

The CouchDB Project

a document oriented database

Damien Katz, - Project Leader
January 24, 2008



CouchDB
relax

About Me

- Iris/IBM - Lotus Notes & Domino
- MySQL - Server Runtime
- IBM - Information Management, CTO's Office

What is a document?

- “any discrete representation of meaning“
- readable, writable, addressable: RESTful

CouchDB Documents

- Documents are JSON objects.
 - Name/Value pairs
 - Arbitrary Nested Structure
- Can contain file attachments
 - No limit to number or size.
 - one doc can contain an entire web page - HTML, metadata, images, video, flash, etc.

CouchDB JSON Document

```
{
  "_id": "2DA92AAA628CB4156134F36927CF4876",
  "_rev": "75AA3DA9",
  "type": "contact",
  "firstname": "Fred",
  "lastname": "Katz",
  "company": "JFK Construction",
  "kids": [{ "name": "Damien",
             "birthdate": "10/24/1973" },
           { "name": "Ashley",
             "birthdate": "8/28/1997" } ],
  ....
}
```

What is CouchDB?

- RESTful HTTP: GET/PUT/POST/DELETE
- Document database engine - not a wrapper
 - Optimistic commits with ACID, MVCC
- Replication - Fully updatable, occasionally connected replicas. Work with data offline.
- Javascript as query language
 - Views: incremental map/reduce

Why CouchDB?

- Closer to the common problem domains
- Simplicity.
- Locality matters. Offline/occasionally connected support. Data physically near you.
- Accessibility. Use from any language with a HTTP and JSON library.
- Admin tasks using common command line utilities - Incremental backup and restore

Document Oriented Applications

- Activities that, if not computerized, would require the use of paper documents
- Email, to dos, notes, wikis, forums, CMS, CRM, cataloging, approval workflow, expense tracking & reporting, status reports, asset management, project management, bug/issue tracking....
- Most web applications!

Project History

- Began coding in 2005.
- Storage, view and custom query language (Fabric) originally written in C++
- Discarded C++ for Erlang.
- Early public versions (0.1 - 0.6) Erlang and Fabric w/ XML HTTP API.

XML BEGONE!

- Fed up with XML - best as a markup language
 - Too many XML situps
 - “Tags in a sea of text”
- JSON - Maps better to native language types, easier to type, often more readable
- Javascript - obvious choice for core language
- CouchDB 0.7.0 - JSON & Javascript

What are views?

- Similar to RDBMS views
 - read-only, exists independent of the data
- Static queries of dynamic data.
 - Map/Reduce model
 - On-demand, incrementally updated indexes
- View operations: key match, range, limit, etc

A Simple Example

CRM Database

Example contact document:

```
{type:"contact",  
  firstname:"Fred",  
  lastname: "Katz",  
  company:"JFK Construction",  
  ... }
```

A View Function

View of contact documents keyed by lastname:

```
function (doc) {  
  if (doc.type == "contact")  
    map(doc.lastname, doc.firstname);  
}
```

The view functions are stored in special “design documents”, views results accessible via HTTP GET.

Get View Results

```
GET /db/_view/contacts/by_last?key="Katz"
```

Result:

```
{ "total_rows": 3,
  "rows": [
    { "id": "2DA92AAA628CB4156134F36927CF4876",
      "key": "Katz",
      "value": "Roseanna" },
    { "id": "4156134F36927CF48762DA92AAA628CB",
      "key": "Katz",
      "value": "Fred" },
    { "id": "4F36927CF48762DA92AAA628CB415613",
      "key": "Katz",
      "value": "Gwendolyn" }
  ]
}
```

Completely Different

SQL

CouchDB

Predefined, Explicit Schema	Dynamic, Implicit Schema
Uniform Tables Of Data	Collection of named docs with varying structure.
Normalized. Objects spread across tables. Duplication reduced.	Denormalized. Docs usually self contained. Data often duplicated.
Must know schema to read/ write a complete object	Must know only document name
Dynamic queries of static schemas	Static queries of dynamic schemas

SQL can be dynamic too

- EAV - Entity Attribute Value.
 - Considered an anti-pattern.
 - Cumbersome to query. Doesn't scale.
- Keep dynamic structures in strings/blobs
 - Difficult/impossible to query from SQL
- Static Schema - Best for OLTP and OLAP

Replication

- Not master/slave replication, more like "sync"
- multi-master, bi-directional replication usually just works without planning
- Enabled by self-contained doc model. No hierarchy, dependencies or schemas to sync.
- Distributed conflict management
- Selective replication - Replicate a subset

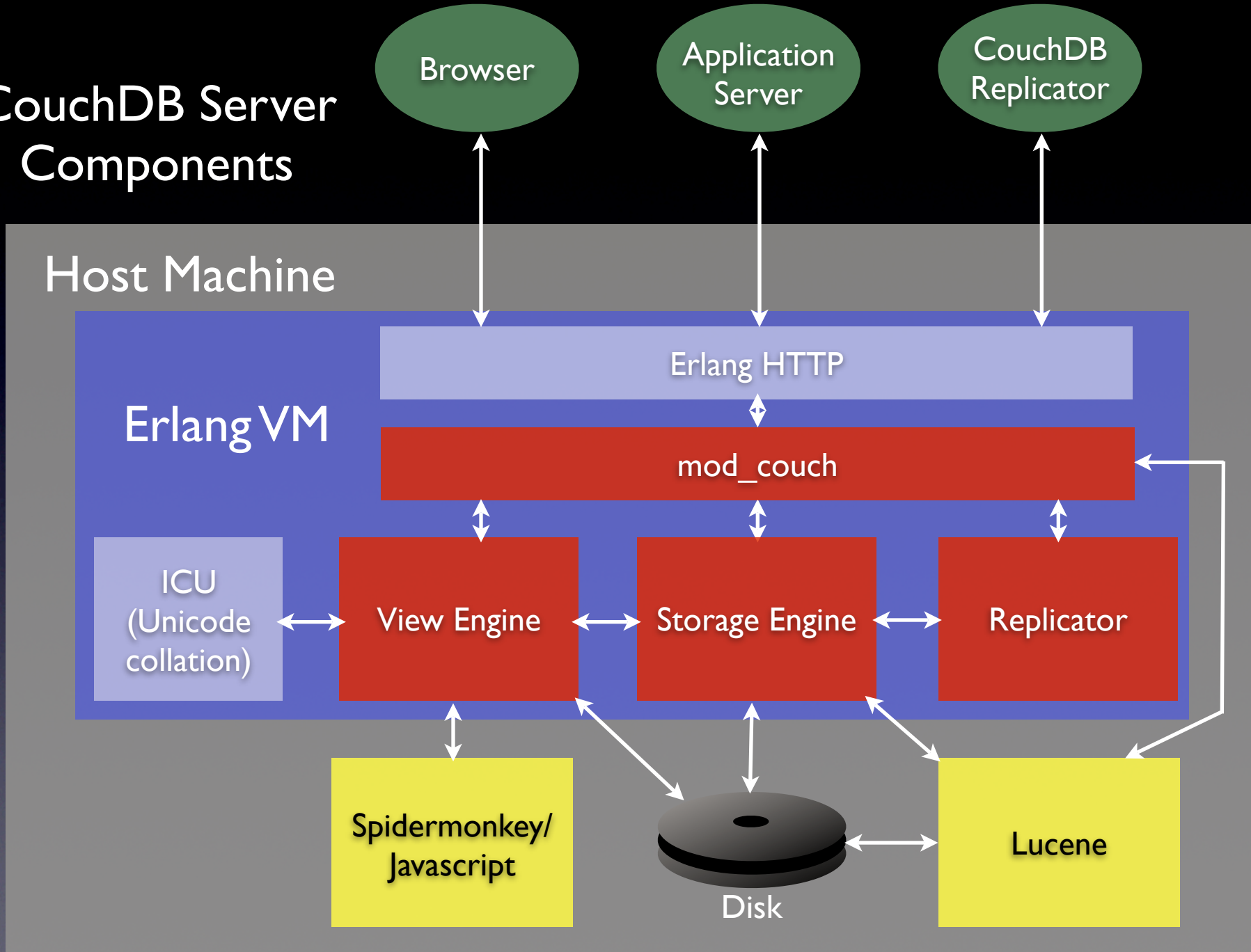
Full Text Indexing

- Experimental integration with Lucene
- Piped interface - any FT engine can be used
- Unlike RDBMS - pretty much just works. No reversing schema to get result “object”
- View intersection - Search only docs in view

Implementation

- Core written in Erlang/OTP
 - HTTP Server, storage, view, replication.
 - Custom storage engine. No Mnesia
 - Designed for telecom. Ericsson AXD301 Phone switch claims 99.99999999% uptime
- Javascript based test suit
- AJAX GUI Admin client
- GNU Build system

CouchDB Server Components



ACID

- MVCC - Postgres, Oracle, BerkleyDB
- Write-once storage resembles file journaling
 - ~~Wastes~~ Takes advantage of cheap disk space
 - Instant restart on crash
- No locking - optimistic commits
 - Attachments written concurrently in isolation, then committed serially

View Engine

- Disk layout and engine optimized for incremental indexing.
- MVCC lockless design, many simultaneous readers, even while actively updated.
- Can accommodate multiple query languages. Python and Ruby versions already exist
- language runtime/environment should be “hardened”

High Availability

C.O.U.C.H

- Work in progress
- Cluster Of Unreliable Commodity Hardware
- Erlang OTP, designed for hardware and software failure
 - Fail fast/crash-only design
 - Instant start after crash or power failure
- Replication makes clustering easy

Limitations

- Size of single file instance limited by OS
 - With node partitioning, unlimited
- No limit on number or size of documents
 - but doc more expensive than RDMS row
- Performance favors read heavy loads
- Testing: 400k docs and 6GB - No problems

Project Source

- Mozilla JavaScript engine aka SpiderMonkey (235 files)
- Internet functions implemented in Erlang (74 files)
- Config and build files (50 files)
- Web server content files (37 files)
- CouchDB engine (22 files)

Project Members

- Damien Katz - Core server code.
- Jan Lenhardt - Code, documentation, examples, community
- Christopher Lenz - AJAX Admin client
- Noah Slater - GNU build/deployment system

Community

- Growing Wiki documentation
 - Dozens of contributors
 - Examples in 10+ languages
- Growing list of client libraries
- Ajatus Project, open source distributed PIM/
Document management system.
- Actively submitting to Apache Incubator

Business Interest

- David Ascher - Head of Mozilla Thunderbird/Mailco
 - Core storage.
 - Strong interest, solves lots problems.
 - Flexible model, replication.



Core 1.0 Work Outstanding

- Live Compaction (recover wasted space)
- Security/Validation model
- User programmable Reduce (map/reduce)
- Performance/scalability work
- Admin tools and interfaces
- Finish full text search integration

Future work

- Partitioning/Horizontal scaling
 - Scale to Google sized databases
 - Data and query models designed from the start for this. Nothing inherently hard about scaling across machines, unlike SQL
 - Still, lots of engineering and testing necessary
- Client Side/Browser integration
 - Local server

Thank You