[¿Que es Cassandra?](#_oqyo7ya52fx)

[Características de Cassandra](#_almj1wq0hw4b)

[Distribuido](#_7ggcamalgf5n)

[Descentralizado](#_mcatw5rk2vk3)

[Alta Escalabilidad](#_nbozg21rwucs)

[Tolerancia a Fallos](#_7bwtpqc0ilj)

[Lenguaje CQL](#_lc9rd0wknvsy)

[Cassandra y el teorema de CAP](#_ejxrawh2jxlu)

[Eventualmente Consistente](#_ys2bmx3388tt)

[Consistencia estricta](#_isdox7ssm4pk)

[Consistencia casual](#_dua6x7arp4qy)

[Consistencia débil o eventual.](#_6wpcgbs1pvw2)

[Arquitectura: escritura de datos](#_z1eev55s3msc)

[Pedido de escritura y almacenamiento en la memtable](#_vgrfpokibc3u)

[Flushing (vaciando) datos de la memtable](#_td3c6pdrxcb3)

[Almacenamiento de datos en el disco (SSTables)](#_mk5s1zkaxq2n)

[Arquitectura](#_b892g729ttkl)

[Espacio de Claves](#_63reo9nzds5u)

[Peer to Peer (P2P)](#_rqled2yape07)

[Pedido de Coordinación (entendiendo el proceso)](#_501dkwcurpaw)

[Particionado](#_6z8w34ikky6)

[Replicación](#_ld4ev5jhs0vm)

[Gossiping](#_f0koj3jv2f2z)

[Snitch](#_tix707hk52zw)

[Anti entropía](#_dqe1esu1hnaj)

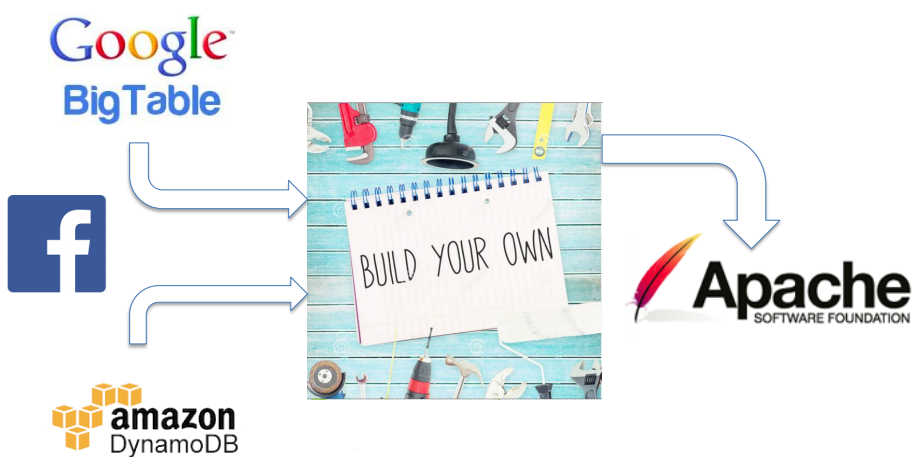
[Diferencias entre MongoDB y Cassandra](#_qv3r663a3eo5)

# ¿Que es Cassandra?



El desarrollo inicial de Cassandra tiene su origen en Facebook, que lo diseñó para potenciar la funcionalidad de búsqueda en el inbox. En 2008 fue liberado como proyecto open source y en febrero de 2010 se convirtió en un proyecto top-level de la fundación Apache. Está inspirado e influenciado por los papers de Amazon Dynamo de 2007 y de Google BigTable de 2006. Hoy en día está mantenido y desarrollado por la compañía Datastax.

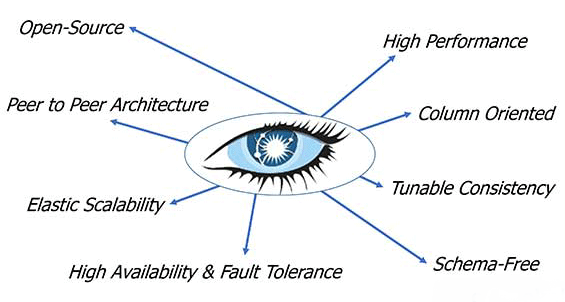
Su nombre está inspirado por la sacerdotisa Cassandra de la mitología griega, que tenía el don de la profecía, y predijo el engaño del [Caballo de Troya](https://es.wikipedia.org/wiki/Caballo_de_Troya).



Actualmente Cassandra se utiliza en varios de los sitios más conocidos y con más necesidades en cuanto a tráfico y almacenamiento tienen en el mundo, como por ejemplo, Netflix, Ebay, Spotify, Twitter, Reddit, SoundCloud o Digg entre otros.

Algunas particularidades:

* Cassandra puede manejar varios terabytes de datos si lo necesita y puede, fácilmente, manejar millones de archivos, incluso en un clúster pequeño (Big Data).
* La información en las bases de datos relacionales, se almacenan en forma de filas, pero en Cassandra la información se almacena en columnas con pares key-value.
* Esquema dinámico. El esquema que define la estructura de los datos puede cambiar en tiempo de ejecución.
* No hay un único punto de fallo. Los datos se replican automáticamente a varios nodos. Perder un nodo no causa la baja del clúster.
* Alta disponibilidad. Los datos están disponibles la mayor parte del tiempo gracias a la redundancia que introduce la replicación de datos.
* Particionamiento de los datos. La topología de Cassandra es la de un anillo a través del cual se distribuyen los datos para minimizar cuellos de botella en el acceso a los mismos (P2P).
* Escalabilidad horizontal. Hasta un alto número de máquinas la capacidad de cómputo aumenta linealmente con el número de máquinas.
* Capacidad para manejar cientos de gigabytes de datos.
* Soporte profesional: actualmente varias empresas dan soporte y construyen productos sobre Cassandra.
* Está escrito en Java, es multiplataforma



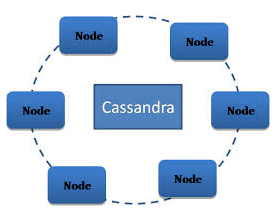
## Características de Cassandra

### Distribuido

Cassandra es distribuido, esto significa que puede mantener cuántos nodos sean necesarios para su correcto funcionamiento, pero para un cliente o una aplicación externa siempre parecerá un sistema con un punto de entrada único.

Cassandra está especialmente diseñado y preparado para funcionar en modo distribuido. Esto no significa que no se puedan sacar buenos rendimiento de un único servidor con Cassandra instalado y funcionando, pero cuando realmente se obtienen beneficios de Cassandra es cuando está distribuido (como con la mayoría de las bases de datos noSQL).

### Descentralizado

Que Cassandra sea descentralizado significa que es un sistema que ha sido pensado de forma que n*o exista ningún punto de único de fallo (SPF)*. Todos los nodos en Cassandra desempeñan la misma funcionalidad: no hay maestros, no existe un nodo que coordine las actividades que desempeñan los demás.

En la mayoría de las bases de datos distribuidas funcionan de forma que uno de los nodos recibe las escrituras, y el resto de nodos del clúster replican esa información y mantienen copias de lectura de la base de datos de cara a las aplicaciones. Incluso bases de datos relacionales poseen modos de funcionamiento que les permiten desarrollar una funcionalidad similar a la que se acaba de describir.

El problema de mantener una solución centralizada de este tipo es el punto de fallo único del sistema. Este tipo de arquitecturas están pensadas para escalar las lecturas de forma que la carga se reparta entre varias máquinas, pero ¿qué sucede si se tiene un sistema que realiza muchas más escrituras que lecturas? Además, si en el esquema descrito anteriormente, el maestro (master) falla, se está poniendo en riesgo el correcto funcionamiento del sistema completo.

Por tanto, la descentralización proporciona principalmente dos beneficios:

* funcionalmente es más sencillo que los sistemas maestro – esclavo (master/slave).
* te ayuda a evitar pérdidas en el servicio (disponibilidad).

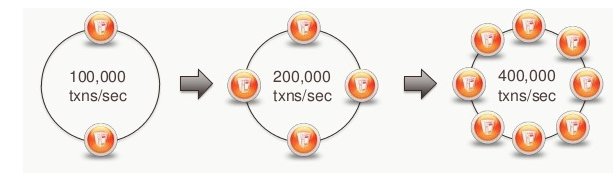
### Alta Escalabilidad

La escalabilidad podría ser definida como la capacidad de un sistema para, ante un incremento considerable de la demanda en un corto periodo de tiempo, continuar ofreciendo el servicio sin degradar la calidad del mismo.

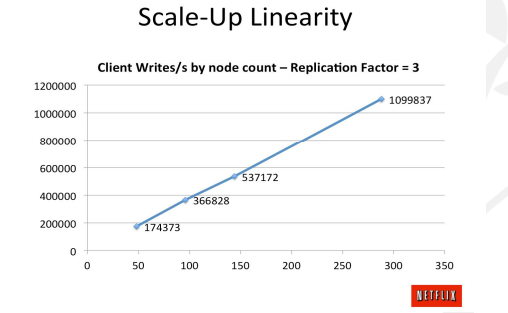
La escalabilidad horizontal es la mejor manera de alcanzar esta meta. Agregar más máquinas cuando el sistema lo necesita, unido a la filosofía de este tipo de sistemas de utilizar hardware de coste moderado, es la opción que mejor se perfila para empezar con un sistema pequeño y gastos menores, e ir ampliando el sistema a medida que las necesidades lo van haciendo necesario.

Por supuesto, también está la opción de hacer un escalado “hacia abajo”, o lo que es lo mismo, sacar máquinas del clúster cuando debido a la poca carga que está recibiendo el

sistema ya no son necesarias.



Escala linealmente, lo que quiere decir que el rendimiento de forma lineal respecto al número de nodos que añadamos. Por ejemplo, si con 2 nodos soportamos 100.000 operaciones por segundo, con 4 nodos soportaremos 200.000. Esto da mucha predictibilidad a nuestros sistemas.



### Tolerancia a Fallos

Cassandra es tolerante a fallos. Si uno de los nodos tiene un error de comunicaciones, el resto pueden cumplir su función sin problema, y en caso de que sea necesario, meter un nuevo nodo en lugar del que está fallando. Si el error es de hardware, el nodo puede ser retirado hasta que se solucione el problema que le afecta. Si es necesario, puede incluso ser totalmente retirado del clúster y sustituido por otro nuevo.

### Lenguaje CQL

Cassandra Query Language (CQL) es el lenguaje de acceso a datos en Cassandra, es un derivado reducido de SQL. En Cassandra los datos están desnormalizados de manera que el concepto de joins o subqueries no existe.

Podemos interactuar con Cassandra mediante CQL a través de la shell. de CQL, cqlshell. También podemos usar herramientas gráficas como [DevCenter](http://www.datastax.com/what-we-offer/products-services/devcenter) o a través de los drivers soportados para múltiples lenguajes de programación.

## Cassandra y el teorema de CAP

### Eventualmente Consistente

Le eventualidad en la consistencia le viene “impuesto” por [el teorema de CAP](https://es.wikipedia.org/wiki/Teorema_CAP):

Cassandra es un sistema AP. Esto significa que antepone la disponibilidad de la información y la capacidad del sistema a funcionar con el sistema particionado, a la consistencia de los datos.

La consistencia básicamente hace referencia a la capacidad del sistema de devolver, ante una petición de lectura, siempre la información más actualizada que se había escrito con anterioridad.

Decir que Cassandra es eventualmente consistente no es del todo cierto. Cassandra es altamente parametrizable, y como parte de esta parametrización, el sistema permite incrementar la consistencia del sistema todo lo que se desee. Eso sí, a cambio de sacrificar disponibilidad.

La consistencia eventual es tan solo uno de los múltiples modelos que los arquitectos de sistemas pueden escoger para su instalación de Cassandra.

Otras posibles configuraciones serían:

### Consistencia estricta

La consistencia estricta significa que cada lectura que se realice en el sistema devolverá la escritura más recientemente recibida. En una máquina única y local en el que se realizan escrituras no es excesivamente complicado saber qué es lo último que se ha escrito, pero en un sistema que permite escrituras en todos sus nodos, que probablemente esté distribuido geográficamente por varios CPDs, y que previsiblemente va a recibir un elevado número de escrituras, las labores de sincronización y de asegurar que las últimas escrituras serás las respuestas devueltas cuando se realice una lectura, tendrán un elevado impacto en la disponibilidad del servicio, o en su defecto, en los tiempos de respuesta del mismo.

### Consistencia casual

La consistencia casual trata de relajar la carga de sincronización de la anterior solución aplicando una visión más semántica de las operaciones. El sistema “descubre” que operaciones de escritura están relacionadas de una u otra manera, y hace consistentes las llamadas de lectura relacionadas con esa información.

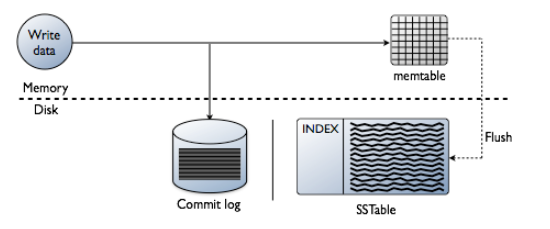
Si una escritura se produce tras otra ordenadamente, se puede deducir que están casualmente relacionadas. La consistencia casual dice que las escrituras casuales se deben leer de forma secuencial.

### Consistencia débil o eventual.

La consistencia eventual dice que la información que se escribe en un nodo del clúster actualizará al resto de nodos en algún momento, pero que podría tomar algún tiempo. Eventualmente las réplicas serán consistentes.

## 

# Arquitectura: escritura de datos



Cassandra procesa los datos en varias etapas, comenzando con el pedido de una escritura y finalizando con una escritura de datos en el disco:

* Registra datos en el *commit log* (log de confirmación).
* Escribe datos en el *memtable* (tabla en memoria).
* *Flush* (vacía) esos datos de la *memtable*
* Almacena los datos en disco en la *SSTables*

### Pedido de escritura y almacenamiento en la *memtable*

Cuando se produce una escritura, Cassandra almacena los datos en una estructura de memoria llamada *memtable*, y para proporcionar durabilidad (configurable), también agrega esas mismas escrituras a un registro de confirmación en el disco (commit log).

El registro de confirmación (commit log) recibe cada escritura realizada en un nodo de Cassandra, y estas escrituras duraderas permanecen incluso si falla la alimentación de un nodo. El memtable es [una caché de write-back](https://en.wikipedia.org/wiki/Cache_(computing)) de particiones de datos que Cassandra busca por clave. La *memtable* almacena escrituras ordenadas hasta que alcanza un límite configurable y entonces se vacía.

### Flushing (vaciando) datos de la *memtable*

Para eliminar los datos de la *memtable*, Cassandra escribe los datos en el disco, en el orden en el que estaban guardados en la *memtable*. También se crea un índice de partición en el disco que asigna los tokens para una ubicación en el disco. Cuando el contenido de la *memtable* excede el umbral configurado (threshold) o el espacio de registro de commit (commit log) excede el commitlog\_total\_space\_in\_mb, la *memtable* se coloca en una queue, para luego ser bajada al disco.

Una buena práctica es eliminar el memtable antes de reiniciar los nodos. Si un nodo deja de funcionar, al volver a estar operativo, del commit log se restauran las escrituras que estaban allí antes de que se detuvieran.

Los datos en el commit log (registro de confirmación) se purgan después de que sus datos correspondientes en la memoria se descarguen a un *SSTable* en el disco.

### Almacenamiento de datos en el disco (*SSTables*)

*Memtables* y *SSTables* se mantienen por tabla (de base de datos). El *commit log* se comparte entre las tablas de la base de datos. Los archivos *SSTables* son inmutables, no se escriben de nuevo después de que se haya limpiado la memoria. En consecuencia, en una partición se almacena típicamente varios archivos SSTable. Existen varias otras estructuras SSTable para ayudar a las operaciones de lectura:

Para cada SSTable, Cassandra crea estas estructuras (data files):

**Data (Data.db)**

Los datos (SSTable).

**Primary Index (Index.db)**

Índice de las "row keys" con punteros a sus posiciones en el archivo de datos.

**Bloom filter (Filter.db)**

Una estructura almacenada en la memoria que comprueba si los datos de fila existen en la *memtable* antes de acceder a *SSTables* en el disco

**Información de Compresión (CompressionInfo.db)**

Un archivo que contiene información sobre el tamaño de los datos sin comprimir y otra información de compresión

**Estadísticas (Statistics.db)**

Metadatos estadísticos sobre el contenido del SSTable

**Digest (Digest.crc32, Digest.adler32, Digest.sha1)**

A file holding adler32 checksum of the data file

**CRC (CRC.db)**

A file holding the CRC32 for chunks in an a uncompressed file.

**SSTable Index Summary (SUMMARY.db)**

A sample of the partition index stored in memory

**SSTable Table of Contents (TOC.txt)**

A file that stores the list of all components for the SSTable TOC

**Secondary Index (SI\_.\*.db)**

Built-in secondary index. Multiple SIs may exist per SSTable

Actividad práctica

Vemos un ejemplo con el [club atlético independiente:](https://es.wikipedia.org/wiki/Club_Atl%C3%A9tico_Independiente)

CREATE KEYSPACE *pablo\_espaciorojo* WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication\_factor' : 1 };

USE pablo\_espaciorojo;

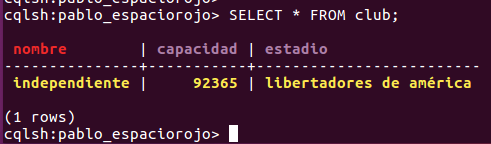
CREATE TABLE club(  
 nombre text,  
 estadio text,

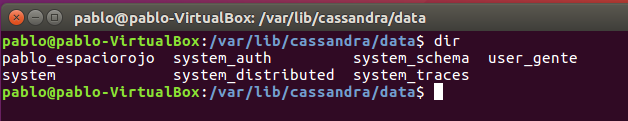
capacidad text,

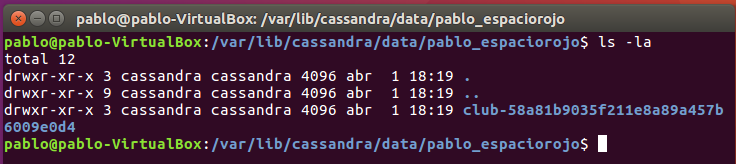
PRIMARY KEY(nombre));

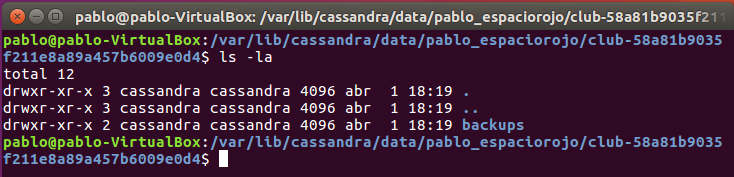
INSERT INTO club (nombre, estadio, capacidad) VALUES(‘independiente’, ‘libertadores de américa’, ‘92364’);





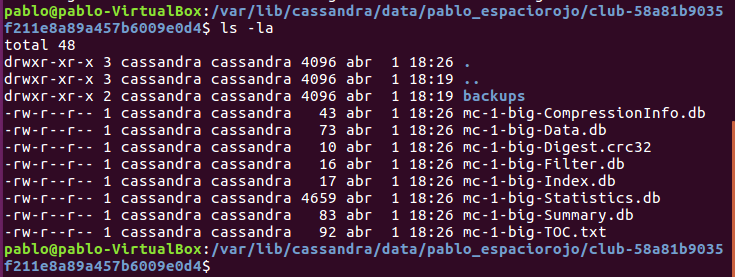






No tenemos nada….(en algunos casos puede que sí).

Vamos a bajar el servicio y subir y volvemos a revisar:



## Arquitectura

### Espacio de Claves

Cassandra mantiene un espacio en la base de datos llamado *system* que utiliza para gestionar la información sobre la metadata. Este esquema almacena metadata relacionada con el nodo local, como por ejemplo: el token del nodo, el nombre del clúster, el espacio de claves, la definición del esquema, etc.

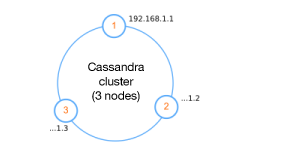
Concretamente, existen dos *column families* importantes. Uno gestiona el espacio de claves del usuarios y las definiciones de los esquemas, y el otro los cambios que se hacen sobre el espacio de claves. Son column families: *Schema* y *Migrations*.

### Peer to Peer (P2P)

El modelo que propone Cassandra es un modelo basado en la comunicación *peer-to-peer* entre nodos. De esta manera se consigue que todos los nodos sean funcionalmente idénticos entre ellos. Es decir, no existe un maestro que controle las escrituras, o cierta parte del clúster.

De este modo, el diseño P2P hace que el sistema escale de una forma mucho más sencilla, además de mejorar la disponibilidad de los datos, ya que la pérdida de un nodo no supone una degradación del servicio demasiado elevado.

Agregar nuevos nodos al clúster se convierte en una tarea casi trivial, puesto que todos los nodos desempeñan el mismo rol, lo único que hay que hacer es añadir un nodo nuevo al clúster y él solo aprenderá la topología existente en ese momento, y cuando esté preparado, se unirá al anillo de servidores para ser un miembro más del anillo.



En el estilo arquitectónico P2P, no hay nodos “principales”. Los clientes pueden conectarse a cualquiera de los nodos para realizar cualquier operación de I/O.

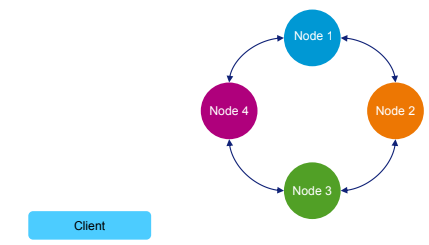
El nodo al que se conecta el cliente actúa como coordinador entre éste y el resto de nodos en dónde se encuentran los datos afectados por la consulta.

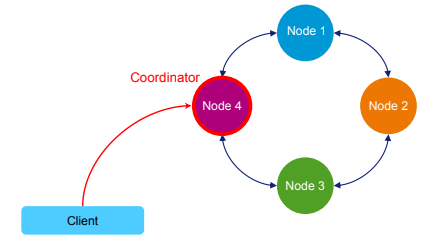
Una arquitectura Peer-to-Peer elimina los puntos de fallo único (SPF) ya que no sigue patrones del estilo maestro-esclavo como otros sistemas de almacenamiento (MongoDB).

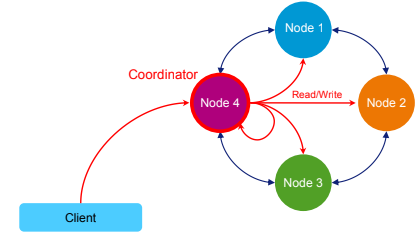
De esta manera cualquiera de los nodos puede tomar el rol de coordinador de una query. El coordinador determina qué nodos deben responder a la consulta. Internamente Cassandra replicará los datos entre los nodos con la política que le definamos, por ejemplo con el factor de replicación. Además soporta el concepto de data center para agrupar los nodos lógicamente y tener los datos más cerca del usuario.

### Pedido de Coordinación (entendiendo el proceso)

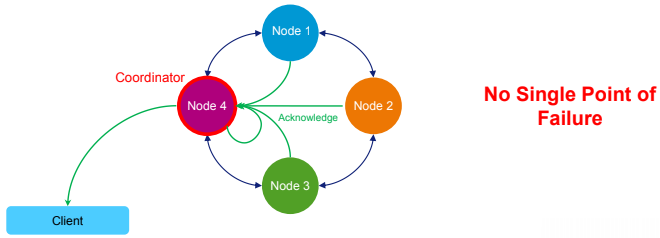
Coordinador: el cliente va a elegir un nodo quién recibirá una solicitud de lectura o escritura particular a su cluster.





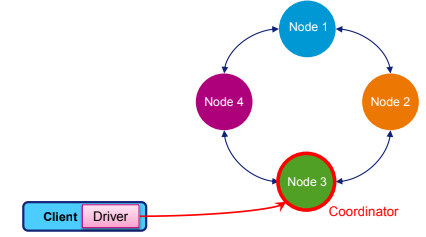


Cualquier nodo puede coordinar cualquier solicitud. Cada solicitud del cliente puede ser coordinada por un nodo diferente.



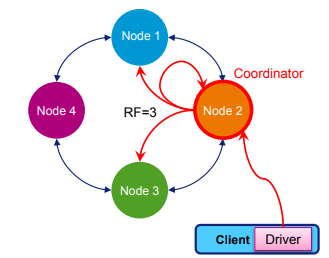
El driver de Cassandra elige quien es el nodo coordinador:

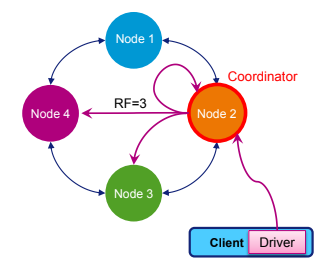
* Patrón [Round-Robin](https://es.wikipedia.org/wiki/Planificaci%C3%B3n_Round-robin), patrón token-aware (política de load balancing).
* La librería del cliente es quien gestiona dichas solicitudes
* En Cassandra existen muchos controladores de código abierto para muchos lenguajes de programación.



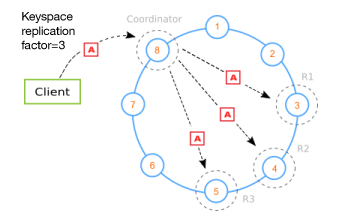
El coordinador maneja el proceso de replicación

* Factor de replicación (RF): en cuántos nodos se debe copiar una escritura.
* La escritura ocurrirá en los nodos involucrados para esa partición
* 1 ≤ RF ≤ (nodos # en el clúster)
* Cada escritura tiene un [timestamp](https://es.wikipedia.org/wiki/Marca_temporal) (marca de tiempo).





### Particionado



Cassandra, al igual que DynamoDB, particiona la información a través de todo el clúster utilizando Consistent Hashing.

Que Cassandra utilice Consistent Hashing para particionar sus datos significa que cada nodo es responsable de una parte de los datos contenidos en la base de datos. Igualmente, la salida o entrada de un nodo en el sistema afectará únicamente a sus vecinos directos en el anillo, que tendrán que hacerse cargo o repartir parte del anillo con sus nodos vecinos.

### Replicación

Cada dato de la base de datos es replicado N veces entre los nodos del clúster, donde N es el *factor de replicación*, y se configura individualmente por instancia. Cada servidor será el encargado de gestionar la replicación de esa parte concreta de la base de datos al resto de miembros que conforman el sistema.

Cassandra proporciona diversas políticas de replicación, tales como políticas sensibles al [rack](https://es.wikipedia.org/wiki/Bastidor_de_19_pulgadas) donde están ubicados los datos, o sensibles al [CPD](https://es.wikipedia.org/wiki/Centro_de_procesamiento_de_datos), de forma que al final se consiga tener la información distribuida por distintos racks de un mismo CPD, y distintos racks de otros CPDs.

Además, Cassandra proporciona garantías de disponibilidad de la información en presencia de errores a nivel de red y de CPD. Problemas de refrigeración o ventilación en los centros de datos, o problemas de red internas en los CPDs, son salvados por Cassandra permitiendo que los datos se distribuyan homogéneamente entre CPDs evitando así pérdidas de servicio.

### Gossiping

Cassandra hace uso de un protocolo de [*gossiping*](http://www.wordreference.com/es/translation.asp?tranword=gossiping) para hacer que sus nodos se comuniquen entre ellos y obtengan información los unos de los otros.

Primero se explicará este término. Los algoritmos de gossiping, a veces también llamados “protocolos epidémicos” tienen muy buen rendimiento, especialmente en grandes clústeres.

Gossip en castellano se traduce como “rumor”, y recibe este peculiar nombre porque la forma que tiene el sistema para divulgar la información se asemeja bastante a la forma en la que las personas transmitimos un rumor o chisme.

En Cassandra, cuando un nodo arranca, la clase Gossiper será la encargada de gestionar todas las comunicaciones con el resto de nodos de aquí en adelante. Esta clase mantendrá una lista de nodos vivos y muertos, y actúa de la siguiente manera:

Cada cierto tiempo, el gossiper elige un nodo del anillo de forma aleatoria e inicia una comunicación con él. Estas comunicaciones se basan en el intercambio de tres mensajes

inspirados en el método de apertura de conexión de las conexiones TCP:

1. Se envía un mensaje GossipDigestSynMessage,
2. A continuación el receptor responde con un GossipDigestAckMessage
3. Y finalmente el emisor nuevamente responde, esta vez con un GossipDigestAck2Message.

De esta manera, si el nodo objetivo no responde estos mensajes, se deducirá que es un nodo inaccesible y pasará a formar parte de la lista de nodos “muertos”. Al final, el conjunto de nodos

### Snitch

Un snitch determina desde qué centros de datos (CPDs) y racks se va a escribir y leer. El trabajo de un snitch es simplemente determinar la proximidad relativa del host. Por lo tanto, si un nodo tiene 3 opciones para copiar los datos, ¿qué host debe seleccionar? ¿De qué host debería preferir los datos?  
  
Si este es el tipo de información que el host anfitrión desea recibir, llamará a un snitch particular para ver qué host está relativamente más cerca. Los Snitches recopilan información sobre la topología de la red. Dependiendo del tipo de snitch que se utilice, conocerán la topología de red que está utilizando una persona y podrán direccionar las solicitudes de manera eficiente.

### Anti entropía

La anti entropía es un mecanismo por el cual los sistemas distribuidos de bases de datos son capaces de asegurar que los datos que contienen sus nodos están en la última de las versiones.

DynamoDB utiliza los [árboles Merkle](https://es.wikipedia.org/wiki/%C3%81rbol_de_Merkle) para lograr este objetivo. Cassandra también lo utilizará, pero su implementación es algo distinta, ya que Cassandra mantiene un árbol Merkle por cada *column family*.

Procedimiento de escritura

Puesto que Cassandra es un sistema que proporciona alto rendimiento para las escrituras, parece interesante analizar este proceso con más detenimiento, describiendo con más detalle los aspectos más relevantes del proceso (Hewitt 2011).

Commit Log

El proceso de escritura en la base de datos debe pasar por varias fases antes de que sea efectivo. El primer paso antes de llevar a cabo la escritura es el de dejarla reflejada en el *commit log*. De esta manera se asegura la consistencia de los datos incluso en el caso de que suceda un problema y la operación se quede a medias.

Memtable

Después de haber escrito en el commit log, el dato es creado el memtable, es decir, en una estructura en memoria preparada para tal efecto. Cuando el número de objetos contenidos en un memtable supera el límite impuesto, se vuelcan los datos a disco y se crea un nuevo memtable. Una vez que esta operación se ha completado, se informa al *commit log* de que la operación ya ha sido finalizada correctamente, para que elimine esa entrada de sus registros.

SSTable

Cuando las *memtables* se vuelcan a disco, éstas lo hacen sobre las *SSTables*. Estas tablas son inmutables y no pueden ser modificadas por la aplicación. Todas las escrituras en Cassandra se realizan de forma secuencial. Es por esto por lo que el sistema experimenta un tan alto rendimiento en lo que a escrituras se refiere. No se requieren de lecturas previas o posicionados aleatorios en disco, tan solo escribir de forma secuencial.

Tombstones

Los [tombstones](https://en.wikipedia.org/wiki/Tombstone_(data_store)) son elementos que se utilizan en Cassandra cuando se quiere eliminar un objeto del sistema (su significado literal en castellano es “lápida”). En lugar de eliminar físicamente la información del clúster, el procedimiento que lleva a cabo es el de “marcar” la información como (o mejor dicho para ser) borrada y eso se escribe en SSTables. Esta marca de borrado, junto con todos los datos a los que esta marca está haciendo referencia, serán borrados la próxima vez que se ejecute *el proceso de compactación*.

Compactación

La compactación es un proceso de agrupación de datos y liberado de información. Se produce cuando el número de SSTables en disco comienza a ser elevado. La compactación realiza una fusión de todas las SSTables que se han estado escribiendo a disco desde que la última compactación fue procesada: las claves son entrelazadas, las columnas son combinadas, los tombstones son eliminados y se recrean de nuevo los índices.

El objetivo de hacer esto es el de liberar espacio ocupado en disco innecesariamente, unificar los datos y recrear los índices para que la información pueda ser fácil y rápidamente accesible. Toda esta información ordenada y unificada es nuevamente escrita en una única SSTable.

Además, en Cassandra existe un límite de SSTables que un proceso puede consultar para encontrar el dato que busca. Si una clave es modificada frecuentemente, es probable que se alcance este límite. Realizar una labor de compactación cada cierto tiempo ayuda a mantener la velocidad y la estabilidad del clúster en general y de cada nodo en particular.

Filtros Bloom

Los filtros Bloom (llamados así en honor a su creador, [Burton Bloom](https://es.wikipedia.org/wiki/Filtro_de_Bloom)), son un mecanismo que emplea Cassandra para mejorar el rendimiento del clúster.

Los filtros Bloom son un tipo de algoritmos no deterministas que determinan si un objeto dado se encuentra dentro de un conjunto. Que estos algoritmos sean no determinísticos significa que nunca darán un falso negativo, pero que sí pueden dar un falso positivo. Es decir, en caso de que, como resultado de la aplicación del filtro, un elemento no resulte ser miembro de un conjunto, no hay opción a la duda, pero en caso de que sí lo sea, el siguiente paso debe ser ir a comprobarlo. Los filtros Bloom funcionan manteniendo en memoria un array reducido de información que representa toda una colección completa de datos. Este array será un resumen de los objetos que pertenecen a esa colección, y debe tener un tamaño tal que se pueda mantener siempre en memoria. Con este array en memoria y los filtros de Bloom, el objetivo es tratar de minimizar en la medida de lo posible que se realicen accesos innecesarios a disco.

SEDA

[SEDA](https://en.wikipedia.org/wiki/Staged_event-driven_architecture) es el acrónimo de **Staged Event Driven Architecture**, o lo que es lo mismo, Arquitectura Orientada a Eventos por Etapas. Cassandra implementa SEDA, una arquitectura propia de los servicios de Internet que tienen unas necesidades muy grandes a la hora de escalar (Welsh 2006).

En una aplicación típica, habitualmente para gestionar operaciones entrantes al sistema, la forma de proceder es: se crea un hilo que atiende esa petición entrante, el hilo hace el trabajo que tenga que hacer, devuelve el resultado de la petición al cliente y muere. Con SEDA esto no es exactamente así.

Con SEDA un hilo puede comenzar a atender una petición, después pasarle el trabajo a otro hilo que continúa realizando la parte que le corresponda de la operación, y finalmente podría delegar en otro hilo diferente la finalización de la misma.

SEDA propone ciertas bases:

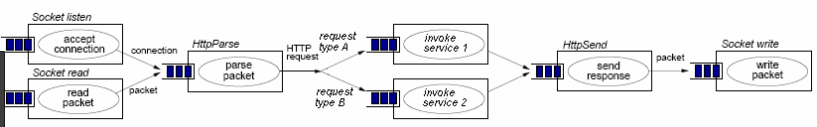
1. El trabajo es dividido en etapas o stages.

2. Se crean pools de hilos que serán los encargados de ejecutar las peticiones que

lleguen al sistema.

De este modo, las etapas serían la unidad mínima en la que se dividen las peticiones, y éstas se asocian a pools de hilos que ejecutan todas estas tareas. Puesto que cada tarea puede ser asociada a un pool de hilos distinto, esta técnica hace que Cassandra obtenga un aumento del rendimiento de manera considerable.

Además, los pools de hilos son conscientes del hardware del que disponen, y harán asignaciones de recursos entre los hilos que más lo necesiten. Es decir, si un hilo está haciendo un trabajo intensivo a nivel de entrada / salida, u otro esta ocupando mucho espacio en memoria principal, o hay un thread que está empleando muchos ciclos de CPU, etc., el pool conoce los recursos de los que dispone y sabe asignarlos entre los hilos en función de las necesidades de cada uno.



Modelo de Datos

Cassandra es una bases de datos NoSQL orientada a columnas. Su modelo de datos se basa en una serie de column families que a su vez se contienen columnas, que pueden darse ninguna, una o varias veces para un mismo elemento.

Cassandra, al igual que HBase, basa su modelo de datos en el paper sobre Bigtable que

liberó Google en 2006 (Chang, et al. 2006).

Cassandra y HBase poseen muchos aspectos comunes en lo que al modelo de datos se

refiere, por lo que aquí se intentará no repetir lo ya expuesto con anterioridad pero sí dar

una visión de lo que nuevo que aporta Cassandra.

**Clúster**

Cassandra es una base de datos que está diseñada e ideada para funcionar formando un clústers de máquinas. Es decir, si se quiere aprovechar al máximo todo el rendimiento y la funcionalidad que nos ofrece esta base de datos, no se deberá correr en un único nodo, sino de forma distribuida en varios nodos. Y cuanto más distribuido mejor: si es en varios racks mejor, y si además se ubican en varios CPDs, Cassandra dará un mejor rendimiento a la aplicación. El protocolo peer-to-peer hace que los nodos se comuniquen entre ellos para conocer su estado y repliquen los datos de forma transparente a diferentes usuarios.

**Keyspaces**

Un clúster está conformado por uno o más keyspaces. En un sistema de bases de datos relacionales, un keyspace correspondería a lo que conocemos como database. Cada keyspace tiene asociado un conjunto de atributos que definen el comportamiento global del keyspace.

Los atributos más importantes que se pueden configurar son:

* Factor de replicación.

El factor de replicación hace referencia al número de copias que debe guardarse de cada dato de la base de datos. El factor de replicación indicará cuántos nodos tendrán copia de los objetos de los que ese nodo sea responsable.

Este parámetro impacta directamente en el nivel de consistencia y de rendimiento del sistema. A mayor factor de replicación, más rendimiento y menos consistencia.

* Estrategia de localización de réplicas.

Este parámetro establece qué política se seguirá para alojar las réplicas en el anillo, y para decidir qué nodos serán los que alojen qué copias. Las políticas de localización de réplicas tienen en cuenta aspectos como la tomar en consideración el rack donde están alojados los servidores, los centros de datos, la topología de la red, etc.

**Column families**

Los keyspaces en Cassandra actúan como contenedores de column families, de igual manera que en el modelo relacional una database sería un contenedor de tablas. Cada column family contendrá varias columnas, y las filas se conforman de estas estructuras para almacenar la información final.

Cuando se escribe en una column family, se especifican valores para una o más columnas. Al conjunto de esos valores, junto con una clave identificativa, es lo que se le llama fila. En Cassandra, la estructura de datos es frecuentemente llamada hash cuatridimensional se estructura de la siguiente forma.

[Keyspace][ColumnFamily][Key][Column]

La clave key en este caso es una clave primaria formada por números, letras y símbolos especiales. La column family se llama “Hotel”, y cada una de las columnas de esa column family son las que aparecen con valores: name, address, city, etc.

**Columas**

Las columnas son las unidades mínimas que alojan información en Cassandra. La información que contiene el valor de cada columna está conformado por un array de bytes.

Además, en Cassandra no es necesario definir todas las columnas en el momento de diseñar la base de datos, pero sí las column families. Una vez éstas estén creadas, el resto de tablas se pueden crear en tiempo de ejecución, incluso aunque previamente no existieran.

**Supercolumnas**

Las supercolumnas son un tipo especial de columna. Ambos tipos de columnas son pares nombre-valor, pero mientras que una columna normal almacena el valor para esa columna, una super columna contiene un mapa de columnas, que estas ya sí, almacenan el valor en forma de array de bytes para esa columna.

Con las superscolumnas, Cassandra pasa de ser un hash de 4 dimensiones a serlo de 5, con la siguiente ordenación en sus dimensiones.

[Keyspace][ColumnFamily][Key][SuperColumn][Subcolumn]

Para definir una supercolumna, simplemente hay que generar una column family como si fuera de tipo super. De esta manera, este tipo de column families se pueden utilizar como si fueran un tipo de column family normal, únicamente habría que indicar explícitamente el nombre de la super columna.

Una cosa a tener en cuenta es que no se pueden generar índices de los datos almacenados en las súper columnas, sino que cada vez que se carga una súper columna en memoria, se cargan todas las columnas asociadas también.

# Diferencias entre MongoDB y Cassandra

| **Información editorial proporcionada por** [**DB-Engines**](https://db-engines.com/en/system/Cassandra%3BMongoDB) |
| --- |

| **Nombre** | **Cassandra** | **MongoDB** |
| --- | --- | --- |
| **Modelo de Base de datos** | Familia de columnas | Orientada a documentos |
| **Desarrollador** | Apache Software Foundation. | MongoDB, Inc. |
| **Primer Release** | 2008 | 2009 |
| **Licencia** | Open Source | Open Source |
| **Lenguaje** | JAVA | C++ |
| **Sistemas Operativos Soportados** | BSD - Linux - OS X - Windows | Linux - OS X - Solaris - Windows |
| **Esquema de Datos** | Polimórfico | Polimórfico |
| **SQL** | SQL-like DML and DDL statements (**CQL**) | no |
| **Lenguajes de programación soportados** | C# C++ Clojure Erlang Go Haskell Java JavaScript Perl PHP Python Ruby Scala | Actionscript C C# C++  Clojure ColdFusion D Dart Delphi Erlang Go Groovy Haskell Java JavaScript Lisp Lua MatLab Perl PHP PowerShell Prolog Python R Ruby Scala Smalltalk |
| **Scripts (del lado del server)** | no | si |
| **Triggers** | si | no |
| **Particionamiento** | Sharding | Sharding |
| **Replicación** | Factor de replicación | Maestro-Esclavo |
| **Foreign Key** | no | no |
| **Concepto de transacción** | no | no |
| **Concurrencia** | si | si |
| **Gestión de la seguridad** | Los permisos se pueden definir a nivel de objeto | Los permisos se definen mediante roles |
| **Ventaja competitiva** | No existe SPF | MongoDB proporciona un modelo de datos flexible, escalabilidad y alto rendimiento y disponibilidad. |
| **Aplicaciones** | Internet de las cosas (IOT), Aplicación de detección de fraudes, engines, catálogos de productos and playlists and aplicaciones de mensajerías. | Internet of Things (Bosch, Silver Spring Networks).  Mobile (The Weather Channel, ADP, O2).  Single View (MetLife)  Real Time Analytics (Buzzfeed, City of Chicago, Crittercism)  Personalization (Expedia, eHarmony, Gilt)  Catalogs (Under Armour, Otto)  Content Management (eBay, Forbes) |