

# Get to know CouchDB

My struggle:

## My struggle:

- Oh, look another NoSQL DB!

## My struggle:

- Oh, look another NoSQL DB!
- I know HTTP!

## My struggle:

- Oh, look another NoSQL DB!
- I know HTTP!
- -> CouchDB should be easy

## My struggle:

- Oh, look another NoSQL DB!
- I know HTTP!
- -> CouchDB should be easy
- Weeks later still **no clue how it works** :(

## My struggle:

- Oh, look another NoSQL DB!
- I know HTTP!
- -> CouchDB should be easy
- Weeks later still **no clue how it works** :(
- Visited CouchDB Meetup / CouchDB Day

## My struggle:

- Oh, look another NoSQL DB!
- I know HTTP!
- -> CouchDB should be easy
- Weeks later still **no clue how it works** :(
- Visited CouchDB Meetup / CouchDB Day
- After *months* i grasped the concept



## My struggle:

- Oh, look another NoSQL DB!
- I know HTTP!
- -> CouchDB should be easy
- Weeks later still **no clue how it works** :(
- Visited CouchDB Meetup / CouchDB Day
- After *months* i grasped the concept
- Abled to use it now :)

## My struggle:

- Oh, look another NoSQL DB!
- I know HTTP!
- -> CouchDB should be easy
- Weeks later still **no clue how it works** :(
- Visited CouchDB Meetup / CouchDB Day
- After *months* i grasped the concept
- Abled to use it now :)

## My struggle:

- Oh, look another NoSQL DB!
- I know HTTP!
- -> CouchDB should be easy
- Weeks later still **no clue how it works** :(
- Visited CouchDB Meetup / CouchDB Day
- After *months* i grasped the concept
- Abled to use it now :)

Let me tell you how it works in ~40 Minutes!

## I need YOU to ...

- “Speak” HTTP fluently
- Read JSON
- Understand JavaScript

- What is CouchDB?
- Simple usage
  - Databases, Documents, CRUD
- How CouchDB works
  - B-tree, ranges, pagination
- Advanced usage
  - Views, Grouping
- Extended topics
  - More fancy stuff if time left...

Julius Beckmann

Silpion IT-Solutions

- PHP / Symfony2
- Erlang / Elixir
- NodeJS
- Continuous Everything!

# What is CouchDB?

# What is CouchDB?

- **Cluster Of Unreliable Commodity Hardware DB**
- Started 2005 by [@damienkatz](#) at Lotus Notes.
- Since 2008 Apache Project.
- Open Source: Apache Software license 2.0.



- RESTful HTTP as Interface.
- Schemaless JSON documents.
- Written in Erlang.
- Version 1.6.1 is current.
- Version 2.0 with *real clustering* on the way!



[www.meetup.com/CouchDB-Meetup-Hamburg](http://www.meetup.com/CouchDB-Meetup-Hamburg)

## Organized by

- Robert Kowalski @robinson\_k
- Andreas Wenk @awenkh
- Klaus Trainer @KlausTrainer

# Installation

## Debian/Ubuntu

```
apt-get install couchdb
```

*# 12.04 LTS: v1.0.1 | 14.04 LTS: v1.5.0 | 15.04/10: v1.6.0*

<http://wiki.ubuntuusers.de/couchdb>

## Docker image

Using this Image: [klaemo/docker-couchdb](#)

- `docker pull klaemo/couchdb:1.6.1`
- `docker run -d -p 5984:5984 --name couchdb klaemo/couchdb`

Stop and start container:

- `docker stop couchdb`
- `docker start couchdb`

# Webinterface: Futon

Apache CouchDB - Futon

localhost:5984/\_utils/

Overview

+

 Create Database ...

| Name               | Size    | Number of Documents | Update Seq |
|--------------------|---------|---------------------|------------|
| <b>_replicator</b> | 4.1 KB  | 1                   | 1          |
| <b>_users</b>      | 28.1 KB | 2                   | 7          |

Showing 1-2 of 2 databases ← Previous Page | Rows per page: 10 ▼ | Next Page →



**CouchDB**  
relax

Tools

Overview

Configuration

Replicator

Status

Documentation

Manual

Diagnostics

Verify Installation

Recent Databases

# Webinterface: Fauxton

Project Fauxton

localhost:5984/\_utils/fauxton/

**CouchDB** relax

**Databases**

Active Tasks

Config

Log

Statistics

Replication

Documentation

Verify

Login

**Databases**

API URL +

Add New Database

Database name

| Name                        | Size | # of Docs | Update Seq | Actions |
|-----------------------------|------|-----------|------------|---------|
| <a href="#">_replicator</a> | 0    | 1         | 1          |         |
| <a href="#">_users</a>      | 0    | 2         | 7          |         |

« 1 »

Fauxton on Apache CouchDB 1.6.1

## Simple usage

# Say hello to CouchDB

```
$ curl -X GET http://localhost:5984/
```

```
{  
  "couchdb": "Welcome",  
  "uuid": "460a0cc31a109fe7cdcdd17f2109e515",  
  "version": "1.6.1",  
  "vendor": {  
    "version": "1.6.1",  
    "name": "The Apache Software Foundation"  
  }  
}
```

# List Databases

```
$ curl -X GET http://localhost:5984/_all_dbs
```

```
[  
  "_replicator",  
  "_users"  
]
```



- 1 What is a Database
- 2 What is a Document
- 3 CRUD for Documents
- 4 CRUD for Attachments

# 1. What is a Database

- Single container = No tables.
- Permissions granted on database level by CouchDB.

# 1. What is a Database

- Single container = No tables.
- Permissions granted on database level by CouchDB.

## Create Database

```
$ curl -X PUT http://localhost:5984/my_database
```

```
{"ok":true}
```

# 1. What is a Database

- Single container = No tables.
- Permissions granted on database level by CouchDB.

## Create Database

```
$ curl -X PUT http://localhost:5984/my_database
```

```
{"ok":true}
```

## Remove Database

```
$ curl -X DELETE http://localhost:5984/my_database
```

```
{"ok":true}
```

# Database Metadata

```
$ curl -X GET http://localhost:5984/my_database
```

```
{
  "db_name": "my_database",
  "doc_count": 0,
  "doc_del_count": 0,
  "update_seq": 0,
  "purge_seq": 0,
  "compact_running": false,
  "disk_size": 79,
  "data_size": 0,
  "instance_start_time": "1441012193869017",
  "disk_format_version": 6,
  "committed_update_seq": 0
}
```

## 2. What are Documents

- JSON-Format.
- Unicode charset.
- Schemaless structure.
- Underscore prefix is reserved for first level keys:
  - `_id`
  - `_rev`
  - `_attachments`
  - `_deleted`

# Creating a Document

```
// test.json
{
  "hello": "world",
  "age": 42,
  "tags": ["couchdb", "is", "cool"]
}
```

# Creating a Document

```
// test.json
{
  "hello": "world",
  "age": 42,
  "tags": ["couchdb", "is", "cool"]
}
```

```
$ curl -X POST http://localhost:5984/my_database
  -H "Content-Type: application/json"
  --data-binary @test.json
```

```
{
  "ok": true,
  "id": "22d26e752e31bd1572f1d20d94000c47",
  "rev": "1-ff73263039516726e7917f24111c80ec"
}
```



- By default UUIDs.
- Must be unicode **string**.
- Will be part of URL.
- Can be *chosen* for each document.

# CouchDB IDs

- By default UUIDs.
- Must be unicode **string**.
- Will be part of URL.
- Can be *chosen* for each document.

```
$ curl -X PUT http://localhost:5984/my_database/hello-world  
-H "Content-Type: application/json"  
--data-binary @test.json
```

```
{  
  "ok":true,  
  "id":"hello-world",  
  "rev":"1-ff73263039516726e7917f24111c80ec"  
}
```

- *Multi-Version Concurrency Control.*
- Revision of document.
- 'Strictly monotonic ascending'.

- *Multi-Version Concurrency Control.*
- Revision of document.
- 'Strictly monotonic ascending'.
- Optimistic locking => lock-free access.
- Needed for **replication**.
- Initial  $i = 1$ .

```
rev = i + "-" + md5(body,attachments,deleted)  
  
"1-ff73263039516726e7917f24111c80ec"
```

# Fetching a Document

*Skipping **http://localhost:5984** in url from now on...*

```
$ curl -X GET /my_database/hello-world
```

```
{
  "_id": "hello-world",
  "_rev": "1-ff73263039516726e7917f24111c80ec",
  "hello": "world",
  "age": 42,
  "tags": ["couchdb", "is", "cool"]
}
```

# Updating a Document

```
// test_update.json
{
  "hello": "world!",
  "age": 43,
  "tags": ["couchdb", "is", "cooler"]
}
```

# Updating a Document

```
// test_update.json
{
  "hello": "world!",
  "age": 43,
  "tags": ["couchdb", "is", "cooler"]
}
```

```
$ curl -X PUT /my_database/hello-world?rev=1-ff732630...
  -H "Content-Type: application/json"
  --data-binary @test_update.json
```

```
{
  "ok": true,
  "id": "hello-world",
  "rev": "2-b9f6bbfa08e0f6a8cf3d0dcd4c153fc7"
}
```

# Deleting a Document

```
$ curl -X DELETE /my_database/hello-world?rev=2-b9f6bbfa...
```

```
{  
  "ok":true,  
  "id":"hello-world",  
  "rev":"3-e4f06ea31419b24495c94bcd952777b6"  
}
```



# Deleting a Document

```
$ curl -X DELETE /my_database/hello-world?rev=2-b9f6bbfa...
```

```
{  
  "ok":true,  
  "id":"hello-world",  
  "rev":"3-e4f06ea31419b24495c94bcd952777b6"  
}
```

```
$ curl -X GET http://localhost:5984/my_database/hello-world  
{"error":"not_found","reason":"deleted"}
```

- Document attribute `_deleted`: `true`
- *OLD* revisions still there till **compact**:

```
$ curl -X GET /my_database/hello-world?rev=2-b9f6bbfa...
```

- Files attached to documents
- Binary data of any kind: Images, PDF, Mails ...

## API

- **PUT** /products/apple-macbook-air/preview.jpg?rev=XXX
- **GET** /products/apple-macbook-air/preview.jpg
- **DELETE** /products/apple-macbook-air/preview.jpg?rev=XXX

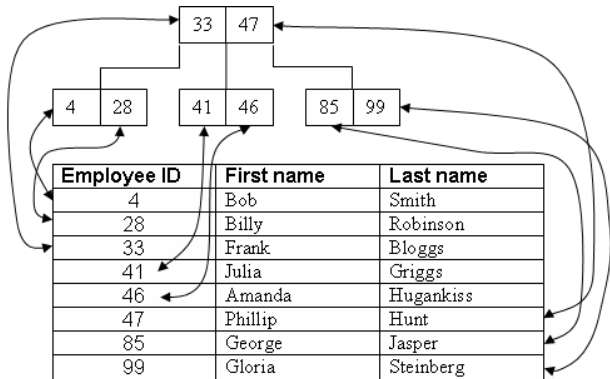
## Insights we learned:

- 1 Documents have `_id`'s and `_rev`'s.
- 2 Update/Delete needs latest `_rev`.
- 3 Optimistic locking via `_rev`.
- 4 Update **replaces** document.
- 5 Binary attachments.

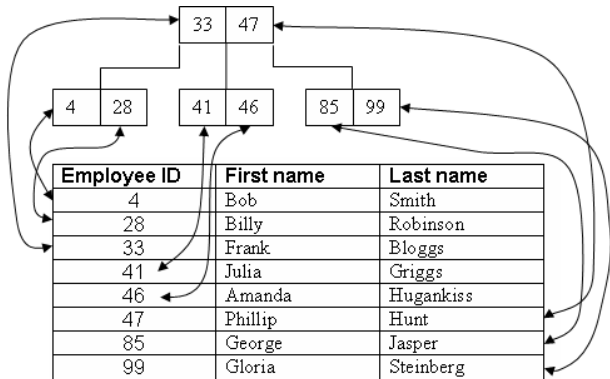
## How CouchDB works

# Databases use Trees to be efficient

## Databases use Trees to be efficient

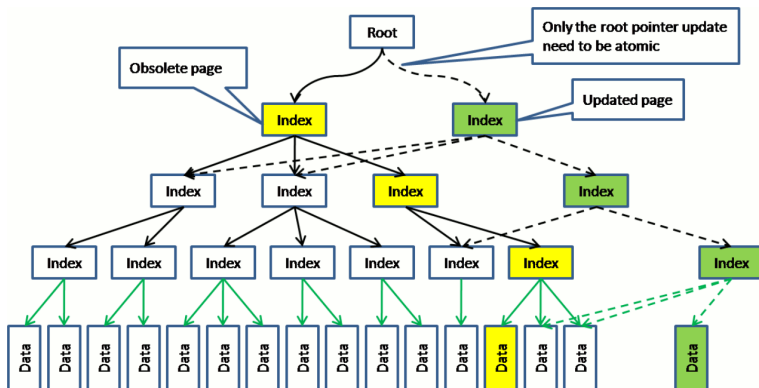


## Databases use Trees to be efficient



CouchDB is using a B+ Tree

# Understanding the B-Tree



- Using B-Tree to access documents.
- Efficient low tree structure.



# Realization:

Document ID's can have features!

IDs are unicode *strings*

IDs are unicode *strings*

There is a order defined by their binary value:

- A = U0041 to Z = U005A
- First: U0000 - Last: UFFFF

IDs are unicode *strings*

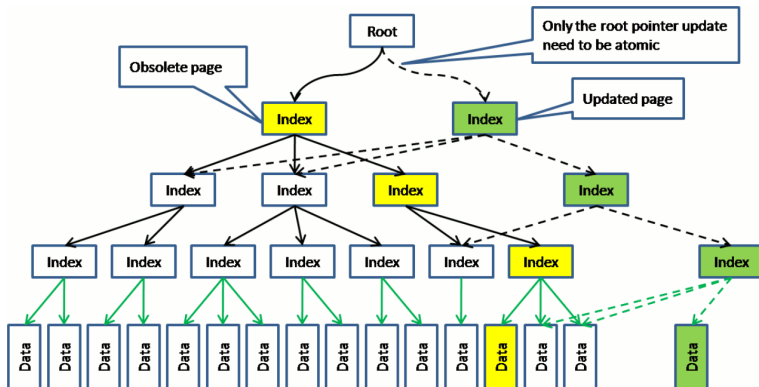
There is a order defined by their binary value:

- A = U0041 to Z = U005A
- First: U0000 - Last: UFFFF

IDs are held in *index* of Tree, not in *data*.

=> **Range iteration** possible.

# Understanding the B-Tree



- From U0000 on the *left*
- To UFFFF on the *right*

## Documents

```
{ "_id": "lenovo-thinkpad-t440", ... }  
{ "_id": "lenovo-edge-e130", ... }  
{ "_id": "apple-macbook-2013", ... }  
{ "_id": "apple-macbook-pro-2015", ... }
```

# Usefull IDs

## Documents

```
{ "_id": "lenovo-thinkpad-t440", ... }  
{ "_id": "lenovo-edge-e130", ... }  
{ "_id": "apple-macbook-2013", ... }  
{ "_id": "apple-macbook-pro-2015", ... }
```

## “Query” all apple products

```
$ curl '/products/_all_docs?startkey="apple-\  
                                &endkey="apple-\\uffff"'
```

- `_all_docs` is the resource for the *whole B-Tree* of the database.

# Typical CouchDB IDs

- user-alice, user-bob
- product-apple-macbook-2013, product-lenovo-thinkpad-t440
- order-user-alice-20150102, order-user-bob-20141212



# Typical CouchDB IDs

- user-alice, user-bob
- product-apple-macbook-2013, product-lenovo-thinkpad-t440
- order-user-alice-20150102, order-user-bob-20141212

## All users

```
$ curl '/my_database/_all_docs?startkey="user-"\
      &endkey="user-\\uffff"'
```

# Typical CouchDB IDs

- user-alice, user-bob
- product-apple-macbook-2013, product-lenovo-thinkpad-t440
- order-user-alice-20150102, order-user-bob-20141212

## All users

```
$ curl '/my_database/_all_docs?startkey="user-"\&endkey="user-\\uffff"'
```

## All products by lenovo

```
$ curl '/my_database/_all_docs?startkey="product-lenovo-"\&endkey="product-lenovo-\\uffff"'
```

# Typical CouchDB IDs

- user-alice, user-bob
- product-apple-macbook-2013, product-lenovo-thinkpad-t440
- order-user-alice-20150102, order-user-bob-20141212

## All users

```
$ curl '/my_database/_all_docs?startkey="user-"\
      &endkey="user-\\uffff"'
```

## All products by lenovo

```
$ curl '/my_database/_all_docs?startkey="product-lenovo-"\
      &endkey="product-lenovo-\\uffff"'
```

## All orders by bob in 2014

```
$ curl '/my_database/_all_docs?startkey="order-user-bob-2014"\
      &endkey="order-user-bob-2014\\uffff"'
```

# Pagination using Keys

## Simple pagination

```
$ curl '/my_database/_all_docs?limit=10&skip=0'  
$ curl '/my_database/_all_docs?limit=10&skip=10'
```

# Pagination using Keys

## Simple pagination

```
$ curl '/my_database/_all_docs?limit=10&skip=0'  
$ curl '/my_database/_all_docs?limit=10&skip=10'
```

## Paginate user documents

*# First page:*

```
$ curl '/my_database/_all_docs?startkey="user-"\n      &endkey="user-\\uffff"&limit=10'
```

*# Second page via `skip`:*

```
$ curl '/my_database/_all_docs?startkey="user-"\n      &endkey="user-\\uffff"&limit=10&skip=10'
```

*# Second page via `startkey\_docid`:*

```
$ curl '/my_database/_all_docs?startkey_docid=user-alice\n      &endkey="user-\\uffff"&limit=10'
```

# Realization:

CouchDB is a  
restfull HTTP API  
for B+ Trees!

# Fetching multiple keys at once

```
// keys.json
{
  "keys" : [
    "user-alice",
    "order-user-alice-20150102"
  ]
}
```

```
$ curl -X POST /my_database/_all_docs
-H "Content-Type: application/json"
--data-binary @keys.json
```

## Insights we learned:

- ➊ Restfull HTTP API to B+ Tree.
- ➋ Meaningfull IDs instead of UUIDs.
- ➌ IDs can determine type of a document.
- ➍ Iteration over IDs is usefull.
- ➎ Combination of keys is very usefull.
- ➏ Dates in IDs can be handy.



But this is not enough to work with ...

**We need:**  
queries!  
grouping!  
database logic!

## Advanced usage

# Realization:

“Querys” **always** need a Tree.

# Realization:

“Querys” **always** need a Tree.

---

We have to build them ourself!

# Views

CouchDB offers **Map**-Reduce to create indexes.

## Map function

```
function(doc) {  
    //    key        value  
    emit(doc.age, doc.hello);  
}
```

## Result

- Key: 42, Value: "world"
- Key: 43, Value: "world!"

CouchDB offers Map-**Reduce** to create indexes.

## Reduce function

```
function (key, values, rereduce) {  
    return values.join(',');  
}
```

## Result

- "world,world!"



## Map Functions:

- Each map function called *once* for any new/changed doc
- Defines key and value of index
- Convert data to needed format (*SELECT ...*)
- Filter documents (*WHERE ...*)
- Create multiple entries from one doc

## Reduce Functions:

- Will be called on collection of mapped entries
- Included functions: `_sum`, `_count`, `_stats`
- *Summarizes* mapped data

## Example: Query for Tags

### Map function:

```
function(doc) {  
    doc.tags.forEach(function(tag) {  
        emit(tag, doc._id);  
    });  
}
```

### Result:

```
"cool": "22d26e752e31bd1572f1d20d94000c47"  
"cooler": "hello-world",  
"couchdb": "22d26e752e31bd1572f1d20d94000c47"  
"couchdb": "hello-world"  
"is": "22d26e752e31bd1572f1d20d94000c47"  
"is": "hello-world"
```

### Reduce function:

```
_count
```

### Result with reduce:

```
"cool": 1
```

```
"cooler": 1
```

```
"couchdb": 2
```

```
"is": 2
```

## Example: Index date ranges

*Example for flexibility of CouchDB views*

### Map function:

```
function(doc) {  
  
    var dates = date_range(doc.start, doc.end);  
    // ['2015-04-01', '2015-04-02', '2015-04-03', ...]  
  
    dates.forEach(function(date) {  
        emit(date, doc._id);  
    });  
}
```

- Index will contain data, that not existed explicitly in doc!

# Design Documents

- JSON document
- Collection of javascript code
  - for views
  - for callbacks
- Container for “application code”
- Multiple Design Documents common
  - for users
  - for products
  - ...
- *Short: **DDoc***

# Design Documents

```
{
  "_id": "_design/tags",
  "_rev": "1-753abddc33070499276277bd90decabd",
  "language": "javascript",
  "views": {
    "by-tag": {
      "map": "function(doc) {
                doc.tags.forEach(function(tag) {
                  emit(tag, doc._id);
                });
              }",
      "reduce": "_count"
    }
  }
}
```

Access the *by-tags* view from ddoc *tags*:

```
$ curl -X GET '/my_database/_design/tags/_view/by-tag'
```



Access the *by-tags* view from ddoc tags:

```
$ curl -X GET '/my_database/_design/tags/_view/by-tag'
```

```
{  
  "rows": [  
    {  
      "key": null,  
      "value": 6  
    }  
  ]  
}
```

Whats that? - Ahh, its reduced ...

# Query View Reduce Flag

```
$ curl -X GET '/my_database/_design/tags/_view/by-tag\
?reduce=false'
```

```
{
  "offset": 0,
  "total_rows": 6,
  "rows": [
    {
      "id": "22d26e752e31bd1572f1d20d94000c47",
      "key": "cool",
      "value": "22d26e752e31bd1572f1d20d94000c47"
    },
    {
      "id": "hello-world",
      "key": "cooler",
      "value": "hello-world"
    },
    ...
  ]
}
```

# Query View Group Level

```
$ curl -X GET '/my_database/_design/tags/_view/by-tag\
?group_level=0'
```

```
{
  "rows": [
    {
      "key": null,
      "value": 6
    }
  ]
}
```

The same as the **first request**.

Default is ?group\_level=0  
and reduce=true if reduce function exists.

# Query View Group Level

```
$ curl -X GET '/my_database/_design/tags/_view/by-tag\  
?group_level=1'
```

```
{  
  "rows": [  
    {  
      "key": "cool",  
      "value": 1  
    },  
    {  
      "key": "couchdb",  
      "value": 2  
    },  
    ...  
  ]  
}
```

# Query View by Key

```
$ curl -X GET '/my_database/_design/tags/_view/by-tag\
?key="couchdb"&reduce=false'
```

```
{
  "offset": 2,
  "total_rows": 6,
  "rows": [
    {
      "id": "22d26e752e31bd1572f1d20d94000c47",
      "key": "couchdb",
      "value": "22d26e752e31bd1572f1d20d94000c47"
    },
    {
      "id": "hello-world",
      "key": "couchdb",
      "value": "hello-world"
    }
  ]
}
```

# Realization:

Every needed “Query”  
needs to be programmed  
with **Map-Reduce** first.

## Insights we learned:

- 1 Views instead of Queries.
- 2 Views programmed with **map-reduce**.
- 3 Each view is a own accessible Index.
- 4 Code for views inside a Design-Document.
- 5 No arbitrary WHERE `this=$that AND ....`

## Extended Topics



## Different documents in same database

- Using prefix for id
- Having type field

**Simply using both:**

```
{  
  "_id": "user-alice",    // for _all_docs  
  "_rev": "1-753abddc33070499276277bd90decabd",  
  "type": "user",        // for views  
  "name": "Alice"  
}
```

- Can contain multiple views and other functions.
- callbacks:
  - views
  - filters
  - lists
  - rewrites
  - shows
  - updates
  - validate\_doc\_update
- In Javascript, Ruby, Python, Perl, Lisp, Erlang, ...
- Changing DDoc = rebuilding views.
- Building views costs time!

# Changes feed

- Like twitter for your database, with less drama.
- Easy notification system.

```
$ curl /my_database/_changes?since=5
```

```
{
  "last_seq":6,
  "results":[
    {
      "changes":[{"rev":"3-e4f06ea31419b24495c94bcd952777b6"}],
      "deleted":true,
      "id":"hello-world",
      "seq":6
    }
  ]
}
```

# Replication

CouchDB can **replicate DB** from other CouchDB via its own HTTP API.

- 1 “Master” - n “Slaves”
- n “Master”
- ...
- “Slaves” of “Slaves” possible
- *“Automatic conflict handling” ...*

For CouchDB in “*Multi-Master cluster*”-Setup.

## CAP

- Partition Tolerance: YES
- Availability: YES
- Consistency: Eventual

## Pick Two

- CA: **MySQL**, Postgres, RDBMS
- CP: **MongoDB**, Google BigTable
- AP: **CouchDB**, Cassandra

- CouchDB fits best for Read >> Write.

- CouchDB fits best for Read >> Write.
- Use HTTP correctly (ETag, ...)



- CouchDB fits best for Read >> Write.
- Use HTTP correctly (ETag, ...)
- Create new docs instead of updating.

- CouchDB fits best for Read >> Write.
- Use HTTP correctly (ETag, ...)
- Create new docs instead of updating.
- Need to design to avoid conflicts.

- CouchDB fits best for Read >> Write.
- Use HTTP correctly (ETag, ...)
- Create new docs instead of updating.
- Need to design to avoid conflicts.
- Design for your access pattern.

- CouchDB fits best for Read >> Write.
- Use HTTP correctly (ETag, ...)
- Create new docs instead of updating.
- Need to design to avoid conflicts.
- Design for your access pattern.
- Avoid highly relational data.

- CouchDB fits best for Read >> Write.
- Use HTTP correctly (ETag, ...)
- Create new docs instead of updating.
- Need to design to avoid conflicts.
- Design for your access pattern.
- Avoid highly relational data.
- Small attachment files only.

- CouchDB fits best for Read >> Write.
- Use HTTP correctly (ETag, ...)
- Create new docs instead of updating.
- Need to design to avoid conflicts.
- Design for your access pattern.
- Avoid highly relational data.
- Small attachment files only.
- Doc revisions ARE NOT A Revision Control System!

- CouchDB fits best for Read >> Write.
- Use HTTP correctly (ETag, ...)
- Create new docs instead of updating.
- Need to design to avoid conflicts.
- Design for your access pattern.
- Avoid highly relational data.
- Small attachment files only.
- Doc revisions ARE NOT A Revision Control System!
- Reduce function `_sum` and `_count` are really cheap.

- Lots of disk space because of appending file format.
- Compaction of database/views:
  - Rewriting whole file.
  - Removing old `_rev`'s.
- Can be done automatically:
  - [wiki.apache.org/couchdb/Compaction](http://wiki.apache.org/couchdb/Compaction)



# Further Topics to explore

- Clustering
  - via HTTP Replication
  - Master-Slave
  - Master-Master
- Browser Local Storage
  - PouchDB
  - Syncing Browser of offline Apps
- Empowering HTTP
  - Proxy
  - Caching (Etag, ...)
- Fulltext search
  - River to Elasticsearch
  - Push to Lucene

## Links

- [couchdb.apache.org](http://couchdb.apache.org)
- [wiki.apache.org/couchdb](http://wiki.apache.org/couchdb)
- [guide.couchdb.org](http://guide.couchdb.org)
- <http://docs.ehealthafrica.org/couchdb-best-practices/>

## Other talks

- <https://speakerdeck.com/renatosnrg/couchdb>
- <https://speakerdeck.com/dzuelke/an-introduction-to-couchdb-ipc2011se-2011-06-01>

The End - Thanks!