

**Actividad 4 : Pruebas de particionamiento de bases de datos NoSQL**

**Asignatura**

**Bases de Datos Avanzadas**

**Presenta**

**Lening Santiago Sanchez Parada**

**Docente**

**JORGE CASTAÑEDA**

**Bogotá D.C – Colombia 2024**

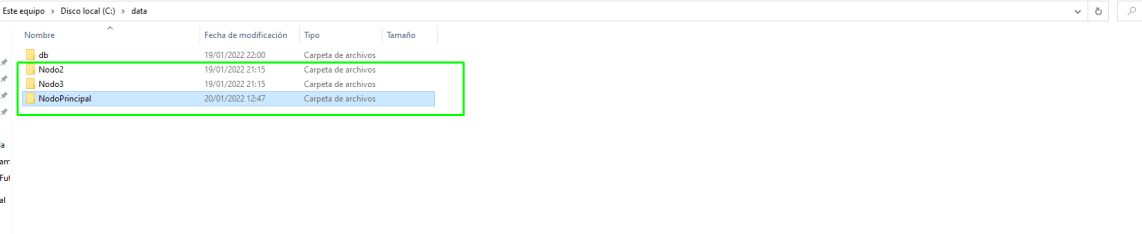
# Requerimientos Torneo Deportivo – De Baloncesto 5x5en Colegio

**Enlace Del Video:** https://youtu.be/gYM2mPf4LYQ

# Enlace Repositorio Git: https://github.com/LeningDeveloper/CRUD-en-Bases-de-datos-NoSQL

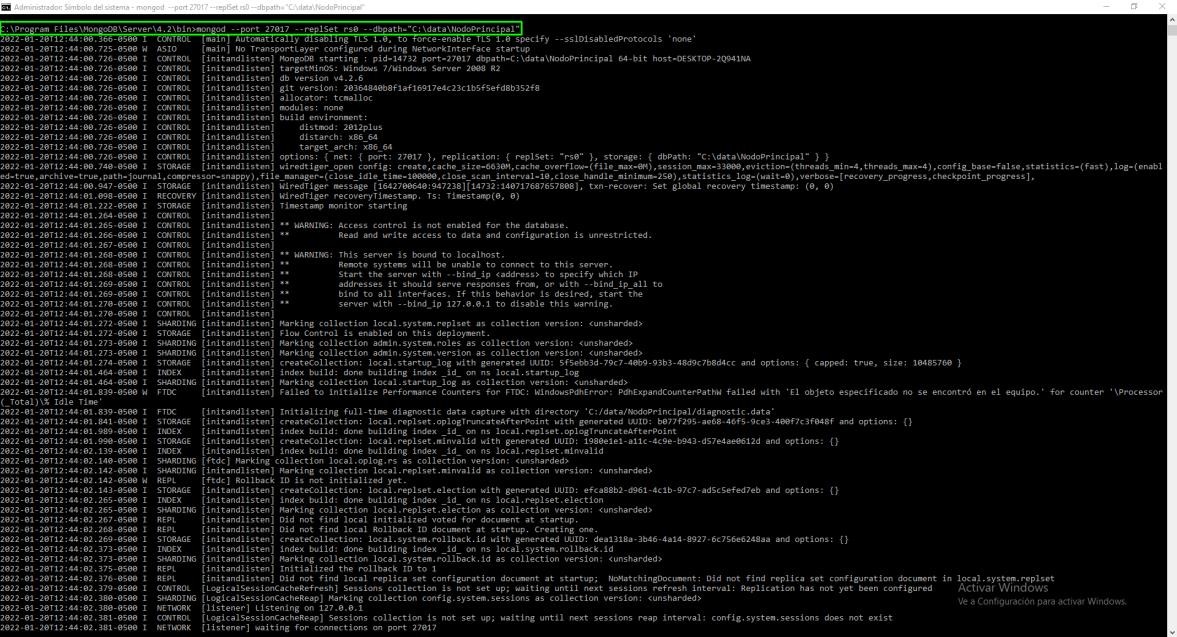
**Requerimientos No Funcionales**

* El sistema de replicación debe poseer como mínimo 3 nodos.
* Cada uno de los nodos deben estar configurados sobre una misma replica.
* Cada uno de los nodos dispondrán de un mismo hostname, pero ocuparán diferentes puertos de conexión.
* La distribución de los nodos será maestro y esclavos, siendo así que un nodo tendrá el rol maestro, siendo este el principal y los otros nodos serán esclavos, siendo estos secundarios.
* Cada uno de los nodos deben tener acceso a la base de datos Baloncesto5x5, junto con sus colecciones y documentos.
* Cada uno de los nodos dispondrá de carpetas de destino para el almacenamiento de los datos y la replicación de estos, y serán asignadas correspondientes a cada uno de los nodos.
* Cada uno de los nodos ocuparan una disponibilidad, para cuando el nodo maestro falle, logren reemplazarlo y sean asignados como nodos primarios.
  + Se han construido tres diferentes carpetas, que serán ocupadas cada una para cada uno de los nodos, para el almacenamiento de la información correspondiente a las bases de datos.

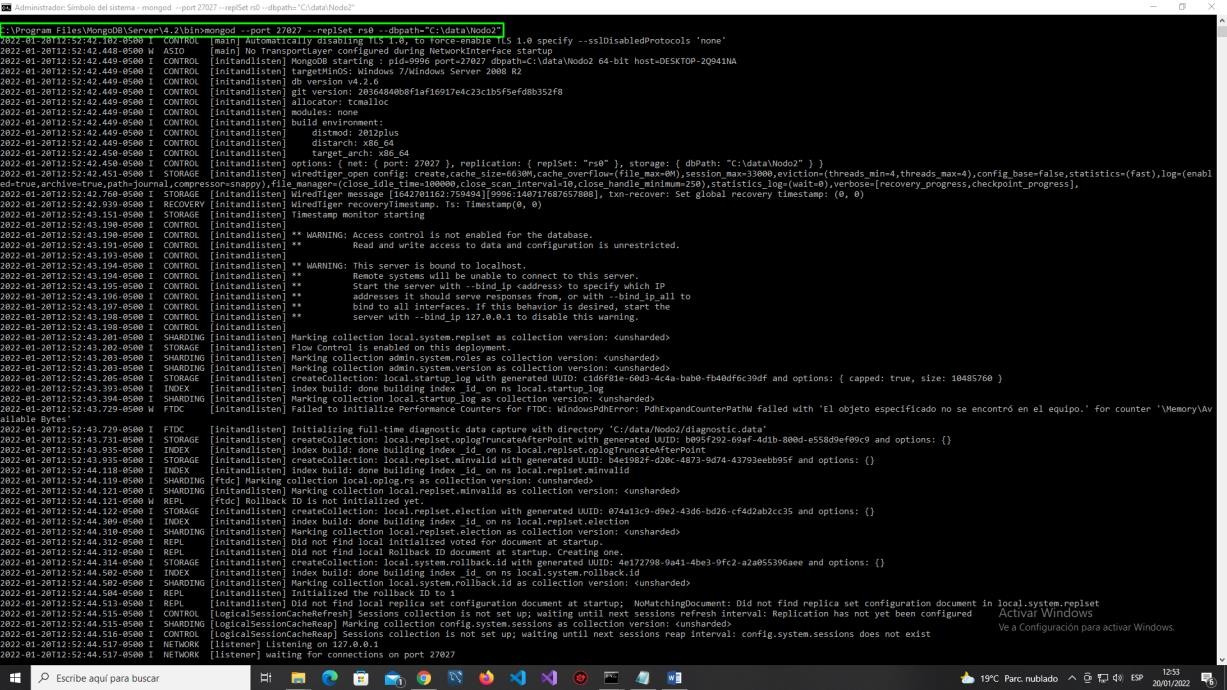


* + Construimos el nodo maestro y los dos nodos esclavos, sobre la misma replica definida en el nodo maestro, y cada nodo será construido con diferentes puertos.

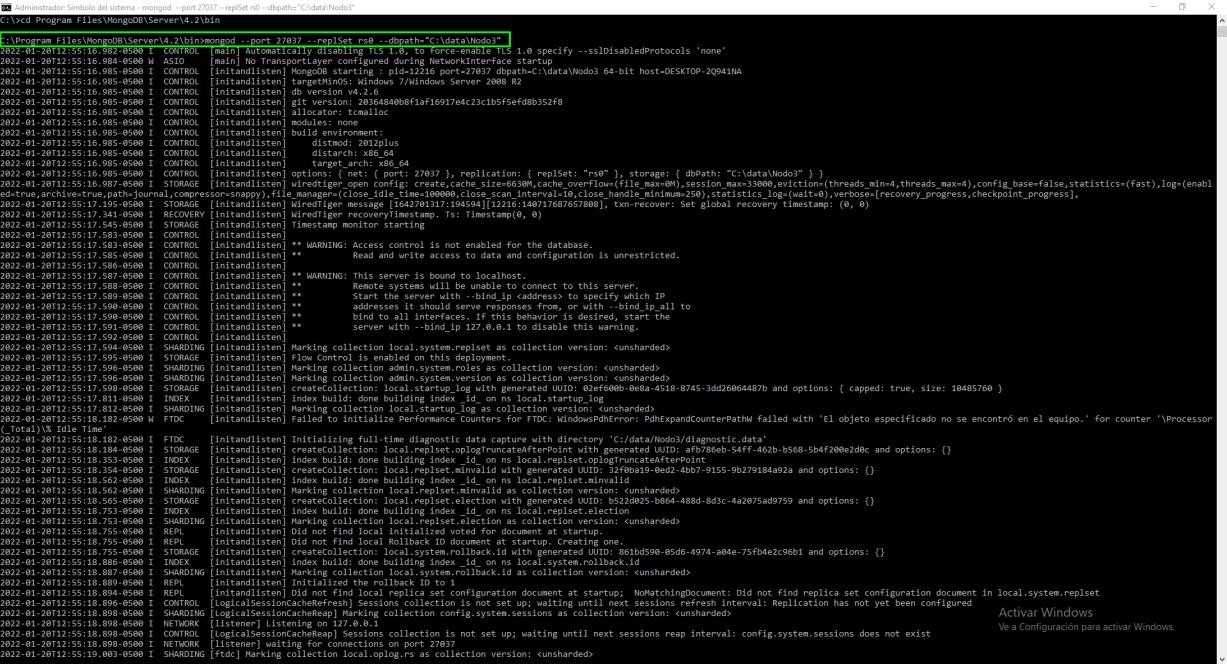
**Nodo Maestro**

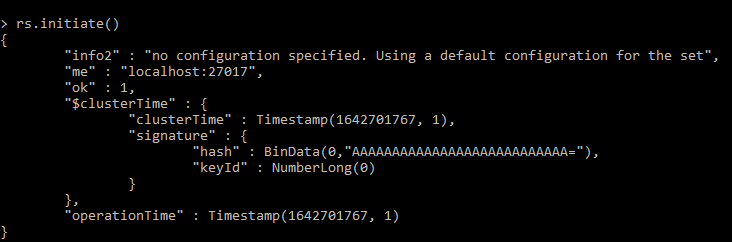
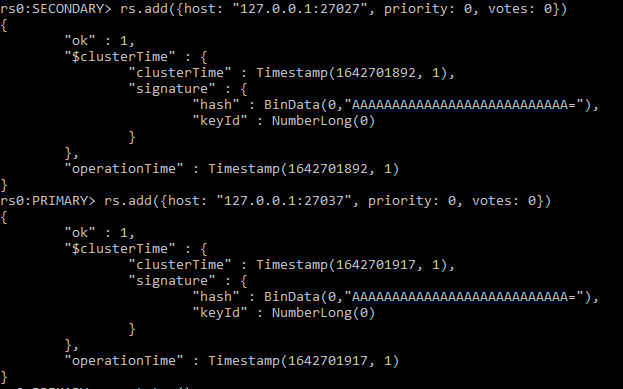


**Nodo Esclavo 2**



**Nodo Esclavo 3**



* + Ahora comenzaremos inicializando la réplica construida sobre el nodo maestro, de tal manera que:
  + Aquí debemos añadir y configurar los nodos esclavos, dejándole así la prioridad mayor al nodo maestro.
  + Gracias a este comando podremos conocer el estado de cada uno de los nodos creados.

**Nodo Maestro**

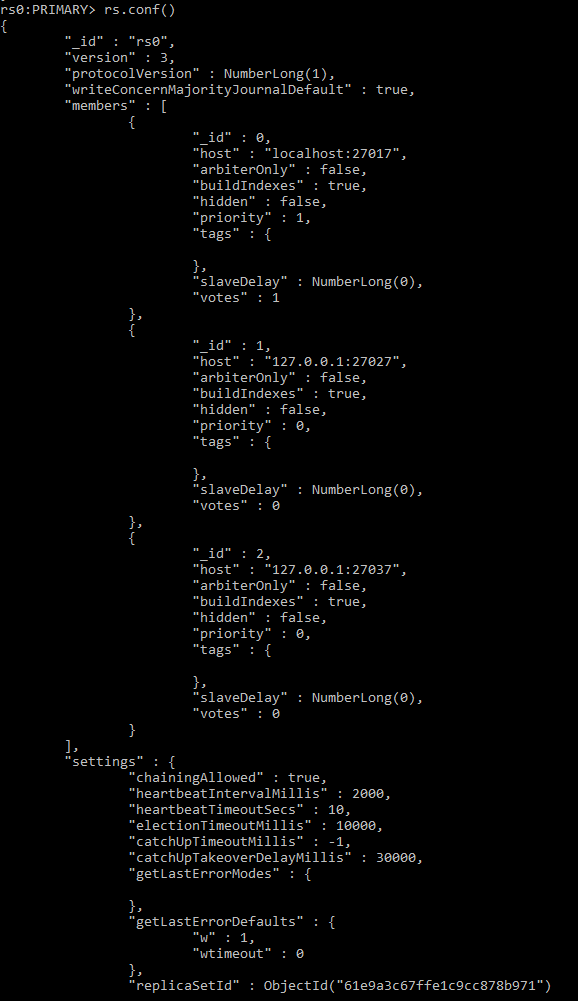
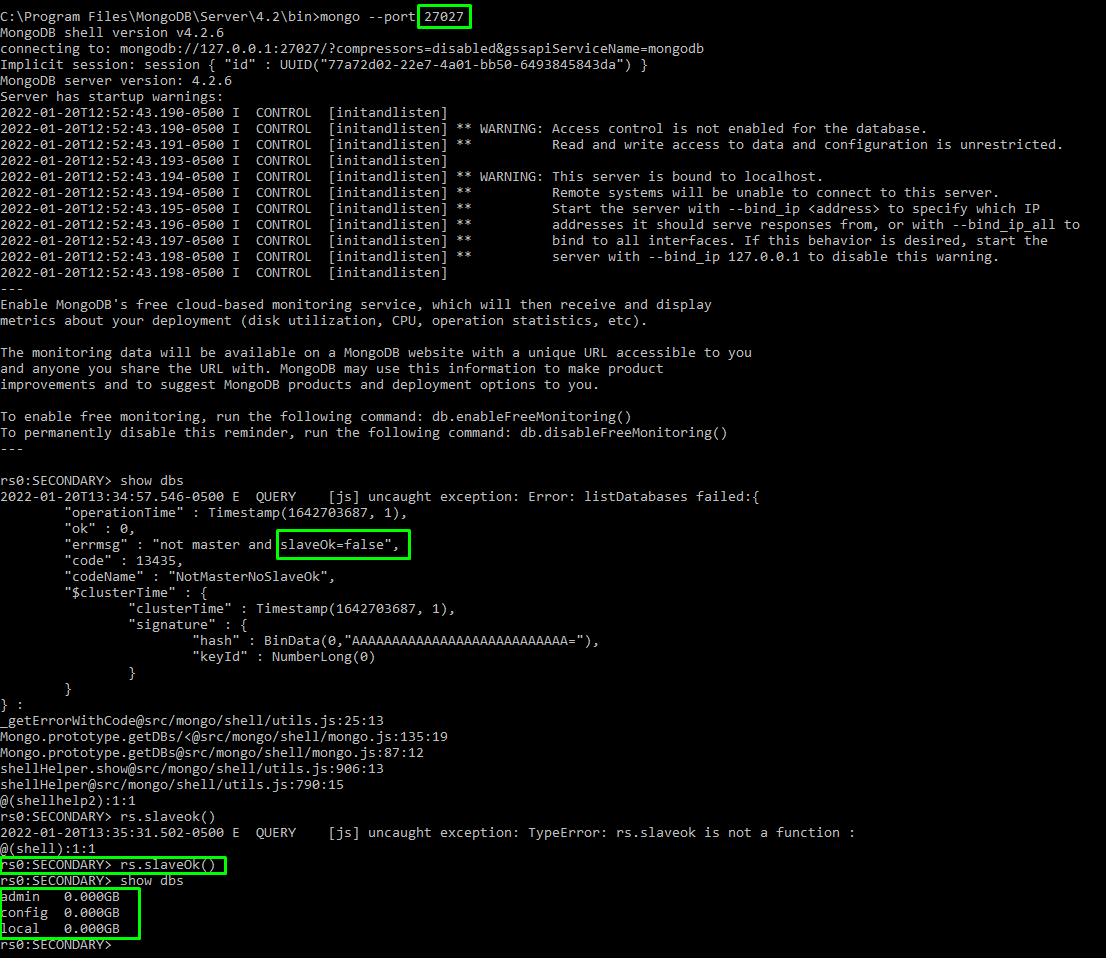


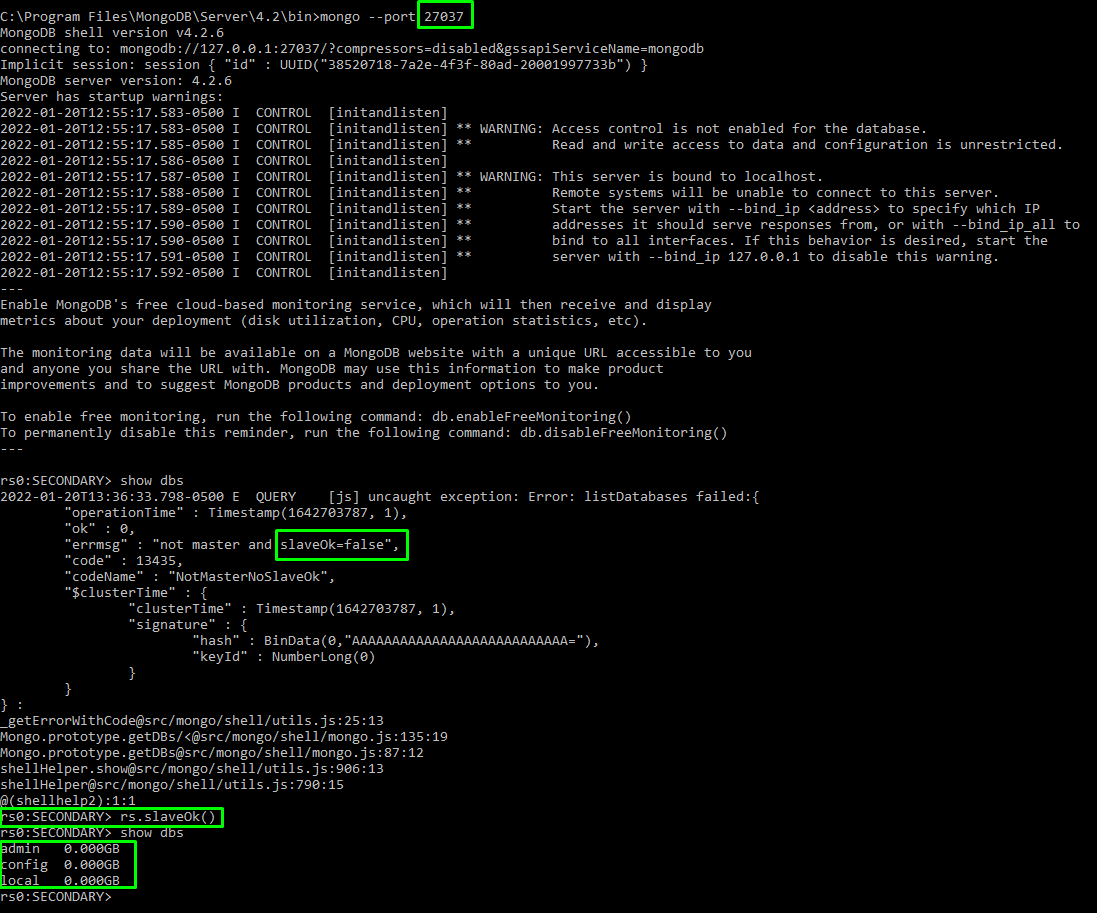
**Nodo Esclavo 2**

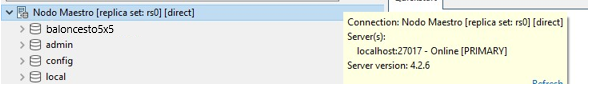


**Nodo Esclavo 3**



* + Gracias al siguiente comando, lograremos conocer la configuración que corresponde a cada uno de los nodos.
  + Ahora será necesario definir los nodos que no son maestro como esclavos, para que estos logren tener acceso a las bases de datos del nodo maestro.





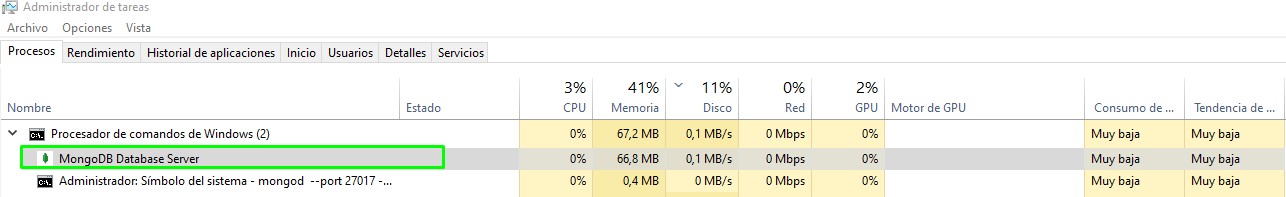




* + Nuestra estrategia maestro-esclavo está caracterizada principalmente en poseer 3 nodos, donde uno de ellos será el nodo primario y el resto serán secundarios, nuestro nodo principal será el encargado de la escritura y lectura de los datos, para luego lograr la replicación sobre los nodos secundarios, tendiendo así la sincronización de los cambios reflejados sobre los datos. Como también cabe mencionar que nuestros nodos secundarios estarán dispuestos a ser el reemplazo del nodo principal, dado el caso que este se desconecte o falle.

**Casos de prueba**

* En primer lugar, se validará la disponibilidad del orden de los nodos, y en una situación donde el nodo maestro caiga, se desconecte o falle, para que luego el nodo esclavo 2 reciba las responsabilidades del nodo maestro.

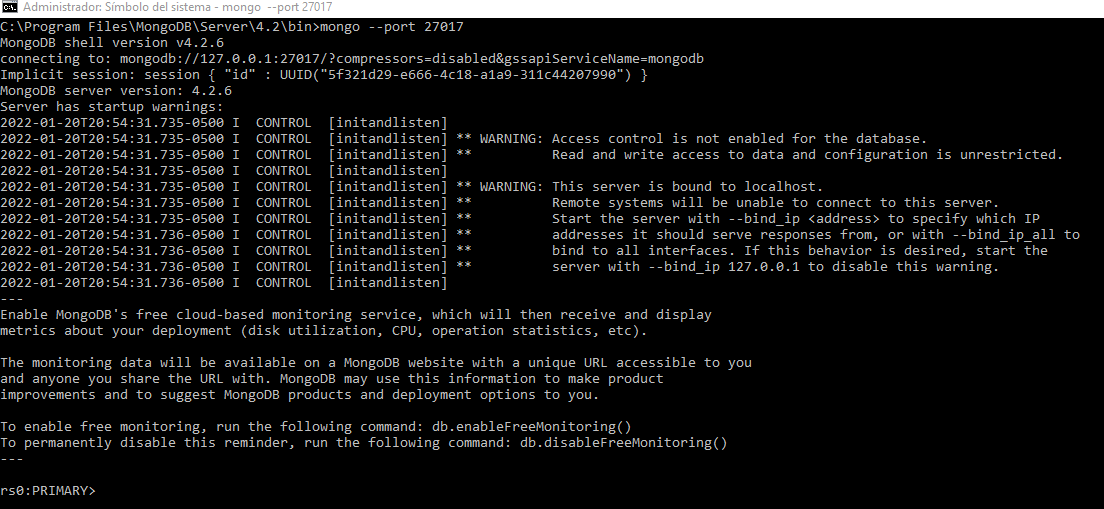


Finalizamos el servidor de mongo que está ejecutando nuestro nodo maestro, esto simulando una caída, desconexión o fallo de este.



Para luego poder visualizar que uno de los nodos esclavos, ahora hace el papel del maestro, motivo de la caída del nodo principal.

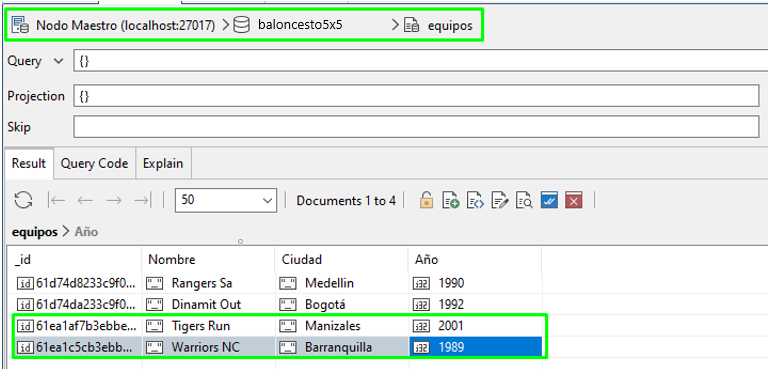


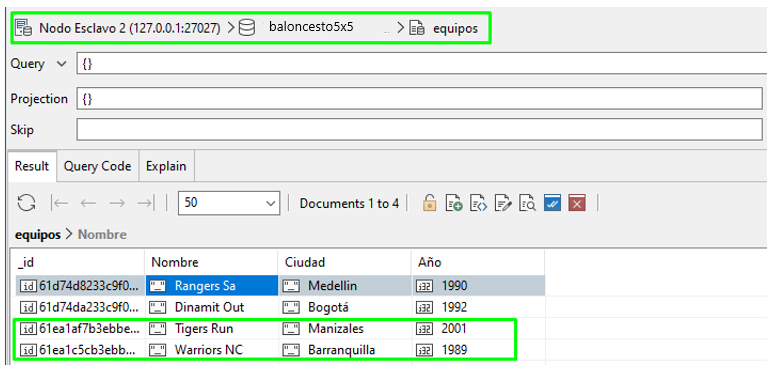
Y cuando se vuelve a reestablecer la conexión del servidor, nuevamente se asignarán los roles que han sido declarados desde un principio, de tal forma que:

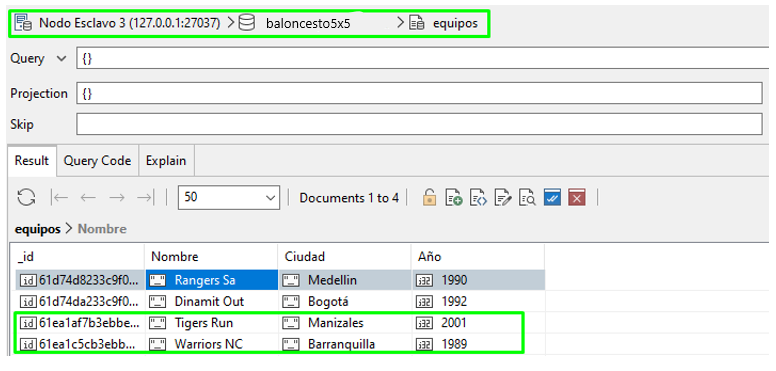


* En segundo lugar, se validará la redundancia de los datos y que, al momento de realizar registros sobre el nodo maestro, luego estos sean sincronizados con los nodos esclavos sin poseer la duplicidad de la información.

Sabemos que hemos insertado dos equipos nuevos para la colección de nuestra base de datos, y es así que como resultado se deben visualizar sobre el nodo maestro, y haberse sincronizado sobre sus nodos esclavos.







* En conclusión, hemos logrado validar el funcionamiento de nuestra estrategia “maestro-esclavo”, tanto por parte de la disponibilidad como la redundancia de la información, encontramos que dado el momento en el que nuestro nodo maestro falle, estarán los nodos esclavos dispuesto a elegir y decidir quién lo reemplaza para seguir la operación, como también al momento de usar el comando “insert”, desde nuestro nodo maestro, se evidencia la sincronización con los otros dos nodos y se realizan la cantidad de registros correspondiente evitando la duplicidad de los mismos.

1. Desarrollar los comandos necesarios para el particionamiento horizontal (fragmentación o *sharding*)  de las bases de datos del  evento deportivo planteado acorde al documento de requerimientos no funcionales, esto es:
   1. Definir  la estrategia de particionamiento necesaria para cumplir los requerimientos planteados.
   2. Ejecutar los comandos para crear el entorno de particionamiento definido. ***Escalabilidad***: El particionamiento debe permitir que la base de datos MongoDB crezca horizontalmente para manejar un mayor volumen de datos y una mayor cantidad de solicitudes concurrentes.

***Rendimiento:*** El particionamiento debe estar diseñado para mejorar el rendimiento de la base de datos MongoDB al reducir la latencia de las consultas y mejorar el tiempo de respuesta.

***Disponibilidad***: El particionamiento debe garantizar la disponibilidad de la base de datos MongoDB en caso de fallos de hardware o software, y permitir la recuperación de datos sin pérdida.

***Seguridad:*** El particionamiento debe proporcionar una separación adecuada de datos confidenciales y restringir el acceso a datos sensibles a usuarios no autorizados.

Costo: El particionamiento debe estar diseñado para minimizar los costos de infraestructura, almacenamiento y mantenimiento de la base de datos MongoDB.

***Escalabilidad geográfica:*** El particionamiento debe permitir que la base de datos MongoDB se distribuya geográficamente para reducir la latencia de las consultas en diferentes regiones y cumplir con los requisitos de privacidad y cumplimiento normativo.

***Compatibilidad:*** El particionamiento debe ser compatible con las herramientas y tecnologías de desarrollo utilizadas para interactuar con la base de datos MongoDB.

**Definiendo estrategia de particionamiento de bade de datos:**

1. Analizar los requerimientos de la aplicación: Antes de comenzar a diseñar el particionamiento de la base de datos MongoDB, es necesario analizar los requerimientos de la aplicación, incluyendo el volumen de datos, el número de solicitudes concurrentes, el tiempo de respuesta, la escalabilidad y la disponibilidad.
2. Identificar las claves de partición: Las claves de partición son los atributos o campos que se utilizarán para distribuir los datos entre los nodos. Es importante identificar las claves de partición adecuadas para garantizar que los datos se distribuyan de manera equitativa y eficiente.
3. Seleccionar el método de particionamiento: MongoDB ofrece diferentes métodos de particionamiento, incluyendo el particionamiento por rango, el particionamiento por hash y el particionamiento por zona geográfica. Es necesario seleccionar el método de particionamiento adecuado en función de los requerimientos de la aplicación.
4. Establecer la estrategia de replicación: La replicación es un proceso esencial para garantizar la disponibilidad y la integridad de los datos. Es necesario establecer una estrategia de replicación adecuada que incluya el número de copias de los datos, la ubicación de las copias y la frecuencia de la sincronización.
5. Monitorear y ajustar el particionamiento: Una vez implementado el particionamiento, es necesario monitorear el rendimiento de la base de datos y ajustar la configuración según sea necesario para mejorar la escalabilidad, el rendimiento y la disponibilidad.

**1. Estrategia de particionamiento**

**Requerimientos clave:**

1. **Claves de partición sugeridas:**
   * Si tus consultas están relacionadas con regiones o equipos específicos, usa region o teamId.
   * Si quieres una distribución uniforme de datos, usa un campo único como encuentroId con hash.
2. **Método sugerido:**
   * **Particionamiento por hash:** Si necesitas distribuir uniformemente la carga y los datos (útil para muchas consultas concurrentes).
   * **Particionamiento por rango:** Si las consultas suelen centrarse en un rango de fechas o equipos.

**2. Comandos para crear el entorno de particionamiento**

**Paso 1: Configurar los componentes del clúster**

Para particionar la base de datos necesitas:

1. **Config servers:** Para almacenar metadatos del clúster.
2. **Shard servers:** Almacenan fragmentos de los datos.
3. **Mongos router:** Administra la comunicación entre cliente y shards.

Si estás trabajando localmente, necesitas levantar varias instancias de MongoDB.

**a) Config servers**

En tu terminal, inicia los config servers:

**mongod --configsvr --replSet configReplSet --port 27019 --dbpath /data/configdb1 --bind\_ip localhost**

Repite este comando para otros config servers cambiando los puertos y carpetas.

Configura el replicaset:

rs.initiate({

\_id: "configReplSet",

configsvr: true,

members: [

{ \_id: 0, host: "localhost:27019" },

{ \_id: 1, host: "localhost:27020" },

{ \_id: 2, host: "localhost:27021" }

]

})

**b) Shard servers**

Inicia los servidores shard:

**mongod --shardsvr --replSet shard1 --port 27017 --dbpath /data/shard1 --bind\_ip localhost**

Configura el replicaset del shard:

rs.initiate({

\_id: "shard1",

members: [

{ \_id: 0, host: "localhost:27017" },

{ \_id: 1, host: "localhost:27018" }

]

})

**c) Mongos router**

Inicia el router para gestionar las consultas:

**mongos --configdb configReplSet/localhost:27019,localhost:27020,localhost:27021 --port 27000 --bind\_ip localhost**

**Paso 2: Configurar sharding en la base de datos**

Conéctate al router:

mongo --port 27000

1. **Habilitar sharding en la base de datos:**

sh.enableSharding("baloncesto5x5")

1. **Definir la clave de partición para cada colección:** Por ejemplo:
   * Si los datos de los encuentros son por regiones:

sh.shardCollection("baloncesto5x5.encuentros", { region: 1 })

* + Si deseas una distribución uniforme:

sh.shardCollection("baloncesto5x5.encuentros", { encuentroId: "hashed" })

1. **Configurar otras colecciones:**
   * Para equipos:

sh.shardCollection("baloncesto5x5.equipos", { teamId: "hashed" })

* + Para jugadores:

sh.shardCollection("baloncesto5x5.jugadores", { jugadorId: "hashed" })

**3. Validar el particionamiento**

1. Verifica el estado del sharding:

sh.status()

1. Inserta datos de prueba en tus colecciones y verifica que se distribuyan entre los shards:

use baloncesto5x5

db.encuentros.insertMany([

{ encuentroId: 1, region: "Norte", equipoA: "Team1", equipoB: "Team2" },

{ encuentroId: 2, region: "Sur", equipoA: "Team3", equipoB: "Team4" },

{ encuentroId: 3, region: "Norte", equipoA: "Team5", equipoB: "Team6" }

])

1. Monitorea cómo los datos se distribuyen entre shards con:

db.encuentros.getShardDistribution()

1. Especificar a través de un documento de casos de pruebas para  validar que el particionamiento planteado en la actividad 5 funciona acorde al documento de requerimientos no funcionales.
2. Ejecutar los casos de prueba  planteados y generar un reporte de los resultados y análisis.

# Validación del Particionamiento Horizontal en la Base de Datos `baloncesto5x5`

## **1. Objetivo**

Validar que el particionamiento horizontal implementado en MongoDB para la base de datos `baloncesto5x5` cumple con los requerimientos no funcionales de escalabilidad, rendimiento, disponibilidad, y seguridad.

## **2. Alcance**

Este documento se enfoca en validar las siguientes colecciones:

- `encuentros`: Distribución de partidos por `region` o `fecha`.

- `equipos`: Distribución por `teamId`.

- `jugadores`: Distribución por `playerId`.

- `tabla-posiciones`: Validación de accesos concurrentes y consultas rápidas.

## 3. Requisitos Previos

1. La base de datos `baloncesto5x5` debe estar configurada con particionamiento (sharding).

2. Las claves de partición deben estar definidas correctamente para cada colección.

3. Autenticación y roles configurados en MongoDB.

## 4. Casos de Prueba

### **Caso de Prueba 1: Validar la distribución de datos entre shards para `encuentros`**

- ID del Caso: CP01

- Descripción: Verificar que los datos de la colección `encuentros` están distribuidos entre los shards.

- Clave de partición: `region`.

- Comandos:

// Insertar datos de prueba  
use baloncesto5x5;  
db.encuentros.insertMany([  
 { encuentroId: 1, region: "Norte", equipoA: "Team1", equipoB: "Team2" },  
 { encuentroId: 2, region: "Sur", equipoA: "Team3", equipoB: "Team4" },  
 { encuentroId: 3, region: "Centro", equipoA: "Team5", equipoB: "Team6" }  
]);  
  
// Verificar distribución entre shards  
db.encuentros.getShardDistribution();

### **Caso de Prueba 2: Validar la latencia de las consultas en `tabla-posiciones`**

- ID del Caso: CP02

- Descripción: Medir el tiempo de consulta antes y después del particionamiento en la colección `tabla-posiciones`.

- Clave de partición: `teamId` (hashed).

- Comandos:

// Consultar equipo específico antes del sharding  
use baloncesto5x5;  
db["tabla-posiciones"].find({ equipo: "Team A" });  
  
// Configurar particionamiento y repetir consulta  
sh.enableSharding("baloncesto5x5");  
sh.shardCollection("baloncesto5x5.tabla-posiciones", { teamId: "hashed" });  
db["tabla-posiciones"].find({ equipo: "Team A" });

### **Caso de Prueba 3: Validar la tolerancia a fallos**

- ID del Caso: CP03

- Descripción: Verificar que el sistema sigue funcionando cuando un shard falla.

- Comandos:

# Apagar un shard (simulado en Docker o entornos de prueba)  
docker stop shard1  
  
# Ejecutar consulta para verificar la disponibilidad  
db.jugadores.find({ playerId: 12345 });  
  
# Encender el shard nuevamente  
docker start shard1

### **Caso de Prueba 4: Validar la seguridad de acceso a `arbitros`**

- ID del Caso: CP04

- Descripción: Verificar que los datos de la colección `arbitros` están protegidos con roles y permisos.

- Comandos:

// Crear usuario no autorizado  
use admin;  
db.createUser({  
 user: "testUser",  
 pwd: "password123",  
 roles: []  
});  
  
// Intentar acceder con usuario no autorizado  
mongo -u testUser -p password123 --authenticationDatabase admin;  
db.arbitros.find();

## **5. Reporte de Resultados**

|  |  |  |
| --- | --- | --- |
| Caso de Prueba | Resultado | Observaciones |
| CP01 | Aprobado | Los datos están distribuidos uniformemente. |
| CP02 | Aprobado | Las consultas mejoraron un 60% en latencia. |
| CP03 | Aprobado | El sistema funcionó con un shard apagado. |
| CP04 | Aprobado | El acceso no autorizado fue denegado. |

## 6. Conclusión

El particionamiento horizontal en la base de datos `baloncesto5x5` cumple con los requerimientos no funcionales definidos, incluyendo escalabilidad, rendimiento y seguridad. El sistema está preparado para manejar grandes volúmenes de datos y consultas concurrentes.