11
Replicating your Data

With our application live for the world to see, we need to know the proper steps to scale it to meet demand.

In this chapter we're going to:

- Learn about CouchDB's powerful master-master replication
- Play with replication locally and dig into how it works
- Handle conflicts that can occur as a result of replication
- Use replication to push your local databases to Cloudant
- Play with continuous replication
- Save room in our database by compacting data
- Talk about the next steps that you can take with your application

We played with replication in Chapter 10, *Deploying your Application*, but we didn't completely look into the inner workings to see what it is and what it actually does. Let's get right into it!

What is replication?

Basic replication, as it relates to the IT world, can be defined as the process of sharing information so as to ensure consistency between redundant resources. So, if we were to define replication as it relates to CouchDB, we would have the ability to synchronize two copies of the same database using master-master replication. You may have heard of the more common master-slave database architecture that most databases use, but CouchDB uses a very powerful master-master replication model. Before we go any further, let's dive deeper into what master-slave and master-master mean, and how their architectures differ.

Master-slave replication

Master-slave is the most common database architecture when there is more than one database server. In a master-slave relationship, there is one database that, as you will have guessed, serves as the master, and the rest of the databases act as its slave. Here's what it would look like:

- An application would connect to the master and update data
- The data would then ripple through the slaves until all of the data is consistent across the servers

In the simplest implementation, this setup allows you to write to the master server, and when you are doing queries or reads, you can connect to the slave. This practice helps you reduce the burden and number of connections to any one server.

This all sounds great in theory. However, in practice, there is still a bit of a bottleneck here. If the master server goes down for whatever reason, the data will still be available via the slave, but new writes won't be possible. There are ways around this, but that's out of the scope of this discussion. I'm not arguing that this approach is bad (because it's not); I'm just defining some basic terms here.

Now that you understand the basics of master-slave replication, let's talk about master-master replication and how CouchDB uses it.

Master-master replication

Master-master replication is unique in the fact that there is no residing master database; each database server can act as the master at the same time as other servers are being treated as masters. At some point in time, all of the masters sync up to make sure that they all have correct and up-to-date data. Here's what master-master replication allows:

- If one master fails, the other database servers can operate normally and pick up the slack. When the database server is back online, it will catch up using replication.
- Masters can be located in several physical sites and can be distributed across the network.

This sounds like the perfect scenario, right? Well, it kind of is. But, there is still the possibility of running into data conflicts and all kinds of other issues. If you've ever tried master-master or multi-master replication in relational databases, then you can breathe a sigh of relief, because it's much easier in CouchDB, as it was built from the ground up with replication in mind.

Remember the revision field that we saw in every document called `_rev`? This `_rev` field keeps track of the current revision of a document. If two databases are trying to sync documents, the `_rev` field will be compared. If the `_rev` values are not identical, then the out of date one will update its revision to match by comparing revision history. You might be wondering, what happens when conflicts happen, like one document has two different pieces of data?". I'll answer that question shortly, but let's start by getting our feet wet with replication.

Playing with replication

Replication is flexible in that it can work with multiple databases on the same or different servers. This flexibility will make it easy for us to test and get comfortable before we do anything with our production database.

Time for action – replicating verge to another local database

One of the nice things about replication is that we can test and play with it anywhere. So, let's create another database on our local machine, and replicate the entire `verge` database there.

1. Open the command-line.
2. Let's create a new database called `verge-replica` by running the following command in Terminal, and replacing the username and password with your administrator's username and password:

   ```
   curl -X PUT username:password@localhost:5984/verge-replica
   ```

3. Terminal will respond with the following message:

   ```
   {"ok":true}
   ```

4. Now that the `verge-replica` database has been created, let's replicate all of the data from `verge` to `verge-replica` using a simple `curl` statement that talks to CouchDB's `_replicate` function:

   ```
   curl -X POST -H "Content-Type:application/json" -d '{"source":
   "verge", "target":"verge-replica"}' http://localhost:5984/_
   replicate
   ```

5. Terminal will respond with a nice long status message telling you that everything turned out okay.

```
{"session_id":"51e4f994c97874cfd70d46c9eafee865","start_
time":"Sat, 24 Dec 2011 05:56:33 GMT","end_time":"Sat, 24 Dec 2011
05:56:33 GMT","start_last_seq":0,"end_last_seq":102,"recorded_
seq":102,"missing_checked":0,"missing_found":102,"docs_
read":102,"docs_written":102,"doc_write_failures":5}
```

What just happened?

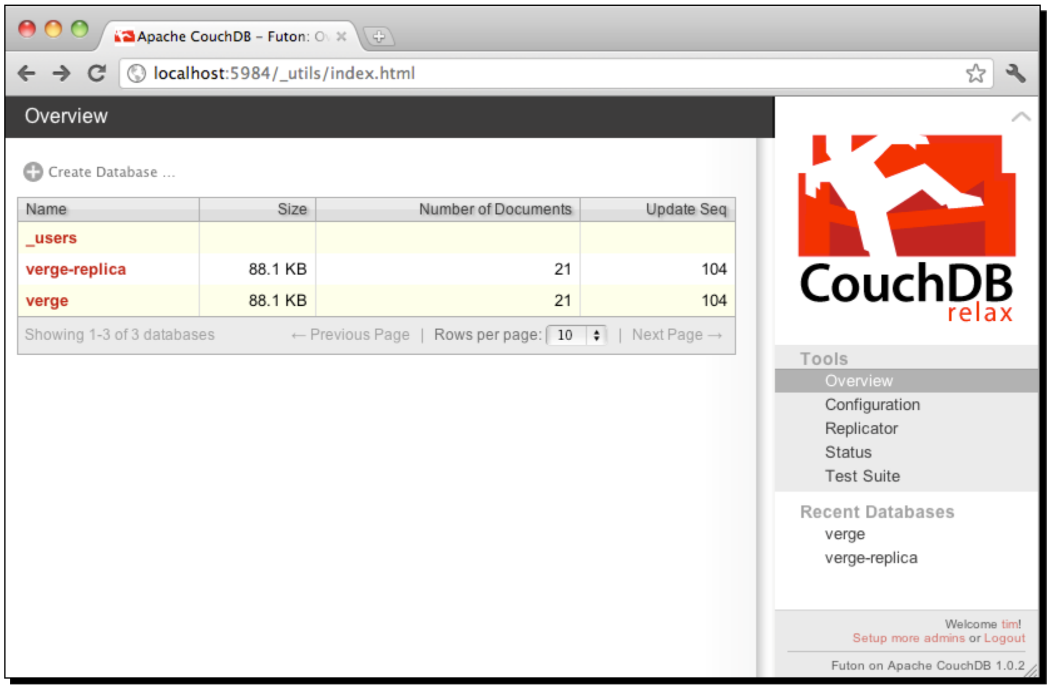
We just used Terminal to create a database called `verge-replica` to which we intend to push data from `verge`. We then used a simple (but long) `POST` `curl` statement to CouchDB's `_replicate` function. We included the source database, which is where we want the data to come from, and the target, which is the database that we want to push the changes to. After the replication has run, the results will be returned to you. Let's go through each of the keys in the response to see exactly what's happening here:

- `session_id` is the unique identifier of the replication process.
- `start_time` is the time when the replication began.
- `end_time` is the time when the replication ended. You can see that this value is very close to `start_time`, because we were running all this locally, and we didn't replicate all that many documents.
- `start_last_seq` is the time where we are starting out from. Since this is our first replicating time, its value will be zero.
- `end_last_seq` is the time where the sequence ended up at the end of replication.
- `recorded_seq` is the sequence that is recorded on the target database at the end of replication.
- `missing_checked` is the number of documents that already exist on the target, therefore they won't need to be copied. Since this is our first time replication, its value will be zero.
- `missing_found` is the number of documents that need to be replicated to the target. Again, since we've never replicated to this database, this will be the total number of documents in the source.
- `docs_read` is the number of documents read from the source database, which is the `verge` database in this instance.

- `docs_written` is the number of documents written to the target database, which is the `verge-replica` database in this instance.
- `doc_write_failures` is the number of failures that occurred. Failures can occur due to server timeouts or a `validate_doc_update` function rejecting the write of a document. If all went well, this value should be equal to zero.

It's pretty amazing to see that we just have to run one simple command, and all of the data in `verge` can be copied over to `verge-replica`. Just to make sure that `verge-replica` is intact, let's open Futon and double check the data.

1. Go to Futon by opening your browser to `http://localhost:5984/_utils`.
2. You'll notice that the **Size** and **Number of Documents** in the `verge` database is equal to the **Size** and **Number** of Documents in the `verge-replica` databases. That's enough proof for us to confidently say that all the documents replicated cleanly between `verge` and `verge-replica`. But, let's dig deeper and edit a document.

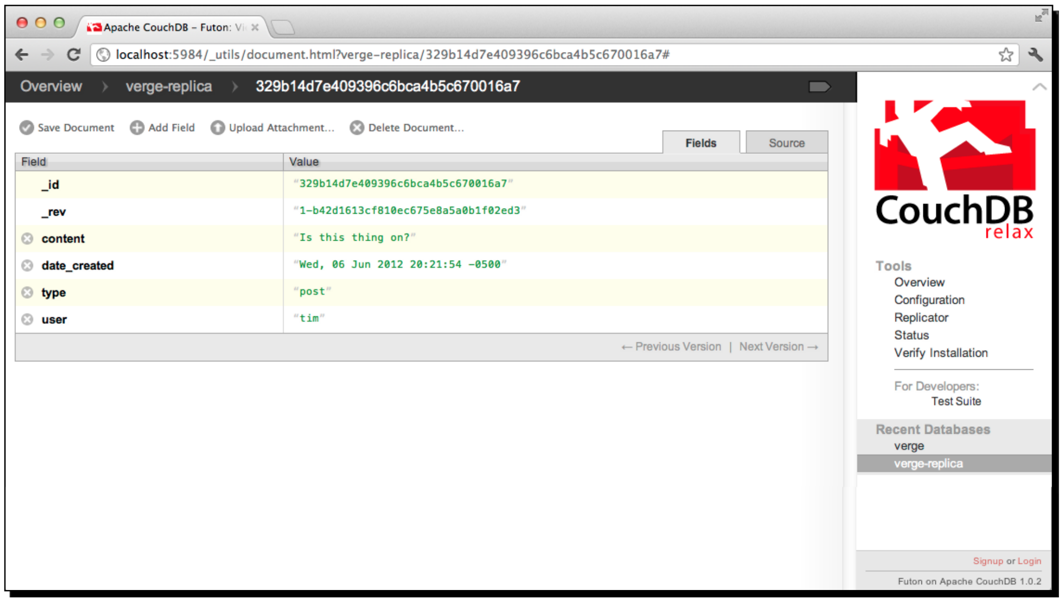


Name	Size	Number of Documents	Update Seq
_users			
verge-replica	88.1 KB	21	104
verge	88.1 KB	21	104

Showing 1-3 of 3 databases ← Previous Page | Rows per page: 10 | Next Page →

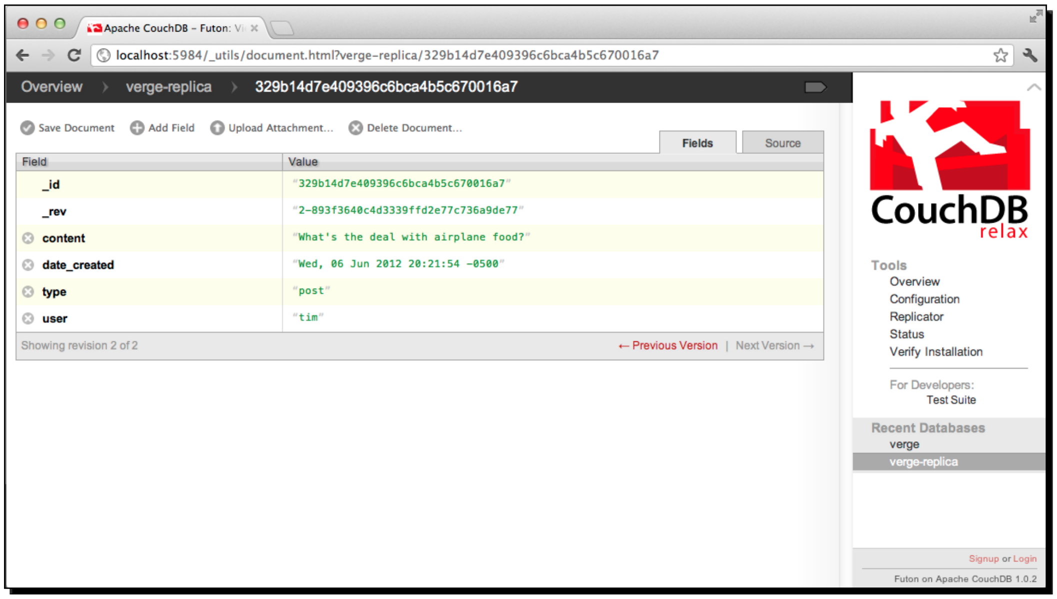
3. Click on `verge-replica` in the database list.
4. Pick any post to edit, but you'll need to remember which one it was. So, it's probably easiest to click the first one.

5. This is the post that I'm going to edit.



Field	Value
_id	"329b14d7e409396c6bca4b5c670016a7"
_rev	"1-b42d1613cf810ec675e8a5a0b1f02ed3"
content	"Is this thing on?"
date_created	"Wed, 06 Jun 2012 20:21:54 -0500"
type	"post"
user	"tim"

6. Change the content from **Is this thing on?** to **What's the deal with airplane food?**.
7. Click on **Save Document**.



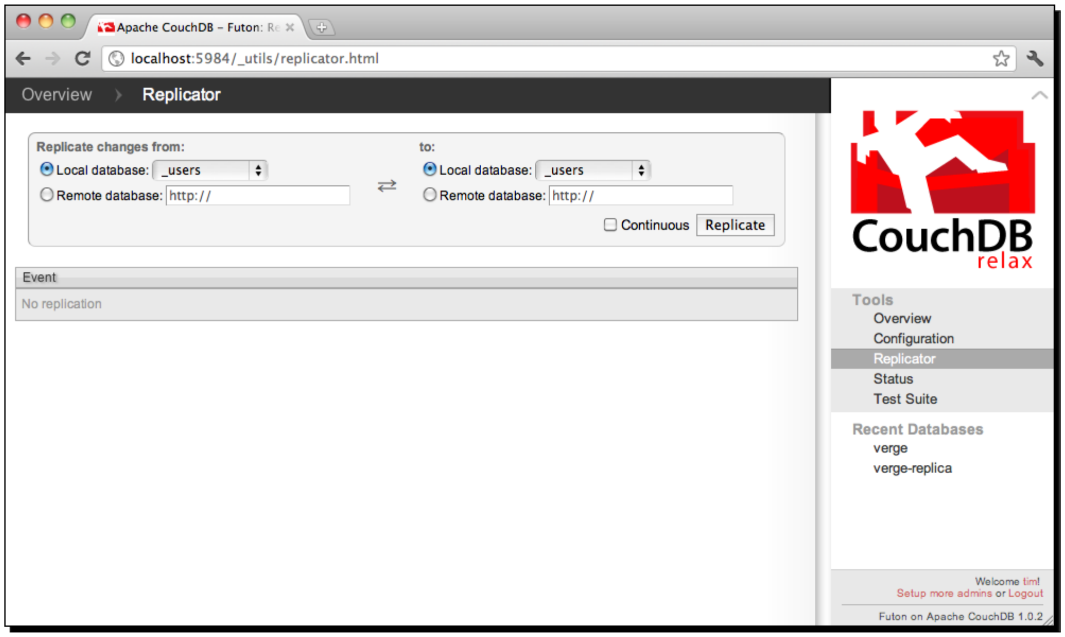
Field	Value
_id	"329b14d7e409396c6bca4b5c670016a7"
_rev	"2-893f3640c4d3339ffd2e77c736a9de77"
content	"What's the deal with airplane food?"
date_created	"Wed, 06 Jun 2012 20:21:54 -0500"
type	"post"
user	"tim"

Let's see replication in action by replicating the updated data from `verge-replica` back to `verge`.

Time for action – using Futon to replicate again

Typing the replication statement through the command-line is definitely the most universal way to communicate with CouchDB, but it can become pretty irritating if you miss quotation marks or forget the tricky syntax. Luckily, Futon has our back with a nice and simple **Replicator** interface. Let's replicate our data again. But, this time, let's switch things up and replicate our newly updated data from `verge-replica` to `verge`, and use Futon to speed things up along the way.

1. Go to Futon by opening your browser to `http://localhost:5984/_utils`.
2. If you're not already logged in, make sure to click on **Login** at the bottom of the page and enter your administrator's credentials.
3. Click on **Replicator** in the side bar. You will be presented with a form that will allow you to tell CouchDB which database you would like to replicate from (the source database), and the database you would like to replicate the changes to (the target database).



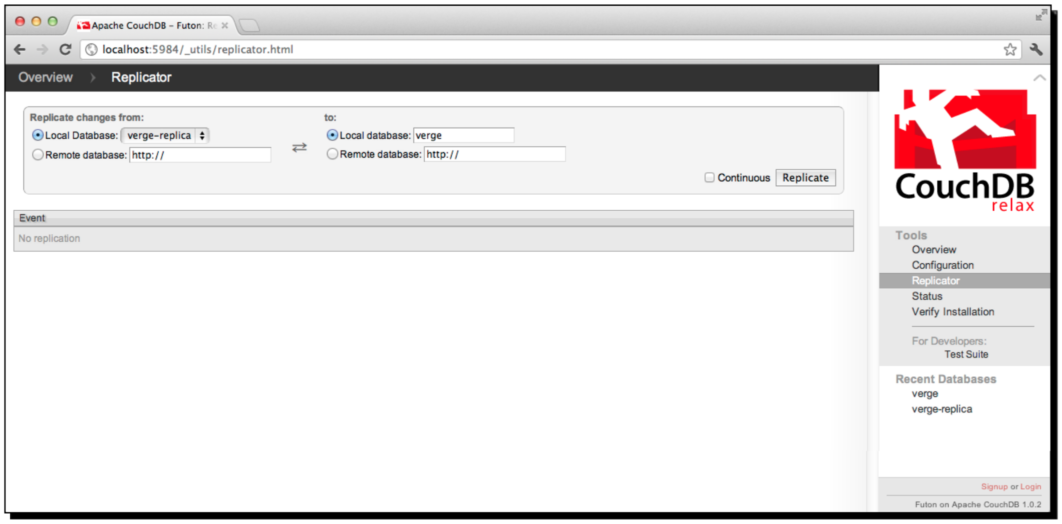
Replicate changes from:
(● Local database: _users) Remote database: http://

to:
(● Local database: _users) Remote database: http://

☐ Continuous Replicate

Event
No replication

4. Remembering that replication can go both ways, lets replicate our data from `verge-replica`, by putting it in the from the database back to `verge` by putting it into the to database.



Replicate changes from:
(●) Local Database: verge-replica
( ) Remote database: http://

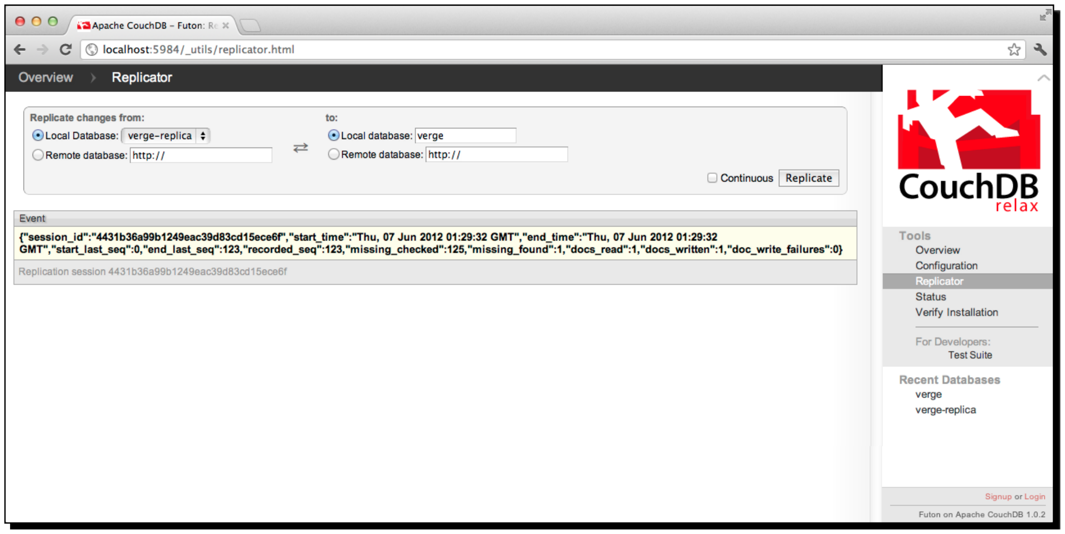
to:
(●) Local database: verge
( ) Remote database: http://

☐ Continuous [Replicate]

Event
No replication

5. When you're ready, click on `Replicate`, and CouchDB will replicate our `verge-replica` database back to the `verge` database.

6. You'll see a replication result that is pretty similar to what you saw from the command-line earlier.



Replicate changes from:
(●) Local Database: verge-replica
( ) Remote database: http://

to:
(●) Local database: verge
( ) Remote database: http://

☐ Continuous [Replicate]

Event

```
{"session_id":"4431b36a99b1249eac39d83cd15ece6f","start_time":"Thu, 07 Jun 2012 01:29:32 GMT","end_time":"Thu, 07 Jun 2012 01:29:32
GMT","start_last_seq":0,"end_last_seq":123,"recorded_seq":123,"missing_checked":125,"missing_found":1,"docs_read":1,"docs_written":1,"doc_write_failures":0}
Replication session 4431b36a99b1249eac39d83cd15ece6f
```

What just happened?

We just replicated back and forth between two databases. This is known as **bi-lateral replication**. Bi-lateral replication means that we can replicate back and forth between databases without having to name a database as the master; we simply want to maintain the data across all of the servers.

To look a bit deeper into what happened, look closely at the fields in the replication results in the previous image. Do you notice anything different? CouchDB determined that there was a document missing in `verge` that was present in `verge-replica`, and outputted the field `missing_found` with a value of `1`. Likewise, a new document was created in `verge` with the data from `verge-replica`; it saved successfully. So, it outputted the field `docs_written` with a value of `1`. The writing of the new document was successful without any problems, so `doc_write_failures` is still set to `0`.

As your application and database grow, you may have a variety of database servers. You can ensure that with bi-lateral replication, you can write to a server and the data will eventually be consistent across all of the other servers.

Managing conflicts

The concept of bi-lateral replication brings up an important issue: what happens when two documents are changed on two separate database servers? We try to replicate these databases against each other. This occurrence is called a **conflict**.

Conflicts, while annoying, are a part of most distributed systems. For instance, in this book, we've been using Git to manage our source control. We are the only developer accessing the source code. But, if you had a whole team of people working on this project, then you'd run into a case where two developers will eventually change the same line of code. In this example, the developer who is trying to merge the database against the other database will have to say "mine is the correct one", or "yours is the correct one", or "pieces of yours are right and pieces of mine or right".

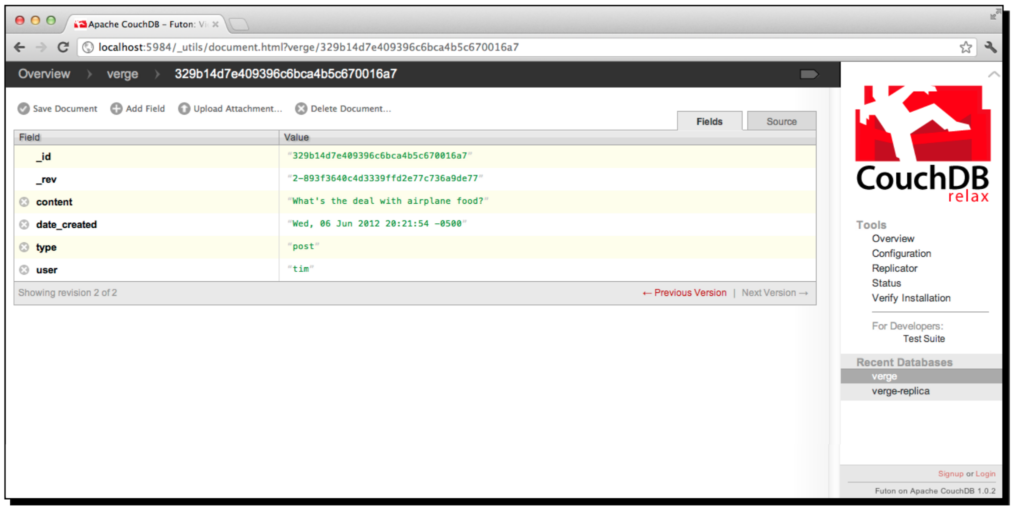
This same exact exchange and workflow is something that we'll experience inside of CouchDB.

Let's go through it step-by-step in our real database to show you how you might resolve these conflicts.

Time for action – creating and fixing a conflict

Creating a conflict is pretty easy for you to simulate between two databases. So let's do that now.

1. Let's go into the `verge` database first, and open the document that we've been playing with.



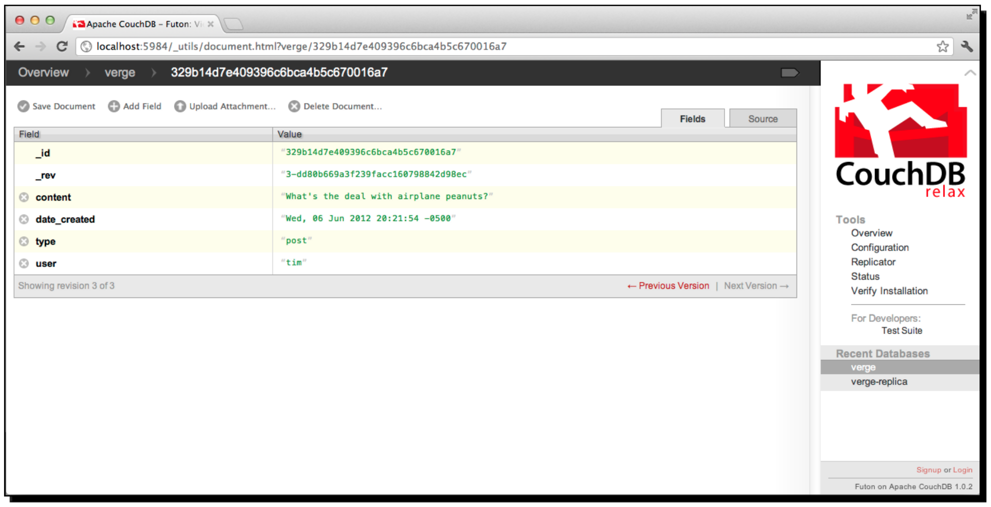
localhost:5984/_utils/document.html?verge/329b14d7e409396c6bca4b5c670016a7

Overview › verge › 329b14d7e409396c6bca4b5c670016a7

Field	Value
_id	"329b14d7e409396c6bca4b5c670016a7"
_rev	"2-893f3640c4d3339ffd2e77c736a9de77"
content	"What's the deal with airplane food?"
date_created	"Wed, 06 Jun 2012 20:21:54 -0500"
type	"post"
user	"tim"

Showing revision 2 of 2

2. The data in this document is equal on both the database servers. So, let's change the value of the content field to something different; I made mine `What's the deal with airplane peanuts?`. Click on **Save Document** when you're done.



localhost:5984/_utils/document.html?verge/329b14d7e409396c6bca4b5c670016a7

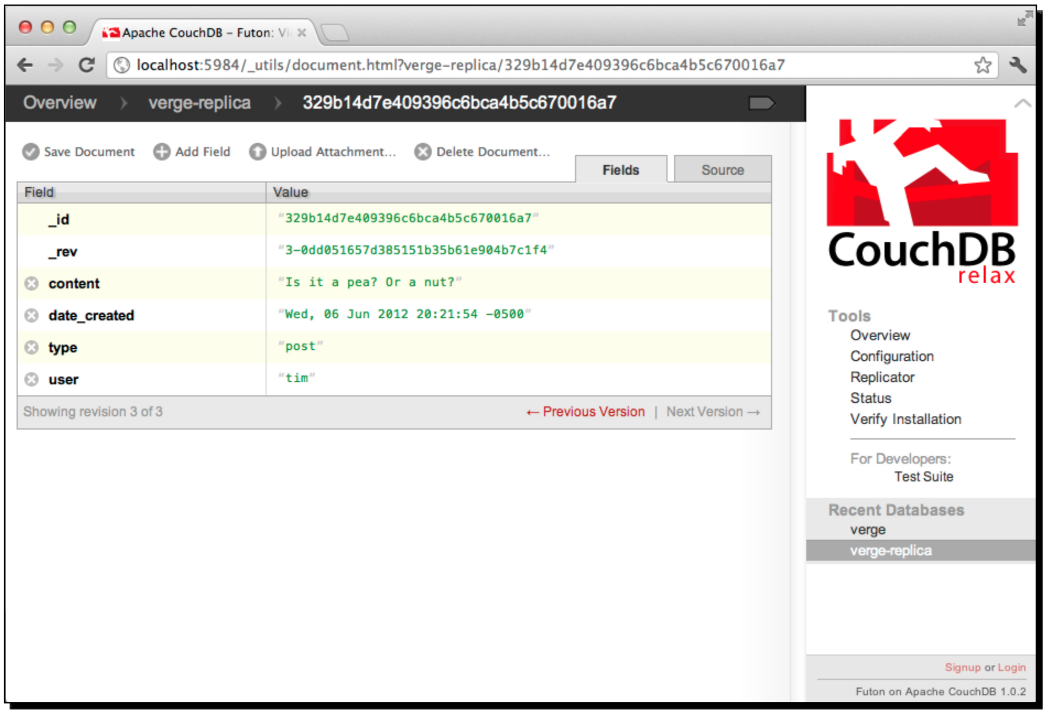
Overview › verge › 329b14d7e409396c6bca4b5c670016a7

Field	Value
_id	"329b14d7e409396c6bca4b5c670016a7"
_rev	"3-dd80b669a3f239facc160798842d98ec"
content	"What's the deal with airplane peanuts?"
date_created	"Wed, 06 Jun 2012 20:21:54 -0500"
type	"post"
user	"tim"

Showing revision 3 of 3

3. Notice that the `_rev` field has been updated.

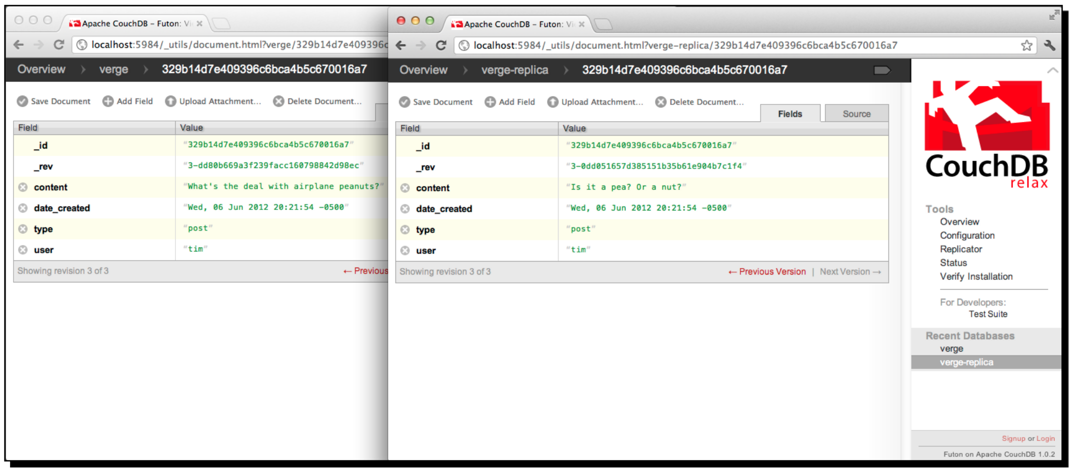
4. Now, let's go to the `verge-replica` database and find the same document.



Field	Value
_id	"329b14d7e409396c6bca4b5c670016a7"
_rev	"3-0dd051657d385151b35b61e904b7c1f4"
content	"Is it a pea? Or a nut?"
date_created	"Wed, 06 Jun 2012 20:21:54 -0500"
type	"post"
user	"tim"

5. Let's change the value of the content field to something different to what we entered on `verge`. I made mine `Is it a pea? Or a nut?`. Click on **Save Document** when you're done.

6. What we have now is one document with conflicting data. You'll notice that the `_rev` fields do not match, which is what the replicator will pick up on in just a moment.



localhost:5984/_utils/document.html?verge/329b14d7e409396c

Overview › verge › 329b14d7e409396c6bca4b5c670016a7

Field	Value
_id	"329b14d7e409396c6bca4b5c670016a7"
_rev	"3-dd80b669a3f239facc160798842d98ec"
content	"What's the deal with airplane peanuts?"
date_created	"Wed, 06 Jun 2012 20:21:54 -0500"
type	"post"
user	"tim"

Showing revision 3 of 3

localhost:5984/_utils/document.html?verge-replica/329b14d7e409396c6bca4b5c670016a7

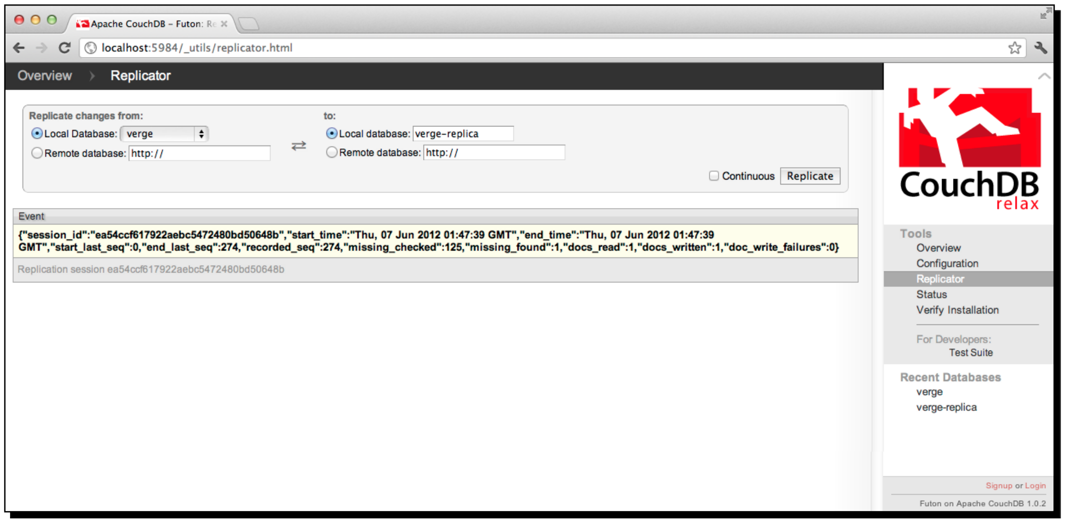
Overview › verge-replica › 329b14d7e409396c6bca4b5c670016a7

Field	Value
_id	"329b14d7e409396c6bca4b5c670016a7"
_rev	"3-0dd051657d385151b35b61e904b7c1f4"
content	"Is it a pea? Or a nut?"
date_created	"Wed, 06 Jun 2012 20:21:54 -0500"
type	"post"
user	"tim"

Showing revision 3 of 3

7. All right, let's cause some chaos. Click on **Replicator** in the side bar.

8. In the **Replicator** screen, put `verge` in the from database and `verge-replica` in the to database. Cross your fingers and click on **Replicate**.



localhost:5984/_utils/replicator.html

Overview › Replicator

Replicate changes from:
[✓ Local Database:] verge
Remote database: http://

to:
[✓ Local database:] verge-replica
Remote database: http://

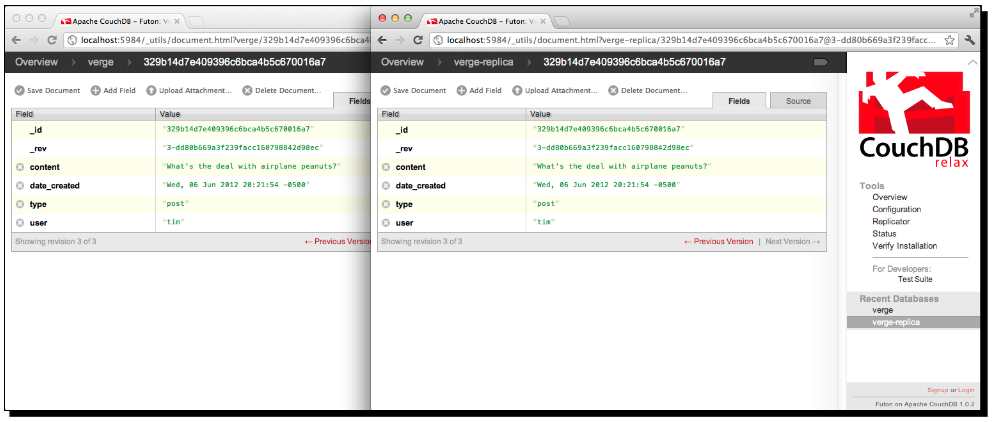
Continuous

Event

```
{"session_id":"ea54ccf617922aebc5472480bd50648b","start_time":"Thu, 07 Jun 2012 01:47:39 GMT","end_time":"Thu, 07 Jun 2012 01:47:39
GMT","start_last_seq":0,"end_last_seq":274,"recorded_seq":274,"missing_checked":125,"missing_found":1,"docs_read":1,"docs_written":1,"doc_write_failures":0}
```

Replication session ea54ccf617922aebc5472480bd50648b

9. It looks like everything went off without a hitch again. But we know better than that—we know that a conflict occurred. But, if we were to look at both of our documents, you'll see that their values are the same.



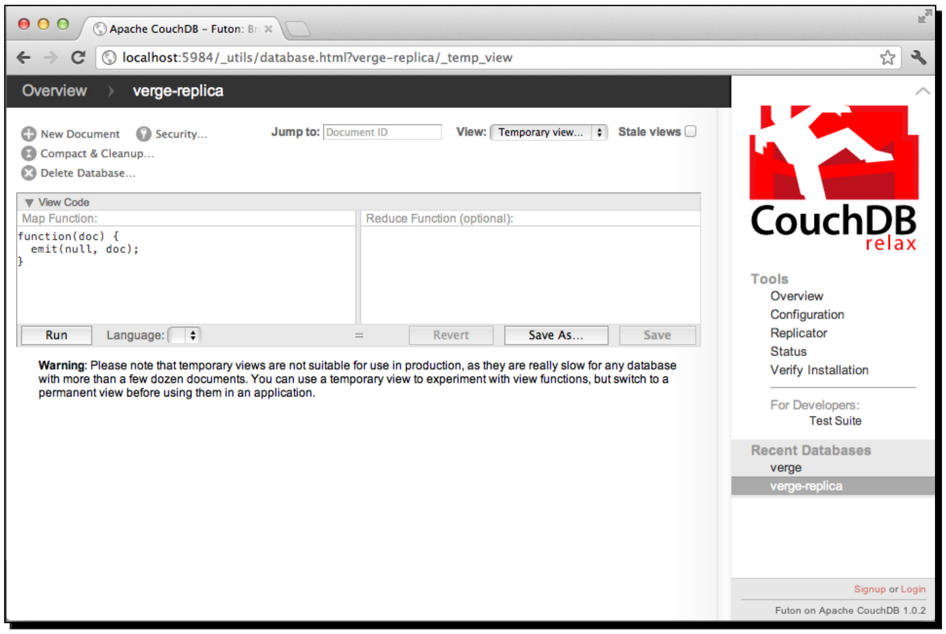
Field	Value
_id	"329b14d7e409396c6bca4b5c670016a7"
_rev	"3-dd80b669a3f239facc160798842d98ec"
content	"What's the deal with airplane peanuts?"
date_created	"Wed, 06 Jun 2012 20:21:54 -0500"
type	"post"
user	"tim"

Showing revision 3 of 3

localhost:5984/_utils/document.html?verge/329b14d7e409396c6bca4
localhost:5984/_utils/document.html?verge-replica/329b14d7e409396c6bca4b5c670016a7@3-dd80b669a3f239facc...
Overview › verge › 329b14d7e409396c6bca4b5c670016a7
Overview › verge-replica › 329b14d7e409396c6bca4b5c670016a7
Save Document
Add Field
Upload Attachment...
Delete Document...
Fields
Source
← Previous Version | Next Version →
CouchDB relax
Tools
Overview
Configuration
Replicator
Status
Verify Installation
For Developers:
Test Suite
Recent Databases
verge
verge-replica
Signup or Login
Futon on Apache CouchDB 1.0.2

10. CouchDB assumed that the data in the source database was the "actual" data to save, and it overwrote the target database. In order to see what actually happened, we need to create a temporary view to list out all of the documents that have some type of conflict.

11. Go back to the database overview, click on the view drop-down, and select the **Temporary view...** option.



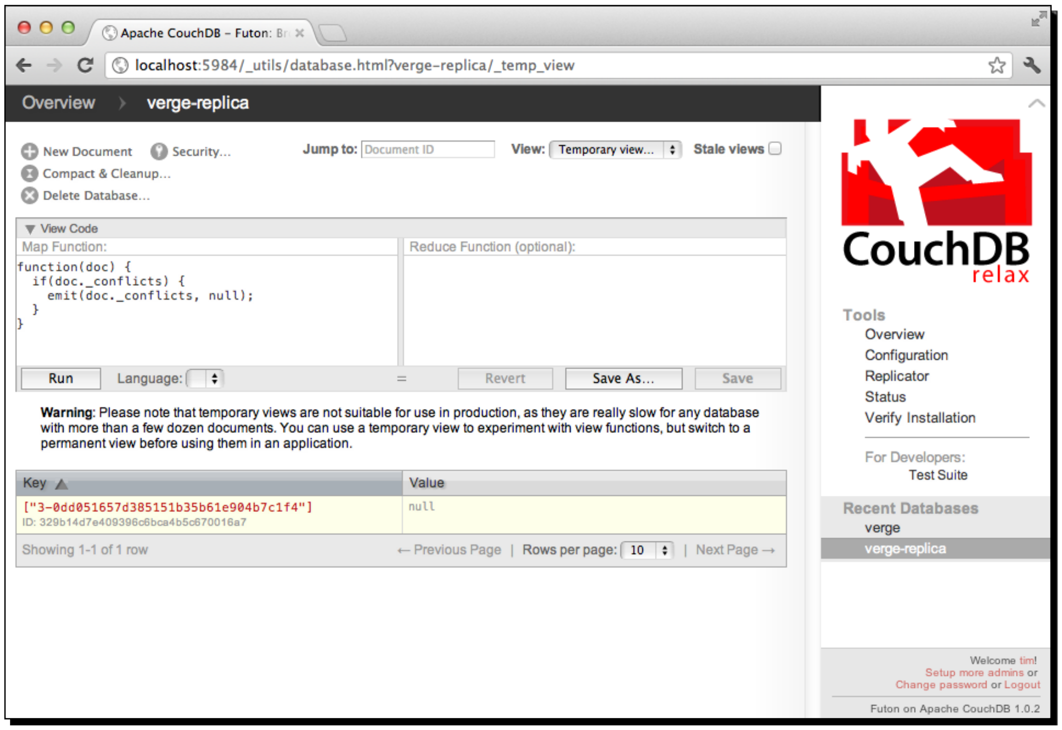
```
function(doc) {
  emit(null, doc);
}
```

Warning: Please note that temporary views are not suitable for use in production, as they are really slow for any database with more than a few dozen documents. You can use a temporary view to experiment with view functions, but switch to a permanent view before using them in an application.

12. Type the following code into the Map function text area, and click on **Run**.

```
function(doc) {
  if(doc._conflicts) {
    emit(doc._conflicts, null);
  }
}
```

13. You'll see that the ID of the document that we've been playing with is listed.



```
function(doc) {
  if(doc._conflicts) {
    emit(doc._conflicts, null);
  }
}
```

Warning: Please note that temporary views are not suitable for use in production, as they are really slow for any database with more than a few dozen documents. You can use a temporary view to experiment with view functions, but switch to a permanent view before using them in an application.

Key	Value
["3-0dd051657d385151b35b61e904b7c1f4"] ID: 329b14d7e409396c6bca4b5c670016a7	null

Showing 1-1 of 1 row ← Previous Page | Rows per page: 10 | Next Page →

What just happened?

We just simulated the occurrence of a conflict between two databases in CouchDB. Essentially, we changed the content of a document on two different databases and replicated them together. The source database document essentially overwrote the value of the target database's document content during replication.

We could just consider this fine, but what if the conflict deleted the data that was mission-critical for you?

In order to solve this problem, we created a temporary view to show us a list of all of the documents that were conflicting. From there, if you decided that you wanted to build a conflict system, then you could turn this into an actual view, and poll it with a background process to look for conflicts. You could then develop business rules around it and fix the conflicts programmatically, or alert one of your technical staff to fix it manually. To fix a conflict, you need to delete the old revisions of the document, and update the document with the value that you actually want to use.

This is an issue that you may encounter, but if you do, please check out the Github repository that I'll tell you about at the end of this chapter. I'll be sure to add a working example of this for you to play with.

Continuous replication

Doing one-off replication between two databases is great, but chances are that for most data, you'll need to do the replication on a continual basis. We'll call this **continuous replication**. Luckily, for us to do continuous replication, we just need to pass an additional parameter to the replicate function. I'll not only show you how to perform continuous replication through Futon, but I'll also give you the command-line statement at the end, if you want to try that as well.

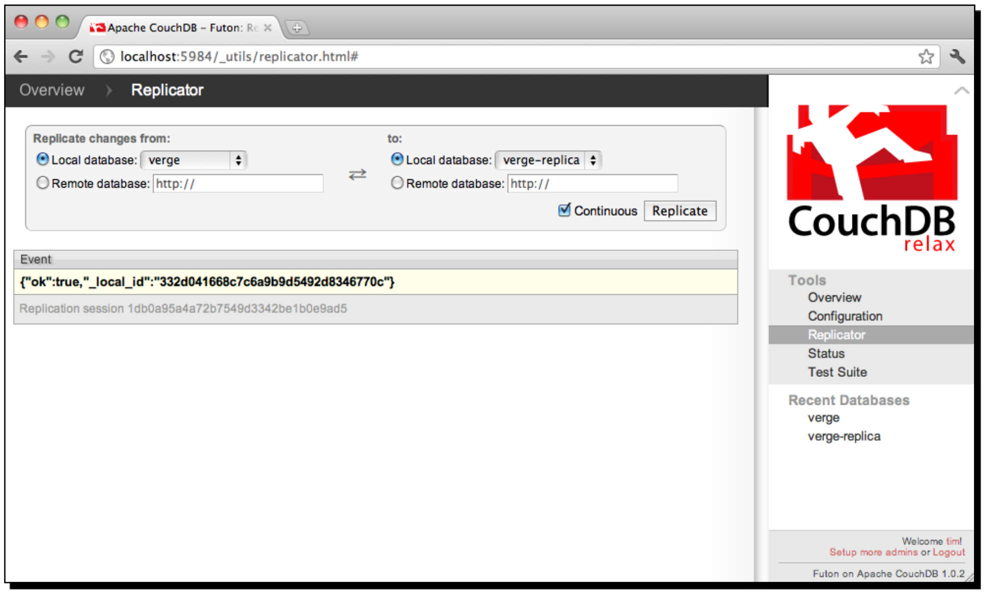
Time for action – setting up continuous replication using Futon

Let's set up continuous replication from our `verge` database to `verge-replica` using Futon. I think you'll enjoy that as all we really need to do is click on a checkbox, and CouchDB will take it from there!

1. Open up Futon and click on **Replicator**, or navigate directly to `http://localhost:5984/_utils/replicator.html`.
2. Select `verge` as the from (source) database and select `verge-replica` as the to (target) database
3. The only thing that is different is that we'll want to click the **Continuous** checkbox.

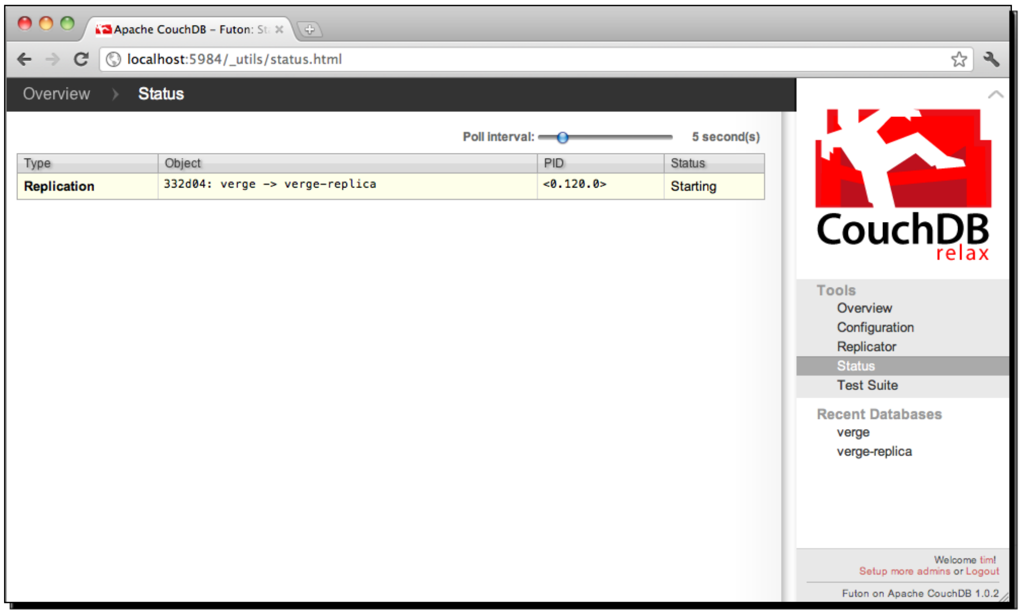
4. Click on **Replicate**.

Futon will show you a different kind of response than you've seen before, but it says that everything went okay.



Event
{"ok":true,"_local_id":"332d041668c7c6a9b9d5492d8346770c"}
Replication session 1db0a95a4a72b7549d3342be1b0e9ad5

5. Let's check to make sure the replication has started by clicking on **Status** in the sidebar.



Type	Object	PID	Status
Replication	332d04: verge -> verge-replica	<0.120.0>	Starting

What just happened?

We just used the familiar Futon console that we've used before to perform one-off replication, but this time we clicked on the **Continuous** checkbox. By checking this checkbox, we are telling CouchDB that we want it to continually poll for changes from the source database and the target database. When any changes are found, we want to replicate those documents from the source database to the target database. This replication runs as a separate process, so we can check our **Status** panel to see the **PID** (process ID) and the **Status** of the replication running in the background.

Since we might not always have access to Futon to perform this action, we can also just add `"continuous":"true"` to the same curl statement we used before in Terminal as follows:

```
curl -X POST http://localhost:5984/_replicate -d '{"source":"db",
"target":"db-replica", "continuous":true}' -H "Content-Type: application/
json"
```

It's worth noting that, in this example, we are only replicating from one database to another. But, if any changes occur in the `verge-replica` database, then we aren't currently replicating them back to `verge`. If we really want to have bi-directional continuous replication, we totally could. Going even further, if this cluster grew, we could replicate to multiple servers just as easily. Designing a solid and scalable replication system is a bit out of the realm of this book. But, with some research and experimenting, it shouldn't be too hard to come up with a solid solution.

Now that we have continuous replication set up, let's go through and test to make sure that the content is being pushed to the `verge-replica` database when we create new content, since our application is solely accessing the `verge` database. We could do this in a number of ways, but the easiest way is to actually create a new post through our application that is connected to the `verge` database.

1. Open your browser to `http://localhost/verge/`, and log in as one of your users.
2. Click on **My Profile** and create a new post.
3. Now, let's check to see if our changes have been replicated from `verge` to `verge-replica`. Go to `http://localhost:5984/_utils`.

The easiest way to check that our data has replicated is to simply look at the number of documents in each database. If they are the same, then they are synched up.

You could dig deeper and find the newly created document in both databases, but it's overkill for us to do all of that. We can trust that CouchDB will do its job in replicating the data. Now that you've learned the ropes, let's make sure that we remember how to replicate the data from our local `verge` database to our `production` database on Cloudant.

Replicating the local data to production

This section has a bit of overlap with Chapter 10, but it's important for you to understand, So I'll go over it quickly again. Let's replicate both the `_users` and `verge` databases, so that all of our local data is available on the production server. Believe it or not, replicating data to another server is no different than what we have been toying around with locally.

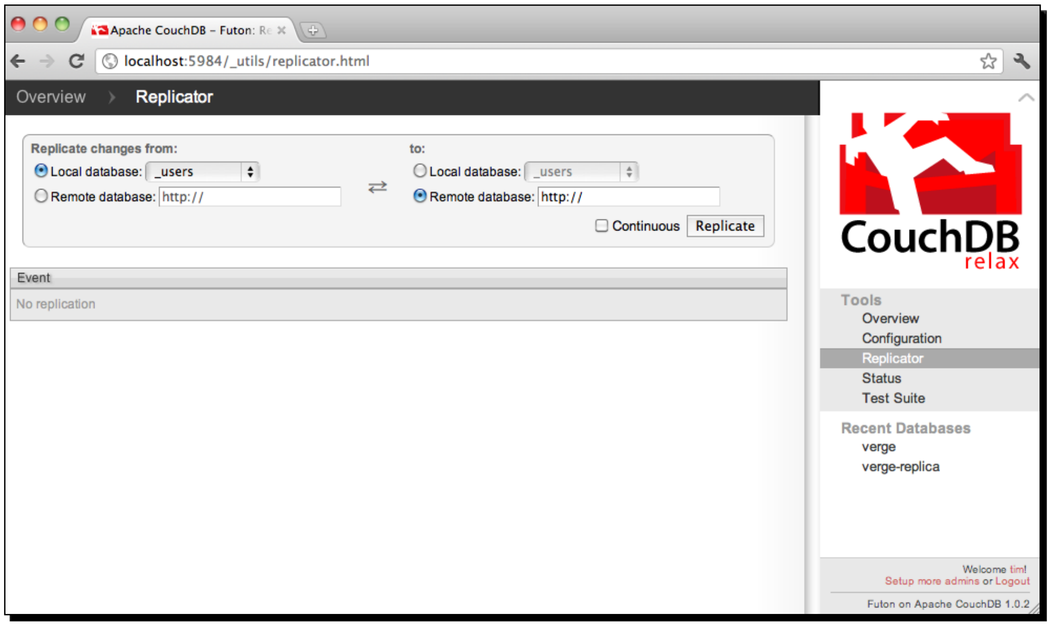
Just so that we get some variety, let's replicate the `_users` database through Futon, and the `verge` database through Terminal.

Once we've done this, our application will be operational with all of the testing data that you've created locally. You don't have to worry if your application has been live for a few minutes or even a few days; the best part of replication is that if someone is already using your application, then all of their data will remain intact, and we'll just be adding our local data.

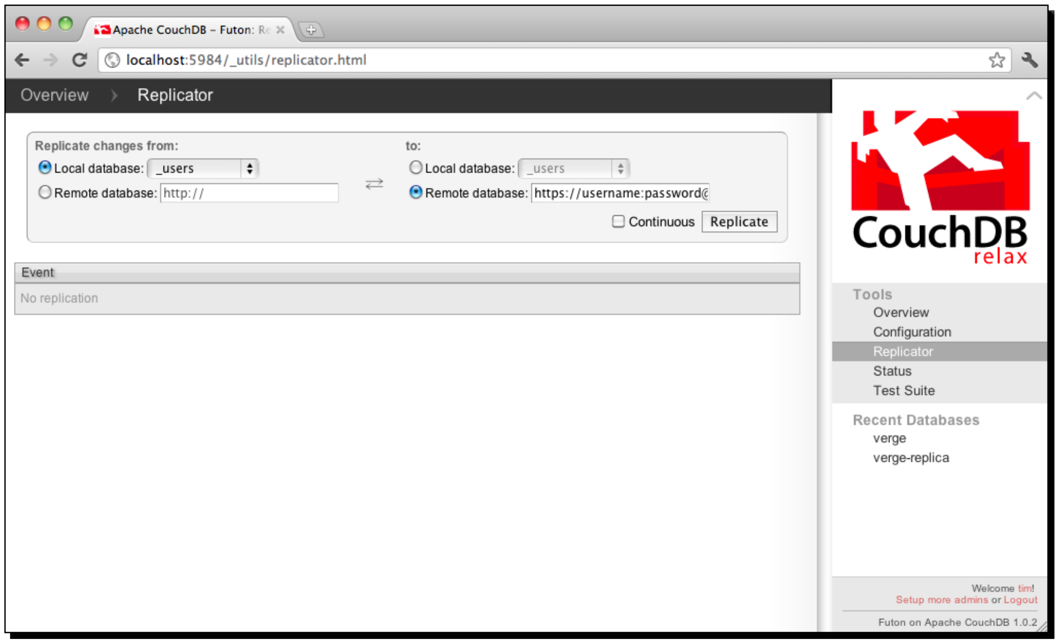
Time for action – replicating our local _users database to Cloudant

Let's replicate our local `_users` database to the `_users` database we've created on Cloudant.

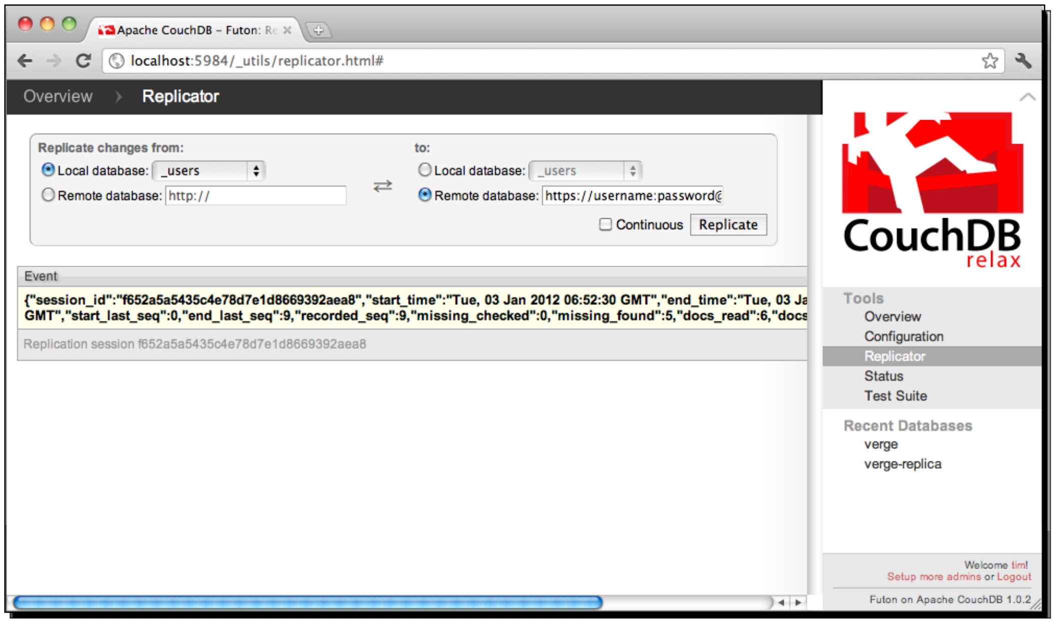
1. Open Futon in the browser and click on **Replicator**, or you can navigate directly to `http://localhost:5984/_utils/replicator.html`.
2. Make sure that you are signed in as the `administrator`; if you are not, click on **Login** and sign in as an **administrator**.
3. Select the `_users` database in the **Replicate changes from** dropdown.
4. Click on the **Remote database** radio button in the **to** section.



4. In the **Remote database** text field, enter the URL of the database at Cloudant along with the credentials. The format of the URL will look similar to `https://username:password@username.cloudant.com/_users`.



5. Click on **Replicate**, and CouchDB will push your local database to Cloudant. You'll see the familiar results from Futon.



Replicate changes from:
(●) Local database: _users
( ) Remote database: http://

to:
( ) Local database: _users
(●) Remote database: https://username:password@

☐ Continuous

Event

```
{"session_id":"f652a5a5435c4e78d7e1d8669392aea8","start_time":"Tue, 03 Jan 2012 06:52:30 GMT","end_time":"Tue, 03 Ja
GMT","start_last_seq":0,"end_last_seq":9,"recorded_seq":9,"missing_checked":0,"missing_found":5,"docs_read":6,"docs
```

Replication session f652a5a5435c4e78d7e1d8669392aea8

What just happened?

We just used Futon to replicate our local `_users` database to our `_users` production database hosted with Cloudant. The process was exactly the same as we've done before, however, we used **Remote Database** in the `to` section, and used the URL of the database along with our credentials. When the replication was complete, we received a familiar report saying that everything went okay. It looks like everything went off well without a hitch. Let's move on to replicating our `verge` database as well.

It's worth mentioning that if you attempted to replicate the `users` database from the command-line, then you would have to include the username and password in your call. This is because we have the `users` database completely locked to anonymous users. The function would look something similar to the following:

```
curl -X POST http://user:password@localhost:5984/_replicate -d
'{"source":"_users", "target":"https://timjuravich:wookie@timjuravich.
cloudant.com/_users"}' -H "Content-Type: application/json"
```

Have a go hero – replicating the local verge database to Cloudant

As we've seen before, it's easy to replicate databases from the command-line. Do you think you can figure out the command to replicate your local `verge` database to the `verge` database on Cloudant? It's almost impossible to mess anything up at this stage in the game, so don't be scared to try a few things if you don't get it the first time.

Give it a shot. When you're done, flip to the next page, and we'll go over the command that I used.

How did everything go? Hopefully, you got it without too much effort. If you couldn't get it to work, here's an example of a command that you could have run:

```
curl -X POST localhost:5984/_replicate -d '{"source":"verge",
"target":"https://timjuravich:wookie@timjuravich.cloudant.com/verge"}' -H
"Content-Type: application/json"
```

In this example, we're using our local CouchDB instance to replicate our local `verge` database to the target Cloudant `verge` database. For the local database, we can simply put the name as `verge`, but for the target database, we had to pass the full database location.

With all of our data live and on our production server, you can log in as any of the users that you created locally, and see all of our content is live and ready for the world to see. But, it's not quite the end of our journey yet; as our application grows we will need to know about some of the maintenance that we can perform on CouchDB, and how we might easily scale using PHPFog and Cloudant.

Compacting databases

As we've seen a few times in this book, CouchDB's revision system is a powerful thing. But, with CouchDB keeping track of each revision on a document, there will be a lot of data overhead that you might no longer need. With that in mind, CouchDB has a nice and simple compaction function that allows you to easily compact a database.

Compaction compresses your database by removing the document revision history, which is created during updates. Deleted documents and old revisions of documents will also be removed from memory.

Let's compact our local database, so that we're familiar with how to do this in the real world if we ever need to.

Time for action – compacting our local verge database

Compaction is a manually triggered process, which, as you may have guessed, can be triggered through CouchDB's RESTful JSON API.

1. Open Terminal.
2. Trigger compaction on our local server by running the following command, and replacing the username and password with the credentials of your database `admin` account.

   ```
   curl -H "Content-Type: application/json" -X POST http://
   username:password@localhost:5984/verge/_compact
   ```

3. If all goes well, you'll be returned the simple and always satisfying response that everything went okay.

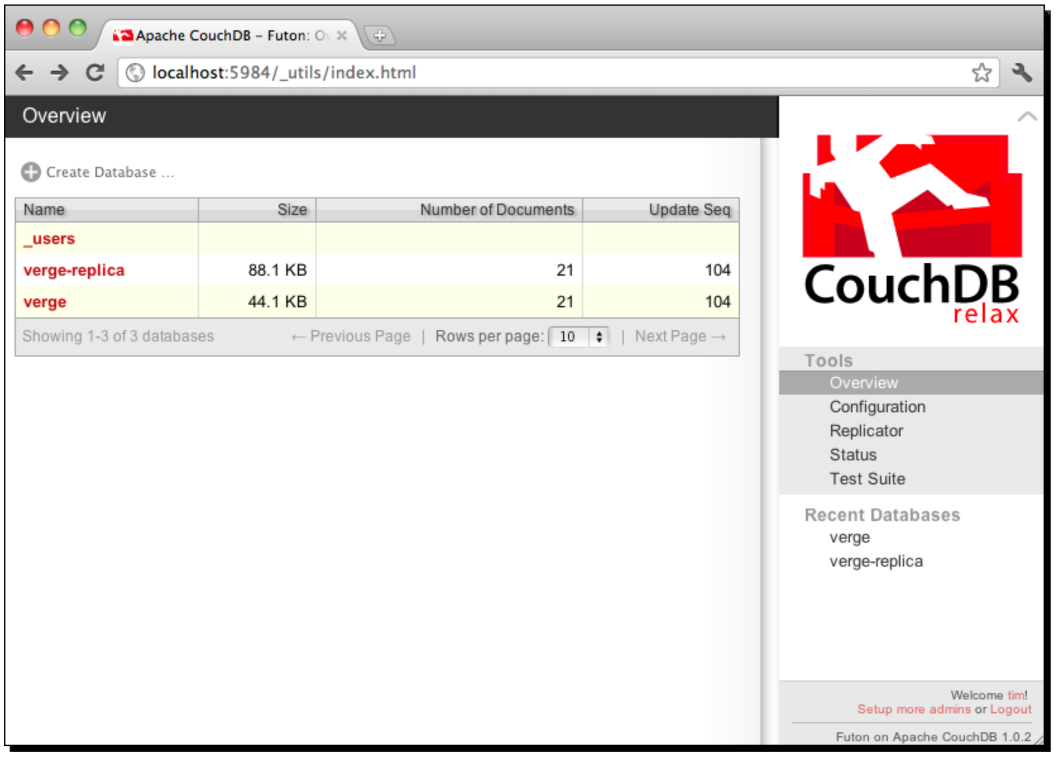
   ```
   {"ok":true}
   ```

What just happened?

We just used `curl` to trigger a `POST` call to the `verge` database `_compact` function. Once we called the `_compact` function, we were returned `{"ok":true}`, so we know that compaction completed successfully. Behind the scenes, CouchDB removed all of the revision history for each document.

Testing compaction is as easy as you might expect. We just need to look at the size of the database in Futon.

1. Open your browser and go to `http://localhost:5984/_utils`.
2. Compare `verge` against `verge-replica`, and you'll notice right away that compaction just cut the size of our database in half!



Name	Size	Number of Documents	Update Seq
_users			
verge-replica	88.1 KB	21	104
verge	44.1 KB	21	104

Showing 1-3 of 3 databases ← Previous Page | Rows per page: 10 | Next Page →

You can compact your database as many times as you like without any side effects. But, trying to run compaction more than once on the same data is a futile effort as CouchDB has already removed as much data as it could.

Compacting local databases is great, but it doesn't do much for us on our local machine, where compression is more important than on our production database. Luckily, Cloudant automatically handles compaction for us, so we don't have to worry about doing it ourselves. If you try to trigger the `_compact` function using `curl`, you'll get a friendly reminder that Cloudant has our back.

```
{"error":"forbidden","reason":"compaction is automatic on Cloudant"}
```

If you ever end up spinning up your own instance of CouchDB, then you'll want to make sure you have a good methodology for handling compaction. Compaction isn't something you want to run in a panicked situation, where you are seeing your server about to max out. This is because if you are attempting to run compaction on a database that is nearing its write capacity, then the compaction process may not complete, and worse, if the writes keep happening during replication, then you may run out of disk space. So, with that in mind, it's preferable to run compaction at off-peak times, and have it run semi-regularly to keep everything clean.

Additionally, if you were to scale out your CouchDB instances into a clustered environment, you could switch off `write` access to a node before compaction, and have it returned when replication is complete. This will keep the `write` buffer from overloading.

What's next?

You should stop reading and start developing something new! As I mentioned before, I will continue to incrementally add features like these and more to the Verge repository on GitHub here `https://github.com/timjuravich/verge`. So, make sure to watch the repository for updates and fork it if you'd like.

Again, I really appreciate the time we spent together in this book. Please feel free to reach out to me on Twitter `@timjuravich` if you have any questions.

Happy developing!

Summary

I hope you enjoyed this chapter and the great features that CouchDB continues to throw our way. Let's recap on everything we learned in this chapter:

- We talked about replication and how to use it
- We created some replication conflicts and handled them nicely
- We played with continuous replication and database compaction
- We pushed all of our data to Cloudant, so that our application had all of our data
- We finished the book and can dominate the world using PHP and CouchDB