PRÁCTICA GUIADA: Arquitectura Distribuida en MongoDB

Del Standalone al Sharded Cluster - Experimentación Práctica

Objetivos de la práctica

Al finalizar esta práctica, serás capaz de:

1. Configurar un Replica Set desde cero
2. Simular un failover y observar la elección automática
3. Monitorizar el lag de replicación
4. Configurar un Sharded Cluster completo
5. Observar cómo se distribuyen los datos entre shards
6. Comparar el rendimiento de consultas targeted vs broadcast

Requisitos previos

- Docker y Docker Compose instalados
- Terminal/shell disponible
- mongosh instalado localmente (o usar el de Docker)
- Navegador web (para visualizar métricas opcionales)

PARTE 1: Configuración de un Replica Set

Paso 1.1: Preparar el entorno con Docker

Vamos a crear un Replica Set de 3 nodos usando Docker Compose.

Crear el directorio de trabajo:

```
mkdir -p ~/mongodb-practica/replica-set
cd ~/mongodb-practica/replica-set
```

Crear `docker-compose.yml`:

```
version: '3.8'

services:
  mongo1:
    image: mongo:7.0
    container_name: mongo1
    command: mongod --replSet rsLab --port 27017 --bind_ip_all
    ports:
      - "27017:27017"
    volumes:
```

```
      - mongo1_data:/data/db
      - mongo1_config:/data/configdb
    networks:
      - mongo-network
    healthcheck:
      test: echo 'db.runCommand("ping").ok' | mongosh --quiet
      interval: 10s
      timeout: 5s
      retries: 3

  mongo2:
    image: mongo:7.0
    container_name: mongo2
    command: mongod --replSet rsLab --port 27017 --bind_ip_all
    ports:
      - "27018:27017"
    volumes:
      - mongo2_data:/data/db
      - mongo2_config:/data/configdb
    networks:
      - mongo-network
    healthcheck:
      test: echo 'db.runCommand("ping").ok' | mongosh --quiet
      interval: 10s
      timeout: 5s
      retries: 3

  mongo3:
    image: mongo:7.0
    container_name: mongo3
    command: mongod --replSet rsLab --port 27017 --bind_ip_all
    ports:
      - "27019:27017"
    volumes:
      - mongo3_data:/data/db
      - mongo3_config:/data/configdb
    networks:
      - mongo-network
    healthcheck:
      test: echo 'db.runCommand("ping").ok' | mongosh --quiet
      interval: 10s
      timeout: 5s
      retries: 3

volumes:
  mongo1_data:
  mongo1_config:
  mongo2_data:
  mongo2_config:
  mongo3_data:
  mongo3_config:

networks:
```

```
  mongo-network:
    driver: bridge
```

Iniciar los contenedores:

```
docker-compose up -d
```

Verificar que están corriendo:

```
docker ps
```

Deberías ver 3 contenedores: `mongo1`, `mongo2`, `mongo3`.

Paso 1.2: Inicializar el Replica Set

Conectar a mongo1:

```
docker exec -it mongo1 mongosh
```

Dentro del shell de MongoDB, inicializar el Replica Set:

```
rs.initiate({
  _id: "rsLab",
  members: [
    { _id: 0, host: "mongo1:27017", priority: 2 },
    { _id: 1, host: "mongo2:27017", priority: 1 },
    { _id: 2, host: "mongo3:27017", priority: 1 }
  ]
})
```

Esperar unos 10-20 segundos y verificar el estado:

```
rs.status()
```

📝 **EJERCICIO 1.1:**

> Observa la salida de `rs.status()` e identifica:
>
> - ¿Cuál es el nodo PRIMARY?
> - ¿Cuál es el `electionTime` del primario?
> - ¿Cuál es el `syncSourceHost` de cada secundario?

Script para monitorizar en tiempo real:

```
// Guardar este script como monitor.js
function monitorRS() {
  while (true) {
    print("\n=== ESTADO DEL REPLICA SET ===");
    print("Timestamp: " + new Date().toISOString());

    rs.status().members.forEach(m => {
      let lag = m.optimeDate && m.state === 2
        ? Math.round((new Date() - m.optimeDate) / 1000)
        : 0;

      print(`${m.name.padEnd(20)} | ${m.stateStr.padEnd(12)} | Lag:
${lag}s`);
    });

    sleep(5000);  // Actualizar cada 5 segundos
  }
}

// Ejecutar
monitorRS();
```

Ejecutar el script:

```
docker exec -it mongo1 mongosh --file /dev/stdin < monitor.js
```

Paso 1.3: Insertar datos de prueba

Conectar al PRIMARY (mongo1) y crear una base de datos de prueba:

```
use laboratorio

// Crear colección con datos de ejemplo
for (let i = 0; i < 10000; i++) {
  db.experimentos.insertOne({
    experimento_id: i,
    tipo: ["temperatura", "presion", "humedad"][i % 3],
    valor: Math.random() * 100,
    timestamp: new Date(),
    ubicacion: ["Madrid", "Barcelona", "Valencia"][i % 3]
  });

  if (i % 1000 === 0) {
    print("Insertados: " + i);
  }
}
```

```
print("10,000 documentos insertados");
```

📝 **EJERCICIO 1.2:**

> Verifica que los datos se han replicado a los secundarios:
>
> 1. Sal del PRIMARY (exit)
> 2. Conecta a mongo2: `docker exec -it mongo2 mongosh`
> 3. Ejecuta: `rs.secondaryOk()` (permite leer en secundarios)
> 4. Ejecuta: `use laboratorio; db.experimentos.countDocuments()`
> 5. ¿Cuántos documentos hay?

Paso 1.4: Simular un failover

Ahora vamos a detener el PRIMARY y observar cómo uno de los secundarios se convierte en el nuevo primario.

Identificar el PRIMARY actual:

```
docker exec -it mongo1 mongosh --eval "rs.status().members.find(m =>
m.stateStr === 'PRIMARY')"
```

Detener el contenedor PRIMARY (asumiendo que es mongo1):

```
docker stop mongo1
```

📝 **EJERCICIO 1.3:**

> 1. Conecta rápidamente a mongo2: `docker exec -it mongo2 mongosh`
> 2. Ejecuta `rs.status()` cada 5 segundos
> 3. Observa el cambio de estado: SECONDARY → (posiblemente RECOVERING) → PRIMARY
> 4. ¿Cuánto tardó en completarse la elección?
> 5. Anota el nuevo `electionTime`

Verificar que el cluster sigue funcionando:

```
use laboratorio

// Intentar insertar un nuevo documento
db.experimentos.insertOne({
  experimento_id: 10001,
  tipo: "test_failover",
  timestamp: new Date()
})
```

Restaurar mongo1:

```
docker start mongo1
```

EJERCICIO 1.4:

> 1. Observa cómo mongo1 se reincorpora al cluster como SECONDARY
> 2. Ejecuta `rs.status()` en mongo1
> 3. Verifica su `syncSourceHost` (de dónde está replicando)
> 4. ¿Cuánto tarda en sincronizarse completamente?

Paso 1.5: Monitorizar el lag de replicación

Vamos a crear un script que genere escrituras continuas y monitorice el lag.

Script de escritura continua (`write-load.js`):

```
use laboratorio

let count = 0;
const startTime = new Date();

print("🚀 Iniciando carga de escritura continua...");
print("Presiona Ctrl+C para detener\n");

while (true) {
  try {
    db.experimentos.insertOne({
      experimento_id: 10000 + count,
      tipo: "load_test",
      valor: Math.random() * 100,
      timestamp: new Date()
    });

    count++;

    if (count % 100 === 0) {
      const elapsed = (new Date() - startTime) / 1000;
      const rate = count / elapsed;
      print(`Insertados: ${count} | Tasa: ${rate.toFixed(2)} docs/s`);
    }

    sleep(10);  // 10ms entre inserts = ~100 inserts/seg
  } catch (e) {
    print(" Error: " + e);
    break;
  }
}
```

Ejecutar en una terminal:

```
docker exec -it mongo1 mongosh < write-load.js
```

Script de monitorización de lag (`monitor-lag.js`):

```
function monitorLag() {
  while (true) {
    const status = rs.status();
    const primary = status.members.find(m => m.state === 1);

    print("\n=== REPLICATION LAG ===");
    print("Time: " + new Date().toISOString());

    status.members.forEach(m => {
      if (m.state === 2 && primary) {
        const lag = Math.round((primary.optimeDate - m.optimeDate) /
1000);
        const lagColor = lag > 5 ? "⚠" : "✅ ";
        print(`${lagColor} ${m.name}: ${lag} segundos de lag`);
      } else if (m.state === 1) {
        print(` ${m.name}: PRIMARY`);
      }
    });

    sleep(2000);
  }
}

monitorLag();
```

Ejecutar en otra terminal:

```
docker exec -it mongo2 mongosh < monitor-lag.js
```

** EJERCICIO 1.5:**

> Mientras corre la carga de escritura:
>
> 1. Observa el lag en los secundarios
> 2. Detén mongo2 temporalmente: `docker pause mongo2`
> 3. Espera 30 segundos
> 4. Reactiva mongo2: `docker unpause mongo2`
> 5. ¿Cuánto tarda en recuperarse del lag?