Apuntes Bases de Datos II - Clase 10/03/2023

Gerald Núñez Chavarría - 2021023226

Profesor: Nereo Campos Araya.

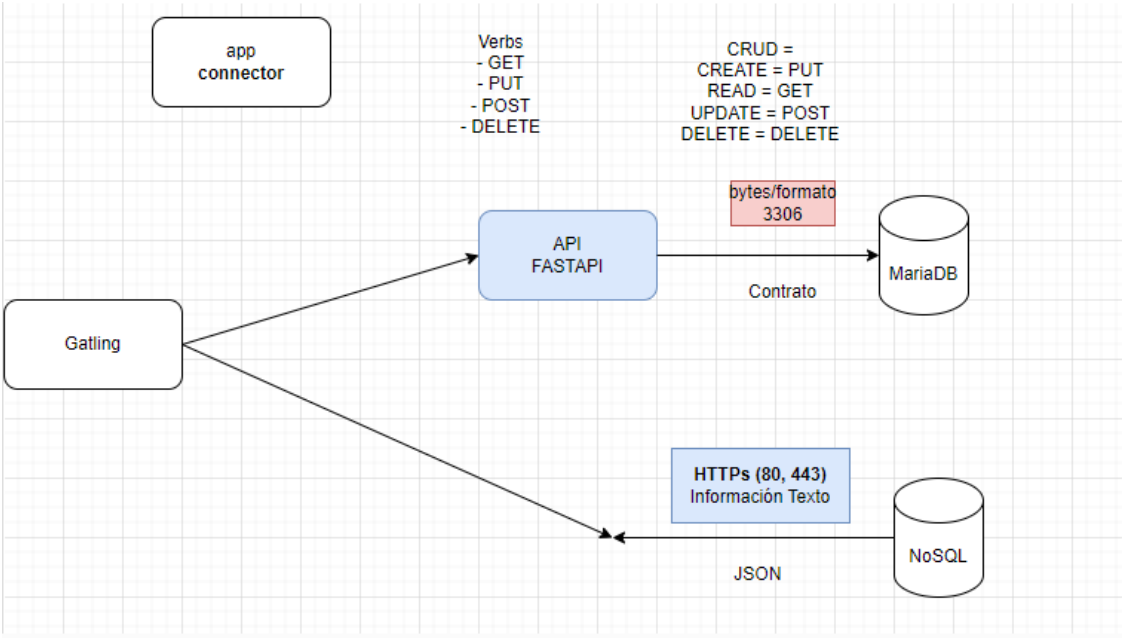
1. Dudas sobre la tarea corta Observability.

1. Para saber la versión que se debe instalar para la base de datos se puede entrar al link que viene en la especificación de cada una de las bases de datos, de ahí se busca el link del repositorio y se desciende hasta encontrar las versiones.
   Otra opción es ejecutar el comando 'helm repo add bitnami https://charts.bitnami.com/bitnami' esto para añadir el repositorio de bitnami si no lo tienen, y luego 'helm search repo 'nombre de la bd''.
2. Una base de datos relacional como mariadb expone el puerto 3306 y envía información en bytes, por eso se necesita un connector para comunicarse entre la aplicación y la base. Pueden buscar mariadb python client para obtener información de como conectarse. Utiliza CRUD (create, read, update and delete) para comunicarse.

   La mayoría de bases de datos no sql utilizan el protocolo http como protocolo de comunicación. Se comunican mediante archivos json y utilizan el puerto estándar http. Utilizan verbos para comunicarse que son get, put, post and delete. Estos cuatro verbos realizan la misma acción que los cuatro CRUD respectivamente.

   En cuanto a la tarea, se necesita que Gatling realice pruebas de carga para diferentes bases de datos. Gatling realiza las pruebas de carga conectándose a un endpoint http. Con las bases no sql se comunica de manera directa, pero para las sql necesita una api intermediaria para poder comunicarse.

   Observe el siguiente diagrama de ejemplo:

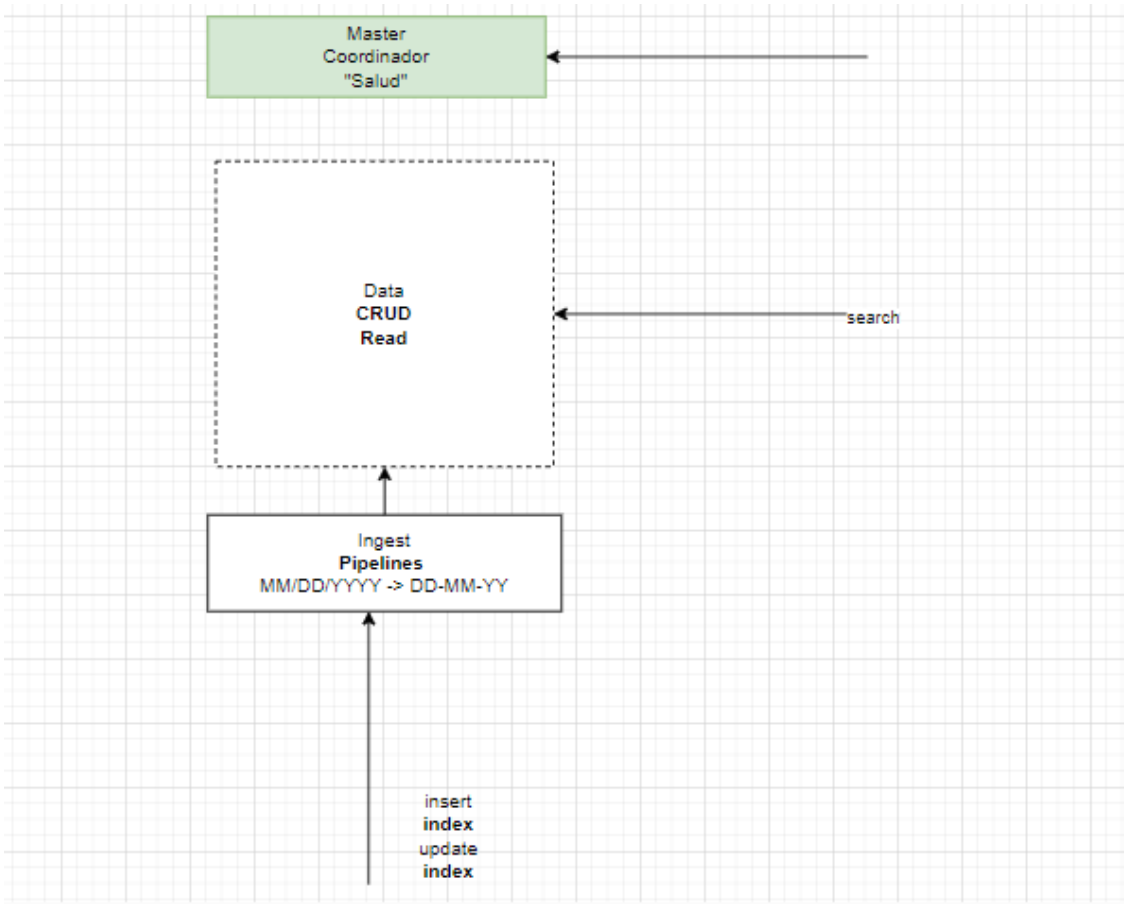


3. Las pruebas unitarias (unit test) se implementan para probar una parte del código de manera aislada. Si la parte del código que se esta probando genera la salida correcta, entonces se libera el código con los cambios y se vuelve a integrar, si no, entonces no se puede liberar hasta que se corrija. Dependiendo de la empresa una prueba unitaria puede pasar por Unittest, Smoke, Test, Integration Test, Staging Test, QA y Proceso Formal.

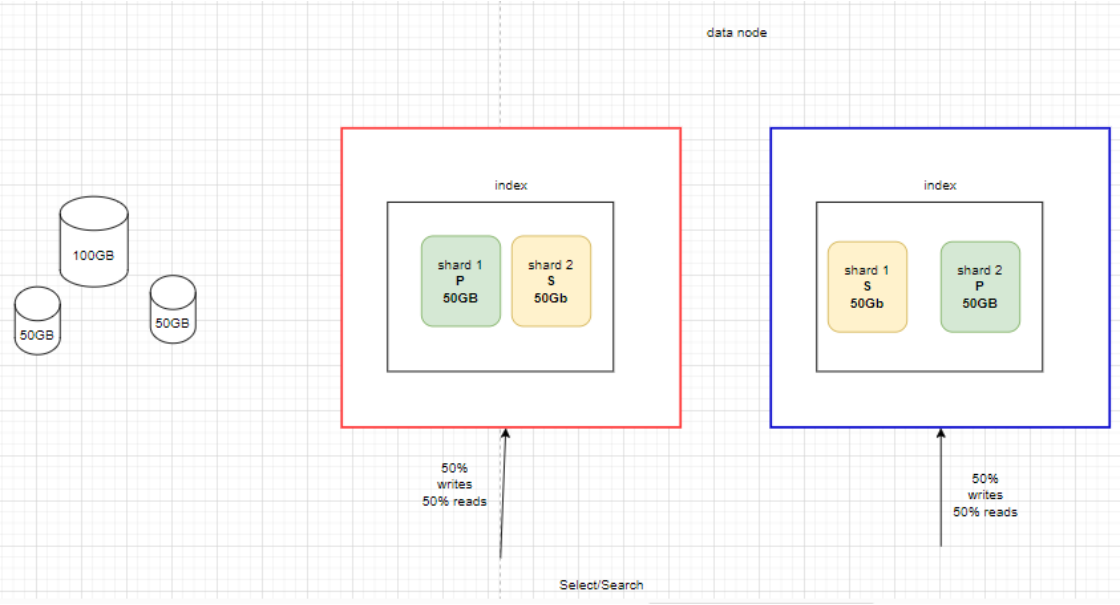
2. Desarrollo de la clase.

1. Arquitectura de Elasticsearch: Tiene tres tipos básicos de nodo: 1- Master: El que coordina y revisa el estado de "salud" del clúster. 2- Data: Realizan pipelines. 3- Ingest: Realizan CRUD.

   Al proceso de insertar o actualizar un dato se le denomina "index" en Elasticsearh. Lo ideal es que un proceso entre primero al ingest para hacer las conversiones de datos, luego al data para que realice la operación respectiva. También es ideal que el master se encargue solo de su función y no reciba operaciones. Cuando se realiza una búsqueda (search) es ideal entrar entondes directamente al nodo de data.



2. Rendimiento de Elasticsearch: Recurrir a un solo nodo de Data en Elasticsearch, con un shard puede causar un ponto de falla único, ya que al fallar el nodo no hay manera de recuperar el dato. Por eso, se recomiendo repartir recursos y tener al menos dos data nodes, pero que cada uno tenga dos shards, un principal y otro secundario, para dividir en un 50% el funcionamiento del sistema. Además también dividir la memoria para asignarle una parte igual a cada nodo. Se pueden implementar más nodos dependiendo de las necesidades del sistema, pero siempre teniendo en cuenta que lo ideal es tener tantos shards como nodos. A continuación un ejemplo ilustrado:



3. En las bases de datos NO SQL o SQL pero a gran escala los datos que se manejan siguen un patrón de tiempo, entre más cercanos a la hora actual tienen más relevancia. Recordar que existen datos que se pueden llamar hot (mucha lectura y escritura), warm (), cold, fronzen y glasier. Idealmente sería tener la misma cantidad de memoria como de Raw Data en disco, pero como esto es sumamente caro casi no sucede, entonces se han creado recomendaciones de la cantidad de disco que se debería tener por cada 1gb de memoria según el dato: hot = 30, warm = 60, cold= 90, frozen = 120 y glacier = se asume que no hay memoria.
4. Elasticsearch utiliza los tipos de datos de la siguiente manera:

   Content: Tipo de dato que es muy leído y modificado. Entonces se necesitan muchas lecturas y escrituras, esto requiere mucha memoria, cpu y espacio en discos.

   Hot: Son datos que se leen bastante pero la escritura es moderada.

   Warm: Se leen de manera moderada y la escritura de estos datos es muy baja.

   Cold: Son datos de solo lectura.
   Frozen: Aquí se implementa un Object Storage, que es una herramienta de la nube que funciona como una interfaz http donde se exponen los verbos de comunicación get, put, post y delete. Entonces un dato frozen se guarda un object storage y cada vez que se va a utilizar se tiene que ir a traer desde la red y montarlos en el servidor solo para la búsqueda, luego se desmontan, se borra y se sigue trabajando. Estas búsquedas son muy caras y son llamadas Fully mounted indexes.