MongoDB

Sharded Cluster

Carlos García Muñoz

Índice

1. Introducción

Las bases de datos no relacionales surgieron a raíz de la necesidad de almacenar de forma rápida y sencilla cualquier tipo de información, sin importar el formato. Por lo tanto las bases de datos no relacionales no necesitan de la existencia de un esquema para guardar la información.

Otra de la ventajas que ofrecen estas bases de datos es la facilidad y sencillez de su escalabilidad horizontal para hacer frente al constante crecimiento de los datos.

Las bases de datos no relacionales no son la solución a todos los problemas, pero en los casos con gran variedad de datos y alto crecimiento de información son ideales.

Las características que presentan son las siguiente:

- Pueden incorporar literalmente cualquier tipo de dato, al tiempo que proporcionan todas las características necesarias para crear aplicaciones ricas en contenido.

- El escalado está integrado en la base de datos. Es automático y transparente. Pueden escalar a medida que crece el volumen de datos agregando nuevos nodos sin ningún tipo de problema.

A continuación se explicará con detalle el sistema de bases de datos no relacional Mongodb y su Cluster sharding que se utiliza para el escalado mencionado anteriormente.

2. Definiciones y conceptos

2.1. MongoDB

2.1.1. ¿Qué es MongoDB?

MongoDB es una base de datos orientada a documentos. Esto quiere decir que en lugar de guardar los datos en registros, guarda los datos en documentos. Estos documentos son almacenados en BSON, que es una representación binaria de JSON.

Una de las diferencias más importantes con respecto a las bases de datos relacionales, es que no es necesario seguir un esquema. Los documentos de una misma colección (concepto similar a una tabla de una base de datos relacional), pueden tener esquemas diferentes.

Aunque las colecciones de MongoDB no necesitan definir un esquema, es importante que diseñemos nuestra aplicación para seguir uno. Tendremos que pensar si necesitamos normalizar los datos, desnormalizarlos o utilizar una aproximación híbrida. Estas decisiones pueden afectar al rendimiento de nuestra aplicación. En definitiva el esquema lo definen las consultas que vayamos a realizar con más frecuencia.

2.1.2. ¿Cómo funciona MongoDB?

MongoDB está escrito en C++, aunque las consultas se hacen pasando objetos JSON como parámetro. Es algo bastante lógico, dado que los propios documentos se almacenan en BSON.

MongoDB viene de serie con una consola desde la que podemos ejecutar los distintos comandos. Esta consola está construida sobre JavaScript, por lo que las consultas se realizan utilizando ese lenguaje. Además de las funciones de MongoDB, podemos utilizar muchas de las funciones propias de JavaSciprt. En la consola también podemos definir variables, funciones o utilizar bucles.

2.1.3. ¿Dónde no se debe usar MongoDB?

En esta base de datos no existen las transacciones. Aunque nuestra aplicación puede utilizar alguna técnica para simular las transacciones, MongoDB no tiene esta capacidad. Solo garantiza operaciones atómicas a nivel de documento. Si las transacciones son algo indispensable en nuestro desarrollo, deberemos pensar en otro sistema.

Tampoco existen los JOINS. Para consultar datos relacionados en dos o más colecciones, tenemos que hacer más de una consulta. En general, si nuestros datos pueden ser estructurados en tablas, y necesitamos las relaciones, es mejor que optemos por un RDBMS clásico.

2.1.4. Características principales.

- **Alto rendimiento.**

MongoDB proporciona persistencia de datos de alto rendimiento. En Concreto:

- El soporte para modelos de datos incorporados reduce la actividad de E / S en el sistema de base de datos.

- Los índices admiten consultas más rápidas y pueden incluir claves de documentos incrustados y matrices.

- **Lenguaje de consulta.**

MongoDB admite un lenguaje de consulta enriquecido para dar soporte a las operaciones de lectura y escritura (CRUD), así como:

- Agregación de datos.

- Búsqueda de texto y consultas geoespaciales.

- **Alta disponibilidad.**

La facilidad de replicación de MongoDB, llamada conjunto de réplicas, proporciona:

- Failover automático.

- Redundancia de datos.

Un conjunto de réplicas es un grupo de servidores MongoDB que mantienen el mismo conjunto de datos, proporcionando redundancia y aumentando la disponibilidad de datos.

- **Soporte para múltiples motores de almacenamiento.**

MongoDB soporta múltiples motores de almacenamiento, tales como WiredTiger y MMAPv1.

Además, MongoDB proporciona una API de motor de almacenamiento, plugin que permite a terceros desarrollar motores de almacenamiento para MongoDB.

- **Escalabilidad horizontal.**

MongoDB proporciona escalabilidad horizontal como parte de su funcionalidad principal:

Sharding distribuye datos a través de un grupo de máquinas.
MongoDB soporta la creación de zonas de datos basadas en la clave shard. En un cluster equilibrado, MongoDB dirige las lecturas y escrituras cubiertas por una zona sólo a aquellos fragmentos dentro de la zona.

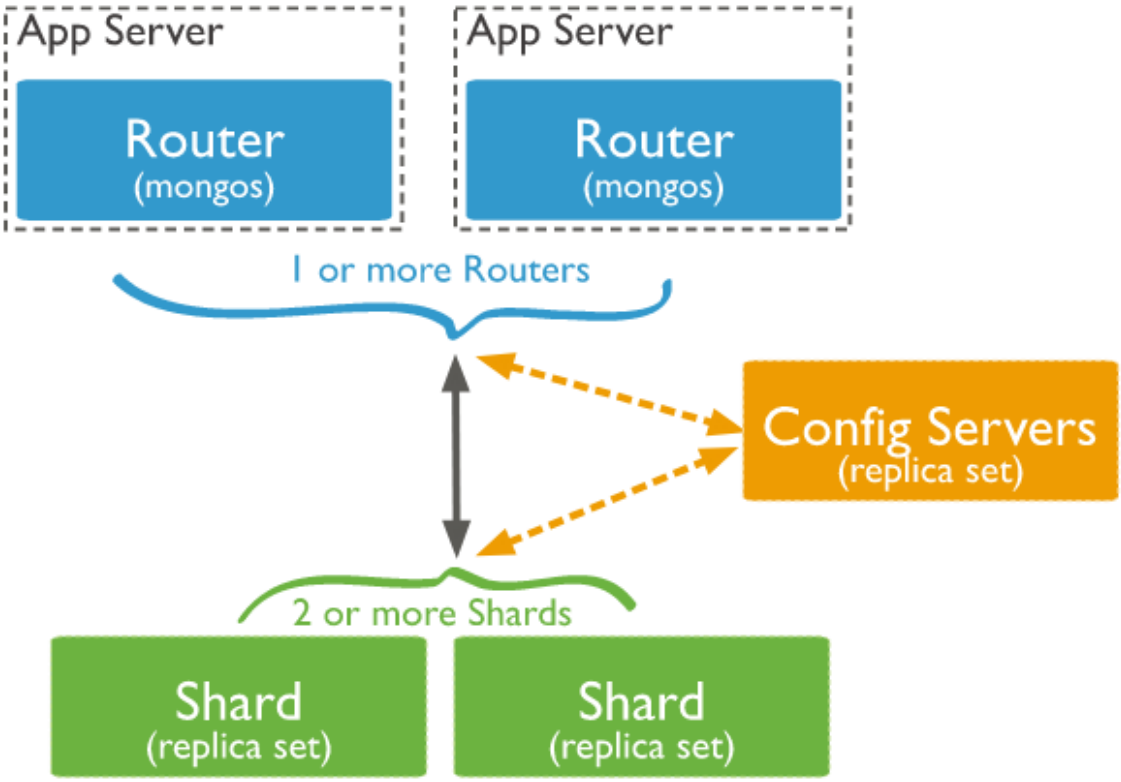
2.2. Sharded Cluster

2.2.1. Sharded Cluster o cluster fragmentado.

Un clúster sharded de MongoDB consta de los siguientes componentes:

- Shard o fragmento: Cada fragmento contiene un subconjunto de los datos fragmentados. Cada fragmento se puede implementar como un conjunto de réplicas.
- Mongos: Los mongos actúan como router de consulta, proporcionando una interfaz entre las aplicaciones cliente y el clúster.
- Servidores de configuración: los servidores de configuración almacenan los metadatos y los valores de configuración del clúster. A partir de MongoDB 3.4, los servidores de configuración deben implementarse como un conjunto de réplicas (CSRS).

El siguiente gráfico describe la interacción de componentes dentro de un clúster fragmentado:



MongoDB fragmenta los datos a nivel de colección, distribuyendo los datos de recopilación a través de los fragmentos del clúster.

2.2.2. Sharding o fragmentar.

La fragmentación es un método para distribuir datos entre múltiples máquinas. MongoDB utiliza la fragmentación para soportar despliegues con conjuntos de datos muy grandes y operaciones de alto rendimiento.

Los sistemas de base de datos con grandes conjuntos de datos o aplicaciones de alto rendimiento pueden desafiar la capacidad de un solo servidor. Por ejemplo, las altas tasas de consulta pueden agotar la capacidad de la CPU del servidor.

Existen dos métodos para abordar el crecimiento del sistema: escalamiento vertical y horizontal:

Escalado vertical implica el aumento de la capacidad de un solo servidor, como el uso de una CPU más potente, añadir más memoria RAM, o aumentar la cantidad de espacio de almacenamiento. Las limitaciones en la tecnología disponible pueden restringir que una sola máquina sea suficientemente potente para una carga de trabajo determinada. Además, los proveedores basados en la nube tienen techos rígidos basados en configuraciones de hardware disponibles. Como resultado, existe un máximo práctico para la escala vertical.

La escala horizontal implica dividir el conjunto de datos del sistema y la carga en varios servidores, agregando servidores adicionales para aumentar la capacidad según sea necesario. Aunque la velocidad o capacidad de una sola máquina puede no ser alta, cada máquina maneja un subconjunto de la carga de trabajo global, proporcionando potencialmente una mejor eficiencia que un único servidor de alta velocidad. La ampliación de la capacidad de la implementación sólo requiere la adición de servidores adicionales según sea necesario, lo que puede ser un costo total menor que el hardware de gama alta para una sola máquina.

MongoDB soporta la escala horizontal a través de la fragmentación.

2.2.3. Shard Key o clave de fragmentación.

Para distribuir los documentos en una colección, MongoDB particiona la colección usando la clave de fragmentación. La clave de fragmentación consiste en un campo o campos inmutables que existen en cada documento de la colección de destino.

La clave de fragmentación se elige cuando se fragmenta una colección. La clave de fragmentación no se puede cambiar después de la fragmentación de la colección, una colección fragmentada solo puede tener una clave de fragmento.

Para fragmentar una colección no vacía, la colección debe tener un índice que comience con la clave de fragmentación. Para las colecciones vacías, MongoDB crea el índice si la colección aún no tiene un índice apropiado para la clave de fragmentación especificada.

La elección de la clave de fragmentación afecta el rendimiento, la eficiencia y la escalabilidad de un clúster fragmentado. Un clúster con el mejor hardware e infraestructura posible puede ser bloqueado por la elección de la clave de fragmentación. La elección de la clave de fragmentación y su índice de respaldo también puede afectar a la estrategia de fragmentación que puede utilizar el clúster.

2.2.4. Chunks o trozos.

Las particiones MongoDB fragmentan los datos en trozos. Cada trozo tiene un rango inclusivo inferior y un rango exclusivo superior basado en la clave de fragmentación. MongoDB migra trozos a través de los fragmentos en el clúster utilizando el balanceador del clúster. El balanceador intenta lograr un equilibrio uniforme de trozos a través de todos los fragmentos del clúster.

2.2.5. Ventajas de la fragmentación.

- **Lectura / Escritura**

MongoDB distribuye la carga de trabajo de lectura y escritura a través de los fragmentos del clúster, permitiendo que cada fragmento procese un subconjunto de operaciones del clúster. Tanto las cargas de trabajo de lectura como de escritura se pueden escalar horizontalmente en el clúster añadiendo más fragmentos.

Para consultas que incluyen la clave de fragmentos o el prefijo de una clave de fragmentos compuestos, los mongos pueden orientar la consulta en un fragmento específico o conjunto de fragmentos. Estas operaciones específicas son generalmente más eficientes que la radiodifusión a cada fragmento del clúster.

- **Capacidad de almacenamiento.**

El fragmentado distribuye datos a través de los fragmentos del clúster, permitiendo que cada fragmento contenga un subconjunto del total de datos del clúster. A medida que el conjunto de datos crece, los fragmentos adicionales aumentan la capacidad de almacenamiento del clúster.

- **Alta disponibilidad.**

Un clúster fragmentado puede continuar realizando operaciones de lectura / escritura parciales incluso si uno o más fragmentos no están disponibles. Aunque no se puede acceder al subconjunto de datos de los fragmentos no disponibles durante el tiempo de inactividad, las lecturas o escrituras dirigidas a los fragmentos disponibles pueden tener éxito.

MongoDB puede desplegar servidores de configuración como conjuntos de réplicas. Un clúster fragmentado con un Conjunto de réplicas de servidor de configuración (CSRS) puede continuar procesando lecturas y escrituras siempre y cuando la mayoría del conjunto de réplicas esté disponible. En los entornos de producción, los fragmentos individuales deben desplegarse como conjuntos de réplicas, proporcionando mayor redundancia y disponibilidad.

2.3. Consideraciones antes de la fragmentación.

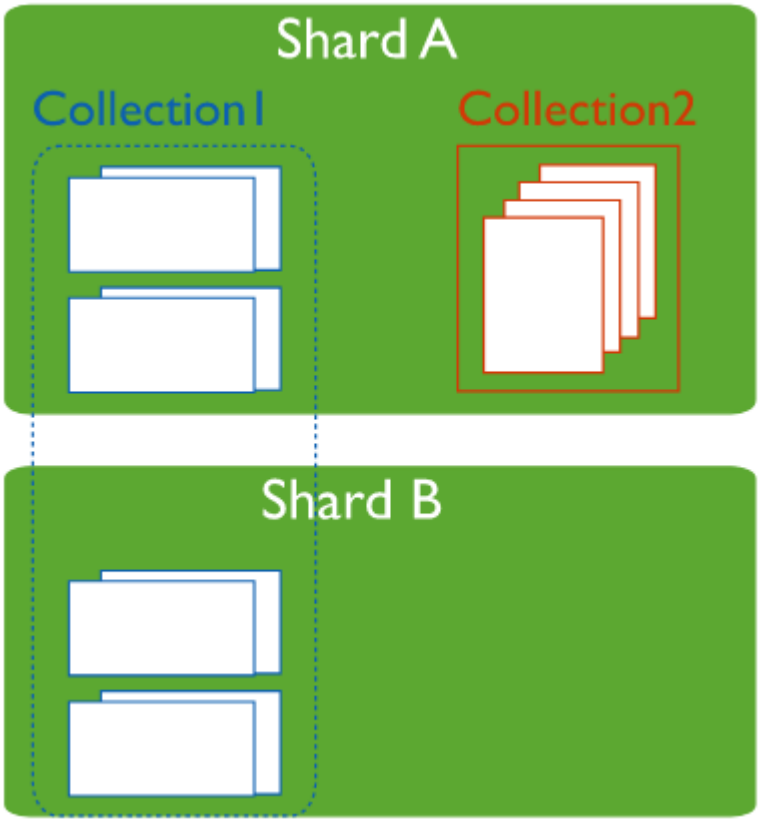
Los requisitos y la complejidad de las infraestructuras de clúster fragmentadas requieren una cuidadosa planificación, ejecución y mantenimiento.

Una consideración cuidadosa en la elección de la clave de fragmentación es necesaria para asegurar el rendimiento y la eficiencia del clúster. No se puede cambiar la clave después de la fragmentación, ni se puede revertir la fragmentación de una colección.

Si las consultas no incluyen la clave de fragmentación o el prefijo de una clave compuesta, mongos realiza una operación de difusión, consultando todos los fragmentos del clúster fragmentado. Estas consultas de dispersión / recopilación pueden ser operaciones largas.

2.4. Colecciones fragmentadas y no fragmentadas.

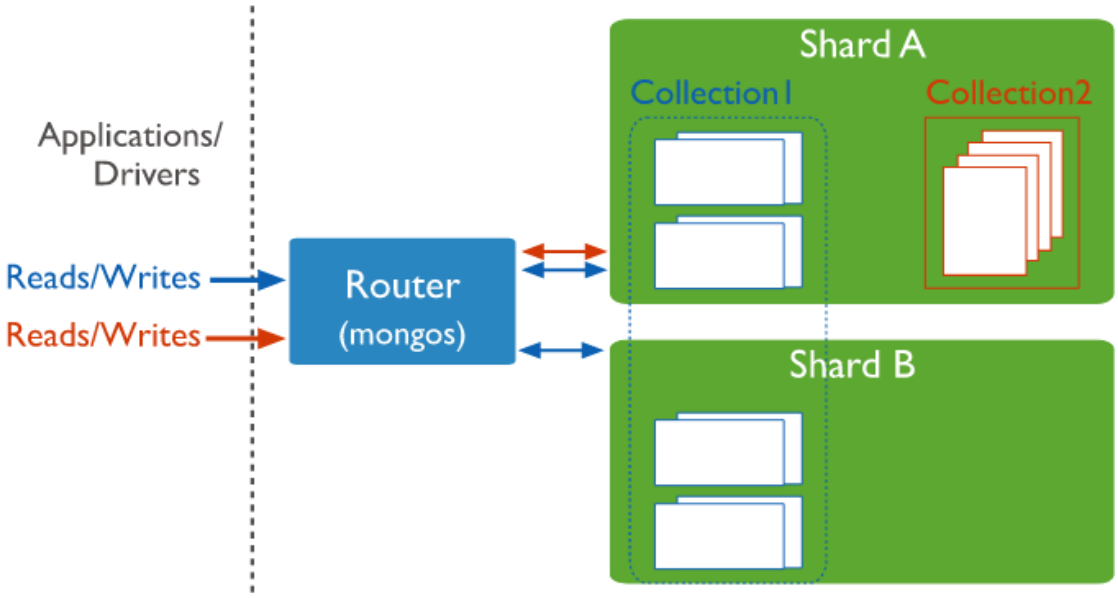
Una base de datos puede tener una mezcla de colecciones fragmentadas y no fragmentadas. Las colecciones fragmentadas se dividen y se distribuyen a través de los fragmentos del clúster. Las colecciones no fragmentadas se almacenan en un fragmento primario. Cada base de datos tiene su propio fragmento primario.



2.5. Conectarse a un clúster fragmentado.

Hay que conectarse a un router mongos para interactuar con cualquier colección del clúster fragmentado. Esto incluye, tanto colecciones fragmentadas como las no fragmentadas. Los clientes nunca deben conectarse a un solo fragmento para realizar operaciones de lectura o escritura.

Es posible conectarse a un mongos de la misma manera que se conectaría a un mongod, como a través de la shell de mongo o un controlador MongoDB.

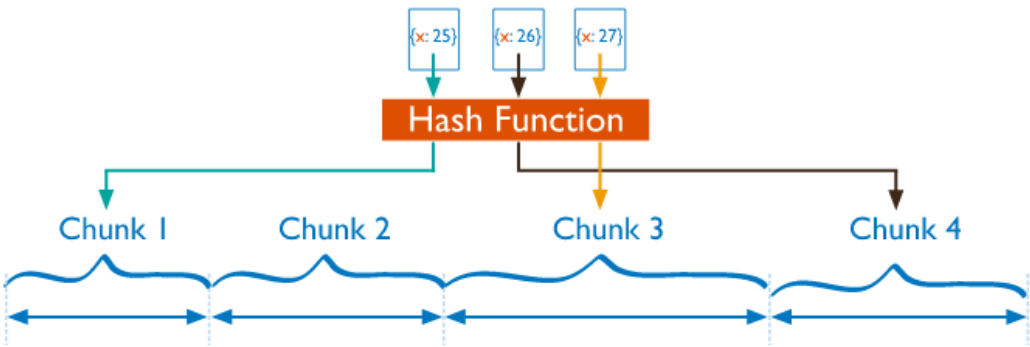


2.6. Estrategia de fragmentado.

MongoDB soporta dos estrategias de fragmentado para distribuir datos entre los clústeres.

- **Hashed Sharding**

Hashed Sharding implica calcular el hash del valor del campo de la clave de fragmentación. A cada trozo se le asigna un rango basado en los valores del hash de la clave. MongoDB calcula automáticamente los hashes cuando se resuelven consultas utilizando índices de hash. Las aplicaciones no necesitan calcular hashes.

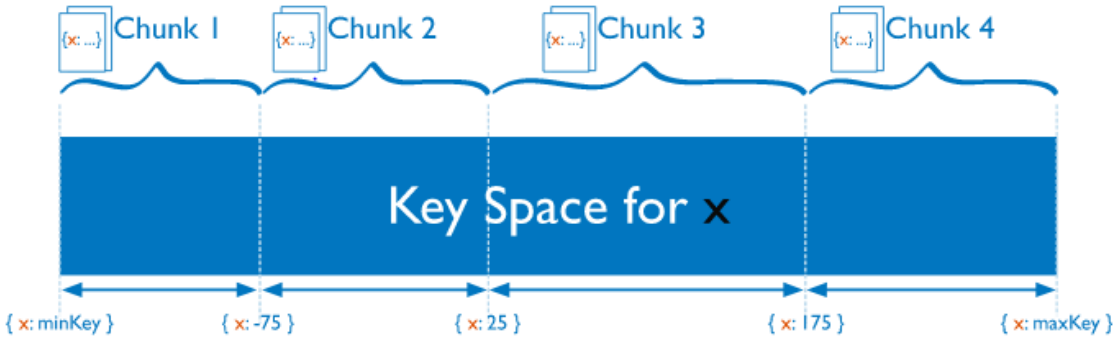


Mientras que un rango de claves de fragmentos puede estar limitado, es improbable que sus valores de hash estén en el mismo trozo. La distribución de datos basada en valores de hash facilita una distribución más uniforme de los datos, especialmente en conjuntos de datos en los que la clave del fragmento cambia de forma monótona.

Sin embargo, la distribución hash significa que las consultas basadas en rangos de la clave son menos propensas a apuntar a un único fragmento, lo que resulta en más operaciones de difusión en todo el clúster.

- **Ranged Sharding.**

El ranged sharding implica dividir datos en intervalos basados en los valores de la clave de fragmentos. A cada trozo se le asigna un rango basado en los valores de la clave.



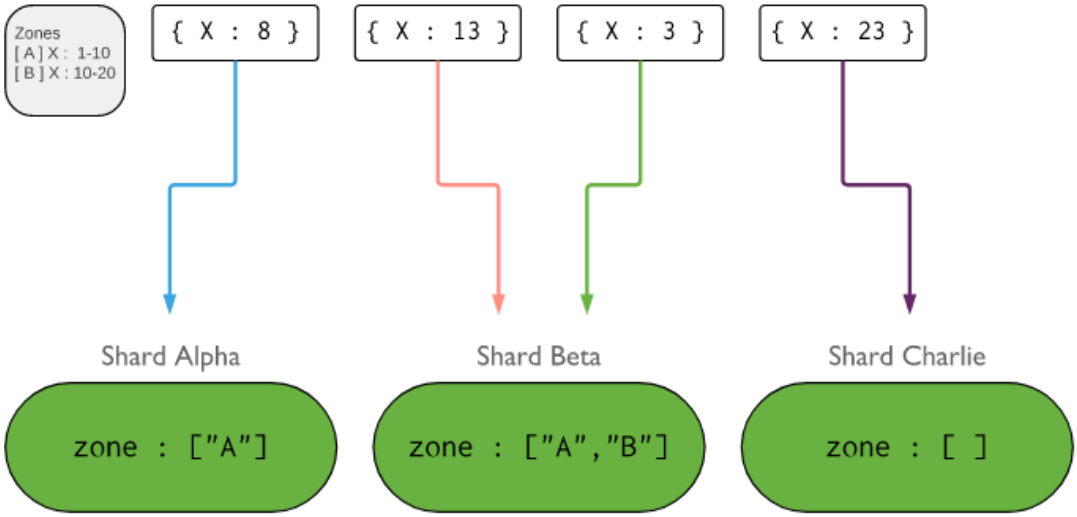
Un rango de claves de fragmentos cuyos valores son "cercanos" tienen más probabilidades de residir en el mismo trozo. Esto permite operaciones específicas, ya que los mongos pueden encaminar las operaciones sólo a los fragmentos que contienen los datos requeridos.

La eficiencia del ranged sharding depende de la clave de fragmento elegida. Las claves mal consideradas pueden resultar en una distribución desigual de los datos, lo que puede anular algunos beneficios de la fragmentación o puede provocar cuellos de botella en el rendimiento.

2.7. Zonas en clústeres fragmentados.

En los clústeres fragmentados, se pueden crear zonas de datos fragmentados basados en la clave de fragmentos. Se asocian cada zona con uno o más fragmentos del clúster. Un fragmento puede asociarse con cualquier número de zonas no conflictivas. En un cluster equilibrado, MongoDB migra los trozos cubiertos por una zona sólo a los fragmentos asociados con la zona.

Cada zona cubre una o más gamas de valores de la clave. Cada rango que cubre una zona siempre incluye su límite exclusivo inferior y su límite superior.



Se deben utilizar los campos de la clave de fragmentos al definir un nuevo rango para que una zona se cubra. Si se utiliza una clave de fragmentos compuestos, el rango debe incluir el prefijo de la clave de fragmentos.

Al elegir una clave de fragmentos, hay que considerar cuidadosamente la posibilidad de usar las zonas en el futuro, ya que no se puede cambiar la clave de fragmentos después de fragmentar una colección.

Más comúnmente, las zonas sirven para mejorar la localización de los datos en los clústeres fragmentados que abarcan múltiples centros de datos.

3. Creación y despliegue de un cluster fragmentado.

Un cluster fragmentado consta de 3 elementos:

- Config servers: Se encargan de guardar los metadatos de todos los elementos del cluster.
- Mongos: Esta instancia actúa como router entre el cliente y el cluster, redirigiendo la query del cliente al shard adecuado según los metadatos del config server.
- Shards: Contienen las bases de datos y colecciones.

A continuación se desarrollaran y explicaran las características de estos elementos ademas de su despliegue y configuración.

3.1 Config servers o servidores de configuración.

Los servidores de configuración almacenan los metadatos del clúster, estos reflejan el estado y la organización de todos los datos y componentes dentro del clúster. Los metadatos incluyen la lista de trozos en cada fragmento y los rangos que definen los trozos.

Los routers mongos almacenan en caché estos datos y los utilizan para dirigir las operaciones de lectura y escritura a los fragmentos correctos. Mongos actualiza el caché cuando hay cambios de metadatos para el clúster, como la división de trozos o agregar un nuevo fragmento al cluster, además del router mongos, los fragmentos también leen los metadatos de los servidores de configuración.

Los servidores de configuración también almacenan información para la autenticación, como el control de acceso basado en roles o la configuración de autenticación interna del clúster. MongoDB también utiliza los servidores de configuración para administrar los bloqueos distribuidos.

Cada clúster debe tener sus propios servidores de configuración, no es posible utilizar los mismos servidores de configuración para diferentes clústeres.

3.1.1 Replica set en los servidores de configuración

Los servidores de configuración de los clústeres se pueden implementar como un conjunto de réplicas (replica set) en lugar de un solo servidor duplicado varias veces. El uso de un conjunto de réplicas para los servidores de configuración mejora la coherencia entre los servidores, ya que MongoDB puede aprovechar los protocolos de lectura y escritura establecidos por la réplica estándar para los datos de configuración.

Además, el uso de un conjunto de réplicas para los servidores permite que un clúster tenga más de 3 servidores de configuración, ya que un conjunto de réplicas puede tener hasta 50 miembros.

Este tipo de replicación proporciona redundancia y aumenta la disponibilidad de datos. Con múltiples copias de datos en diferentes servidores de bases de datos, la replicación proporciona un nivel de tolerancia a fallos frente a la pérdida de un único servidor de base de datos.

En algunos casos, la replicación puede proporcionar una mayor capacidad de lectura, ya que los clientes pueden enviar operaciones de lectura a diferentes servidores. El mantenimiento de copias de datos en diferentes centros de datos puede aumentar la localidad de los datos y la disponibilidad de las aplicaciones distribuidas. También puede mantener copias adicionales para propósitos específicos, como recuperación de desastres, informes o copia de seguridad.

Por otro lado, el uso de un conjunto de replicación para los servidores de configuración tiene ciertas limitaciones o restricciones:

- No puede tener arbiters, es necesario que exista quorum entre los servidores únicamente.

- Los servidores deben generar índices ,es decir, ningún miembro debe tener la configuración buildIndexes establecida en false).

3.1.2 Operaciones de lectura y escritura.

Las bases de datos de administración y de configuración están en los servidores de configuración.

Escritura.

La base de datos admin contiene las colecciones relacionadas con la autenticación y la autorización, así como con las demás colecciones del sistema para uso interno.

La base de datos de configuración contiene las colecciones que tienen los metadatos del clúster. MongoDB escribe datos en esta base de datos de cuando cambian los metadatos, como por ejemplo después de una migración de trozos o una división de fragmentos.

Los usuarios deben evitar escribir directamente a la base de datos de configuración durante el funcionamiento normal o el mantenimiento.

Lectura

MongoDB lee desde la base de datos de administración los datos de autenticación y autorización y otros usos internos.
MongoDB lee de la base de datos de configuración cuando se inicia un mongos o después de un cambio en los metadatos. Los fragmentos también leen los metadatos de los servidores de configuración.

3.1.3 Disponibilidad.

Si el conjunto de réplicas de los servidores de configuración pierde su servidor primario y no puede elegir uno nuevo, los metadatos del clúster se vuelven de sólo lectura. Todavía es posible leer y escribir datos de los fragmentos, pero no se producirá ninguna migración o otra actividad hasta que el conjunto de réplicas pueda elegir un servidor primario nuevamente. Si todas las bases de datos de configuración no están disponibles, el clúster puede volverse inoperable.

Las instancias mongos almacenan en caché los metadatos de los servidores de configuración. Por lo tanto, si todos los miembros del servidor de configuración no están disponibles, se puede seguir utilizando el clúster si no se reinician las instancias mongos hasta que los servidores de configuración sean accesibles de nuevo. Si se reinician las instancias mongos antes de que los servidores estén disponibles, los mongos no pueden enrutar lecturas o escrituras.

Los clústeres se vuelven inoperables sin los metadatos del clúster. Para garantizar que los servidores de configuración permanezcan disponibles e intactos, las copias de seguridad de los servidores son críticas. Los datos del servidor de configuración son pequeños en comparación con los datos almacenados en un clúster y el servidor de configuración tiene una carga de actividad relativamente baja.

3.1.4 Configuración de los servidores

Es posible configurar el servidor utilizando un fichero con todas las opciones que queramos o bien poner las opciones directamente en el comando.

Si queremos usar un fichero es simple, utilizaremos la opción “--config” del comando “mongo” de la siguiente forma:

→ mongo --config <path o nombre del fichero>

En este caso se utilizara la segunda opción, el resultado es el mismo.

Antes de ejecutar el comando es necesario que el servicio de mongo este detenido, ya que vamos a lanzar un nuevo servicio mongo con unas características diferentes y si ya existe un proceso no nos dejará.

Además hay que crear el directorio /data/configdb, puesto que es el directorio que utiliza mongo por defecto para los servidores de configuración.

En esta ocasión voy a utilizar 3 servidores de configuración, los iniciamos con el siguiente comando:

→ mongod --configsvr --replSet rs --port 27019

Con la opción “--configsrv” especificamos que se trata de un servidor de configuración, con la opción “--replSet <nombre de la replica>” elegimos el nombre de la replica (esta

debe ser la misma en todos los servidores que se usen) y con la opción “--port” especificamos el puerto de escucha para el servicio.

Al ejecutar este comando el servicio se quedará en espera de otras conexiones, hay que hacerlo en todos los servidores. Una vez iniciados, nos conectaremos a uno de ellos para iniciar el replica set.

→ mongo --host localhost --port 27019

Cuando estemos conectados iniciamos el replica set con el comando “rs”

→ rs.initiate()

A continuación añadimos los demás miembros al replica set

→ rs.add('192.168.100.7:27019')
→ rs.add('192.168.100.8:27019')

Después de un tiempo para que los servidores se sincronicen, podemos ver el estado del set con el comando “rs.status()”

→ rs.status()

```
rs:PRIMARY> rs.status()
{
        "set" : "rs",
        "date" : ISODate("2017-06-14T18:55:01.897Z"),
        "myState" : 1,
        "term" : NumberLong(2),
        "configsvr" : true,
        "heartbeatIntervalMillis" : NumberLong(2000),
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(1497466496, 1),
                        "t" : NumberLong(2)
                },
                "readConcernMajorityOpTime" : {
                        "ts" : Timestamp(1497466496, 1),
                        "t" : NumberLong(2)
                },
                "appliedOpTime" : {
                        "ts" : Timestamp(1497466496, 1),
                        "t" : NumberLong(2)
                },
                "durableOpTime" : {
                        "ts" : Timestamp(1497466496, 1),
                        "t" : NumberLong(2)
                }
        },
```

Según bajamos por el resultado del comando veremos el estado de los servidores.

Aquí vemos que el servidor primario es confserver, el primario suele ser el servidor en el que hemos iniciado el replica set.

```
"members" : [
        {
                "_id" : 0,
                "name" : "confserver:27019",
                "health" : 1,
                "state" : 1,
                "stateStr" : "PRIMARY",
                "uptime" : 623,
                "optime" : {
                        "ts" : Timestamp(1497466496, 1),
                        "t" : NumberLong(2)
                },
                "optimeDate" : ISODate("2017-06-14T18:54:56Z"),
                "electionTime" : Timestamp(1497465924, 1),
                "electionDate" : ISODate("2017-06-14T18:45:24Z"),
                "configVersion" : 3,
                "self" : true
        },
```

Aquí podemos ver los servidores secundarios.

```
        "_id" : 1,
        "name" : "192.168.100.7:27019",
        "health" : 1,
        "state" : 2,
        "stateStr" : "SECONDARY",
        "uptime" : 1291,
        "optime" : {
                "ts" : Timestamp(1497467204, 1),
                "t" : NumberLong(2)
        },
        "optimeDurable" : {
                "ts" : Timestamp(1497467204, 1),
                "t" : NumberLong(2)
        },
        "optimeDate" : ISODate("2017-06-14T19:06:44Z"),
        "optimeDurableDate" : ISODate("2017-06-14T19:06:44Z"),
        "lastHeartbeat" : ISODate("2017-06-14T19:06:45.678Z"),
        "lastHeartbeatRecv" : ISODate("2017-06-14T19:06:46.227Z"),
        "pingMs" : NumberLong(0),
        "syncingTo" : "confserver:27019",
        "configVersion" : 3
},
```

Una vez comprobado esto, ya están listos los servidores de configuración.

3.2 Servidores fragmentados o shard servers.

Un fragmento contiene un subconjunto de datos para un clúster. Juntos, los fragmentos del clúster contienen todo el conjunto de datos del clúster.

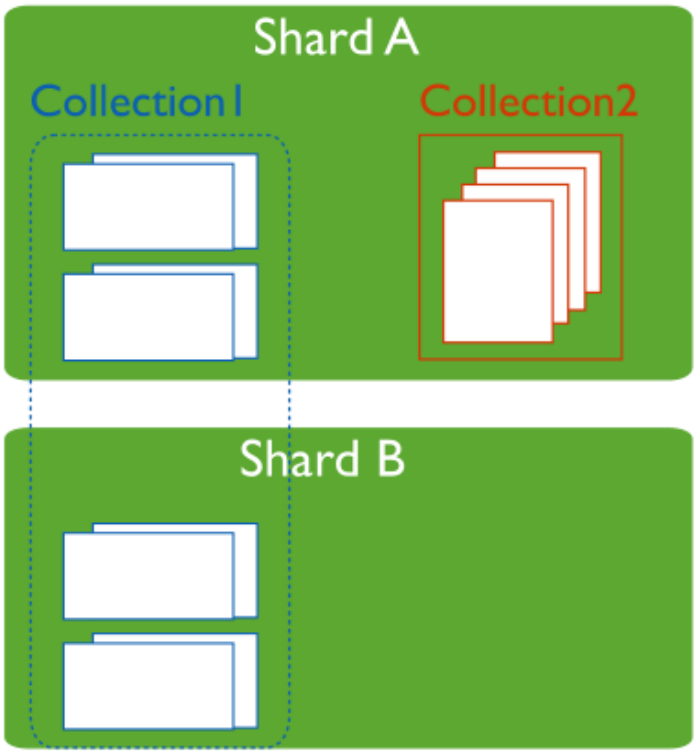
Los fragmentos deben desplegarse como un conjunto de réplicas para proporcionar redundancia y alta disponibilidad.

Los usuarios, clientes o aplicaciones sólo deben conectarse directamente a un fragmento para realizar operaciones administrativas y de mantenimiento locales y el resto de consultas o acciones deben realizarse mediante una instancia mongos.

Si por ejemplo realizamos las consultas en un solo fragmento este sólo devuelve un subconjunto de datos, es necesario usar un mongos para que este realice las operaciones a nivel de clúster y así unificar los datos que están repartidos.

Cada base de datos de un clúster fragmentado tiene un fragmento primario que contiene todas las colecciones no fragmentadas de esa base de datos, el resto son repartidas entre los fragmentos. Cada base de datos tiene su propio fragmento primario, es decir un fragmento por defecto en el que almacenar las colecciones. Este fragmento primario no tiene relación con el primario en un conjunto de réplicas.

Los mongos seleccionan el fragmento primario al crear una nueva base de datos escogiendo el fragmento en el clúster que tiene la menor cantidad de datos, Mongos utiliza el campo “totalSize” devuelto por el comando “listDatabase” como parte de los criterios de selección.



Es posible cambiar el fragmento primario de una base de datos, hay que usar el comando “movePrimary”. El proceso de migración del fragmento primario puede tardar un tiempo considerable en completarse y se perderá el acceso a las colecciones asociadas a la base de datos hasta que se complete. Dependiendo de la cantidad de datos que se están migrando, la migración puede afectar las operaciones generales del clúster. Hay que considerar el impacto en las operaciones del clúster y la carga de la red antes de intentar cambiar el fragmento primario.

Algo importante a tener en cuenta es que al implementar un nuevo clúster con fragmentos que se utilizaron anteriormente como conjuntos de réplicas, todas las bases de datos existentes continúan residiendo en sus conjuntos de réplicas originales. Las bases de datos creadas posteriormente pueden residir en cualquier fragmento del clúster.

3.2.1 Configuración y despliegue de los servidores fragmentados.

El proceso a seguir es el mismo que se ha realizado a los servidores de configuración pero con algunas variaciones:

- Hay que cambiar es la opción “--configsrv” por “--shardsrv” para indicar que es un shard
- Hay crear el directorio /data/db, que es el directorio por defecto para los shards del cluster.
- Estos shards deben tener su propio replica set.

Como ya he mencionado, el proceso es el mismo.

→ mongod --shardsvr --replSet shard --port 27018 (en todos los shards)

→ mongo --host localhost --port 27018 (en uno de los shards)

→ rs.initiate()

→ rs.add('192.168.100.3:27018')
→ rs.add('192.168.100.4:27018')

Después de un tiempo para que los servidores se sincronicen, podemos ver el estado del replica set con “rs.status()”

```
shard:PRIMARY> rs.status()
{
        "set" : "shard",
        "date" : ISODate("2017-06-14T20:18:21.130Z"),
        "myState" : 1,
        "term" : NumberLong(4),
        "heartbeatIntervalMillis" : NumberLong(2000),
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(1497471497, 1),
                        "t" : NumberLong(4)
                },
                "appliedOpTime" : {
                        "ts" : Timestamp(1497471497, 1),
                        "t" : NumberLong(4)
                },
                "durableOpTime" : {
                        "ts" : Timestamp(1497471497, 1),
                        "t" : NumberLong(4)
                }
        },
"members" : [
        {
                "_id" : 0,
                "name" : "shard1:27018",
                "health" : 1,
                "state" : 1,
                "stateStr" : "PRIMARY",
                "uptime" : 4020,
                "optime" : {
                        "ts" : Timestamp(1497471497, 1),
                        "t" : NumberLong(4)
                },
                "optimeDate" : ISODate("2017-06-14T20:18:17Z"),
                "electionTime" : Timestamp(1497467524, 1),
                "electionDate" : ISODate("2017-06-14T19:12:04Z"),
                "configVersion" : 3,
                "self" : true
        },

        "_id" : 1,
        "name" : "192.168.100.3:27018",
        "health" : 1,
        "state" : 2,
        "stateStr" : "SECONDARY",
        "uptime" : 3991,
        "optime" : {
                "ts" : Timestamp(1497471497, 1),
                "t" : NumberLong(4)
        },
        "optimeDurable" : {
                "ts" : Timestamp(1497471497, 1),
                "t" : NumberLong(4)
        },
        "optimeDate" : ISODate("2017-06-14T20:18:17Z"),
        "optimeDurableDate" : ISODate("2017-06-14T20:18:17Z"),
        "lastHeartbeat" : ISODate("2017-06-14T20:18:19.449Z"),
        "lastHeartbeatRecv" : ISODate("2017-06-14T20:18:19.950Z"),
        "pingMs" : NumberLong(0),
        "syncingTo" : "shard1:27018",
        "configVersion" : 3
```

3.3 Configuración y despliegue de un router mongos

Mongos es un servicio de enrutamiento para las configuraciones de fragmentos que procesa las consultas de la capa de aplicación y determina la ubicación de los datos en el clúster para completar estas operaciones.
Desde la perspectiva de la aplicación, una instancia mongos se comporta de forma idéntica a cualquier otra instancia de MongoDB.

La configuración del router mongos es mucho mas simple que el resto de componentes, solo es necesario usar la opción “--configdb”

→ mongos --configdb
rs/192.168.100.6:27019,192.168.100.7:27019,192.168.100.8:27019

Hay que poner el nombre del conjunto de replicas de los servidores de configuración, seguido de la ip/nombre de las maquinas con su correspondiente puerto separados por una coma.

Los shards ya están creados, pero el router mongos no sabe que existen o que están operativos. Es necesario añadir los shards con el comando “sh.addShard()”

→ sh.addShard("shard/192.168.100.2:27018")
→ sh.addShard("shard/192.168.100.3:27018")
→ sh.addShard("shard/192.168.100.4:27018")

Es importante que si los shards pertenecen a algún conjunto de replicas, hay que poner el nombre de este en el comando.

Del mismo modo que podemos ver el estado de los conjuntos de replicas, también podemos ver es estado de los shards y de los datos que contienen.

En esta imagen podemos ver el estado de los shards (mongos permite la opción de incluir un balanceador interno, está habilitado pero no activado).

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("59401fa9dce6edd6b06a6fcc")
}
  shards:
        {  "_id" : "shard",  "host" : "shard/192.168.100.3:27018,192.168.
100.4:27018,shard1:27018",  "state" : 1 }
  active mongoses:
        "3.4.4" : 1
 autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
```

Después de configurar el router mongos, el cluster ya está completo. En este caso tenemos los siguientes componentes:

- Tres servidores de configuración.
  - 192.168.100.6:27019
  - 192.168.100.7:27019
  - 192.168.100.8:27019

- Tres shards.
  - 192.168.100.2:27018
  - 192.168.100.3:27018
  - 192.168.100.4:27018

- Un router mongos.
  - 192.168.100.5 en el puerto por defecto de mongo.

4. Fragmentar una colección.

Al fragmentar una colección se distribuyen los documentos que contenga entre fragmentos.

Para poder fragmentar una colección primero se sebe habilitar la fragmentación en una base de datos, para ello usamos el comando “sh.enebleSharding()”. Vemos un ejemplo:

→ sh.enableSharding("Proyecto")

Una vez fragmentada la base de datos, podemos ver el estado de todas las bases de datos con el comando “sh.status()”.

```
  databases:
        {  "_id" : "Proyecto",  "primary" : "shard",  "partitioned" : true }
        {  "_id" : "Prueba",  "primary" : "shard",  "partitioned" : false }

mongos>
```

Para fragmentar una colección se usa el comando shardCollection.

→ sh.shardCollection("Proyecto.Prueba", { "Index" : 1 })

El formato es el siguiente:

→ sh.shardCollection("<BD>.<Colección>", { "<campo>" : <shard_key> })

Con el comando “sh.status()” podemos ver que se ha fragmentado.

```
databases:
      {  "_id" : "Proyecto",  "primary" : "shard",  "partitioned" : true }
            Proyecto.Prueba
                    shard key: { "Index" : 1 }
                    unique: false
                    balancing: true
                    chunks:
                            shard   1
                    { "Index" : { "$minKey" : 1 } } -->> { "Index" : { "$maxKey"
: 1 } } on : shard Timestamp(1, 0)
```

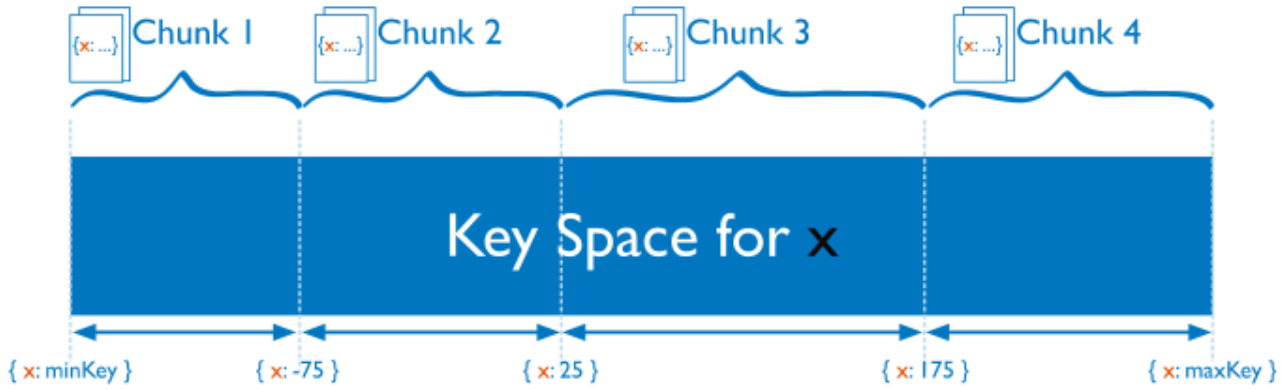
La shard key es el parámetro mas importante a la hora de fragmentar una colección, ya que determina el como se distribuye la colección entre los fragmentos del cluster.

Algo importante a tener en cuenta es que una vez elegida la clave, no es posible cambiarla en ningún momento. La única forma de que una colección pierda su clave es eliminando la colección.

La clave crea un índice para el campo seleccionado de la colección si la colección esta vacía, en caso contrario es necesario crear primero un índice manualmente. En este caso he usado una clave con valores cardinales, este valor indica el numero máximo de trozos o chunks que puede tener la colección.

Por ejemplo, si usamos una clave con valor 4, la colección tendrá 4 trozos como máximo independientemente si el cluster tiene 5 o mas shards. Por eso es importante elegir bien el valor de la clave, ya que puede afectar al rendimiento del cluster completo.

Es posible elegir una clave cuyo valor exceda con creces el numero de shards disponibles, mongo repartirá los valores de la clave a través de los shard que estén disponibles y les asignará un rango de valores para que el router mongos pueda realizar consultas al trozo correspondiente según el valor.

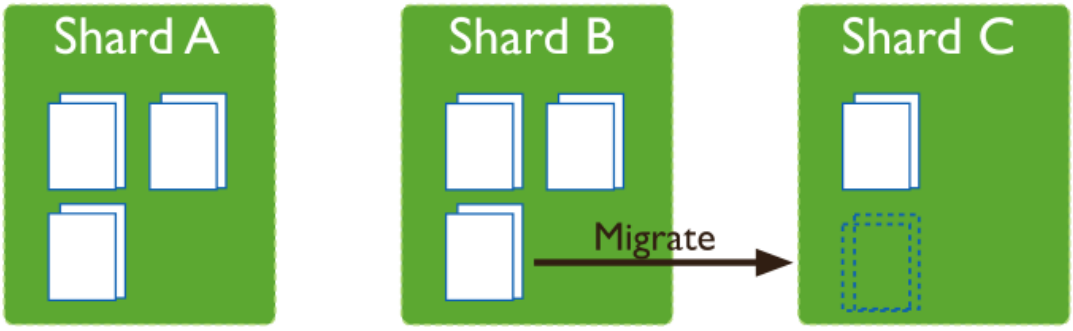


Es importante tener en cuenta que al añadir un nuevo shard, no se reestructuran los valores, como ya se ha mencionado, la clave no puede alterarse.

5. Balanceador del cluster

El balanceador de mongo es un proceso de fondo que supervisa el número de trozos en cada fragmento. Cuando el número de trozos de un fragmento determinado alcanza los limites de migración específicos, el balanceador intenta migrar automáticamente los trozos entre fragmentos y alcanzar un número igual de trozos por fragmento.

El procedimiento de balanceo para los clústeres es totalmente transparente para el usuario y la capa de aplicación, aunque puede haber algún impacto en el rendimiento mientras se lleva a cabo el procedimiento.



Iniciar el balanceador es muy sencillo, primero vemos si esta habilitado (Normalmente esta habilitado por defecto) con el comando “sh.getBalancerState()”

```
mongos> sh.startBalancer()
{ "ok" : 1 }
```

En caso de que no este habilitado, lo activamos con el siguiente comando:

```
db.settings.update(
  { _id: "balancer" },
  { $set: { activeWindow : { start : "<start-time>", stop : "<stop-time>" } } },
  { upsert: true }
)
```

Por ultimo lo iniciamos con “sh.startBalancer()”

```
mongos> sh.startBalancer()
{ "ok" : 1 }
```

Este balanceador solo se ejecuta cuando es necesario migrar algún trozo, podemos ver el estado con “sh.isBalancerRunning()”, ahora esta en false porque no hay ninguna migración en ejecución.

```
mongos> sh.isBalancerRunning()
false
```

6. Bibliografía

Toda la documentación presentada en este documento tiene como fuente la documentación oficial de mongo, que es la mas completa puesto que es oficial y casi la única existente en el aspecto técnico.

https://docs.mongodb.com/manual/

Concretamente la versión 3.4 de MongoDB es la utilizada para el despliegue de este cluster.

Las imágenes son de la misma fuente, por su sencillez, además de muy explícitas y concretas.