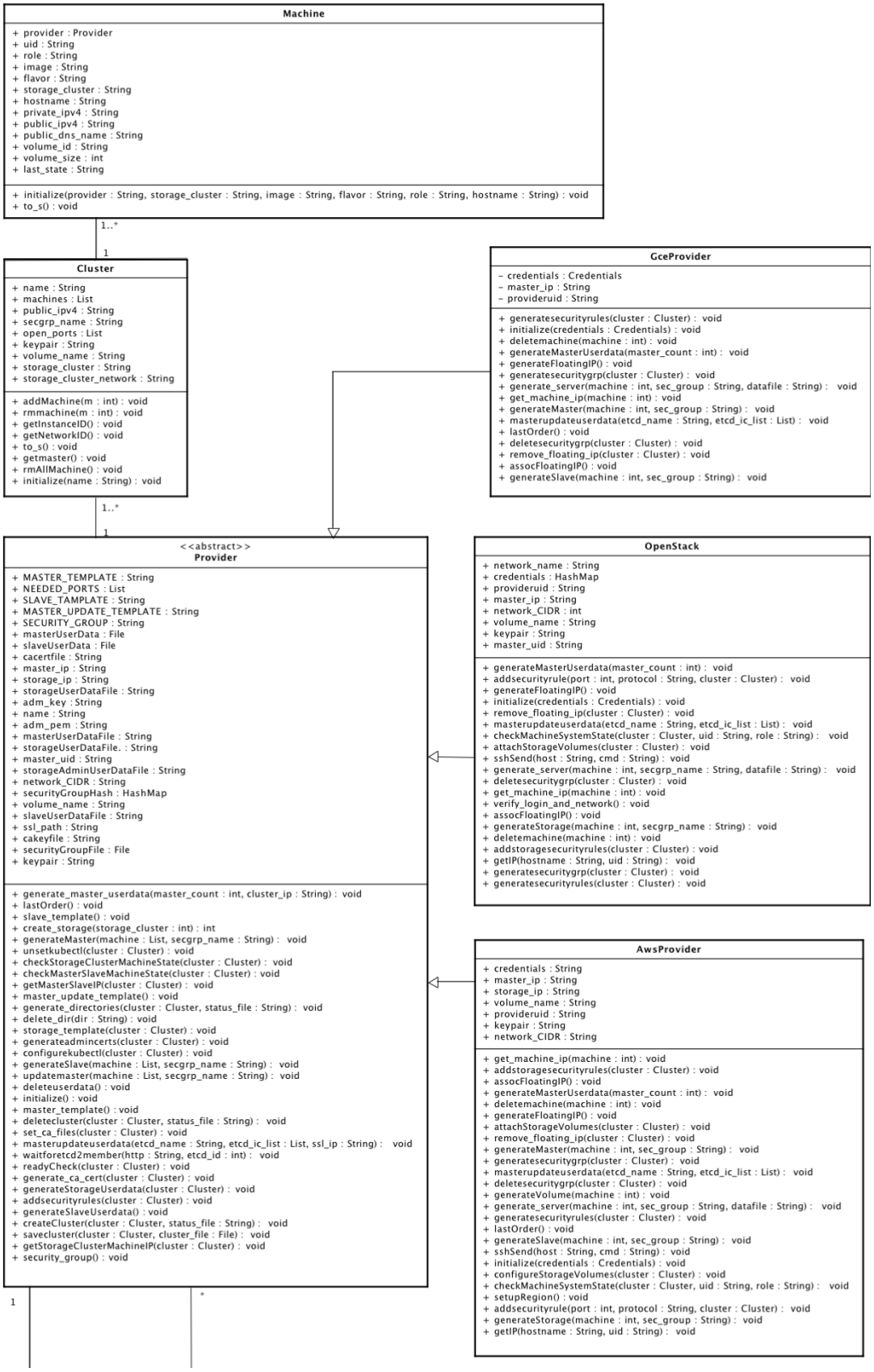

Abbildung D.1: Klassendiagramm

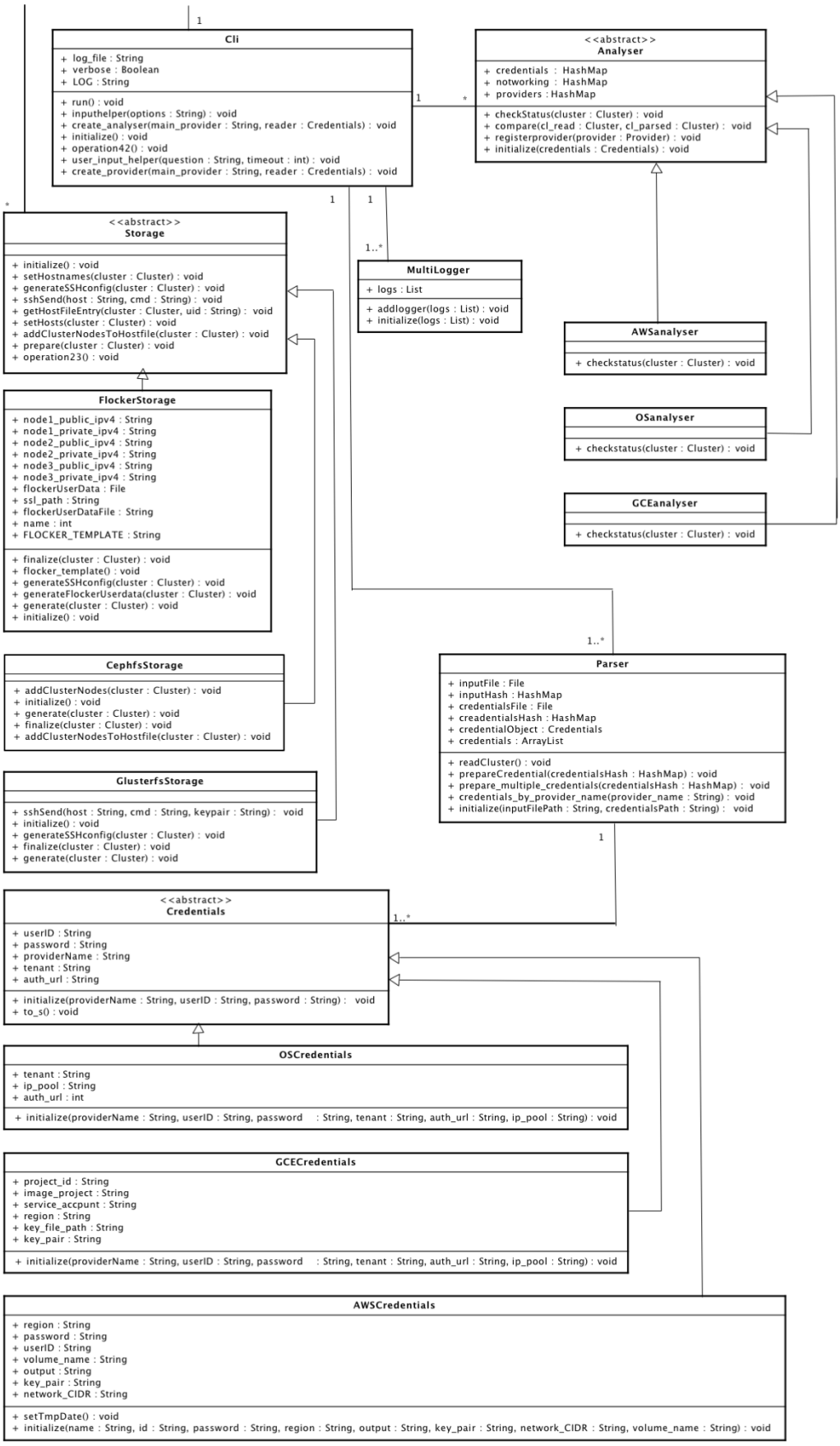

Abbildung D.2: Klassendiagramm (Fortsetzung)

D.3 Flussdiagramm

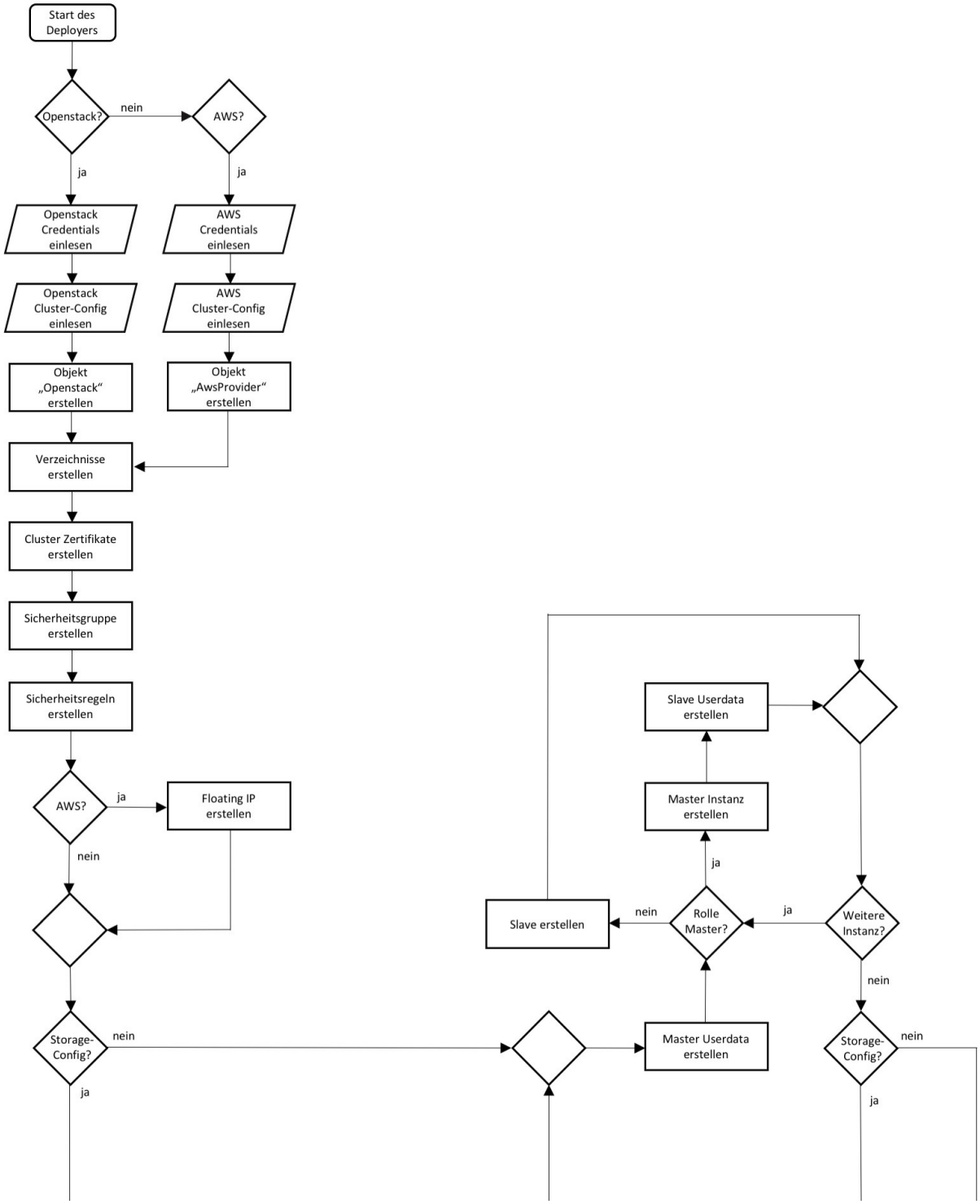

Abbildung D.3: Flussdiagramm

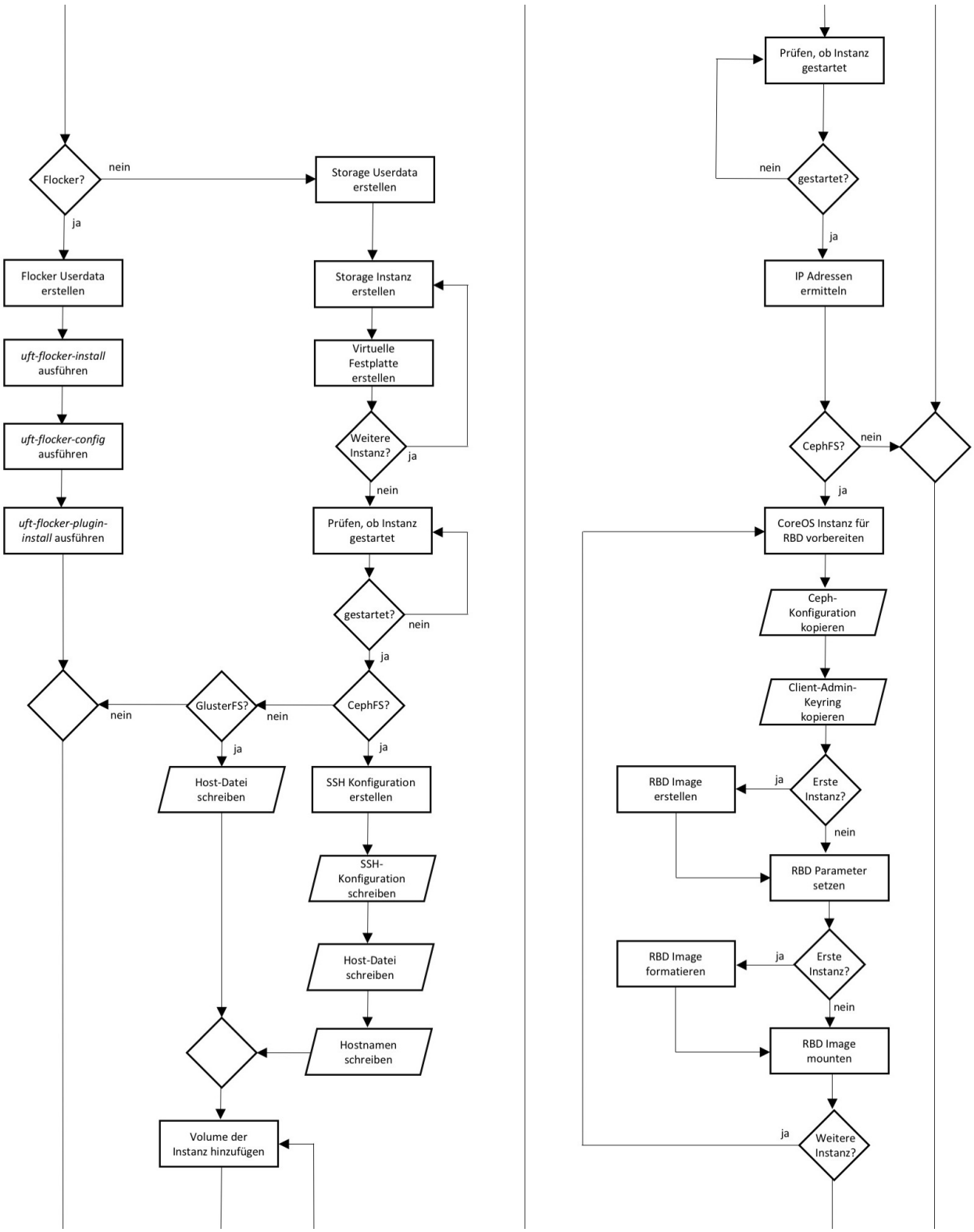

Abbildung D.4: Flussdiagramm (Fortsetzung)

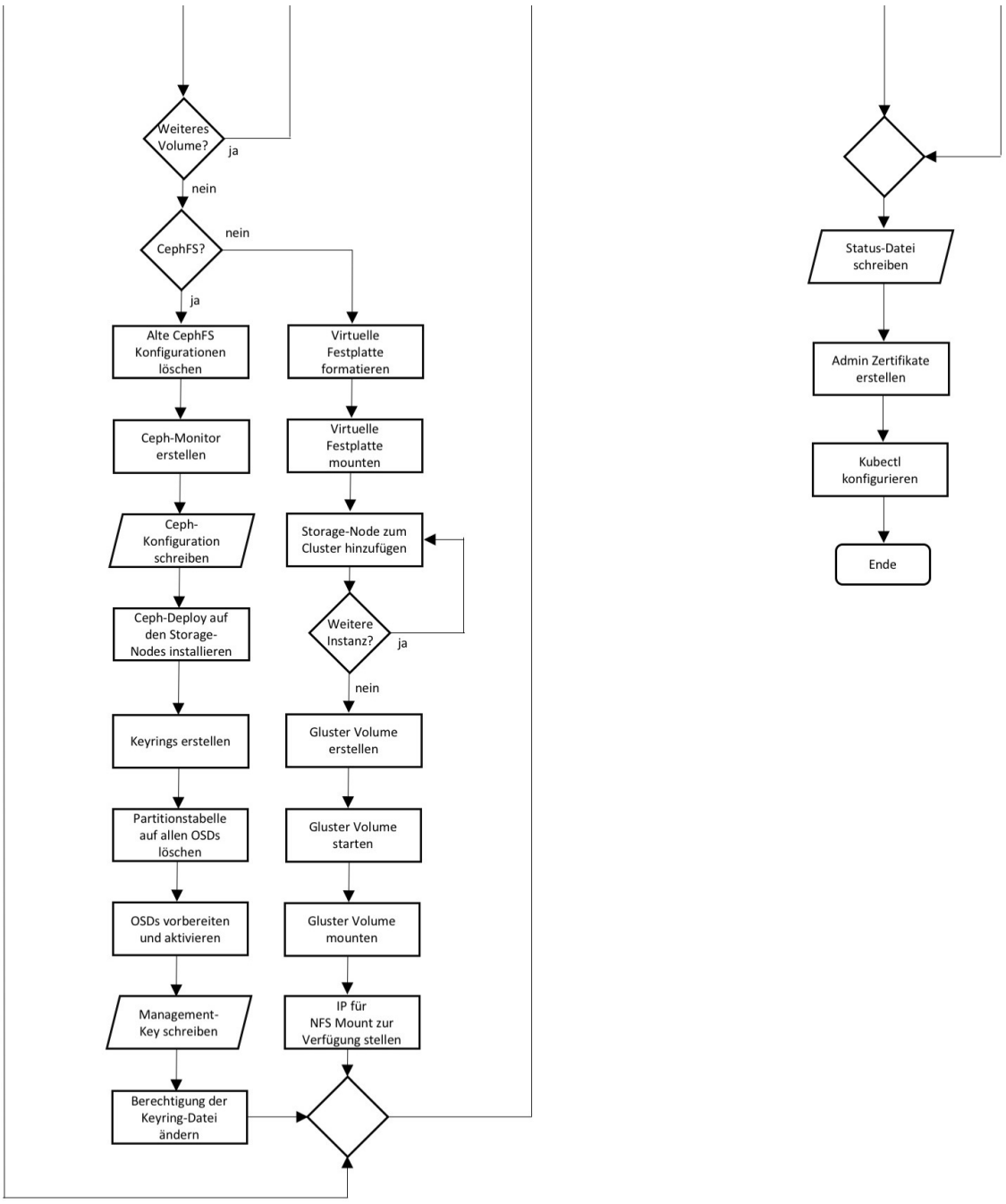

Abbildung D.5: Flussdiagramm (Fortsetzung)

D.4 Test der Deployer Erweiterung

Im folgenden Abschnitt werden die Abläufe für den Test der Deployer Erweiterung beschrieben.

D.4.1 Test von CephFS und GlusterFS

Für den Test ist eine Cluster-Konfiguration mit einer beliebigen CephFS- oder GlusterFS-Konfiguration zu wählen und auf Amazon (AWS) oder OpenStack zu erstellen. Nach dem Ausführen des Deployers steht das Storage-Cluster zusammen mit dem Kubernetes Compute-Cluster zur Verfügung. Für den

Test wird auf einem CoreOS Node in dem Storage-Cluster Mount Verzeichnis */mnt* eine index.html mit dem Text *Storage-Cluster Test* erstellt. Anschließend wird eine Konfigurationsdatei für ein Kubernetes PersistentVolume erstellt:

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: pv-volume
  labels:
    type: local
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt" #GlusterFS /mnt CephFS /mnt/disk
```

Listing D.15: PersistentVolume Konfiguration

Die Konfigurationsdatei *pv-volume.yaml* gibt den Pfad */mnt* für das Storage-Cluster und die Kapazität des PersitentVolumes mit 2 GB an. Anschließend wird das PersistentVolumes mit *kubectl create -f pv-volume.yaml* erstellt. Ein PersitentVolumeClaim wird vom Pod benutzt, um den Speicherplatz zu beanspruchen. Zum Erstellen des PersitentVolumeClaim wird folgende Konfiguration mit dem Befehl *kubectl create -f pv-claim.yaml* erstellt:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pv-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Listing D.16: PersistentVolumeClaim Konfiguration

Im nächsten Schritt wird ein Pod mit der folgenden Konfigurationsdatei mit dem Befehl *kubectl create -f pv-pod.yaml* erstellt:

```
kind: Pod
apiVersion: v1
metadata:
  name: pv-pod
spec:

  volumes:
    - name: pv-storage
      persistentVolumeClaim:
        claimName: pv-claim

  containers:
    - name: pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
```

```
18        volumeMounts:
19        - mountPath: "/usr/share/nginx/html"
20          name: pv-storage
```

Listing D.17: Pod Konfiguration

Anschließend wird die Shell des Containers, welcher in dem Pod ausgeführt wird, mit dem Befehl *kubectl exec -it pv-pod – /bin/bash* aufgerufen. In der Shell werden folgende Kommandos aufgerufen, um den Aufruf der vorher erstellten index.html auszuführen:

```
1 root@pv-pod:/# apt-get update
2 root@pv-pod:/# apt-get install curl
3 root@pv-pod:/# curl localhost
```

Listing D.18: Pod Test Kommandos

Nach dem erfolgreichen Ausführen wird die index.html mit dem Befehl *curl localhost* aufgerufen und der Text *Storage-Cluster Test* erscheint in der Kommandozeile. Beim Test folgender Situationen blieben die Daten stets erhalten:

- Löschen und neu erstellen des Pods
- Neustart der CoreOS Instanz
- Neustart eines Storage-Nodes

Die Tests zeigen, dass die Daten nach der Implementierung der Deployer Erweiterung permanent vorhanden sind.

D.4.2 Test von Flocker

Marsden demonstriert auf der Internetpräsenz von Flocker das Zusammenspiel von Flocker mit CoreOS und Amazon (AWS) [Mar15]. Die Demonstration diente als Grundlage, um den folgenden Testablauf zu definieren.

Voraussetzung für die Durchführung des Tests ist der erfolgreiche Start von Flocker. Nach Abschluss des Startvorgangs wird auf dem ersten CoreOS Node folgendes Skript ausgeführt, welches ein Flocker Volume mit dem Namen *demo* erstellt und dies anschließend mit dem erstellten Amazon Elastic Block Storage (EBS) verknüpft:

```
1 $ NODE1="<CoreOS Node 1 IP-Adresse>"
2 $ NODE2="<CoreOS Node 2 IP-Adresse>"
3 $ KEY="<Verzeichnis der .pem Datei>"
4 $ ssh -i $KEY root@$NODE1 /root/bin/docker run -d -v demo:/data --volume-driver=flocker --name
      =redis redis:latest
5 $ ssh -i $KEY root@$NODE1 /root/bin/docker run -d -e USE_REDIS_HOST=redis --link redis:redis -p
       80:80 --name=app binocarlos/moby-counter:latest
6 $ uft-flocker-volumes list
```

Listing D.19: Beispielkonfiguration CephFS und Amazon (AWS)

Das Erstellen und Hinzufügen des EBS zum Flocker demo Volume dauert ungefähr eine Minute. Anschließend wird im Browser die Adresse *http://<CoreOS Node 1 IP-Adresse>* geöffnet und mit mehreren Mausklicks werden auf dem Bildschirm Moby Dock Logos platziert. Anschließend wird die Applikation mit dem Aufruf des folgenden Skriptes auf dem zweiten CoreOS Node gestartet:

```
1 $ ssh -i $KEY root@$NODE1 /root/bin/docker rm -f app
2 $ ssh -i $KEY root@$NODE1 /root/bin/docker rm -f redis
3 $ ssh -i $KEY root@$NODE2 /root/bin/docker run -d -v demo:/data --volume-driver=flocker --name
      =redis redis:latest
4 $ ssh -i $KEY root@$NODE2 /root/bin/docker run -d -e USE_REDIS_HOST=redis --link redis:redis -p
       80:80 --name=app binocarlos/moby-counter:latest
5 $ uft-flocker-volumes list
```

Listing D.20: Beispielkonfiguration CephFS und Amazon (AWS)

Wenn das Verschieben des Flocker Volumes funktioniert hat, sind die Moby Dock Logos ebenfalls beim Aufruf der Adresse *http://<CoreOS Node 2 IP-Adresse>* vorhanden.

Anhang E

Testergebnisse

E.1 Testergebnisse Netzwerkbandbreite

E.1.1 TCP Netzwerkbandbreite Amazon (AWS) in Richtung OpenStack

```
ubuntu@storage1:~$ iperf -c 212.201.22.189 -i 1 -r
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
------------------------------------------------------------
Client connecting to 212.201.22.189, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 172.31.29.92 port 42400 connected with 212.201.22.189 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec  16.4 MBytes  137 Mbits/sec
[  3]  1.0- 2.0 sec  17.1 MBytes  144 Mbits/sec
[  3]  2.0- 3.0 sec  18.8 MBytes  157 Mbits/sec
[  3]  3.0- 4.0 sec  18.5 MBytes  155 Mbits/sec
[  3]  4.0- 5.0 sec  16.1 MBytes  135 Mbits/sec
[  3]  5.0- 6.0 sec  17.9 MBytes  150 Mbits/sec
[  3]  6.0- 7.0 sec  18.9 MBytes  158 Mbits/sec
[  3]  7.0- 8.0 sec  20.0 MBytes  168 Mbits/sec
[  3]  8.0- 9.0 sec  20.2 MBytes  170 Mbits/sec
[  3]  9.0-10.0 sec  22.4 MBytes  188 Mbits/sec
[  3]  0.0-10.0 sec   186 MBytes  156 Mbits/sec
[  5] local 172.31.29.92 port 5001 connected with 212.201.22.189 port 52916
[  5]  0.0- 1.0 sec  36.5 MBytes  306 Mbits/sec
[  5]  1.0- 2.0 sec  44.1 MBytes  370 Mbits/sec
[  5]  2.0- 3.0 sec  44.2 MBytes  370 Mbits/sec
[  5]  3.0- 4.0 sec  44.5 MBytes  373 Mbits/sec
[  5]  4.0- 5.0 sec  44.6 MBytes  374 Mbits/sec
[  5]  5.0- 6.0 sec  40.2 MBytes  337 Mbits/sec
[  5]  6.0- 7.0 sec  37.5 MBytes  315 Mbits/sec
[  5]  7.0- 8.0 sec  38.2 MBytes  320 Mbits/sec
[  5]  8.0- 9.0 sec  43.6 MBytes  366 Mbits/sec
[  5]  9.0-10.0 sec  35.6 MBytes  298 Mbits/sec
[  5]  0.0-10.1 sec   411 MBytes  343 Mbits/sec
```

Listing E.1: TCP Netzwerkbandbreite Amazon (AWS) in Richtung OpenStack

E.1.2 UDP Netzwerkbandbreite Amazon (AWS) in Richtung OpenStack

```
ubuntu@storage1:~$ iperf -c 212.201.22.189 -i 1 -u -r
------------------------------------------------------------
Client connecting to 212.201.22.189, UDP port 5001
Sending 1470 byte datagrams UDP buffer size: 208 KByte (default)
------------------------------------------------------------
[ 3] local 172.31.29.92 port 52222 connected with 212.201.22.189 port 5001
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)
------------------------------------------------------------
[ ID] Interval Transfer Bandwidth
[ 3] 0.0- 1.0 sec 59.0 MBytes 495 Mbits/sec
[ 3] 1.0- 2.0 sec 60.3 MBytes 506 Mbits/sec
[ 3] 2.0- 3.0 sec 60.5 MBytes 508 Mbits/sec
[ 3] 3.0- 4.0 sec 62.0 MBytes 520 Mbits/sec
[ 3] 4.0- 5.0 sec 62.8 MBytes 527 Mbits/sec
[ 3] 5.0- 6.0 sec 61.5 MBytes 516 Mbits/sec
[ 3] 6.0- 7.0 sec 60.5 MBytes 508 Mbits/sec
[ 3] 7.0- 8.0 sec 61.0 MBytes 512 Mbits/sec
[ 3] 8.0- 9.0 sec 60.6 MBytes 509 Mbits/sec
[ 3] 9.0-10.0 sec 60.9 MBytes 511 Mbits/sec
[ 3] 0.0-10.0 sec 609 MBytes 511 Mbits/sec
[ 4] local 172.31.29.92 port 5001 connected with 212.201.22.189 port 48180
[ 4] 0.0- 1.0 sec 17.5 MBytes 147 Mbits/sec
[ 4] 1.0- 2.0 sec 17.5 MBytes 147 Mbits/sec
[ 4] 2.0- 3.0 sec 17.5 MBytes 146 Mbits/sec
[ 4] 3.0- 4.0 sec 16.8 MBytes 141 Mbits/sec
[ 4] 4.0- 5.0 sec 17.3 MBytes 145 Mbits/sec
[ 4] 5.0- 6.0 sec 17.1 MBytes 144 Mbits/sec
[ 4] 6.0- 7.0 sec 17.6 MBytes 148 Mbits/sec
[ 4] 7.0- 8.0 sec 17.1 MBytes 144 Mbits/sec
[ 4] 8.0- 9.0 sec 17.6 MBytes 147 Mbits/sec
[ 4] 9.0-10.0 sec 16.8 MBytes 141 Mbits/sec
[ 4] 10.0-11.0 sec 21.1 MBytes 177 Mbits/sec
[ 4] 11.0-12.0 sec 20.3 MBytes 170 Mbits/sec
[ 4] 12.0-13.0 sec 20.0 MBytes 168 Mbits/sec
[ 4] 0.0-13.0 sec 235 MBytes 151 Mbits/sec
```

Listing E.2: UDP Netzwerkbandbreite Amazon (AWS) in Richtung OpenStack

E.1.3 TCP Netzwerkbandbreite OpenStack in Richtung Amazon (AWS)

```
ubuntu@storage2:~$ iperf -c 52.59.225.21 -i 1 -r
------------------------------------------------------------
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
Client connecting to 52.59.225.21, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[ 3] local 212.201.22.189 port 52902 connected with 52.59.225.21 port 5001
[ ID] Interval Transfer Bandwidth
[ 3] 0.0- 1.0 sec 23.9 MBytes 200 Mbits/sec
```

```
[  3]  1.0- 2.0 sec  19.2 MBytes  161 Mbits/sec
[  3]  2.0- 3.0 sec  21.1 MBytes  177 Mbits/sec
[  3]  3.0- 4.0 sec  24.0 MBytes  201 Mbits/sec
[  3]  4.0- 5.0 sec  23.2 MBytes  195 Mbits/sec
[  3]  5.0- 6.0 sec  26.1 MBytes  219 Mbits/sec
[  3]  6.0- 7.0 sec  26.1 MBytes  219 Mbits/sec
[  3]  7.0- 8.0 sec  24.8 MBytes  208 Mbits/sec
[  3]  8.0- 9.0 sec  26.4 MBytes  221 Mbits/sec
[  3]  9.0-10.0 sec  25.1 MBytes  211 Mbits/sec
[  3]  0.0-10.0 sec   240 MBytes  201 Mbits/sec
[  5] local 212.201.22.189 port 5001 connected with 52.59.225.21 port 42394
[  5]  0.0- 1.0 sec  8.48 MBytes 71.2 Mbits/sec
[  5]  1.0- 2.0 sec  8.85 MBytes 74.2 Mbits/sec
[  5]  2.0- 3.0 sec  8.42 MBytes 70.6 Mbits/sec
[  5]  3.0- 4.0 sec  8.99 MBytes 75.4 Mbits/sec
[  5]  4.0- 5.0 sec  9.65 MBytes 80.9 Mbits/sec
[  5]  5.0- 6.0 sec  10.1 MBytes 84.8 Mbits/sec
[  5]  6.0- 7.0 sec  9.27 MBytes 77.8 Mbits/sec
[  5]  7.0- 8.0 sec  9.57 MBytes 80.3 Mbits/sec
[  5]  8.0- 9.0 sec  9.04 MBytes 75.8 Mbits/sec
[  5]  9.0-10.0 sec  8.31 MBytes 69.7 Mbits/sec
[  5]  0.0-10.2 sec  92.5 MBytes 75.9 Mbits/sec
```

Listing E.3: TCP Netzwerkbandbreite OpenStack in Richtung Amazon (AWS)

E.1.4 UDP Netzwerkbandbreite OpenStack in Richtung Amazon (AWS)

```
ubuntu@storage2:~$ iperf -c 52.59.225.21 -i 1 -u -r
------------------------------------------------------------
Client connecting to 52.59.225.21, UDP port 5001
Sending 1470 byte datagrams UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 212.201.22.189 port 52222 connected with 52.59.225.21 port 5001
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec  96.5 MBytes  809 Mbits/sec
[  3]  1.0- 2.0 sec  95.8 MBytes  804 Mbits/sec
[  3]  2.0- 3.0 sec  96.2 MBytes  807 Mbits/sec
[  3]  3.0- 4.0 sec  95.8 MBytes  804 Mbits/sec
[  3]  4.0- 5.0 sec  96.2 MBytes  807 Mbits/sec
[  3]  5.0- 6.0 sec  95.6 MBytes  802 Mbits/sec
[  3]  6.0- 7.0 sec  96.4 MBytes  809 Mbits/sec
[  3]  7.0- 8.0 sec  96.6 MBytes  811 Mbits/sec
[  3]  8.0- 9.0 sec  96.2 MBytes  807 Mbits/sec
[  3]  9.0-10.0 sec  95.6 MBytes  802 Mbits/sec
[  3]  0.0-10.0 sec   961 MBytes  806 Mbits/sec
[  3] Sent 685402 datagrams
[  4] local 212.201.22.189 port 5001 connected with 52.59.225.21 port 37618
[  4]  0.0- 1.0 sec  33.1 MBytes  278 Mbits/sec
[  4]  1.0- 2.0 sec  33.2 MBytes  279 Mbits/sec
[  4]  2.0- 3.0 sec  33.4 MBytes  280 Mbits/sec
[  4]  3.0- 4.0 sec  32.9 MBytes  276 Mbits/sec
[  4]  4.0- 5.0 sec  33.3 MBytes  279 Mbits/sec
```

```
[  4]  5.0- 6.0 sec  33.2 MBytes  278 Mbits/sec
[  4]  6.0- 7.0 sec  33.6 MBytes  282 Mbits/sec
[  4]  7.0- 8.0 sec  34.0 MBytes  285 Mbits/sec
[  4]  8.0- 9.0 sec  33.7 MBytes  282 Mbits/sec
[  4]  9.0-10.0 sec  33.9 MBytes  285 Mbits/sec
[  4] 10.0-11.0 sec  35.3 MBytes  296 Mbits/sec
[  4] 11.0-12.0 sec  34.8 MBytes  292 Mbits/sec
[  4]  0.0-12.6 sec   426 MBytes  283 Mbits/sec
```

Listing E.4: UDP Netzwerkbandbreite OpenStack in Richtung Amazon (AWS)

E.2 Testergebnisse Schreib- und Lesegeschwindigkeit der Storage-Cluster

Für den Test der Schreib- und Lesegeschwindigkeit wurde das Open Source Programm IOzone[1] genutzt. IOzone eignet sich für die Installation für das im Testaufbau verwendete Linux Betriebssystem. Zur IOzone Installation sind folgende Schritte notwendig:

```
wget http://www.iozone.org/src/current/iozone3_465.tar
tar xvf iozone3_465.tar
cd iozone3_465/src/current
make
make linux
```

Listing E.5: IOzone Installation

Die Storage-Cluster, die exklusiv bei Amazon (AWS) und OpenStack getestet wurden, wurden mit Hilfe des Deployers erstellt. Da zum Zeitpunkt dieser Arbeit noch keine Multicluster Konfiguration zur Verfügung stand, mussten die plattformübergreifenden Storage-Cluster manuell erstellt werden. Folgende Schritte sind zum Erstellen der plattformübergreifenden Storage-Cluster notwendig:

1. Starten der t2.micro bzw. t2.large Instanzen auf Amazon (AWS) und der m1.mirco bzw. m1.large.storage Instanzen auf OpenStack jeweils mit dem Linux Betriebsystem Ubuntu 16.04.02 LTS. Je nach Größe des Storage-Clusters werden jeweils eine, zwei oder drei Instanzen gestartet.

2. Installieren von GlusterFS mit dem Befehl *sudo apt-get install glusterfs-server attr -y* auf jeder Instanz.

3. Erstellen der Verzeichnisse */mnt/storage* und */mnt/gluster* auf jeder Instanz.

4. Sicherstellen, dass alle Instanzen sich gegenseitig erreichen können. Gegebenenfalls ist ein Anpassen der Sicherheitsgruppen notwendig.

5. Hinzufügen aller GlusterFS Nodes zum Storage-Cluster mit dem Befehl *sudo gluster peer probe <IP-Adresse des GlusterFS Nodes>*. Der Befehl wird wird so oft auf einem GlusterFS Node mit den IP-Adressen aller anderen GlusterFS Nodes ausgeführt, bis alle GlusterFS Nodes eingeladen wurden. Bei einem plattformübergreifenden Storage-Cluster mit zwei GlusterFS Nodes muss der Befehl einmal ausgeführt werden. Bei vier GlusterFS Nodes muss der Befehl dreimal und bei sechs GlusterFS Nodes fünfmal ausgeführt werden.

6. Erstellen des GlusterFS Volumes für zum Beispiel zwei GlusterFS Nodes mit dem Befehl *sudo gluster volume create vol1 replica 2 transport tcp <IP-Adresse des ersten GlusterFS Nodes>:/mnt/gluster <IP-Adresse des zweiten GlusterFS Nodes>:/mnt/gluster force*. Zum Erstellen eines GlusterFS Volume mit mehr als zwei GlusterFS Nodes müssen die IP-Adressen der anderen GlusterFS Nodes ebenfalls zu dem Befehl zum Erstellen des GlusterFS Volumes hinzugefügt werden.

[1] http://www.IOzone.org/

7. Starten des GlusterFS Volumes mit dem Befehl *sudo gluster volume start vol1*.

8. Erstellen des Storage-Cluster Zugriffs mit dem Befehl *sudo mount -t glusterfs 127.0.0.1:vol1 /mnt/storage*.

Nach dem Fertigstellen des plattformübergreifenden Storage-Cluster wurden die Tests in dem Verzeichnis */mnt/storage* ausgeführt.

E.2.1 Storage-Cluster basierend auf OpenStack

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	2,02	4,76	3,00	3,43	4,93								
128	6,30	9,63	10,43	10,78	11,05	8,01							
256	9,17	9,39	16,47	18,33	18,81	19,36	18,97						
512	10,92	10,19	25,85	25,64	27,42	28,13	28,18	21,97					
1024	13,11	21,96	23,95	32,22	37,94	38,63	38,50	21,60	38,66				
2048	12,70	17,40	23,39	12,93	21,47	28,60	46,20	47,24	46,72	45,21			
4096	13,23	19,41	40,54	44,47	32,53	52,34	52,12	43,01	38,19	52,57	41,49		
8192	12,99	20,60	29,55	26,87	29,16	54,51	40,27	52,49	53,87	27,30	41,79	50,65	
16384	15,77	43,19	22,42	21,99	54,81	47,51	41,34	52,12	29,40	34,02	58,47	51,87	30,87
32768					22,12	59,27	45,94	54,91	45,66	29,96	54,93	59,70	56,03
65536					34,48	49,88	30,49	38,13	57,84	55,51	61,11	44,19	33,54
131072					51,08	50,85	39,07	78,36	40,17	59,42	66,03	71,08	62,59
262144					66,91	91,41	84,48	64,27	63,39	76,95	83,78	57,42	94,02
524288					42,35	66,54	65,59	67,47	70,72	59,04	82,43	83,85	68,76
1048576					57,93	62,26	64,91	70,04	64,64	61,33	63,49	74,57	63,44
2097152					59,70	63,05	60,82	80,21	63,59	73,90	64,86	66,28	58,06

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.1: Schreibgeschwindigkeit OpenStack mit zwei GlusterFS Nodes und Einkern-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	12,03	12,38	11,63	13,97	13,97								
128	5,74	21,37	22,08	24,29	27,45	28,33							
256	38,83	32,84	42,36	43,13	43,67	46,83	54,56						
512	57,08	58,75	60,35	58,02	60,71	58,62	77,32	104,77					
1024	70,26	68,79	74,05	71,58	76,18	77,28	84,50	116,87	223,38				
2048	79,61	39,57	20,40	25,63	26,87	31,71	96,04	116,42	168,88	354,75			
4096	89,15	88,75	94,25	91,23	95,10	95,17	99,04	118,06	28,12	220,59	231,10		
8192	28,60	19,33	99,84	20,08	104,31	100,77	102,18	108,88	156,07	179,40	302,59	997,79	
16384	19,87	99,46	21,68	132,38	20,91	108,85	99,45	110,22	153,14	34,14	287,99	270,10	1326,64
32768					19,74	97,96	104,75	120,52	27,13	179,40	46,28	334,08	388,29
65536					52,31	109,40	107,68	123,65	91,56	187,60	210,48	142,38	54,57
131072					24,66	123,10	112,32	134,87	136,25	39,55	207,67	229,31	250,49
262144					25,44	22,17	59,67	25,47	54,03	48,53	35,68	50,66	256,10
524288					37,01	23,65	45,46	39,70	77,34	86,33	70,14	91,66	108,67
1048576					24,93	28,64	26,87	28,25	67,76	67,22	92,03	91,44	100,52
2097152					33,03	27,81	24,67	34,53	64,03	80,69	86,72	85,16	100,87

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.2: Lesegeschwindigkeit OpenStack mit zwei GlusterFS Nodes und Einkern-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	4,14	4,64	3,10	6,51	6,85								
128	9,66	10,22	10,85	5,64	10,92	10,81							
256	13,95	16,08	17,06	10,55	6,37	10,94	16,39						
512	17,60	11,90	23,60	23,78	23,82	8,73	23,28	23,29					
1024	16,85	20,05	29,35	30,27	28,85	29,57	19,85	24,66	21,80				
2048	17,12	26,85	30,73	33,79	32,51	32,80	12,45	33,30	33,22	33,75			
4096	18,58	25,86	36,22	35,78	35,23	36,09	23,90	36,10	35,98	36,12	36,29		
8192	19,82	30,17	37,52	37,54	37,48	32,76	37,44	26,36	37,68	37,71	37,83	37,63	
16384	21,89	28,26	21,89	37,87	31,45	38,69	25,12	26,37	38,70	38,69	38,70	38,74	38,71
32768					38,45	39,25	39,28	34,38	33,83	39,17	39,27	39,33	36,18
65536					41,08	36,78	40,03	40,56	40,64	40,80	40,82	37,66	34,20
131072					47,84	49,23	49,23	48,78	47,03	47,94	49,12	49,14	49,43
262144					51,49	51,58	48,78	50,55	50,59	51,86	51,69	47,13	51,04
524288					51,19	50,78	52,71	53,72	48,37	53,60	47,39	50,76	46,79
1048576					50,91	47,95	51,08	51,17	51,96	50,27	49,51	50,36	50,43
2097152					48,83	49,73	49,97	49,96	49,21	50,65	49,58	49,78	48,74

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.3: Schreibgeschwindigkeit OpenStack mit vier GlusterFS Nodes und Einkern-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	9,82	9,90	9,02	12,79	13,33								
128	14,12	14,42	17,00	18,26	26,24	24,71							
256	24,30	23,60	29,32	28,83	38,50	56,14	50,18						
512	35,77	37,64	39,35	39,82	42,60	49,29	58,78	99,30					
1024	38,73	39,21	41,50	40,73	39,14	32,12	35,30	79,12	205,01				
2048	37,05	36,41	40,23	38,39	37,84	39,98	42,53	56,43	97,70	323,33			
4096	29,90	37,86	38,08	38,65	40,11	37,77	41,39	50,19	73,92	117,29	346,01		
8192	38,67	38,97	38,31	38,99	40,01	40,12	41,02	48,46	70,59	96,51	118,13	575,73	
16384	39,37	39,53	40,29	39,55	39,61	40,25	40,47	48,34	65,82	90,83	117,79	184,47	462,20
32768					38,94	38,12	38,35	47,44	53,43	65,73	86,58	122,69	203,53
65536					40,04	40,22	40,22	46,72	67,14	86,15	102,44	83,99	142,42
131072					39,73	40,23	40,37	47,09	66,87	84,64	95,45	111,22	124,49
262144					39,11	39,41	40,82	45,90	66,03	81,75	91,10	108,04	116,77
524288					41,21	40,52	40,33	44,72	61,62	79,14	97,27	91,07	105,83
1048576					41,22	41,07	41,38	50,32	68,16	86,45	97,42	102,34	93,71
2097152					40,30	41,20	41,22	50,64	67,58	85,52	94,07	97,81	86,35

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.4: Lesegeschwindigkeit OpenStack mit vier GlusterFS Nodes und Einkern-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	3,29	3,18	4,65	4,89	4,74								
128	6,14	9,71	10,25	11,21	11,09	10,43							
256	8,85	15,87	18,68	19,30	19,00	18,53	18,43						
512	7,73	17,66	23,33	27,92	11,51	8,55	27,80	26,86					
1024	10,30	13,95	32,37	36,02	37,79	39,06	36,00	38,29	37,88				
2048	11,00	21,86	23,17	16,87	17,66	24,64	31,09	26,71	34,11	17,36			
4096	13,93	20,87	26,49	17,79	50,37	52,56	50,84	51,87	51,67	34,21	51,80		
8192	12,88	34,26	15,48	42,71	48,29	54,49	53,46	55,39	44,89	40,04	20,80	48,52	
16384	14,27	23,39	28,76	50,67	48,07	42,31	37,50	25,82	25,86	57,40	31,21	28,38	53,24
32768					49,20	58,07	59,33	52,09	59,36	52,73	56,18	53,57	57,50
65536					39,38	39,29	61,09	60,11	50,75	42,25	61,28	55,62	34,56
131072					58,98	65,89	45,48	49,25	78,88	79,05	62,08	72,62	71,00
262144					59,41	63,91	56,58	71,21	65,96	84,20	77,89	73,89	80,36
524288					46,69	99,44	59,85	64,35	64,68	84,76	64,83	79,21	54,55
1048576					69,72	72,54	73,37	77,29	90,09	63,98	62,43	62,11	75,81
2097152					63,40	60,40	59,72	61,76	75,83	74,66	69,73	74,31	66,46

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.5: Schreibgeschwindigkeit OpenStack mit sechs GlusterFS Nodes und Einkern-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	11,99	14,92	14,17	17,49	15,74								
128	23,27	24,18	24,42	28,02	33,93	30,47							
256	46,03	44,07	47,29	45,95	47,45	51,08	64,94						
512	63,05	57,97	60,97	60,19	59,38	71,29	79,79	117,00					
1024	42,95	32,55	78,32	75,91	77,71	80,41	89,99	114,13	211,75				
2048	19,14	83,61	21,34	22,40	47,65	26,13	22,57	35,51	41,36	102,29			
4096	90,93	90,09	20,82	94,44	95,39	95,50	100,53	114,59	156,16	72,92	663,45		
8192	125,10	121,40	21,31	98,82	98,77	98,00	103,36	117,40	30,91	47,54	271,81	447,48	
16384	18,37	129,66	96,83	102,32	21,33	104,04	19,21	115,32	151,77	188,06	44,79	322,20	1409,87
32768					34,04	102,60	101,27	120,52	157,65	187,11	213,09	259,67	390,38
65536					106,21	26,53	113,86	123,31	45,87	189,51	212,31	96,87	279,58
131072					22,91	21,04	23,18	148,29	161,39	192,53	212,07	194,78	245,02
262144					19,43	22,48	34,14	34,61	62,39	33,82	48,62	54,40	62,96
524288					32,88	46,67	40,37	70,10	90,04	85,33	99,73	99,89	108,82
1048576					19,04	32,99	44,28	46,99	76,45	85,93	96,11	100,48	104,16
2097152					27,63	19,59	27,43	30,07	78,95	83,59	86,74	95,93	97,69

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.6: Lesegeschwindigkeit OpenStack mit sechs GlusterFS Nodes und Einkern-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	3,86	5,14	3,86	5,15	5,09								
128	8,35	11,75	11,05	11,20	2,63	10,42							
256	8,96	11,98	16,37	16,35	15,06	16,20	18,58						
512	14,06	23,15	24,41	26,63	27,53	27,92	27,32	24,09					
1024	15,44	29,36	30,88	35,65	39,25	37,98	34,70	39,82	38,93				
2048	22,56	30,84	41,82	42,09	44,38	43,87	40,88	47,49	46,65	46,14			
4096	10,97	38,42	45,31	46,59	43,60	52,03	30,04	51,75	49,43	52,38	52,26		
8192	15,13	22,52	32,59	42,48	53,42	55,73	39,33	47,61	55,53	55,24	52,91	55,02	
16384	20,46	22,48	39,56	51,55	50,02	56,82	57,29	57,79	55,71	57,85	58,29	57,99	57,92
32768					44,13	51,53	59,24	59,05	58,87	55,88	57,01	59,26	58,89
65536					48,02	56,06	59,11	58,03	59,49	57,10	59,85	59,70	59,94
131072					48,02	55,65	60,08	59,77	60,44	59,85	60,58	60,19	58,96
262144					48,45	54,30	56,95	59,70	59,05	60,95	59,98	58,02	58,47
524288					49,83	62,32	60,16	62,29	68,23	71,08	71,27	69,31	69,21
1048576					69,08	74,57	80,25	78,19	72,83	85,29	83,38	78,87	84,52
2097152					73,52	79,96	70,54	73,00	76,00	84,46	82,30	80,23	79,97

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.7: Schreibgeschwindigkeit OpenStack mit zwei GlusterFS Nodes und Zwei-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	11,41	10,46	11,89	13,75	13,99								
128	21,47	21,56	23,69	24,88	29,66	27,55							
256	40,19	39,16	43,37	42,70	40,18	60,98	62,24						
512	55,78	58,98	72,33	62,00	52,82	79,90	75,76	103,90					
1024	70,60	72,62	80,95	91,26	89,27	63,15	95,48	122,88	214,10				
2048	84,00	86,43	99,80	90,42	103,65	101,04	102,30	121,71	136,66	369,34			
4096	90,04	95,29	106,53	99,33	103,31	106,73	114,71	147,92	164,46	235,23	131,64		
8192	106,65	101,84	70,37	110,00	95,05	113,79	73,14	170,61	163,33	277,28	316,92	918,98	
16384	82,34	104,26	77,64	77,93	78,04	107,37	94,71	81,89	188,20	202,46	312,44	416,72	1367,39
32768					96,46	101,70	102,29	111,35	173,58	120,51	206,22	305,82	333,75
65536					103,12	112,93	116,77	128,16	137,79	176,65	195,82	165,02	185,03
131072					99,97	95,49	113,04	114,75	156,61	160,71	204,95	205,80	223,32
262144					101,40	118,33	101,72	131,40	172,84	209,64	189,53	171,37	289,34
524288					92,33	105,78	107,98	110,30	132,15	173,88	191,57	200,84	242,69
1048576					105,83	110,95	107,07	118,35	148,83	160,93	179,38	190,72	192,42
2097152					91,39	94,44	95,11	108,96	129,50	134,59	140,11	146,89	147,80

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.8: Lesegeschwindigkeit OpenStack mit zwei GlusterFS Nodes und Zwei-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	2,82	4,31	4,65	3,45	4,31								
128	8,14	9,25	9,12	5,65	9,96	9,95							
256	11,64	16,59	17,92	18,87	16,34	19,21	19,39						
512	16,82	18,23	25,83	28,24	29,87	28,84	29,57	29,18					
1024	16,36	21,91	32,72	33,57	30,70	33,09	38,69	37,36	37,98				
2048	23,94	30,62	40,57	46,46	44,40	46,91	44,72	46,58	46,12	40,90			
4096	17,65	31,00	45,07	43,07	49,12	51,79	51,19	51,66	51,69	52,35	51,05		
8192	21,39	25,95	37,93	51,11	55,54	54,98	55,65	54,91	55,23	52,74	55,28	52,29	
16384	18,73	41,64	48,04	54,72	53,12	58,21	57,74	57,59	58,04	57,80	57,50	57,24	57,43
32768					59,16	59,44	59,19	58,31	57,72	59,47	59,21	59,12	58,10
65536					59,31	59,65	59,87	59,86	59,85	58,00	58,91	59,43	60,31
131072					59,48	60,54	59,31	60,57	60,77	60,33	59,59	60,40	60,97
262144					56,57	60,86	60,71	60,01	60,88	60,78	60,83	59,91	60,74
524288					69,76	71,58	71,26	71,37	57,80	71,97	71,56	71,04	71,45
1048576					78,09	75,81	76,54	83,33	86,32	83,00	83,16	82,34	81,80
2097152					87,66	80,66	79,22	85,50	70,98	84,03	80,38	89,51	90,86

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.9: Schreibgeschwindigkeit OpenStack mit vier GlusterFS Nodes und Zwei-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	12,81	11,22	12,89	15,74	14,44								
128	17,46	22,55	22,85	24,12	28,97	28,66							
256	35,05	36,75	43,64	41,08	43,76	50,81	54,63						
512	67,09	59,12	72,60	62,24	61,98	68,34	85,57	113,78					
1024	67,02	87,45	89,94	84,16	65,67	93,52	85,11	174,54	196,54				
2048	96,84	113,85	58,16	99,76	118,47	114,55	103,79	145,37	207,83	367,94			
4096	109,77	115,26	109,87	125,62	145,93	110,12	126,71	141,28	192,76	275,94	571,98		
8192	106,69	130,09	117,45	106,66	75,11	100,22	149,30	154,13	182,58	241,96	351,77	922,93	
16384	71,61	134,88	138,99	97,30	123,43	141,46	85,65	115,54	116,57	162,59	177,88	396,85	1308,72
32768					114,48	126,73	147,50	142,60	194,15	185,97	284,26	344,97	455,58
65536					114,80	128,35	114,28	123,51	190,66	181,97	264,37	275,28	318,81
131072					125,25	116,68	119,34	125,71	192,30	226,55	229,28	285,00	284,73
262144					121,87	128,13	130,42	131,97	151,52	202,08	220,16	257,46	252,64
524288					123,45	121,41	126,44	139,73	189,23	209,15	260,95	290,00	274,14
1048576					121,33	124,17	120,29	140,88	183,18	205,77	227,67	234,29	267,42
2097152					108,65	111,85	111,99	125,95	143,55	157,68	166,53	169,21	176,61

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.10: Lesegeschwindigkeit OpenStack mit vier GlusterFS Nodes und Zwei-Prozessoren

E.2.2 Storage-Cluster basierend auf Amazon (AWS)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	9,19	12,44	11,10	3,20	12,46								
128	17,82	19,85	21,94	21,96	22,16	14,01							
256	12,40	32,49	34,44	14,39	32,25	34,63	30,89						
512	31,87	47,01	49,11	46,22	49,64	45,84	48,13	23,41					
1024	30,88	62,37	48,09	38,55	21,43	53,50	18,76	14,60	59,47				
2048	36,83	48,51	64,55	48,17	53,87	62,59	10,34	54,64	15,72	45,06			
4096	47,99	67,36	65,54	37,14	27,95	55,63	41,08	19,62	19,56	26,55	77,47		
8192	28,63	51,46	25,96	35,06	50,19	29,69	33,81	52,21	36,79	84,92	79,52	46,14	
16384	20,15	56,10	59,65	39,01	35,71	52,33	81,23	33,40	34,07	31,77	36,48	48,15	55,04
32768					36,69	32,94	40,42	47,70	61,54	37,24	40,27	32,68	38,74
65536					34,64	37,15	49,43	44,63	34,62	32,89	40,60	36,17	34,42
131072					44,06	44,64	42,97	45,67	47,00	66,46	46,44	56,42	46,09
262144					52,46	45,84	48,51	56,30	49,36	46,94	48,36	45,56	49,99
524288					51,13	54,75	46,03	48,44	49,37	49,40	49,25	50,01	44,58
1048576					56,86	53,67	52,25	51,22	53,26	51,96	50,13	48,26	46,26
2097152					46,67	45,20	25,83	11,19	9,29	8,81	8,80	8,80	8,80

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.13: Schreibgeschwindigkeit AWS mit zwei GlusterFS Nodes und Einkern-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	3,64	2,99	4,57	4,60	4,33								
128	8,49	11,05	9,87	10,29	11,99	11,71							
256	8,09	16,93	16,13	18,23	18,36	17,61	15,51						
512	16,14	22,48	24,28	25,14	28,10	21,99	28,24	20,87					
1024	15,91	22,52	29,05	38,34	39,17	29,55	38,46	37,24	36,79				
2048	18,45	25,66	38,98	30,38	45,39	44,44	45,87	46,18	37,14	30,64			
4096	16,27	26,97	45,93	50,68	49,85	52,51	51,52	45,41	52,08	51,59	51,61		
8192	27,64	41,10	41,84	52,31	52,45	49,83	53,86	55,41	55,11	54,94	54,49	54,80	
16384	18,07	40,72	51,53	53,15	57,60	58,02	57,25	57,26	57,99	57,20	56,91	57,50	57,93
32768					57,72	58,65	59,20	58,05	59,13	59,50	59,48	55,27	58,94
65536					58,95	59,87	57,86	60,11	60,11	60,07	59,44	58,24	59,96
131072					59,75	60,75	60,84	60,93	60,85	60,79	60,76	59,62	60,64
262144					59,95	60,89	60,76	60,86	59,99	61,09	61,00	61,04	60,87
524288					62,46	71,39	71,77	70,54	72,43	71,22	71,63	70,11	71,98
1048576					82,83	81,13	76,13	86,93	82,34	86,34	81,62	87,45	89,43
2097152					83,11	80,67	84,73	79,13	72,19	80,06	84,51	77,18	85,89

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.11: Schreibgeschwindigkeit OpenStack mit sechs GlusterFS Nodes und Zwei-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	12,47	11,09	12,13	13,97	14,65								
128	23,13	22,50	21,08	26,40	30,61	29,80							
256	45,65	41,44	50,39	44,21	14,59	56,80	62,24						
512	63,97	74,47	72,50	80,98	67,79	91,01	105,11	112,72					
1024	19,37	91,62	97,38	88,10	98,99	114,74	94,61	136,21	192,45				
2048	113,31	70,75	81,25	113,16	90,56	119,49	69,85	127,39	135,02	394,76			
4096	47,32	95,99	71,44	102,68	113,22	86,81	100,77	81,68	91,42	255,87	637,41		
8192	116,26	40,48	112,27	124,79	112,77	157,10	120,89	161,74	187,64	265,28	317,87	1029,14	
16384	126,40	93,59	117,21	111,47	88,15	117,09	104,93	152,76	109,98	238,91	320,93	420,38	1530,20
32768					120,80	107,62	145,39	117,29	148,60	236,00	250,27	303,72	458,97
65536					139,38	119,09	109,84	126,35	179,15	239,74	179,69	301,33	301,55
131072					130,33	114,79	114,39	133,08	171,37	238,25	258,53	252,94	318,18
262144					130,42	110,31	126,40	143,64	215,16	229,12	225,84	266,11	294,39
524288					137,74	134,86	128,47	150,49	200,95	209,54	249,42	279,90	277,09
1048576					132,29	133,70	130,21	143,69	194,32	231,25	264,42	273,43	278,05
2097152					108,61	109,68	116,92	127,73	147,23	156,98	166,46	166,62	177,74

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.12: Lesegeschwindigkeit OpenStack mit sechs GlusterFS Nodes und Zwei-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	31,36	32,85	32,36	39,58	40,32								
128	54,63	60,84	70,29	65,21	82,16	79,69							
256	105,93	109,40	110,82	115,21	57,68	112,58	47,64						
512	137,37	146,87	103,56	158,46	152,98	152,20	199,38	285,73					
1024	97,40	163,39	152,27	173,58	13,22	207,38	238,52	79,34	589,17				
2048	197,11	204,80	205,21	30,76	25,90	32,03	65,26	51,95	100,23	1122,31			
4096	208,68	204,19	220,70	210,34	214,15	216,37	29,69	60,02	382,09	475,18	1162,31		
8192	216,38	24,92	227,01	222,34	19,53	20,00	23,78	250,76	330,80	437,82	383,61	1865,17	
16384	20,25	223,55	17,26	226,78	217,78	213,43	219,56	261,77	329,14	400,36	449,46	632,68	819,20
32768					227,49	238,20	22,20	259,80	40,99	396,21	446,52	500,91	662,27
65536					236,12	237,26	234,58	267,59	343,36	405,39	447,26	366,14	538,17
131072					236,54	237,98	239,57	277,54	344,83	402,13	434,70	452,87	512,72
262144					237,57	16,79	17,64	18,47	360,41	396,66	437,40	455,79	465,87
524288					18,64	19,85	33,36	32,44	31,32	35,59	36,61	38,02	38,35
1048576					36,73	32,88	18,85	27,11	30,09	35,14	37,75	38,59	38,86
2097152					33,70	22,24	25,16	46,69	64,09	64,43	64,46	64,48	64,56

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.14: Lesegeschwindigkeit AWS mit zwei GlusterFS Node und Einkern-Prozessorens

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	1,49	10,41	10,68	9,34	6,40								
128	11,53	14,33	5,46	10,74	9,90	18,00							
256	24,48	15,96	12,51	11,40	7,68	13,10	9,73						
512	19,48	25,86	32,63	20,59	31,69	8,76	11,54	9,38					
1024	18,64	24,57	29,86	32,93	34,16	15,64	12,96	9,62	14,28				
2048	19,70	41,47	23,65	15,16	19,00	19,72	17,27	17,71	13,39	12,89			
4096	24,46	50,29	26,63	40,58	25,43	22,51	25,70	24,83	26,11	19,82	20,89		
8192	29,83	35,31	28,54	32,08	25,03	27,05	22,65	27,06	23,72	25,58	24,28	25,11	
16384	22,06	27,64	29,98	26,07	26,70	41,67	26,40	30,14	30,99	29,96	29,31	26,02	28,76
32768					34,41	28,78	26,33	30,60	30,52	30,79	29,64	30,14	35,46
65536					36,35	34,63	33,65	31,80	32,20	33,06	37,98	35,62	32,27
131072					36,49	39,60	39,42	36,90	39,34	38,21	35,14	37,94	34,98
262144					36,78	38,19	38,42	38,68	37,28	36,64	37,26	36,95	36,27
524288					37,08	37,72	38,11	36,09	37,78	37,41	35,80	34,96	30,33
1048576					4,72	4,48	4,44	4,54	4,47	4,39	4,32	4,25	4,27
2097152					4,07	4,41	4,46	4,50	4,42	4,27	4,24	4,20	4,27

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.15: Schreibgeschwindigkeit AWS mit vier GlusterFS Nodes und Einkern-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	16,42	11,92	18,20	30,71	25,33								
128	24,48	17,25	21,91	27,85	60,75	58,31							
256	42,98	40,88	45,54	22,43	53,77	53,04	107,97						
512	27,11	13,93	61,35	58,59	67,61	53,79	90,49	257,28					
1024	35,71	29,95	23,13	55,20	38,32	18,34	20,20	105,76	440,07				
2048	45,87	12,32	29,59	35,76	46,04	26,18	52,72	18,71	121,28	741,23			
4096	34,38	45,61	19,67	22,73	32,80	35,72	48,76	14,23	87,86	51,14	1471,79		
8192	31,16	44,88	39,14	32,24	47,10	30,06	31,62	43,54	80,69	119,05	156,00	593,11	
16384	46,08	45,38	41,87	32,19	27,75	35,49	31,93	37,28	70,47	95,39	87,04	193,26	671,92
32768					36,55	39,37	38,79	51,47	73,34	59,40	52,51	75,20	76,51
65536					42,70	38,14	38,53	35,45	64,81	75,07	55,27	122,35	53,85
131072					33,05	36,06	34,96	37,60	59,95	74,15	60,37	62,66	69,96
262144					29,64	35,27	33,65	37,52	48,76	61,82	70,72	59,21	63,79
524288					31,90	31,96	29,50	34,07	43,10	52,54	62,12	63,69	53,83
1048576					27,56	29,16	25,48	28,06	31,81	36,72	42,74	41,37	40,43
2097152					30,01	27,86	26,51	29,93	40,09	43,64	47,80	40,52	44,79

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.16: Lesegeschwindigkeit AWS mit vier GlusterFS Nodes und Einkern-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	6,41	7,77	12,92	12,70	13,52								
128	7,15	20,89	20,75	22,55	23,57	19,99							
256	30,74	18,18	25,14	34,48	34,51	12,97	36,26						
512	35,67	12,97	42,35	45,86	47,41	48,58	49,56	49,37					
1024	31,27	64,08	64,26	60,04	57,05	37,12	53,78	66,15	18,49				
2048	20,26	55,25	61,43	66,87	67,84	73,20	22,68	17,80	12,05	72,22			
4096	29,17	24,02	45,36	70,27	28,16	34,85	66,06	21,06	82,92	23,97	23,24		
8192	23,93	26,14	47,62	50,70	41,00	24,51	52,24	34,68	23,63	61,25	28,14	41,10	
16384	22,09	38,04	39,00	33,66	88,68	47,93	40,71	47,30	44,24	41,29	40,56	40,52	41,24
32768					52,06	34,56	36,30	39,15	55,56	36,22	58,65	38,78	46,77
65536					36,22	91,17	92,48	54,40	34,76	45,64	47,05	86,98	76,80
131072					79,75	72,26	79,06	79,34	47,75	49,15	43,85	51,36	50,07
262144					60,65	57,20	64,54	58,80	51,18	56,30	57,92	56,71	66,06
524288					58,66	55,50	61,86	62,22	65,70	57,26	63,03	57,49	59,60
1048576					59,58	59,87	49,98	62,21	51,35	62,66	50,02	46,15	45,32
2097152					44,77	43,61	39,50	13,25	10,30	10,35	8,80	9,66	9,74

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.17: Schreibgeschwindigkeit AWS mit sechs GlusterFS Nodes und Einkern-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	35,16	32,13	31,85	42,16	39,73								
128	60,46	54,58	70,72	65,27	16,46	77,76							
256	112,78	23,04	120,24	53,13	124,75	142,07	157,72						
512	154,88	12,36	39,94	172,98	159,95	185,71	241,84	330,75					
1024	155,95	175,92	198,14	182,92	183,02	196,69	232,62	265,90	618,35				
2048	78,44	34,13	224,14	203,60	216,54	229,24	227,60	76,97	119,89	1199,91			
4096	71,94	225,62	215,88	218,82	23,31	52,39	15,72	17,76	64,13	54,04	230,75		
8192	17,96	19,04	18,26	225,66	21,20	235,71	237,53	188,38	34,47	55,27	548,51	1174,14	
16384	16,17	16,79	17,93	224,80	229,29	231,26	226,90	235,03	304,21	392,23	430,37	602,40	1170,71
32768					224,21	236,39	240,01	257,44	355,37	420,12	457,98	476,75	812,90
65536					251,80	61,36	248,20	269,31	369,91	40,19	441,80	471,53	417,73
131072					254,87	246,97	245,20	284,97	369,81	433,18	459,67	455,87	524,76
262144					17,46	18,00	17,78	17,82	51,86	432,71	43,12	469,07	500,28
524288					26,66	19,06	32,29	34,31	30,99	33,68	37,30	38,96	39,72
1048576					32,14	19,39	27,35	27,83	33,53	33,41	36,31	38,81	41,34
2097152					21,44	21,80	18,63	31,24	30,73	64,50	39,02	37,90	40,31

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.18: Lesegeschwindigkeit AWS mit sechs GlusterFS Nodes und Einkern-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	9,91	13,32	6,60	6,59	13,78								
128	16,31	21,13	21,48	13,62	23,31	8,96							
256	26,92	34,28	36,62	37,50	37,87	19,69	17,47						
512	26,62	42,63	51,21	51,70	52,80	27,94	52,04	50,51					
1024	31,12	49,35	64,49	36,34	43,13	37,74	48,79	61,28	64,21				
2048	22,63	57,77	62,61	29,65	40,57	56,29	76,84	63,02	39,13	57,56			
4096	29,79	44,72	32,48	76,11	33,82	59,82	65,04	76,37	64,52	50,19	36,94		
8192	27,72	54,38	41,94	59,88	52,19	28,02	39,36	47,29	30,27	49,79	37,43	47,26	
16384	30,96	55,06	90,88	76,18	68,03	85,86	54,13	39,52	40,73	41,36	38,63	35,50	48,04
32768					43,08	52,76	49,98	48,40	40,60	31,62	39,65	42,73	43,55
65536					48,44	37,96	44,12	37,45	50,79	44,86	40,87	47,70	50,29
131072					42,35	47,67	45,94	48,80	43,14	43,64	40,71	45,26	40,83
262144					41,52	42,87	38,45	40,92	36,39	44,20	41,75	41,63	41,81
524288					41,56	50,19	40,73	38,96	41,93	37,14	39,50	34,42	39,01
1048576					47,35	42,06	43,53	45,56	44,83	46,71	43,91	43,96	44,57
2097152					51,07	55,88	60,68	59,87	64,60	59,77	58,27	54,26	61,14

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.19: Schreibgeschwindigkeit AWS mit zwei GlusterFS Nodes und Zweikern-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	30,76	33,49	32,69	43,54	38,62								
128	60,60	58,34	18,73	69,98	73,69	81,42							
256	108,99	108,02	118,96	30,65	114,12	169,20	167,33						
512	39,19	44,70	70,45	51,59	53,74	54,11	264,74	323,04					
1024	115,86	91,48	231,15	110,45	66,72	279,41	295,70	415,78	561,15				
2048	250,37	85,81	22,26	183,92	279,90	207,60	168,63	161,63	209,05	1096,94			
4096	38,02	65,58	181,09	289,80	132,26	126,72	104,17	86,50	535,08	771,51	1869,45		
8192	244,71	198,91	64,11	113,78	131,35	86,52	124,67	216,51	220,93	653,01	270,45	2699,16	
16384	128,98	98,00	71,60	90,08	86,86	79,60	82,47	256,74	303,13	239,24	548,34	276,63	2625,20
32768					86,04	132,15	101,40	137,35	283,68	222,65	248,38	323,80	302,46
65536					124,56	164,06	128,07	224,22	176,98	263,57	387,04	510,90	499,76
131072					155,06	158,03	172,86	181,78	228,65	272,82	294,11	320,57	469,26
262144					166,98	183,91	166,51	183,97	245,36	321,28	265,07	353,37	412,48
524288					157,63	156,42	150,13	202,78	226,99	292,62	300,60	380,26	434,50
1048576					132,55	153,46	159,27	171,37	258,64	348,68	344,38	473,41	463,66
2097152					144,65	147,07	155,97	194,48	248,07	280,12	323,50	533,83	544,06

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.20: Lesegeschwindigkeit AWS mit zwei GlusterFS Nodes und Zweikern-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	10,16	10,76	11,01	12,10	11,42								
128	14,56	18,69	9,67	20,42	20,70	19,96							
256	21,07	16,51	32,63	34,32	32,12	32,43	33,20						
512	18,15	38,15	24,52	47,92	46,93	36,31	46,26	36,91					
1024	26,64	42,82	56,34	61,21	62,95	48,12	59,28	59,64	47,94				
2048	28,25	49,70	70,74	44,99	65,80	54,27	70,98	58,20	69,82	62,93			
4096	27,09	45,02	56,47	37,18	46,80	36,46	77,12	35,07	60,42	48,75	76,04		
8192	27,80	48,10	44,01	24,39	44,85	36,58	38,96	37,91	42,78	46,72	52,00	35,80	
16384	28,05	47,70	77,87	78,31	61,81	75,33	37,57	35,45	80,49	42,18	78,22	82,80	36,00
32768					73,42	55,66	44,97	44,57	42,54	47,56	56,47	80,72	49,96
65536					43,25	38,98	40,42	44,08	40,93	41,03	35,53	33,84	42,70
131072					37,57	38,42	36,18	39,03	44,26	35,88	37,58	35,73	36,83
262144					38,16	38,69	40,38	35,28	37,36	37,52	39,69	35,77	34,56
524288					36,88	44,02	40,53	32,68	43,26	36,01	38,94	34,24	36,30
1048576					47,11	38,80	40,69	42,51	44,96	56,42	39,34	37,69	44,25
2097152					64,19	58,62	53,49	69,47	58,29	56,53	64,43	58,71	56,48

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.21: Schreibgeschwindigkeit AWS mit vier GlusterFS Nodes und Zweikern-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	29,55	29,43	28,30	34,63	35,40								
128	53,51	55,97	56,14	61,63	68,16	67,47							
256	99,41	96,75	97,97	108,01	109,97	143,67	140,66						
512	139,39	146,87	61,17	176,79	176,79	186,72	23,53	280,69					
1024	129,52	199,33	218,34	214,86	71,04	261,29	253,27	114,34	523,21				
2048	215,51	29,24	246,56	29,61	274,09	164,33	261,86	396,44	589,69	962,48			
4096	142,77	255,49	267,29	172,36	294,80	43,66	154,62	193,30	542,09	505,81	1552,66		
8192	50,23	73,88	90,08	195,35	97,66	117,99	73,01	128,84	356,73	664,13	425,65	2303,12	
16384	96,33	94,54	97,24	83,97	99,46	62,33	165,17	133,14	130,77	256,58	525,21	1009,56	2570,51
32768					131,13	152,86	101,64	177,64	192,34	158,78	354,12	795,11	861,59
65536					144,13	118,33	122,89	142,26	204,86	277,87	299,18	429,14	264,16
131072					141,56	127,45	148,89	220,40	287,56	300,67	325,72	243,20	505,81
262144					167,40	155,36	185,42	208,68	261,15	272,27	345,47	499,14	461,11
524288					120,68	152,42	167,03	172,07	222,95	294,39	349,48	407,43	388,19
1048576					142,50	168,01	175,04	192,25	274,66	281,84	343,78	425,31	449,29
2097152					148,40	156,68	159,28	198,54	287,40	327,14	295,81	467,23	562,84

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.22: Lesegeschwindigkeit AWS mit vier GlusterFS Nodes und Zweikern-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	5,98	9,34	3,84	11,28	12,30								
128	14,69	19,55	19,45	20,91	12,69	16,93							
256	20,84	26,40	31,45	14,07	29,98	31,41	23,56						
512	27,28	24,55	36,24	39,89	45,62	44,03	43,44	35,80					
1024	28,64	47,47	38,04	39,99	52,08	59,66	42,89	35,54	54,05				
2048	29,08	36,38	67,77	70,49	57,00	50,41	67,06	58,29	54,45	67,56			
4096	29,07	54,53	30,81	61,96	63,71	31,37	76,68	70,97	19,51	35,16	73,36		
8192	29,16	56,37	77,46	79,81	39,00	80,20	52,63	76,56	74,61	81,04	35,86	79,68	
16384	31,82	54,66	58,63	79,54	79,01	80,57	80,75	81,91	82,27	76,29	72,58	79,19	57,59
32768					79,00	68,96	65,16	68,63	78,49	63,39	80,59	79,01	48,59
65536					71,69	70,99	64,33	62,81	58,74	68,05	63,52	69,61	67,61
131072					60,47	78,25	67,87	64,21	76,98	69,58	63,79	59,04	72,45
262144					59,70	65,07	81,24	81,13	70,89	69,62	60,77	53,69	73,89
524288					59,97	63,10	60,82	63,39	55,54	60,57	68,46	52,62	60,63
1048576					62,85	63,50	59,10	62,20	60,96	60,29	60,64	71,41	65,33
2097152					81,58	88,85	88,17	83,79	88,60	87,73	90,09	89,90	89,00

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.23: Schreibgeschwindigkeit AWS mit sechs GlusterFS Nodes und Zweikern-Prozessoren

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	24,80	26,72	25,80	32,06	31,42								
128	12,61	47,76	51,45	51,61	61,21	61,19							
256	81,17	87,67	95,98	97,83	97,63	115,15	119,40						
512	72,22	135,92	142,98	36,63	150,64	54,42	59,90	245,68					
1024	85,51	183,77	107,12	211,35	212,76	223,54	94,31	319,81	449,13				
2048	239,50	22,39	120,49	247,79	136,20	262,13	287,36	352,01	548,92	895,46			
4096	116,22	82,99	166,57	260,82	108,45	115,46	103,93	217,19	478,00	701,48	1408,09		
8192	62,76	68,55	62,44	155,29	178,42	191,00	77,48	229,04	116,80	120,12	882,86	2063,42	
16384	104,10	113,28	119,74	200,49	189,54	128,73	133,53	246,59	474,26	577,35	474,43	911,18	2464,49
32768					68,23	126,58	171,16	125,96	287,55	327,81	490,71	289,33	873,49
65536					136,62	172,43	135,84	213,78	246,88	227,76	309,39	390,39	678,86
131072					182,81	158,51	120,05	157,74	277,08	219,45	432,56	348,41	533,87
262144					112,06	158,42	154,66	204,52	248,15	285,44	356,53	415,18	477,76
524288					138,97	164,65	147,63	220,38	248,38	305,00	359,30	459,65	384,90
1048576					130,35	154,10	160,77	158,89	238,66	299,60	341,72	403,80	444,99
2097152					150,39	155,38	160,49	175,41	261,68	265,37	310,05	430,47	443,98

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.24: Lesegeschwindigkeit AWS mit sechs GlusterFS Nodes und Zweikern-Prozessoren

E.2.3 Plattformübergreifendes Storage-Cluster mit Amazon (AWS) und OpenStack mit jeweils einem GlusterFS Node

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	0,37	0,52	0,51	0,52	0,52								
128	0,99	1,01	0,95	0,92	0,96	0,93							
256	1,87	1,85	1,85	1,85	1,90	1,83	1,85						
512	3,52	3,59	3,71	3,72	3,65	3,68	3,68	3,71					
1024	6,35	6,36	6,67	6,70	6,63	6,63	6,64	6,18	6,57				
2048	9,66	10,84	10,86	10,29	10,70	9,91	9,50	10,79	10,72	10,42			
4096	13,55	16,05	15,69	15,05	16,08	16,23	16,11	16,10	16,18	15,78	15,80		
8192	14,52	19,66	20,35	19,74	20,27	20,42	19,62	19,71	20,51	19,14	19,80	19,90	
16384	16,46	22,44	23,10	23,07	20,15	22,39	22,68	21,83	23,29	23,30	22,56	22,76	21,01
32768					23,98	24,21	24,29	24,43	24,50	21,08	22,17	21,55	24,41
65536					22,38	26,14	26,13	25,97	26,04	25,81	26,70	26,32	25,07
131072					27,87	28,80	27,73	25,94	26,68	26,01	26,63	29,33	29,01
262144					32,34	26,13	30,72	30,59	32,67	27,61	31,04	31,63	31,64
524288					28,82	31,30	30,53	31,84	32,56	32,47	29,85	32,10	31,01
1048576					31,19	32,22	31,86	30,50	30,50	29,05	32,71	29,38	18,31
2097152					19,45	18,83	31,60	15,57	10,45	8,53	8,32	8,73	8,02

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.25: Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils einem GlusterFS Node und Einkern-Prozessoren (Messung auf der OpenStack Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	0,18	0,24	0,25	0,25	0,25								
128	0,43	0,51	0,51	0,51	0,51	0,50							
256	0,99	1,00	1,01	1,00	1,00	1,00	1,00						
512	1,91	1,93	1,84	1,95	1,93	1,95	1,95	1,94					
1024	3,52	3,57	3,60	3,58	3,58	3,60	3,62	3,59	3,60				
2048	4,59	5,62	5,58	5,75	5,62	5,35	4,47	5,63	5,66	4,80			
4096	7,66	7,38	8,04	8,31	8,32	8,34	8,40	8,37	8,37	8,34	8,38		
8192	7,62	10,58	10,75	10,90	10,93	10,99	10,78	10,84	11,00	10,93	10,97	10,89	
16384	12,87	13,00	13,12	13,12	13,10	13,15	13,11	13,10	12,93	13,04	13,15	13,18	13,14
32768					14,45	14,64	14,73	14,55	14,69	14,39	14,67	14,70	14,70
65536					15,49	15,39	15,29	15,37	15,40	15,57	15,59	15,42	15,25
131072					18,63	18,71	19,21	18,59	18,71	18,77	19,39	18,53	18,73
262144					19,90	19,24	19,58	19,17	19,38	17,86	18,05	19,12	19,02
524288					20,57	20,23	19,50	19,94	19,13	20,73	20,65	16,49	20,04
1048576					21,05	20,39	19,42	17,97	15,48	15,83	16,46	19,13	14,19
2097152					15,32	12,58	11,26	9,42	4,98	4,22	4,22	4,32	4,22

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.26: Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils einem GlusterFS Node und Einkern-Prozessoren (Messung auf der AWS Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	0,35	0,49	0,52	0,49	0,53								
128	1,01	1,03	1,03	1,04	1,04	1,00							
256	1,99	1,85	2,03	2,04	1,99	2,07	1,98						
512	3,65	3,87	3,97	3,99	3,91	3,81	3,87	4,01					
1024	6,91	7,17	7,56	7,47	7,51	4,74	7,58	6,87	4,93				
2048	11,14	11,62	12,27	8,64	12,40	12,46	12,45	12,02	12,24	11,92			
4096	14,53	16,20	18,13	17,44	17,78	16,72	18,35	18,57	18,18	18,51	14,03		
8192	17,49	21,74	21,82	22,68	23,10	23,21	22,52	22,39	22,68	23,44	22,35	22,38	
16384	20,07	25,31	24,82	25,73	24,00	26,65	26,95	27,29	27,14	27,29	27,25	27,16	27,14
32768					29,04	29,17	29,48	29,31	29,39	28,78	29,56	29,47	29,75
65536					30,06	30,28	29,54	29,89	30,24	30,07	30,42	29,92	30,93
131072					30,90	30,79	30,98	30,70	30,67	30,55	31,13	31,03	30,95
262144					31,60	31,76	32,33	31,84	31,78	31,83	32,14	31,98	31,89
524288					31,90	32,28	32,24	32,26	32,24	32,04	32,11	32,15	32,18
1048576					37,38	32,11	35,74	37,35	38,03	37,32	35,60	35,77	35,27
2097152					35,71	36,62	36,10	36,26	36,14	37,10	35,95	36,64	37,34

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.27: Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils einem GlusterFS Node und Zweikern-Prozessoren (Messung auf der OpenStack Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	0,36	0,51	0,52	0,48	0,52								
128	1,00	1,03	1,04	1,03	1,02	1,05							
256	1,99	0,55	1,51	1,31	2,03	1,94	1,95						
512	3,02	3,33	3,36	3,36	3,09	2,94	2,91	2,91					
1024	4,07	4,03	4,09	4,12	4,16	4,50	4,49	4,37	4,50				
2048	5,78	5,85	6,19	6,18	5,13	5,21	5,41	5,49	5,80	5,79			
4096	7,03	7,37	7,62	8,00	6,54	7,02	7,31	7,41	7,73	6,28	6,58		
8192	7,78	8,54	9,12	7,74	8,58	7,10	7,60	6,76	7,56	8,22	6,99	7,79	
16384	9,31	10,62	11,77	10,03	8,85	8,07	7,67	9,12	8,36	9,71	10,97	9,53	8,42
32768					10,94	12,03	12,57	10,86	13,18	11,89	13,67	12,30	14,01
65536					16,59	14,92	10,56	13,37	16,73	12,07	12,71	13,21	16,08
131072					10,06	11,93	12,45	13,30	17,29	13,92	15,37	16,98	12,70
262144					15,45	15,86	19,35	15,58	14,67	17,45	19,40	19,05	16,90
524288					18,98	18,21	19,62	16,77	16,99	17,74	16,31	16,60	15,25
1048576					18,75	15,03	10,50	10,07	10,49	10,50	10,25	10,26	10,24
2097152					10,21	10,11	10,24	10,26	9,93	10,33	10,41	10,31	10,45

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.28: Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils einem GlusterFS Node und Zweikern-Prozessoren (Messung auf der AWS Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	2,57	2,59	2,58	2,65	2,64								
128	5,03	5,05	5,13	5,13	5,31	5,29							
256	9,93	9,90	9,88	10,11	10,03	10,57	10,55						
512	18,04	18,16	19,29	19,35	19,47	19,20	19,56	20,85					
1024	17,88	30,72	20,34	17,27	18,56	32,99	32,97	36,51	41,88				
2048	46,26	47,12	48,97	46,42	17,93	22,71	23,45	26,25	39,41	72,52			
4096	23,92	61,47	18,32	65,87	66,54	66,62	23,06	75,17	33,83	113,00	91,91		
8192	19,48	80,25	19,41	81,20	81,76	82,36	20,06	93,75	115,92	37,25	52,75	296,81	
16384	29,72	89,75	91,90	91,74	92,29	92,65	19,81	106,15	134,04	153,80	176,41	245,15	342,82
32768					94,58	91,92	101,95	109,50	138,26	172,64	193,21	219,70	297,20
65536					101,25	108,09	104,24	120,00	138,96	181,59	192,02	221,16	242,82
131072					33,24	133,32	119,71	123,17	159,07	171,29	207,11	216,78	234,19
262144					25,02	41,16	23,11	84,21	62,10	52,29	51,60	53,08	198,22
524288					20,21	42,54	31,14	39,00	76,31	87,98	94,16	86,08	102,03
1048576					35,67	34,35	19,61	35,64	72,64	91,04	91,48	99,68	108,91
2097152					33,39	24,72	27,87	56,09	86,51	82,00	88,40	95,10	89,13

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.29: Lesegeschwindigkeit zwischen AWS und OpenStack mit jeweils einem GlusterFS Node und Einkern-Prozessoren (Messung auf der OpenStack Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	2,52	2,53	2,51	2,56	2,56								
128	5,00	5,01	5,02	5,02	5,09	5,09							
256	9,90	9,84	9,92	10,02	9,92	10,15	10,25						
512	18,97	18,90	19,06	19,29	18,96	19,40	19,87	20,62					
1024	34,42	34,54	34,80	35,11	35,08	35,44	36,28	38,04	40,61				
2048	61,23	60,64	61,02	60,22	32,99	42,04	64,38	46,88	46,53	71,19			
4096	33,71	41,49	95,33	94,02	96,56	96,62	98,87	104,19	115,64	132,21	147,36		
8192	18,08	134,67	132,75	132,15	135,38	134,22	137,21	147,65	169,37	187,81	212,16	271,26	
16384	169,39	165,95	168,92	168,41	169,87	168,95	169,59	185,01	222,50	253,92	288,00	317,91	479,39
32768					200,11	199,62	202,01	220,80	274,99	306,26	333,09	369,72	499,94
65536					211,93	216,82	198,60	242,17	299,45	341,65	377,50	391,46	443,57
131072					223,56	225,37	231,17	260,26	318,81	378,12	399,56	422,21	447,67
262144					17,93	231,31	230,67	265,43	325,29	376,28	411,21	427,36	428,93
524288					19,58	19,04	19,42	28,27	41,91	46,33	55,40	70,98	51,03
1048576					22,53	19,49	30,73	28,29	39,64	45,75	51,03	52,46	53,34
2097152					25,19	23,45	25,86	31,73	64,36	64,39	54,86	64,44	64,52

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.30: Lesegeschwindigkeit zwischen AWS und OpenStack mit jeweils einem GlusterFS Node und Einkern-Prozessoren (Messung auf der AWS Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	2,43	2,60	2,53	2,66	2,64								
128	4,76	5,10	5,16	5,03	5,33	5,28							
256	9,61	9,45	9,81	9,98	10,13	9,36	10,78						
512	15,11	14,58	18,49	18,68	18,54	18,35	20,15	20,74					
1024	27,54	29,69	32,07	26,55	34,31	31,97	33,84	38,52	41,91				
2048	39,89	34,77	40,52	54,09	50,72	32,16	58,39	56,43	66,22	81,82			
4096	58,92	63,31	67,61	60,99	67,48	67,24	67,69	69,00	59,80	110,80	154,91		
8192	83,13	79,20	84,64	88,72	60,40	82,99	91,41	99,75	105,60	103,15	175,43	291,93	
16384	69,04	95,15	90,40	102,77	78,52	79,85	78,90	90,76	135,74	150,20	137,36	242,74	436,65
32768					85,29	101,37	106,28	98,46	96,40	132,05	207,89	215,30	297,50
65536					91,58	88,76	100,20	111,35	141,23	162,47	213,96	252,00	296,04
131072					96,12	93,63	108,32	105,30	148,71	160,05	200,91	214,10	224,52
262144					99,42	103,21	105,68	151,67	157,92	179,87	200,40	205,35	219,29
524288					102,26	104,09	109,36	118,47	138,69	170,36	202,48	213,52	227,07
1048576					106,34	108,08	108,09	123,33	145,65	167,15	171,59	195,08	199,24
2097152					96,28	95,72	91,95	109,56	132,55	135,44	169,21	178,06	172,64

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.31: Lesegeschwindigkeit zwischen AWS und OpenStack mit jeweils einem GlusterFS Node und Zweikern-Prozessoren (Messung auf der OpenStack Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	2,55	2,51	2,56	2,57	2,60								
128	4,94	5,11	5,13	5,12	5,15	5,15							
256	10,09	10,18	10,11	8,11	8,74	10,50	10,32						
512	19,33	19,10	19,60	19,61	19,80	19,83	7,06	20,99					
1024	35,34	24,73	37,28	24,76	26,87	37,84	38,17	39,31	41,45				
2048	26,50	43,51	65,85	15,36	34,63	43,20	69,89	73,58	30,75	83,02			
4096	80,88	33,86	54,48	50,17	100,42	31,29	92,03	124,05	129,64	144,25	161,53		
8192	71,00	88,13	125,24	67,44	58,81	60,01	87,37	158,89	152,92	230,56	221,28	307,77	
16384	78,19	114,24	123,64	79,17	93,04	84,51	62,28	203,82	255,92	325,51	265,26	299,17	574,83
32768					87,77	152,49	158,74	192,40	215,08	347,90	431,14	386,84	558,09
65536					155,86	156,46	156,77	248,22	207,35	294,94	275,72	522,92	318,79
131072					159,22	134,03	104,90	201,90	202,27	262,77	294,28	292,56	454,94
262144					126,88	179,46	175,05	210,05	250,10	350,05	340,25	437,96	526,89
524288					116,84	145,65	157,45	205,91	254,27	339,91	417,05	401,19	402,18
1048576					143,21	162,16	147,09	161,71	264,01	352,36	420,93	474,99	549,70
2097152					132,22	151,73	154,02	171,92	284,20	315,19	338,07	455,08	512,88

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.32: Lesegeschwindigkeit zwischen AWS und OpenStack mit jeweils einem GlusterFS Node und Zweikern-Prozessoren (Messung auf der AWS Plattform)

E.2.4 Plattformübergreifendes Storage-Cluster mit Amazon (AWS) und OpenStack mit jeweils zwei GlusterFS Nodes

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	1,87	3,06	4,49	4,80	5,20								
128	3,71	5,47	10,23	11,39	11,56	4,98							
256	7,27	14,37	17,78	19,94	19,76	18,92	15,59						
512	10,15	19,95	25,02	27,54	26,49	29,34	12,25	8,72					
1024	10,76	14,19	32,74	35,98	37,84	38,37	20,18	15,31	13,65				
2048	11,24	24,61	36,86	40,53	35,18	44,03	42,04	43,68	43,90	44,28			
4096	13,47	16,76	15,65	17,09	22,63	27,59	31,09	17,51	27,69	29,62	41,63		
8192	11,69	22,50	33,03	42,35	30,56	39,51	21,79	23,52	50,21	55,53	53,05	52,90	
16384	14,72	19,95	28,85	20,95	25,42	47,23	37,95	58,23	54,52	34,96	23,41	56,18	57,80
32768					31,26	28,21	52,82	57,85	57,53	59,26	28,64	46,76	56,07
65536					44,10	55,92	38,23	38,50	58,08	60,55	57,11	60,53	56,67
131072					46,64	59,42	71,55	74,57	38,62	55,78	39,55	58,47	59,31
262144					46,89	48,50	57,29	53,50	63,72	52,15	49,19	59,18	64,81
524288					63,40	64,15	53,60	57,06	69,66	64,93	62,21	52,11	64,81
1048576					43,67	67,80	64,23	55,79	64,66	63,21	72,84	62,65	60,53
2097152					57,45	67,81	59,26	60,97	59,46	64,50	63,92	63,78	64,39

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.33: Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils zwei GlusterFS Nodes und Einkern-Prozessoren (Messung auf der OpenStack Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	0,37	0,50	0,51	0,51	0,50								
128	0,88	0,96	0,97	1,01	0,99	1,02							
256	1,63	1,70	1,63	1,70	1,71	1,71	1,71						
512	2,87	2,58	2,60	2,88	2,90	2,90	2,89	2,88					
1024	4,07	4,04	4,11	3,99	4,45	4,47	4,05	4,47	4,49				
2048	5,66	5,84	5,83	6,08	5,84	6,24	6,27	7,13	7,06	7,13			
4096	9,03	8,00	9,10	8,21	9,26	8,38	9,63	8,60	9,15	8,37	8,14		
8192	10,45	10,94	10,93	10,64	9,75	8,98	9,28	10,27	10,64	10,09	11,32	11,08	
16384	13,22	13,87	14,04	12,04	13,53	13,60	13,95	14,44	14,60	15,14	15,37	16,02	16,07
32768					16,67	18,54	19,38	19,81	16,00	16,87	14,23	15,08	16,40
65536					15,40	15,62	16,78	19,12	21,08	21,93	21,85	17,93	17,51
131072					19,18	21,13	21,71	17,54	20,14	21,67	17,55	19,09	19,73
262144					15,88	16,05	13,54	15,42	18,73	17,47	18,03	13,40	15,81
524288					13,63	13,05	12,83	14,32	8,69	5,30	4,75	4,50	4,43
1048576					4,06	13,21	13,15	13,12	6,85	5,60	4,75	4,63	4,43
2097152					4,03	12,96	12,93	12,78	6,95	5,28	4,73	4,51	4,50

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.34: Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils zwei GlusterFS Nodes und Einkern-Prozessoren (Messung auf der AWS Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	3,53	3,91	4,71	4,91	4,54								
128	8,99	10,53	11,89	10,83	11,60	6,50							
256	9,75	19,23	18,49	19,81	19,54	19,26	19,68						
512	21,30	25,32	21,36	28,52	27,70	22,31	28,15	28,26					
1024	27,06	29,76	36,39	28,64	38,73	38,99	39,35	39,26	37,77				
2048	26,26	41,56	44,96	39,60	47,32	42,84	47,50	44,98	46,05	46,51			
4096	24,57	32,35	39,00	53,10	52,98	51,63	49,33	51,98	51,36	51,62	51,41		
8192	24,54	32,89	41,89	56,15	55,24	55,87	52,94	55,01	54,06	55,75	55,39	55,59	
16384	25,48	31,01	58,45	42,53	56,62	56,92	55,38	55,78	57,11	57,39	58,15	57,65	58,25
32768					51,60	58,44	58,37	59,00	59,39	57,65	57,33	58,54	59,34
65536					52,66	59,60	60,26	60,51	60,34	60,07	60,21	55,75	60,60
131072					55,68	57,17	60,92	58,63	59,57	60,58	59,08	61,18	60,65
262144					54,05	59,30	60,42	60,38	60,94	60,88	60,44	59,58	60,48
524288					53,07	60,14	57,88	58,69	60,83	60,12	59,96	60,95	60,22
1048576					53,90	67,25	69,01	69,24	69,08	69,92	67,77	69,30	69,03
2097152					72,87	82,36	83,39	80,48	82,20	81,63	77,08	82,43	79,63

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.35: Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils zwei GlusterFS Nodes und Zweikern-Prozessoren (Messung auf der OpenStack Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	0,32	0,32	0,33	0,37	0,37								
128	0,58	0,58	0,59	0,64	0,65	0,65							
256	0,96	1,04	0,52	0,95	1,05	0,98	1,16						
512	1,51	1,62	1,75	1,89	1,89	1,90	2,08	2,06					
1024	2,74	2,94	2,97	3,16	3,13	3,16	3,43	3,42	3,45				
2048	4,52	4,51	4,80	4,75	5,08	4,82	4,62	4,53	4,76	4,79			
4096	5,98	5,91	6,21	6,44	6,69	5,92	6,40	6,67	6,64	5,71	5,75		
8192	6,90	7,78	8,35	8,46	7,58	6,51	7,32	7,42	6,94	6,92	6,55	6,62	
16384	8,24	9,86	10,99	9,59	8,60	7,95	9,19	8,50	7,85	7,52	8,93	8,96	7,98
32768					10,32	12,28	9,01	9,14	11,64	13,65	11,34	11,15	10,47
65536					8,59	11,52	11,91	10,92	10,26	9,50	12,86	13,41	12,05
131072					14,25	15,16	15,64	13,45	12,22	10,78	12,92	11,69	11,17
262144					10,93	12,24	12,14	12,13	12,06	10,57	11,81	11,60	12,84
524288					13,09	16,06	10,20	10,66	10,64	10,20	10,27	11,35	10,59
1048576					10,37	10,65	10,53	10,60	10,42	9,68	10,46	10,45	11,07
2097152					10,46	10,65	10,93	10,91	10,78	10,52	10,70	10,42	10,74

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.36: Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils zwei GlusterFS Nodes und Zweikern-Prozessoren (Messung auf der AWS Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	13,02	12,63	13,47	15,83	16,82								
128	24,03	24,11	26,80	27,74	33,32	33,20							
256	40,64	47,77	45,03	48,99	50,52	47,34	57,88						
512	54,21	57,10	60,56	36,00	55,75	62,65	98,44	107,43					
1024	67,03	60,03	80,60	72,72	75,60	74,01	53,98	57,93	211,05				
2048	19,00	82,21	91,91	85,21	89,53	89,62	98,70	118,46	165,94	385,05			
4096	17,39	22,86	21,86	91,84	19,62	23,49	19,22	27,65	36,77	60,56	640,02		
8192	33,74	22,77	97,72	97,00	95,30	23,11	20,95	117,90	152,29	184,47	246,78	1012,61	
16384	20,00	22,29	21,38	23,57	103,26	20,73	104,64	118,76	154,74	34,66	154,62	314,40	815,53
32768					33,18	102,74	106,79	122,48	158,60	187,99	215,52	263,93	358,66
65536					105,37	60,12	21,05	122,91	159,80	188,68	212,10	238,01	284,14
131072					120,89	117,82	123,73	142,21	178,48	166,18	207,93	225,00	223,37
262144					31,26	20,37	21,59	29,49	57,54	31,54	47,76	212,40	271,14
524288					23,38	37,79	25,13	29,77	59,04	76,45	71,14	91,10	99,47
1048576					21,31	35,68	28,64	31,56	77,28	94,38	93,19	95,30	94,62
2097152					24,36	24,99	25,79	31,61	61,74	85,20	85,82	98,59	103,14

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.37: Lesegeschwindigkeit zwischen AWS und OpenStack mit jeweils zwei GlusterFS Nodes und Einkern-Prozessoren (Messung auf der OpenStack Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	0,64	0,64	0,64	1,28	1,27								
128	1,04	1,04	1,27	1,29	2,54	2,57							
256	1,66	1,65	2,06	2,40	2,54	5,13	5,11						
512	2,53	2,55	2,95	3,35	3,38	5,07	6,63	10,32					
1024	3,14	3,20	3,26	3,86	4,09	4,48	5,58	11,32	20,52				
2048	3,85	3,84	3,89	4,06	4,51	4,72	5,30	8,92	17,62	40,93			
4096	4,04	4,00	4,03	4,20	4,67	4,88	4,99	8,30	15,14	25,63	80,23		
8192	4,15	4,26	4,27	4,31	3,83	5,00	4,52	7,19	10,77	14,68	18,23	153,97	
16384	4,41	4,45	4,50	4,48	4,90	4,98	4,73	6,94	10,74	16,84	17,44	16,37	21,00
32768					4,89	5,03	5,08	7,57	11,70	15,76	19,93	22,50	45,44
65536					4,94	4,93	4,77	7,51	11,58	16,73	20,24	20,60	23,88
131072					5,03	5,00	5,10	7,43	12,40	16,93	19,87	22,76	24,60
262144					4,99	5,06	5,03	7,75	11,71	16,97	19,70	23,03	24,53
524288					5,04	5,03	5,04	7,58	11,97	16,45	20,50	22,34	22,38
1048576					4,99	5,01	5,01	7,75	10,58	16,27	18,10	22,18	22,96
2097152					5,02	5,03	5,05	7,60	11,91	16,43	19,96	20,30	24,06

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.38: Lesegeschwindigkeit zwischen AWS und OpenStack mit jeweils zwei GlusterFS Nodes und Einkern-Prozessoren (Messung auf der AWS Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	15,53	15,30	15,41	18,66	17,77								
128	27,41	30,65	30,73	30,50	37,42	38,48							
256	43,84	52,15	50,40	57,98	56,60	69,57	78,87						
512	80,12	86,57	77,14	78,37	78,37	86,14	103,54	124,94					
1024	54,97	94,10	121,52	111,32	122,81	137,03	120,97	167,10	247,15				
2048	123,01	135,04	148,20	134,66	138,76	150,41	162,88	107,98	309,93	456,24			
4096	69,93	101,16	53,55	161,08	155,95	109,70	48,55	124,37	142,83	243,91	476,84		
8192	83,80	96,20	74,64	171,96	115,72	116,13	124,70	130,82	113,88	104,85	310,70	781,76	
16384	142,09	116,38	107,40	154,26	71,38	85,58	177,83	155,42	210,20	362,13	348,44	276,34	1253,76
32768					91,99	106,18	120,00	108,76	157,90	189,40	136,90	280,75	395,71
65536					95,57	104,82	122,87	119,67	120,56	153,12	335,06	259,91	275,54
131072					105,61	107,21	114,50	138,59	186,18	154,07	155,10	223,60	294,49
262144					110,93	127,37	126,02	122,60	146,98	222,46	220,34	194,89	280,49
524288					88,76	103,77	107,86	122,32	116,50	147,81	155,00	163,10	178,63
1048576					101,33	99,26	95,19	115,76	135,15	144,30	167,89	173,37	192,84
2097152					124,19	127,70	115,57	120,59	164,42	177,59	171,37	198,55	207,36

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.39: Lesegeschwindigkeitzwischen AWS und OpenStack mit jeweils zwei GlusterFS Nodes und Zweikern-Prozessoren (Messung auf der OpenStack Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	9,83	9,93	10,14	19,02	19,37								
128	15,95	15,75	19,86	19,47	38,48	38,03							
256	26,04	26,07	32,01	39,41	38,81	75,47	75,57						
512	39,32	39,75	45,86	52,31	53,37	77,21	85,38	154,47					
1024	52,59	57,38	56,93	62,58	63,90	78,30	88,89	149,61	311,93				
2048	63,81	63,20	68,22	68,78	71,52	80,87	83,21	162,59	288,57	599,42			
4096	71,64	71,37	73,41	76,10	72,36	78,65	81,56	155,52	242,90	471,71	1199,90		
8192	74,57	73,73	77,46	75,42	75,02	76,52	82,26	149,21	221,16	365,64	651,45	2353,56	
16384	78,33	78,74	79,01	79,43	80,01	78,84	80,88	149,96	217,49	313,98	477,57	812,60	3130,86
32768					78,17	80,88	79,49	145,23	207,90	311,93	426,72	576,02	928,29
65536					80,69	78,39	80,07	145,77	212,01	306,05	413,46	504,56	637,77
131072					80,01	79,14	81,15	143,87	210,87	306,75	392,54	471,86	566,69
262144					80,40	79,25	79,28	143,51	204,98	298,26	388,41	458,40	520,73
524288					80,09	79,16	79,46	143,57	204,74	299,01	380,55	459,89	499,46
1048576					80,30	80,37	80,57	143,12	206,99	293,82	381,29	429,06	500,18
2097152					79,11	79,65	79,76	144,74	205,31	296,40	382,80	450,87	493,64

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.40: Lesegeschwindigkeitzwischen AWS und OpenStack mit jeweils zwei GlusterFS Nodes und Zweikern-Prozessoren (Messung auf der AWS Plattform)

E.2.5 Plattformübergreifendes Storage-Cluster mit Amazon (AWS) und OpenStack mit jeweils drei GlusterFS Nodes

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	2,20	4,05	3,77	4,23	4,37								
128	7,53	9,62	9,94	9,43	9,91	6,11							
256	11,18	14,54	5,82	16,31	5,19	17,13	16,96						
512	13,29	20,71	22,37	11,67	11,01	10,70	21,91	25,28					
1024	15,25	21,71	23,70	34,13	35,44	35,81	36,35	24,66	12,02				
2048	14,98	24,42	36,59	19,17	17,39	18,18	14,90	20,97	12,89	13,65			
4096	14,52	24,74	39,95	19,47	15,04	20,59	20,73	28,17	20,34	23,82	29,65		
8192	16,13	26,21	37,28	15,33	21,73	24,15	26,37	21,35	54,42	33,25	26,58	42,66	
16384	16,34	28,67	37,97	25,58	16,78	31,89	31,41	34,66	31,34	27,27	25,23	54,44	26,12
32768					25,84	44,72	43,97	32,56	50,49	24,94	59,30	52,88	57,59
65536					27,47	49,16	47,71	56,39	48,77	46,39	50,62	51,92	36,25
131072					47,25	66,55	71,73	62,03	61,35	58,58	63,33	40,73	57,83
262144					43,37	66,13	52,80	54,42	53,17	61,38	56,22	80,41	57,82
524288					51,49	77,11	57,95	63,77	69,40	66,36	61,00	62,48	68,64
1048576					52,29	57,33	64,88	61,53	63,97	62,32	62,95	61,96	73,38
2097152					48,54	61,36	60,05	72,33	70,96	56,97	65,44	62,07	63,67

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.41: Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils drei GlusterFS Nodes und Einkern-Prozessoren (Messung auf der OpenStack Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	0,37	0,50	0,50	0,51	0,48								
128	1,01	1,01	1,01	1,01	1,01	1,02							
256	1,94	1,97	1,89	1,87	1,98	1,97	1,98						
512	3,72	3,71	3,74	3,70	3,74	3,58	3,67	3,52					
1024	6,65	6,75	6,32	6,34	5,69	6,25	6,25	6,43	6,41				
2048	8,81	8,19	8,29	8,55	8,51	9,18	8,79	9,26	9,10	10,30			
4096	14,38	11,67	11,98	11,50	12,02	11,48	12,27	12,06	12,96	11,75	11,78		
8192	11,45	11,24	11,34	12,12	11,46	12,04	12,39	12,01	11,81	12,09	12,69	12,05	
16384	15,32	15,61	15,61	16,12	17,70	18,26	19,03	20,98	22,23	22,34	22,59	23,93	19,97
32768					23,87	23,64	21,15	21,65	22,73	22,91	22,62	22,68	23,11
65536					22,86	23,67	24,32	25,03	26,25	26,72	24,97	27,38	20,82
131072					24,15	26,93	26,82	28,76	31,81	27,68	21,42	34,73	25,71
262144					28,77	30,79	30,04	29,00	27,63	29,35	30,91	25,10	34,02
524288					32,05	21,52	27,54	23,33	24,33	20,82	25,32	25,34	21,59
1048576					21,09	20,74	22,13	22,24	19,60	16,91	15,11	12,91	5,89
2097152					4,01	12,33	12,46	13,30	7,10	5,29	5,37	4,58	4,94

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.42: Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils drei GlusterFS Nodes und Einkern-Prozessoren (Messung auf der AWS Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	4,94	9,23	3,52	8,12	10,41								
128	14,59	15,84	7,89	17,40	4,37	17,10							
256	21,78	25,96	27,82	17,08	20,05	14,21	15,79						
512	28,35	34,30	42,60	20,27	45,20	22,25	33,15	25,82					
1024	31,39	48,46	36,39	48,52	58,66	17,44	58,63	42,15	55,82				
2048	35,36	57,36	70,08	51,04	70,70	68,81	49,11	61,58	70,01	52,75			
4096	34,51	58,42	78,28	74,71	64,40	69,01	59,42	62,29	51,21	79,04	70,74		
8192	33,26	45,35	76,50	74,47	52,91	77,97	79,38	81,27	69,82	41,12	84,21	82,91	
16384	34,20	58,24	81,32	83,55	81,46	85,03	85,52	58,37	86,92	46,36	85,94	56,42	80,62
32768					83,29	83,96	81,54	79,74	85,36	83,37	84,34	84,00	85,64
65536					77,06	84,78	82,77	74,45	82,94	86,93	85,34	56,44	76,55
131072					80,96	86,45	80,36	87,06	80,48	79,85	76,38	77,68	85,92
262144					76,29	77,14	78,93	78,62	79,10	77,14	77,54	77,86	74,43
524288					68,11	68,78	68,67	68,66	67,98	68,64	68,63	68,57	68,82
1048576					74,24	74,24	73,59	73,45	72,52	73,78	71,97	74,06	74,10
2097152					90,09	90,83	90,77	90,62	90,71	89,65	90,47	89,32	89,20

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.43: Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils drei GlusterFS Nodes und Zweikern-Prozessoren (Messung auf der OpenStack Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	0,38	0,49	0,50	0,48	0,52								
128	0,97	1,03	1,02	0,99	0,94	1,01							
256	1,84	1,97	1,97	1,75	1,88	2,01	1,86						
512	3,74	3,86	3,42	3,91	3,91	3,61	3,90	3,79					
1024	7,02	5,96	6,26	5,89	5,40	6,65	6,70	5,78	5,26				
2048	11,00	9,71	8,33	10,08	8,39	8,91	9,68	9,87	10,90	10,84			
4096	15,18	12,93	12,91	12,85	12,90	12,45	12,74	12,89	13,95	14,16	15,50		
8192	20,55	18,53	18,56	18,80	17,79	19,09	19,07	18,82	18,91	18,82	19,31	19,44	
16384	20,82	23,60	24,85	23,49	24,90	25,12	24,98	25,24	24,28	25,38	24,55	25,14	25,20
32768					29,64	29,76	29,44	28,99	29,91	30,01	29,92	30,03	29,98
65536					32,28	32,82	33,09	32,26	32,32	32,56	32,31	32,76	32,65
131072					34,23	33,19	34,78	34,88	34,31	34,55	34,99	34,38	34,58
262144					33,96	34,15	34,29	35,54	33,66	34,20	34,20	34,47	34,31
524288					32,57	32,53	32,45	32,46	32,61	32,57	32,54	32,51	32,55
1048576					35,44	38,78	37,21	34,51	35,04	34,54	35,48	34,84	34,60
2097152					34,75	39,98	37,47	39,18	39,79	39,47	38,54	38,85	38,01

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.44: Schreibgeschwindigkeit zwischen AWS und OpenStack mit jeweils drei GlusterFS Nodes und Zweikern-Prozessoren (Messung auf der AWS Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	11,95	11,30	11,38	12,38	13,95								
128	21,38	22,20	24,28	23,81	28,66	27,71							
256	41,02	43,28	35,52	45,51	47,07	48,61	53,33						
512	57,59	52,11	58,08	50,45	54,20	63,36	76,58	103,04					
1024	62,54	66,39	74,11	70,56	72,88	76,34	83,58	44,94	72,46				
2048	77,04	81,25	88,60	28,70	18,97	26,21	38,20	39,47	29,78	221,16			
4096	82,77	81,49	87,99	19,30	22,15	22,03	19,12	24,51	36,01	59,83	589,10		
8192	22,13	82,81	17,03	22,55	19,67	23,05	21,64	114,44	145,60	47,12	63,43	432,91	
16384	59,98	19,80	94,02	20,69	21,24	21,13	20,51	20,45	31,34	189,47	221,35	137,66	783,88
32768					20,16	116,88	19,91	119,86	48,27	182,18	207,96	247,53	313,60
65536					103,26	25,71	103,90	120,20	156,54	175,21	195,80	219,66	272,90
131072					126,16	106,59	129,32	136,72	159,20	194,90	178,85	225,58	231,14
262144					45,42	61,27	28,08	26,84	32,18	48,20	54,10	85,15	58,29
524288					26,16	24,78	33,22	31,24	71,54	64,61	73,89	68,57	96,31
1048576					29,52	36,99	21,74	36,39	85,96	88,19	93,96	105,13	105,05
2097152					37,40	27,00	30,91	54,79	84,11	85,10	91,68	98,30	102,26

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.45: Lesegeschwindigkeitzwischen AWS und OpenStack mit jeweils drei GlusterFS Nodes und Einkern-Prozessoren (Messung auf der OpenStack Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	0,64	0,65	0,64	1,27	1,26								
128	1,02	1,01	1,28	1,26	2,56	2,52							
256	1,68	1,68	2,07	2,42	2,53	5,15	5,13						
512	2,55	2,52	2,89	3,24	3,31	5,10	6,62	10,09					
1024	3,34	3,33	3,67	3,91	4,01	5,03	5,46	10,49	20,28				
2048	3,98	3,93	4,10	4,29	4,22	4,79	5,08	8,85	17,26	40,09			
4096	4,18	4,23	4,38	4,43	4,40	4,95	5,05	8,03	14,44	25,71	74,87		
8192	4,68	4,65	4,65	3,76	3,49	3,57	3,80	5,69	6,19	11,95	18,30	88,23	
16384	3,50	3,50	3,59	3,73	3,60	3,79	4,62	7,36	11,52	15,50	22,72	35,20	192,27
32768					4,79	5,00	4,98	7,06	12,16	16,22	22,53	27,17	30,63
65536					4,95	4,92	5,02	7,68	11,85	16,29	21,30	24,82	31,40
131072					4,96	4,99	4,96	7,53	12,15	16,82	19,29	24,41	26,18
262144					4,99	5,01	5,02	7,42	12,33	15,68	20,16	23,21	22,96
524288					5,00	4,97	4,98	7,56	11,75	16,24	19,45	23,21	24,48
1048576					5,06	5,03	5,06	7,48	12,35	16,48	19,82	21,90	24,79
2097152					5,05	5,05	5,05	7,52	11,85	12,04	18,68	15,06	16,95

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.46: Lesegeschwindigkeitzwischen AWS und OpenStack mit jeweils drei GlusterFS Nodes und Einkern-Prozessoren (Messung auf der AWS Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	12,73	15,81	12,88	38,11	25,32								
128	19,17	19,29	26,30	38,22	66,50	71,75							
256	42,60	44,43	13,86	61,31	76,36	153,70	153,21						
512	52,78	56,00	67,19	68,44	78,76	123,13	199,87	307,08					
1024	61,77	75,81	70,38	72,56	74,47	102,31	95,06	209,00	158,14				
2048	70,30	68,59	70,35	74,91	65,31	82,77	88,48	111,30	225,45	404,40			
4096	69,86	73,37	71,60	72,73	80,59	81,37	92,05	118,03	160,25	328,31	802,47		
8192	81,26	74,65	68,18	78,42	75,31	77,20	81,46	111,74	154,54	247,63	387,61	1539,99	
16384	74,70	75,52	78,79	76,01	78,24	78,27	77,63	97,79	147,44	208,60	266,92	388,54	2944,33
32768					79,58	77,08	78,13	96,51	134,50	186,15	228,11	274,49	222,17
65536					76,42	78,62	75,46	94,71	136,36	181,98	216,54	252,71	360,76
131072					75,92	77,17	75,63	94,63	138,82	176,70	212,77	261,52	278,26
262144					76,82	76,62	76,18	92,77	139,64	179,19	212,05	235,07	270,74
524288					75,95	75,17	79,11	97,24	139,32	178,64	204,77	223,79	264,86
1048576					80,27	79,95	77,17	95,47	136,85	174,84	210,28	234,15	249,66
2097152					78,47	77,98	75,98	96,66	138,68	172,26	203,93	227,16	245,49

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.47: Lesegeschwindigkeit zwischen AWS und OpenStack mit jeweils drei GlusterFS Nodes und Zweikern-Prozessoren (Messung auf der OpenStack Plattform)

Dateigröße [kByte] \ Record Size [kByte]	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
64	32,50	33,85	26,63	13,24	37,74								
128	57,79	50,72	62,50	64,32	64,65	82,52							
256	97,48	100,12	104,07	31,86	116,10	144,05	47,88						
512	159,30	41,71	39,00	46,14	169,93	228,17	208,30	305,87					
1024	67,29	86,96	219,19	222,08	223,58	68,48	110,70	374,30	522,77				
2048	239,06	225,52	141,93	112,08	277,78	276,20	28,59	407,07	550,08	566,40			
4096	274,83	116,71	262,83	110,14	107,93	168,67	216,39	219,74	504,80	512,70	1685,49		
8192	67,87	87,11	209,09	51,89	102,86	167,28	161,94	410,52	275,21	571,86	725,08	2528,35	
16384	201,53	166,16	88,60	112,82	169,55	186,46	242,57	91,50	203,81	470,91	155,03	413,57	2985,36
32768					71,68	94,74	191,17	224,88	241,41	333,56	278,24	248,93	1104,67
65536					158,20	99,47	152,93	145,30	245,52	302,56	277,92	409,59	584,79
131072					110,29	163,03	164,03	193,25	231,82	234,96	250,98	266,74	427,05
262144					122,09	163,49	166,82	189,68	246,39	312,22	285,15	425,85	480,76
524288					143,67	143,40	153,61	190,76	283,01	294,85	397,60	481,04	466,21
1048576					130,82	150,36	150,93	219,68	260,35	263,97	366,84	483,68	451,82
2097152					139,67	150,20	147,80	199,54	257,90	317,89	395,05	465,23	485,61

Der Bereich wird von IOzone nicht gemessen

Legende: höchste Datenübertragungsrate [MByte/s] — niedrigste Datenübertragungsrate [MByte/s]

Abbildung E.48: Lesegeschwindigkeit zwischen AWS und OpenStack mit jeweils drei GlusterFS Nodes und Zweikern-Prozessoren (Messung auf der AWS Plattform)

Literaturverzeichnis

[AFS+10] Noraziah Ahmad, Ainul Azila Che Fauzi, Roslina Mohd Sidek, Noriyani Mat Zin, and Abul Hashem Beg. Lowest data replication storage of binary vote assignment data grid. In *International Conference on Networked Digital Technologies*, pages 466–473. Springer, 2010.

[Avi76] A Aviziens. Fault-tolerant systems. *IEEE Transactions on Computers*, 25(12):1304–1312, 1976.

[Ben14] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

[Ber14] David Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84, 2014.

[Bie06] Eric W Biederman. Multiple instances of the global linux namespaces. In *Proceedings of the Linux Symposium*, volume 1, pages 101–112. Citeseer, 2006.

[BLRK03] Markus Böhm, Stefanie Leimeister, Christoph Riedl, and Helmut Krcmar. Cloud Computing: Outsourcing 2.0 oder ein neues Geschäftsmodell zur Bereitstellung von IT-Ressourcen? *IM-Fachzeitschrift für Information Management und Consulting*, 24:6–14, 2003.

[Bre00a] Eric Brewer. Keynote – towards robust distributed systems. In *19th ACM Symposium on Principles of Distributed Computing*, Portland, 2000.

[Bre00b] Eric A Brewer. Towards robust distributed systems. In *PODC*, volume 7, 2000.

[Bre12] Eric Brewer. Cap twelve years later: How the"rules"have changed. *Computer*, 45(2):23–29, 2012.

[BRF14] Zhang Baojun, Pan Ruifang, and Ye Fujun. Analyzing and improving load balancing algorithm of moosefs. *International Journal of Grid and Distributed Computing*, 7(4):169–176, 2014.

[CDK02] George Coulouris, Jean Dollimore, and Tim Kindberg. *Verteilte Systeme: Konzepte und Design*. Addision-Wesley, 2002.

[DG16] Elsie DeBrie and David Goeschel. Open source software licenses: Legal implications and practical guidance. *The Nebraska Lawyer*, 2016.

[DHH+03] Stephanie Donovan, Gerrit Huizenga, Andrew J Hutton, C Craig Ross, Martin K Petersen, and Philip Schwan. Lustre: Building a file system for 1000-node clusters. In *Proceedings of the Linux Symposium*, 2003.

[DIN16] *Informationstechnik - Cloud Computing - Übersicht und Vokabular*. DIN ISO/IEC 17788. Beuth, April 2016.

[FFRR14] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An updated performance comparison of virtual machines and linux containers. Technical report rc25482 (aus1407-001), IBM Research Division, 2014.

[FL14] Martin Fowler and James Lewis. Microservices. *http://www.martinfowler.com/articles/microservices.html (aufgerufen am 02.01.2017)*, 2014.

[FLdM95] Kazi Farooqui, Luigi Logrippo, and Jan de Meer. The iso reference model for open distributed processing: an introduction. *Computer Networks and ISDN Systems*, 27(8):1215–1229, 1995.

[flo] Flocker. *https://clusterhq.com/flocker (aufgerufen am 20.01.17)*.

[FMN+05] Michael Factor, Kalman Meth, Dalit Naor, Ohad Rodeh, and Julian Satran. Object storage: The future building block for storage systems a position paper. 2005.

[GL02] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2):51–59, 2002.

[glu] Glusterfs. *https://gluster.readthedocs.io/en/latest/Administrator(aufgerufen am 20.01.17)*.

[Ham07] James Hamilton. On designing and deploying internet-scale services. In *LISA*, volume 18, pages 1–18, 2007.

[HCK+08] Felix Hupfeld, Toni Cortes, Björn Kolbeck, Jan Stender, Erich Focht, Matthias Hess, Jesus Malo, Jonathan Marti, and Eugenio Cesario. The xtreemfs architecture—a case for object-based file systems in grids. *Concurrency and computation: Practice and experience*, 20(17):2049–2060, 2008.

[Hei] Jan Heichler. An introduction to beegfs. *https://www.beegfs.io/content (aufgerufen am 20.01.17)*.

[HHDL15] Juliana Hildebrandt, Dirk Habich, Patrick Damme, and Wolfgang Lehner. Modularization of lightweight data compression algorithms. Technical report, Technical report, Department of Computer Science, Technische Universität Dresden, 2015.

[HMP10] Denis Hünich and Ralph Müller-Pfefferkorn. Managing large datasets with irods—a performance analysis. In *Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on*, pages 647–654. IEEE, 2010.

[inf] Infinit. *https://infinit.sh (aufgerufen am 03.02.2017)*.

[JGG15] Pragya Jain, Anita Goel, and SC Gupta. Object storage as a service. *International Journal of Innovations Advancement in Computer Science*, Volume 4, Special Issue, 2015.

[JW00] Prasad Jogalekar and Murray Woodside. Evaluating the scalability of distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 11(6):589–603, 2000.

[KM12] Daniel Kailer and Peter Mandl. Hoch verfügbare und konsistente datenhaltung in der cloud. *HMD Praxis der Wirtschaftsinformatik*, 49(6):69–77, 2012.

[KP16] Nane Kratzke and R Peinl. Clouns-a cloud-native application reference model for enterprise architects. In *Enterprise Distributed Object Computing Workshop (EDOCW), 2016 IEEE 20th International*, pages 1–10. IEEE, 2016.

[Kra14] Nane Kratzke. Lightweight virtualization cluster how to overcome cloud vendor lock-in. *Journal of Computer and Communications*, 2(12):1, 2014.

[leo] Leofs. *https://github.com/leo-project/leofs (aufgerufen am 26.01.2017)*.

[Leu03] Jens Leuschner. Massenspeicher-netze auf ip-basis. Master's thesis, Technische Universität Chemnitz - Fakultät für Informatik, 2003.

[liz] Lizardfs. *https://lizardfs.com/about-lizardfs/ (aufgerufen am 20.01.17)*.

[LPC+12] Cheng Li, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno M Preguiça, and Rodrigo Rodrigues. Making geo-replicated systems fast as possible, consistent when necessary. In *OSDI*, volume 12, pages 265–278, 2012.

[Mar15] Luke Marsden. Demo: Flocker on coreos on aws. *https://cluster-hq.com/2015/09/01/flocker-runs-on-coreos/*, 2015.

[MCT08] Lijun Mei, W.K. Chan, and T.H. Tse. A tale of clouds: Paradigm comparisons and some thoughts on research issues. In *Proceedings Of The 3rd IEEE Asia-Pacific Services Computing Conference*, pages 464–469, 2008.

[Mey97] Bertrand Meyer. *Object-oriented software construction*, volume 2. Prentice Hall, 1997.

[MG07] Franco Milicchio and Wolfgang Alexander Gehrke. *Distributed services with openafs: for enterprise and education*. Springer Science & Business Media, 2007.

[MG11a] Peter Mell and Tim Grance. The nist definition of cloud computing. *Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg*, 2011.

[MG11b] Peter Mell and Timothy Grance. The nist definition of cloud computing. Technical report, National Institute o Standards and Technology, 2011.

[MGR03] Mike Mesnier, Gregory R Ganger, and Erik Riedel. Object-based storage. *IEEE Communications Magazine*, 41(8):84–90, 2003.

[MK15] Karl Matthias and Sean P Kane. *Docker: Up & Running*. O'Reilly Media, Inc., 2015.

[MPR15] Gerald Münzl, Michael Pauly, and Martin Reti. Cloud computing als neue herausforderung für management und it. 2015.

[Mül11] Klaus-Rainer Müller. Sicherheitsarchitektur. In *IT-Sicherheit mit System*, pages 178–378. Springer, 2011.

[obj] Objectivefs. *https://objectivefs.com (aufgerufen am 20.01.17)*.

[ORR+13] Michael Ovsiannikov, Silvius Rus, Damian Reeves, Paul Sutter, Sriram Rao, and Jim Kelly. The quantcast file system. *Proceedings of the VLDB Endowment*, 6(11):1092–1101, 2013.

[PDE+14] Dimitri Pertin, Sylvain David, Pierre Evenou, Benoît Parrein, and Nicolas Normand. Distributed file system based on erasure coding for i/o-intensive applications. In *CLOSER*, pages 451–456, 2014.

[PHS+16] Leandro Pacheco, Raluca Halalai, Valerio Schiavoni, Fernando Pedone, Etienne Riviere, and Pascal Felber. Globalfs: A strongly consistent multi-site file system. In *Reliable Distributed Systems (SRDS), 2016 IEEE 35th Symposium on*, pages 147–156. IEEE, 2916.

[PW] Tom Preston-Werner. Semantic versioning 2.0.0. *http://semver.org (aufgerufen am 15.02.17)*.

[QK16] Peter-Christian Quint and Nane Kratzke. Taming the complexity of elasticity, scalability and transferability in cloud computing - cloud-native applications for smes. *International Journal on Advances in Networks and Services*, 9(34):389–400, 2016.

[Qui16] Peter-Christian Quint. Vendor lock-in im cloud computing! Was bringen Container und Container-Cluster? *Online Themenspecial Microservices und Docker*, 2016.

[Riv92] Ronald Rivest. The md5 message-digest algorithm. *RFC 1321*, 1992.

[RMH+10] Arcot Rajasekar, Reagan Moore, Chien-yi Hou, Christopher A Lee, Richard Marciano, Antoine de Torcy, Michael Wan, Wayne Schroeder, Sheau-Yen Chen, and Lucas Gilbert. irods primer: integrated rule-oriented data system. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 2(1):1–143, 2010.

[s3q] S3ql. *https://bitbucket.org/nikratio/s3ql (aufgerufen am 20.01.17)*.

[SCG+12] Da-Wei Sun, Gui-Ran Chang, Shang Gao, Li-Zhong Jin, and Xing-Wei Wang. Modeling a dynamic data replication strategy to increase system availability in cloud computing environments. *Journal of computer science and technology*, 27(2):256–272, 2012.

[Sch01] Rüdiger Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 101–102. IEEE, 2001.

[Sch13] Alexander Schaaf. *Open-Source-Lizenzen: Untersuchung der GPL, LGPL, BSD und Artistic License*. Diplomica Verlag, 2013.

[sea] Seaweedfs. *https://github.com/chrislusf/seaweedfs (aufgerufen am 20.01.17)*.

[SGLM08] Mark W Storer, Kevin Greenan, Darrell DE Long, and Ethan L Miller. Secure data deduplication. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 1–10. ACM, 2008.

[SH02] Frank B Schmuck and Roger L Haskin. Gpfs: A shared-disk file system for large computing clusters. In *FAST*, volume 2, 2002.

[SKRC10] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*, pages 1–10. IEEE, 2010.

[SL16] Manikandan Selvaganesan and Mohamed Ashiq Liazudeen. An insight about glusterfs and its enforcement techniques. In *Cloud Computing Research and Innovations (ICCCRI), 2016 International Conference on*, pages 120–127. IEEE, 2016.

[SMM14] Matthew NO Sadiku, Sarhan M Musa, and Omonowo D Momoh. Cloud computing: opportunities and challenges. *IEEE potentials*, 33(1):34–36, 2014.

[SPEW11] Amir Ali Semnanian, Jeffrey Pham, Burkhard Englert, and Xiaolong Wu. Virtualization technology and its impact on computer hardware architecture. In *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on*, pages 719–724. IEEE, 2011.

[SPF+07] Stephen Soltesz, Herbert Pötzl, Marc E Fiuczynski, Andy Bavier, and Larry Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 275–287. ACM, 2007.

[TLM15] Alexander Tormasov, Anatoly Lysov, and Emil Mazur. Distributed data storage systems: Analysis, classification and choice. 27(6):225–252, 2015.

[TMM+02] Osamu Tatebe, Youhei Morita, Satoshi Matsuoka, Noriyuki Soda, and Satoshi Sekiguchi. Grid datafarm architecture for petascale data intensive computing. In *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on*. IEEE, 2002.

[tor] Torus. *https://github.com/coreos/torus (aufgerufen am 20.01.17)*.

[TVS07] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems*. Prentice-Hall, 2007.

[VBR06] Srikumar Venugopal, Rajkumar Buyya, and Kotagiri Ramamohanarao. A taxonomy of data grids for distributed data sharing, management, and processing. *ACM Computing Surveys (CSUR)*, 38(1):3, 2006.

[Wei07] Sage A Weil. *Ceph: reliable, scalable, and high-performance distributed storage*. PhD thesis, University of California, Santa Cruz, 2007.

[WPG+10] Jiyi Wu, Lingdi Ping, Xiaoping Ge, Ya Wang, and Jianqing Fu. Cloud storage as the infrastructure of cloud computing. In *Intelligent Computing and Cognitive Informatics (ICICCI), 2010 International Conference on*, pages 380–383. IEEE, 2010.

[YLIQ11] Shuangyang Yang, Walter B Ligon III, and Elaine C Quarles. Scalable distributed directory implementation on orange file system. *Proc. IEEE Intl. Wrkshp. Storage Network Architecture and Parallel I/Os (SNAPI)*, 2011.

[YVCJ07] Weikuan Yu, Jeffrey Vetter, R Shane Canon, and Song Jiang. Exploiting lustre file joining for effective collective io. In *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, pages 267–274. IEEE, 2007.