Proyecto Fin de Ciclo G.S

Cassandra NoSQL

Nombre: Juan José López Roldán
C.F.G.S Administración de Sistemas Informáticos en Red
Promoción 2016-2018

Índice

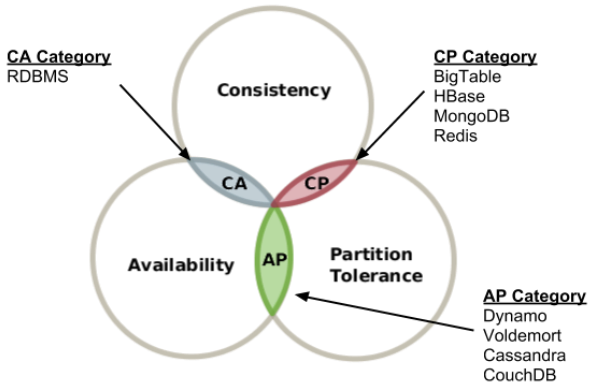
1. Introducción

Cassandra es una base de datos de software libre ya que pertenece a la Apache software foundation. Es una base de datos tipo NoSQL (El concepto de NoSQL se explicará más adelante) distribuida y basada en un modelo de almacenamiento de "Clave-Valor", está escrita en Java por ello es necesario de su uso para que funcione. Permite manejar con grandes volúmenes de datos de forma distribuida. Por ello su uso en el ámbito de Big Data es muy recomendable ya que Cassandra es capaz de manejar un tamaño de datos máximo de 2 billones de registros y en el caso de tener que manejar con más registros, se tienen que particionar para poder trabajar con ellos.

2. ¿Qué es NoSQL?

NoSQL es un término que describe las bases de datos no relacionales de alto desempeño. Las bases de datos NoSQL utilizan varios modelos de datos, incluidos los de documentos, gráficos, claves-valores y columnas. Las bases de datos NoSQL son famosas por la facilidad de desarrollo, el desempeño escalable, la alta disponibilidad y la resiliencia.

En el sector de NoSQL existen un amplio abanico de bases de datos, y dependiendo de la necesidad que tengamos tendremos que utilizar una base de datos u otra. A continuación aparece una imagen en la que se muestra las diferentes bases de datos y a que va orientada cada una de ellas, dependiendo de las necesidades deberemos elegir uno y otro gestor de base de datos (la imagen que aparece a continuación hace referencia al teorema CAP) .



Como se puede apreciar en la imagen existen tres cualidades en las cuales a la hora de elegir un gestor de base de datos, pueden darse un máximo de dos pero en ningún caso se dan las tres, por ello tendremos tres opciones a la hora de elegir una base de datos. Para que quede más claro lo veremos en el siguiente ejemplo.

- En el caso que deseemos **consistencia** (quiere decir que se vea el mismo dato en caso de que se realice una transacción del mismo, en el nodo o los nodos para que no exista incoherencia de los datos) y **disponibilidad** (que esté el nodo/s de la base de datos siempre disponible), tendremos que elegir las bases de datos relacionales tales como MySQL, Oracle, etc.
- En el caso que deseemos **consistencia** (quiere decir que se vea el mismo dato en caso de que se realice una transacción del mismo, en el nodo o los nodos para que no exista incoherencia de los datos) y **tolerancia a fallos** (quiere decir que el sistema sigue funcionando a pesar de que se caiga un nodo o varios nodos), tendremos que elegir bases de datos como MongoDB, Redis, etc.
- Pero si surge el caso en el cual necesitamos **tolerancia a fallos** (quiere decir que el sistema sigue funcionando a pesar de que se caiga un nodo o varios nodos) y **disponibilidad** (que esté el nodo/s de la base de datos siempre disponible), tendremos que elegir **Cassandra** como base de datos.

3. ¿Por qué aparecen los sistemas NoSQL?

Las base de datos relacionales como MySQL, Oracle o PostgreSQL no tienen nada malo ya que están orientadas a un amplio sector, pero tienen un fallo y es que con la llegada de los servicios como los servicios en nube, ofrecen problemas de alta escalabilidad a la hora de manejar los datos. Si bien los modelos relacionales se pueden adaptar para hacerlos escalar incluso en los entornos más difíciles, no son los adecuados ni tampoco es la técnica más apropiada para ello.

Entonces este problema se resuelve creando una estructura de almacenamiento más versátil (NoSQL), aunque sea a costa de perder cierta funcionalidades como las transacciones que engloban operaciones en más de una colección de datos, o la incapacidad de ejecutar el producto cartesiano de dos tablas (más conocido como JOIN) teniendo que recurrir a la desnormalización de los datos (por ello en Cassandra no se puede realizar JOIN).

4. ¿Qué puntos diferencian NoSQL de las Bases de datos relacionales?

Las diferencias que existen entre NoSQL frente a las bases de datos relacionales son las siguientes:

Base de datos relacional	Base de datos NoSQL
Soporta potente lenguaje de consulta.	Soporta lenguaje de consulta muy simple.
Tiene un esquema fijo.	No posee un esquema fijo.

Sigue ACID (atomicidad, coherencia, aislamiento y durabilidad).	Es solamente “con el tiempo coherente”.
Soporta transacciones.	No es compatible con las transacciones.

5. Pero, ¿en que es la diferencia realmente?

En el caso de que no quede claro la diferencia que existe, podemos resumir que las características que existen son las tres siguientes:

- Ausencia de esquema en los registro de datos.
- Escalabilidad horizontal sencilla.
- Velocidad (en el caso de Cassandra las consultas son veloces ya que los datos a obtener son muy pocos por que se usa para consultas simples).

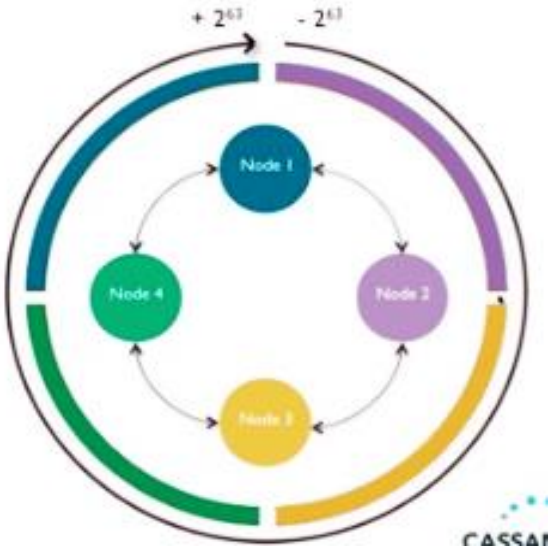
6. Arquitectura y características de Cassandra

Cassandra nos proporciona tolerancia a fallos y disponibilidad, pero a cambio de ser eventualmente consistente, tal y como se define en el teorema CAP (ya que la actividad de inserción de datos no es su fuerte, si no su consulta de información de forma rápida), que ha sido explicado anteriormente. El nivel de consistencia puede ser configurado, según nos interese incluso a nivel de query, dicha configuración está repartida de la siguiente forma:

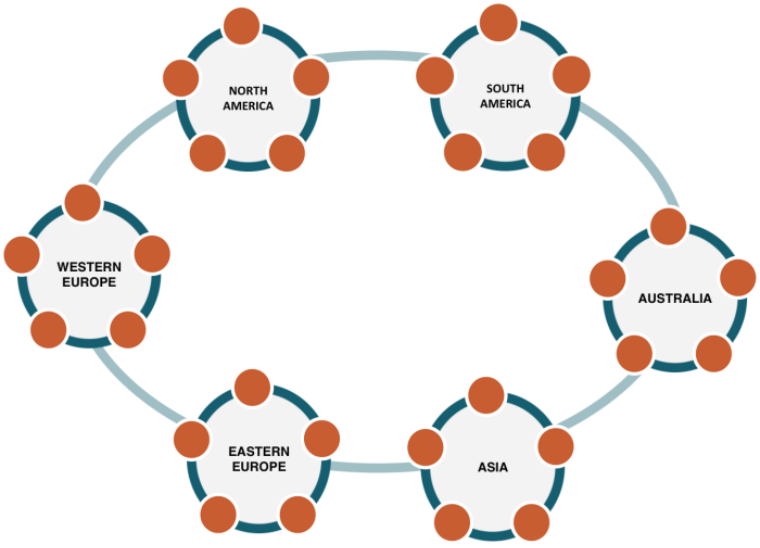
- Escala Distribuida: lo que quiere decir que la información está repartida a lo largo de los nodos del cluster. Además ofrece alta disponibilidad, de manera que si alguno de lo nodos se cae el servicio no se degradará.
- Escala linealmente: lo que quiere decir que el rendimiento de forma lineal respecto 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.
- Escala de forma horizontal: lo que quiere decir que podemos escalar nuestros sistema añadiendo nuevos nodos basados en hardware commodity de bajo coste.

Implementa una arquitectura Peer-to-Peer, lo que quiere decir es que todos los nodos tienen la misma importancia, por lo cual el patrón de maestro-esclavo es inexistente, así que ningún nodo tendrá alguna información de más con respecto al resto. De esta forma cualquier nodo puede tomar el rol de coordinador de una query. Será el driver el que decida qué nodo quiere que sea el coordenador (esta gestión la realiza internamente cassandra).

Los datos son repartidos a lo largo del cluster en base a un token único calculado para cada fila por una función hash. Los nodos se reparten equitativamente el rango de tokens que va de -2^{63} a 2^{63} , esto define el nodo primario.Internamente Cassandra replicará los datos entre los nodos con la política que le definamos, por ejemplo definiendo el factor de replicación.



Además soporta el concepto de data center para agrupar los nodos lógicamente y tener los datos más cerca del usuario.



7. Componentes de Cassandra

Una vez que ya sabemos la arquitectura que tiene Cassandra y como funciona internamente, además de las posibilidades que nos brinda vamos a conocer de forma resumida cómo está compuesta para así ser consciente en todo momento qué componentes tiene esta base de datos.

- **Cluster:** es un componente que contiene uno o más centro de datos (*datacenter*).
- **Centro de datos (*datacenter*):** se trata de una colección que almacena los nodos relacionados.
- **Nodo:** es el lugar donde se almacena los datos de Cassandra.
    - **Commit log:** es un fichero en donde se almacena la información sobre los cambios en los datos. Sirve para recuperar los datos en caso de una fallo en el sistema.
    - **MemTable:** estructura de almacenamiento en memoria. Contiene los datos que aún no han sido escritos en un SSTable.
    - **SSTable:** es un fichero que almacenan los datos escritos en disco. Cada fichero *SSTable* es inmutable (que no cambia) una vez creado.
- **About internode communications (gossip):** es el protocolo de comunicación peer-to-peer para descubrir y compartir información sobre la localización y estado de los nodo en un cluster de Cassandra.
- **Partitioner:** este componente determina cómo se distribuyen los datos entre los nodos (las copias entre los nodos).
- **Replica placement strategy:** define la estrategia a seguir para almacenar las copias de los mismo datos en diferentes nodos, de forma que se aseguren la accesibilidad y la tolerancia a fallos. Se pueden definir diferentes estrategias. Hay que tener en cuenta que no existen copias principales ni secundarias, todas las copias, incluida la primera, con réplicas.
- **Snitch:** define la topología que utilizan las estrategias de replicación para colocar las réplicas y dirigir las consultas de forma eficiente.

8. Funcionamiento de Cassandra

8.1. Proceso de escritura

Una vez teniendo claro los componentes principales de vamos a comprobar como sería el proceso de escritura de un dato en Cassandra.

1. En el momento en el que se realiza la petición de escritura, se envía la información a **Memtable** que almacenará las escritura de cada familia de columna (algo parecido a una tabla en una BD relacional) de forma temporal en memoria ram (de ahí que en Cassandra las escrituras son “baratas”).

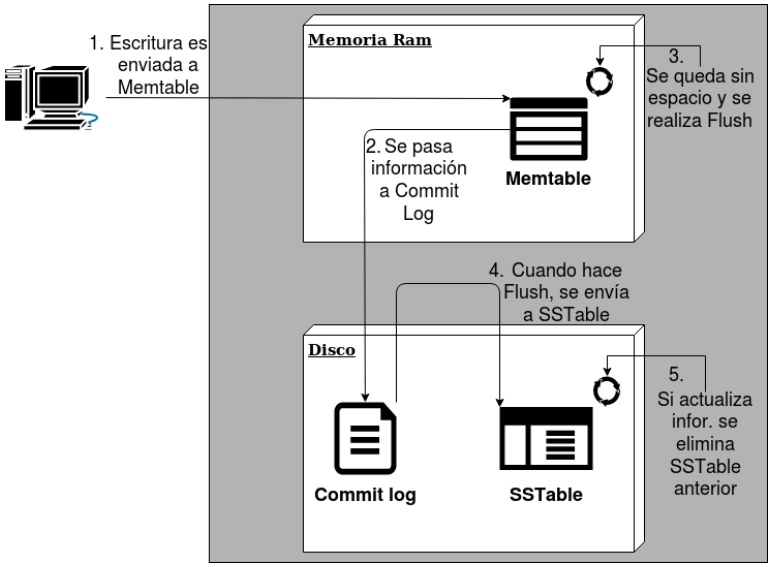
2. Acto seguido procederá a realizar la escritura en cada nodo, el **commit log** almacenará toda la actividad e información de escritura para garantizar la durabilidad de los datos. De esta forma si el sistema se apaga se puede recuperar las escrituras ya que el **commit log** se encuentra en disco.

3. En el momento en el que la **MemTable** se quede sin espacio, se elimina los datos, liberando así la memoria (este concepto se denomina como un **Flush** de la **MemTable**).

4. Los datos que han sido eliminados en la fase de **Flush** de la **MemTable** se escriben en disco, en ficheros llamados **SSTables,** los cuales son inmutables y no se vuelve a escribir en ellos después de volcar los datos. Por ello, los datos de una misma partición pueden estar repartidos en varias SSTables.

   Este es el motivo por el cual las lecturas masivas de información son más "caras" o "costosas" ya que la lectura suele requerir el acceso y búsqueda en varias **SSTables**.

5. Una vez que ya esté todo escrito en las SSTables, de forma periódica Cassandra trata de reducir el espacio de las SSTables**,** eliminando aquellas SSTables que se han quedado obsoletas por que han sido escritas con datos actualizados, pero este dato actualizado lo escribirá en otra SSTable y no en la misma ya que son inmutables. Este proceso de eliminación de ficheros SSTable se denomina **compactación**.



8.1.1. Datos a tener en cuenta en la escritura de datos

Hay que tener en cuenta que las escrituras con "costosas" o "caras" en el momento en el que la información se escribe en las **SSTables**, pero si esta en memoria en el **Memtable** su escritura y acceso es mas rapida y menos costosa ya que inicialmente se escribe en memoria.

¿Siempre se puede escribir aunque haya algún nodo caído?

No, porque se puede dar el caso en el que se realice una escritura y en el nodo en el que se realice la réplica, se encuentre caído pero es poco probable que ocurra.

¿Es posible realizar una escritura de una fila existente?

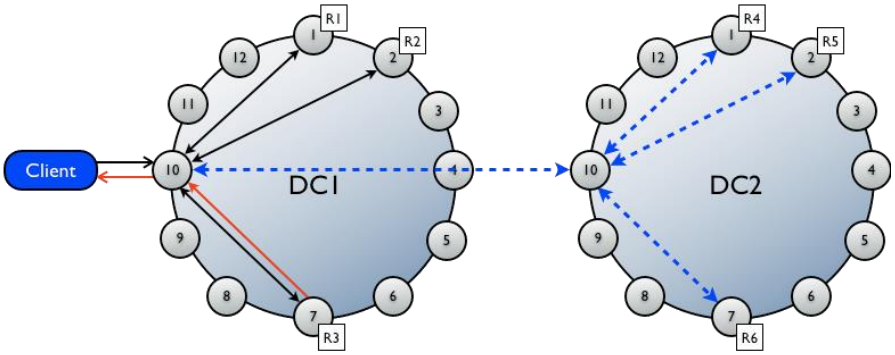
Si es posible, ya que cuando se realiza una escritura, Cassandra no realiza la búsqueda si existe o no el dato en el momento de la escritura para darnos el aviso (*Primary Key* existente en la BD), si existe ese dato lo que realizará es sobreescribir el dato (realizando un *upsert*).

- Si el dato está en *Memtable*, se sobrescribe, siendo el valor de *Primary Key* único para cada fila.
- En el caso que se encuentre en el Commit log, se almacenan los logs de todos los insertados de datos, por lo que habrá tantos logs como escrituras se hagan (no se sobrescriben, si tienen la misma *Primary Key*).

¿En un cluster como se realiza el proceso de escritura en el nodo?

El nodo coordinador envía una petición de escritura a todas las réplicas que contengan la fila a ser escrita (que de forma aleatoria le va correspondiendo el rol de coordinador a cada nodo dependiendo de la configuración que le hayamos predefinido a la base de datos).

- Los datos se escriben en todos los nodos disponibles que reciben la petición.
- El nivel definido de consistencia de la escritura determina cuántos nodos han de responder a la petición para determinar que la escritura se ha realizado con éxito.
- El éxito de la escritura significa que el dato ha sido escrito en el *Commit log* y en el *MemTable*.



8.2. Proceso de borrado de datos

Cuando una fila es eliminada, no se borra del disco (*SSTables* inmutables), sino que la marca como eliminada (**tombstone**) y durante la **compactación**(descrita anteriormente), las filas marcadas son eliminadas definitivamente del disco.

En el caso en el que un nodo contenga la fila a ser eliminada y se encuentra caído en ese momento, aún tiene la posibilidad de recibir la orden de eliminación si se recupera en un intervalo de tiempo predefinido en su configuración, este intervalo de tiempo se denomina **Tiempo de Gracia.**

Si no se recupera en ese tiempo predefinido, la fila no será marcada como eliminada, por lo que pueden existir inconsistencias en las réplicas. Por ello se recomienda que los administradores realicen cada cierto tiempo tareas de mantenimiento en Cassandra.

Además de ello los datos pueden tener una **fecha de caducidad** (denominado **LLTS**) a partir de la cual son marcados por Cassandra como eliminados.

8.3. Proceso de lectura de datos

El proceso de lectura se realiza de diferentes formas dependiendo de la localización de la información, siendo las dos siguientes formas de acceso a la información:

- Si los datos se encuentra en la *MemTable*, se procede a la recuperación de ese dato de forma rápida y eficaz y se envía.

- Pero si el dato no se encuentra en la *MemTable*, procederá a acceder a las *SSTable* para encontrar esa información. El problema que existe en esta forma de acceder a la información es la lentitud que ofrece a la hora de encontrar el dato, ya que si empieza a buscar el dato y no se ha realizado la tarea de **compactación** recientemente, es probable que haya que leer en varias *SSTables* para recuperar todas las filas requeridas.

Dentro de estas dos formas de obtener la información, existen 3 tipos de lecturas en Cassandra, siendo las siguientes:

- **Direct read request**: el nodo coordinador contacta con un nodo que contenga la réplica requerida.

- **Digest request**: contacta con tantos nodos con réplicas como se determine en el nivel de consistencia. Se comprueba la consistencia de los datos retornados por el nodo contactado en el **Direct read request**.
    - Se envía la consulta aquello nodos que estén respondiendo “más rápido”.

    - Si existen inconsistencias, se consideran como válidos las réplicas con un *timestamp* más reciente. Estos son los retornados al cliente.

    - Una vez contactados los nodos determinados por el *consistency level*, se envía un *Digest request* al resto de réplicas, para determinar si son también consistentes.

- **Background read repair request**: en el caso de existir inconsistencias en las réplicas, las réplicas con un *timestamp* más antigua son sobrescritas con los datos de la réplica “más actual”.

9. Administración de Cassandra

9.1. Seguridad

La seguridad existente en Cassandra se encuentra a nivel de usuarios como *logins* con *password* y permisos de gestión administración vía *GRANT/REVOKE*, como es utilizada en los sistemas de bases de datos relacionales como por ejemplo ORACLE, a continuación aparece los privilegios que podemos gestionar y a los objetos que son aplicables.

Privilegios:

- ALL PERMISSIONS
- ALTER
- AUTHORIZE
- CREATE
- DESCRIBE
- DROP
- EXECUTE
- MODIFY
- SELECT

Objetos a los que se le aplican:

- ALL FUNCTIONS
- ALL FUNCTIONS IN KEYSPACES “nombre keyspace”
- FUNCTION “nombre de función”
- ALL KEYSPACE
- KEYSPACE “nombre keyspace”
- TABLE “nombre tabla”
- ALL ROLES
- ROLE “nombre rol”

Sintaxis a la hora de aplicar/revocar un privilegio:

Asignación de privilegios.

```
GRANT "privilegio a conceder" ON "objeto al que concede" TO "usuario/rol";
```

Revocación de privilegios.

```
REVOKE "privilegio a revocar" ON "objeto al que revocar" FROM
"usuario/rol";
```

Además de ello nos ofrece opciones de encriptación tanto entre clientes y clústeres como entre nodos. En el caso que deseemos un nivel mayor de seguridad en nuestro sistema Cassandra, tendremos que hacer uso de ***DataStax*** Enterprise que ofrece opciones avanzadas de seguridad, como autenticación externa, encriptación de tablas, y auditoría de datos.

9.2. Copias de respaldo

En el ámbito de copias de respaldo, Cassandra nos ofrece varias opciones de copias de seguridad o *Backups*. Es recomendable realizarlos de forma regular ante errores comunes como por ejemplo borrados accidentales.

Nuevamente en el caso que necesitemos tener un control de backups de forma gráfica e intuitiva, ***DataStax OpsCenter*** provee de una interfaz de visualización para su mantenimiento y restauración de copias de seguridad.

9.3. Consistencia de los datos

Ya sabemos que en Cassandra existen diferentes mecanismos para que no exista el riesgo de encontrar datos inconsistentes en los nodos pero aún así existe la posibilidad de que se generen inconsistencia en los datos por ello es recomendable seguir las siguiente tarea de forma rutinaria que nos permite disminuir la probabilidad de que ocurra:

- En el caso en el que un nodo caído recibe una petición de borrado que no puede ejecutar. El nodo se recupera tiempo después de que el "*tiempo de gracia*" de esa orden haya caducado, por lo que tiene datos inconsistentes que tienen que ser borrado para que los datos estén en la base de datos de forma correcta, ya que no se realiza ninguna consulta sobre esos datos, por lo que no hay posibilidad de recibir un *Read Repair Request* (porque si ya ese dato "existe" no se puede consultar).

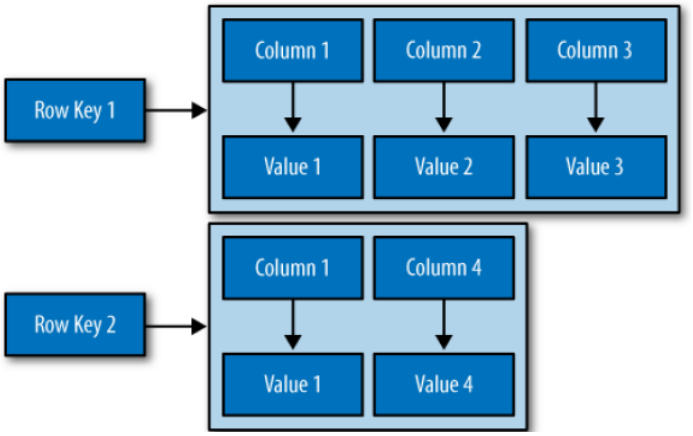
- En estos casos Cassandra ofrece una operación denominada ***repair***, que puede utilizarse por parte del usuario para asegurar que todos los nodos son consistentes (en otras palabras que no haya réplicas con diferentes valores).

10. Modelo de Datos de Cassandra

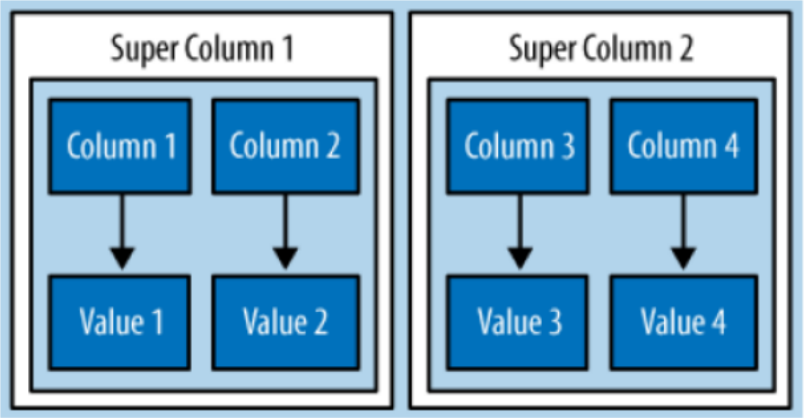
El modelo de datos que se utiliza en Cassandra es algo complejo de entender por cómo se organiza y reparte la información, como ya sabemos, la base de datos de Cassandra se organiza en un sistema de "Clave-Valor" enriquecido, lo que ofrece una visión totalmente diferente a lo que estamos acostumbrado a ver en los sistemas relacionales.

El modelado de datos existente en Cassandra está dividido en diferentes apartados, siendo los siguientes:

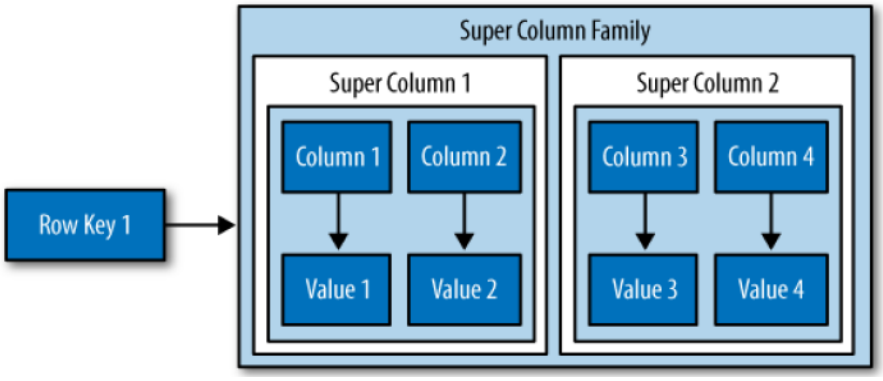
- **Columna (Column)**: es el elemento básico del sistema, es una estructura de tres campos que contienen: el nombre de la columna, el valor de la columna y una marca de tiempo.
    - La clave y el valor se almacenan como datos binarios y el timestamp indica la última vez que se actualizó la columna.
    - Los tres valores son proporcionados por el cliente de alto nivel (por la aplicación), incluido el *timestamp*. Por ello es necesario tener sincronizado el reloj del sistema con el reloj del clúster.
    - El *timestamp* se utiliza para la resolución de conflictos y así diferencias cuando se introdujo un dato de otro.
    - El *timestamp* puede ser de cualquier tipo pero la convención marca que sea el valor en microsegundos.
    - Dicha columna tiene asociada las ***filas (rows)*** que es el lugar en donde va a almacenado el valor que hace referencia al nombre de columna.
    - Y la información almacenada que hace referencia a esas columnas se le aplica un ***hash*** para generar una ***key (denominada row-key)*** que diferencia ese dato con el resto para que en el momento en el que se realice la búsqueda de ese dato, sea capaz de encontrarlo.
    - La ***row-key*** es el equivalente a la clave primaria del modelo relacional. y está compuesta por dos partes:
        - **Parte del particionado (partition key)**: el objetivo es identificar en la partición o el nodo en el que se encuentra esa fila/dato
        - **Parte del agrupamiento (clustering key)**: determina el orden físico en el que se almacenan las filas.



- **Súper Columna (SuperColumn)**: es un elemento que se encuentra compuesto por varias columnas (descrito anteriormente). El aspecto negativo que tiene la *súper columna* es que no aconsejan el uso de ella ya que afecta al rendimiento en la búsqueda de información ya que "desglosar" las columnas por lo cual tarda más en mostrar el resultado, así que por ello no se recomienda su usabilidad.

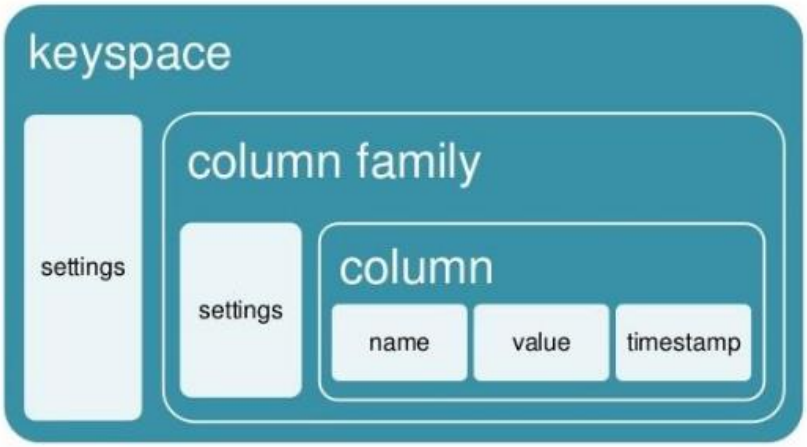


- **Familia de Columnas (ColumnFamily)**: sería un equivalente a una tabla en un esquema de datos *relacional*. Se trata de un contenedor para una colección ordenada de *columnas*. Cada entrada *(row)* se identifica y accede a ella mediante una *row-KEY* (como hemos explicado anteriormente).
    - Cada Familia de Columna es guardada en un fichero separado y es ordenado por la *KEY*.
    - Los tipos de columnas que almacena la *Familia de Columnas* es la siguiente:
        - *Standard Columns*
        - *Counter Columns*
        - *Collection Columns: SET, LIST, MAP*
        - *User-defined type*
        - *Tuple-type*
        - *Timestamp type*



- **Espacio de clave**: es un contenedor para la *familia de columnas*, más o menos el equivalente a una **base de datos (schema)** en el modelo relacional. Es una colección ordenada de una *familia de columnas o columnas.*
    - Los atributos básicos de un *Keyspace o Espacio de Clave son:*
        - **Replication factor**: cuánto quieres asignar en rendimiento a favor de consistencia.
        - **Replica placement strategy**: indica cómo se colocan las réplicas en el anillo:
            - *SimpleStrategy (solo un centro de datos)*.
            - *NetworkTopologyStrategy (varios centro de datos)*.
    - Cassandra ofrece soporte para particionado distribuido de datos.
        - **RandomPartitioner**: ofrece un correcto balanceo de carga.
        - **OrderPreservingPartitioner**: nos permite ejecutar consultas de rangos, pero exista más trabajo eligiendo node *tokens.*

    - Además de ello Cassandra tiene consistencia reconfigurable.
        - En el momento de escritura, *consistency level* determina en cuántas réplicas se debe escribir para confirmar a la aplicación cliente.
        - Y al leer, también se especifica cuántas réplicas deben responder para retornar los datos a la aplicación cliente.



- **Cluster**: es un conjunto de máquinas o nodos que ejecutan Cassandra y proporcionan al sistema escalabilidad horizontal, de manera transparente para el cliente.
    - Los datos en Cassandra se guardan en el Clúster (*"Ring"*) donde se asignan datos a los nodos dentro de un anillo.
    - Un nodo tiene réplicas para diferentes rangos de datos.
    - Si un nodo se cae, su réplica puede responder.

- El protocolo utilizado en Cassandra es ***P2P*** que hace que los datos se repliquen entre nodos acorde a un factor de replicación definido.

10.1. Objetivos del modelado de datos

1. Distribuir los datos por todo el clúster.
    - Es deseable que cada nodo del clúster tenga un volumen de datos similar (equilibrio).
    - Como las filas filas se distribuyen en base a la *partition key* es conveniente escoger una clave primaria adecuada para la aplicación que se trate.
2. Minimizar el número de particiones a leer.
    - Las particiones son grupos de filas que comparten la misma *partition key*.
    - Cuantas menos particiones tengan que ser leídas, más rápida será la lectura.

10.2. Comparación entre base de datos relacional y Cassandra

A continuación aparece una pequeña tabla en la cual se muestra las posibles similitudes entre Cassandra y los sistemas de bases de datos relacionales del mercado.

Base de datos relacional	**Cassandra**
Base de datos	Keyspace (Espacio de Clave)
Tabla	Column Family (Familia de columnas)
Clave primaria	Row Key

10.3. Índices en Cassandra

10.3.1. Índices Primarios

Los índices primarios son únicos por cada fila ya que cada fila es asignada a cada nodo por el *partitioner* y él *replica placement strategy* en base a su índice primario. Como cada nodo conoce el rango de valores del índice primario que almacena, las consultas pueden realizarse de forma eficiente escaneando únicamente los índices de las réplicas de las filas buscadas

10.3.2. Índices Secundario

Este tipo de índices son permitidos en Cassandra ya que se utilizan sobre las columnas y se implementan como una tabla oculta, separada de la tabla que contiene los valores originales.

Por ello no es recomendable usar ese tipo de índices para consultas que requieran comprobar un gran volumen de datos para finalmente retornar un pequeño conjunto de resultados

11. Lenguaje CQL

Cassandra Query Language (**CQL**) es el lenguaje que utiliza para realizar el acceso a la base de datos. En Cassandra los datos están desnormalizado por lo cual como hemos comentado antes los JOIN no puede ser posible realizar como ocurre en las bases de datos relacionales.

Para poder interactuar con Cassandra mediante **CQL** tendremos que introducir a través de la shell de la propia base de datos el comando **cqlshell**, aunque también podemos utilizar herramientas gráficas o a través de los driver soportados por múltiples lenguajes de programación.

12. Trabajando con CQL

A la hora de trabajar con CQL, vamos a utilizarla en su versión CQL3. Para empezar lo más llamativo que tiene Cassandra que su lenguaje con el que interactuamos en cuanto a sintaxis se asemeja muchísimo a los sistemas de bases de datos relacionales. Pero no debemos olvidar que aunque se parezca sintácticamente, en su interior el funcionamiento es completamente diferente.

12.1. Consultas CQL

Una de las particularidades que posee Cassandra con su lenguaje CQL es que de cara a las consultas que se realizan en ella se hacen de forma muy reducida en cuanto a la condicion de busqueda se refiere, de esta forma las consultas a realizar son del siguiente tipo (las condiciones tienen que ser especificadas detrás del ***WHERE***):

- Como hemos comentado anteriormente no podemos realizar consultas con *JOIN*.
- Podemos especificar cláusulas de igualdad a un objeto concreto, como por ejemplo:

```
SELECT * WHERE Nombre = 'Pepito';
```

- Podemos especificar cláusulas de igualdad a varios objetos con el valor **“IN”**, como por ejemplo:

```
SELECT * WHERE Nombre = 'Pepito' and numero IN (2, 3, 4);
```

- Y también podremos realizar una query cuya cláusula sea a través de rangos, como por ejemplo:

```
SELECT * WHERE Nombre = 'Pepito' and numero >=6 and numero <=10;
```

- Unas de las consultas que no podremos realizar en Cassandra, son consultas de negación o consultas que indiquen, por ejemplo que el nombre empiezan por “P” o que no empiece por “P”, ya que en todo momento en Cassandra lo que se está evitando es lo escaneos para que en el tiempo de consulta tenga la menor demora posible.
  A continuación vamos a ver en un par de ejemplos, que tipos de consultas no se puede realizar por las limitaciones que ofrece Cassandra.

```
select * from personas where apellidos like 'M%';
```

```
select * from personas where cod_hospital !=2;
```

- Al igual que ocurre en las bases de datos relacionales, podremos realizar consultas con criterios de ordenamiento como es la utilización de un **order by (**dicho ***order by*** se puede realizar si se hace sobre la ***clustering key,*** y que en el ***where*** aparezca la ***partition key*****)**.

```
select idusuario from personas where idusuario = '0s66' order by
nombre;
```

- Una de las opciones que nos ofrece Cassandra en el momento de realizar la consulta es la posibilidad de definir el número de filas que deseemos que nos retorne en la consulta, para ello tendremos que introducir el siguiente comando.

```
select idusuario from personas where idusuario = '0s66' order by
nombre limit n;
```

- Y por último una de las opciones que podemos utilizar es la opción denominada Allow Filtering que nos permite realizar las consultas en las que en el WHERE no haya condiciones en la partition key (aunque no se recomienda su uso, a menos que sea estrictamente necesario que seria en el caso de acceso secuencial a los datos).

```
select idusuario from personas where mes>9 allow filtering;
```

12.2. Creación, Modificación y Eliminación de Objetos con CQL

Como hemos explicado anteriormente, **CQLSH** es la consola estándar para interactuar con Cassandra a través de CQL, además de poder realizar las consultas como hemos visto en el punto anterior también podremos realizar la creación de objetos como por ejemplo la creación de un Keyspace o la creación de una tabla (que es el equivalente a una familia de columnas con sus respectivas columnas en Cassandra).

12.2.1. Crear, alterar y eliminar un Keyspace

Como hemos aprendido anteriormente, un keyspace es el contenedor más exterior y debería ser creado antes que cualquier cosa, así que para poder realizar la creación de un keyspace tendremos que introducir el siguiente comando:

```
CREATE KEYSPACE IF NOT EXISTS "nombre keyspace" WITH replication = {
'class': 'SimpleStrategy', 'replication_factor': '1'} AND durable_writes =
true;
```

El comando anterior está indicando que se realice la creación de un keyspace con el nombre deseado en el caso que no exista (si existe, no lo creará), además de ello realizará la replicación en un solo nodo, además de que estamos indicando la estrategia de cómo gestionará la réplica de los datos(este apartado se explica a continuación) y utilizará la política durable_writes.

Hay un dato a tener en cuenta y es que tenemos que definir como deseamos realizar la réplica de los datos, dependiendo de la opción que pongamos, nos realizará un tipo de replica u otro, existen dos opciones de replicado que se explican a continuación.

- **SimpleStrategy**: se define cuando deseamos que los datos se almacenen en un solo data center.
- **NetworkTopologyStrategy**: se define si se planea desplegar el cluster en múltiples data centers.

En el caso que deseemos realizar desde fuera de la base de datos (fuera de la shell de CQL) tendremos que realizarlo con el siguiente comando:

```
cqlsh -u "usuario" -p "password" localhost -e "CREATE KEYSPACE IF NOT
EXISTS "nombre keyspace" WITH replication = { 'class': 'SimpleStrategy',
'replication_factor': '1'} AND durable_writes = true;"
```

Si en algún momento deseamos realizar la modificación de ese keyspace tendremos que introducir el siguiente comando que nos permite alterar las características del keyspace:

```
ALTER KEYSPACE "nombre keyspace" WITH REPLICATION = { 'class' :
'SimpleStrategy', 'replication_factor' : 3 };
```

Con esta modificación estamos indicando que deseamos ajustar el número de réplicas disponibles de este keyspace, cambiando el valor de 1 a 3.

En el momento que deseemos usar este keyspace lo único que tendremos que realizar es la introducción del siguiente comando.

```
USE "nombre de keyspace";
```

Y si deseamos realizar el borrado de un Keyspace tendremos que introducir el siguiente comando.

```
DROP KEYSPACE "nombre de keyspace";
```

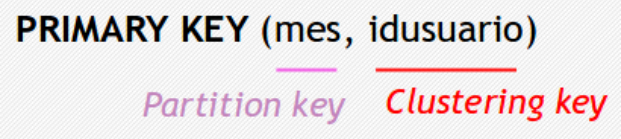
12.2.2 Crear, alterar y eliminar una Tabla

Como hemos visto anteriormente el concepto más equivalente en Cassandra de una Tabla en una base de datos relacional es una Familia de Columnas, que dicha familias de columnas almacenan una Super Columna o Columna, las cuales almacena las filas con los datos que son introducidos en la base de datos.

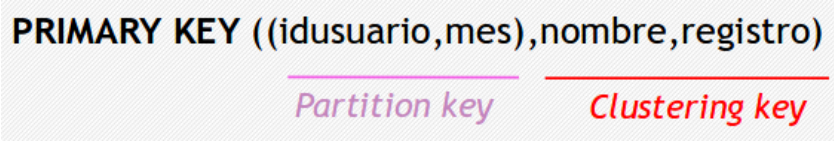
Para poder realizar la creación de una "tabla", tendremos que introducir el siguiente comando en el cual estamos indicando los tipos de campos de las columnas, además de indicar su correspondiente primary key.

```
CREATE TABLE personas (
      id text,
      name text,
      surname text,
      date time,
      PRIMARY KEY (id, name));
```

Es recomendable indicar una fecha de registro e indicar dos primary keys, para que la primera sirva para definir la partitionkey y la segunda sea para definir el clustering key y van en ese orden.



Aunque también podremos definir la primary key y la clustering key de forma compuesta (que cada una está compuesta por más de uno).



12.2.2.1 Tipos de Datos en las Tablas

Además de ello en el momento de la creación de la tabla podremos definir el tipo de dato que vamos a almacenar, por ello tendremos una amplia diversidad de opciones a elegir dependiendo del dato a introducir en esa columna, dichos tipos de datos se clasifican en los siguientes:

- Numéricos:
    - **Int**: enteros de hasta 32 bits.
    - **Bigint**: enteros de hasta 64 bits.
    - **Smallint**: enteros de hasta 2 bytes.
    - **Tinyint**: enteros de hasta 1 byte.
    - **Varint**: enteros de precisión arbitraria.
    - **Decimal**: decimales de precisión variable.
    - **Float**: coma-flotante de 32 bits.
    - **Double**: coma-flotante de 64 bits.
    - **Counter**: contador de enteros de hasta 64 bits.

- Texto:
    - **Ascii**: strings US-ASCII.
    - **Text**: strings UTF-8.
    - **Varchar**: strings UTF-8 de longitud variable.
    - **Inet**: strings que almacenan direcciones IPv4 o IPv6.

- Identificación:
    - **UUID**: tipo de dato que sigue el estándar UUID. Es único para cada fila insertada en una tabla. Se utiliza para identificar a las filas
    - **Timeuuid**: es un UUID que además sirve como timestamp único para cada fila.

- Fecha
    - **Date**: fechas desde el 1 de Enero de 1970 en formato yyyy-mm-dd, representada como string.
    - **Time**: hora, minutos y segundos en formato hh:mm:ss.sss, representado como string.
    - **Timestamp**: fecha y hora con precisión al milisegundo en formato yyyy-mm-dd hh:mm:ss.sss. Puede ser representada como string.

- Colección:
    - **List**: colección de uno o más elementos ordenados.
    - **Map**: colección con pares clave-valor.
    - **Set**: colección de uno o más elementos.

- Otros tipos:
    - **Blob**: almacena bytes expresados en hexadecimal.

En el momento que deseemos revisar la estructura de la tabla creada, tendremos que realizar la introducción del comando describe.

```
DESCRIBE "nombre tabla";
```

Y si deseamos alterar la tabla creada anteriormente tendremos que realizar la introducción del siguiente comando (en el ejemplo que aparece a continuación añadimos la columna email de tipo texto).

```
ALTER TABLE personas ADD email text;
```

Además de añadir columnas, podremos modificar el tipo de dato que es capaz de almacenar esa columna y como en el ejemplo anterior, lo mismo que añadimos columnas nuevas, podremos también eliminarlas. Los dos ejemplos que aparecen a continuación muestran cómo se realizaría la modificación del tipo de columna y la eliminación de la misma.

- Modificación de tipo de columna:

```
ALTER TABLE personas ALTER email TYPE varchar;
```

- Eliminación de la columna.

```
ALTER TABLE personas DROP email;
```

En el momento que lo hayamos añadido podremos comprobar si se ha realizado correctamente con el comando **describe**.

Por último si deseamos realizar la eliminación de una *TABLE* tendremos que realizar la introducción del siguiente comando.

```
DROP TABLE "nombre tabla";
```

12.2.3. Insertar y Actualizar Datos

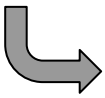
En el caso en el que deseemos insertar datos en una tabla concreta, tendremos que realizarlo de forma similar como lo hacemos en los sistemas de bases de datos relacionales, tal y como aparece a continuación.

```
INSERT INTO personas (id, name, surname, email) VALUES ('001', 'juan jose', 'lopez', 'juanjose@correo.es')
```

Un dato de interés a tener en cuenta es que Cassandra por defecto, no comprueba la unicidad de la Primary Key. Si existe otro dato con el mismo valor que el que se está insertando, el dato antiguo será sobreescrito, realizando así una actividad conocida como **UPSERT.**

Por ejemplo en el caso que deseemos realizar el insert que aparece a continuación sobreescribirá la información de juan josé por antonio, por que antonio que es el que se está insertando, tiene la misma primary key que juan josé.

```
INSERT INTO personas (id, name, surname, email) VALUES ('001', 'antonio', 'jimenez', 'antonio@correo.es');
```



001	juan jose	lopez	juanjose@correo.es
002	pepe	dominguez	pepe@correo.es
003	miguel	rivas	miguel@correo.es

Además del dato de interés anterior tenemos diferentes opciones que nos ofrece Cassandra a la hora de realizar la inserción de datos, siendo las siguientes opciones:

- **USING TTL**: esta opción nos permite realizar un insert por tiempo limitado, dicho tiempo se define en segundo y en el momento en el que se cumpla ese tiempo, Cassandra automáticamente eliminará ese dato introducido.

```
INSERT INTO personas (id, name, surname, email) VALUES ('001', 'juan jose', 'lopez', 'juanjose@correo.es') USING TTL 87900;
```

- **USING TIMESTAMP**: esta opción está indicada en microsegundos y permite realizar un insert en el cual si no se usa la el tiempo especificado en microsegundos , se usará el tiempo en el que se hizo la escritura.

```
INSERT INTO personas (id, name, surname, email) VALUES ('001', 'juan jose', 'lopez', 'juanjose@correo.es') USING TIMESTAMP 123456789;
```

- **IF NOT EXISTS**: esta opción permite evitar la inserción de esa fila en el caso que exista otra fila con la misma *Primary Key* (evitando un ***UPSERTS***). Existe el problema que disminuye el rendimiento de dicho *insert* ya que va comprobandolo fila por fila.

Y si se da el caso de tener que actualizar unos datos existentes para que aparezcan correctamente en la base de datos tendremos que introducir el siguiente comando para actualizar el dato correctamente.

```
UPDATE personas SET email='correojuanjose@correo.com' WHERE id='001';
```

12.2.4. Eliminación de datos

En el caso de la eliminación de registros en una tabla concreta se puede realizar con la cláusula DELETE, utilizándose así de la misma forma que en los sistemas de bases de datos relacionales, pero en el caso de Cassandra puede eliminar la fila o el contenido de una columna concreta, a continuación se ven unos ejemplos que nos permitirá entender mejor como va todo el tema de la eliminación de datos.

- Esta opción elimina el contenido de la columna "nombre" de la fila con el "id=001"

```
DELETE nombre FROM personas WHERE id='001';
```

- Esta segunda opción nos permite eliminar la fila con el "id=001"

```
DELETE FROM personas WHERE id='001';
```

12.2.5. Creación y eliminación de Índices

Al igual que ocurre en los sistemas de bases de datos relacionales, podremos realizar a creación de índices en Cassandra. Dichos índices se pueden aplicar sobre columnas o tablas, para su creación se tiene que introducir el siguiente comando.

```
CREATE INDEX idx_nombreusuario ON personas(nombre);
```

Y si deseamos eliminar dicho index que hemos creado, solamente tendremos que introducir el siguiente comando, que nos permitirá quitarlo de nuestra base de datos Cassandra, para ello vamos a ver un ejemplo de ese comando.

```
DROP INDEX  idx_nombreusuario;
```

12.2.6. Otros elementos que pueden ser creados en Cassandra

Además de todas las posibilidades que nos brinda Cassandra explicadas anteriormente, podremos realizar la creación de diferentes objetos que son los que nos aparece a continuación.

12.2.6.1 TRIGGERS

- Creación/Eliminación de trigger: podremos realizar la creación de triggers o eliminado de triggers con los siguientes comandos.
    - Creación:

```
CREATE TRIGGER IF NOT EXISTS trigger_name ON table_name USING 'java_class';
```

    - Eliminación:

```
DROP TRIGGER trigger_name;
```

12.2.6.2. USERS

- Creación/Eliminación de usuarios: podremos realizar la creación o eliminado de usuario para así gestionar la base de datos y su contenido, todo ello se realiza con los siguientes comandos.

    - Creación:

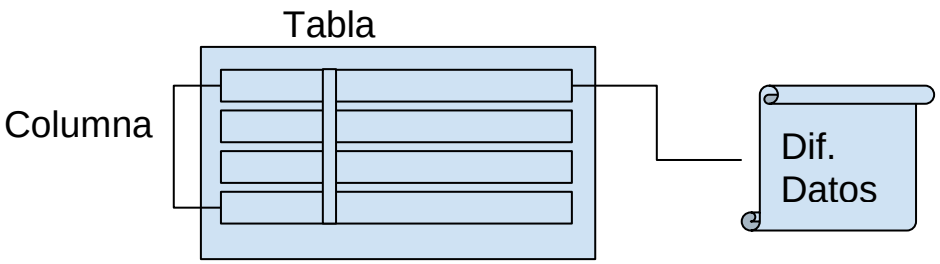
```
CREATE USER IF NOT EXISTS "nombre usuario" WITH PASSWORD 'password usuario' NOSUPERUSER | SUPERUSER;
```

    - Eliminación:

```
DROP USER "nombre usuario";
```

12.2.6.3. TYPE

Otro de los elementos que puede ser administrado por Cassandra, son los datos ***TYPE***. Estos datos ***TYPE*** no son más que espacios proporcionados al usuario (*cuya creación y gestión es parecida a la de una tabla*) que puede crear y usar tipos de datos definidos por él mismo. Puede crear un tipo de datos para manejar múltiples campos. Este tipo de componente se suele utilizar para asignarle a una columna de una tabla, que el tipo de columna sea de colección (*collection*) para que sean almacenados todos esos datos en un solo registro.



- Creación/Eliminación de TYPE: para realizar la creación o eliminación de los TYPE tendremos que introducir los siguientes comandos.
    - Creación:

```
CREATE TYPE direcciones (
  calle text,
  ciudad text,
  cod_postal int,
  movil set<text>
);
```

    - Eliminación:

```
DROP TYPE direcciones;
```

13. Cuándo Cassandra es la mejor solución

Una vez que ya hemos conocido a fondo cómo funciona Cassandra y cómo se interacciona con sus datos, además de ver diferentes cosas de interés relacionada con esta base de datos tan peculiar, vamos a plantearnos cómo solucionar el problema de cómo satisfacer nuestras necesidades de cara a si es recomendable usar esta base de datos o no, ya que hay que conocer a fondo el problema y tener una visión muy técnica de las diferentes soluciones que se pueden adoptar. Por eso, Cassandra puede ser la mejor solución, la peor o una más, dependerá de muchos factores que vamos a intentar resumir a continuación.

Para elegir Cassandra como la solución a adoptar nos tenemos que fijar en las características que hemos explicado anteriormente y ver si encajan con nuestras necesidades. Los casos que aparecen a continuación son los que permitirán decantarnos por Cassandra.

- Si nuestro caso es que necesitamos una base de datos que nos permita una escalación lineal con alto rendimiento y además de ello ofrecer alta disponibilidad. Esta base de datos es la ideal ya que encaja con sus grandes características.
- Siendo el caso que necesites una arquitectura distribuida y la manera en que se almacenan los datos va a permitir tener una alta disponibilidad de la base de datos evitando caídas que pueden suponer pérdida de dinero o pérdida de datos, incluso dándose el caso de las dos a la vez.
- Si tenemos grandes cantidades de datos y preferimos la velocidad para acceder a los datos antes que la normalización de estos , entonces Cassandra también puede ser la base de datos a elegir. Pero cuidado si estás acostumbrado a trabajar con base de datos relacionales deberás cambiar la forma de pensar a la hora de diseñar tu modelo de datos.

Como hemos visto en el caso en el que tendremos que elegir Cassandra es en los momentos en lo cuales vamos a almacenar una gran cantidad de información (uso indicado en BigData) y que dicha información

esté siempre accesible y a salvo, y sobre todo la relación no sea necesaria, en ese caso nos tendremos que decantar en las bases de datos relacionales como por ejemplo MySQL, PostgreSQL u Oracle.

14. Casos prácticos implementados actualmente

En la actualidad el uso de Cassandra está más implementado de lo que aparentemente parece por ello a continuación mostraremos de forma resumida los usos que se están dando en la actualidad para así tener una idea del uso de esta base de datos y a lo que está orientado de forma práctica para tener una mejor idea de su uso.

- Análisis de datos de las redes sociales, lo que les permite hacerle recomendaciones a sus clientes.
- Es usado para el manejo y búsqueda de catálogos de productos de tiendas online como es el caso de eBay.
- Para IoT (internet de las cosas), para el manejo de los datos de multitud de sensores, instalados en diversos lugares.
- Se usa para el manejo de aplicaciones de datos de series temporales, como es el caso de datos del clima, gracias a la velocidad de lectura/escritura que provee.
- Para aplicaciones de mensajería, para almacenar los datos de las conversaciones y demás contenido compartido.
- Se usa para rastrear y monitorear la actividad de los usuarios, por ejemplo al escuchar música, ver videos, sitios web, etc.

En todos estos usos que se realizan en producción están detrás empresas como Facebook (que fue la primera que empezó a desarrollar Cassandra), Twitter, Netflix, Apple, Google, Amazon, Microsoft, IBM, etc.

15. Comparativa entre bases de datos NoSQL.

A continuación vamos a ver una comparativa entre diferentes bases de datos NoSQL que se suelen usar, para ver qué diferencia tiene las bases de datos NoSQL del mercado dependiendo de su estructura y funcionamiento, ya que están clasificadas por como maneja los datos y los tiene estructurado. La comparativa que se visualizará será de una forma resumida, escogiendo a dos de las más utilizadas en su sector para así tener en una primera aproximación que diferencias existen entre ellas.

15.1. NoSQL Orientadas a Documentos

Las bases de datos NoSQL orientadas a documentos son aquellas que se gestionan datos semi estructurados. Es decir documentos. Estos datos son almacenados en algún formato estándar como puede ser XML, JSON o BSON. Para hacernos una idea de lo que es un documento, suele ser algo parecido a lo que aparece a continuación.

```
{
  _id:1,
  nombre:"Juan José López",
  telefonos:["465465456","4545654"],
  direccion:
  {
    calle:"C\ Herberos",
    numero:3,
    extra: "Portal 3, 3º B",
    ciudad:"Sevilla"
  }
}
```

Este tipo de bases de datos NoSQL son las más versátiles. Se puedes utilizar en gran cantidad de proyectos, incluyendo muchos que tradicionalmente funcionan sobre bases de datos relacionales.

MongoDB	**CouchDB**
Probablemente la base de datos NoSQL más famosa del momento.Llegando a conseguir hasta 150 millones de dólares en financiación. convirtiéndose en una de las startups más prometedoras. Algunas compañías que actualmente la utilizan son Foursquare o eBay.	Es la base de datos orientada a documentos de Apache. Una de sus interesantes características es que los datos son accesibles a través de una API Rest. Este sistema es utilizado por compañías como Credit Suisse y la BBC.

15.2. Orientada a Columnas

Este tipo de bases de datos están pensadas para realizar consultas y agregaciones sobre grandes cantidades de datos. Funcionan de forma parecida a las bases de datos relacionales, pero almacenando las columnas de datos en lugar de registros.

Cassandra	HBase
Incluida en este apartado, aunque en realidad sigue un modelo híbrido entre orientada a columnas y clave-valor. Es utilizada por Facebook y Twitter como hemos comentado anteriormente (entre otras empresas ya comentadas con anterioridad).	Escrita en Java y mantenida por el Proyecto Hadoop de Apache, se utiliza para procesar grandes cantidades de datos. La utiliza Facebook y Twitter o Yahoo.

15.3. De Clave-Valor

Estas son más sencillas de entender. Simplemente guardan tuplas que contienen una clave y su valor. Cuándo se quiere recuperar un dato, simplemente se busca por su clave y se recupera el valor que va asociado a esa clave.

DynamoDB	Redis
Desarrollada por Amazon, es una opción de almacenaje que podemos usar desde los Amazon Web Services. La utilizan el Washington Post y Scopely.	Desarrollada en C y de código abierto, es utilizada por Craiglist y Stack Overflow (a modo de caché)

15.4. Orientadas a Grafo

Basadas en la teoría de grafos, utilizan nodos y aristas para representar los datos almacenados. Son muy útiles para guardar la información en modelos con muchas relaciones, como redes y conexiones sociales.

Infinite Graph	Neo4j
Escrita en Java y C++ por la compañía Objectivity. Tiene dos modelos de licenciamiento: uno gratuito y otro de pago.	Base de datos de código abierto, escrita en Java por la compañía Neo Technology. Utilizada por compañías como HP, infojobs o Cisco.

16. Librerías Cliente

Además de todas las las características de Cassandra que hemos descrito anteriormente de forma detallada, tiene una comunidad muy activa desarrollando librerías para diferentes lenguajes, por lo cual el soporte para Cassandra de cara a la conexión desde un cliente y su interacción con la base de datos se realiza de una forma más rápida, eficaz y sencilla de cara al usuario final.

Java	Astyanax. Hector.
PHP CQL	Cassandra-PDO.
Python	Pycassa.
Ruby	Ruby Gem.
REST	Virgil.
.NET	Cassandra Sharp.
Node.js	Cassandra Node.

17. Instalación de Cassandra

Todo el proceso de instalación de los paquetes necesarios para cassandra como el propio software cassandra se realizará utilizando la distribución de Ubuntu 16.04 Xential.

17.1. Instalación de Oracle Java VM

Para el correcto funcionamiento de Cassandra, es necesario el uso de Java, asi que tendremos que proceder a instalarlo, para ello tendremos que seguir los siguientes pasos descritos a continuación.

En el directorio del usuario (en nuestro caso root) crearemos un directorio temp.

```
cd ~
mkdir temp
```

Acto seguido nos descargamos el paquete de java de la página de Oracle, dependiendo de la arquitectura de nuestro sistema operativo nos tendremos que descargar la versión de 32bits o 64bits, en nuestro caso al ser de 64bits, nos descargamos la versión para 64bits. La versión de java que nos tenemos que descargar es la versión 8 (Este enlace muestra la página de descarga de java para todas las plataformas).

```
wget http://download.oracle.com/otn-pub/java/jdk/8u162-
b12/0da788060d494f5095bf8624735fa2f1/jdk-8u162-linux-x64.tar.gz
```

Una vez descargado el paquete, procederemos a extraer el contenido del fichero.

```
tar -zxf jdk-8u162-linux-x64.tar.gz
```

En el momento en el que extraemos el contenido nos dirigimos a la ruta ***/usr/local*** y en dicha ruta vamos a crear el siguiente directorio denominado ***java***.

```
cd /usr/local/
mkdir java
```

Y una vez creado el directorio anterior procederemos a realizar el movimiento de datos para que se aloje en esta nueva ubicación.

```
mv jdk1.8.0_162/ /usr/local/java/
```

Acto seguido vamos a introducir las siguientes líneas en el fichero de configuración llamado ***profile*** que se encuentra en ***/etc/*** y dichas líneas lo que está indicando es que se utilice java de forma predeterminada (en la parte en la que se indica en la ruta del directorio Java, tendremos que indicar la ruta que la tengamos en nuestro caso).

```
JAVA_HOME=/usr/local/java/jdk1.8.0_162
PATH=$PATH:$HOME/bin:$JAVA_HOME/bin
JRE_HOME=/usr/local/java/jdk1.8.0_162
PATH=$PATH:$HOME/bin:$JRE_HOME/bin
export JAVA_HOME
export JRE_HOME
export PATH
```

Una vez que ya hemos modificado el fichero, guardaremos los cambios y acto seguido introduciremos los siguientes comandos que nos permitirán la ejecución correcta de lo modificado en el fichero de configuración (en la parte en la que se indica en la ruta del directorio Java, tendremos que indicar la ruta que la tengamos en nuestro caso).

```
sudo update-alternatives --install "/usr/bin/java" "java"
"/usr/local/java/jdk1.8.0_162/bin/java" 1
sudo update-alternatives --install "/usr/bin/javac" "javac"
"/usr/local/java/jdk1.8.0_162/bin/javac" 1
sudo update-alternatives --install "/usr/bin/javaws" "javaws"
"/usr/local/java/jdk1.8.0_162/bin/javaws" 1
sudo update-alternatives --set java /usr/local/java/jdk1.8.0_162/bin/java
sudo update-alternatives --set javac /usr/local/java/jdk1.8.0_162/bin/javac
sudo update-alternatives --set javaws
/usr/local/java/jdk1.8.0_162/bin/javaws
```

En el momento en el que hayamos ejecutado los comandos anteriores, tendremos que reiniciar el sistema para que en el momento del inicio, se nos ejecute el fichero que hemos editado anteriormente.

```
reboot
```

Una vez que ya tengamos operativo el sistema después del reinicio, vamos a comprobar la versión de Java para verificar que lo tenemos en funcionamiento correctamente, así que para ello vamos a introducir el siguiente comando.

```
java -version
echo $JAVA_HOME
```

Una vez introducido estos comandos nos debería aparecer un resultado parecido al siguiente.

```
ubuntu@cassandra-1:~$ java -version
java version "1.8.0_162"
Java(TM) SE Runtime Environment (build 1.8.0_162-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.162-b12, mixed mode)
ubuntu@cassandra-1:~$ echo $JAVA_HOME
/usr/local/java/jdk1.8.0_162
```

17.2. Instalación de Cassandra

En la instalación de Cassandra tenemos dos posibilidades de instalar el software y es realizar la instalación a través de la descarga directa del paquete de su página oficial o su instalación a través de paquetería haciendo uso de los repositorios de apache, en nuestro caso vamos a contemplar las dos posibilidades(en todo momento el sistema operativo que vamos a utilizar es Ubuntu 16.04 Xential).

17.2.1. Instalación desde la web

Para la instalación de Cassandra tendremos que descargarnos el paquete de su página oficial y para ello vamos a utilizar el comando wget, accederemos a la carpeta ***temp*** que nos hemos descargado anteriormente y lo descargamos ahí.

```
cd ~/temp
wget http://apache.uvigo.es/cassandra/3.11.2/apache-cassandra-3.11.2-
bin.tar.gz
```

Una vez descargado tendremos que extraer los datos del fichero descargado con el comando tar y lo moveremos a un directorio llamado Cassandra (este directorio lo creamos nosotros).

```
ubuntu@cassandra-1:~/temp$ tar -zxf apache-cassandra-3.11.2-bin.tar.gz
ubuntu@cassandra-1:~/temp$ ls
apache-cassandra-3.11.2/  apache-cassandra-3.11.2-bin.tar.gz

ubuntu@cassandra-1:~/temp$ cd ..
ubuntu@cassandra-1:~$ mkdir cassandra
ubuntu@cassandra-1:~$ mv temp/apache-cassandra-3.11.2 cassandra/
```

Cuando ya hayamos realizado el desempaquetado del fichero, procederemos realizar la creación de los directorios en los cuales se va a instalar Cassandra, así que para ello vamos a realizar los siguiente.

```
sudo mkdir /var/lib/cassandra
sudo mkdir /var/log/cassandra
sudo chown -R $USER:$GROUP /var/lib/cassandra
sudo chown -R $USER:$GROUP /var/log/cassandra
```

A continuación procederemos a cargar las siguientes variables necesarias.

```
export CASSANDRA_HOME=~/cassandra
```

```
export PATH=$PATH:$CASSANDRA_HOME/bin
```

17.2.2. Iniciando Cassandra (Inicio simple 1 nodo)

Una vez que hemos exportado las variables anteriores, procederemos a iniciar cassandra, así que para ello vamos a introducir el siguiente comando necesario para el inicio del servicio de Cassandra (Este inicio de Cassandra se realiza con la configuración por defecto que es un cluster con un solo nodo, siendo la configuración más simple de Cassandra).

```
sudo sh ~/cassandra/bin/cassandra
```

Acto seguido iniciaremos el intérprete de comandos de CQL para poder interaccionar con la base de datos y desde este momento ya tendremos una base de datos Cassandra funcionando correctamente

```
sudo sh ~/cassandra/bin/cqlsh
```

En el momento que esté introducido dicho comando, se nos mostrará la línea de comandos de Cassandra.

```
ubuntu@cassandra-1:~$ sudo sh ~/cassandra/bin/cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh>
```

17.2.3. Instalación por repositorios

Para realizar la instalación a través de repositorio tendremos que empezar añadiendo el repositorio del que vamos a obtener Cassandra, así que para ello vamos a introducir el siguiente comando.

```
echo "deb http://www.apache.org/dist/cassandra/debian 311x main" | sudo tee
-a /etc/apt/sources.list.d/cassandra.sources.list
```

Acto seguido añadiremos el repositorio de claves en el sistema.

```
curl https://www.apache.org/dist/cassandra/KEYS | sudo apt-key add -
sudo apt-key adv --keyserver pool.sks-keyservers.net --recv-key
A278B781FE4B2BDA
```

Una vez añadido el repositorio de claves, procederemos a realizar la actualización de paquetes para así utilizar el nuevo repositorio.

```
sudo apt-get update
```

A continuación procederemos a instalar Cassandra con el siguiente comando.

```
sudo apt-get install cassandra
```

17.2.4. Iniciando Cassandra (Inicio simple 1 nodo)

En el momento que lo hayamos instalado procederemos a realizar el inicio del servicio, ya que cassandra se ha instalado pero no se ha iniciado, por ello tendremos que introducir la siguiente instrucción, hay que tener en cuenta que se iniciará con la configuración predefinida que será la configuración de un solo nodo.

```
sudo systemctl start cassandra.service
```

Por consiguiente una vez que lo hayamos iniciado, introduciremos el siguiente comando para abrir la shell de Cassandra e interactuar con ella.

```
ubuntu@cassandra-1:~$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh>
```

18. Primeros pasos en Cassandra

Una vez visto de forma teórica cómo se realiza el modelado de datos, vamos a llevarlo a la práctica en este único nodo en el cual hemos empezado instalando Cassandra, vamos a ver cómo se realiza la gestión básica de Cassandra.

Para comenzar, abrimos la shell de Cassandra con el siguiente comando, lo iniciaremos como administrador de la base de datos para así realizar la creación de objetos.

```
ubuntu@cassandra-1:~$ cqlsh -u cassandra -p cassandra
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
root@cqlsh>
```

Acto seguido crearemos el keyspace que en nuestro caso se denominará "*prueba"* pero antes listamos los keyspaces que tenemos creados para así comprobar que junto con la instalación de Cassandra, ya se nos ha creado varios que son los keyspaces necesarios para el sistema.

```
cassandra@cqlsh> describe keyspaces;
system_traces  system_schema  system_auth  system  system_distributed

cassandra@cqlsh> create keyspace if not exists prueba with replication = {
'class' : 'SimpleStrategy', 'replication_factor' : 1};

cassandra@cqlsh> describe keyspaces;
prueba  system_schema  system_auth  system  system_distributed
system_traces

cassandra@cqlsh>
```

En el proceso de creación del keyspace le estamos indicando lo siguiente:

```
create keyspace if not exists prueba with replication = { 'class' :
'SimpleStrategy', 'replication_factor' : 1};
```

Le indicamos que nos cree el keyspace en el caso de que no exista (si existe no nos lo creará para que no sobreescriba la información), y en cuanto a las condiciones de creación que nos lo cree con la política replicado en un solo nodo, ya que no disponemos de mas *"SimpleStrategy"* y que conste con un sola réplica *"replication_factor"*.

Una vez creado el Keyspace, vamos realizar la creación del usuario que se llamará *"juanjose"* y es el que se encargará de la gestión de ese keyspace, pero antes de realizar ninguna creación de usuario, vamos a cambiar un parámetro en la configuración de Cassandra ya que en el caso de querer realizar la creación de ese usuario nos aparece un error el cual no nos permite realizar la creación del mismo, siendo los siguientes errores.

Creación de usuario con privilegios de superusuario.

```
cassandra@cqlsh> create user if not exists juanjose with password
'juanjose' superuser;
Unauthorized: Error from server: code=2100 [Unauthorized] message="Only
superusers can create a role with superuser status"
```

Creación de usuario sin privilegios de superusuario.

```
cassandra@cqlsh> create user if not exists juanjose with password
```

```
'juanjose' nosuperuser;
InvalidRequest: Error from server: code=2200 [Invalid query]
message="org.apache.cassandra.auth.CassandraRoleManager doesn't support
PASSWORD"
```

En ambos casos el fallo que nos está dando es que Cassandra por defecto no nos permite realizar la creación de usuario mediante contraseña y por ello tampoco podremos asignarle los privilegios de superusuario. Así que, para que Cassandra nos permita la creación de los usuario tendremos que realizar un cambio en los parámetros de autenticación para que así nos permita crear los usuario mediante contraseñas.

Para ello vamos a salirnos de la shell de Cassandra y vamos a modificar el fichero ***cassandra.yaml*** que se encuentra en ***/etc/cassandra*** y como root cambiaremos la siguiente línea:

```
sudo nano /etc/cassandra/cassandra.yaml
```

Cambiamos esta línea:

```
authenticator: AllowAllAuthenticator
```

Por esta:

```
authenticator: PasswordAuthenticator
```

Una vez que hemos cambiado este parámetro procederemos a guardar los cambios y acto seguido vamos reiniciamos el servicio para aplicar los cambios realizados.

```
sudo systemctl restart cassandra.service
```

En el momento que ya se han aplicado los cambios, procederemos a acceder de nuevo a la shell de cassandra y ya podremos realizar la creación de los usuarios sin problemas, sean con privilegios de superusuario o no.

```
cassandra@cqlsh> create user if not exists juanjose2 with password
'juanjose2';

cassandra@cqlsh> create user if not exists juanjose with password
'juanjose' superuser;
```

Acto seguido a este usuario creado vamos a asignarle privilegios sobre ese keyspace para que ese usuario pueda gestionarlo y dichos privilegios los asignaremos con la instrucción GRANT.

En el momento de realizar la asignación de privilegios, nos vuelve a salir otro error y es que en la configuración de Cassandra no están los parámetros definidos correctamente para poder asignar privilegios, por ello vamos a dirigirnos al fichero que hemos editado anteriormente y vamos a realizar lo siguiente.

```
sudo nano /etc/cassandra/cassandra.yaml
```

Estando dentro del fichero cambiaremos la siguiente línea:

```
authorizer: AllowAllAuthorizer
```

Por esta otra:

```
authorizer: CassandraAuthorizer
```

Guardaremos los cambios y procederemos a reiniciar el servicio nuevamente para que se apliquen los cambios guardados.

```
sudo systemctl restart cassandra.service
```

Una vez que ya se haya reiniciado el servicio para aplicar los cambios podremos asignar los permisos deseados a un usuario concreto, ya que la línea que hemos cambiado en el fichero le estamos indicando que nos guarde la información de los privilegios en la tabla denominada ***system_auth.role***, para que así pueda asociarse los privilegios de un usuario a una tabla, porque en el caso de la otra opción no permitía almacenar esa información, por ello no era soportada la opción de asignar los privilegios.

Volveremos a acceder a la shell de Cassandra y asignaremos los correspondientes privilegios para que el usuario *juanjose* pueda administrar el Keyspace denominado *prueba*.

```
cassandra@cqlsh> GRANT ALL ON KEYSPACE prueba TO juanjose;
```

En el momento en el que ya esté asignado ese privilegio vamos a proceder a salir de la shell de Cassandra como usuario cassandra y accederemos a la misma esta vez con el usuario que hemos creado con anterioridad denominado juanjose.

```
ubuntu@cassandra-1:~$ cqlsh -u juanjose -p juanjose
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
```

```
juanjose@cqlsh>
```

Acto seguido estando dentro con el usuario *juanjose,* listamos los keyspaces y veremos como nos aparece el keyspace prueba y accederemos a dicho keyspace usando el comando use.

```
juanjose@cqlsh> describe keyspaces;
prueba  system_schema  system_auth  system  system_distributed
system_traces

juanjose@cqlsh> use prueba;
juanjose@cqlsh:prueba>
```

Una vez que ya estemos haciendo uso de ese keyspace procederemos a realizar la creación de una tabla, siendo la siguiente:

```
 CREATE TABLE personas (
      dni text,
      nombre text,
      apellidos text,
      correo text,
      organizacion text,
      PRIMARY KEY (dni));
```

En el momento que hemos realizado la creación de la siguiente tabla procederemos a comprobar que se ha creado correctamente, por ello vamos a hacer un ***describe*** de la misma para ver sus características.

```
juanjose@cqlsh:prueba> describe personas;
CREATE TABLE prueba.personas (
      dni text PRIMARY KEY,
      apellidos text,
      correo text,
      nombre text,
      organizacion text
) WITH bloom_filter_fp_chance = 0.01
      AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
      AND comment = ''
      AND compaction = {'class':
'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy',
'max_threshold': '32', 'min_threshold': '4'}
      AND compression = {'chunk_length_in_kb': '64', 'class':
```

```
'org.apache.cassandra.io.compress.LZ4Compressor'}
      AND crc_check_chance = 1.0
      AND dclocal_read_repair_chance = 0.1
      AND default_time_to_live = 0
      AND gc_grace_seconds = 864000
      AND max_index_interval = 2048
      AND memtable_flush_period_in_ms = 0
      AND min_index_interval = 128
      AND read_repair_chance = 0.0
      AND speculative_retry = '99PERCENTILE';
```

En el momento en el que ya esté la tabla creada correctamente, procederemos a poblar dicha tabla, en la cual vamos a insertar los siguientes datos que aparecen a continuación.

```
INSERT INTO personas (dni, nombre, apellidos, correo, organizacion) VALUES
('49231685Y', 'Mario', 'Perez', 'mario@correo.com', 'marketing');
INSERT INTO personas (dni, nombre, apellidos, correo, organizacion) VALUES
('51968723Z', 'Jose', 'Marquez', 'jm@correo.com', 'ventas');
INSERT INTO personas (dni, nombre, apellidos, correo, organizacion) VALUES
('33968574L', 'Laura', 'Romero', 'lauritar@correo.com', 'atencion al
cliente');
INSERT INTO personas (dni, nombre, apellidos, correo, organizacion) VALUES
('47859632H', 'Elisabet', 'Lopez', 'elilopez@correo.com', 'directora');
INSERT INTO personas (dni, nombre, apellidos, correo, organizacion) VALUES
('98765448A', 'Juan Jose', 'Lopez', 'jjlr@correo.com', 'gerente');
```

Una vez que hemos introducido los datos comprobaremos que todo está correcto realizando una consulta. Como se realiza de la misma forma que en los sistemas relacionales pues lo que realizaremos será una ***SELECT***.

```
juanjose@cqlsh:prueba> select * from personas;

 dni       | apellidos | correo                  | nombre    | organizacion
-----------+-----------+-------------------------+-----------+---------------------
 98765448A | Lopez     | jjlr@correo.com         | Juan Jose | gerente
 49231685Y | Perez     | mario@correo.com        | Mario     | marketing
 51968723Z |  Marquez  | jm@correo.com           | Jose      | ventas
 47859632H | Lopez     | elilopez@correo.com     | Elisabet  | directora
 33968574L | Romero    | lauritar@correo.com     | Laura     | atencion al cliente

(5 rows)
```

Si se dá el caso en el que existe un dato en el que no esté registrado correctamente o que tiene que ser actualizado por que ha cambiado, podremos realizar una corrección con la instrucción ***UPDATE***. En esta ocasión vamos a cambiar la organización a la que pertenece *Laura*.

```
update personas set organizacion = 'ventas' where dni = '33968574L';
```

Acto seguido, en el momento que ya se ha actualizado, volvemos a comprobar que se ha cambiado correctamente.

```
juanjose@cqlsh:prueba> select * from personas;

 dni       | apellidos | correo                   | nombre   | organizacion
-----------+-----------+--------------------------+----------+--------------------
 98765448A | Lopez     | jjlr@correo.com          | Juan Jose | gerente
 49231685Y | Perez     | mario@correo.com         | Mario     | marketing
 51968723Z |  Marquez  | jm@correo.com            | Jose      | ventas
 47859632H | Lopez     | elilopez@correo.com | Elisabet  | directora
 33968574L | Romero    | lauritar@correo.com | Laura     | ventas

(5 rows)
```

Además de ello, vamos a ver cómo se realiza el borrado de un dato de forma práctica si se dá el caso de que tengamos que eliminar a una persona de la tabla creada anteriormente, en nuestro caso vamos a eliminar a la persona llamada *Mario* cuyo dni es *49231685Y*.

```
delete from personas where dni = '49231685Y';
```

Por consiguiente, una vez borrado, procederemos a realizar de nuevo la consulta para comprobar que se ha eliminado correctamente el dato indicado anteriormente.

```
juanjose@cqlsh:prueba> select * from personas;

 dni       | apellidos | correo                   | nombre   | organizacion
-----------+-----------+--------------------------+----------+--------------------
 98765448A | Lopez     | jjlr@correo.com          | Juan Jose | gerente
 51968723Z |  Marquez  | jm@correo.com            | Jose      | ventas
 47859632H | Lopez     | elilopez@correo.com | Elisabet  | directora
 33968574L | Romero    | lauritar@correo.com | Laura     | ventas
```

19. Configuración de cluster multi-nodo con Cassandra

En este apartado vamos a ver cómo se realiza la configuración de Cassandra para que funcione en un cluster multi-nodo cuya configuración hará que la información este replicada y repartida entre los nodos en ***un data center*** (que será un tipo de configuración) y que la información esté repartida y replicada por los nodos de la red estando los nodos en ***diferentes data centers***. En ambas configuraciones vamos a partir siempre de que tenemos los nodos con Cassandra instalada y lista para ser configurada correctamente.

19.1. Configuración de clúster horizontal con la información replicada/repartida en un data center

En este caso práctico en el que se realiza un cluster con la información replicada y repartida, las máquinas a utilizar se especifican a continuación.

- Cassandra-1:
  - S.O : Ubuntu 16.04 Xential
  - IP: 10.0.0.7
- Cassandra-2:
  - S.O : Ubuntu 16.04 Xential
  - IP: 10.0.0.9

Para empezar, en ambos nodos lo que tendremos que realizar es detener el proceso de Cassandra y acto seguido eliminamos la configuración predeterminada para aplicarle la adecuada para que funcione en multi-nodo con la información replicada.

```
sudo service cassandra stop
sudo rm -rf /var/lib/cassandra/data/system/*
```

Una vez que ya se ha puesto las dos bases de dato con los valores predeterminado eliminados, procederemos a modificar los ficheros de configuración de Cassandra en ambos nodos, por ello editaremos en ambos equipos el fichero ***cassandra.yaml*** que se encuentra en la ruta ***/etc/cassandra/*** y en dicho fichero indicaremos las siguientes líneas.

```
sudo nano /etc/cassandra/cassandra.yaml
```

- Cassandra-1

```
cluster_name: 'cluster replicado'
num_tokens: 256
authenticator: PasswordAuthenticator
authorizer: CassandraAuthorizer
seed_provider:
      - class_name: org.apache.cassandra.locator.SimpleSeedProvider
      parameters:
            - seeds: "10.0.0.7,10.0.0.9"
listen_address: 10.0.0.7
rpc_address: 10.0.0.7
endpoint_snitch: SimpleSnitch
```

- Cassandra-2

```
cluster_name: 'cluster replicado'
num_tokens: 256
authenticator: PasswordAuthenticator
authorizer: CassandraAuthorizer
seed_provider:
      - class_name: org.apache.cassandra.locator.SimpleSeedProvider
      parameters:
            - seeds: "10.0.0.7,10.0.0.9"
listen_address: 10.0.0.9
rpc_address: 10.0.0.9
endpoint_snitch: SimpleSnitch
```

En el momento en el que ya tengamos indicado todos esos parámetros en ambos nodos vamos a explicar para qué sirve cada parámetro que le estamos pasando para saber en todo momento lo que estamos configurando.

- **cluster_name**: es el nombre que describe el cluster que estamos configurando, puede definir el deseado por el usuario, puede tener espacios, siempre y cuando los datos introducidos estén dentro de las comillas.
- **num_tokens**: este parámetro que indicamos es la cantidad de nodos virtuales dentro de una instancia de Cassandra. Esto se usa para dividir los datos y difundir lo datos a través del clúster. El valor recomendado de partida es de 256.
- **authenticator**: este parámetro indica que el usuario se autentica a través de una cuenta de usuario con su password.
- **authorizer**: este valor que le estamos indicando le decimos a Cassandra que la gestión de permisos sea a través de privilegios con GRANT/REVOKE.
- **seed**: en este valor indicamos las direcciones ip's de los nodos/semillas que van a estar disponible en ese cluster, ya que en todo momento las direcciones ip's especificadas están indicando a

Cassandra que puede confiar en esos nodos par que así en caso de fallo no exista un único punto, si no que existan varios puntos donde repartir la información y así tener los datos a salvo

- **listen_address**: es la dirección ip por la que Cassandra escuchara las peticiones para la comunicación interna (de Cassandra a Cassandra). En el caso en el que la dirección ip la deje en blanco, Cassandra intentará averiguar la dirección que tiene, pero siempre es mejor y recomendable especificarla.
- **rpc_address**: es la dirección ip por la que Cassandra escuchara para la comunicación basada en el cliente, como a través del protocolo CQL. En nuestro caso es la misma dirección ya que solamente disponemos de una sola interfaz.
- **endpoint_snitch**: en esta opción le indicamos a Cassandra a que centro de datos y rack pertenece un nodo dentro de un clúster. Hay varios tipos de opciones que se pueden usar aquí, que en el mismo fichero de configuración nos describe las diferentes opciones disponibles.

En el momento que ya hayamos configurado todo lo anterior, procederemos a editar el fichero de configuración denominado ***cassandra-rackdc.properties*** que se encuentra en ***/etc/cassandra***. En dicho fichero vamos a indicar los atributos de cada nodo en el cual definiremos si dicho nodo pertenece al mismo **rack** o centro de datos (**dc**).
En nuestro caso vamos a definir que estamos en el mismo centro de datos (mismo cluster) pero que cada nodo estará en un rack diferente, para que así Cassandra al estar diseñado para ser tolerante a fallos, distribuirá los datos y los backups de los mismo entre los diferentes nodos para así minimizar el riesgo de pérdida de datos.

Así que para ello editaremos este fichero en ambos nodos e indicaremos el mismo **dc** pero un **rack** diferente, tal y como aparece a continuación.

```
nano /etc/cassandra/cassandra-rackdc.properties
```

- Cassandra-1

```
dc=dc1
rack=rack1
```

- Cassandra-2

```
dc=dc1
rack=rack2
```

Para finalizar eliminamos el fichero denominado ***cassandra-topology.properties*** que se encuentra en ***/etc/cassandra***, ya que dicha configuración que contiene ese fichero no es compatible con la configuración que en este caso le estamos definiendo, así que la eliminaremos utilizando el siguiente comando (este fichero se elimina en ambos nodos).

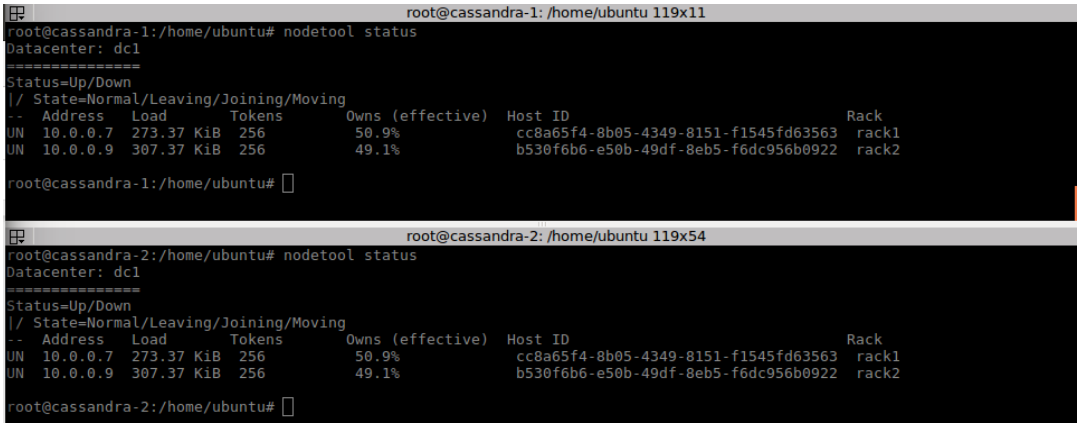
```
rm /etc/cassandra/cassandra-topology.properties
```

19.1.1. Iniciando cluster horizontal de Cassandra

En el momento en el que ya se ha eliminado el fichero, procederemos a iniciar el servicio de Cassandra nuevamente para así aplicar todos los cambios que le hemos realizado desde que habíamos detenido dicho proceso, para ello vamos a introducir el siguiente comando (el inicio del proceso se realiza en ambos nodos).

```
service cassandra start
```

Una vez que ya hemos iniciado Cassandra en ambos nodos, procederemos a listar el estado de dicho cluster, mostrando los nodos que lo componen, dicha información se muestra con el comando denominado ***nodetool status***.

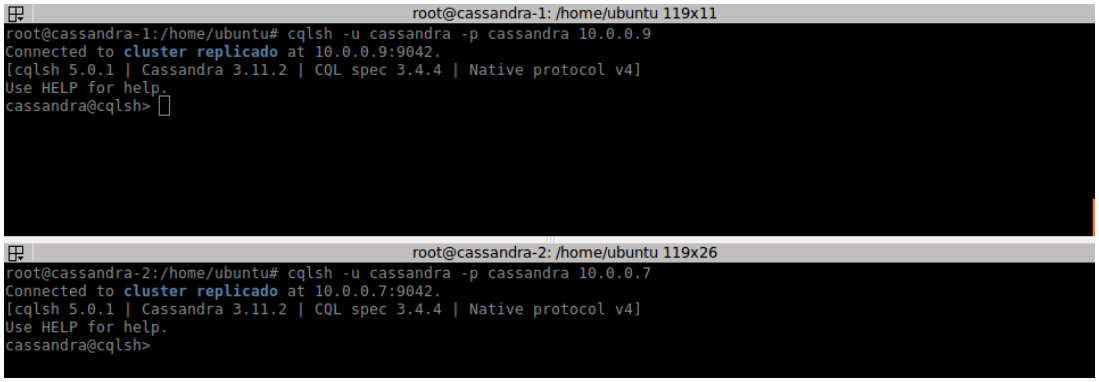


Como aparece en la anterior captura, la ejecución de ese comando nos permite ver la cantidad de nodos que estén en el mismo ***dc*** en el que se encuentre este nodo de Cassandra. Como se puede apreciar en la imagen, tenemos un ***dc*** denominado *dc1* en el cual se encuentra los dos equipos que se encuentra en diferentes rack para así diferenciar uno del otro en cuanto a su localización física, y junto a ella nos aparece las características de cada nodo, estando entre ellos sincronizados y mostrando así la eficiencia que disponen cada uno de ellos.

19.1.2. Conexión con la base de datos de Cassandra (entre servidores)

En el caso en el que deseemos comprobar si podemos acceder a las bases de datos de los nodos, procederemos a introducir los siguientes comandos en ambos nodos, que no es más que el acceso anterior a la base de datos incluyendo así la dirección ip del nodo al que se quiere conectar para diferenciar un

nodo de otro a la hora de desear conectarnos a uno de ellos (la ip que estemos indicando será la misma que hemos indicado en el fichero de configuración de *cassandra.yaml* en la opción denominada *rpc_address* que servía para indicar porqué ip, iba a escuchar la base de datos Cassandra).



En la captura anterior se muestra como se pueden acceder a ambos nodos, teniendo como prueba de conexión, el acceso de Cassandra-1 a Cassandra-2 y de Cassandra-2 a Cassandra-1.

19.1.3. Conexión con la base de datos de Cassandra (desde el cliente CQLSH)

Además de poder acceder a los diferentes nodos de Cassandra entre los propios servidores, también podemos acceder a la base de datos de Cassandra o Cluster de Cassandra a través del cliente de Cassandra (***CQLSH***) que lo instalaremos en un entorno virtual para mayor comodidad de cara a su usabilidad e instalación para así, instalarlo en el equipo que sea necesario de forma rápida y sencilla.

Para empezar tendremos que instalar en el cliente como usuario root, el siguiente paquete que es el que nos permite realizar la creación de los entornos virtuales.

```
apt install virtualenv
```

En el momento en el que ya esté instalado el paquete, procederemos a crear el entorno virtual, para ello vamos a introducir el siguiente comando que se encargará de dicha acción (a dicho entorno le vamos a asignar el nombre de CASSANDRA).

```
virtualenv CASSANDRA
```

Una vez que hayamos creado el entorno virtual, procederemos a activarlo, para ello vamos a entrar en dicho entorno (dentro de ese directorio) y dentro del directorio bin, ejecutaremos el fichero denominado ***activate***, y todo ello se realizara con el comando *source*.

```
source CASSANDRA/bin/activate
```

Cuando ya se haya ejecutado el fichero para activar el entorno veremos algo parecido a lo siguiente en la línea de comandos.

```
(CASSANDRA) debian@cliente-cassandra:~$
```

A continuación realizaremos la instalación del paquete necesario para utilizar el cliente de Cassandra utilizando el instalador ***pip*** que nos ofrece el entorno virtual. El paquete a instalar se llama ***cqlsh***.

```
(CASSANDRA) debian@cliente-cassandra:~$ pip install cqlsh
Collecting cqlsh
  Downloading cqlsh-5.0.4.tar.gz (99kB)
      100% |████████████████████████████████| 102kB 1.4MB/s
Collecting cql (from cqlsh)
Collecting cassandra-driver (from cqlsh)
Collecting thrift (from cql->cqlsh)
Collecting six>=1.9 (from cassandra-driver->cqlsh)
  Using cached six-1.11.0-py2.py3-none-any.whl
Collecting futures (from cassandra-driver->cqlsh)
  Using cached futures-3.2.0-py2-none-any.whl
Building wheels for collected packages: cqlsh
  Running setup.py bdist_wheel for cqlsh ... done
  Stored in directory:
/home/debian/.cache/pip/wheels/67/bc/b9/8fd53967b3eac0cacb31d17653b68be630c
44d5ca933f56d44
Successfully built cqlsh
Installing collected packages: six, thrift, cql, futures, cassandra-driver,
cqlsh
Successfully installed cassandra-driver-3.13.0 cql-1.4.0 cqlsh-5.0.4
futures-3.2.0 six-1.11.0 thrift-0.11.0
```

Una vez instalado el paquete procederemos a conectarnos a los diferentes servidores del cluster de Cassandra. Para ello vamos a introducir el siguiente comando que nos permitirá realizar la conexión.

```
cqlsh -u cassandra -p cassandra 10.0.0.7 9042
```

En el comando anterior lo que estamos indicando es el usuario y su correspondiente contraseña del usuario que queremos iniciar en la base de datos, seguido de la dirección ip del nodo de Cassandra al que nos queremos conectar y su correspondiente puerto.

Al realizar esta conexión, es probable que nos devuelva un error en la conexión, siendo el que aparece a continuación.

```
(CASSANDRA) debian@cliente-cassandra:~$ cqlsh -u cassandra -p cassandra
10.0.0.7 9042

Connection error: ('Unable to connect to any servers', {'10.0.0.7':
ProtocolError("cql_version '3.3.1' is not supported by remote (w/ native
protocol). Supported versions: [u'3.4.4']",)})
```

Este error que nos devuelve, lo que nos quiere decir es que en la conexión que deseamos realizar con el servidor estamos utilizando una versión de CQLSH diferente a la que posee el servidor, entonces no es posible realizar la conexión.

Para ello tendremos que especificar la versión con la que nos queremos conectar, que dicha versión a utilizar es la que nos devuelve en el mensaje de error anterior, así que especificaremos la versión con la opción denominada ***--cqlversión='versión elegida'***, y ya tendremos conexión correctamente con la base de datos Cassandra.

Para comprobar que todo funciona correctamente, realizaremos la conexión con los dos nodos que tenemos disponibles y viendo cómo todo funciona como debe ser.

- Cassandra-1

```
(CASSANDRA) debian@cliente-cassandra:~$ cqlsh -u cassandra -p
cassandra 10.0.0.7 9042 --cqlversion=3.4.4
Connected to cluster replicado at 10.0.0.7:9042.
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol
v4]
Use HELP for help.écassandra@cqlsh>
```

- Cassandra-2

```
(CASSANDRA) debian@cliente-cassandra:~$ cqlsh -u cassandra -p
cassandra 10.0.0.9 9042 --cqlversion=3.4.4
Connected to cluster replicado at 10.0.0.9:9042.
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol
v4]
Use HELP for help.
cassandra@cqlsh>
```

19.1.4. Prueba de cluster horizontal en un data center

A continuación vamos a poblar la base de datos de Cassandra, creando su correspondiente keyspace, usuario y tabla con sus datos incluidos y acto seguido apagaremos uno de los nodos para comprobar cómo

toda la información sigue replicada en el segundo nodo que sigue ofreciendo el servicio y en todo momento el cliente o clientes que utilizan ese servicio, no se percatan de lo ocurrido.

Como ejemplo vamos a introducir los mismos datos que hemos introducido en el caso anterior (en el apartado de primeros pasos con cassandra), pero modificando varios parámetros para adaptarlos así a esta estructura. A continuación aparecerá los datos que vamos a introducir.

- Para empezar vamos a crear el keyspace, que en esta ocasión al utilizar un solo ***dc (centro de datos)*** de dos nodos, se dejará la opción del modo de réplica de datos en simple, pero su cantidad la cambiaremos a 4, por lo que realizará cuatro réplicas de los datos de ese keyspace por todos los nodos que existan en ese ***dc***, repartiendo así la información por el cluster.

```
cassandra@cqlsh> create keyspace if not exists comvive with
replication = { 'class' : 'SimpleStrategy', 'replication_factor' :
4};
```

- Además de ello tendremos que alterar la configuración del keyspace de autenticación denominado ***"system_auth"*** del nodo1 de cassandra, que es el nodo que estamos poblando con el keyspace denominado *comvive* y por ello en ese mismo nodo como registraremos el usuario *juanjose*, para que también se propague la configuración de ese usuario, tendremos que replicar el keyspace de autenticación para que todos los nodos tengan esa información de ese usuario y se pueda acceder con ese usuario en cualquier nodo de ese ***centro de datos (dc) o desde otro.*** Por ello modificaremos ese keyspace de la siguiente forma.

```
alter keyspace system_auth WITH replication = {'class':
'SimpleStrategy', 'replication_factor': 4};
```

  De forma resumida, en el nodo en el que vayamos a registrar los datos tendremos que indicar la configuración de replicado, para que la información sea accesible desde todos los nodos y además de ello, en el nodo en el que se vaya a realizar la creación y gestión de usuarios y privilegios, realizaremos el replicado del keyspace de autenticación denominado ***"system_auth"*** para que todos los nodos tengan los mismos usuarios y se puedan acceder desde todos los usuarios disponibles.
- Acto seguido crearemos el usuario para gestionar ese keyspace y además de ello vamos a asignarle sus respectivos privilegios para poder administrarlo.

```
cassandra@cqlsh> create user if not exists juanjose with password
'juanjose' superuser;
cassandra@cqlsh> GRANT ALL ON KEYSPACE comvive TO juanjose;
```

- A continuación vamos a acceder con ese usuario al que se ha asignado los permisos sobre ese keyspace y con él, vamos a usar el keyspace creado.

```
(CASSANDRA) kiki@msi:~$ cqlsh -u juanjose -p juanjose 10.0.0.7 9042 -
-cqlversion=3.4.4
```

```
Connected to cluster replicado at  10.0.0.7:9042.
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol
v4]
Use HELP for help.
juanjose@cqlsh> describe keyspaces;
comvive  system_schema  system_auth  system  system_distributed
system_traces
juanjose@cqlsh> use comvive ;
juanjose@cqlsh:comvive>
```

- Una vez que ya estemos dentro del keyspace procederemos a crear una tabla y la poblamos para que se pueda realizar la prueba correctamente, intentando equivaler la prueba a un caso real. Como vemos a continuación al tratarse de una prueba lo más asemejada a un caso real, pues la primary key está compuesta por la parte de la partition key y la parte de la cluster key, explicada anteriormente.

  Creación de tabla.

```
CREATE TABLE empleados (
     dni text,
     nombre text,
     apellidos text,
     correo text,
     puesto text,
     contratacion timestamp,
     PRIMARY KEY (contratacion,dni));
```

Poblado de tabla.

```
INSERT INTO empleados (dni, nombre, apellidos, correo, puesto,
contratacion) VALUES ('49231685Y', 'Mario', 'Perez',
'mario@correo.com', 'marketing','2017-02-15 15:00:00');
INSERT INTO empleados (dni, nombre, apellidos, correo, puesto,
contratacion) VALUES ('51968723Z', 'Jose', 'Marquez',
'jm@correo.com', 'ventas','2017-05-22 12:30:30.000');
INSERT INTO empleados (dni, nombre, apellidos, correo, puesto,
contratacion) VALUES ('33968574L', 'Laura', 'Romero',
'lauritar@correo.com', 'atencion al cliente','2017-05-22
14:12:10.020');
INSERT INTO empleados (dni, nombre, apellidos, correo, puesto,
```

```
contratacion) VALUES ('47859632H', 'Elisabet', 'Lopez',
'elilopez@correo.com', 'directora','2017-06-22 14:59:00.00');
INSERT INTO empleados (dni, nombre, apellidos, correo, puesto,
contratacion) VALUES ('98765448A', 'Juan Jose', 'Lopez',
'jjlr@correo.com', 'gerente','2018-02-28 19:45:00.060');
```

- En el momento en el que esté la tabla creada y los datos introducidos, procederemos a listar los datos para ver que todo está correcto, pero como suponemos que se ha replicado dicha información en ambos nodos, esa consulta la realizaremos en los dos nodos para comprobar que se ha realizado el replicado correctamente.

Cassandra-Nodo1

```
(CASSANDRA) kiki@msi:~$ cqlsh -u juanjose -p juanjose 10.0.0.7 9042 --cqlversion=3.4.4
Connected to cluster replicado at 10.0.0.7:9042.
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
juanjose@cqlsh> use comvive ;
juanjose@cqlsh:comvive> SELECT * from empleados ;

 contratacion                    | dni       | apellidos | correo              | nombre    | puesto
---------------------------------+-----------+-----------+---------------------+-----------+--------------------
 2017-05-22 12:12:10.020000+0000 | 33968574L |    Romero | lauritar@correo.com |     Laura | atencion al cliente
 2017-02-15 14:00:00.000000+0000 | 49231685Y |     Perez |    mario@correo.com |     Mario |           marketing
 2017-06-22 12:59:00.000000+0000 | 47859632H |     Lopez | elilopez@correo.com |  Elisabet |           directora
 2018-02-28 18:45:00.060000+0000 | 98765448A |     Lopez |     jjlr@correo.com | Juan Jose |             gerente
 2017-05-22 10:30:30.000000+0000 | 51968723Z |   Marquez |       jm@correo.com |      Jose |              ventas

(5 rows)
```

Cassandra-Nodo2

```
(CASSANDRA) kiki@msi:~$ cqlsh -u juanjose -p juanjose 10.0.0.9 9042 --cqlversion=3.4.4
Connected to cluster replicado at 10.0.0.9:9042.
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
juanjose@cqlsh> use comvive ;
juanjose@cqlsh:comvive> SELECT * from empleados ;

 contratacion                    | dni       | apellidos | correo              | nombre    | puesto
---------------------------------+-----------+-----------+---------------------+-----------+--------------------
 2017-05-22 12:12:10.020000+0000 | 33968574L |    Romero | lauritar@correo.com |     Laura | atencion al cliente
 2017-02-15 14:00:00.000000+0000 | 49231685Y |     Perez |    mario@correo.com |     Mario |           marketing
 2017-06-22 12:59:00.000000+0000 | 47859632H |     Lopez | elilopez@correo.com |  Elisabet |           directora
 2018-02-28 18:45:00.060000+0000 | 98765448A |     Lopez |     jjlr@correo.com | Juan Jose |             gerente
 2017-05-22 10:30:30.000000+0000 | 51968723Z |   Marquez |       jm@correo.com |      Jose |              ventas

(5 rows)
```

- Una vez que hayamos comprobado que en los dos nodos se puede visualizar la información correctamente, procederemos a listar el estado de los nodos y por consiguiente vamos a tirar uno de los dos nodos para simular que no está operativo (en nuestro caso detendremos el nodo 1 de

cassandra) y acto seguido vemos con el comando ***"nodetool status"*** como está caído uno de ellos y seguimos accediendo a la misma información ya que la ofrece el segundo nodo teniendo así un servicio ininterrumpido.

Listar estado de nodos.

```
root@cassandra1:/home/usuario# nodetool status
Datacenter: dc1
===============
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address         Load       Tokens       Owns (effective)  Host ID                               Rack
UN  10.0.0.7  228,17 KiB  256          100,0%            e5316223-2467-4951-b27f-d8e953f6bbc1  rack1
UN  10.0.0.9  218,85 KiB  256          100,0%            28413744-5586-4e7e-ac9c-a05e8625c65c  rack2
```

Para simular que el nodo 1 está detenido, lo que vamos a realizar es, detener el proceso de cassandra para que ya no pueda ofrecer el servicio al otro nodo.

```
root@cassandra1:/home/usuario# systemctl stop cassandra.service
```

Listamos el estado de los nodos en el segundo nodo que sigue operativo y vemos como nos muestra que el nodo 1 de cassandra no está ofreciendo servicio(mostrando un ***DN*** al principio de la línea, lo cual indica que esta apagada y su estado es normal).

```
root@cassandra2:/home/usuario# nodetool status
Datacenter: dc1
===============
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address         Load       Tokens       Owns (effective)  Host ID                               Rack
DN  10.0.0.7  285,61 KiB  256          100,0%            e5316223-2467-4951-b27f-d8e953f6bbc1  rack1
UN  10.0.0.9  218,85 KiB  256          100,0%            28413744-5586-4e7e-ac9c-a05e8625c65c  rack2
```

Acto seguido vamos a realizar una consulta desde el cliente al segundo nodo y vemos cómo responde correctamente y además de ello muestra la información correctamente.

```
(CASSANDRA) kiki@msi:~$ cqlsh -u juanjose -p juanjose 10.0.0.9 9042 --cqlversion=3.4.4
Connected to cluster replicado at 10.0.0.9:9042.
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
juanjose@cqlsh> use comvive ;
juanjose@cqlsh:comvive> SELECT * from empleados ;

 contratacion                    | dni       | apellidos | correo              | nombre    | puesto
---------------------------------+-----------+-----------+---------------------+-----------+--------------------
 2017-05-22 12:12:10.020000+0000 | 33968574L |    Romero | lauritar@correo.com |     Laura | atencion al cliente
 2017-02-15 14:00:00.000000+0000 | 49231685Y |     Perez |    mario@correo.com |     Mario |          marketing
 2017-06-22 12:59:00.000000+0000 | 47859632H |     Lopez | elilopez@correo.com |  Elisabet |          directora
 2018-02-28 18:45:00.060000+0000 | 98765448A |     Lopez |     jjlr@correo.com | Juan Jose |            gerente
 2017-05-22 10:30:30.000000+0000 | 51968723Z |   Marquez |       jm@correo.com |      Jose |             ventas

(5 rows)
```

- A continuación, una vez probado que el segundo nodo sigue respondiendo correctamente, procederemos a iniciar de nuevo el servicio de Cassandra del primer nodo, para que se vuelvan a sincronizar los nodos y así vuelven a estar disponibles ambos nodos, respondiendo a todas la peticiones correctamente.

```
root@cassandra1:/home/usuario# nodetool status
Datacenter: dc1
===============
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address         Load       Tokens       Owns (effective)  Host ID                               Rack
UN  10.0.0.7  332,77 KiB  256          100,0%            e5316223-2467-4951-b27f-d8e953f6bbc1  rack1
UN  10.0.0.9  280,29 KiB  256          100,0%            28413744-5586-4e7e-ac9c-a05e8625c65c  rack2
```

```
root@cassandra1:/home/usuario# systemctl start cassandra.service
```

19.2. Configuración de clúster horizontal con la información replicada/repartida en dos data centers

En este segundo caso práctico, el ámbito de la repartición de la información lo veremos más claro ya que dispondremos de más equipos, y dichos equipos estarán en el mismo cluster con varios nodos que irán repartidos en los dos ***dc*** (data center) para que así exista la comunicación entre los dos centro de datos y sea lo más real posible como por ejemplo que esos dos centro de datos estén en diferentes ciudades.

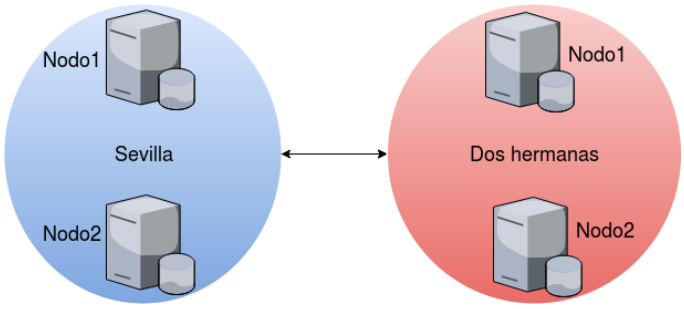
- Cassandra1 (Sevilla - Nodo1):
    - S.O : Ubuntu 16.04 Xential
    - IP: 10.10.10.101

- Cassandra2 (Sevilla - Nodo2):
    - S.O : Ubuntu 16.04 Xential
    - IP: 10.10.10.102

- Cassandra3 (Dos hermanas - Nodo1):
    - S.O : Ubuntu 16.04 Xential
    - IP: 10.10.10.103

- Cassandra4 (Dos hermanas - Nodo2):
    - S.O : Ubuntu 16.04 Xential
    - IP: 10.10.10.104

El escenario que vamos a recrear es el siguiente:



Una vez teniendo claro que escenario vamos a recrear, procederemos a la configuración del mismo. Como ya hemos comentado anteriormente, vamos a partir de que tenemos Cassandra instalado y lo que realizaremos es el proceso de configuración, así que, empezaremos con la configuración de Cassandra en dos nodos.

Para comenzar nos dirigimos a las máquinas y detendremos el proceso de Cassandra, como hemos realizado en el proceso anterior, además de eliminar la configuración que tiene Cassandra para que no entre en conflicto con ella (la configuración a eliminar es la que viene definida por defecto).

```
service cassandra stop
rm -rf /var/lib/cassandra/data/system/*
```

Acto seguido configuraremos el fichero cassandra.yaml que se encuentra en la ruta /etc/cassandra, y en él tendremos que definir los siguientes parámetros (cabe recordar que en cada nodo tendremos que adaptar la configuración a sus necesidades).

```
nano /etc/cassandra/cassandra.yaml
```

Las líneas a modificar de los ficheros de configuración de cada máquina son los que aparecen a continuación adaptando los parámetros definidos a cada máquina.

- Cassandra1:

```
cluster_name: 'CR2DC2'
num_tokens: 256
authenticator: PasswordAuthenticator
authorizer: CassandraAuthorizer
seed_provider:
      - class_name: org.apache.cassandra.locator.SimpleSeedProvider
      parameters:
      - seeds: "10.10.10.101,10.10.10.102,10.10.10.103,10.10.10.104"
listen_address: 10.10.10.101
rpc_address: 10.10.10.101
endpoint_snitch: GossipingPropertyFileSnitch
```

- Cassandra2:

```
cluster_name: 'CR2DC2'
num_tokens: 256
authenticator: PasswordAuthenticator
authorizer: CassandraAuthorizer
seed_provider:
      - class_name: org.apache.cassandra.locator.SimpleSeedProvider
      parameters:
      - seeds: "10.10.10.101,10.10.10.102,10.10.10.103,10.10.10.104"
listen_address: 10.10.10.102
rpc_address: 10.10.10.102
endpoint_snitch: GossipingPropertyFileSnitch
```

- Cassandra3:

```
cluster_name: 'CR2DC2'
num_tokens: 256
authenticator: PasswordAuthenticator
authorizer: CassandraAuthorizer
seed_provider:
      - class_name: org.apache.cassandra.locator.SimpleSeedProvider
      parameters:
      - seeds: "10.10.10.101,10.10.10.102,10.10.10.103,10.10.10.104"
listen_address: 10.10.10.103
rpc_address: 10.10.10.103
endpoint_snitch: GossipingPropertyFileSnitch
```

- Cassandra4:

```
cluster_name: 'CR2DC2'
num_tokens: 256
authenticator: PasswordAuthenticator
authorizer: CassandraAuthorizer
seed_provider:
      - class_name: org.apache.cassandra.locator.SimpleSeedProvider
      parameters:
      - seeds: "10.10.10.101,10.10.10.102,10.10.10.103,10.10.10.104"
listen_address: 10.10.10.104
rpc_address: 10.10.10.104
endpoint_snitch: GossipingPropertyFileSnitch
```

Una vez definido los parámetros en los ficheros de configuración en los cuatro nodos, podemos comprobar que los parámetros son similares a la configuración de un solo **dc** (centro de datos), y es que a la hora de configurar varios centro de datos, el diferenciar una máquina de un centro de datos a otros se define en cada maquina en el fichero *cassandra-rackdc.properties* (por ello en las semillas/seeds tendremos que indicar todos los nodos que van a estar disponibles).

Además de ello podemos observar como en la opción de **endpoint_snitch** hemos indicado una opción diferente que en la configuración en un dc (explicada anteriormente), ya que tiene la siguiente explicación:

En esa opción como ya hemos explicado anteriormente sirve para indicar a qué centro de datos y rack pertenece un nodo dentro de un clúster, pero podemos comprobar que en dicho fichero de configuración nos indica los diferentes modos de configuración que vamos a ver de forma resumida a continuación.

- Dynamic snitching.
  Supervisa el rendimiento de las lecturas de varias réplicas y elige la mejor réplica basada en este historial.

- SimpleSnitch.
  Se usa solo para implementaciones en centros de datos únicos (parámetro que viene definido por defecto).

- RackInferringSnitch.
  Determina la ubicación de los nodos por rack y centro de datos correspondiente a las direcciones IP.

- PropertyFileSnitch.
  Determina la ubicación de los nodos por rack y centro de datos.

- GossipingPropertyFileSnitch.
  Actualiza automáticamente todos los nodos usando Snitch al agregar nuevos nodos.

- Ec2Snitch.
  Esta opción se utiliza para implementaciones en amazon con una sola región.

- Ec2MultiRegionSnitch
  Se utiliza esta opción para implementaciones en Amazon EC2 donde el clúster se extiende por varias regiones.

- GoogleCloudSnitch
  Esta opción se utiliza para las implementaciones de Cassandra en Google Cloud Platform en una o más regiones.

- CloudstackSnitch
  Se utiliza esta opción para entornos con Apache Cloudstack.

Como vemos en las descripciones de las opciones que tenemos disponibles, la opción definida en nuestro caso es la única opción que nos permite agregar nuevos nodos automáticamente, por ello es la opción más recomendable para utilizar en producción y sobre todo a la hora de montar varios **dc**, ya que si necesitamos añadir algún nodo nuevo, se realizará de forma sencilla.

En el caso de que no se realice la configuración con está opción, cada vez que deseemos configurar un nuevo nodo deberemos eliminar los parámetros que se nos genera en ***/var/lib/cassandra/data/system/***, quitando así la configuración que estaba anteriormente establecida, que cassandra lo cachea y lo almacena en ese directorio para acceder a ella de forma más eficiente.

Una vez definido los parámetros anteriores procederemos a definir en cada nodo a que cluster y centro de datos pertenece. Por ello vamos a configurar el fichero de configuración que hemos indicado anteriormente denominado *cassandra-rackdc.properties* que se encuentra en */etc/cassandra* e indicaremos los siguientes datos en cada nodo que vamos a configurar.

```
nano /etc/cassandra/cassandra-rackdc.properties
```

- Cassandra1:

```
dc=sevilla
rack=nodo1
```

- Cassandra2:

```
dc=sevilla
rack=nodo2
```

- Cassandra3:

```
dc=doshermanas
rack=nodo1
```

- Cassandra4:

```
dc=doshermanas
rack=nodo2
```

Una vez que ya hayamos configurado ese fichero en todos los nodos, procederemos a eliminar el fichero denominado cassandra-topology.properties ya que los parametros de configuración son incompatibles con el tipo de configuración que estamos definiendo, así que lo eliminaremos(la eliminación de este fichero se realizará en todos los nodos que estemos configurando).

```
rm /etc/cassandra/cassandra-topology.properties
```

Acto seguido procederemos a iniciar el servicio de Cassandra en todos los nodos para que así se sincronicen los dos ***dc*** y además de ello que empiecen a cachear los nodos existentes de la red, incluyendo otros parámetros que guarda cassandra que no es más que información del resto de nodos de la red, incluyendo el propio nodo (esto lo realizan todos lo nodos de la red, por que como ya hemos visto anteriormente ningún nodo es maestro, ni esclavo, si no que todos tienen la misma importancia en el clúster).

```
service cassandra start
```

Por consiguiente, en el momento que ya se hayan iniciado todos los nodos (y esperamos unos segundos para que se sincronicen), procederemos a realizar un ***nodetool status*** para así comprobar que todos los nodos están sincronizados y centro de datos están sincronizados y repartiendo la información correctamente.

```
root@cassandra1:/home/usuario# nodetool status
Datacenter: doshermanas
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address       Load       Tokens       Owns (effective)  Host ID                               Rack
UN  10.10.10.104  273,1 KiB  256          53,9%             22c3e952-8322-4ce7-800b-2f6d4ff12868  nodo2
UN  10.10.10.103  282,91 KiB  256          46,0%             b6786ee7-b32b-4431-86f8-4c5abd53803f  nodo1
Datacenter: sevilla
===================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address       Load       Tokens       Owns (effective)  Host ID                               Rack
UN  10.10.10.102  282,42 KiB  256          50,1%             33dd12f5-a1c6-4ca3-a372-1453c6192961  nodo2
UN  10.10.10.101  254,68 KiB  256          50,0%             a70511ae-2609-4b64-b67e-0f1e30cbf847  nodo1
```

Si se dá el caso en el que nos aparece en algún nodo menos o data center de lo esperado, procederemos a dirigirnos a ese nodo y realizaremos un ***nodetool status*** para comprobar que error nos devuelve.

En el caso que nos devuelva el error en el cual que nos diga que no puede conectar con localhost o 127.0.0.1, por que el socket no se ha generado o no lo encuentra, quiere decir que existe una incoherencia entre lo que tenemos indicado en la configuración y lo que tenemos cacheado en la caché de Cassandra. Por ello, detendremos el proceso de Cassandra, además de eliminar la información del directorio system de Cassandra y volveremos a iniciar el servicio de Cassandra para que así aplique la configuración que le hemos indicado.

```
service cassandra stop
rm -rf /var/lib/cassandra/data/system/*
service cassandra start
```

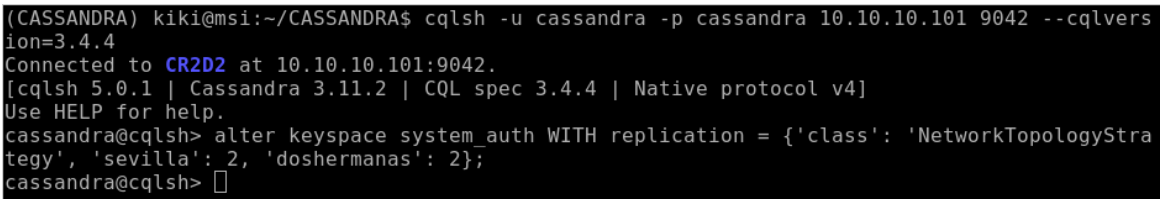
Una vez solucionado este error, procederemos a empezar a poblar los dos data centers de Cassandra de la siguiente forma.

19.2.1. Prueba de cluster horizontal en dos data centers

A continuación procederemos a poblar el clúster de los dos centro de datos que hemos creado, para empezar lo que vamos a realizar es la creación del keyspace, siendo este el primer paso que cambia con respecto a la configuración de un solo datacenter.

- Para empezar accederemos a uno de los nodos de los dos data centers (no importa al que accedemos ya que la información estará replicada y repartida por todos los nodos y será accesible desde cualquier lado), en nuestro caso accederemos al nodo1 del datacenter denominado Sevilla. Y lo primero que realizaremos será alterar la configuración del keyspace denominado ***system_auth***, para que la creación de los usuario se replique por todos los nodos de ambos data centers para así poder acceder con el mismo usuario en cualquier nodo.

  Accederemos al nodo y procederemos a introducir dicha línea.




```
alter keyspace system_auth WITH replication = {'class':
'NetworkTopologyStrategy', 'sevilla': 2, 'doshermanas': 2};
```

En comparación con la configuración de un solo data center, cambiaremos el tipo de replicado, se simple a modo ***'NetworkTopologyStrategy'*** que consiste en replicar la información el número de veces por los data centers indicados en dicha línea, en nuestro caso vamos a replicar tantas veces como nodos tenemos en cada data center.

- Acto seguido crearemos el keyspace en el que vamos a trabajar con el usuario que crearemos en el siguiente paso y le asignaremos sus correspondientes privilegios para así desplegar la información en ese keyspace en los dos ***dc***.

```
cassandra@cqlsh> create keyspace if not exists comvive with
replication = { 'class' : 'NetworkTopologyStrategy', 'sevilla': 2,
'doshermanas': 2};
```

- Una vez configurado el keyspace y creado el keyspace denominado ***comvive*** procederemos a crear el usuario por el cual, accederemos desde cualquier punto de la red.

```
cassandra@cqlsh> create user if not exists juanjose with password
'juanjose' superuser;
cassandra@cqlsh> GRANT ALL ON KEYSPACE comvive TO juanjose;
```

- A continuación, saldremos del usuario root denominado cassandra y accederemos con el usuario que acabamos de crear, pero en este caso accederemos desde otro nodo de otro dc para ver como el keyspace de system_auth se ha replicado correctamente.

```
(CASSANDRA) kiki@msi:~$ cqlsh -u cassandra -p cassandra 10.10.10.101
9042 --cqlversion=3.4.4
Connected to CR2D2 at 10.10.10.101:9042.
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol
v4]
Use HELP for help.
cassandra@cqlsh> exit

(CASSANDRA) kiki@msi:~$ cqlsh -u juanjose -p juanjose 10.10.10.104
9042 --cqlversion=3.4.4
Connected to CR2D2 at 10.10.10.104:9042.
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol
v4]
Use HELP for help.
juanjose@cqlsh>
```

- Una vez accedido con el usuario que gestionará el keyspace, procederemos a poblarlos, realizando en ese keyspace la creación de las siguientes tablas e inserción de datos en las mismas, siendo los siguientes datos a introducir (pero antes realizaremos el use de ese keypace):

```
juanjose@cqlsh> use comvive;
```

Creación de tablas

```
CREATE TABLE empleados (
      dni text,
      nombre text,
      apellidos text,
      correo text,
      puesto text,
      contratacion timestamp,
      PRIMARY KEY (contratacion,dni));

CREATE TABLE transportes (
      tipo text,
      origen text,
      destino text,
      duracion time,
      PRIMARY KEY (tipo, duracion));
```

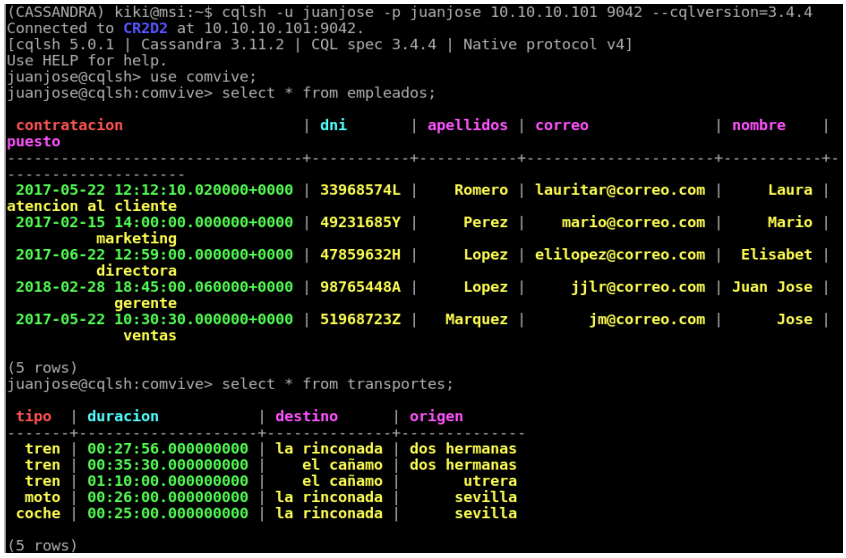
Inserción de datos tabla empleados

```
INSERT INTO empleados (dni, nombre, apellidos, correo, puesto,
contratacion) VALUES ('49231685Y', 'Mario', 'Perez',
'mario@correo.com', 'marketing','2017-02-15 15:00:00');
INSERT INTO empleados (dni, nombre, apellidos, correo, puesto,
contratacion) VALUES ('51968723Z', 'Jose', 'Marquez',
'jm@correo.com', 'ventas','2017-05-22 12:30:30.000');
INSERT INTO empleados (dni, nombre, apellidos, correo, puesto,
contratacion) VALUES ('33968574L', 'Laura', 'Romero',
'lauritar@correo.com', 'atencion al cliente','2017-05-22
14:12:10.020');
INSERT INTO empleados (dni, nombre, apellidos, correo, puesto,
contratacion) VALUES ('47859632H', 'Elisabet', 'Lopez',
'elilopez@correo.com', 'directora','2017-06-22 14:59:00.00');
INSERT INTO empleados (dni, nombre, apellidos, correo, puesto,
contratacion) VALUES ('98765448A', 'Juan Jose', 'Lopez',
'jjlr@correo.com', 'gerente','2018-02-28 19:45:00.060');
```

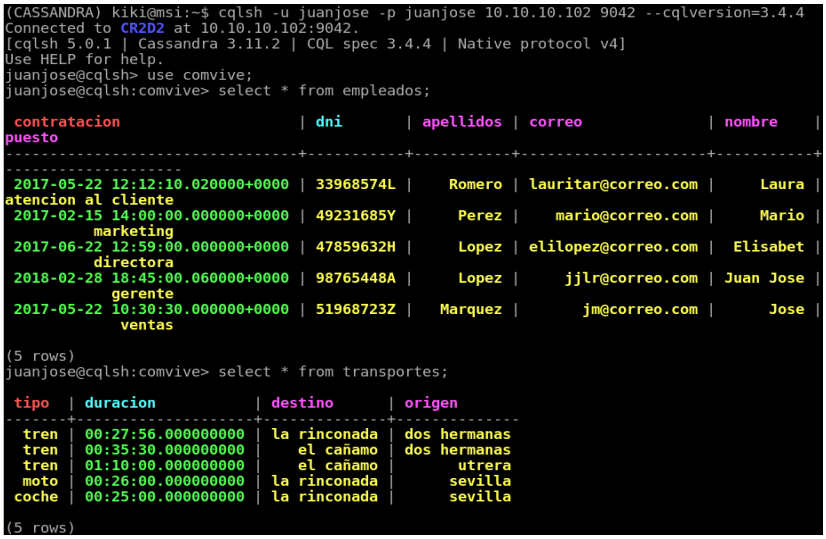
Inserción de datos tabla transportes

```
INSERT INTO transportes (tipo, origen, destino, duracion) VALUES
('tren','dos hermanas','el cañamo','00:35:30');
INSERT INTO transportes (tipo, origen, destino, duracion) VALUES
('tren','dos hermanas','la rinconada','00:27:56');
INSERT INTO transportes (tipo, origen, destino, duracion) VALUES
('tren','utrera','el cañamo','01:10:00');
INSERT INTO transportes (tipo, origen, destino, duracion) VALUES
('coche','sevilla','la rinconada','00:25:00');
INSERT INTO transportes (tipo, origen, destino, duracion) VALUES
('moto','sevilla','la rinconada','00:26:00');
```

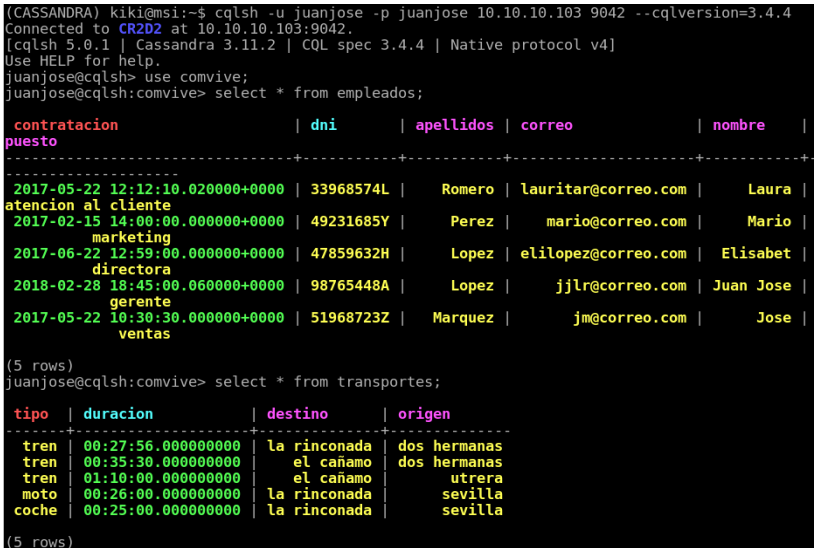
- En el momento en el que tengamos todos los datos introducidos, procederemos a realizar el listado de los datos desde los 4 nodos de los dos dc para así comprobar que se ha replicado correctamente.
    - Cassandra1



- Cassandra2



- Cassandra3



- Cassandra4

```
(CASSANDRA) kiki@msi:~$ cqlsh -u juanjose -p juanjose 10.10.10.104 9042 --cqlversion=3.4.4
Connected to CR2D2 at 10.10.10.104:9042.
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
juanjose@cqlsh> use comvive;
juanjose@cqlsh:comvive> select * from empleados;

 contratacion                    | dni       | apellidos | correo              | nombre    |
puesto
---------------------------------+-----------+-----------+---------------------+-----------+-
------------------
 2017-05-22 12:12:10.020000+0000 | 33968574L |    Romero | lauritar@correo.com |     Laura |
atencion al cliente
 2017-02-15 14:00:00.000000+0000 | 49231685Y |     Perez |    mario@correo.com |     Mario |
         marketing
 2017-06-22 12:59:00.000000+0000 | 47859632H |     Lopez | elilopez@correo.com |  Elisabet |
          directora
 2018-02-28 18:45:00.060000+0000 | 98765448A |     Lopez |     jjlr@correo.com | Juan Jose |
          gerente
 2017-05-22 10:30:30.000000+0000 | 51968723Z |   Marquez |       jm@correo.com |      Jose |
           ventas

(5 rows)
juanjose@cqlsh:comvive> select * from transportes;

 tipo  | duracion          | destino      | origen
-------+-------------------+--------------+-------------
  tren | 00:27:56.000000000 | la rinconada | dos hermanas
  tren | 00:35:30.000000000 |    el cañamo | dos hermanas
  tren | 01:10:00.000000000 |    el cañamo |       utrera
  moto | 00:26:00.000000000 | la rinconada |      sevilla
 coche | 00:25:00.000000000 | la rinconada |      sevilla

(5 rows)
```

- Acto seguido, cuando ya tengamos todos los datos introducidos y replicados en todos los nodos procederemos a listar los nodos para su posterior prueba, siendo en este caso como vemos que la efectividad a cambiado, mostrando que cada nodo es 100% efectivo, ya que la información está replicada por partes iguales en todos los nodos de forma equitativa.

```
root@cassandra1:/home/usuario# nodetool status
Datacenter: doshermanas
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address       Load       Tokens       Owns (effective)  Host ID                               Rack
UN  10.10.10.104  336,88 KiB  256          100,0%            22c3e952-8322-4ce7-800b-2f6d4ff12868  nodo2
UN  10.10.10.103  312,51 KiB  256          100,0%            b6786ee7-b32b-4431-86f8-4c5abd53803f  nodo1
Datacenter: sevilla
===================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address       Load       Tokens       Owns (effective)  Host ID                               Rack
UN  10.10.10.102  311,84 KiB  256          100,0%            33dd12f5-a1c6-4ca3-a372-1453c6192961  nodo2
UN  10.10.10.101  319,47 KiB  256          100,0%            a70511ae-2609-4b64-b67e-0f1e30cbf847  nodo1
```

19.2.2. Prueba de clúster horizontal con dos data centers

A continuación cuando ya tengamos listo los nodos con toda la información introducida en Cassandra, procederemos a ponerlo a prueba, por ello vamos a situarnos en el caso que se cae el centro de datos de sevilla, pudiendo así seguir ofreciendo el servicio a todos los clientes el segundo centro de datos denominado dos hermanas.

Para ello vamos a detener el servicio de Cassandra en los nodos que conforman el centro de datos de Sevilla, impidiendo el acceso a ellos y viéndose en la necesidad de responder las peticiones, el nodo denominado dos hermanas.

```
service cassandra stop
```

En el momento en el que se haya detenido el servicio, podemos comprobar cómo todo sigue funcionando sin problema alguno, aunque en realidad se ha detenido todo un centro de datos(pero lo que importa es que el servicio sea ininterrumpido).

```
root@cassandra3:/home/usuario# nodetool status
Datacenter: doshermanas
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address       Load       Tokens       Owns (effective)  Host ID                               Rack
UN  10.10.10.104  336,88 KiB  256          100,0%            22c3e952-8322-4ce7-800b-2f6d4ff12868  nodo2
UN  10.10.10.103  312,51 KiB  256          100,0%            b6786ee7-b32b-4431-86f8-4c5abd53803f  nodo1
Datacenter: sevilla
===================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address       Load       Tokens       Owns (effective)  Host ID                               Rack
DN  10.10.10.102  311,84 KiB  256          100,0%            33dd12f5-a1c6-4ca3-a372-1453c6192961  nodo2
DN  10.10.10.101  319,47 KiB  256          100,0%            a70511ae-2609-4b64-b67e-0f1e30cbf847  nodo1
```

Los nodos afectados muestran la coletilla de ***DN*** que significa que está en modo normal y está en Down (que quiere decir apagado o bajado).

Cuando ya hayamos verificado que todo el datacenter de Sevilla este caido, comprobamos si los datos lo sigue ofreciendo la base de datos a pesar de lo ocurrido y como vemos en la siguiente imagen, si, está ofreciendo los datos correctamente.

```
(CASSANDRA) kiki@msi:~$ cqlsh -u juanjose -p juanjose 10.10.10.104 9042 --cqlversion=3.4.4
Connected to CR2D2 at 10.10.10.104:9042.
[cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
juanjose@cqlsh> use comvive;
juanjose@cqlsh:comvive> select * from transportes;

 tipo  | duracion          | destino     | origen
-------+-------------------+-------------+-------------
  tren | 00:27:56.000000000 | la rinconada | dos hermanas
  tren | 00:35:30.000000000 |    el cañamo | dos hermanas
  tren | 01:10:00.000000000 |    el cañamo |      utrera
  moto | 00:26:00.000000000 | la rinconada |      sevilla
 coche | 00:25:00.000000000 | la rinconada |      sevilla

(5 rows)
juanjose@cqlsh:comvive> select * from empleados;

 contratacion                    | dni       | apellidos | correo             | nombre    | puesto
---------------------------------+-----------+-----------+--------------------+-----------+--------------------
 2017-05-22 12:12:10.020000+0000 | 33968574L |    Romero | lauritar@correo.com |     Laura | atencion al cliente
 2017-02-15 14:00:00.000000+0000 | 49231685Y |     Perez |    mario@correo.com |     Mario |           marketing
 2017-06-22 12:59:00.000000+0000 | 47859632H |     Lopez | elilopez@correo.com |  Elisabet |           directora
 2018-02-28 18:45:00.060000+0000 | 98765448A |     Lopez |     jjlr@correo.com | Juan Jose |             gerente
 2017-05-22 10:30:30.000000+0000 | 51968723Z |   Marquez |       jm@correo.com |      Jose |              ventas

(5 rows)
```

Y si se dá el caso en el que se recupera los nodos caídos o el datacenter caído, automáticamente se volverá a añadir gracias al modo que hemos definido en la configuración de Cassandra.

```
root@cassandra1:/home/usuario# nodetool status
Datacenter: doshermanas
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address       Load       Tokens       Owns (effective)  Host ID                               Rack
UN  10.10.10.104  336,88 KiB  256          100,0%            22c3e952-8322-4ce7-800b-2f6d4ff12868  nodo2
UN  10.10.10.103  312,51 KiB  256          100,0%            b6786ee7-b32b-4431-86f8-4c5abd53803f  nodo1
Datacenter: sevilla
===================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address       Load       Tokens       Owns (effective)  Host ID                               Rack
UN  10.10.10.102  311,84 KiB  256          100,0%            33dd12f5-a1c6-4ca3-a372-1453c6192961  nodo2
UN  10.10.10.101  319,47 KiB  256          100,0%            a70511ae-2609-4b64-b67e-0f1e30cbf847  nodo1
```

Pero si se da el caso en el que, la base de datos se lleva el suficiente tiempo como para que en otros nodos se realicen algún tipo de movimiento de los datos ¿Que pasa con la información de esos nodos caídos?

19.2.3. ¿Qué ocurriría si se modifican los datos a lo largo del tiempo y los nodos sufren caídas?

En el caso en el que se realizan algún tipo de actualización de datos, inserción de datos nuevos o eliminación de los mismo de forma masiva, estando un nodo, varios nodos o un data center completo caído y se vuelve a levantar y a unir, después de esa actualización de datos se quedaría la base de datos con la información de forma incoherente ya que dependiendo del nodo, la información varía y así no se estaría confirmando uno de los principios de cassandra que es la consistencia de los datos en todos lo nodos.

Por ello Cassandra nos ofrece una herramienta que sirve para sincronizar todos los nodos y así tener siempre la información lo más actualizada posible en todos los nodos o centro de datos de Cassandra. Por esta misma razón en la documentación oficial de Cassandra, nos recomienda pasar con cierta periodicidad el comando repair para así evita que exista información incoherente en la base de datos. Ya que en una base de datos que tenga una gran carga de información y modificación de los mismos, con el paso del tiempo la información entre los nodos no suele ser la misma por ello se utiliza la herramienta denominada ***nodetool repair***.

Esta herramienta no es más que un sistema que interconecta todos los nodos de Cassandra y empiezan a compartir la información eligiendo siempre la que esté más actualizada y por ello descartando la que esté obsoleta (que no es más que marcarla para ser eliminada periódicamente como se explicó anteriormente con las SSTables).

Por que ya no es solo si un nodo se cae, también puede existir la posibilidad que a la hora de actualizar la información, en el nodo en el que se actualiza dicho dato, mande el aviso al resto de los nodos y puede darse el caso en el que la red está congestionada y ese aviso no llegue correctamente a la totalidad de los

nodos, por ello el ***repair*** fuerza ese aviso y así nuestra información siempre estará disponible y tendrá la veracidad que necesita porque en todo los puntos del cluster estará el mismo dato.

Dicha herramienta es invocada con el comando ***nodetool repair*** y ofrece diferentes opciones de reparación que se adapta a las necesidades del usuario, eligiendo así la más adecuada en cada caso, siendo las opciones que aparecen a continuación:

Dato de interés: Todas las opciones que aparecerán a continuación se tienen que ejecutar siempre que todos los nodos involucrados estén activos, por qué de este modo la reparación se realizará de forma completa.

La reparación se puede realizar de varias formas:

- Número de nodos que realicen la reparación:
    - Paralelo ejecuta la reparación en todos los nodos con la misma réplica de datos al mismo tiempo (opción por defecto a partir de Cassandra 3.0 y posteriores).

    - Secuencial ejecuta la reparación en un nodo y después en otro (opción predeterminada en Cassandra 2.1 y posteriores).

```
nodetool repair [-seq, --secuencial]
```

    - Centro de datos en paralelo, combina secuencial y paralelo al ejecutar simultáneamente una reparación secuencial en todos los centro de datos, un solo nodo en cada centro de datos ejecuta la reparación, uno tras otro hasta que se complete la reparación.

```
nodetool repair [-dcpar, --dc-parallel]
```

- Cantidad de datos que reparan:
    - La reparación completa compara todas las réplicas de los datos almacenados en el nodo donde se ejecuta el comando y actualiza cada réplica a la versión mas nueva.(Es opción predefinida a partir de la versión 3.11 y superiores).

```
nodetool repair -full
```

    - La reparación completa con el rango del particionador repara solo las réplicas principales de los datos almacenados en el nodo donde se ejecuta el comando. Esta opción es recomendada para el mantenimiento rutinario.

```
nodetool repair [-pr, --partitioner-range]
```

- La reparación incremental divide los datos por SSTables reparados y no reparados, solamente repara los datos no reparados y va marcando los datos como reparados o no reparados.

```
nodetool repair -inc
```

Además de las formas de reparación descritas anteriormente, también existen multitud de opciones que se describirán a continuación de forma resumida:

Opción corta	Opción larga	Descripción
-h	--host	Nombre de host o la dirección ip del nodo.
-p	--port	Número de puerto.
-pwf	--password-file	Ruta del archivo de contraseña.
-pw	--password	Contraseña.
-u	--username	Nombre de usuario.
-		Separa los parámetros del comando de una lista de opciones.
-dc “dc_name”	--in-dc “dc_name”	Repara nodo en el centro de datos nombrado (los nombres distinguen de mayusculas y minusculas).
-dcpar	--dc-parallel	Repara de forma paralela los dc, reparando un nodo despues de otro en todos los dc a la vez.
-et “end_token”	--end-token “end_token”	Repara un rango de todos comenzando por el primer token (see-token) y terminando con el token indicado.
-full	--full	Hace una reparación completa, comparando todas las replicas de los datos en el nodo donde se ejecuta y actualiza cada replica a la versión mas reciente.
-hosts “host especifico”	--in-hosts “host especifico”	Repara solamente los host especificados.
-inc		Realiza una reparación incremental, que persisten los datos ya reparados y

		calcula solo los árboles de Merkle para SSTables que no se han reparado (se requiere realizar las reparaciones diariamente y no es la opción más recomendable a usar).
-j “job_threads”	--job-threads “job_threads”	Realiza la reparación del número de trabajos que se le indiquen. Por lo general, el número de tablas a reparar al mismo tiempo(tenga en cuenta que esta opción carga de trabajo al nodo por lo cual se recomienda mínimo: 1 y máximo: 4).
-local	--in-local-dc	Se utiliza para reparar nodos en el mismo centro de datos.
-pr	--partitioner-range	Repara solo rangos de particionado primario de los nodos (como mantenimiento rutinario se recomienda realizar lo siguiente nodetool repair -pr).
-pl	--pull	Realiza una reparación unidireccional donde los datos se transmiten desde un nodo remoto a este nodo (en el que se ejecuta).
-seq	--sequential	Realiza una reparación secuencial, que ejecuta la reparación en un nodo tras otro.
-st “start token”	--start-token “start token”	Realiza la reparación empezando por el rango indicado.
-tr	--trace	Rastrea la reparación hecha. Los resultados se registran en ***system_tracer.events***.
keyspace_name table_list		Nombre de keyspace y la lista de tablas separadas por espacios.
--		separa una opción de un argumento que podría confundirse con una opción.

Como ya hemos visto, existen infinidad de opciones disponibles a aplicar a la herramienta de reparación, pero si queremos una solución rápida y efectiva de cara a la reparación podemos introducir en nuestros nodos el comando ***nodetool repair*** con las siguientes opciones:

```
nodetool repair -dcpar -full
```

Con esta opción realizaremos la reparación de forma paralela en todos los dc, reparando un nodo detrás de otro a la vez en todos los nodos y además de ello la realizará de forma completa, por lo cual es una de las opciones más conveniente de cara a mantener la coherencia de datos en todos los nodos de Cassandra.

En el caso práctico utilizaremos esta opción siendo la mayor efectiva bajo mi opinión, por consiguiente procederemos con el caso práctico.

19.2.4. Caso práctico de incoherencia de datos y su recuperación

A continuación vamos a realizar la prueba de insertar un dato estando dos nodos apagados y una vez actualizada dicha información, volveremos a activar dichos nodos y en el momento en el que estén todos incorporados al clúster, realizaremos un ***nodetool repair*** para así sincronizar todos los nodos y establecer la información correctamente en todos los clusters o data centers del sistema y así tener la información siempre correcta en todo los nodos.

Muestra de nodos caídos (detenemos los servicios de cada nodo).

```
root@cassandra3:/home/usuario# nodetool status
Datacenter: doshermanas
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address       Load       Tokens       Owns (effective)  Host ID                               Rack
UN  10.10.10.104  336,88 KiB  256          100,0%            22c3e952-8322-4ce7-800b-2f6d4ff12868  nodo2
UN  10.10.10.103  312,51 KiB  256          100,0%            b6786ee7-b32b-4431-86f8-4c5abd53803f  nodo1
Datacenter: sevilla
===================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address       Load       Tokens       Owns (effective)  Host ID                               Rack
DN  10.10.10.102  311,84 KiB  256          100,0%            33dd12f5-a1c6-4ca3-a372-1453c6192961  nodo2
DN  10.10.10.101  319,47 KiB  256          100,0%            a70511ae-2609-4b64-b67e-0f1e30cbf847  nodo1
```

Inserción de datos.

```
juanjose@cqlsh:comvive> INSERT INTO transportes (tipo, origen, destino,
duracion) VALUES ('bicicleta','la rinconada','el cañamo','00:03:00');
```

Inicio del centro de datos de Sevilla (levantar ambos nodos).

```
service cassandra start
```

Inserción del comando *nodetools repair* en cada nodo (lo introduciremos en cada nodo desactualizado).

Cassandra1:

```
usuario@cassandra1:~$ nodetool repair -dcpar -full
[2018-04-25 18:42:01,528] Starting repair command #1 (94036970-48a7-11e8-
b547-5f840b928375), repairing keyspace comvive with repair options
(parallelism: dc_parallel, primary range: false, incremental: false, job
threads: 1, ColumnFamilies: [], dataCenters: [], hosts: [], # of ranges:
1024, pull repair: false)
[2018-04-25 18:42:10,274] Repair session 94b469f0-48a7-11e8-b547-
5f840b928375 for range [(6256307940113694137,6289315474965287656],
(7387340269506528958,7392072966026730899], (-2236636589844717549,-
2236173174540618007], (8993435237176419172,9017453525252576174], (-
7673706300969988034,-7665127760601001628], (-7304411767142383737,-
7294821414104402254],....
```

Cassandra2:

```
usuario@cassandra2:~$ nodetool repair -dcpar -full
[2018-04-25 18:48:01,246] Starting repair command #1 (6a704b90-48a8-11e8-
b96f-418e8fefd8ca), repairing keyspace comvive with repair options
(parallelism: dc_parallel, primary range: false, incremental: false, job
threads: 1, ColumnFamilies: [], dataCenters: [], hosts: [], # of ranges:
1024, pull repair: false)
[2018-04-25 18:48:07,961] Repair session 6ad94690-48a8-11e8-b96f-
418e8fefd8ca for range [(6256307940113694137,6289315474965287656],
(7387340269506528958,7392072966026730899], (-2236636589844717549,-
2236173174540618007], (8993435237176419172,9017453525252576174], (-
7673706300969988034,-7665127760601001628], (-7304411767142383737,-
7294821414104402254], (7240938126541889995,7280243276098014342],
(8753026677695742557,8766803330776919515], (-5757059749595050128,-
5754949942373548156], (-5013456094685431284,-5009180348737405641], (-
3976886036643671888,-3950379591203266716],
(4883107001221047532,4893827837438276560],....
```

Cuando termina la reparación, nos mostrará algo parecido a lo siguiente.

Cassandra1:

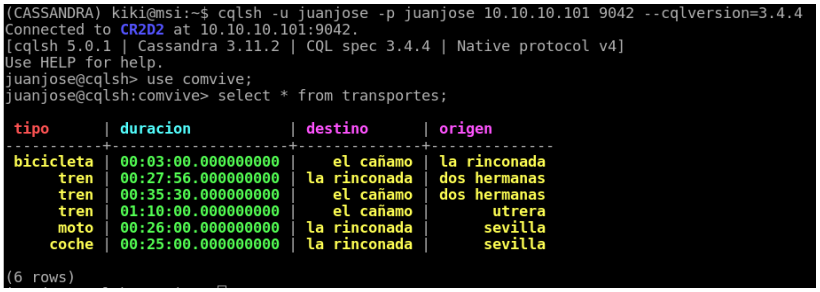
```
...., (6921857072128605034,6934169094749128800], (-5707486827144558971,-
5668690158451203569], (-2460960953865209057,-2444775189174262316],
(1385600283228257119,1388637394549535471], (-1946944510838250234,-
1929238251381291438], (4084305483786575594,4139551067912898908],
(8880935705345435997,8976839849715345987]] finished (progress: 1%)
[2018-04-25 18:42:24,709] Repair completed successfully
[2018-04-25 18:42:24,713] Repair command #3 finished in 2 seconds
usuario@cassandra1:~$
```

Cassandra2:

```
...., (-6265090600188786832,-6252976768090639988],
(6771649013334620566,6774390235346201492], (-2809680545105256900,-
2791381589558361927], (1668160501337984212,1675756008514815243],
(3892884072450831638,3895572792743466378],
(7425333610299928159,7431402605679274420], (-2765923242341103568,-
2686230596836291469], (2812142175315562239,2853452978290577016], (-
2245055674029752320,-2241127322611188102], (-2241127322611188102,-
2236636589844717549]] finished (progress: 1%)
[2018-04-25 18:48:19,771] Repair completed successfully
[2018-04-25 18:48:19,775] Repair command #3 finished in 2 seconds
usuario@cassandra2:~$
```

Por último consultamos la información a esos nodos y veremos cómo está la información actualizada correctamente.

Cassandra1:



Cassandra2:

20. Instalación y configuración de Cassandra en contenedores con Kubernetes

En este apartado veremos cómo además de configurar Cassandra en nodos físicos, también podemos tener Cassandra configurado en contenedores para que así la tolerancia a fallos sea mejor e incluso la facilidad de eliminar y crear contenedores sea más sencilla, consiguiendo así una configuración de Cassandra elástica en cuanto a la cantidad de nodos en ese cluster o data center.

En nuestro caso, el software de orquestación y gestor de contenedores a utilizar es Kubeadm junto con Docker que ya damos por hecho que lo tendremos todo instalado y configurado correctamente para centrarnos solamente en el ámbito de la configuración y despliegue de Cassandra en contenedores. En nuestro caso la instalación de kubernetes con kubeadm la hemos realizado con tres nodo, de los cuales uno de ellos es el master (el manual a seguir en el proceso de instalación a sido el siguiente: kubernetes con kubeadm).

Una vez que ya tengamos Kubernetes listo para su uso, procederemos a realizar la configuración y despliegue de los contenedores de Cassandra (cabe recordar que en todo momento vamos a utilizar la imagen oficial de Cassandra que nos ofrece para los contenedores).

20.1. Creación del servicio sin cabeza de Cassandra

Para empezar crearemos el servicio sin cabeza para que los nodos de cassandra obtengan dirección ip, así que para ello vamos a crear el siguiente fichero .yaml y le indicaremos las líneas que aparece a configuración.

```
nano cassandra-peer-service.yml
```

Indicamos las siguientes líneas:

```
apiVersion: v1
kind: Service
metadata:
  labels:
      name: cassandra-peers
  name: cassandra-peers
spec:
  clusterIP: None
  ports:
      - port: 7000
      name: intra-node-communication
      - port: 7001
      name: tls-intra-node-communication
  selector:
      name: cassandra
```

Acto seguido cargamos la configuración en Kubernetes:

```
kubectl create -f cassandra-peer-service.yml
```

En el caso en el que devuelva algún tipo de información en el momento de la creación del servicio, quiere decir que no se ha creado correctamente, así que es recomendable activar el modo depuración para obtener más información sobre el error.

20.2. Creación del servicio de Cassandra para los pods

A continuación realizaremos la creación del servicio para los Pods de Cassandra el cual posteriormente se realizará el despliegue de los pods, para poder proceder a la creación del servicio de Cassandra, tendremos que crear el fichero denominado cassandra-service.yaml y en él definiremos las características del servicio para Cassandra.

```
nano cassandra-service.yaml
```

En dicho fichero indicaremos las siguientes líneas:

```
apiVersion: v1
kind: Service
metadata:
  labels:
      name: cassandra
  name: cassandra
spec:
  ports:
      - port: 9042
      name: cql
  selector:
      name: cassandra
```

Acto seguido cargaremos la configuración que hemos definido en el fichero:

```
kubectl create -f cassandra-service.yaml
```

Una vez cargada la configuración, procederemos a comprobar que todo se ha creado correctamente y con ello la creación del servicio de Cassandra. Para comprobar que se ha creado correctamente el servicio y que ya está en pleno funcionamiento tendremos que introducir el siguiente comando (el cual mostrará el servicio sin cabeza/descabezado y el servicio para los pods de cassandra):

```
usuario@kubernetes1:~/cassandra-cuarto-intento$ kubectl get svc
NAME              TYPE         CLUSTER-IP   EXTERNAL-IP   PORT(S)              AGE
cassandra         ClusterIP    10.105.84.160    <none>    9042/TCP             24s
cassandra-peers   ClusterIP    None             <none>    7000/TCP,7001/TCP    31s
```

En el comento que ya esté todo correcto, pasaremos al siguiente apartado para seguir configurando cassandra en kubernetes.

20.3. Configuración de la replicación del pod Cassandra

Acto seguido configuraremos la creación y replicación de los pods de Cassandra, por ello vamos a realizar la creación del fichero denominado cassandra-replication-controller.yml y en el indicaremos los parámetros de configuración necesarios para indicar las características de la réplica del pod de Cassandra.

```
nano cassandra-replication-controller.yml
```

En el fichero irán indicadas las siguientes líneas:

```
apiVersion: v1
kind: ReplicationController
metadata:
  labels:
      name: cassandra
  name: cassandra
spec:
  replicas: 1
  selector:
      name: cassandra
  template:
      metadata:
      labels:
      name: cassandra
      spec:
      containers:
      - image: vyshane/cassandra
            name: cassandra
            env:
            - name: PEER_DISCOVERY_SERVICE
            value: cassandra-peers

            # Feel free to change the following:
            - name: CASSANDRA_CLUSTER_NAME
            value: Cassandra
            - name: CASSANDRA_DC
            value: DC1
            - name: CASSANDRA_RACK
            value: Kubernetes Cluster
            - name: CASSANDRA_ENDPOINT_SNITCH
            value: GossipingPropertyFileSnitch
            ports:
            - containerPort: 9042
            name: cql
            volumeMounts:
            - mountPath: /var/lib/cassandra/data
            name: data
      volumes:
      - name: data
            emptyDir: {}
```

En el momento en el que ya esté el fichero configurado y con los cambios guardado, procederemos a cargar dicha configuración realizada.

```
kubectl create -f cassandra-replication-controller.yml
```

Seguidamente configuraremos el número de réplicas existentes con el siguiente comando (en nuestro caso vamos a poner dos réplicas):

```
kubectl scale rc cassandra --replicas=2
```

Acto seguido comprobamos que ya tendremos las réplicas en funcionamiento con el comando utilizando anteriormente:

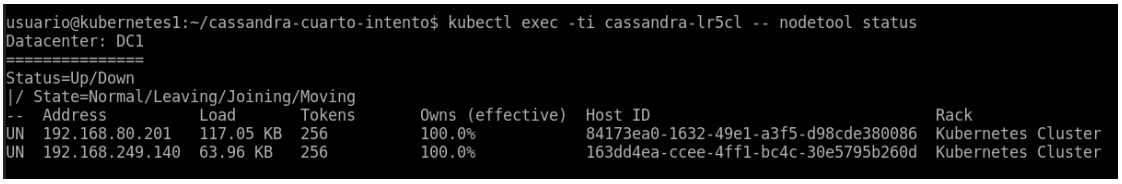
```
usuario@kubernetes1:~/cassandra-cuarto-intento$ kubectl get pods
NAME              READY       STATUS      RESTARTS   AGE
cassandra-lr5cl   1/1         Running    0           2m
cassandra-mkgd6   1/1         Running    0           1m
```

Si nos fijamos, vemos como tiene el nombre de cassandra que es entendible, seguido de un número de caracteres aleatorios, eso es por que es el pod que ha sido generado por el fichero cassandra-replication-controller, por lo que se previene la probabilidad que se repitan los nombre y existan errores, estando siempre todos los nodos identificados de forma única.

20.4. Prueba de funcionamiento

Acto seguido vamos a probar que los dos nodos creados por kubeadm están en pleno funcionamiento, para ello vamos a comprobarlo directamente en el contenedor y se comprueba con el comando exec al que le adjuntamos el comando nodetool status y verificaremos como estan los dos nodo en el mismo cluster configurados automáticamente con kubernetes.

```
kubectl exec -ti cassandra-lr5cl -- nodetool status
```

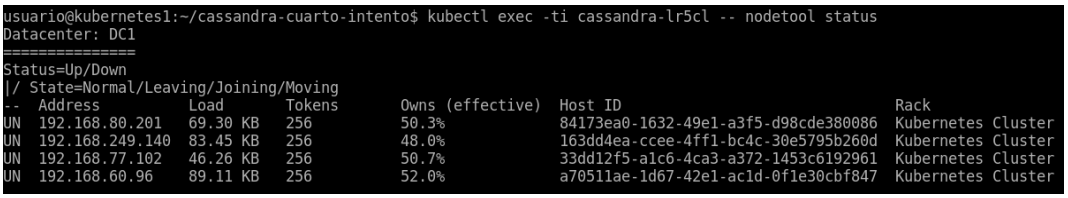


Además de ello en el momento en el que vayamos a crear alguna réplica más de Cassandra por que la necesitemos, automáticamente se nos añadirá como un nuevo nodo disponible al cluster de cassandra con el siguiente comando que utilizaremos en el siguiente ejemplo:

```
kubectl scale rc cassandra --replicas=4
```

En el momento que introduzcamos este numero de replicas, kubeadm comenzará a realizar la creación de las misma, por ello vamos a comprobar que nos lo ha creado todo correctamente y por consiguiente al realizar el nodetool veremos como todo está correctamente añadido.

```
usuario@kubernetes1:~$ kubectl get pods
NAME                READY       STATUS      RESTARTS   AGE
cassandra-5vhfh     1/1         Running      1          1d
cassandra-l82lg     1/1         Running      0          1d
cassandra-lr5cl     1/1         Running      1          1d
cassandra-mkgd6     1/1         Running      2          1d
```



Y si se da el caso en el que algún pod de cassandra se cae o es eliminado, kubeadm automáticamente generará un nuevo pod, el cual se conectará de nuevo a cluster existente y siempre mantendrá el número de réplicas que le hayamos indicado, ofreciendo una mayor tolerancia a fallos de la que ya ofrece Cassandra.

21. Monitorización de Cassandra

21.1. Monitorización con Zabbix

Para poder monitorizar Cassandra, queremos utilizar el software de monitorización por excelencia para Cassandra que es OPSCenter de la compañia DatastaX, pero el uso del mismo es inviable ya que este software actualmente va por la versión 6.5 y a partir de la versión 6.0 no es compatible con el proyecto oficial de Apache Cassandra, ya que tendremos que utilizar la distribución que la propia compañia de Datastax nos ofrece (siendo la única versión compatible), escribiendonos en la versión Enterprise.

Por ello tenemos que utilizar una alternativa que consiste en realizar la recolección de metricas a traves de JMX y así realizar la valoración del estado de los nodos de Cassandra, siendo la opción más viable el recolector de métricas denominado Zabbix (este software lo instalaremos en el nodo Cassandra1 de la instalación de los dos **dc**).

Este sistema de recolección de métricas estará instalado con diferentes componentes que son necesarios para su correcto funcionamiento, trabajando con las siguientes versiones de software y sistema operativo:

- Ubuntu Xential 16.04.
- Base de datos a monitorizar Apache Cassandra 3.11.2 (previamente instalado por toda la documentación anterior).
- Servidor base de datos Mysql 5.7.22.
- Servidor Zabbix 3.2.1.
- Zabbix Java Gateway 3.2.1.
- Zabbix Agent 3.2.1

21.1.1. Instalación de Servidor Web, Servidor de Base de datos, PHP y Extensiones.

Teniendo en cuenta los componentes que vamos a instalar vamos a proceder a realizar la instalación de una parte de todos los componentes y paquetes, introduciremos los siguientes comandos:

- Actualizamos paquetes:

```
sudo apt update
```

- Instalamos Apache:

```
sudo apt-get install apache2
```

- Instalamos MySQL

```
sudo apt-get install mysql-server
```

- Instalamos PHP y sus extensiones.

```
sudo apt-get install php7.0 php7.0-cli php7.0-common php7.0-mysql
php7.0-xml php7.0-bcmath php7.0-mbstring libapache2-mod-php7.0
```

21.1.2. Instalación de Zabbix

En el momento que hayamos instalado los componentes anteriores, procederemos a realizar la instalación de Zabbix, para ello tendremos que introducir los siguientes comandos:

- Añadimos repositorios de Zabbix y lo instalamos por paquetes .deb:

```
sudo wget
http://repo.zabbix.com/zabbix/3.2/ubuntu/pool/main/z/zabbix-
release/zabbix-release_3.2-1+xenial_all.deb
sudo dpkg -i zabbix-release_3.2-1+xenial_all.deb
sudo apt-get update
```

- Instalamos componente para MySQL:

```
sudo apt-get install zabbix-server-mysql
```

- Instalamos el frontal PHP para Zabbix:

```
sudo apt-get install zabbix-frontend-php
```

- Instalamos recolector de las métricas de JMX (métricas de cassandra) para Zabbix:

```
sudo apt-get install zabbix-java-gateway
```

- Instalamos el agente que realizará el envío de las métricas del estado de la máquina (esto lo instalaremos en todos los nodos a monitorizar, incluyendo el nodo que tiene Zabbix-Server).

```
sudo apt-get install zabbix-agent
```

21.1.3. Configuración de Zona Horaria para PHP

Acto seguido procederemos a realizar la configuración de la zona horaria, para tenerle indicada la correcta, para que esté todo correctamente configurado, para ello tendremos que configurar el fichero PHP que utiliza Apache, así que nos dirigimos a /etc/php/7.0/apache2 y editar el fichero php.ini y en dicho fichero tendremos que descomentar la línea de la zona horaria e indicar la zona horaria, en nuestro caso es Europe/Madrid.

```
nano /etc/php/7.0/apache2/php.ini
```

```
date.timezone = Europe/Madrid
```

21.1.4. Configuración del Módulo de Apache en Zabbix

En la ruta /etc/zabbix/apache.conf podemos definir diferentes parámetros de configuración relacionados con el módulo frontal de apache con zabbix, pero en nuestro caso vamos a utilizar los parámetro predefinido, por ello vamos a reiniciar el servicio de apache2 para aplicar la configuración que hemos indicado en el fichero de php anterior.

```
systemctl restart apache2
```

21.1.5. Configuración de la base de datos para Zabbix

El siguiente paso es configurar la base de datos en mysql que va a almacenar la información recogida de los datos de los nodos de Cassandra, para ello tendremos que crear la base de datos con su usuario y sus correspondientes privilegios para que pueda utilizar la base de datos correctamente:

```
CREATE DATABASE zabbixdb;
CREATE USER zabbix@'localhost' identified by 'zabbix';
GRANT ALL on zabbixdb.* to zabbix@localhost IDENTIFIED BY 'zabbix';
FLUSH PRIVILEGES;
```

Acto seguido procederemos a realizar la creación de tablas y resto de información de la siguiente forma:

```
cd /usr/share/doc/zabbix-server-mysql/
zcat create.sql.gz | mysql -u root -proot zabbixdb
```

21.1.6. Configuración del fichero de configuración del Servidor Zabbix

En el momento en el que tengamos la base de datos configurada correctamente para que Zabbix se pueda conectar a la base de datos, procederemos a modificar el fichero de configuración e indicaremos las líneas que aparecen a continuación, el fichero a modificar se denomina **zabbix_server.conf** y se encuentra en

/etc/zabbix.

```
nano /etc/zabbix/zabbix_server.conf
```

- Nombre de host donde está instalada la base de datos que hemos configurado anteriormente.

```
DBHost=localhost
```

- Nombre de la base de datos tal como se creó para el back-end de Zabbix.

```
DBName=zabbixdb
```

- Nombre de usuario de la base de datos para conectarse a la base de datos Zabbix bakc-end.

```
DBUser=zabbix
```

- Contraseña de usuario de base de datos como se especifica cuando se crea el usuario.

```
DBPassword=zabbix
```

21.1.7. Configurar Java Gateway en el servidor Zabbix

Una vez definido los parámetros de configuración anterior y guardado los cambios, procederemos a modificar los parámetros de Zabbix Java Gateway para que pueda recolectar correctamente la información de las métricas de los nodos, ya que este componente es el encargado de recolectar los datos, funciona de la misma forma al agente de Zabbix habitual pero con la diferencia que este agente recolecta la información en JMX.

Los parámetros a modificar para que recolecte correctamente la información son los siguiente y para ello tendremos que modificar el fichero de configuración denominado **zabbix_java_gateway.conf** y se encuentra en /etc/zabbix/.

```
nano /etc/zabbix/zabbix_java_gateway.conf
```

- Dirección IP que Zabbix Java Gateway escucha (para la conexión del servidor Zabbix)

```
LISTEN_IP="10.10.10.101"
```

- Número de puerto que Zabbix Java Gateway escucha en

```
LISTEN_PORT=10052
```

- Número de subprocesos de trabajo iniciados por Zabbix Java Gateway

```
START_POLLERS=5
```

Después de estos parámetros que le hemos indicado, lo guardaremos y procederemos a modificar de nuevo el fichero anterior denominado **zabbix_server.conf** que se encuentra en /etc/zabbix y en dicho fichero indicaremos los parámetros necesarios para que sepa cómo conectarse zabbix a Zabbix Java Gateway.

- Dirección IP o nombre de host donde se instala Zabbix Java Gateway

```
JavaGateway=10.10.10.101
```

- Número de puerto que Zabbix Java Gateway escucha en

```
JavaGatewayPort=10052
```

- La cantidad de manejadores de procesos dentro del servidor Zabbix que están comenzó a sondear desde los subprocesos de Zabbix Java Gateway

```
StartJavaPollers=5
```

21.1.8. Configuración de los nodos a monitorizar

En el momento en el que se ha configurado el Zabbix-Server (que es Cassandra1), procederemos a instalar los paquetes necesarios y a configurar los parámetros necesarios para que sean monitorizados los diferentes nodos de Cassandra.

Para comenzar, procederemos a instalar el agente de zabbix en todos los nodos que queramos monitorizar.

```
sudo wget http://repo.zabbix.com/zabbix/3.2/ubuntu/pool/main/z/zabbix-release/zabbix-release_3.2-1+xenial_all.deb

sudo dpkg -i zabbix-release_3.2-1+xenial_all.deb

sudo apt-get update
```

```
sudo apt-get install zabbix-agent
```

Una vez instalando los componentes necesarios, procederemos con su configuración. Para empezar, nos dirigimos al fichero ***zabbix_agentd.conf*** que se encuentra en ***/etc/zabbix/*** y lo modificaremos, indicando las siguientes líneas (estas líneas vamos a indicarlas en todos los nodos que vamos a monitorizar ya que monitorizará el estado del host).

- Indicamos que dirección ip tiene el Servidor Zabbix en el valor server.

```
Server=10.10.10.101
```

- Definiremos el puerto que vamos a utilizar.

```
ListenPort=10050
```

- Además de ello indicaremos el servidor o servidores Zabbix que están activos, en nuestro caso en cada nodo tendremos que indicar los mismos parámetros ya que en todo momento el recolector de las métricas es el mismo servidor.

```
ServerActive=10.10.10.101
```

- Y por último, en cada nodo en el apartado denominado *Hostname* tendremos que indicar el nombre de host de su respectivo host para que en el servidor zabbix pueda diferenciar un nodo de otro.

```
Hostname="Nombre de Host de ese Nodo"
```

El resto de parámetros que nos aparece los dejaremos con los valores predefinidos, ya que cumple con nuestras necesidades. Después de modificar los parámetros anteriores, guardaremos los cambios que hemos realizado en ese fichero.

Acto seguido procederemos a modificar los valores de Java de Cassandra para que pueda enviar los parámetros JMX a *Zabbix server*, ya que en todo momento lo tiene disponible solamente en el localhost, pero no de cara al exterior de la red, así que procederemos a modificar varios valores del fichero denominado ***cassandra-env.sh*** que se encuentra en ***/etc/cassandra*** y las líneas a modificar son las siguientes (esta modificación también la tendremos que realizar en cada nodo que deseamos monitorizar para que pueda responder a las peticiones del servidor Zabbix).

Se cambia el valor de **YES**:

```
if [ "x$LOCAL_JMX" = "x" ]; then
    LOCAL_JMX=yes
fi
```

Por el valor **NO**:

```
if [ "x$LOCAL_JMX" = "x" ]; then
      LOCAL_JMX=no
fi
```

Descomentamos la linea que aparece a continuación:

```
JVM_OPTS="$JVM_OPTS -Djava.rmi.server.hostname=
```

Y le indicaremos la dirección ip por la que deseamos que escuche las peticiones del servidor Zabbix

```
JVM_OPTS="$JVM_OPTS -
Djava.rmi.server.hostname=ip_por_la_que_escucha_las_peticiones"
```

Por último comentaremos la línea de autenticación a través de SSL, para que automáticamente obtenga los valores de cada nodo de Cassandra sin tener que autenticarse, esta línea se encuentra en el ***if*** que le sigue a la condición anterior de **LOCAL_JMX=no**.

```
  # JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.authenticate=true"
```

En este caso, el resultado final de este ***if*** es el siguiente:

```
if [ "x$LOCAL_JMX" = "x" ]; then
    LOCAL_JMX=no
fi

# Specifies the default port over which Cassandra will be available for
# JMX connections.
# For security reasons, you should not expose this port to the internet.  Firewall it if needed.
JMX_PORT="7199"

if [ "$LOCAL_JMX" = "yes" ]; then
  JVM_OPTS="$JVM_OPTS -Dcassandra.jmx.local.port=$JMX_PORT"
  JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.authenticate=false"
else
  JVM_OPTS="$JVM_OPTS -Dcassandra.jmx.remote.port=$JMX_PORT"
  # if ssl is enabled the same port cannot be used for both jmx and rmi so either
  # pick another value for this property or comment out to use a random port (though see CASSANDRA-7087 for origins)
  JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.rmi.port=$JMX_PORT"

  # turn on JMX authentication. See below for further options
  # JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.authenticate=true"

  # jmx ssl options
  #JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.ssl=true"
  #JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.ssl.need.client.auth=true"
  #JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.ssl.enabled.protocols=<enabled-protocols>"
  #JVM_OPTS="$JVM_OPTS -Dcom.sun.management.jmxremote.ssl.enabled.cipher.suites=<enabled-cipher-suites>"
  #JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.keyStore=/path/to/keystore"
  #JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.keyStorePassword=<keystore-password>"
  #JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.trustStore=/path/to/truststore"
  #JVM_OPTS="$JVM_OPTS -Djavax.net.ssl.trustStorePassword=<truststore-password>"
fi
```

Acto seguido guardaremos los cambios que hemos realizado en el fichero de configuración de ***cassandra-env.sh*** y reiniciaremos el servicio para aplicar los cambios con el siguiente comando:

```
systemctl restart cassandra
```

En el momento que hayamos reiniciado el servicio de Cassandra, podemos comprobar como ya el servicio de Java no está escuchando solamente para localhost, para comprobarlo tenemos que hacerlo con el siguiente comando:

```
netstat -putan | grep java
```

Una vez modificado todo lo anterior, procederemos a aplicar los cambios para que el servidor Zabbix pueda utilizar todo lo que le hemos indicado, por ello vamos a reiniciar todos los servicios de Zabbix. Reiniciamos los servicios tanto en los nodos como en el servidor Zabbix (los servicio de zabbix-server y zabbix-java-gateway solamente se reiniciará en el servidor Zabbix ya que es donde está instalado).

Servidor Zabbix.

```
sudo service zabbix-java-gateway restart
sudo service zabbix-agent restart
sudo service zabbix-server restart
```

Nodos monitorizados.

```
sudo service zabbix-agent restart
```

Si deseamos que se nos inicie todos los servicios automáticamente al arrancar la máquina después de haber sufrido un apagado, tendremos que habilitar el inicio con el arranque del sistema, con el siguiente comando (al igual que hemos explicado en el párrafo anterior el servicio de zabbix-java-gateway y zabbix-server solamente será habilitado en el servidor de Zabbix).

Servidor Zabbix.

```
systemctl enable zabbix-java-gateway
systemctl enable zabbix-agent
systemctl enable zabbix-server
```

Nodos monitorizados.

```
systemctl enable zabbix-agent
```

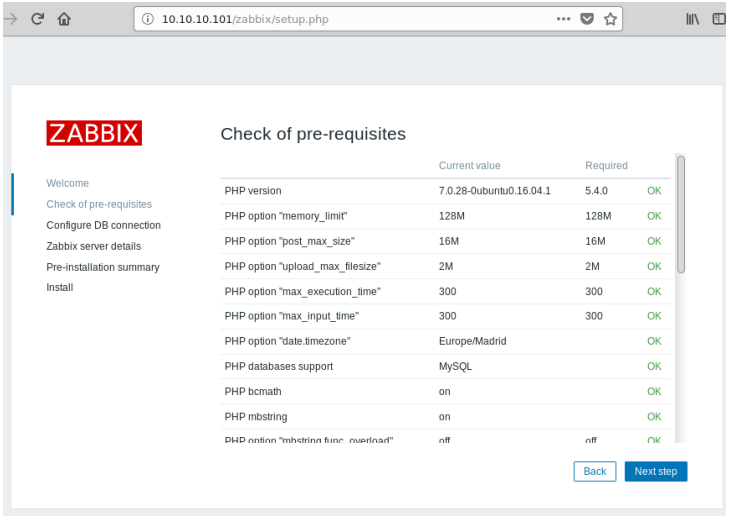
21.1.9. Instalación de zabbix a través del panel web

Cuando ya tengamos configurado todos los parámetros, procederemos a realizar la instalación del panel web de Zabbix para que sea administrado, para ello empezar, accederemos al panel web pendiente de instalar, introduciendo en la URL la dirección ip del nodo en el que esté el zabbix server instalado, seguido del nombre zabbix, tal y como aparece en el siguiente ejemplo.

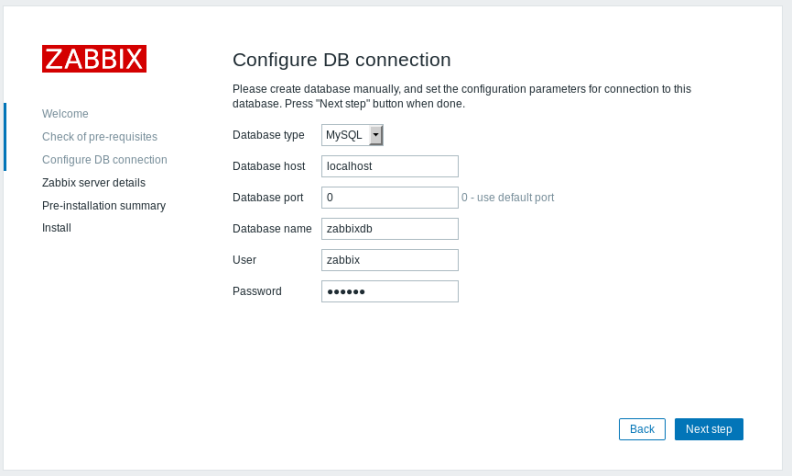
```
http://10.10.10.101/zabbix
```

En el momento que accedemos a esa URL, nos mostrará el panel de instalación de zabbix y con ello vamos a ir pasando los diferentes punto del proceso de instalación del software de monitorización.

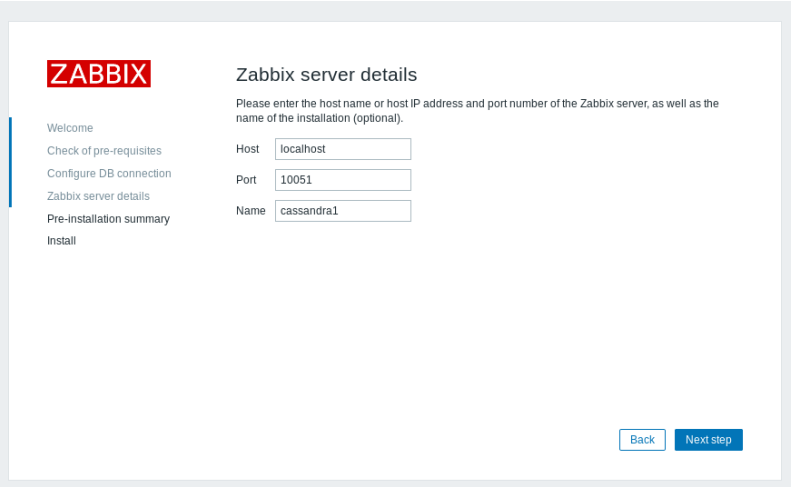
Para empezar, pasaremos por un chequeo de pre-requisitos para que el entorno en el que se va a instalar Zabbix cumpla con todos los requisitos necesarios para que funcione correctamente y para que los pueda pasar, tendremos que tener todos los valores que nos requiere en OK



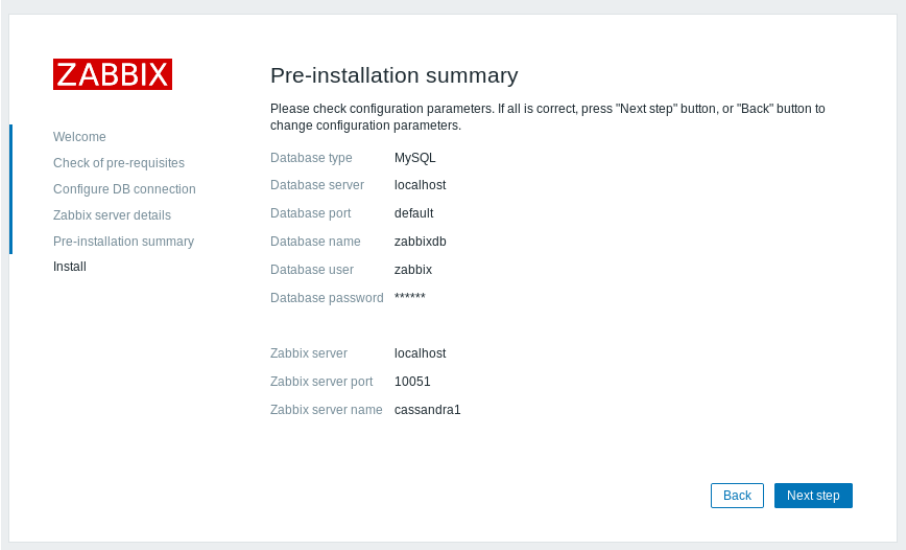
En el momento en el que esté todo correcto, procederemos a configurar donde se encuentra la base de datos que hemos creado anteriormente, en nuestro caso como la base de datos se encuentra en el mismo equipo que en el Zabbix Server, pues lo conectaremos a través de localhost.



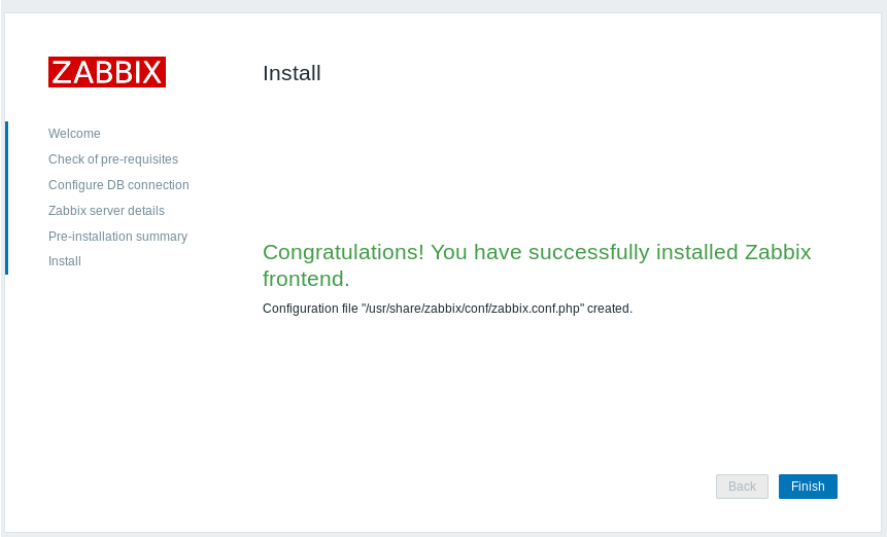
Seguidamente indicaremos en el puerto en el que Zabbix Server estará escuchando, siendo en nuestro caso los valores por defecto.



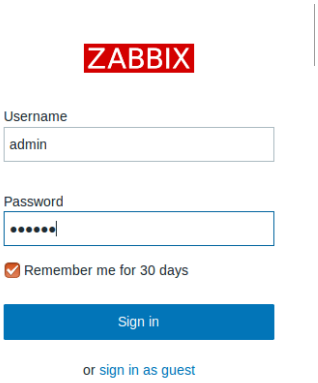
Una vez definido todos los parámetros anteriores, nos aparecerá una breve descripción de lo que hemos definido en la configuración y estando conforme con ello, seleccionaremos siguiente para proceder con la instalación.



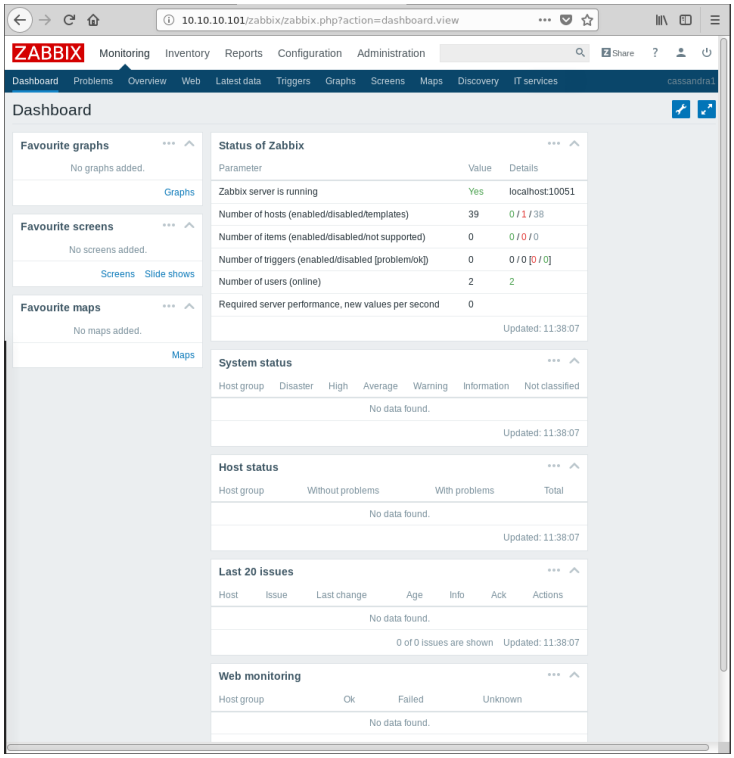
Cuando haya finalizado la configuración inicial de zabbix, nos aparecerá un mensaje diciéndonos que todo ha salido de forma correcta.



En el momento que seleccionemos finish, nos aparecerá el panel de inicio de sesión para zabbix, de forma predefinida el usuario con el que podemos acceder es de nombre de usuario *admin* y su contraseña *zabbix*.

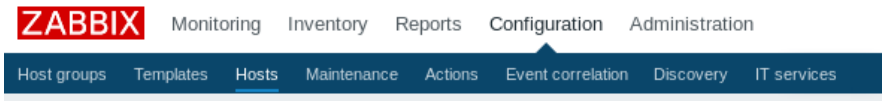


Una vez que iniciemos sesión con ese usuario, veremos el panel de administración de zabbix que estará sin ningún tipo de información a mostrar, ya que tendremos que definir los hosts (máquinas) que haya en la red que en nuestro caso son los nodos de Cassandra que son los que vamos a monitorizar.



21.1.10. Configuración de los cliente en la interfaz web de Zabbix Server

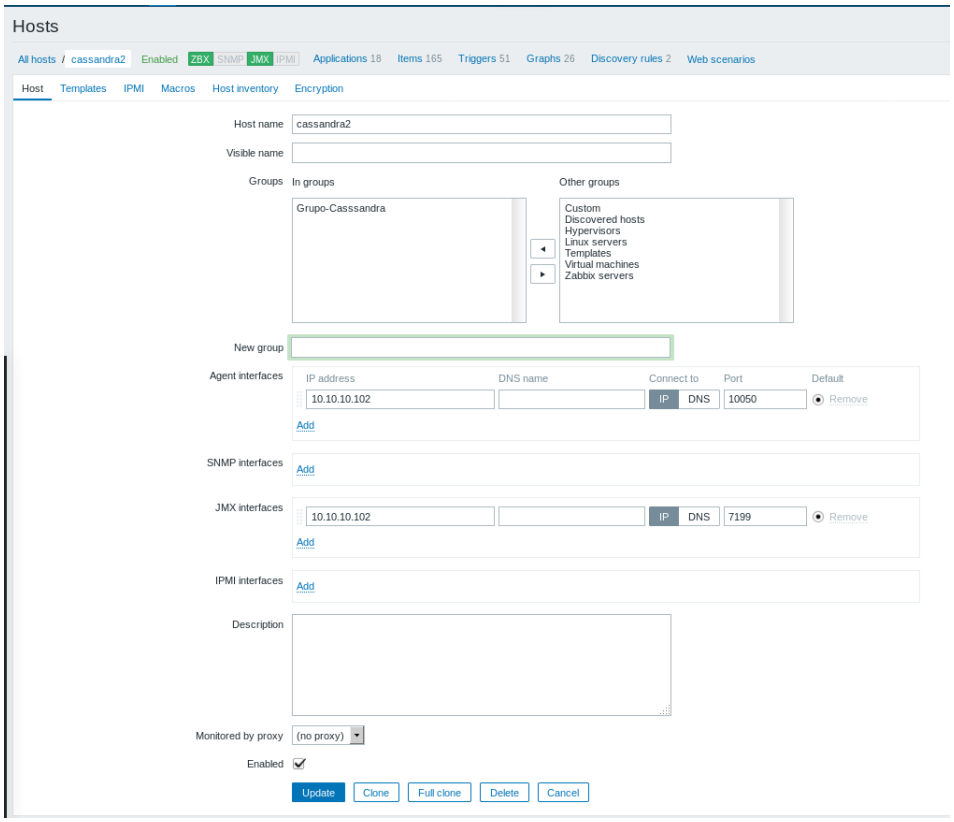
Cuando ya tengamos iniciado correctamente Zabbix Server y comprobado que podemos acceder al panel de administración, vamos a configurar los hosts que hace referencia a cada nodo, por ello nos vamos a dirigir a la pestaña de *Configuración*, en el apartado Hosts y empezaremos a definir los diferentes hosts que vamos a configurar.



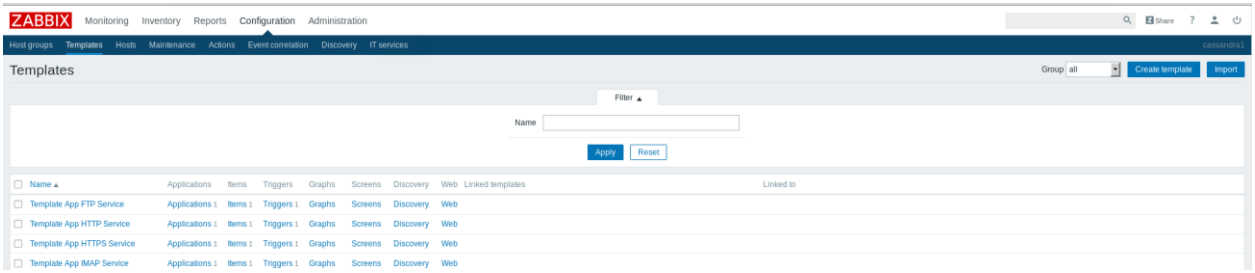
Seleccionaremos la opción denominada ***create hosts*** y empezaremos a definir los parámetros necesarios para que pueda ser monitorizado ese nodo, siendo los siguientes (esta configuración la realizaremos para cada nodo que deseamos monitorizar) :

- **Host name**: definiremos el nombre del host que vamos a monitorizar (tiene que ser el mismo nombre que le definimos en el agente de zabbix en el nodo a monitorizar).
- **Visible name**: nombre que estará disponible en el panel de monitorización.

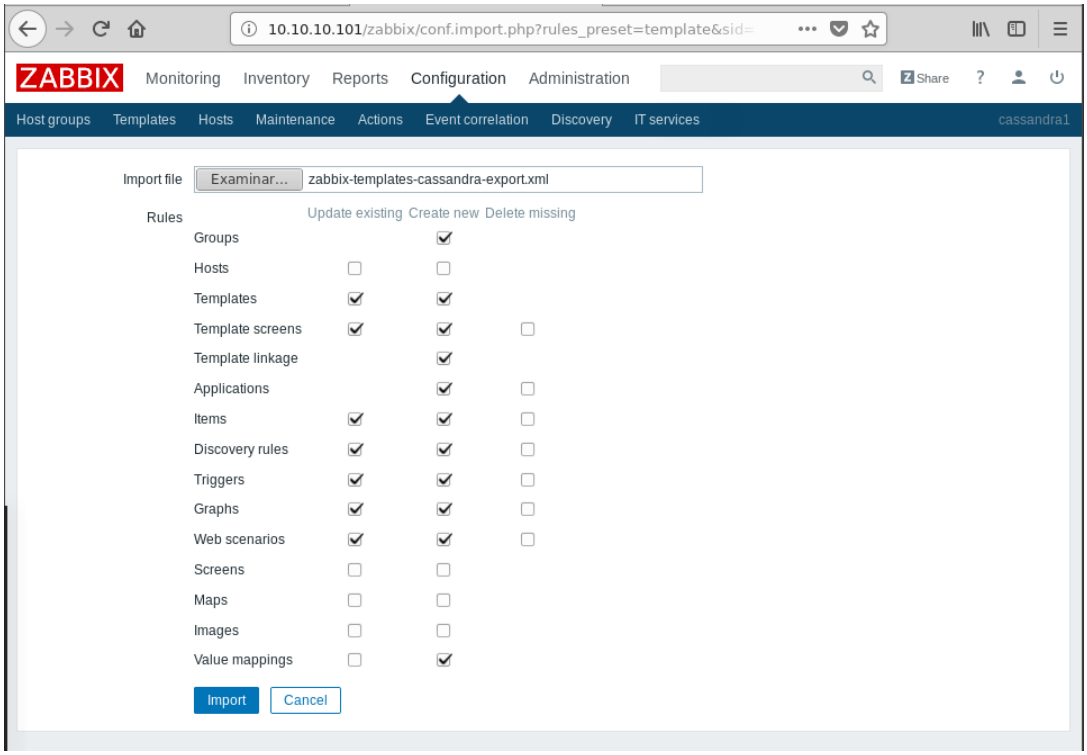
- **Groups**: nombre del grupo al que pertenece ese host para agrupar los tipos de host por grupos y que así sea más fácil de encontrar los diferentes hosts.
- **New group**: este apartado lo utilizaremos la primera vez en la configuración del primer host para crear el grupo, pero en los siguiente solamente seleccionaremos el grupo creado por que ya existirá permanentemente como un grupo de hosts más en zabbix.
- **Agent interfaces**: definiremos la dirección ip por la que está escuchando el agente de zabbix para poder recolectar los datos de las métricas.
- **JMX interfaces**: definiremos la interfaz JMX que recolectarán las métricas de Java para llevar la monitorización de cassandra.
- **Description**: breve descripción del nodo para diferenciarlo del resto.



Una vez definido los parámetros anteriores, aplicaremos los cambios establecidos y ya tendremos creado un host que hace referencia a un nodo de Cassandra para poder ser monitorizado (esto lo realizaremos en cada nodo a monitorizar), pero hasta ahora solamente tenemos configurado el host que deseamos monitorizar pero todavía no le tenemos indicado que es lo que deseamos monitorizar por ello tendremos que dirigirnos al apartado denominado ***Templates*** en la pestaña de ***Configuration*** y seleccionaremos ***Import*** ya que en nuestro caso vamos a importar un template específico para Cassandra (el template a importar nos lo descargamos del siguiente enlace: [fichero de monitorización para zabbix]()).

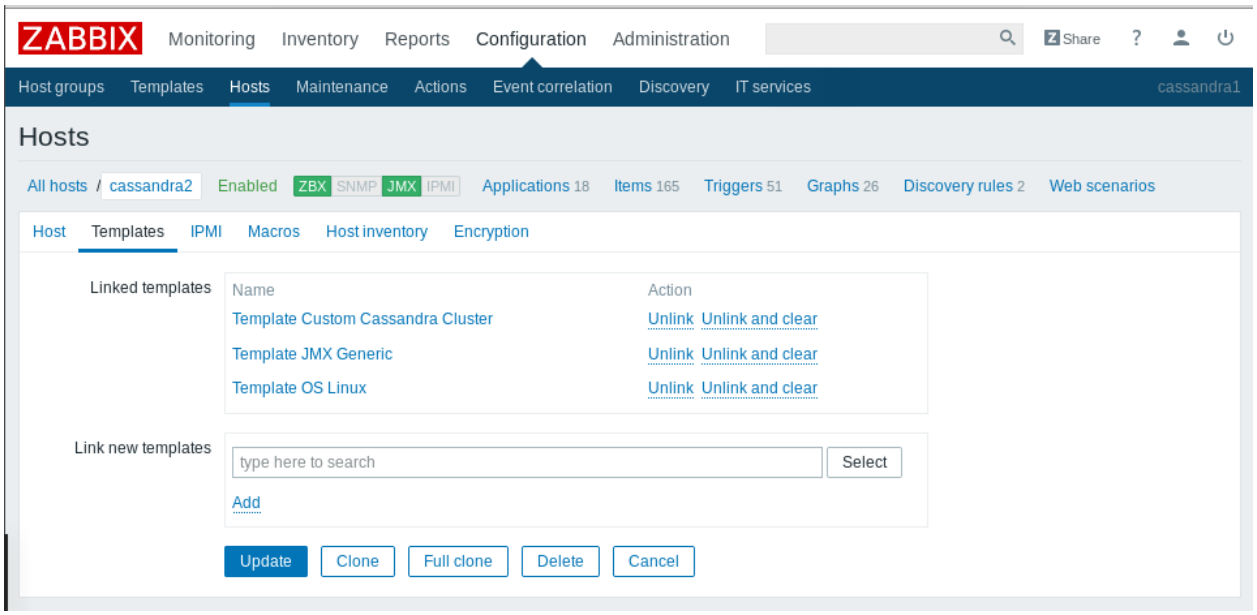


Seleccionaremos importar y dentro de él vamos a seleccionar examinar e importamos el fichero que nos descargamos del anterior enlace.



Cuando ya lo hayamos importado, volveremos al apartado de configuración de cada host y añadiremos el template que hemos importado para así indicarle a zabbix que tipo de información va a monitorizar, por ello los template que vamos a cargar son: el que hemos importado, un template genérico de JMX que nos proporciona Zabbix, además de un template que se utilizará para el Agente de Zabbix que nos permitirá monitorizar no solo el estado de Cassandra en cada nodo, si no que también monitorizará el estado de cada equipo en todo momento.

Para ello nos dirigimos al apartado de configuración de los host, como hemos realizado anteriormente y en cada host en los que vamos a monitorizar Cassandra y el estado del equipo vamos a acceder a su configuración y nos dirigimos al apartado de configuración de los template y le asignaremos los tres templates que hemos comentado (la selección de los template se tiene que realizar en cada configuración de cada Host).



Una vez que ya hayamos añadido todos los templates anteriores en cada host, procederemos a comprobar que todo se está monitorizando correctamente.

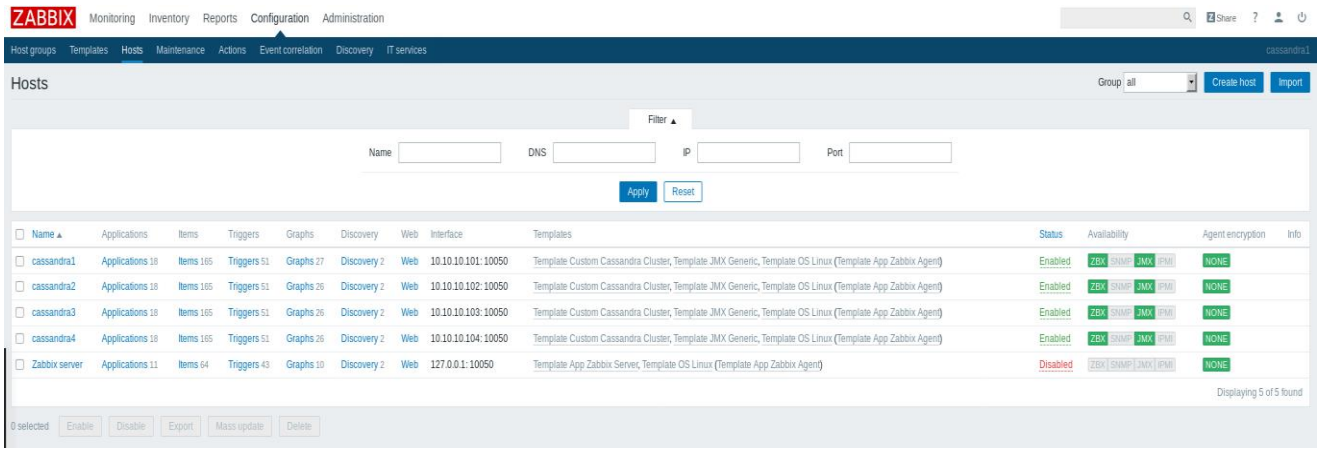
21.1.11. Comprobación de la monitorización

Cuando ya tengamos todos los parámetros configurados correctamente y además de ello hayamos importado los templates y lo hayamos añadido, procederemos a comprobar que la monitorización es satisfactoria y se está realizando correctamente.

Para ello vamos a dirigirnos al apartado de los hosts y comprobaremos como nos muestra que está activo y funcionando correctamente los dos tipo de recolecciones que le hemos definido, siendo JMX para cassandra y ZBX para la recolección a través del agente de Zabbix para obtener el estado de los nodos físicos.

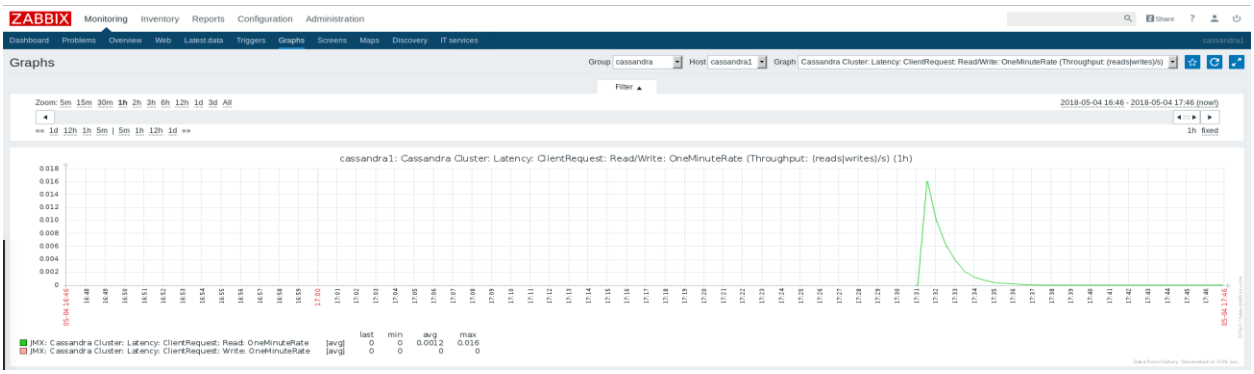
Una vez comprobado que esta todo correcto nos dirigimos al apartado de Graph que se encuentra en la pestaña de monitorización. Y desde ahí veremos cómo podemos monitorizar cassandra y el estado de los diferentes equipos que tenemos en monitorización a través del desplegable que nos ofrece Zabbix para ir

visualizando los diferentes nodos disponibles, a continuación se visualiza un ejemplo de visualización de la monitorización de los nodos.

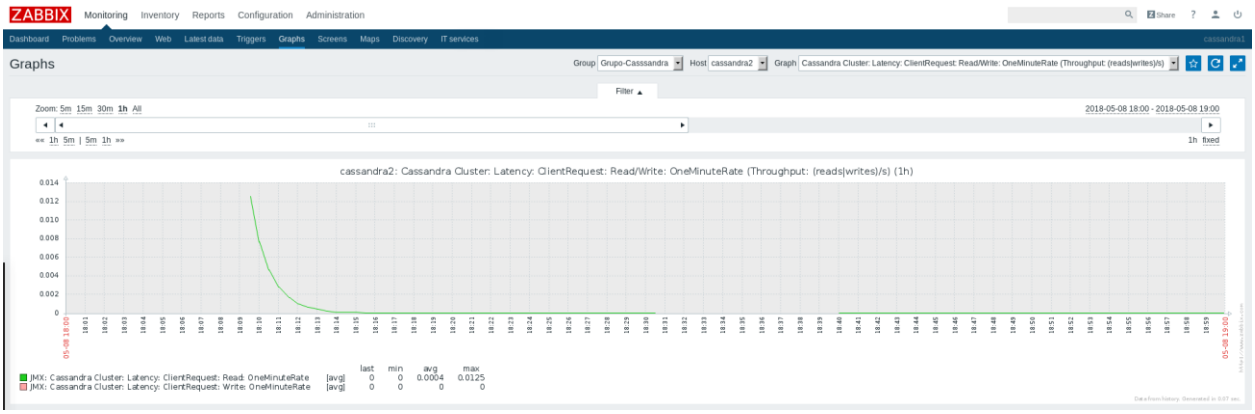


Monitorización de Cassandra (JMX)

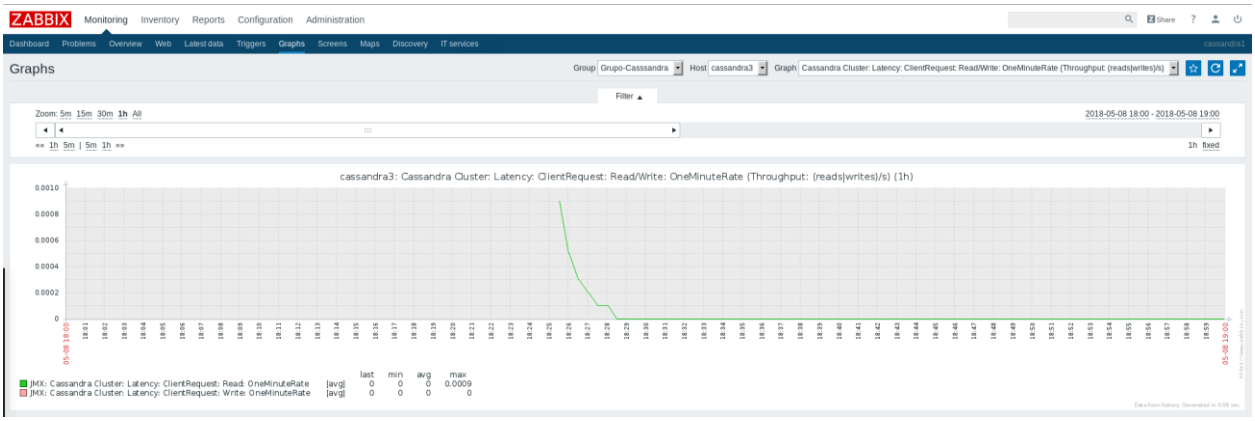
- **Cassandra1**



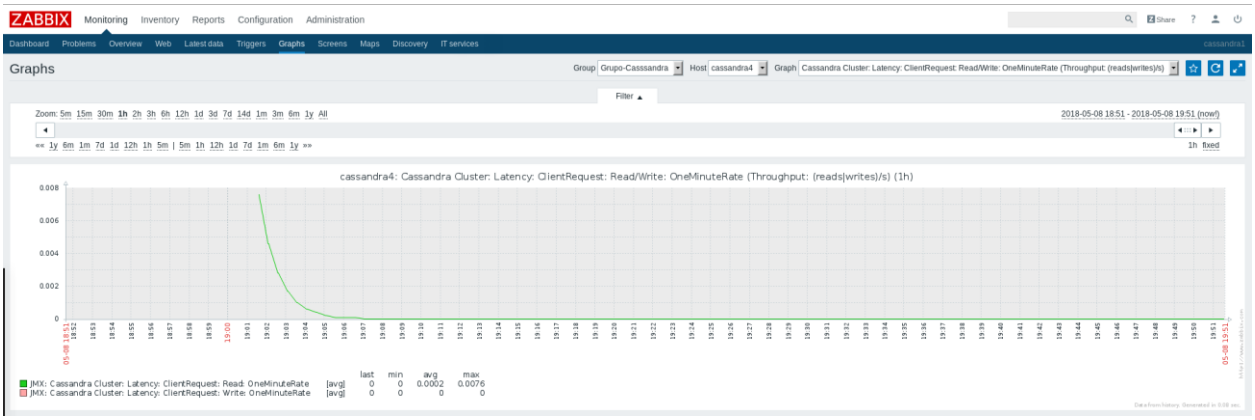
- **Cassandra2**



- **Cassandra3**

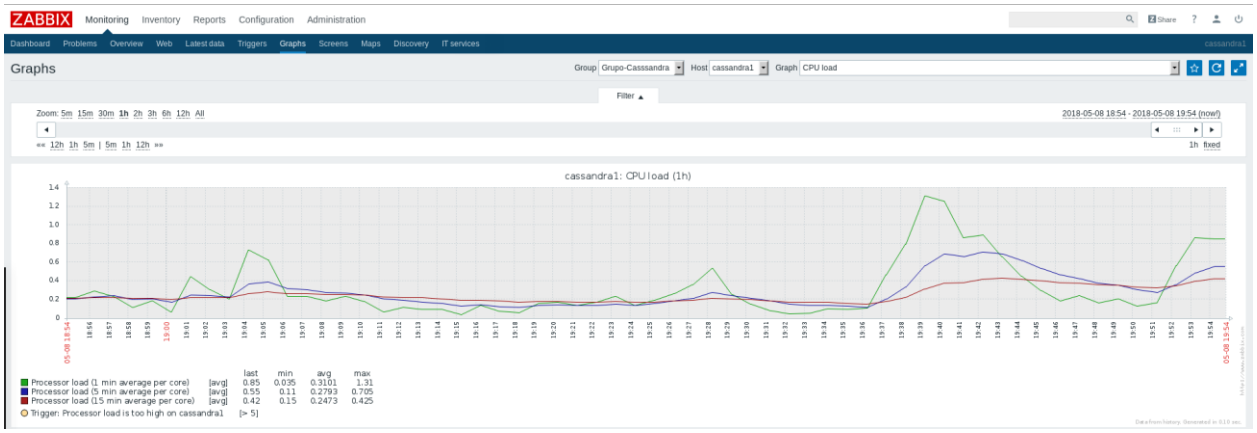


- **Cassandra4**

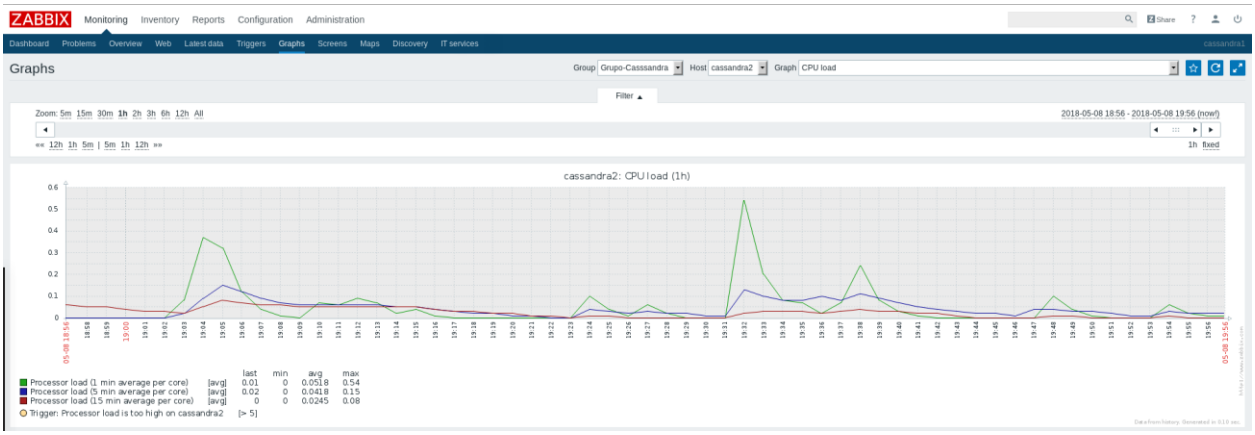


Monitorización física de los nodos (ZBX)

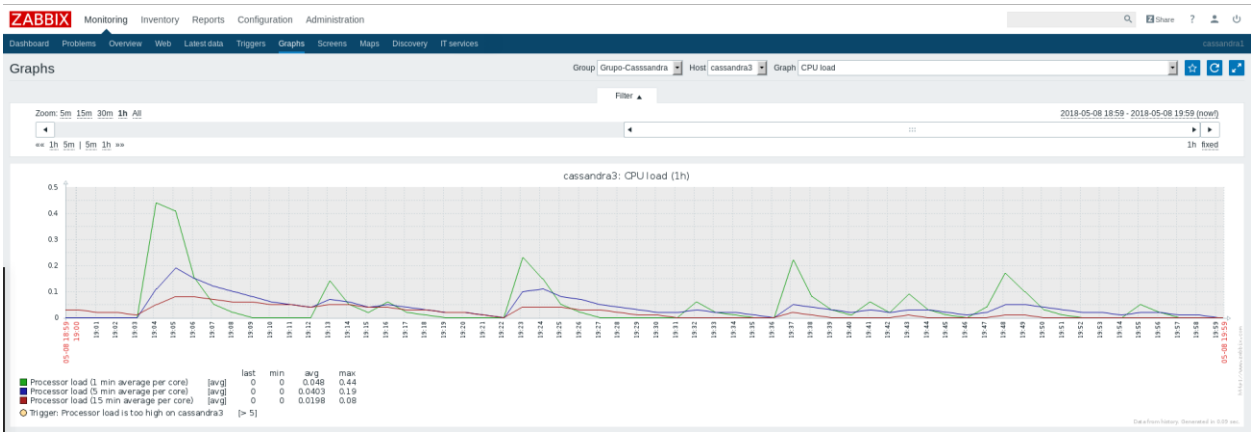
- **Cassandra1**



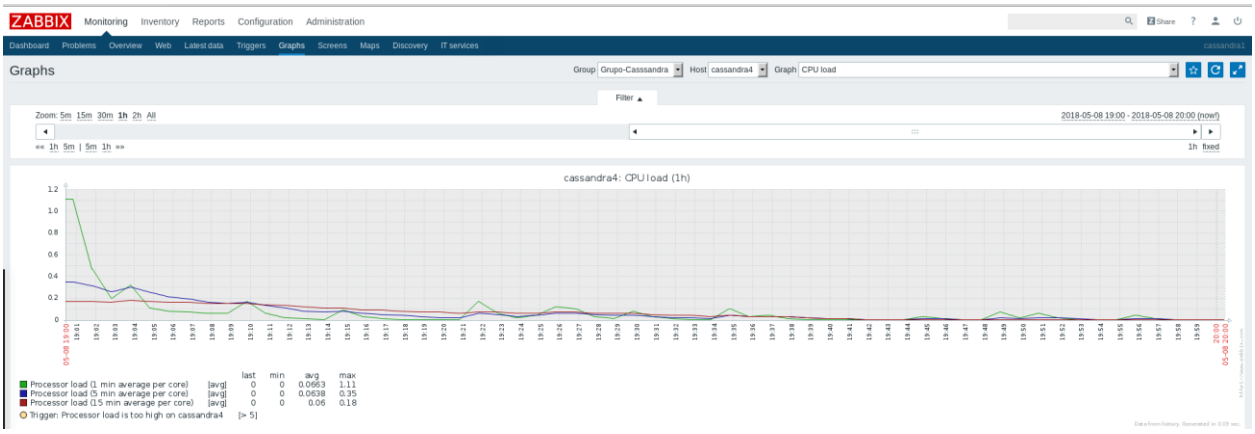
- **Cassandra2**



- **Cassandra3**



- **Cassandra4**



22. Información de interés sobre Cassandra

En este último apartado del proyecto sobre cassandra, se tratará diversos aspectos del software de interés, que son muy prácticos el conocimiento de ellos para su uso diario, ya que en las explicaciones anteriores no era apropiado divagar por diferentes temas y no profundizar en el que estábamos tratando en cada apartado.

22.1. Eliminación de nodo en el Cluster de Cassandra

Como hemos visto en el apartado de la configuración del cluster en dos data centers distintos, el aspecto de la reparación de los datos de los nodos para que la información permanezca coherente se utiliza con el comando ***nodetool***, aparte de esa utilidad que nos ofrece la instrucción ***nodetool***, también nos ofrece diversas funciones de muchísima utilidad estando entre ella la posibilidad de eliminar algún nodo que no esté operativo o que haya sido retirado, para así evitar errores en los nodos o confusiones por parte del administrador ya que en todo momento listará todos los nodos.

Para poder realizar la eliminación de un nodo concreto en un cluster tendremos que utilizar el comando ***nodetool*** indicando la opción denominada ***assassinate*** lo cual el nodo indicado con esa opción lo eliminará. A continuación vamos a conocer las diferentes opciones que le podemos indicar a esta instrucción y acto seguido realizaremos un ejemplo práctico para ver cómo se debe utilizar en el caso de querer eliminar un nodo de un cluster.

22.1.1. Eliminación básica de un nodo

El comando que vemos en el siguiente ejemplo muestra cómo se realiza la eliminación de un nodo concreto indicando la dirección ip del nodo a eliminar y utilizando las credenciales del usuario predefinido que nos ofrece cassandra que es usuario cassandra y contraseña cassandra .

```
nodetool -u cassandra -pw cassandra assassinate "ip del nodo a eliminar"
```

22.1.2. Opciones disponibles para Nodetool assassinate

Además de eliminar un nodo de la forma anteriormente especificada, también podemos indicarle las siguientes opciones disponibles.

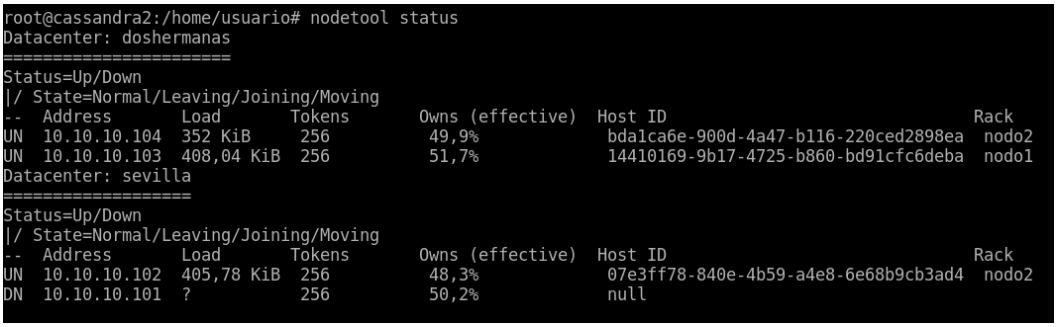
Opción corta	Opción larga	Descripción
-h	--host	Nombre de host o dirección IP
-p	--port	Número de puerto a utilizar
-pwf	--password-file	Ruta del archivo de la password
-pw	--password	Password del usuario
-u	--username	Nombre de usuario del agente remoto JMX
dirección ip		Dirección del nodo a eliminar
--		Separa una opción de un argumento.

A continuación realizaremos una pequeña práctica, en la cual se realiza la eliminación de un nodo en un cluster de Cassandra.

22.1.3. Caso práctico de eliminación de un nodo

En este caso práctico disponemos de dos data center con sus respectivos cluster con Cassandra, en los cuales un nodo de ello se ha dañado y se tiene que retirar de dicho cluster. Por ello se ha procedido a apagar dicho nodo y lo eliminaremos de la lista de nodos de Cassandra.

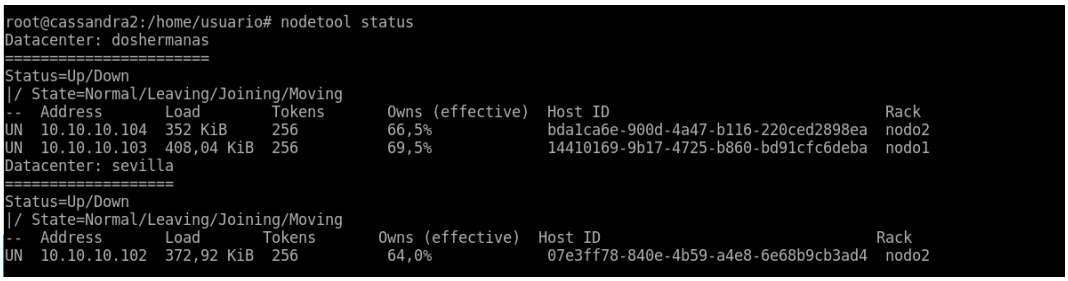
Para empezar listamos los nodos que tenemos disponibles en la red.



Como vemos en la imagen, hay un nodo el cual no está operativo, por ello vamos a introducir el siguiente comando que indicando las credenciales de Cassandra e indicando la ip del nodo que deseamos eliminar, procederá a realizar la eliminación de dicho nodo.

```
nodetool -u cassandra -pw cassandra assassinate 10.10.10.101
```

Una vez introducido el comando anterior, ya tendremos eliminado correctamente el nodo de nuestro cluster, para comprobarlo, introduciremos de nuevo el comando ***nodetool status*** y veremos como ya tenemos el nodo anterior eliminado.



22.2. Limpieza de Keyspaces y Partition Keyspaces que ya no son utilizables

Como hemos visto en la parte teórica explicada anteriormente, a la hora de eliminar un keyspace o partición de keyspace no la eliminamos realmente, lo que Cassandra realiza es marcar dicho keyspace o partición de keyspace como no valida (que no es más que marcar las SSTables pertenecientes al keyspace que deseamos eliminar) y de forma periódica, Cassandra realiza una limpieza de todas aquellas SSTables que hayan sido marcadas como no válidas.

22.2.1. Limpieza básica en un nodo

Esa limpieza se realizar a traves de la herramienta ***nodetool*** indicando la opción ***cleanup*** y con dicha opción tenemos disponible diferentes parámetros que nos ayuda a definir como deseamos que nos realice la limpieza. A continuación vamos a ver un ejemplo de cómo se formaría la sintaxis del comando:

```
nodetool -h 127.0.0.1 -p 7199 cleanup
```

Partiendo de esta sintaxis base como ejemplo, lo que estamos realizando es ejecutar una limpieza de todos aquellos keyspaces que ya no los utiliza el nodo en el que se está utilizando ya que el host que estamos indicando es localhost (127.0.0.1) y además de ello se conecta a través del puerto 7199 para poder realizar la limpieza. En el caso que hagamos uso de este ejemplo si disponemos en nuestro cluster más de un nodo, tendremos que ejecutar el comando en cada uno de los nodos.

22.2.2. Opciones disponibles de Nodetool cleanup

Además del ejemplo explicado anteriormente, vamos a ver las diferentes opciones que tenemos disponibles y son las siguientes:

Opción corta	Opción larga	Descripción
-h	--host	Nombre de Host o la dirección IP del mismo.
-j	--job	Número de SSTables a limpiar simultáneamente; por defecto usa todos los hilos de compactación disponibles.
-p	--port	Número de puerto a utilizar.
-pwf	--password-file	Ruta del archivo de contraseña.
-u	--username	Nombre de usuario del agente remoto JMX.
-pw	--password	Contraseña del usuario.
keyspace		Nombre del Keyspace a limpiar.
tabla		Uno o más nombres de tablas a limpiar, separados por un espacio.
--		Separa una opción de un argumento que podría confundirse con una opción.

Una de las recomendaciones de uso es de forma periódica, que podemos implantarlo por ejemplo en una tarea de cron en todos los nodos que deseemos para que de forma automática el nodo realice su limpieza automática. Aunque otra de las recomendaciones en cuanto a su uso es ejecutar el este comando de limpieza después de haber añadido un nuevo nodo al cluster, para que sean todos los nodos lo más equivalentes en cuanto a información se refiere.

A continuación vamos a ver cómo se realiza el uso de este comando en un caso práctico, el cual lo vamos a aplicar en nuestros dos data center conectados entre sí para que tengan la menos información residual posible. Por ello comencemos el caso práctico de limpieza de información en los nodos.

22.2.3. Caso práctico de limpieza en los nodos

En este caso práctico vamos a realizar una limpieza en los cuatro nodos que tenemos disponibles que están repartidos en los dos data centers que ofrecen el servicio de base de datos. Para comenzar introduciremos el siguiente comando en cada nodo que tenemos disponible ya que bajo mi punto de vista, es más seguro que cada equipo ejecute el suyo a que se ejecute remotamente que siempre dependerá del

estado de la red, por ello vamos a introducir el siguiente comando en cada nodo de nuestro escenario con Cassandra.

```
nodetool -h 127.0.0.1 -p 7199 cleanup -j5
```

En nuestro caso práctico en cada nodo de Cassandra, ejecutaremos el comando especificado en el cual vamos a indicar el número de hilos de a utilizar simultáneamente para así no saturar o comprometer el rendimiento y eficacia del nodo.

A continuación vamos a introducir el comando especificado en los distintos nodos para mantenerlos limpios de SSTables innecesarias.

- **Cassandra1:**

```
usuario@cassandra1:~$ nodetool -h 127.0.0.1 -p 7199 cleanup -j5
WARN  23:20:18,865 Small commitlog volume detected at /var/lib/cassandra/commitlog;
setting commitlog_total_space_in_mb to 3506.  You can override this in
cassandra.yaml
WARN  23:20:18,870 Small cdc volume detected at /var/lib/cassandra/cdc_raw; setting
cdc_total_space_in_mb to 1753.  You can override this in cassandra.yaml
WARN  23:20:18,871 Only 10,504GiB free across all data volumes. Consider adding
more capacity to your cluster or removing obsolete snapshots
jobs (5) is bigger than configured concurrent_compactors (2) on this host, using at
most 2 threads
jobs (5) is bigger than configured concurrent_compactors (2) on this host, using at
most 2 threads
jobs (5) is bigger than configured concurrent_compactors (2) on this host, using at
most 2 threads
jobs (5) is bigger than configured concurrent_compactors (2) on this host, using at
most 2 threads
```

- **Cassandra2:**

```
usuario@cassandra2:~$ nodetool -h 127.0.0.1 -p 7199 cleanup -j5
WARN  23:20:21,436 Small commitlog volume detected at /var/lib/cassandra/commitlog;
setting commitlog_total_space_in_mb to 3506.  You can override this in
cassandra.yaml
WARN  23:20:21,461 Small cdc volume detected at /var/lib/cassandra/cdc_raw; setting
cdc_total_space_in_mb to 1753.  You can override this in cassandra.yaml
WARN  23:20:21,464 Only 10,946GiB free across all data volumes. Consider adding
more capacity to your cluster or removing obsolete snapshots
jobs (5) is bigger than configured concurrent_compactors (2) on this host, using at
most 2 threads
```

```
jobs (5) is bigger than configured concurrent_compactors (2) on this host, using at
most 2 threads
jobs (5) is bigger than configured concurrent_compactors (2) on this host, using at
most 2 threads
jobs (5) is bigger than configured concurrent_compactors (2) on this host, using at
most 2 threads
```

- **Cassandra3:**

```
usuario@cassandra3:~$ nodetool -h 127.0.0.1 -p 7199 cleanup -j5
WARN  23:20:22,501 Small commitlog volume detected at
/var/lib/cassandra/commitlog; setting commitlog_total_space_in_mb to 3506.
You can override this in cassandra.yaml
WARN  23:20:22,527 Small cdc volume detected at /var/lib/cassandra/cdc_raw;
setting cdc_total_space_in_mb to 1753.  You can override this in
cassandra.yaml
WARN  23:20:22,529 Only 11,002GiB free across all data volumes. Consider
adding more capacity to your cluster or removing obsolete snapshots
jobs (5) is bigger than configured concurrent_compactors (2) on this host,
using at most 2 threads
jobs (5) is bigger than configured concurrent_compactors (2) on this host,
using at most 2 threads
jobs (5) is bigger than configured concurrent_compactors (2) on this host,
using at most 2 threads
jobs (5) is bigger than configured concurrent_compactors (2) on this host,
using at most 2 threads
```

- **Cassandra4:**

```
usuario@cassandra4:~$ nodetool -h 127.0.0.1 -p 7199 cleanup -j5
WARN  23:20:23,805 Small commitlog volume detected at
/var/lib/cassandra/commitlog; setting commitlog_total_space_in_mb to 3506.
You can override this in cassandra.yaml
WARN  23:20:23,826 Small cdc volume detected at /var/lib/cassandra/cdc_raw;
setting cdc_total_space_in_mb to 1753.  You can override this in
cassandra.yaml
WARN  23:20:23,829 Only 11,003GiB free across all data volumes. Consider
adding more capacity to your cluster or removing obsolete snapshots
jobs (5) is bigger than configured concurrent_compactors (2) on this host,
using at most 2 threads
jobs (5) is bigger than configured concurrent_compactors (2) on this host,
using at most 2 threads
```

```
jobs (5) is bigger than configured concurrent_compactors (2) on this host,
using at most 2 threads
jobs (5) is bigger than configured concurrent_compactors (2) on this host,
using at most 2 threads
```

Una vez introducido los comando anteriores en los diferentes nodos conseguiremos liberar el espacio que podemos aprovecharlo en otro tipo de información de mayor importancia que las SSTables no válidas para la base de datos.

22.3. Descripción de los clusters de Cassandra

Todo el proceso de eliminación de datos en desuso o eliminación de un nodo en un clúster es muy útil para el uso rutinario de Cassandra para intentar tener todo el cluster o clusters lo más optimizados posibles, pero sí al realizar alguna acción de las descritas en el nodo equivocado podemos poner en peligro los datos de dichos nodos. Por ello vamos a conocer una nueva opción que nos ofrece el comando ***nodetool*** y es cuando lo ejecutamos con la opción ***describecluster***.

22.3.1. Comando básico a introducir en el nodo

Este comando ***nodetool*** con la opción ***describecluster*** tiene que ser indicada en todo momento en el nodo del cual queremos obtener la información, por ello tendremos que introducir el comando de su forma más básica, siendo la siguiente:

```
nodetool describecluster
```

22.3.2. Opciones disponibles de Nodetool describecluster

En cuanto a parámetros que le indicamos al pasar a este comando con la opción ***describecluster*** es bastante similar a la herramienta anterior ya que las únicas opciones que le podemos indicar son muy básicas y son las siguientes:

Opción corta	Opción larga	Descripción
-h	--host	Nombre de Host o la dirección IP del mismo.
-p	--port	Número de puerto a utilizar.
-pwf	--password-file	Ruta del archivo de contraseña.
-u	--username	Nombre de usuario del agente remoto JMX.

-pw	--password	Contraseña del usuario.
--		Separa una opción de un argumento que podría confundirse con una opción.

Una vez que ya tengamos claro las opciones que tenemos disponible para esta herramienta, que no son más que para permitirnos realizarlo de forma remota, procederemos a realizar una prueba en nuestros nodos de Cassandra.

22.3.3. Caso practico de información de los nodos

En este caso práctico vamos a ver la información que nos ofrece los nodos con la inserción del comando.

22.3.3.1. Recabar información del cluster

Para empezar vamos a ver como muestra la información del cluster a la hora de introducir el comando deseado.

```
usuario@cassandra1:~$ nodetool describecluster
Cluster Information:
      Name: CR2D2
      Snitch: org.apache.cassandra.locator.GossipingPropertyFileSnitch
      DynamicEndPointSnitch: enabled
      Partitioner: org.apache.cassandra.dht.Murmur3Partitioner
      Schema versions:
                  54ccbbc8-d0c4-331e-863a-3c41aeb7cab3: [10.10.10.104,
10.10.10.102, 10.10.10.103, 10.10.10.101]
```

Como aparece en el resultado del comando, nos muestra como información que en este cluster disponemos de cuatro nodos, los cuales nos vienen descrito especificando la dirección ip de cada uno de ellos (no importa desde el nodo en el que se ejecute, ya que este comando recaba información de todo el cluster al completo, preguntando a todos los nodos existentes).

22.3.3.2. Conexión errónea con el resto de nodos

En el caso que no se conecte con alguno de los nodos o no exista ningún tipo de negociación con alguno de los mismo, como resultado de ese comando nos mostrará cuales han sido los nodos con los que no ha sido posible negociar para así solucionar los fallos de los nodos indicados. A continuación como ejemplo apagaremos uno de los cuatro nodos disponibles (el nodo Cassandra4) e introduciremos este comando para ver cómo nos devuelve el error que estamos comentando.

```
usuario@cassandra1:~$ nodetool describecluster
Cluster Information:
      Name: CR2D2
      Snitch: org.apache.cassandra.locator.GossipingPropertyFileSnitch
      DynamicEndPointSnitch: enabled
      Partitioner: org.apache.cassandra.dht.Murmur3Partitioner
      Schema versions:
                  54ccbbc8-d0c4-331e-863a-3c41aeb7cab3: [10.10.10.102,
10.10.10.103, 10.10.10.101]

                  UNREACHABLE: [10.10.10.104]
```

22.4. Copia de seguridad de los clusters de Cassandra

Como último punto a ver para ir finalizando la documentación, vamos a ver cómo se realiza las copias de seguridad con la herramienta nodetool, utilizando en concreto la opción snapshot, esta opción realizará una instantánea de uno o más keyspaces, o de una tabla para hacer la copia de seguridad de los datos de la misma.

A continuación veremos un ejemplo básico de como realiza la copia de seguridad de los datos.

22.4.1. Comando básico para la copia de seguridad de un nodo

Para que esta opción realice una copia de seguridad básica del nodo, lo que tendremos que realiza es introducir el comando en el nodo del que queremos realizar la copia de seguridad, y no tenemos que indicarle ningún parámetro opcional, lo único que tendremos que hacer es introducir el comando en su forma más básica y realizará la copia de seguridad de todos los keyspaces existentes.

```
nodetool snapshot
```

En el momento que hayamos introducido el comando, lo que realizará Cassandra será vaciar el nodo de tareas que puedan alterar los datos a copiar y una vez terminada esa tarea, procederá a realizar la copia de seguridad. Si en el momento de introducir el comando, no indicamos el directorio en el que deseamos que se guarde con la opción -t , automaticamente Cassandra le pondrá el nombre de marca de tiempo de cuando realiza la instantánea y realizará la copia de toda la información.

22.4.2. Opciones disponible de nodetool snapshot

A continuación vamos a ver las opciones que nos ofrece esta herramienta a la hora de utilizar la opción de snapshot, las opciones disponibles son las siguientes:

Opción Corta	Opción Larga	Descripción
-h	--host	Nombre de host o su dirección IP.
-p	--port	Número de puerto.
-pwf	--password-file	Ruta del archivo de contraseña.
-u	--username	Nombre del usuario del agente remoto JMX.
-cf “nombre tabla”	--column-family “nombre tabla”	Nombre de la tabla a la que se le hace la copia. Solo se le especifica uno y su correspondiente keyspace.
	--table	Nombre de la tabla a la que se le hace la copia. Solo se le especifica uno y su correspondiente keyspace.
-kc “nombre ktlist”	--kc-list “nombre ktlist”	La lista de keyspaces.tablas para copiar. Requiere una lista de los keyspaces
-kt “nombre ktlist”	--kt-list “nombre ktlist”	La lista de keyspaces.tablas para copiar. Requiere una lista de los keyspaces
-sf	--skip-flush	Ejecuta la copia sin realizar un flush primero en las tablas
-t	--tag	Nombre de la copia a realizar
kesypace		Nombres de los keyspaces separados por un espacio. Por defecto son todos los keyspaces
--		Separa una opción de un argumento que podría confundirse con una opción.

22.4.3. Caso práctico de copias de seguridad en los nodos

A continuación procederemos a realizar una copia de seguridad de forma práctica utilizando el mismo escenario que hemos utilizado en los casos prácticos anteriores, así que procederemos a realizar la copia de seguridad en el nodo cassandra1.

En dicho nodo lo que realizaremos es una copia de seguridad del keyspace denominado ***comvive*** y lo guardaremos con la descripción ***copia-seg-comvive***, así que para ello vamos a introducir el comando con la siguiente sintaxis.

```
nodetool snapshot comvive -t copia-seg-comvive
```

Y el resultado del comando a la hora de introducirlo en el nodo cassandra1 es el siguiente:

```
usuario@cassandra1:~$ nodetool snapshot comvive -t copia-seg-comvive
Requested creating snapshot(s) for [comvive] with snapshot name [copia-seg-
comvive] and options {skipFlush=false}
Snapshot directory: copia-seg-comvive
```

En el momento en que haya sido introducir el comando anterior, Cassandra guardará por defecto la copia de seguridad del keyspace con sus respectivas tablas en el directorio de datos de Cassandra que es en ***/var/lib/cassandra/data/,*** dentro del directorio del keyspace ***comvive***, en cada tabla almacenará la copia. A continuación mostramos las rutas donde lo almacena las copias de cada tabla que contiene el keyspace comvive.

```
/var/lib/cassandra/data/comvive/transportes-
7c0a92a052eb11e898a0ab41457431bc/snapshots/copia-seg-comvive

/var/lib/cassandra/data/comvive/empleados-
749da20052eb11e898a0ab41457431bc/snapshots/copia-seg-comvive
```

A continuación veremos de forma resumida que podremos realizar con la copia de seguridad. Ya que podemos realizar diferentes acciones una vez realizada la copia de seguridad.

22.4.3.1. ¿Qué podemos hacer con las copias de seguridad?

La razón por la cual se realizan copias de seguridad es simplemente cuando queremos tener un respaldo de la información para que en caso de que se produzca alguna anomalía en el funcionamiento habitual del software, si se produce la pérdida de datos tener un punto donde recuperarlas o si se da el caso de que se realiza cambios en el software el cual comprometa en algún momento a la integridad de los datos, por ello se realizan las copias para que encaso de perdida podamos volver a un estado anterior.

Pero una vez que ya hemos hecho la copia ¿Qué podemos hacer con ella? Pues bien las tareas o acciones que podemos hacer con la copia de seguridad es la siguiente:

- **nodetool listsnapshot**: podemos listar la cantidad de copias de seguridad realizadas con este comando para así llevar un control de las copias que se realizan a los nodo, a continuación se muestra un ejemplo de la ejecución del comando (la ejecución se realiza en el nodo directamente para obtener la información).

```
root@cassandra1:~# nodetool listsnapshots
Snapshot Details:
Snapshot name     Keyspace name Column family name True size Size on disk
copia-seg-comvive comvive     transportes       9,94 KiB  10,9 KiB
copia-seg-comvive comvive     empleados         5,34 KiB  6,31 KiB

Total TrueDiskSpaceUsed: 15,29 KiB
```

- **nodetool clearsnapshot**: podemos eliminar las copias de seguridad en uno o más keyspaces, si se da el caso que ya las copias con más antigüedad no son necesarias o simplemente queremos realizar una limpieza de copias de seguridad. Podemos especificar la copia que deseamos eliminar, pero si se da el caso que queremos eliminarlas todas, bastará con no indicar ninguna en concreto, esto hará que se eliminen todas las copias realizadas. En el ejemplo que aparece a continuación eliminaremos todas las copias que existan en el nodo.

```
root@cassandra1:~# nodetool clearsnapshot
Requested clearing snapshot(s) for [all keyspaces]
```

Además de las diferentes opciones que hemos descrito anteriormente, la herramienta nodetool está dotada de una gran variedad de funcionalidad y opciones, pero en este apartado solamente he indicado las que bajo mi punto de vista son las más interesantes de cara al uso diario de Cassandra para su gestión y administración. El resto de opciones que tenemos disponible para esta herramienta está en el siguiente enlace:

Opciones de Nodetool: http://cassandra.apache.org/doc/latest/tools/nodetool/nodetool.html

23. Conclusiones

En definitiva, Cassandra es un software muy potente a la vez que versátil y ofrece diversas soluciones gracias a su flexibilidad en cuanto a la configuración del mismo. Su principal fuerte es el uso para el Big Data como se ha comentado en varias partes del documento ya que tiene su cuota de mercado en ese tipo de uso..

Debido a sus cualidades se puede usar en muchas aspectos fuera del Big Data pero siendo consciente que no es la solución más adecuada. Lo ideal es tener claro su uso para saber sacarle el máximo partido, contemplando así las consultas, el tipo de diseño de la base de datos ya que dependiendo de cómo se realice el diseño podrá aprovechar mejor las ventajas de esta potente base de datos distribuida.

24. Trabajos futuros para la mejora del proyecto

Los aspectos a mejorar en el proyecto para un posible progreso del mismo en el futuro son los siguientes:

- Las mejoras en cuanto a configuración que ofrece cassandra con configuraciones en cloud con compañías como Amazon o Google, ya que tiene sus propios parámetros de configuración.
- Investigar las mejoras que ofrece la empresa Datastax a esta base de datos y ya no por su versión propia de Cassandra si no por el software adicional que ofrece si te registras en la web como Enterprise.
- Mejorar e investigar la configuración de Zabbix para sacarle un mejor provecho al software de monitorización por que es una herramienta con gran funcionalidad. Pero este aspecto ya se escapa en gran parte del ámbito de Cassandra por que se enfoca mas en la monitorización como tal (por la funcionalidad en cuanto a parámetros de notificación, límites de aviso, etc).
- Investigar y optimizar Cassandra con el uso de aplicaciones que usen este tipo de bases de datos y así personalizar el rendimiento software para un mejor rendimiento y una mejor optimización.
- Realizar la creación de una aplicación web basada en python que funcione como frontal para la administración de Cassandra, gracias a la utilización de su librería en python.

25. Referencias

- Videos de youtube.
    - Apache Cassandra (Introdución a Cassandra en entorno Big Data): https://www.youtube.com/watch?v=Xq1ociMAOmw
    - Cassandra para impacientes (Commit conf): https://www.youtube.com/watch?v=jxwtmMTMrJA

- Enlaces Web.
    - Documentación Zabbix: https://www.zabbix.com/documentation/3.2/start
    - JMX: http://wiki.intrusos.info/doku.php/seguridad:monitorizacion:zabbix2:jmx
    - Cliente Zabbix: https://clouding.io/kb/como-anadir-servidores-a-zabbix-agente-zabbix/
    - Documentación de Cassandra con Kubernetes:
        - https://github.com/vyshane/cassandra-kubernetes
        - https://github.com/fabric8io/gitcontroller/tree/master/vendor/k8s.io/kubernetes/examples/cassandra#tl-dr
        - https://kubernetes.io/docs/tutorials/stateful-application/cassandra/
    - Añadir DC a un Cluster: https://docs.datastax.com/en/cassandra/2.1/cassandra/operations/ops_add_dc_to_cluster_t.html
    - Cluster de dos DC: https://www.datastax.com/dev/blog/deploying-cassandra-across-multiple-data-centers
    - Monitorización con software de Datastax: https://docs.datastax.com/en/opscenter/6.1/opsc/install/opscInstallDeb_t.html
    - Definición keys de Casasndra: https://www.elconspirador.com/2018/01/29/definicion-claves-cluster-y-particion-apache-cassandra-parte-1/
    - Instalación 3 nodos de Cassandra:
        - https://www.jamescoyle.net/how-to/2438-install-datastax-cassandra-3-on-debain-ubuntu
        - https://www.jamescoyle.net/how-to/2448-create-a-simple-cassandra-cluster-with-3-nodes
    - Conf. distribuida: https://docs.datastax.com/en/archived/cassandra/1.2/cassandra/configuration/configCassandra_yaml_r.html
    - Conf. Multinodo:

        - https://www.ibm.com/developerworks/ssa/library/ba-set-up-apache-cassandra-architecture/index.html
        - https://www.digitalocean.com/community/tutorials/how-to-run-a-multi-node-cluster-database-with-cassandra-on-ubuntu-14-04

- Primeros pasos cassandra: https://geekytheory.com/primeros-pasos-con-apache-cassandra
- Diferencia con RDB: https://fireosoft.com.co/blogs/que-es-la-base-de-datos-apache-cassandra/
- Curiosidades de Cassandra: https://unpocodejava.com/2012/07/12/un-poco-de-cassandra/
- Tipos de insert: https://docs.datastax.com/en/cql/3.1/cql/cql_reference/insert_r.html
- CQL: https://code.tutsplus.com/es/articles/getting-started-with-cassandra-using-cql-api-and-cqlsh--cms-28026
- Modelado Datos: https://elmanadelainformatica.wordpress.com/2015/06/09/apache-cassandra-modelo-de-datos/
- Contenedores Cassandra: https://docs.docker.com/samples/library/cassandra/
- Contenedores Cassanda (github de Bitnami): https://github.com/bitnami/bitnami-docker-cassandra
- Tipo de replicado de datos: https://docs.datastax.com/en/cassandra/2.1/cassandra/architecture/architectureDataDistributeReplication_c.html
- Explicación resumida de Cassandra: https://www.paradigmadigital.com/dev/cassandra-la-dama-de-las-bases-de-datos-nosql/
- Cluster de 2 nodos: http://chuwiki.chuidiang.org/index.php?title=Cluster_Cassandra_con_dos_servidores
- Explicación teorica de cassandra: https://es.scribd.com/document/371439932/cassandra
- Instalación Cassandra:

- https://www.rosehosting.com/blog/how-to-install-apache-cassandra-on-ubuntu-16-04/
- https://www.digitalocean.com/community/tutorials/how-to-install-cassandra-and-run-a-single-node-cluster-on-a-ubuntu-vps