Captura y visualización en tiempo real de eventos en MongoDB: Integración con Elasticsearch mediante Change Streams

Julio de 2023

Información de los autores:

Ing. Arturo Castagnino (arturocastagnino@gmail.com) - Maldonado, Uruguay
Maestría en Ciencia de Datos y Aprendizaje Automático
Facultad de Ingeniería - Universidad de la República

Ing. Franco Fontana (franco@fontana.com.uy) - Montevideo, Uruguay
Maestría en Ciencia de Datos y Aprendizaje Automático
Facultad de Ingeniería - Universidad de la República

Resumen

En este artículo, se presenta una arquitectura orientada a eventos que combina la potencia de MongoDB, una base de datos NoSQL altamente escalable, con Elasticsearch, una herramienta de búsqueda y análisis de datos en tiempo real. Esta arquitectura utiliza el mecanismo de "Change Streams" de MongoDB para capturar eventos en colecciones específicas. A medida que los eventos ocurren, se envían casi en tiempo real a una base de datos Elasticsearch para su procesamiento y visualización en tableros interactivos.

1. Introducción

La necesidad de capturar y analizar eventos en tiempo real es fundamental en muchos escenarios de aplicaciones modernas. En este sentido, la combinación de MongoDB y Elasticsearch proporciona una solución potente y escalable. MongoDB, una base de datos NoSQL flexible y de alto rendimiento, ofrece el mecanismo de "Change Streams" que permite la observación de cambios en las colecciones sin pérdida de datos. Por otro lado, Elasticsearch es una herramienta de búsqueda y análisis distribuida, diseñada para el procesamiento eficiente de grandes volúmenes de datos en tiempo real, que en conjunto con Kibana como herramienta de visualización, permite al usuario realizar un análisis exhaustivo de la información.

2. Motivación

Contar con un servicio que transmita los cambios casi en tiempo real desde una base de datos MongoDB hacia un clúster EK (Elasticsearch - Kibana) resulta de gran interés. Esto permite tomar decisiones oportunas, monitorear eventos en tiempo real, realizar análisis actualizados y visualizar datos en tableros interactivos. La transmisión casi instantánea de datos facilita la toma de decisiones basadas en información actualizada y precisa, el monitoreo en tiempo real, el análisis en tiempo real, la exploración de datos interactiva y la capacidad de manejar grandes volúmenes de datos de manera eficiente.

3. Análisis de Opciones Disponibles

Para definir una solución concreta al problema planteado se realiza un breve estudio de las opciones disponibles. A continuación se mencionan algunas de ellas conforme a los artículos:

- https://code.likeagirl.io/5-different-ways-to-synchronize-data-from-mongodb-to-elasticsearch-d8456b83d44f
- https://www.linkedin.com/pulse/5-way-sync-data-from-mongodb-es-kai-hao/

3.1. Mongo-connector

Mongo-connector[1] es un servicio que permite la sincronización de datos en tiempo real entre un clúster de MongoDB y otros sistemas de bases de datos como Elasticsearch u otro clúster de MongoDB. Constituye una poderosa herramienta para propagar los cambios realizados en las colecciones de MongoDB a otros sistemas y viceversa, y se usa comúnmente en escenarios donde existe la necesidad de integrar MongoDB con motores de búsqueda, bases de datos o plataformas de análisis, entre otros.

Ventajas:

- Servicio de sincronización en tiempo real.
- Puede crear una canalización desde un clúster de MongoDB a uno o más sistemas de destino.

Desventajas:

- Necesita MongoDB para ejecutarse en modo *replica set*.
- Necesita un paquete adicional llamado "elastic2_doc_manager"[2] para escribir datos en Elasticsearch.

3.2. Plugin para Elasticsearch "elasticsearch-river-mongodb"

Elasticsearch brinda la capacidad de mejorar la funcionalidad básica mediante complementos, que son fáciles de usar y desarrollar. Se pueden usar para análisis, descubrimiento, monitoreo, sincronización de datos y muchos otros. Este plugin[3] escrito en Java, monitorea las colecciones y propaga las nuevas operaciones automáticamente a Elasticsearch.

Ventajas:

- Fácil de usar.

Desventajas:

- No soporta objetos anidados.
- No tiene mantenimiento activo desde hace años.

3.3. Plugin JDBC para Logstash

Logstash[4] es un componente del stack ELK (Elastisearch/Logstash/Kibana), el cual crea un pipeline para el procesamiento de datos desde diversas fuentes, los transforma y luego los envía a Elasticsearch.

Ventajas:

- Es muy flexible y se puede configurar para manejar una amplia variedad de fuentes y formatos de datos.
- Se puede escalar fácilmente de forma horizontal para manejar grandes volúmenes de datos.

[1] `https://github.com/yougov/mongo-connector/wiki`
[2] `https://github.com/yougov/elastic-doc-manager`
[3] `https://github.com/richardwilly98/elasticsearch-river-mongodb`
[4] `https://www.elastic.co/guide/en/logstash/current/plugins-inputs-jdbc.html`

- Proporciona un poderoso conjunto de filtros que se pueden usar para transformar y enriquecer los datos antes de indexarlos en Elasticsearch. Esto puede incluir tareas como limpieza, análisis y normalización de datos.
- Brinda capacidades detalladas de monitoreo y registro que se pueden usar para solucionar problemas y optimizar el rendimiento.

Desventajas:

- Puede ser complejo de instalar y configurar, especialmente para los usuarios que no están familiarizados con su sintaxis y opciones de configuración.
- Mientras ejecuta, puede presentar sobrecargas de rendimiento, especialmente cuando se trata de grandes volúmenes de datos.
- Puede consumir muchos recursos y requiere una cantidad significativa de memoria y potencia de CPU para ejecutarse de manera efectiva.

3.4. MongoDB Atlas Search

No podía faltar mencionar este servicio[5] de MongoDB que incorpora un índice de búsqueda de Apache Lucene (similar a Elasticsearch) directamente junto con la base de datos. Los datos se sincronizan automáticamente entre los dos sistemas, los desarrolladores trabajan con una sola API, no hay un sistema separado para ejecutar y pagar, aliviando la carga operativa. En este caso no hay necesidad de sincronizar operaciones con Elasticsearch/Kibana, se trabaja únicamente con MongoDB.

Ventajas:

- Integrado con MongoDB.
- Al estar integrado con MongoDB puede escalar automáticamente para manejar grandes volúmenes de datos.
- Proporciona una búsqueda de texto completo que puede manejar consultas complejas y proporcionar resultados precisos para consultas que incluyen múltiples palabras clave.
- Fácil de configurar y utilizar, especialmente para los usuarios que ya están familiarizados con MongoDB.

Desventajas:

- Menos funcionalidad que Elasticsearch, aunque MongoDB Atlas Search proporciona una funcionalidad de búsqueda de texto completo, no ofrece todas las características avanzadas que Elasticsearch proporciona, como agregaciones complejas, análisis de texto y enriquecimiento de datos.
- Menos maduro que Elasticsearch, aunque MongoDB Atlas Search ha mejorado significativamente en los últimos años, todavía es una tecnología relativamente nueva y no tiene la misma madurez y estabilidad que Elasticsearch.
- Dependencia de la nube de MongoDB, MongoDB Atlas Search está disponible solo en la plataforma de MongoDB Atlas y no se puede utilizar fuera de esta plataforma, lo que puede ser una limitación para algunos usuarios.
- No se dispone de herramientas de visualización predefinidas tales como Kibana para la construcción de visualizaciones y tableros.

[5] `https://www.mongodb.com/compare/mongodb-atlas-search-vs-elastic-elasticsearch`

3.5. Transporter

Transporter[6] es una herramienta de sincronización de datos que permite a los usuarios replicar datos de diversas fuentes, incluyendo bases de datos MongoDB y Elasticsearch.

Ventajas:

- Es fácil de configurar y utilizar, lo que lo hace ideal para usuarios que no están familiarizados con herramientas más complejas como Logstash o la API de replicación de MongoDB.
- Puede replicar datos de diversas fuentes, lo que permite a los usuarios sincronizar datos de múltiples fuentes en Elasticsearch.
- Está diseñado para ser escalable y puede manejar grandes volúmenes de datos, lo que lo hace ideal para aplicaciones que requieren una alta disponibilidad y rendimiento.
- Está diseñado específicamente para trabajar con bases de datos MongoDB, lo que significa que los usuarios pueden aprovechar la funcionalidad integrada de MongoDB para replicar datos.

Desventajas:

- Limitaciones de funcionalidad: aunque Transporter es fácil de usar y escalable, no proporciona la misma funcionalidad avanzada que herramientas como Logstash o la API de replicación de MongoDB.
- Transporter es un software de terceros, desarrollado por la empresa Compose[7] adquirida por IBM[8], lo que significa que los usuarios pueden estar sujetos a limitaciones y dependencias de la herramienta.

Existen más opciones que por razones de alcance no corresponde analizar en profundidad, tales como el uso de colas de mensajes Kafka[9] para guardar cambios, triggers[10] de MongoDB, o la librería nodejs Mongoosastic[11], la cual se utiliza como plugin de la conocida librería Mongoose[12] para conectarse a MongoDB y definir esquemas.

4. Trabajo Realizado

El trabajo se centra principamente en el estudio de los mecanismos de sincronización existentes entre MongoDB y Elasticsearch. Luego de analizar distintas posibilidades, se halló que MongoDB tiene una funcionalidad nativa y específica llamada "Change Streams"[13] para la captura de eventos tanto a nivel de una base de datos como de una colección específica la cual, si es utilizada adecuadamente, garantiza la no pérdida de datos.

El trabajo simula una situación hipotética donde existe una base de datos MongoDB que sistemáticamente recibe datos desde una fuente determinada, los cuales deben ser transmitidos no bien sea posible a un stack Elasticsearch/Kibana para posteriormente realizar búsquedas e implementar visualizaciones.

La justificación para la elección de "Change Streams" se basa en:

I. *Near Real-Time Replication*: las secuencias de cambios de MongoDB permiten la replicación casi en tiempo real de los datos de MongoDB en una base de datos Elasticsearch. Esto significa que, dependiendo del hardware y tiempos de red, los datos se pueden indexar y buscar en Elasticsearch casi inmediatamente después de haberse actualizado en MongoDB.

[6] https://github.com/compose/transporter
[7] https://compose.io/
[8] https://www.prnewswire.com/news-releases/ibm-acquires-compose-to-expand-cloud-data-services-300117883.html
[9] https://kafka.apache.org/
[10] https://www.mongodb.com/docs/atlas/triggers/
[11] https://github.com/mongoosastic/mongoosastic/
[12] https://mongoosejs.com/
[13] https://www.mongodb.com/docs/manual/changeStreams/

II. Integración nativa: "Change Streams" es una característica nativa de MongoDB, lo que significa que no se necesita una herramienta de terceros para realizar la sincronización entre MongoDB y Elasticsearch. Esto reduce la complejidad de la arquitectura y reduce la cantidad de herramientas y software que se necesitan para mantener el sistema.

III. Escalabilidad: "Change Streams" se integran directamente con la arquitectura de replicación de MongoDB, lo que permite la escalabilidad horizontal y la distribución de datos en múltiples nodos de Elasticsearch.

IV. Flexibilidad: "Change Streams" permite la sincronización selectiva de datos, lo que significa que los usuarios pueden elegir qué datos se sincronizan entre MongoDB y Elasticsearch. Esto puede ayudar a reducir la cantidad de datos que se envían a Elasticsearch y reducir el costo de almacenamiento y procesamiento de datos.

V. Seguimiento de cambios: "Change Streams" proporciona un registro completo de todos los cambios en MongoDB, lo que facilita el seguimiento de los cambios y la solución de problemas en caso de errores o problemas de sincronización.

5. Componentes de la solución

5.1. Datos

La fuente de datos seleccionada a los efectos del presente trabajo proviene de las API[14] contenidas en el sitio web OpenAQ[15], organización sin fines de lucro que integra, en una plataforma gratuita, datos abiertos de calidad de aire provenientes de estaciones de monitoreo en más de 136 países de todo el mundo. Cualquier usuario puede tener acceso sin restricciones a los datos para su conocimiento, análisis y comunicación.

En tanto se trabajará con datos de calidad de aire y, como fuera arriba expuesto, se tienen datos de estaciones en centenares de ciudades, se decidió realizar una comparativa de la calidad del aire en las principales ciudades de América Latina en las cuales existen mediciones, tomando en consideración solamente una ciudad por país, enfatizando la elección de las ciudades capitales cuando ello fuera posible[16].

En tal sentido, se seleccionaron las siguientes ciudades:

- Buenos Aires, Argentina.
- Santiago, Chile.
- San Pablo, Brasil.
- Sucre, Bolivia.
- Lima, Perú.
- Guayaquil, Ecuador.
- Bogotá, Colombia.
- Puerto Ordaz, Venezuela.

A continuación se expone en un mapa de América Latina la ubicación de las ciudades seleccionadas.

[14] Application Programming Interfaces

[15] `https://openaq.org`

[16] Se entiende pertinente observar que Montevideo ni ninguna ciudad o localidad de Uruguay cuenta con datos de calidad de aire en la fuente seleccionada

Figura 1: Ubicación de las estaciones de medición de calidad del aire

Con la idea de limitar el alcance del presente trabajo, de la lista de párametros de calidad de aire que se miden en las estaciones relevadas por OpenAQ, se seleccionan los parámetros PM10 y PM25. Los parámetros PM10 y PM25 refieren al material particulado (particularmente a las fracciones de tamaño menor a 10 micras y 2,5 micras, respectivamente). El material particulado constituye un buen indicador del estado de calidad del aire y es el contaminante que más afecta a los seres humanos. Este contaminante tiene su origen o emisión al aire a partir de diversas fuentes, como el tránsito vehicular y la erosión eólica sobre caminos sin pavimentar, las obras de construcción, la quema de combustibles fósiles, el acopio de materiales granulares, los incendios forestales, así como también los procesos de quema de biomasa, entre otros. En los seres humanos el PM10 puede viajar a través del sistema respiratorio y alojarse dentro de los pulmones, mientras que el PM25 pueden atravesar la barrera pulmonar y entrar en el sistema sanguíneo. La exposición crónica a partículas contribuye al riesgo de desarrollar enfermedades cardiovasculares y respiratorias, así como cáncer de pulmón.[17]

Estos datos se cargan en una colección de MongoDB para lo cual se diseña un script que ejecuta a demanda (actuando como un simulador para la generación de eventos en MongoDB). Luego, estos eventos son propagados hacia Elasticsearch (por otro script) sin pérdida alguna de datos.

5.2. Bases de datos

La solución va a contar con las bases de datos que a continuación se exponen.

5.2.1. MongoDB

MongoDB recibe los datos de calidad de aire, los almacena en una colección y disponibiliza el mecanismo de "Change Streams" para su uso en la propagación de los datos hacia Elasticsearch.

[17] Wark and Warner, C.F. (1999). Contaminación del aire, su origen y control. Editorial Limusa S.A. – Grupo Noriega Editores. Balderas 95, México D.F.

5.2.2. Elasticsearch

Elasticsearch recibe en casi tiempo real los datos que son propagados por "Change Streams" desde MongoDB y los almacena en un índice.

5.3. Kibana

Complementariamente a la instancia de Elasticsearch, se disponibiliza una instancia de Kibana para la visualización y análisis de los datos cargados.

5.4. Scripts

Todos los scripts están implementados en Python y almacenados en un repositorio GitLab[18]. Aquellos scripts que se utilicen en ambientes de producción deberán ser desplegados en servidores o plataformas como Kubernetes u Openshift. A continuación se describe brevemente la función de cada uno de ellos y luego, en la sección de implementación, se profundiza acerca de cómo funcionan e interactúan entre sí.

La solución consta de varios scripts:

I. "mongoLoader.py": es un script que ejecuta a demanda y carga los datos de calidad de aire (utilizando la API que provee el sitio OpenAQ[19]) hacia la colección de MongoDB, que se denomina "material_particulado". Ésta será la colección sobre la cual se capturarán los cambios; cualquier cambio en esta colección va a ser reflejado en Elasticsearch.

II. "changeStream.py": es el script principal de la solución, contiene la lógica de captura de eventos y recuperación frente a fallos. Además contiene la lógica de limpieza y transformación de cada documento para llevarlo a Elasticsearch con los campos y la estructura que se necesita para luego implementar visualizaciones en Kibana.

III. "elastic.py": contiene la lógica para las operaciones CRUD sobre Elasticsearch

IV. "changeStreamTakeGustTime.py": básicamente es el mismo script que "changeStream.py" pero con lógica adicional para realizar las pruebas de desempeño.

V. "testDeleteRun.bat" y "testInsertRun.bat": archivos de lotes para lanzar las pruebas de desempeño.

6. Tecnología Utilizada

Como se mencionara anteriormente, el eje de la solución se basa en la tecnología "Change Streams" de MongoDB, la cual utiliza dos componentes adicionales, llamados "oplog" y "resume_token", para garantizar la captura sin pérdida de datos de los cambios que ocurren en una base de datos o colecciones MongoDB.

El "oplog" (registro de operaciones) es un registro especial en MongoDB que almacena todas las operaciones de escritura realizadas en la base de datos. "Change Streams" utiliza el "oplog" como una fuente de eventos para capturar los cambios en tiempo real. Cada cambio en la base de datos se registra en el "oplog" como una entrada individual.

Cuando se inicia un "Change Streams" y ocurren cambios, MongoDB proporciona un "resume_token"que representa la posición actual en el "oplog". Este token se guarda y se utiliza para reiniciar el "Change Streams" en caso de interrupciones o reinicios.

Durante el funcionamiento normal, el "Change Streams" monitorea constantemente el "oplog" y captura los cambios en tiempo real. Cada cambio capturado incluye información como el tipo de operación (inserción, actualización, eliminación, etc.), el documento afectado y los detalles relacionados. Las capturas que realiza son configurables por cualquier tipo de filtro que se quiera definir. Por ejemplo, según el tipo de operación, es

[18] `https://gitlab.com/fing4/bdnr_final`
[19] `https://openaq.org/`

posible configurar sólo inserciones y eliminaciones. También sería posible configurar que capture los cambios que ocurren cuando determinado valor en un campo supere un umbral predefinido.

Si ocurre una interrupción en el servicio de "Change Streams", como un corte de comunicación o una incidencia que detenga su funcionamiento, se puede reiniciar utilizando el "resume_token" guardado anteriormente. Al reiniciar, el "Change Streams" se coloca en la posición exacta del "oplog" donde se detuvo, lo que garantiza que no se pierdan cambios durante el tiempo fuera de servicio.

7. Implementación

7.1. Descripción general

La implementación de esta solución requiere la utilización de diversas tecnologías al mismo tiempo. Teniendo en cuenta las limitaciones de infraestructura que existen para desarrollar una solución de este tipo con los reducidos recursos que se cuentan en el marco de un trabajo universitario de investigación, se priorizó - donde fuera posible- la utilización de cuentas gratis en la nube de proveedores de las tecnologías involucradas.

Es así que, para la implementación, se cuenta con todos los componentes que se mencionaron en la sección anterior, de los cuales importa mencionar que:

- MongoDB: se despliega en un clúster MongoDB Atlas en la nube, plan gratis.[20]
- Elasticsearch/Kibana: se despliega un clúster (de un nodo) Elasticsearch en un playground Red Hat Openshift en la nube.[21]
- Los scripts Python y bat se ejecutan localmente en un equipo con conexión a Internet y acceso a ambas bases de datos remotas.

En la siguiente figura se muestra un esquema simple de cómo es la implementación de la solución con las tecnologías involucradas.

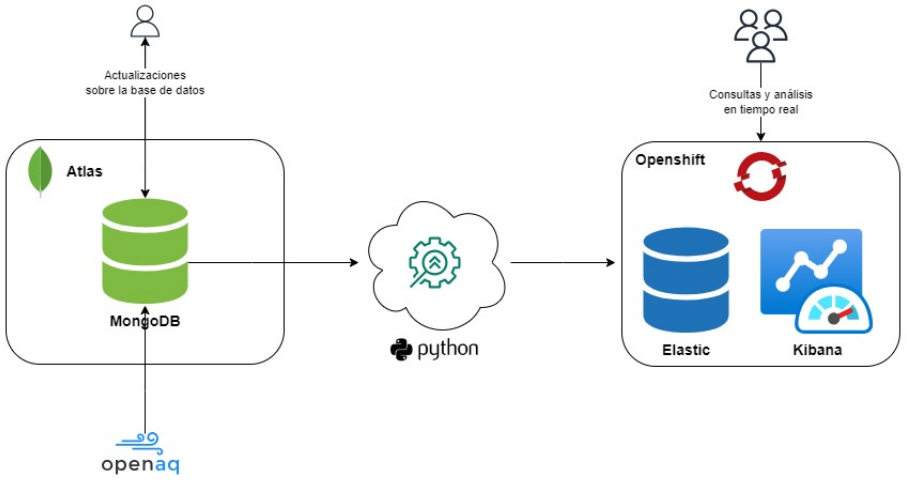

Figura 2: Esquema de la implementación

Para dejar esta solución preparada, se debe levantar el servicio de "Change Streams" para lo cual se ejecuta el script "changeStream.py". Este script establece la conexión con MongoDB y dispara un hilo de

[20] https://www.mongodb.com/atlas/database
[21] https://developers.redhat.com/courses/explore-openshift/openshift-playground

ejecución donde se implementa la lógica de detección de cambios. Este hilo queda ejecutando permanentemente y está configurado para capturar los cambios que ocurren en la colección "material-particulado", la cual tiene todos los datos de calidad de aire que interesa propagar hacia Elasticsearch. En cualquier momento es posible detener la ejecución del proceso de captura pulsando la tecla enter en la terminal.

La función "look_for_changes()" es la que invoca el hilo principal y la primer actividad que realiza es detectar si existe un "resume_token". Para implementar la lógica de recuperación frente a fallos, se utilizó un archivo de texto auxiliar "resume_token.txt" ubicado físicamente junto al script "changeStream.py", el cual cumple la función de persistir el token actual de cada una de las operaciones que van ocurriendo en MongoDB. Así es que al comienzo de la función, se abre el archivo de texto y se verifica si existe token o no. Si no existe token, simplemente comienza a ejecutar esperando por cambios; si existe token, lo lee y procesa todos los cambios que hayan ocurrido desde ese token en adelante y luego sigue esperando por más cambios.

Luego de comprobar la existencia de un token, la función ingresa al bucle "for change in change_stream:" donde quedará ejecutando en forma indefinida hasta que un operador pulse enter. Este bucle es quien procesa cada uno de los documentos que se detectan con cambios; en caso de haber obtenido un token, procesará todos los pendientes desde la última lectura, en caso contrario quedará a la espera de nuevos cambios. Es importante señalar que los cambios de los documentos se procesan uno a uno en tiempo real y en el mismo instante que van ocurriendo en MongoDB. Para el caso del presente trabajo se configuró "Change Streams" para que detecte únicamente inserciones, actualizaciones y eliminaciones de documentos. Se incluye también el tipo de operación reemplazo "replace", que se diferencia de una actualización en aquellos casos en que el documento se sustituye completamente por uno nuevo y no parcialmente (como ocurre en el caso de actualizaciones). Existen más opciones de captura de cambios que pueden consultarse en la documentación de "Change Streams".

Todos los cambios que "Change Streams" registra se almacenan en una colección auxiliar local en MongoDB llamada "material_particulado_log". Esta colección puede servir como mecanismo de control para que, en caso de fallos, se pueda tener un lugar donde estén persistidos todos los cambios que se propagaron a Elasticsearch. El documento tipo de "Change Streams" tiene una estructura determinada que incluye campos como "_id._data" (el token), "operationType" (el tipo de operación), entre otros. El que sigue es un ejemplo genérico de documento para un operación de update:

```
{
    "_id": {"_data": "82647400860000001F2B022C0100296E5(...)"},
    "operationType": "update",
    "clusterTime": {"$timestamp": {"t": 1685323910, "i": 31}},
    "wallTime": "2023-05-29T01:31:50.707Z",
    "ns": {"db": "samples", "coll": "inventory"},
    "documentKey": {"_id": 10},
    "updateDescription": {
        "updatedFields": {"item": "beans"},
        "removedFields": [],
        "truncatedArrays": [],
    },
    "fullDocument": {"_id": 10, "item": "beans", "quantity": 50, "price": 2.99},
}
```

Listing 1: Ejemplo genérico de documento para una actualización

Es importante observar que el tipo de documento que genera "Change Streams" no es el documento que interesa propagar en sí mismo. Por tanto, cuando se necesite procesar cada documento es necesario obtenerlo. Para ello, se hicieron configuraciones adicionales en "Change Streams" de forma que por cada cambio que ocurre en las operaciones de inserción y actualización de documentos se incluya el campo "fullDocument". De esta manera, no hay necesidad de recuperarlo, acelerando el tiempo total de procesamiento. No ocurre lo mismo para la operación de eliminación de documentos, ya que a la estructura de documento de "Change Streams" no se le puede incluir el campo "fullDocument". En este caso simplemente con el valor del campo "documentKey._id" se propaga hacia Elasticsearch un comando de eliminación para ese "_id".

Esta configuración no viene por defecto en “Change Streams”, se configura cuando se inicia el servicio con el parámetro “full_document=“updateLookup”” en el comando “collection.watch”.[22]

La primer operación que se realiza dentro de este bucle principal “for change in change_stream:” de la función “look_for_changes()” es la obtención del documento “puro” que interesa propagar hacia Elasticsearch. Esto lo implementa la función “getRealDoc()” donde según al campo “operationType” se obtiene el campo “fullDocument” o el campo “documentKey._id” para las eliminaciones. Este documento “puro”se tranforma -a través de la función “streamDoc2Elastic()”- a la estructura que se define para almacenar en Elasticsearch. Esta estructura contendrá solo los campos que se van a utilizar para visualizaciones, además de completar los campos ciudad y país ya que no están siempre completos en la base de calidad de aire:

```
{
    "id": doc["_id"],
    "locationId": doc["locationId"],
    "location": doc["location"],
    "parameter": doc["parameter"],
    "value": doc["value"],
    "unit": doc["unit"],
    "dateutc": doc["date"]["utc"],
    "datelocal": doc["date"]["local"],
    "city": city,
    "country": country,
}
```

Listing 2: Estructura para almacenamiento

Al mismo tiempo que ocurren estas operaciones dentro del bucle principal de ejecución, se almacena también el documento tipo de “Change Streams” en la colección “material_particulado_log”, se obtiene el siguiente token por cada ciclo del bucle (resume_token = change[“_id”]) y, finalmente, se guarda el valor del token en el archivo de texto auxiliar.

Los documentos se van enviando uno a uno a Elasticsearch en la medida que van ocurriendo los cambios en MongoDB. Es importante señalar que en la función “streamDoc2Elastic()” se define como “id” clave de documento Elasticsearch, el ObjectId() que ya está presente en cada documento MongoDB (podría ser cualquier otro campo o combinación única de campos). De esta forma, cada vez que vengan “id” repetidos, la función se encargará de hacer el update del documento correspondiente. Es importante tener en cuenta este aspecto, ya que si se hacen inserciones en Elasticsearch sin especificar un “id”, la base va a generar uno arbitrario, lo que puede generar problemas en situaciones donde se envíe dos veces el mismo documento a Elasticsearch; esto es, en vez de el último actualizar al primero, se crearían dos documentos que en realidad son el mismo.

Finalmente en la medida que van quedando los datos en Elasticsearch, se implementan algunas visualizaciones en Kibana.

7.2. Ejecución

Para que toda esta maquinaria comience a operar, es necesario que se produzcan los cambios previamente configurados para la colección “material_particulado” de MongoDB. Como se mencionó en secciones anteriores, esta solución emula un problema donde existe una base de datos MongoDB que sistemáticamente recibe cambios y éstos deben ser propagados a Elasticsearch. Por temas de alcance y tiempos de desarrollo, en el presente trabajo no se implementó un proceso que cargue los datos de calidad de aire en forma continua hacia la colección de MongoDB. Lo que se hizo fue implementar el script “mongoLoader.py” el cual ejecuta a demanda y carga para un rango de fechas determinado los documentos de calidad de aire en MongoDB. Se entiende que este script es suficiente a los efectos de verificar el correcto funcionamiento de la solución ya que es posible elegir períodos de fechas que generen cambios en menor o en gran escala en MongoDB. Adicionalmente, se verá más adelante en la sección de “Resultados” más pruebas donde se eliminan y se insertan distintos tamaños de lotes de documentos.

[22] https://www.mongodb.com/docs/manual/changeStreams

Luego la ejecución de “mongoLoader.py”, van a ingresar nuevos registros a MongoDB. Al tener previamente ejecutando el script “changeStreams.py”, como se explicó en la sección anterior, todos los cambios serán propagados en tiempo casi real hacia Elasticsearch. Vale mencionar que el término “casi” refiere a las demoras que pueden ocurrir tanto sea por el hardware que se utiliza, tiempos de red y volumen de datos a propagar.

Para verificar la solución, también se realizaron conexiones directas a la base MongoDB y se procedió a actualizar o eliminar registros para confirmar el correcto funcionamiento. La siguiente imagen ilustra el esquema de ejecución:

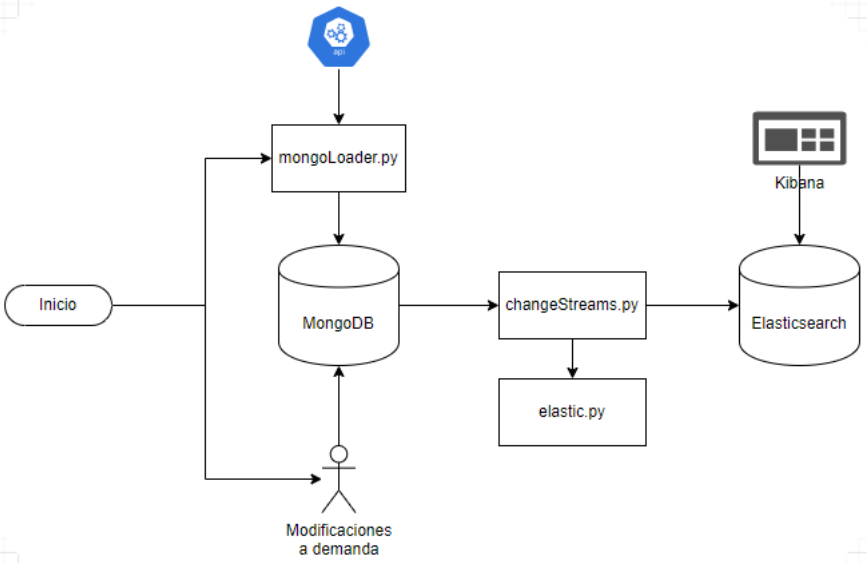

Figura 3: Esquema de ejecución

8. Pruebas, resultados y desempeño

En esta sección se presentan los resultados obtenidos en base a la ejecución de la solución implementada. Se analiza el rendimiento considerando un escenario en la nube y otro local (MongoDB, Elasticsearch y scripts en un mismo equipo Windows), evaluando la latencia y la consistencia de los datos replicados.

8.1. Descripción de las pruebas

Para ambos escenarios se armaron archivos json con lotes de documentos (extraídos de la propia base MongoDB) de diferentes cantidades de documentos con el objetivo de simular ráfagas abruptas de cambios sobre MongoDB, las cuales van de menos a más en volumen. Con estas ráfagas se pretende estudiar los tiempos de respuesta de la solución en su conjunto, además de verificar que en ningún caso se pierdan datos.

Los lotes de documentos considerados son de: 1, 20, 50, 100, 200, 500, 1000, 2000 y 3000, los cuales son estudiados para dos operaciones seleccionadas, eliminación e inserción sobre MongoDB. Como se menciona en la sección de Implementación, debido a que “Change Streams” desde que se inicia queda permanentemente esperando por cambios en el bucle “for change in change_stream:”, no resulta adecuado utilizar la implementación tal cual está en “changeStream.py” ya que dificulta la inicialización y toma de tiempos entre ráfagas. Por tanto, se clonó el código en “changeStreamTakeGustTime.py” donde se hicieron modificaciones para que quede esperando por una ráfaga, tome el tiempo, lo muestre y salga. El parámetro “TEST_FILE_DOCS_SIZE” de la configuración define el tamaño del archivo a utilizar -dentro de los tamaños predefinidos- y los scripts “mongoTestDelete.py” y “mongoTestInsert.py” ejecutan las eliminaciones e inserciones de prueba, respectivamente.

El lanzamiento de cada prueba se basa en un par de archivos de lotes .bat "testDeleteRun.bat" y "testInsertRun.bat" en donde primero se levanta el servicio de "Change Streams", se esperan unos pocos segundos para dar tiempo a que se arme el mecanismo de captura, e inmediatamente se ejecuta la actualización que corresponda sobre MongoDB. También existe un par de archivos .bat idénticos para las pruebas en clústers locales, con la única diferencia que por tratarse de ambientes locales (más rápidos) el tiempo de espera por el mecanismo de captura es menor.

La secuencia de lanzamiento de cada una de las pruebas primero invoca al "testDeleteRun.batz procede a borrar n (parámetro "TEST_FILE_DOCS_SIZE") documentos de MongoDB. Una vez finalizada la prueba, se ejecuta "testInsertRun.bat" y se insertan nuevamente los mismos n documentos. Luego de cada ciclo de eliminación/inserción la cantidad de registros en MongoDB y Elasticsearch debe ser la misma que antes de lanzar la prueba. De esta forma se asegura la no pérdida de datos.

En primera instancia y como se menciona en la sección de Implementación, se utilizó un despliegue de las bases de datos en las nubes de MongoDB y Red Hat para el clúster Elasticsearch. Como son dos proveedores distintos de infraestructura en la nube, con planes gratis, es esperable que los tiempos de red entre la transmisión desde MongoDB hacia Elasticsearch sean elevados. Por tanto, para tener otra óptica de las pruebas y en particular de la latencia, se realizaron las mismas pruebas en clústers MongoDB (*replica set*) y Elasticsearch locales en un mismo equipo Windows.

8.2. Resultados y desempeño

A continuación se presentan los resultados obtenidos en dos partes: remoto y local.

8.2.1. Remoto

En ninguna de las pruebas se detectaron pérdidas de datos pero, como era esperable, la latencia resultó alta. Tanto para las operaciones de eliminación como de inserción, al comienzo con lotes pequeños de documentos se obtienen tiempos aceptables que van desde 2 segundos cuando se tiene un documento, hasta 22 minutos con una ráfaga de 3000 documentos.

Seguidamente se presenta la tabla completa de resultados.

Ráfaga de documentos	Tiempo de eliminación (s)	Tiempo de inserción (s)
1	2.19	2.11
20	10.66	14.74
50	27.54	27.08
100	49.71	50.64
200	93.06	95.12
500	227.54	228.94
1000	446.49	451.28
2000	892.29	897.7
3000	1347.91	1341.92

Cuadro 1: Rendimiento de pruebas - Remoto

A continuación se expone el gráfico asociado al cuadro anterior.

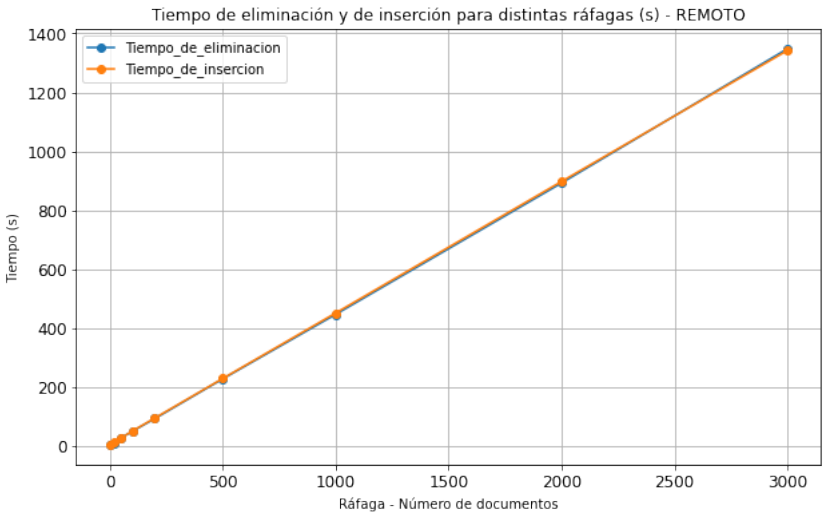

Figura 4: Tiempo de eliminación y de inserción para distintas ráfagas (s) - REMOTO

Como puede observarse, no existe prácticamente diferencia en el tiempo de eliminación y de inserción para las distintas ráfagas testeadas.

8.2.2. Local

En este caso tampoco en ninguna de las pruebas se detectaron pérdidas de datos y notoriamente se apreciaron mejoras en la latencia. Las operaciones de eliminación e inserción al comienzo registraron tiempos similares a los del ambiente remoto, pero al ir aumentando la cantidad de documentos de las ráfagas se comienza a detectar claras mejoras en los tiempos de respuesta.

A continuación se presenta la tabla completa de resultados.

Ráfaga de documentos	Tiempo de eliminación (s)	Tiempo de inserción (s)
1	2.27	2.66
20	3.51	3.84
50	5.26	5.42
100	7.65	9.59
200	16.25	16.68
500	31.09	34.01
1000	59.42	56.63
2000	121.64	126.3
3000	197.73	179.6

Cuadro 2: Rendimiento de pruebas - Local

A continuación se expone el gráfico asociado a la tabla anterior.

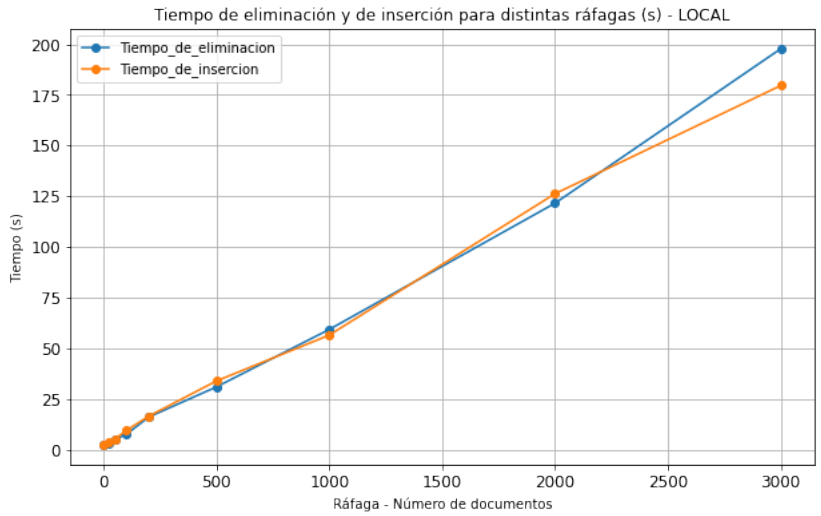

Figura 5: Tiempo de eliminación y de inserción para distintas ráfagas (s) - LOCAL

8.2.3. Comparación de desempeño: Local vs Remoto

Como parte final de esta sección, se exponen gráficos para comparar (para un mismo tipo de operación) el desempeño en entorno Local versus entorno Remoto, a los efectos de visualizar claramente las eficiencias obtenidas.

En el siguiente gráfico se expone la mencionada comparación para la operación de eliminación.

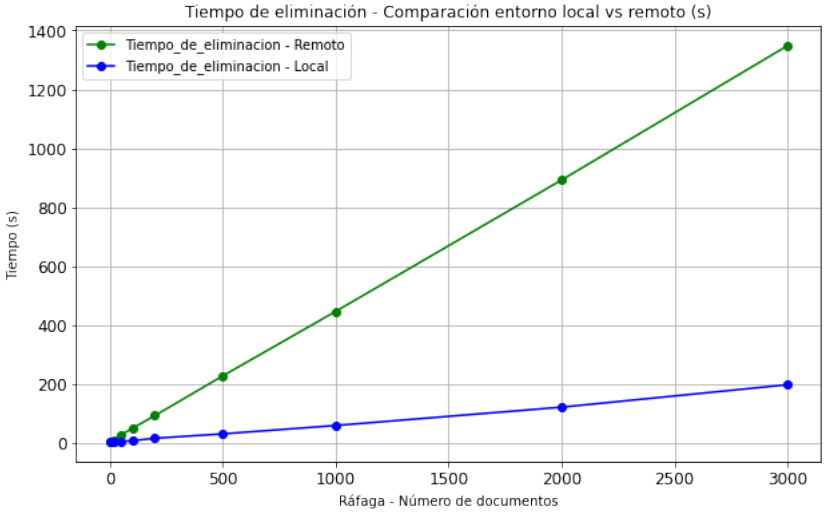

Figura 6: Tiempo de eliminación - Comparación entorno local vs remoto (s)

A continuación se expone el gráfico de comparación para la operación de inserción.

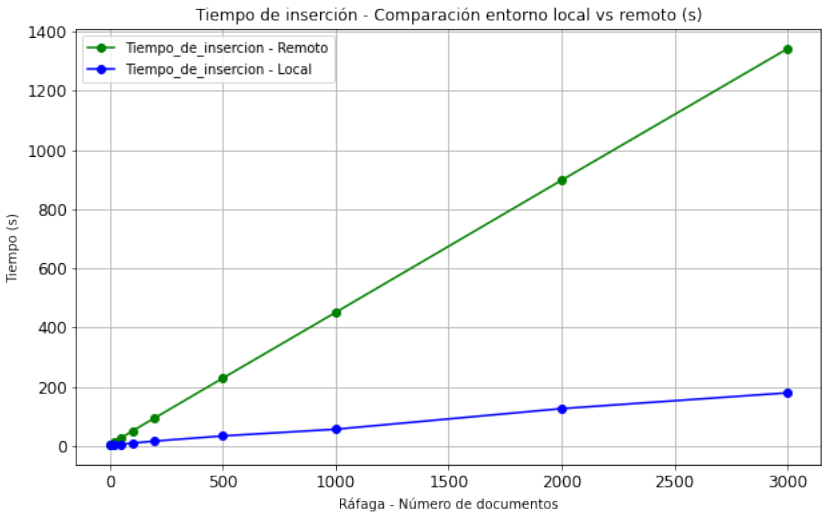

Figura 7: Tiempo de inserción - Comparación entorno local vs remoto (s)

Puede verse finalmente que, tanto para la operación de eliminación como para la de inserción, es claro -y tal como es esperable- el mejor desempeño en términos de tiempo de ejecución cuando se trabaja en entorno local.

8.3. Visualizaciones generadas en Kibana

Aprovechando que Kibana es una poderosa herramienta de visualización que ayuda a explorar, analizar y comunicar datos de manera efectiva, se procedió a crear un panel de control con distintas visualizaciones en lo referente a los datos colectados para el parámetro material particulado (en sus formas PM10 y PM25) para las distintas ciudades ya mencionadas en una sección anterior.

En tal sentido, se creó un panel que incluye (para los parámetros analizados PM10 y PM25 y para todas las ciudades consideradas) su evolución temporal para el período cargado, un gráfico de áreas que muestra -en base al promedio de cada serie temporal- en qué ciudades o países se encuentran los mayores niveles de contaminación y, finalmente, un gráfico que muestra valores de excedencia en base a un mapa de calor, considerado útil a los efectos de visualizar rápidamente en qué días fue superado un cierto umbral de concentración.

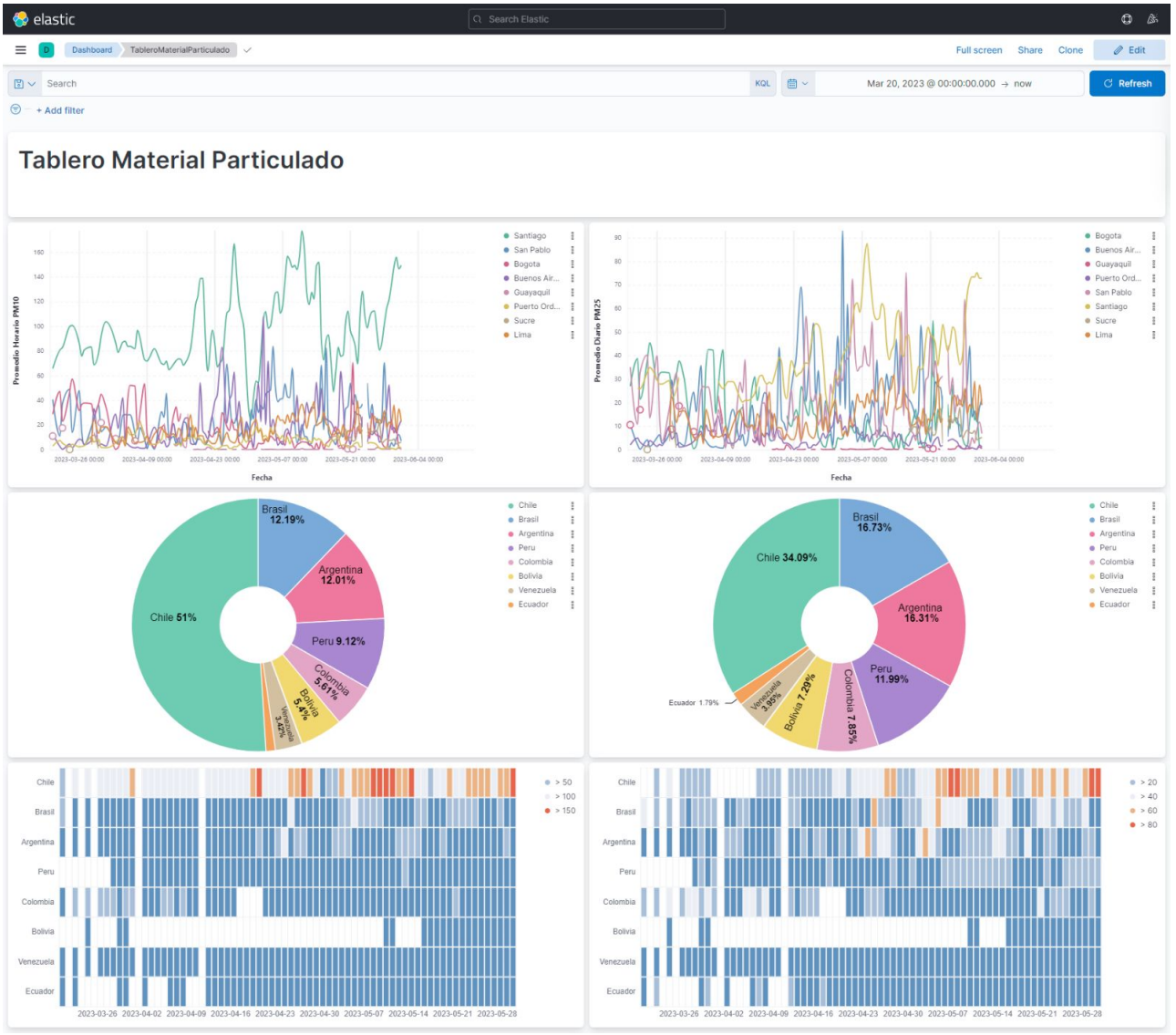

Figura 8: Panel de visualización - Kibana

A modo de referencia, puede observarse rápidamente cómo efectivamente la ciudad de Santiago resulta ser la más contaminada de América Latina en lo que refiere al material particulado, asunto ya ampliamente estudiado y conocido en lo que refiere a la ciencia de la contaminación atmosférica.[23]

9. Conclusiones y trabajo futuro

Luego de concluida la exploración de diversas opciones, implementación y pruebas de la solución elegida, se presentan a continuación las conclusiones del estudio divididas en dos partes: conclusiones sobre la solución implementada y, finalmente, variantes sobre la misma que podrían determinar futuros trabajos de investigación.

[23] https://www.futuro360.com/desafiotierra/santiago-de-chile-ranking-contaminacion_20220615/

9.1. Conclusiones sobre la solución

En cualquier circunstancia que se presente un problema de sincronización de datos es necesario relevar claramente las necesidades de los usuarios finales. Ello significa, por ejemplo, tener en claro si es aceptable que los datos lleguen a destino con una tolerancia de tiempo determinada (minutos, horas, días, etc). En el presente trabajo, se prioriza la sincronización en tiempo real o casi tiempo real desde el momento que ocurre una actualización en MongoDB hasta que llega a Elasticsearch, sin admitir pérdida alguna de datos. Asimismo, también deben considerarse cuestiones relativas a características de los datos y posibles entornos de trabajo. Ejemplos de estos son: el volumen de los datos a sincronizar, formatos de documentos, entornos, ancho de banda/estabilidad de red y hardware a utilizar. En atención a estas cuestiones es que se realizaron despliegues de la misma solución en ambientes en la nube y locales, además de realizar pruebas con ráfagas abruptas de actualizaciones sobre MongoDB variando los volúmenes de carga.

Observando los resultados obtenidos en las pruebas, se considera que la solución principal aquí expuesta es indudablemente aplicable cuando todos los componentes de la misma están desplegados dentro de una red local. Los tiempos obtenidos en las pruebas en entorno local para las operaciones de eliminación e inserción, en el peor de los casos llegó al orden de los 3 minutos para ráfagas de 3000 documentos y, por otra parte, con ráfagas de cientos de documentos se obtienen tiempos del orden de segundos.

En cuanto a los resultados observados en el ambiente remoto (un ambiente con dos proveedores de nube distintos MongoDB - Red Hat), se observa que los tiempos aumentan considerablemente a medida que se aumentan los volúmenes de las ráfagas de documentos, llegando a tener un tiempo al menos 7 veces superior que la misma operación realizada en un ambiente de red local. Sin duda alguna que estos tiempos en ambientes remotos mejorarían si todos los componentes estuvieran desplegados en el mismo proveedor, pero por razones de disponibilidad eso no fue posible lograr en el presente estudio. Aún así, se considera que la solución desplegada en la nube puede resultar aplicable en muchos casos, en particular donde exista mayor tolerancia con la latencia.

9.2. Trabajo futuro

Son múltiples las variantes que pueden lograrse en base a la solución aquí planteada, así como también considerando otras tecnologías tales como las mencionadas en la sección de “Análisis de Opciones Disponibles”.

9.2.1. Variantes Respecto a Solución Planteada

La primer variante que en algunos casos podría resultar en una mejora, es no procesar uno a uno los documentos al momento de realizar la propagación desde MongoDB hacia Elasticsearch. En la solución aquí planteada, en el mismo instante que ocurre una actualización de un documento en MongoDB, lo propaga hacia Elasticsearch (uno a uno). Se podría optar otro enfoque donde la propagación hacia Elasticseach se haga de una forma desacoplada del momento en que ocurren actualizaciones en MongoDB. La idea sería contar con un proceso asincrónico donde cada determinado período de tiempo lea la colección de log (material_particualdo_log) donde se almacenan todos los cambios con sus documentos asociados. Esta lectura asincrónica se podría hacer por lotes de documentos, lo que podría determinar una mejora en los tiempos de procesamiento. La desventaja de esta variante es que el proceso para garantizar la no pérdida de datos frente a fallos, debería de alguna manera llevar la cuenta de los documentos actualizados en Elasticsearch de forma que, en caso de ocurrir una falla, al retomar la sincronización lo haga a partir de la última actualización recibida del lado de Elasticsearch.

Otra línea de investigación a futuro refiere al estudio de la concurrencia. Por razones de tiempo y alcance, el tema no fue abordado en el presente trabajo. Podría estudiarse el comportamiento de esta solución (con o sin variantes) cuando desde distintos orígenes se actualizan al mismo tiempo ráfagas de documentos de diversos tamaños en MongoDB. Es esperable que los tiempo de procesamiento bajen a medida que se aumentan las fuentes de actualización y volúmenes sobre MongoDB, pero aquí no fue posible cuantificar tiempos ni verificar confiablidad de la solución.

9.2.2. Otras soluciones

Si bien en la sección de "Análisis de Opciones Disponibles" se vieron algunas alternativas a la solución finalmente implementada, en esta sección se plantea otra solución que es particularmente interesante cuando los volúmenes de carga tienden a superar los vistos en el presente documento.

Esta nueva solución requiere la integración de un nuevo tipo de componente conocido como cola de mensajes. En particular, Kafka es una de las opciones más utilizadas en el área. La idea sería utilizar un conector MongoDB-Kafka que sincronice los datos en ambas direcciones. Este conector sería capaz de capturar los cambios tanto a nivel de colección como de la base de datos. La utilización de Kafka aportaría una solución flexible, confiable y escalable capaz de manejar grandes volúmenes de datos sin pérdida de desempeño. Está fuera del alcance de este trabajo detallar los aspectos de instalación y configuración de Kafka, pero sin duda alguna serían temas adicionales de los cuales ocuparse en caso de adoptar este esquema.[24][25]

[24] `https://rockset.com/blog/3-ways-to-offload-read-heavy-applications-from-mongodb`
[25] `https://debezium.io/blog/2018/01/17/streaming-to-elasticsearch`

Repositorio web de este trabajo

https://gitlab.com/fing4/bdnr_final

Referencias

[1] Apache Kafka (s.f.), https://kafka.apache.org/

[2] Cloud database solutions - IBM (s.f.), http://compose.io/

[3] Elastic - Jdbc input plugin (s.f.), https://www.elastic.co/guide/en/logstash/current/plugins-inputs-jdbc.html

[4] [Github1] Compose Transporter helps with database transformations from one store to another. It can also sync from one to another or several stores (2022, 26 de mayo), https://github.com/compose/transporter

[5] [Github2] Mongo-Connector doc manager for Elastic 1.x (2018, 14 de setiembre), https://github.com/yougov/elastic-doc-manager

[6] [Github3] MongoDB River Plugin for ElasticSearch (2016, 22 de marzo), https://github.com/richardwilly98/elasticsearch-river-mongodb

[7] [Github4] Mongoosastic is a mongoose plugin that can automatically index your models into elasticsearch (2022, 19 de setiembre), https://github.com/mongoosastic/mongoosastic/

[8] [Github5] Welcome to the Mongo Connector wiki! Read any of the sections below to find detailed information about the connector (2017, 23 de junio), https://github.com/yougov/mongo-connector/wiki

[9] IBM Acquires Compose to Expand Cloud Data Services (2015, 23 de julio), https://www.prnewswire.com/news-releases/ibm-acquires-compose-to-expand-cloud-data-services-300117883.html

[10] [MongoDB1] Change Streams (s.f.), https://www.mongodb.com/docs/manual/changeStreams/

[11] [MongoDB2] Configure Database Triggers (s.f.), https://www.mongodb.com/docs/atlas/triggers/

[12] [MongoDB3] Database. Deploy a multi-cloud database (s.f.), https://www.mongodb.com/atlas/database

[13] [MongoDB4] Elasticsearch vs MongoDB Atlas Search (s.f.), https://www.mongodb.com/compare/mongodb-atlas-search-vs-elastic-elasticsearch

[14] Mongoose, elegant mongodb object modeling for node.js (s.f.), https://mongoosejs.com/

[15] Openaq.org - Fighting air inequality through open data (s.f.), https://openaq.org

[16] Red Hat OpenShift Playground (s.f.), https://developers.redhat.com/courses/explore-openshift/openshift-playground

[17] Santiago de Chile se posicionó como una de las 10 ciudades más contaminadas del mundo, según ranking IQAir (2022, 15 de junio), https://www.futuro360.com/desafiotierra/santiago-de-chile-ranking-contaminacion_20220615/

[18] Streaming Data Changes from Your Database to Elasticsearch (2018, 17 de enero), https://debezium.io/blog/2018/01/17/streaming-to-elasticsearch/

[19] Wark and Warner, C.F. (1999), Contaminación del aire, su origen y control. Editorial Limusa S.A. – Grupo Noriega Editores. Balderas 95, México D.F.

[20] 3 Ways to Offload Read-Heavy Applications from MongoDB (2020, 25 de setiembre), https://rockset.com/blog/3-ways-to-offload-read-heavy-applications-from-mongodb

[21] 5 Different ways to synchronize data from MongoDB to ElasticSearch (2016, 13 de diciembre), https://code.likeagirl.io/5-different-ways-to-synchronize-data-from-mongodb-to-elasticsearch-d8456b83d44f

[22] 5 ways to synchronize data from MongoDb to ElasticSearch (2016, 29 de marzo), https://www.linkedin.com/pulse/5-way-sync-data-from-mongodb-es-kai-hao/