CouchDB Notes

NoSQL databases are classified in 3 types :

1. Key-Value Pair

2. Column Store

3. Document Store : example is couchDB and MongoDB

It uses JSON, to store data (documents), java script as its query language to transform the documents, http protocol for api to access the documents, query the indices with the web browser. It is a multi master application released in 2005 and it became an apache project in 2008.

* Open source
* Mapreduce and HTTP for an API
* Document metadata contains revision information, which makes it possible to merge the differences occurred while the databases were disconnected.
* CouchDB implements a form of Multi-Version Concurrency Control ([MVCC](https://en.wikipedia.org/wiki/Multiversion_concurrency_control)) so it doesn't have to lock the database file during writes. Conflicts are left to the application to resolve. Resolving a conflict generally involves first merging data into one of the documents, then deleting the stale one.[[](https://en.wikipedia.org/wiki/CouchDB#cite_note-2)

Difference between MongoDB and CouchDb

|  |  |
| --- | --- |
| MongoDB | COUCHdb |
| CP | AP |
| If you need dynamic queries. If you prefer to define indexes, not map/reduce functions. If you need good performance on a big DB. If you wanted CouchDB, but your data changes too much, filling up disks. | For accumulating, occasionally changing data, on which pre-defined queries are to be run. Places where versioning is important. |
| Master-Slave Replication ONLY | * Master-Master Replication |
| If your code will run only on the server then go with MongoDB | If you plan to have a mobile component, or need desktop users to work offline and then sync their work to a server you need CouchDB. |

Once the data is entered in to the disc it will not be overwritten. Document updates (add, edit, delete) follow Atomicity, i.e., they will be saved completely or not saved at all. The database will not have any partially saved or edited documents.

Compaction is an operation to avail extra disc space for the database by removing unused data. While performing compaction operation on a particular file, a file with the extension **.compaction** is created and all the active/actual data is copied (cloned) to that file, when the copying process is finished then the old file is discarded. The database remains online during the compaction and all updates and reads are allowed to complete successfully.

After installation, open built-in web interface of CouchDB by visiting the following **link:** [**http://127.0.0.1:5984/**](http://127.0.0.1:5984/)**.**

**Futon is web interface of couchDB**

**To start futon**

http://127.0.0.1:5984/\_utils/

**CouchDB is a web interface**

**1.To access list of all dbs**

curl -X GET http://127.0.0.1:5984/\_all\_dbs

note: by default there are 2 databases in couchdb “\_replicator” and “\_users”

2. Creating a database

curl -X PUT http://127.0.0.1:5984/**database\_name**

3, to get database info

curl -X GET http://127.0.0.1:5984/my\_database

curl -X PUT http://127.0.0.1:5984/database\_name/"001" -d

'{ " Name " : " Raju " , " age " :" 23 " , " Designation " : " Designer " }'

curl –X PUT <http://127.0.0.1:5984/database_name/”001>” -d ‘{ “Name” : “Raju”}’

HTTP URL Paths

There are certain url paths using which, you can interact with the database directly. Following is the tabular format of such url paths.

|  |  |
| --- | --- |
| **URL** | **Operation** |
| **PUT /db** | This url is used to create a new database. |
| **GET /db** | This url is used to get the information about the existing database. |
| **PUT /db/document** | This url is used to create a document/update an existing document. |
| **GET /db/document** | This url is used to get the document. |
| **DELETE /db/document** | This url is used to delete the specified document from the specified database. |
| **GET /db/\_design/design-doc** | This url is used to get the definition of a design document. |
| **GET /db/\_design/designdoc/\_view/view-name** | This url is used to access the view, view-name from the design document from the specified database. |

Each document in CouchDB has a unique ID. You can choose your own ID that should be in the form of a string. Generally, UUID (Universally Unique IDentifier) is used, which are random numbers that have least chance of creating a duplicate. These are preferred to avoid collisions.

* **"rev",** this indicates the revision id. Every time you revise (update or modify) a document a **\_rev** value will be generated by CouchDB. If you want to update or delete a document, CouchDB expects you to include the **\_rev** field of the revision you wish to change. When CouchDB accepts the change, it will generate a new revision number. This mechanism ensures concurrency control.

Attaching files using curl

$ curl -vX PUT http://127.0.0.1:5984/database\_name/database\_id

/filename?rev=document rev\_id --data-binary @filename -H "Content-Type:

type of the content"

The request has various options that are explained below.

* **--data-binary@** - This option tells cURL to read a file’s contents into the HTTP request body.
* **-H** - This option is used to mention the content type of the file we are going to upload.

Replication:

<http://guide.couchdb.org/draft/replication.html>

<https://wiki.apache.org/couchdb/Replication_and_conflicts>

curl -X POST http://127.0.0.1:5984/\_replicate -d '{"source":"http://example.org/database", "target":"http://admin:password@127.0.0.1:5984/database"}' -H "Content-Type: application/json"

for continuous replication

curl -X POST http://127.0.0.1:5984/\_replicate -d '{"source":"db", "target":"db-replica", "continuous":true}' -H "Content-Type: application/json"’

If you do GET /db/bob?conflicts=true, and the document is in a conflict state, then you will get the winner plus a \_conflicts member containing an array of the revs of the other, conflicting revision(s). You can then fetch them individually using subsequent GET /db/bob?rev=xxxx operations.

the winning revision is picked by these rules, in descending order of priority:  
  
1. A live revision wins over a deleted one.  
2. Higher generation number [the numeric prefix of the revision ID] wins.  
3. Lexicographically-higher revision ID wins.

Once you have retrieved all the conflicting revisions, your application can then choose to display them all to the user. Or it could attempt to merge them, write back the merged version, and delete the conflicting versions - that is, to resolve the conflict permanently.

As described above, you need to update one revision and delete all the conflicting revisions explicitly. This can be done using a single POST to \_bulk\_docs, setting"\_deleted":true on those revisions you wish to delete.

Push Replication: When a replication task is initiated on the sending node

Pull replication :  if it is initiated by the receiving node

## 4.1.3. Master - Master replication

One replication task will only transfer changes in one direction. To achieve master-master replication, it is possible to set up two replication tasks in opposite direction. When a change is replicated from database A to B by the first task, the second task from B to A will discover that the new change on B already exists in A and will wait for further changes.

## 4.1.4. Controlling which Documents to Replicate

There are two ways for controlling which documents are replicated, and which are skipped. Local documents are never replicated (see[Local (non-replicating) Documents](http://docs.couchdb.org/en/stable/api/local.html#api-local)).

Additionally, [Filter functions](http://docs.couchdb.org/en/stable/couchapp/ddocs.html#filterfun) can be used in a replication (see [Replication Settings](http://docs.couchdb.org/en/stable/json-structure.html#replication-settings)). The replication task will then evaluate the filter function for each document in the changes feed. The document will only be replicated if the filter returns true.

<http://guide.couchdb.org/draft/consistency.html>

<https://wiki.apache.org/couchdb/Replication_and_conflicts>

<https://wiki.apache.org/couchdb/HTTP_Document_API>