Symcloud: Filesync and Collaboration platform

Wachter Johannes

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Seit den Abhörskandalen durch die NSA und anderen Geheimdiensten ist es immer mehr Menschen wichtig, die Kontrolle über die eigenen Daten zu behalten. Aufgrund dessen erregen Projekte wie Diaspora[1], ownCloud[2] und ähnliche Softwarelösungen immer mehr Aufmerksamkeit. Die beiden genannten Softwarelösungen decken zwei sehr wichtige Bereiche der persönlichen Datenkontrolle ab.

Diaspora ist ein dezentrales soziales Netzwerk. Die Benutzer von diesem Netzwerk sind durch die verteilte Infrastruktur nicht von einem Betreiber abhängig. Es ermöglicht, seinen Freunden bzw. der Familie, eine private Plattform anzubieten. Das Interessante daran ist, dass sich sogenannten Pods (dezentrale Knoten), beliebig untereinander vernetzen lassen und damit ein P2P Netzwerk aufbauen lässt. Pods können von jedem installiert und betrieben werden; dabei kann der Betreiber bestimmen, wer in sein Netzwerk eintreten darf und welche Server mit seinem verbunden sind. Die verbundenen Pods tauschen ohne einen zentralen Knoten, Daten aus und sind dadurch unabhängig. Dies garantiert die volle Kontrolle über seine Daten im Netzwerk (siehe "Was Ist Dezentralisierung" 2015).

Das Projekt "ownCloud" ist eine Software, die es ermöglicht, Daten in einer privaten Cloud zu verwalten. Mittels Endgeräte-Clients können die Daten synchronisiert und über die Plattform auch geteilt werden. Insgesamt bietet die Software einen ähnlichen Funktionsumfang gängiger kommerzieller Lösungen an (siehe "Owncloud Features" 2015). Zusätzlich bietet es eine Kollaborationsplattform, mit der zum Beispiel Dokumente über einen online Editor, von mehreren Benutzern gleichzeitig, bearbeitet werden können. Diese Technologie basiert auf

[1] https://diasporafoundation.org/
[2] https://owncloud.org/

der JavaScript Library WebODF[3].

1.1 Projektbeschreibung

Symcloud ist eine private Cloud-Software, die es ermöglicht, über dezentrale Knoten (ähnlich wie Diaspora) Daten über die Grenzen des eigenen Servers hinweg zu teilen. Verbundene Knoten tauschen über sichere Kanäle Daten aus, die anschließend über einen Client mit dem Endgerät synchronisiert werden können.

TODO genauere Beschreibung

Die Software baut auf modernen Web-Technologien auf und verwendet als Basis das PHP-Framework Symfony2[4]. Dieses Framework ist eines der beliebtesten in der Open-Source Community. Es bietet neben der Abstraktion von HTTP-Anfragen auch einen Dependency-Injection-Container und viele weitere Komponenten wie zum Beispiel Routing und Event Dispatcher. Zusätzlich erleichtert es die Entwicklung von großen PHP-Projekten, durch die Möglichkeit den Code in Komponenten, sogenannten Bundles, zu gliedern. Diese können mit der Community geteilt werden.

Als Basis für die Plattform verwendet Symcloud das Content-Management-Framework SULU[5] der Vorarlberger Firma MASSIVE ART WebServices[6] aus Dornbirn. Es bietet ein erweiterbares Admin-UI, eine Benutzerverwaltung und ein Rechtesystem. Diese Features ermöglichen Symcloud eine schnelle Entwicklung der Oberfläche und deren zugrundeliegenden Services.

[3] http://webodf.org/
[4] http://symfony.com/
[5] http://www.sulu.io
[6] http://www.massiveart.com/de

1.2 Inspiration

TODO Noch einmal ownCloud - Diaspora und Ted Nelson mit dem Xanadu Projekt

1.3 Anforderungen

TODO Anforderungen an das Projekt (auch in Bezug auf xanadu)

2 Stand der Technik

In diesem Kapitel werden moderne Anwendungen und ihre Architektur analysiert.

TODO bessere Einleitung (=

2.1 Verteilte Systeme

Andrew Tannenbaum definiert “verteilte Systeme” in seinem Buch folgendermaßen:

```
"Ein verteiltes System ist eine Menge voneinander unabhängiger
Computer, die dem Benutzer wie ein einzelnes kohärentes
System erscheinen"
```

Diese Definition beinhaltet zwei Aspekte. Der eine Aspekt besagt, dass die einzelnen Maschinen in einem verteilten System autonom sind. Der zweite Aspekt bezieht sich auf die Software, die die Systeme miteinander verbinden. Durch die Software glaubt der Benutzer, dass er es mit einem einzigen System zu tun hat (siehe Tanenbaum and Steen 2003, 18).

Eines der besten Beispiele für verteilte Systeme sind Cloud-Computing Dienste. Diese Dienste bieten verschiedenste Technologien an. Sie umfassen Rechnerleistung, Speicher, Datenbanken und Netzwerke. Der Anwender kommuniziert hierbei immer nur mit einem System, allerdings verbirgt sich hinter diesen Anfragen ein komplexes System, das sehr stark auf Virtualisierung setzt.

Gerade im Bereich der verteilten Dateisysteme, bietet sich die Möglichkeit, Dateien über mehrere Server zu verteilen. Dies ermöglicht eine Verbesserung von Datensicherheit, durch Replikation über verschiedene Server und Steigerung der Effizienz, durch paralleles Lesen der Daten. Diese Dateisysteme trennen meist die Nutzdaten von ihren Metadaten und halten diese, als Daten zu den Daten, in einer effizienten Datenbank. Um zum Beispiel Informationen zu einer Datei zu erhalten, wird die Datenbank nach den Informationen durchsucht und ohne auf die Nutzdaten zugreifen zu müssen, an den Benutzer weitergeleitet. Dies steigert sehr stark die Effizienz der Anfrage (siehe Seidel 2013). Das Kapitel 4.3 befasst sich genauer mit verteilten Dateisystemen.

2.2 Dropbox

Dropbox-Nutzer können jederzeit von ihrem Desktop aus, über das Internet, mobile Geräte oder mit Dropbox verbundene Anwendungen auf Dateien und Ordner zugreifen.

Alle diese Clients stellen Verbindungen mit sicheren Servern her, über die sie Zugriff auf Dateien haben und Dateien für andere Nutzer freigeben können. Wenn Daten auf einem Client geändert werden, werden diese automatisch mit dem Server synchronisiert. Verknüpfte Geräte aktualisieren sich automatisch. Dadurch werden Dateien, die hinzugefügt, verändert oder gelöscht werden, auf allen Clients aktualisiert bzw. gelöscht.

Der Dropbox-Service betreibt verschiedenste Dienste, die sowohl für die Handhabung und Verarbeitung von Metadaten, als auch für die Verwaltung des Blockspeichers verantwortlich sind. (siehe “Wie Funktioniert Der Dropbox-Service” 2015)

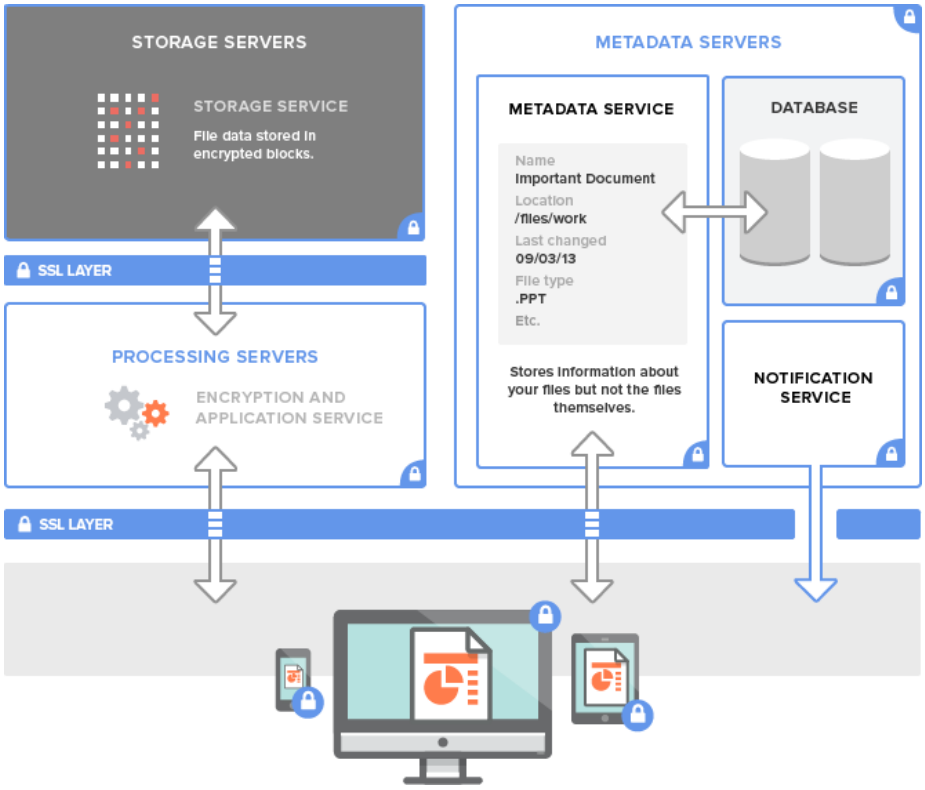

Abbildung 1: Blockdiagramm der Dropbox Services (Quelle https://www.dropbox.com/help/1968)

In der Abbildung 1 werden die einzelnen Komponenten in einem Blockdiagramm dargestellt. Es gliedert sich in drei größere Blöcke, die jeweils über einen Service angesprochen werden können:

- Metadata
- Storage

- Processing

Wie im Kapitel 2.1 beschrieben trennt Dropbox intern die Dateien von ihren Metadaten. Der Metadata Service speichert die Metadaten und Informationen zu ihrem Speicherort in einer Datenbank, aber der Inhalt der Daten liegt in einem separaten Storage Service. Dieser Service verteilt die Daten wie ein "Load Balancer" über viele Server.

Der Storage Service ist wiederum von außen durch einen Application Service abgesichert. Die Authentifizierung erfolgt über das OAuth2 Protokoll (siehe "Core API Dokumentation" 2015). Diese Authentifizierung wird für alle Services verwendet, auch für den Metadata Service und der Notification Service.

2.3 ownCloud

Nach den neuesten Entwicklungen arbeitet ownCloud an einem ähnlichen Feature wie Symcloud. Unter dem Namen "Remote shares" wurde in der Version 7 eine Erweiterung in den Core übernommen, mit dem es möglich ist, sogenannte "Shares" mittels einem Link auch in einer anderen Installation einzubinden. Dies ermöglicht es Dateien auch über die Grenzen des Servers hinweg zu teilen. (siehe "Server2Server - Sharing" 2015)

Die kostenpflichtige Variante von ownCloud geht hier noch einen Schritt weiter. In Abbildung 2 ist zu sehen, wie ownCloud als eine Art Verbindungsschicht zwischen verschiedenen "On-Site", also Daten die vor Ort bereitstehen, und Daten aus der Cloud, dienen soll. (siehe ownCloud 2015, 1)

Um die Integration in ein Unternehmen zu erleichtern, bietet es verschiedenste Services an. Unter anderem ist es möglich Benutzerdaten über LDAP oder ActiveDirectory zu verwalten und damit ein doppeltes Verwalten der Benutzer zu vermeiden. (siehe ownCloud 2015, 2)

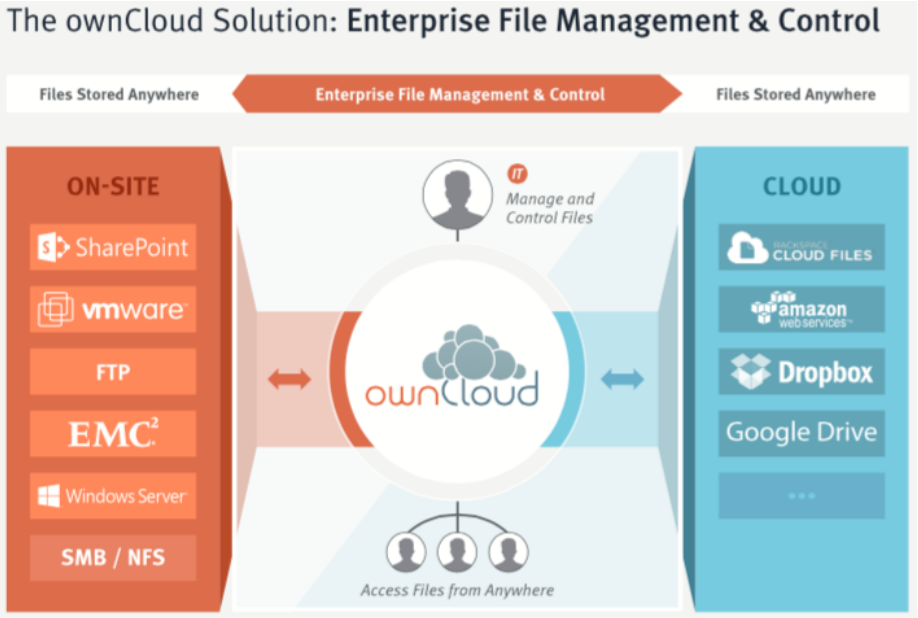

Abbildung 2: ownCloud Enterprise Architektur Übersicht (Quelle ownCloud 2015)

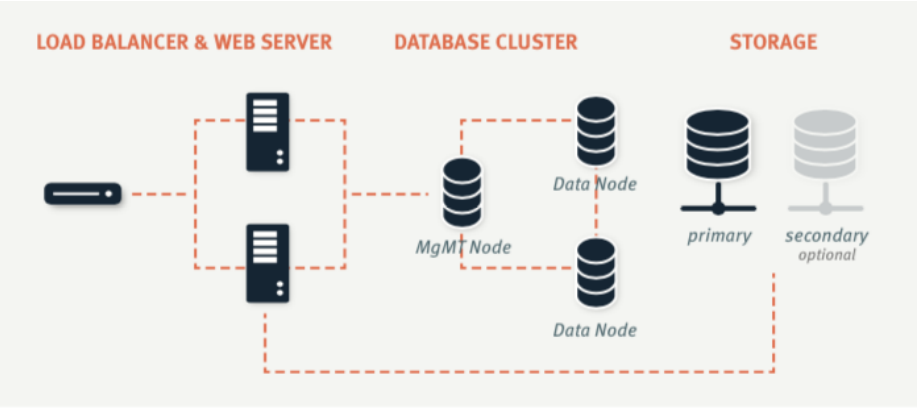

Abbildung 3: Bereitstellungsszenario von ownCloud(Quelle ownCloud 2015)

Für einen produktiven Einsatz wird eine skalierbare Architektur, wie in Abbildung 3, vorgeschlagen. An erster Stelle steht ein Load-Balancer, der die Last der Anfragen an mindestens zwei Webserver verteilt. Diese Webserver sind mit einem MySQL-Cluster verbunden, in dem die User-Daten, Anwendungsdaten und Metadaten der Dateien gespeichert sind. Dieser Cluster besteht wiederum aus mindestens zwei Datenbankservern. Dies ermöglicht auch bei stark frequentierten Installationen eine horizontale Skalierbarkeit. Zusätzlich sind die Webserver mit dem File-Storage verbunden. Auch hier ist es möglich, diesen redundant bzw. skalierbar aufzubauen, um die Effizienz und Sicherheit zu erweitern. (siehe ownCloud 2015, 3–4)

2.4 Diaspora

TODO kaum Infos gefunden, weitere suche notwendig

2.5 Zusammenfassung

TODO Zusammenfassung state of the art Kapitel

3 Cloud Computing

3.1 Amazon S3

3.2 Eucalyptus

3.3 Heroku

3.4 Docker

3.5 Evaluation

4 Speicherverwaltung

Ein wichtiger Aspekt von Cloud-Anwendungen ist die Speicherverwaltung. Es bieten sich verschiedenste Möglichkeiten der Datenhaltung in der Cloud an. Dieses Kapitel beschäftigt sich mit der Evaluierung von verschiedenen Diensten bzw. Lösungen, mit denen Speicher verwaltet und möglichst effizient zur Verfügung gestellt werden kann.

Aufgrund der Anforderungen des Projektes werden folgende Anforderungen an die Speicherlösung gestellt.

Ausfallsicherheit Die Speicherlösung ist das Fundament einer jeder Cloud-Anwendung. Ein Ausfall dieser Schicht bedeutet oft ein Ausfall der kompletten Anwendung. ???

Skalierbar Die Datenmengen einer Cloud-Anwendung sind oft schwer abschätzbar und können sehr große Ausmaße annehmen. Daher ist eine wichtige Anforderung an eine Speicherlösung die Skalierbarkeit. ???

Datenschutz Der Datenschutz ist ein wichtiger Punkt beim betreiben der eigenen Cloud-Anwendung. Meist gibt es eine kommerzielle Konkurrenz, die mit günstigen Preisen die Anwender anlockt, um ihre Daten zu verwerten. Die Möglichkeit, Daten Privat auf dem eigenen Server zu speichern, sollte somit gegeben sein. Damit Systemadministratoren nicht auf einen Provider angewiesen sind.

Flexibilität Um Daten flexibel speichern zu können, sollte es möglich sein, Verlinkungen und Metadaten direkt in der Speicherlösung abzulegen. Dies erleichtert die Implementierung der eigentlichen Anwendung.

Versionierung Ein optionale Eigenschaft ist die integrierte Versionierung der Daten. Dies würde eine Vereinfachung der Anwendungslogik ermöglichen, da Versionen nicht in einem separaten Speicher abgelegt werden müssen.

Performance ???

4.1 Datenhaltung in Cloud-Infrastrukturen

Wenn man Speicherstrukturen in der Cloud genauer betrachtet, gibt es grundsätzlich drei Möglichkeiten.

Objekt-Speicherdienste wie zum Beispiel Amazon S3[7], ermöglichen das Speichern von sogenannten Objekten (Dateien, Ordner und Metadaten). Sie sind optimiert für den parallelen Zugriff von mehreren Instanzen einer Anwendung.

Verteilte Dateisysteme fungieren als einfache Laufwerke und abstrahieren dadurch den komplexen Ablauf der darunter liegenden Services. Der Zugriff

[7] http://aws.amazon.com/de/s3/

auf diese Dateisysteme erfolgt meist über system-calls wie zum Beispiel `fopen` oder `fclose`.

Datenbank gestützte Dateisysteme wie zum Beispiel GridFS[8] von MondoDB, erweitern Datenbanken um große Dateien effizient und sicher abzuspeichern.

4.2 Amazon Simple Storage Service (S3)

Amazon Simple Storage Service bietet Entwicklern einen sicheren, beständigen und sehr gut skalierbaren Objektspeicher. Es dient der einfachen und sicheren Speicherung großer Datenmengen (siehe "Amazon S3" 2015). Daten werden in sogenannten Buckets gegliedert. Jeder Bucket kann unbegrenzt Objekte enthalten. Die Gesamtgröße der Objekte ist jedoch auf 5TB beschränkt. Sie können nicht verschachtelt werden, allerdings können sie Ordner enthalten, um die Objekte zu gliedern.

Die Kernfunktionalität des Services besteht darin, Daten in sogenannten Objekten zu speichern. Diese Objekte können bis zu 5GB groß werden. Zusätzlich wird zu jedem Objekt ca. 2KB Metadaten abgelegt. Die Tabelle 1 enthält eine Liste von systemdefinierten Metadaten. Einige dieser Metadaten können vom Benutzer überschrieben werden, wie zum Beispiel `x-amz-storage-class`, andere werden vom System automatisch gesetzt, wie zum Beispiel `Content-Length` (siehe "Object Key and Metadata" 2015).

Tabelle 1: Objekt Metadaten

Name	Description
Date	Object creation date.

[8] http://docs.mongodb.org/manual/core/gridfs/

Name	Description
Content-Length	Object size in bytes.
Last-Modified	Date the object was last modified.
Content-MD5	The base64-encoded 128-bit MD5 digest of the object.
x-amz-server-side-encryption	Indicates whether server-side encryption is enabled for the object, and whether that encryption is from the AWS Key Management Service (SSE-KMS) or from AWS-Managed Encryption (SSE-S3).
x-amz-version-id	Object version. When you enable versioning on a bucket, Amazon S3 assigns a version number to objects added to the bucket.
x-amz-delete-marker	In a bucket that has versioning enabled, this Boolean marker indicates whether the object is a delete marker.
x-amz-storage-class	Storage class used for storing the object.
x-amz-website-redirect-location	Redirects requests for the associated object to another object in the same bucket or an external URL.
x-amz-server-side-encryption-aws-kms-key-id	If the x-amz-server-side-encryption is present and has the value of aws:kms, this indicates the ID of the Key Management Service (KMS) master encryption key that was used for the object.
x-amz-server-side-encryption-	Indicates whether server-side encryption with customer-provided encryption keys (SSE-C) is enabled.

Name	Description
customer-algorithm	

Zusätzlich zu diesen systemdefinierten Metadaten ist es möglich benutzerdefinierte Metadaten zu speichern. Das Format dieser Metadaten entspricht einer Key-Value Liste. Sie sind auf 2KB limitiert.

4.2.1 Versionierung

Die Speicherlösung bietet eine Versionierung der Objekte an. Diese kann über eine Rest-API, mit folgendem Inhalt, in jedem Bucket aktiviert werden.

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Enabled</Status>
</VersioningConfiguration>
```

Ist die Versionierung aktiviert, gilt diese für alle Objekte, die dieser enthält. Wird anschließend ein Objekt überschrieben, resultiert dies in einer neuen Version, dabei wird die Version-ID im Metadaten Feld `x-amz-version-id` auf einen neuen Wert gesetzt (siehe “Using Versioning” 2015). Dies veranschaulicht die Abbildung 4.

4.2.2 Skalierbarkeit

Die Skalierbarkeit ist aufgrund der, von Amazon verwalteten Umgebung, sehr einfach. Es wird soviel Speicherplatz zur Verfügung gestellt, wie benötigt wird. Der Umstand, dass mehr Speicherplatz benötigt wird, zeichnet sich nur auf der Rechnung des Betreibers ab.

Abbildung 4: Versionierungsschema von Amazon S3 (Quelle “Using Versioning” 2015)

4.2.3 Datenschutz

Amazon ist ein US-Amerikanisches Unternehmen und wir aus diesem Grund, wie andere Dienste, seit Jahren kritisiert. Um dieses Problem zu kompensieren, können Systemadministratoren sogenannte “Availability Zones” ausgewählt werden und damit steuern, wo ihre Daten gespeichert werden. Zum Beispiel werden Daten aus einem Bucket mit der Zone Irland, auch wirklich in Irland gespeichert. Zusätzlich gewährt Amazon die Verschlüsselung der Daten (siehe Wikipedia 2015a).

Wer bedenken hat, seine Daten aus den Händen zu geben, kann auf verschiedene kompatible Lösungen zurückgreifen.

4.2.4 Alternativen zu Amazon S3

Es gibt einige Amazon S3 kompatible Anbieter, die einen ähnlichen Dienst bieten. Diese sind allerdings meist auch US-Amerikanische Firmen und daher gleich vertrauenswürdig wie Amazon. Wer daher auf Nummer sicher gehen will und seine Daten bzw. Rechner-Instanzen ganz bei sich behalten will, kommt nicht um eine Installation von Cluster Lösungen.

Eucalyptus ist eine Open-Source-Infrastruktur zur Nutzung von Cloud-Computing auf Rechner Cluster. Der Name ist ein Akronym für "Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems". Die hohe Kompatibilität macht diese Software-Lösung zu einer optimalen Alternative zu Amazon-Web-Services (siehe Wikipedia 2015b). Dieser Dienst bietet keine Versionierung

Riak Cloud Storage ist eine Software, mit der es möglich ist, ein verteilter Objekt-Speicherdienst zu betreiben. Es implementiert die Schnittstelle von Amazon S3 und ist damit kompatibel zu der aktuellen Version (siehe Basho Technologies 2015). Es unterstützt die meisten Funktionalitäten, die Amazon bietet. Die Installation von Riak-CS ist im Gegensatz zu Eucalyptus sehr einfach und kann daher auf nahezu jedem System durchgeführt werden.

Beide vorgestellten Dienste bieten momentan keine Möglichkeit Objekte zu versionieren. Außerdem ist das vergeben von Berechtigungen für andere Benutzer ebenfalls nur mit Amazon S3 möglich.

4.2.5 Performance

HostedFTP veröffentlichte im Jahre 2009 in einem Perfomance Report ihre Erfahrungen mit der Performance zwischen EC2 (Rechner Instancen) und S3 (siehe "Amazon S3 and EC2 Performance Report – How Fast Is S3" 2009). Über ein Performance Modell wurde festgestellt, dass die Zeit für den Download einer Datei in zwei Bereiche aufgeteilt werden kann.

Feste Transaktionszeit ist ein fixer Zeitabschnitt, der für die Bereitstellung oder Erstellung der Datei benötigt wird. Beeinflusst wird diese Zeit kaum,

allerdings kann es aufgrund schwankender Auslastung zu Verzögerungen kommen.

Downloadzeit ist liniar abhängig zu der Dateigröße und kann aufgrund der Bandbreite schwanken.

Ausgehend von diesen Überlegungen kann davon ausgegangen werden, dass die Upload- bzw. Downloadzeit einen linearen Verlauf über die Dateigröße aufweist. Diese These wird von den Daten unterstützt. Aus dem Diagramm (Abbildung 5) kann die feste Transaktionszeit von ca. 140ms abgelesen werden.

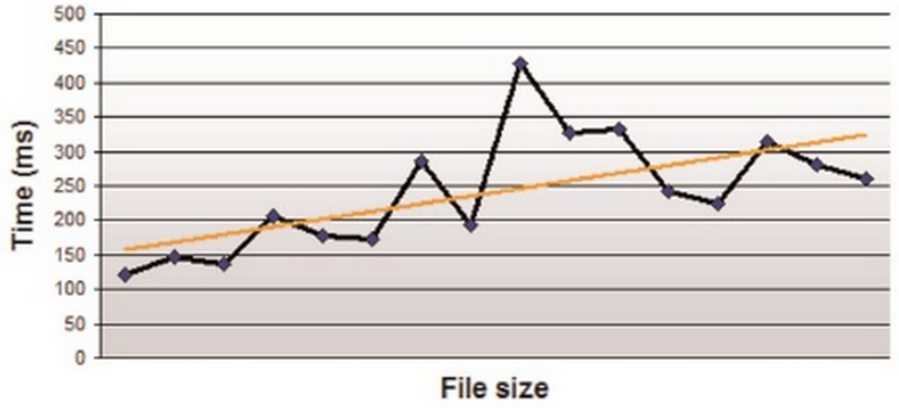

Abbildung 5: Upload Analyse zwischen EC2 und S3 (Quelle "Amazon S3 and EC2 Performance Report – How Fast Is S3" 2009)

Für den Download von Dateien entsteht laut den Daten aus dem Report keine fixe Transaktionszeit. Die Zeit für den Download ist also nur von der Größe der Datei und der Bandbreite abhängig.

4.3 Verteilte Dateisysteme

Verteilte Dateisysteme unterstützen die gemeinsame Nutzung von Informationen in Form von Dateien. Es bietet Zugriff auf Dateien, die auf einem entfernten Server

abgelegt sind, wobei eine ähnliche Leistung und Zuverlässigkeit erzielt wird, wie für lokal gespeicherte Daten. Wohldurchdachte Dateisysteme erzielen oft bessere Ergebnisse in Leistung und Zuverlässigkeit als lokale Systeme. Die entfernten Dateien werden genauso verwendet wie lokale Dateien, da verteilte Dateisysteme die Schnittstelle des Betriebssystems emulieren. Dadurch können die Vorteile von verteilten Systemen in einem Programm genutzt werden, ohne dieses anzupassen. Die Schreibzugriffe bzw. Lesezugriffe erfolgen über ganz normale `system-calls` (siehe Coulouris, Dollimore, and Kindberg 2003 S. 363ff.).

Dies ist auch ein großer Vorteil zu Speicherdiensten wie Amazon S3. Da die Schnittstelle zu den einzelnen Systemen abstrahiert werden, muss die Software nicht angepasst werden, wenn das Dateisystem gewechselt wird.

TODO Anforderungen an ein verteiltes Dateisystem?

4.3.1 NFS

Das verteilte Dateisystem Network File System wurde von Sun Microsystem entwickelt. Das grundlegende Prinzip von NFS ist, dass jeder Dateiserver eine standardisierte Dateischnittstelle implementiert und über diese Dateien des lokalen Speichers, den Benutzern zur Verfügung stellt. Das bedeutet, dass es keine Rolle spielt, welches System dahinter steht. Ursprünglich wurde es für UNIX Systeme entwickelt. Mittlerweile gibt es aber Implementierungen für verschiedenste Betriebssysteme (siehe Tanenbaum and Steen 2003 S. 645ff.).

NFS ist also weniger ein Dateisystem als eine Menge von Protokollen, die in der Kombination mit den Clients ein verteiltes Dateisystem ergeben. Die Protokolle wurden so entwickelt, dass unterschiedliche Implementierungen einfach zusammenarbeiten können. Auf diese Weise können durch NFS eine heterogene Menge von Computern verbunden werden. Dies gilt sowohl für Benutzer, als

auch für die Serverseite (siehe Tanenbaum and Steen 2003 S. 645ff.).

TODO finde Informationen zu den einzelnen Punkten

- http://xtreemfs.org/how_replication_works.php
- http://xtreemfs.org/xtfs-guide-1.5.1/index.html#tth_sEc2.4

4.3.1.1 Architektur

4.3.1.2 Kommunikation

4.3.1.3 Synchronisierung

4.3.1.4 Replikation

4.3.1.5 Fehlertoleranz

4.3.1.6 Sicherheit

4.3.2 XtreemFS

Als Alternative zu konventionellen verteilten Dateisystemen, bietet XtreemFS eine unkomplizierte und moderne Variante eines verteilten Dateisystems. Es wurde speziell für die Anwendung in einem Cluster mit dem Betriebssystem XtreemOS entwickelt. Mittlerweile gibt es aber Server und Client Anwendungen für fast alle Linux Distributionen. Außerdem Clients für Windows und MAC.

4.3.2.1 Architektur

4.3.2.2 Kommunikation

4.3.2.3 Replikation

4.3.2.4 Sicherheit

4.3.3 Speichergeschwindigkeit

TODO überhaupt notwendig? * http://member.wide.ad.jp/~shima/publications/20120924-dfs-performance.pdf

4.4 Datenbank gestützte Dateiverwaltungen

Einige Datenbanksysteme, wie zum Beispiel MongoDB[9] bieten eine Schnittstelle an, um Dateien abzuspeichern. Viele dieser Systeme sind meist nur begrenzt für große Datenmengen geeignet. MongoDB und GridFS sind jedoch genau für diese Anwendungsfälle ausgelegt, daher wird diese Technologie im folgenden Kapitel genauer betrachtet.

4.4.1 MongoDB & GridFS

MongoDB bietet von Haus aus die Möglichkeit, BSON-Dokumente in der Größe von 16MB zu speichern. Dies ermöglicht die Verwaltung kleinerer Dateien ohne zusätzlichen Layer. Für größere Dateien und zusätzliche Features bietet MongoDB mit GridFS eine Schnittstelle an, mit der es möglich ist, größere Dateien und ihre Metadaten zu speichern. Dazu teilt GridFS die Dateien in Chunks einer bestimmten Größe auf. Standardmäßig ist die Größe von Chunks auf 255Byte

[9] http://docs.mongodb.org/manual/core/gridfs/

gesetzt. Die Daten werden in der Kollektion `chunks` und die Metadaten in der Kollektion `files` gespeichert.

Durch die verteilte Architektur von MongoDB werden die Daten automatisch auf allen Systemen synchronisiert. Außerdem bietet das System die Möglichkeit über Indexes schnell zu suchen und Abfragen auf die Metadaten durchzuführen.

Beispiel:

```
$mongo = new Mongo();                       // connect to database
$database = $mongo->selectDB("example");    // select mongo database

$gridFS = $database->getGridFS();           // use GridFS class for handling files

$name = $_FILES['Filedata']['name'];        // optional - capture the
                                            // name of the uploaded file
$id = $gridFS->storeUpload('Filedata', $name);
                                            // load file into MongoDB
```

Bei der Verwendung von MongoDB ist es sehr einfach Dateien in GridFS abzulegen. Die fehlenden Funktionen wie zum Beispiel ACL oder Versionierung, machen den Einsatz in Symcloud allerdings schwierig.

4.5 Performance

TODO überhaupt notwendig?

4.6 Evaluation

Am Ende dieses Kapitels, werden die Vor- und Nachteile der jeweiligen Technologien zusammengefasst. Dies ist notwendig, um am Ende ein optimales

Speicherkonzept für Symcloud zu entwickeln.

Speicherdienste wie Amazon S3 sind für einfache Aufgaben bestens geeignet, sie bieten alles an, was für ein schnelles Setup der Applikation benötigt wird. Jedoch haben gerade die Open-Source Alternativen zu S3 wesentliche Mankos, die gerade für das aktuelle Projekt unbedingt notwendig sind. Zum einen ist es bei den Alternativen die fehlenden Funktionalitäten, wie zum Beispiel ACLs oder Versionierung, zum anderen ist auch Amazon S3 wenig flexibel, um eigene Erweiterungen hinzuzufügen. Jedoch können wesentliche Vorteile bei der Art der Datenhaltung beobachtet werden.

- Rest-Schnittstelle
- Versionierung
- Gruppierung durch Buckets
- Berechtigungssystem

Verteilte Dateisysteme bieten durch ihre einheitliche Schnittstelle einen optimalen Abstraktionslayer für Datenintensive Anwendungen. Die Flexibilität, die diese Systeme verbindet, bietet sich für Anwendungen wie Symcloud an. Jedoch sind fehlende Zugriffsrechte und Versionierung ein Problem, dass auf Storageebene nicht gelöst wird. Aufgrund dessen könnte ein solches verteiltes Dateisystem nicht als Ersatz für eine eigene Implementierung, sondern lediglich als Basis hergenommen werden.

Datenbankgestützte Dateiverwaltung sind für den Einsatz in Anwendungen geeignet, die die darunterliegende Datenbank verwendet. Die nötigen Erweiterungen um Dateien in eine Datenbank zu schreiben, sind aufgrund der Integration sehr einfach umzusetzen. Sie bieten eine gute Schnittstelle, um Dateien zu verwalten. Die fehlenden Möglichkeiten von ACL und Versionierung macht jedoch die Verwendung von GridFS sehr schwierig. Für

einen geeigneten Storage wäre ein chunking der Daten hilfreich, um auch Teile einer Datei effizient zu laden.

Da aufgrund verschiedenster Schwächen keine der Technologien eine adäquate Lösung für die Datenhaltung in Symcloud bietet, wird im nächsten Kapitel versucht ein optimales Speicherkonzept für das aktuelle Projekt zu entwickeln.

5 Konzept für Symcloud

TODO nur Notizen Dieses Kapitel befasst sich mit der Konzeption des Konzeptes für Symcloud. Symcloud wird als eigenständige Server-Applikation gestaltet. Es baut auf einer verteilten Datenbank RIAK auf und ist dadurch für sich gesehen sehr sicher. Als Ganzes gesehen, fungiert Symcloud als eigenständiges Storage-System. Bei der Verbindung mit anderen Symcloud-Servern fungiert dieses Netzwerk als verteiltes System, mit dem es möglich ist, konfigurierbare Replikationen von Nutz- und Metadaten durchzuführen. Konzeptionell gibt es bei Symcloud auch eine Suchmaschine, mit der es möglich sein sollte, Daten verteilt zu suchen.

Die Sulu-Oberfläche wird über eine Rest-Schnittstelle mit Symcloud kommunizieren. Bestenfalls könnten Benutzerdaten über den `UserProvider` von Sulu mitverwendet werden. Mittels Cookie Authentifizierung könnte JavaScript direkt mit der Schnittstelle kommunizieren ohne sich erneut anzumelden.

Ebenfalls denkbar wäre auch eine Authentifizierung mittels Key und Secret, wie es auch bei Amazon S3 implementiert wurde. Diese Kompatibilität könnte auch auf die REST-Schnittstelle ausgeweitet werden.

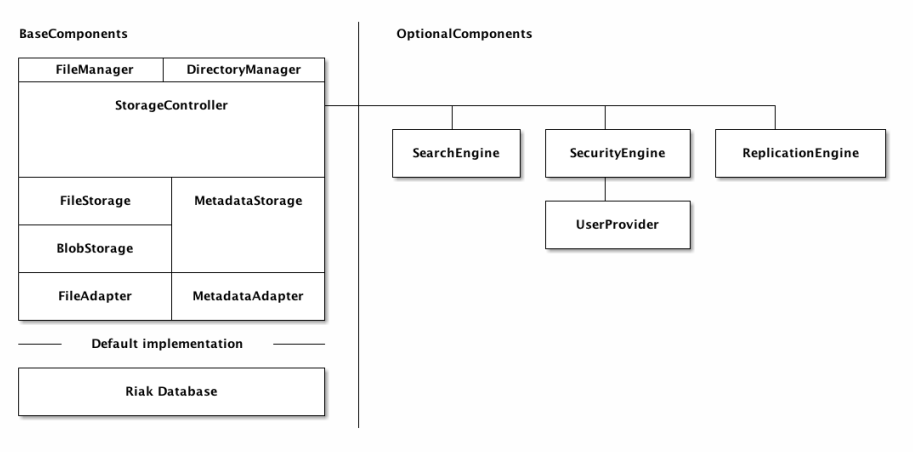

Abbildung 6: Architektur für "Symcloud-DistributedStorage"

5.1 Überblick

Die Architektur ist gegliedert in Kern-Komponenten und optionale Komponenten. In der Abbildung 6 ist die Kopplung der Komponenten zu erkennen. Die Schichten sind jeweils über ein Interface entkoppelt und ein Adapter entkoppelt den jeweiligen Datenbankzugriff.

Im Rahmen dieser Arbeit entstand eine Beispiel Implementierung mit der verteilten Datenbank Riak für die Speicherung aller Informationen.

5.1.1 PHP Stream & Rest API

TODO nur Notizen Eventuell Zugriff über PHP Stream wenn die Library in die Applikation eingebettet ist oder eine Rest-API, falls dies als eigenständige Applikation (long running process) umgesetzt wird. Die REST-API könnte, bis zu einem gewissen grade, kompatibel zu der S3 Schnittstelle sein.

5.1.2 StorageController

TODO nur Notizen Zentrale Zugriffsschnittstelle

5.1.3 SecurityController

TODO nur Notizen Bearbeitet und überprüft Datei Berechtigungen.

5.1.4 Metadaten Storage

TODO nur Notizen Entweder auch auf RIAK oder MySQL je nach dem wo unstrukturierte Daten abgelegt werden. Hier werden die Daten zum File abgelegt. Zum einen die Struktur / ACL / Größe / Name / Replikationen (auf welchen Servern) / ... Strukturelle Zugriffe wie `ls` / `mkdir` / `prop` gehen nur über diese Schnittstelle und die Daten müssen nicht gelesen werden.

Später können hier unstrukturierte Daten wie Titel / Beschreibung / Referenzen und Includes (aus xanadu) zusätzlich abgelegt werden.

"Buckets" (evtl. anderer Name) dienen zur Gruppierung. Diese Gruppierungen können gemeinsame Optionen und Benutzerrechte besitzen. Benutzerrechte auf einzelne Objekte ist nicht vorgesehen, da es nicht in den Anforderungen benötigt wird.

5.1.5 FileStorage

TODO nur Notizen Eventuell Speicherkonzept aufbauend auf einem Blob Storage. Dateien werden in z.b. 8MB große Blöcke geteilt und anhand ihres Hash-wertes in eine Datei geschrieben. Der Hash fungiert hier als eine Art ID. Diese Daten könnten dann in einer Objekt-Datenbank wie RIAK gespeichert werden. Alternativ wäre auch eine Speicherung auf dem Filesystem möglich.

Dies könnte durch XtreemFS ebenfalls verteilt aufgebaut sein. Ein zusätzliches Objekt mit einem Array aus Blob-IDs würde dann eine Datei darstellen. Diese bekäme dann die den Hash-wert der Datei als seine ID.

Diese Schicht übernimmt auch die Versionierung der Dateien. Dies geschieht im Zusammenspiel mit dem Metadaten-Storage, da die Informationen zu einer Version ebenfalls dort abgelegt werden.

Ein mögliches Datenmodell wäre das von GIT.

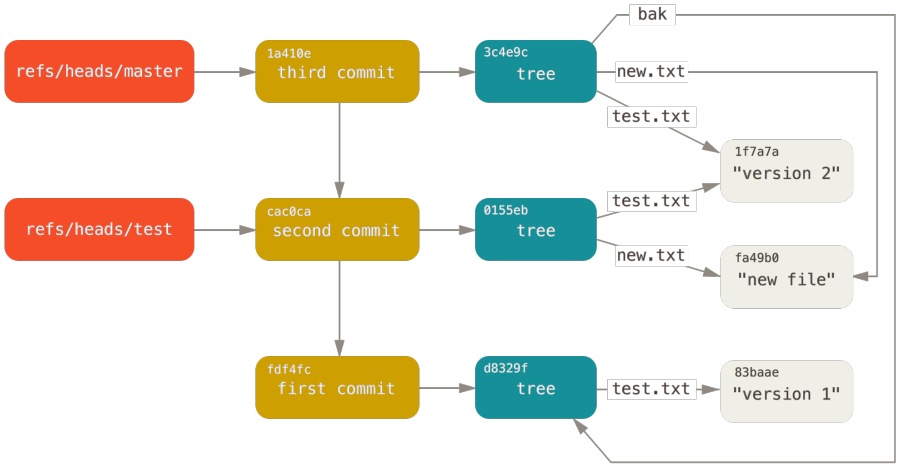

Abbildung 7: Git Datenmodell [Quelle http://git-scm.com/book/it/v2/Git-Internals-Git-References]

5.2 Zusammenfassung

Symcloud verwendet nicht nur ein verteiltes System (RIAK), es ist auch selbst ein verteiltes System, das die Metadaten/Nutzdaten trennt und diese über eine Schnittstelle zur Verfügung stellt. Zusätzlich bietet es die Möglichkeit Replikationen zu konfigurieren und eine Suche über alle Verknüpften Server durchzuführen.

Es kann pro Bucket festgelegt werden, welcher Benutzer Zugriff auf diesen hat bzw. ob er diese durchsuchen darf. Dies bestimmt die Einstellungen des Replikators, der die Daten anhand dieser Einstellungen über die verbundenen Instanzen verteilt.

Beispiel:

- Bucket 1 hat folgende Policies:
- SC1 User1 gehört der Bucket
- SC2 User2 hat Leserechte
- SC3 User3 hat Lese- und Schreibrechte

Der Replikator wird nun folgendermaßen vorgehen.

1. Die Metadaten des Buckets werden auf die Server SC2 und SC3 repliciert.
2. Die Nutzdaten (aktuellste Version) des Buckets werden auf den Server SC3 repliciert und aktuell gehalten.
3. Beides wird automatisch bei Änderungen durchgeführt.
4. Beim lesen der Datei wird SC2 bei SC1 oder SC3 (je nach Verfügbarkeit) die Daten holen und bei sich persistieren. Diese Kopie wird nicht automatisiert von SC3 upgedated, sie wird nur bei Bedarf aktualisiert.
5. Bei Änderung einer Datei des Buckets auf SC3 werden die Änderungen automatisch auf den Server S1 gespielt.

Die Suchschnittstelle wird bei der Suche nach Dateien für den User2 oder User3 auf das Bucket durchsuchen. Jedoch wird der User3 die Daten in seinem eigenen Server suchen und nicht bei S1 nachfragen. Da S2 nicht immer aktuelle Daten besitzt, setzt er bei der Schnittstelle S1 eine Anfrage ab, um die Suche bei sich zu Vervollständigen.

Dieses Konzept vereint die größten Vorteile, die im vorherigen Kapitel beschrieben wurden.

6 Datenmodel

TODO nur Notizen In diesem Kapitel wird das Datenmodel für die Datenhaltung behandelt. Es sollte alle Anforderungen an das Projekt erfüllen, um eine optimale und effiziente Datenhaltung zu gewährleisten.

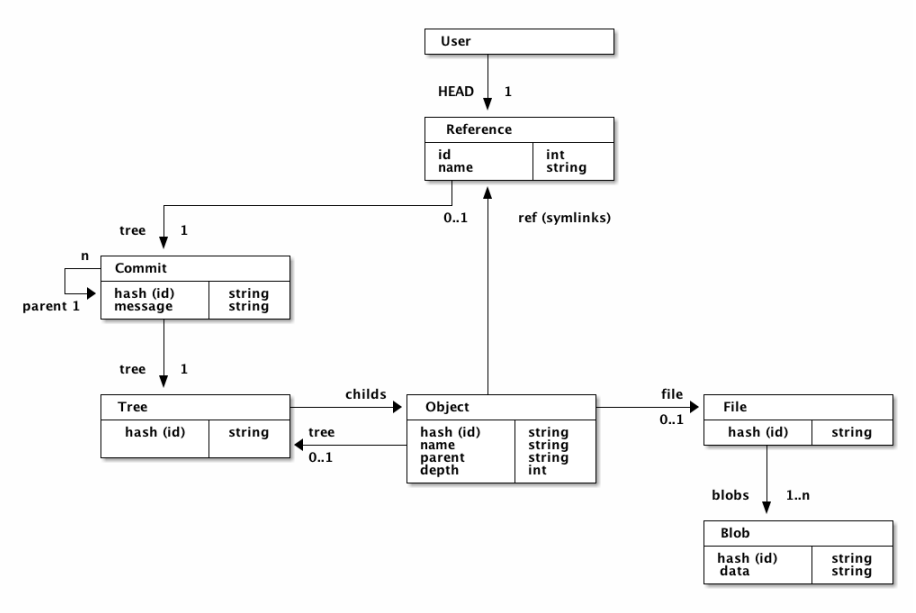

Abbildung 8: Datenmodel für “Symcloud-DistributedStorage”

Das Datenmodell ist an GIT angelehnt. Details zu diesem Model kann in der Abbildung 8 begutachtet werden.

6.1 Exkurs: GIT

GIT[10] ist ein verteilte Versionsverwaltung, das ursprünglich entwickelt wurde, um den Source-Code des Linux Kernels zu verwalten.

Abbildung 9: GIT-Logo

Die Software ist im Grunde eine Key-Value Datenbank. Es werden Objekte in Form einer Datei abgespeichert, in dem jeweils der Inhalt des Objekts abgespeichert wird. Der Name der Datei enthält den Key des Objektes. Dieser Key wird berechnet indem ein sogenannter SHA berechnet wird. Der SHA ist ein mittels Secure-Hash-Algorith berechneter Hashwert der Daten. Das Listing **??** zeigt, wie ein SHA in einem Terminal berechnet werden kann.

```
$ OBJECT='blob 46\0{"name": "Johannes Wachter", "job": "Web-Developer"}'
$ echo -e $OBJECT | shasum
6c01d1dec5cf5221e86600baf77f011ed469b8fe -
```

Im Listing **??** wird ein GIT-Objekt vom Typ BLOB erstellt und in den `objects` Ordner geschrieben.

```
$ OBJECT='blob 46\0{"name": "Johannes Wachter", "job": "Web-Developer"}'
$ echo -e $OBJECT | git hash-object -w --stdin
6c01d1dec5cf5221e86600baf77f011ed469b8fe
```

[10] http://git-scm.com/

```
$ find .git/objects -type f
    .git/objects/6c/01d1dec5cf5221e86600baf77f011ed469b8fe
```

Die Objekte in GIT sind immutable also nicht veränderbar. Ein einmal erstelltes Objekt wird nicht mehr aus der Datenbank gelöscht. Bei der Änderung eines Objektes wird ein neues Objekt mit einem neuen Key erstellt.

6.1.1 Objekt Typen

GIT kennt folgende Typen:

Ein BLOB repräsentiert eine einzelne Datei in GIT. Der Inhalt der Datei wird in einem Objekt gespeichert. Bei Änderungen ist GIT auch in der Lage Inkrementelle DELTA-Dateien zu speichern. Beim wiederherstellen werden diese DELTAs der Reihe nach aufgelöst. Ein BLOB besitzt für sich gesehen keinen Namen.

Der TREE beschreibt ein Ordner im Repository. Ein TREE enthält andere TREE bzw. BLOB Objekte und definiert damit eine Ordnerstruktur. In einem TREE werden auch die Namen zu BLOB und TREE Objekten festgelegt.

Der COMMIT ist ein Zeitstempel eines einzelnen TREE Objektes. Im folgenden Listing **??** ist der Inhalt eines COMMIT Objektes auf einem Terminal ausgegeben.

```
1 $ git show -s --pretty=raw 6031a1aa
2 commit 6031a1aa3ea39bbf92a858f47ba6bc87a76b07e8
3 tree 601a62b205bb497d75a231ec00787f5b2d42c5fc
4 parent 8982aa338637e5654f7f778eedf844c8be8e2aa3
```

```
5 author Johannes Wachter <johannes.wachter@massiveart.at> 1429190646 +0200
6 committer Johannes Wachter <johannes.wachter@massiveart.at> 1429190646 +0200
7
8     added short description gridfs and xtreemfs
```

Das Objekt enthält folgende Werte:

Zeile	Name	Beschreibung
2	commit	SHA des Objektes
3	tree	TREE-SHA des Stammverzeichnisses
4	parent(s)	Ein oder mehrere Vorgänger
5	author	Verantwortlicher für die Änderungen
6	committer	Ersteller des COMMITs
8	comment	Beschreibung des COMMITs

Anmerkungen:

- Ein COMMIT kann mehrere Vorgänger haben wen sie zusammengeführt werden. Zum Beispiel würde dies bei einem MERGE verwendet werden.
- Der Autor und Ersteller des COMMITs können sich unterscheiden, wenn zum Beispiel ein Benutzer einen PATCH erstellt ist er der Verantwortliche für die Änderungen. Der Benutzer, der den Patch nun auflöst und den `git commit` Befehl ausführt, ist der Ersteller.

Eine REFERENCE ist ein Verweis auf einen bestimmten COMMIT. Aufbauend auf diese Verweise, ist das Branching-Model von GIT aufgebaut.

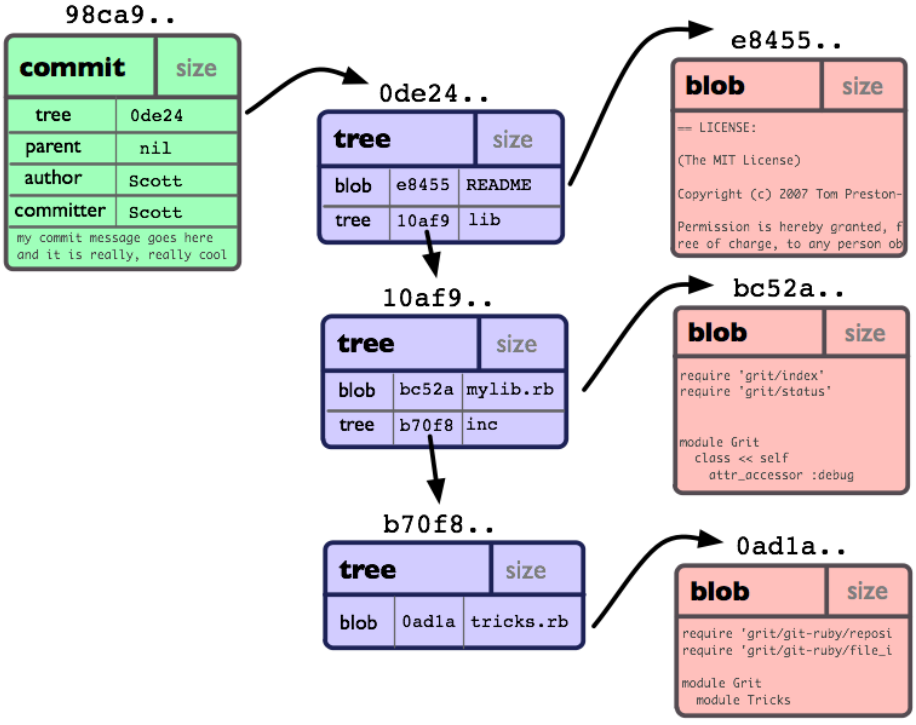

Abbildung 10: Beispiel eines Repositories (Chacon 2015)

6.1.2 Anforderungen

Das Datenmodell von GIT erfüllt folgende Anforderungen von Symcloud:

Versionierung ??? (commits)

Buckets ??? (referenzen)

Symlinks ??? (referencen)

7 Implementierung

In diesem Kapitel werden die einzelnen Komponenten, die für Symcloud entwickelt wurden, genauer betrachtet.

7.1 OAuth2

Für die Authentifizierung wurde das Protokoll OAuth in der Version 2 implementiert. Dieses offene Protokoll erlaubt eine standardisierte, sichere API-Autorisierung für Desktop, Web und Mobile-Applikationen. Initiiert wurde das Projekt von Blaine Cook und Chris Messina. ("OAuth – Wikipedia, Die Freie Enzyklopädie" 2015)

Der Benutzer kann einer Applikation den Zugriff auf seine Daten autorisieren, die von einer andere Applikation zur Verfügung gestellt wird. Dabei werden nicht alle Details seiner Zugangsdaten preisgegeben. Typischerweise wird die Weitergabe eines Passwortes an Dritte vermieden. ("OAuth – Wikipedia, Die Freie Enzyklopädie" 2015)

7.1.1 Begriffe

In OAuth2 werden folgende vier Rollen definiert:

Resource owner Besitzer einer Ressource, die er für eine Applikation bereitstellen will.

Resource server Der Server, der die Geschützen Ressourcen verwaltet. Er ist in der Lage Anfragen zu akzeptieren und die geschützten Ressourcen zurückzugeben, wenn ein geeignetes und valides Token bereitgestellt wurde.

Client Die Applikation stellt Anfragen, im Namen des Ressourceneigentümers, an den sesource server. Sie holt sich vorher die Genehmigung zu diesen geschützten Ressourcen.

Authorization server Der Server, der Zugriffs-Tokens, nach der erfolgreichen Authentifizierung des Ressourceneigentümers, bereitstellt.

Die Interaktion zwischen "Resource server" und "Authorization server" ist nicht spezifiziert. Der Autorisierungsserver und Ressourcenserver können auf dem selben Server bzw. in der selben Applikation betrieben werden. Eine andere Möglichkeit wäre es, dass die beiden Server auf verschiedenen Server zu betreiben. Ein Autorisierungsserver kann auch Zugriffstoken für mehrere Ressourcenserver bereitstelle. (Hardt 2012, Seite 5)

7.1.2 Protokoll Ablauf

Der Ablauf einer Autorisierung (Hardt 2012, Seiten 6 ff) mittels Oauth2, der in der Abbildung 11 abgebildet ist, enthält folgende Schritte:

A) Der Client fordert die Genehmigung des "Resource owner". Diese Anfrage kann direkt an den Benutzer gemacht werden (wie in der Abbildung

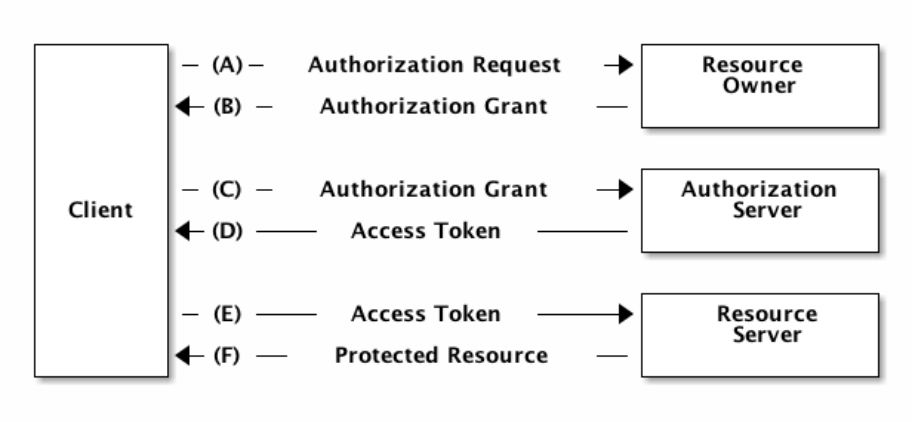

Abbildung 11: Ablaufdiagramm des OAuth

dargestellt) oder vorzugsweise indirekt über den "Authorization server" (wie zum Beispiel bei Facebook).

B) Der Client erhält einen "authorization grant". Er repräsentiert die Genehmigung des "Resource owner" die geschützten Ressourcen zu verwenden.

C) Der Client fordert einen Token beim "Autorization server" mit dem "authorization grant" an.

D) Der "Autorization server" authentifiziert den Client, validiert den "authorization grant" und gibt einen Token zurück.

E) Der Client fordert eine geschützte Ressource und autorisiert die Anfrage mit dem Token.

F) Der "Resource server" validiert den Token, validiert ihn und gibt die Ressource zurück.

7.1.3 Anwendung

OAuth2 wird verwendet um es externen Applikationen zu ermöglichen auf die Dateien der Benutzer zuzugreifen. Der Sync-Client verwendet diese Authorizie-

rungsmöglichkeit um Dateien des Benutzers zu synchronisieren.

8 Dokumentation

Dieses Kapitel enthält eine kurze Dokumentation wie Symcloud installiert und deployed werden kann. Es umfasst eine einfache Methode auf einem System und ein verteiltes Setup (sowohl RIAK als auch Symcloud).

9 Ausblick

Welche Teile des Konzeptes konnten umgesetzt werden und wie gut funktionieren diese?

Anhang

Literaturverzeichnis

"Amazon S3." 2015. http://aws.amazon.com/de/s3/.

"Amazon S3 and EC2 Performance Report – How Fast Is S3." 2009. https://hostedftp.wordpress.com/2009/03/02/.

Basho Technologies, Inc. 2015. "Riak CS." http://docs.basho.com/riakcs/latest/.

Chacon, Scott. 2015. "Git Book - The Git Object Model." http://schacon.github.io/gitbook/1_{t}he$_{g}$it$_{o}$bject$_{m}$odel.ht

"Core API Dokumentation." 2015. https://www.dropbox.com/developers/core/docs.

Coulouris, G.F., J. Dollimore, and T. Kindberg. 2003. *Verteilte Systeme: Konzepte Und Design*. Informatik - Pearson Studium. Pearson Education Deutschland. http://books.google.at/books?id=FfsQAAAACAAJ.

Hardt, Dick. 2012. "The OAuth 2.0 Authorization Framework."

"OAuth – Wikipedia, Die Freie Enzyklopädie." 2015. http://de.wikipedia.org/wiki/OAuth.

"Object Key and Metadata." 2015. http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingMetadata.html.

ownCloud. 2015. *OwnCloud Architecture Overview*. https://owncloud.com/de/owncloud-architecture-overview.

"Owncloud Features." 2015. https://owncloud.org/features.

Seidel, Udo. 2013. "Dateisystem-Ueberblick." *Linux Magazin*.

"Server2Server - Sharing." 2015. https://www.bitblokes.de/2014/07/server-2-server-sharing-mit-der-owncloud-7-schritt-fuer-schritt.

Tanenbaum, A.S., and M. van Steen. 2003. *Verteilte Systeme: Grundlagen Und Paradigmen*. I : Informatik. Pearson Education Deutschland GmbH. https://books.google.at/books?id=qXGnOgAACAAJ.

"Using Versioning." 2015. http://docs.aws.amazon.com/AmazonS3/latest/dev/Versioning.html.

"Was Ist Dezentralisierung." 2015. https://diasporafoundation.org/about.

"Wie Funktioniert Der Dropbox-Service." 2015. https://www.dropbox.com/help/1968.

Wikipedia. 2015a. "Amazon Web Services - Wikipedia, Die Freie Enzyklopädie." http://de.wikipedia.org/w/index.php?title=Amazon_Web_Services&oldid=139854883.

———. 2015b. “Eucalyptus (Software) - Wikipedia, Die Freie Enzyklopädie.” http://de.wikipedia.org/w/index.php?title=Eucalyptus_(Software)&oldid=137397846.