ntic master
revolucionamos la comunicación

UNIVERSIDAD COMPLUTENSE DE MADRID

Arquitectura MongoDB

Alvaro Bravo

Índice

1. Arquitectura

- MongoDB (de la palabra en inglés ''Humongous'' que significa enorme) es una base de datos NoSQL de tipo documental, es decir, orientada a documentos, desarrollado bajo el concepto de código abierto.

- Implementado en C++ por la empresa de software MongoDB Inc. (antes 10gen).

- El Shell de MongoDB es un intérprete JavaScript interactivo, que permite utilizar código JavaScript directamente o ejecutar archivos JavaScript en standalone.

- Su posicionamiento en el Teorema CAP es **CP** (garantizan consistencia y tolerancia a particiones sacrificando la disponibilidad), pero no es definitivo. MongoDB puede configurarse para cambiar su comportamiento.
    - Ej. Configurar el nivel de consistencia eligiendo el nº de nodos a los que se replicarán los datos o si se permite leer de un nodo secundario, sacrificaremos consistencia pero ganaremos disponibilidad.

- Permite operaciones de **alta, baja, modificación y consulta** sobre los documentos almacenados (acceso por documento o por atributo).

- Todo documento, además de los datos, tiene que tener un campo ***_id*** único que puede ser creado o dejarle a MongoDB que lo haga. Se trata de un campo **ObjectId** de 12 bytes (4 de timestamp, 3 de id. Máquina, 2 de id. Proceso y 3 de un contador).

- **Consultas ad-hoc**: búsquedas por campos, consultas de rangos, expresiones regulares, funciones JavaScript, etc.

- **Indexación**: definición de índices a nivel de colección (máx. 64 por colección). Soporta índices por cualquier atributo de los documentos, incluso de los documentos embebidos. Tipos de índice: **primarios**, **simples**, **multikeys**, **texto** (búsquedas por texto libre), **geoespaciales** y **time to live**. Ante falta de índice, lanza en paralelo n planes de ejecución candidatos y gana el que devuelva 100 resultados con mayor rapidez (el ganador se guarda para futuras queries similares).

- **Replicación**: nativa. Soporta la replicación maestro-esclavo. Al grupo de instancias que poseerán la misma información se les denomina **Replica Set** y está compuesto por 2 tipos principales de miembros, instancias primarias y secundarias.

- **Balanceo de carga:** la escalabilidad se consigue mediante **Sharding**. El **Sharding** es un conjunto de **Replica Sets** cuya función es repartir uniformemente la carga de trabajo, de tal manera que nos permite escalar horizontalmente las aplicaciones para así poder trabajar con grandes cantidades de datos.

- **Almacenamiento de archivos**: cuenta con una arquitectura de almacenamiento muy flexible que incorpora los motores **WiredTiger y In-Memory** para lograr un mejor rendimiento y una mayor compresión. Los documentos son almacenados en formato BSON (JSON codificado de manera binaria) con un tamaño máximo de 16MB. Para ficheros con un tamaño superior (ej. videos) se usa la utilidad **GridFS**.

- **Agregación**: para operaciones más complejas dispone de 2 funcionalidades: **Aggregation** Framework (ejecución de operaciones secuenciales sobre una colección para realizar cálculos, filtrados, etc.) y **Map Reduce** (implementación en JavaScript).

Otras características:

- Permite almacenar cualquier tipo de datos (datos estructurados, semi-estructurados o no estructurados) siempre enfocados para un ámbito documental.
    - Ej. Aplicaciones CRUD o desarrollos web.

- Modelo de datos **flexible**: las colecciones no necesitan definir un modelo de datos estático, aunque se recomienda diseñar la aplicación para seguir uno, ya que las consultas más frecuentes marcarán el modelo de datos a definir.

- **Document Validation**: Capacidad para validar documentos mediante reglas de validación o expresiones durante operaciones de update o inserts. Incluso con las validaciones activas sobre una colección, advertir que esta puede contener documentos inválidos insertados antes de su activación

- Dispone de conectores/drivers con un gran número de tecnologías y soluciones empresariales.

- No hay integridad referencial.

- **Lookup Dinámicos**: incluye el operador **$lookup**, que permite realizar left outer joins sobre varios documentos.

- **Text Search**: soporta índices sobre los elementos de texto. El operador *$regexp* permite realizar búsquedas por patrones mediante el uso de expresiones regulares.

- No es una base de datos transaccional. Por defecto, las operaciones en MongoDB son atómicas sobre un único documento. La utilización del patrón ***two-phase-commit*** permitirá tratar varios documentos de forma transaccional, implementando la transaccionalidad desde la aplicación. A partir de la versiones 4.0 permiten transacciones múltiples y distribuidas.

- No se recomienda para realizar Cubos OLAP o cálculos complejos.

Nomenclatura: MongoDB se organiza en bases de datos, colecciones, documentos y campos.

Equivalencias:

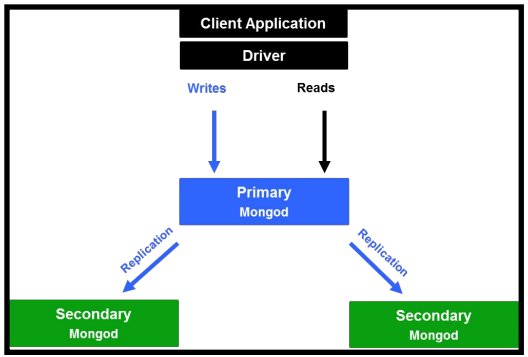
RDBMS	MongoDB
TableSpace/Schema/Database	Database
Table	Collection
Row	Document (BSON)
Transaction	Atomic Operation
Partitioning	Sharding
HADR / GoldenGate / DataGuard	Replication
SELECT/INSERT/UPDATE	FIND/INSERT/UPDATE

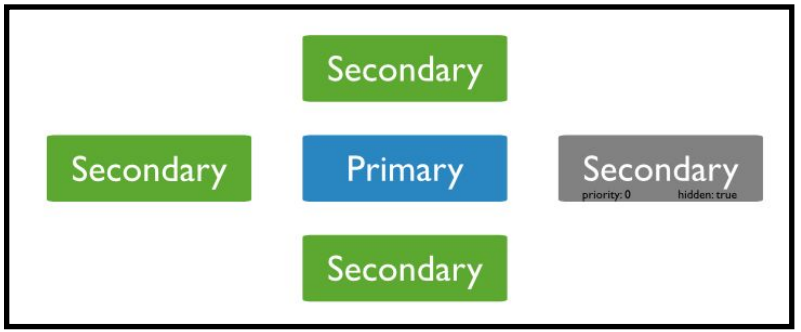
Relaciones:

- Una base de datos puede tener una o más colecciones.
- Una colección puede estar compuesta por 0 o más documentos.
- Un documento está compuesto por 1 o varios campos.

2. Replicaset

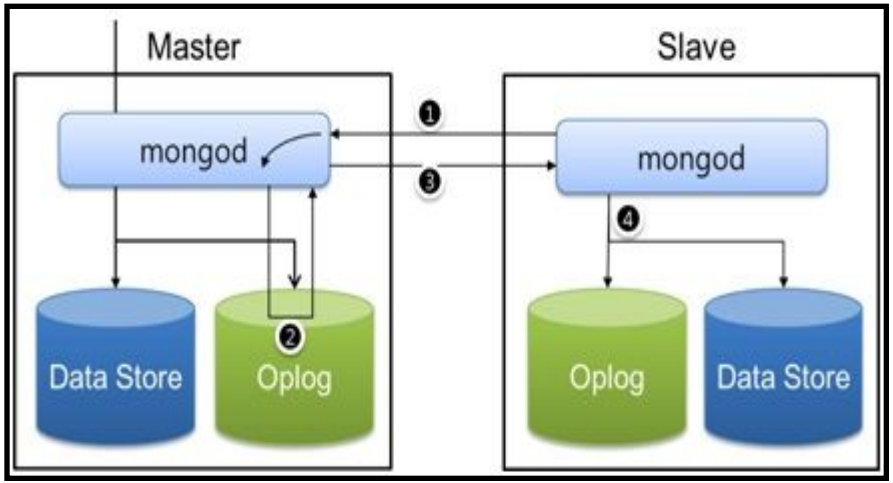
- En MongoDB la replicación es la forma de proporcionar alta disponibilidad y tolerancia a fallos de forma nativa y transparente a las aplicaciones que lo utilizan.
- La replicación parte de una colección de instancias o nodos de MongoDB, denominado conjunto de réplicas, en donde siempre debe de existir un nodo primario para poder estar activo.
- Configuración mínima para replicación: 3 nodos por **Replica Set** (Nodo primario + secundarios).
- MongoDB para proporcionar HA no requiere de componentes adicionales.
- Los datos se escriben en el nodo primario, y de forma asíncrona se copian a los nodos secundarios.
- Si el nodo primario se cae, el resto de nodos deciden cuál de ellos pasará a ser el nuevo nodo primario, que asumirá el control en un tiempo entre 1 y 3 segundos.
- Normalmente durante este tiempo, las aplicaciones recibirán un mensaje diciéndoles que lo vuelvan a intentar, esto sería transparente para los usuarios finales.

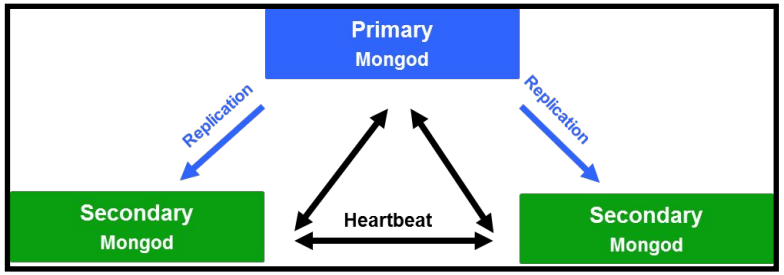




Proceso de replicación:

- Un nodo primario recibe todas las peticiones y registra los cambios en un **Oplog**.
- Los nodos secundarios leen el **Oplog** (colección) y replican los cambios.
- Todos los miembros del conjunto poseen una copia de dicho **Oplog** en la colección: ''local.oplog.rs'' para poder mantener actualizada su BD. Esto se hace a través de heartbeats o pings a todos los miembros permitiendo importar registros de cualquiera del resto de los nodos del conjunto.
- MongoDB implementa dos tipos de sincronización para mantener actualizados todos los nodos de un conjunto de réplicas, el primero es la sincronización inicial para cargar miembros nuevos con todos los datos del conjunto, y el segundo, es la replicación que mantiene los datos actualizados del conjunto después de la sincronización inicial.

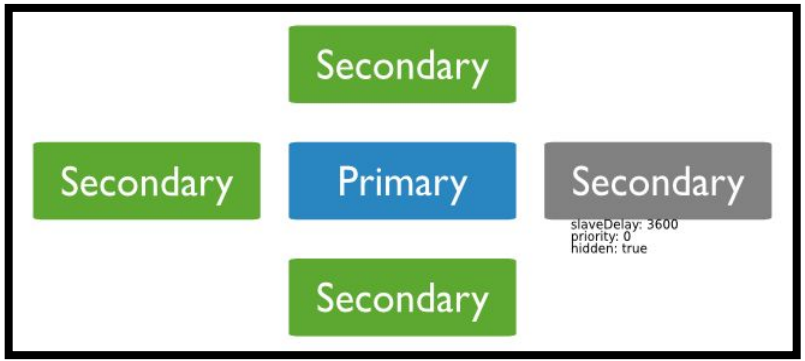


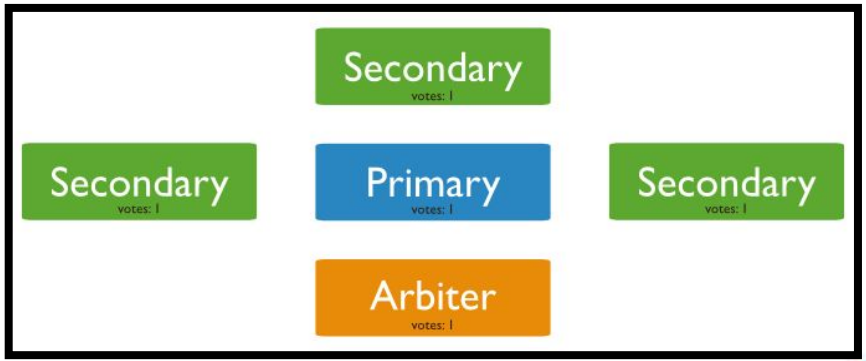


Existen los siguientes tipos de nodos para configurar un ReplicaSet:

Tipo	Permiso de voto	Puede ser primario	Descripción
Regular	SI	SI	Este es el tipo más común de nodo. Puede actuar como un nodo primario o secundario.
Arbiter	SI	NO	No es un nodo real, y se utiliza para deshacer mayorías en el caso de que el número de nodos sea par.
Delayed	SI	NO	Se utiliza como nodo de recuperación en caso de desastre. Se puede ajustar para estar con un retraso de un tiempo determinado.
Hidden	NO	NO	Se suele usar para análisis en el ReplicaSet.

- Ejemplos:





- Todas las escrituras se realizan sobre el nodo primario.
- Si sólo se lee y se escribe en el nodo primario, tendríamos garantía de consistencia.
- El nivel de garantía de escritura (**Write Concern**) proporciona la información sobre el éxito en una operación de escritura.
- Se configura para cada operación, colección, base de datos o conexión.
- MongoDB tiene los siguientes niveles de **Write Concern**, la lista va del más débil al más fuerte:
    - **Unacknowledged**: no hace Acknowledged de la recepción. Es similar a ignorar errores.
    - **Acknowledged**: confirma que ha recibido la orden de escritura y que ha aplicado el cambio en memoria. Permite capturar errores de red, claves duplicadas, etc. **(*)**
    - **Journaled**: hace Acknowledged de la orden de escritura después de haber hecho el commit al **journal**. Esto permite que MongoDB se reestablezca de un shutdown.
    - **Replica Acknowledged**: se garantiza que se ha replicado la operación de escritura al resto de miembros.

Los niveles anteriores se pueden configurar en base a 3 parámetros principales, por separado o a la vez:

Opción w: nivel de confirmación de escritura .El resto de escrituras se harán de forma asíncronas entre replicas en base al **oplog** del nodo primario.

- ***w=0***: Unacknowledged, no garantiza escritura en memoria.
- ***w=1***: garantiza la escritura en memoria para standalone mongod o nodo primario junto con oplog.
- ***w=n***: garantiza la escritura en memoria en n replicas. Nunca debe ser superior al numero total de réplicas para evitar bloqueos.
- ***w="majority"***: garantiza la escritura en la mayoría de replicas con derecho a voto.

Opción j: nivel de confirmación en Journal. Asegura o no que se realiza la escritura en disco.

- ***j=false***: no garantiza la escritura en disco (journal).
- ***j=true***: garantiza la escritura en disco (journal) en nodo primario. Tiene más prioridad que opción w=0.

Opción wtimeout (milisegundos): para las escrituras con Replica Acknowledged (w>1) se puede establecer un tiempo de espera de respuesta. Esta opción devolverá error si dicho tiempo se cumple sin recibir señal de aprobación, aunque la operación pueda finalizar correctamente.

- ***wtimeout 0***: si alguna replicación cancela, la operación no finaliza.
- ***wtimeout n***: espera a la respuesta de confirmación de las replicas durante n milisegundos.

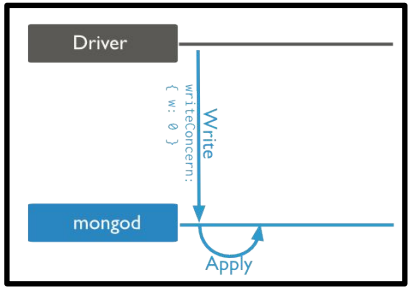
- Para cargas masivas (**bulk**), se suele utilizar un nivel de **Write Concern** de **Replica Acknowledged**. Una gran cantidad de operaciones se registran en el **oplog** del primario cuyo tamaño es limitado y este puede verse afectado antes de que comience la replica asíncrona. De esta forma, se aseguran N replicas antes de lanzar la confirmación al driver, asegurando la consistencia de la operación en N replicas. El resto de replicaciones continúa de forma asíncrona basándose en el **oplog** de la replica origen.

- Existe un proceso que realiza la copia en disco cada intervalo de tiempo . Marcando la opción **j=true** se confirma que se ha registrado en disco.

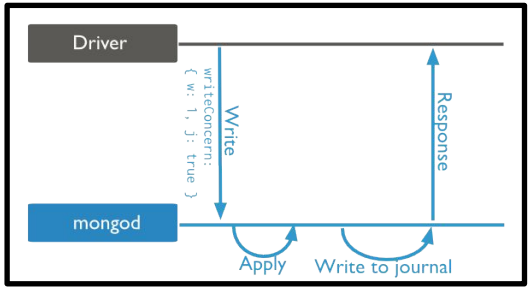
- Con un valor alto de **w**, el tiempo para cada operación aumenta ya que se debe replicar tantas veces como se indica en el valor.

- La atomicidad se establece a nivel de documento. Para asegurar el orden en las operaciones, se puede configurar el parámetro ***$isolated***. Consigue que las operaciones se realicen de forma consecutiva sin que otras operaciones puedan mezclarse antes de finalizar. Devuelve el control cuando finaliza (error o no). No existe la opción de **rollback** para las operaciones que han finalizado correctamente hasta el error. Esta opción no trabaja con shard.

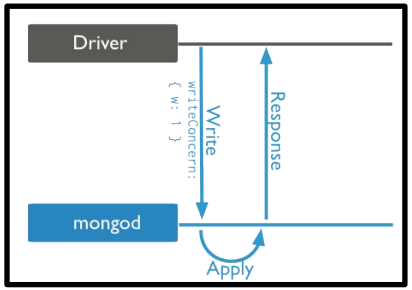
Unacknowledged



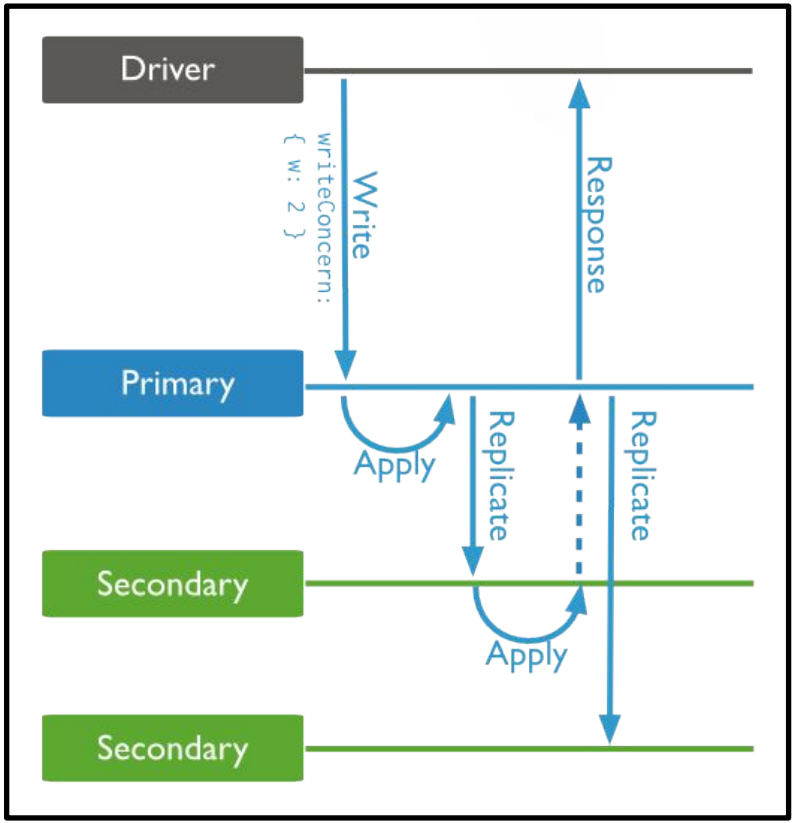
Journaled



Acknowledged

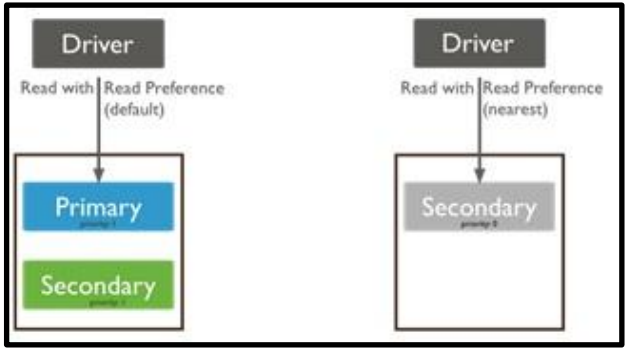


Replica Acknowledged



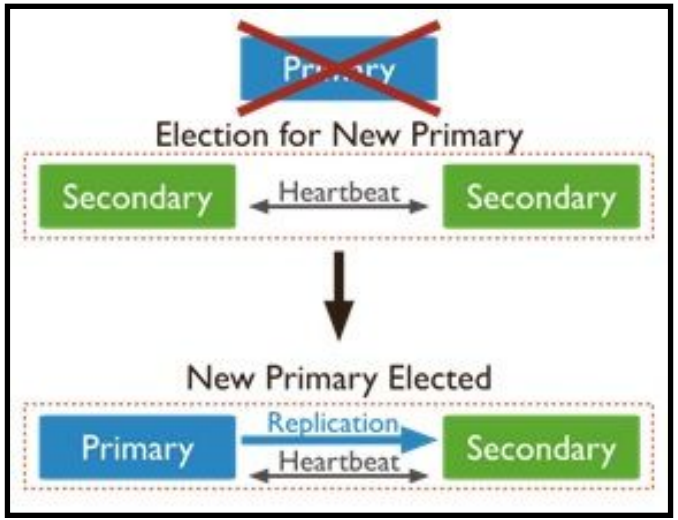
MongoDB realiza sus lecturas al primario para proporcionar una consistencia fuerte entre los datos escritos con los leídos, sin embargo, permite que este comportamiento sea modificado en su configuración de acuerdo a las necesidades que se tengan y poder leer de los secundarios:

- **primary**: modo por defecto, todas las operaciones de lectura van al primario.
- **primaryPrefered**: permite que las operaciones de lectura vayan a secundarios en caso que no exista primario disponible ese momento.
- **secondary**: todas las operaciones de lectura van a los nodos secundarios.
- **secondaryPrefered**: permite que las operaciones de lectura vayan a primarios en casos que no existan secundarios disponibles en ese momento.
- **nearest**: las operaciones de lectura van al miembro del **Replica Set** que tenga la menor latencia de red, indistintamente de si es primario o secundario.

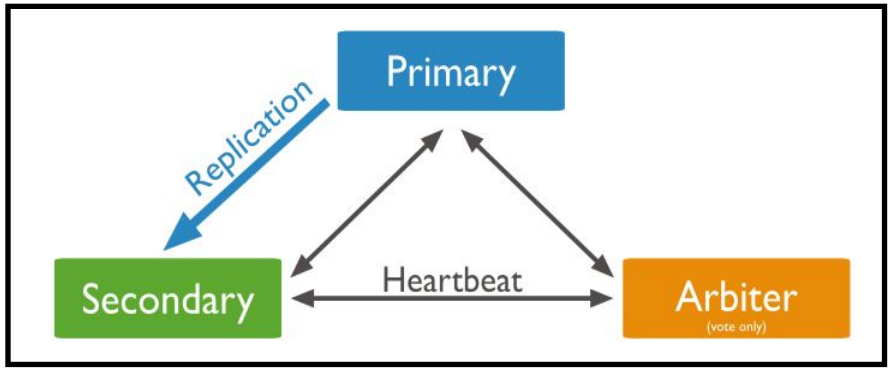


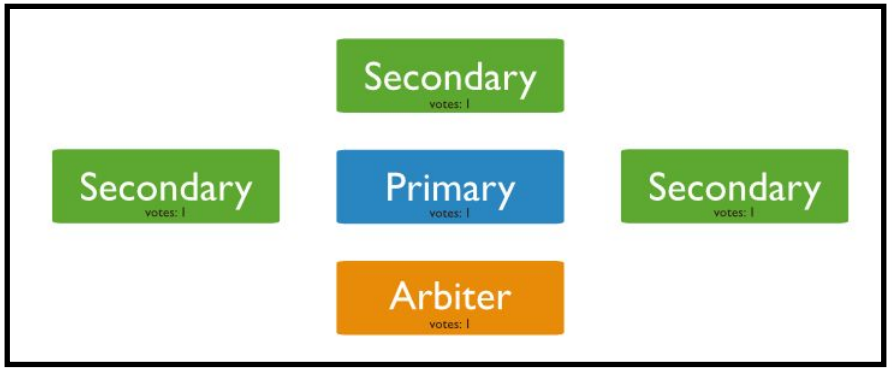
- Las operaciones de lectura desde miembros secundarios del **Replica Set** no garantizan que reflejen el estado actual del primario.

- El número mínimo de nodos para conformar un conjunto de réplicas es de tres, ya que en caso de un fallo en el primario, un proceso de elección es activado para buscar entre los nodos restantes un único sustituto para seguir proporcionando el servicio.
- Cuando un nodo primario no tiene comunicación con los otros miembros del conjunto durante más de 10 segundos, el Replica Set intentará seleccionar otro miembro que se convertirá en el nuevo primario. El primer secundario que reciba la mayoría de votos se convertirá en primario.
- Si sólo existen dos nodos, no existiría mayoría en la elección y no se seleccionaría un nuevo nodo primario quedando el conjunto inactivo.



- Para facilitar el proceso de elección del nodo primario, se puede designar un nodo árbitro por **Replica Set**.
- Un árbitro no tiene copia del conjunto de datos y no puede convertirse en primario.
- El propósito de un árbitro es mantener el quórum en un conjunto de réplicas respondiendo al heartbeat y solicitudes de elección por parte de otros miembros del **Replica Set**.
- Si el **Replica Set** tiene un número par de miembros, hay que agregar un árbitro para obtener la mayoría de votos en una elección para primario.
- Los árbitros no requieren hardware dedicado.
- No se puede ejecutar un nodo árbitro en sistemas que también alojan miembros primarios o secundarios del **Replica Set**.





3. Sharding

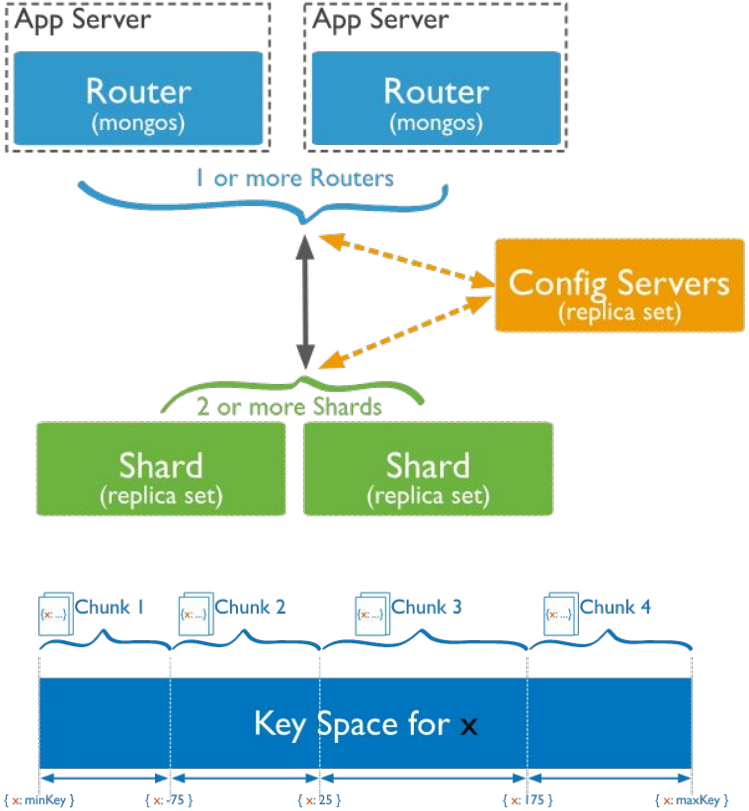
- La escalabilidad se consigue mediante **Sharding**.
- El **Sharding** es una técnica de escalabilidad horizontal que separa los datos a través de varios nodos. Se usa para volúmenes de datos muy grande.
- La forma de operar del **Sharding** consiste en dividir una colección en varios nodos y acceder a sus contenidos a través de un nodo especial que actúa como router (***mongos***).
- Los **Shards**, están formados por **Replica Set**, de modo que si tenemos 3 **Shards**, cada uno de los cuales tendrá un **Replica Set** con tres nodos, para un total de nueve nodos.
- Tipos de **Sharding**: por **rango**, **hash**, o definido por el usuario.
- El **Sharding** es transparente para las aplicaciones, incluso cuando se añaden nuevos nodos al cluster (rebalanceo automático).

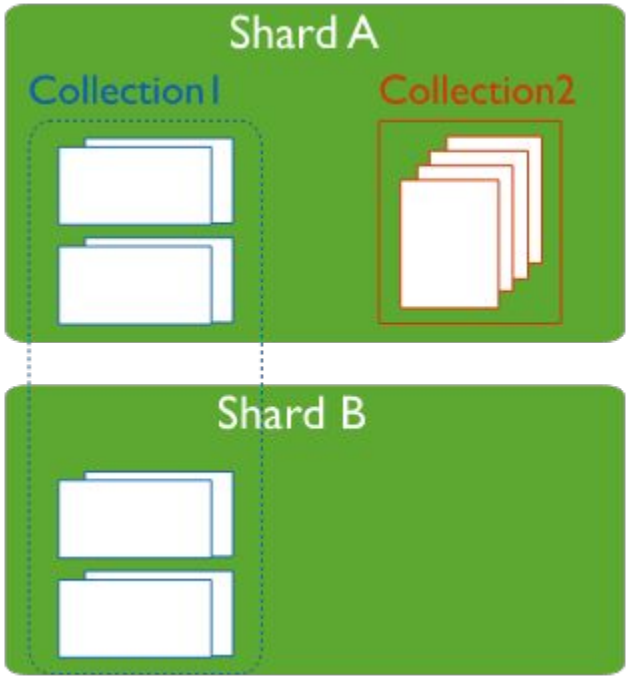
El uso de **Sharding** requiere que cada colección declare un **Shard-Key**:

- Clave definida para una o varios campos de un documento.
- Debe ayudar a dividir la colección en fragmentos (**chunks**).
- Los **chunks** son luego distribuidos entre nodos, lo más equitativamente posible.
- La instancia ***mongos*** usa la clave **Shard-Key** para determinar el **chunk** y así el nodo utilizado.

Las condiciones que un **Shard-Key** debe cumplir:

- Cada documento tiene un **Shard-Key**.
- El valor del **Shard-Key** no puede modificarse.
- Debe ser parte de un índice y debe ser el primer campo de un índice compuesto.
- No puede haber un índice único a no ser que esté conformado por el **Shard-Key**.
- Si no se usa el **Shard-Key** en una operación de lectura, la petición irá a todos los Shards.
- La clave de Shard debe ofrecer suficiente cardinalidad y aleatoriedad para garantizar que el reparto de carga sea equilibrado entre nodos.


App Server
Router
(mongos)
App Server
Router
(mongos)
1 or more Routers
Config Servers
(replica set)
2 or more Shards
Shard
(replica set)
Shard
(replica set)
Chunk 1
Chunk 2
Chunk 3
Chunk 4
Key Space for x
{ x: minKey }
{ x: -75 }
{ x: 25 }
{ x: 175 }
{ x: maxKey }


Shard A
Collection1
Collection2
Shard B

Componentes:

- **Client Application y Driver**: las aplicaciones cuando necesitan comunicarse con la base de datos de MongoDB lo hacen a través de un *driver*, estos tienen implementados los métodos y protocolos necesarios para comunicarse correctamente con la base de datos encapsulando la complejidad del proceso al desarrollador.

- **Shard**: un fragmento o *shard* es aquel que posee los datos fragmentados de las colecciones que componen la base de datos como tal, este suele estar compuesto por un **Replica Set** preferiblemente; sin embargo en ambientes de desarrollo podría ser una única instancia por fragmento.

- **Router**: debido a que las aplicaciones ven la base de datos como un todo, el router es el encargado de recibir las peticiones y dirigir las operaciones necesarias al shard o shards correspondientes. En ambientes de producción es común tener varios routers para balancear la carga de peticiones de los clientes.

- **Config Server**: este tipo de instancias se encargan de almacenar la metadata del cluster de fragmentación, es decir, qué rangos definen un trozo de una colección y qué trozos se encuentran en qué fragmento. Esta información es almacenada en caché por el router para lograr un óptimo tiempo de procesamiento. En ambientes de producción se deben tener 3 servidores de configuración ya que si sólo se posee uno y este falla, el cluster puede quedar inaccesible.

4. Storage Engine

MMAPv1 (deprecado)

WiredTiger (por defecto)

- Los datos en el disco están almacenados en una estructura arborescente autobalanceada, que recupera el espacio de forma regular. Es la opción de storage recomendada por Mongo. Con **WiredTiger**, se soporta compresión para todas las colecciones e índices. La compresión reduce al mínimo el uso de almacenamiento. Por defecto, **WiredTiger** utiliza la compresión ***snappy***, que es una biblioteca de compresión desarrollada en Google. Pero si hay problemas de espacio en disco existe otra opción de compresión en bloques denominada ***zlib***, que necesita aumentar un 25% la CPU de cada nodo. A partir de la versión 4.2 existe un nuevo motor de compresión llamado ***zstd*** que mejora la compresión necesitando menor % de CPU. Los requisitos de memoria necesarios son 50 % de (RAM -1 GB) y como mínimo 256 MB

In-Memory

- Todos los datos y metadatos se escriben en memoria lo que reduce las tiempos de escritura y lectura. No tienen durabilidad, no persisten tras un reinicio. Los requisitos por defecto de memoria son del 50% de RAM menos 1 GB

Recomendaciones sobre qué motor utilizar:

- **WiredTiger** es un motor de almacenamiento de propósito general que debería ser la opción habitual. Su rendimiento en lectura/escritura es excelente, y su capacidad de aprovechamiento de arquitecturas multicore es muy buena. Además, proporciona compresión de datos en el disco, reduciendo las necesidades de almacenamiento, y compresión de índices en memoria, reduciendo las necesidades de memoria del servidor.

- **WiredTiger e In-Memory** proporciona bloqueos a nivel de documento. Esto redunda en un rendimiento muy superior en los escenarios de escritura.

- **WiredTiger** gestiona de forma autónoma los recursos de memoria, y es configurable la forma en que lo hace.

- **In-Memory** mejora la latencia de escritura y lectura al no escribir en disco pero **no persiste** la información tras un reinicio del mongod.

- Una posible **arquitectura** con **Replicaset** con Storage Engine **In-Memory** sería:
  - 2 nodos mongod con In-Memory como Storage Engine
  - 1 nodo mongod con WiredTIger como Storage Engine pero como un miembro hidden.

ntic master
revolucionamos la comunicación

UNIVERSIDAD COMPLUTENSE DE MADRID