

# CouchDB: Investigación e implementación demostrativa de una base de datos NoSQL

Sebastián Hurtado      Diego Linares      Piero Marini

28 de Noviembre del 2019

## 1 Introducción

El objetivo de la presente investigación es entender el funcionamiento de una base de datos NoSQL y realizar una demostración de esta con una colección de datos. Actualmente, la data que resulta del uso en particular de aplicaciones web, redes sociales, etc. se encuentra en un nivel bajo de estructuración. Como resultado de ello, el uso de base de datos relacionales puede no ser el más adecuado para trabajar con este tipo de data [1]. En este caso, exploraremos una de las soluciones NoSQL, que provee soporte ACID y permite trabajar con data no estructurada en formato JSON, CouchDB.

El siguiente artículo se va a encontrar dividido de la siguiente manera: primero se definirá CouchDB y se comparará frente a otras soluciones de este tipo, observando en que casos conviene el uso del primero. A continuación explicaremos más a detalle las características del modelo de datos y la arquitectura de almacenamiento distribuido. Finalmente se hará una demostración de su uso en una colección de datos del servicio web Yelp, y se realizarán conclusiones en base a la misma.

### 1.1 Definición y Propósito

CouchDB es un *cross-platform, open-source*, base de datos de tipo NoSQL. Su arquitectura interna se encuentra diseñada específicamente para la web, y permite trabajar con grandes cantidades de data. Este hace uso de un **Couch Replication Protocol** para sincronizar documentos JSON entre 2 pares a través de un protocolo HTTP. Por ejemplo, los *requests* son realizados a una determinada URL para obtener información con respecto a las bases de datos que se están manejando. Esto permite hacer uso de *CURL* o cualquier otro intérprete para trabajar con los archivos que se encuentran en una base de datos CouchDB.

Las bases de datos de CouchDB almacenan los llamados **documentos**. Estos consisten de un número variable de campos, los cuales pueden ser de diversos tipos, en conjunto con su metadata. Las ediciones por parte del cliente son realizadas a través de una carga, modificación y guardarlos de vuelta. Las

transacciones son de estilo *todo o nada*, por lo cual en la base de datos nunca se van a encontrar documentos parcialmente editados.

Un aspecto que diferencia a CouchDB de otras base de datos NoSQL es el mantenimiento de las propiedades ACID de una transacción. Las actualizaciones de documentos son serializadas, mientras que las lecturas de los mismos se realizan de forma concurrente. Para esta concurrencia es usado un *Multi-Version Concurrency Control*, en el cual cada usuario ve una etapa constante de la base de datos de inicio a fin. La estructura de datos utilizada para organizar los datos es un B-Tree. Las actualizaciones ocurren hasta el final de una transacción, dejando la base de datos en un estado consistente.

El propósito de CouchDB, es el de trabajar con data no estructurada, particularmente para la web. Hacen uso del paradigma *map-reduce*, replicación fácil y una *straightforward RESTful API* [2]. Como resultado de ello resulta útil para un ambiente de producción web. Además de las previas razones para ello, se le suma el echo de que toma de la arquitectura de la web (en palabras del desarrollador de Django *"It's built of the web"*). Otras ventajas particulares son su alta escalabilidad y la ya mencionada facilidad de replicación.

## 1.2 Cuadro comparativo con otras BD NoSQL

Las principales diferencias de CouchDB son 3: el lenguaje de implementación en el que se encuentra (Erlang), el modelo de datos que usa (JSON), y la carencia de un lenguaje de queries fijo.

El beneficio del uso de Erlang es que este se encuentra priorizado para *fault*

DB/ Properties	MongoDB	Cassandra	Accumulo	Couch DB	Hbase	Redis	Riak
Language	C++	Java	Java	Erlang	Java	C,C++	Erlang
Data Model	BSON	Big Table	Big Table	JSON	Big Table and Dynamo	Data Structure	Data structure
Fault Tolerance	Replication	Partitioning and replication	Replication	Replication	Partitioning and replication	Replication	Replication
Data Storage	Memory, file framework	Dynamo for storing data	HDFS	Memory, File framework	HDFS	File system	Bit cast, Memory
Community	AGPL	Facebook	Apache	Apache	Apache	BSD	Apache
MapReduce	YES	YES	Yes	YES	YES	NO	YES
Query Language	-	API calls	Java API, Thrift API	-	XML, Thrift API	API calls	Javascript
Replication Modes	Master-Slave Replication	Master-Slave replication	Multi-master replication	Multi-Master Replication	Master-Slave Replication	Master-Slave Replication	Multi Master Replication
Protocol	TCP/IP	Thrift	Thrift	HTTP/ REST	Thrift, API, tradition	Binary, Similar to telnet	REST

Figura 1: Cuadro comparativo de diferentes tipos de bases de datos NoSQL, tomada de [3].

*tolerance* y en segunda medida, para concurrencia. Como se mencionó anteriormente, los documentos de CouchDB se encuentran guardados en formato JSON, que se encuentra directamente relacionado con su sistema de views. El protocolo del mismo se encuentra basado en JSON, en conjunto con servidores de socket

externo; además de no haber límite para el *text-size* o número de elementos de cada documento. CouchDB carece de soporte para un lenguaje de queries declarativo, en cambio, estas son basadas en REST/HTTP. Los requests son hechos de forma directa a una URL, lo cual permite realizar pruebas de queries de forma directa, y que el acceso a las bases de datos sea estandarizado.

### 1.3 Escenarios de uso

Es necesario considerar que CouchDB, actualmente, tiene un relativo pequeño campo de mercado (usado por 3900 empresas) en comparación a otras bases de datos NoSQL. Es utilizado en casos de manejo de grandes volúmenes de data y almacenamiento basado en la nube. Razón para ello es su alta escalabilidad (las BDR tienden a no soportar más de 1024 columnas de forma estable). Muchas de sus aplicaciones toman ventaja de la replicación y *attachment management features* que son específicas de CouchDB [4]. Los ámbitos en los cuales esto es aplicable son varios ámbitos, el mismo [4] lo menciona en el caso de hospitales de radiología.

Además, CouchDB ofrece beneficios particulares para portales y gateways [5], gracias a que provee un API RESTful. Permite un acceso a data más rápido para los usuarios. Uno de los costos de esto, y debe ser tomado en consideración para el uso de CouchDB, es espacio adicional en disco disponible. Esto es debido a que sus índices Btree y los views se encuentran almacenados en este. Uno de los ejemplos, de este beneficio estuvo presente en el portal de usuarios de Teragrid en el cual se logró un aumento de velocidad de 8.24x [5] para permitir que la información se encuentre mucho más disponible al usuario.

## 2 Características

### 2.1 Modelo de datos

#### 2.1.1 Inserciones

- Fauxton o Futon (interfaz gráfica): La interfaz tiene una opción de inserción de documentos (elementos).
- Línea comandos (curl): Usando comandos HTTP, podemos interactuar con la base de datos.

– Conexión con el puerto 5984:

```
curl http://127.0.0.1:5984/
```

– Crear la base de datos ciudades:

```
curl -X PUT http://127.0.0.1:5984/ciudades
```

– Insertar el elemento Lima con id 1:

```
curl -X PUT http://127.0.0.1:5984/ciudades/1 -d '{"  
  ↪ nombre:"Lima", "pais":"Peru", "habitantes  
  ↪ ":"8575000"}'
```

- Para obtener el elemento de id 1:

```
curl -X GET http://127.0.0.1:5984/ciudades/1
```

- Python: Podemos basarnos en anteriores proyectos para insertar 100 datos a la base de datos.

```
import couchdb  
import json  
  
couch = couchdb.Server()  
db = couch.create('yelpmain')  
with open('example.json', 'r') as myfile:  
    data=myfile.read()  
  
obj = json.loads(data)  
print(obj)  
for doc in obj:  
    db.save(doc)
```

### 2.1.2 Actualizaciones

- Revision number: Es la versión que couchdb le da al documento. Para hacer una correcta actualización debes indicar la correcta versión.
- Puedes modificar todos los campos o solo uno.
- Modificar la siguiente inserción:

```
curl -X PUT localhost:5984/yelpmain/101 -d '{"id": 101, "  
  ↪ first_name": "Diana", "last_name": "Davila", "email": "  
  ↪ diana@tripod.com"}'
```

- Modificación:

```
curl -X PUT http://127.0.0.1:5984/yelpmain/101 -d '{"_id  
  ↪ ":"101", "_rev": "1-9facd005099ce22bf8cc79d63b32b494", "  
  ↪ email": "diana98@tripod.com"}'
```

### 2.1.3 Búsqueda

La búsqueda se realiza mediante un POST y find. El selector da las condiciones para el atributo a ser juzgado y fields es el filtro de campos a ser mostrados. Podemos centrarnos en los Mango Queries. Podemos usarlos en la interfaz gráfica fauxton para hacer búsquedas. Por ejemplo, seleccionamos a Hortensia para ver su correo electrónico.

```
{
  "selector":
  {
    "first_name": {"$eq": "Hortensia"}
  },
  "fields": ["email"]
}
```

O podemos usar múltiples condiciones con operadores lógicos. En este caso, una de las condiciones es que el nombre sea igual a Sebastian y la otra que haya nacido entre 1998 y 2000.

```
{
  "selector":
  {
    "$and":
    [
      {"first_name": {"$eq": "Sebastian"}},
      {"year": {"$in": [1998, 2000]}}
    ]
  },
  "fields": ["email", "year"]
}
```

### 2.1.4 Indexación

Para la indexación, vamos a manage indexes y agregamos el índice.

```
{
  "index":
  {
    "fields": ["first_name"]
  },
  "name" : "name-index",
  "type" : "json"
}
```

También podemos crear un partial index, que tiene una mejora en performance con respecto a usar un selector y el índice.

```
{
  "index":
  {
    "partial_filter_selector": {"year": {"$gt": 1998}},
    "fields": ["first_name"]
  }
}
```

Por ejemplo, en el caso anterior, se filtran a los que nacieron antes de o en 1998.

## 2.2 Arquitectura de almacenamiento distribuido

Es posible dividir la arquitectura de CouchDB en 3 componentes principales: engine de almacenamiento, engine de vistas, y replicador [6]. El almacenamiento es basado en una estructura de *BTree*, y accedidas por llave o rango (haciendo uso de las operaciones del árbol), de forma rápida. El engine de vistas se encuentra escrito en JavaScript, basado en jobs de MapReduce, por lo que sirve para índices y extracción de data. El replicador se conecta a bases de datos locales o remotas y las sincronizan.

Las relaciones entre los componentes mencionados pueden ser observados en el siguiente gráfico.

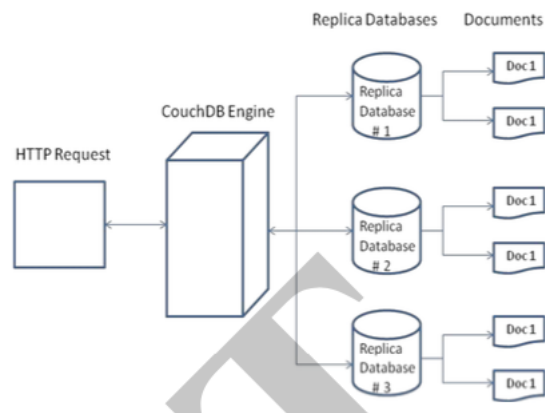


Figura 2: Gráfico de relación entre los componentes distribuidos de CouchDB, tomado de [6].

### 2.2.1 Fragmentación, Asignación, Replicación

Las inserciones en CouchDB son realizadas en forma de *append-only*, de forma que se evita la Fragmentación. Los appends son realizados al final del documento que se busca modificar. No hay una necesidad de realocación de memoria. De

la misma manera, permite realizar compactación, a través de la escritura de un *revision* a los archivos de una nueva base de datos (en la que incluye los documentos eliminados). Finalmente, la antigua base de datos es remplazada con la nueva que se encuentra compactada.

La replicación se puede realizar de dos maneras, transiente, y persistente. Una replicación transiente se refiere a la carencia de un documento que sirva de backup durante la replicación; este hace uso del api endpoint `_replicate`, y no permite realizar queries de su estado hasta que el trabajo se encuentre terminado. Las replicaciones persistentes mantienen un documento con los parametros de replicación utilizados, los cuales se guardan en una base de datos `_replicator`, en donde cada documento describe un proceso tomando lugar. Las replicaciones funcionan comparando el objetivo con el source. Unicamente se transfieren aquellos documentos que difieren y se eliminan los que no se encuentran en la fuente.

Una replicación finaliza cuando el stream de cambios se ha acabado, con algunos check-points realizado en medio en caso de fallas o caída del servidor. Las replicaciones son hechas en una única dirección por, lo cual es necesario dos procesos para realizar una replicación de master a master.

De acuerdo a la documentación oficial de CouchDB, el control de documentos a replicar puede ser hecho a través de tres métodos:

1. Local: si los documentos son locales estos nunca se replican.
2. Selector: se encuentran en los documentos de parámetro y contienen una query que determina si un determinado documento debe ser replicado o no.
3. Funciones de filtrado: por cada uno de los documentos que se van a replicar, estos pasan por una función de filtrado, que en caso de retornar verdadero, continúan con el proceso.

## 2.2.2 Procesamiento de consultas distribuidas

La computación distribuida es posible en CouchDB gracias al uso del modelo de programación MapReduce. Es esencialmente un patrón que permite implementar queries de gran escala para un motor de búsqueda [7]. El mapeo produce valores que son clasificados a través de llaves, y el reduce agrega estos resultados. La función se encuentra embebida como parte de JavaScript. Esto significa que la base de datos hace la mayor parte del trabajo, mientras que el developer se debe enfocar en la extracción de la información.

La arquitectura de CouchDB posee un feature conocido como *direct shard connection only*, donde un *shard* es una partición horizontal de data en la base de datos, con el objetivo de generar durabilidad. El componente que realiza la resolución del shard-key es el coordinador de queries. Una desventaja de CouchDB, es que este no ofrece soporte para queries que usen non-shard key values. Por otra parte, el merge de los resultados múltiples provenientes de múltiples shards es también realizado por el coordinador de queries y enviado a cada uno de los

clientes. El ordenamiento de estos resultados se encuentra soportado. Cabe resaltar la existencia de CouchDB Lounge, un paquete que permite clustering de nodos que reciben hashing constante. Este permite que estos nodos se mezclen de una forma más sencilla. Actualmente este se encuentra en proceso de traducción a Erlang para su integración en CouchDB regular.

### 3 Implementación Demostrativa

Utilizaremos instancias de docker para simular el ambiente de trabajo y poder crear un cluster de CouchDB. Para este caso específico, se crearon 3 instancias de Docker, todas corriendo una instancia de CouchDB. Cada una de estas instancias consiste de 3 replicas, cada una con 4 shards o fragmentos.

Esto se logró a través de un docker-compose, programa que permite orquestrar y configurar varias instancias de Docker para que trabajen juntas y, en este caso, dentro de la misma red para poder comunicarse entre nodos.

```
version: "3.5"
networks:
  network:
services:
  server-0:
    environment:
      COUCHDB_PASSWORD: -pbkdf2-847043
      ↪ acc65626c8eb98da6d78682fbc493a1787,
      ↪ f7b1a3e4b624f4f0bbfe87e96841eda0,10
      COUCHDB_SECRET: 0123456789abcdef0123456789abcdef
      COUCHDB_USER: admin
      NODENAME: couchdb-0.docker.com
    image: couchdb:2.3.0
    networks:
      network:
        aliases:
          - couchdb-0.docker.com
    ports:
      - "5984:5984"
      - "5986:5986"
    volumes:
      - "volume-0:/opt/couchdb/data"
  server-1:
    environment:
      COUCHDB_PASSWORD: -pbkdf2-847043
      ↪ acc65626c8eb98da6d78682fbc493a1787,
      ↪ f7b1a3e4b624f4f0bbfe87e96841eda0,10
      COUCHDB_SECRET: 0123456789abcdef0123456789abcdef
      COUCHDB_USER: admin
      NODENAME: couchdb-1.docker.com
```



```

image: couchdb:2.3.0
networks:
  network:
    aliases:
      - couchdb-1.docker.com
ports:
  - "15984:5984"
  - "15986:5986"
volumes:
  - "volume-1:/opt/couchdb/data"
server-2:
  environment:
    COUCHDB_PASSWORD: -pbkdf2-847043
    ↪ acc65626c8eb98da6d78682fbc493a1787,
    ↪ f7b1a3e4b624f4f0bbfe87e96841eda0,10
    COUCHDB_SECRET: 0123456789abcdef0123456789abcdef
    COUCHDB_USER: admin
    NODENAME: couchdb-2.docker.com
image: couchdb:2.3.0
networks:
  network:
    aliases:
      - couchdb-2.docker.com
ports:
  - "25984:5984"
  - "25986:5986"
volumes:
  - "volume-2:/opt/couchdb/data"
volumes:
  volume-0:
  volume-1:
  volume-2:

```

Luego, se creó la base de datos en uno de los nodos y esta fue replicada en los otros 2.

```

curl -X PUT "http://admin:password@localhost:5984/db2-project3
↪ ?q=4&n=3"

```

Para comprobar:

```

curl -s "http://admin:password@localhost:5984/db2-project3/
↪ _shards" | jq .

```

```

{
  "shards": {
    "00000000-3fffffff": [

```

```

        "couchdb@couchdb-0.docker.com",
        "couchdb@couchdb-1.docker.com",
        "couchdb@couchdb-2.docker.com"
    ],
    "40000000-7fffffff": [
        "couchdb@couchdb-0.docker.com",
        "couchdb@couchdb-1.docker.com",
        "couchdb@couchdb-2.docker.com"
    ],
    "80000000-bfffffff": [
        "couchdb@couchdb-0.docker.com",
        "couchdb@couchdb-1.docker.com",
        "couchdb@couchdb-2.docker.com"
    ],
    "c0000000-ffffffff": [
        "couchdb@couchdb-0.docker.com",
        "couchdb@couchdb-1.docker.com",
        "couchdb@couchdb-2.docker.com"
    ]
}
}

```

### 3.1 Carga de una colección de datos

```

curl -X POST http://admin:password@localhost:5984/db2-project3
  ↪ /_bulk_docs -d @data.json -H "Content-Type:application/
  ↪ json"

```

Desafortunadamente, cargando un archivo json de datos con más de 1 millón de líneas no fue posible para couchdb. En este caso decidimos hacer un split del archivo en batches de 100000 elementos y subirlos individualmente para poder cargar toda la data que teníamos disponible.

### 3.2 Consulta de datos de forma distribuida

Queries:

- Búsqueda por *compliment\_count*
- Búsqueda por *date* (Rango)
- Búsqueda por *business\_id* (Match exacto)

```

curl -X POST http://admin:password@localhost:5984/db2-project3
  ↪ /_find/ -d '{ "selector": { "compliment_count": { "$gt
  ↪ ": 5 } } }' -H "Content-Type:application/json"

```

```
curl -X POST http://admin:password@localhost:5984/db2-project3
  ↳ /_find/ -d '{ "selector": { "date": { "$gt":
  ↳ "2012-02-06 01:30:34", "$lt": "2012-02-08 06:40:50" } }
  ↳ }' -H "Content-Type:application/json"
curl -X POST http://admin:password@localhost:5984/db2-project3
  ↳ /_find/ -d '{ "selector": { "business_id": { "$eq": "
  ↳ UOST50KiVrSFTPMXGH9njg" } } }' -H "Content-Type:
  ↳ application/json"
```

## 4 Conclusiones

Sintetizando, posterior a investigar y realizar una simple implementación demostrativa de CouchDB de forma satisfactoria, se pueden plantear las siguientes conclusiones.

1. Los beneficios de replicación de CouchDB permite ventajas tangibles para trabajar con bases de datos distribuidas de gran tamaño. La replicación transiente permite *backwards compatibility* con versiones antiguas de la plataforma. Por otra parte la replicación persistente permite controlar con mayor libertad los parametros y que un desarrollador este al tanto de como esta tomando lugar el proceso en cuestión. Además, se ofrecen varias formas de realizar, lo cual permite tomar la decisión optima (entre selector y funciones de filtrado). En resumen, esto da lugar a una base de datos altamente escalable que además es resistente a fallos gracias al uso de checkpoints y los documentos de diferencia. La única desventaja a considerar es la carencia de replicación de master a master, para lo cual son necesario dos procesos (de ida y vuelta).
2. Los mango queries usados en CouchDB tienen formato de diccionario, por lo que su parseo es sencillo. Con esto, en una implementación más profesional, podríamos alterar los selectores según parámetros definidos de manera sencilla en un lenguaje como *Python* usando la librería *json*. Por ejemplo, podríamos llevar la cuenta del acceso a cada uno de los atributos de las tablas y comenzar a crear índices o particiones en la tabla a partir de una heurística. Podríamos mantener el campo como una variable e ir cambiando esta variable según el algoritmo. En resumen, este tipo de query pueden ser fácilmente parseados, configurados y aplicados en *running time*, dándole mucha más flexibilidad a nuestra base de datos.
3. La habilidad de Couchdb de manejar su fragmentación, junto con sus habilidades de replicación, crean un ambiente de desarrollo muy amigable y fácil de aprender y aplicar. Couchdb solo necesita saber cuantas *réplicas* quieres crear, y cuantas *shards* quieres en cada réplica y el sistema distribuirá equivalentemente entre todos los nodos presentes. Esto no quita la posibilidad que el usuario puede asignar manualmente cada shard y

cada réplica al lugar específico que la requiere. Gracias a esto pudimos crear un ambiente de desarrollo utilizando Docker en muy poco tiempo y estar listos para cargar la data y crear nuestros queries.

## Referencias

- [1] Sangeeta Gupta et al. Efficient query analysis and performance evaluation of the nosql data store for bigdata. *Proceedings of the First International Conference on Computational Intelligence and Informatics. Advances in Intelligent Systems and Computing*, 507:549–558, 2016.
- [2] Reuven M. Lerner. At the forge: Couchdb. *Linux Journal*, 195:8, 2010.
- [3] Sandeep Kumar et al. Comparison of nosql database and traditional database-an emphatic analysis. *International Journal on Informatics Visualization*, 2:51–55, 2018.
- [4] Matthew R. Hanlon et al. Informatics in radiology use of couchdb for document-based storage of dicom objects. *Radiographics*, 32:913–27, 2012.
- [5] Matthew R. Hanlon et al. Benefits of nosql databases for portals and science gateways. *Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery*, 1:36, 2011.
- [6] Rabi Prasad Padhy et al. Rdbms to nosql: Reviewing some next-generation non-relational database’s. *International Journal of Advanced Engineering Sciences and Technologies*, 11:15–22, 2011.
- [7] Michele Sciabarrà. *Learning Apache OpenWhisk: Developing Open Serverless Solutions*. O’Reilly Media, Inc., Sebastopol, California, 2019.