

Couchbase

ABGABEZEITRAUM: 21.04.2016 - UNBEGRENZT

INSY, BORM/ROSC

BORSOS ROBERT, KOCSIS PATRICK

Inhaltsverzeichnis

Aufgabenstellung.....	3
Ziele.....	3
Aufgabenstellung	3
Arbeitsaufteilung.....	4
Installation	6
CLI Befehle.....	9
Couchbase-cli bucket-list	9
Couchbase-cli bucket-create	9
Couchbase-cli bucket-delete	10
Couchbase-cli bucket-edit	10
Couchbase-cli bucket-flush.....	10
cbtransfer.....	10
Die wichtigsten CRUD Befehle sind folgende:	11
Befehle für Data Bucket.....	11
Befehl für die View	11
API Installation und Verwendung	11
Installation.....	11
Connection.....	11
CRUD	12
Create	12
Read	12
Update	12
Delete.....	13
Dokumentenstruktur (JSON, GSON)	13
Erläutern der Indizierung und des Map/Reduce Vorgangs	13
Erstellung und Verwendung von Views	14
Production View (Development).....	14
Format.....	15
View schreiben.....	15
View lesen	15
Quellenangaben	16
GitHub.....	16

Aufgabenstellung

Das Wissen über Couchbase erweitern.

Ziele

Ziel ist es, ein Nachschlagewerk für die zukünftige Verwendung von Couchbase zu schaffen.

Aufgabenstellung

Protokollieren Sie die einzelnen Schritte zur Installation und Inbetriebnahme sowie die Verwendung einer API (Java, PHP, Python, Node.js oder C) mit Couchbase. Gehen Sie dabei näher auf das Dokumentenformat und die Abfrage (Views) der Daten ein. Verwenden Sie dabei auch die CLI um auch in der Konsole mit Couchbase arbeiten zu können.

Folgende Eckpunkte sollen enthalten sein:

- Installation
- CLI Befehle
- API Installation und Verwendung
- Dokumentenstruktur (JSON, GSON)
- Erläutern der Indizierung und des Map/Reduce Vorgangs
- Erstellung und Verwendung von Views

Arbeitsaufteilung

Die Erstellung des Protokolls wurde in Aufgabenbereiche unterteilt, jede Person muss Bestimmte Teilbereiche ausarbeiten. Natürlich wird dann aber das ganze Protokoll von beiden überprüft.

Borsos

Aufgaben	geschätzter Zeitaufwand (in Minuten)	tatsächlicher Zeitaufwand (in Minuten)
Installation	30	24
CLI Befehle	50	51
API Installation und Verwendung	50	27
Dokumentenstruktur (JSON, GSON)	70	85
Summe	200	187

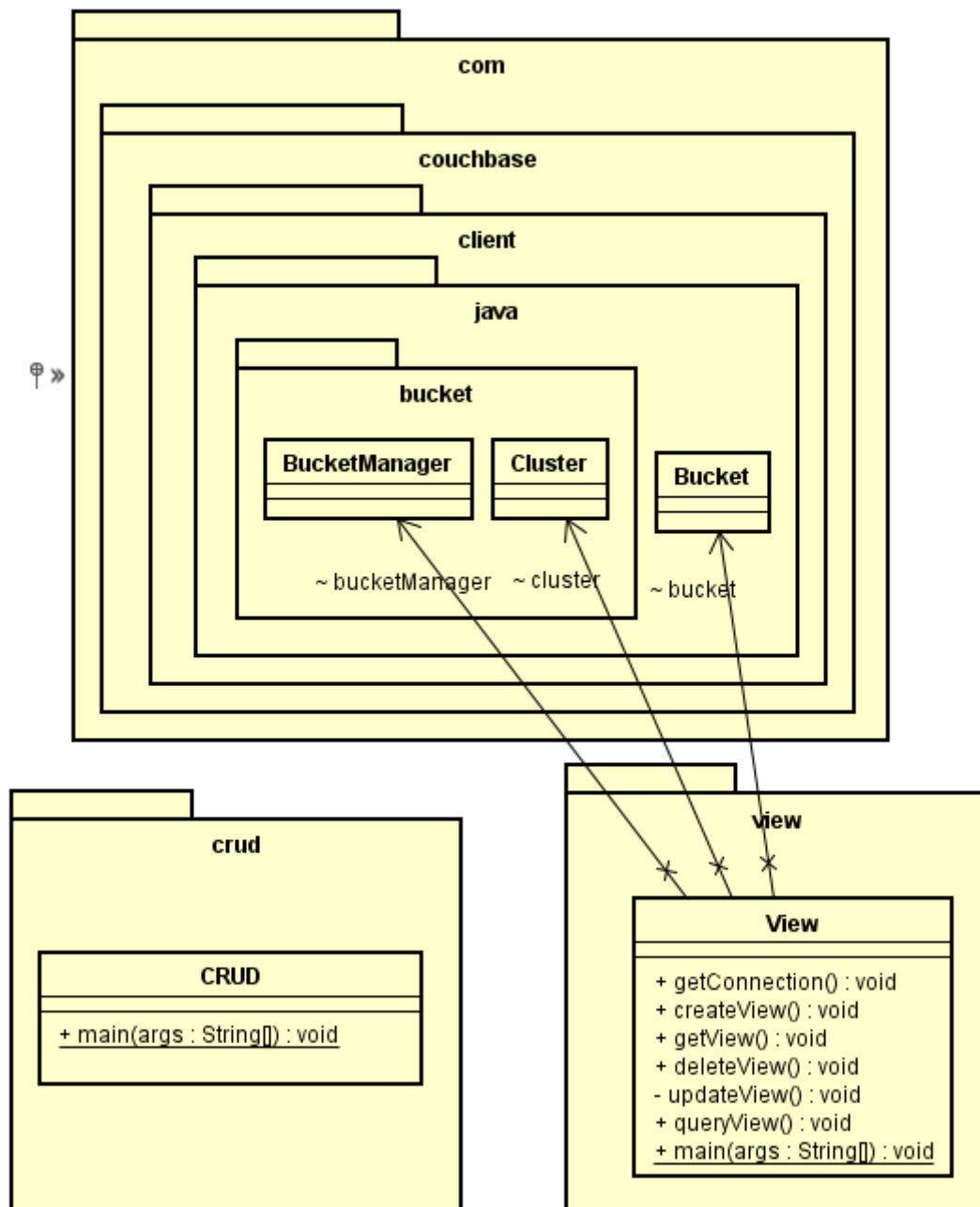
Kocsis

Aufgaben	geschätzter Zeitaufwand (in Minuten)	tatsächlicher Zeitaufwand (in Minuten)
Installation	30	20
Erläutern der Indizierung und des Map/Reduce Vorgangs	120	135
Erstellung und Verwendung von Views	70	68
Summe	220	205

Gesamt Arbeitszeit

	geschätzter Zeitaufwand (in Minuten)	tatsächlicher Zeitaufwand (in Minuten)
Gesamt	440	392

UML



Installation

Installiert wird Couchbase in diesem Fall in eine VM. Hierbei wird ein grafisches Betriebssystem benötigt, weil Couchbase über eine grafische Oberfläche verwaltet wird.

Nun um Couchbase zu installieren öffne man ein Terminal (CLI) Fenster und ladet Couchbase wie folgt in ein Verzeichnis

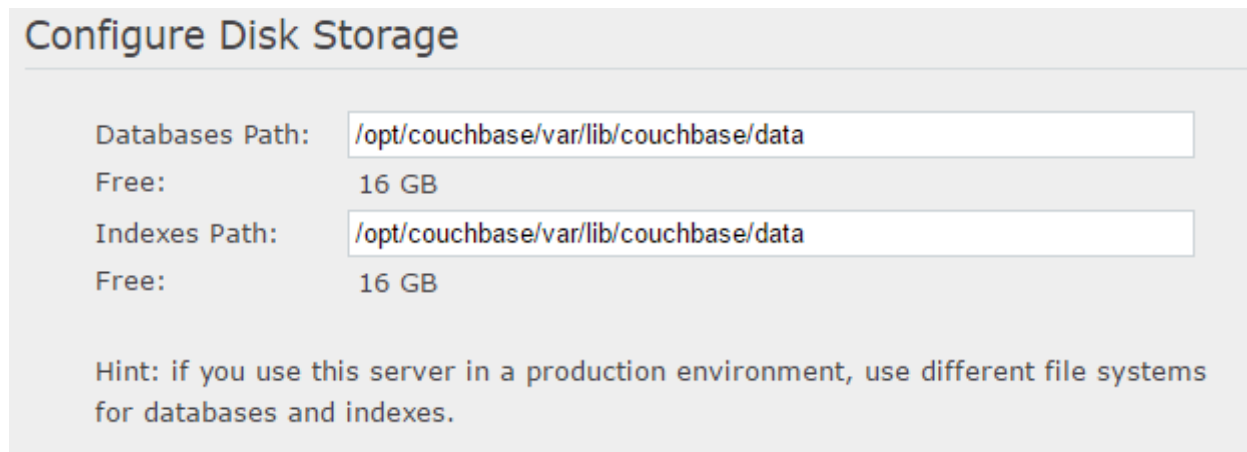
```
wget http://www.couchbase.com/get-started-developing-nosql#download_form
```

Danach wird mittels dem Befehl dpkg Couchbase installiert: (Achtung bei der VM: Couchbase benötigt mindestens 4 Prozessor Kerne und 4 GB RAM)

```
dpkg -i couchbase-server-enterprise_4.5.0-beta-debian8_amd64.deb
```

Nun kann man die grafische Oberfläche von Couchbase im Browser mit `server_ip:8091` aufrufen.

Beim ersten Aufruf nach der Installation muss man zuerst den Server konfigurieren. Dazu wird der Speicherpfad für die Datenbank angegeben. Bei größeren Projekten sollten die Speicherorte unterschiedlich sein (Abbildung 1).



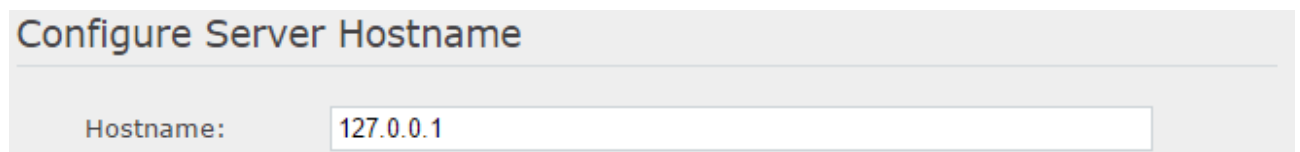
Configure Disk Storage

Databases Path:
Free: 16 GB

Indexes Path:
Free: 16 GB

Hint: if you use this server in a production environment, use different file systems for databases and indexes.

Danach wird 127.0.0.1 als Localhost beim Hostname angegeben (Abbildung 2).



Configure Server Hostname

Hostname:

Im letzten Punkt wird ausgewählt ob der Server zu einem bereits bestehenden Cluster gehört oder ein neues Cluster erstellt werden soll.

Cluster:

Ein Cluster ist eine Zusammenfassung von mehreren Nodes. Alles innerhalb eines Clusters sind vollkommen ident in folgenden Arten:

- Funktion
- Interface

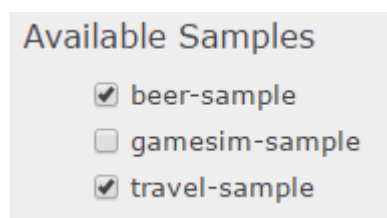
- Components
- Systems

Alle Änderungen an einzelnen Nodes in einem Cluster werden überall geändert, sodass keine parent/child Beziehung entstehen kann.

Buckets

Buckets sind isolierte, virtuelle Container, welche eine logische Gruppe bilden. Im Vergleich mit Relationalen Datenbanken sind Buckets die einzelnen Datenbanken. Bei einem neuen Cluster kann ausgewählt werden ob der Server nur für die Daten oder auch für Indizes und Querys zuständig ist.

Danach kann man zwischen einigen sample Buckets aussuchen (optional):



Available Samples

- ☒ beer-sample
- ☐ gamesim-sample
- ☒ travel-sample

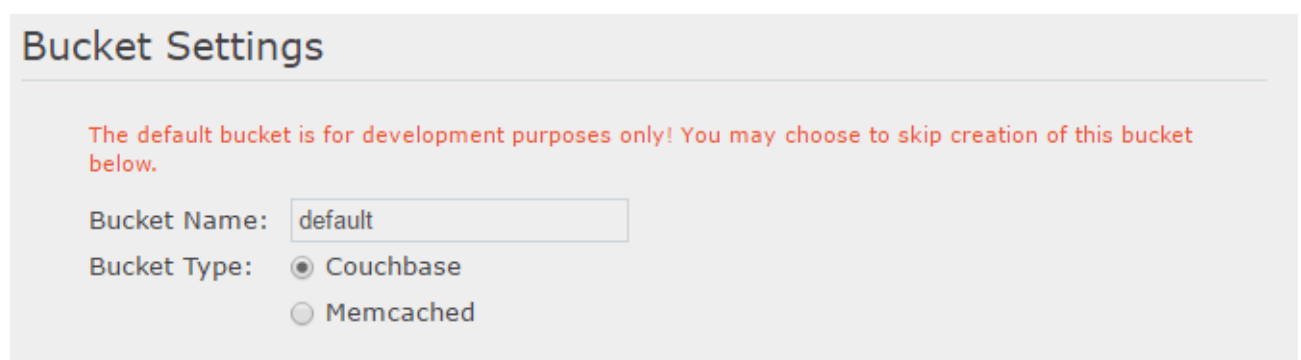
Danach kann man zwischen 2 Bucket Typen ausgesucht werden:

Couchbase:

Bietet hochverfügbare und dynamisch konfigurierbaren verteilte Datenspeicherung, mit Ausdauer und Replikationsdienste. Von Couchbase abgeleiteter JSON-basierter Dokument Store mit Memcached-kompatiblen Interface.

memchaed:

In-memory Key-Value Store, ursprünglich entwickelt für Caching.



Bucket Settings

The default bucket is for development purposes only! You may choose to skip creation of this bucket below.

Bucket Name:

Bucket Type: ☒ Couchbase ☐ Memcached

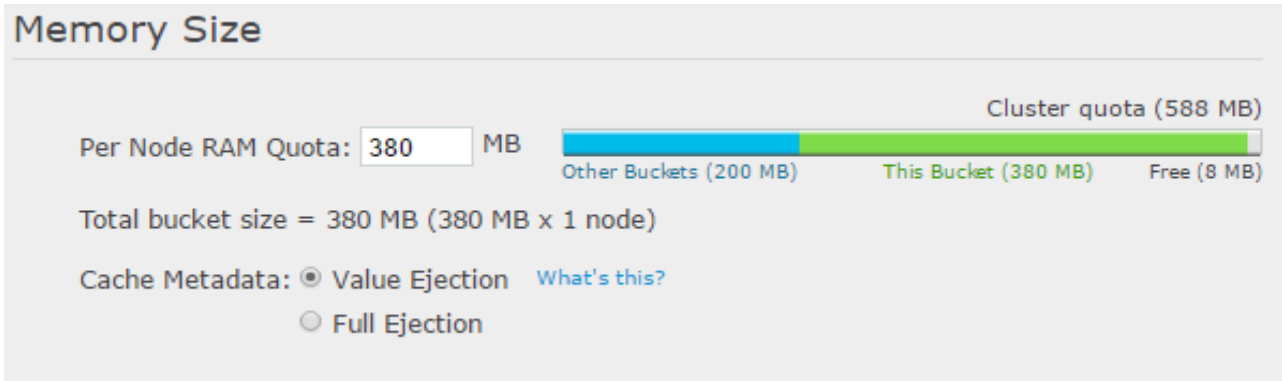
Es wird der RAM per Node eingetragen was eine zusammenfassend Cluster heißt. Danach wird das Cache Metadata bestimmt:

Value Ejection:

Entfernt die Daten aus dem Cache und behält alle Keys.

Full Ejection

Löscht alle Daten inclusive der Keys, metadata und key-values aus dem Cache.



Weiteres wird dann die Anzahl der Backups festgelegt.

Nun kann man die I/O optional optimieren.

Damit hier das Löschen für das Gesamte Bucket möglich ist, muss die Flushfunktion zu „enabled“ werden.

Replicas

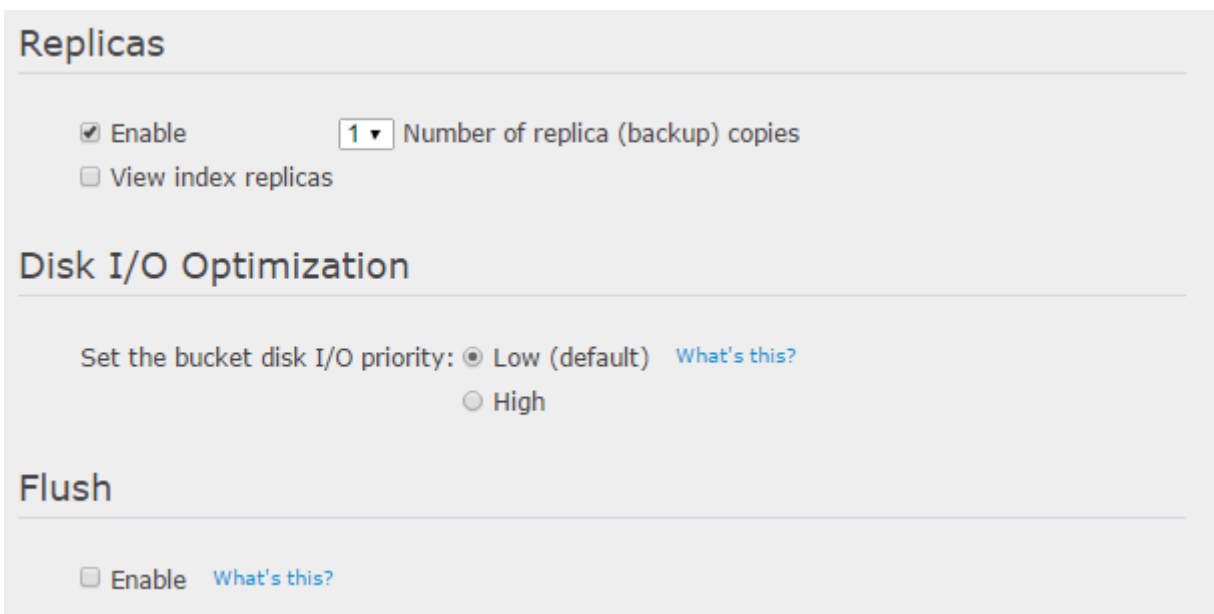
☒ Enable Number of replica (backup) copies
☐ View index replicas

Disk I/O Optimization

Set the bucket disk I/O priority: ☒ Low (default) [What's this?](#)
☐ High

Flush

☐ Enable [What's this?](#)

The figure shows three sections of a configuration interface. The 'Replicas' section has a checked 'Enable' checkbox, a dropdown menu set to '1', and a 'View index replicas' checkbox. The 'Disk I/O Optimization' section has a radio button set to 'Low (default)' with a 'What's this?' link, and an unselected 'High' radio button. The 'Flush' section has an unselected 'Enable' checkbox and a 'What's this?' link.

CLI Befehle

Das Couchbase-CLI wird mit der normalen Couchbase-Installation mitgeliefert, jedoch steht das Programm (oder ein Link) nicht im PATH. Die Tools sind nach der Installation des Couchbase Servers unter `/opt/couchbase/bin` zu finden. Fast alle Operationen können in der GUI (Webinterface) sowie in dem CLI durchgeführt werden.

Nun werden einige der wichtigsten Bucket Operationen gezeigt.

Couchbase-cli bucket-list

@bucket-list

Gibt eine Liste mit allen Buckets (und deren Konfiguration) in einem Cluster zurück.

```
couchbase-cli bucket-list -c [host]:8091 -u [admin] -p [password]
```

Zum Beispiel:

```
/opt/couchbase/bin/couchbase-cli bucket-list -u root -p rootroot -c localhost:8091
```

Möglicher Output:

```
beer-sample
bucketType: membase
authType: sasl
saslPassword:
numReplicas: 1
ramQuota: 104857600
ramUsed: 53885632
default
bucketType: membase
authType: sasl
saslPassword:
numReplicas: 1
ramQuota: 314572800
ramUsed: 49213176
gamesim-sample
bucketType: membase
```

Couchbase-cli bucket-create

@bucket-create

Mit diesem Befehl kann ein bestimmter Bucket in einem Cluster erstellt werden:

```
couchbase-cli bucket-create -c [host]:8091 -u [admin] -p [password]
    --bucket=[bucket-name]
    --bucket-eviction-policy=[valueOnly | fullEviction]
    --bucket-type=[type]
    --bucket-port=[port]
    --bucket-ramsize=[size]
    --bucket-priority=high
    --bucket-replica=[replicas]
```

```
--enable-flush=[0 | 1]
--wait
```

Couchbase-cli bucket-delete

@bucket-delete

Löschen eines vorhandenen Buckets in einem bestimmten Cluster.

```
couchbase-cli bucket-delete -c [host]:8091 -u [admin] -p [password]
--bucket=[bucket-name]
```

Couchbase-cli bucket-edit

@bucket-edit

Hiermit kann die Konfiguration eines bestimmten Buckets bearbeitet werden (e.g. Name, RAM Quota, ...).

```
couchbase-cli bucket-edit -c [host]:8091 -u [admin] -p [password]
--bucket=[bucket-name]
--bucket-eviction-policy=[valueOnly | fullEviction]
--bucket-port=[port]
--bucket-ramsize=[ramsizeMB]
--enable-flush=[0 | 1]
```

Couchbase-cli bucket-flush

@bucket-flush

Das Bucket kann hierbei geflusht werden was heißt, dass alle Daten eines Buckets gelöscht werden. Bei memcached-Buckets werden sie nur für das Entfernen markiert, bei Couchbase-Buckets werden die Daten sofort gelöscht.

```
couchbase-cli bucket-flush -c [host]:8091 -u [admin] -p [password]
--bucket=[bucket-name]
--enable-flush
--force
```

Zum Beispiel: Leeren des default-Buckets:

```
/opt/couchbase/bin/couchbase-cli bucket-flush -c localhost:8091 -u root -p rootroot --
bucket=default
```

Running this command will totally PURGE database data from disk. Do you really want to do it? (Yes/No)Yes

Database data will be purged from disk ...
SUCCESS: bucket-flush

cbtransfer

@cbtransfer

Tool, für den Transfer von Daten von URL/File zu URL/File. cbtransfer wird von cbbackup und cbrestore verwendet. Es werden sehr viele Optionen unterstützt, beispielsweise kann man die Daten auch als CSV exportieren.

```
cbtransfer [options] source destination
```

Zum Beispiel: Kopieren von Daten aus Bucket in File(s)

```
/opt/couchbase/bin/cbtransfer couchbase://localhost:8091 -b beer-
sample/home/rborsos/beer-sample
[#####] 100.0% (7303/estimated 7303 msgs)
bucket: beer-sample, msgs transferred...
:      total |   last |  per sec
byte :      2541549 | 2541549 | 1027257.5
done
```

Somit wurden 7303 Dokumente erfolgreich übertragen.

Die wichtigsten CRUD Befehle sind folgende:

- *cbc help* → gibt eine Liste aller Befehle aus
- *cbc create* → erstellt ein neues Item bzw. ändert ein bestehendes
- *cbc rm* → löscht ein Item
- *cbc cat* → schreibt in den Standard Output

Befehle für Data Bucket

- *cbc bucket-create* → erstellt einen neuen Bucket
- *cbc bucket-delete* → löscht einen Bucket
- *cbc bucket-flush* → benötigt aktiviertes flush in der Datenbank

Befehl für die View

- *cbc view* → gibt eine view aus

API Installation und Verwendung

Installation

Die Java-Client-API kann

von <http://developer.couchbase.com/documentation/server/4.1/sdks/java-2.2/download-links.html>

heruntergeladen werden. Die JARs müssen im Build-Path eingebunden werden. Die Bibliotheken sind auch in einem Maven Repository verfügbar, könnten also einfach als Abhängigkeiten in der pom.xml eingetragen werden.

Connection

Die grundsätzliche Verbindung zu einem Couchbase Cluster erfolgt über die Klasse CouchbaseCluster. Es muss nur ein Teil der Nodes des Clusters übergeben werden, da die eigentliche Verbindung (also das Erstellen des Sockets) erst hergestellt wird, wenn man sich

zu einem Bucket verbindet. Die Verbindung wird in einem Cluster-Objekt gespeichert und kann über dieses verwaltet werden. @java-intro

```
Cluster cluster = CouchbaseCluster.create(„192.168.0.12“);  
Bucket buce = cluster.openBucket();
```

Mit `cluster.openBucket()` wird eine Verbindung zu einem Bucket hergestellt (es können Name des Buckets, eventuell Passwort und die Timeout-Zeit übergeben werden, ohne Parameter wird das default-Bucket verwendet).

Über `cluster.disconnect()` werden die Verbindungen zu den Buckets sowie zu dem Cluster geschlossen.

CRUD

Create

Dokumente können im `JSONDocument`-Format in einen Bucket eingefügt werden. Ein `JSONDocument` besteht aus `JSONObjects` (bzw. ein `JSONObjects-Array`). In einem Dokument werden zusätzlich noch Meta-Daten (wie ID, Ablaufzeit, CAS-Value,...) gespeichert. Ein `JSONDocument` wird über `JsonDocument.create` erstellt, wobei ID und Inhalt (also das `JSONObject`) übergeben werden.

```
JsonDocument docu = JsonDocument.create("rborsos", person1);
```

Das Dokument kann über `bucket.upsert` oder `bucket.insert` eingefügt werden. **Upsert** überschreibt ein eventuell vorhandenes Dokument (kann somit auch für „Update“ verwendet werden). **Insert** speichert das Dokument, wirft jedoch eine `DocumentAlreadyExistsException` falls der Key bereits existiert.

Read

Ein Dokument kann über `bucket.get(id)` (null falls das Dokument nicht gefunden wurde) gelesen werden. Das gelesene Dokument wird in einem `JSONDocument`-Objekt gespeichert.

Über `JSONDocument.content().get()` können Daten aus dem Dokument ausgelesen werden.

Über `bucket.getAndLock(id, locktime)` kann ein Dokument für Schreibzugriffe gesperrt werden (nicht für Lesezugriffe). Ein Dokument kann für maximal 30 Sekunden gelockt werden (15 Sekunden per default) und kann mit `unlock()` (frühzeitig) wieder freigegeben werden.

Über `bucket.getAndTouch(„id“, expiryTime)` erhält man das Dokument und es wird die Ablaufzeit erneuert (funktioniert auch mit nur `touch()` falls nur Ablaufzeit erneuert werden soll).

Update

Ein `JSONDocument` kann über `JSONDocument.content().put()` geändert werden. Über `bucket.replace()` bzw. `bucket.upsert()` wird das Dokument an den Cluster gesendet. `Replace` wirft jedoch eine `DocumentDoesNotExistException`. **Upsert** überschreibt das Dokument und fügt ein neues Dokument ein, falls das Dokument vorher nicht existierte.

```
JSONDocument loaded = bucket.get(„Robert“);
if(loaded == null) {
    System.err.println(“Document not available”);
} else {
    loaded.content().put(“age”, 18);
    JsonDocument updated = bucket.replace(loaded);
    System.out.println(“Updated:” + updated.id())
}
```

Delete

JSONDocument können über `remove()` gelöscht werden.

```
bucket.remove(„rborsos“);
```

Dokumentenstruktur (JSON, GSON)

In Couchbase können Daten entweder als JSON-Dokumente oder in einem binären Format (GSON) gespeichert werden.

Ein Dokument in Couchbase repräsentiert üblicherweise eine Instanz eines Objektes im Code des Clients. Es ist vergleichbar mit einem Datensatz, einer Zeile in einer Tabelle, in einem RDBMS. Die Attribute des Dokuments sind ähnlich wie die Spalten einer Tabelle in einem RDBMS. Jedoch gibt es kein Schema, dass die Struktur eines Dokumentes vorgibt (wie in RDBMS die Tabelle die Spalten vorgibt), jedes Dokument kann verschiedene Attribute haben. Wenn der Client zwischen verschiedenen Typen unterscheiden muss, kann man bspw. ein Attribut „Typ“ hinzufügen (Abfrage des Typens in View bietet sich an).

Dokumente können ineinander verschachtelte Strukturen (JSON: Objekte können weitere Objekte enthalten) enthalten (sog. „Sub-Dokumente“). So können Beziehungen oder Hierarchien natürlicher und kompakter als in einem RDBMS dargestellt werden.

\cite{couchbase-data-model}

Erläutern der Indizierung und des Map/Reduce Vorgangs

Ein Index ist eine Liste von Daten, die das Durchsuchen von Dokumenten effizienter macht. Dadurch müssen nicht mehr alle einzelnen Dokumente durchsucht werden, sondern es wird nur der Index durchsucht.

Eine Map/Reduce-View (oder auch nur View) erstellt einen angepassten Index, je nach Abfrage der View.

Eine View besteht aus JavaScript-Funktionen (\texttt{map} und \texttt{reduce}) die mit den Dokumenten eines Buckets arbeiten.

Views sind ausgewählte Daten aus Dokumenten. Eine View durchsucht ein Bucket und speichert jene Informationen die in der View angegeben sind in einem Index. Dabei gibt es 2 Arten von Views:

- Bei einer Development View wird nur ein Subset (d.h. ein kleiner Teil) des Ergebnisses indiziert.
- Bei einer Production View werden alle Ergebnisse zum dem Index hinzugefügt.

Die map-Funktion ist dabei für die Auswahl der Daten zuständig. Das Ergebnis der map-Funktion kann über die reduce-Funktion zusammengefasst bzw. „reduziert“ werden. Die reduce-Funktion ist optional und kann über Funktionen wie bspw. count, sum, oder avg (von Couchbase bereits definiert) Ergebnisse auf genau einen Wert reduzieren.

Es können auch eigene Reduce-Funktionen verwendet werden. @writing-views Dabei ist zu beachten, dass eine Reduce-Funktion aus mehreren sogenannten *rereduce*-Funktionen besteht. Die Ergebnisse der map-Funktion werden in mehrere „Gruppen“ eingeteilt. Auf diese einzelnen Gruppen wird anschließend die Reduce-Funktion angewandt und die Ergebnisse in einem B-Baum gespeichert. Dieser Vorgang wird solange wiederholt, bis ein einzelnes Ergebnis vorhanden ist.

Falls sich ein Dokument, auf das sich eine Abfrage bezieht, ändert bzw. ein neues Dokument hinzugefügt wird, muss neu die Datenmenge neu indiziert werden. Dabei gibt es 3 Möglichkeiten:

- Es wird sofort indiziert (nicht empfehlenswert wegen Performanceverlust)
- Es wird bei der nächsten Abfrage indiziert
- Es wird indiziert, wenn der Cluster unausgelastet ist, jedoch spätestens der nächsten Abfrage

Neben einer Map/Reduce-View gibt es auch andere Views, die Indizes erzeugen: @indexing

- Spatial Views\ enthalten nur eine Funktion (ähnlich der map-Funktion) und sind für geometrische bzw. räumliche Abfragen.
- Global Secondary Index\ (GIS) erzeugt Indizes von einem gesamten Cluster.

Erstellung und Verwendung von Views

Views generieren einen Index auf die gespeicherte Information im Databucket, und erlauben es spezifische Felder/Daten auszulesen. Durch eine View werden die kaum- oder garnicht strukturierten Daten, welche am Server gespeichert sind, auf die gesuchten Felder bzw. Daten untersucht, und es wird eine strukturierte Ansicht auf den gewünschten Output gewährleistet.

Production View (Development)

Im Back-end von Couchbase können Views erstellt werden. Zuerst muss man entscheiden ob man eine Development View oder eine Production View erstellen will.

Dabei empfiehlt es sich, solange die View entworfen und getestet wird, eine Development View zu erzeugen. Der Grund dafür ist, dass das der Aufwand des Indizierens bei schlecht entwickelten Views, die Performance des Clusters enorm beeinträchtigen kann.

Development Views, sind für Testzwecke Performance schonender. Sobald eine View ausreichend getestet wurde, kann sie dann zur Production View gepublikt werden.

Format

Die einzelnen Views werden in sogenannte Design Documents geschrieben. Auf dem Bucket können auch mehrere Design Documents (JavaScript) mit jeweils mehreren Views erstellt werden. Wichtig dabei ist, dass eine View immer nur auf Daten des Buckets zugreifen kann, in welchem das eigene Design-Dokument gespeichert ist.

View schreiben

Erläuterung an Beispiel:

Beim map function Aufruf werden Dokumente und die Metadaten über das Dokument als Parameter übergeben. Im eingebetteten emit aufruf wird dann angegeben nach was sortiert werden soll (doc.id) und welche Values ausgegeben werden sollen [doc.job, doc.age].

```
Function (doc, meta) {
  emit(doc.id, [doc.age]);
}
```

View lesen

URL:

Als URL folgendes eintippen: *localhost:8092/default/_design/dev_Test/_view/schueler*

```
{ "total_rows": 2, "rows": [
  { "id": "Robert", "key": null, "value": [17] },
  { "id": "Patrick", "key": null, "value": [19] }
]}
```

JAVA:

Mit dem folgenden Java Code wird der Inhalt unserer erstellten View gelesen:

```
ViewResult result1 = bucket.query(ViewQuery.from("dev_Test", "schueler"));
Iterator<ViewRow> iterator = result.rows();
while (iterator.hasNext()){
  ViewRow row = iterator.next();
  JsonDocument docu = row.document();
  System.out.println("" + docu.content());
}
```

```
{ "firstname": "Robert", "age": "17", "lastname": "Borsos" }
{ "firstname": "Patrick", "age": "19", "lastname": "Kocsis" }
```

Quellenangaben

<http://developer.couchbase.com/documentation/server/4.1/introduction/editions.html>

http://www.couchbase.com/get-started-developing-nosql#download_form

<http://developer.couchbase.com/documentation/server/4.1/cli/cbtransfer-tool.html>

<http://developer.couchbase.com/documentation/server/current/getting-started/installing.html>

<http://docs.couchbase.com/developer/dev-guide-3.0/concepts.html>

<http://docs.couchbase.com/admin/admin/Views/views-writing.html>

GitHub

<https://github.com/rborsos-tgm/Couchbase>