

John C. Zablocki
Development Manager, HealthcareSource
Organizer, Beantown ALT.NET
Philly Code Camp 2011.2
2011-10-15

.NET and NoSQL

Relaxing with CouchDB

Agenda

- NoSQL Overview
- CouchDB Basic Concepts
- CouchDB and cURL
- CouchDB and .NET
- LoveSeat
- Document Design Considerations
- Briefly: Meringue
- Questions?

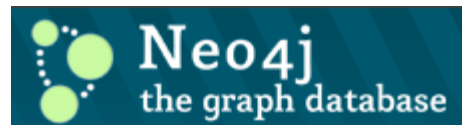
NoSQL

Not Only SQL

What is NoSQL?

- Coined in 1998 by Carlos Strozzi to describe a database that did not expose a SQL interface
- In 2008, Eric Evans reintroduced the term to describe the growing non-RDBMS movement
- Broadly refers to a set of data stores that do not use SQL or a relational data model
- Popularized by large web sites such as Google, Facebook and Digg

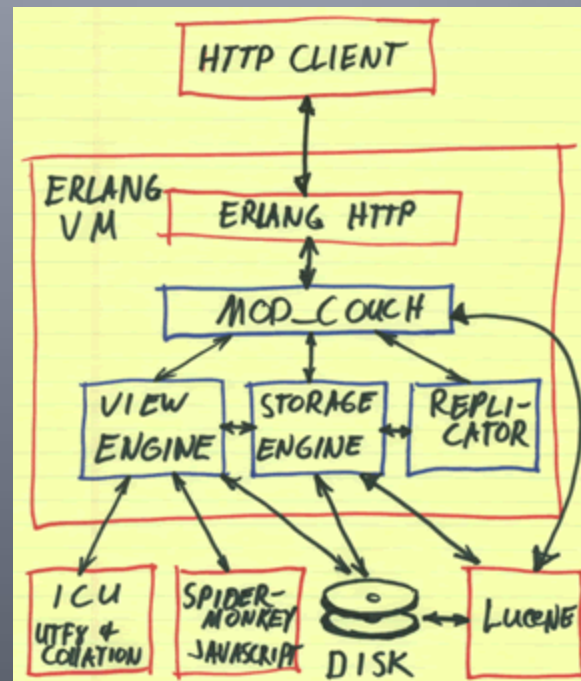
NoSQL Databases



NoSQL Databases

- NoSQL databases come in a variety of flavors
 - XML (myXMLDB, Tamino, Sedna)
 - Wide Column (Cassandra, Hbase, Big Table)
 - Key/Value (Redis, Memcached with BerkleyDB)
 - Object (db4o, JADE)
 - Graph (Trinity, neo4j, InfoGrid)
 - Document store (CouchDB, MongoDB)

Introducing CouchDB



About CouchDB

- Open source, Apache supported project
- Document-oriented database
- Written in Erlang
- RESTful API (POST/PUT/GET/DELETE) for managing CouchDB:
 - Servers
 - Databases
 - Documents
 - Replication
- Uses Multiversion Concurrency Control (MVCC)

CouchDB - Concepts

- Schema-less documents stored as JSON
- RESTful API for database and document operations (POST/PUT/GET/DELETE)
- Each document has a unique field named “_id”
- Each document has a revision field named “_rev” (used for change tracking)
- Related documents may have common “type” field by convention – vaguely analogous to collections or tables

Design Documents

- Design Documents special documents containing application logic
- Generally mapped to application boundaries (users, blogs, posts, etc.)
- Used to define views, shows, lists and validation functions, attachments, etc.
 - Views allow for efficient querying of documents
 - Show and List functions allow for efficient document and view transformations
 - Validation functions place constraints on document creation

Sample Design Document

```
{ "_id": "_design/artist",
  "validate_doc_update": "function(newDoc, oldDoc) { if (!newDoc.name) { throw({ forbidden : 'Name is required'}); } }",
  "shows" :
  {
    "csv" : "function(doc, req) { return doc._id + ',' + doc.name }"
  },
  "views":
  {
    "all" : {
      "map" : "function(doc) { emit(null, doc) }"
    },
    "by_name" : {
      "map" : "function(doc) { emit(doc.name, doc) }"
    },
    "by_name_starts_with" : {
      "map" : "function(doc) { var match = doc.name.match(/^{0,3}/i)[0]; if (match) { emit(match, doc) } }"
    },
    "by_tag" : {
      "map" : "function(doc) { for(i in doc.tags) { emit(doc.tags[i], doc) } }"
    },
  },
  "lists" :
  {
    "all_csv" : "function(head, row ) { while(row = getRow()) { send(row.value._id + ',' + row.value.name + '\\r\\n'); } }"
  }
}
```

Installing CouchDB on Windows

- Download an installer from <https://github.com/dch/couchdb/downloads>
- Download curl at <http://curl.haxx.se/download/curl-7.19.5-win32-ssl-sspi.zip>, unzip and set path
- Run the following from the command line
`curl.exe http://127.0.0.1:5984`
 - If all is running, response should be
`{"couchdb" : "Welcome", "version", "1.1.0"}`
- Check out http://wiki.apache.org/couchdb/Quirks_on_Windows for some gotchas

cURL Basics

- cURL is an open source, command line utility for transferring data to and from a server
- cURL supports all common Internet protocols, including SMTP, POP₃, FTP, IMAP, GOPHER, HTTP and HTTPS
- Examples:
 - curl <http://www.bing.com>
 - curl -F email=test@live.com <http://www.live.com>
 - curl -X GET <http://www.bing.com?q=couchdb>

cURL and CouchDB

- Check server version
 - curl <http://localhost:5984>
- Create database
 - curl -X PUT <http://localhost:5984/albums>
- Delete database
 - curl -X Delete <http://localhost:5984/cds>
- Get a UUID
 - curl http://localhost:5984/_uuids
- Create document
 - curl -X POST <http://localhost:5984/albums>
-d '{"artist": "The Decembrists"}'
-H "Content-Type: application-json"
- Get document by ID
 - curl <http://localhost:5984/artists/a10a5006d96c9e174d28944994042946>

CouchDB - Futon

The screenshot shows the CouchDB Futon web interface. The browser address bar displays the URL `127.0.0.1:5984/_utils/index.html`. The main content area is titled "Overview" and features a "+ Create Database ..." button. Below this is a table listing four databases:

Name	Size	Number of Documents	Update Seq
_replicator	8.1 KB	1	1
_users	4.1 KB	1	1
albums	8.1 KB	2	2
vtcodecamp	8.1 KB	4	6

Below the table, it indicates "Showing 1-4 of 4 databases" and provides navigation options: "← Previous Page", "Rows per page: 10", and "Next Page →".

The right sidebar contains the CouchDB logo with the tagline "relax". Under the "Tools" section, there are links for "Overview", "Configuration", "Replicator", "Status", and "Test Suite". The "Recent Databases" section lists "albums" and "vtcodecamp". At the bottom of the sidebar, a message reads "Welcome to Admin Party! Everyone is admin. [Fix this](#)". The footer of the sidebar indicates "Futon on Apache CouchDB 1.1.0".

CouchDB - Futon

- Futon is a simple web admin for managing CouchDB instances and is accessible at http://127.0.0.1:5984/_utils/
- Used for setting server configuration
- Allows for database administration (create/delete, compact/cleanup, security)
- Allows for CRUD operations on documents
- Creating and testing views
- Creating design documents

My Dog on a Couch



CouchDB .NET Client Libraries

- SharpCouch – simple CouchDB wrapper and GUI client. Last commit 2008
- Divan – Nearly API complete. Some LINQ support. Last commit 2010
- Relax – Built with CQRS consideration. Complex library. Recent commit (May 2011)

LoveSeat

- Document, View, List and Show API complete.
- Fluent HTTP API for non-implemented API features, such as creating design documents
- Support for strongly typed documents, using generics and Type convention
- Last commit August 2011 by jzablocki.

CouchDB LoveSeat – The Basics

```
private const string DESIGN_DOC = "artist";  
private const string DATABASE = "vtcodecamp";
```

```
private static CouchClient _couchClient = null;  
private static CouchDatabase _couchDatabase = null;
```

```
static Program() {
```

```
    //create the client and database, set the default design doc to "artist"  
    _couchClient = new CouchClient("127.0.0.1", 5984, null, null);  
    _couchDatabase = _couchClient.GetDatabase(DATABASE);  
    _couchDatabase.SetDefaultDesignDoc(DESIGN_DOC);
```

```
}
```

CouchDB LoveSeat – Design Doc

//Create map and reduce functions for tag counts

```
var design = string.Format(
    @"{{ ""_id"": ""_design/artist"",
    ""all"": {{
        ""map"": ""function(doc) {{ emit(null, doc) }}"
    }},
    ""by_name"": {{
        ""map"": ""function(doc) {{ emit(doc.name, doc) }}"
    }}
    });
```

```
var request= new CouchRequest("http://127.0.0.1:5984/music/_design/artist");
```

```
var response = request.Put().Form()
    .ContentType("multipart/formdata")
    .Data(JObject.Parse(design)).GetResponse();
```

CouchDB LoveSeat - CRUD

```
//Create POCO instance
var artist = new Artist() { Name = "The Decembrists", TourStops
= { "Boston", "Boston", "Hartford", "Burlington" } };

//Inserting a document into a typed collection
//- GUID Id will be created prior insert in property, not by driver
var result = _couchDatabase.CreateDocument(new Document<Artist>(artist));

//Updating (replacing) a document in a typed collection
//after creating document, doc rev is in result, but POCO not updated
artist.Rev = result["rev"].ToString();
artist.Name = "The Decemberists";
result = _couchDatabase.SaveDocument(new Document<Artist>(artist));

//Updating a nested collection
artist.Rev = result.Rev;
artist.Albums = new List<string>()
{ "Castaways and Cutouts", "Picaresque", "Hazards of Love", "The Crane Wife" };
result = _couchDatabase.SaveDocument(new Document<Artist>(artist));
```

CouchDB LoveSeat –CRUD

```
//Find all documents in a typed view
```

```
var artists = _couchDatabase.View<Artist>("all");
```

```
Console.WriteLine("Artist name: " + artists.Items.FirstOrDefault().Name);
```

```
//Find a single document by name
```

```
var options =
```

```
    new ViewOptions() { Key = new KeyOptions("The Decemberists") };
```

```
var artist
```

```
    = _couchDatabase.View<Artist>("by_name", options).Items.First();
```

```
Console.WriteLine("Album count: " + artist.Albums.Count);
```

```
//Count the documents in a view
```

```
long count = _couchDatabase.View<Artist>("all").Items.Count();
```

```
Console.WriteLine("Document count: " + count);
```

CouchDB LoveSeat - MapReduce

```
//Add some tags
```

```
var artist = _couchDatabase.View<Artist>("all").Items.First();  
artist.Tags = new List<string> { "Folk rock", "Indie" };  
_couchDatabase.SaveDocument(new Document<Artist>(artist));
```

```
//add a new artist with some tags
```

```
var newArtist = new Artist() {  
    Name = "Sunny Day Real Estate",  
    Albums = { "How it Feels to be Something On", "Diary" },  
    Tags = { "Indie", "Emo" },  
    TourStops = { "Boston", "Philadelphia", "Philadelphia", "Philadelphia",  
                 "New York", "New York", "Hartford" }  
};  
_couchDatabase.SaveDocument(new Document<Artist>(newArtist));
```

```
var options = new ViewOptions() { Key = new KeyOptions("Indie"), Group = true };  
var tagCounts = _couchDatabase.View("by_tag_count", options);  
Console.WriteLine("Indie tag count: " + tagCounts.Rows.First()["value"]);
```

CouchDB LoveSeat - MapReduce

```
//Create map and reduce functions
```

```
var map = @"function() {  
    if (!this.Tags ) { return; }  
    for (index in this.Tags) { emit(this.Tags[index], 1); }  
}";
```

```
var reduce = @"function(previous, current) {  
    var count = 0;  
    for (index in current) { count += current[index]; }  
    return count;  
}";
```

```
//Snippet below would be found in Design document, with  
//map and reduce replacing the format strings {0} and {1}
```

```
""by_tag_count"" : {{  
    ""map"" : ""{0}"" , ""reduce"" : ""{1}""  
}},
```

CouchDB LoveSeat – Group By

```
//add one more artist for good measure
_couchDatabase.CreateDocument(new Document<Artist>(
    new Artist() { Name = "Blind Pilot",
        Albums = { "3 Rounds and a Sound" },
        TourStops = { "Philadelphia", "Providence", "Boston" } }));

var tourStopGroupBy = _couchDatabase.View("by_tour_stop",
    new ViewOptions() { Group = true });

Func<JToken, string> stripQuotes = (j) => j.ToString().Replace("\"", "");
foreach (var row in tourStopGroupBy.Rows) {
    Console.WriteLine("{0} played {1} {2} time(s)"
        , stripQuotes(row["key"][1]), stripQuotes(row["key"][0]), row["value"])
    }
}
```

CouchDB LoveSeat - Group By

```
var tourStopMap = @"function(doc) {  
    for(i in doc.tourStops) {  
        emit([doc.tourStops[i], doc.name], 1) }  
    }";
```

```
var tourStopReduce = @"function(keys, values) { return sum(values) }";
```

```
//Snippet below would be found in Design document, with  
//map and reduce replacing the format strings {0} and {1}  
""by_tour_stop_and_name"": {{  
    ""map"": ""{0}"" , ""reduce"": ""{1}""  
}},
```

CouchDB LoveSeat – View Queries

```
//Find items in typed collection
```

```
var options = new ViewOptions() { Key = new KeyOptions("The") };
```

```
var artistsStartingWithThe =
```

```
_couchDatabase.View<Artist>("by_name_starts_with", options);
```

```
    Console.WriteLine("First artist starting with The:
```

```
" + artistsStartingWithThe.Items.First().Name);
```

```
//Find artists with a given tag
```

```
options = new ViewOptions() { Key = new KeyOptions("Indie") };
```

```
var artistsWithIndieTag = _couchDatabase.View<Artist>("by_tag", options);
```

```
foreach (var artist in artistsWithIndieTag.Items) {
```

```
    Console.WriteLine("Found artist with indie tag:
```

```
" + artist.Name);
```

```
    }
```

CouchDB LoveSeat – Show and List

```
var artist = _couchDatabase.View<Artist>("all").Items.First();
var csv = _couchDatabase.Show("csv", artist.Id.ToString());
Console.WriteLine("Show: {0}", csv);

var csvList = _couchDatabase.List("all_csv", "by_tag",
    new ViewOptions() { Key
        = new KeyOptions("Indie") });
Console.WriteLine("List:
{0}", csvList.RawString.Split(Environment.NewLine.ToCharArray(),
StringSplitOptions.RemoveEmptyEntries).First());
```

Briefly: Meringue

[Sign In](#) or [Register](#)

Meringue

When life gives you lemons, make meringue...



Recent Posts

[On Defining ALT.NET](#)

I had the opportunity to be part of an open spaces session at the Philly.NET Code Camp this past weekend. Part of the conversation centered around defining what ALT.NET really means or more specifically, what does it mean to be an ALT.NET developer. It's pretty clear that as ubiquitous as the term is, its definition is still largely subjective. Though I'm hardly the authority on this bit of lexicography, I'll take a stab...

[Grunge.NET](#)

I'm a huge fan of early 90s Seattle grunge rock. Though most of these acts had clear roots in acts such as Led Zeppelin and Jimmy Hendrix, they were often simply labeled "Alternative Rock." I think the whole ALT.NET label is pretty similar. After all, techniques and technologies such as DDD and ORMs have deep roots in late 90's OOPD patterns. Yet, the ALT.NET label gets applied, probably because most .NET devs grew up in the Microsoft space, where data-centric, procedural patterns were commonplace

[ALT.NET is Elitist](#)

This is a charge I hear often in podcasts, blogs and even in the open spaces event. Though I disagree that ALT.NET is fundamentally elitist, there is merit to this notion. In some ways, ALT.NET is a reaction to the often thoughtless ways .NET developers work. I've interviewed hundreds of developers over the past few years and I'm generally shocked at how many .NET devs work in drag and drop shops. I've also found that most .NET devs don't worry about things like open source, patterns or OOPD.

In no way am I trying to put down the community at large nor am I attempting to be elitist myself. The point is that there are a lot of devs, particularly in the .NET space, who don't think out of the toolbox. The ALT.NET movement evolved in part as a reaction to this behavior. ALT.NET developers learned from other platforms and languages and sought to use tools that didn't come from Redmond. ALT.NET developers don't typically like drag and drop. So by virtue of being a reaction against the way people work, I believe a lot of people feel they are the target of some dev angst.

Another reason I think devs feel ALT.NET is elitist has to do with the fact that many popular ALT.NET bloggers are quite opinionated. Let's face it, a lot of proponents of ALT.NET techniques and technologies often speak as if they have discovered nirvana (the metaphysical state, not the grunge band). I mean, even the language of DDD feels a bit elitist if not weird (ubiquitous language, really?). Even worse, are developers who treat ALT.NET as if it were some sort of scripture, blindly applying patterns to problems they don't understand. Being preachy only works in church. When I interviewed those candidates who knew nothing of common architectural patterns or OOP idioms, I took the opportunity to share what I've learned to give them another perspective. But, I hope, I never made them feel stupid for not knowing how to configure Spring.NET's XML files.

[What Makes an ALT.NET Developer an ALT.NET Developer?](#)

Contrary to popular belief, there is no ALT.NET stack to know. We have no secret handshake to join our society.

Tags

[alt.net](#) [mapreduce](#)
[mercurial](#)
[mongodb](#)
[norm](#)

[All Tags](#)

Design Considerations

- Your object graph is your data model
- Don't be afraid to store data redundantly
 - Your graph might be redundant!
- Not everything has to fit in 1 document
- Don't be afraid to store aggregate statistics with a document.

Object Graph as Data Model

- Generally speaking, most MongoDB drivers will serialize an object graph as a single document
 - The relationships of your classes creates an implied schema!
 - Migrating this schema is not trivial if you are trying to deserialize properties that did not or no longer exist
- Consider use cases carefully to avoid inefficiently structured documents
- Projection queries will be your friend

Redundant Data is OK

- Optimize documents for quick reads and writes
- Your application layer will have to maintain referential integrity!
- If every time you access a Post document, you need some of an Author document's data, store that data with Post
- Design simple classes for this redundant data for reusability (see AuthorInfo in Meringue)

Don't Stuff it All In One Document

- Nothaving formal relationships does not mean throwing away relationships
- Consider a user and his or her logged actions
 - The user would likely have a User class/doc with properties for name, email, etc.
 - User actions are generally write heavy and read out of band.
 - Don't clutter user documents - create a separate collection for user actions

Don't Be Afraid of Extra Data

- The schema-less nature of documents makes it easy to store meta data about that document – particularly aggregate data
- Consider a blog post with a rating feature
 - Each rating would be stored as a nested document of the post
 - Rather than compute vote totals and averages real time, simply add these properties to the document and update on writes

Final Thoughts

God is Dead.

- Nietzsche

Final Thoughts

Codd is dead.

- John Zablocki

Links

- <http://dllHell.net> - my blog
- <http://www.CodeVoyeur.com> - my code
- <http://www.linkedin.com/in/johnzablocki>
- <http://twitter.com/codevoyeur>
- <http://couchdb.org> - Official CouchDB site
- <http://guide.couchdb.org/> - Free eBook
- <http://bitbucket.org/johnzablocki/meringuecouch>
- <http://bitbucket.org/johnzablocki/codevoyeur-samples>
- <http://about.me/johnzablocki>

Questions?
