



Formación MongoDB

{ paradigma

Índice

1. Introducción
2. Instalación básica
3. Shell de MongoDB
4. Modelado de datos
5. Replica Set y Sharding
6. Recomendaciones Generales
7. Operations Manager
8. Conclusiones

Material del curso

Todo los ejercicios los puedes encontrar en el siguiente repo de github

<https://github.com/evalero/Curso-MongoDB>

1 Introducción

Índice

1. ¿Qué es MongoDB?
2. Empresa y fundadores
3. Principales casos de uso
4. Analogía MongoDB y mundo relacional
5. Documentos
6. Colecciones
7. Arquitecturas de despliegue
8. Storage Engine

1. ¿Qué es MongoDB?

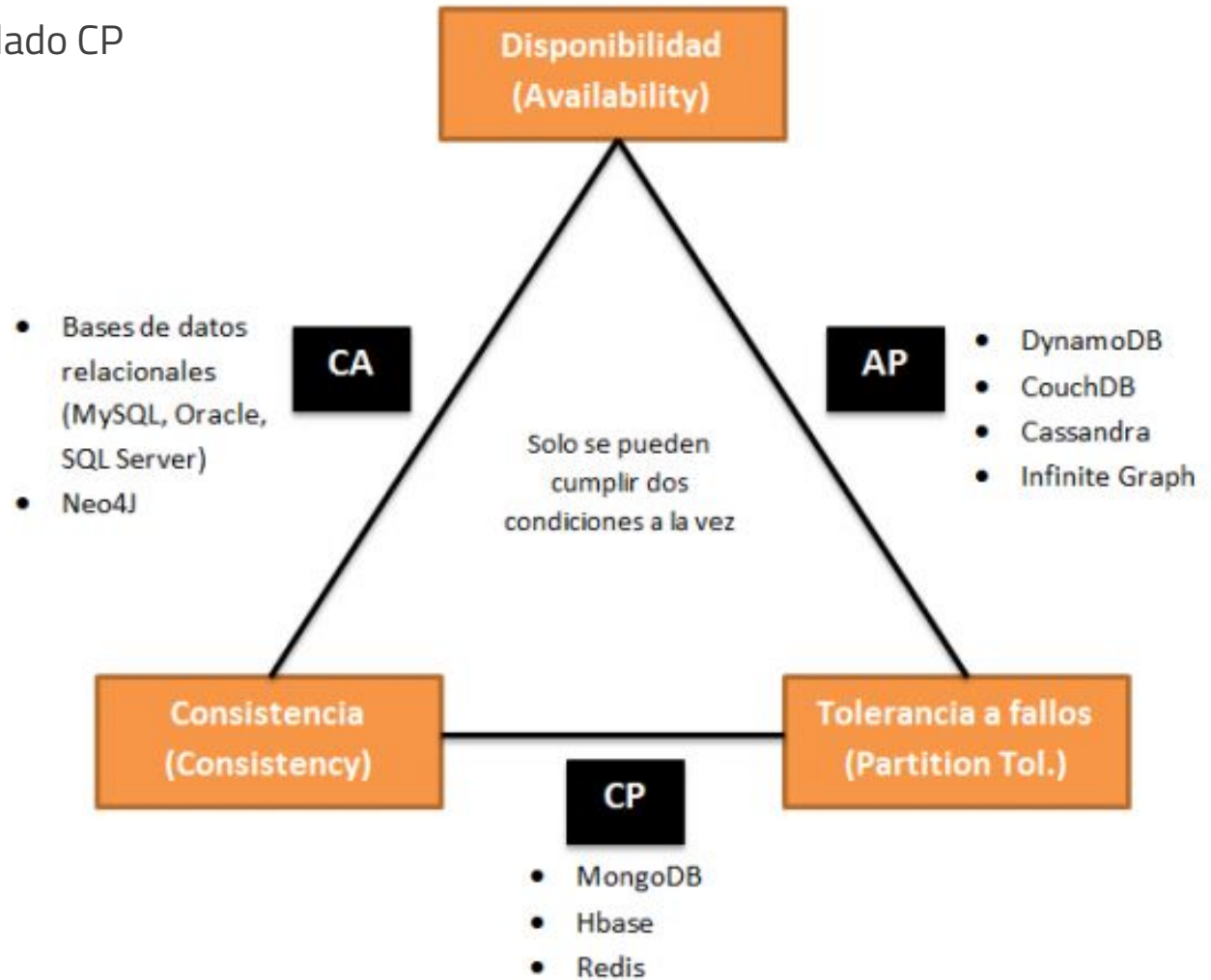
- Es una base de datos NoSQL orientada a **documentos**
- La estructura de los documentos es en formato **JSON**. Internamente los documentos son almacenados en formato **BSON**
- Al tener documentos más ricos, se **reduce el I/O** sobre la BD (no hay Joins).
- Permite operaciones **CRUD** (Create, Read, Update Delete) con una **sintaxis parecida a JavaScript**
- Proporciona **replicación** y **alta disponibilidad** a través de Replica Sets
- También **dispone de balanceo** y **escalado horizontal** usando **Sharding**. El rebalanceo de los datos se realiza automáticamente



- Ofrece un mecanismo de procesamiento masivo de datos a través de **operaciones de agregación**.
- El grueso de los **datos reside en memoria**, por lo que las lecturas y las escrituras son muy rápidas.
- Permite también crear colecciones de tipo circular de tamaño fijo y mantiene el orden según se han ido insertando los datos (Capped Collections)
- Tienen múltiples storage engine diferentes, nmap, WiredTiger, In-Memory, y Encrypted
- **Homongous** : Gigantesco, enorme



En el teorema CAP, MongoDB se encuentra en el lado CP
(Consistencia y Persistencia)



2 Empresa y fundadores

- La empresa detrás del proyecto es MongoDB inc. (anteriormente 10gen)
- Los fundadores de este proyecto son:



Kevin P. Ryan



Dwight Merriman

3 Casos de uso

MongoDB es un producto de propósito general y es muy útil para múltiples casos de uso tales como:

- CMS
- Aplicaciones móviles
- Gaming
- E-commerce
- Analytics
- Bussiness intelligence
- Proyectos Big Data
- Web caché

Transacciones JOIN'S



4 Analogía MongoDB y mundo relacional

A modo introductorio, vamos a exponer algunos términos utilizados en MongoDB y su “correspondencia” en el mundo relacional

RELACIONAL	MONGO DB
Base de datos	Base de datos
Tabla	Colección
Fila	Documento
Índice	Índice
Insert	Insert
Select	Find
Update	Update
Delete	Remove

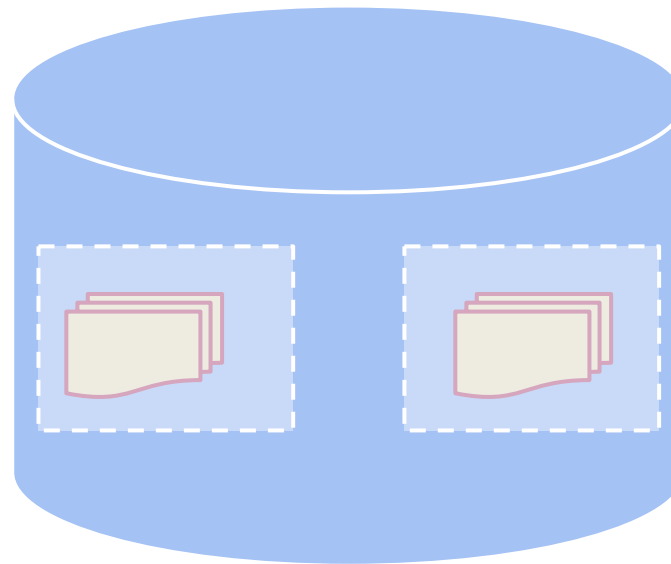
5 Documentos

- Cada entrada de una colección es un documento
- Son estructuras de datos compuestas por campos de clave-valor
- Los documentos tienen una estructura similar a objetos JSON
- Los documentos se corresponden con tipos de datos nativos en los lenguajes de programación
- En un documento es posible embeber otros documentos o arrays
- Se tiene un esquema dinámico que permite polimorfismo de manera fluida
- Las operaciones de escritura son sólo atómicas a nivel de documento

```
{
  "_id" : ObjectId("5457a502e308f720d8999e97"),
  "Nombre": "Ernesto",
  "Apellidos" : "Valero Marcelo",
  "Edad" : 28,
  "Aficiones": "{ \"Comics\" : null,
                  \"Deportes\": [\"squash\", \"natación\"]
                },
  "Empresa" : "Red Hat",
  "Cargo" : "MongoDB DBA",
  "Tecnologías" : [\"Openstack\" , \"Openshift\"],
  "Proyectos" : "{ \"Openstack\": [Cliente1,Cliente2],
                   \"Openshift\": [Cliente4]
                 }
}
```

6 Colecciones

- Una colección es una agrupación de documentos existente en una única base de datos.
- Las colecciones no tienen un esquema predefinido
- Los documentos de una misma colección pueden tener diferentes campos
- Típicamente, todos los documentos de una misma colección tienden a ser similares



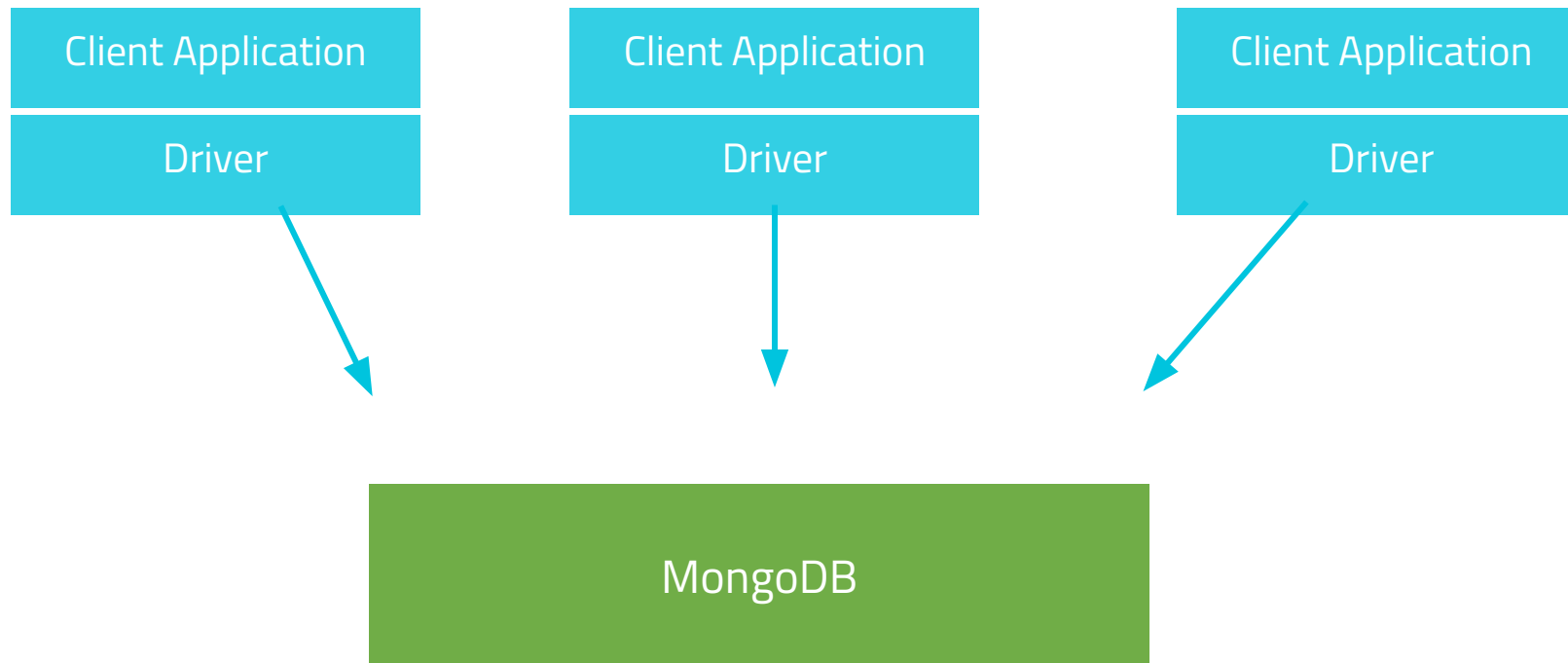
Ejemplo de documentos de la colección libros

```
{  
  "Título": " MongoDB: The Definitive Guide",  
  "Autor" : "Kristina Chodorow, Michael Dirolf",  
  "Género" : "Manual tecnológico",  
  "Tecnologías": "[\"MongoDB\", \"NoSQL\"]"  
}
```

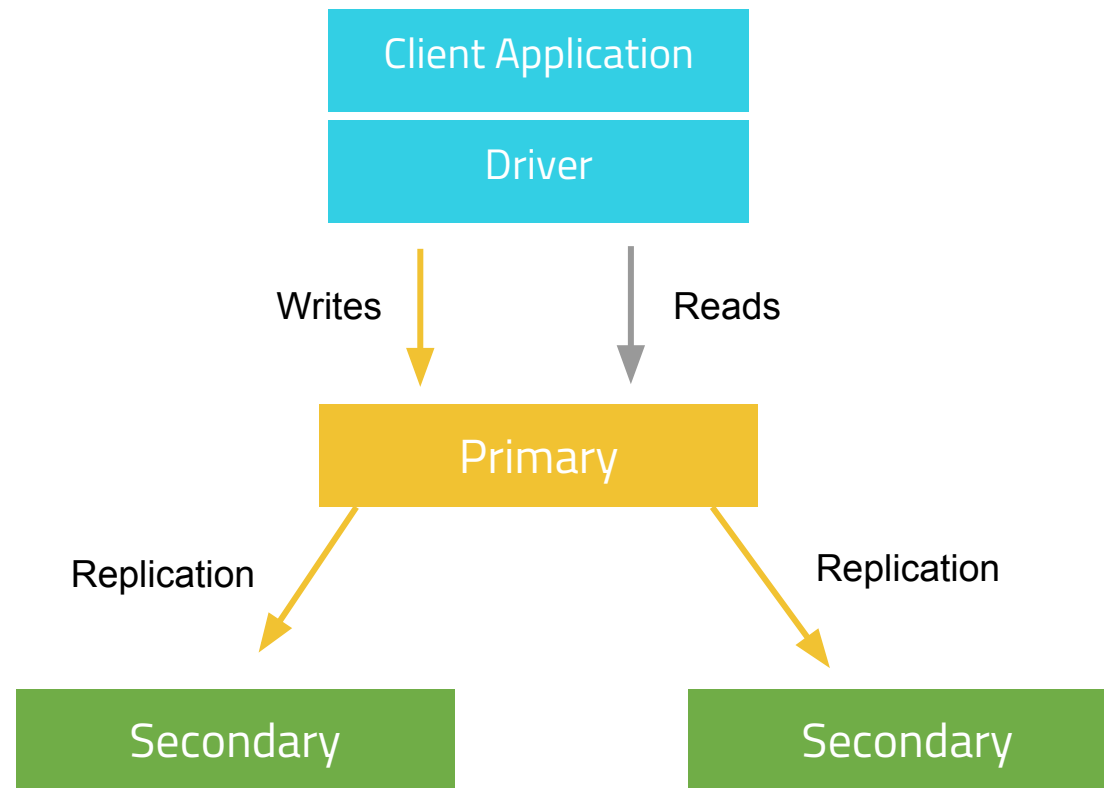
```
{  
  "Título": "La isla del tesoro",  
  "Autor" : "Robert Louis Stevenson",  
  "Género" : "Aventuras"  
}
```


7 Arquitecturas de despliegue

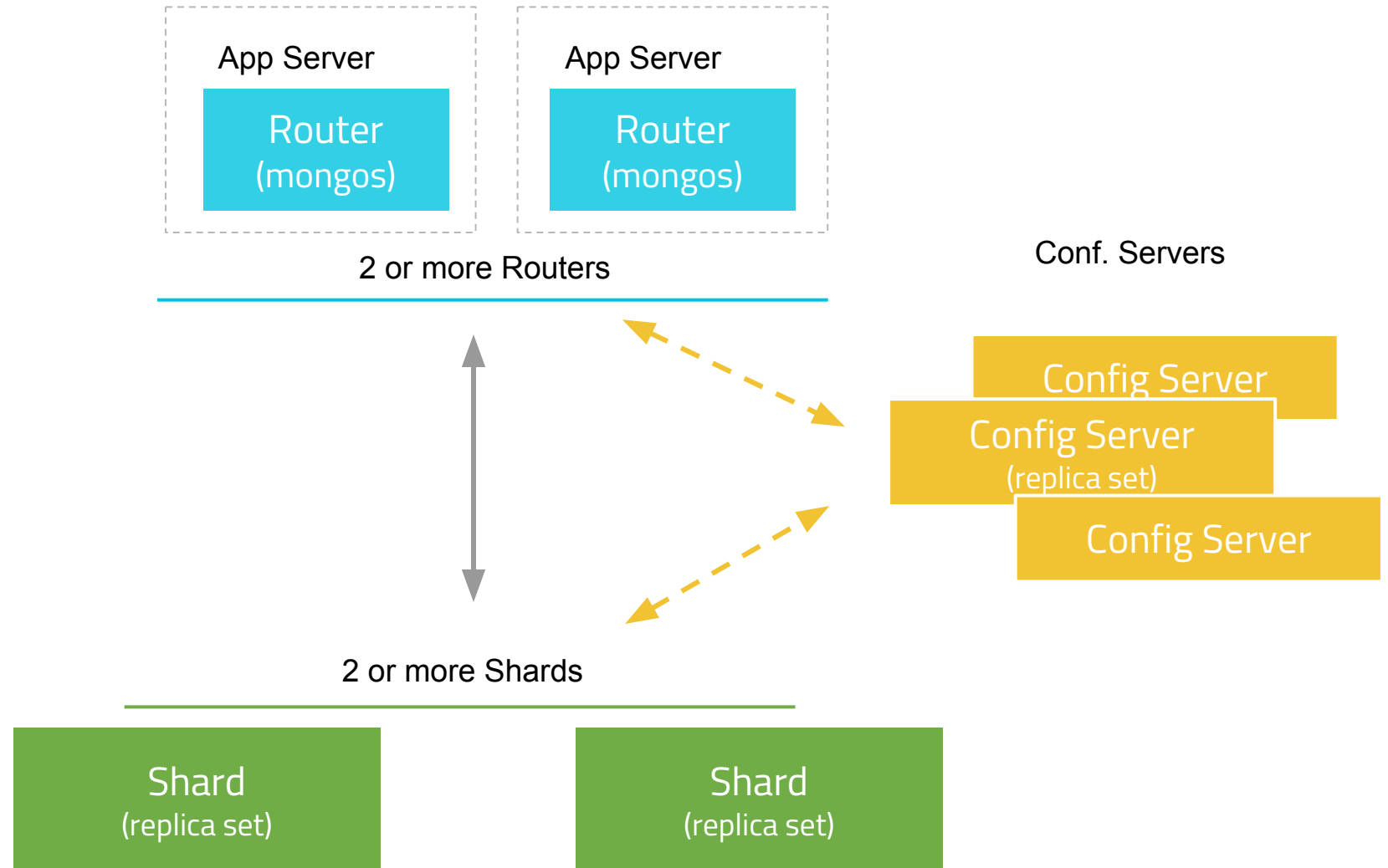
- Standalone



- Replica set



- Sharding



8 Storage engine

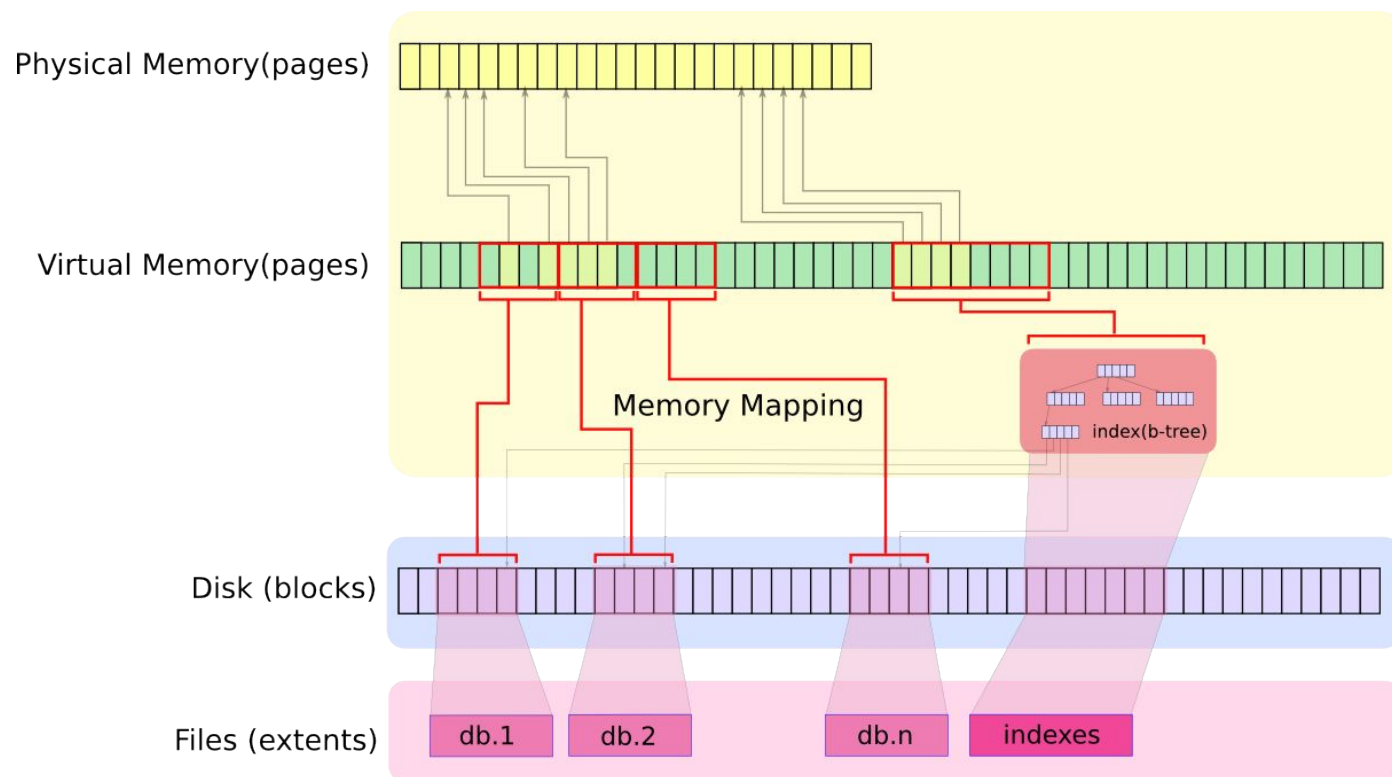
WiredTiger: Ofrece capacidad de comprimir los datos y mejora el paralelismo para el acceso a los datos. Permite determinar una caché máxima para almacenar datos en memoria. Este motor de almacenamiento es muy útil para :

- Albergar grandes bases de datos por tener un buen ratio de compresión
- Aplicaciones que sean altamente escritoras
- Aplicaciones que mezclen escrituras y lecturas
- Al ser concurrente a nivel de documento, permite mayor concurrencia de operaciones



MongoDB ofrece la posibilidad de trabajar con distintos Storage Engine dependiendo de la naturaleza de nuestra aplicación.

MMAPv1: es el motor de almacenamiento por defecto. Utiliza ficheros mapeados en memoria. Utiliza la llamada del sistema `mmap()` para mapear bloques de ficheros de datos en memoria virtual. **MMAP es recomendado para aplicaciones “lectoras”**. El nivel de bloqueo es a nivel de **colección**



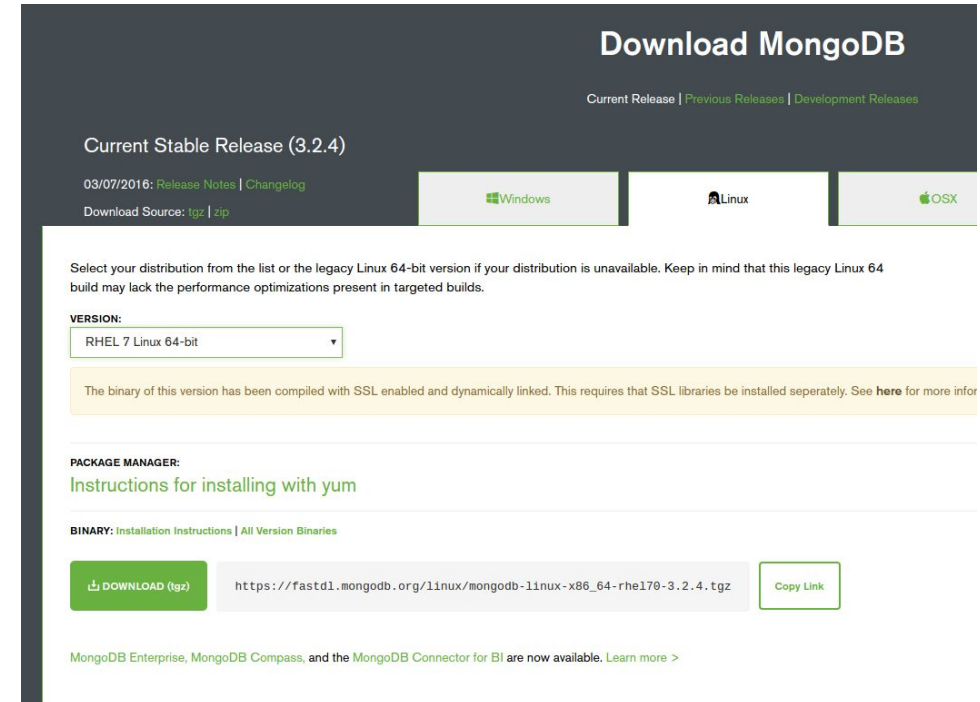
2 Instalación básica

Índice

1. Instalación básica
2. Binarios
3. Fichero de configuración

En la web de [MongoDB](#) encontramos los enlaces de descarga en los siguientes formatos:

- Código fuente
- Paquetería para distintos S.O. y distribuciones
- Binarios precompilados



The screenshot shows the 'Download MongoDB' page. At the top, it says 'Current Stable Release (3.2.4)' with links for 'Current Release', 'Previous Releases', and 'Development Releases'. Below this, there are tabs for 'Windows', 'Linux', and 'OSX'. The 'Linux' tab is selected. The page instructs users to select their distribution from a list, with a dropdown menu currently showing 'RHEL 7 Linux 64-bit'. A note states that the binary is compiled with SSL enabled and dynamically linked, requiring separate SSL library installation. Under the 'PACKAGE MANAGER' section, it provides 'Instructions for installing with yum'. The 'BINARY' section includes a 'DOWNLOAD (tgz)' button and a 'Copy Link' button next to the URL 'https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-rhel70-3.2.4.tgz'. At the bottom, it mentions 'MongoDB Enterprise, MongoDB Compass, and the MongoDB Connector for BI are now available' with a 'Learn more >' link.

PRACTIQUEMOS

Descarga de binario

1. Descargamos los paquetes de [mongodb.org](https://fastdl.mongodb.org)
wget https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-rhel70-3.2.4.tgz
2. Descomprimos el tar en el directorio deseado
tar -xzvf mongodb-linux-x86_64-rhel70-3.2.4.tgz -C /opt
3. Actualizamos los ficheros de `.bash_profile` para añadir la ruta donde hemos instalado mongodb

Yum

1. Instalamos la public key de MongoDB

```
sudo rpm --import https://www.mongodb.org/static/pgp/server-3.2.asc
```

2. Movemos el fichero ~/Labs/Tema2/mongodb-org-3.2.repo a la ruta /etc/yum.repos.d/

3. Instalamos el paquete

```
sudo yum install -y mongodb-org
```

4. Para las prácticas, SELINUX está deshabilitado. Para entornos productivos, se recomienda habilitarlo y añadir la siguiente regla para utilizar el puerto por defecto:

```
semanage port -a -t mongod_port_t -p tcp 27017
```

El paquete de MongoDB contiene los siguientes binarios. Los binarios para arrancar el **servidor de BBDD y acceso a la shell** de MongoDB son:

- **mongod**: Es el servicio principal de MongoDB. Maneja los accesos a los datos, las peticiones de datos y ejecuta tareas de mantenimiento en background. Su fichero de configuración es **mongod.conf**.
- **mongo**: Es la shell interactiva de MongoDB. Aporta un entorno funcional completo para ser usado con la BBDD.
- **mongos**: Es un servicio propio del modo de despliegue Shard. Su función es la de enrutar las peticiones de la capa de aplicación y determinar la ubicación de los datos en los diferentes shards del despliegue.

Herramientas para **backup y restore** de datos :

- **mongodump**: Es una utilidad para crear un export binario del contenido una base de datos. Podemos considerar MongoDB como una herramienta más para realizar copias de seguridad. Podremos usar esta herramienta contra “mongod” o “mongos” , teniendo en cuenta que “mongod” podrá estar arrancado o parado indistintamente.
- **mongorestore**: En conjunción con mongodump, mongorestore se utiliza para restaurar los respaldos realizados con mongodump.
- **mongooplog**: Es una herramienta que permite para hacer “polling” del oplog de un servidor remoto y aplicarlo sobre el servidor local. Esta utilidad lo podemos usar para realizar cierto clase de migraciones en tiempo-real, dónde se requiere que el servidor fuente se mantenga Online y en funcionamiento.

Herramientas para **exportación e importación** de datos :

- **bsondump**: Convierte ficheros BSON a algún formato legible por humanos, incluido a JSON. Se trata de una herramienta de análisis, en ningún caso debe ser utilizada para otro tipo de actividades.
- **mongoexport**: Utilidad que permite exportar los datos de una instancia de MongoDB en formato JSON o CSV. En conjunción con mongoimport son útiles para hacer backup de una parte bien definida de los datos de la BBDD MongoDB o para casos concretos de inserción de datos.
- **mongoimport**: Utilidad que permite importar los datos de una instancia de MongoDB desde ficheros con formato JSON o CSV.
- **mongofiles**: Utilidad que permite manejar ficheros en una instancia de MongoDB con objetos GridFS desde la línea de comandos. En Replica Set sólo podrá leer desde el primario.

Herramientas para **análisis de performance y actividad**:

- **mongoperf**: Utilidad para comprobar el performance I/O de forma independiente a MongoDB.
- **mongostat**: Utilidad que proporciona una rápida visión del estado actual de los servicios mongod y mongos. Es similar a la utilidad vmstat.
- **mongotop**: Proporciona un método para trazar el tiempo que una instancia de MongoDB gasta en las operaciones de Lectura/Escritura de datos. Proporciona estadísticas a nivel de colección, por defecto cada segundo.
- **mongosniff**: Es una utilidad que proporciona una visión a bajo nivel de la actividad de la BBDD a través de la actividad de red. Es equivalente a un analizador de tráfico de red TCP/IP

Para iniciar una instancia de MongoDB tenemos dos opciones:

- Ejecutar por línea de comandos el comando **mongod** con los parámetros de configuración que necesitemos:
`mongod --dbpath /data/db/ --port 27017 --fork --logpath/var/log/mongod/mongod.log`
- Crear un fichero donde exponemos toda la configuración del servidor y le pasamos por parámetro la información del fichero de configuración
`mongod -f /etc/mongod.conf`



```
systemLog:
  destination: file
  path: "/var/log/mongod/mongod.log"
  logAppend: true
storage:
  dbPath: /data/db
processManagement:
  fork: true
net:
  port: 27017
```

PRACTIQUEMOS

1. Accedemos al directorio de donde se encuentra el material del ejercicio descargado del repositorio

```
cd ~/RedHat-Mongodb/Tema2/ficheroConfiguracion
```

2. Observamos el contenido del fichero de configuración ejecutando el comando

```
cat mongod.conf
```

3. Ejecutamos el comando

```
mongod -f mongod.conf
```

3 Shell de MongoDB

Índice

1. Introducción
2. Primeros pasos
3. Uso de helpers de la shell
4. Aplicación de scripts desde la shell
5. GUI's
6. Comandos administrativos
7. Scripting con la shell
8. Funciones almacenadas en servidor

Para interactuar con nuestra base de datos utilizamos la shell **mongo**

- **Es una shell interactiva en JavaScript**
- Permite ejecutar scripts escritos en JavaScript para manipulación de datos, ejecución de comandos en la BBDD, aplicación de índices, etc.
- La shell puede ser utilizada tanto para la visualización de datos como para la administración de la base de datos, sea Standalone, Replica Set o Sharding

Para conectarnos con los datos de conexión por defecto (puerto 27017 y BBDD test): **mongo**

PRACTIQUEMOS

1. Nos conectamos a la instancia que hemos levantado anteriormente ejecutando el comando **mongo**
2. Una vez obtenemos el prompt, vamos a ejecutar los siguientes comandos:

```
show dbs
use miBD
show collections
db.holaMundo.insert({"Nombre": "<tu nombre>" , "Apellido" : "<tu
apellido>" , edad : <tu_edad> })
//Por ejemplo
db.holaMundo.insert({"Nombre" : "Ernesto", "Apellido" :
"Valero", "edad":30})
show collections
db.holaMundo.find({})
```

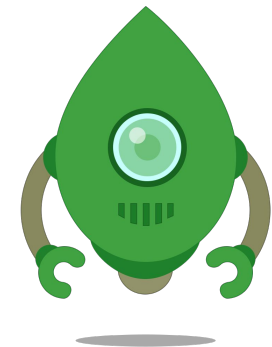
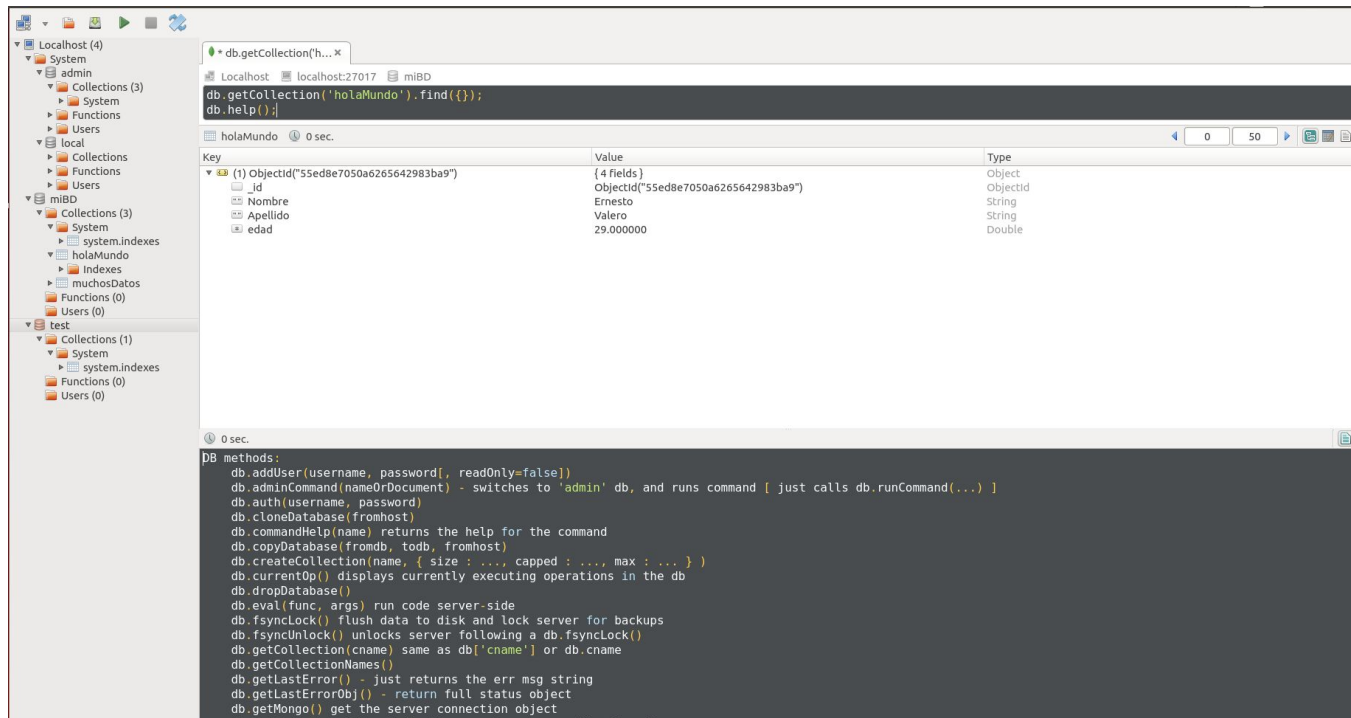
En la shell de Mongo podemos encontrar una serie de comandos que nos facilita la realización de alguna tarea o mostrar información sobre la BBDD o a r. Destacamos algunos de los más usados

Helper	Descripción
help	Muestra ayuda general
db.help()	Muestra ayuda sobre comandos de ejecutables sobre BBDD
db.<collection>.help()	Muestra ayuda sobre comandos de ejecutables sobre colecciones
show dbs	Muestra las BBDD del servidor
db	Devuelve el nombre de la BBDD donde nos encontramos posicionados

Helper	Descripción
show collections	Muestra las colecciones contenidas en la BBDD donde estamos posicionados
use <db>	Nos posicionamos sobre la BBDD db
show users	Muestra los usuarios sobre la BBDD actual
load("<ruta_script>")	Carga en la sesión actual el script contenido en la ruta ruta_script
it	Itera el cursor sobre el que se haya hecho una query

1. Salimos de la shell de mongo usando el comando **exit**
2. Ejecutamos los siguientes comandos
`cd ~/RedHat-Mongodb/Tema3/shell`
`mongo --shell generaDatos.js`
3. Ejecutamos dentro de la shell de mongo los siguientes comandos
`show dbs`
`use miBD`
`show collections`
`db.muchosDatos.find({x:{$lte:10}})`
`db.muchosDatos.remove({z:{$gt:100}})`

Existen herramientas gráficas que también nos permiten acceder a nuestra BBDD y administrarla de manera más intuitiva.



Robomongo

Existe una serie de comandos administrativos que son interesantes para obtener información sobre la BBDD o una colección

Helper	Descripción
<code>db.setProfilingLevel(<int>)</code>	Activa el log de operaciones sobre la base de datos referenciada. El parámetro puede ser 0-> desactivado 1->queries lentas 2->Todas las queries
<code>db.stats()</code>	Devuelve un documento con las estadísticas de almacenamiento de la BBDD actual
<code>db.collection.stats()</code>	Similar al comando anterior pero referenciado sobre una colección particular
<code>db.collection.find().explain()</code>	Devuelve el resultado del query planner para observar como se ha ejecutado la query

```
use miBD
db.stats()
db.muchosDatos.find({x:{$gte:1040}}, {y:1})
db.muchosDatos.find({x:{$gte:1040}}, {y:1}).explain()

db.muchosDatos.createIndex({x:1,y:1})
db.stats()
db.muchosDatos.find({x:{$gte:1040}}, {y:1})
db.muchosDatos.find({x:{$gte:1040}}, {y:1}).explain()

db.setProfilingLevel(2)
db.muchosDatos.insert({"mensaje":"Este mensaje va a ser traceado"})
db.system-profile.find()
```

En ciertos casos nos puede resultar interesante hacer scripts para ejecutar sobre la Base de Datos algún tipo de operación. Cuando se desarrolla un script para ejecutar contra la base de datos, **no pueden usarse los helpers**

Helper	Equivalencia
show dbs, show databases	<code>db.adminCommand('listDatabases')</code>
use db	<code>db = db.getSiblingDB('<db>')</code>
show collections	<code>db.getCollectionNames()</code>
show users	<code>db.getUsers()</code>
db	Devuelve el nombre de la BBDD donde nos encontramos posicionados

Helper	Equivalencia
it	<pre>cursor = db.collection.find() if (cursor.hasNext()){ cursor.next(); }</pre>

1. Abrimos el directorio ~/RedHat-Mongodb/Tema3/scripts
#cd ~/RedHat-Mongodb/Tema3/scripts
2. Cargamos los datos
#mongo generaDatos.js
3. Nos metemos en la instancia de Mongo y comprobamos que colecciones tienen datos
#mongo Script
show collections
db.coleccionX.find()
4. Salimos de la estancia de Mongo y ejecutamos el script
#mongo borradoColecciones.js
5. Repetimos el paso 3

En ocasiones puede interesarnos almacenar funciones en nuestro servidor para una Base de Datos. Para ello usamos la colección `system.js` añadiendo un documento con el nombre de la función y la implementación de la misma:

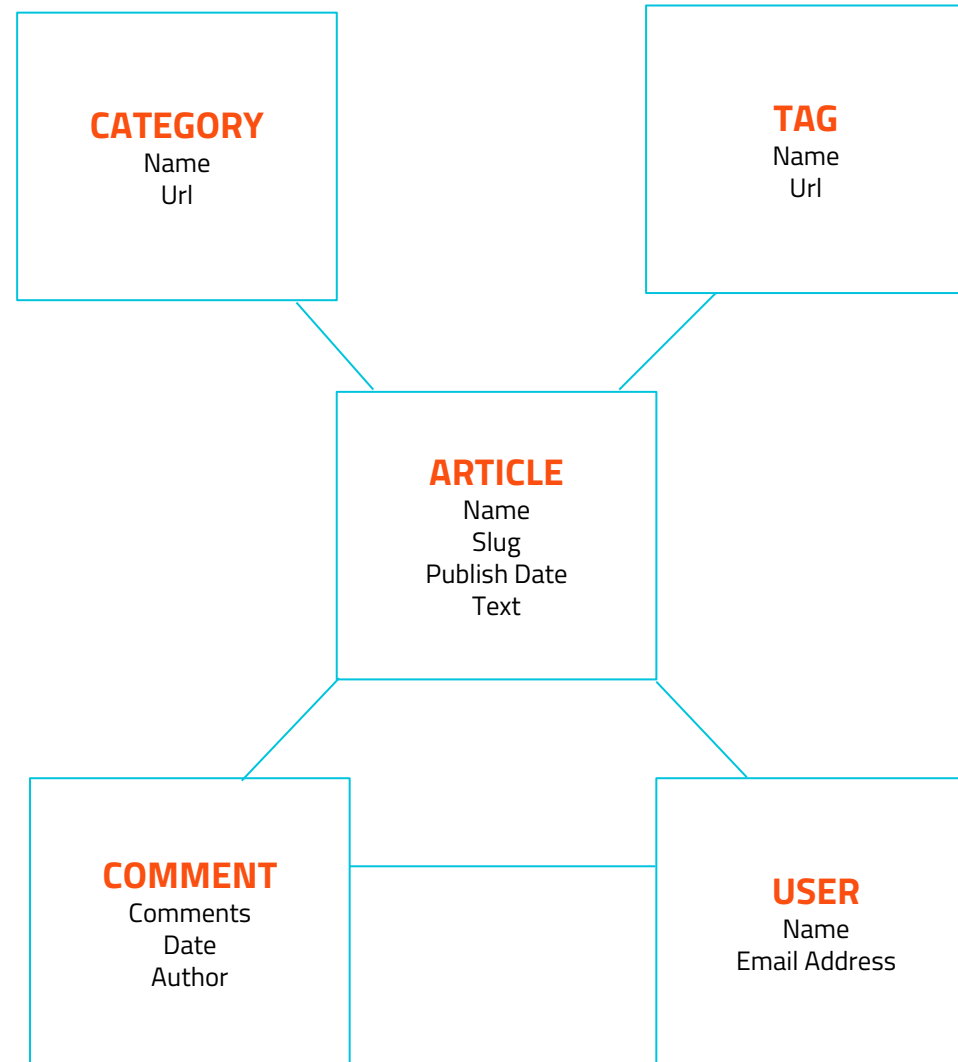
```
db.system.js.save(  
  {  
    "_id" : "listaColl",  
    "value" : function (x){return db.getCollectionNames();}  
  })  
  
db.loadServerScripts();  
  
listaColl();
```

4 Modelado de datos

Índice

1. Consideraciones generales
2. Relaciones entre entidades
3. Relaciones 1 : n
4. Relaciones m : n
5. Consideraciones

- La unidad de información es el **documento**.
- Una **colección** es un conjunto de documentos.
- No existe una definición de esquema tal cual existe en SQL. En cualquier colección pueden convivir documentos de diferente definición (tanto diferentes campos como diferentes tipos). **Normalmente** los documentos de la misma colección tienen los mismos campos.
- MongoDB no exige “ninguna restricción” a los documentos antes de insertarlos en una colección.
- La definición de las colecciones y documentos debe estar orientado a las **necesidades de la aplicación** (en contra de la tercera forma normal de SQL).
- NO existen joins, **no es posible en una misma consulta recuperar información de dos colecciones**.
- Podemos definir **índices** para mejorar el acceso a la información.

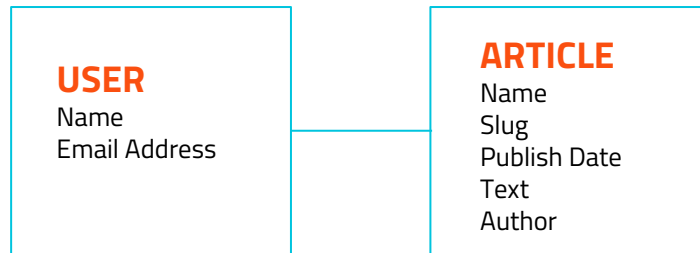


Embebidas



```
{
  name : "Lorem ipsum",
  publis_date : "27/03/1983",
  text : "Lorem ipsum ...",
  comment : [
    {
      comment : "comment lorem... ",
      date : "28/09/1983" ,
    },
    {
      comment : "comment lorem... ",
      date : "30/09/1983" ,
    }
  ]
}
```

Relación



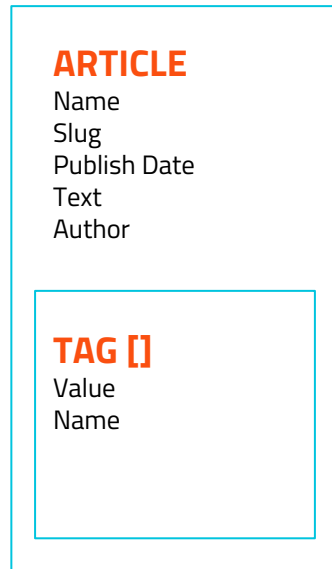
USER:

```
{  
  name : "user1",  
  email : "user@user.es"  
}
```

ARTICLE:

```
{  
  name : "Lorem"  
  publis_date : "27/03/1983"  
  text : "Lorem ipsum"  
  author : "user1"  
}
```

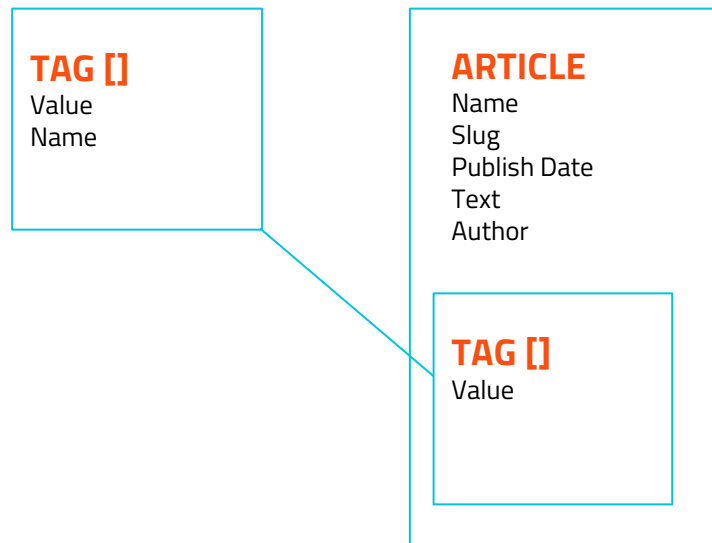
Embebida



ARTICLE:

```
{
  name : "Lorem"
  publis_date : "27/03/1983"
  text : "Lorem ipsum"
  author : "user1"
  tag: [{
    name : "tag1",
    value : "tag1"
  }]
}
{
  name : "Lorem 2"
  publis_date : "30/03/1983"
  text : "Lorem ipsum"
  author : "user1"
  tag: [{
    name : "tag2",
    value : "tag2"
  },
  {
    name : "tag1",
    value : "tag1"
  }]
}
```

Relacional



ARTICLE:

```

{
  name : "Lorem"
  publis_date : "27/03/1983"
  text : "Lorem ipsum"
  author : "user1"
  tag: [{
    value : "tag1"
  }]
}
{
  name : "Lorem 2"
  publis_date : "30/03/1983"
  text : "Lorem ipsum"
  author : "user1"
  tag: [{
    value : "tag1"
  },
  {
    value : "tag2"
  }]
}

```

TAG:

```

{
  name : "tag1",
  value : "tag1"
},
{
  name : "tag2",
  value : "tag2"
}

```

No existe una receta para el modelado de datos en MongoDB.

- El “esquema” de información debe estar guiado por las necesidades de la aplicación.
- Tener en cuenta las operaciones de **lectura/escritura**
- Realizar operaciones en bloque (que afecten a muchos documentos).
- Consideraciones a tener en cuenta:
 - Rendimiento de operaciones
 - Lecturas a múltiples colecciones (relaciones) o a una sola (embebidas)
 - Escrituras sobre entidades embebidas
 - Limitar el posible crecimiento del documento (escrituras)
 - Complejidad de las operaciones
 - Lecturas sobre campos de entidades embebidas
 - Actualizaciones sobre entidades embebidas
 - Gestión de la duplicidad de información
 - Las escrituras son atómicas a nivel de **documento**
 - Tener en cuenta los tamaño de N en las relaciones 1:N y N:M

No existe una receta para el modelado de datos en MongoDB.

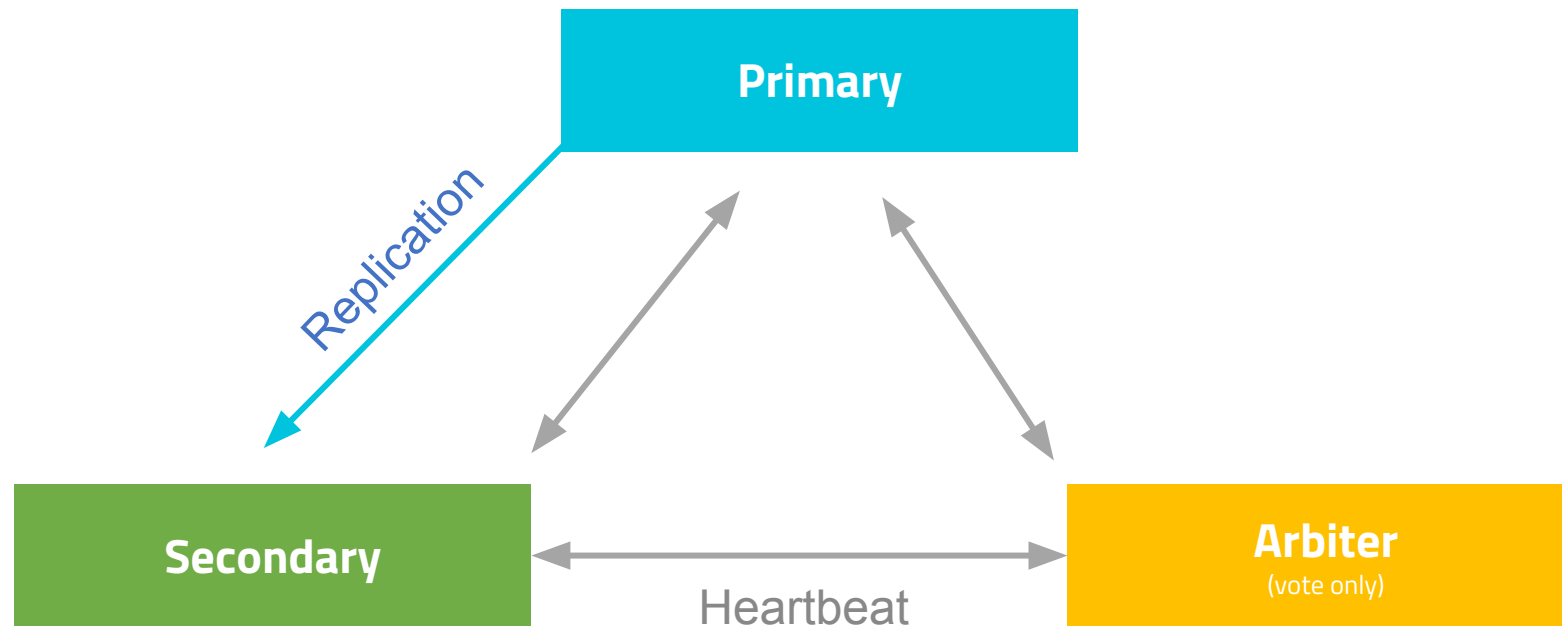
- Recomendaciones generales:
 - Intentar embeber entidades siempre que sea posible.
 - Tener que acceder únicamente a un objeto embebido es una señal para no embeberlo
 - Listas de objetos muy grandes, es otro motivo para no embeberlos
 - Uso adecuado de **índices** y **proyecciones**.

5 Replica Set y Sharding

Índice

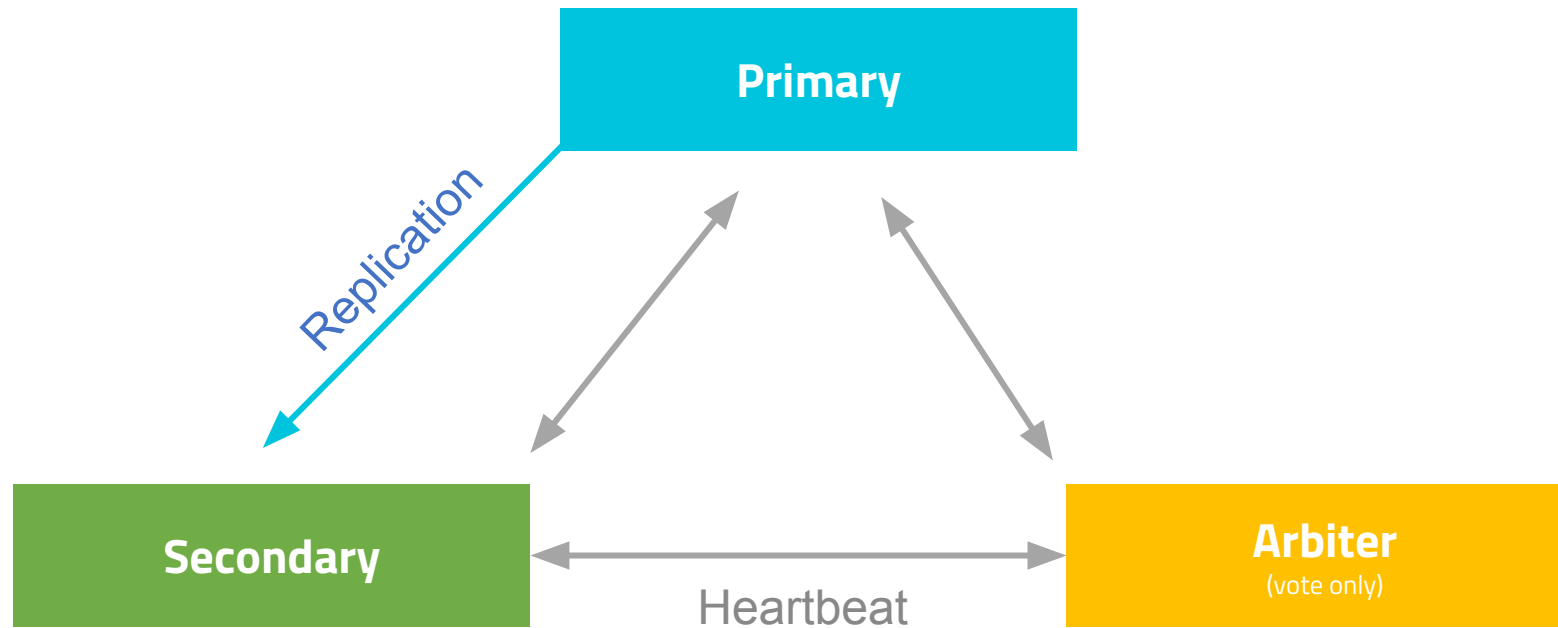
1. Introducción a Replica Set
2. Arquitectura y componentes de Replica Set
3. Oplog y journal
4. Read Preference y Write Concern
5. Despliegue y configuración de RS
6. Introducción a Sharding
7. Arquitectura y componentes de Sharding
8. Shard Key
9. Despliegue y configuración de sharding

La arquitectura que nos permite la replicación de los datos se conoce como **Replica Set**



Un despliegue en Replica Set nos permite la replicación de datos y la alta disponibilidad del servidor.

- La replicación funciona en modelo maestro-esclavo
- El failover es automático
- Todos los nodos guardan el mismo dataset
- El sistema es eventualmente consistente
- El nodo maestro es conocido como nodo Primario y los esclavos como Secundarios
- Las **escrituras** siempre se llevan a cabo contra el **nodo primario**
- Las **lecturas consistentes** deberán hacerse sobre el nodo primario
- El número de nodos en la arquitectura deberá ser **impar** para garantizar el correcto funcionamiento de la HA



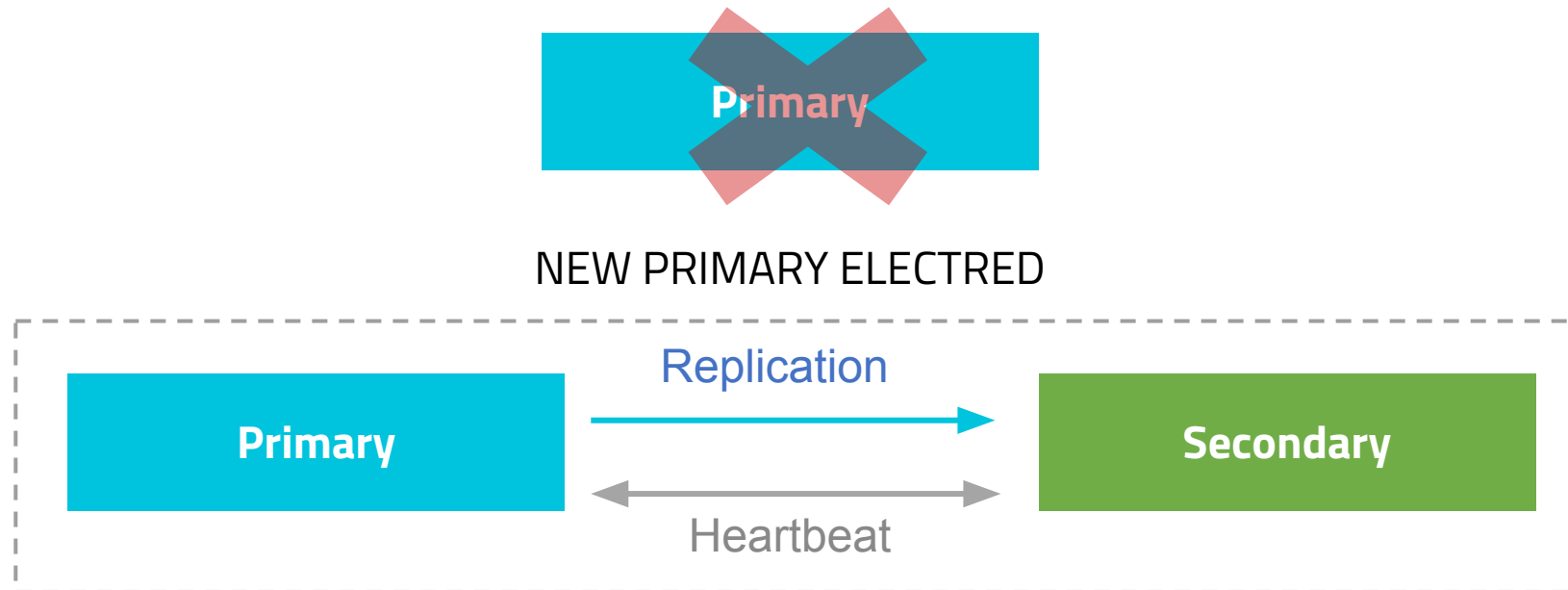
Un replica set está compuesto por al menos

Primario : Nodo que recibe todas las peticiones de escritura

Secundario: Nodo de réplica de datos

Y opcionalmente

Árbitro: nodo que no almacena copia de los datos pero sirve para el proceso de votación que promueve a primario alguno de los nodos secundarios en caso de empate. Un nodo árbitro deberá ser utilizado siempre que tengamos un **número par de nodos que almacenan datos**



Una pieza fundamental para que funcione la replicación de los datos es el oplog.

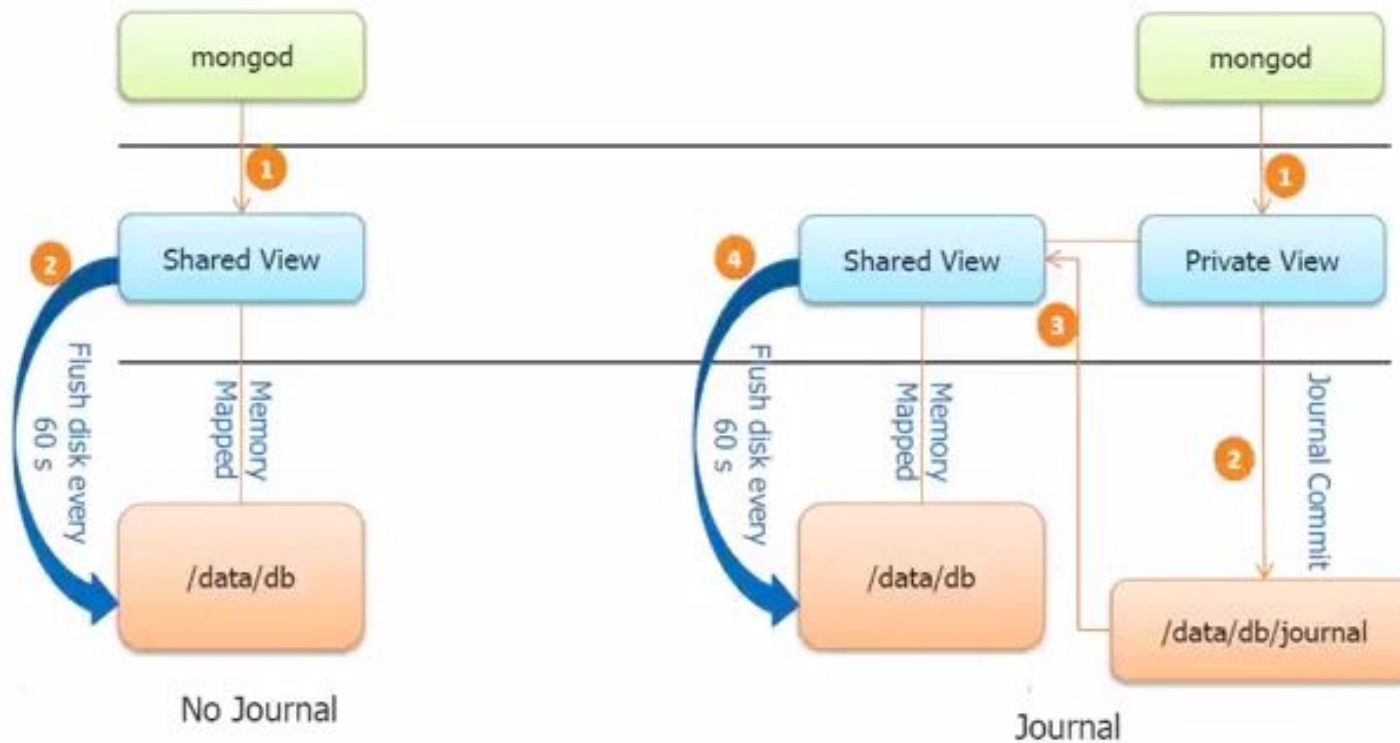
- Es una capped collection en la que se almacenan todas las operaciones de modificación de datos ejecutadas sobre el nodo primario.
- Los nodos secundarios copian de manera asíncrona la información almacenada en el oplog abriendo un cursor contra el nodo contra el que tenga menos latencia del replica set
- Las operaciones almacenadas en el el oplog cumplen el principio de **idempotencia**
- **El tamaño recomendado es al menos un 5 % del tamaño del volumen de datos**
- Si un nodo secundario cae, y se restaura en un corto periodo de tiempo, podrá replicar los datos utilizando las entradas del oplog. Si hay demasiadas operaciones por replicar, necesitará realizar una resincronn completa.izació

El mecanismo de journal nos asegura la integridad que los datos escritos en memoria se escriben en los datafiles en disco. El proceso difiere dependiendo del storage engine.

En MMAPV1 :

1. El dato de la operación de escritura se almacena en la vista privada de memoria.
2. Después aplica las operaciones en los ficheros de journal en batch como un grupo de commits cada 100 ms .
3. Tras aplicar el commit del journal, aplica los cambios sobre la vista compartida
4. Tras pasar los cambios a la vista compartida, aplica los cambios en los datafiles cada 60 segundos. Si el SO tiene “presión” en el uso de memoria, el flush a disco puede ser más frecuente.

Journaling Mechanics



El Journal de Wired Tiger se llama write-ahead-log. Funciona con un sistema de “checkpoints” por cada transacción. El funcionamiento del journaling es el siguiente:

1. Cuando llega una escritura, se almacena en un buffer en memoria las operaciones. (log-ahead)
2. Cuando el buffer se llena(100 MB), se copia a disco al fichero de journal (si no se llena, WiredTiger se encarga de vaciarlo varias veces por segundo)

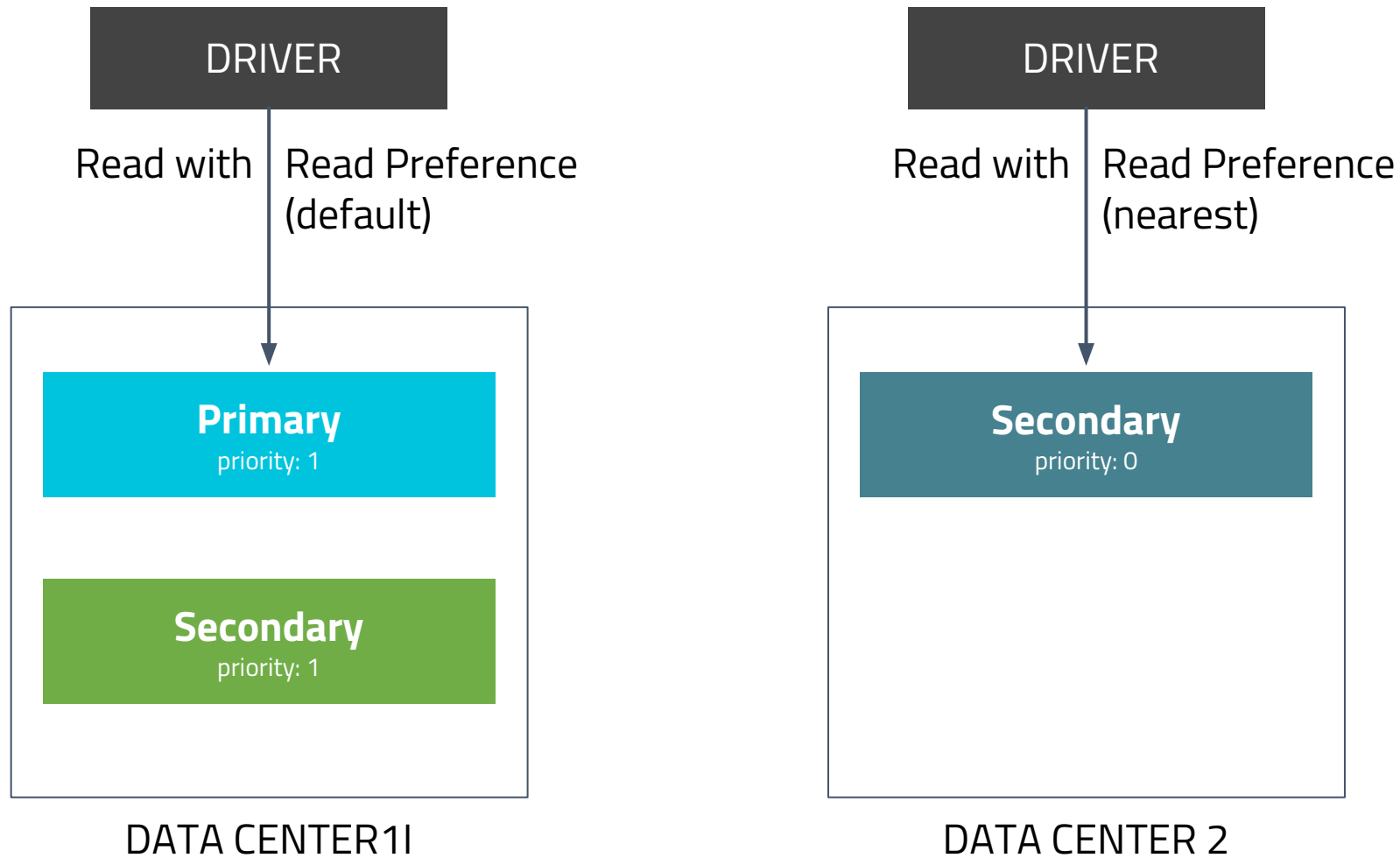
Después WiredTiger lleva los cambios a los datafiles bajo las siguientes condiciones:

1. Cuando el journal alcanza un tamaño de 2 GB o cada 60 segundos, lo que suceda antes
2. Como utiliza ficheros de log-ahead de 100 MB, cada vez que un fichero se llena, reserva otro nuevo fichero de 100 MB y sincroniza a disco el anterior
3. Si hay Write Concern a journal, Wired Tiger fuerza la escritura a disco del log-ahead

Los datos en un replica set son eventualmente consistentes y nuestra aplicación puede ser más o menos permisiva con la lectura de datos no frescos.

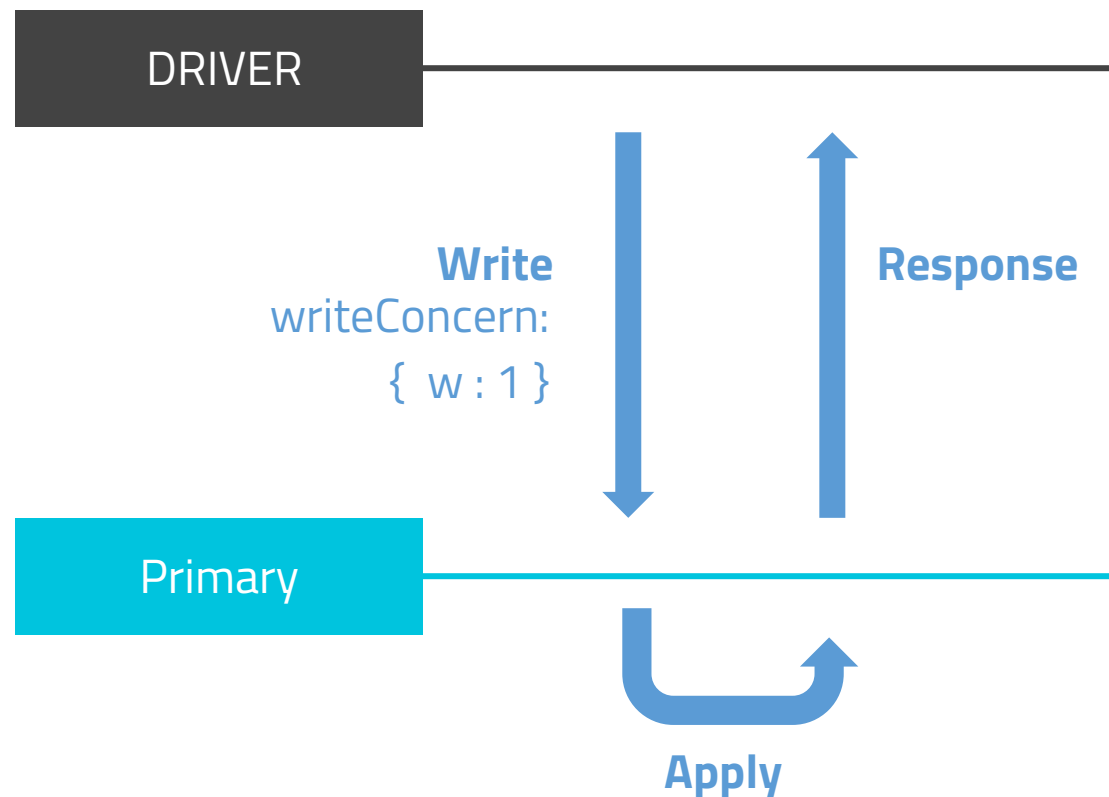
Para ello podemos configurar nuestra aplicación para indicarle sobre que nodos del replica set leer datos (read preference):

- **Primary** : (por defecto) Los datos se leen siempre de un nodo primario
- **Primary preferred** : Los datos preferentemente serán leídos de un primario, y en caso de no ser alcanzable o disponible, se leerá de un nodo secundario.
- **Secondary** : Los datos se leen siempre de un nodo secundario
- **Secondary preferred** : Los datos preferentemente serán leídos de un secundario, y en caso de no ser alcanzable o disponible, se leerá de un nodo primario.
- **Nearest** : Los datos se leen del nodo del réplica set con menor latencia sin importar el rol dentro del replica set

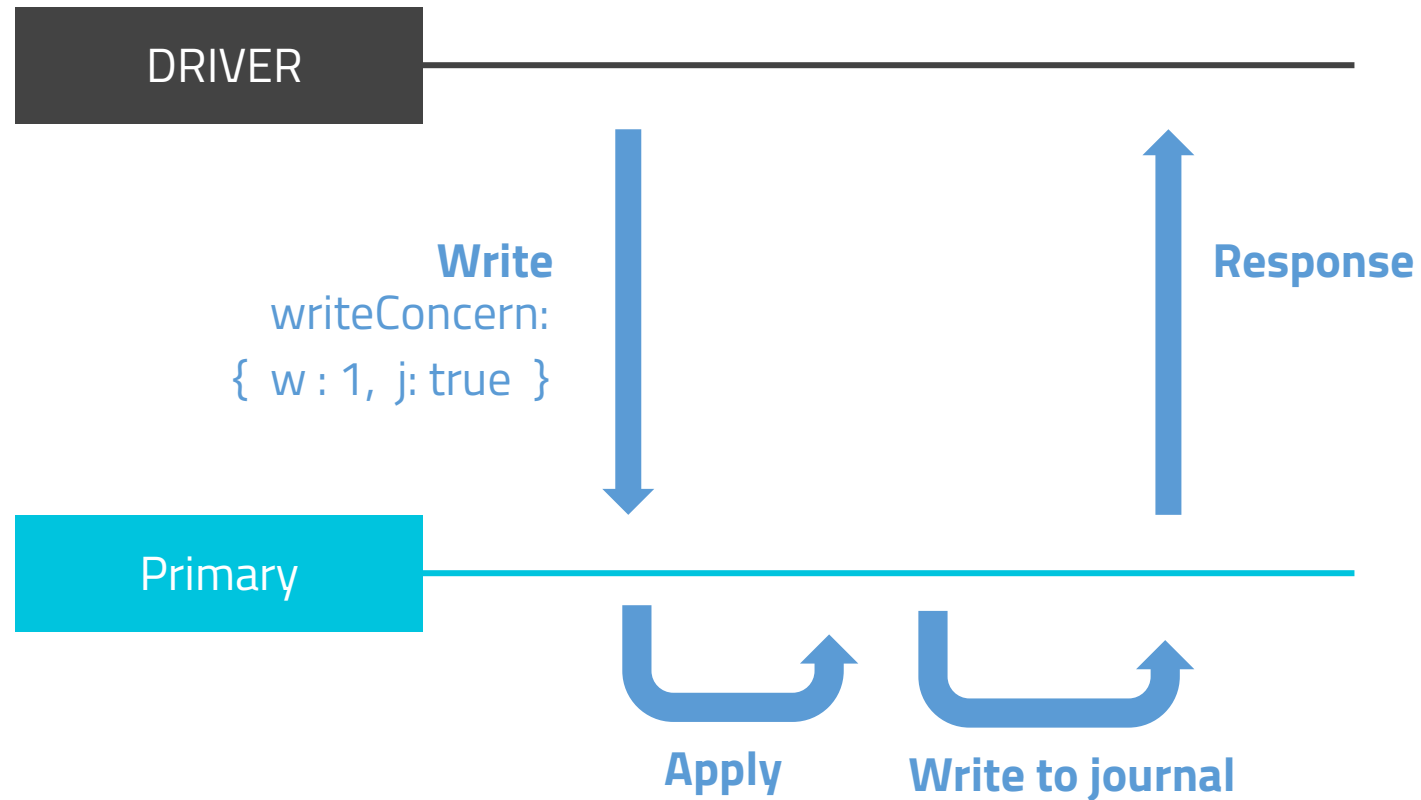


Al igual que con la lectura, dependiendo de la seguridad que queremos dar de que el dato ha sido escrito, podemos configurar diferentes niveles de acknowledge de escritura del dato.

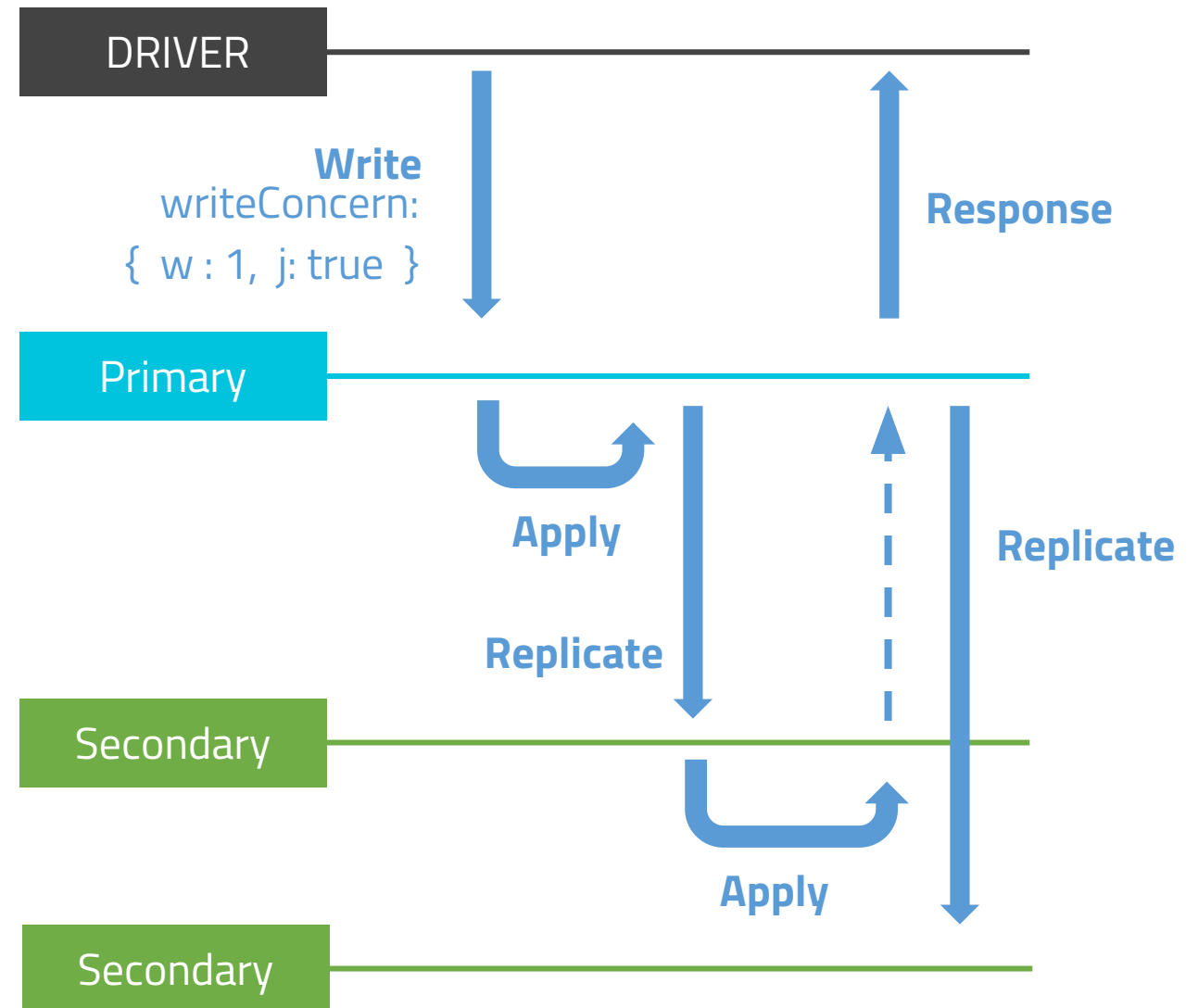
Por defecto, el dato se da por escrito cuando se encuentra aplicado sobre el nodo primario



El modo journaled, el dato se da por escrito cuando se encuentra escrito en el journal del nodo primario



Por último, podemos especificar el número de nodos en los que debe estar replicado el dato para considerarlo como escrito en el replica set



Para configurar en replica set añadimos en el fichero de configuración la siguiente opción:

replication:

 replSetName: <nombre_replica_set>

Si además queremos autenticación:

security:

 keyFile : <Ruta_key_file>

El nombre del Replica set tiene que coincidir en el fichero de configuración en todos los nodos

Levantamos nuestro proceso Mongo en cada una de las máquinas que van a conformar el replica set con la configuración apropiada. Nos conectamos a uno de los nodos del replica set y ejecutamos el siguiente comando tantas veces como máquinas conformen nuestro RS:

```
rs.add("<hostname>:<puerto>")
```

Por ejemplo:

```
rs.add("mongolab1:27017")
```

Si queremos añadir un nodo árbitro

```
rs.addArb("<hostname>:<puerto>")
```

Si utilizamos el material del repositorio ~/RedHat-Mongodb/Tema5/Replicaset/ veremos los siguientes scripts:

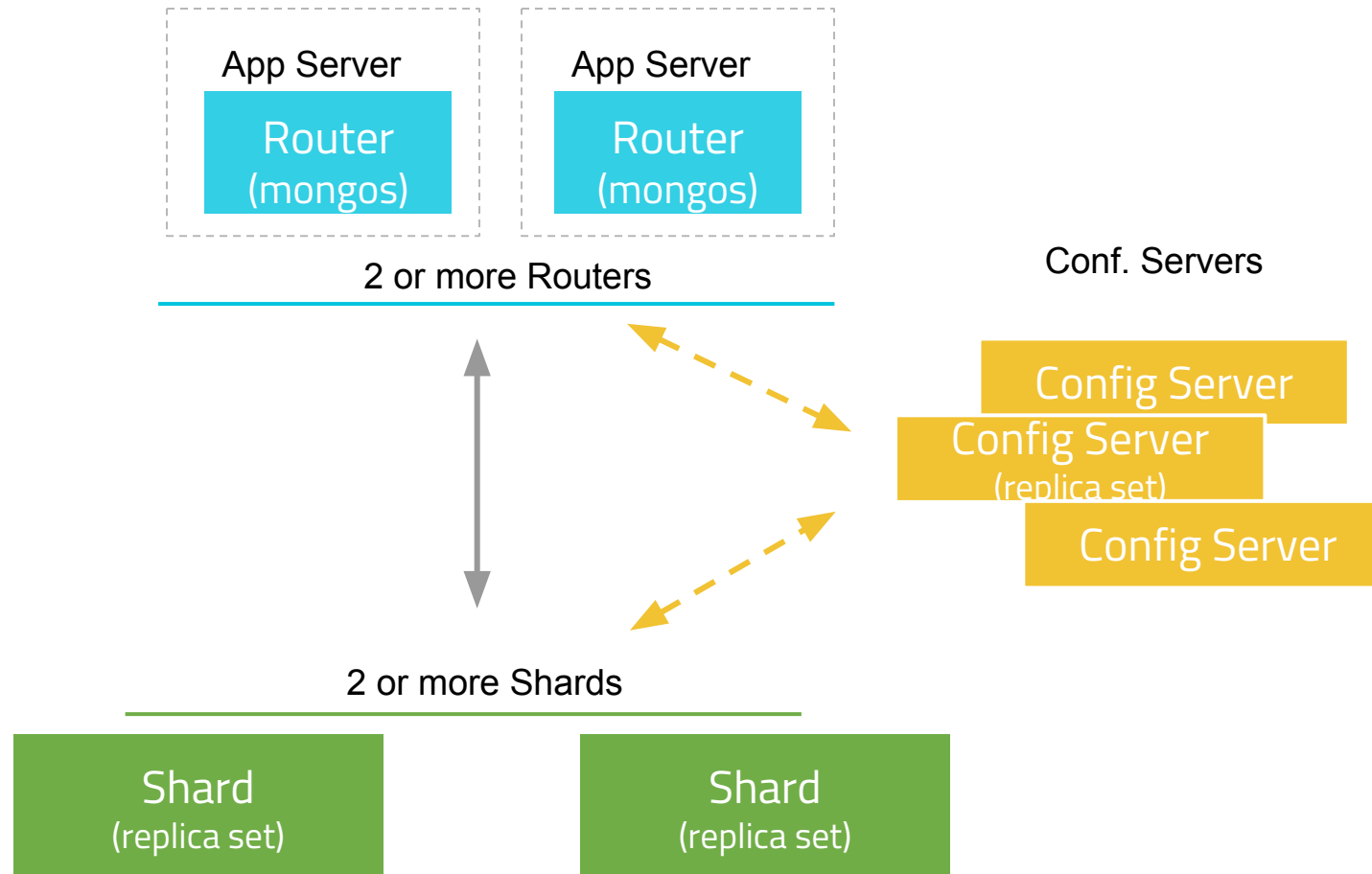
startRS.sh : inicializa un replica set de 3 nodos y puebla la bbdd miBD con datos

stopEnv.sh : para las instancias del Replica Set

cleanEnv.sh : Para las instancias y elimina la información almacenada.

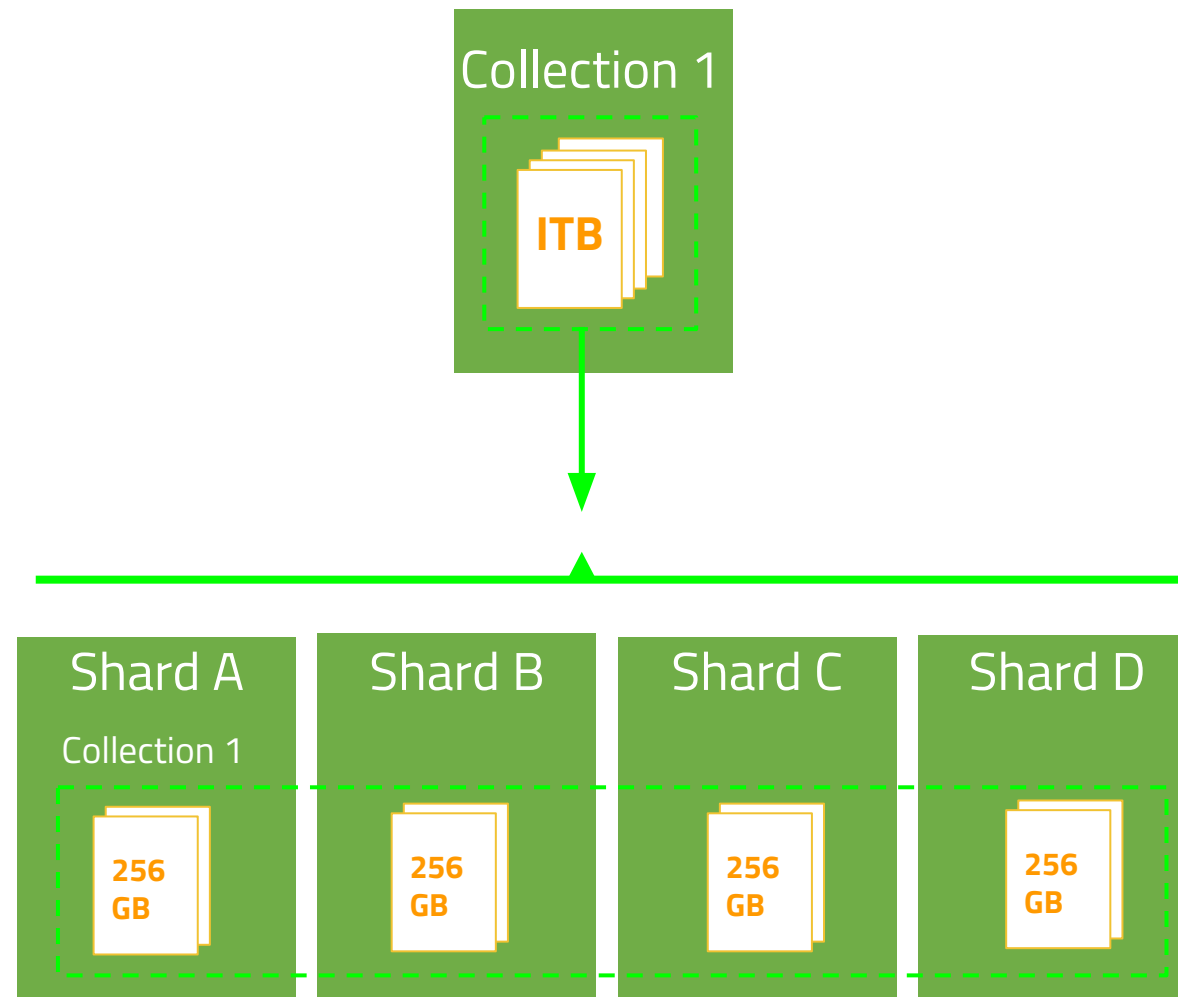
PRACTIQUEMOS

La arquitectura que nos permite el escalado horizontal, el particionamiento de los datos y la distribución de la capacidad de procesamiento y almacenamiento se conoce como Sharding



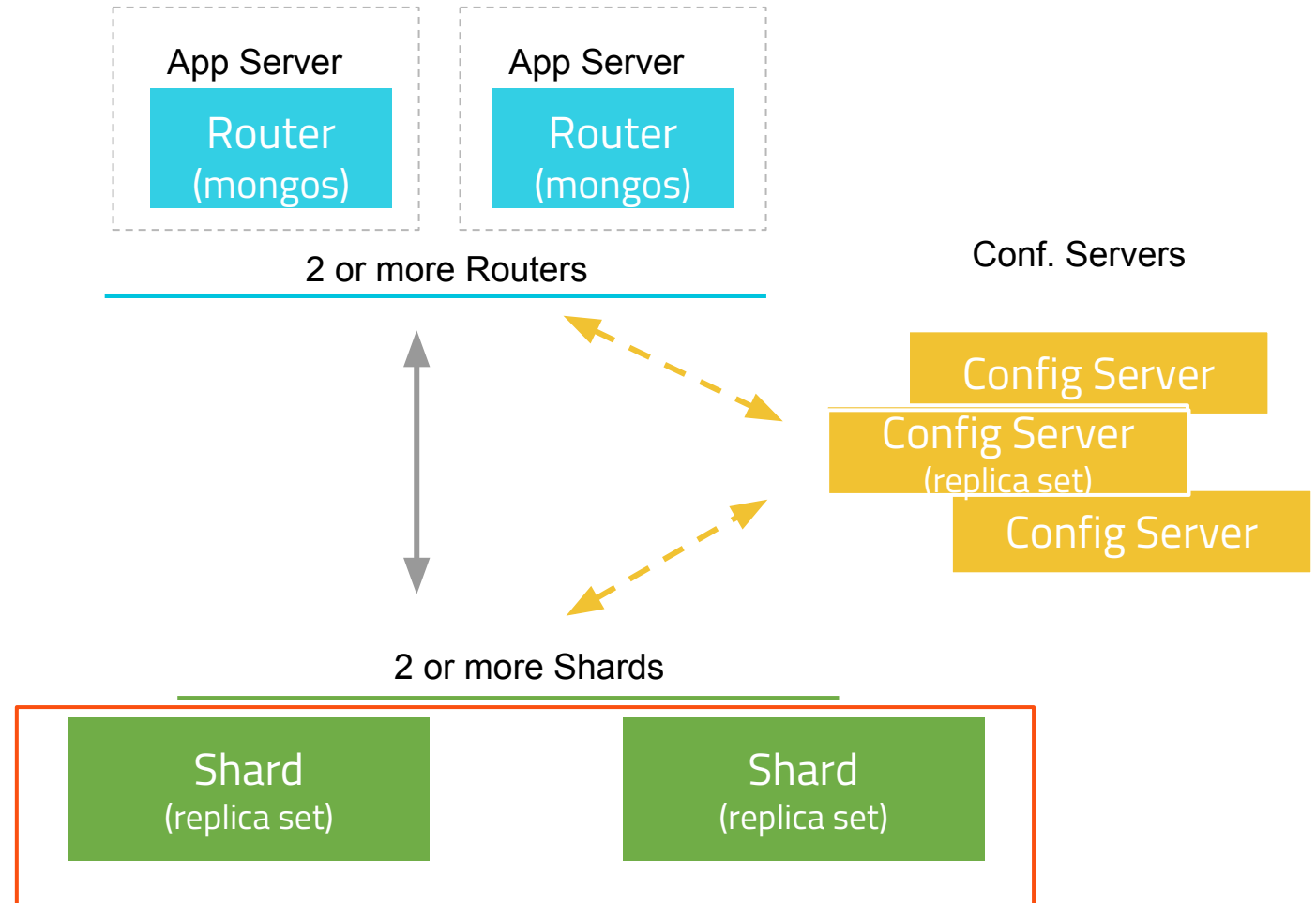
Un despliegue en sharding nos permite dividir el almacenamiento de los datos entre distintas máquinas. Este mecanismo nos facilita el **escalado horizontal**

- El dividir la información entre distintas máquinas, nos permite también dividir las tareas de procesamiento. Cuanto más crezca el clúster, la capacidad de procesamiento aumenta al dividir entre más shards la información.
- Es muy útil también cuando el tamaño de dataset es tan grande **que supera el tamaño de la memoria física**
- El particionado se realiza a nivel de **colección**. Tiene gran importancia la **clave de particionamiento**
- Los datos son particionados en intervalos cerrados por la izquierda y abiertos por la derecha llamados **chunks**.



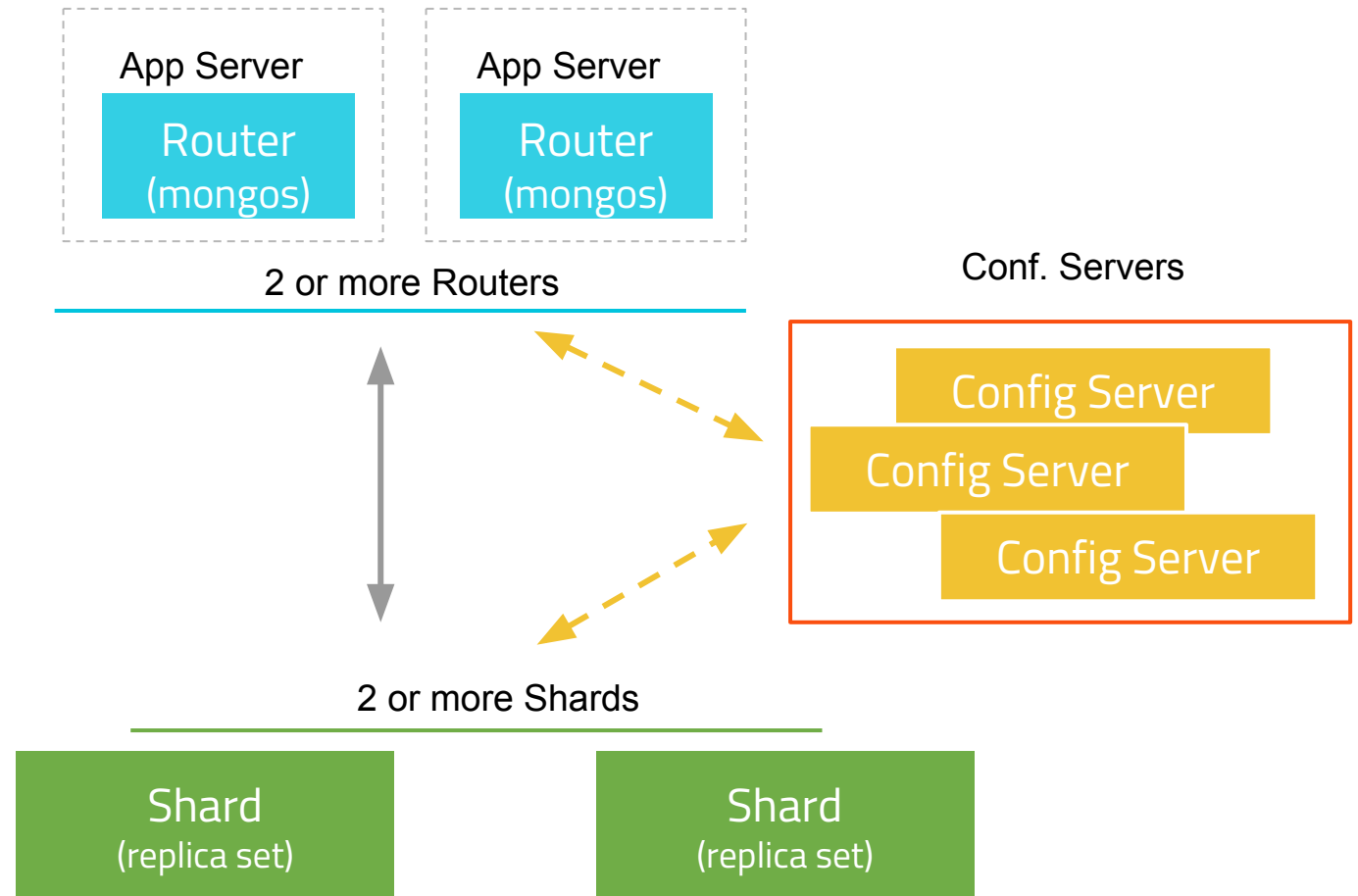
Shards:

- Son los nodos que almacenan los **datos**
- En entornos productivos, **cada shard** está compuesto por **un replica set**
- Por cada BBDD del entorno, existe un shard principal que almacena **el resto de colecciones no particionadas**
- Los shards no pueden atacarse directamente salvo por **labores de mantenimiento**



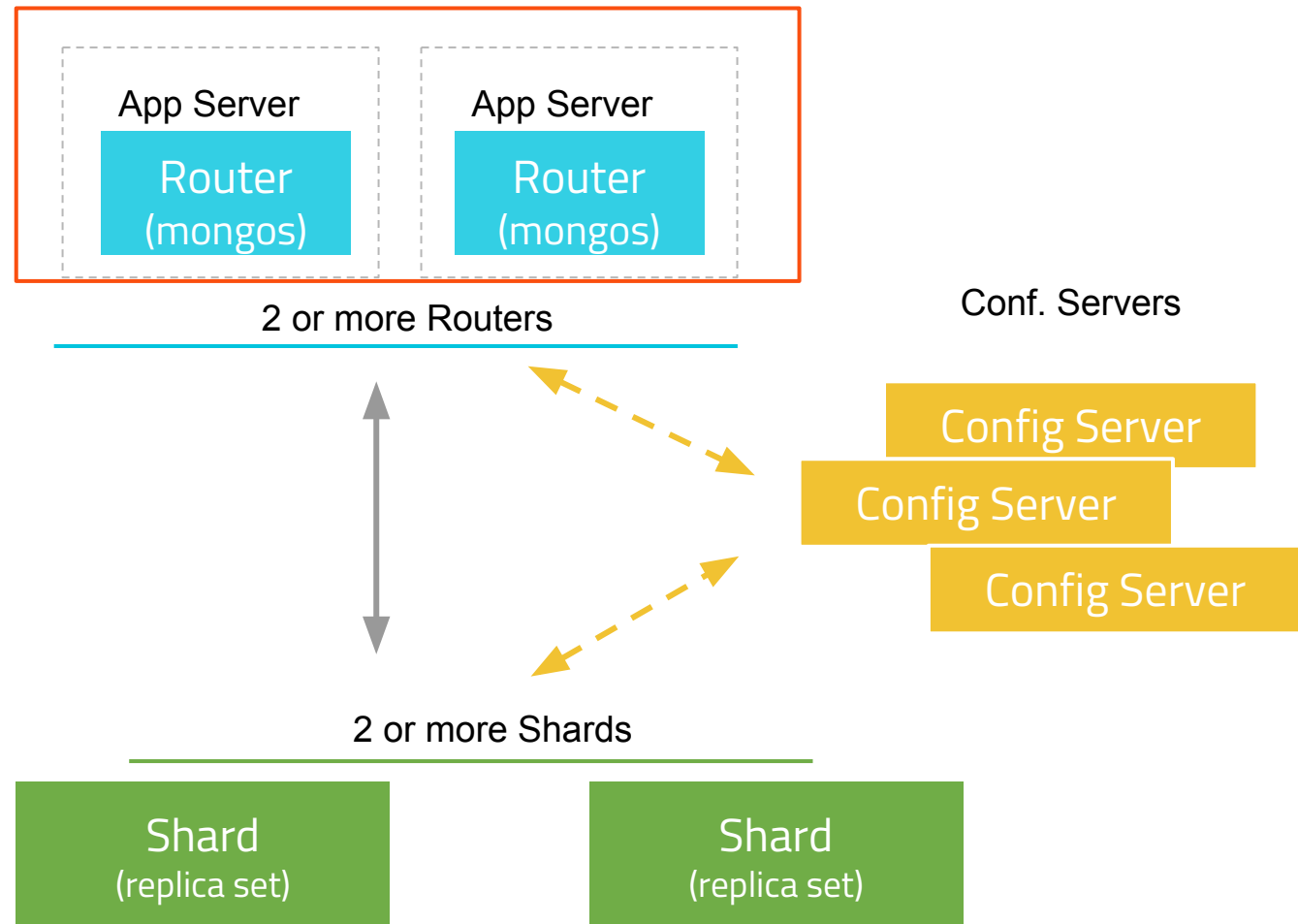
Config servers:

- Almacenan **metadatos de las colecciones particionadas y los chunks**
- Se despliegan como un replica set



Mongos:

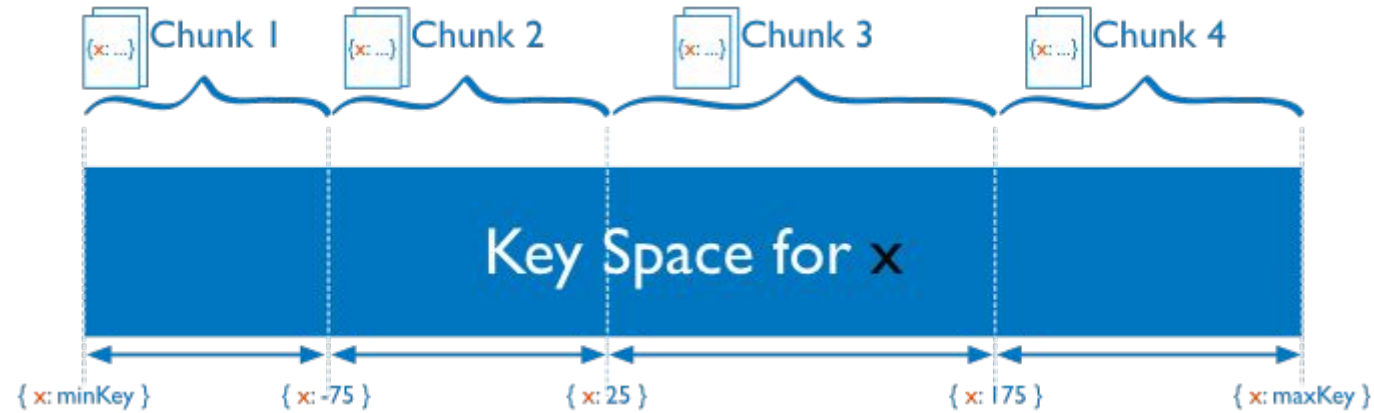
- Son los nodos **enrutadores de peticiones** de las aplicaciones.
- Consultan los config server periódicamente para cachear la información de los metadatos de particionamiento de los datos
- Son procesos muy ligeros que pueden residir en la misma máquina **que las aplicaciones**



Para particionar una colección es necesaria una **shard key**. Es un campo que puede ser un **índice simple o compuesto que debe existir en todos los documentos de la colección particionada**.

- MongoDB se encarga de dividir la shard key en valores en **chunks** y distribuir los chunks a en los distintos **shards** del clúster.
- La shard key debe ser utilizada en la gran mayoría (si no en todas) las queries sobre la colección particionada
- La estrategia de partición de los chunks puede ser por **rangos, hash key o tags**
- **Una vez elegida una shard key, esta no puede ser modificada.**
- La shard key debe cumplir con algunas de estas propiedades para que sea apropiada (pero es imposible cumplir todas):
 - Alta cardinalidad
 - Alto grado de aleatoriedad (dispersión)
 - Operaciones sobre un solo shard
 - Usar un índice compuesto
 - No puede ser un array
 - No debe ser monótonamente creciente

Shard key basada en rango

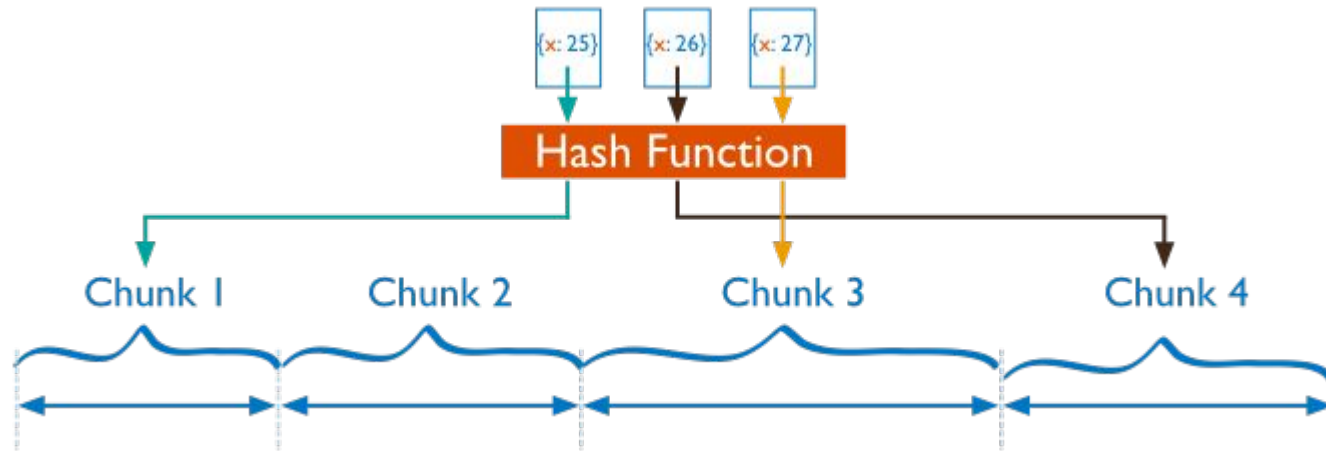


Búsquedas por rango más eficientes al estar dentro de un mismo shard



Peligro de existir chunks descompensados o indivisibles

Shard key basada en hash

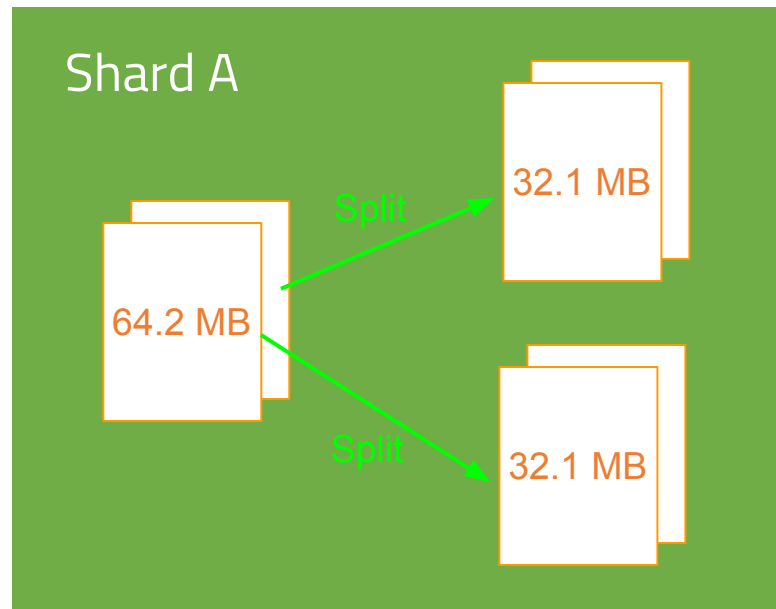


Distribución muy balanceada de los chunks, muy divisible y gran cardinalidad.

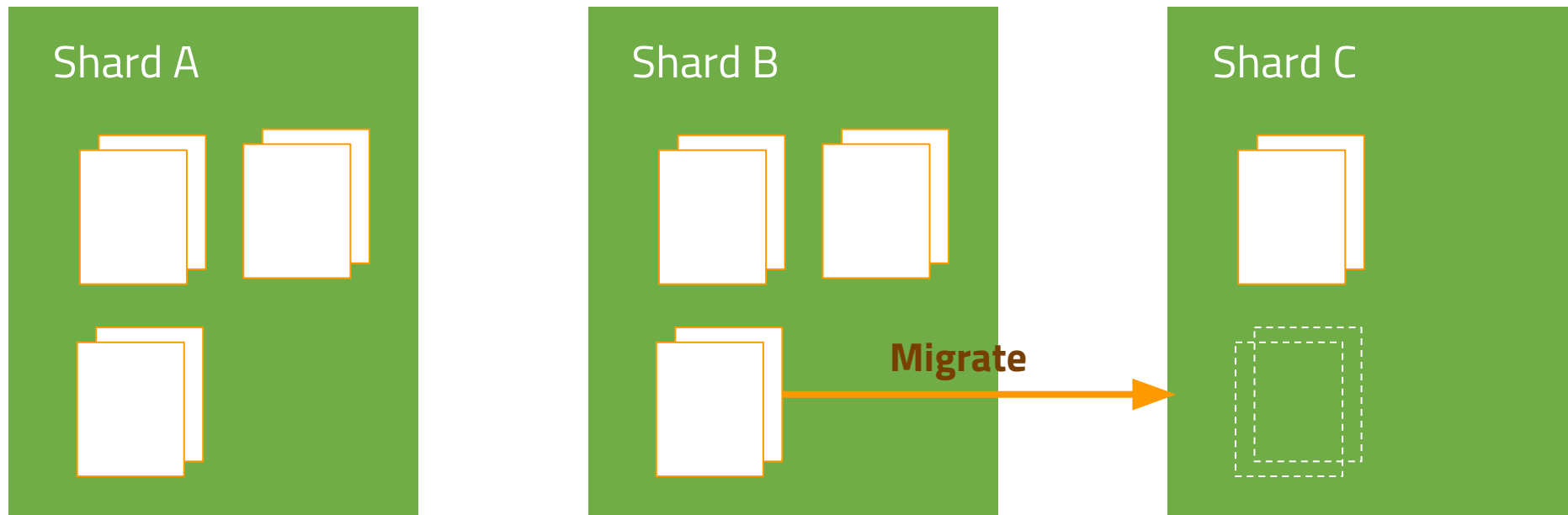


Imposibilidad de realizar queries sobre rango en un único shard

Splitting



Balanceo



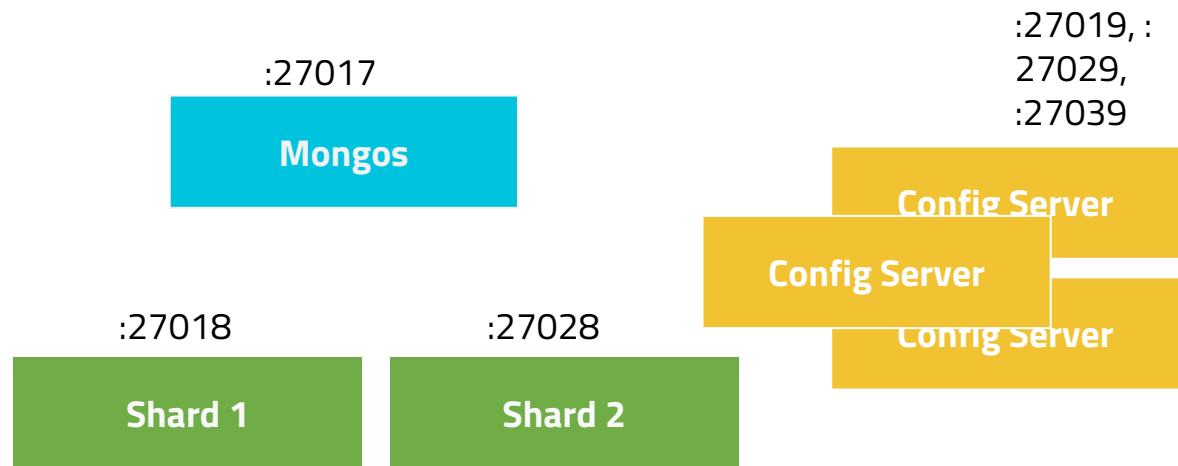
- Las operaciones de lectura/escritura (actualizaciones) si **NO** tienen la shard key **pueden ser muy ineficientes.**
- En las operaciones de agregación:
 - Se va a intentar siempre que las operaciones se realicen sobre cada shard, por ejemplo:
 - match sobre shard key
 - projections
 - sort sobre shard key
 - Si no se puede hacer así, se escoge un shard y la agregación se realiza en esa máquina: **exige mucho movimiento de información**
- **No utilizar la shard key implicar ejecutar la operación sobre cada nodo y luego unificar el resultado.**

El despliegue habitualmente se hace en el siguiente orden:

1. Config Servers
2. Mongos
3. Shards

Puede utilizarse el script del repositorio del curso

Arquitectura del laboratorio:



Config Server:

Creemos un directorio para almacenar data files (uno por instancia) :

```
mkdir -p /data/configdb1
mkdir -p /data/configdb2
mkdir -p /data/configdb3
```

Y levantamos 3 instancias:

```
mongod --configsvr --replSet config
--dbpath /data/configdb1 --port 27019 --logpath /data/configdb1/mongod.log &
mongod --configsvr --replSet config
--dbpath /data/configdb2 --port 27029 --logpath /data/configdb2/mongod.log &
mongod --configsvr --replSet config
--dbpath /data/configdb3 --port 27039 --logpath /data/configdb3/mongod.log &
```

Mongos:

Ejecutamos el comando siguiente (en la máquina de laboratorio):

```
mongos --configdb config/mongodb:27019,mongodb:27029,mongodb:27039 &
```

Shards:

Creamos los directorios de datos:

```
mkdir -p /data/shard1  
mkdir -p /data/shard2
```

Y levantamos el proceso mongod con las opciones para sharding, dbpath y puertos

```
mongod --shardsvr --replSet RS1 --dbpath /data/shard1 --port 27018 --  
noprealloc --nojournal &  
mongod --shardsvr --replSet RS2 --dbpath /data/shard2 --port 27028 --  
noprealloc --nojournal &
```

Shards:

Inicializamos el replica set en cada shard. Nos conectamos a cada instancia de los shards:

```
mongo --port 27018
```

```
mongo --port 27028
```

Y ejecutamos el comando

```
rs.initiate()
```

Mongos:

Por último, nos conectamos al mongos para añadir los shards:

```
sh.addShard("RS1/mongodb:27018")  
sh.addShard("RS2/mongodb:27028")
```

Nos posicionamos sobre la BBDD mishard, creamos un índice en la colección partición, y particionamos la BBDD:

```
use mishard  
db.particion.createIndex({x:1})  
sh.enableSharding("mishard")  
sh.shardCollection("mishard.particion",{x:1})  
sh.status()
```

Usando repositorio y máquina de laboratorio

startSharding.sh

Inicializa un shard con 3 config servers 2 shards y un mongo y puebla la BBDD mishard con 1M de registros

cleanEnv.sh

Para todas las instancias y borra todos los datos para limpiar el entorno

stopEnv.sh

Para los config servers, los shards y el mongos

6 Recomendaciones generales

Índice

1. Best Practices en Producción
2. Operativas de mantenimiento en RS y Sharding
3. Backups
4. Autenticación
5. Arquitecturas de RS
6. Arquitecturas Sharding

Existe un decálogo de buenas prácticas recomendadas por MongoDB para asegurar el correcto funcionamiento del servidor de BBDD que pueden consultarse en la documentación:

<https://docs.mongodb.org/manual/administration/production-notes/>



Orientadas al software

- Utilizar los paquetes para los SS00 ofrecidos por la web y la versión más reciente y estable
- Usar siempre SS00 de 64 bits
- Usar **siempre** journaling
- Utilizar Write Concern para asegurar la persistencia del dato
- Controlar el pool de conexiones contra la BBDD
- Asegurar la BBDD con usuario y password
- Utilizar firewalls para limitar únicamente conexiones legítimas

Orientadas al hardware y S.O.

MMAPv1:

- Incrementar la cantidad de Memoria Física más que CPU.

Wired Tiger

- El throughput aumenta cuanto mayor sea el número de CPU's y disminuye si el número de operaciones activas sobre la BBDD supera el número de CPU's

Para ambos S.E.

- Usa discos de estado sólido en la medida de lo posible
- Asigna siempre espacio de Swap (Linux)
- Utiliza RAID 10
- No usar NFS
- Separa en distintos filesystems los logs, datos y journal (cuidado con backup por snapshot LVM)
- En entornos virtuales deshabilitar ballooning
- Deshabilitar transparent huge transparent pages

Orientadas a Arquitectura

Replica sets:

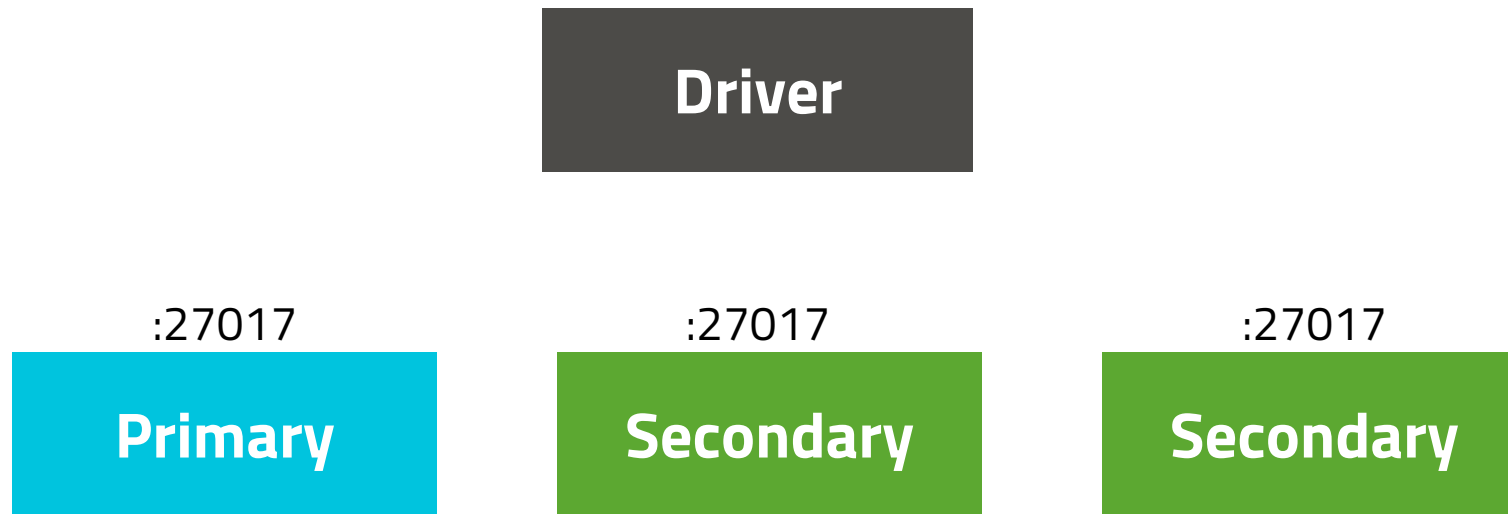
- Usar SIEMPRE número impar de nodos
- Utilizar nodos Hidden o Delayed para backups o data analytics
- Distribuir lecturas sobre secundarios
- Distribuye los nodos de réplica geográficamente
- Usa Journaling

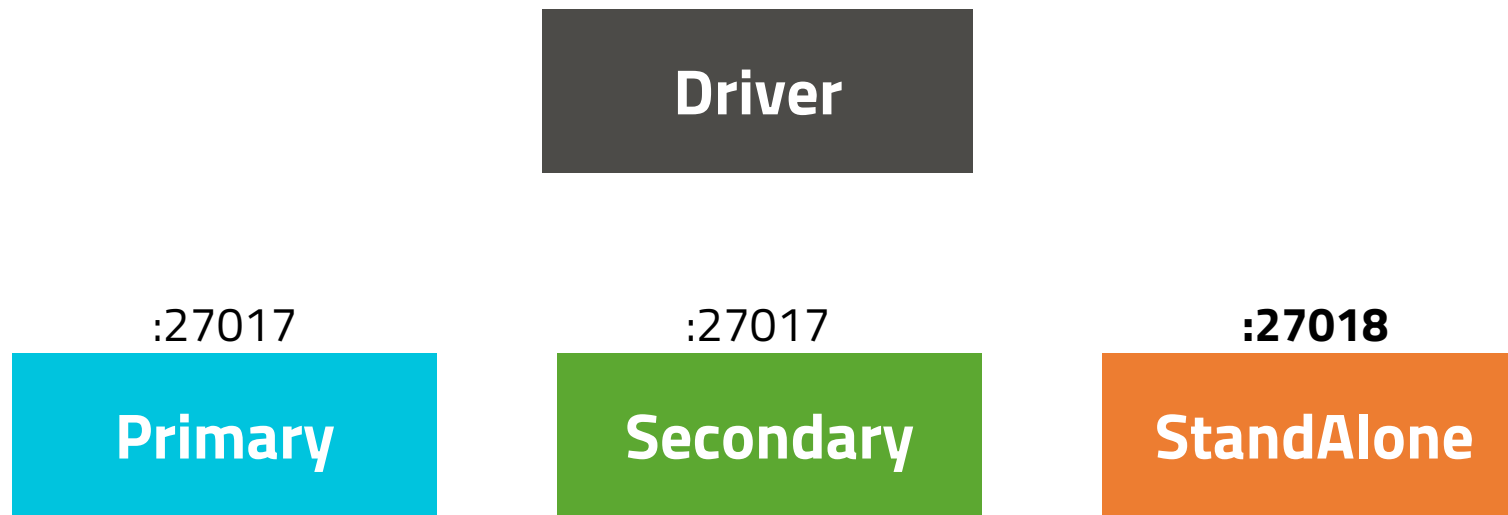
Sharding

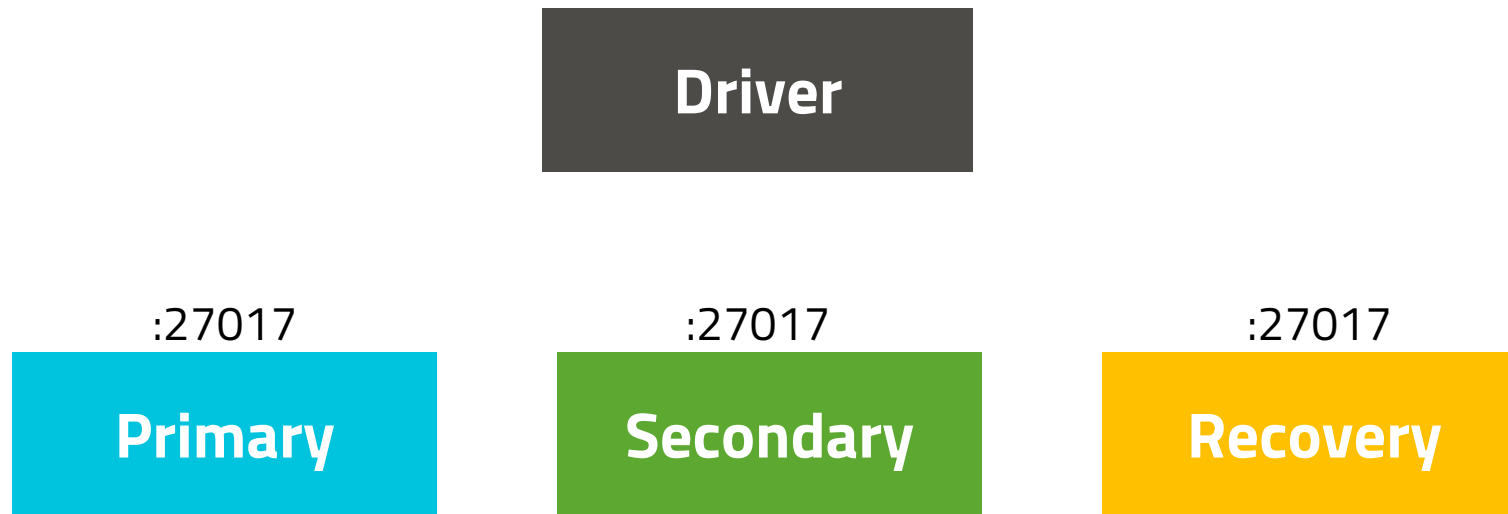
- Usa **siempre** 3 config servers en diferentes máquinas separadas
- Despliega al menos 2 shards en **Replica Set**
- Despliega un **mongos** por cada app server

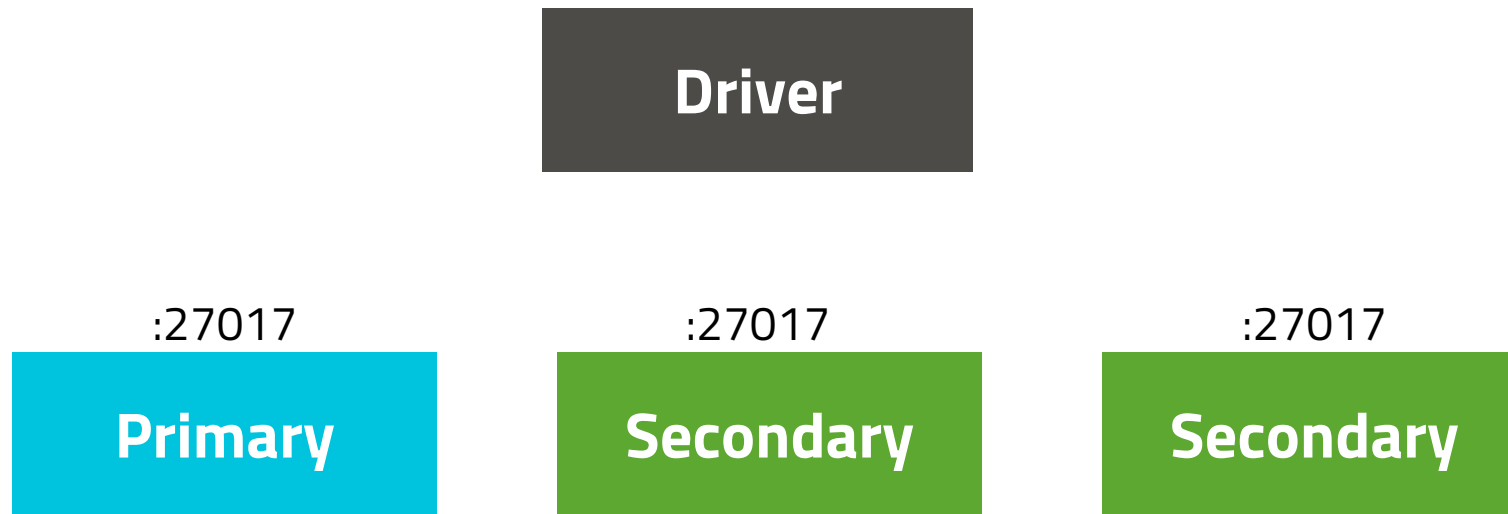
- Creación de índices sin bloqueo
- Escalado vertical de máquina
- Compactación de datos

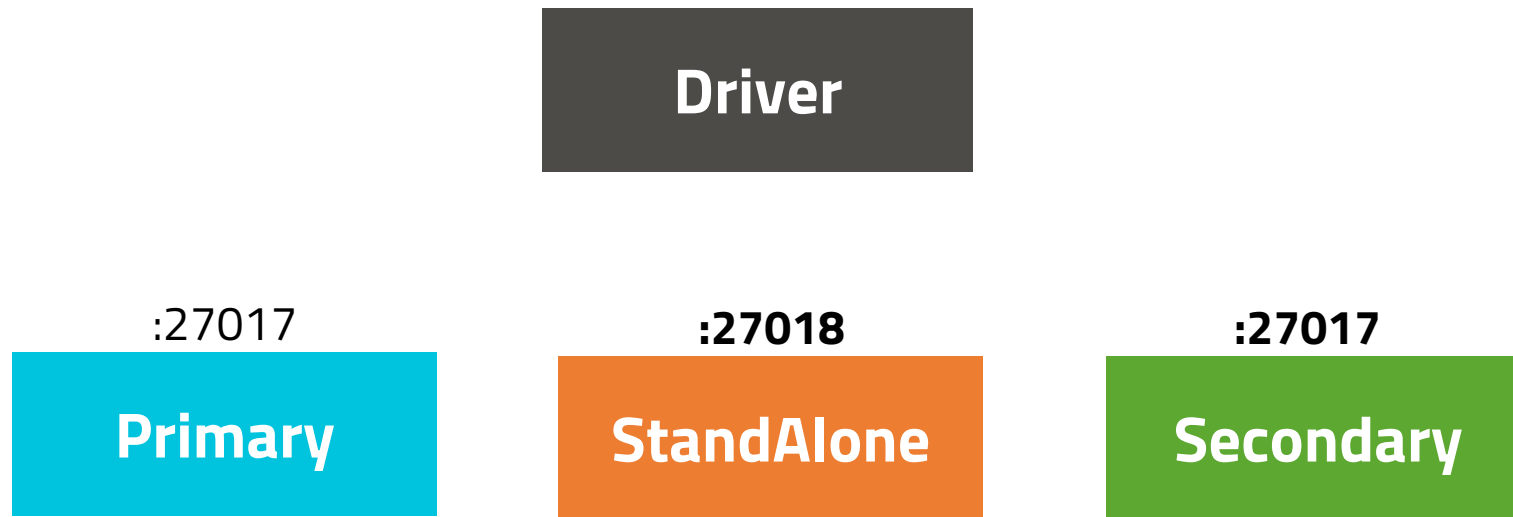
0 Downtime



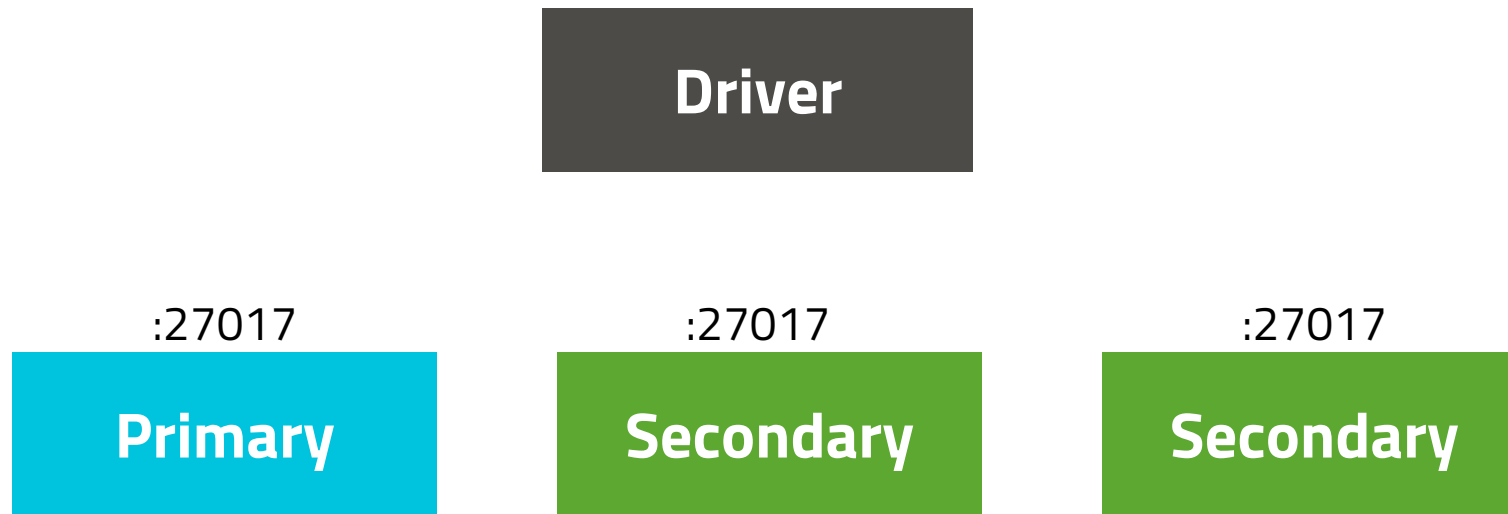


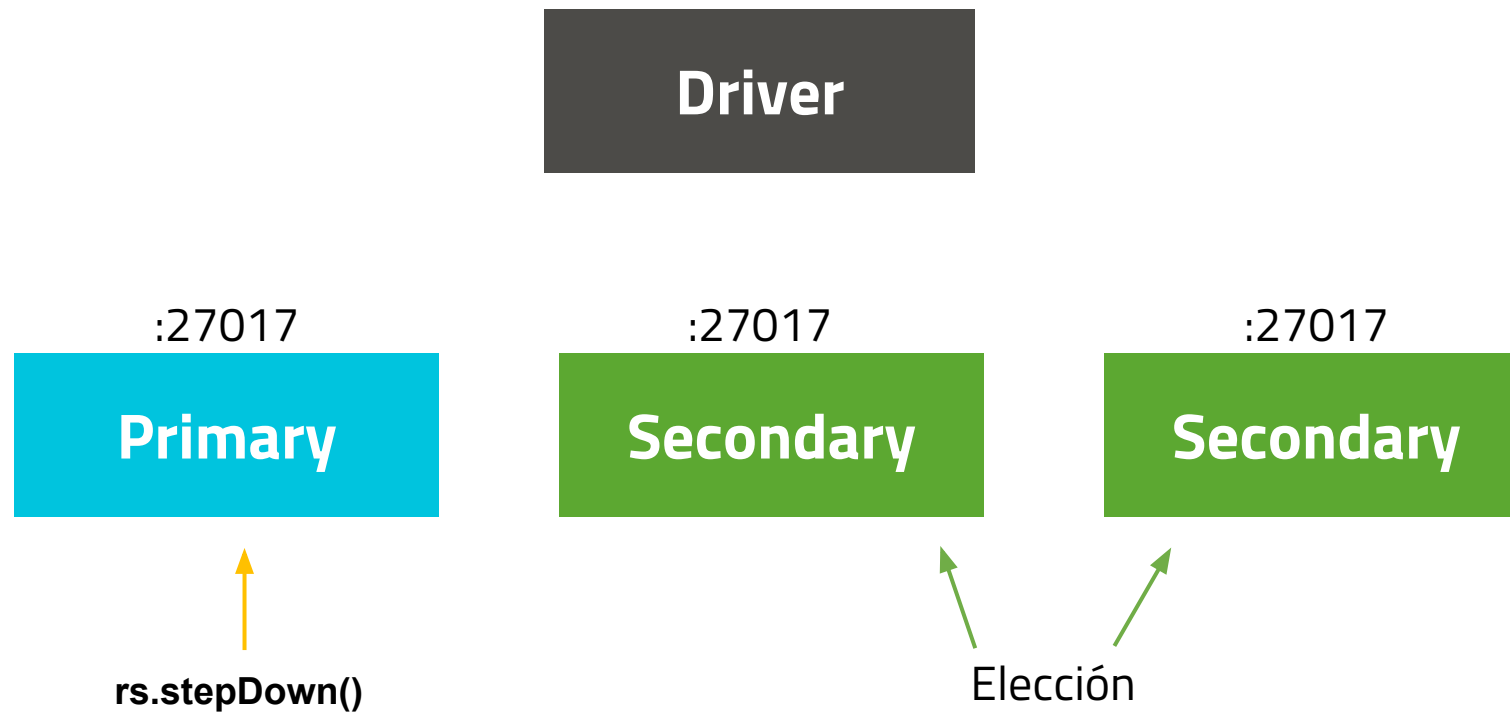


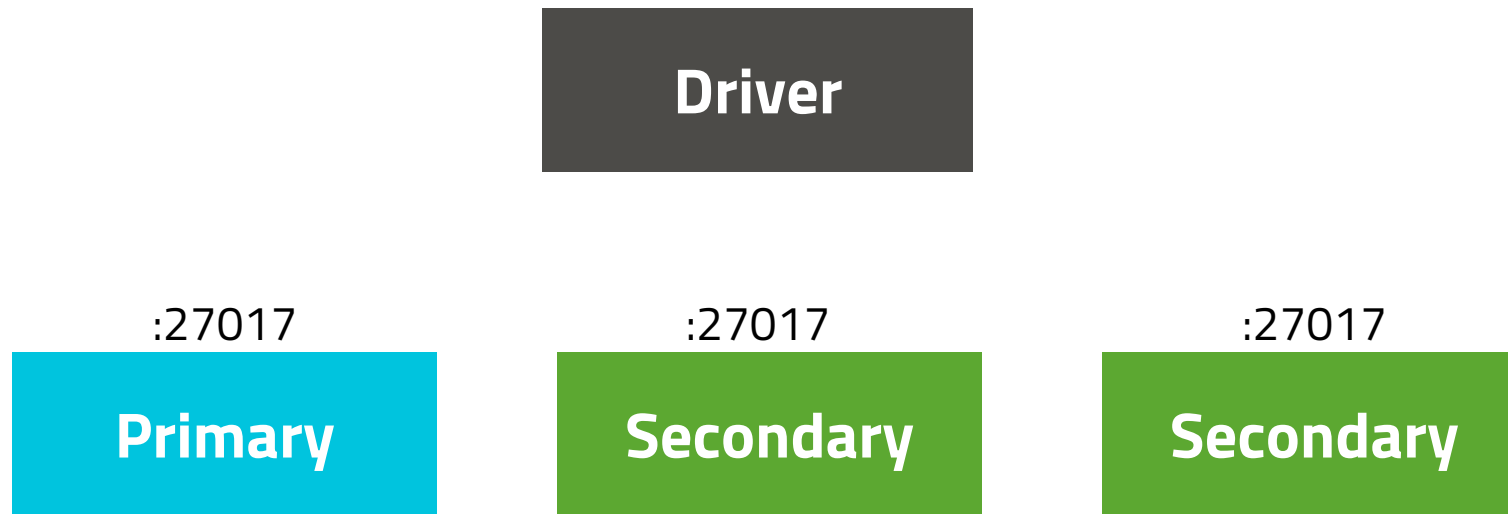


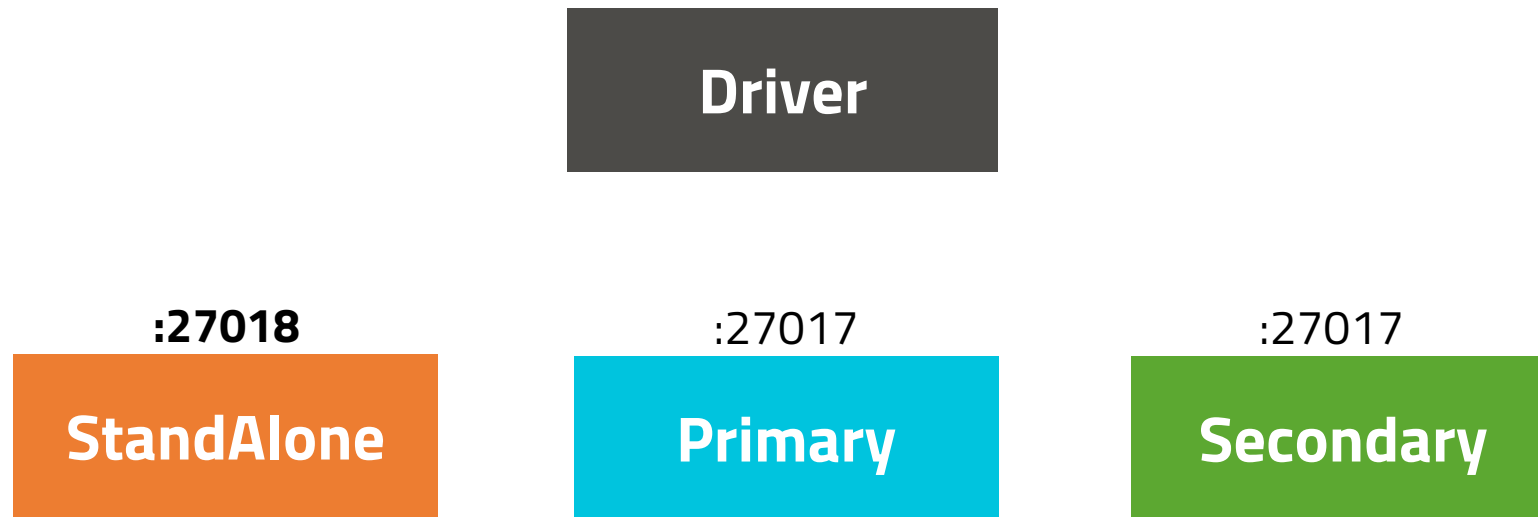


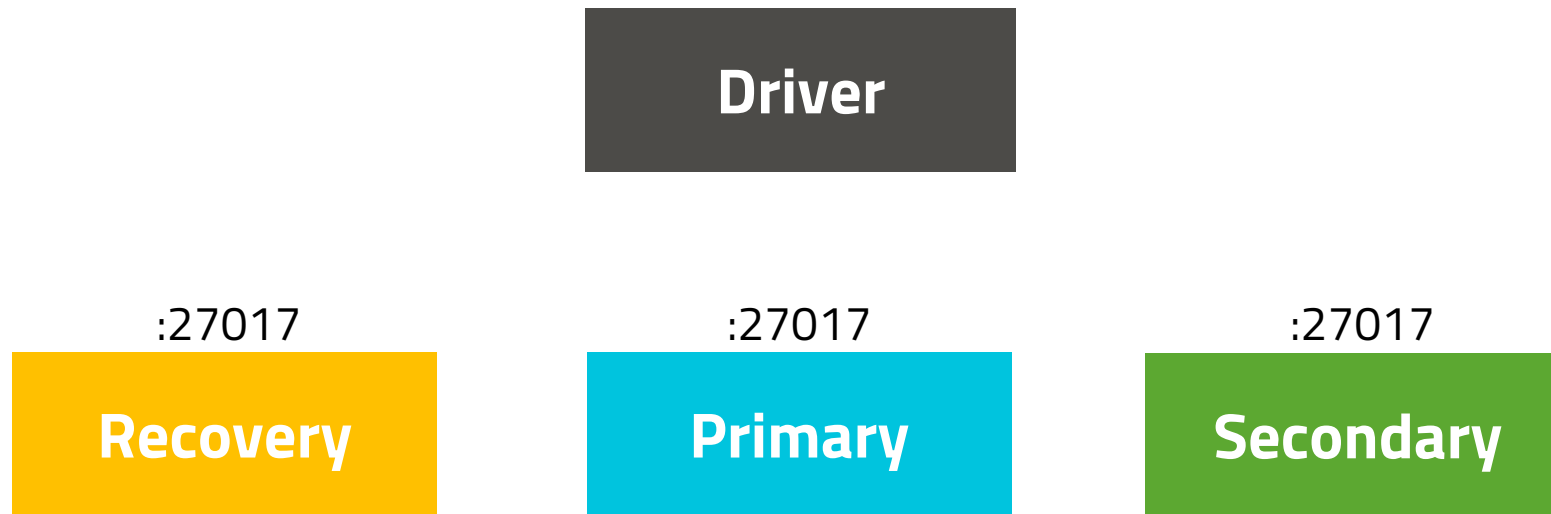


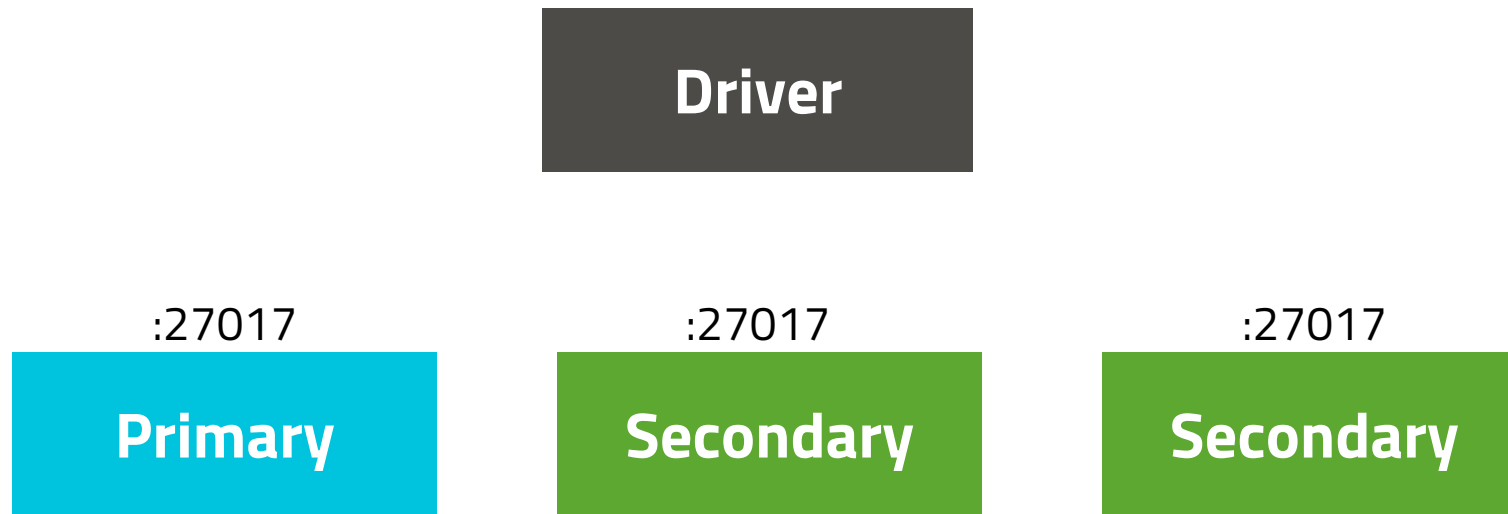












Existen distintas estrategias para realizar backup sobre una instalación de MongoDB

- Copia de datafiles
 - exige la parada de un nodo para que sea consistente
- mongodump
 - Puede realizarse en caliente y en el punto en el tiempo exacto en Replica Sets y Shards
 - Para Shards exige parar el balancer de chunks
 - Es recomendable usar un nodo específico que no sea primario para labor de backup
- Snapshots de filesystems
 - Muy rápido
 - Exige que el journal y datafiles en el mismo filesystem
- Operations Manager (Onpremise o Cloud)
 - De pago o licenciado
 - Gestión de backups y monitorización completa
 - Consola unificada de gestión de backup y monitorización

- Es altamente recomendable establecer algún mecanismo de autenticación de usuarios para el acceso la BBDD.
- Para ello en el fichero de configuración deberá habilitarse el tipo de autenticación necesitada, generar el keyfile o certificado y referenciarlo en el fichero de configuración.
- Elegir el método de autenticación deseado:
 - Usuario y password
 - Certificados
 - LDAP (Enterprise)
 - Directorio Activo (Enterprise)
 - Kerberos(Enterprise)

Vamos a autenticar nuestra BBDD

1. Accedemos al directorio de donde se encuentra el material del ejercicio

```
cd ~/RedHat-Mongodb/Tema6/autenticación
```

2. Observamos el contenido del fichero de configuración (fijándonos en el apartado security)

3. ejecutando el comando:

```
cat mongod.conf
```

Y levantamos el proceso con dicho fichero de configuración : **mongod -f mongod.conf**

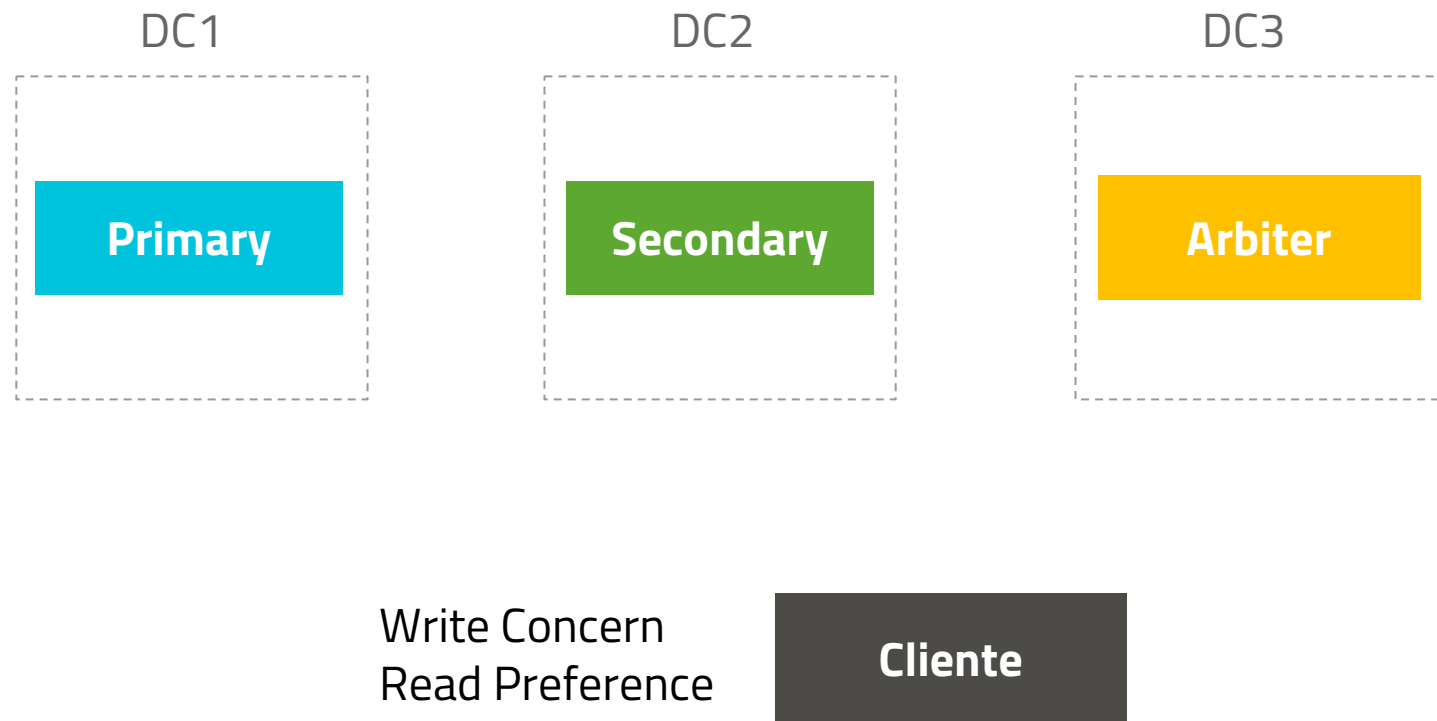
4. Nos conectamos a la BBDD y ejecutamos los siguientes comando

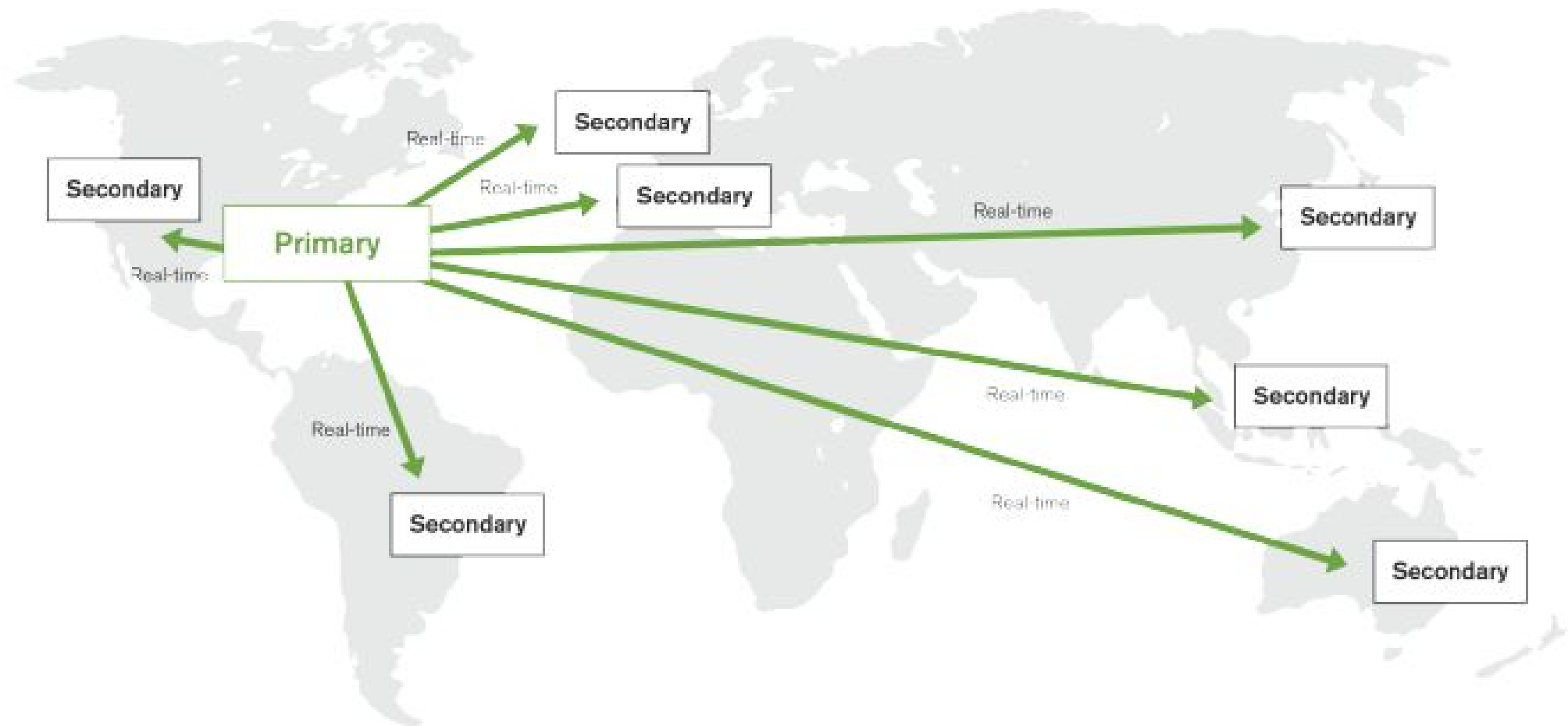
```
use admin
```

```
db.createUser({user:"root",pwd:"pass",roles: [{role:"root",db:"admin"}]})
```

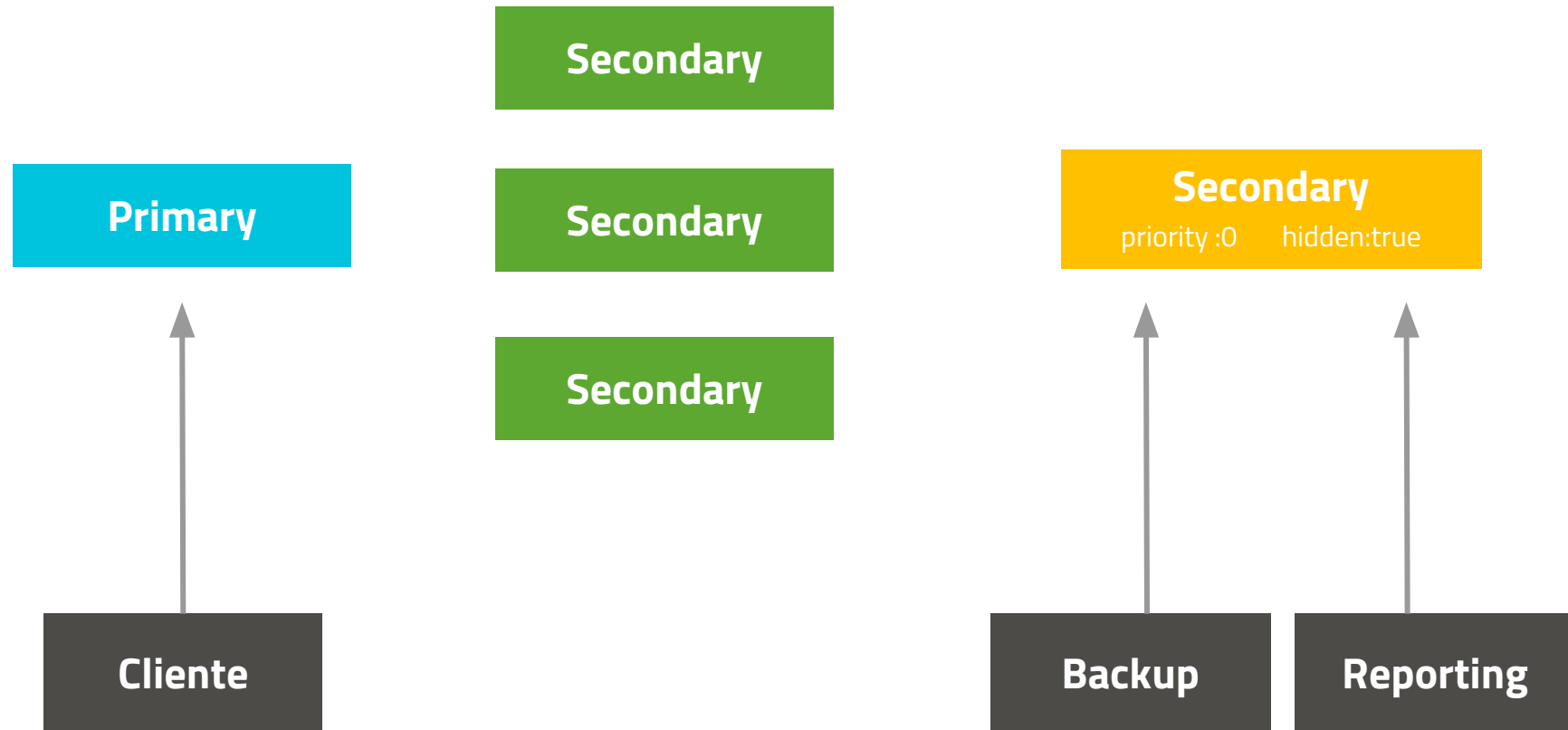
5. Salimos de la BBDD y volvemos a entrar con el usuario recién creado

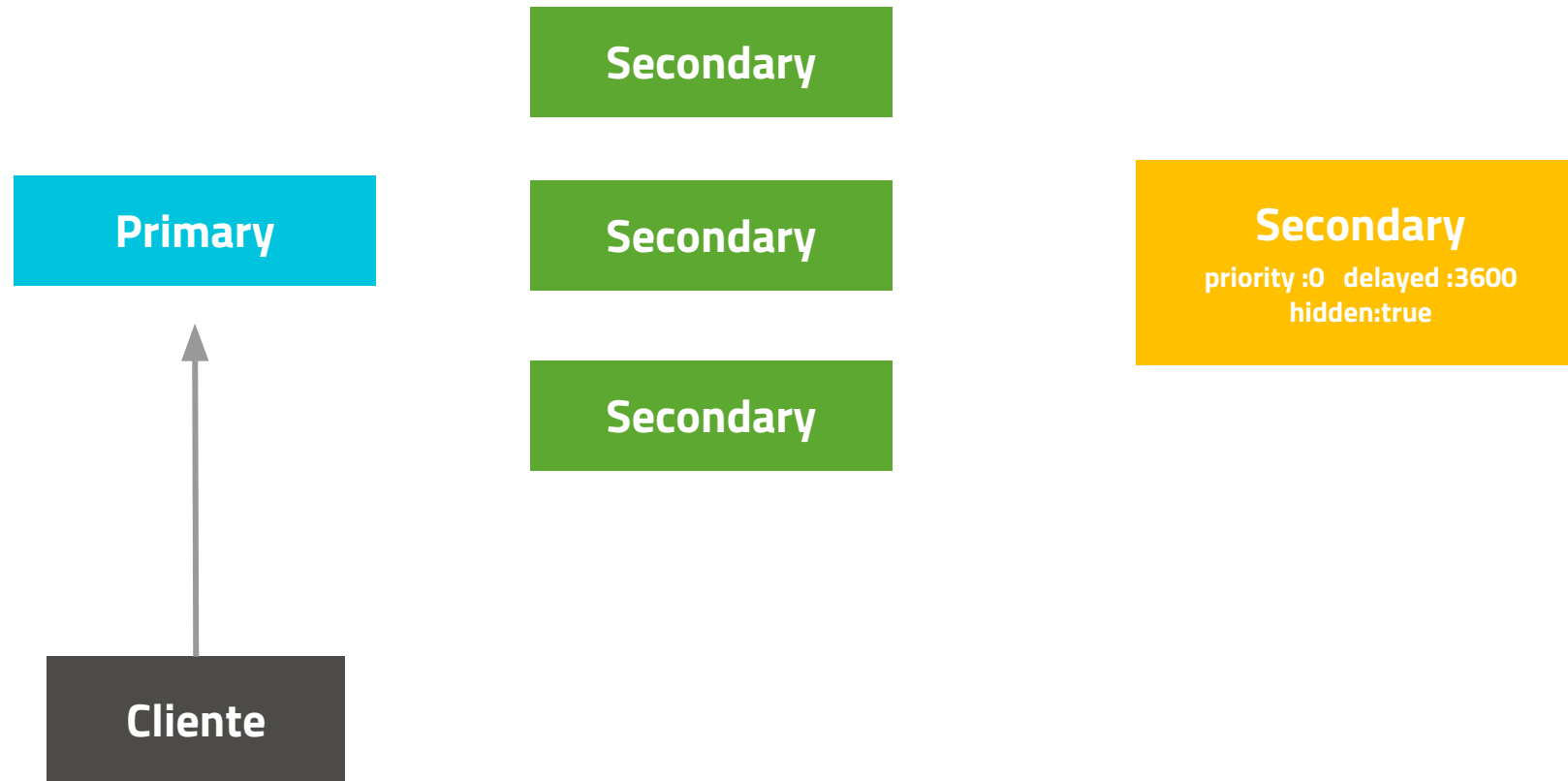
```
mongo admin -uroot -p
```

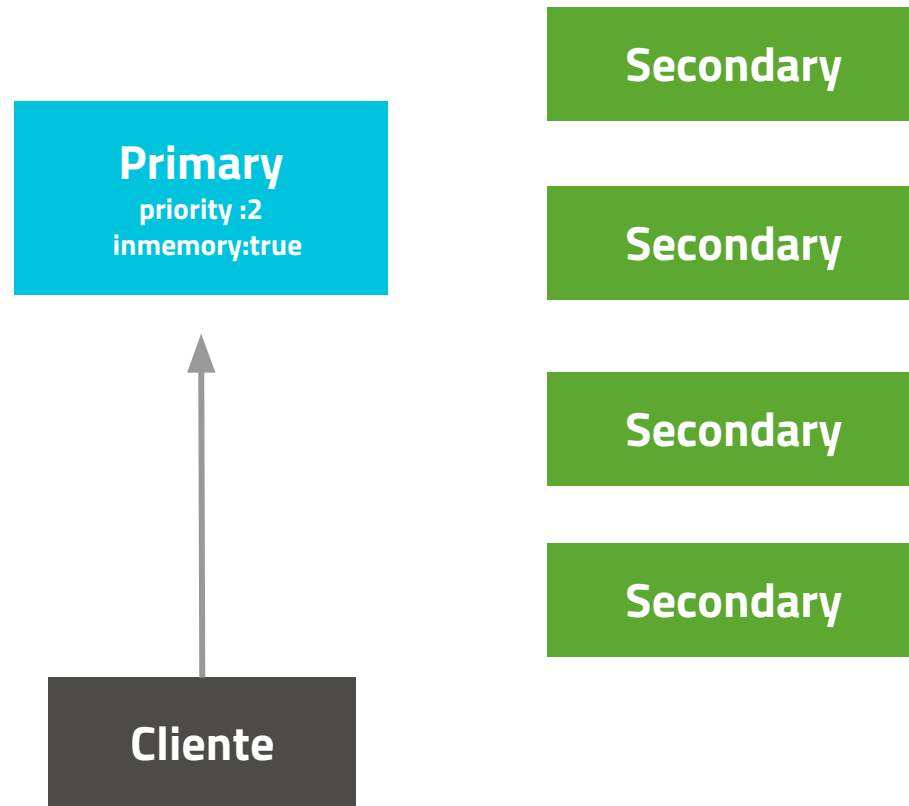


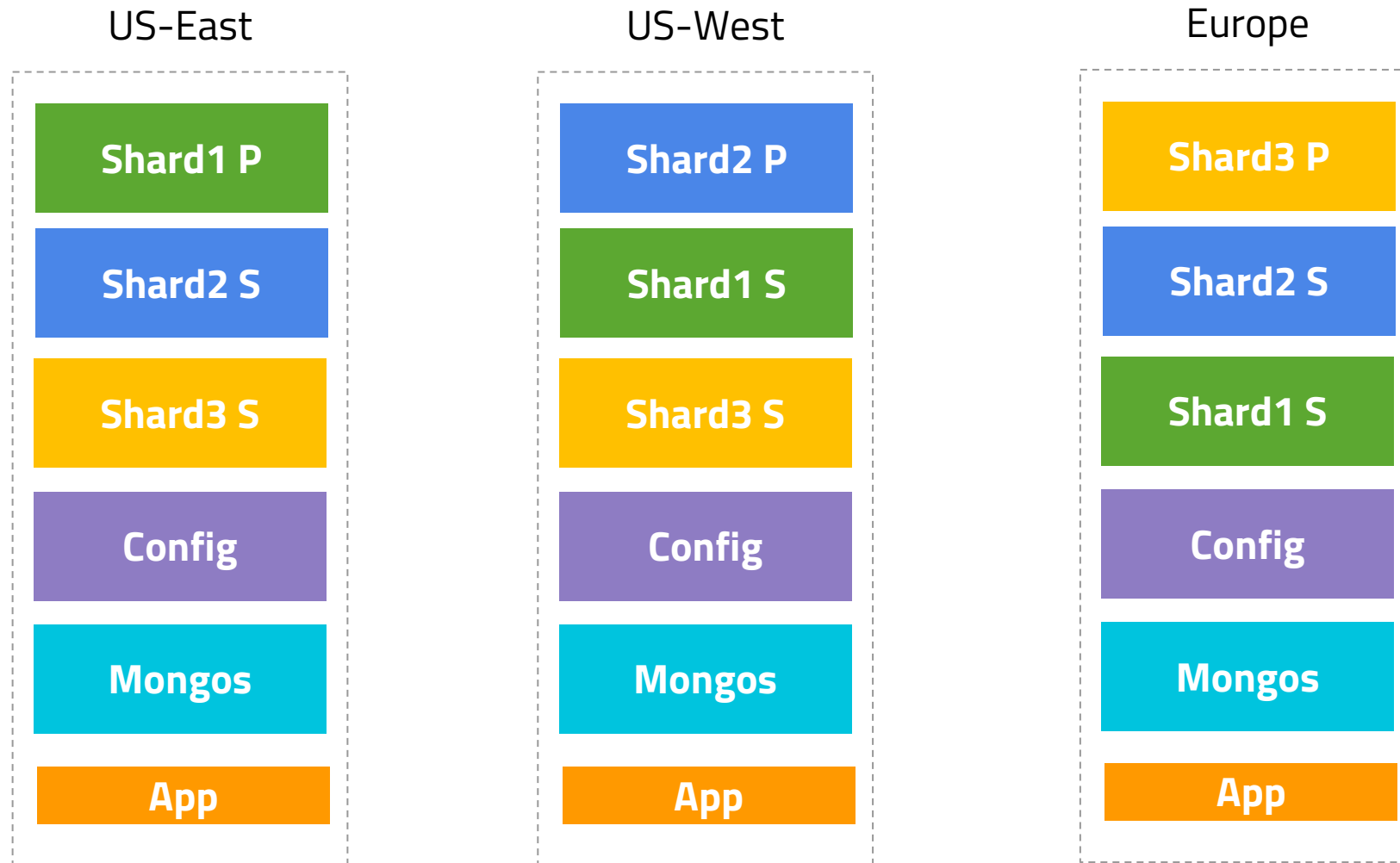


http://s3.amazonaws.com/info-mongodb-com/MongoDB_Multi_Data_Center.pdf







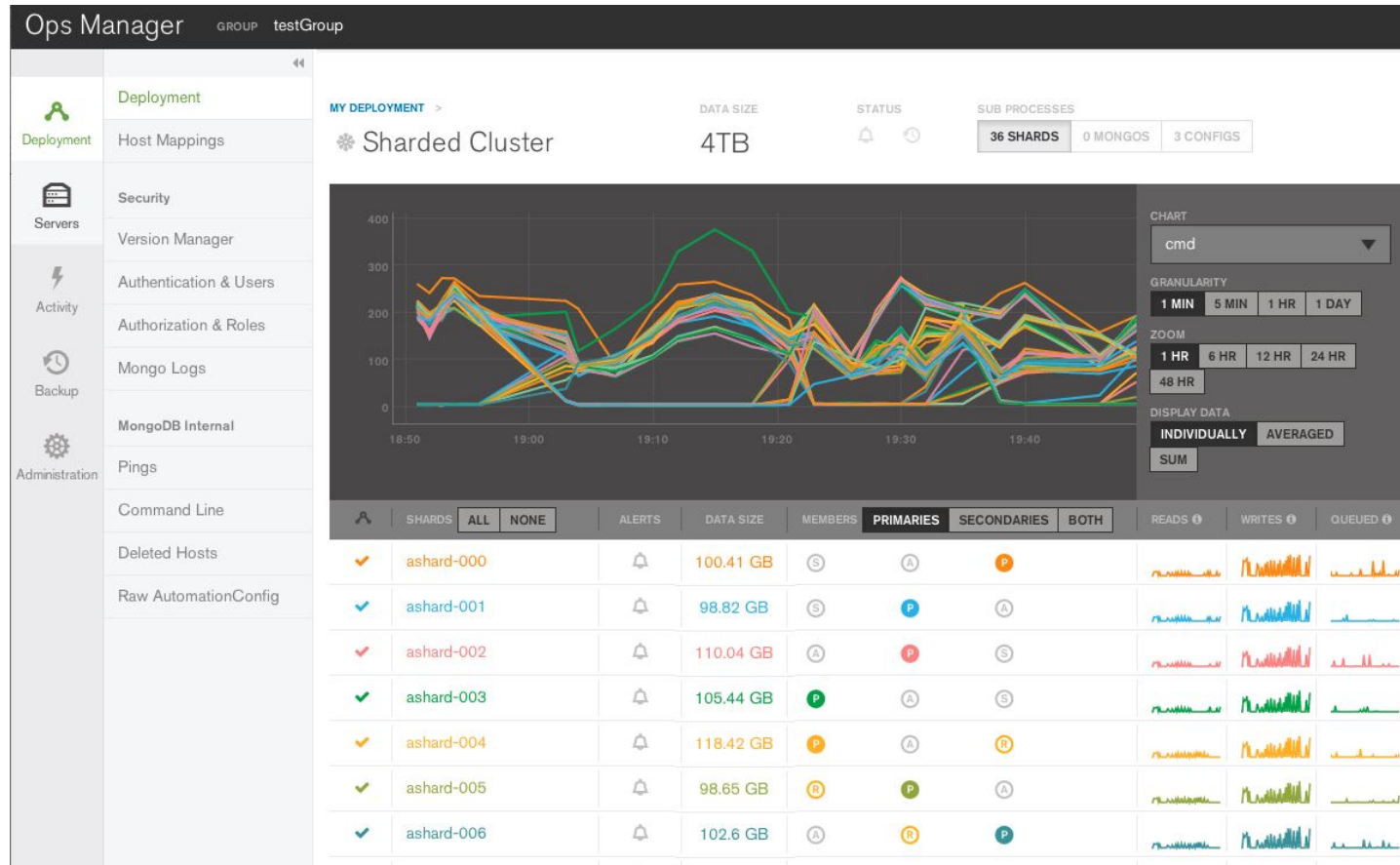


7 Operations Manager

Índice

1. Introducción a Ops Manager
2. Monitorización
3. Backup
4. Automatización
5. API

Ops Manager es una consola unificada que sirve para gobernar, monitorizar y salvaguardar datos de instancias de MongoDB



Ops Manager se presenta en dos formatos:

- Cloud Manager: <https://www.mongodb.com/cloud/>
 - Obtenemos la consola de Ops Manager en la nube. Tiene coste por la monitorización, automatización de tareas y almacenamiento de los backup de sharding y replica set
- Ops Manager On Premise
 - Desplegamos en nuestra infraestructura o cloud privada todos los componentes que son necesarios para albergar los servidores del servicio de Ops Manager

- Visualizar gráficas sobre un replica set o un shard completo o sobre una instancia en particular
- Ver gráficas de rendimiento o datos del profiling
- Visualizar logs de una instancia
- Visualizar y configurar alertas relacionadas con un despliegue

DEMO

- Crear nuevas tareas de backup sobre Replica Sets y Shard Clusters
- Administrar las políticas de retención
- Restaurar backup

DEMO

- Crear y reconfigurar despliegues
- Crear índices
- Calcular índices sugeridos
- Editar configuración de despliegues
- Parar y arrancar instancias
- Migrar agentes entre máquinas
- Migrar versiones de MongoDB de un entorno

DEMO

- Todas las tareas realizadas desde la consola pueden realizarse mediante el API ofrecido por Ops Manager.
- Pueden crearse usuarios para realizar peticiones a la API con distintos niveles de permisos
- A través de la API podemos orquestar flujos que podrían realizarse a través de la consola web

DEMO

8 Conclusiones

- MongoDB es muy útil para desarrollos con un bajo time to market
- Tiene un gran throughput de lectura y escritura, por lo que es útil para aplicaciones con número de peticiones muy exigentes y que manejen un volumen muy grande de datos.
- Tiene una gran variedad de casos de uso en los que MongoDB se adapta perfectamente
- Permite una gran flexibilidad en la definición de la información a almacenar
- La BBDD se adapta a la aplicación y no la aplicación a la BBDD
- Si buscas joins y transaccionalidad entre colecciones, MongoDB no es tu BBDD, aunque a partir de la versión 3.2 permite hacer left joins el framework de agregación
- MongoDB te ofrece capacidades de escalado, replicación y HA muy sencillos de configurar.
- El potente framework de agregación permite queries complejas y puede ser muy útil para la explotación de datos complejos

GRACIAS



mongoDB

{ paradigm