# **REPLICACIÓN DE BASES DE DATOS**

1.- Concepto de replicación, tipos

2.- Criterios de selección de una tecnología de replicación:

* Coherencia de los datos
* Autonomía de los sitios
* Particionamiento de datos

3.- Modelos de replicación:

* Componentes
* Replicación de instantáneas
* Replicación transaccional
* Replicación de mezcla o fusión
* Modelos físicos de replicación

4.- Planificación de la replicación:

* Opciones
* Replicación de instantáneas
* Replicación transaccional
* Replicación de mezcla o fusión

5.- Puesta en marcha de los distintos tipos de replicaciones:

* El distribuidor
* El editor
* Las publicaciones
* Las suscripciones

# Concepto

Una **replicación de base de datos** es una técnica mediante la cual copiamos de forma exacta en otra ubicación una instancia de la base de datos. Se utiliza en entornos distribuidos de Sistemas de Gestión de Bases de Datos donde una sola base de datos tiene que ser utilizada y actualizada en varios lugares de forma simultánea.

Permite que el contenido de uno o más servidores (llamados maestros) se refleje en uno o más servidores (llamados esclavos).

Los términos maestro y esclavo se han utilizado históricamente en la replicación, pero ahora se prefieren los términos primario y réplica .

# Tipos

Podemos hablar al menos de 3 tipos de **replicación de base de datos**:

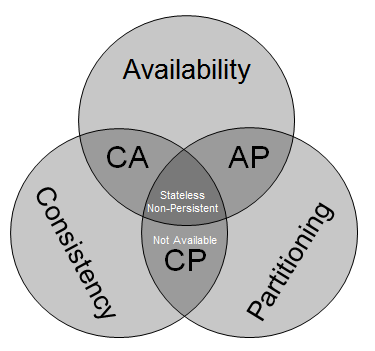
* **Replicación Instantánea**: los datos de un servidor son simplemente copiados a otro servidor o a otra base de datos dentro del mismo servidor. Al copiarse todo no necesitas un control de cambios. Se suele utilizar cuando los datos cambian con muy poca frecuencia.
* **Replicación Transaccional**: primero se envía una copia completa de la base de datos y luego se van enviando de forma periódica (o a veces continua) las actualizaciones de los datos que cambian. Se utiliza cuando necesitas que todos los nodos con todas las instancias de la base de datos tengan los mismos datos a los pocos segundos de realizarse un cambio.
* **Replicación de mezcla**: los datos de dos o más bases de datos se combinan en una sola base de datos. En primer lugar se envía una copia completa de la base de datos. Luego el Sistema de Gestión de Base de Datos va comprobando los cambios que van apareciendo en los distintos nodos y a una hora programada o a petición los datos se sincronizan. Es sobre todo útil cuando cada nodo suele utilizar solo los datos que se actualizan allí pero que por circunstancias necesita tener también los datos de los otros sitios.

# Criterios de selección de una tecnología de replicación (Teorema CAP)

Enunciado como conjetura por Brewer en el año 2000. Dice que en un sistema sólo se pueden dar en un momento dado 2 de las siguientes 3 propiedades:

1. Consistencia (Consistency): Información consistente en todos los nodos al acabar una escritura (Coherencia)
2. Disponibilidad (Availability): toda la información del sistema está siempre disponible y hay autonomía de los sitios.
3. Tolerancia a las Particiones (PartitionTolerance): datos distribuidos entre distintos nodos (particionamiento)

Gráficamente:



fuente: <http://gpd.sip.ucm.es/rafa/docencia/nosql/Replicas.html>

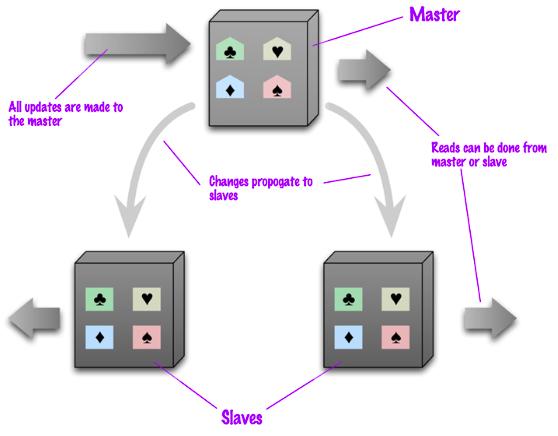


Figure 3. Replicación

La replicación de los datos se utiliza para alcanzar:

* **escalabilidad**, incrementando el rendimiento al poder distribuir las consultas en diferentes nodos, y mejorar la rendundancia al permitir que cada nodo tenga una copia de los datos.
* **disponibilidad**, ofreciendo tolerancia a fallos de hardware o corrupción de la base de datos. Al replicar los datos vamos a poder tener una copia de la base de datos, dar soporte a un servidor de datos agregados, o tener nodos a modo de copias de seguridad que pueden tomar el control en caso de fallo.
* **aislamiento** (la i en ACID - *isolation*), entendido como la propiedad que define cuándo y cómo al realizar cambios en un nodo se propagan al resto de nodos. Si replicamos los datos podemos crear copias sincronizadas que separan procesos de la base de datos de producción, pudiendo ejecutar informes o copias de seguridad en nodos secundarios de modo que no tenga un impacto negativo en el nodo principal, así como ofrecer un sistema sencillo para separar el entorno de producción del de preproducción.

|  |  |
| --- | --- |
|  | No hay que confundir la replicación (copia de los datos en varias máquinas) con la partición (cada máquina tiene un subconjunto de los datos). El entorno más seguro y con mejor rendimiento es aquel que tiene los datos particionados y replicados (cada maquina que tiene un subconjunto de los datos está replicada en 2 o más). |

# Beneficios de la replicación de base de datos

La replicación puede ofrecer grandes beneficios relacionados principalmente con el rendimiento, disponibilidad y seguridad de los datos.

1. **Aumento de la fiabilidad**: mediante la **replicación de base de datos** a través de múltiples servidores, te aseguras que los datos van a estar disponibles incluso en el caso de que una de las máquinas tenga un fallo grave de hardware. El sistema distribuido de gestión de bases de datos debe ser capaz de enrutar a los usuarios afectados a otro de los nodos disponibles.
2. **Mejora en el rendimiento**: al estar los datos distribuidos en diferentes servidores, los múltiples accesos no saturan los servidores. Esto es importante sobre todo en el caso de aplicaciones que pueden tener miles o cientos de miles de peticiones simultáneas. El rendimiento de las aplicaciones aumenta notablemente.
3. **Mejora en la seguridad de los datos**: en un sistema transaccional tradicional, todas las actualizaciones de una base de datos se guardan en un mismo disco. La seguridad de tus datos queda entonces en manos de la estrategia de copias de seguridad que tengas implementada en ese servidor. Con la **replicación de base de datos** aumentas la seguridad de los datos ya que las actualizaciones están siendo escritas en varios servidores. Es decir, varios discos, varias fuentes de alimentación, CPU’s, etc. son utilizadas para asegurar que tus datos estarán a salvo en algunos servidores, aunque pueda ocurrir un desastre en otros.

En definitiva la **replicación de base de datos** se utiliza para propagar los datos en entornos de base de datos distribuidas de forma que se mejora la confiabilidad y el rendimiento de las aplicaciones que la utilizan. Tienes diferentes tipos de **replicación de base de datos** que puedes utilizar. El escoger uno u otro dependerá de la naturaleza y utilización de los mismos.

<https://blog.powerdata.es/el-valor-de-la-gestion-de-datos/beneficios-de-la-replicacion-de-base-de-datos>

<https://es.wikipedia.org/wiki/Replicaci%C3%B3n_(inform%C3%A1tica)#Modelos_de_replicaci%C3%B3n_en_sistemas_distribuidos>

# 

# 

# 

# 

# 

# 

# DIFERENCIAS ENTRE BACKUP Y REPLICACIÓN (Cuadro comparativo)

<https://www.sololinux.es/diferencias-entre-backup-y-replication/>

##### **Comparativa Backup – Replication**

|  |  |  |
| --- | --- | --- |
|  | **Backup** | **Replication** |
| Requisitos | Un disco o dispositivo de almacenamiento. | Dos o más servidores. |
| Recomendado | Almacenamiento de copias de seguridad. | Aplicaciones críticas o siempre online. |
| Coste | Bajo | Alto |
| A favor | Fácil de implementar.  Económico.  Aislamiento ante amenazas. | Alta disponibilidad.  Alta recuperación ante desastres.  Inicio muy rápido. |
| En contra | Los tiempos de creación y de recuperación son largos. | Existe el riesgo de que el malware se expanda por todos los sistemas. |

# Replicación en mariadb:

<https://www.programacion.com.py/varios/db/guia-para-crear-una-replicacion-en-mariadb>

# Replicacion SQL Server:

<https://docs.microsoft.com/es-es/sql/relational-databases/replication/tutorial-preparing-the-server-for-replication?view=sql-server-ver15>

[https://docs.microsoft.](https://docs.microsoft.com/es-es/sql/database-engine/database-mirroring/database-mirroring-and-replication-sql-server?view=sql-server-ver15)[Tutorial: Preparación para la replicación - SQL Server](https://docs.microsoft.com/es-es/sql/relational-databases/replication/tutorial-preparing-the-server-for-replication?view=sql-server-ver15)[com/es-es/sql/database-engine/database-mirroring/database-mirroring-and-replication-sql-server?view=sql-server-ver15](https://docs.microsoft.com/es-es/sql/database-engine/database-mirroring/database-mirroring-and-replication-sql-server?view=sql-server-ver15)

# MySQL: Replicación de bases de datos en MySQL

<https://www.adictosaltrabajo.com/2009/12/08/mysql-replicacion/>

# Replicación master slave con MySQL

<https://codigoxules.org/replicacion-master-slave-con-mysql/>

# Replicación selectiva de tablas en un esquema Maestro – Esclavo de MySQL:

[Replicación selectiva de tablas en un esquema Maestro – Esclavo de MySQL](https://www.cloudcenterandalucia.es/blog/replicacion-selectiva-de-tablas-en-un-esquema-maestro-esclavo-de-mysql/)

### Tutoriales de replicación:

<https://docs.microsoft.com/es-es/sql/relational-databases/replication/replication-tutorials?view=sql-server-ver15>

# Invertir la réplica:

<https://puerto53.com/bases-de-datos/replica-maestro-esclavo-de-una-base-de-datos-mariadb/>

<https://mariadb.com/kb/en/changing-a-slave-to-become-the-master/>

# youtube

[Crear Base de Datos distribuida (Replica) MySQL en Linux](https://www.youtube.com/watch?v=jREw27H8UBE)

<https://www.youtube.com/watch?v=RY-EdBOJWEs>

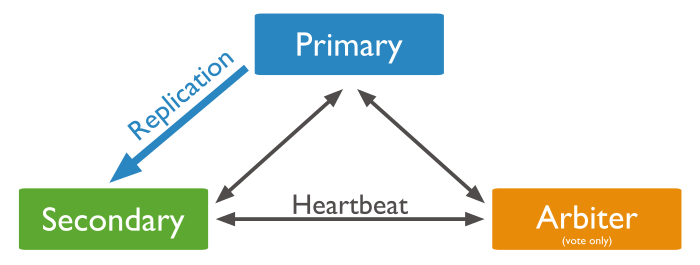
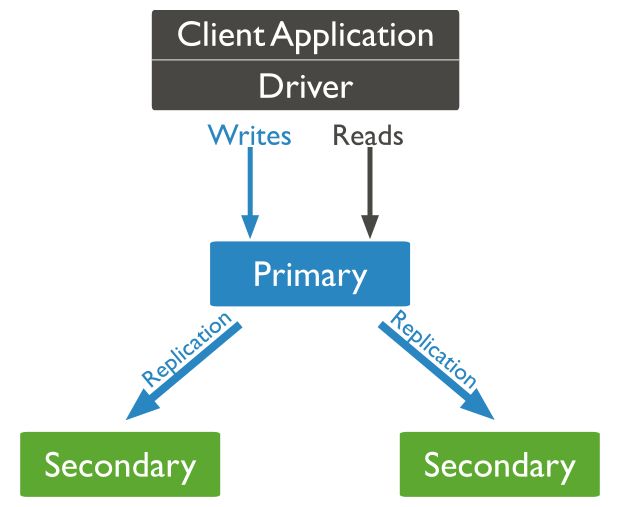
# REPLICACIÓN **NO SQL**

<https://www.fixedbuffer.com/como-crear-una-base-de-datos-en-alta-disponibilidad-con-mongodb/>

*MongoDB* es una de las bases de datos NoSQL más conocidas. Sigue un modelo de datos documental, donde los documentos se basan en JSON.

*MongoDB* destaca porque:

* Soporta esquemas dinámicos: diferentes documentos de una misma colección pueden tener atributos diferentes.
* No soporta *joins*, ya que no escalan bien.
* No soporta transacciones. Lo que en un SGDB puede suponer múltiples operaciones, con *MongoDB* se puede hacer en una sola operación al insertar/actualizar todo un documento de una sola vez.



<http://expertojava.ua.es/si/nosql/nosql.html#_replicaci%C3%B3n_2>

En *MongoDB* se replican los datos mediante un conjunto de réplicas (*Replica Set*), que es un grupo de servidores (nodos) donde hay uno que ejerce la función de **primario** y por tanto recibe las peticiones de los clientes, y el resto de servidores hace de **secundarios**, manteniendo copias de los datos del primario.

Si el nodo primario se cae, los secundarios eligen un nuevo primario entre ellos mismos, en un proceso que se conoce como votación. La aplicación se conectará al nuevo primario de manera transparente. Cuando el antiguo nodo primario vuelva en sí, será un nuevo nodo secundario.

#### **4.4.2. Elementos de un conjunto de réplicas**

Los tipos de nodos que podemos encontrar en un conjunto de réplica son:

* Regular: Es el tipo de nodo más común.

**Primario**: Acepta todas las operaciones de escritura de los clientes. Cada conjunto de réplicas tendrá sólo un primario, y como sólo un miembro acepta operaciones de escritura, ofrece consistencia estricta para todas las lecturas realizadas desde él.

**Secundario**: Los secundarios replican el *oplog* primario y aplican las operaciones a sus conjuntos de datos. De este modo, los nodos secundarios son un espejo del primario. Si el primario deja de estar disponible, el conjunto de réplica elegirá a un secundario para que sea el nuevo primario, mediante un proceso de votación.

* **Árbitro**: se emplea sólo para votar. No contiene copia de los datos, no se puede convertir en primario y no requieren hardware dedicado. Solo está presente cuando hay un número par de nodos.

* Retrasado (*delayed*): nodo que se emplea para la recuperación del sistema ante un fallo. Para ello, hay que asignar la propiedad priority:0. Este nodo nunca será un nodo primario.
* Oculto: empleado para analíticas del sistema.

##### **oplog (operations log)**

Para soportar la replicación, el nodo primario almacena todos los cambios en su ***oplog***.

De manera simplificada, el *oplog* es un diario de todos los cambios que la instancia principal realiza en las bases de datos con el propósito de replicar dichos cambios en un nodo secundario para asegurar que las dos bases de datos sean idénticas. El *oplog* crea un *timestamp* para cada entrada.

**(DESPUÉS DE ESTO HAY OTRA FORMA DE HACER LA REPLICACIÓN)**

<https://www.adictosaltrabajo.com/2016/07/05/replica-de-datos-en-mongodb/#:~:text=La%20r%C3%A9plica%20de%20datos%20es,que%20estamos%20persistiendo%20o%20recuperando.>

### 

### **PRÁCTICA**

### 

### **2. Requisitos**

Para hacer una replicación solamente hace falta la consola de Mongo y el API JavaScript que ofrece esta consola.

### **3. Arranque y acceso a la consola con Mongo Shell**

Para iniciar la prueba de ReplicaSet necesitamos arrancar la consola de mongo sin conectar contra ningún servidor en concreto. Esto lo conseguimos con el parámetro –nodb

$ mongo --nodb

MongoDB shell version: 3.2.0

### **4. Creación del ReplicaSet de ejemplo**

Para probar el mecanismo de réplica de MongoDB, necesitaremos crear varias instancias de mongod que actúen como servidores y relacionarlas de forma que todas mantengan una copia de los datos.

Habitualmente esto implica que configuraremos estos procesos de forma externa, pero para facilitar la configuración y arranque de estas instancias, MongoDB incluye en el API Javascript de su shell un objeto ReplSetTest que nos permite levantar estas instancias de una forma directa.

Para arrancarlo crearemos un nuevo objeto ReplSetTest. Como argumento al constructor del objeto podemos pasar un JSON con la configuración que queremos dar al grupo de réplica.

El conjunto de atributos que podemos configurar para este ReplicaSet está descrito en el Anexo A de este tutorial.

> sampleReplicaSet = new ReplSetTest({name : “myReplicaSet”, nodes : 3})

Al ejecutar el comando, por consola nos imprimirá el contenido del objeto ReplSetTest que acabamos de crear. Este efecto se produce porque la consola de mongoDB, por defecto siempre imprime el resultado de la evaluación de la última expresión. En este caso, la evaluación de la última expresión es la propia variable sampleReplicaSet que contiene la configuración del grupo de réplica que vamos a crear.

Si queremos evitar este efecto, podemos crear el ReplSetTest de la siguiente forma:

> sampleReplicaSet = new ReplSetTest({name : “myReplicaSet”, nodes : 3}); print(“done”)

done

>

### **5. Arrancar los procesos mongod del grupo de réplica**

Hasta ahora, sólo hemos configurado el grupo de réplica, pero no hemos iniciado ninguna instancia de los nodos que forma parte de dicho grupo de réplica.

Para arrancar los nodos del grupo de réplica ejecutaremos la función startSet() sobre el objeto que representa el grupo de réplica.

> sampleReplicaSet.startSet()

Como salida del comando veremos la configuración de cada nodo del grupo de réplica y los mensajes de logs que nos indican que están arrancando cada una de las instancias.

**CAPTURA DE LA SALIDA**

Durante la inicialización vemos los puertos en los que se va levantando cada uno de los nodos. En este caso, y por defecto, se levantan en los puertos 20012, 20013 y 20014.

A partir de aquí, es posible que veamos aparecer trazas en esta consola, ya que por defecto, la salida de los tres nodos que se han arrancado con ReplSetTest se volcará en esta shell.

### **6. Arrancar el proceso de réplica**

En este punto, tenemos los tres procesos de servicio mongod que forman parte de nuestro grupo de réplica, pero no ha arrancado todavía la funcionalidad de réplica de datos.

Para activar la réplica tendremos que invocar la funcion initiate() sobre el ReplSetTest.

> sampleReplicaSet.initiate()

# Al ejecutar la función, veremos como salida de consola nos muestra la configuración de miembros del réplica set y a continuación se activa la funcionalidad de réplica en el grupo

***CAPTURA DE RESULTADO***

Una vez activada la funcionalidad de réplica, veremos aparecer trazas en esta consola, ya que por defecto, la salida de los tres nodos que se han arrancado con ReplSetTest se volcará en esta consola.

### **7. Prueba del grupo de réplica**

Una vez que ya está configurado y activo el grupo de réplica, vamos a probar cómo se produce la réplica de datos entre todos los nodos que forman parte del grupo.

#### **7.1. Arranque de una nueva consola**

Para evitar que nuestras pruebas se vean entremezcladas con las trazas que los nodos del grupo de réplica están volcando en la consola con la que hemos configurado y arrancado el ReplSetTest, vamos a utilizar una nueva consola.

$ mongo --nodb

MongoDB shell version: 3.2.0

>

#### **7.2. Conexión al nodo primario del grupo de réplica**

Entre todos los nodos de un grupo de réplica, éstos pueden jugar dos roles distintos:

* **nodo primario**: sólo existe uno y es el único que acepta operaciones tanto de escritura como de lectura.
* **nodo secundario**: pueden existir más de uno. Son nodos que replican los datos del nodo primario, pero no aceptan operaciones de escritura ni de lectura (por defecto, aunque puede configurarse para permitir las operaciones de lectura).
* **nodos árbitro**: son nodos que no almacenan réplica de datos, su única función es participar en el proceso de elección de un nuevo nodo primario entre los nodos secundarios cuando el nodo primario anterior se cae

Para probar el funcionamiento del grupo de réplica que hemos levantado, es necesario que nos conectemos al nodo primario. Como a priori no podemos saber cuál es el nodo primario (el nodo primario se elige en un proceso de votación entre todos los nodos), deberemos conectarnos a cada uno de ellos y preguntar si es el nodo primario.

Para obtener la conexión a uno de los demonios mongod crearemos un nuevo objeto Mongo pasando como argumento la cadena de conexión compuesta del hostname y el puerto (que corresponderá con los puertos que vimos en la salida al arrancar los procesos en el [apartado 5](https://www.adictosaltrabajo.com/2016/07/05/replica-de-datos-en-mongodb/#05-arranque-grupo-mongod)):

> conn = new Mongo("localhost:20013")

connection to localhost:20013

Una vez obtenida la conexión, obtenemos la BD sobre la que realizaremos la prueba. En nuestro caso, para esta prueba vamos a utilizar la propia bd de test.

> testDB = conn.getDB(“test”)

test

Por último, sobre la BD, preguntaremos si es el nodo primario, utilizando la función isMaster():

> testDB.isMaster()

**CAPTURA DE RESULTADO**

Del objeto devuelto, nos fijaremos en la propiedad ismaster y secondary. Si el nodo al que nos hemos conectado no es primario, volveremos a conectarnos al siguiente nodo del grupo de réplica, hasta que encontremos el nodo primario (o podemos identificarlo como el que está marcado con el atributo primary)

> conn = new Mongo("localhost:20012")

connection to localhost:20012

> testDB = conn.getDB("test")

test

> testDB.isMaster()

**CAPTURA DE RESULTADO**

#### **7.3. Insertamos un conjunto de datos sobre el nodo primario**

Una vez que ya estamos conectados al nodo primario, vamos a ejecutar una inserción de un conjunto de datos en la colección de ejemplo (en nuestro caso, por ejemplo una serie de entradas de un blog).

> for (i = 0; i < 1000; i++) {

... testDB.blog\_posts.insert(

... {

... author: "author " + i,

... blog\_title : "Blog post entry by author " + i

... });

... }

WriteResult({ "nInserted" : 1 })

Por último, comprobamos que se han almacenado los registros en la colección.

> testDB.blog\_posts.count();

1000

#### **7.4. Comprobación de la réplica sobre los nodos secundarios**

Una vez que hemos insertado los datos a través del nodo primario, vamos a conectarnos a alguno de los nodos secundarios, y comprobar si se han replicado los datos a ese nodo.

Empezamos por conectarnos a uno de los nodos secundarios, obtener la conexión y comprobamos que, efectivamente, el nodo es secundario:

> connSecondary = new Mongo("localhost:20014")

connection to localhost:20014

> secondaryTestDB = connSecondary.getDB("test")

test

> secondaryTestDB.isMaster()

**CAPTURA DE RESULTADOS**

Ahora que estamos ya conectados a la base de datos en el nodo secundario dentro del grupo de réplica, vamos a comprobar si los datos que hemos insertado en el nodo primario, se han replicado en este secundario.

Para ello hacemos una consulta a la colección (debería devolver los mismos datos que en la colección del nodo primario):

> secondaryTestDB.blog\_post.count();

**CAPTURA DE RESULTADOS**

En este caso obtenemos un error porque por defecto, tal y como comentamos en el [7.2 sobre la conexión al nodo primario del grupo de réplica](https://www.adictosaltrabajo.com/2016/07/05/replica-de-datos-en-mongodb/#071-nueva-consola), los nodos secundarios en un grupo de réplica, no admiten operaciones ni de escritura ni de lectura. Todas las operaciones deben realizarse siempre sobre el nodo principal.

Sin embargo, podemos activar el permiso para realizar operaciones de lectura sobre un nodo secundario.

Para ello utilizaremos la función setSlaveOK() sobre la conexión al nodo secundario. La invocación de esta función significa que le estamos indicando a mongoDB que somos conscientes de que estamos trabajando sobre un nodo secundario. A partir de aquí la responsabilidad de lo que hagamos es nuestra,

> connSecondary.setSlaveOk()

>

En este momento, ya podemos lanzar la operación de consulta y comprobar que la réplica de datos funciona:

> secondaryTestDB.blog\_posts.count();

1000

> secondaryTestDB.blog\_posts.findOne();

{

"\_id" : ObjectId("56cdeeb47b058dd8d549a12d"),

"author" : "author 1",

"blog\_title" : "Blog post entry by author 1"

}

En este caso, comprobamos que los datos que insertamos en el nodo principal se han replicado en el nodo secundario.

### **8. Promoción automática de un nodo secundario ante la caída del primario**

Como hemos explicado, los nodos pueden tomar varios roles (primario, secundario y árbitro) en un grupo de réplica, siendo sólo el nodo primario el que admite las operaciones de escritura y de consulta.

En este apartado, vamos ahora a probar qué ocurre cuando el nodo primario se cae, y comprobar que automáticamente uno de los nodos secundarios toma el papel de primario y empieza a admitir las operaciones.

#### **8.1. Parada del nodo primario**

Empezamos por parar específicamente el nodo primario, para simular una caída (o una pérdida de conexión). Para ello podríamos :

* ejecutar el comando kill a nivel de sistema operativo
* o simplemente enviar el comando de parada del nodo a través del API JS de la consola.

Optaremos por la segunda opción, pero comprobando previamente que el nodo al que nos hemos conectado es el primario:

> connPrimary = new Mongo("localhost:20014")

connection to localhost:20014

> primaryDB = connPrimary.getDB("test")

test

> primaryDB.isMaster()

**OTRA CAPTURA MÁS**

En este caso, vemos que, el nodo primario es el 20014. Vamos a parar este nodo, para ello emitimos el comando de apagado:

> primaryDB.adminCommand({shutdown : 1});

En este momento, en la consola de mongo en la que habíamos creado el ReplSetTest, y donde se están volcando las trazas, veremos que se ha detectado una pérdida de conexión (“Error in heartbeat”) con el nodo que hemos apagado e, inmediatamente, se produce un proceso de election para elegir al nuevo nodo primario:

**CAPTURA DE ALGO ASÍ:**

[...]

d20012| 2016-02-25T16:00:15.419+0100 I REPL [ReplicationExecutor] Error in heartbeat request to Irensaga-2.local:20014; HostUnreachable Connection refused

d20012| 2016-02-25T16:00:18.948+0100 I REPL [ReplicationExecutor] Starting an election, since we've seen no PRIMARY in the past 10000ms

d20012| 2016-02-25T16:00:18.949+0100 I REPL [ReplicationExecutor] conducting a dry run election to see if we could be elected

d20012| 2016-02-25T16:00:18.949+0100 I REPL [ReplicationExecutor] dry election run succeeded, running for election

d20012| 2016-02-25T16:00:18.950+0100 I REPL [ReplicationExecutor] election succeeded, assuming primary role in term 2

d20012| 2016-02-25T16:00:18.950+0100 I REPL [ReplicationExecutor] transition to PRIMARY

En las trazas vemos que el demonio d20012 ha perdido conectividad con el pulso del nodo que corre en el puerto 20014, y cómo este inicia un proceso de elecciones en el que se erige en candidato a ser el primario.

Después de ser validado para ser el nuevo primario, promociona a nodo PRIMARY, o master.

#### **8.2. Comprobación del nuevo nodo primario**

Vamos a ahora a comprobar que el nodo efectivamente es el primario. Primero obtenemos una conexión al nuevo nodo primario:

> connNewPrimary = new Mongo("localhost:20012")

connection to localhost:20012

> newPrimaryDB = connNewPrimary.getDB("test")

test

> newPrimaryDB.isMaster()

**OTRA CAPTURA MÁS**

### **9. Parada del ReplicaSet de pruebas**

Por último, una vez finalizada la prueba de cómo funciona el mecanismo de réplica, vamos a parar el grupo de réplica.

Para ello, en la consola de mongo donde configuramos y arrancamos el grupo de réplica (la consola donde estamos viendo las trazas de los nodos), ejecutamos el siguiente comando para parar el grupo de réplica:

> sampleReplicaSet.stopSet()

**OTRA CAPTURA MÁS**

### **A. Configuración avanzada en la construcción del ReplSetTest**

En la creación del ReplSetTest, se puede indicar como argumento un objeto JSON con la configuración detallada de cómo queremos crear el RéplicaSet.

Este objeto JSON puede tener las siguientes propiedades:

* **name**: de tipo String, con el nombre que queremos asignar al grupo de réplica. Por defecto toma el valor *testReplSet*
* **host** : de tipo String, con el nombre de la máquina donde se va a crear el grupo de réplica. Por defecto toma el *hostname*.
* **useHostName**: de tipo boolean, indica si se debe utilizar el hostname de la máquina como host (en caso de true) o utiliza “localhost” (en caso de ser false) para los nodos. Por defecto a *true*.
* **nodes**: Indica las instancias de mongod que formarán parte del grupo de réplica. El valor de este atributo de configuración puede ser de varios tipos:
  + de tipo entero, indica el número de instancias de mongod que se crearán en el grupo de réplica. Por defecto toma el valor 0.
  + de tipo objeto JSON, con la configuración de la instancia que quiere que sea parte del grupo de réplica. Este objeto JSON puede tener la siguiente estructura (atributos) de configuración
    - **useHostName** : de tipo boolean, cuando se configura a true utiliza el hostname de la máquina.
    - **forceLock**: de tipo boolean, si está puesto a true, borra el fichero de *lock*
    - **dbpath**: de tipo string, con la ruta de los ficheros de base de datos. Por defecto /data/db/{nodename}
    - **cleanData**: de tipo boolean, si está configurado a true, elimina los ficheros que hubiese en la ruta dbpath antes de arrancar la instancia
    - **startData**: equivalente *cleanData*.
    - **noCleanData**: de tipo boolean, si está puesto a true mantiene los ficheros que ya existiesen en la ruta dbpath. Este valor tiene prioridad si está especificado también la propiedad cleanData.
    - **arbiter:**: de tipo boolean, indica si la instancia se quiere que tome el papel de árbitro para las votaciones en caso de caída del nodo primario del grupo de réplica.
  + de tipo array, con los distintos objetos JSON de configuración (según están descritos en el punto anterior) de cada una de las instancias que se quieren incluir en el grupo de réplica.
* **nodeOptions**: de tipo objeto JSON, con las opciones que se quieren aplicar a todos las instancias que formarán parte del grupo de réplica. Este objeto toma los valores de los argumentos que pasaríamos por linea de comandos al comando mongod arrancar las instancias de cada uno de los nodos del grupo de réplica.
* **opLogSize**: de tipo numérico, tamaño del registro de operaciones que se mantienen para poder sincronizar los distintos nodos del grupo de réplica después de una caída de uno o varios de ellos. Por defecto toma el valor 40.
* **useSeedList**: de tipo boolean, si se configura, la cadena de conexión se utilizará como nombre para el replicaSet. Este valor sobreescribe el valor asignado al atributo name, si se ha configurado. Por defecto toma el valor *false*
* **protocolVersion**: de tipo numérico, indica la versión del protocolo a utilizar en la inicialización del grupo de réplica.

### **Resolución de problemas**

#### **startSet() no arranca los demonios mongod del grupo de réplica, por error en la ruta.**

### **Configuración avanzada en la construcción de ReplSetTest**

Si al ejecutar el comando de arranque de los nodos del del grupo de réplica no vemos que se levanten los procesos mongod y obtenemos un error de permiso denegado o de acceso a la ruta donde están los ficheros de la BD:

**CAPTURA DEL ERROR**

> sampleReplicaSet.startSet()

ReplSetTest Starting....

Resetting db path '/data/db/myReplicaSet-0'

2016-02-24T16:19:32.710+0100 E QUERY [thread1] Error: Caught std::exception of type boost::filesystem::filesystem\_error: boost::filesystem::create\_directory: Permission denied: "/data/db/myReplicaSet-0" :

MongoRunner.runMongod@src/mongo/shell/servers.js:615:13

ReplSetTest.prototype.start@src/mongo/shell/replsettest.js:771:16

ReplSetTest.prototype.startSet@src/mongo/shell/replsettest.js:249:16

deberemos comprobar que tenemos permisos de acceso y escritura en la ruta de configurada como dbpath (por defecto /data/db):

* Podemos asignar permisos en /data/db al usuario con el que ejecutamos la consola mongo
* Podemos ejecutar mongo –nodb como superusuario
* Podemos crear el ReplSetTest utilizando un dbpath diferente para los nodos del grupo de réplica. Ver propiedad dbpath del atributo nodes del objeto JSON que pasamos como configuración al constructor.

# 

# 

# 

# 

# <http://expertojava.ua.es/si/nosql/nosql.html>

1.1.3 y 4.4, 4.5

Replicación para crear una BD de alta disponibilidad

<https://www.fixedbuffer.com/como-crear-una-base-de-datos-en-alta-disponibilidad-con-mongodb/>

REPLICACIÓN PASO A PASO BIEN EXPLICADA:

[adictosaltrabajo.com/2016/07/05/replica-de-datos-en-mongodb/#:~:text=La%20réplica%20de%20datos%20es,que%20estamos%20persistiendo%20o%20recuperando.](http://adictosaltrabajo.com/2016/07/05/replica-de-datos-en-mongodb/#:~:text=La%20r%C3%A9plica%20de%20datos%20es,que%20estamos%20persistiendo%20o%20recuperando.)

<https://docs.mongodb.com/manual/replication/>

# No SQL

<http://expertojava.ua.es/si/nosql/nosql.html>

**PRACTICA CON 3 MÁQUINAS VIRTUALES**

<https://github.com/RafaMunoz/Replica-Set-MongoDB>

<http://labs.conekia.es/mongodb-replicacion-de-bases-de-datos-nosql/>

<https://github.com/rickardo10/distributed-db-project>

<https://docs.mongodb.com/manual/tutorial/deploy-replica-set/>

<https://docs.mongodb.com/manual/tutorial/deploy-replica-set-for-testing/>

posible error

<https://stackoverrun.com/es/q/8334304>

<http://labs.conekia.es/mongodb-replicacion-de-bases-de-datos-nosql/>