Ejecución del Proyecto: Red Social de Viajes

Este documento describe los pasos correctos para levantar el clúster shardeado de MongoDB, cargar los datos sintéticos y verificar la correcta distribución de datos.

Requisitos Previos

- Tener instalado Docker y Docker Compose.
- Tener acceso a una terminal Bash. (Linux y MacOS incluyen Bash de forma predeterminada. En Windows, se recomienda instalar Git Bash).
- Abrir una terminal Bash (o Git Bash en Windows) en el directorio del proyecto.

Preparación Inicial

1. Dar permisos de ejecución a los scripts:

```
chmod +x setup_cluster.sh
```

```
chmod +x verify_cluster.sh
```

Pasos de Ejecución

1. Inicialización del Cluster

Ejecutar el siguiente comando para configurar el clúster:

```
./setup_cluster.sh
```

Este script:

- Configura los Replica Sets.
- Agrega los Shards.
- Crea las colecciones `usuarios`, `posts` y `comentarios`.
- Configura el Sharding correspondiente.

2. Construcción y Ejecución del Generador de Datos

Luego, construir la imagen del generador de datos y ejecutarlo:

```
# Construir y levantar el data-generator
docker-compose up --build data-generator
```

Debe esperar unos minutos ya que la inserción puede tardar un tiempo.

Esto insertará aproximadamente:

- 5000 usuarios.
- 10 000 posts.
- 50 000 comentarios.

3. Verificación Final del Cluster

Finalmente, ejecutar el script de verificación para comprobar el estado del cluster y la distribución de los datos:

```
./verify_cluster.sh
```

Consulta de Datos Manual (Opcional)

Para visualizar algunos documentos insertados:

```
# Ver 5 comentarios
docker exec -it mongos1 mongosh --eval
'db.getSiblingDB("travel_social").comentarios.find().limit(5).pretty()'
```

```
# Ver 5 usuarios
docker exec -it mongos1 mongosh --eval
'db.getSiblingDB("travel_social").usuarios.find().limit(5).pretty()'
```

```
# Ver 5 posts
docker exec -it mongos1 mongosh --eval
'db.getSiblingDB("travel_social").posts.find().limit(5).pretty()'
```

Consulta de Datos mediante script de Python

Requisitos:

- Tener Python 3.9 o superior instalado
- Instalar la dependencia de `pymongo`

```
pip install pymongo
```

¿Cómo correr el script?

1. Asegúrese de que el clúster de MongoDB esté levantado, especialmente el contenedor `mongos1`
2. Ejecute el siguiente script

```
python query_demo.py
```

> El script intentará conectarse al clúster de MongoDB mediante múltiples routers mongos (mongos1, mongos2, mongos3).

Asegúrese de que todos los servicios estén accesibles y que los puertos estén correctamente expuestos, para garantizar alta disponibilidad en caso de que alguno no responda.

¿Qué cosas se pueden hacer?

El menú intentará las siguientes funciones:

- Ver posts de un usuario.
- Ver comentarios de un usuario.
- Ver comentarios de un post.
- Buscar un usuario por nombre.

Limpieza (opcional)

Para detener y eliminar todos los contenedores y volúmenes:

```
docker-compose down -v
```

Pruebas de Failover (Opcional)

Se puede simular una caída del nodo primario de un Replica Set y observar la recuperación automática:

1. Identificar el nodo primario ejecutando:

```
docker exec -it mongors1n1 mongosh --eval 'rs.status()'
```

Busque en la salida algo como:

```
"stateStr" : "PRIMARY"
```

y verá el nombre del contenedor que es el primario, por ejemplo `mongors1n1`

2. Detener el nodo primario una vez identificado

```
docker stop nombre_del_contenedor_primario
```

3. Esperar una segundos para permitir la elección de un nuevo nodo primario.

4. Consultar el estado en otro nodo secundario:

```
docker exec -it mongors1n2 mongosh --eval 'rs.status()'
```

Debería observarse que el contenedor detenido aparece como "(not reachable/healthy)" y que otro nodo ahora es PRIMARY.

5. Reiniciar el contenedor detenido:

```
docker start nombre_del_contenedor_primario
```

6. Verificar nuevamente el estado:

```
docker exec -it mongors1n2 mongosh --eval 'rs.status()'
```

El contenedor reincorporado debería aparecer como SECONDARY.

Esta prueba demuestra que el clúster se mantiene disponible incluso ante fallos de nodos individuales.