Joshua Catoe

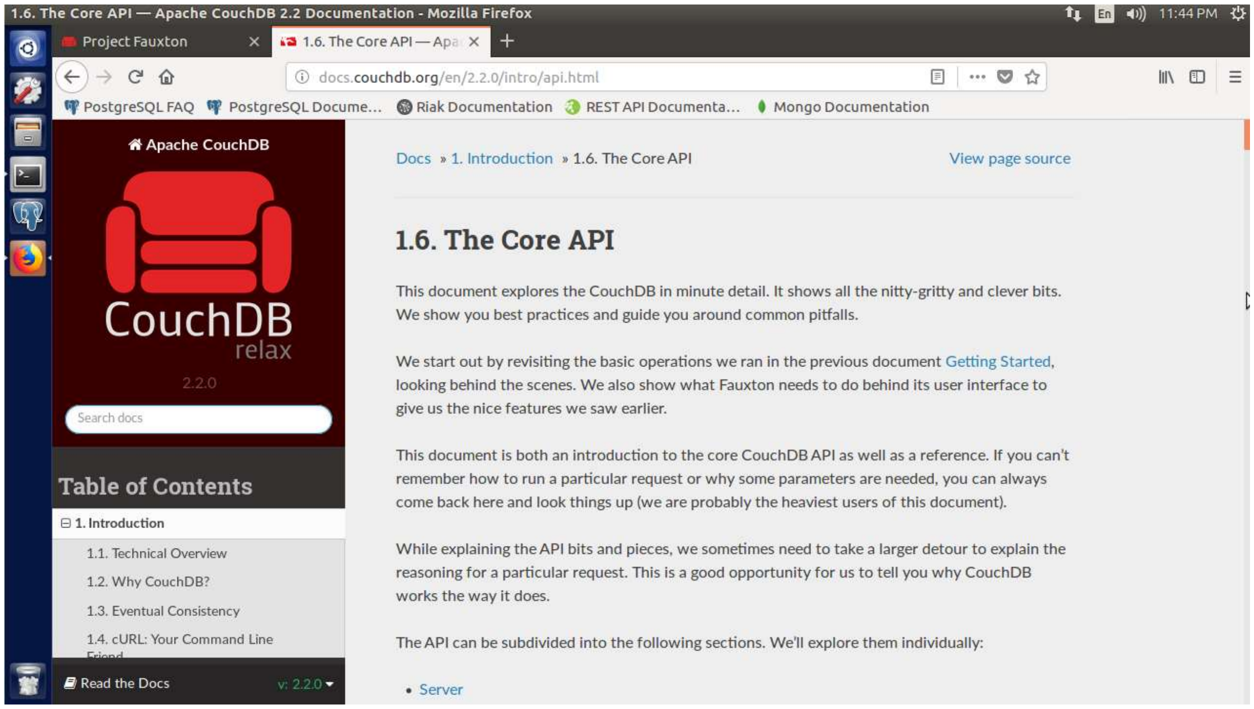
07/29/18

CouchDB

Day 1

Find:

1. Find the CouchDB HTTP Document API documentation online.



Docs » 1. Introduction » 1.6. The Core API   View page source

Table of Contents

- 1. Introduction
  - 1.1. Technical Overview
  - 1.2. Why CouchDB?
  - 1.3. Eventual Consistency
  - 1.4. cURL: Your Command Line Friend

1.6. The Core API

This document explores the CouchDB in minute detail. It shows all the nitty-gritty and clever bits. We show you best practices and guide you around common pitfalls.

We start out by revisiting the basic operations we ran in the previous document Getting Started, looking behind the scenes. We also show what Fauxton needs to do behind its user interface to give us the nice features we saw earlier.

This document is both an introduction to the core CouchDB API as well as a reference. If you can't remember how to run a particular request or why some parameters are needed, you can always come back here and look things up (we are probably the heaviest users of this document).

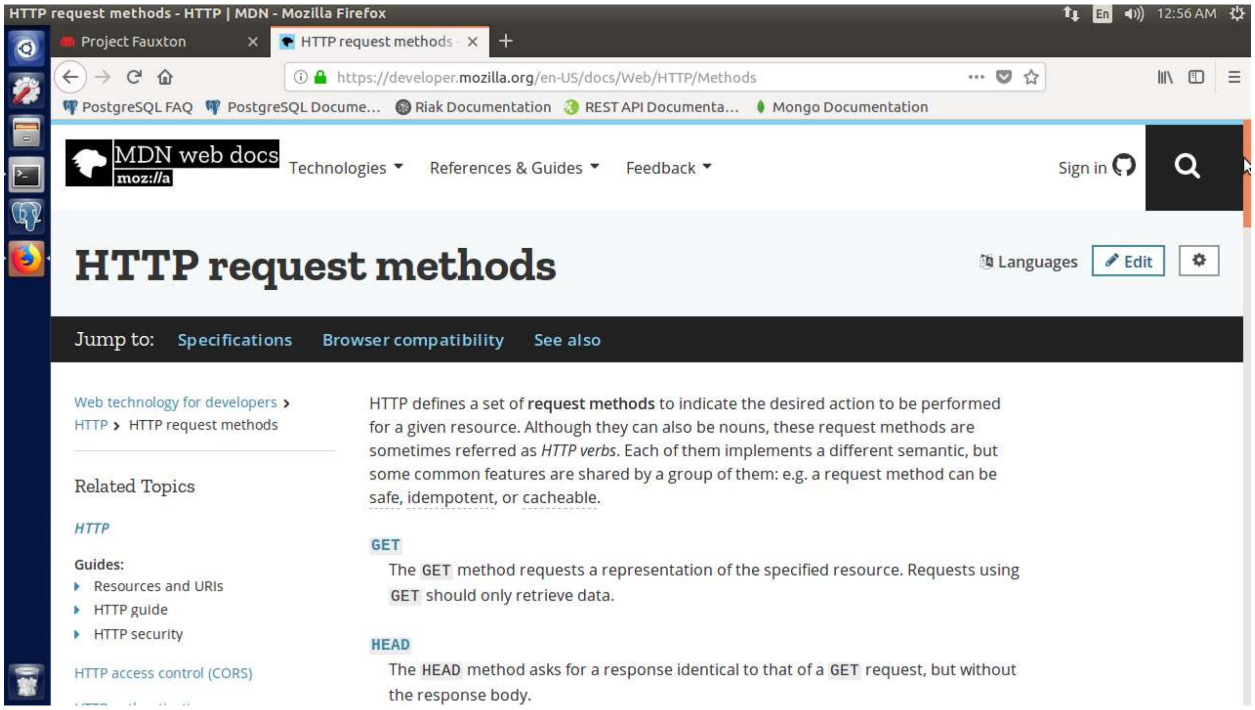
While explaining the API bits and pieces, we sometimes need to take a larger detour to explain the reasoning for a particular request. This is a good opportunity for us to tell you why CouchDB works the way it does.

The API can be subdivided into the following sections. We'll explore them individually:

- Server

The API documentation is linked to on the main docs page of the CouchDB website. Strangely, the links from a Google search page are for older versions and lead to a page with a "Permission denied" error. It looks like the only way to get to the docs for the API is through the CouchDB webpage or a direct link.

2. We’ve already used GET, POST, PUT, and DELETE. What other HTTP commands are supported?



HTTP request methods

HTTP defines a set of **request methods** to indicate the desired action to be performed for a given resource. Although they can also be nouns, these request methods are sometimes referred as *HTTP verbs*. Each of them implements a different semantic, but some common features are shared by a group of them: e.g. a request method can be safe, idempotent, or cacheable.

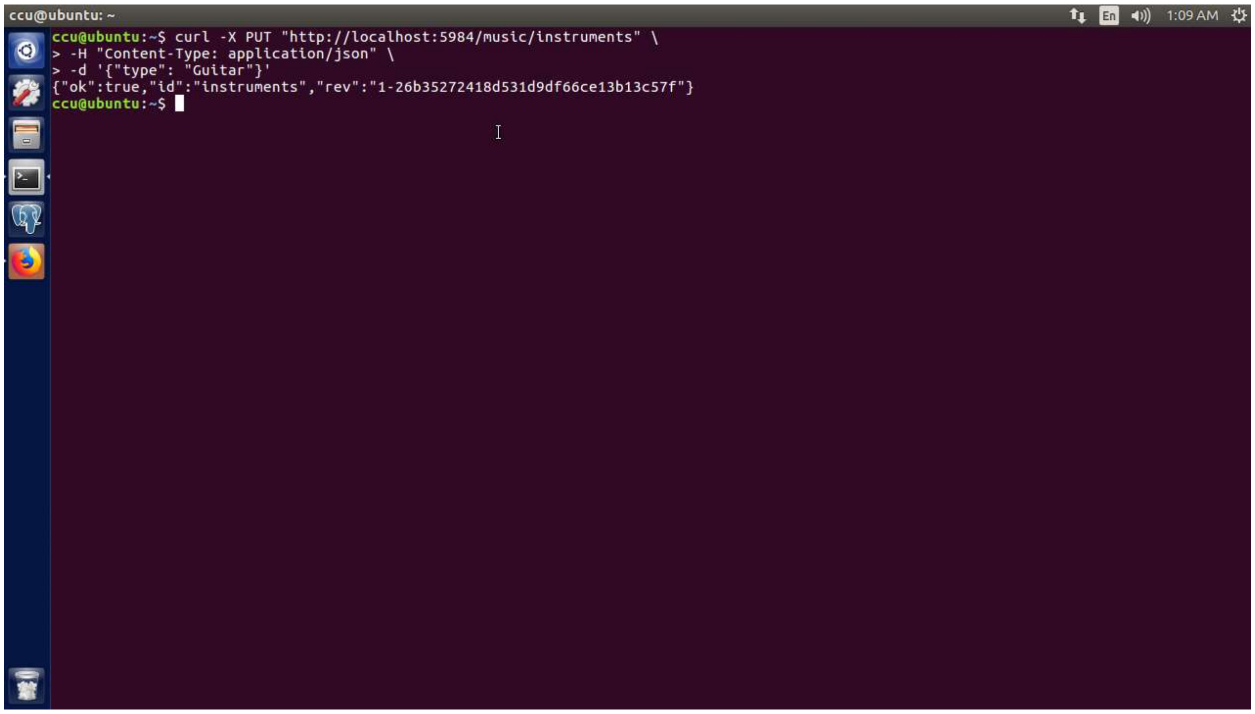
`GET`
The `GET` method requests a representation of the specified resource. Requests using `GET` should only retrieve data.

`HEAD`
The `HEAD` method asks for a response identical to that of a `GET` request, but without the response body.

I found a list of HTTP request methods from mozilla.org. Other than GET, POST, PUT, DELETE, there are HEAD, CONNECT, OPTIONS, TRACE, and PATCH.

Do:

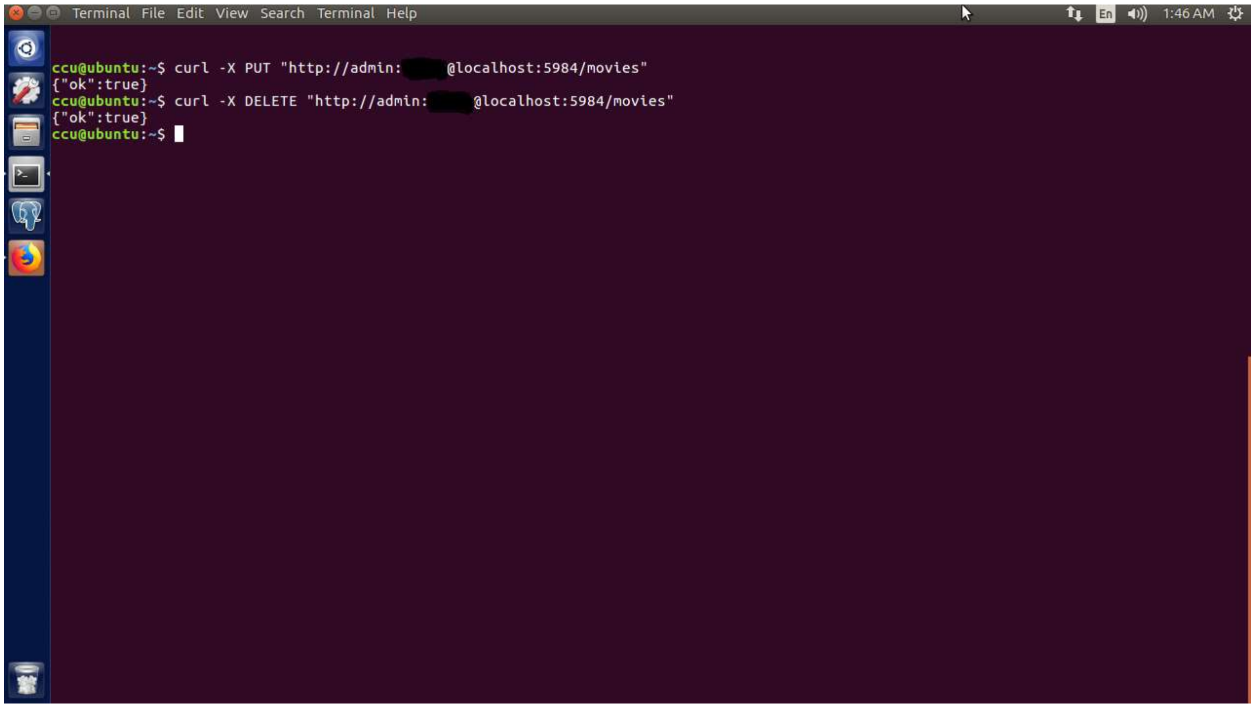
1. Use cURL to PUT a new document into the music database with a specific _id of your choice.



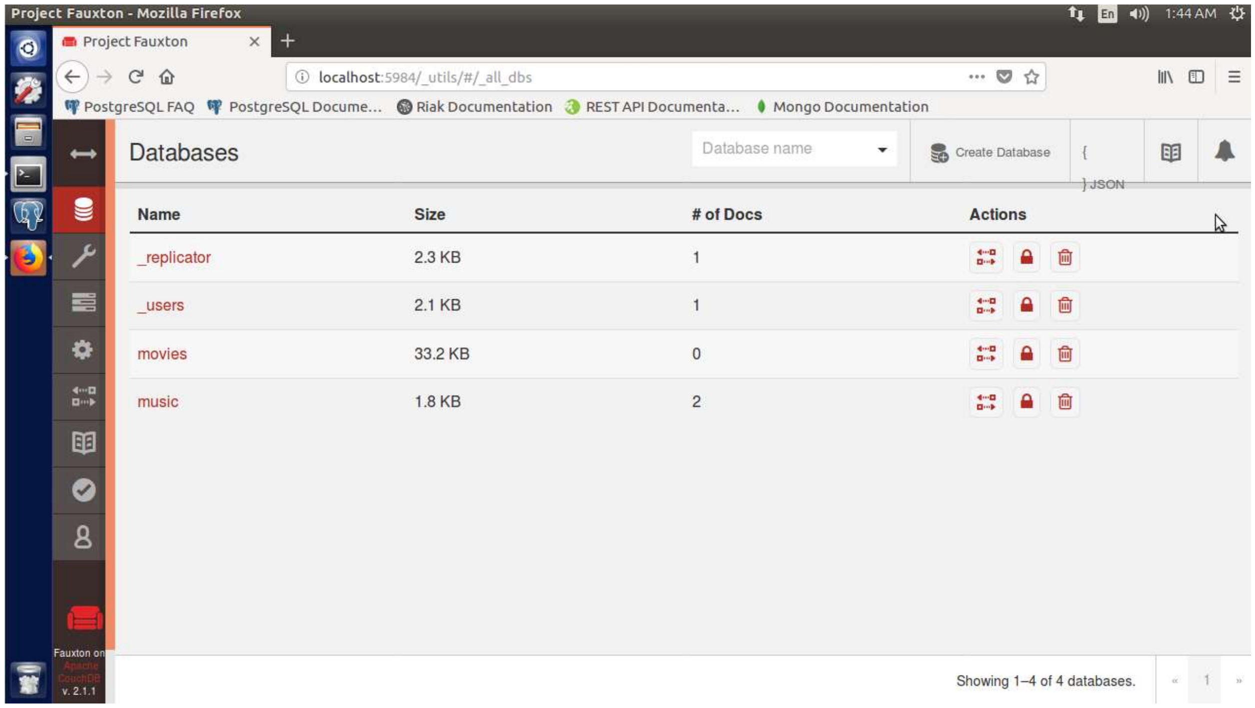
```
ccu@ubuntu:~$ curl -X PUT "http://localhost:5984/music/instruments" \
> -H "Content-Type: application/json" \
> -d '{"type": "Guitar"}'
{"ok":true,"id":"instruments","rev":"1-26b35272418d531d9df66ce13b13c57f"}
ccu@ubuntu:~$
```

All this took was a simple cURL PUT command. For the **_id**, I decided on **instruments**, keeping with the music theme, and I also went ahead and put a in a **type** field with the **"Guitar"** type.

2. Use curl to create a new database with a name of your choice, and then delete that database also via cURL.



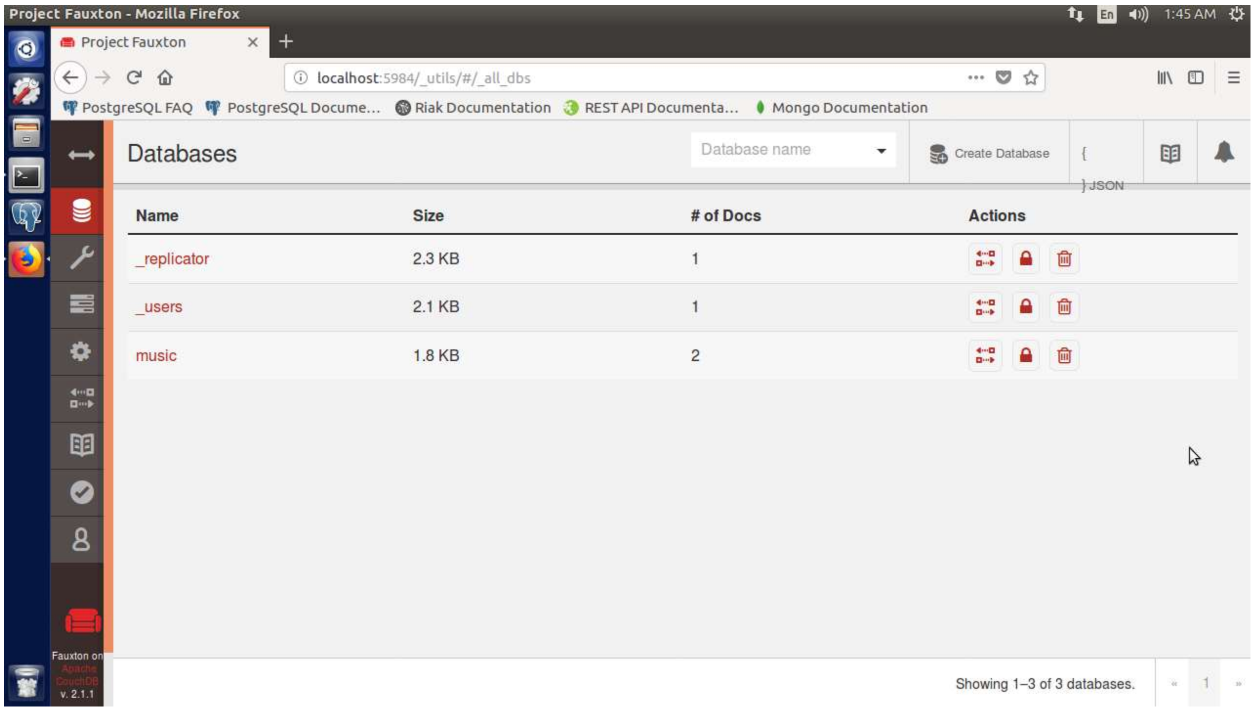
```
ccu@ubuntu:~$ curl -X PUT "http://admin:[redacted]@localhost:5984/movies"
{"ok":true}
ccu@ubuntu:~$ curl -X DELETE "http://admin:[redacted]@localhost:5984/movies"
{"ok":true}
ccu@ubuntu:~$
```



Databases

Name	Size	# of Docs	Actions
_replicator	2.3 KB	1	
_users	2.1 KB	1	
movies	33.2 KB	0	
music	1.8 KB	2	

Showing 1–4 of 4 databases.

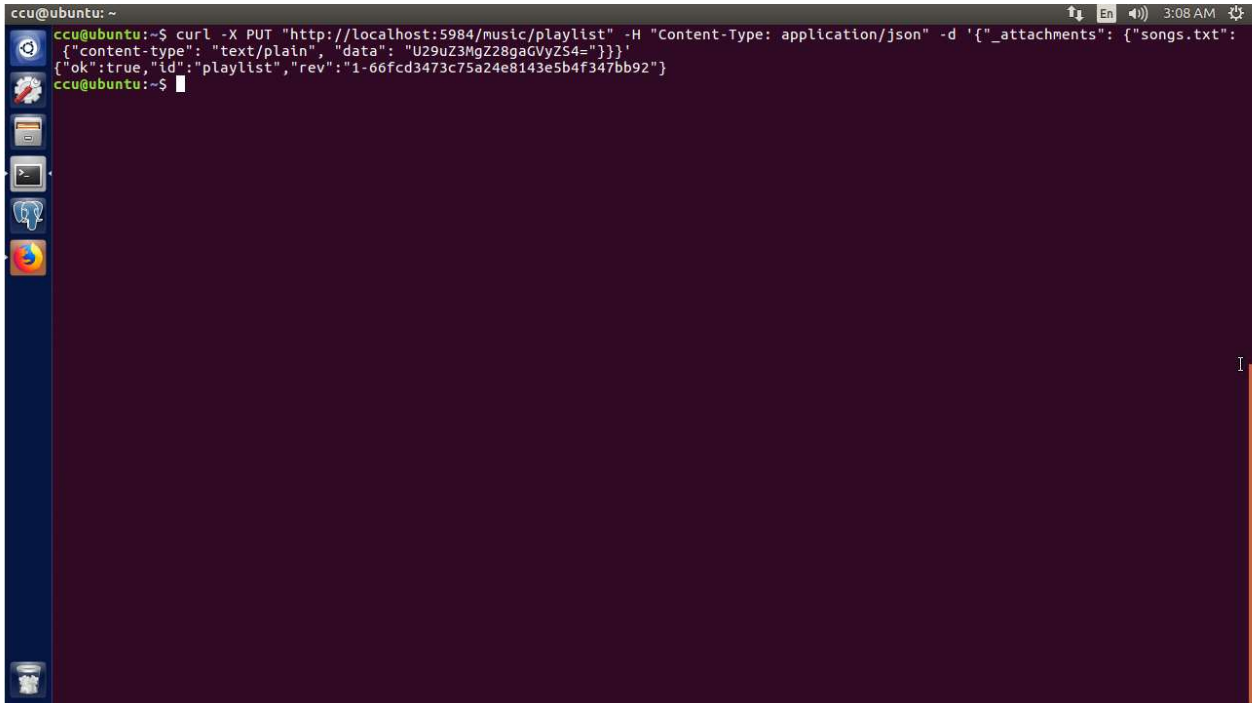


Name	Size	# of Docs	Actions
_replicator	2.3 KB	1	
_users	2.1 KB	1	
music	1.8 KB	2	

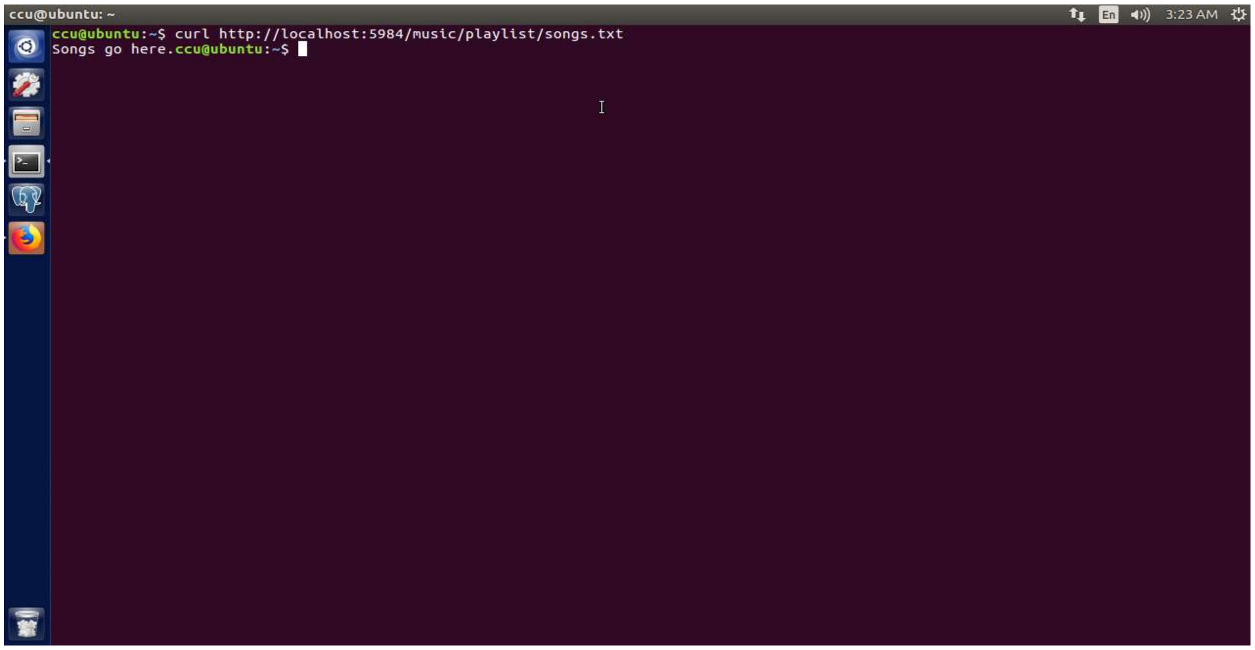
Showing 1–3 of 3 databases.

When I first tried creating a database, I was getting an error that said "You are not a server admin.". I found out that to create a database, you have to be a server admin, but creating and modifying documents does not require you to be one. The process to become an admin was easy, as I just went to the account section of Fauxton (no longer Futon), and created an admin account. Now, when creating a database, I just had to use the standard PUT command with my admin username and password in the URL, like so: "http://admin:password@localhost:5984/database". Deleting the database also required using a username and password.

3. Again using cURL, create a new document that contains a text document as an attachment. Lastly, craft and execute a cURL request that will return just that document's attachment.



```
ccu@ubuntu:~$ curl -X PUT "http://localhost:5984/music/playlist" -H "Content-Type: application/json" -d '{"_attachments": {"songs.txt":
 {"content-type": "text/plain", "data": "U29uZ3MgZ28gaGVyZS4="}}}'
{"ok":true,"id":"playlist","rev":"1-66fcd3473c75a24e8143e5b4f347bb92"}
ccu@ubuntu:~$
```



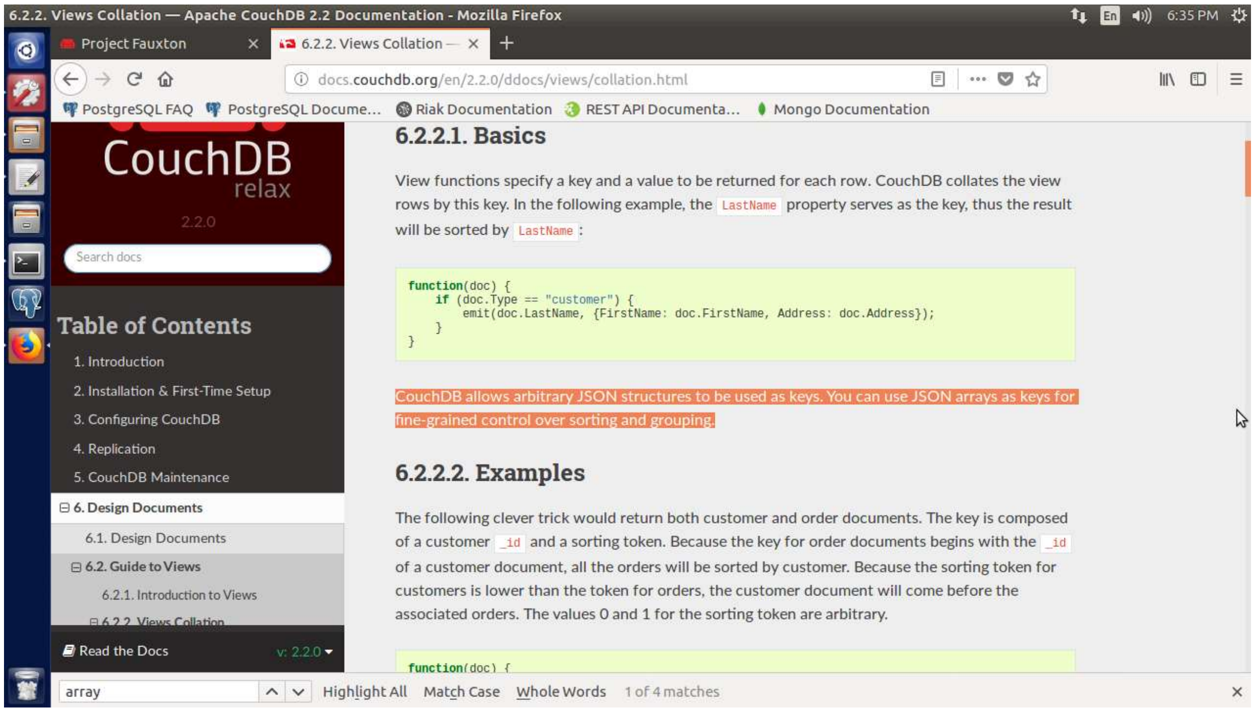
```
ccu@ubuntu:~$ curl http://localhost:5984/music/playlist/songs.txt
Songs go here.ccu@ubuntu:~$
```

I had to do some research to learn how to create a document with an attachment. All it involves is adding the **"_attachments"** field in the data portion of the JSON and under that field, adding the attachment name and the content type and data of the attachment. Since I used plain text as the attachment type, its data had to be encoded in Base64, so I went to base64encode.org and encoded the contents to be used in the file. To display just the attachment contents, I used a standard cURL GET request and pointed to the name of the attachment.

Day 2

Find:

1. We've seen that the emit() method can output keys that are strings. What other types of values does it support? What happens when you emit an array of values as a key?



6.2.2.1. Basics

View functions specify a key and a value to be returned for each row. CouchDB collates the view rows by this key. In the following example, the `LastName` property serves as the key, thus the result will be sorted by `LastName`:

```
function(doc) {
    if (doc.Type == "customer") {
        emit(doc.LastName, {FirstName: doc.FirstName, Address: doc.Address});
    }
}
```

CouchDB allows arbitrary JSON structures to be used as keys. You can use JSON arrays as keys for fine-grained control over sorting and grouping.

6.2.2.2. Examples

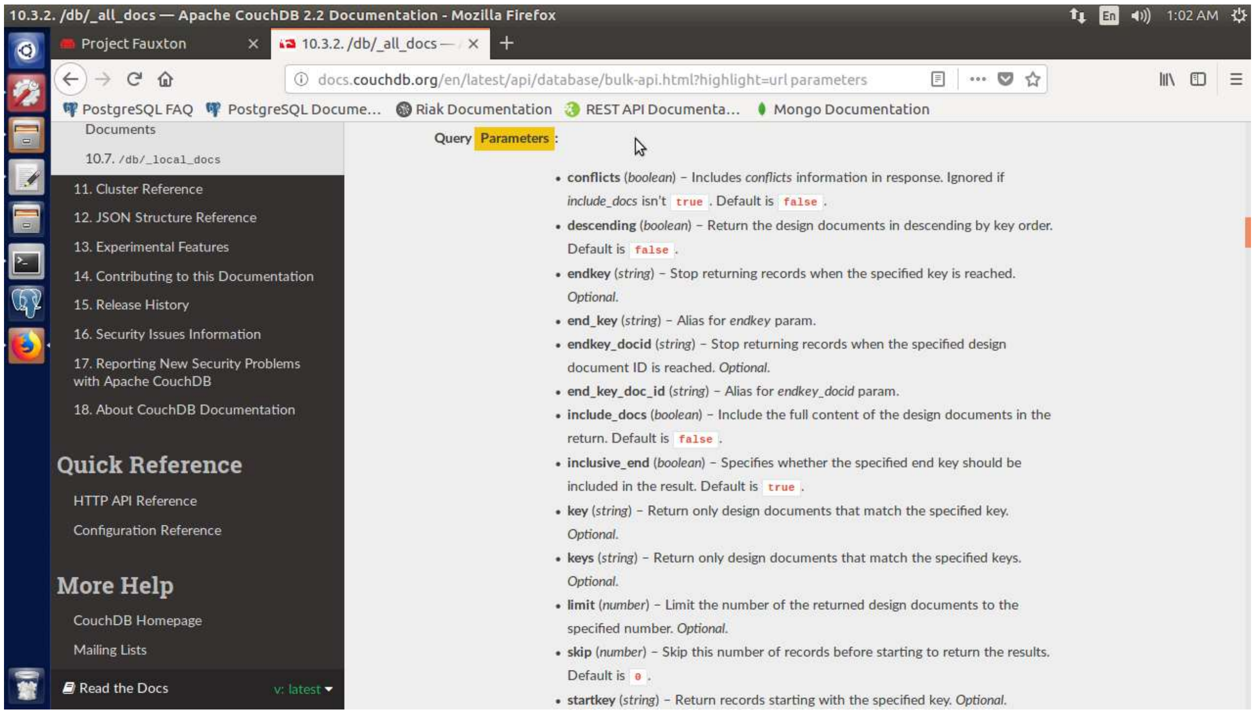
The following clever trick would return both customer and order documents. The key is composed of a customer `_id` and a sorting token. Because the key for order documents begins with the `_id` of a customer document, all the orders will be sorted by customer. Because the sorting token for customers is lower than the token for orders, the customer document will come before the associated orders. The values 0 and 1 for the sorting token are arbitrary.

```
function(doc) {
```

According to the documentation under 6. Design Documents -> 6.2 Guide to Views, any JSON object can be used as a key, including arrays which offer greater control over grouping/sorting.

Unfortunately, I was unable to find a handy list of supported values, however, from all of the examples I have seen, **emit()** can emit anything as long as it is a key/value pair in the document.

2. Find a list of available URL parameters (like limit and startkey) that can be appended to view requests and what they do.


10.3.2. /db/_all_docs — Apache CouchDB 2.2 Documentation - Mozilla Firefox
Project Fauxton
10.3.2. /db/_all_docs —
docs.couchdb.org/en/latest/api/database/bulk-api.html?highlight=url parameters
PostgreSQL FAQ
PostgreSQL Docume...
Riak Documentation
REST API Documenta...
Mongo Documentation
Documents
10.7. /db/_local_docs
11. Cluster Reference
12. JSON Structure Reference
13. Experimental Features
14. Contributing to this Documentation
15. Release History
16. Security Issues Information
17. Reporting New Security Problems with Apache CouchDB
18. About CouchDB Documentation
Quick Reference
HTTP API Reference
Configuration Reference
More Help
CouchDB Homepage
Mailing Lists
Read the Docs
v: latest

Query Parameters:

- **conflicts** (*boolean*) – Includes *conflicts* information in response. Ignored if *include_docs* isn't `true`. Default is `false`.
- **descending** (*boolean*) – Return the design documents in descending by key order. Default is `false`.
- **endkey** (*string*) – Stop returning records when the specified key is reached. *Optional.*
- **end_key** (*string*) – Alias for *endkey* param.
- **endkey_docid** (*string*) – Stop returning records when the specified design document ID is reached. *Optional.*
- **end_key_doc_id** (*string*) – Alias for *endkey_docid* param.
- **include_docs** (*boolean*) – Include the full content of the design documents in the return. Default is `false`.
- **inclusive_end** (*boolean*) – Specifies whether the specified end key should be included in the result. Default is `true`.
- **key** (*string*) – Return only design documents that match the specified key. *Optional.*
- **keys** (*string*) – Return only design documents that match the specified keys. *Optional.*
- **limit** (*number*) – Limit the number of the returned design documents to the specified number. *Optional.*
- **skip** (*number*) – Skip this number of records before starting to return the results. Default is `0`.
- **startkey** (*string*) – Return records starting with the specified key. *Optional.*

A list of valid query parameters can be found in the CouchDB docs by searching for "query parameters", selecting section 10.3.2 and scrolling down to section 10.3.3, which covers design documents. What the book calls "URL parameters" seem to be more accurately referred to as "query parameters" around the internet, so it was hard to find information on them until I discovered that.

Do:

1. The import script import_from_jamendo.rb assigned a random number to each artist by adding a property called random. Create a mapper function that will emit key-value pairs where the key is the random number and the value is the band's name. Save this in a new design document named _design/random with the view name artist.

I could not import the Jamendo data, as something was wrong with the libxml package, so I could not complete this task as well as the next two.

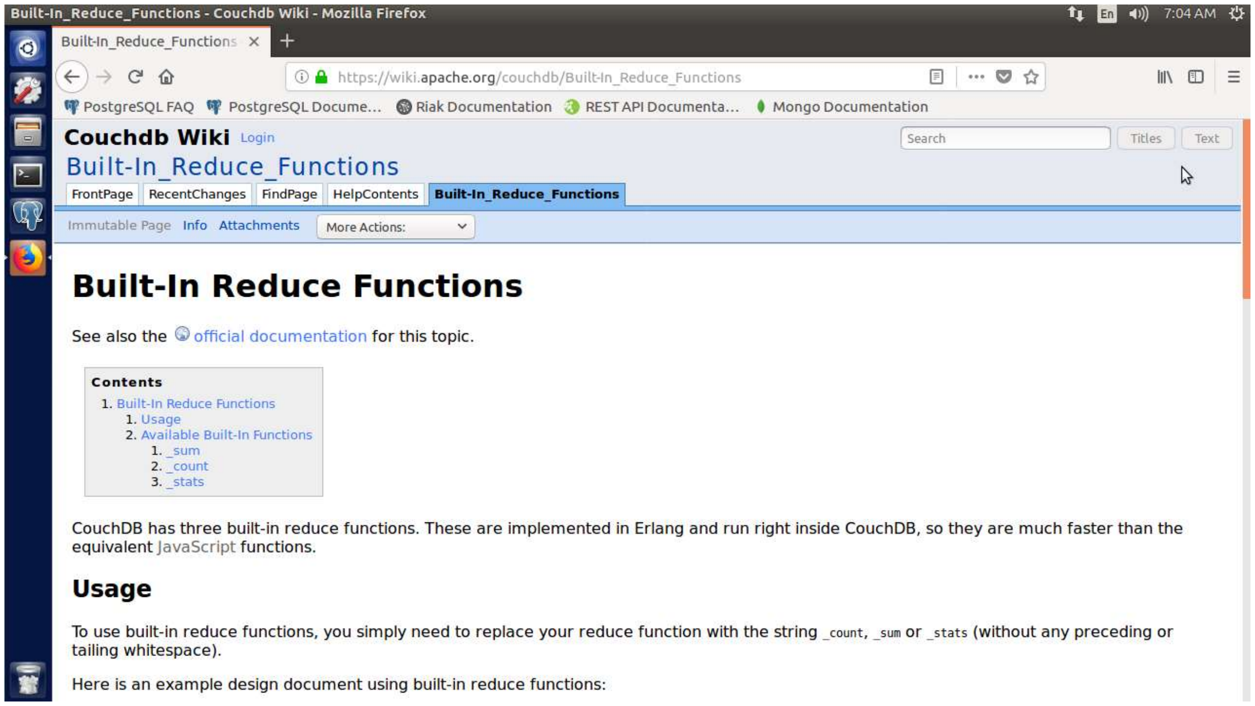
2. Craft a cURL request that will retrieve a random artist.

3. The import script also added a random property for each album, track, and tag. Create three additional views in the _design/random design document with the view names album, track, and tag to match the earlier artist view.

Day 3

Find:

1. What native reducers are available in CouchDB? What are the benefits of using native reducers over custom JavaScript reducers?


Built-In_Reduce_Functions - Couchdb Wiki - Mozilla Firefox

https://wiki.apache.org/couchdb/Built-In_Reduce_Functions

Couchdb Wiki Login

Built-In_Reduce_Functions

Built-In Reduce Functions

See also the official documentation for this topic.

Contents
1. Built-In Reduce Functions
   1. Usage
   2. Available Built-In Functions
      1. _sum
      2. _count
      3. _stats

CouchDB has three built-in reduce functions. These are implemented in Erlang and run right inside CouchDB, so they are much faster than the equivalent JavaScript functions.

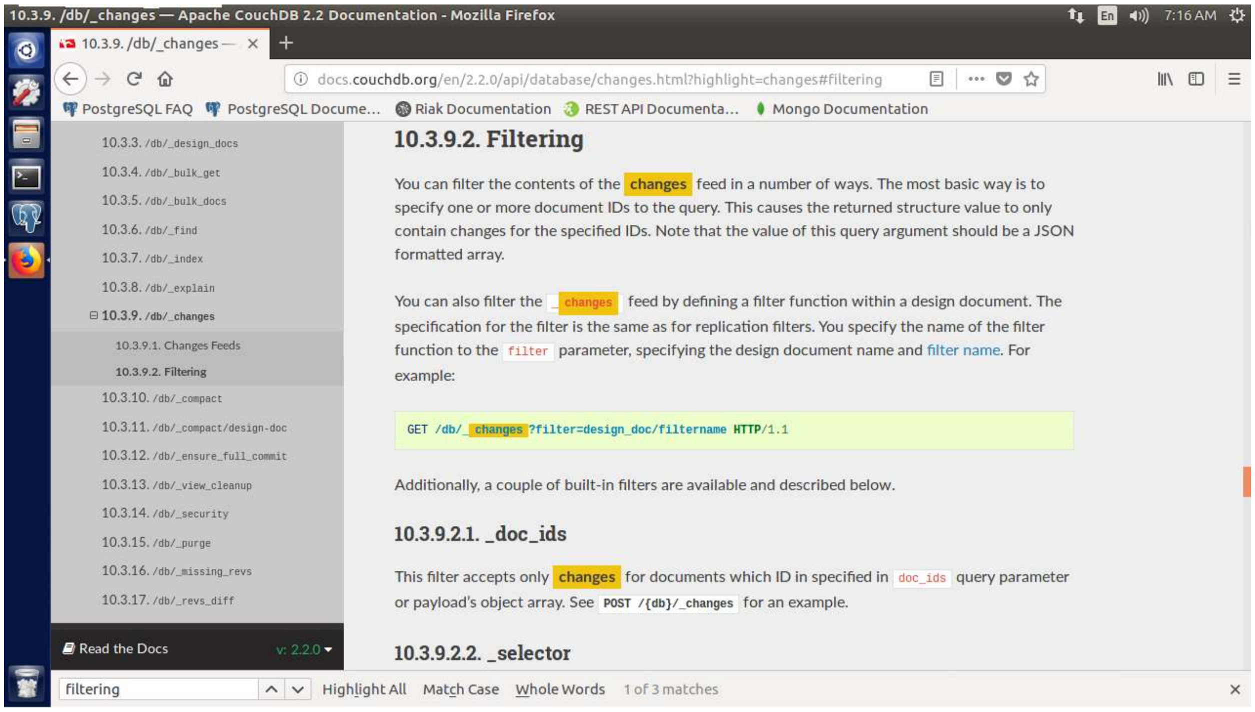
Usage

To use built-in reduce functions, you simply need to replace your reduce function with the string `_count`, `_sum` or `_stats` (without any preceding or tailing whitespace).

Here is an example design document using built-in reduce functions:

The CouchDB wiki has a list of native reduce functions. According to the wiki page, native reducers run inside of CouchDB unlike JavaScript, so the benefit is that they are faster.

2. How can you filter the changes coming out of the _changes API on the server side?



10.3.9.2. Filtering

You can filter the contents of the changes feed in a number of ways. The most basic way is to specify one or more document IDs to the query. This causes the returned structure value to only contain changes for the specified IDs. Note that the value of this query argument should be a JSON formatted array.

You can also filter the _changes feed by defining a filter function within a design document. The specification for the filter is the same as for replication filters. You specify the name of the filter function to the `filter` parameter, specifying the design document name and filter name. For example:

```
GET /db/_changes?filter=design_doc/filtername HTTP/1.1
```

Additionally, a couple of built-in filters are available and described below.

10.3.9.2.1. _doc_ids

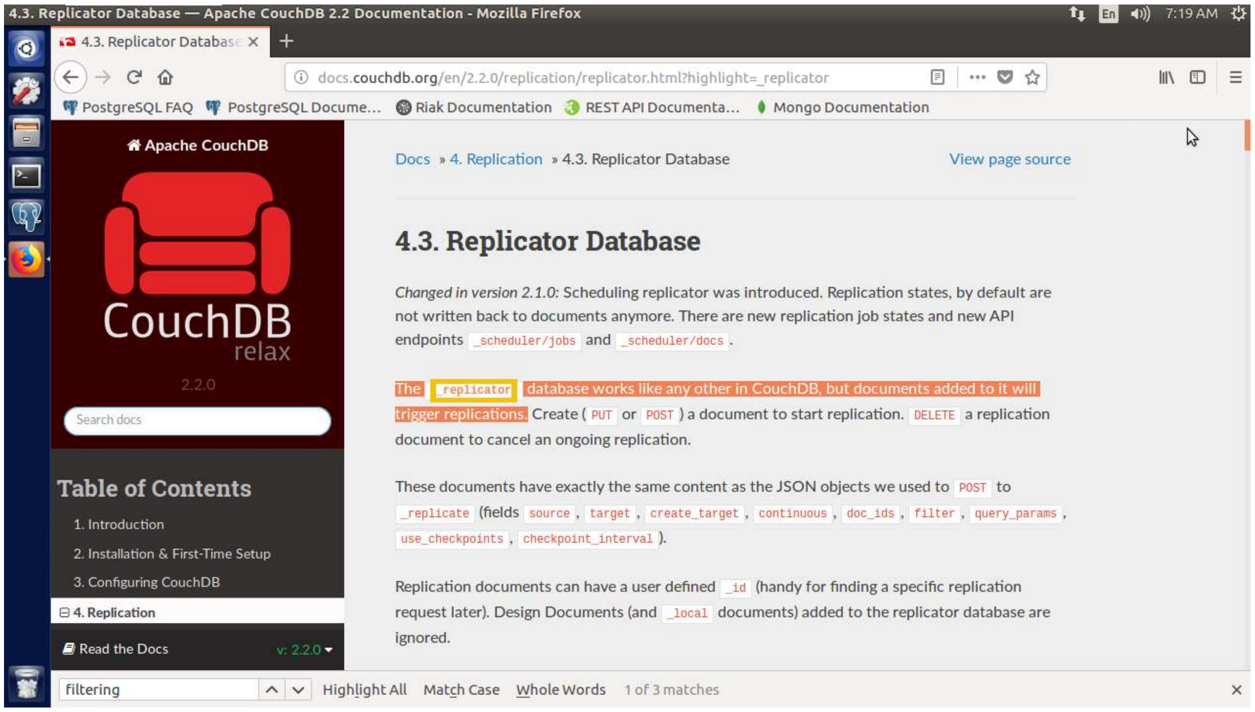
This filter accepts only changes for documents which ID in specified in `doc_ids` query parameter or payload's object array. See `POST /{db}/_changes` for an example.

10.3.9.2.2. _selector

A list of filtering methods can be found in section 10.3.9.2 of the CouchDB docs.

3. Like everything else in CouchDB, the tasks of initializing and canceling replication are controlled by HTTP commands under the hood. What are the REST commands to set up and remove replication relationships between servers?

4. How can you use the _replicator database to persist replication relationships?


4.3. Replicator Database — Apache CouchDB 2.2 Documentation - Mozilla Firefox

docs.couchdb.org/en/2.2.0/replication/replicator.html?highlight=_replicator

Docs » 4. Replication » 4.3. Replicator Database

View page source

4.3. Replicator Database

Changed in version 2.1.0: Scheduling replicator was introduced. Replication states, by default are not written back to documents anymore. There are new replication job states and new API endpoints `_scheduler/jobs` and `_scheduler/docs`.

The `_replicator` database works like any other in CouchDB, but documents added to it will trigger replications. Create (`PUT` or `POST`) a document to start replication. `DELETE` a replication document to cancel an ongoing replication.

These documents have exactly the same content as the JSON objects we used to `POST` to `_replicate` (fields `source`, `target`, `create_target`, `continuous`, `doc_ids`, `filter`, `query_params`, `use_checkpoints`, `checkpoint_interval`).

Replication documents can have a user defined `_id` (handy for finding a specific replication request later). Design Documents (and `_local` documents) added to the replicator database are ignored.

According to the documentation, documents inside the **_replicator** database trigger replications, so all you would have to do is put the document in this database.