Oct 26 / 4:41pm

Massive CouchDB Brain Dump

by Matthew Woodward

The following is a semi-unorganized brain dump of everything of interest I've come across while learning the incredibly cool CouchDB document-oriented database system. In this brain dump I pull things from many different resources including my own head, so there may be literal quotes from some of these resources without inline attributions. For that I apologize, but rest assured I'm not trying to plaigarize anyone or take credit where it's not due; I was just merely taking notes as I perused a lot of different resources and organized them in a way that made sense to me. I ***do*** have a complete list of all of the resources I used at the end to let you explore on your own. Again, my apologies to the creators of the resources from which I pull for not attributing inline.

I'll be presenting CouchDB to the ColdFusion Meetup on December 17 (Charlie did a great job of booking a full schedule through the end of the year) so don't miss it!

CouchDB: General Concepts

- document-oriented
- schemaless
- NOT RELATIONAL
- JSON-based
- REST-based
- MapReduce
    - http://labs.google.com/papers/mapreduce.html
- basically throw out everything you know about databases and you'll pick this up a lot faster
- Calls are made to the database via HTTP
    - Yes, that means via a browser, curl, cfhttp ... anything that talks HTTP
- Responses come back as JSON
- Lock-free design--reads don't have to wait for writes or other reads
- Why are these good things?
    - More like how data works in the real world
        - e.g. business cards--if one has a fax and another doesn't, in an RDBMS you have to have a fax field that's going to be null for anyone who doesn't have a fax
        - with CouchDB, you can have one record with a fax field, and another with no fax field, but they're both considered business cards since every document in CouchDB is 100% self-contained
    - Simple
        - some of this stuff is so simple you'll be amazed there isn't more to it
    - Fast
        - Push/Pull of JSON data over HTTP
        - No messy, time-consuming joins between tables--a document contains all its data
    - Scalable
    - Takes the object-relational mismatch out of the picture to a certain extent
    - It works like the web does
- History of CouchDB

        - development started in 2004
        - originally written in C++, now in Erlang
        - Damien Katz quit his job and self-funded development full-time for 2 years
            - was formerly with IBM working on Lotus Notes, also a brief stint at MySQL
        - Damien Katz is now a full-time employee at IBM and gets paid to work on CouchDB full time
        - CouchDB is a top-level Apache project and is released under the Apache 2.0 license
- CouchDB's motto: RELAX.
    - we shouldn't have to worry about data so much

Why Use CouchDB?

- throws out the relational model and looks at what matters with data in the majority of applications
- vastly simplifies data modeling and interaction with data
- extremely flexible since there are no preset schemas
- in relational databases, data does get to a point where it's unwieldy and slow to access
- relational model is hard to scale, and doesn't do so very naturally or quickly
- CouchDB offers ...
    - robust, dead simple replication to any number of servers
    - bi-directional conflict detection and resolution
    - fantastic performance on huge databases
        - number of records in a database has very little impact on performance
    - fantastic scalability
        - Erlang was designed for real-time telcom apps in the 1980s, so it's ideal for high scalability and highly concurrent apps like database servers
        - early testing with CouchDB shows it can handle 20,000 concurrent connections with no problems, and they haven't even done and performance profiling yet
            - lead developer said in an interview that using conventional threading in C++ you'd be lucky to handle 500 concurrent connections
        - Erlang also can help with multi-machine scalability, failover, etc. but CouchDB is not taking advantage of any of this yet
- CouchDB speaks the language of the web
    - REST, HTTP, and JSON are how CouchDB works natively
- already gaining a huge amount of traction and becoming very popular
    - libraries for CouchDB already exist for: (see full, current list at http://wiki.apache.org/couchdb/Basics)
        - Amazon EC2 (this is a nice way to play around with CouchDB without installing anything; just costs a few cents an hour)
        - C
        - C#
        - Erlang
        - ExtJS
        - Haskell
        - Java
        - JavaScript

        - Lisp
        - LotusScript
        - Objective-C
        - Perl
        - PHP
        - Python
        - Ruby
        - Smalltalk
        - VMWare (another great way to play around with it without installing anything; might be a very old version though)
    - what about CFML?
        - one CFC wrapper project on RIAForge -- http://couchdb.riaforge.org/
            - seems to provide basic "friendly messages" for what CouchDB spits back

Is the Relational Model Dead?

- there is an increasing indication that the relational model will begin being seen as *a* solution, not *the* solution
- Map/Reduce is simply a better model for dealing with large datasets and taking advantage of parallel processing
    - http://labs.google.com/papers/mapreduce.html
- "Map/Reduce will kill every traditional data warehousing vendor in the market. Those who adapt to it as a design/deployment pattern will survive, the rest won't."
    - Might think this came from a non-relational database vendor, but it's actually from Brian Aker, one of the original authors of MySQL and currently working on the Drizzle (http://drizzle.org/wiki/Main_Page) fork of MySQL
- document-based databases like CouchDB scale far better and easier than relational databases do
    - both Amazon and Google came up with their own database solutions for their cloud computing platforms as opposed to using a traditional RDBMS--this should tell you something
- Better and more natural fit for applications

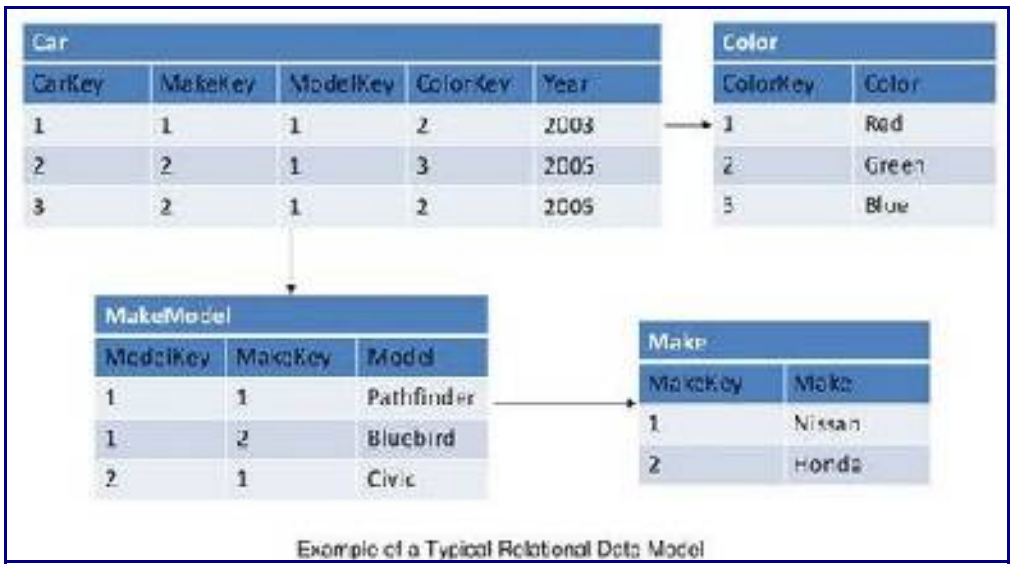
More on "Better Fit for Applications"

- self-contained documents
    - no more taking a real world construct and deconstructing it into a relatonal model
- flexible schemas
    - two documents can be of the same type and not contain the same fields--don't have to have a bunch of nulls involved, worry about foreign keys, etc. etc. since every document is self-contained
    - if you need a change in your schema, it's dead simple to do--just start using the new schema
        - if you don't care that the old documents don't have the new field, you don't have to worry about them
- speaks our language as application developers
    - REST and JSON--doesn't get much simpler than that
- since it's all web based you take advantage of the following at the database level

    - can handle more traffic since connections aren't left open
    - clustering, proxying, caching, security, etc. behaves just as it would with an HTML document
- Creator of CouchDB said one of the goals was to have users feel like "you could touch your data ... like it was right there in your hands"
    - eliminating all the layers between your application and your data

Relational Model vs. Document-Based aka "Key/Value Store" Databases

- relational diagram



Example of a Typical Relational Data Model

Download this gallery (ZIP, undefined KB)

Download full size (37 KB)

- key/value diagram

Car	
Key	Attributes
1	Make: Nissan Model: Pathfinder Color: Green Year: 2003
2	Make: Nissan Model: Pathfinder Color: Blue Color: Green Year:2005 Transmission: Auto

Example of a Typical Key/Value Domain

- CouchDB pros
    - ideally suited for cloud computing
    - more natural fit with the code we write -- no ORM mismatch nonsense to worry about
- CouchDB cons
    - relational databases enforce integrity at the database level
    - schemaless nature of CouchDB means your data integrity is that the APPLICATION level
        - bugs in application code using RDBMS don't lead to data integrity issues
        - bugs in application code using CouchDB CAN lead to data integrity issues
        - really this just puts this concern in a different place, but it's something to be aware of
    - no shared standards between key/value database vendors
        - much easier to move from SQL Server to MySQL than it would be to move from CouchDB to Amazon or Google
- other concerns with cloud databases ...
    - limtations on analytics -- e.g. Amazon queries cannot take longer than 5 seconds to run
    - limitations on data returned -- e.g. Google queries cannot return more than 1000 rows
- from an application development standpoint, in my experience thus far this does bring your data repository a bit more into the realm of your application
    - again, this isn't SQL, so your code isn't running queries and dealing with query objects; instead it's making HTTP calls and dealing with JSON
    - less friction between your app and your data, but be aware that it's a bit of a whole new world when you're working with CouchDB

Should You Consider CouchDB?

- yes if you ...

        - have tables with lots of columns of which you typically only use/display a few
        - have lots of joins in your queries
        - are serializing JSON or XML data into single columns in your relational database
        - have data that is more heirarchical or flat than it is relational
        - have systems that require frequent schema changes
        - are reaching the performance capacity of a single database server and need to scale out
        - have an amount of data that is difficult for a single server to hold
        - have background processes running on your database that impact performance of the database as a whole
- the nice thing about CouchDB is that it's highly and easily scalable by its very nature
    - but if you don't need scalability now, you don't have to worry about it; you just get it when you need it practically for free
- Why not just dump JSON data into a relational database?
    - because RDBMSes don't know anything about JSON, so you don't get any of the huge efficiency and functionality advantages you get with CouchDB

Other Document-Based Databases

- Amazon SimpleDB
  http://aws.amazon.com/simpledb/
- Google BigTable
  http://labs.google.com/papers/bigtable.html
- Voldemort
  http://project-voldemort.com/
- Cassandra
  http://incubator.apache.org/cassandra/
- Dynomite
  http://github.com/cliffmoon/dynomite/tree/master
- HBase
  http://hadoop.apache.org/hbase/
- Hypertable
  http://hypertable.org/
- VPork
  http://wiki.github.com/trav/vpork/vpork
- MongoDB
  http://www.mongodb.org/
- ThruDB
  http://code.google.com/p/thrudb/
- Jackrabbit
  http://jackrabbit.apache.org/

Building/Installing CouchDB

- basic requirements: Erlang, SpiderMonkey (JavaScript engine), other miscellany
- on Linux you'll need to install some prerequisites/dependencies if you don't have them; here's the list for Ubuntu ...
    - sudo apt-get install subversion

        - sudo apt-get install libtool
        - sudo apt-get install automake
        - sudo apt-get install libmozjs-dev
        - sudo apt-get install libicu-dev
        - sudo apt-get install curl
        - sudo apt-get install libcurl4-gnutls-dev
        - sudo apt-get install erlang-dev
        - sudo apt-get install erlang-nox
        - sudo apt-get install openssl
        - sudo apt-get install libssl-dev
            - double-check you have openssl and libssl installed; otherwise you may not get an error until you first try to run CouchDB
    - alternatively you can try ...
        - sudo apt-get build-dep couchdb
        - you can also check http://wiki.apache.org/couchdb/Installing_on_Ubuntu for the latest info
    - then grab the code and build it (do this from your home directory or wherever you like)
        - svn co http://svn.apache.org/repos/asf/couchdb/trunk couchdb
        - cd couchdb
        - sudo ./bootstrap
            - You should see "You have bootstrapped Apache CouchDB, time to relax." If not, fix any dependency issues it lists (the error messages are very explicit).
        - sudo ./configure
            - You should see "You have configured Apache CouchDB, time to relax."
        - sudo make
        - sudo make install
            - if you don't get any errors with make or make install you should be able to launch CouchDB!
    - Check http://wiki.apache.org/couchdb/Installing_on_Windows for information about installing on Windows. Haven't tried this myself, likely won't, so best of luck.
        - You'll want/need to install cygwin (a bash shell for Windows) to get this all rolling -- http://www.cygwin.com/
        - As of version 0.10 support for Windows is part of the standard build process, so at least you have that going for you.
        - There are some binaries available at http://wiki.apache.org/couchdb/Windows_binary_installer but as usual with binaries they're a few versions behind, which for a fast-moving project like CouchDB is less than ideal. I'm sure as the project progresses the Windows binaries for the latest versions will be more immediately available.
        - Other Windows links:
            - http://www.brunomlopes.com/software/couch-db-binaries (general info; binaries for 0.8.1)
            - http://people.virginia.edu/~lmb7s/couch/ (binary for 0.9.1)

Running CouchDB

- on Linux the install process puts the couchdb script in your path, so you can open a terminal and type sudo couchdb
    - you should see:
      Apache CouchDB has started. Time to relax.
      [info] [<version_number>] Apache CouchDB has started on http://127.0.0.1:5984/
    - If you see an error along the lines of {"init terminating in do_boot",{undef,[{crypto,start, []} ... that means you don't have erlang-nox and/or libssl-dev installed, so you'll have to go back through the steps above once you have those dependencies resolved.
    - If you have other errors when trying to start CouchDB check http://wiki.apache.org/couchdb/Error_messages

Interacting with CouchDB with CURL

- some basic examples
    - curl -X GET http://localhost:5984/
        - returns basic server info
    - curl -X GET http://localhost:5984/_all_dbs
        - returns list of all databases on the server
    - curl -X PUT http://localhost:5984/contacts
        - creates a new database called contacts
    - curl -X PUT http://localhost:5984/contacts/6e1295ed6c29495e54cc05947f18c8af -d '{"firstName":"Matt", "lastName":"Woodward", "email":"matt@mattwoodward.com"}'
        - creates a new document in the contacts database; the string after the database name is a UUID
    - curl -vX PUT http://localhost:5984/contacts/6e1295ed6c29495e54cc05947f18c8af/headshot.jpg?rev=2-2739352689 -d@headshot.jpg -H "Content-Type: image/jpg"
        - attaches a headshot jpeg to the document with the ID provided
    - curl -X GET http://localhost:5984/_uuids
        - CouchDB returns a new UUID; can add ?count=N to get back N UUIDs if you need more than one
    - curl -X GET http://localhost:5984/contacts/6e1295ed6c29495e54cc05947f18c8af
        - returns the document with the UUID provided
    - curl -X DELETE http://localhost:5984/contacts/6e1295ed6c29495e54cc05947f18c8af?rev=2-212344
        - deletes the document with the ID provided; note that you must provide the latest revision number for the document in order for the delete to succeed
    - curl -X DELETE http://localhost:5984/contacts
        - deletes the contacts database
    - curl -X POST http://localhost:5984/_replicate -d

'{"source":"contacts","target":"contacts-replica"}'
    - replicates the contacts database to the contacts-replica database
- of course since this is just HTTP, you can use CURL's -v flag to get a verbose listing of everything CouchDB is doing on each request
- performing updates is a bit different
    - if you do a PUT of a document with the same ID but don't include a revision number, the update will fail
    - you have to include the latest revision number in CouchDB in updates for them to work
    - what this means in practice is that you'll pull the document you want to update back, update the JSON (or update the data in your application code), and then do a put of the updated document with the new data since this will contain the most recent revision in CouchDB
    - the updated document gets a new revision number, and the original document is retained in CouchDB as a previous revision

Versioning of Documents

- CouchDB uses a multi-version concurrency control (MVCC) system
- each document in CouchDB gets a revision number
- previous versions of documents are saved in CouchDB
    - BUT ... unlike a version control system, there is no guarantee how long the previous versions will be retained
    - you can tell CouchDB you want to retain the previous versions of a document if you need to
- remember that all communication with CouchDB is done over HTTP
    - HTTP is stateless--you open a connection to CouchDB, make a request, then the connection is terminated
        - this is good because it means CouchDB can handle a lot of traffic since connections are short-lived
- If you're familiar with Etags in the HTTP world, CouchDB uses its revision numbers as Etags in HTTP responses
    - Etags are very useful for caching
    - since all documents in CouchDB are really just resources in the HTTP/REST sense, your data behaves like any other HTML resource

Futon: CouchDB's Web-Based Interface

- browse to http://localhost:5984/_utils for the web-based interface to CouchDB
- handy way of perusing your documents, managing datbases, etc.
- definitely handy for creating design documents and views
    - mini editor for creating temporary and permanent views--can execute temporary views from within the editor
- can kick off replication and a ton of other stuff from Futon

Creating a Document

- new documents have _id and _rev fields added automatically
- documents are versioned much like code is in SVN, so every version of every document in the db is stored
- click "add field" in Futon to add a new field to a document
- double-click value (default is null) to edit
- values must be JSON valid data
    - strings have to have quotes around them, e.g. "hello" not just hello
    - valid datatypes are string, number, boolean, list, and key/value dictionaries
- you can do a "view source" on a document from Futon to see the JSON version of the document
- as you update a document, the version number will change with each revision
    - if another process changes the document before you save your changes, a conflict will arise
- CouchDB has no concept of "types"
    - e.g. in a blog application we would think of "posts" and "comments" as types
    - remember that CouchDB is schemaless, so there is no inherent structure to documents contained within the database itself
    - common to use a type field on a document containing a string that defines the type
        - makes it easy to write a view that pulls back specific document types
        - CouchDB does NOT CARE what field you use to define type--you can call this anything because again, CouchDB has no concept of document types
    - remember also that even if you define a document as a type, it does NOT have to literally match the structure of other documents with that same type
        - e.g. music library--could define "album" as a type, and if one album has a year field and another doesn't, they're both still "album" types since we defined the type explicitly
        - BUT, if you do want to require a specific structure for a document type, you can do that with validation functions and, e.g., reject an addition or update of an album to a music library if it didn't contain a year field
    - also handy to infer type based on fields for more flexibility
        - e.g. in a blog app we could use if (doc.title && doc.body) and assume if those fields are present that this is a blog post as opposed to a comment

How Documents Are Not Like Database Records

- self-contained--no joins across table to put together a single record
    - documents in CouchDB map directly to an object instance in your application
- typically documents will automatically have authors and publish dates associated with them
    - very easy to publish documents of any type in the future
    - if you create user accounts for CouchDB it automatically keeps track of who created and modified records
- don't break documents into smaller units than you need to!
    - a blog post will have an author--don't have the author be a separate document
- JSON document format
    - CouchDB documents all have _id and _rev fields
        - _id can be anything, so long as its unique--UUID, plain old string, whatever
        - _rev is the revision number--this changes with each update to a document

        - to update a document, you have to provide the most recent value of _rev so CouchDB knows you're working with the latest revision
    - if users have been configured, documents will have an author field
    - Do I really look like a guy with a plan? You know what I am? I'm a dog chasing cars. I wouldn't know what to do with one if I caught it. You know, I just… do things. The mob has plans, the cops have plans, Gordon's got plans. You know, they're schemers. Schemers trying to control their little worlds. I'm not a schemer. I try to show the schemers how pathetic their attempts to control things really are.

      — *The Joker*, The Dark Knight (this quote is used in the forthcoming O'Reilly CouchDB book)
    - NEED INFO ABOUT THINGS LIKE ARRAYS, ETC. HERE

Running Queries

- again, forget everything you know
- there is no SQL here
- instead of running queries in the traditional sense, data is filtered using map and reduce functions, which are written in javascript
    - DEFINE MAPREDUCE HERE
- the map and reduce functions combined create a CouchBD view
- views are stored as rows sorted by key
    - extremely efficient even for millions of records
- can create temporary views for testing, but these are rather inefficient, so views that are going to be used regularly are stored in the database as documents
    - once a view is stored in the database as a document, CouchDB indexes behind the scenes for efficiency
- main points:
    - map functions allow you to sort your data using any key you choose
    - CouchDB is designed to provide extremely fast access to data by key and key range
    - you don't really run queries against CouchDB, you query a view
    - when you query a view, CouchDB runs the map function against every document in the database in which the map function is defined
- map functions have a single "doc" parameter which is each individual document in your database
- the emit() function is used to spit out matching documents, and you can specify the fields you want to output
- if you're querying every document in the database every time, isn't that inefficient?
    - you'd think so, but no
    - CouchDB only runs through all the documents the first time the view is queried
    - as documents change, CouchDB only has to update what's changed
    - everything is stored in a B-Tree, which is very efficient
- creating multiple views specific to how you want to access the data helps with efficiency
- to execute a view, you just--surprise--hit a URL over HTTP and get JSON back
    - e.g. http://localhost:5984/database/_design/designdocname/_view/viewname
    - to add a key to this, it's just an argument in the query string of the url, e.g.
        - http://localhost:5984/database/_design/designdocname/_view/viewname?

key=value
    - can also retrieve documents by key range
        - http://localhost:5984/database/_design/designdocname/_view/viewname?startkey=startvalue&endkey=endvalue
- default query engine or "view server" in CouchDB is JavaScript, but you can write your own in any language
    - Remember it's all just HTTP and JSON!
- MAP FUNCTIONS take a document as an argument and emit key/value pairs
    - Btrees are very efficient--even with lots of documents the tree is "shallow" and it's pre-indexed so searches are very fast
- REDUCE FUNCTIONS operate on the rows returned by map functions and act as filters on the documents

Replication

- can replicate local -> local, local -> remote, or remote -> remote from Futon
- as with everything in CouchDB, this is all HTTP/REST based
    - initial replication may be time consuming
    - subsequent replications are diffs only
    - if you trigger replication from Futon, you have to leave the browser window open!
    - but since all this is HTTP based, easy to set up cron jobs that use curl to do replication
- a POST to CouchDB containing the source and target of replication is all that's needed to kick off replication
    - CouchDB maintains a session history of replication sessions, again in JSON
- can replicate among local databases or between databases not on the same physical box
- CouchDB has automatic conflict detection and resolution
    - remember that documents in the database are versioned, so conflicts are handled quite gracefully
    - documents that are in conflict when a replication occurs get a new _conflict:true attribute added to them
        - one of the two competing documents is given the latest revision number, the other is given a previous revision number
        - these conflicts are also replicated, so all databases will have the same information
- CouchDB takes the approach of "eventual consistency"
    - traditional RDBMS systems enforce consistency--put consistency above all in replication situations
    - “Each node in a system should be able to make decisions purely based on local state. If you need to do something under high load with failures occurring and you need to reach agreement, you’re lost… If you’re concerned about scalability, any algorithm that forces you to run agreement will eventually become your bottleneck. Take that as a given.”

      — *Werner Vogels*, Amazon CTO and Vice President
    - consistency between nodes is not guaranteed on writes, but the nodes will eventually be consistent on reads

Design Documents

- documents that contain application code
- IDs must start with _design as the ID, e.g. "_design/myapp"
- possible to write entire apps in HTML/JavaScript, store this code as a design document in CouchDB, and run the entire app from the CouchDB database
    - dynamic code (views and validation) written as JSON and stored as a document in CouchDB
        - MapReduce queries stored in the views field
        - data output CAN be things other than JSON using the show field, e.g. CouchDB can output RSS without any middleware
    - static HTML pages stored as attachments to the design document

Validation

- validation functions are used to do things like prevent users who aren't logged into an app from performing document updates
- validation functions are stored in design documents under the validate_doc_update field
    - can only have one validation function per design document
    - but remember you can have multiple design documents per database
- documents must pass all the validation rules on all design documents in the database in order to be saved
    - the order in which validation functions are executed is arbitrary
- most common example, since CouchDB is schemaless, is to require that certain fields be included if a document is declared to be of a particular type
    - e.g. require "title" and "body" for a document type of post

```
function(newDoc, oldDoc, userCtx) {
   function require(field, message) {
     message = message || "Document must have a " + field;
     if (!newDoc[field]) throw({forbidden : message});
   };

  if (newDoc.type == "post") {
     require("title");
     require("body");
   }
}
```

Show Functions

- since everything in CouchDB is JSON, HTTP, and JavaScript, it works well in any programming environment
- CouchDB doesn't, however, address things like outputting HTML
- easy enough to have CFML call CouchDB over HTTP and output the results in HTML
- CouchDB can, however, generate HTML natively using show functions
- basic show function

```
function(doc, req) {
   return '<h1>' + doc.title + '</h1>';
}
```

    - the "return" bit here is sent back to the browser as a HTTP response

- you can even write full HTML templates that embed CouchDB-specific scripting so you don't have to embed HTML in javascript functions

Attachments

- Documents in CouchDB, which are JSON, can have file attachments
- doing an HTTP PUT with a -d@filename.ext flag tells CouchDB to attach the file to the document ID provided in the PUT request
    - as with other updates to documents, you DO need to provide the current revision number of the document to attach a file to the document
    - unlike other updates you do NOT need to provide the data for the document itself in order to add an attachment to it
- documents may have mutliple attachments
- attachments are made available as, surprise, HTTP resources
    - http://localhost:5984/contacts/6e1295ed6c29495e54cc05947f18c8af/headshot.jpg would display the headshot.jpg file attached to the document with the ID provided
- if you pull a document back from CouchDB that has an attachment, the attachment file name and meta information such as type, size, etc. are contained in the JSON with a key of "_attachments"
    - adding ?attachments=true returns the attachments in base64 format as part of the JSON

Applications

- you can build applications entirely in CouchDB
- if you replicate your databases to another server, you replicate your app as well
    - means if you replicate to a local instance of CouchDB, you get offline data mode "for free"
- applications are stored in CouchDB as design documents
- can use couchapp for developing native CouchDB apps

Security

- can lock down databases by editing a simple config file
    - by default it's in /usr/local/etc/couchdb/local.ini
    - there's also a default.ini file, but any changes made to default.ini are overwritten when CouchDB is upgraded
- e.g. adding admin accounts to CouchDB
    - uncomment [admins] section and add authorized users as user = pass for each line
    - when CouchDB is restarted the passwords are hashed so they aren't stored in the config file in plain text
- remember--this is all just HTTP so you can apply the same HTTP-based security, proxies, reverse proxies, etc. as you would to any web resource
    - e.g. putting a web server in front of CouchDB and using HTTP authentication would be trivial

General Tips/Tricks

- In your applications, you'll want to create your own UUIDs for document IDs instead of letting CouchDB auto-create them
    - WHY? stated this in book but didn't elaborate
- Since replication and resynching is so dead simple, easy to replicate to a local DB for offline use, then resynch when back online
- For bulk conversions of existing databases to CouchDB, couple of performance tips
    - http://www.atypical.net/archive/2009/05/12/couchdb-090-bulk-document-post-performance
    - use the bulk document API instead of looping and doing individual document additions http://wiki.apache.org/couchdb/HTTP_Bulk_Document_API
    - don't use CouchDB's auto-assigned IDs--increases db size and has a big performance hit during conversion

There Are Some Cons ...

- it's new--they call it the bleeding edge for a reason
    - stuff WILL change between versions that will break your apps!
- it's a completely new skillset--there is no sql here
- views take a long time to build the first time they're saved, but after that they're incredibly fast regardless of the number of documents involved
- large databases can take up a lot of disk space
    - raw data is one consideration, but the views often take up much more disk space than the data itself
    - this is trading disk space for performance, which is a good tradeoff, so you just need to plan for disk capacity
- CouchDB does not deal well with relational data
    - that being said, we all likely spend a lot of time dealing with the shortcomings of relational data, specifically how horrendously bad the relational model is at dealing with heirarchical data, so I'm not sure this is a straight con as compare with the relational model
    - recurring theme in the CouchDB literature is DON'T GO AGAINST THE GRAIN! Don't try to force CouchDB to behave like an RDBMS, because it's not.
- CouchDB does not support transactions well
    - e.g. check to see if a user name is unique, then assign it--no way to isolate this from another simultaneous request in the same way you can with relational database transactions
- Reads on single documents, and writes in general, are slower than you might be used to with an RDBMS
    - but, the CouchDB model scales much better
- Have to write all of your "queries" (views) in advance--no on the fly SQL allowed
- map-reduce not as flexible as sql--sometimes you'll have to return more data than you want or need and process on the application side

Common Use Case That Is a Bit Odd in CouchDB: Unique Constraints

- e.g. guarantee a unique user name or email address
- thread here http://markmail.org/thread/qwgqql74b2gg5hc5

- other than the _id field, CouchDB has no way of guaranteeing uniqueness in any other field
- you CAN do a check to see if the record already exists, but remember ...
    - This is HTTP. There are no transactions.
    - There is no locking in CouchDB
- On the other hand, guaranteed uniqueness doesn't scale
- Some solutions
    - you can use anything as your _id, so use one piece of data that has to be unique AS your _id
    - use a relational table in an RDBMS to store any unique values and put a unique constraint on that field in the RDBMS
        - you could still have problems, but this would reduce the likelihood from slim (without doing this) to extremely slim

Resources

- CouchDB web site
  http://couchdb.apache.org
- O'Reilly CouchDB Book
  http://books.couchdb.org/relax/
- Why CouchDB?
  http://books.couchdb.org/relax/intro/why-couchdb
- Manning CouchDB Book (Early Access Edition)
  http://www.manning.com/chandler/
- APress CouchDB Book
  http://www.apress.com/book/view/9781430272373
- CouchDB in 15 Minutes
  http://wiki.apache.org/couchdb/CouchIn15Minutes
- Svenson: Java <-> JSON Convertor
  http://code.google.com/p/svenson/
- Java 5 CouchDB Driver
  http://code.google.com/p/jcouchdb/
- CouchDB4J - another Java library for use with CouchDB
  http://github.com/mbreese/couchdb4j/tree/master
- CouchDB Case Study
  http://johnpwood.net/2009/06/15/couchdb-a-case-study/
- CouchDB Wiki
  http://wiki.apache.org/couchdb
- CouchApp - IDE for building CouchDB applications natively
  http://github.com/jchris/couchapp/tree/master
- Sofa - Sample blog app built in CouchDB
  http://github.com/jchris/sofa/tree/master
- Building CouchBD on Linux
  http://japhr.blogspot.com/2009/03/yak-shaving-is-new-dependency-hell.html
- CouchBD from 10,000 Feet - video presentation from QCon
  http://www.infoq.com/presentations/couchDB-from-10K-feet
- Interview from RubyFringe with Damien Katz, Creator of CouchDB
  http://www.infoq.com/interviews/CouchDB-Damien-Katz

- Damien Katz Presentation from RubyFringe (mostly non-technical; talks about why he quit his job to build CouchDB)
  http://www.infoq.com/presentations/katz-couchdb-and-me
- Installing SpiderMonkey
  http://wiki.apache.org/couchdb/Installing_SpiderMonkey
- Is the relational database doomed?
  http://www.readwriteweb.com/enterprise/2009/02/is-the-relational-database-doomed.php
- Should you go beyond relational databases?
  http://carsonified.com/blog/dev/should-you-go-beyond-relational-databases/
- No SQL Meeting announcing the end of RDBMS?
  http://www.infoq.com/news/2009/08/NoSQL-and-the-End-of-RDBMS-Era
- Why CouchDB Sucks
  http://www.eflorenzano.com/blog/post/why-couchdb-sucks/
- Why CouchDB Rocks (same author as "Why CouchDB Sucks")
  aaaa

7 comments

Oct 27, 2009

Russ said...
Yikes! That is massive. Lots of good info in there though.
Have you tried out CouchDB for Coldfusion (from RIAforge)? Some feedback would be nice!

Oct 27, 2009

dswitzer2 said...
Thanks for the great write up on CouchDB. The only thing I would have liked to seen in more info on views.

Oct 27, 2009

Matthew Woodward said...
@Russ--thanks! Definitely be looking into your library before my CFMeetup presentation.

@Dan--there are definitely holes in there but I wanted to make this available instead of sitting on it. I'll fill in the gaps before long.

Oct 27, 2009

Martin Klepsch said...
while scrolling i wondered how huge it is :)

nice compilation and thoughts
Oct 28, 2009

John Allen said...
Awesome.
Apr 08, 2010

johnnygoodman said...
Epic. Thanks for taking the time to make it!
Aug 13, 2010

Pedro Landeiro said...
Excelent! thanks!