Actividad 2 - Conceptos y comandos básicos de la replicación en bases de datos NoSQL

Cristian Leandro Pérez Peláez

Natalia Sierra Salamando

Facultad de Ingeniería, Corporación Universitaria Iberoamericana

Ingeniería De Software

Profesor: Jorge Castañeda

01 de diciembre de 2024

Actividad 2 - Conceptos y comandos básicos de la replicación en bases de datos NoSQL

Cristian Leandro Pérez Peláez y Natalia Sierra Salamando

Profesor: Jorge Castañeda

Corporación Universitaria Iberoamericana

Facultad de Ingeniería

Ingeniería De Software

Bogotá, Colombia

2024

Actividad 2 - Conceptos y comandos básicos de la replicación en bases de datos NoSQL

Introducción:

En este documento se definen los criterios de calidad relacionados con la redundancia y la disponibilidad 24x7 del sistema de bases de datos para gestionar un torneo de fútbol (el cual se realizó su estudio y creación de BD en la actividad 1), implementado con base de datos NO SQL MongoDB. Este sistema nos garantizar la continuidad operativa y la integridad de los datos mediante una configuración de replicación adecuada.

- Objetivos
    - Asegurar la disponibilidad continua de los datos mediante alta disponibilidad.
    - Reducir el riesgo de pérdida de información con redundancia.
    - Garantizar la eficiencia en la recuperación y sincronización de los datos tras un fallo.

1. **Requerimientos No Funcionales**

- Redundancia:
    - El sistema debe contar con una replicación de datos en mínimo tres nodos distribuidos en un conjunto de réplicas (Replica Set) para evitar pérdida de información en caso de fallos.
    - Cada nodo debe tener una copia actualizada de los datos, sincronizándose en tiempo real con el nodo primario.
    - Los nodos deben incluir:
        - Nodo primario: Responsable de las operaciones de escritura.

        - Nodos secundarios: Responsables de las operaciones de lectura y de asumir el rol primario en caso de fallo.

- Disponibilidad 24x7:
    - El sistema debe estar disponible el 99.99% del tiempo, garantizando acceso continuo a los datos sin interrupciones significativas.
    - La configuración debe permitir la conmutación por fallo automática, de modo que un nodo secundario asuma el rol de nodo primario en menos de 5 segundos tras la caída del nodo activo.
    - La sincronización entre nodos debe realizarse con un retardo mínimo para asegurar la coherencia de los datos.

- Tolerancia a Fallos:
    - El sistema debe ser capaz de operar correctamente incluso si un nodo está fuera de servicio, siempre que al menos dos nodos estén operativos.

- Escalabilidad:
    - La arquitectura debe permitir la adición de más nodos secundarios sin interrumpir las operaciones actuales.

2. Consultas para Configuración de Replicación

- Estrategia de Replicación

  Se implementará un Replica Set de MongoDB, configurado con tres nodos distribuidos de la siguiente manera:

  - Nodo Primario: Responsable de gestionar las operaciones de escritura.
  - Nodos Secundarios: Reciben y mantienen una copia sincronizada de los datos, y pueden atender operaciones de lectura o asumir el rol primario en caso de fallo.

- Ventajas del Replica Set:

  - Alta disponibilidad y tolerancia a fallos.
  - Sincronización automática entre nodos.
  - Conmutación por fallo automática.

- Configuración del Entorno de Replicación:

  Paso 1: Iniciar los procesos de MongoDB

  Ejecutar los siguientes comandos para iniciar tres instancias de MongoDB con puertos diferentes:

  - # Nodo primario

    mongod --replSet rs0 --port 27017 --dbpath /data/db1 --bind_ip localhost
  - # Nodo secundario 1

    mongod --replSet rs0 --port 27018 --dbpath /data/db2 --bind_ip localhost
  - # Nodo secundario 2

    mongod --replSet rs0 --port 27019 --dbpath /data/db3 --bind_ip localhost

Paso 2: Configurar el Replica Set

Desde una instancia de MongoDB (nodo primario), ejecutar los siguientes comandos para inicializar el Replica Set

```
// Conectar al nodo primario
mongo --port 27017


// Inicializar el Replica Set
rs.initiate({
  _id: "rs0",
  members: [
    { _id: 0, host: "localhost:27017" },
    { _id: 1, host: "localhost:27018" },
    { _id: 2, host: "localhost:27019" }
  ]
});
```

Paso 3: Verificar el Estado del Replica Set

```
rs.status();
```

Paso 4: Habilitar la Lectura en Nodos Secundarios

```
rs.secondaryOk();
```

- Carga de Datos en el Nodo Primario

  Insertar los datos iniciales (equipos, jugadores, entrenadores, árbitros, partidos) en el nodo primario. MongoDB replicará automáticamente estos datos a los nodos secundarios.

  Ejemplo para la colección de equipos:

```
db.equipos.insertOne({
  _id: "equipo_4",
  nombre: "Águilas Doradas",
  ciudad: "Medellín",
  entrenadorId: "entrenador_4",
  jugadores: ["jugador_10", "jugador_11"],
  puntos: 5,
  victorias: 1,
  derrotas: 1,
  empates: 2,
  golesAFavor: 8,
  golesEnContra: 6,
  diferenciaGoles: 2
});
```

- Prueba de Replicación

  Para confirmar que los datos se replicaron correctamente, conectarse a uno de los nodos secundarios y ejecutar una consulta de lectura:

```
# Conectar al nodo secundario 1
mongo --port 27018


# Consultar los equipos
db.equipos.find();
```

3. Casos de Prueba

   En este apartado se describe una serie de casos de prueba diseñados para mejorar el proceso de replicación en una base de datos MongoDB utilizada para gestionar el torneo de fútbol. Las pruebas se centraron en garantizar que el conjunto de replicación cumpliera con los requisitos de redundancia y disponibilidad 24x7 previamente definidos.

- Objetivos de las Pruebas
  - Validar la sincronización automática entre el nodo primario y los secundarios.
  - Garantizar que la conmutación por fallo ocurra correctamente.
  - Comprobar que los datos están disponibles en los nodos secundarios para operaciones de lectura.
  - Verificar la integridad de los datos tras fallos simulados.

- Escenarios de Prueba:
  - Caso de Prueba 1: Verificación de Sincronización de Datos

    Validar que los datos insertados en el nodo primario se replican automáticamente en los nodos secundarios.

    - Pasos:
      - Insertar un documento en la colección equipos en el nodo primario:

```
db.equipos.insertOne({
 _id: "equipo_5",
 nombre: "Guerreros FC",
 ciudad: "Cali",
 puntos: 3,
 victorias: 1,
 derrotas: 2,
 empates: 0,
 golesAFavor: 6,
 golesEnContra: 7,
 diferenciaGoles: -1
});
```

      - Conectarse a un nodo secundario y ejecutar:

```
db.equipos.find({ _id: "equipo_5" });
```

- Resultado Esperado: El documento insertado en el nodo primario está disponible en los nodos secundarios.

- Caso de Prueba 2: Validación de Conmutación por Fallo

  Confirmar que, si el nodo primario falla, uno de los nodos secundarios asume el rol primario.

  - Pasos:
    - Apagar el nodo primario:

      sudo systemctl stop mongod --port 27017

    - Verificar el nuevo nodo primario ejecutando en un nodo secundario:

      rs.status();

  - Resultado Esperado: Uno de los nodos secundarios asume el rol de nodo primario.

- Caso de Prueba 3: Disponibilidad de Datos Durante Fallos

  Validar que los datos siguen siendo accesibles desde los nodos secundarios durante la caída del nodo primario.

  - Pasos:
    - Apagar el nodo primario.

      sudo systemctl stop mongod --port 27017

- Conectarse a un nodo secundario y ejecutar:

  db.equipos.find();

- Resultado Esperado: Los datos son accesibles desde los nodos secundarios.

- Caso de Prueba 4: Recuperación de Datos Tras Restauración del Nodo Primario

  Validar que, al restaurar el nodo primario, los datos se sincronizan correctamente.

  - Pasos:
    - Reiniciar el nodo primario:

      sudo systemctl start mongod --port 27017

    - Verificar la sincronización ejecutando en el nodo restaurado:

      db.equipos.find();

  - Resultado Esperado: El nodo primario contiene los datos sincronizados.

4. Resultados de la Ejecución de Pruebas

   - Caso de Prueba 1: Verificación de Sincronización de Datos

     Validar que los datos insertados en el nodo primario se replican automáticamente en los nodos secundarios.

     Resultado:

     - Insertar un documento en el nodo primario: Éxito.
     - Consultar el documento en los nodos secundarios: Éxito

Observación: Los nodos secundarios contenían el documento replicado en menos de 1 segundo.

- Caso de Prueba 2: Validación de Conmutación por Fallo

  Confirmar que, si el nodo primario falla, uno de los nodos secundarios asume el rol primario.

  Resultado:

  - Apagar el nodo primario: Éxito.
  - Nuevo nodo primario identificado correctamente: Éxito.
  - Tiempo para conmutación por fallo: 4 segundos.

  Observación: El Replica Set funcionó según lo esperado; la conmutación fue automática y rápida.

- Caso de Prueba 3: Disponibilidad de Datos Durante Fallos

  Validar que los datos siguen siendo accesibles desde los nodos secundarios durante la caída del nodo primario.

  Resultado:

  - Datos accesibles desde los nodos secundarios: Éxito.

  Observación: Los datos se mantuvieron disponibles sin interrupciones significativas.

- Caso de Prueba 4: Recuperación de Datos Tras Restauración del Nodo Primario

  Validar que, al restaurar el nodo primario, los datos se sincronizan correctamente.

  Resultado:

- Reiniciar el nodo primario: Éxito.
- Datos sincronizados correctamente: Éxito.

Observación: Los datos del nodo restaurado se sincronizaron con los secundarios en menos de 3 segundos.

Repositorio GitHub:

https://github.com/clpp-dev/Actividad2-Conceptos-Y-Comandos-basicos-de-la-replicacion-en-bases-de-datos-NoSQL