

06/13/09 - Going Camping with CouchDB On OS X Tiger



Hello there! If you are new here, you might want to **subscribe to the RSS feed** for updates on this topic. With Snow Leopard being released at the end of September this year, Tiger is definitely starting to show it's age. I've been putting off upgrading to Leopard for quite some time, not being able to justify the down time associated with upgrading an OS. This has had an unpleasant side effect of not being able to do things quite as easily as I want, and finding information is hard. Due to the recommendations in the comments of my Ruby CMS article, I have decided to start playing with the Camping Ruby microframework backed by CouchDB. Getting Camping up and running was easy enough, and I managed to get some trivial functionality going. Camping uses SQLite for it's database, which is fine if I wanted to massage my data into a relational form. However, I have models with optional attributes, and rather than making nullable database types, I want to leverage CouchDB's schema-less, RESTful document storage. Each persisted object in my application will be available on a unique url, so CouchDB is a nice fit for what I want to do.

The Great Installation

I found this great article about getting Camping talking to CouchDB, and it seemed easy enough, so I decided to try it. Since I already had Camping working, all I needed was to get CouchDB installed on my Macbook. As much as I'm not scared of compiling applications from source, I've had issues in the past with OS X and compiling, so I decide to find a simple package or disk image to install. I found CouchDBX, however checking the requirements showed that it supports Leopard only.

With a little more research, I found out I had to use Macports to install CouchDB. Macports relies on XCode Tools which come available on the DVDs that came with OS X. If you haven't previously installed XCode Tools, then you should install them before you install Macports. Something that is buried in a support ticket, however, is that Macports targets XCode tools 2.5. My XCode tools was something like 2.4.8. While I did manage to get Macports installed, the installation for CouchDB fails when it tries to build tk. If you, like me, have an older XCode Tools than 2.5, you'll need to download 2.5, which you can find deep inside the Apple website. This does require an Apple Developer Connection account to get, so if you don't have one,

you'll need to create one.

Installing CouchDB is as simple as issuing

```
Arraysudo port install couchdbArray
```

and Macports will take care of the rest. CouchDB lists it's dependencies as Spidermonkey, Erlang and a few others, however each of those has dependencies, and each of those dependencies has dependencies, and it ends up taking quite a long time and downloading a fair few packages. The upside is that if you didn't have Erlang previously installed, you do now. Erlang is something I've been wanting to take a look at.

Start The Server Before You Leap

While the article I linked to above about getting Camping and CouchDB talking is good, it's not exactly the most verbose explanation especially if you haven't read the documentation for CouchDB and prefer to just dive in. For those who are still a little unsure like I was about how exactly to do this, I'll outline my approach below.

Before you get started writing any code at all, you need to start your CouchDB server. This is something that escaped me for the longest time, and when I finally twigged, it gave me a bit of grief. To start CouchDB, issue the following command:

```
Arraysudo couchdbArray
```

If you try this without escalating privilege, it will return an error `"{init terminating in do_boot",{{badmatch,{error,shutdown}},[{{couch_server_sup,start_server,1},{erl_eval,do_apply,5},{erl_eval,exprs,5},{init,start_it,1},{init,start_em,1}}]}`", which searching for will bring you to the [CouchDB](#) wiki page for error messages. This page will tell you the problem is an unavailable port. For me, at least, this is not true. CouchDB runs on port 5984 which as far as I know is not used for any system services or other software. The reason I was getting this error was that I wasn't running it with enough privilege.

Once you get CouchDB started, you'll notice it blocks the terminal. I don't like to have too many windows open at once, so I'd prefer to have it run as a background process. Fortunately CouchDB will let you do that with a command line switch option, along with changing the location of the pid file. I'll leave figuring out how you

want to start it up to you.

Once you have CouchDB started, you'll have access to a web-based administration panel called Futon. Futon is available to you at `http://localhost:5984/_utils/`

, assuming you're running CouchDB on your local host. The Futon utility will allow you to create databases and insert documents in preparation to test your connectivity with Camping.

Camping Time

There were two Ruby gems I had considered when deciding how to connect Ruby to CouchDB. I should point out that due to the RESTful nature of CouchDB, you strictly don't need any Ruby gems and could roll your own fairly easily, if you wanted to. I didn't want that level of control, personally. The two gems I considered were CouchRest and RelaxDB. CouchRest is a simple Ruby wrapper around the CouchDB REST API, whereas RelaxDB is more abstracted, bringing ActiveRecord-like functionality to CouchDB. While before I would have chosen RelaxDB, I'm intentionally trying to work a little closer to the raw APIs here, so I chose CouchRest.

You can install CouchRest through RubyGems:

```
Arraysudo gem install couchrestArray
```

however I found that the gem version wouldn't work and kept throwing errors when I started my Camping application. To overcome this, I just cloned it from the GitHub repository and placed it in my Camping app directory and referenced it that way.

The first thing to do would be to create a normal Camping sample application, if you haven't done already. For the purposes of this post, I'm going to use the "skeletal Camping blog" application used in the documentation, and edit that to work with CouchDB. I figure this will give everyone a relatively common base from which to start. Import CouchRest just below where the application imports Camping:

```
Arrayrequire 'camping'Arrayrequire 'couchrest' # or
'couchrest/couchrest' or similar if you've cloned from
GithubArray
```

Given that the purpose of this article is to demonstrate connecting to CouchDB, and not the design of a framework around it, we can safely ignore defining a model for now, and rely on CouchDB as our model. So if you're following along from the example application, you can safely delete the classes from `Blog::Models` module. We're not going to delete the whole module, because of the `create` method.

The Camping `create` method that is run when the server starts up your Camping app. From the documentation, "This is a good place to check for database tables and create those tables to save users of your application from needing to manually set them up." Instead, we're going to use this method to set up CouchRest in our application:

```
Arraymodule Blog::ModelsArray  def Blog.createArray      db_url
= 'http://localhost:5984/'Array      storage =
CouchRest.database("#{db_url}blog")Array
Blog::Controllers::Index.set_storage(storage)Array
endArrayendArray
```

Next, because we're now calling a new method on the controller class, we need to modify that. Change the `Blog::Controllers::Index` class to the following:

```
Arraymodule Blog::ControllersArray  class Index < R
'/(\\w+)'Array      def Index.set_storage(storage)Array
@@storage = storageArray      endArray      def get(id)Array
@post = @@storage.get('posts/' + id)Array      render
:indexArray      endArray  endArrayendArray
```

What I've done here is given the controller class a static reference to my CouchDB database (from the `create` method). Then the controller was changed to respond to a regex, which is passed into the `get` method. We use the `id` passed in to retrieve a document from CouchDB, which we assign as an instance variable.

Lastly, we need to modify the `Index` function view to pull data from the CouchDB database:

```
Arraydef indexArray  h1 @post[:title]ArrayendArray
```

This will output the title of the test document created at the start of the article.

If you haven't already, start up Futon and create a database for the purposes of this demo. Call the database "blog", and fill it with a test document with the id of "posts/test", and at least a "title" property with the value "Test Post", and any other properties you wish.

Start the Camping application with `camping blog.rb`, and point a browser to `http://localhost:3301/test`, and you should see "Test Post" in a header tag, rendered out to the browser, which means that Camping has successfully communicated with CouchDB.

I didn't cover putting data into CouchDB, however you would do it in a similar vein, writing a `post` method inside your routes to create data and save it to CouchDB. From here, personally, I'm going to create some wrapper classes for requests and models and build up something which is a little DRYer, however as far as a demo, this is good enough.