Cuaderno de prácticas

Autor :
Ernesto Valero Marcelo
evalero@paradigmadigital.com

Introducción

Mediante esta guía, se facilitan las instrucciones a los alumnos para poder llevar a cabo todos los ejercicios propuestos para mejorar su aprendizaje y aprovechamiento de este curso

Máquina virtual

La máquina virtual proporcionada tiene todo lo necesario para poder llevar a cabo los laboratorios y ejercicios propuestos. El usuario y password para acceder es mongodb / mongodb

Repositorio

Este curso dispone de un repositorio en github donde puede encontrarse material adicional de apoyo.
https://github.com/evalero/Troubleshooting-Mongodb

Advertencia

Si se hace copy paste de algunos de los comandos actualmente mostrados en esta guía sobre la consola de MongoDB, puede introducir caracteres no imprimibles que la consola de mongo procesa pero no se muestran por pantalla.

Tema 1 Configuración de MongoDB

En la máquina virtual ofrecida el paquete **mongodb-org se encuentra instalado en su versión más reciente** para evitar posibles problemas con conexiones a internet. A continuación vamos a proceder a desplegar distintas con diferentes configuraciones

Despliegue en Standalone

Para entornos productivos o estables, es altamente recomendable **utilizar ficheros de configuración y script de arranque para operar una instancia de MongoDB**. En el presente ejercicio vamos a configurar y levantar una instancia de MongoDB en standalone usando fichero de configuración y el script de arranque proporcionado por MongoDB.

Primeramente echamos un vistazo al contenido del fichero:

```
cat  ~Troubleshooting-Mongodb/Tema1/standalone/mongod.conf
systemLog:
   destination: file
   path: "/var/log/mongod/mongod.log"
   logAppend: true
storage:
   dbPath: /data/db
processManagement:
   fork: true
net:
   port: 27017
```

Observamos que necesitamos crear un directorio para almacenar los ficheros de la base de datos y otro para el log.
Ejecutamos los siguientes comandos:

```
sudo mkdir -p /data/db
sudo chown mongod:mongod /data/db
```

sudo mkdir -p /var/log/mongod
sudo chown mongod:mongod /var/log/mongod/

```
sudo cp ~/Troubleshooting-Mongodb/Tema1/standalone/mongod.conf
/etc/
sudo systemctl start mongod
```

Despliegue en Replica Set

Vamos a crear un entorno en replica set usando un único nodo simulando un entorno de 3 instancias con replica factor 3. Para facilitar la tarea en el laboratorio, no vamos a utilizar los scripts de arranque para levantar las instancias y lo haremos directamente por comando.

Paramos todas las instancias de mongodb existentes

```
killall mongod
```

o

systemctl stop mongod

Creamos 3 directorios para almacenar los datos:

```
mkdir -p /data/RS1_1
mkdir -p /data/RS1_2
mkdir -p /data/RS1_3
```

Levantamos 3 instancias

```
mongod -f RS1_1.conf
mongod -f RS1_2.conf
mongod -f RS1_3.conf
```

Nos conectamos a una de las instancias usando el comando **mongo** (este comando nos conectará a la instancia que escucha por la ip localhost y puerto 27017)

Una vez ejecutamos los siguientes comandos:

```
rs.initiate()
rs.add("mongodb:27018")
rs.add("mongodb:27019")
```

Una vez creado el replica set podemos alimentarlo con datos usando el siguiente script:

```
mongo populateDB.js > configuracionReplicaSet.out
```

La salida se va a redirigir a un fichero para guardar la configuración original del replica set para restaurarla más adelante

Tema 2 Configuración de MongoDB

Cambio de prioridad de un nodo de un replica set

Se puede asignar cierta afinidad a un nodo de un replica set para que tenga mayor probabilidad de ser elegido como nodo Primario. Para ello cambiamos la configuración del replica set y le asignamos un valor priority mayor que 1 y menor que 1000

Para hacerlo, nos conectamos al nodo que tiene rol de primario, si no lo sabemos cual es en primera instancia, podemos ejecutar el comando:

```
mongo --host
<nombreReplicaSet>/host1:puerto,host2:puerto,host3:puerto
```

Para nuestro laboratorio:

```
mongo --host training/mongodb:27017,mongodb:27018,mongodb:27019
```

Ejecutamos el siguiente comando:

```
cfg = rs.config()
```

Esto creará una variable llamada cfg que contiene la configuración del Replica Set actualmente creado. Para ver su contenido, ejecutamos el comando:

```
cfg
```

Y obtendremos como salida algo parecido a lo siguiente:

```
{
	"_id" : "training",
	"version" : 3,
	"protocolVersion" : NumberLong(1),
	"members" : [
		{
			"_id" : 0,
			"host" : "mongodb:27017",
			"arbiterOnly" : false,
			"buildIndexes" : true,
			"hidden" : false,
			"priority" : 1,
			"tags" : {

			},
			"slaveDelay" : NumberLong(0),
			"votes" : 1
		},
		{
			"_id" : 1,
			"host" : "mongodb:27018",
			"arbiterOnly" : false,
			"buildIndexes" : true,
			"hidden" : false,
			"priority" : 1,
			"tags" : {

			},
			"slaveDelay" : NumberLong(0),
			"votes" : 1
		},
		{
			"_id" : 2,
			"host" : "mongodb:27019",
			"arbiterOnly" : false,
			"buildIndexes" : true,
			"hidden" : false,
			"priority" : 1,
			"tags" : {

			},
			"slaveDelay" : NumberLong(0),
			"votes" : 1
		}
	],
	"settings" : {
```

```
            "chainingAllowed" : true,
            "heartbeatIntervalMillis" : 2000,
            "heartbeatTimeoutSecs" : 10,
            "electionTimeoutMillis" : 10000,
            "getLastErrorModes" : {

            },
            "getLastErrorDefaults" : {
                  "w" : 1,
                  "wtimeout" : 0
            },
            "replicaSetId" : ObjectId("57b6c1de685142dc1be1d3ff")
      }
}
```

El objetivo es cambiar el valor del parámetro priority de la instancia mongodb:27019 y asignarle el valor 10. Para ello tenemos que acceder dentro de la variable cfg al tercer elemento del array de miembros y cambiar el valor priority a 10.
Para ello ejecutamos el comando :
`cfg.members[2].priority=10`
Y comprobamos ejecutando el comando cfg que el parámetro tiene el valor deseado.
Para reconfigurar el replica set, ejecutamos el comando
`rs.reconfig(cfg)`
Este comando reconfigura el replica set con el valor de la variable cfg que le hemos pasado por parámetro.
Tras la ejecución de este comando ejecutamos nuevamente rs.status() y observamos que la instancia mongodb:27019 tiene el rol de primario. Si la conexión la hubiéramos establecido únicamente sobre la instancia mongodb:27017, el prompt de la consola de MongoDB habría cambiado a SECONDARY y habríamos tenido que reconectar contra el nuevo primario.

Convertir un nodo del replica set a tipo hidden

En ciertas ocasiones, nos puede interesar mantener un nodo que tenga una copia completa de las bases de datos que componen el Replica Set pero que no sea elegible como nodo primario ni que sea visible de cara a los clientes. Este tipo de nodos son buenos para hacer un uso distinto al normal de la aplicación que el resto de nodos que componen el Replica set (por ejemplo, lanzar un proceso de billing de un servicio consultando distintas colecciones que haga que su uso extensivo pudiera afectar a la producción) o para tenerlo como nodo de backup

Estos nodos son miembros con derecho a voto en la elección de un nuevo nodo primario **pero nunca podrán convertirse en nodos con rol de primario.**

Para configurar un nodo con dicho rol debemos cambiar la configuración del replica set como en el ejercicio anterior. En este caso vamos a añadir en el documento de la configuración del replica set al nodo **mongodb:27018** los siguientes parámetros:

```
priority :0,
hidden:true
```

Recuperamos la configuración en la variable cfg como hicimos en el ejercicio anterior

```
cfg = rs.config()
```

Y añadimos los parámetros anteriormente mencionados para el segundo nodo del réplica set

```
cfg.members[1].priority=0
cfg.members[1].hidden=true
```

Y aplicamos la configuración

```
rs.reconfig(cfg)
```

Si ejecutamos el comando db.isMaster():


```
{
        "hosts" : [
                "mongodb:27017",
                "mongodb:27019"
        ],
        "setName" : "training",
        "setVersion" : 5,
        "ismaster" : true,
        "secondary" : false,
        "primary" : "mongodb:27019",
        "me" : "mongodb:27019",
        "electionId" : ObjectId("7fffffff0000000000000005"),
        "maxBsonObjectSize" : 16777216,
        "maxMessageSizeBytes" : 48000000,
        "maxWriteBatchSize" : 1000,
        "localTime" : ISODate("2016-09-09T12:31:18.324Z"),
        "maxWireVersion" : 4,
        "minWireVersion" : 0,
        "ok" : 1
}
```


Únicamente nos aparecen las instancias de los pusertos 27017 y 27019 como miembros elegibles, sin embargo si ejecutamos el comando rs.status(), observamos que las 3 instancias forman parte del replica set.

Convertir un nodo del replica set a delayed

Podemos disponer en un replica set de un nodo que esté por detrás de la replicación en un tiempo, por ejemplo para recuperarnos ante la ejecución de un comando que borre toda la base de datos. Para ello, en el documento de la configuración del replica set debemos asignar al miembro que deseemos que sea delayed los siguientes parámetros:

```
cfg.members[0].priority = 0
cfg.members[0].hidden = true
cfg.members[0].slaveDelay = <segundos>
```

Donde segundos es el número de segundos que queremos que la réplica vaya por detrás en dicho nodo. Como en el ejercicio anterior, hemos configurado la instancia del puerto 27019 como hidden y sin prioridad, vamos a añadirle el parámetro slaveDelay para que vaya 2 minutos por detrás de la réplica. Ejecutamos:

```
cfg = rs.config()
cfg.members[1].slaveDelay=120
rs.reconfig(cfg)
```

Ahora, abrimos una nueva consola de mongo contra el nodo delayed ejecutando el comando

```
mongo --port 27018
```

Y ejecutamos el comando

```
rs.slaveOk()
```

Esto nos permitirá consultar los datos dentro de un nodo secundario (nunca podremos escribir). Volvemos a la otra consola donde tenemos establecida la conexión contra el nodo primario y ejecutamos los siguientes comandos:

```
use testDelayed
db.test.insert({"mensaje":"Se va a replicar en 2 minutos"})
```

Volvemos a la consola donde hemos establecido la conexión contra el nodo delayed y ejecutamos el comando

```
use testDelayed
db.test.find()
```

Ejecutar el último comando hasta que obtengamos el documento insertado en la BBDD

Para dejar la configuración del replica set original, salimos de la consola de mongo y modificamos el fichero replicaSetOriginal.js borrando las líneas siguientes :

```
MongoDB shell version: 3.2.4
connecting to: test
introduciendo datos en la BBDD miBD
Configuración original del replica set :
```

Donde aparece el bloque de configuración del replica set, añadimos delante :

```
cfgOrig={
      "_id" : "training",
[se omite el resto del contenido]
```

Nos conectamos contra la instancia que tenga rol de primario y ejecutamos
`load("./replicaSetOriginal.js")`
y por último ejecutamos
`rs.reconfig(cfgOrig)`

Comprobamos que los cambios que hemos aplicado son correctos ejecutando rs.status () y rs.config()

Creación no bloqueante de índices

Cuando necesitamos realizar algunas operativas que pueden suponer un bloqueo a nivel de BBDD, es necesario aplicarlo en un proceso de rolling update. Para ello **realizaremos primero la tarea sobre los nodos secundarios y por último en los nodos primarios.**

Es importante que antes de aplicar una operativa en modo rolling upgrade, se compruebe que **el oplog tenga suficiente ventana de replicación para ponerse a nivel tras aplicar la operativa de mantenimiento.** Para ello podemos ejecutar el comando rs.printSlaveReplicationInfo()

```
source: mongodb:27017
      syncedTo: Mon Sep 12 2016 18:24:28 GMT+0200 (CEST)
      0 secs (0 hrs) behind the primary
source: mongodb:27018
      syncedTo: Mon Sep 12 2016 18:24:28 GMT+0200 (CEST)
      0 secs (0 hrs) behind the primary
```

Esta información te indica cual es el retraso de la aplicación de las operaciones del oplog escritas en el primario con respecto a cada uno de los nodos secundarios.

```
 rs.printReplicationInfo()
log length start to end: 2494secs (0.69hrs)
oplog first event time:  Mon Sep 12 2016 17:42:54 GMT+0200 (CEST)
oplog last event time:   Mon Sep 12 2016 18:24:28 GMT+0200 (CEST)
now:                     Mon Sep 12 2016 18:25:55 GMT+0200 (CEST)
```

El parámetro log lenght start to end nos indica la diferencia entre la entrada del oplog más antigua y la más moderna para darnos la idea de la ventana de replicación que disponemos antes de que un secundario tuviera que hacer una resincronización completa tras haber estado indispoible. Este parámetro variará en tanto en cuanto la instancia tenga más o menos actividad. A menor actividad, más diferencia habrá entre la primera y la última entrada del oplog. En el ejemplo anterior, la primera entrada mostrada en el oplog es de las 17:42 (CEST) y la última registrada es a las 18:24. Este comando se ejecutó a las 18:25:55

Hemos dejado algo más de dos horas la base de datos levantada un par de horas y hemos ejecutado una inserción. Si volvemos a mostrar la salida del comando :

```
 rs.printReplicationInfo()
```

```
 configured oplog size:   50MB
log length start to end: 10121secs (2.81hrs)
oplog first event time:  Mon Sep 12 2016 17:42:34 GMT+0200 (CEST)
oplog last event time:   Mon Sep 12 2016 20:31:15 GMT+0200 (CEST)
now:                     Mon Sep 12 2016 20:31:21 GMT+0200 (CEST)
```

Observamos que la venana de Oplog es de casi 3 horas teniendo en cuenta la actividad de la BBDD. Ahí reside la importancia de saber la ventana de replicación que disponemos para mantener un nodo secundario caído durante la operativa de mantenimiento.

Para llevar a cabo la aplicación de índices en modo rolling update sobre colecciones de una gran cardinalidad (varios millones de documentos) , seleccionamos uno de los nodos que ejerce el papel de secundario, para ello ejecutamos el comando rs.status()

```
training:PRIMARY> rs.status()
{
	"set" : "training",
	"date" : ISODate("2016-09-12T18:35:48.337Z"),
	"myState" : 1,
	"term" : NumberLong(3),
	"heartbeatIntervalMillis" : NumberLong(2000),
	"members" : [
		{
			"_id" : 0,
			"name" : "mongodb:27017",
			"health" : 1,
			"state" : 1,
			"stateStr" : "PRIMARY",
			"uptime" : 286,
			"optime" : {
				"ts" : Timestamp(1473705075, 1),
				"t" : NumberLong(3)
			},
			"optimeDate" : ISODate("2016-09-12T18:31:15Z"),
			"electionTime" : Timestamp(1473705074, 1),
			"electionDate" : ISODate("2016-09-12T18:31:14Z"),
			"configVersion" : 3,
			"self" : true
		},
		{
			"_id" : 1,
			"name" : "mongodb:27018",
			"health" : 1,
			"state" : 2,
```

```
                "stateStr" : "SECONDARY",
                "uptime" : 280,
                "optime" : {
                        "ts" : Timestamp(1473705075, 1),
                        "t" : NumberLong(3)
                },
                "optimeDate" : ISODate("2016-09-12T18:31:15Z"),
                "lastHeartbeat" :
ISODate("2016-09-12T18:35:46.563Z"),
                "lastHeartbeatRecv" :
ISODate("2016-09-12T18:35:47.423Z"),
                "pingMs" : NumberLong(0),
                "syncingTo" : "mongodb:27017",
                "configVersion" : 3
        },
        {
                "_id" : 2,
                "name" : "mongodb:27019",
                "health" : 1,
                "state" : 2,
                "stateStr" : "SECONDARY",
                "uptime" : 280,
                "optime" : {
                        "ts" : Timestamp(1473705075, 1),
                        "t" : NumberLong(3)
                },
                "optimeDate" : ISODate("2016-09-12T18:31:15Z"),
                "lastHeartbeat" :
ISODate("2016-09-12T18:35:46.563Z"),
                "lastHeartbeatRecv" :
ISODate("2016-09-12T18:35:47.492Z"),
                "pingMs" : NumberLong(0),
                "syncingTo" : "mongodb:27018",
                "configVersion" : 3
        }
    ],
    "ok" : 1
}
```

Las instancias levantadas en el puerto 27018 y 27019 son nodos secundarios. Vamos a comenzar con la instancia 27018.

Abrimos una nueva consola y nos conectamos a la instancia del puerto 27018

```
mongo --port 27018
```

Y procedemos a apagarla usando los siguientes comandos

```
use admin
db.shutDownServer()
exit
```
A continuación, modificamos el fichero de configuración de dicha instancia realizando dos tareas :

1. Cambiar el puerto de escucha
2. Comentar el bloque donde indicamos el nombre del replica set

Para ello ejecutamos
```
cd ~/Troubleshooting-Mongodb/Tema1/ReplicaSet
vim RS1_2.conf
```
En el fichero de configuración tocamos el parámetro marcado en negrita y comentamos con # al inicio de la línea de las opciones para el replica set:

systemLog:
  destination: file
  path: "/data/RS1_2/mongod.log"
  logAppend: true
storage:
  dbPath: /data/RS1_2
  journal:
    enabled: false
processManagement:
  fork: true
  pidFilePath: "/data/RS1_2/mongodb.pid"
net:
  port: **37018**
#replication:
 # oplogSizeMB: 50
 # replSetName: training

Guardamos el fichero y levantamos la instancia con esta nueva configuración
```
mongod -f RS1_2.conf
```

Entramos en la instancia ejecutando el comando
```
mongo --port 37018
```

Y creamos el índice
use training
db.formacion.createIndex({x:1})
{
	"createdCollectionAutomatically" : true,
	"numIndexesBefore" : 1,
	"numIndexesAfter" : 2,
	"ok" : 1
}
Una vez creado el índice, paramos la instancia y salimos de la consola de mongo

```
use admin
db.shutDownServer()
exit
```
Restauramos los valores originales del fichero de configuración:
```
vim RS1_2.conf
… [omitida parte del fichero]
net:
   port: 27018
replication:
   oplogSizeMB: 50
   replSetName: training
y volvemos a levantar la instancia
```
mongod -f RS1_2.conf
nos conectamos al primario o cualquier instancia del replica set, y observamos que el nodo ha sido añadido al replica set correctamente
mongo --host training/mongodb:27017,mongodb:27018,mongodb:27019
rs.status()

Si el nodo al que hemos aplicado aparece en estado SECONDARY, podemos repetir la tarea con el otro nodo secundario.
Por último, para aplicar esta operativa sobre el nodo primario, antes de parar la instancia, debemos ejecutar el comando :
rs.stepDown(300)
De esta forma, se producirá un nuevo proceso de elección de nodo primario y el que era nodo primario no podrá ser reelegido hasta que al menos pasen 300 segundos (5 minutos). Después de ejecutar este comando, realizamos la misma operativa que con los otros nodos secundarios.

Cambio de tamaño de Oplog

Al igual que el ejercicio anterior, vamos a realizar esta operativa en rolling upgrade.
Escogemos uno de los nodos con rol secundario y lo levantamos en standalone. Podemos usar el comando:

```
use admin
db.shutDownServer()
```

Como en el ejercicio anterior, levantamos la instancia comentando el bloque que indica la configuración del replica set y en un puerto de escucha distinto.

Nos conectamos a la instancia en standalone. Vamos a realizar una copia **de la última entrada del oplog** en una colección llamada temp. La colección de oplog se almacena en la base de datos local y se llama oplog.rs. Para ello ejecutamos:

```
use local
db.temp.drop() → (Esto nos borraría la colección temp en caso de
existir previamente)
db.temp.save( db.oplog.rs.find( { }, { ts: 1, h: 1 } ).sort(
{$natural : -1} ).limit(1).next() )
db.temp.find() → Así observamos la entrada que ha almacenado
dentro del oplog
```

Después procedemos a borrar el actual oplog

```
db.oplog.rs.drop()
```

A continuación, creamos un nuevo el nuevo Oplog. Uno de los parámetros a pasar es el tamaño del oplog en bytes. Si queremos crear un oplog de 2 GB podemos pasarle como tamaño 2*1024*1024*1024. Para nuestro ejemplo, vamos a crear un oplog de 300 MB. Ejecutamos el siguiente comando

```
use local
db.runCommand( { create: "oplog.rs", capped: true, size: (3 *
1024 * 1024) } )
```

Insertamos en la nueva colección del oplog, la última entrada del oplog antiguo ejecutando el siguiente comando

```
db.oplog.rs.save( db.temp.findOne() )
db.oplog.rs.find()
```

Apagamos la instancia
use admin
db.shutDownServer()

Y cambiamos el fichero de configuración para volver a añadirla al replica set y su puerto de escucha original. Una vez cambiado, levantamos la instancia, nos conectamos a cualquier

instancia y vemos que el nodo sobre el que hemos realizado la tarea de mantenimiento ha vuelto a formar parte del replica set ejecutando el comando rs.status()

Repetimos el proceso con el resto de nodos, dejando el nodo primario para el final. Recordar que antes de apagar el nodo primario, debemos forzar la elección de un nuevo nodo primario ejecutando el comando
`rs.stepDown(300)`

Compactación de datafiles

Para reclamar espacio en disco que pueda estar siendo desaprovechado debido a la fragmentación de los datafiles se puede realizar de dos maneras:

- Usando el comando db.repairDatabase()
- Borrando el contenido del dbpath de cada miembro del replica set y resincronizando los datos

La primera opción se trata de ejecutar el comando db.repairDatabase() desde la base de datos que queremos reparar. El comportamiento de este comando es que reescribe todos los datafiles partiendo de los datafiles de origen y elimina y compacta los documentos reclamando así el espacio en disco. Sin embargo para poder ejecutar este comando se necesita **el doble de espacio en disco duro, que el tamaño actual que ocupan los datafiles.** (Por ejemplo para una BBDD de 50 GB, necesitas al menos 100 GB)

La otra forma es resincronizando los datos desde otro miembro del replica set. Para realizar esta tarea hacemos lo siguiente. Vamos a cargar un dump de una base de datos donde vamos a provocar cierta fragmentación

Nos situamos en el directorio siguiente:
cd ~/Troubleshooting-Mongodb/Tema2/
y ejecutamos el siguiente comando
mongorestore --db troubleshooting dump/troubleshooting/
Nos contactamos al nodo primario y ejecutamos
use troubleshooting
db.stats()
{
	"db" : "troubleshooting",
	"collections" : 1,
	"objects" : 10000,
	"avgObjSize" : 2542,
	"dataSize" : 25420000,
	"storageSize" : 3756032,
	"numExtents" : 0,
	"indexes" : 1,
	"indexSize" : 94208,
	"ok" : 1

}

Y observamos el valor de los parámetros en negrita
Ejecutamos la siguiente query

db.padding.update({x:{$exists:true}},{$unset:{x:""}},{multi:true})

y volvemos a ejecutar db.stats(). Observamos que el dataSize ha disminuido, pero no ha pasado lo mismo con el parámetro storageSize.
Vamos a intentar recuperar dicho espacio. Para ello, paramos una de las instancias con rol secundario como hemos visto anteriormente y borramos el contenido de su dbpath correspondiente. Por ejemplo si paramos la instancia que escucha por el puerto 27018, según su fichero de configuración el dbpath está en /data/RS1_2
rm -rf /data/RS1_2
Volvemos a levantar la instancia y nos conectamos a la misma
mongo --port 27018
Ejecutamos los comandos:
use troubleshooting
db.stats()
Y comparamos la salida de ese mismo comando con la salida del comando desde el nodo primario.

Reconfiguración de cadena de replicación

Si hacemos la operativa anteriormente mencionada, tal vez nos interese que el nodo secundario sincronice de un nodo que no sea el primario. Para cambiar el nodo del que queremos sincronizar vamos a utilizar el comando db.syncFrom. Hay que tener en cuenta que este cambio **no es persistente** y se perderá si se da cualquiera de estos casos:

- La instancia donde hemos ejecutado el comando se reinicia
- Se cierra la conexión entre el nodo del que se está sincornizando
- La sincronización del nodo objetivo va 30 segundos por detrás de cualquier otro miembro del Replica set

Nos conectamos a un nodo secundario y ejecutamos el comando rs.status() y nos fijamos en el parámetro **syncingTo**
Ejecutamos el comando
rs.syncFrom("mongod:<puerto de otro miembro que sea secundario>")
Y volvemos a ejecutar el comando rs.status()

Reconfiguración de replica set cuando la mayoría de los nodos están caídos

ADVERTENCIA: EL USO DE ESTA OPERATIVA PUEDE CAUSAR QUE LA INFORMACIÓN DE LA BASE DE DATOS PUEDA CORROMPERSE O GENERAR INCONSISTENCIAS. ÚSESE ÚNICAMENTE ÚNICAMENTE COMO MEDIDA DE EMERGENCIA CRÍTICA Y COMO ÚLTIMO RECURSO

Si sufrimos una catástrofe que haga que la mayoría de nuestros nodos caigan y el tiempo de resolución pueda ser realmente elevado, podemos reconfigurar el replica set para dejar únicamente aquellos miembros supervivientes y volver a dar servicio.

El uso de esta operativa debe hacerse de manera responsable y se debe considerar que la consecuencia podría llegar a ser que los datos sean inconsistentes
Vamos a simular el comportamiento de una caída total. Apaga 2 de las 3 instancias del replica set con un kill al proceso de mongo o entrando en el servidor y ejecutando el comando db.shutdownServer()
entra en el nodo superviviente y ejecuta los siguientes comandos
cfg = rs.config()
cfgOrig=rs.config()
miembros=cfg.members[i] donde i es el número del índice del array de la instancia que sigue viva. Si nuestra instancia superviviente es la que corre en el puerto 27019, el comando debería ser :
miembros=cfg.members[2]
Después ejecutamos:
cfg.members=[miembros]
rs.reconfig(cfg,{force:true})

Volver a levantar las instancias paradas y volver a añadirlas al replica set para continuar el resto de laboratorios (observar ejercicios anteriores)

Tema 3 Operación de MongoDB

Obtención de índices y evaluación de uso

Vamos a restaurar un dump de una base de datos para evaluar el uso de sus índices. Para ello, ejecutamos los siguientes comandos:
cd ~/Troubleshooting-Mongodb/Tema3
mongorestore --port <puerto> --db test dump/test
Donde puerto es el puerto de escucha del nodo que es primario en el replica set.

Nos conectamos a la instancia con el comando mongo y nos posicionamos en la base de datos test
use test
Para ver un ejemplo de documento almacenado en la colección, se puede ejecutar el comando db.companies.findOne()
y obtenemos los índices de la colección companies
db.Companies.getIndexes()


```
[
	{
		"v" : 1,
		"key" : {
			"_id" : 1
		},
		"name" : "_id_",
		"ns" : "test.companies"
	},
	{
		"v" : 1,
		"key" : {
			"category_code" : 1,
			"name" : 1
		},
		"name" : "category_code_1_name_1",
		"ns" : "test.companies"
	},
	{
		"v" : 1,
		"key" : {
			"number_of_employees" : 1
		},
		"name" : "number_of_employees_1",
		"ns" : "test.companies"
	}
]
```


Los campos marcados en negrita corresponden a campos indexados en la base de datos
Para analizar el plan de ejecución de una query, usamos el comando explain al final de la consulta. Vamos a lanzar algunas querys y observar sus resultados.

Vamos a ver las siguientes querys usando el comando explain y ver sus resultados con el comando explain()
db.companies.find({name:"Skype"})

db.companies.find({category_code:"news"}

db.companies.find({name:"Skype"},{_id:0,category_code:1,name:1})

db.companies.find({category_code:"news"},{category_code:1,name:1})

db.companies.find({category_code:"news"},{_id:0,category_code:1,name:1})

db.companies.find({category_code:"news"},{_id:0,category_code:1,name:1}).sort({name:-1,category_code:-1})

Habilitación de profiling y querys a profiling

Cuando queremos averiguar qué querys están teniendo unos malos tiempos de respuesta, hacemos uso del profiling. El profiling se habilita a nivel de BBDD y genera una capped collection donde podemos analizar que querys están siendo más lentas, a qué colección corresponden si han producido bloqueos, etc.
Advertencia: el uso del profiling produce impacto en el rendimiento sobre la base de datos, por lo que se recomienda utilizar única y exclusivamente a modo de análisis de una incidencia.
para habilitarlo, basta con usar el comando db.setProfilingLevel(<nivel>,<threshold>)
donde:
<nivel> : 0 deshabilitado, 1 habilitado para querys que superen el threshold pasado por parámetro , 2 activado.
<threshold> el tiempo en milisegundos por el cual se almacenará la query en la colección del profiling

Para nuestra práctica, vamos a habilitar el profiling en la base de datos test para todas las querys que superen los 5 ms

use test
db.setProfilingLevel(1,5)

Y ejecutamos la siguiente query:
db.companies.find({category_code:"news"},{category_code:1,name:1}).sort({name:1})

Ahora vamos a habilitarlo para todas las querys
db.setProfilingLevel(2)
Y ejecutamos esta otra query
db.mensaje.insert({"mensaje":"Este insert aparecerá en el profiling"})

Y por último lo deshabilitamos con
db.setProfilingLevel(0)

Ahora podemos obtener todas las querys ordenadas de mayor a menor tiempo de ejecución usando la consulta :
db.system.profile.find().sort({millis:-1}).pretty()
Si se quieren consultar las 5 más recientes, se puede usar el comando:
show profile

Los campos interesantes a tener en cuenta son:
millis: milisegundos que ha tardado en ejecutarse la query
ns: namespace donde ha sucedido la query (base de datos y colección)
op : el tipo de operación que se ha ejecutado
query: la query que se ha ejecutado

Tema 4

Troubleshooting de instancias que no arrancan

Vamos a intentar simular un entorno con problemas para arrancar la instancia
ejecutamos los siguientes comando
cd ~/Troubleshooting-Mongodb/Tema4
tar -xzvf tema4.tar.gz -C /data
después intentamos levantar la instancia
mongod --dbpath /data/tema4
Intentar identificar el error y ver entre las posibles opciones mostradas más abajo, cual es la causa raíz de que no arranque

Paramos la instancia con control +C y ahora levantamos con el siguiente comando

mongod --dbpath /data/tema4 --logpath /var/log/mongod.log

Por último, una vez levantada la instancia, tratar de levantar dos veces la base de datos usando el comando
mongod --dbpath /data/tema4; mongod --dbpath /data/tema4

Posibles causas raíz del problema:

Existe fichero mongod.lock que no ha sido borrado
El puerto está en uso
No tiene permisos suficientes para escribir en el directorio de dbpath o logs
El storage engine configurado no es el correcto