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

ANDREY LEANDRO VARONA ORJUELA

LUIGY GILBERTO CICUA CABALLERO

CORPORACIÓN UNIVERSITARIA IBEROAMERICANA

FACULTAD DE INGENIERÍA

BASE DE DATOS AVANZADA

IBAGUE – TOLIMA

25 DE MAYO DEL 2024

Disponibilidad 24/7 para el Torneo Deportivo de Tenis

El sistema debe estar disponible las 24 horas del día, los 7 días de la semana, para garantizar la satisfacción del usuario y el éxito del torneo de tenis. Para lograr esto, se deben cumplir los siguientes requerimientos no funcionales:

1. **Tiempo de actividad del sistema**: El sistema debe estar en funcionamiento continuo durante todo el evento, sin interrupciones, y disponible en todo momento para los usuarios.
2. **Monitoreo constante del sistema**: Se deben realizar monitoreos constantes del sistema para detectar cualquier problema que pueda afectar su disponibilidad y solucionarlos en tiempo real.
3. **Personal de soporte técnico disponible**: Se debe contar con personal de soporte técnico disponible las 24 horas del día, los 7 días de la semana, para responder rápidamente a cualquier problema técnico que pueda surgir.

Además, el sistema debe contar con:

- **Infraestructura de alta disponibilidad y escalabilidad** para soportar un alto volumen de tráfico.
- **Balanceador de carga** para distribuir el tráfico de manera equilibrada entre los servidores.
- **Configuración de servidores** que permita la actualización y el mantenimiento en vivo del sistema, sin interrupciones en el servicio.
- **Medidas de seguridad y recuperación de desastres** para garantizar la continuidad del servicio en caso de incidentes.

Replicación y Redundancia

Para asegurar la máxima disponibilidad y fiabilidad del sisvc iktema durante el torneo de tenis, se implementarán las siguientes estrategias:

- **Replicación de datos**: Los datos del sistema serán replicados en múltiples ubicaciones geográficas. Esto asegura que, en caso de una falla en un servidor o centro de datos, los datos seguirán siendo accesibles desde otra ubicación sin pérdida de información.
- **Redundancia de componentes críticos**: Todos los componentes críticos del sistema, como servidores, bases de datos, y redes, tendrán redundancias implementadas. Esto significa que si un componente falla, otro tomará su lugar inmediatamente, garantizando así la continuidad del servicio sin interrupciones.
- **Balanceo de carga redundante**: Se implementarán múltiples balanceadores de carga que trabajen en conjunto para distribuir el tráfico de manera equilibrada y, en caso de fallo de uno, otro puede asumir su función sin afectar la experiencia del usuario.
- **Infraestructura escalable**: La infraestructura estará diseñada para escalar horizontalmente, añadiendo más recursos según sea necesario para manejar el aumento de tráfico durante el torneo sin comprometer el rendimiento del sistema.

Con estas medidas de replicación y redundancia, junto con los requerimientos no funcionales previamente mencionados, se garantiza una alta redundancia y disponibilidad 24/7 del sistema para el torneo de tenis, asegurando así una experiencia fluida y satisfactoria para todos los participantes y espectadores.

Proceso de replicación en Mongo DB

Iniciamos creando 3 instancias para que se conviertan en los servidores que posteriormente vamos a utilizar, están vana a ser usadas a manera que conserven o contengan una copia de los datos.

Creamos por medio de consola los 3 nodos:

```
> setduplicadoback=new ReplSetTest({name:"duplicadoset",nodes:3}); print("Confirmacion terminada")
Starting new replica set duplicadoset
Confirmacion terminada
>
```

Iniciar el grupo de nodos replica que creamos con el comando anterior, ejecutamos:

```
> setduplicadoback.startSet()
```

```
[
        connection to DESKTOP-KI76779:20000,
        connection to DESKTOP-KI76779:20001,
        connection to DESKTOP-KI76779:20002
]
[
        connection to DESKTOP-KI76779:20000,
        connection to DESKTOP-KI76779:20001,
        connection to DESKTOP-KI76779:20002
]
>
```

Ahora vamos a activar la replica de datos para con el siguiente comando:

```
> setduplicadoback.initiate()
```

```
Reconfiguring replica set to add in other nodes
{
        "replSetReconfig" : {
                "_id" : "duplicadoset",
                "protocolVersion" : 1,
                "members" : [
                        {
                                "_id" : 0,
                                "host" : "DESKTOP-KI76779:20000"
                        },
                        {
                                "_id" : 1,
                                "host" : "DESKTOP-KI76779:20001"
                        },
                        {
                                "_id" : 2,
                                "host" : "DESKTOP-KI76779:20002"
                        }
                ],
                "version" : 2
        }
}
```

```
d20002| 2024-05-25T18:06:08.349-0500 I  REPL     [replexec-0] Member DESKTOP-KI76779:20001 is now in s
tate SECONDARY
AwaitNodesAgreeOnPrimary: Waiting for nodes to agree on any primary.
AwaitNodesAgreeOnPrimary: Nodes agreed on primary DESKTOP-KI76779:20000
Waiting for keys to sign $clusterTime to be generated
AwaitLastStableRecoveryTimestamp: Beginning for [
        "DESKTOP-KI76779:20000",
        "DESKTOP-KI76779:20001",
        "DESKTOP-KI76779:20002"
]
AwaitNodesAgreeOnPrimary: Waiting for nodes to agree on any primary.
AwaitNodesAgreeOnPrimary: Nodes agreed on primary DESKTOP-KI76779:20000
AwaitLastStableRecoveryTimestamp: ensuring the commit point advances for [
        "DESKTOP-KI76779:20000",
        "DESKTOP-KI76779:20001",
        "DESKTOP-KI76779:20002"
]
```

Comprobamos como generar la replica de los datos en los diferentes nodos creados, ejecutando el siguiente comando para así establecer una la conexión:

```
> conn=new Mongo("DESKTOP-KI76779:20000")
```

```
d20000| 2024-05-25T18:13:48.804-0500 I  NETWORK  [listener] connection accepted from 192.168.137.1:564
97 #27 (8 connections now open)
d20000| 2024-05-25T18:13:48.814-0500 I  NETWORK  [conn27] received client metadata from 192.168.137.1:
56497 conn27: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", ver
sion: "4.2.25" }, os: { type: "Windows", name: "Microsoft Windows 10", architecture: "x86_64", version
: "10.0 (build 22631)" } }
connection to DESKTOP-KI76779:20000
```

Posteriormente tenemos que seleccionar la base de datos que tengamos almacenada en Mongo DB que anteriormente ya teníamos o que realizamos, esto con el siguiente comando:

```
> testDB=conn.getDB("torneoTenis")
```

```
> testDB=conn.getDB("torneoTenis")
torneoTenis
```

Le vamos a preguntar a el nodo si es un nodo maestro o si por el contrario es un nodo secundario:

```
> testDB.isMaster()
```

```
> testDB.isMaster()
{
        "hosts" : [
                "DESKTOP-KI76779:20000",
                "DESKTOP-KI76779:20001",
                "DESKTOP-KI76779:20002"
        ],
        "setName" : "duplicadoset",
        "setVersion" : 2,
        "ismaster" : true,
        "secondary" : false,
        "primary" : "DESKTOP-KI76779:20000",
        "me" : "DESKTOP-KI76779:20000",
        "electionId" : ObjectId("7fffffff0000000000000001"),
        "lastWrite" : {
                "opTime" : {
                        "ts" : Timestamp(1716679241, 1),
                        "t" : NumberLong(1)
                },
                "lastWriteDate" : ISODate("2024-05-25T23:20:41Z"),
                "majorityOpTime" : {
                        "ts" : Timestamp(1716679241, 1),
                        "t" : NumberLong(1)
                },
                "majorityWriteDate" : ISODate("2024-05-25T23:20:41Z")
        },
        "maxBsonObjectSize" : 16777216,
        "maxMessageSizeBytes" : 48000000,
        "maxWriteBatchSize" : 100000,
        "localTime" : ISODate("2024-05-25T23:21:55.954Z"),
        "logicalSessionTimeoutMinutes" : 30,
        "connectionId" : 27,
        "minWireVersion" : 0,
        "maxWireVersion" : 8,
        "readOnly" : false,
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1716679241, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1716679241, 1)
}
```

Ingresamos al menos 5 datos a una de las colecciones de la base de datos que teniamos creada:

```
...     { "nombre": "Rafael Nadal", "edad": 35, "nacionalidad": "Español", "ranking": 3, "partidos_jugados": 30 },
...     { "nombre": "Serena Williams", "edad": 39, "nacionalidad": "Estadounidense", "ranking": 8, "partidos_jugados": 25 },
...     { "nombre": "Novak Djokovic", "edad": 34, "nacionalidad": "Serbio", "ranking": 1, "partidos_jugados": 35 },
...     { "nombre": "Ashleigh Barty", "edad": 25, "nacionalidad": "Australiana", "ranking": 2, "partidos_jugados": 28 },
...     { "nombre": "Roger Federer", "edad": 40, "nacionalidad": "Suizo", "ranking": 7, "partidos_jugados": 20 }
... ])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("663ea41b9a53952ae97bea0a"),
                ObjectId("663ea41b9a53952ae97bea0b"),
                ObjectId("663ea41b9a53952ae97bea0c"),
                ObjectId("663ea41b9a53952ae97bea0d"),
                ObjectId("663ea41b9a53952ae97bea0e")
        ]
}
```

Luego de almacenar los datos debemos comprobar que efectivamente hayan quedado tanto la cantidad como los datos en si ingresado, con los siguientes comandos:

```
> testDB.deportistas.count();
5
> testDB.deportistas.find().pretty()
{
        "_id" : ObjectId("665274e81be9a22432004974"),
        "nombre" : "Rafael Nadal",
        "edad" : 35,
        "nacionalidad" : "Español",
        "ranking" : 3,
        "partidos_jugados" : 30
}
{
        "_id" : ObjectId("665274e81be9a22432004975"),
        "nombre" : "Serena Williams",
        "edad" : 39,
        "nacionalidad" : "Estadounidense",
        "ranking" : 8,
        "partidos_jugados" : 25
}
{
        "_id" : ObjectId("665274e81be9a22432004976"),
        "nombre" : "Novak Djokovic",
        "edad" : 34,
        "nacionalidad" : "Serbio",
        "ranking" : 1,
        "partidos_jugados" : 35
}
{
        "_id" : ObjectId("665274e81be9a22432004977"),
        "nombre" : "Ashleigh Barty",
        "edad" : 25,
        "nacionalidad" : "Australiana",
        "ranking" : 2,
        "partidos_jugados" : 28
}
{
        "_id" : ObjectId("665274e81be9a22432004978"),
        "nombre" : "Roger Federer",
        "edad" : 40,
        "nacionalidad" : "Suizo",
        "ranking" : 7,
        "partidos_jugados" : 20
}
```

Verificamos que si se realizo la replica de los datos en los nodos secundarios, ejecutamos en orden los siguientes comandos como se aprecia en las siguientes imágenes:

```
> connSecondary=new Mongo("DESKTOP-KI76779:20001")
```

```
> connSecondary=new Mongo("DESKTOP-KI76779:20001")
d20001| 2024-05-25T18:44:14.156-0500 I  NETWORK  [listener] connection accepted from 192.168.137.
1:59840 #18 (4 connections now open)
d20001| 2024-05-25T18:44:14.166-0500 I  NETWORK  [conn18] received client metadata from 192.168.1
37.1:59840 conn18: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Cl
ient", version: "4.2.25" }, os: { type: "Windows", name: "Microsoft Windows 10", architecture: "x
86_64", version: "10.0 (build 22631)" } }
connection to DESKTOP-KI76779:20001
```

```
> secondaryTestDB=connSecondary.getDB("torneoTenis")
torneoTenis
```

Esta tercera consulta la utilizamos para saber si el nodo es un nodo maestro o un nodo secundario:

```
> secondaryTestDB.isMaster()
```

```
> secondaryTestDB.isMaster()
{
        "hosts" : [
                "DESKTOP-KI76779:20000",
                "DESKTOP-KI76779:20001",
                "DESKTOP-KI76779:20002"
        ],
        "setName" : "duplicadoset",
        "setVersion" : 2,
        "ismaster" : false,
        "secondary" : true,
        "primary" : "DESKTOP-KI76779:20000",
        "me" : "DESKTOP-KI76779:20001",
        "lastWrite" : {
                "opTime" : {
                        "ts" : Timestamp(1716680343, 5),
                        "t" : NumberLong(1)
                },
                "lastWriteDate" : ISODate("2024-05-25T23:39:03Z"),
                "majorityOpTime" : {
                        "ts" : Timestamp(1716680343, 5),
                        "t" : NumberLong(1)
                },
                "majorityWriteDate" : ISODate("2024-05-25T23:39:03Z")
        },
        "maxBsonObjectSize" : 16777216,
        "maxMessageSizeBytes" : 48000000,
        "maxWriteBatchSize" : 100000,
        "localTime" : ISODate("2024-05-25T23:47:52.810Z"),
        "logicalSessionTimeoutMinutes" : 30,
        "connectionId" : 18,
        "minWireVersion" : 0,
        "maxWireVersion" : 8,
        "readOnly" : false,
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1716680343, 5),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1716680343, 5)
}
```

Vamos a realizar una simulación de la caída del nodo o de un nodo primario siguiendo los tres comandos que se encuentran a continuación y en orden:

```
> connPrimary=new Mongo("DESKTOP-KI76779:20000")
d20000| 2024-05-25T19:03:06.865-0500 I  NETWORK  [listener] connection accepted from 192.168.137.
1:59979 #28 (9 connections now open)
d20000| 2024-05-25T19:03:06.865-0500 I  NETWORK  [conn28] received client metadata from 192.168.1
37.1:59979 conn28: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Cl
ient", version: "4.2.25" }, os: { type: "Windows", name: "Microsoft Windows 10", architecture: "x
86_64", version: "10.0 (build 22631)" } }
connection to DESKTOP-KI76779:20000
```

```
> primaryDB=connPrimary.getDB("torneoTenis")
torneoTenis
```

```
> primaryDB.isMaster()
{
        "hosts" : [
                "DESKTOP-KI76779:20000",
                "DESKTOP-KI76779:20001",
                "DESKTOP-KI76779:20002"
        ],
        "setName" : "duplicadoset",
        "setVersion" : 2,
        "ismaster" : true,
        "secondary" : false,
        "primary" : "DESKTOP-KI76779:20000",
        "me" : "DESKTOP-KI76779:20000",
        "electionId" : ObjectId("7fffffff0000000000000001"),
        "lastWrite" : {
                "opTime" : {
                        "ts" : Timestamp(1716680343, 5),
                        "t" : NumberLong(1)
                },
                "lastWriteDate" : ISODate("2024-05-25T23:39:03Z"),
                "majorityOpTime" : {
                        "ts" : Timestamp(1716680343, 5),
                        "t" : NumberLong(1)
                },
                "majorityWriteDate" : ISODate("2024-05-25T23:39:03Z")
        },
        "maxBsonObjectSize" : 16777216,
        "maxMessageSizeBytes" : 48000000,
        "maxWriteBatchSize" : 100000,
        "localTime" : ISODate("2024-05-26T00:05:16.907Z"),
        "logicalSessionTimeoutMinutes" : 30,
        "connectionId" : 28,
        "minWireVersion" : 0,
        "maxWireVersion" : 8,
        "readOnly" : false,
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1716680343, 5),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1716680343, 5)
}
```

Procedemos a iniciar la prueba de desconexión del nodo primarios con los siguientes comandos:

Este primer comando nos permite dar de baja o desconectar el nodo primario:

```
> PrimaryDB.adminCommand({shutdown:1})
2023-03-28T22:18:45.634-0500 I  NETWORK  [js] DBClientConnection
2023-03-28T22:18:45.646-0500 E  QUERY    [js] uncaught exception:
calhost:20000'   :
DB.prototype.runCommand@src/mongo/shell/db.js:169:19
DB.prototype.adminCommand@src/mongo/shell/db.js:187:12
@(shell):1:1
```

Luego con estos comandos vamos a realizar la comprobación del nuevo nodo primario:

```
> connNewPrimary=new Mongo("localhost:20001")
d20001| 2024-05-25T19:38:06.433-0500 I  NETWORK  [listener] connection accepted from 127.0.0.1:60304 #21 (5 connectio
ns now open)
d20001| 2024-05-25T19:38:06.434-0500 I  NETWORK  [conn21] received client metadata from 127.0.0.1:60304 conn21: { app
lication: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", version: "4.2.25" }, os: { type: "Win
dows", name: "Microsoft Windows 10", architecture: "x86_64", version: "10.0 (build 22631)" } }
connection to localhost:20001
```

```
> newPrimaryDB=connNewPrimary.getDB("torneoTenis")
torneoTenis
```

Aquí podemos visualizar como el nodo que antes era secundario paso a ser primario y que nos garantiza la disponibilidad de los datos:

```
> newPrimaryDB.isMaster()
{
        "hosts" : [
                "DESKTOP-V8Q56PJ:20000",
                "DESKTOP-V8Q56PJ:20001",
                "DESKTOP-V8Q56PJ:20002"
        ],
        "setName" : "MireplicaSet",
        "setVersion" : 2,
        "ismaster" : true,
        "secondary" : false,
        "primary" : "DESKTOP-V8Q56PJ:20001",
        "me" : "DESKTOP-V8Q56PJ:20001",
        "electionId" : ObjectId("7fffffff0000000000000002"),
        "lastWrite" : {
```

Repositorio GitHub: https://github.com/LuigyC/TorneoDeportivo/tree/main/Actvidad%202

Durante la ejecución de los comandos anteriores por medio de Mongo Shell para este caso de prueba nos dimos de cuenta que:

La replicación se pudo evidenciar en la creación de los nodos primarios y secundarios.

La disponibilidad la pudimos verificar ingresando datos en el nodo primario y posteriormente verificando que estos datos se encontraran en el nodo secundario.

La disponibilidad también se pudo corroborar por medio de la revisión de la sucesión de nodos secundarios como primarios.

La tolerancia a fallos o caídas se ve reflejada con la prueba de desanexió n del nodo primario y verificando que el nodo secundario paso a ser nodo primario para cubrir este rol.

Link Video explicativo :Actividades-20240525_222155-Grabación de la reunión.mp4