

# Actividad 2

Conceptos y comandos básicos de la replicación en bases de datos NoSQL

# Autor.

Daniel Andrés Caicedo Bohórquez ID 100110918 Correo: [dcaice20@ibero.edu.co](mailto:dcaice20@ibero.edu.co)

# Docente.

William Ruiz

**Bases de datos avanzadas Ingeniería de Software virtual** 26 de Mayo de 2024

**ACTIVIDAD 2 – CONCEPTOS Y COMANDOS BASICOS DE LA REPLICACION EN BASES DE DATOS NO SQL**

**AUTOR**

# DANIEL ANDRES CAICEDO BOHORQUEZ

**DOCENTE WILLIAN RUIZ**

# CORPORACIÓN UNIVERSITARIA IBEROAMERICANA INGENIERÍA DE SOFTWARE VIRTUAL BASES DE DATOS AVANZADAS

**26 DE MAYO DE 2024**

**DOCUMENTO DETALLADO DE REQUERIMIENTOS NO FUNCIONALES PARA LA BASE DE DATOS torneo\_champions**

**1. INTRODUCCION**

Este documento detalla los requerimientos no funcionales relacionados con la redundancia y disponibilidad 24x7 de la base de datos **torneo\_champions**, la cual gestiona información específica sobre el torneo de la final de fútbol de la Champions League 2024 que se jugara en Wembley. Esta base de datos es esencial para almacenar y manejar datos específicos sobre los dos equipos **(Real Madrid Borussia & Dortmund)**, jugadores, entrenadores, árbitros, el encuentro deportivo y resultados del partido de la final. Se tiene contemplada la explicación de este conjunto de pasos y estrategias que asegurara que la base de datos torneo\_champions esté configurada para alta disponibilidad, redundancia de datos, seguridad, escalabilidad, recuperación ante desastres, monitoreo y optimización del rendimiento. La implementación de estas prácticas garantizará que la base de datos sea robusta y confiable, cumpliendo con todos los requerimientos no funcionales planteados, por lo cual es importante que implementemos mecanismos robustos que garanticen la operatividad continua y la protección de los datos en todo momento. La implementación adecuada de estos mecanismos nos va a permitir asegurar que la base de datos esté siempre disponible para consultas y actualizaciones, incluso en casos donde se presenten fallos del sistema.

**2. OBJETIVOS**

**2.1** **Alta Disponibilidad 24x7:**

* Descripción: Asegurar que la base de datos esté disponible las 24 horas del día, los 7 días de la semana.
* Requisito: Implementar replica sets en MongoDB.
* Estrategia: Usar al menos tres nodos (un nodo primario y dos nodos secundarios) en el replica set.

**2.2** **Redundancia de Datos:**

* Descripción: Garantizar la redundancia de los datos para prevenir la pérdida de información.
* Requisito: Replicar los datos en múltiples nodos.
* Estrategia: Configurar replica sets con nodos secundarios.

**2.3 Seguridad de los Datos:**

* Descripción: Proteger los datos contra accesos no autorizados.
* Requisito: Implementar autenticación, autorización y encriptación.
* Estrategia: Usar TLS/SSL y roles de usuarios en MongoDB.

**2.4 Escalabilidad:**

* Descripción: Asegurar que la base de datos pueda escalar horizontalmente.
* Requisito: Soportar aumento de nodos y sharding.
* Estrategia: Configurar sharding si el volumen de datos crece significativamente.

**2.5 Recuperación ante vulnerabilidades, fallos o desastres:**

* Descripción: Proveer mecanismos para la recuperación de datos en caso de desastres.
* Requisito: Implementar backups y réplicas geográficamente bien distribuidas.
* Estrategia: Usar mongodump, mongorestore y configurar réplicas en diferentes regiones.

**2.6 Monitoreo y Alertas**

* Descripción: Implementar monitoreo continuo y alertas.
* Requisito: Monitorear métricas clave y configurar alertas para eventos críticos.
* Estrategia: Usar herramientas como MongoDB Cloud Manager, Prometheus y Grafana.

**2.7 Optimización del Rendimiento**

* Descripción: Asegurar alto rendimiento minimizando la latencia.
* Requisito: Indexar colecciones críticas y optimizar consultas.
* Estrategia: Crear índices y revisar consultas con el Profiler de MongoDB.

**3. ESTRATEGIA DE REPLICACION**

Para cumplir con los objetivos de alta disponibilidad, redundancia de datos, y recuperación ante desastres, vamos a realizar la configuración de un replica set en MongoDB, utilizando al menos tres nodos (un nodo primario y dos nodos secundarios).

**3.1 Configuración de Replica Sets**

* Descripción: Un replica set en MongoDB es un grupo de instancias mongod que mantienen las mismas copias de los datos. La configuración debe asegurar que la base de datos **torneo\_champions** esté siempre disponible y que los datos sean redundantes.

Creamos 3 carpetas para nuestros nodos

mkdir C:\data\db1

mkdir C:\data\db2

mkdir C:\data\db3



Para iniciar **mongod** debemos ejecutar en la línea de comandos CMD o PowerShell en Windows. Iniciamos las tres instancias **( nodo 1, nodo 2 y nodo 3),** en puertos diferentes con los siguientes comandos:

mongod --replSet "rs0" --port 27017 --dbpath C:\data\db1 --bind\_ip localhost

mongod --replSet "rs0" --port 27018 --dbpath C:\data\db2 --bind\_ip localhost

# mongod --replSet "rs0" --port 27019 --dbpath C:\data\db3 --bind\_ip localhost

# Asi es para cada una de las instancias

# 

# 3.2 Configurar el Replica Set

# Lo primero es conectarse a uno de los nodos y configurar el replica set. Para ello vamos a conectar al nodo primario y ejecutamos el comando en la Shell de Mongo DB

# mongo --port 27017

# rs.initiate(

# {

# \_id: "rs0",

# members: [

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

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

# { \_id: 2, host: "localhost:27019" }

# ]

# }

# )

# 

# 3.3 Verificar el estado del Replica Set

# Verificamos que el replica set esté funcionando correctamente, para ello vamos a conectar al nodo primario y ejecutamos el comando para validar el estado

# mongo --port 27017

# rs.status()

# 

# 

# 

# 3.4 Inserción de Datos

# Insertamos los datos del evento deportivo en el nodo primario. Esto nos va a asegurar que los datos se propaguen a los nodos secundarios de manera automática.

# Nos conectamos a la base de datos.

# use torneo\_champions

# Insertamos los equipos.

# db.equipos.insert([

# {

# \_id: "RM01",

# nombre: "Real Madrid",

# país: "España",

# entrenador: "id\_entrenador\_real",

# jugadores: ["id\_kross", "id\_modric", "id\_vinicius"],

# fase\_actual: "Final"

# },

# {

# \_id: "BD02",

# nombre: "Borussia Dortmund",

# país: "Alemania",

# entrenador: "id\_entrenador\_dortmund",

# jugadores: ["id\_sancho", "id\_reus", "id\_hummels"],

# fase\_actual: "Final"

# }

# ]);

# 

# Insertamos los jugadores.

# db.jugadores.insert([

# {

# \_id: "id\_kross",

# nombre: "Tony Kross",

# edad: 34,

# nacionalidad: "Alemana",

# posición: "Mediocentro",

# equipo: "RM01"

# },

# {

# \_id: "id\_modric",

# nombre: "Luka Modric",

# edad: 38,

# nacionalidad: "Croata",

# posición: "Mediocentro",

# equipo: "RM01"

# },

# {

# \_id: "id\_vinicius",

# nombre: "Vinicius Junior",

# edad: 23,

# nacionalidad: "Brasileña",

# posición: "Delantero",

# equipo: "RM01"

# },

# {

# \_id: "id\_sancho",

# nombre: "Jadon Sancho",

# edad: 24,

# nacionalidad: "Inglesa",

# posición: "Delantero",

# equipo: "BD02"

# },

# {

# \_id: "id\_reus",

# nombre: "Marco Reus",

# edad: 34,

# nacionalidad: "Alemana",

# posición: "Mediocentro",

# equipo: "BD02"

# },

# {

# \_id: "id\_hummels",

# nombre: "Mats Hummels",

# edad: 35,

# nacionalidad: "Alemana",

# posición: "Defensa",

# equipo: "BD02"

# }

# ]);

# 

# Insertamos los entrenadores.

# db.entrenadores.insert([

# {

# \_id: "id\_entrenador\_real",

# nombre: "Carlo Ancelotti",

# edad: 64,

# nacionalidad: "Italiana",

# historial\_equipos: ["PSG", "Chelsea", "Juventus", "Parma", "AC Milan", "Bayern", "Everton", "Real Madrid"]

# },

# {

# \_id: "id\_entrenador\_dortmund",

# nombre: "Marco Rose",

# edad: 46,

# nacionalidad: "Alemana",

# historial\_equipos: ["Borussia Mönchengladbach", "RB Salzburg"]

# }

# ]);

# 

# Insertamos los árbitros.

# db.arbitros.insert({

# \_id: "id\_arbitro\_final",

# nombre: "Szymon Marciniak",

# edad: 47,

# nacionalidad: "Polaca",

# partidos\_arbitrados: 312

# });

# 

# Insertar encuentro deportivo

# db.encuentrodep.insert({

# \_id: "id\_partido\_final",

# fecha: new Date("2024-05-31"),

# fase: "Final",

# equipo\_local: "RM01",

# equipo\_visitante: "BD02",

# árbitro: "id\_arbitro\_final",

# estadio: "Estadio de Wembley"

# });

# 

# Insertamos el resultado del partido.

# db.resultados.insert({

# encuentro: "id\_partido\_final",

# goles\_equipo\_local: 1,

# goles\_equipo\_visitante: 3,

# detalles: [

# { minuto: 30, autor: "id\_reus", tipo: "gol" },

# { minuto: 60, autor: "id\_sancho", tipo: "gol" },

# { minuto: 75, autor: "id\_vinicius", tipo: "gol" },

# { minuto: 89, autor: "id\_hummels", tipo: "gol" }

# ]

# });

# 

# 3.5 Validación de la replicación

# Verificamos que los datos se han replicado correctamente a los nodos secundarios:

# Nos conectamos al primer nodo secundario.

# mongo --port 27018

# use torneo\_champions

# db.equipos.find().pretty()

# db.jugadores.find().pretty()

# db.entrenadores.find().pretty()

# db.arbitros.find().pretty()

# db.encuentrodep.find().pretty()

# db.resultados.find().pretty()

# Nos conectamos al segundo nodo secundario.

# mongo --port 27019

# use torneo\_champions

# db.equipos.find().pretty()

# 

# db.jugadores.find().pretty()

# 

# db.entrenadores.find().pretty()

# 

# db.arbitros.find().pretty()

# 

# db.encuentrodep.find().pretty()

# 

# db.resultados.find().pretty()

# 

# 4. Seguridad de los Datos

# 4.1 Implementación:

# Autenticación y Autorización: Nos permite configurar usuarios y roles con permisos adecuados.

# Creamos un usuario administrador.

# use admin

# db.createUser({

# user: "admin",

# pwd: "adminAdmin",

# roles: [{ role: "root", db: "admin" }]

# })

# 

# Creamos un usuario para la base de datos torneo\_champions

# use torneo\_champions.

# use torneo\_champions

# db.createUser({

# user: "dbDaniel",

# pwd: "userPassword",

# roles: [{ role: "readWrite", db: "torneo\_champions" }]

# })

# 4.2 Encriptación:

# Configuramos los protocolos TLS/SSL(Transport Layer Security/Secure Sockets Layer) para asegurar y encriptar las comunicaciones entre clientes y servidores.

# Iniciamos mongod con TLS/SSL.

# mongod --replSet "rs0" --port 27017 --dbpath /data/db1 --bind\_ip localhost --tlsMode requireTLS --tlsCertificateKeyFile /path/to/your/cert.pem --tlsCAFile /path/to/your/ca.pem

# 5. Escalabilidad

# 5.1 Implementación:

# Sharding: Configuramos sharding si el volumen de datos crece significativamente. Esto nos va a permitir distribuir los datos de la base de datos torneo\_champions entre varios nodos para mejorar el rendimiento y la capacidad de almacenamiento

# Iniciamos un servidor de configuración.

# mongod --configsvr --replSet configReplSet --port 27019 --dbpath /data/config --bind\_ip localhost

# Iniciamos un servidor de shards.

# mongod --shardsvr --replSet shardReplSet --port 27018 --dbpath /data/shard --bind\_ip localhost

# Iniciamos mongo y nos conectamos al mongos y configuramos sharding.

# mongos --configdb configReplSet/localhost:27019 --bind\_ip localhost --port 27017

# mongo --port 27017

# Añadimos shards

# sh.addShard("shardReplSet/localhost:27018")

# Habilitamos sharding en la base de datos

# sh.enableSharding("torneo\_champions")

# Shardeamos una colección de equipos

# sh.shardCollection("torneo\_champions.equipos", { "\_id": "hashed" })

# 6. Recuperación ante Vulnerabilidades, Fallos o Desastres

# 6.1 Implementación:

# Backups y Restauración: Usar mongodump y mongorestore nos permite realizar backups y restauraciones de la base de datos.

# 

# Hacemos backup de la base de datos.

# mongodump --out /path/to/backup --db torneo\_champions

# Hacemos restauración de la base de datos.

# mongorestore --db torneo\_champions /path/to/backup/torneo\_champions

# 6.2 Réplicas Geográficamente Distribuidas:

# Configuramos réplicas en diferentes regiones para mejorar la recuperación ante desastres.

# Añadimos miembros adicionales al replica set en diferentes regiones.

# rs.add("remoteHost:27017")

# 7. Monitoreo y Alertas

# 7.1 Implementación:

# Herramientas de Monitoreo: Usar MongoDB Cloud Manager, Prometheus y Grafana nos permite monitorear métricas clave.

# 

# Configuramos Prometheus con el Exporter de MongoDB.

# prometheus --config.file=/path/to/prometheus.yml

# Configuramos Grafana para visualizar las métricas de Prometheus.

# grafana-server --config=/path/to/grafana.ini

# 7.2 Configuración de Alertas:

# Configuramos alertas para eventos críticos.

# cloud.mongodb.com

# 8. Optimización del Rendimiento

# 81. Implementación:

# Indexación: Permite crear índices en colecciones críticas para mejorar el rendimiento de las consultas.

# Creamos los siguientes índices en las colecciones

# db.equipos.createIndex({ nombre: 1 })

# db.jugadores.createIndex({ nombre: 1 })

# db.encuentrodep.createIndex({ fecha: 1 })

# 

# Optimización de Consultas: Permite revisar y optimizar consultas utilizando el Profiler de MongoDB.

# Activamos el profiler para registrar la información detallada sobre las operaciones anteriormente ejecutadas

# db.setProfilingLevel(2)

# 

# Consultamos el profiler

# db.system.profile.find().pretty()

# 

# REFERENCIAS BIBLIOGRAFICAS

**Referencias sugeridas por la universidad**

* Sarasa, A. (2016). Introducción a las bases de datos NoSQL usando MongoDB. Editorial UOC. (Capitulo 7- Replicación)

# Referencias de investigación propia

* MongoBD. (2024)What is MongoDB? , Recuperado de: https:/[/www](http://www.mongodb.com/docs/manual/).[mongodb.com/docs/manual/](http://www.mongodb.com/docs/manual/)