The Compleat CouchDB in 10¾ Pages

Peter Lavin

March 8, 2011

Table of Contents

What is CouchDB?

Okay, perhaps the title is a slight exaggeration. But because of CouchDB's ease of use, 10¾ pages (the length of this article in PDF format) gets you a long way. You'll learn how to create a database, bulk load data into it, reconfigure the server for remote access, replicate to an online server, create a view *and* run the server as a daemon. You'll learn how to do this not just from the web UI but also from the command line. That's pretty compleat, especially in 10¾ pages.

But to answer the question posed by the title of this section: CouchDB is a NoSQL database, a project of the Apache software foundation. It's easy to install and comes with its own administration client called "Futon". You can interface with it from the command line through its REST API, by running the CouchDB application or through your favourite browser. If you use the binaries provided by CouchOne you can easily install CouchDB on Mac OS X, Ubuntu or Windows. This means you can quickly check it out without having to mess around with failed dependencies or compiling from source.

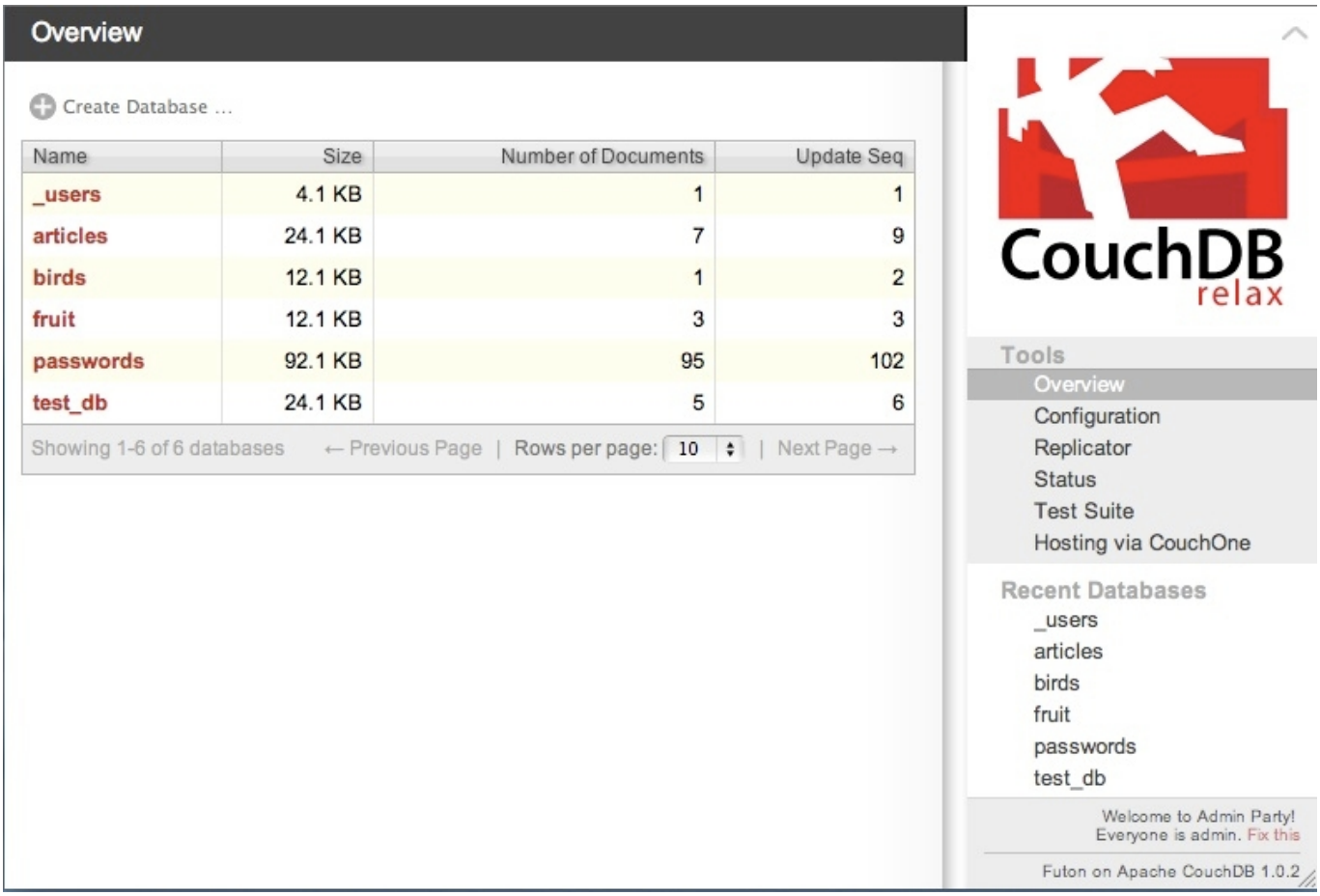
This article deals with CouchDB on Mac OS X. However, apart from file paths and the section on running CouchDB as a service, this article only requires an operating system that supports command line use of curl. Using Futon is a quick way to become familiar with CouchDB and using curl commands to access the REST API is a good, language-neutral way to get a feel for how you can access CouchDB from your preferred programming language.

Exploring the UI

To download and installed CouchDB you just need to point your browser at the url, CouchOne. Download and unzip the file and install the application as you would any other.

Start up CouchDB and you'll see something similar to the following:

Figure 1. The Futon UI



The application opens on the Overview page. From this page you can view all databases and navigate to a specific database by clicking the database name.

On the right sidebar are links to other Futon pages. We'll mostly be concerned with the Overview, Configuration and Replicator links.

Using Futon is an easy way to perform tasks such as:

- Creating a database – perform this task from the Overview page by clicking the Create Database link.
- Entering a record – Once you've created a database, click it and then choose the New Document option.
- Viewing records – Clicking a database lists all its records. Clicking a record displays its details.
- Editing the configuration – Choose Configuration from the sidebar and view the configuration options

- Replicating a database – Choose Replicator from the sidebar and follow the instructions.
- Creating a view – Select a database from the Overview page and then Design Documents from the View drop-down list box. You'll probably want to read on before making any changes here.

Futon is particularly good for some tasks and not quite so good for others—you may already have a sense for which. What it's very good for is giving you a feel for CouchDB. Play with it.

The REST API from the Command Line

This section shows you how to use curl to interface with the REST API. It can serve as a quick reference for basic syntax.

One of the first things you'll want to do is check that you can communicate with the server. You can do this by sending a curl GET to localhost on the default port for CouchDB (the `-X GET` is optional):

```
shell> curl -X GET http://127.0.0.1:5984
{"couchdb":"Welcome","version":"1.0.2"}
```

The JSON response shown above indicates a successful connection to the server.

> **Note**
>
> In this article, all curl commands are followed by the output that is returned. In some cases this output is formatted for easier reading.

Creating a database is as simple as the following PUT:

```
shell> curl -X PUT http://127.0.0.1:5984/test_db
{"ok":true}
```

Inspect your newly created database in the following way:

```
shell> curl http://127.0.0.1:5984/test_db
{"db_name":"test_db","doc_count":0,"doc_del_count":0,
"update_seq":0,"purge_seq":0,"compact_running":false,"disk_size":79,
"instance_start_time":"1298762620491062","disk_format_version":5,
"committed_update_seq":0}
```

Once you've created a database, you'll want to add a record. There are no tables within a CouchDB database—it's not a relational database—so add a record directly to the database by issuing the following command:

```
shell> curl -X POST http://127.0.0.1:5984/test_db/ \
  -H 'Content-Type: application/json' \
  -d '{"name":"blue jay", "location":"Malton"}'
{"ok":true,"id":"e1595400fcee306e82219bb0d400068c",
  "rev":"1-3eab48829ff6a6882f8d049456fa9e21"}
```

By using `POST` you create a new document with a server-generated ID. Notice the Universal Unique Identifier (UUID) in the message returned above. The id field of a record is similar to the autoincrement field used by other databases. However, since there are no tables in a CouchDB database, it makes more sense to use UUIDs rather than incremented integers.

You can use `PUT` to create a document when you do not want an autogenerated ID. Do this by appending the desired ID to the database URI as shown below:

```
shell> curl -X PUT http://127.0.0.1:5984/test_db/fruit \
  -H 'Content-Type: application/json' \
  -d '{"name":"granny smith", "type":"sour"}'
{"ok":true,"id":"granny_smith","rev":"1-a9a02531ffa21cf0da0f9e49658ef642"}
```

Notice that the id is specified as part of the URI rather than being autogenerated.

> **Note**
>
> You can also create an id by including it in the JSON object as the value associated with the special `_id` field.

To view all the records in a database, use the following syntax:

```
shell> curl -X GET http://127.0.0.1:5984/test_db/_all_docs
{"total_rows":2,"offset":0,"rows":[
{"id":"820510f01e98a2a20dcffdb8f0000052","key":"820510f01e98a2a20dcffdb8f0000052",
  "value":{"rev":"1-e49ecdd681345e490f1061ecd54d06dc"}},
{"id":"fruit","key":"fruit","value":{"rev":"1-c7410567a14b274b7b931674520082de"}}
]}
```

Look at the output of the `id` field and you can readily see the difference between an autogenerated id field and one that has been specified.

Using the REST API `test_db/_all_docs` as shown above doesn't retrieve data in the manner of a `SELECT` statement such as `SELECT * FROM tblname;`. To make a "wildcard" selection from a database add the `include_docs=true` query parameter:

```
shell> curl -X GET http://127.0.0.1:5984/test_db/_all_docs?include_docs=true
{"total_rows":2,"offset":0,"rows":[
{"id":"820510f01e98a2a20dcffdb8f0000052","key":"820510f01e98a2a20dcffdb8f0000052",
"value":{"rev":"1-e49ecdd681345e490f1061ecd54d06dc"},
"doc":{"_id":"820510f01e98a2a20dcffdb8f0000052",
"_rev":"1-e49ecdd681345e490f1061ecd54d06dc",
"name":"blue jay","location":"Malton"}},
{"id":"fruit","key":"fruit",
"value":{"rev":"1-c7410567a14b274b7b931674520082de"},
"doc":{"_id":"fruit","_rev":"1-c7410567a14b274b7b931674520082de",
"name":"granny smith","type":"sour"}}
]}
```

Adding Data

It's easy to add single records to a database using Futon but that's not the ideal way to add records to a database especially if you have existing data that you would like to bulk load. You can't bulk load data from the web UI. You must use the REST API directly.

Importing Data

You can bulk load data using the bulk document API which simply requires that you append `_bulk_docs`

to the base database URI.

To test use of this method, create a file named `mydata.json` in the following format:

```
{
  "docs": [
    {"url":"example.com","user":"fred","password":"secret", "type":"personal"},
    {"url":"other.example.com","user":"admin","password":"super_secret",
      "type":"personal"}
  ]
}
```

Using the same `test_db` database, try bulk loading from the command line by issuing the following curl command:

```
shell> curl -X POST http://127.0.0.1:5984/test_db/_bulk_docs \
 -H "Content-type: application/json" -d @mydata.json
```

When using **curl** with the `-d` option you must precede the data file name with the '`@`' character.

The data file does not contain an `_id` field so UUIDs will be generated for the records in the `mydata.json` file.

Exporting Data From MySQL

You may well want to export data from another database for bulk loading into CouchDB. This section describes how to export data from MySQL—a quick way to get up and running.

You can export data from a MySQL database in JSON format using a SQL statement such as the following:

```
SELECT CONCAT("{\"field1_name\":\"",field1,"\",\"field2_name\":\"",field2, "\",
  \"field3\":\"",field3,"\",\"field4\":\"",field4,"\"},")
  FROM tablename
    INTO OUTFILE '/var/tmp/myout.json';
```

Note

If any of your data fields contain quotation marks they will need to be escaped.

Given a table with the column names, `url, type, username` and `password` the output will look something like this:

```
{"url":"http://localhost","type":"personal","username":"peter","password":"secret"},
{"url":"http://example.com","type":"work","username":"root","password":"secret"},
...
```

To load this data file as described in the section called "Importing Data", you'll need to adjust the output file and place its contents inside `{"docs": [` *file_contents_here* `]}`. As you've already seen, the bulk load REST API call is as follows:

```
shell> curl -X POST http://127.0.0.1:5984/db_name/_bulk_docs \
  -H "Content-type: application/json" -d @myout.json
```

Remember that a unique ID will be created if you do not include one. If your existing database table contains a unique ID that you would like to use in CouchDB, export data using the field name `_id` for that unique ID field.

> **Note**
>
> You can nest a `GROUP_CONCAT` statement inside a `CONCAT` statement to create a complete file that doesn't need to be adjusted manually but data will be truncated unless you change the MySQL `group_concat_max_len` option from its default, `1024`. However, this technique only makes sense for small data sets.

Replicating

Easy replication is one of the attractive features of CouchDB and once you have some data in a database you'll want to try replicating that data.

By default, CouchDB is configured for local access only so let's set up CouchDB for remote access because replicating to or from a remote database is much more fun.

Remote Access

If you want to replicate to a server other than one running on your local machine, you'll have to change the default CouchDB configuration. In the default configuration the `bind_address` option is set to `127.0.0.1`. This setting limits REST requests to the local machine. If you change the `bind_address` option to `0.0.0.0`, then you'll be able to access CouchDB remotely.

You can easily modify this option through Futon. Click the `Configuration` tool on the right sidebar and scroll down until you find the `httpd` section. Double click the value opposite `bind_address` and replace it with `0.0.0.0`. Once you've made this change you can access CouchDB using the IP address or hostname of the machine hosting CouchDB. You can also still use `127.0.0.1` or `localhost` to access it locally. Note: Restart CouchDB after making this change.

To change the configuration from the command line open `/Applications/CouchDBX.app/Contents/Resources/couchdbx-core/couchdb_1.0.2/etc/couchdb/local.ini` in your favourite text editor, locate the `[httpd]` section and adjust the value of the `bind_address` option.

> **Note**
>
> Once you allow remote access it may be time to end the "Admin Party". With Futon open, click the Fix this link on the lower right to add a username and password. You can also do this by altering the `local.ini` file.

Replicating on a LAN

With remote access enabled you can now replicate a database across your LAN. You will, of course, need to

have CouchDB running on another machine on your network. Install CouchDB elsewhere on your network. You can use the web UI to perform the replication. The following command shows how to do this from the command line:

```
shell> curl -X POST http://127.0.0.1:5984/_replicate \
 -d '{"source":"test_db", "target":"http://192.168.0.196:5984/test_db"}' \
 -H "Content-type: application/json"
{"ok":true,"session_id":"c5fde0fc0905238800cc89f03f1c2fb8",
"source_last_seq":103,"history":[{"session_id":"c5fde0fc0905238800cc89f03f1c2fb8",
"start_time":"Sun, 27 Feb 2011 14:55:06 GMT","end_time":"Sun, 27 Feb 2011 14:55:06
GMT",
"start_last_seq":0,"end_last_seq":103,"recorded_seq":103,"missing_checked":0,
"missing_found":99,"docs_read":99,"docs_written":99,"doc_write_failures":0}]}
```

Add `"continuous":true` to the JSON object if you want the database to update continuously.

> **Note**
>
> Continuous updates work unless the server stops. You also might want to read up about push versus pull replication (http://wiki.apache.org/couchdb/Replication).

Replicating to an Online Database

What's even more fun than replicating a database locally is replicating to an online database. You can do this for no charge by signing up for CouchOne hosting at http://www.couchone.com/get. Do this by clicking the Hosting via CouchOne link on the right sidebar.

From the command line you can replicate a local database to a CouchOne online database in the following way:

```
shell> curl -X POST http://127.0.0.1:5984/_replicate  \
 -d '{"source":"test_db",
"target":"http://admin:password@yourUI.couchone.com/test_db"}' \
 -H "Content-type: application/json"
  {"ok":true,"session_id":"21dcf771f6c780bec0223f4dd6feeb55","source_last_seq":1,
  "history":[{"session_id":"21dcf771f6c780bec0223f4dd6feeb55",
  "start_time":"Sun, 27 Feb 2011 21:14:01 GMT","end_time":"Sun, 27 Feb 2011 21:14:01
GMT",
  "start_last_seq":0,"end_last_seq":1,"recorded_seq":1,"missing_checked":0,"missing_fou
nd":1,
  "docs_read":1,"docs_written":1,"doc_write_failures":0}]}
```

> **Note**
>
> The preceding curl POST assumes that a database named `birds` already exists locally and online. If the online database does not yet exist, you can create it during replication by adding `"create_target":true` to the JSON object. Also, the online database requires that a username and password be included as part of the URI. Replace `admin:password@yourUI` with appropriate vales.

Use your browser to navigate to the CouchOne web interface for your online database and view the replicated database. That was easy. No master, no slave. Just democratic, peer-to-peer replication.

Creating Views

If CouchDB is a NoSQL database with no tables, how do you look at your data? Views substitute for the `SELECT` statements used in a Relational Database Management system (RDBMS). You can create temporary or permanent views.

From the command line create a temporary view in the following way:

```
shell> curl -X POST http://127.0.0.1:5984/test_db/_temp_view  \
  -d  '{"map":"function(doc) { emit(null, doc);}"}'  -H "Content-type:
application/json"
{"total_rows":2,"offset":0,"rows":[
{"id":"820510f01e98a2a20dcffdb8f0000052","key":null,
  "value":{"_id":"820510f01e98a2a20dcffdb8f0000052",
  "_rev":"1-e49ecdd681345e490f1061ecd54d06dc",
  "name":"blue jay","location":"Malton"}},
{"id":"fruit","key":null,
  "value":{"_id":"fruit",
  "_rev":"1-c7410567a14b274b7b931674520082de",
  "name":"granny smith","type":"sour"}},
]}
```

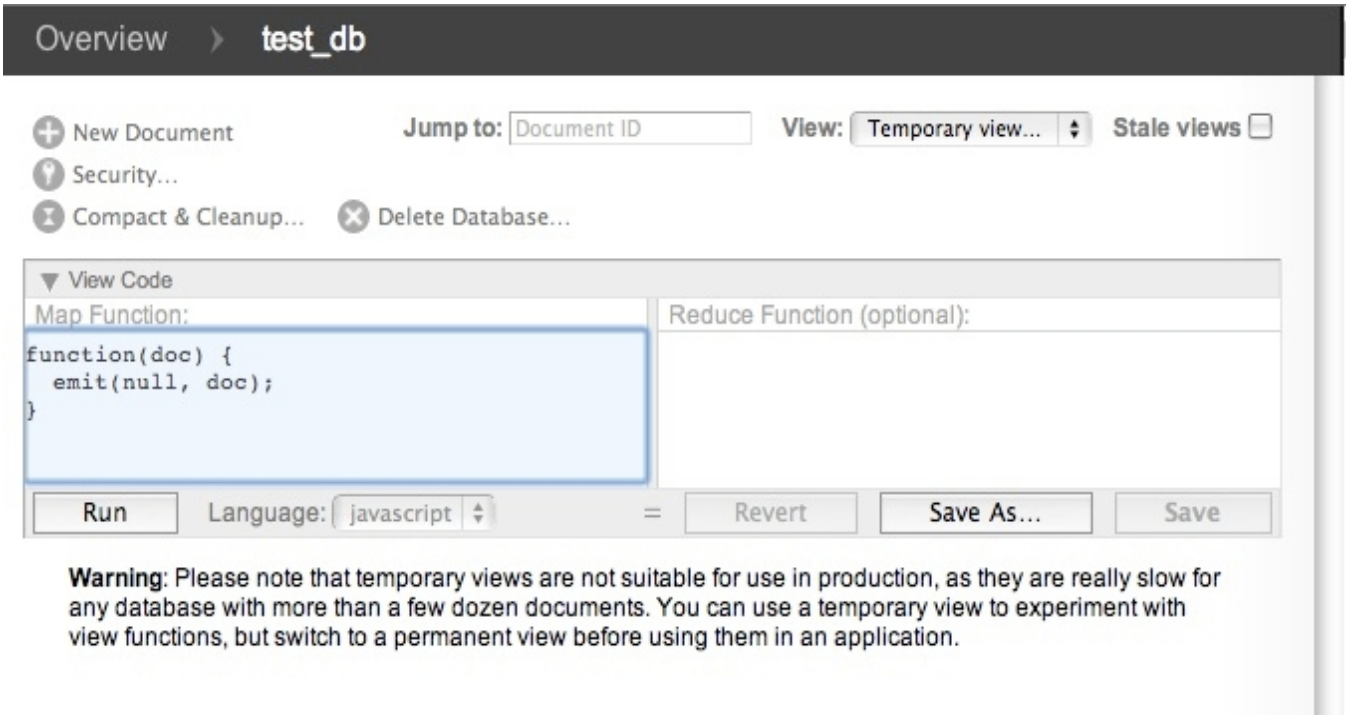
If your syntax is incorrect, as it can quite easily be with even simple views, you might see a message such as:

```
{"error":"bad_content_type","reason":"Content-Type must be application/json"}
```

If you see such a message, check quotation marks and open and closing braces.

Whereas bulk-loading is a task for the REST API, creating views is most easily done from the web UI. With Futon open, select a database and choose `Temporary view` from the `View` list box. You should see something like the following:

Figure 2. Designing a view



The View Code box shows a map function written in JavaScript that will be invoked against every record in the database. The `doc` parameter passed in is a single document in the database and `emit` is a built-in function that takes a key and a value argument. Consequently, this default map function outputs each document in the current database. Add the line `if(doc._id=="fruit")` before `emit(null, doc);` and, if you entered data as described in the section called "The REST API from the Command Line", you will see the record with the id field `fruit` when this view is run.

You can turn temporary views into permanent views by clicking the Save As button. You'll be asked to provide a design document name and a view name. If you created a view for the `test_db` database with the design document name `id` and the view name `fruit`, you can invoke it from the command line like so:

```
shell> curl -X GET http://127.0.0.1:5984/test_db/_design/id/_view/fruit?
limit=11&descending=true
[1] 3534
macbook:articles peterlavin$ {"total_rows":1,"offset":0,"rows":[
{"id":"fruit","key":null,"value":{"_id":"fruit",
  "_rev":"1-c7410567a14b274b7b931674520082de",
  "name":"granny smith","type":"sour"}}
]}
```

Running CouchDB as a Service

Once you've got things up and running and are familiar with CouchDB, you'll want to run it in the background as a service. On Mac OS X this means using **launchctl** with a plist file. There is an existing plist file called `org.apache.couchdb.plist` and it's found in the `/Applications/CouchDBX.app/Contents/Resources/couchdbx-core/couchdb_1.0.2/Library/LaunchDaemons` directory. Copy this file to your home directory so that you can easily make changes to it. The `org.apache.couchdb.plist` file is reproduced below:

```
  <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
  "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>Label</key>
    <string>org.apache.couchdb</string>
    <key>EnvironmentVariables</key>
    <dict>
      <key>HOME</key>
      <string>~</string>
      <key>DYLD_LIBRARY_PATH</key>
      <string>/opt/local/lib:$DYLD_LIBRARY_PATH</string>
    </dict>
    <key>ProgramArguments</key>
    <array>
      <string>/Users/jan/usr/src/couchdbx-core/dist/couchdb_1.0.2/bin/couchdb</string>
    </array>
    <key>UserName</key>
    <string>couchdb</string>
    <key>StandardOutPath</key>
    <string>/dev/null</string>
    <key>StandardErrorPath</key>
    <string>/dev/null</string>
    <key>RunAtLoad</key>
    <true/>
    <key>KeepAlive</key>
    <true/>
  </dict>
```

```
</plist>
```

You can't use this file "as is". Make the following changes:

- The `EnvironmentVariables` key is not required so remove that key and its dictionary.
- Locate the couchdb startup script on your computer. It should be in the `/Applications/CouchDBX.app/Contents/Resources/couchdbx-core/couchdb_1.0.2/bin` directory. Change the string in the `ProgramArguments` key array to this value and also change the `UserName` key string to your system username.
- One final change is required because the **couchdb** script expects to be run from the `/Applications/CouchDBX.app/Contents/Resources/couchdbx-core` directory. Add a `WorkingDirectory` key immediately followed by `<string>/Applications/CouchDBX.app/Contents/Resources/couchdbx-core</string>`.

The final result should look like the following, with your username replacing `your_username`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>Label</key>
    <string>org.apache.couchdb</string>
    <key>ProgramArguments</key>
    <array>
      <string>/Applications/CouchDBX.app/Contents/Resources/couchdbx-
core/couchdb_1.0.2/bin/couchdb</string>
    </array>
    <key>UserName</key>
    <string>your_username</string>
    <key>StandardOutPath</key>
    <string>/dev/null</string>
    <key>StandardErrorPath</key>
    <string>/dev/null</string>
    <key>RunAtLoad</key>
    <true/>
    <key>WorkingDirectory</key>
    <string>/Applications/CouchDBX.app/Contents/Resources/couchdbx-core</string>
    <key>KeepAlive</key>
    <true/>
  </dict>
</plist>
```

Once you've made changes to the `org.apache.couchdb.plist` file, shut down the CouchDB application if it is running, and copy the newly created plist file to the `/Library/LaunchDaemons/` directory.

```
shell> sudo cp ~/org.apache.couchdb.plist \
  /Library/LaunchDaemons/org.apache.couchdb.plist
```

Start up the daemon in the following way:

```
shell> sudo launchctl load /Library/LaunchDaemons/org.apache.couchdb.plist
```

You can confirm that CouchDB is running by pointing your browser at `http://127.0.0.1:5984/_utils/index.html`. You can do everything from a browser that you can do when running the CouchDB application—except shut down the server. If you want to stop the server use the command:

```
shell> sudo launchctl unload /Library/LaunchDaemons/org.apache.couchdb.plist
```

> **Note**
>
> If there are errors in your plist file, there will be no notification at the command line. If CouchDB does not start up properly you can open the Console App and check the messages. If you wish to relaunch CouchDB, you must first unload it as shown above.

Just How Compleat Have We Been?

A surprising number of topics can be covered in a minimal number of pages but lots more could be said, especially about the following topics:

- Views
- CouchDB on Android
- How CouchDB handles revisions

Those topics will be covered in "The More Compleat Guide to CouchDB" coming sometime soon.

Resources

http://wiki.apache.org/couchdb/ – the CouchDB project website

http://www.couchone.com/get – the CouchOne website

http://guide.couchdb.org/ – "CouchDB: The Definitive Guide", a free online book from O'Reilly

http://www.ibm.com/developerworks/opensource/library/os-couchdb/index.html – "Exploring CouchDB", an article at IBM developerWorks

About the Author

Peter Lavin has been published in a number of print and online magazines. He is also the author of Object Oriented PHP, published by No Starch Press and a contributor to PHP Hacks by O'Reilly Media.

Being a full-time technical writer, Peter occasionally feels the need to depart from the restrained style typical of his profession by writing articles with code snippets that use background colours other than grey.