> ***FECHA: 23 DE SETIEMBRE, 2022***
> ***ESTUDIANTE: VALERIA SELL SÁENZ***

Información acerca del proyecto

Callback: Llamada asincrónica que toma el mensaje y lo convierte en un .json. El **Dockerfile** es el encargado de empaquetar la aplicación. Además, las imágenes se publican en *Docker Hub* a medida de que se vayan construyendo. Una vez que estas son publicadas se levanta la aplicación. *Nota*: Las variables de entorno deben ser inyectadas al consumidor.

Primer paso: Incluir lo realizado en la Tarea Corta (monitoreo y bd), los nombres son libres pero es importante que sean significativos. Los nombres de los *helmcharts* definen los nombres de los recursos a usar.

Para automatizar el proceso, se deben conocer los nombres de los recursos a crear de antemano. Así se puede configurar el *secret* y el rervicio del *RabbitMQ*. Si use sa el *ElasticSearch* de ECK Operator, este genera los nombres con los valores que tiene por defecto. Es decir, todos los servicios llevan ese nombre por delante, de manera que son sencillos de identificar.

Detalle importante: En el caso de realizar una desintalación, el disco virtual permanece. Esto puede llegar a generar un conflicto de contraseñas ya que el sistema intentará usar el disco anterior. Por ello, se recomienda borrar también el disco virtual (persistent volume) antes de volver a instalar.

Materia de Clase

Perfomance

ACID, ¿por qué es importante?

Esto define el por qué una base de datos relacional no se comporta tan bien como una no-sql. Esto es debido a que las no-sql tienden a borrar una de estas características.

Atomicity

Las operaciones se ejecutan en un ciclo de reloj y son indivisibles. El problema radica en que operaciones se han vuelto cada vez más complejas, por lo que un ciclo de reloj no es suficiente tiempo y se corre el riesgo de perder el procesador en media transacción.

Un "Acceso Exclusivo" para un recurso compartido es difícil de garantizar si las operaciones no son atómicas. La idea consiste en garantizar que un conjunto de sentencias sean ejecutadas de manera atómica.

Consistency

¿En qué consiste hacer un commit? Es hacer un cambio *persistente*. Al ejecutar una transacción y hacer un commit, ese cambio perdurará pase lo que pase. Si esto no sucede, la base de datos queda en un estado *inconsistente*. Para tener consitencia, es importante tener atomicidad. Para una base de datos no-sql, la consistencia no es tan importante ya que es lo "más libre".

Rollback en Cascada: Un rollback es una operación que devuelve la base de datos a algún estado anterior. Por tanto, un rollback en cascada sucede cuando una transacción (T1) falla y se debe hacer un rollback. Otras transacciones que dependen de las

acciones de T1 también deben revertirse debido a la falla de T1, provocando un efecto en cascada. [1] Una base de datos puede ser diseñada para que no se de el rollback en cascada. **lock**: es el bloque de un recurso compartido para establecer la necesidad de uso y regularlo. **read**: lock compartido **write**: lock exclusivo

Compatibilidad de locks: | | read | write | | --- | --- | --- | | **read** | X | --- | | **write** | --- | --- |

Una operación read no es compatible con una operación write debido a la consistencia, ya que el valor del dato puede cambiar.

Isolation

Una transacción siempre debe estar corriendo sola, con la intención de no tener conflictos por medio de locks, pero eso realmente no se da. En su lugar, se da la *apariencia* a la transacción de que está sola.

Durability

La base de datos debe permanecer ante fallas de poder, caídas, errores, usuarios... Una vez que una transacción es ejecutada y se le hace commit, se debe garantizar que la transacción va a perdurar.

¿Cuál es el problema con los locks?

- **Overhead**
- **Blocking**: Baja el rendimiento de la base de datos.

Multiversioning: Como su nombre lo dice, es la creación y manejo de varias versiones de un producto. Esto elimina el concepto del lock de read en la mayor parte de los casos. *Deadlocks*: Sucede cuando dos entidades usan los mismos recursos compartidos en orden diferente. Una posible solución para evitarlos, es procurar que las entidades usen los recursos en el mismo orden. *Two-Phase Locking*: Reduce la probabilidad de un deadlock. Sus dos fases son Expanding/Growing y Shrinking/Retracting.

Tipos de Two-Phase Locking

- **Conservative**: Obtiene todos los locks anten de que inicie la transacción. Sus problemas se traducen en un timeout para los usuarios y en caídas en rendimiento para los desarolladores.
- **Strict**: Los locks adquiridos de tipo exclusivo se liberan solo cuando la transacción termine. Su ventaja es que evita los rollbacks en cascada pero su desventaja radica en que comparte los problemas del tipo Conservador.
- **Strong-Strict Two-Phase**: Suelta los locks hasta el final de la transacción, los va solicitando a medida que va avanzando.

Si la base de datos está en varios servidores, se necesita de un coordinador. **Two-Phase Commit Protocol**

1. *Commit Request Phase*: redo log - undo log.
2. *Commit Phase*: Failure o Success.

El Coordinador hace una petición a que sus nodos ejecuten la transacción distribuida, con dos posibles resultados: funciona o no funciona. Los nodos llegan hasta el punto del commit, dejando los recursos bloqueados. Cuando el coordinador recibe los resultados, es todo o nada. Con solo un nodo que no haya logrado la transacción, el commit phase arranca en modo failure (se hace rollback y se liberan recursos). Si todos son exitosos, entonces se arranca en modo success (se hace commit a disco y se hace la transacción persistente).

- Redo log: operaciones que se van a ejecutar en la base de datos para hacer el commit.
- Undo log: operaciones que se van a ejecutar en la base de datos para hacer el rollback.

En este protocolo, se prepara para ambos escenarios.

Referencias

> 1. *tok.wiki. (n.d.). Rollback (gestión de datos) Retroceso en cascadaySQL. Retrieved October 4, 2022, from https://hmong.es/wiki/Rollback_(data_management)*