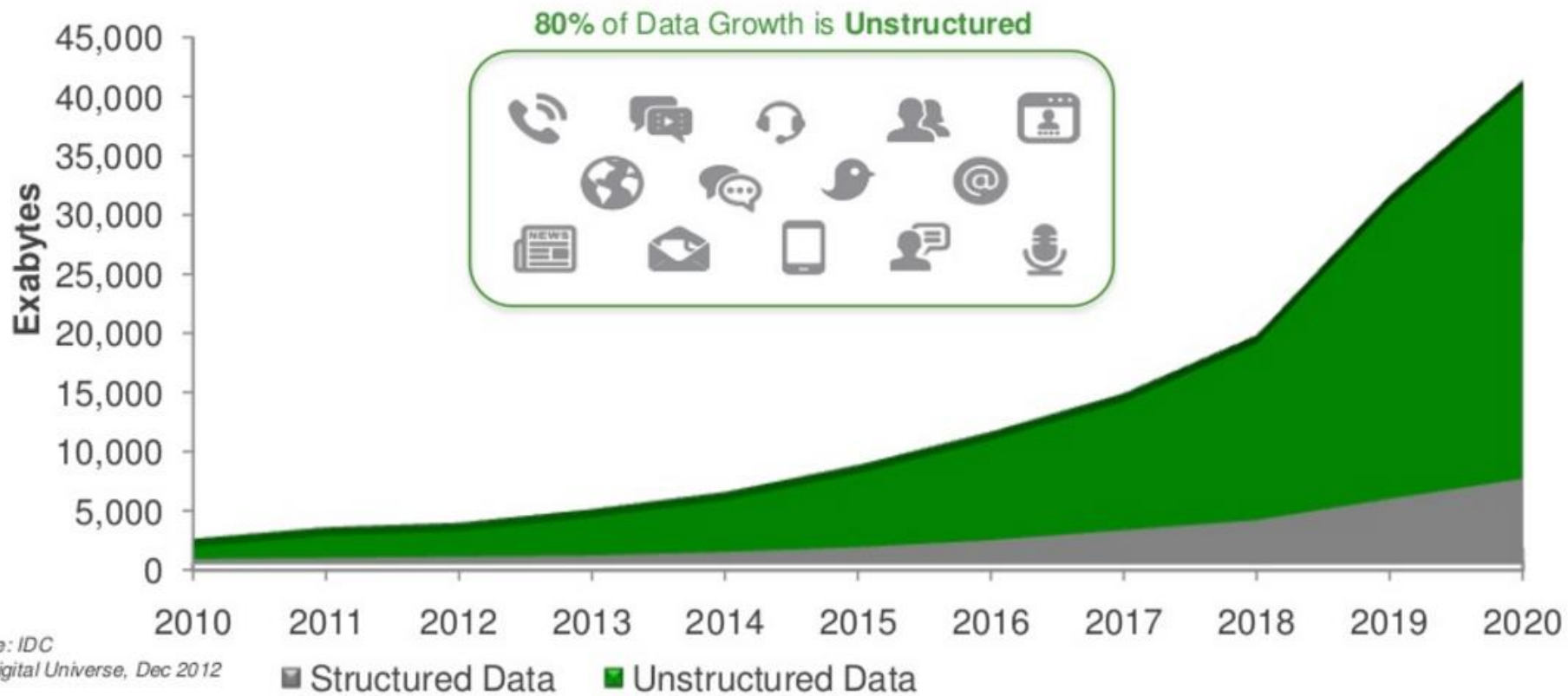




COUCHDB INTRODUCCIÓN

Enrique Barra

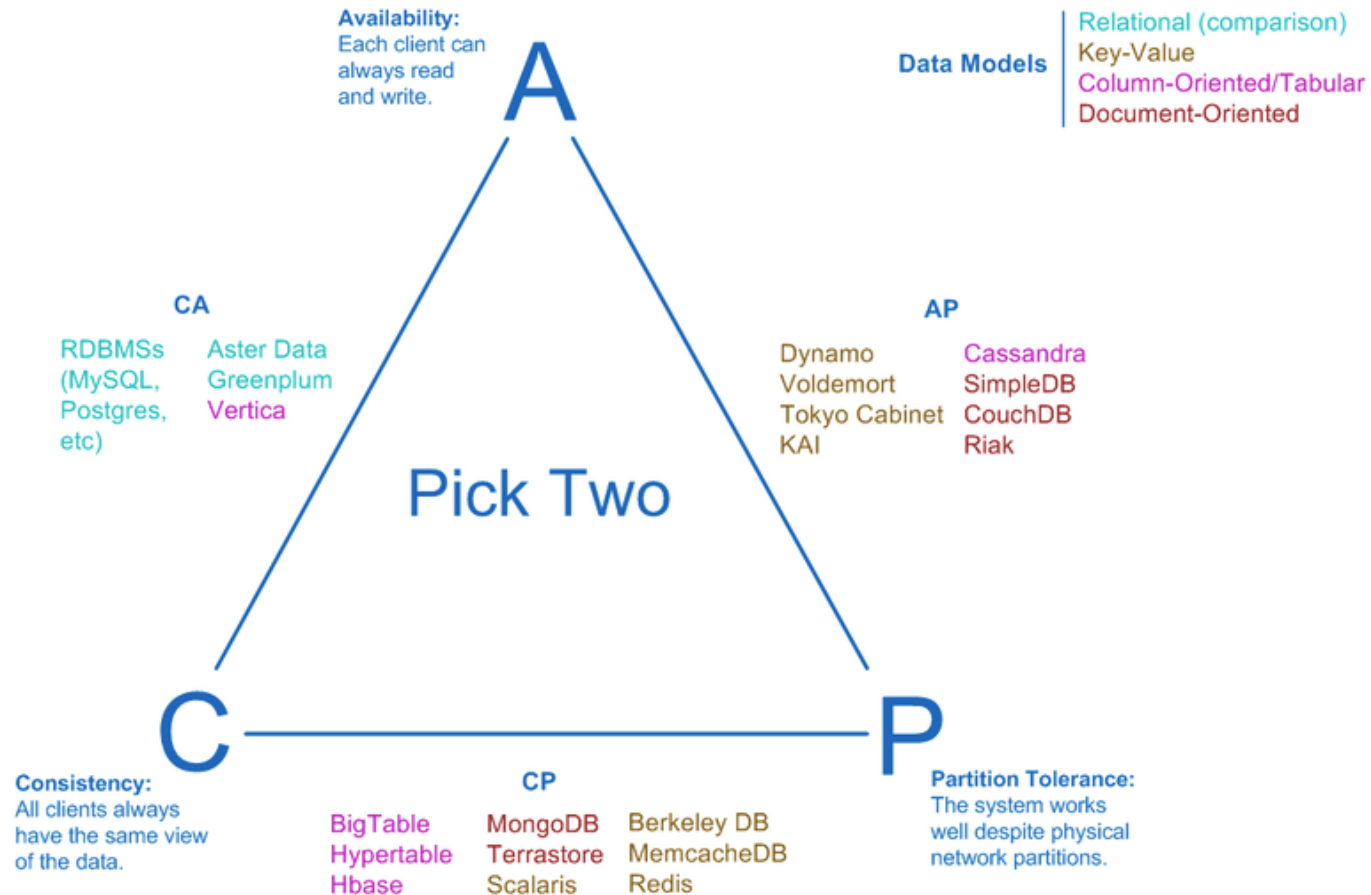
Worldwide Corporate Data Growth



CAP

- CouchDB ofrece consistencia eventual.

<http://guide.couchdb.org/draft/consistency.html>

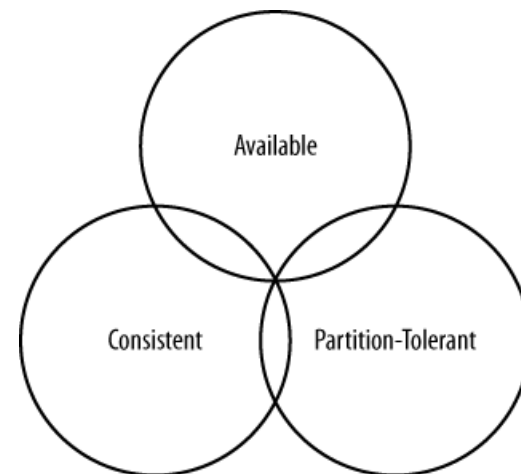


EL TEOREMA CAP

- **Teorema de Brewer:** “es imposible para un sistema computacional distribuido ofrecer simultáneamente las siguientes tres garantías”:
 - **Consistencia** (*Consistency*) – todos los nodos ven los mismos datos al mismo tiempo. Siempre que un dato es actualizado, todos los usuarios tienen acceso a esa última versión
 - **Disponibilidad** (*Availability*) – garantiza que cada petición recibe una respuesta acerca de si tuvo éxito o no. Cualquier operación puede ser ejecutada sin demora
 - **Tolerancia a la partición** (*Partition*) – los datos son distribuidos a través de dos o más nodos de la red y el sistema puede seguir funcionando, incluso, cuando algunos de estos nodos son totalmente inaccesibles

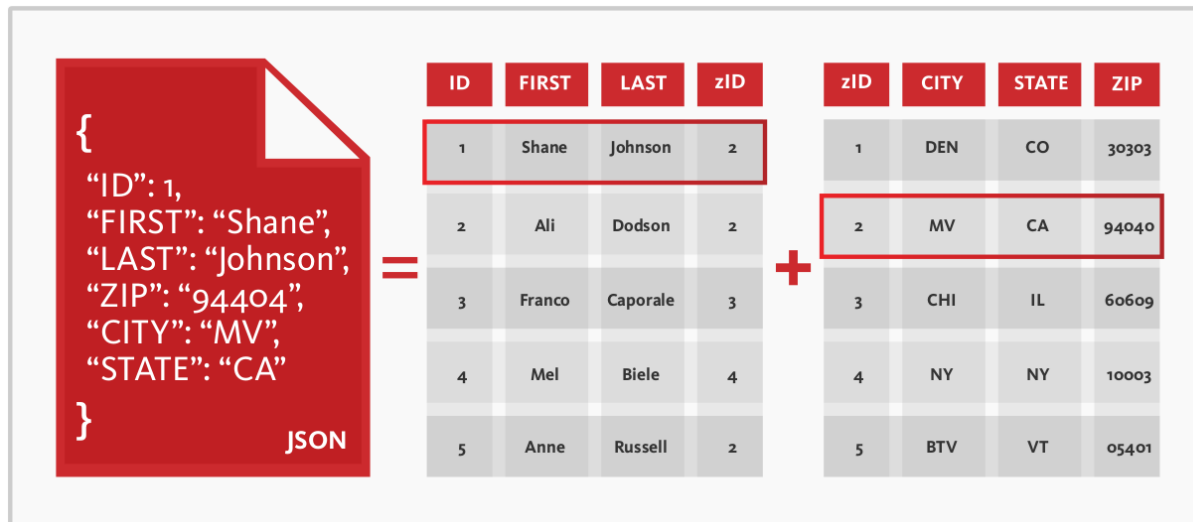
Equivalente a:

“You can have it good,
you can have it fast, you
can have it cheap: pick
two.”



BBDD ORIENTADAS A DOCUMENTOS

- La precursora fue **Lotus Notes**
- Modelo de datos: colecciones de documentos (JSON, XML, BSON) que contienen colecciones de claves-valor
- Ejemplos: **CouchDB**, MongoDB
- Buenas en:
 - Permiten trabajar con datos más complejos, admitiéndose documentos (objetos) anidados
 - Se corresponde con la manera en que se modelan los objetos y sus propiedades en los lenguajes OO
 - Orientas a la web: CRUD



RELAX

Apache CouchDB has started. Time to relax.

- <https://docs.couchdb.org/en/stable/intro/why.html>
- Razones:
 - Aprender CouchDB y entender sus conceptos básicos debería ser **natural** para casi todo el mundo que ha trabajado en web
 - Arquitectura interna es **tolerante a fallos**
 - **Escala**, impone limitaciones pero siempre pensando en que se imponen para favorecer el escalado (te impone limitaciones que luego te meterán en problemas de escalado si no estuviesen)



¿DÓNDE SE USA COUCHDB?

- CERN. El colisionador de Hadrones.
<http://readwrite.com/2010/08/26/lhc-couchdb/>
- BBC
- ...
- <https://cwiki.apache.org/confluence/display/COUCHDB/CouchDB+in+the+wild>



CouchDB Overview

DEFINICIÓN

- Apache CouchDB, comúnmente llamada **CouchDB**, es un gestor de bases de datos de código abierto, cuyo foco está puesto en la **facilidad de su uso** y en ser "una base de datos que asume la web de manera completa". Se trata de una base de datos **NoSQL** que emplea **JSON** para almacenar los datos, **JavaScript** como lenguaje de consulta por medio de **MapReduce** y HTTP para su **API RestFul**
- Las labores de administración se facilitan por medio de una aplicación web incorporada, llamada **Fauxton** (Futón)
- Características: **replicaciones**, consistencia eventual, replicación incremental, tolerancia a los fallos ...
- CouchDB fue liberada por primera vez en 2005, transformándose en un proyecto Apache en 2008

CARACTERÍSTICAS

1. Almacenamiento de documentos
2. Semántica ACID
3. Vistas e índices MapReduce
4. Arquitectura distribuida con replicación
5. Interfaz REST
6. Consistencia Eventual
7. Hecha para operar offline
8. Ultra Durable

CARACTERÍSTICA I - BBDD DE DOCUMENTOS

- CouchDB almacena los datos como documentos JSON. Todos los documentos en una base de datos CouchDB tienen un identificador único y no requieren un esquema determinado.
- Son autocontenidos, no hay relaciones, no hay joins, ...
- Pueden contener datos binarios en el campo `_attachments`

```
{  
  "_id" : "2d7f015226a05b6940984bbe39004fde",  
  "_rev" : "2-477f6ab2dec6df185de1a078d270d8",  
  "first_name" : "Enrique",  
  "last_name" : "Barra",  
  "interests" : ["programación", "baloncesto", "series"],  
  "projects" : [  
    { "name" : "ViSH", "stars" : 5, "url": "http://vishub.org" },  
    { "name": "EducaInternet", "stars": 8, "url": "http://educainternet.es"}  
  ]  
}
```

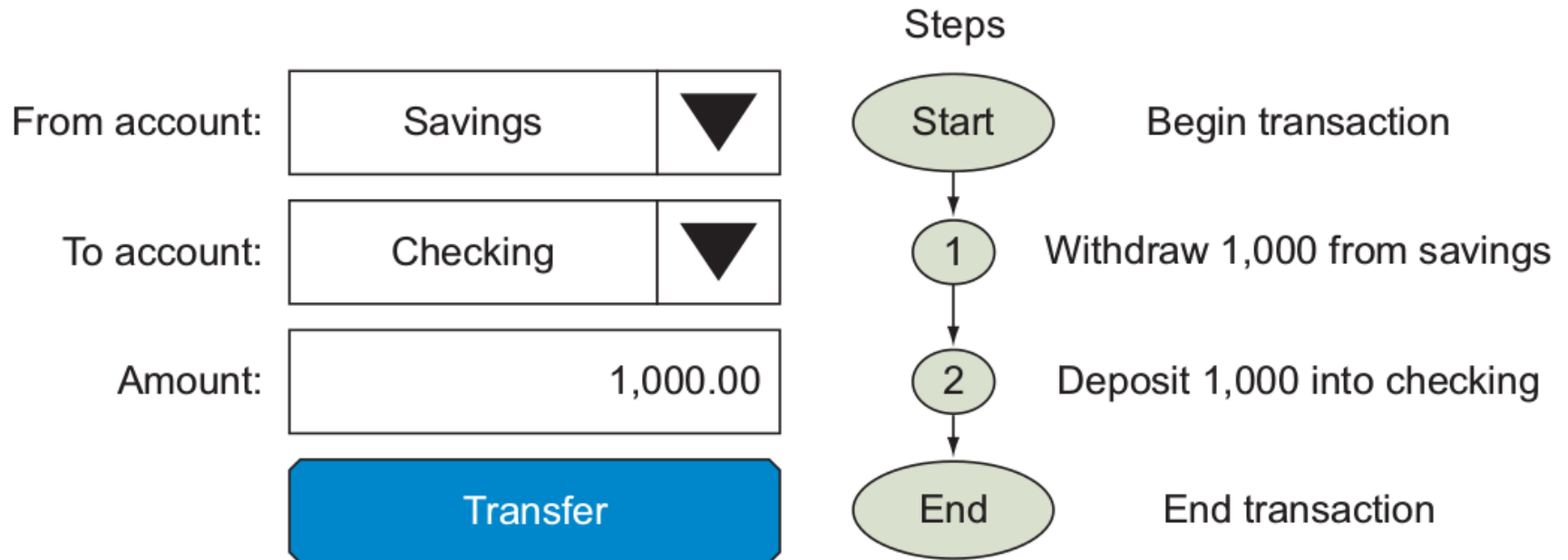
CARACTERÍSTICA II - SEMÁNTICA ACID

- CouchDB provee una semántica ACID (atomicidad, consistencia, aislamiento y durabilidad).
- Lo hace implementando una forma de control de concurrencia multiversión, lo que significa que CouchDB puede manejar un gran número de lecturas y escrituras en paralelo, sin que surjan conflictos.
- Más info:
- <https://docs.couchdb.org/en/latest/intro/overview.html#acid-properties>

RECORDEMOS ACID

- *Atomicity - Consistency - Isolation - Durability*
- Atomicidad, Consistencia, aislamiento y Durabilidad
- Una secuencia de operaciones (transacción):
 - Se ejecutará del todo o nada (**A**)
 - Una vez completada, la BD quedará en un estado en el que no se viola ninguna restricción de integridad (**C**)
 - Las transacciones concurrentes son independientes y no se afectan unas a otras (**I**)
 - Las modificaciones efectuadas por una transacción podrán recuperarse ante fallas del sistema (**D**)
- En el mundo relacional estamos familiarizados con las transacciones ACID, que garantizan la consistencia y estabilidad de las operaciones pero requieren lockings sofisticados

EJEMPLO TRANSACCIÓN ACID



CARACTERÍSTICA III - VISTAS E ÍNDICES MAPREDUCE

- En CouchDB, cada vista se construye por medio de una función JavaScript que actúa como **la mitad Map de una operación MapReduce**. La función recibe un documento y lo transforma en un valor, retornándolo (emit)
- CouchDB **indexa** las vistas y mantener actualizados esos índices a medida de que se agregan, eliminan o actualizan documentos
- Estas funciones no modifican los datos almacenados, solo la presentación al acceder a la vista

```
// Map
```

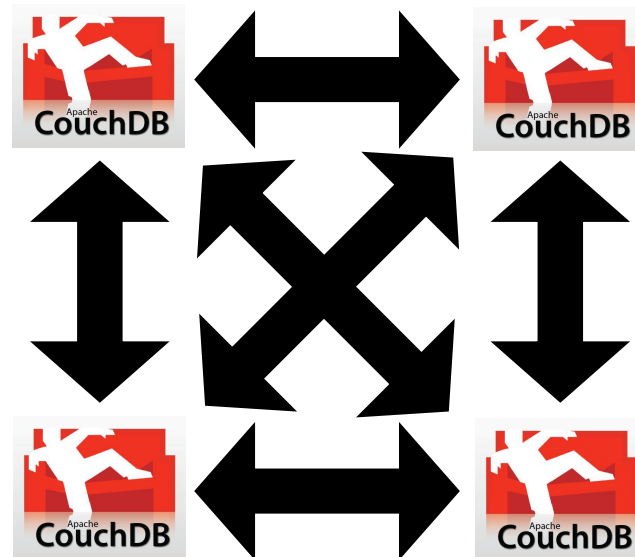
```
function(doc) {  
    emit(doc._id, 1);  
}
```

```
// Reduce
```

```
function(keys, values, rereduce) {  
    return sum(values);  
}
```

CARACTERÍSTICA IV - ARQUITECTURA DISTRIBUIDA CON REPLICACIÓN

- **Arquitectura distribuida con replicación:** CouchDB se diseñó teniendo en mente la replicación bidireccional (o sincronización) y la **operación offline**. Eso significa que múltiples réplicas pueden tener cada una sus propias copias de los mismos datos, modificarlas y luego sincronizar esos cambios en un momento posterior.



CARACTERÍSTICA V - INTERFAZ REST

- Todos los ítems tienen una URI única que queda expuesta vía HTTP.
- REST usa los métodos HTTP POST, GET, PUT y DELETE para las cuatro operaciones básicas CRUD (Create, Read, Update, Delete) con todos los recursos.
- Ver: <https://docs.couchdb.org/en/stable/api/basics.html#api-basics>

Create

POST <http://localhost:5984/employees>

Read

GET <http://localhost:5984/employees/1>

Update

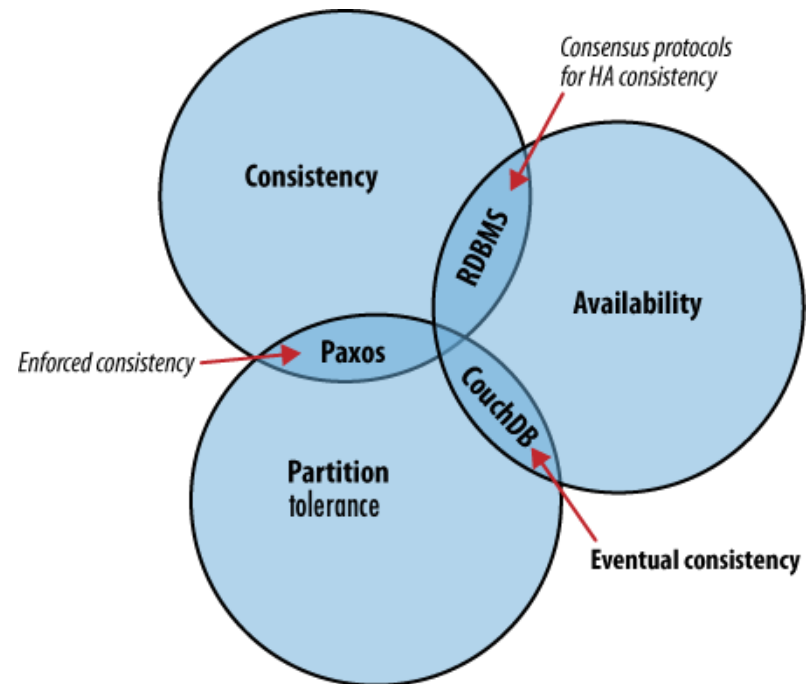
PUT <http://localhost:5984/employees/1>

Delete

DELETE <http://localhost:5984/employees/1>

CARACTERÍSTICA VI - CONSISTENCIA EVENTUAL

- **Consistencia Eventual:** CouchDB garantiza consistencia eventual para poder ofrecer tanto disponibilidad como tolerancia a las particiones
- Ver:
- <http://guide.couchdb.org/draft/consistency.html>
- <https://docs.couchdb.org/en/stable/intro/consistency.html>



CARACTERÍSTICA VII – OFFLINE MODE

- **Hecha para operar offline:** CouchDB puede replicar datos a dispositivos (como smartphones) que pueden quedar offline y manejar automáticamente la sincronización de los datos cuando el dispositivo vuelve a estar en línea.

CARACTERÍSTICA VIII - ULTRA DURABLE

- **Ultra durable:** CouchDB está diseñada de tal modo que el servidor no tiene un proceso de apagado. Esto es gracias a las propiedades ACID (recordar que en Mongo se escribe en cache y luego a disco)
- De hecho la única manera de apagar el servidor es matándolo
- CouchDB utiliza control de concurrencia multiversión, nunca sobrescribe los datos, siempre añade datos, esto reduce siempre los riesgos de corrupción de datos



ERLANG OTP


CouchDB is built on the Erlang OTP platform,
a functional, concurrent programming language and development platform.
Erlang was developed for real-time telecom applications with an extreme
emphasis on reliability and availability.



COMPARACIÓN CON MONGODB

<https://db-engines.com/en/system/CouchDB%3BMongoDB>

<https://blog.panoply.io/couchdb-vs-mongodb>

Name	CouchDB ⓘ X	MongoDB X
Description	A native JSON - document store inspired by Lotus Notes, scalable from globally distributed server-clusters down to mobile phones.	One of the most popular document stores available both as a fully managed cloud service and for deployment on self-managed infrastructure
Primary database model	Document store	Document store
Secondary database models		Search engine ⓘ
DB-Engines Ranking ⓘ  Trend Chart	Score 18.12 Rank #34 Overall #5 Document stores	Score 421.12 Rank #5 Overall #1 Document stores
Website	couchdb.apache.org	www.mongodb.com
Technical documentation	docs.couchdb.org/en/stable	docs.mongodb.com/manual
Developer	Apache Software Foundation ⓘ	MongoDB, Inc
Initial release	2005	2009
Current release	2.3.1, March 2019	4.2, August 2019
License ⓘ	Open Source ⓘ	Open Source ⓘ
Cloud-based only ⓘ	no	no ⓘ
DBaaS offerings (sponsored links) ⓘ		MongoDB Atlas : Deploy a fully managed cloud database on-demand and ready for use in minutes. Available on AWS, Azure, and GCP.
Implementation language	Erlang	C++

Vistas

VISTAS EN BBDD

- Una vista se puede considerar una tabla virtual o una consulta almacenada.
- Las vistas tienen la misma estructura que una tabla: filas y columnas. La única diferencia es que sólo se almacena de ellas la definición, no los datos (si la vista no está indexada), es decir sus datos no se almacenan en la base de datos como un objeto diferente. Lo que se almacena en la base de datos es una instrucción SELECT.
- El conjunto de resultados de la instrucción SELECT forma la tabla virtual que devuelve la vista.

```
CREATE VIEW myview AS
```

```
    SELECT city, temp_lo, temp_hi, prcp, date, location  
    FROM weather, cities  
    WHERE city = name;
```

```
SELECT * FROM myview;
```


VISTAS

- <https://docs.couchdb.org/en/stable/ddocs/views/index.html>
- **Filtrar** documentos de la bbdd para encontrar los relevantes
- **Extraer datos** de documentos y presentarlos en **orden**
- Construir **índices** para encontrar documentos según algún valor o estructura que tengan
- Utilizar estos índices para representar **relaciones** entre documentos
- Finalmente, con las vistas se puede hacer todo tipo de **cálculos** en los documentos.
 - Por ejemplo, si un documento representa las transacciones financieras, una vista puede responder a la pregunta de cuanto se ha gastado en la última semana, mes o año

MAPREDUCE

- Es un modelo de programación para dar soporte a la **computación paralela** sobre grandes colecciones (petabytes) de datos en clusters (o grupos de computadoras)
- No todo se puede hacer con MapReduce, “algoritmos susceptibles de ser paralelizados”
- El origen fue Hadoop
- Suele usarse un HDFS, sistema de archivos distribuido, aunque en nuestro caso son los datos de la BBDD
- Explicación con cartas:
<https://www.youtube.com/watch?v=bcjSe0xCHbE>
- Ejemplo del censo desde Roma.
 - Para contar a la gente se puede mandar a una persona ciudad por ciudad e ir contando
 - Pero será mejor mandar a una persona a cada ciudad y luego que vuelvan a Roma (eso sería el Map) y una vez en Roma se toman los datos de todas las personas y se “reducen”, se agregan (eso sería el Reduce)

MAPREDUCE

○ Workflow estándar

- Se tienen muchos datos como entrada
- **MAP**: La función map recorre los datos, extrae algo que nos interesa de cada elemento y construye una lista de pares clave/valor. La clave puede ser un tipo básico o un array, el valor puede ser cualquier cosa (tipo básico, hash o array)
- Reordenación de la lista por clave (en orden ascendente o descendente)
- **REDUCE**: La función reduce, reduce la lista a un solo valor

○ En CouchDB:

- Map recorre todos los documentos JSON de una colección y obtiene (usando emit()) datos que son organizados en un índice
- CouchDB automáticamente añade el campo `_id` al par clave/valor
- Las vistas (resultado del MapReduce) se guardan en disco a parte de la BBDD
- Una vez que se ha construido la vista, ésta se modifica cada vez que se añade o modifica un documento, esta es la principal diferencia con el MapReduce estándar de Hadoop

EJEMPLO DE MAP

JSON doc

```
function(doc) {  
  emit(doc.username, doc.email);  
}
```

Crear fila

Clave del índice
(tipo básico o array)

Valor/es (tipo básico,
hash o array)

_id	username	email
U1	Ebarra	ebarra@dit.upm.es
U2	Jlopez	jlopez@gmail.com
U3	Pedro	lodope@gmail.com
U4	Juanin	juanpe@fa.com

- CouchDB automáticamente añade el campo `_id` al par clave/valor
- La fase reduce es opcional

EJEMPLO DE MAP

- query que cuenta los docs cuyo “first_name” es “Enrique”
- Función MAP:

```
function(doc) {  
  if (doc.first_name == "Enrique") {  
    emit(doc._id, 1);  
  }  
}
```

- Función Reduce:

```
function(keys, values, rereduce) {  
  return sum(values)  
}
```

REDUCE

- CouchDB viene con algunas funciones Reduce estándar y muy útiles: `_count`, `_sum`, y `_stats`
- https://wiki.apache.org/couchdb/Built-In_Reduce_Functions
- Pero puedo usar cualquier cosa de JavaScript, por ej el equivalente a `_sum` sería:

```
function(keys, values, rereduce) {  
    return sum(values);  
}
```

MAPREDUCE

```
// Map
function(doc) {
  if (doc.dependents) {
    for (i in doc.dependents) {
      emit(doc._id, doc.dependents[i]);
    }
  }
}
```

```
// Reduce
_count
```

DESIGN DOCUMENTS

- Las vistas se almacenan en unos documentos especiales que se llaman “design documents”
- <https://docs.couchdb.org/en/stable/ddocs/index.html>

```
{
  "_id": "_design/stats",
  "views": {
    "total_employees": {
      "map": "function(doc) { emit(doc._id, 1); }",
      "reduce": "function(keys, values, rereduce) { return sum(values); }"
    },
    "by_lastname": {
      "map": "function(doc) { emit(doc.last_name, null); }"
    },
    "dependents": {
      "map": "function(doc) { if (doc.dependents) { for (i in doc.dependents) { emit(doc._id, doc.dependents[i]); } } }",
      "reduce": "_count"
    }
  }
}
```



```
{
  "_id": "1",
  "first_name": "Robert",
  "last_name": "Johnson",
  "date_hired": "2010/01/10",
  "dependents": [
    { "first_name": "Margie",
      "last_name": "Johnson" },
    { "first_name": "Charlie",
      "last_name": "Johnson" },
    { "first_name": "Sophie",
      "last_name": "Johnson" }
  ],
  "salary": 250000
}
```

```
{
  "_id": "2",
  "first_name": "Jim",
  "last_name": "Jones",
  "date_hired": "2010/02/11",
  "salary": 150000
}
```

```
{
  "_id": "3",
  "first_name": "Sally",
  "last_name": "Stevenson",
  "date_hired": "2010/04/23",
  "salary": 100000
}
```

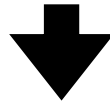
```
{
  "_id": "4",
  "first_name": "Bob",
  "last_name": "Smith",
  "salary": 80000,
  "date_hired": "2010/03/11",
  "dependents": [
    { "first_name": "Susan",
      "last_name": "Smith" }
  ]
}
```



MapReduce:

Función Map “dependents”:

```
function (doc) {
  if (doc.dependents) {
    for (i in doc.dependents) {
      emit(doc._id, doc.dependents[i]);
    }
  }
}
```



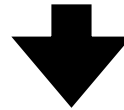
```
{"total_rows":4,"offset":0,"rows":[
  {"_id":"1","key":"1","value":{"first_name":"Margie","last_name":"Johnson"}},
  {"_id":"2","key":"1","value":{"first_name":"Charlie","last_name":"Johnson"}},
  {"_id":"3","key":"1","value":{"first_name":"Sophie","last_name":"Johnson"}},
  {"_id":"4","key":"4","value":{"first_name":"Susan","last_name":"Smith"}}
]}
```

```
{
  "_id": "1",
  "first_name": "Robert",
  "last_name": "Johnson",
  "date_hired": "2010/01/10",
  "dependents": [
    { "first_name": "Margie",
      "last_name": "Johnson" },
    { "first_name": "Charlie",
      "last_name": "Johnson" },
    { "first_name": "Sophie",
      "last_name": "Johnson" }
  ],
  "salary": 250000
}
```

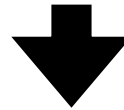
```
{
  "_id": "2",
  "first_name": "Jim",
  "last_name": "Jones",
  "date_hired": "2010/02/11",
  "salary": 150000
}
```

```
{
  "_id": "3",
  "first_name": "Sally",
  "last_name": "Stevenson",
  "date_hired": "2010/04/23",
  "salary": 100000
}
```

```
{
  "_id": "4",
  "first_name": "Bob",
  "last_name": "Smith",
  "salary": 80000,
  "date_hired": "2010/03/11",
  "dependents": [
    { "first_name": "Susan",
      "last_name": "Smith" }
  ]
}
```



MapReduce: función reduce (_count)



```
{"rows":[
  {"key":"1","value":3},
  {"key":"4","value":1}
]}
```

MAPREDUCE CONSIDERACIONES FINALES

- Lo que hemos visto son microejemplos
- MapReduce y las vistas de CouchDB pueden hacer queries mucho más complejas y avanzadas
- <https://docs.couchdb.org/en/stable/ddocs/views/intro.html>
- <https://docs.couchdb.org/en/stable/ddocs/views/collation.html>
- <https://docs.couchdb.org/en/stable/ddocs/views/joins.html>
- <https://docs.couchdb.org/en/stable/ddocs/views/nosql.html>
- <https://docs.couchdb.org/en/stable/ddocs/views/pagination.html>

```

function(keys, values, rereduce) {
  function sum(type_counts, totals, status) {
    if (type_counts[status]) { // OK or ERR
      if (!totals[status]) {
        totals[status] = new Object();
      }
      var status_totals = totals[status];
      var status_type_counts = type_counts[status];
      for (key in status_type_counts) { // MO, MT, CM, etc.
        var count = status_type_counts[key];
        if (!status_totals[key]) {
          status_totals[key] = count;
        } else {
          status_totals[key] += count;
        }
      }
    }
  }
}

var totals = new Object();
// values should be something like
// {"OK":{"MO":1234,"MT":1000,"CM":20},"ERR":{"MO":1,"MT": 1}}
for (i = 0; i < values.length; i++) {
  var message_count = values[i];
  sum(message_count, totals, 'OK');
}
return totals;
}

```

Replicación

REPLICACIÓN

- Simplemente hay que avisarle bbdd origen y destino
- La replicación gestiona los errores de modo transparente
- <https://docs.couchdb.org/en/stable/replication/index.html>

```
POST /_replicate {"source":"database", "target":"http://example.org/db"}
```

UNIDIRECCIONAL

POST /_replicate HTTP/1.1

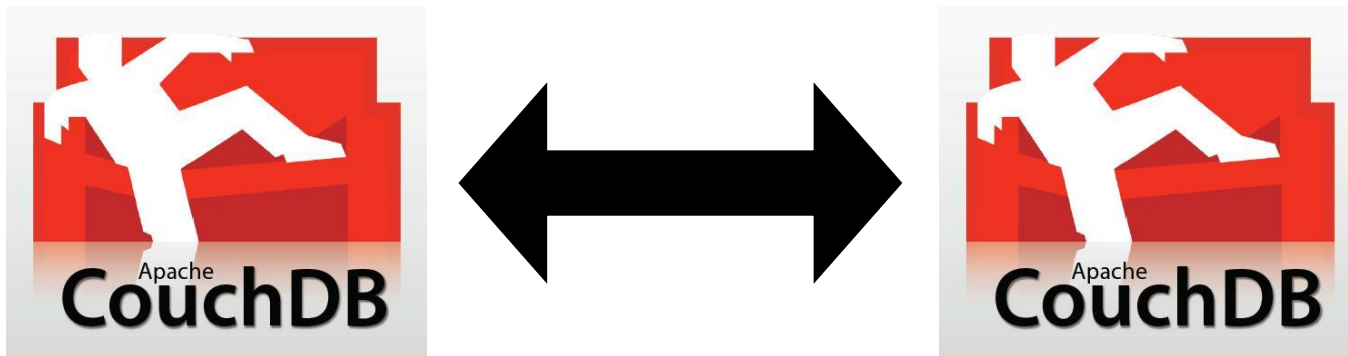
```
{"source":"database","target":"http://example.org/database"} -H "Content-Type: application/json"
```



BIDIRECCIONAL

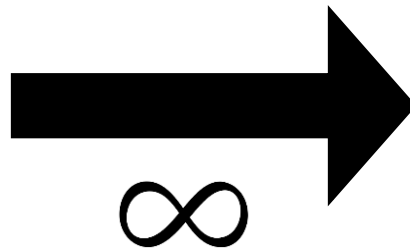
- Habrá que hacer el POST en el sentido contrario y ya tenemos bidireccional

```
POST /_replicate HTTP/1.1  
Content-Type: application/json  
{  
  "source": "http://example.org/database",  
  "target": "database"  
}
```



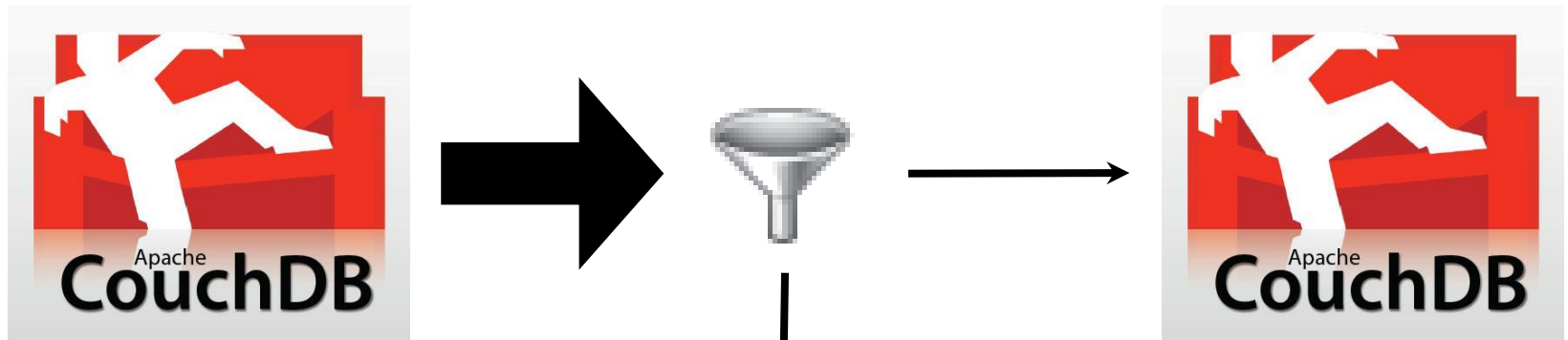
CONTINUO

```
curl -X POST http://127.0.0.1:5984/_replicate -d  
'{"source":"db", "target":"db-replica", "continuous":true}' -H "Content-Type: application/json"
```



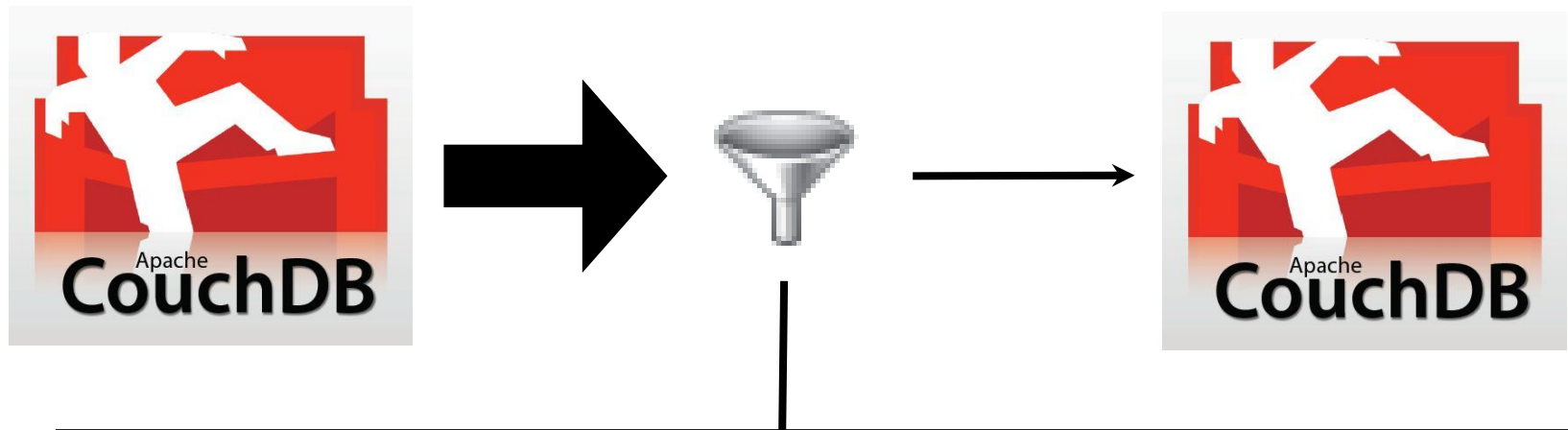
FILTRADO

<https://docs.couchdb.org/en/stable/ddocs/ddocs.html#filterfun>



```
function(doc, req) {  
  if (doc.type && doc.type == "foo") {  
    return true;  
  } else {  
    return false;  
  }  
}
```

FILTRADO



```
function(doc, req) {  
  if (doc.type == req.query.doc_type) {  
    return true;  
  } else {  
    return false;  
  }  
}
```

CONFLICTOS



- CouchDB está diseñada para operar offline
- Si dos bases de datos continúan operando offline mientras están desconectadas puede haber conflictos al volver online
- **Un conflicto ocurre** cuando un mismo documento se actualiza en ambas bases de datos
- Durante la replicación CouchDB detectará que hay múltiples versiones del mismo documentos y registrará un conflicto
- El ganador lo elige CouchDB porque necesita un documento final que guardar en ambas réplicas
- La versión que pierde también guarda en una versión previa del documento que está disponible para poder “mergearla” y la identifica con una propiedad `_conflicts`, CouchDB no intenta hacer merge
- <https://docs.couchdb.org/en/stable/replication/conflicts.html>



ENCONTRANDO CONFLICTOS

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

```
{ "total_rows": 1, "offset": 0, "rows": [  
  
  { "id": "foo", "key": ["2-7c971bb974251ae8541b8fe045964219"], "value": null }  
  
]}
```

RESOLVIENDO CONFLICTOS

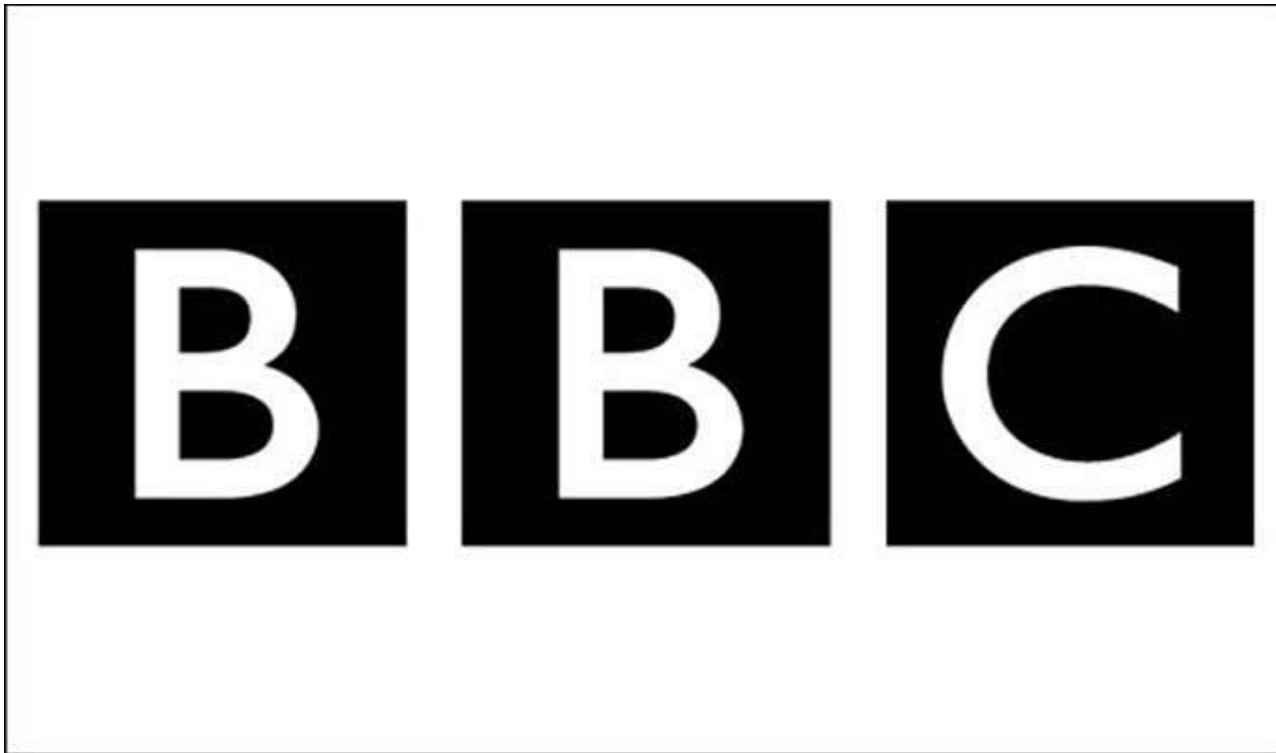
Step 1

```
PUT /db/document {... merged data ...}
```

Step 2

```
DELETE /db/document?rev=2-7c971bb974251ae8541b8fe045964219
```

EJEMPLO REAL



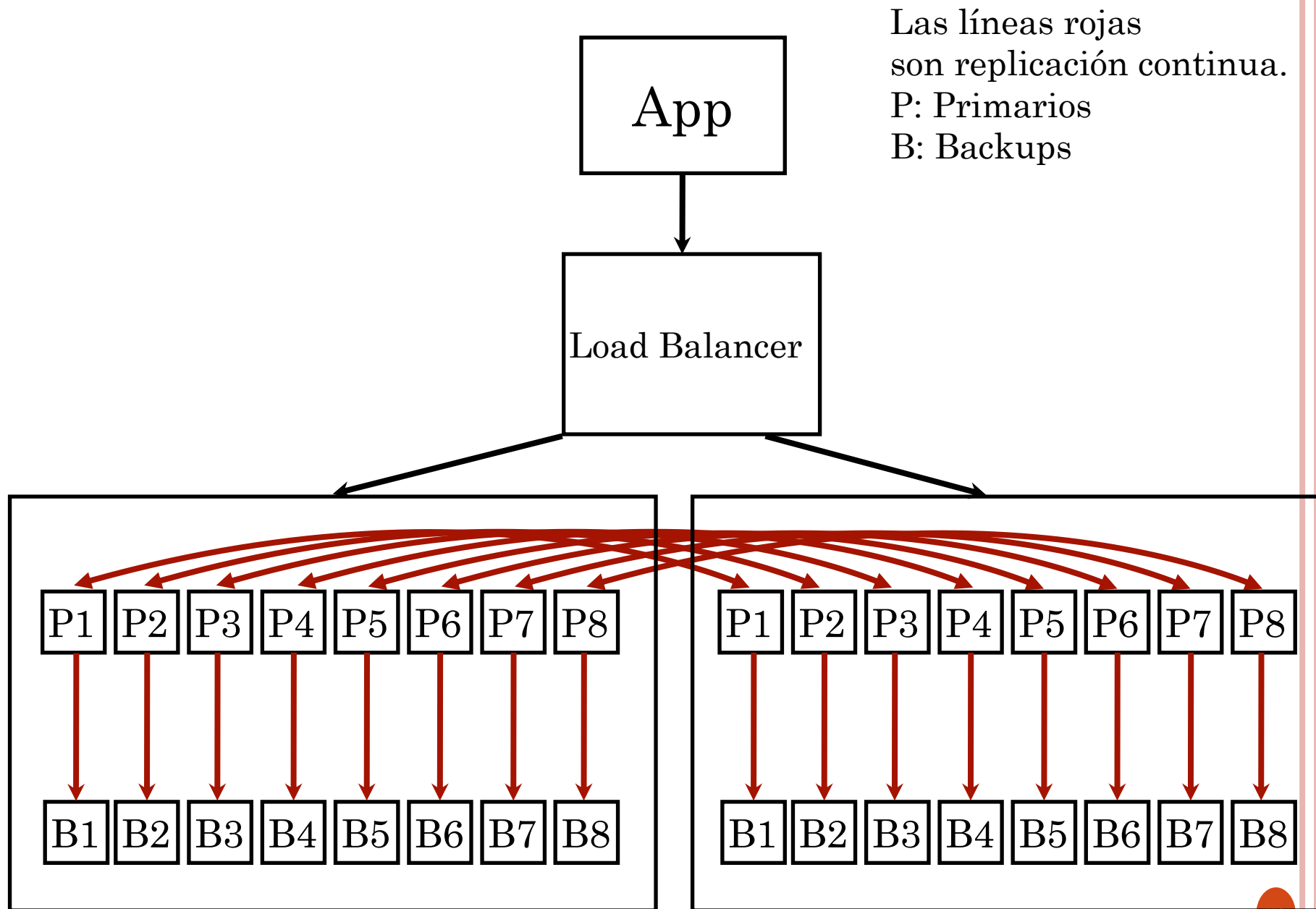
<http://www.couchbase.com/case-studies/bbc>

EL PROBLEMA

- Necesitaban asegurarse de que el sitio estaba siempre arriba y disponible, incluso aunque hubiese una catástrofe en un data center
- Necesitaban una solución que pudiese replicar fácilmente entre dos o más data centers
- Necesitaban que la solución almacenase datos en modo seguro y confiable

LA SOLUCIÓN

- Utilizar CouchDB para crear un multi data center con configuración multi master y tolerancia a fallos
- 32 nodos en el cluster
 - 16 nodos in cada uno de los dos data centers
 - 8 nodos primarios, 8 nodos de backup
- Terabytes de datos
- 150-170 millones de requests por día



Change Notifications

CHANGE NOTIFICATIONS

- Para crear servicios de mensajes o apps que necesiten notificaciones de cambio
- CouchDB crea una bbdd `_changes` a la que se pueden pedir los cambios
- `GET /db/_changes`
- `GET /db/_changes?since=1`
- `GET /db/_changes?since=1&include_docs=true`
- `GET /db/_changes?feed=longpoll&since=2`
- `GET /db/_changes?feed=continuous`
- `GET /db/_changes?filter=filters/signal_employees`

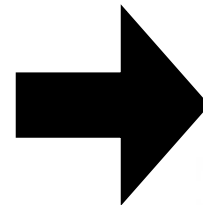
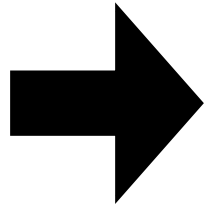
```
{"seq":12,"id":"foo","changes":[{"rev":"1-23202479633c2b380f79507a776743d5"}]}
```

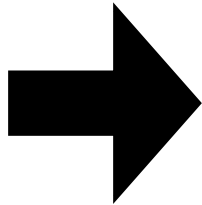
- <http://guide.couchdb.org/draft/notifications.html>
- <https://docs.couchdb.org/en/stable/api/database/changes.html>

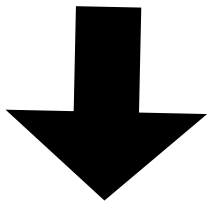
Soporte a móviles

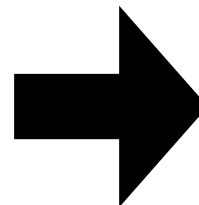
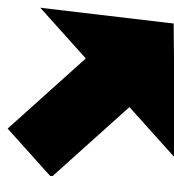
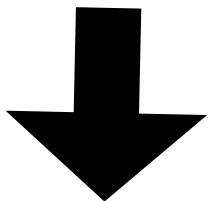
- CouchDB funciona de forma nativa en iOS y Android







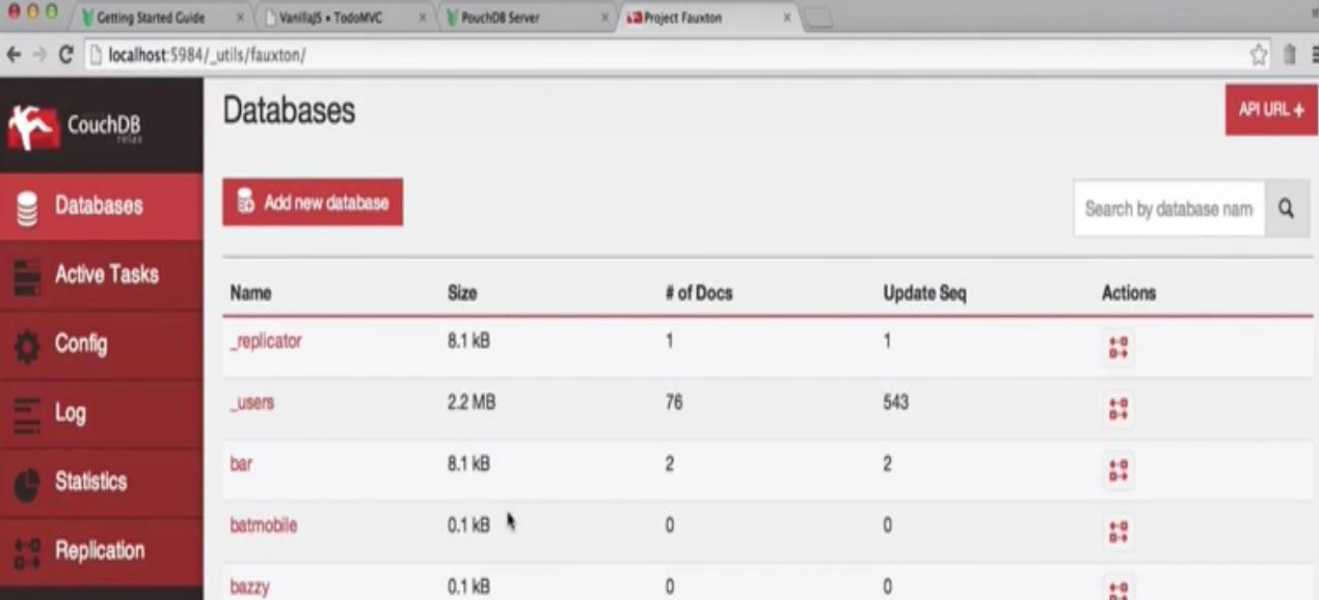














Recursos

INSTALAR COUCHDB

```
MacBook-Air-de-Jessy:~ jessyaviles$ brew install couchdb
```

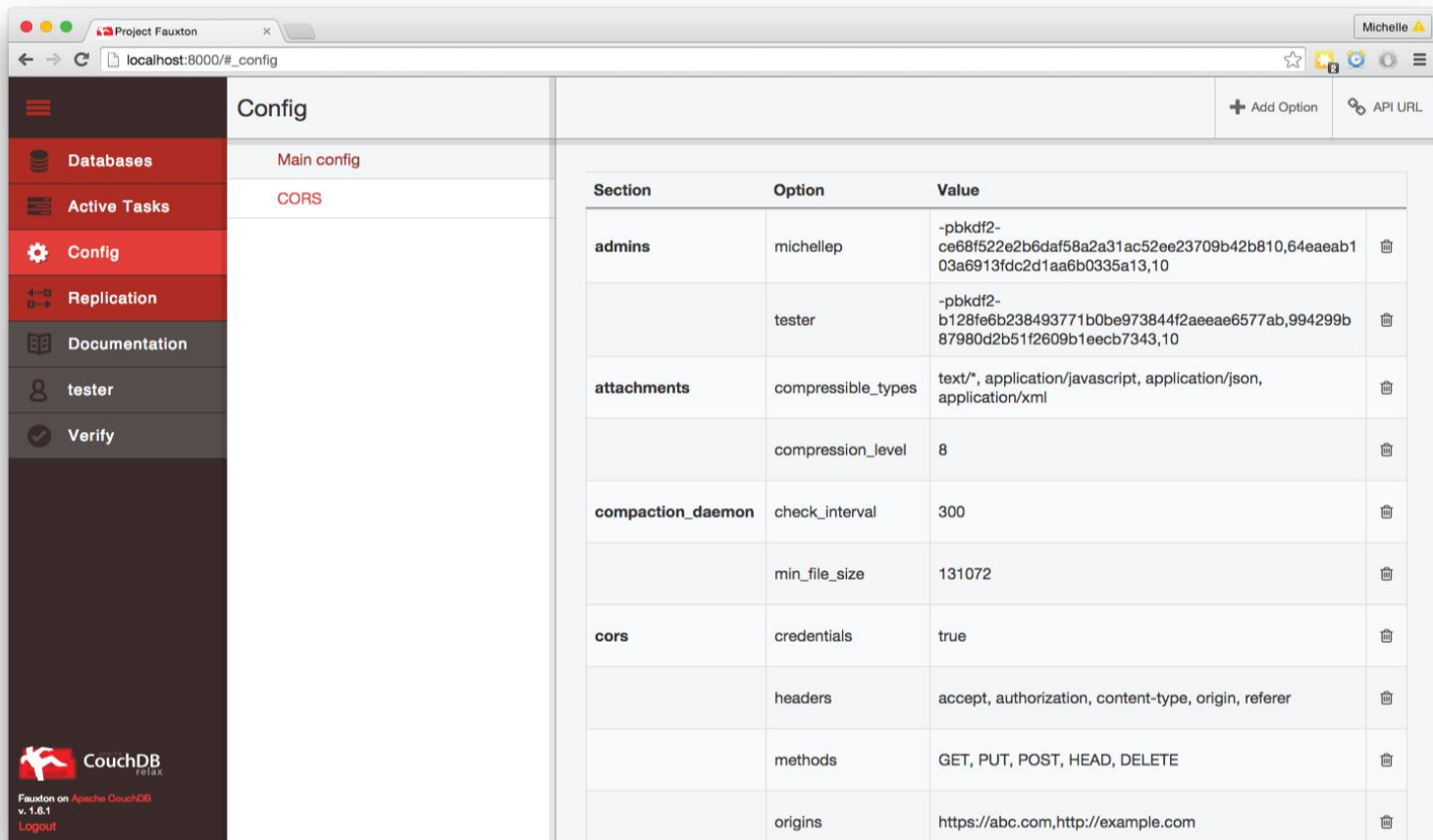


The screenshot shows the CouchDB web interface in a browser. The address bar indicates the URL is `localhost:5984/_utils/fauxton/`. The interface has a dark red sidebar on the left with navigation links: Databases, Active Tasks, Config, Log, Statistics, and Replication. The main content area is titled "Databases" and features a red "Add new database" button and a search bar labeled "Search by database name". Below these is a table listing the databases.

Name	Size	# of Docs	Update Seq	Actions
<code>_replicator</code>	8.1 kB	1	1	 
<code>_users</code>	2.2 MB	76	543	 
<code>bar</code>	8.1 kB	2	2	 
<code>batmobile</code>	0.1 kB	0	0	 
<code>bazzy</code>	0.1 kB	0	0	 

HABILITAR CORS

```
$ npm install -g add-cors-to-couchdb
$ add-cors-to-couchdb
```



The screenshot shows the Project Fauxton web interface in a browser window. The address bar indicates the URL is `localhost:8000/#_config`. The left sidebar contains navigation links: Databases, Active Tasks, Config (selected), Replication, Documentation, tester, and Verify. The main content area is titled 'Config' and shows the 'Main config' section with 'CORS' highlighted. A table displays the configuration options for various sections.

Section	Option	Value
admins	michellep	-pbkdf2-ce68f522e2b6daf58a2a31ac52ee23709b42b810,64eaeab103a6913fdc2d1aa6b0335a13,10
	tester	-pbkdf2-b128fe6b238493771b0be973844f2aeae6577ab,994299b87980d2b51f2609b1eeeb7343,10
attachments	compressible_types	text/*, application/javascript, application/json, application/xml
	compression_level	8
compaction_daemon	check_interval	300
	min_file_size	131072
cors	credentials	true
	headers	accept, authorization, content-type, origin, referer
	methods	GET, PUT, POST, HEAD, DELETE
	origins	https://abc.com,http://example.com

RESOURCES

- Presentación basada en “Real World CouchDB by John Wood”
<https://vimeo.com/26383445>
- CouchDB Project Website
<http://couchdb.apache.org>
- CouchDB: The Definitive Guide (deprecated)
<http://guide.couchdb.org>
- CouchDB Project Wiki
<https://cwiki.apache.org/confluence/display/couchdb>
- CouchApps (deprecated)
<http://couchapp.org>

COUCHDB Y COUCHBASE

- Son similares aunque tienen diferencias, no confundirlos
- <https://www.couchbase.com/couchbase-vs-couchdb>
- Comparativa de features entre MongoDB, CouchDB y CouchBase
- <http://db-engines.com/en/system/CouchDB%3BCouchbase%3BMongoDB>
- “Couchbase Server takes all the chewy NoSQL goodness of CouchDB, and gives it the crisp hard edge of a memcache frosting”

