



COUCHBASE

María Romero Angulo

Proyecto final del Ciclo de Formación Superior de Administración de Sistemas
Informáticos en Red

2ºASIR

IES Gonzalo Nazareno

ÍNDICE

1. INTRODUCCIÓN.....	4
2. BASES DE DATOS RELACIONALES vs NO RELACIONALES.....	6
2.1 VENTAJAS Y DESVENTAJAS DE BASES DE DATOS RELACIONALES.....	6
2.2 VENTAJAS Y DESVENTAJAS DE BASE DE DATOS NO RELACIONALES.....	7
3. COUCHBASE Y SU ESTRUCTURA.....	9
3.1 ESTRUCTURA.....	10
3.1.1 SERVICIOS BÁSICOS.....	12
3.1.2 OTROS SERVICIOS.....	16
4. PREPARACIÓN DEL ENTORNO.....	21
4.1 DNS.....	21
5. INSTALACIÓN.....	25
6. CONFIGURACIÓN BÁSICA.....	27
7. N1QL.....	31
7.1 EXPRESIONES.....	31
7.2 EJEMPLOS.....	36
8. RÉPLICAS.....	43
8.1 REPLICACIÓN XDCR.....	47
9. COPIAS DE SEGURIDAD.....	50
9.1 CCBACKUP Y CBRESTORE.....	50
9.2 CBBACKUP.....	52
9.2.1 EJEMPLOS.....	54
9.3 CBRESTORE.....	55
9.3.1 EJEMPLOS.....	57

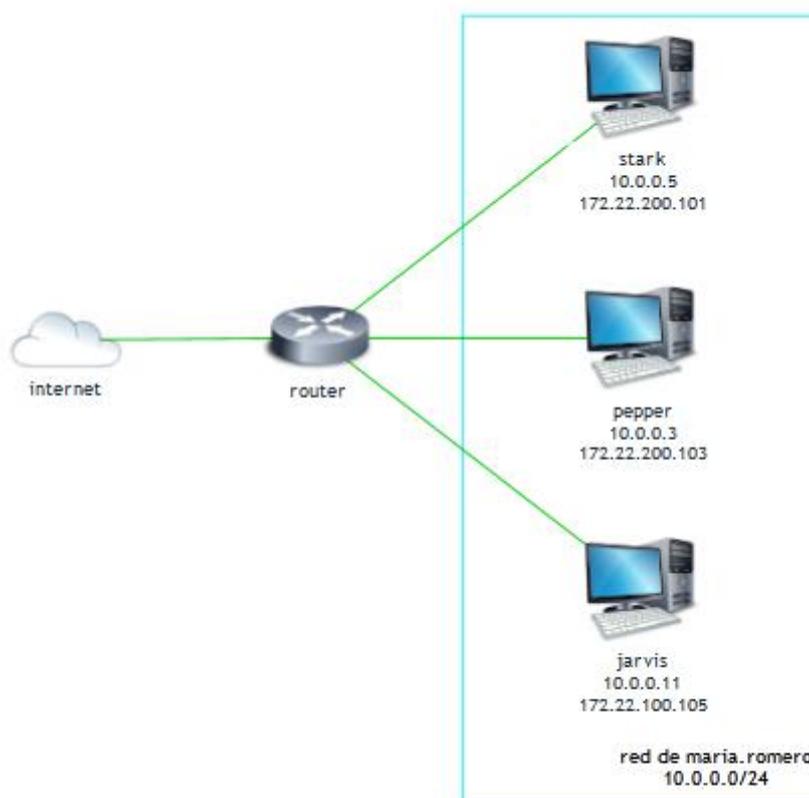
10. IMPORTACIÓN Y EXPORTACIÓN.....	59
10.1 EXPORTACIÓN.....	59
10.1.1 EJEMPLOS.....	61
10.2 IMPORTACIÓN.....	62
10.2.1 EJEMPLOS.....	63
11. MÉTRICA.....	65
11.1 INSTALACIÓN DEL SERVIDOR (JARVIS).....	65
11.2 INSTALACIÓN EN LOS CLIENTES (STARK Y PEPPER).....	68
12. APLICACIÓN.....	71
13. CONCLUSIONES.....	72
14. BIBLIOGRAFÍA.....	73
15. POSIBLES MEJORAS.....	74

1. INTRODUCCIÓN

En este documento se va explicar el funcionamiento de las bases de datos no relacionales en general y luego aplicado a Couchbase, las diferencias con otras de su tipo y con las relacionales, ventajas y desventajas. Una vez comprendidos estos conceptos, se pasará a la instalación y configuración de un entorno en alta disponibilidad y a la creación de una aplicación de gestión gracias a al API que no proporciona.

También se explicará el lenguaje propio de consultas llamado N1QL.

El escenario constará de 3 máquinas siguiendo el siguiente esquema



- stark
 - ◆ Es el servidor principal de la base de datos

➤ **pepper**

- ◆ Es la réplica del servidor principal que estará a la espera de fallo para evitar la caída del servicio.

➤ **jarvis**

- ◆ Tendrá los servicios complementarios como el servidor web para la aplicación, un servidor DNS y todo lo relacionado con la representación de métricas.

2. BASES DE DATOS RELACIONALES vs NO RELACIONALES

Actualmente nos encontramos en una situación en la que la tecnología tiene un papel muy importante en la vida cotidiana, tanto personal como laboral, por lo que ha habido muchas herramientas que han tenido que adaptarse, como es el caso de algo tan básico como las bases de datos.

Las “clásicas”, es decir, las relacionales, se han quedado atrás en cuanto a escalabilidad y rendimiento, características imprescindibles para muchas de las aplicaciones actuales y futuras. Para cumplir con esta exigencia, tenemos las bases de datos no relacionales, con mayor rapidez en ciertas tareas y muy fácilmente adaptables a la demanda de la aplicación en cualquier momento.

El concepto básico es el mismo en ambas: almacenar datos. Pero si vemos su funcionamiento interno, coinciden en muy pocas cosas. Por ejemplo: las bases de datos no relacionales tienen una estructura fija de tablas como la de las relacionales, es decir, las noSQL pueden ser basadas en documentos (MongoDB), pares clave-valor (cassandra), grafos (Neo4j), tabular (BigTable) u orientadas a objetos (ObjectDB).

2.1 VENTAJAS Y DESVENTAJAS DE BASES DE DATOS RELACIONALES

- Ventajas
 - ◆ Tienen herramientas que **evitan la duplicidad de registros**.

- ◆ **Integridad referencial:** al eliminar un registro, desaparecen también los que estén relacionados.
 - ◆ Facilita la **normalización** en el diseño al ser más comprensible.
 - ◆ **Mayor soporte comercial**
 - ◆ **Mucha madurez**
 - ◆ Hay bastantes desarrolladores con una **amplia experiencia** en su manejo.
 - ◆ Hay ciertos **estándares** que hacen que sean más sencillas tareas como migraciones.
- Desventajas:
- ◆ **Escalabilidad vertical**
 - ◆ **Escalamiento complejo**
 - ◆ No se pueden manipular de forma sencilla los bloques de texto como tipo de datos.
 - ◆ Su **rendimiento baja notablemente al manejar grandes cantidades de datos**, y recae sobre el administrador o el desarrollador afinar la instalación.
 - ◆ Requieren **equipos potentes** para hacer tareas que se hacen con bastante frecuencia.

2.2 VENTAJAS Y DESVENTAJAS DE BASE DE DATOS NO RELACIONALES

- Ventajas
- ◆ **Escalabilidad horizontal**
 - ◆ Pueden manejar **grandes cantidades de datos**

- ◆ No generan cuellos de botella, **gran rendimiento**
 - ◆ **Escalamiento sencillo**
 - ◆ **Diferentes tipos** para diferentes proyectos
 - ◆ **No hacen falta máquinas muy potentes**
- Desventajas:
- ◆ **Menos soporte comercial** al ser de código abierto. Hay que recurrir a la comunidad open source, que si bien nos podrá solucionar el problema, los tiempos de espera suelen ser mayores.
 - ◆ **Poca madurez:**
 - ◆ **Falta de experiencia**
 - ◆ **Falta de compatibilidad:** las bases de datos relacionales tiene ciertos estándares que en este caso no existen, es decir, cada una tiene su API, su interfaz de consultas y peculiaridades propias.
 - ◆ Puede haber duplicidad de registros.

3. COUCHBASE Y SU ESTRUCTURA

Couchbase es una base de datos no relacional orientada a documentos, distribuida, de alto rendimiento, muy escalable y de propósito general.

API

Una de las ventajas de couchbase es que permite comunicar dispositivos con diferentes sistemas y arquitecturas entre ellos gracias a su API, facilitando la tarea de hacer aplicaciones con diferentes lenguajes, conectores y herramientas.

IoT

Couchbase proporciona una plataforma completa para aplicaciones móviles y relacionadas con IoT con sincronización de datos en tiempo real, alta seguridad e integración de los datos ya existentes en el entorno de trabajo.

ESCALABILIDAD

Si necesitásemos más recursos, se pueden añadir de forma rápida y el propio servidor se encarga de repartir la carga entre todos los nodos de forma automática y uniforme, manteniendo el servicio activo y sin que los clientes se den cuenta.

RENDIMIENTO

Couchbase está pensado para procesar el pesado tráfico de lectura-escritura que requieren aplicaciones web, para móviles y las relacionadas con IoT sin problemas. Esto lo puede llevar a cabo con su sistema que detecta que documentos, metadatos e índices se acceden más veces y los almacena en la memoria RAM.

ALTA DISPONIBILIDAD

Todas las acciones de configuración como añadir o quitar nodos, construcción de índices, actualizaciones o cambios de hardware se pueden realizar si reiniciar el servicio o el mismo equipo gracias a la orientación hacia la alta disponibilidad y el manejo sencillo del servidor.

Con la tolerancia a fallos por defecto, el sistema está protegido frente a posibles caídas del servicio no planeadas, como errores en el propio servidor. Esto se consigue replicando los datos en varios nodos, asegurando copias de seguridad automáticas.

Se pueden hacer réplicas en otras localizaciones geográficas con XDCR (Cross Data Center Replication), aumentando la facilidad de recuperación en caso de fallo y el balanceo de carga.

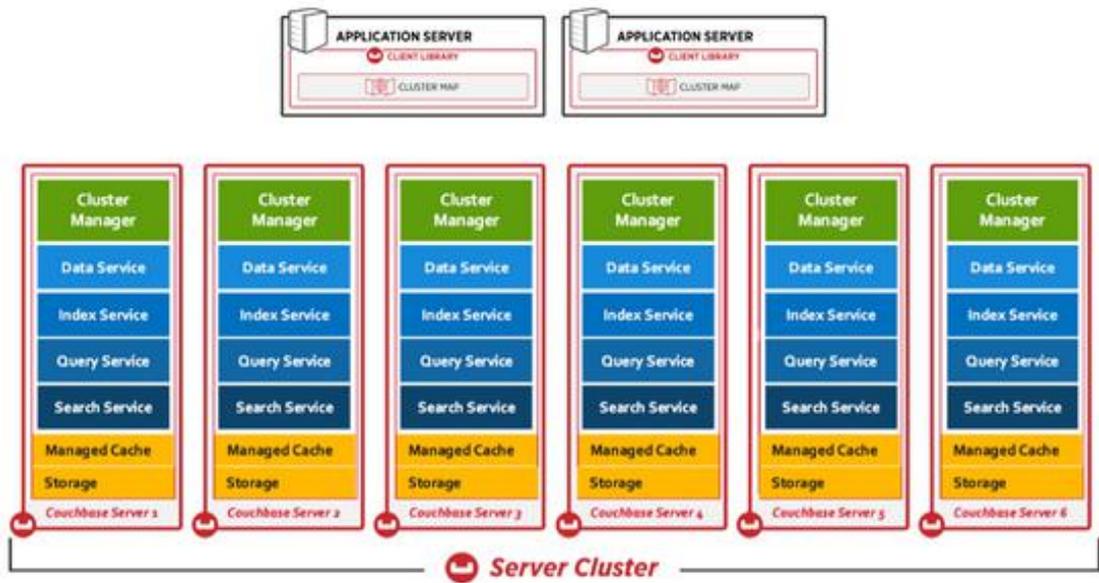
EJEMPLOS DE USO

- AOL: publicidad
- PayPal: estadísticas en tiempo real
- Tesco: almacena todos los datos de los productos
- Ryanair: aplicación web
- Liveperson: chat en tiempo real

3.1 ESTRUCTURA

Couchbase es un único paquete que se instala en todos los nodos y con los SDKs correspondientes se pueden hacer aplicaciones en el lenguaje que se quiera y luego se conectan al servidor para leer/escribir datos de forma rápida.

La estructura básica sigue este esquema:



Los componentes básicos (core) de un servidor de couchbase son:

- Conexiones (connectivity architecture)
- Réplicas (replication architecture)
- Almacenamiento (storage architecture)
- Cache (caching layer architecture)
- Seguridad (security architecture)

Los demás componentes son:

- Cluster (cluster manager)
- Datos (data service)
- Índices (index service)
- Consultas (query service)
- Búsquedas (search service)

3.1.1 SERVICIOS BÁSICOS

CONEXIONES (CONNECTIVITY ARCHITECTURE)

Dentro del servidor, hay varios tipos de conexiones (cliente-cluster, nodo-nodo, cluster-cluster, cluster-equipos externos).

Los pasos para establecer un canal de comunicación son:

1. Autenticación: se comprueban las credenciales del cliente para acceder al bucket que se indique. En caso de acceder como administrador, ve el cluster completo.
2. Discovery: la conexión recibe un mapa de cluster que representa su topología, con nodos, como se distribuyen los datos en ellos y qué servicios están activos en cada uno.
3. Conexión: el cliente determina qué conexiones de todas las anteriores necesita y accede. Las conexiones a los componentes no básicos requieren una segunda autenticación para poder realizar varias operaciones.

RÉPLICAS (REPLICATION ARCHITECTURE)

Está desarrollado en el apartado de réplicas (página [43](#))

ALMACENAMIENTO (STORAGE ARCHITECTURE)

Los distintos componentes de couchbase requieren diferentes tipos de almacenamiento adaptados a su carga de trabajo, por lo que tiene dos mecanismos: couchstore y forestDB.

Couchbase permite controlar el almacenamiento de datos y de índices, pudiendo elegir distintos directorios para ambos, de forma que cada uno puede tener un método de los anteriores y así mejorar el rendimiento.

Los métodos son:

Data Service, MapReduce Views, Spatial Views, y Couchstore

Cada bucket se guarda como un fichero distinto en el sistema. Usa una estructura B+tree para acceder de forma rápida a los elementos a través de sus claves. Para mejorar la eficiencia y la consistencia de los datos, se usa un modelo de “append-only”.

Index Service, Search Service, y ForestDB

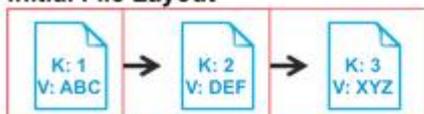
Con forestDB, cada índice se representa como un fichero distinto en el sistema y usa una estructura B+trie, que proporciona mayor eficiencia que B+tree y está indicado para grandes cantidades de datos (índices). Se puede configurar para “append-only” o para “circular reuse”.

“Append-only”

Con cada cambio, los nuevos datos se añaden nuevas páginas al final del fichero que invalidan los enlaces a las anteriores. Es necesario un proceso de limpieza de los nodos huérfanos y una especial atención a la fragmentación del disco.

En la imagen vemos primero el estado inicial, luego cómo quedan esos enlaces invalidados y finalmente cómo queda el fichero tras la limpieza.

Initial File Layout



Update Some Data



After Compaction



“Circular reuse”

Cuando se realiza un cambio, en vez de añadir al final del fichero, busca los nodos huérfanos y los sobrescribe. Si no hubiera suficientes, lo añade al final como en el método anterior. Es necesario un proceso de control para que no haya errores en el proceso de continua desfragmentación, que en este caso se ejecuta en segundo plano sin penalización del rendimiento.

CACHE (CACHING LAYER ARCHITECTURE)

Al ser una arquitectura en la que prima la memoria, para alcanzar unos altos niveles de escalabilidad y rendimiento es necesario un manejo muy efectivo de dicho componente.

Capa de cache

Cada servicio del servidor tiene su cache definida según sus necesidades. Por ejemplo, el servicio de datos tiene una cache configurada para tener operaciones de lectura/Escritura basadas en claves con baja latencia y alta concurrencia, pero el servicio de índices está preparado para mantener la consistencia y el escaneo rápido en las consultas.

El sdk de couchbase nunca se comunica directamente con los datos persistentes, siempre recurre a la cache, que es la que busca los datos.

Cuando se realiza una consulta, la propia cache es capaz de determinar si ese dato es accedido con mucha frecuencia, y en caso de que no lo sea, da la orden de moverlo de la RAM, al disco, dando prioridad a los datos más utilizados en RAM.

Cuotas en RAM

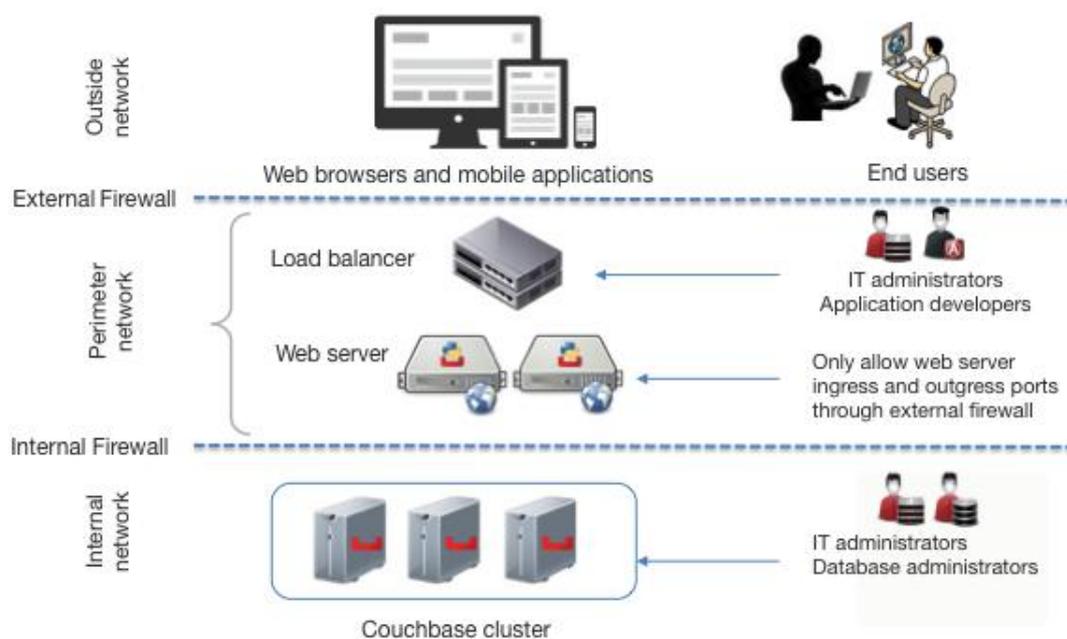
Cada servicio tiene un máximo de RAM asignado según sus necesidades. El servicio de indexado es el que mayor cuota posee, seguido de las búsquedas y los datos.

SEGURIDAD (SECURITY ARCHITECTURE)

Dado que van a entrar grandes volúmenes de datos a altas velocidades es importante mantener seguro nuestro ecosistema.

Couchbase proporciona diversas herramientas a los administradores para implementar controles de seguridad de varios tipos desde el acceso a la pila completa a las aplicaciones que dependen de couchbase pasando por la protección física de la infraestructura de red (red, dispositivos de almacenamiento, máquinas físicas y virtuales con sus sistemas operativos, ...)

También tiene en cuenta que hay aplicaciones que por su naturaleza (orientadas a empresas) necesitan unos niveles extra de seguridad. En este sentido, couchbase recomienda seguir un esquema de este tipo:



Con esto, las peticiones externas pasan primero por un firewall externo y se permite el balanceo de carga añadiendo también filtros y bloqueos de ciertos sitios y direcciones. Una vez se haya procesado todo esto, pasa por un firewall interno en el que solo se permiten conexiones desde y hacia los puertos de couchbase.

3.1.2 OTROS SERVICIOS

CLUSTER (CLUSTER MANAGER)

Se ejecuta en todos los nodos del cluster y permite el manejo de diversas operaciones a nivel de cluster como:

- Tipo de cluster y nodos miembros
- Balanceo de carga
- Salud de los nodos
- Almacenamiento
- Autenticación

La arquitectura del administrador de cluster es la siguiente:



Tiene los siguientes módulos:

- API y Auth: autenticación y ejecución de los comandos realizados a través de CLI o el portal de administración.
- Servicios maestros: localización de réplicas, autorrecuperación de fallos y rebalanceo de carga.
- Servicios de bucket: operaciones a nivel de bucket.
- Servicios de nodos: salud y monitorización de nodos.
- Elementos genéricos

Detección de fallos en nodos

Cada nodo se comunica con los demás a través de “latidos” de forma que cuando no hay respuesta, se pone en marcha el mecanismo de recuperación.

Un nodo es elegido por los demás como principal y es el responsable de hacer el seguimiento de todos los nodos y los servicios de datos y administración de cluster.

Administración, estadísticas y logging centralizados

Las estadísticas de uso son accesibles a través de todas las herramientas administrativas (cbsats, API, consola web) y recolectan datos de rendimiento de nodos y buckets individuales mostrándolas en gráficos en tiempo real agrupado por categorías.

DATOS (DATA SERVICE)

Couchbase almacena los datos como items, formados por una clave (ID) y un documento, junto con los metadatos asociados. Los datos se agrupan en buckets.

Proporciona métodos GET y SET simples y eficientes para realizar cambios y ver las claves y métodos para mejorar las consultas como filtros, agrupaciones y agregaciones de datos.

El acceso a datos está permitido desde varias aplicaciones y métodos de forma simultánea debido a su mínima latencia y a la eficiencia de la cache.

Para tener una buena rapidez en las consultas, los datos más accedidos se guardan en la memoria RAM y todos los demás en el disco, pudiendo pasar de uno a otro según la frecuencia de acceso y las necesidades de las aplicaciones.

Para mantener la consistencia de los datos, siempre se accede al bucket principal y en ciertos casos muy puntuales a los demás, pero esto se intenta evitar por todos los medios para que solo haya modificaciones simultáneas en un solo nodo.

Los datos almacenados pueden estar en formato JSON o binario, incluyendo cadenas estructuradas y no estructuradas, objetos con números de serie, datos binarios como audio e imágenes,... Puede usarse solo un formato o ambos, pero los datos almacenados en JSON serán accesibles de forma más sencilla.

También podemos crear documentos con una fecha de caducidad, de forma que llegado el día, automáticamente esos datos se borran. Se suele utilizar para sesiones. El valor por defecto es infinito.

Hay que tener en cuenta que para realizar consultas con N1QL hay que tener el servicio de consultas (Query service) activado.

ÍNDICES (INDEX SERVICE)

Se pueden consultar índices y vistas al igual que los buckets a través de:

- Vistas de MapReduce a través de la API.

- Vistas de Spatial a través de la API.
- Consultas N1QL con GSI y vistas de MapReduce.

Tipos de índices:

- MapReduce: es una forma de indexado incremental, de forma que se ahorra tiempo en consultas que devuelvan muchos valores.
- Spatial: proporciona procesamiento de información geográfica y permite consultas a servidores remotos.
- GSI: es muy similar a los índices convencionales. Se utiliza para realizar consultas N1QL de forma rápida. Depende del servicio de indexado pero no del de datos.

CONSULTAS (QUERY SERVICE)

Se pueden realizar consultas de varias formas: claves de documentos, vistas, índices o N1QL.

Claves de documentos

Se realizan a través de la API, que en base a la clave que se le pase, devuelve un documento. Solo es válido si se conocen las claves aunque es la más rápida.

Vistas

Son útiles para ver datos de forma interactiva y cuando se trabajan con grandes cantidades de datos complejos.

Spatial (índices)

Muestran la información geográfica de los documentos remotos. Es utilizado en aplicaciones que requieren localizaciones como las relacionadas con mapas.

N1QL

Es un lenguaje de consultas propio, muy similar al SQL de las base de datos relacionales como Oracle, MySQL o PostgreSQL. Es la más sencilla de todas, pero no la más rápida como desventaja.

4. PREPARACIÓN DEL ENTORNO

Servicios como DNS, o el servidor web de la aplicación que se realizará más adelante van a estar en otro equipo, evitando así caídas de los servicios en caso de que o bien el servidor principal o la réplica fallen. De esta forma también se evita sobrecargar el servidor principal.

4.1 DNS

Lo primero es instalar el paquete bind9 y dnsutils para hacer comprobaciones.

Éste último lo voy a instalar en todos los equipos, tanto en los 3 del escenario como en el mío propio desde el que me conectaré al panel web y a la aplicación.

```
apt install bind9 dnsutils
```

Con el paquete ya instalado, pasamos a la configuración como tal.

En el fichero /etc/bind/named.conf.local añadimos las zonas que vayamos a definir:

```
zone "maria.org"{  
type master;  
file "db.maria.org";  
};  
zone "200.22.172.in-addr.arpa"{  
type master;  
file "db.maria.org.inv";  
};
```

En el fichero `/etc/bind/zones.rfc1981` comentamos las líneas correspondientes a las redes que vayamos a definir (172.22.200.0/24)

```
#zone "22.172.in-addr.arpa" { type master; file
"/etc/bind/db.empty"; };
```

Ahora creamos los ficheros de las zonas:

```
/var/cache/bind/db.maria.org
$ORIGIN maria.org.
$TTL 86400 ; 1 day
@      IN      SOA      jarvis.maria.org.
admin.maria.org. (
        1 ; serial
        21600 ; refresh (6 hours)
        3600 ; retry (1 hour)
        604800 ; expire (1 week)
        21600 ; minimum (6 hours)
)
@      IN      NS       jarvis
@      IN      MX       10 correo
jarvis IN      A        172.22.200.105
pepper IN      A        172.22.200.103
stark  IN      A        172.22.200.101
couchweb IN     CNAME    jarvis
```

```
/var/cache/bind/db.maria.org.inv
```

```
$ORIGIN 200.22.172.in-addr.arpa.
$TTL 86400 ; 1 day
```

```
@      IN      SOA      jarvis.maria.org.
admin.maria.org. (
    1 ; serial
    21600 ; refresh (6 hours)
    3600 ; retry (1 hour)
    604800 ; expire (1 week)
    21600 ; minimum (6 hours)
)
@      IN      NS      jarvis.maria.org.
105    IN      PTR     jarvis.maria.org.
103    IN      PTR     pepper.maria.org.
101    IN      PTR     stark.maria.org.
```

Para aplicar los cambios, reiniciamos el servicio

```
systemctl restart bind9
```

Es importante tener en cuenta que para usarlo, tenemos añadir la siguiente línea al fichero /etc/resolv.conf de nuestro equipo

```
nameserver 172.22.200.105
```

```
maria@holmes:~$ cat /etc/resolv.conf
nameserver 172.22.200.105
nameserver 127.0.1.1
search home
```

Por último, con el comando dig comprobamos que está bien configurado:

```

maria@holmes:~$ dig stark.maria.org

; <<> DiG 9.10.3-P4-Ubuntu <<> stark.maria.org
;; global options: +cmd
;; Got answer:
;; ->HEADER<<- opcode: QUERY, status: NOERROR, id: 3057
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags;; udp: 4096
;; QUESTION SECTION:
;stark.maria.org.                IN      A

;; ANSWER SECTION:
stark.maria.org.                86400  IN      A      172.22.200.101

;; AUTHORITY SECTION:
maria.org.                      86400  IN      NS     jarvis.maria.org.

;; ADDITIONAL SECTION:
jarvis.maria.org.              86400  IN      A      172.22.200.105

;; Query time: 92 msec
;; SERVER: 172.22.200.105#53(172.22.200.105)
;; WHEN: Wed Mar 28 13:30:03 CEST 2018
;; MSG SIZE rcvd: 97

```

```

maria@holmes:~$ dig -x 172.22.200.103

; <<> DiG 9.10.3-P4-Ubuntu <<> -x 172.22.200.103
;; global options: +cmd
;; Got answer:
;; ->HEADER<<- opcode: QUERY, status: NOERROR, id: 20968
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags;; udp: 4096
;; QUESTION SECTION:
;103.200.22.172.in-addr.arpa.    IN      PTR

;; ANSWER SECTION:
103.200.22.172.in-addr.arpa.    86400  IN      PTR    pepper.maria.org.

;; AUTHORITY SECTION:
200.22.172.in-addr.arpa.       86400  IN      NS     jarvis.maria.org.

;; ADDITIONAL SECTION:
jarvis.maria.org.              86400  IN      A      172.22.200.105

;; Query time: 85 msec
;; SERVER: 172.22.200.105#53(172.22.200.105)
;; WHEN: Wed Mar 28 13:43:12 CEST 2018
;; MSG SIZE rcvd: 123

```

5. INSTALACIÓN

Lo primero es instalar la fuente del paquete y las claves públicas de couchbase:

```
ubuntu@stark:~/descargas$ curl -O
http://packages.couchbase.com/releases/couchbase-
release/couchbase-release-1.0-4-amd64.deb
ubuntu@stark:~/descargas$ sudo dpkg -i couchbase-release-1.0-4-
amd64.deb
```

Con todo configurado, podemos descargar el paquete:

```
ubuntu@stark:~/descargas$ sudo apt update
ubuntu@stark:~/descargas$ sudo apt install couchbase-server
```

Al finalizar la instalación, nos muestra una serie de requisitos, de forma que si no se cumple alguno, antes de hacer ninguna configuración, podamos cambiarlo.

```
Minimum RAM required : 4 GB
```

```
System RAM configured : 3.86 GB
```

```
Minimum number of processors required : 4 cores
```

```
Number of processors on the system : 2 cores
```

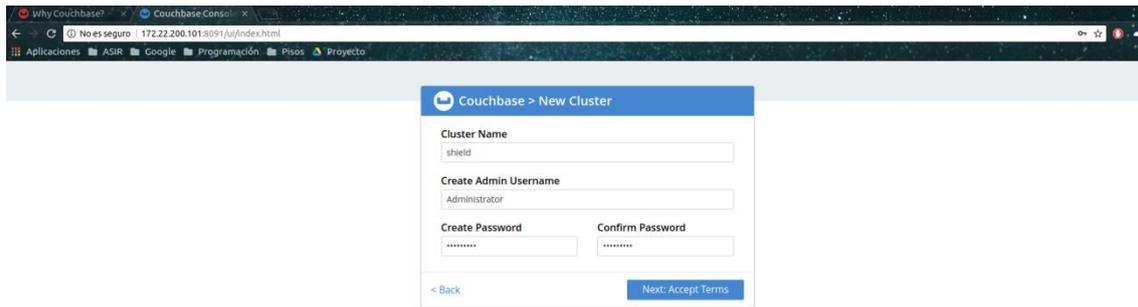
Si nos vamos a la url <http://172.22.200.101:8091/> o <http://stark.maria.org:8091/> vemos:



6. CONFIGURACIÓN BÁSICA

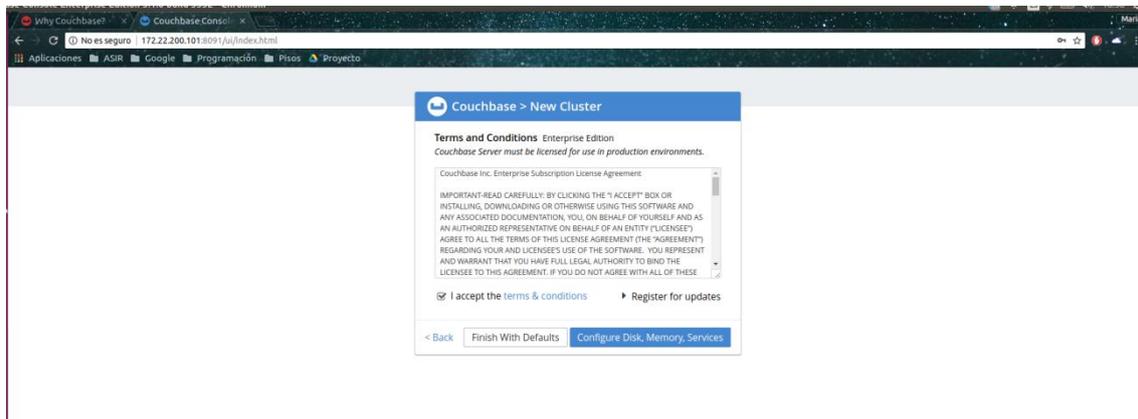
Una vez hayamos completado lo anterior, en el navegador podemos instalar y configurar nuestra base de datos. Los pasos a seguir son:

1. Dar nombre al cluster y proporcionar las credenciales del administrador.



2. Aceptar términos y condiciones.

Para continuar, nos da dos opciones: instalar con la configuración por defecto, o elegir nosotros ciertos datos, que es la que he seleccionado.



3. En este paso podemos definir varias opciones:

- Host name / IP address
 - ◆ Nombre del host o dirección ip en el que se va a realizar la instalación.
- Data disk path

Directorio del nodo actual en el que se almacenarán los ficheros.

➤ Indexes data path

Directorio del nodo actual en el que se almacenarán los índices.

En este caso, al ser un entorno de pruebas no hay problema en que estén en el mismo directorio ficheros e índices, sin embargo, en un entorno de producción es muy recomendable que estén separados.

➤ Couchbase memory quotas

Campos que nos permiten especificar la memoria que cada servicio asociado puede usar de nuestro nodo y los que se añadan más tarde. Cada servicio tendrá un mínimo y la suma de todos no puede superar la memoria total disponible.

Data Service: si se está creando un cluster nuevo, este servicio, necesario para el funcionamiento básico del cluster, aparece deshabilitado. En caso de estar en un entorno de desarrollo, se podrían añadir hasta 3 servicios, pero si estamos en producción, se recomienda solo uno por nodo.

Index Service: memoria para *Global Secondary Indexes*. Tiene un mínimo de 256 MB.

Search Service : memoria para *Full Text Service*. Tiene un mínimo de 256 MB.

Query Service : no necesita RAM mínima,

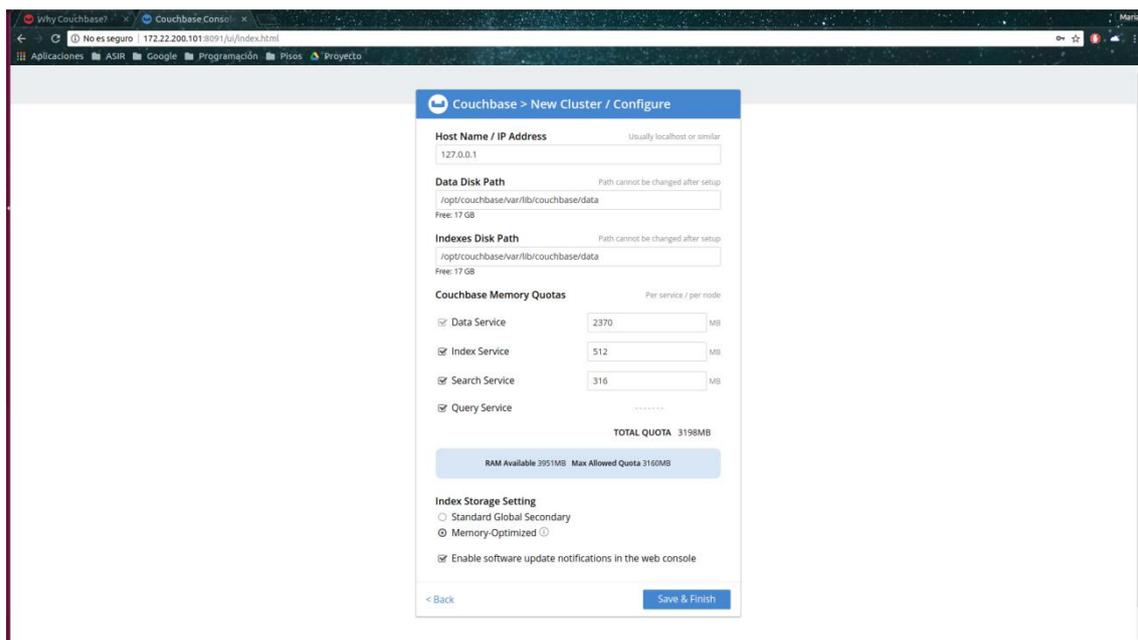
Si la asignación es excesiva, nos aparece una notificación para que se pongan valores más adecuados.

➤ Index storage settings: si Index Service ha sido activado, podemos elegir entre dos tipos de almacenamiento:

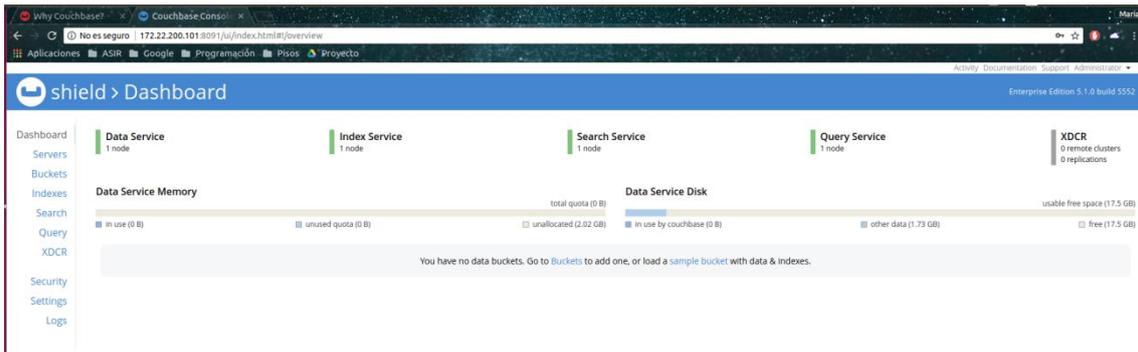
- ◆ Standard Global Secondary Indexes: en este caso cada índice tiene un fichero para él solo
- ◆ Memory-Optimized Global Secondary Indexes: con esta opción tenemos una menor latencia y mejor uso del espacio disponible.

Al seleccionarlo nos aparece el siguiente aviso, que nos advierte de que, aunque tiene un buen rendimiento, hay que estar pendientes de la memoria asignada a los índices.

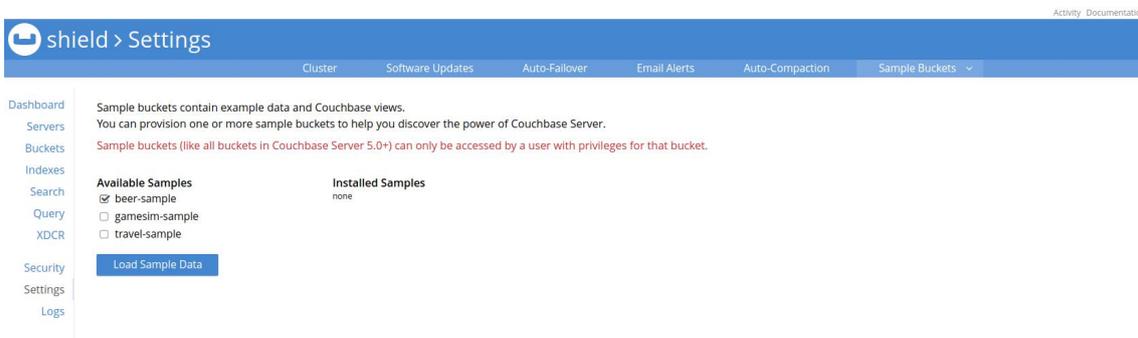
Memory-optimized indexing is highly performant but requires careful attention to your index RAM quota. It is an Enterprise-only feature.



4. Accedemos con las credenciales del primer paso y ya tenemos la instalación completada.



5. Para poder tener datos de prueba, en la pestaña de Settings tenemos 3 ejemplos para poder empezar a trabajar. Para añadirlos, seleccionamos los que queramos y le damos a Load Sample Data.



7. N1QL

Las sentencias se dividen en 2 grupos:

- **DDL**: todo lo relacionado con los índices.
- **DML**: todo lo relacionado con los datos de los documentos JSON.

Al igual que en SQL, se permiten hacer subconsultas y consultas a otras tablas.

7.1 EXPRESIONES

De agregación

Toman múltiples valores de los documentos, realizan cálculos y devuelven un solo valor. Las funciones hacen distinción de mayúsculas y minúsculas.

Solo se pueden usar en SELECT, LETTING, HAVING y ORDER BY.

Las funciones más comunes son *AVG(expression)*, *COUNT(expression)*, *MAX(expression)*, *MIN(expression)*, *SUM(expression)*.

Aritméticas

Son las operaciones matemáticas básicas dentro de una expresión que devuelve un valor . N1QL incluye un operador para cambiar el signo del valor. Si uno de los operandos no existe, el resultado será 'MISSING'. Si uno de los operandos es NULL o no es un número, el resultado será 'NULL'

Los operadores son + , - , * , / , %(modulo)

De colección

Permiten evaluar expresiones sobre colecciones u objetos. Los operadores son:

ANY, *EVERY*, *ARRAY*, *FIRST*, *EXISTS*, *IN* y *WITHIN*.

- ANY: si al menos uno de los elementos cumple con la expresión, devuelve la cadena completa, si no, devuelve una cadena vacía.

```
ANY var1 ( IN | WITHIN ) expr1
      [ , var2 ( IN | WITHIN ) expr2 ]*
SATISFIES condition END
```

Ejemplo:

Obtener los vuelos de Albuquerque (ABQ) a Atlanta (ATL) si alguno de los vuelos sale a partir de las 23:40.

```
SELECT * FROM `travel-sample` WHERE type="route" AND
airline="KL" AND sourceairport="ABQ" AND
destinationairport="ATL" AND ANY departure IN schedule SATISFIES
departure.utc > "23:41" END;
```

Array

El operador ARRAY permite mapear y filtrar los elementos de una colección o un objeto.

```
ARRAY var1 FOR var1 ( IN | WITHIN ) expr1
      [ , var2 ( IN | WITHIN ) expr2 ]*
      [ ( WHEN cond1 [ AND cond2 ] ) ] END
```

Devuelve la cadena que cumple con las condiciones o una cadena vacía.

Ejemplo:

Mostrar las cadenas de los vuelos nocturnos de los Viernes desde Albuquerque a Atlanta después de las 7 de la tarde.

```

SELECT ARRAY v FOR v IN schedule WHEN v.utc > "19:00" AND v.day
= 5 END AS fri_evening_flights
FROM `travel-sample`
WHERE type="route" AND airline="KL" AND sourceairport="ABQ" AND
destinationairport="ATL"
AND ANY v IN schedule SATISFIES v.utc > "19:00" END;

```

Every

Comprueba que una condición booleana sobre los elementos u objetos de una colección se cumpla. Se utiliza con los operadores IN y WITHIN. Devuelve todos los items que cumplan con la condición o una cadena vacía.

```

EVERY var1 ( IN | WITHIN ) expr1
[ , var2 ( IN | WITHIN ) expr2 ]*
SATISFIES condition END

```

Ejemplo:

Obtener los horarios de los vuelos desde Albuquerque (ABQ) a Atlanta (ATL) si todos los vuelos salen después de las 00:35.

```

SELECT * FROM `travel-sample`
WHERE type="route" AND airline="KL" AND sourceairport="ABQ" AND
destinationairport="ATL"
AND EVERY departure IN schedule SATISFIES departure.utc >
"00:35" END;

```

Exists

Se usa en combinación con subconsultas y se cumple si devuelve al menos un valor la segunda consulta. Se puede utilizar en sentencias SELECT, INSERT, UPDATE o DELETE.

```
search_expr EXISTS target_expr
```

Ejemplo:

De las 1641 ciudades con aeropuerto, mostrar todas las que tengan también algún monumento.

```
SELECT DISTINCT city FROM `travel-sample` AS l WHERE type =
"landmark"
AND EXISTS (SELECT city FROM `travel-sample` AS a WHERE type =
"airport");
```

First

Devuelve el primer elemento que cumpla con la condición, normalmente establecida con WHERE o una cadena vacía en caso de que no haya coincidencias.

```
FIRST var1 FOR var1 ( IN | WITHIN ) expr1
[ , var2 ( IN | WITHIN ) expr2]*
[ ( WHEN cond1 [ AND cond2 ] ) ] END
```

Ejemplo:

Mostrar los vuelos de Albuquerque a Atlanta a partir de las 7 de la tarde.

```
SELECT FIRST v FOR v IN schedule WHEN v.utc > "19:00" END AS
evening_flights
FROM `travel-sample`
```

```
WHERE type="route" AND airline="KL" AND sourceairport="ABQ" AND
destinationairport="ATL"
AND ANY v IN schedule SATISFIES v.utc > "19:00" END;
```

In

Se utiliza para determinar el nivel de “profundidad” de búsqueda dentro del documento, es decir, hasta en cuántos hijos va a buscar.

```
search_expr [ NOT ] IN target_expr
```

Ejemplo:

Mostrar todas las aerolíneas del Reino Unido o Francia,

```
SELECT * FROM `travel-sample` AS t
WHERE type = "airline" AND country IN ["United Kingdom",
"France"];
```

Comparación

Los operadores de comparación permiten evaluar dos expresiones. Los operadores son: =, ==, !=, <>, >, >=, <, <=, BETWEEN, NOT BETWEEN, LIKE, NOT LIKE, IS NULL, IS NOT NULL, IS MISSING, IS NOT MISSING, IS VALUED, IS NOT VALUED.

Condicionales

Evalúan expresiones lógicas.

```
simple-case-expression | searched-case-expression
```

Missing

La cláusula MISSING es un literal específico de N1QL para los casos en los que no existe el valor buscado.

Otros tipos

Existen otros tipos de expresiones que son:

- Constructores de cadenas y objetos
- Alias (funcionan como en SQL)
- Literales
- Booleanos (funcionan como en SQL)
- Números (funcionan como en SQL)
- NULL
- Expresiones anidadas (funcionan como en SQL)
- Cadenas: CONTAINS, INITCAP, LENGTH, LOWER, LTRIM, RTRIM, POSITION, REPLACE, REVERSE, RTRIM, SPLIT, SUBSTR, SUFFIXES, TITLE (alias de INITCAP), TOKENS, TRIM, UPPER.
- Lógicos: AND, OR, NOT (funcionan como en SQL)

Comentarios:

```
/* [ [text] | [newline] ]+ */
```

7.2 EJEMPLOS

Para ver los campos;

```
cbq> SELECT OBJECT_NAMES(b) FROM `beer-sample` b limit 1;
```

```
{
  "requestID": "288ec6e7-274a-4177-93fc-82e947703b6c",
  "signature": {
    "$1": "array"
  },
  "results": [
    {
      "$1": [
        "address",
        "city",
        "code",
        "country",
        "description",
        "geo",
        "name",
        "phone",
        "state",
        "type",
        "updated",
        "website"
      ]
    }
  ],
  "status": "success",
  "metrics": {
    "elapsedTime": "15.673349ms",
    "executionTime": "15.615218ms",
    "resultCount": 1,
  }
}
```

```

    "resultSize": 352
  }
}

```

Para ver todos los estados posibles:

```
cbq> select distinct state from `beer-sample`;
```

```
cbq> select distinct state, count(state) from `beer-sample`
group by state;
```

```

    {
      "$1": 2,
      "state": "Mississippi"
    },
    {
      "$1": 2,
      "state": "Manchester"
    },
    {
      "$1": 1,
      "state": "Nova Scotia"
    },
    {
      "$1": 1,
      "state": "OH"
    },
    {
      "$1": 6,
      "state": "Wyoming"
    },
    {
      "$1": 12,
      "state": "Iowa"
    },
    {
      "$1": 1,
      "state": "ME"
    },
    {
      "$1": 4,
      "state": "Niedersachsen"
    },
    {
      "$1": 17,
      "state": "Nebraska"
    },
    {
      "$1": 2,
      "state": "Bedford"
    },
    {
      "$1": 1,
      "state": "Phoenix"
    }
  ],
  "status": "success",
  "metrics": {
    "elapsedTime": "740.273744ms",
    "executionTime": "740.242592ms",
    "resultCount": 169,
    "resultSize": 10915
  }
}

```

Mostrar el nombre y el ID de 2 tiendas si existe ese ID

```
cbq> SELECT name, brewery_id from `beer-sample` WHERE brewery_id  
IS NOT MISSING LIMIT 2;
```

```
{  
  "requestID": "fb75057a-93b1-4c94-901f-93acd88e8545",  
  "signature": {  
    "brewery_id": "json",  
    "name": "json"  
  },  
  "results": [  
    {  
      "brewery_id": "21st_amendment_brewery_cafe",  
      "name": "21A IPA"  
    },  
    {  
      "brewery_id": "21st_amendment_brewery_cafe",  
      "name": "563 Stout"  
    }  
  ],  
  "status": "success",  
  "metrics": {  
    "elapsedTime": "30.021285ms",  
    "executionTime": "29.991321ms",  
    "resultCount": 2,  
    "resultSize": 198  
  }  
}
```

Mostrar el nombre y el ID de 2 tiendas si no existe ese ID

```
cbq> SELECT name, brewery_id from `beer-sample` WHERE brewery_id  
IS MISSING LIMIT 2;
```

```
{  
  "requestID": "d9c92df7-d2b1-45c0-af52-cc5f74938645",  
  "signature": {  
    "brewery_id": "json",  
    "name": "json"  
  },  
  "results": [  
    {  
      "name": "21st Amendment Brewery Cafe"  
    },  
    {  
      "name": "357"  
    }  
  ],  
  "status": "success",  
  "metrics": {  
    "elapsedTime": "34.887968ms",  
    "executionTime": "34.868429ms",  
    "resultCount": 2,  
    "resultSize": 98  
  }  
}
```

Eliminar el documento con nombre "563 Stout"

```
cbq> DELETE FROM `beer-test` WHERE name = "563 Stout";
```

```
{
  "requestID": "b088f22b-d0b8-41e8-b32b-32ebc397c638",
  "signature": null,
  "results": [
  ],
  "status": "success",
  "metrics": {
    "elapsedTime": "651.592766ms",
    "executionTime": "651.563722ms",
    "resultCount": 0,
    "resultSize": 0,
    "mutationCount": 1
  }
}
```

```
}
```

```
cbq> SELECT name, brewery_id from `beer-test` WHERE name = "563
Stout";
```

```
{
  "requestID": "a01cd268-d884-4013-b86b-e9668ca649e6",
  "signature": {
    "brewery_id": "json",
    "name": "json"
  },
  "results": [
  ],
  "status": "success",
  "metrics": {
    "elapsedTime": "630.009645ms",
    "executionTime": "629.969079ms",
```

```
    "resultCount": 0,  
    "resultSize": 0  
  }  
}
```

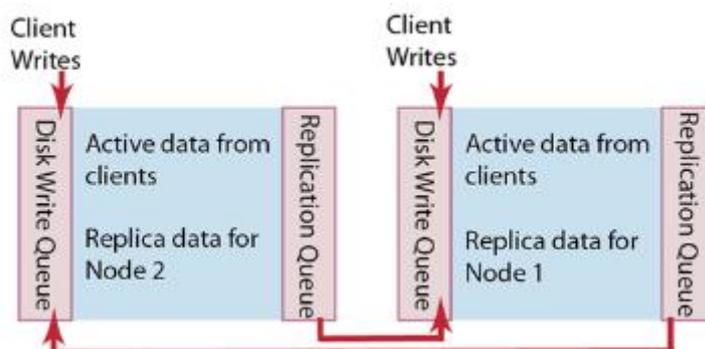
8. RÉPLICAS

La replicación está distribuida a través del cluster para prevenir puntos de fallo y es configurable a nivel de bucket y nodo.

Dentro de un cluster, tenemos datos activos y datos de replica en cada nodo. Los datos activos son los que se han escrito directamente en ese nodo y los de réplica, son copias de datos activos de otro nodo.

La replicación está basada en peer-to-peer de forma que la información se copia directamente entre los nodos. No existe topología, jerarquía o relación maestro-esclavo en este caso.

En la siguiente imagen vemos cómo se realiza este proceso:



Cuando un cliente escribe en un nodo, esos datos se almacenan en una cola de replicación y luego una copia se manda al otro nodo. Los datos replicados se guardan en la RAM del otro nodo y entran en una cola de escritura a disco para ser almacenados de forma definitiva en el segundo nodo.

En caso de haber demasiada información para almacenar a la vez, ya sea de réplica o activa, el nodo saturado manda un mensaje de “backoff” para que el nodo reduzca la velocidad a la que manda información hasta que se estabilice el nodo que haya mandado el mensaje.

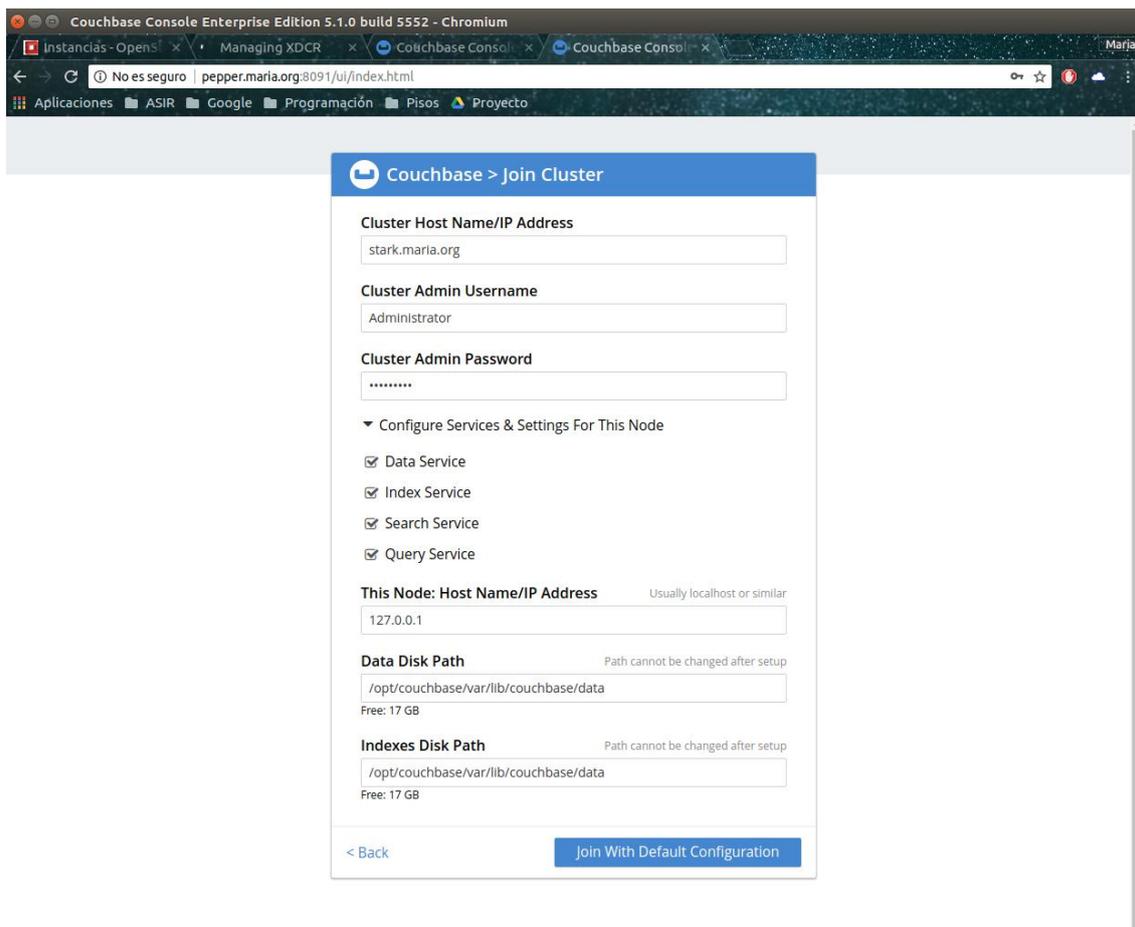
Si mientras está en la cola un documento se modifica varias veces, couchbase para mejorar la eficiencia, “para” el envío hasta que deja de modificarse un tiempo y manda únicamente la última versión.

En caso de fallo del primer nodo, los datos siguen estando disponibles en el nodo secundario y almacenados en RAM. En los clientes no es necesario configurar nada ya que el propio servidor es el que decide si las consultas van al nodo principal o a alguno de los secundarios.

Para añadir una réplica, hay que instalar en el otro nodo el paquete completo, y en los pasos de configuración, en vez de crear un nuevo cluster, lo añadimos al que ya existe. Los pasos serían:

1. Instalación del paquete

```
ubuntu@pepper:~/descargas$ curl -O
http://packages.couchbase.com/releases/couchbase-
release/couchbase-release-1.0-4-amd64.deb
ubuntu@pepper:~/descargas$ sudo dpkg -i couchbase-release-1.0-4-
amd64.deb
ubuntu@pepper:~/descargas$ sudo apt update
ubuntu@pepper:~/descargas$ sudo apt install couchbase-server
```



Consulta de prueba

```
root@pepper:/opt/couchbase/bin# ./cbq -u Administrator -  
engine=http://172.22.200.103:8091/
```

Enter Password:

```
Connected to : http://172.22.200.103:8091/. Type Ctrl-D or  
\QUIT to exit.
```

Path to history file for the shell : /root/.cbq_history

```
cbq> select * from `beer-sample` limit 1;  
{  
  "requestID": "f7c2b9a0-88b9-4376-b653-a998c7062484",  
  "signature": {
```

```

    "*": "*"
  },
  "results": [
    {
      "beer-sample": {
        "address": [
          "563 Second Street"
        ],
        "city": "San Francisco",
        "code": "94107",
        "country": "United States",
        "description": "The 21st Amendment Brewery
offers a variety of award winning house made brews and American
grilled cuisine in a comfortable loft like setting. Join us
before and after Giants baseball games in our outdoor beer
garden. A great location for functions and parties in our semi-
private Brewers Loft. See you soon at the 21A!",
        "geo": {
          "accuracy": "R00FTOP",
          "lat": 37.7825,
          "lon": -122.393
        },
        "name": "21st Amendment Brewery Cafe",
        "phone": "1-415-369-0900",
        "state": "California",
        "type": "brewery",
        "updated": "2010-10-24 13:54:07",
        "website": "http://www.21st-amendment.com/"
      }
    }
  ]
}

```

```
        }
    }
],
"status": "success",
"metrics": {
    "elapsedTime": "22.244385ms",
    "executionTime": "22.21335ms",
    "resultCount": 1,
    "resultSize": 1055
}
}
```

Este método es adecuado para entornos de prueba o que no sean críticos, pero para escenarios reales, es mucho más adecuada la replicación XDCR.

8.1 REPLICACIÓN XDCR

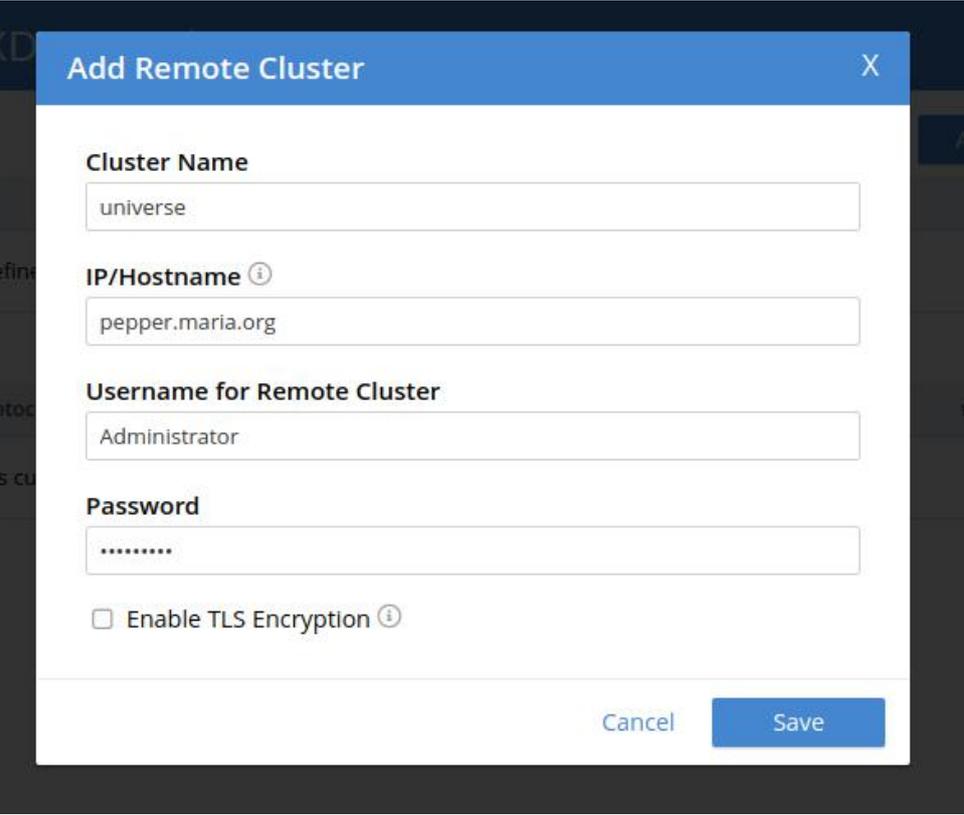
En este caso, los clusters están repartidos geográficamente en diversos centros de datos ya sea por protección ante daños naturales o para dar mayor velocidad a los clientes según donde estén. Se puede configurar como unidireccional o bidireccional y proporciona resolución de conflictos.

Es muy importante tener en cuenta que solo se replican datos, es decir, no se incluyen vistas o índices. Para que también se repliquen hay que hacerlo a mano y de esta forma se creará la vista o el índice en el otro cluster.

Ejemplo de réplica unidireccional

En la consola web nos encontramos con la pestaña XDCR, que nos va a permitir manejar todas las réplicas. Para crear una, primero tenemos que añadir un cluster remoto y luego la réplica como tal.

Para crear el cluster, le damos a “Add remote cluster” y nos muestra este formulario



Add Remote Cluster X

Cluster Name
universe

IP/Hostname ⓘ
pepper.maria.org

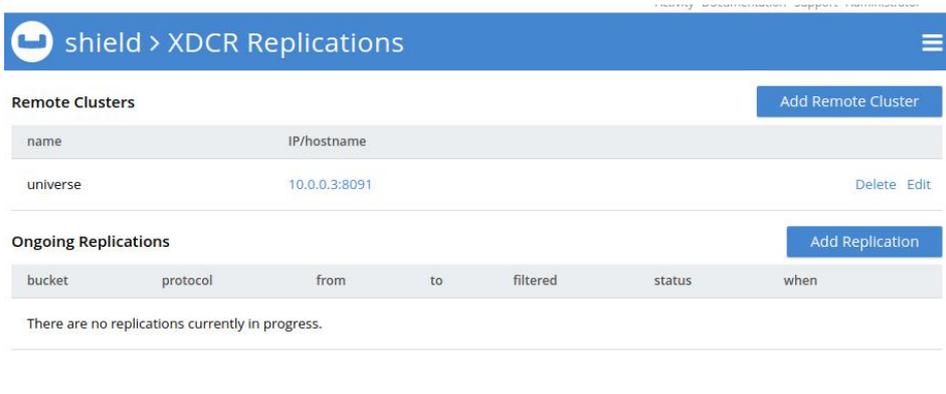
Username for Remote Cluster
Administrator

Password

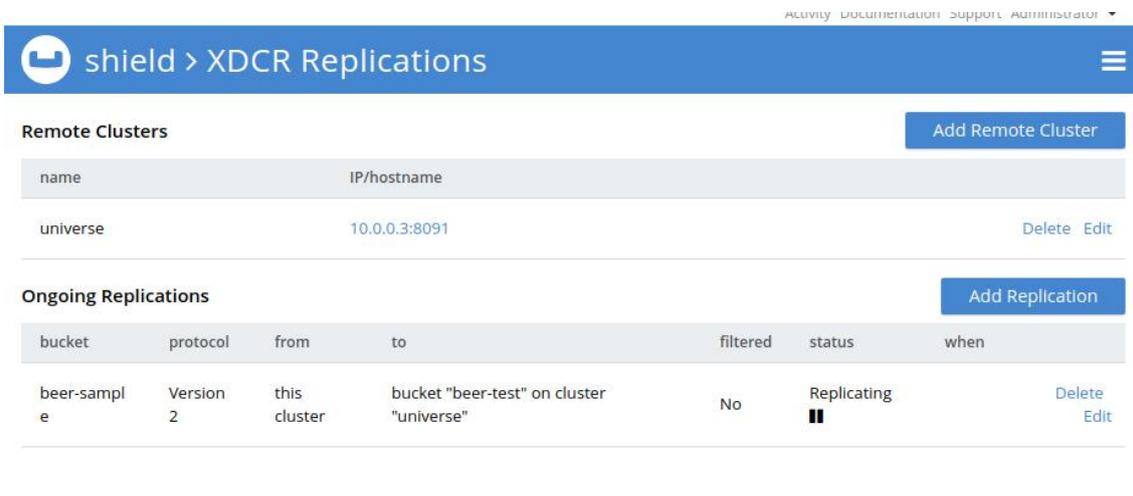
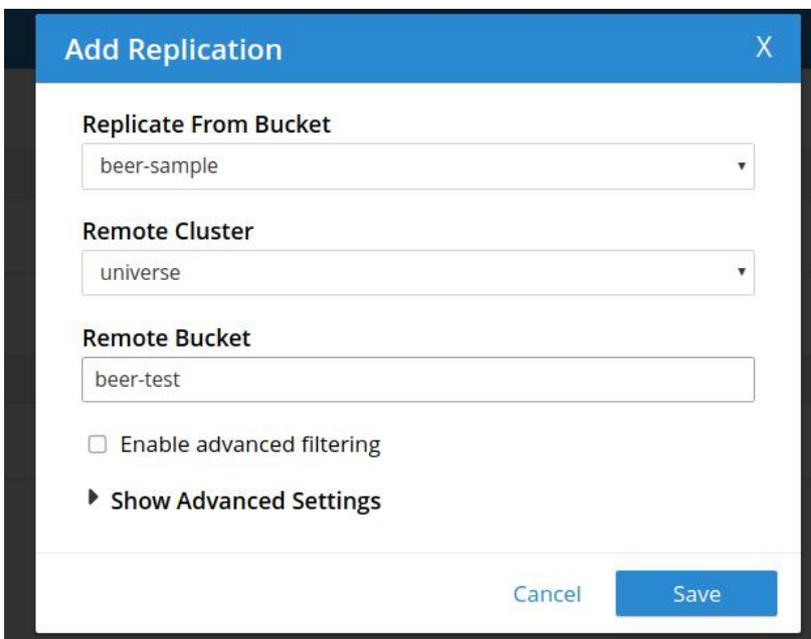
Enable TLS Encryption ⓘ

Cancel Save

Una vez rellenado y si está todo correcto, ya está creado el cluster remoto.



Lo siguiente es crear la réplica en “Add replication”. Al igual que en el caso anterior, nos muestra un formulario en el que en “Remote Bucket” tenemos que poner el nombre de un bucket que ya exista en el destino.



Con estos simples pasos ya tenemos una réplica XDCR unidireccional lista.

9. COPIAS DE SEGURIDAD

En un entorno en producción hay que hacer copias de los datos de cara a posibles fallos y así evitar la pérdida de datos y minimizar la inconsistencia de datos en caso de tener que hacer alguna restauración. Para esto couchbase proporciona dos herramientas

- cbackupmgr (solo disponible en la versión de pago)
- cbackup y cbrestore (disponible en todas las versiones)

Dada la naturaleza de los servidores de Couchbase, no se puede realizar una copia en tiempo de ejecución y un snapshot del cluster completo ya que los datos están en constante cambio.

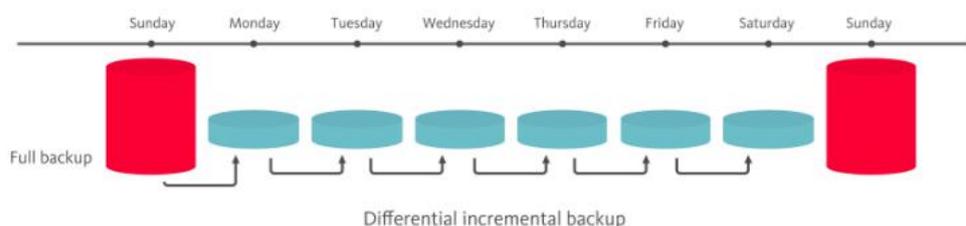
9.1 CBACKUP Y CBRESTORE

Proporcionan copias incrementales y completas.

Existen dos tipos de copias incrementales: diferenciales u acumulativas.

Diferenciales

Almacenan los cambios desde la ultima copia. Se crean rápido ya que la cantidad de datos suele ser pequeña, pero su restauración es más larga que las acumulativas.



Lunes: cambios desde la ultima copia completa del domingo

Martes: cambios desde el lunes

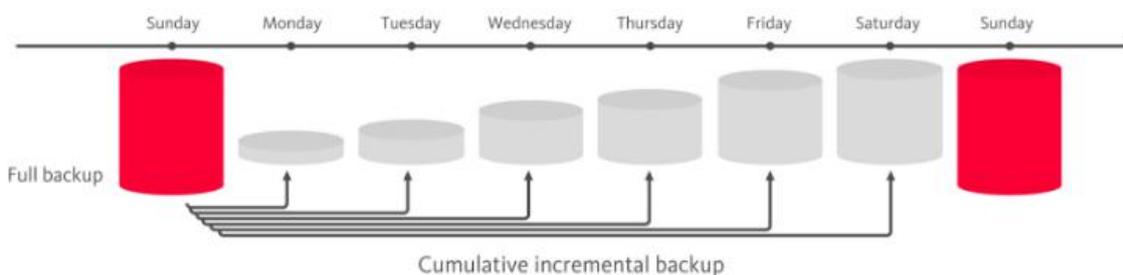
Miércoles: cambios desde el martes

...

Una restauración hecha el miércoles usaría la completa del domingo más las incrementales del lunes y martes.

Acumulativas

Contienen todos los cambios desde la última copia completa. Sus restauraciones son mas rápidas, pero las copias requieren mas tiempo y espacio en disco.



Lunes: cambios desde la ultima del domingo

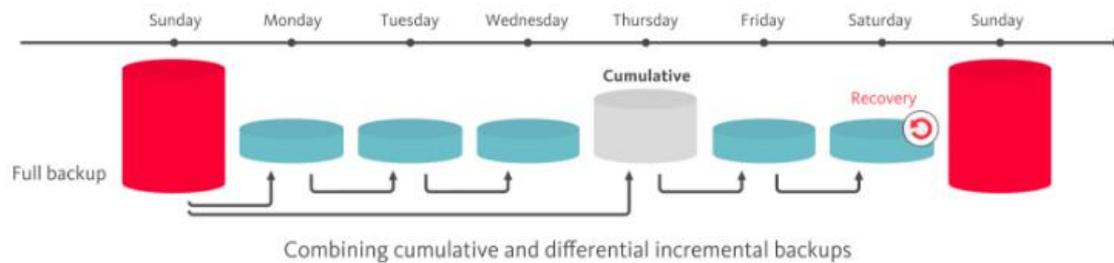
Martes: cambios desde la ultima del domingo

...

Una restauración el miércoles usará la copia del domingo y la acumulativa del martes

Combinación de ambas

Para mejorar el rendimiento del proceso de restauración, se pueden combinar ambos métodos.



En función del día, se hace una u otra.

Jueves: acumulativa

Resto: diferencial

Si se restaura el sábado se usa la del domingo y la del jueves y la del viernes

9.2 CBBACKUP

Puede crear copias de un cluster, un bucket completo , en un nodo o un solo bucket en un solo nodo.

```
cbbbackup [options] [source] [backup-dir] -u [admin] -p [password]
```

Donde *[backup-dir]* es el directorio en el que se almacenaran los datos. Si no existe, se crea, siempre cuando el directorio padre exista. Para crear una copia nueva, hay que crear un nuevo directorio.

Para la versión gratuita, solo esta disponible la copia completa.

Para crear una copia, el nodo (o cluster) tiene que estar operativo. Se tiene que ejecutar en local (o a través de ssh o similares)

Nos permite realizar copias de:

- todos los buckets de un cluster
- un solo bucket de un cluster
- todos los buckets de un solo nodo

- un solo bucket en un nodo

Por defecto, se hace una copia de los índices secundarios del cluster. Si se hace un copia de un solo nodo, solo se harán copia de los índices de dicho nodo y solo si tiene el servicio de indexado activado

La herramienta se encuentra en `/opt/couchbase/bin/cbbackup`

Opciones:

- h, --help** ayuda
- b BUCKET_SOURCE, --bucket-source=BUCKET_SOURCE** bucket de origen
- single-node** para indicar que se hará en un solo nodo
- m MODE, --mode=MODE** tipo de backup
 - full para copias completas
 - diff para diferenciales
 - accu para acumulativas
- i ID, --id=ID** copiar elementos que contengan ese id
- k KEY, --key=KEY** copiar elementos cuya clave contengan la expresión regular que se indique
- n, --dry-run** no se hace copia, solo comprueba la conexión
- u USERNAME, --username=USERNAME** usuario
- p PASSWORD, --password=PASSWORD** contraseña
- s, --ssl** habilitar ssl
- t THREADS, --threads=THREADS** hilos simultáneos para la transferencia
- v, --verbose** nivel de los mensajes, a mas v, mas mensajes
- silent** solo muestra los errores

-x EXTRA, --extra=EXTRA para indicar configuraciones especiales

9.2.1 EJEMPLOS

Copia de un cluster completo

```
ubuntu@stark:~$ cbackup http://172.22.200.101:8091 backups -u
Administrator -p $CBPASS
```

```
[#####] 100.0% (7303/estimated 7303 msgs)
bucket: beer-sample, msgs transferred...
      :          total |      last |    per sec
byte  :          2542924 |    2542924 |    733938.2
done
```

Un solo bucket en un cluster

```
ubuntu@stark:~$ cbackup http://172.22.200.101:8091
backups/backup-20180612 -u Administrator -p $CBPASS -b beer-
sample
```

```
[#####] 100.0% (7303/estimated 7303 msgs)
bucket: beer-sample, msgs transferred...
      :          total |      last |    per sec
byte  :          2542924 |    2542924 |    1026626.5
done
```

Vemos como en el directorio backups tenemos creado el directorio backup-20180612

```
ubuntu@stark:~$ ls backups/
backup-20180612  bucket-beer-sample
```

Todas las claves de un bucket con el prefijo 'object'

```
ubuntu@stark:~$ cbbbackup http://172.22.200.101:8091
backups/backup-object -u Administrator -p $CBPASS -b beer-sample
-k '^object.*'
```

Vemos que se han creado los ficheros correspondientes:

```
ubuntu@stark:~$ ls backups/backup-object/2018-06-
12T104002Z/2018-06-12T104002Z-full/bucket-beer-sample/node-
10.0.0.5%3A8091/
data-0000.cbb meta.json
```

9.3 CBRESTORE

Herramienta de restauración de datos. La sintaxis es:

```
cbrestore [options] [backup-dir] [destination]
```

Donde *[backup-dir]* es el directorio en el que se almacenaran los datos. Si no existe, se crea, siempre cuando el directorio padre exista.

Aspectos a tener en cuenta

- Los datos escritos a un fichero en disco, se restauran en la RAM
- Sobre los índices secundarios
 - ◆ Si la copia los contiene, también son parte del proceso de restauración
 - ◆ Se almacenan sin los nombres de los nodos, que son pasados posteriormente (CREATE INDEX). Si almacena los id de los índices.

- ◆ Cuando los índices ya están restaurados, se tienen que construir a mano con BUILD INDEX.
- Resolución de conflictos
 - ◆ Por defecto, se comprueba que los datos no se hayan sobrescrito de manera incorrecta
 - ◆ A veces, hay documentos que no coinciden con el contenido de la copia si se han eliminado muy recientemente.

La herramienta esta en `/opt/couchbase/bin/cbrestore`

Opciones

- h, --help** ayuda
- a, --add** añade sin sobrescribir datos
- b BUCKET_SOURCE, --bucket-source=BUCKET_SOURCE** bucket de origen
- B BUCKET_DESTINATION, --bucket-destination=BUCKET_DESTINATION** bucket de destino. Si no se especifica, coge el mismo que el origen
- from-date=FROM_DATE** fecha desde la que se quiere restaurar. Por defecto, restaura desde el primer día.
- i ID, --id=ID** copiar elementos que contengan ese id
- k KEY, --key=KEY** copiar elementos cuya clave contengan la expresión regular que se indique
- m MODE, --mode=MODE**
- n, --dry-run** no se hace copia, solo comprueba la conexión
- u USERNAME, --username=USERNAME** usuario
- p PASSWORD, --password=PASSWORD** contraseña
- single-node** para indicar que se hará en un solo nodo

- t THREADS, --threads=THREADS hilos simultáneos para la transferencia
- v, --verbose nivel de los mensajes, a mas v, mas mensajes
- x EXTRA, --extra=EXTRA para indicar configuraciones especiales

9.3.1 EJEMPLOS

Para hacer la prueba, he borrado la información del documento con nombre "563 Stout"

```
cbq> DELETE FROM `beer-test` WHERE name = "563 Stout";
cbq> SELECT name, brewery_id from `beer-test` WHERE name = "563
Stout";
{
  "requestID": "a01cd268-d884-4013-b86b-e9668ca649e6",
  "signature": {
    "brewery_id": "json",
    "name": "json"
  },
  "results": [
  ],
  "status": "success",
  "metrics": {
    "elapsedTime": "630.009645ms",
    "executionTime": "629.969079ms",
    "resultCount": 0,
    "resultSize": 0
  }
}
```

Para restaurar, el comando sería

```
cbrestore backups/ http://172.22.200.101:8091 -b beer-sample -u
Administrator -p $CBPASS
```

Si hacemos la consulta de nuevo:

```
cbq> SELECT name, brewery_id from `beer-test` WHERE name = "563
Stout";
```

```
{
  "requestID": "129129325c0-5347-526a-c7b2-527c24b41f8f",
  "signature": {
    "brewery_id": "json",
    "name": "json"
  },
  "results": [
    {
      "brewery_id": "21st_amendment_brewery_cafe",
      "name": "563 Stout"
    }
  ],
  "status": "success",
  "metrics": {
    "elapsedTime": "635.157459ms",
    "executionTime": "635.092341ms",
    "resultCount": 1,
    "resultSize": 100
  }
}
```

10. IMPORTACIÓN Y EXPORTACIÓN

10.1 EXPORTACIÓN

La sintaxis es:

```
cbexport [--version] [--help] <command> [<args>]
```

Con cbexport se pueden exportar los datos en diversos formatos. El más utilizado es json, siendo su sintaxis específica:

```
cbexport json [--cluster <url>]
             [--bucket <bucket_name>]
             [--format <data_format>]
             [--username <username>][--password <password>]
             [--include-key <key>][--cacert <path>][--no-ssl-
             verify]
             [--threads <num>][--log-file <path>]
```

El comando copia documentos JSON ya existentes a un fichero con un documento por línea o un fichero que tiene una lista en JSON en la que cada elemento es un documento.

Opciones obligatorias

-c, --cluster <url> host del nodo del cluster del que se van a exportar los datos.

-u, --username <username> usuario del cluster. Necesita tener los permisos adecuados.

-p, --password <password> contraseña del usuario anterior.

- b, --bucket <bucket_name> nombre del bucket.
- f, --format <format> formato de los datos (lineas o lista)
- o, --output <path> fichero en el que se guardarán los datos exportados

Formato de host

Al especificar un host, podemos ponerlo de 3 formas:

- > couchbase://<addr> (la mas recomendada si se comunica por el puerto por defecto)
- > <addr>:<port>
- > http://<addr>:<port>

Formato de datos

Los datos se pueden presentar de distintas formas:

Líneas: Contiene un documento JSON por línea en el fichero.

```
{"key": "mykey1", "value": "myvalue1"}  
  
{"key": "mykey2", "value": "myvalue2"}  
  
{"key": "mykey3", "value": "myvalue3"}  
  
{"key": "mykey4", "value": "myvalue4"}
```

Lista: Contiene una lista donde cada elemento es un documento JSON,

```
[  
  
  {  
  
    "key": "mykey1",  
  
    "value": "myvalue1"  
  
  },  
  
  {"key": "mykey2", "value": "myvalue2"},
```

```
{ "key": "mykey3", "value": "myvalue3"},  
{ "key": "mykey4", "value": "myvalue4"}  
]
```

10.1.1 EJEMPLOS

Exportar con formato de linea

```
ubuntu@pepper:/opt/couchbase/bin$ ./cbexport json -c  
couchbase://172.22.200.103 -u Administrator -p $CBPASS -b beer-  
test -o /home/ubuntu/data/lines.json -f lines  
Json exported to `/home/ubuntu/data/lines.json` successfully  
ubuntu@pepper:/opt/couchbase/bin$ ls -l  
/home/ubuntu/data/lines.json  
-rw-rw---- 1 ubuntu ubuntu 2549798 Jun 12 11:24  
/home/ubuntu/data/lines.json
```

Exportar con formato de lista

```
ubuntu@pepper:/opt/couchbase/bin$ ./cbexport json -c  
couchbase://172.22.200.103 -u Administrator -p $CBPASS -b beer-  
test -o /home/ubuntu/data/list.json -f list  
ubuntu@pepper:/opt/couchbase/bin$ ls -l  
/home/ubuntu/data/list.json  
-rw-rw---- 1 ubuntu ubuntu 2557101 Jun 12 11:26  
/home/ubuntu/data/list.json
```

10.2 IMPORTACIÓN

Al igual que en el caso de la exportación, se puede realizar sobre varios tipos de ficheros, siendo el más común JSON con la siguiente sintaxis

```
cbimport json [--cluster <url>]
               [--bucket <bucket_name>]
               [--dataset <path>]
               [--format <data_format>]
               [--username <username>]
               [--password <password>]
               [--generate-key <key_expr>]
               [--cacert <path>]
               [--no-ssl-verify]
               [--threads <num>]
               [--error-log <path>]
               [--log-file <path>]
```

Opciones obligatorias

-c, --cluster <url> host del nodo del cluster del que se van a exportar los datos.

-u, --username <username> usuario del cluster. Necesita tener los permisos adecuados.

-p, --password <password> contraseña del usuario anterior.

-b, --bucket <bucket_name> nombre del bucket.

-d, --dataset <uri> URI de los datos que van a ser importados (local o URL) Si el un fichero local, tiene que comenzar con file://. Si es una URL, empezará por http:// o https://.

-f, --format <format> formato del fichero (lineas, lista, muestra)

Formato de datos

Soporta los formatos descritos anteriormente más el formato muestra, en el que se especifica un ZIP con los documentos. Está pensado para los buckets de ejemplo de couchbase. A diferencia de las anteriores, contiene vista, índice y definiciones de índices completas. No hay que especificar el directorio con file://.

Todos los documentos en este formato están en el mismo directorio y solo hay un fichero por documento. El nombre del fichero es la clave para el documento JSON del fichero.

El directorio design_docs tiene los índices. El fichero indexes.json está reservado para índices secundarios. Todos los demás ficheros están reservados para vistas

Generación de claves

Se utilizan para asignar una clave única para cada documento. Se suelen generar combinando columnas, valores arbitrarios o generadores personalizados.

10.2.1 EJEMPLOS

Importar de /data/lines.json usando como clave "name"

```
ubuntu@pepper:/opt/couchbase/bin$ ./cbimport json -c
couchbase://172.22.200.103 -u Administrator -p $CBPASS -b beer-
test -d file:///home/ubuntu/data/lines.json -f lines -g
key: :%name%
```

```
Json `file:///home/ubuntu/data/lines.json` imported to  
`http://172.22.200.103:8091` successfully
```

Importar de /data/lines.json usando como clave "name" y el generador de UUID

```
ubuntu@pepper:/opt/couchbase/bin$ ./cbimport json -c  
couchbase://172.22.200.103 -u Administrator -p $CBPASS -b beer-  
test -d file:///home/ubuntu/data/list.json -f list -g  
key::%name%::#UUID#  
Json `file:///home/ubuntu/data/list.json` imported to  
`http://172.22.200.103:8091` successfully
```

11. MÉTRICA

Para poder tener controlados los equipos del entorno, he elegido prometheus y grafana para mostrar los datos. Ambas son dos herramientas potentes que permiten obtener información de prácticamente todos los componentes del entorno, facilitando así la prevención de fallos que provoquen paradas o, en caso de que ocurran, poder encontrar la fuente del origen con mayor rapidez.

En el equipo jarvis están instalados grafana, prometheus y node-exporter (que es el agente encargado de coger los datos). En los equipos stark y pepper está únicamente node-exporter.

11.1 INSTALACIÓN DEL SERVIDOR (JARVIS)

1. Descarga del paquete

```
cd /tmp
```

```
wget
```

```
https://github.com/prometheus/prometheus/releases/download/v2.0.0/prometheus-2.0.0.linux-amd64.tar.gz
```

```
tar -xvzf prometheus-2.0.0.linux-amd64.tar.gz
```

```
cd prometheus-2.0.0.linux-amd64
```

2. Añadimos los usuarios necesarios para prometheus y node-exporter

```
sudo useradd --no-create-home --shell /bin/false prometheus
```

```
sudo useradd --no-create-home --shell /bin/false node_exporter
```

3. Creamos los directorios y asignamos los permisos correspondientes

```
sudo mkdir /etc/prometheus
```

```
sudo mkdir /var/lib/prometheus
sudo chown prometheus:prometheus /etc/prometheus
sudo chown prometheus:prometheus /var/lib/prometheus
```

4. Copiamos los binarios a sus directorios correspondientes

```
sudo cp prometheus-2.0.0.linux-amd64/prometheus /usr/local/bin
sudo cp prometheus-2.0.0.linux-amd64/promtool /usr/local/bin
```

5. Descargamos el paquete de grafana

```
wget https://s3-us-west-2.amazonaws.com/grafana-
releases/release/grafana_4.6.2_amd64.deb
sudo apt install libfontconfig
sudo dpkg -i grafana_4.6.2_amd64.deb
sudo systemctl daemon-reload
sudo systemctl start grafana-server
sudo systemctl status grafana-server
```

6. Descargamos el node-exporter

```
wget
https://github.com/prometheus/node_exporter/releases/download/v0.
15.1/node_exporter-0.15.1.linux-amd64.tar.gz
tar xvf node_exporter-0.15.1.linux-amd64.tar.gz
```

7. Copiamos los binarios y asignamos los permisos correspondientes

```
sudo cp node_exporter-0.15.1.linux-amd64/node_exporter
/usr/local/bin
```

```
sudo chown node_exporter:node_exporter
/usr/local/bin/node_exporter
```

8. Creamos la unidad de systemd para facilitar la depuración

```
sudo nano /etc/systemd/system/node_exporter.service
```

```
[Unit]
```

```
Description=Node Exporter
```

```
Wants=network-online.target
```

```
After=network-online.target
```

```
[Service]
```

```
User=node_exporter
```

```
Group=node_exporter
```

```
Type=simple
```

```
ExecStart=/usr/local/bin/node_exporter
```

```
[Install]
```

```
WantedBy=multi-user.target
```

9. Añadimos la unidad e iniciamos el servicio

```
sudo systemctl daemon-reload
```

```
sudo systemctl start node_exporter
```

```
sudo systemctl status node_exporter
```

```
sudo systemctl enable node_exporter
```

10. En el fichero de configuración de prometheus, añadimos node-exporter con todos los hosts

```
sudo nano /etc/prometheus/prometheus.yml

global:
  scrape_interval: 15s # By default, scrape targets every 15
seconds.
  evaluation_interval: 15s # By default, scrape targets every 15
seconds.

scrape_configs:
  - job_name: 'prometheus'
    scrape_interval: 5s
    scrape_timeout: 5s
    static_configs:
      - targets: ['localhost:9090']
  - job_name: 'node_exporter'
    scrape_interval: 5s
    static_configs:
      - targets:
        ['localhost:9100', '172.22.200.101:9110', '172.22.200.103:9110']
```

11. Para aplicar los cambios, reiniciamos prometheus

```
sudo systemctl restart prometheus
```

11.2 INSTALACIÓN EN LOS CLIENTES (STARK Y PEPPER)

1. Descargamos el paquete

```
cd /tmp/
```

wget

```
https://github.com/prometheus/node_exporter/releases/download/v0.15.1/node_exporter-0.15.1.linux-amd64.tar.gz
tar xvf node_exporter-0.15.1.linux-amd64.tar.gz
```

2. Copiamos el binario a su carpeta correspondiente y asignamos los permisos

```
sudo cp node_exporter-0.15.1.linux-amd64/node_exporter
/usr/local/bin
sudo chown node_exporter:node_exporter
```

3. Creamos la unidad de systemd

```
sudo nano /etc/systemd/system/node_exporter.service
[Unit]
Description=Node Exporter
Wants=network-online.target
After=network-online.target

[Service]
User=node_exporter
Group=node_exporter
Type=simple
ExecStart=/usr/local/bin/node_exporter --web.listen-
address :9110

[Install]
WantedBy=multi-user.target
```

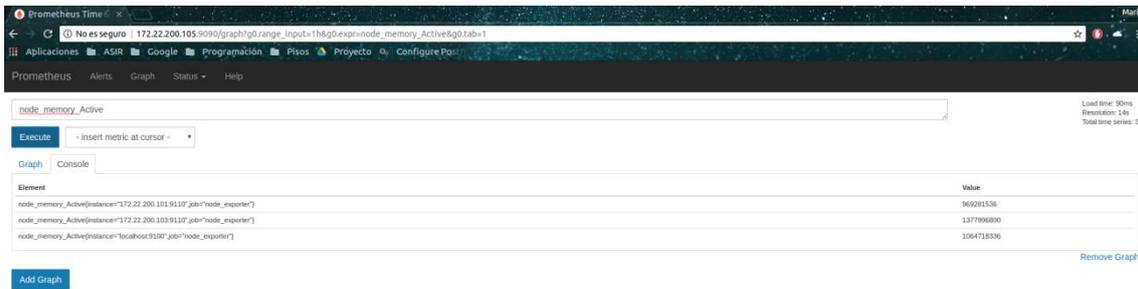
4. Añadimos la unidad e iniciamos el servicio

```
systemctl daemon-reload
```

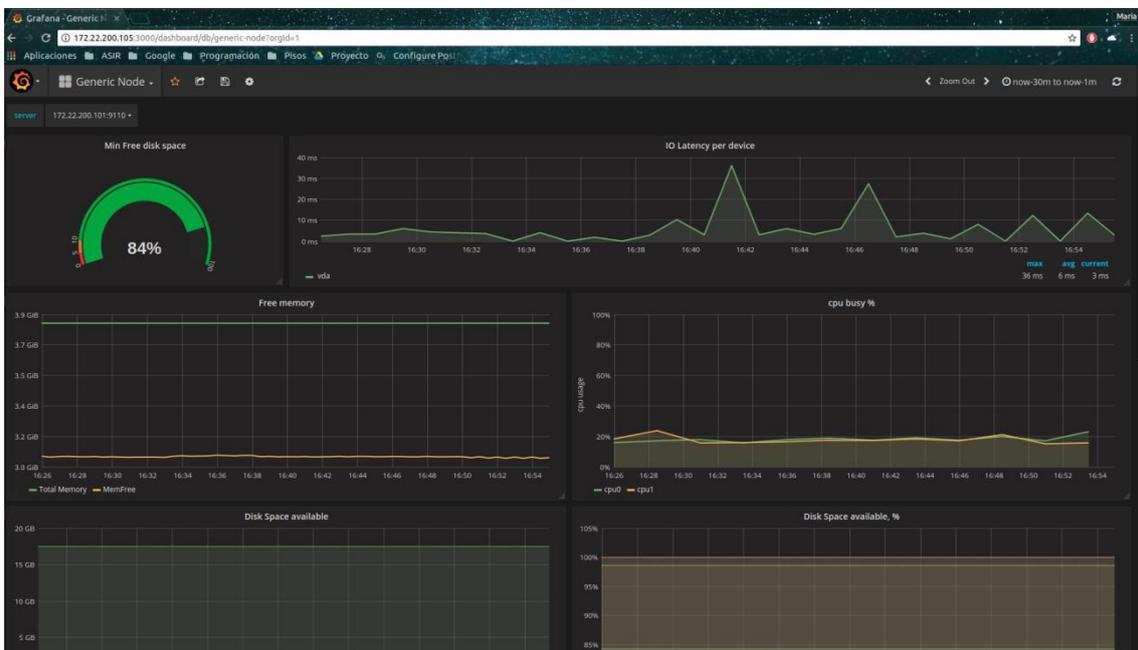
```
systemctl start node_exporter
```

Para comprobar que prometheus está recogiendo datos, accedemos a la url

<http://172.22.200.105:9090>



Una vez añadido el dashboard, vemos como se muestran ya los datos de forma gráfica.



12. APLICACIÓN

Todo el código y la explicación de la aplicación está en el repositorio

<https://github.com/marromang/couchweb>

13. CONCLUSIONES

Dadas las necesidades actuales de los servidores en cuanto a velocidad de acceso a datos y fácil replicación couchbase se presenta como una alternativa potente a las bases de datos relacionales.

Es fácil de manejar con una curva de aprendizaje relativamente suave y la gran similitud de su lenguaje N1QL con el SQL convencional facilita la tarea del cambio.

Permite realizar miles de aplicaciones, en casi la gran mayoría de lenguajes y con una muy buena documentación. Su consola web es muy completa, salvo por las copias de seguridad, motivo por el cual he hecho la aplicación web.

En resumen, es una base de datos potente, sencilla y permite una alta personalización.

14. BIBLIOGRAFÍA

- <https://blog.pandorafms.org/es/bases-de-datos-nosql/>
- <https://es.slideshare.net/JuanRebes/the-top10enterpriseno-sqlusecases>
- <https://www.wikitechy.com/tutorials/couchbase/couchbase-tutorial>
- <https://riunet.upv.es/bitstream/handle/10251/55530/Memoria.pdf?sequence=1>
- <https://es.wikipedia.org/wiki/NoSQL>
- https://es.wikipedia.org/wiki/Base_de_datos_relacional
- <http://slashmobility.com/blog/2016/12/nosql-vs-sql-cual-elegir/>
- <https://adminlte.io/>
- <https://marromang.wordpress.com/2017/12/11/metricas-con-telegraf-prometheus-y-grafana/>

15. POSIBLES MEJORAS

- Montar el escenario con kubernetes para mejorar la automatización de la alta disponibilidad.
- Añadir a la aplicación web toda la gestión completa para no tener una para copias de seguridad y monitorización y otra para todo lo demás.
- Crear un buen sistema de alertas para mejorar el tiempo de respuesta ante fallos.
- Meter la aplicación web en apache para no tener que lanzarla a mano.
- Añadir usuarios y autenticación con LDAP.