Instituto Tecnológico de Costa Rica

Campus Tecnológico Central Cartago

Escuela de Computación

Base de Datos II: IC4302

Trabajo Corto #2: MSSQL Replica

GitHub: https://github.com/LUISGM1501/BD_TC2

Profesor:

Kenneth Roberto Obando Rodríguez

Integrantes:

Luis Gerardo Urbina Salazar

Carnet 2023156802

Andres Mora Urbina

Carnet 2023064564

Fecha de entrega 12 de septiembre del 2024

Contenidos

1. Introducción

En el avance continuo en las tecnologías de base de datos, se han desarrollado métodos para asegurar la confiabilidad y la estabilidad de la información en formato digital, de este modo, convirtiéndose en el método predilecto para el manejo de información actual, siendo eficiente y permitiendo el constante acceso a la información por diferentes medios. Entre estas técnicas, se ha desarrollado el método de replicación *failover*, que se conoce como un mecanismo que proporciona a una base de datos funcionalidades para permitir la transacción automática de información desde un repositorio central a un repositorio secundario en caso de que el primario presente fallos o pierda estabilidad de forma inesperada. El *failover* es un método centrado en asegurar la estabilidad de un servicio, minimizando el tiempo de inactividad que puede presentar un sistema al fallar cambiando rápidamente a un repositorio de datos de respaldo, evitando así las interrupciones prolongadas de conectividad de un sistema o la perdida de datos en el peor de los casos (Fiade et al., 2019).

Para profundizar en las aplicaciones y formas de implementar el *failover* en una base de datos, en el presente documento se expone su implementación utilizando Docker con una imagen de MySQL y se detalla la configuración necesaria para el proceso. A su vez, se presentan distintas pruebas que aseguren el funcionamiento del *failover* en una base de datos secundaria. Por último, se realiza una breve investigación con respecto a las diferentes ventajas y desventajas del *failover*, su escalabilidad y rendmiento en un sistema, y diferentes alternativas a este método.

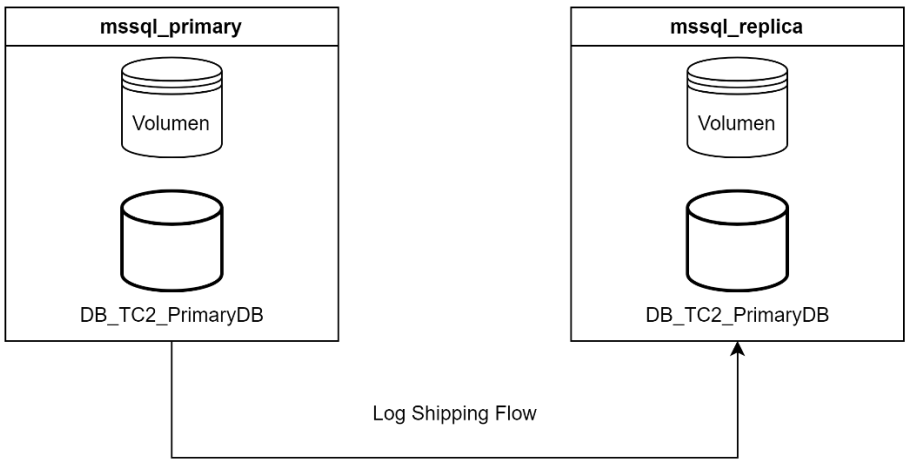
2. Procedimientos de Implementación

2.1 Configuración de Docker y Base de Datos

Se configuraron dos contenedores Docker para SQL Server utilizando un archivo docker-compose.yml. El primer contenedor, mssql_primary, aloja la base de datos primaria DB_TC2_PrimaryDB, mientras que el segundo contenedor, mssql_replica, alberga la réplica de la base de datos configurada para Log Shipping. Cada contenedor se configuró con variables de entorno específicas para aceptar la EULA de SQL Server y establecer la contraseña de administrador.

Los contenedores se levantaron con el comando docker-compose up -d, lo que permitió la creación automática de los volúmenes persistentes para almacenar los datos de las bases de datos, asegurando la persistencia de los datos incluso si los contenedores se detienen o reinician.

2.2 Diagramas de Arquitectura



El diagrama de arquitectura muestra la configuración de Log Shipping entre las instancias primaria y secundaria en contenedores Docker. El diagrama destaca la comunicación entre los contenedores a través de una red compartida, los volúmenes de datos montados en cada contenedor, y las rutas de backup de los logs de transacciones.

Los contenedores están configurados para permitir la conexión a través de puertos específicos (1435 para la primaria y 1436 para la secundaria), asegurando la replicación continua y sincronización de los datos mediante la transferencia de logs de transacciones.

3. Resultados de Pruebas

Las pruebas realizadas incluyeron la inserción de datos en la base de datos primaria y la verificación de la replicación de estos datos en la secundaria. Se forzaron backups y restauraciones manuales para validar la sincronización continua de la base de datos secundaria con los logs de transacciones del primario.

Las pruebas también incluyeron la simulación de un fallo en la instancia primaria deteniendo el contenedor mssql_primary, tras lo cual se completó la restauración de la base de datos secundaria para ponerla en modo de recuperación completa. Esta prueba confirmó la capacidad de la secundaria para asumir el control y servir datos sin la intervención del primario, validando la configuración de failover manual.

4. Análisis Critico

La configuración de Log Shipping en SQL Server es una solución eficaz para asegurar la alta disponibilidad y la recuperación de desastres, especialmente en entornos donde la simplicidad y la replicación a nivel de logs son suficientes para los requisitos de negocio. Sin embargo, este enfoque tiene sus limitaciones, como la falta de un failover completamente automático y la necesidad de intervenciones manuales para restaurar la disponibilidad completa de la base de datos secundaria.

Comparado con otras soluciones de replicación como Always On Availability Groups, Log Shipping es más sencillo de implementar y administrar, pero puede no satisfacer los requisitos de recuperación y disponibilidad en tiempo real para aplicaciones críticas. Es una opción viable para entornos que pueden tolerar algún nivel de interrupción y donde los recursos para configuraciones más avanzadas son limitados.

Se recomienda esta configuración para escenarios de prueba, desarrollo, o aplicaciones de producción con requisitos de replicación menos estrictos. Para despliegues en entornos de misión crítica, se debería considerar una evaluación más exhaustiva de opciones de replicación más robustas.

4.1 Ventajas

La implementación del *failover* en una base datos tiene como principal enfoque la recuperación automática de la información en caso de fallos del sistema durante su funcionamiento o del servidor primerio donde se encuentre el repositorio principal. A su vez, esto permite la continuidad del sistema ante fallos, puesto que la disponibilidad

ininterrumpida de los datos ante los fallos mantiene un funcionamiento constante del servicio evitando inconvenientes y caídas del sistema de forma inesperada, reduciendo el tiempo de inactividad del servicio. Por otro lado, este método implementa una protección sobre los datos almacenados, que en casos de error el repositorio secundario recibe toda la información, obteniendo una copia reciente de los datos y minimizando los daños sobre el sistema en cuestión (Singh et al., 2013).

4.2 Desventajas

A pesar de todas las ventajas que brinda su implementación, a la hora de utilizar una base de datos que aplique el *failover* para la reposición de sus datos es necesario tomar en cuenta que el respaldo que realiza el sistema no replicará el estado más reciente de los datos al momento del fallo, dado que esta recuperación se realiza mediante el ultimo guardado o copia de la base de datos principal, lo cual implica una regresión temporal en la información y en el estado del sistema, significando la perdida inevitable de cierta información, generando escenarios críticos para un servicio o sistema (Terneborg et al., 2021).

Otro aspecto a tomar en cuenta es la sobrecarga de datos que se genera en una base de datos al integrar la trasferencia constante de datos que requiere el *failover* entre las instancias primarias y secundarias, lo cual produce un uso de recursos de red desmedido en el sistema y una gran necesidad de almacenamiento, provocando una baja en el rendimiento general del sistema (Terneborg et al., 2021).

4.3 Rendimiento y Escalabilidad

El rendimiento de una base de datos que implementa el *failover* depende de la capacidad de redistribuir eficientemente las cargas que se ejercen sobre los sistemas que

componen a una aplicación, donde, en caso de errores, es necesario que la optimización de los recursos restantes se realice de forma equilibrada para evitar sobrecargas y tiempos de respuesta prolongados, degradando la calidad y la continuidad del servicio.

En cuanto a la escalabilidad, los sistemas que implementan este método de replicación son capaces de ajustarse a un funcionamiento en tiempo real, minimizando al máximo la pérdida de datos y permitiendo que la conexión entre el repositorio principal y secundario continúe constantemente. Aún así, al escalar un sistema que implemente *failover* es necesario tener en cuenta el tipo de arquitectura del código, donde arquitecturas centralizadas pueden interferir con su funcionamiento y generar problemas como un cuello de botella.

4.4 Alternativas

Si bien es cierto que el *failover* es una de las técnicas predilectas de la replicación de bases de datos, existen diversas alternativas que tienen diferentes especificaciones que pueden llegar a ser más eficientes dependiendo de las condiciones de un sistema de software. Entre estas se encuentran:

4.4.1 Snapshot Replication

El método *snapshot replication* consiste en una copia programa de todos los datos del repositorio en un momento especifico y los distribuye a las réplicas, permitiendo una arquitectura simple que no genera sobrecarga de red a costa de generar inconsistencias temporales con mayor frecuencia (Pupezescu y Rădescu, 2016).

Esta técnica comparada al *failover* no garantiza el continuo funcionamiento de un sistema, no obstante, es más adecuada a un entorno donde no se prevea una alta demanda

del sistema, recortando los costos que una arquitectura que implementa el *failover* necesita para tener un funcionamiento eficiente.

4.4.2 Transactional Replication

El *transactional replication* propone una metodología en donde se propagan los datos del repositorio principal a las replicas de manera continua y en tiempo real, permitiendo una total consistencia entre la base de datos principal y las secundarias, reduciendo aún más el riesgo de perdida de datos al tener una copia por cambio en el repositorio principal (Pupezescu y Rădescu, 2016).

El funcionamiento que propone esta técnica es ideal para sistemas en donde sea fundamental asegurar la integridad de los datos en todo momento del sistema, aún así, las constantes actualizaciones de las replicas con respecto al repositorio original provoca una sobrecarga de redes y un impacto mayor a un sistema que los sistemas con failover a costa de maximizar la consistencia de los datos del sistema. Por el contrario, a pesar de que la funcionalidad del *failover* es similar, esta se centra en mantener la estabilidad de un sistema, por lo que su actualización de réplica no es tan constante y por ende no requiere de tantos recursos en comparación.

4.4.3 Merge Replication

Finalmente, la *merge replication* consigue que los servidores principales y las replicas realicen cambios en el sistema de forma independiente entre si, otorgando una gran flexibilidad en su funcionamiento, en especial para entornos distribuido, permitiendo el funcionamiento en conjunto de diferentes bases de datos y luego se sincronicen en un momento determinado (Pupezescu y Rădescu, 2016).

Su metodología basada en independencias se bastante utilizada en ambientes que requieren un sistema flexible, no obstante, introduce una alta complejidad al sistema en su funcionamiento al tener que cambiar entre bases de datos y luego sincronizarlas entre sí. En cambio, el *failover* no implica la interacción por ambas direcciones de las bases de datos, únicamente realizando un cambio cuando la base principal falla y es necesario un respaldo para asegurar un funcionamiento continuo del sistema, adaptándose más a sistemas que necesiten soportar una alta demanda y reducir tiempos de inactividad.

5. Conclusiones

El failover es una técnica de replicación con cierta complejidad que permite a muchos sistemas de bases de datos mantener la integridad de la información en escenarios críticos, que, a diferencia de otros métodos, ofrece una solución neutra y balanceada para asegurar la continuidad y estabilidad de un sistema, lo cual es necesario en entornos donde el tiempo de inactividad y la pérdida de datos podrían tener consecuencias graves.

Si bien es cierto que el failover puede requerir un uso intensivo de recursos, como mayor capacidad de almacenamiento y una constante transferencia de datos entre los servidores primario y secundario, su eficiencia y confiabilidad lo convierten en una herramienta valiosa. Aun así, su implementación a gran escala puede elevar considerablemente los costos, tanto en términos de infraestructura como de mantenimiento, lo que puede limitar su adopción en sistemas más pequeños o con recursos limitados. A pesar de estas desventajas, el failover sigue siendo una opción adaptable y segura para la mayoría de los sistemas modernos, ofreciendo una protección robusta contra fallos imprevistos y garantizando la continuidad operativa de servicios críticos.

6. Referencias

Dharam, P., & Dey, M. (2021). *A mechanism for controller failover in distributed software-defined networks*. En *2021 8th International Conference on Computer and Communication Engineering (ICCCE)* (pp. 196-201). IEEE. https://doi.org/10.1109/ICCCE50029.2021.9467174

Dharam, P., & Dey, M. (2021). *A mechanism for controller failover in distributed software-defined networks*. En *2021 8th International Conference on Computer and Communication Engineering (ICCCE)* (pp. 196-201). IEEE. https://doi.org/10.1109/ICCCE50029.2021.9467174

Fiade, A., Agustian, M. A., & Masruroh, S. U. (2019). *Analysis of failover link system performance in OSPF, EIGRP, RIPv2 routing protocol with BGP*. 7th International Conference on Cyber and IT Service Management (CITSM).

Pupezescu, V., & Rădescu, R. (2016). The influence of data replication in the knowledge discovery in distributed databases process. En ECAI 2016 – 8th International Conference on Electronics, Computers and Artificial Intelligence (pp. 1-6). IEEE. https://doi.org/10.1109/ECAI.2016.7877104

Singh, T., Sandhu, P. S., & Bhatti, H. S. (2013). *Replication of data in database systems for backup and failover – an overview*. International Journal of Computer and Communication Engineering, 2(4), 535-538. https://doi.org/10.7763/IJCCE.2013.V2.243

Terneborg, M., Karlsson Rönnberg, J., & Schelén, O. (2021). *Application agnostic container migration and failover*. 46th Conference on Local Computer Networks (LCN), IEEE.