A blue and black logo

Description automatically generated

**FACULTAD**

**DE**

**CIENCIAS**

**GESTIÓN CENTRALIZADA DE LOGS PARA SERVICIOS BIG DATA DESPLEGADOS SOBRE UNA ARQUITECTURA CENTRADA EN EL DATO**

CENTRALIZED LOG MANAGEMENT FOR BIG DATA SERVICES DEPLOYED ON A DATA-CENTRIC ARCHITECTURE.

Trabajo de Fin de Grado de Ingeniería Informática

Autor: Jesús Saiz Gutiérrez

Director: Marta Elena Zorrilla Pantaleón

Julio 2024

**Agradecimientos**

**Resumen**

En el ámbito empresarial actual, se utilizan numerosas aplicaciones y servicios que necesitan funcionar de manera eficaz. Cada uno de estos componentes produce datos propios sobre su rendimiento. Sin embargo, la gestión de esta información a menudo no es óptima debido a la gran cantidad de servicios y datos generados.

Este proyecto busca desarrollar un sistema de gestión de registros sobre una plataforma big data distribuida y escalable que incluye el uso de Kafka como gestor de eventos, entre otras tecnologías.

Más específicamente, el objetivo es implementar un sistema de gestión de registros utilizando el conjunto de herramientas ELK (Logstash, Elasticsearch y Kibana) para recopilar, procesar y visualizar el estado de los diversos servicios de la plataforma big data.

**Palabras clave**

Logs, Elasticsearch, Logstash, Kibana, Big Data, Kafka

**Abstract**

In today's business environments, a vast array of applications and services is utilized, each of which must operate correctly. Each of these components generates its own information about its status. However, due to the high number of services and the volume of information produced, this information is often not adequately monitored.

This project aims to create a log management system on a scalable and distributed big data platform utilizing the Kafka event manager among other technologies.

More specifically, it seeks to implement a log management system using the ELK stack (Logstash, Elasticsearch, and Kibana) to collect, process, and visually display the status of various services within the big data platform.

**Keywords**

Logs, Elasticsearch, Logstash, Kibana, Big Data, Kafka

# Índice

[Índice 7](#_Toc164091872)

[1. Introducción 10](#_Toc164091873)

[1.1. Objetivos 10](#_Toc164091874)

[1.2. Tecnologías sobre las que se realiza la gestión y análisis de logs 11](#_Toc164091875)

[1.3. Elección de tecnologías para la gestión y análisis de logs 11](#_Toc164091876)

[2. Stack ELK 12](#_Toc164091877)

[2.1. Funcionamiento de Logstash 13](#_Toc164091878)

[2.2. Funcionamiento de Elasticsearch 14](#_Toc164091879)

[2.3. Funcionamiento de Kibana 15](#_Toc164091880)

[2.4. Filebeat 15](#_Toc164091881)

[3. Logs de los servicios 16](#_Toc164091882)

[3.1. Logs de Kafka 17](#_Toc164091883)

[3.1.1. Ficheros de logs de Kafka 17](#_Toc164091884)

[3.1.2. Autenticación vs Autorización en Kafka 18](#_Toc164091885)

[3.1.3. Configuración de la autenticación/autorización de acceso 18](#_Toc164091886)

[4. Despliegue del proyecto 20](#_Toc164091887)

[4.1. Volúmenes Docker 21](#_Toc164091888)

[4.2. Configuración del fichero .yaml 21](#_Toc164091889)

[5. Configuración del Stack ELK 26](#_Toc164091890)

[5.1. Configuración de Logstash 26](#_Toc164091891)

[5.1.1. Ficheros de configuración 26](#_Toc164091892)

[5.1.2. Filtrado con el plugin Grok 27](#_Toc164091893)

[5.2. Configuración de Elasticsearch 32](#_Toc164091894)

[5.3. Configuración de Filebeat 34](#_Toc164091895)

[6. Visualización en Kibana 36](#_Toc164091896)

[6.1. Observabilidad 36](#_Toc164091897)

[7. Despliegue local 36](#_Toc164091898)

[7.1. Limitaciones del despliegue local 36](#_Toc164091899)

[7.2. Explicación del problema de almacenamiento 36](#_Toc164091900)

[7.3. Explicación de la solución local 36](#_Toc164091901)

[8. Explicación de la solución en la nube 37](#_Toc164091902)

[9. Objetivos de desarrollo futuro 37](#_Toc164091903)

[10. Bibliografía 38](#_Toc164091904)

**Índice de tablas**

[Tabla 1: Localización de los logs de los servicios 18](#_Toc164089967)

**Índice de imágenes**

[Imagen 1: Esquema productor-consumidor de Kafka. 12](#_Toc164090521)

[Imagen 2: Arquitectura general del sistema (sin Filebeat). 14](#_Toc164090522)

[Imagen 3: Formato de logs proporcionado por Logstash por defecto. 14](#_Toc164090523)

[Imagen 4: Arquitectura de la solución final usando Filebeat. 17](#_Toc164090524)

[Imagen 5: Formato de logging de 5 ficheros de Kafka 31](#_Toc164090525)

[Imagen 6: Formato de logging de 1 fichero de Kafka 31](#_Toc164090526)

[Imagen 7: Visualización de campos en Kibana para los 5 ficheros 32](#_Toc164090527)

[Imagen 8: Visualización de campos en Kibana para el fichero independiente 32](#_Toc164090528)

[Imagen 9: Índices por defecto de Elasticsearch 34](#_Toc164090529)

# Introducción

El presente Trabajo de Fin de Grado se ha desarrollado en el contexto del proyecto de investigación: Monitorización y análisis de datos en streaming en arquitecturas distribuidas y escalables, del grupo ISTR a través de una beca de colaboración con el Departamento de Ingeniería Informática de la Universidad de Cantabria. El objetivo era desarrollar un servicio de monitorización y análisis de los datos disponibles en una plataforma big data, bajo una arquitectura Kappa, esto es distribuida, escalable, en tiempo real y acorde a la filosofía del metamodelo RAI4.0 desarrollado por el grupo.

Este metamodelo, tiene por objeto proponer una solución a la implementación de aplicaciones intensas en el uso de datos en entornos distribuidos donde los sistemas Internet of Things (IoT), la computación en la nube y el procesado distribuido están presentes para abordar el volumen, velocidad y variedad de datos que se generan.

El proyecto realizado pretende, por lo tanto, exponer de forma práctica un caso de uso proveniente del IoT industrial, trabajando sobre una plataforma big data distribuida y escalable, utilizando el gestor de eventos Kafka, así como otras tecnologías específicas para la visualización gráfica de los diversos datos procesados y la monitorización de los servicios que dan soporte a la solución.

TODO: Modificar introducción (es la misma que Mario)

## Objetivos

Este trabajo de fin de grado se centra en crear un sistema para gestionar, monitorizar y analizar registros utilizando las tecnologías Logstash, Elasticsearch y Kibana, agrupadas bajo el nombre de stack ELK.

El proyecto se estructura en varios objetivos específicos:

* **Implementar y configurar una plataforma de servicios**: Antes de gestionar los registros de ciertos servicios, es necesario implementar y configurar el conjunto de servicios que se utilizarán para la gestión de registros mediante el stack ELK.
* **Establecer un pipeline para el procesamiento de registros**: El objetivo es desarrollar un flujo de datos con Filebeat, Logstash, Elasticsearch y Kibana para facilitar la captura, procesamiento y visualización de los registros.
* **Desarrollar estrategias para el filtrado y procesamiento de registros**: Es esencial diseñar métodos para estructurar los registros filtrándolos y procesándolos adecuadamente, de modo que se facilite su consulta.
* **Crear cuadros de control o dashboards**: Se empleará Kibana para crear interfaces visuales que simplifiquen la visualización y el análisis de los registros de los servicios utilizados en la plataforma de big data.
* **Despliegue en la nube**: El proyecto se expandirá para incluir el despliegue del sistema de gestión de registros en un entorno en la nube, lo cual mejorará el almacenamiento y aprovechará las ventajas de escalabilidad, disponibilidad y flexibilidad que los entornos en la nube ofrecen para manejar grandes volúmenes de datos y registros en tiempo real.

## Tecnologías sobre las que se realiza la gestión y análisis de logs

Para llevar a cabo el propósito principal de gestionar los logs de un conjunto de servicios ha sido necesario el despliegue de una arquitectura con los siguientes elementos:

* **Apache Kafka:** Apache Kafka es una conocida plataforma para el manejo de eventos en tiempo real que facilita la recolección, el procesamiento y el almacenamiento de flujos de datos continuos. Utiliza un sistema de intercambio de datos que opera bajo el modelo de productor y consumidor. En este proyecto va a ser el servicio principal para la gestión de logs, además de que se usarán sus funciones de autenticación y autorización para la visualización de logs en Kibana.

En la imagen 1 se muestra un esquema con la representación gráfica del modelo productor-consumidor:

A diagram of a circle with three lines

Description automatically generated with medium confidence

Imagen 1: Esquema productor-consumidor de Kafka.

* **Zookeeper:** Facilita la coordinación entre procesos distribuidos mediante un espacio de nombres jerárquico común de registros de datos, conocidos como znodes. Es un servicio útil para el funcionamiento de los demás, ya que almacena información como los ACL’s definidos en Kafka.
* **Apache Spark:** Apache Spark es una plataforma integrada de análisis que maneja datos a gran escala e incluye módulos para SQL, transmisión en tiempo real, aprendizaje automático y análisis de grafos. En este proyecto, se emplea una aplicación Java para captar datos de Kafka, procesarlos y almacenarlos en Cassandra.
* **Cassandra:** Sistema de gestión de bases de datos NoSQL, en este proyecto se utiliza para almacenar los datos procesados recibidos desde Spark.

## Elección de tecnologías para la gestión y análisis de logs

Existe una gran variedad de tecnologías que permiten crear un sistema de gestión de logs similar, por ejemplo:

* Splunk
* Graylog
* Fluentd
* Prometheus y Grafana
* Loki
* Datadog
* Sumo Logic
* Stack ELK

Para el desarrollo del proyecto era necesario que la tecnología fuese open source y gratis, por lo que se descartan opciones como Splunk. Además, sería recomendable que la tecnología empleada tuviese una gran comunidad de usuarios para evitar tener que cambiar todo el sistema en un futuro en caso de que se quede obsoleta la tecnología empleada. También sería recomendable que su uso esté extendido para la función que vamos a desarrollar en el proyecto, es decir, gestión de logs. Teniendo en cuenta todos estos parámetros se ha decidido emplear el stack ELK para este proyecto.

# Stack ELK

Para este proyecto, se ha optado por utilizar el stack ELK de código abierto, que incluye tres tecnologías principales: Elasticsearch, Logstash y Kibana, destinadas a la gestión de registros (logs).

* **Logstash**: Funciona como un pipeline para el procesamiento de datos. En este proyecto se utilizará para captar los registros provenientes de Filebeat, filtrarlos y/o enriquecerlos y enviarlos a su índice correspondiente en Elasticsearch.
* **Elasticsearch**: Sirve como una base de datos NoSQL, cuya tarea es almacenar los registros enviados por Logstash. Además, ofrece un motor de búsqueda de texto completo y capacidades de análisis distribuido, posteriormente se explicará mas detalladamente, ya que es el elemento más complejo del stack ELK.
* **Kibana**: Es el componente final del stack ELK, y se utiliza para visualizar los datos almacenados en Elasticsearch de forma visual, accediendo a Elasticsearch a través de su API de tipo REST.

Estas son las 3 aplicaciones principales utilizadas en el proyecto, sin embargo, hay otras aplicaciones secundarias, desarrolladas por los mismos equipos que pueden ser útiles o incluso necesarias para el desarrollo del proyecto. Por ejemplo, también se ha utilizado Filebeat, que se considera parte del Elastic Stack, aunque no sea un elemento principal del stack ELK.

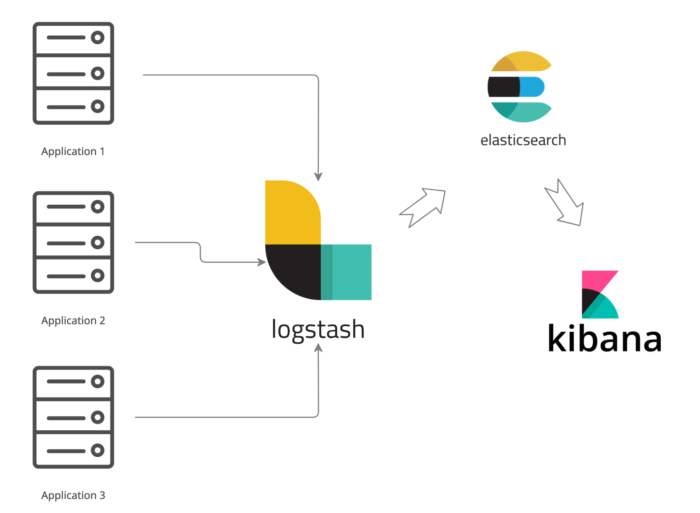
En la imagen 2 se muestra un esquema con la representación gráfica general de las tecnologías utilizadas para el análisis de logs.

Imagen 2: Arquitectura general del sistema (sin Filebeat).

## Funcionamiento de Logstash

Como mencioné anteriormente, Logstash opera como un pipeline para el procesamiento de datos, captando información (logs), procesándola y finalmente enviándola a un repositorio, Elasticsearch en este contexto. La función principal de Logstash es transformar datos inicialmente sin estructura, como los logs crudos obtenidos de varios servicios, organizándolos al añadir detalles como tipo, fecha, hora, host y mensaje, y convertirlos en formato JSON para su almacenamiento en Elasticsearch.

Logstash capta datos, en este contexto logs, y facilita su procesamiento y transformación antes de su envío a Elasticsearch. Por ejemplo, permite la aplicación de filtros para extraer campos específicos de los logs. Después de procesar los datos, se configura Logstash para dirigir los logs ya procesados hacia Elasticsearch.

En la imagen que se mostrará a continuación, se puede ver el formato estándar que Logstash aplica a los logs que recibe.A group of text boxes

Description automatically generated

Imagen 3: Formato de logs proporcionado por Logstash por defecto.

Como se ve en la imagen anterior, Logstash por defecto, sin añadir ningún filtro, añade los siguientes campos a los logs para darles una estructura consultable:

* + **type:** Indica el servicio del que provenga el log, a no ser que se esté usando Filebeat, en ese caso, tomará el valor “Filebeat” independientemente del servicio que haya generado el log.
  + **@version:** Este campo es irrelevante, hace referencia a la versión del esquema del log en Logstash, normalmente tomará el valor 1.
  + **path:** Es la ruta hasta el fichero de donde se leyó el log. Si se usa Logstash o Filebeat, será el fichero de log que se ha mapeado previamente al contenedor de Logstash/Filebeat.
  + **host:** En este proyecto, este campo identifica el contenedor donde se generó el log, en otros casos podria contener el nombre del host de donde proviene el log.
  + **@timestamp:** Es el valor temporal de referencia que utiliza Logstash, indica la fecha y hora del momento en que Logstash recibió el registro.
  + **message:** Indica el mensaje principal del log, suele contener gran parte del log.
  + **\_id:** Este campo sirve para identificar cada log (documento json) dentro de cada índice de Elasticsearch, tiene un valor único obviamente.
  + **\_type:** Este campo ya no se utiliza, está depreciado.
  + **\_index:** Como es lógico, este es el nombre del índice de Elasticsearch donde se ha almacenado este log.
  + **\_score:** Este campo es irrelevante, indica como de importante es un documento en referencia a una consulta de búsqueda determinada. En este proyecto no se ha utilizado para nada.

Cabe destacar que, estos campos se generaran por defecto por Logstash en caso de que el input recibido por Logstash sea un log en crudo, sin embargo, si se utiliza Filebeat, el log recibido en Logstash ya será un documento json y tendrá un formato de campos determinado por Filebeat.

## Funcionamiento de Elasticsearch

Para almacenar la información, Elasticsearch se basa en índices, un índice es un conjunto de documentos json y cada documento json es un conjunto de campos clave-valor, los cuales contienen los datos del log.

Cuando Elasticsearch recibe un registro, lo recibirá ya en el formato json que hayan creado Filebeat y Logstash, lo indexa junto con un tipo de dato timestamp que es añadido automáticamente por Elasticsearch para guardar el instante en que llegaron esos datos y facilitar futuras busquedas.

* RDBMS => Bases de datos => Tablas => Columnas/Filas
* Elasticsearch => Cluster => Nodos => Índices => Shards => Documentos con pares clave-valor

Una característica de Elasticsearch muy utilizada en este proyecto es que no es necesario indicar el tipo de dato que se va a enviar a Elasticsearch, es decir, puedes importar datos sin necesidad de haber definido el esquema anteriormente.

Con los datos ya almacenados, se puede empezar a usar la funcionalidad catalogada como la más potente de Elasticsearch, la búsqueda y análisis de datos. Elasticsearch proporciona una API basada en JSON RESTful para buscar, indexar, actualizar y eliminar documentos dentro de un índice mediante peticiones http. Una de las principales características de Elasticsearch es que través de esta interfaz se pueden realizar búsquedas por texto completo (full-text) casi en tiempo real.

Otra característica importante que mencionar es que, Elasticsearch, para permitir una gran escalabilidad, emplea shards en los índices, es decir, crea réplicas de los índices (no necesariamente completas, pueden ser partes del índice) para permitir consultas en paralelo y almacenamiento distribuido.

Como he mencionado previamente, en este proyecto se utiliza la funcionalidad de Elasticsearch de importar logs sin haber definido un esquema previamente, en referencia a ello, se puede configurar los índices de dos formas:

* + **Estáticamente:** Se consideran índices estáticos los índices sobre los que se ha definido previamente el tipo de datos que va a almacenar. Es útil en casos en los que se quieran restringir los campos de los documentos que formarán parte del índice.
  + **Dinámicamente:** Todos los índices sobre los que no se define un esquema previamente son dinámicos por defecto. En este caso Elasticsearch intenta detectar el tipo de dato recibido y darle el formato que cree determinado.

Respecto a la API de Elasticsearch, que es utilizada principalmente por Kibana, esta permite buscar, indexar, actualizar y eliminar datos almacenados en Elasticsearch mediante el lenguaje de consulta Query DSL basado en JSON.

## Funcionamiento de Kibana

Kibana es el punto final del flujo de datos, es una plataforma de código abierto que permite la visualización de datos mediante dashboards, gráficos y todo tipo de elementos visuales. Kibana es ampliamente usado para análisis de logs, análisis de seguridad y análisis de datos en diferentes sectores.

Kibana se suele usar integrado con Elasticsearch, para obtener los datos a mostrar utiliza la API basada en JSON RESTful disponible en Elasticsearch, a través de la cual se pueden hacer consultas de todo tipo, por palabras, campos, fechas, etc.

Kibana contiene múltiples funcionalidades, algunas de ellas de pago, en este proyecto se usarán principalmente funciones de observabilidad.

Más adelante se explicarán con un caso de uso las diferentes funcionalidades de Kibana.

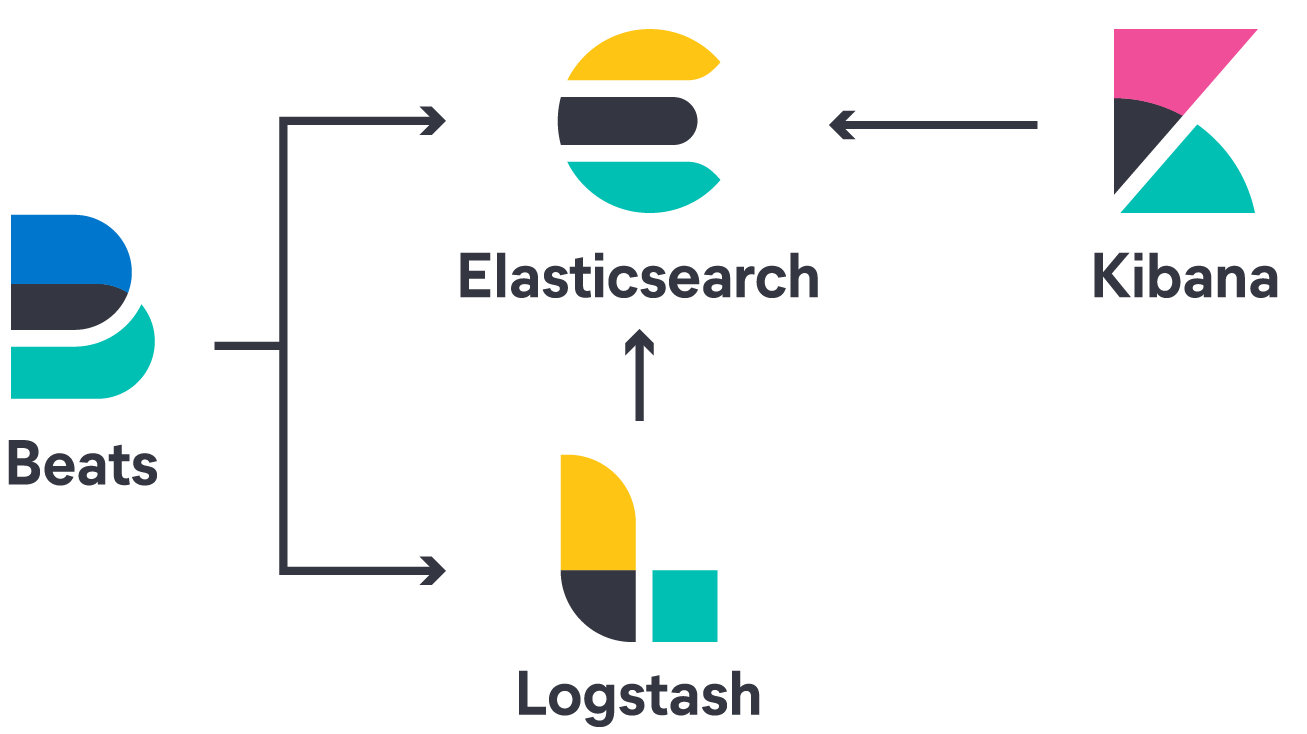
## Filebeat

Los beats son agentes ligeros que se pueden instalar en los servicios para enviar directamente los logs a Elasticsearch o Logstash.

Filebeat permite enviar los logs tanto a Logstash como a Elasticsearch, en caso de que los logs necesiten ser procesados o filtrados sería necesario pasar por Logstash, sin embargo, en caso contrario se podrían almacenar en Elasticsearch directamente.

Filebeat es especialmente útil cuando se generan logs en múltiples servidores, ya que instalando y configurando Filebeat en cada servidor sería suficiente para enviar los logs a Logstash o Elasticsearch, en el caso de un despliegue local en un mismo equipo se podria evitar el uso de Filebeat almacenando los logs en el equipo e indicando a Logstash su localización mediante bid mounts o volúmenes. Sin embargo, para darle un contexto más realista se empleará Filebeat en el proyecto.

En la imagen 4 se muestra como quedaría la arquitectura de la solución para la gestión de logs usando Filebeat:

Imagen 4*: Arquitectura de la solución final usando Filebeat.*

# Logs de los servicios

Para poder llevar a cabo el análisis de logs, primero será necesario localizar donde se escriben los logs de los servicios que queremos monitorizar. El lugar de almacenamiento de los logs por los diferentes servicios se puede modificar en sus ficheros de configuración, en este proyecto la ruta de almacenamiento de los logs viene determinada por las imágenes que se utilizan para desplegar los contenedores de los servicios.

En la siguiente tabla se puede visualizar la ruta de almacenamiento de logs de los diferentes servicios.

|  |  |
| --- | --- |
| SERVICIO | RUTA EN EL CONTENEDOR |
| Cassandra | /var/log/cassandra  /opt/cassandra/logs es un enlace simbólico a /var/log/cassandra |
| Zookeeper | /var/log/zookeeper |
| Kafka | /opt/kafka/logs  /kafka/kafka-logs-kafka |
| Spark Master | /opt/bitnami/spark/logs |
| Spark Worker | /opt/bitnami/spark/logs |

Tabla 1: Localización de los logs de los servicios

Una vez has localizado los logs del servicio que quieres gestionar, el siguiente paso es conocer que información se almacena en cada fichero de log, para poder elegir que ficheros te interesan y cual es el formato de logging en ese fichero.

## Logs de Kafka

En Kafka, los registros se clasifican generalmente en dos tipos: registros de bróker (o aplicación) y registros de datos (o tópicos). La configuración de estos registros se puede ajustar en el directorio “/opt/kafka/config”. Los archivos de configuración más cruciales incluyen “server.properties”, que especifica la ubicación de los registros de datos, y “log4j.properties”, que establece el nivel de registro y los archivos para los registros del bróker. A continuación, se explican los dos tipos de logs mencionados previamente.

* + **Logs de datos/tópicos:** En este proyecto se encuentran en la ruta /kafka/kafka-logs-kafka, estos ficheros están relacionados con el almacenamiento de los datos en los tópicos, contienen los propios datos y metadatos.
  + **Logs de aplicación/bróker:** En este proyecto se encuentran en la ruta /opt/kafka/logs, incluyen información sobre el funcionamiento del servidor de Kafka.

Para un sistema de gestión de logs, los logs más relevantes son los logs de aplicación, ya que proporcionan información directa sobre el estado de salud, seguridad y rendimiento del cluster de Kafka.

### Ficheros de logs de Kafka

**controller.log:** En este proyecto, como hay una sola instancia de Kafka, es decir, un solo bróker, el controlador es la misma instancia, y en este fichero se registran las actividades del controlador, como gestionar errores en otros brokers, gestionar el líder de las particiones, hacer el rebalanceo de cargas si cambia el numero de brokers del cluster, gestionar los cambios de estado de las particiones e incluso gestionar los metadatos del cluster.

**kafka-authorizer.log:** Este fichero será muy utilizado en este proyecto, inicialmente no se generan logs en este fichero, es necesario configurar la autorización y autenticación de Kafka para ello. Como se deduce, aquí se almacenan los errores de autorización.

**kafka-request.log:** En este fichero se almacena información sobre las solicitudes de los clientes, es útil para visualizar el comportamiento del cliente y posibles errores durante la comunicación cliente-servidor.

**log-cleaner.log:** Este fichero registra información sobre el proceso conocido como “compactación de logs”, el cual ayuda a mantener el tamaño de los logs manejable y optimiza el uso del almacenamiento eliminando entradas de log antiguas o duplicadas que ya no son necesarias.

**server.log:** Es el fichero más importante, ya que registra la actividad general del servidor. Entre otros logs, también registra los fallos de autenticación (si está activada la autenticacion).

**state-change.log:** Es un fichero poco importante, almacena detalles cada vez que hay un cambio en el estado de las particiones dentro de un broker de Kafka. Esto puede incluir eventos como cuando una partición se vuelve activa, se desactiva, o cuando cambia de líder.

**kafkaServer-gc.log:** También es un fichero poco importante, simplemente contiene información sobre la recolección de basura de Java.

### Autenticación vs Autorización en Kafka

De cara a monitorizar los logs de autenticación y autorización de Kafka, es importante conocer como Kafka maneja la autenticación y la autorización y diferenciar los dos conceptos:

* **Autenticación:** Los fallos de autenticación aparecen en el fichero “server.log”. La autenticación es correcta si el usuario con el que se quiere acceder a Kafka está definido en las credenciales válidas de Kafka, incorrecta en caso contrario.
* **Autorización**: La autorización se refiere al proceso de decidir si un usuario, ya previamente autenticado de forma correcta, tiene permiso para realizar una acción específica, como leer o escribir en un tópico. Los eventos relacionados específicamente con la autorización se registran en “kafka-authorizer.log”. Para que funcione la autorización es necesario haber definido ACL’s previamente. Aquellas acciones sobre las que no se defina un permiso con ACL’s serán denegadas por Kafka.

### Configuración de la autenticación/autorización de acceso

Como se ha mencionado previamente, se va a activar la autenticación y autorizacion de Kafka para utilizarlo como un caso de uso en las visualizaciones de Kibana.

Para activar la autenticación/autorización es necesario llevar a cabo modificaciones en el fichero de configuración “server.properties” de Kafka, en este caso al estar trabajando con contenedores la mejor opción es modificar el fichero a través de las variables de entorno de la imagen de Kafka en el fichero .yaml de docker que utilizamos para el despliegue.

Se han llevado a cabo los siguientes pasos para configurar la autenticación/autorización SASL\_PLAINTEXT de Kafka:

1. Modificaciones realizadas para configurar el mecanismo de autorización SASL\_PLAIN:

KAFKA\_ADVERTISED\_LISTENERS: SASL\_PLAINTEXT://localhost:9092

KAFKA\_LISTENERS: SASL\_PLAINTEXT://:9092

KAFKA\_INTER\_BROKER\_LISTENER\_NAME: SASL\_PLAINTEXT

KAFKA\_AUTHORIZER\_CLASS\_NAME:kafka.security.authorizer.AclAuthorizer

KAFKA\_SUPER\_USERS: User:admin;User:admin2

KAFKA\_SASL\_MECHANISM\_INTER\_BROKER\_PROTOCOL: PLAIN

KAFKA\_SASL\_ENABLED\_MECHANISMS: PLAIN

KAFKA\_OPTS: "-Djava.security.auth.login.config=/jaas.conf"

1. Crear un archivo JAAS para configurar las credenciales válidas de los usuarios. Este archivo definirá los usuarios y sus contraseñas.

KafkaServer {

   org.apache.kafka.common.security.plain.PlainLoginModule required

   username="admin"

   password="admin"

   user\_admin="admin"

   user\_usuario1="usuario1"

   user\_usuario2="usuario2";

};

Para utilizar SASL\_PLAIN desde el cliente será necesario crear un fichero “.properties” donde se indiquen las credenciales que está usando el cliente, y este fichero se referenciará en los comandos de producción, consumición o configuración de Kafka.

Un ejemplo de cliente valido para la configuración anterior sería el siguiente:

security.protocol=SASL\_PLAINTEXT

sasl.mechanism=PLAIN

sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="admin" password="admin";

Aquí un ejemplo de creación de un tópico empleando la autenticación:

docker exec kafka /opt/kafka/bin/kafka-topics.sh --create --topic topico1 --bootstrap-server kafka:9092 --replication-factor 1 --partitions 1 --command-config /admin.properties

Y para generar un error de autenticación valdría con utilizar un fichero “.properties” con credenciales incorrectas.

Con el servidor y el cliente configurados ya está configurada la autenticación de Kafka, ahora será necesario configurar la autorización, es decir, crear ACL’s para definir los permisos de acceso y posteriormente comprobar mediante un acceso correcto y un acceso denegado si aparecen sus logs correspondientes en el fichero kafka-authorizer.log.

La creación de una ACL para permitir a el usuario usuario1 producir datos a el topico1 y al usuario2 leer del topico1 se hace mediante los siguientes comandos:

docker exec -it kafka kafka-acls.sh --bootstrap-server localhost:9092 --command-config /admin.properties --add --allow-principal User:usuario1 --operation Write --topic topico1

docker exec -it kafka kafka-acls.sh --bootstrap-server localhost:9092 --command-config /admin.properties --add --allow-principal User:usuario2 --operation Read --group '\*' --topic topico1

Para revisar la lista de ACL’s creadas:

docker exec -it kafka kafka-acls.sh --bootstrap-server localhost:9092 --command-config /admin.properties --list

Para enviar un mensaje al topico1:

# CORRECTO (el usuario1 puede escribir en el topico1)

docker exec -it kafka kafka-console-producer.sh --broker-list localhost:9092 --topic topico1 --producer.config producer.properties

# INCORRECTO (el usuario2 no puede escribir en el topico1)

docker exec -it kafka kafka-console-producer.sh --broker-list localhost:9092 --topic topico1 --producer.config consumer.properties

Y para leer el topico1:

# CORRECTO (el usuario2 puede leer del topico1)

docker exec -it kafka kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic topico1 --from-beginning --consumer.config consumer.properties

# INCORRECTO (el usuario1 no puede leer del topico1)

docker exec -it kafka kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic topico1 --from-beginning --consumer.config producer.properties

Con esta configuración ya podemos generar logs tanto de autenticación (en server.log) como de autorización (kafka-authorizer.log).

# Despliegue del proyecto

Para el despliegue del proyecto se ha decidido emplear Docker y Docker Compose. Los principales motivos son los siguientes:

* + **Aislamiento entre servicios:** Cada aplicación se ejecuta con sus dependencias en un contenedor independiente y aislado, lo que mejora la seguridad y reduce los conflictos entre aplicaciones.
  + **Entornos de desarrollo y producción idénticos:** Docker soluciona el conocido problema "funciona en mi máquina" ya que los contenedores se pueden ejecutar en cualquier entorno que soporte Docker sin realizar ningún cambio.
  + **Facilidad de despliegue y modificación:** Con Docker Compose, el despliegue se hace únicamente a través de un archivo y un solo comando.
  + **Portabilidad:** Docker encapsula aplicaciones y dependencias en contenedores, esto permite que puedan ser ejecutados en cualquier entorno compatible con Docker sin necesidad de realizar ningún cambio.
  + **Escalabilidad:** Docker permite la creación de múltiples réplicas de un contenedor para repartir la carga de trabajo.
  + **Uso de recursos:** A diferencia de las máquinas virtuales, los contenedores de Docker utilizan el mismo kernel que el sistema operativo, además son más ligeros y ocupan menos espacio en memoria RAM y disco.

## Volúmenes Docker

Los volúmenes en Docker sirven para persistir y compartir datos entre contenedor-contenedor y máquina local-contenedor, hay dos tipos principales de montajes en Docker:

* + **Bind mounts**: pueden ser almacenados en cualquier lugar en el sistema de archivos del host. Pueden ser muy eficientes, pero dependen de la estructura de directorios del sistema de archivos del host.
  + **Volúmenes**: son gestionados por Docker y son la mejor manera de persistir datos en Docker. Los datos no se almacenan en una parte específica del sistema de archivos del host, sino en una parte del sistema de archivos gestionada por Docker.

En este proyecto, se ha probado a gestionar el almacenamiento de logs y la relación entre servicios-logstash mediante bind mounts, sin embargo, después de probar su funcionamiento se ha optado por utilizar volúmenes para evitar almacenar ninguna información en la maquina local.

## Configuración del fichero .yaml

Se ha utilizado un fichero .yaml para el despliegue de la arquitectura de los contenedores utilizando Docker Compose. A continuación se muestra el fichero que se ha utilizado para el despliegue de los contenedores utilizados en el proyecto.

version: '3.7'

services:

   zookeeper:

      image: zookeeper

      container\_name: zookeeper

      restart: always

      networks:

         - storm

      environment:

         ZOO\_MY\_ID: 1

         ZOO\_SERVERS: server.1=0.0.0.0:2888:3888;2181

      ports:

         - "2181:2181"

         - "2888:2888"

         - "3888:3888"

spark-master:

      image: bitnami/spark:2.4.5

      networks:

         - storm

      restart: always

      environment:

         - SPARK\_MODE=master

      ports:

         - "7077:7077"

         - "8080:8080"

spark-worker:

      image: bitnami/spark:2.4.5

      networks:

         - storm

      environment:

         - SPARK\_MODE=worker

         - SPARK\_MASTER\_URL=spark://spark-master:7077

      ports:

         - "8081:8081"

      restart: always

kafka:

      image: wurstmeister/kafka

      container\_name: kafka

      ports:

         - "9092:9092"

      environment:

         HOSTNAME: kafka

         HOSTNAME\_COMMAND: "docker info | grep 'Node Address: ' | cut -d' ' -f 5"

         KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181

         # Modificación para configurar autenticación y autorización

         KAFKA\_ADVERTISED\_LISTENERS: SASL\_PLAINTEXT://localhost:9092

         KAFKA\_LISTENERS: SASL\_PLAINTEXT://:9092

         KAFKA\_INTER\_BROKER\_LISTENER\_NAME: SASL\_PLAINTEXT

         KAFKA\_AUTHORIZER\_CLASS\_NAME: kafka.security.authorizer.AclAuthorizer

         KAFKA\_SUPER\_USERS: User:admin;User:admin2

         KAFKA\_SASL\_MECHANISM\_INTER\_BROKER\_PROTOCOL: PLAIN

         KAFKA\_SASL\_ENABLED\_MECHANISMS: PLAIN

         KAFKA\_OPTS: "-Djava.security.auth.login.config=/jaas.conf"

      volumes:

         - /var/run/docker.sock:/var/run/docker.sock

# Volumen con los logs de Kafka

         - kafka\_logs:/opt/kafka/logs

         # Definición de credenciales

         - C:/TFGLogs/jaas.conf:/jaas.conf

         # Credenciales incorrectas para autenticación

         - C:/TFGLogs/client.properties:/client.properties

         # Credenciales correctas para autenticación

         - C:/TFGLogs/admin.properties:/admin.properties

         # Credenciales usuario1 correctas

         - C:/TFGLogs/producer.properties:/producer.properties

         # Credenciales usuario2 correctas

         - C:/TFGLogs/consumer.properties:/consumer.properties

      networks:

         - storm

cassandra:

      image: cassandra

      container\_name: cassandra

      networks:

         - storm

      volumes:

         - C:/TFGLogs/cassandra\_data/apache:/home/apache

         - C:/TFGLogs/cassandra\_data/logs:/var/log/cassandra

      environment:

         CASSANDRA\_BROADCAST\_ADDRESS: cassandra

         CASSANDRA\_CLUSTER\_NAME: POLLUTION\_CLUSTER

         CASSANDRA\_ENDPOIN\_SNITCH: GossipingPropertyFileSnitch

         CASSANDRA\_DC: dc1

         CASSANDRA\_RACK: rack1

         CASSANDRA\_SEEDS: cassandra

      ports:

         - "9042:9042"

elasticsearch:

      image: docker.elastic.co/elasticsearch/elasticsearch:7.9.3

      environment:

         - discovery.type=single-node

      volumes:

         - C:/TFGLogs/elasticsearch\_data:/usr/share/elasticsearch/data

      ports:

         - "9200:9200"

         - "9300:9300"

      networks:

         - storm

logstash:

      image: docker.elastic.co/logstash/logstash:7.9.3

      volumes:

         # Ficheros de configuracion de Logstash

         - C:/TFGLogs/logstash/logstash\_config:/usr/share/logstash/config

         - C:/TFGLogs/logstash/logstash\_pipeline:/usr/share/logstash/pipeline

      ports:

         - "5044:5044"

      depends\_on:

         - elasticsearch

      networks:

         - storm

kibana:

      image: docker.elastic.co/kibana/kibana:7.9.3

      ports:

         - "5601:5601"

      environment:

         ELASTICSEARCH\_URL: http://elasticsearch:9200

         ELASTICSEARCH\_HOSTS: http://elasticsearch:9200

      depends\_on:

         - elasticsearch

      networks:

         - storm

networks:

   storm:

      driver: bridge

filebeat:

      image: docker.elastic.co/beats/filebeat:7.9.3

      user: root # Necesario para leer los logs de Docker

      volumes:

        - /var/lib/docker/containers:/var/lib/docker/containers:ro

        - /var/run/docker.sock:/var/run/docker.sock:ro

        # Archivo de configuración

        - C:/TFGLogs/filebeat.yml:/usr/share/filebeat/filebeat.yml

        # Volumen con los logs de Kafka

        - kafka\_logs:/usr/share/filebeat/kafka\_logs

      networks:

        - storm

      depends\_on:

        - kafka

        - zookeeper

        - cassandra

        - elasticsearch

        - logstash

      command: filebeat -e -strict.perms=false

      networks:

         - storm

volumes:

   kafka\_logs: # Define el volumen para logs de Kafka

     external: true

# Configuración del Stack ELK

Se ha mostrado previamente el despliegue de los contenedores con Docker, después del despliegue es necesario configurar las herramientas del stack ELK para su funcionamiento.

## Configuración de Logstash

Como se ha explicado en el funcionamiento de Logstash, este será el encargado de recibir los logs desde Filebeat, filtrarlos y enviarlos a Elasticsearch para su almacenamiento.

### Ficheros de configuración

Logstash tiene principalmente dos ficheros de configuración:

* **logstash.conf:** archivo que se usa para definir la configuración de entrada, filtro y salida de logs. Se adjunta un ejemplo de configuración utilizada para recibir los logs de autenticación y autorización desde Filebeat, filtrarlos y enviarlos a su índice correspondiente en Elasticsearch, en este caso se usará un índice para los logs de autorización y otro para los de autenticación.

input {

  beats {

    port => 5044

  }

}

Previamente, en Filebeat se ha añadido el campo descriptivo “log\_type” a los logs de autenticación (server.log) y autorización (kafka-authorizer.log).

En estos filtros, se deja pasar directamente a Elasticsearch los logs de autorización, ya que todos los logs del fichero kafka-authorizer.log son de autorización. Sin embargo, en el fichero server.log hay muchos más logs que los de autenticación, por lo que se filtra que contengan el mensaje “Failed authentication”.

filter {

    if [log\_type] == "autenticacion" {

      # Filtrar solo los mensajes de autenticación fallida

      if [message] !~ /Failed authentication/ {

        drop { }

      }

    }

}

La salida es un índice para los logs de autenticación y otro para los de autorización.

output {

  if [log\_type] == "autenticacion" {

    elasticsearch {

      hosts => ["http://elasticsearch:9200"]

      index => "logstash-kafka-autenticacion-%{+YYYY.MM}"

    }

  } else if [log\_type] == "autorizacion" {

    elasticsearch {

      hosts => ["http://elasticsearch:9200"]

      index => "logstash-kafka-autorizacion-%{+YYYY.MM}"

    }

  }

  stdout { codec => rubydebug }

}

La siguiente línea de código sirve para enviar la salida de los eventos procesados a la consola estándar (stdout) de Logstash, además imprime los eventos en un formato fácil de leer para el usuario. Esto puede ser útil durante el desarrollo ya que permite ver en tiempo real como Logstash está procesando los datos. Sin embargo, en entornos de producción o cuando se procesan grandes volúmenes de datos, es común eliminar esta línea para evitar una sobrecarga innecesaria y mejorar el rendimiento.

stdout { codec => rubydebug }

* **logstash.yml:** se utiliza para configurar opciones de Logstash, como parámetros de JVM, nivel de logging y otros ajustes del sistema, se adjunta la configuración utilizada.

path.config: /usr/share/logstash/pipeline

log.level: info

# Los niveles de log pueden ser trace, debug, info, warning, error, y fatales.

path.logs: /usr/share/logstash/logs

### Filtrado con el plugin Grok

Una de las formas más comunes de filtrar en Logstash es mediante el plugin Grok, este permite analizar texto aleatorio y convertirlo a un formato estructurado y consultable.

Se ha comentado previamente que Logstash por defecto procesa los logs con unos determinados campos identificativos, esta tarea la realiza únicamente el plugin Grok.

Por ejemplo, a los campos diseñados por defecto puede ser necesario añadir otros como el nivel de logging (INFO, WARN, ERROR), de forma que en Kibana se pueda filtrar por este campo. Para modificar la configuración de los campos tendremos que utilizar también el plugin Grok.

Dependiendo del formato de los logs que se quieran procesar, se tendrá que utilizar el plugin Grok de una forma u otra, por lo que es importante examinar el formato de los archivos de logs previamente.

Para entender mejor la sintaxis de Grok se pondrá un ejemplo concreto realizado en el proyecto. En este caso estamos tratando los logs de Kafka, más específicamente de este log del fichero de log server.log:

[2024-03-04 13:40:19,449] INFO [Log partition=topico1-0, dir=/kafka/kafka-logs-kafka] Loading producer state till offset 0 with message format version 2 (kafka.log.Log)

Aplicando el siguiente filtro con el plugin Grok se podría estructurar los logs de este fichero en sus campos identificativos según su formato. Cabe destacar que los filtros se deben personalizar para cada fichero de log de cada servicio, ya que no hay un formato estandarizado, y si defines en los filtros un campo que no se detecta en ese fichero en concreto habrá errores en la visualización del log en Kibana.

Para darle un formato correcto a este log en concreto se podría usar este filtro Grok:

\[%{TIMESTAMP\_ISO8601:log\_timestamp}\] %{LOGLEVEL:log\_level} \[Log partition=%{DATA:partition}, dir=%{PATH:log\_dir}\] %{GREEDYDATA:message} \(%{JAVACLASS:java\_class}\)

**@version**:1 **@timestamp**: Mar 4, 2024 @ 14:40:19.976 **type**: kafka **path**: /usr/share/logstash/kafka\_logs/broker/server.log **host**:126f1c66a4db **message**: [2024-03-04 13:40:19,449] INFO [Log partition=topico1-0, dir=/kafka/kafka-logs-kafka] Loading producer state till offset 0 with message format version 2 (kafka.log.Log) **\_id**: rACxCY4B0A2jgDkHo9Bp **\_type**: \_doc

**\_index**: logstash-kafka-2024.10 **\_score**: -

En el cuadro anterior vemos como quedaría la visualización en Kibana sin personalizar ningún filtro en Logstash, a simple vista está que la organización en campos es mejorable, ya que se añade la fecha en el campo “MESSAGE” además de en su propio campo, no existe un campo “LEVEL” con el nivel del log (INFO, WARN, ERROR), por lo que no podríamos filtrar en el cuadro de mando por este campo tan importante.

Sin embargo, la configuración de filtros para el fichero server.log de Kafka no acaba aquí, ya que este filtro solo será útil para el log previo, pero en este fichero se almacenan decenas o cientos de logs con formatos un poco diferentes, por ejemplo:

[2024-03-04 13:38:45,525] INFO [GroupCoordinator 1001]: Starting up. (kafka.coordinator.group.GroupCoordinator)

Para este otro log podríamos usar un filtro diferente, como el siguiente:

\[%{TIMESTAMP\_ISO8601:log\_timestamp}\] %{LOGLEVEL:log\_level} \[%{DATA:component} %{INT:coordinator\_id}\]: %{GREEDYDATA:message} \(%{JAVACLASS:java\_class}\)

Ahora, ¿si para un mismo fichero de logs de un servicio tenemos que usar diferentes filtros cómo lo hacemos?

La respuesta es mediante el uso de la directiva match y el modificador “break\_on\_match => false”, esto permite intentar cada patrón de filtrado hasta que el formato del log coincida con uno de ellos.

Para complementar los dos filtros puestos como ejemplo previamente sobre el mismo fichero server.log del servicio Kafka el filtrado quedaría así:

filter {

if [path] =~ /server\.log$/ and [type] == "kafka" {

grok {

break\_on\_match => false

match => {

"message" => [

"\[%{TIMESTAMP\_ISO8601:log\_timestamp}\] %{LOGLEVEL:log\_level}\[Log partition=%{DATA:partition}, dir=%{PATH:log\_dir}\] %{GREEDYDATA:log\_message} \(%{JAVACLASS:java\_class}\)",

"\[%{TIMESTAMP\_ISO8601:log\_timestamp}\] %{LOGLEVEL:log\_level} \[%{DATA:component} %{INT:coordinator\_id}\]: %{GREEDYDATA:log\_message} \(%{JAVACLASS:java\_class}\)"

]

}

tag\_on\_failure => ["\_grokparsefailure\_serverlog"]

}

}

}

En el cuadro superior podemos ver un ejemplo de configuración de filtrado sobre el fichero server.log de Kafka, este filtrado se podria repetir para cada fichero incluyendo todos los formatos de logs que se detecten en cada uno de ellos.

La anotación “tag\_on\_failure” indica que a los logs que no coincidan con ninguno de los formatos de campos indicados se les añadirá el campo “tag\_on\_failure” con valor “\_grokparsefailure\_serverlog”. En caso de no querer que exista este campo y se trate por defecto a los logs que no coincidan se omitirá esta línea de código.

Como hemos visto previamente, puede ser muy complejo definir tantos campos específicos para cada log de cada fichero y servicio, por ello lo normal sería utilizar o los campos por defecto de Logstash o unos campos generales que sirvan para todos los logs, y posteriormente si hay un tipo de log que se quiera monitorizar especialmente, diseñar campos adicionales para ese caso en específico.

Por ejemplo, para la gestión de logs del servicio Kafka, vemos que de los 6 ficheros de logs solo hay uno que no siga un patrón general de FECHA-NIVEL-MENSAJE (aunque dentro del mensaje se podrían especificar más campos).

En la siguiente imagen se ve la estructura de 5 de los 6 ficheros de logs de Kafka:

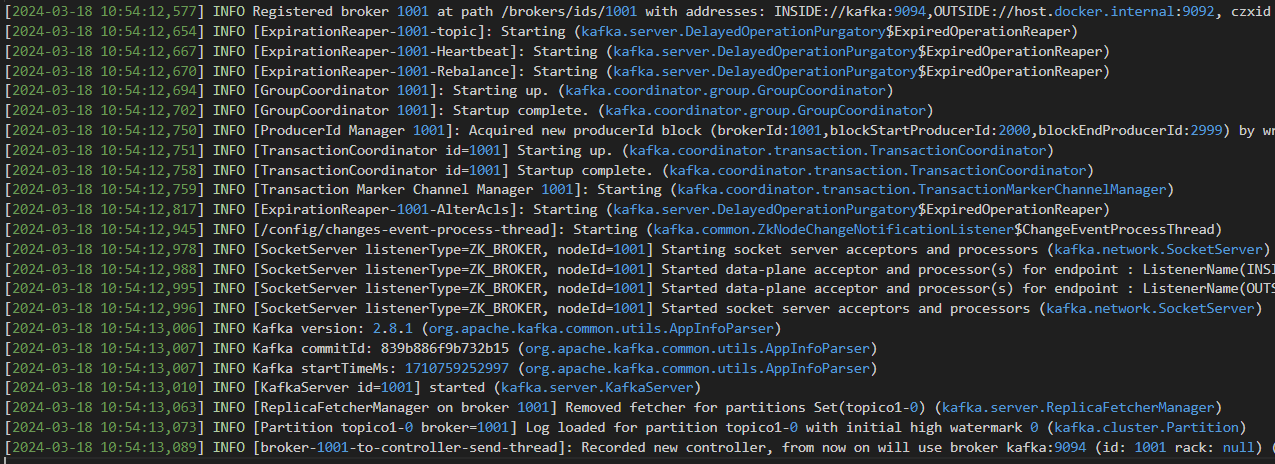


Imagen 5: Formato de logging de 5 ficheros de Kafka

En la siguiente imagen se ve la estructura del único fichero de Kafka que no cumple la estructura anterior:

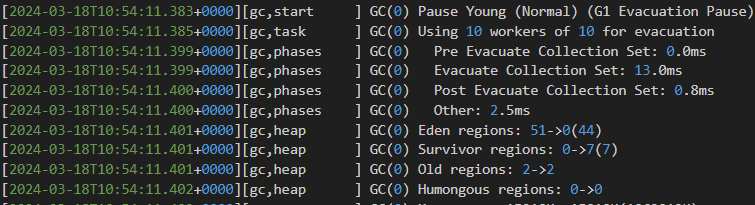


Imagen 6: Formato de logging de 1 fichero de Kafka

Para hacer una gestión simple y general de estos ficheros podríamos definir unos campos para los 5 ficheros similares y otros campos distintos para el otro fichero, aquí un ejemplo de un posible filtrado para este caso:

filter {

  if [type] == "kafka" {

    if [path] =~ /kafkaServer-gc\.log$/ {

      grok {

        match => {

          "message" => [

            "\[%{TIMESTAMP\_ISO8601:timestamp}\]\[%{DATA:gc\_tags}\] %{GREEDYDATA:gc\_message}"

          ]

        }

      }

    }

    else {

      grok {

        match => {

          "message" => [

            "\[%{TIMESTAMP\_ISO8601:timestamp}\] %{LOGLEVEL:loglevel} %{GREEDYDATA:message}"

          ]

        }

      }

    }

  }

}

Con esta última configuración de filtrado mostrada, vamos a visualizar desde Kibana como se han generado los campos:

Los logs del fichero server.log (de los 5 ficheros iguales) quedan así, y como podemos ver, se ha generado el campo “loglevel”:

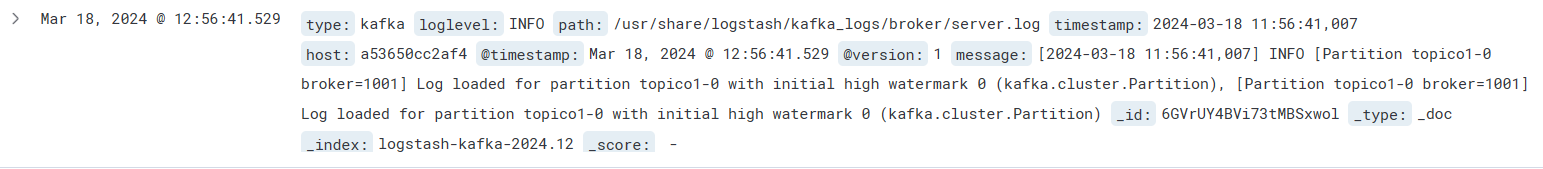


Imagen 7: Visualización de campos en Kibana para los 5 ficheros

Y los logs del fichero tratado de forma diferente quedan con los siguientes campos, podemos ver que se ha añadido el campo “gc\_tags”:

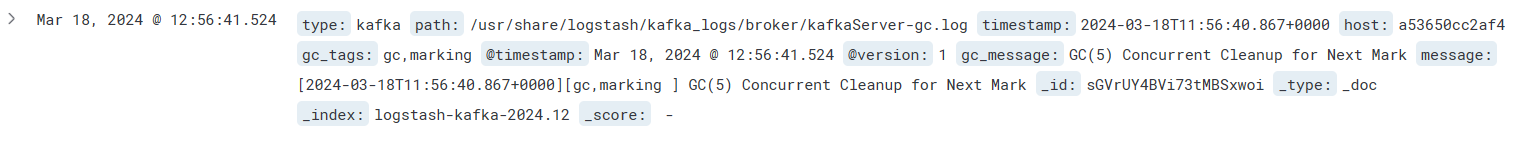


Imagen 8: Visualización de campos en Kibana para el fichero independiente

Como vemos en las imágenes superiores, además de los 3 campos indicados para cada fichero, Logstash ha añadido los restantes campos por defecto, si quisiéramos eliminar alguno de estos campos podríamos usar la siguiente directiva:

mutate {

remove\_field => ["path", "host", "@version", "type"]

}

Hemos repasado el funcionamiento del plugin Grok con diferentes ejemplos y llegamos a la conclusión de que hay infinitas formas de utilizar Logstash para filtrar y enriquecer los logs, por lo que el uso que se le dé dependerá en gran parte del contexto.

## Configuración de Elasticsearch

Una vez sabemos cómo funciona Elasticsearch, podemos empezar a configurarlo. Como se ha explicado previamente, Elasticsearch se basa para almacenar la información en índices y documentos json.

Para determinar cómo almacenar la información en Elasticsearch, es esencial tener claro cómo queremos analizar los logs posteriormente, es decir, si en el contexto del proyecto se va a querer visualizar KPIs o indicadores con la información de los logs de la última semana, quizás lo más apropiado sería crear un índice mensual para cada servicio, de forma que en Kibana se pueda hacer consultas directamente sobre el índice determinado.

En este proyecto en particular, la estrategia será la mencionada previamente, se creará un índice semanal para cada servicio empleado, además de un índice para autenticación y otro para autorización de Kafka.

En la práctica, la forma de configurar la creación automática de un índice mensual se gestiona desde el fichero de configuración de Logstash, donde se indica el output de los logs.

Se adjunta un ejemplo de output en Logstash para crear un índice mensual por servicio.

output {

  elasticsearch {

    hosts => ["http://elasticsearch:9200"]

    index => "logstash-%{type}-%{+YYYY.MM}"

  }

  stdout { codec => rubydebug }

}

Al acceder a Kibana desde “localhost:5601” tenemos que seguir unos pasos de configuración:

1. Crear uno o varios patrones para la nomenclatura de los índices: en este proyecto se utiliza únicamente el patrón siguiente:

logstash-\*

Esto indica que todos los índices tendrán un nombre siguiendo este formato, acorde con “logstash-{servicio}”.

1. Selección del campo utilizado como filtro temporal: normalmente se utiliza el campo “timestamp” como referencia temporal.
2. Configuración del tipo dato de cada campo que contendrá ese índice en concreto. Como vamos a utilizar el mapeo dinámico de Elasticsearch, no es necesario modificar nada.

Una vez tenemos Elasticsearch desplegado y configurado, es importante monitorizar el estado del cluster, para ello podemos usar comandos como los siguientes:

Este comando sirve para verificar el estado de salud del cluster de Elasticsearch. Lo ideal sería que se encontrase en estado “Green”:

curl -X GET "http://localhost:9200/\_cat/health?v"

El siguiente comando sirve para verificar los índices que existen en el cluster, de forma que puedas verificar la configuración de índices que hayas creado previamente:

curl http://localhost:9200/\_cat/indices

Además, el comando curl también permite eliminar índices de esta forma:

curl -X DELETE "localhost:9200/index-name"

Se muestra a continuación una imagen con la salida al comando de consulta de los índices, de forma que se observe los índices creados por defecto por Elasticsearch.

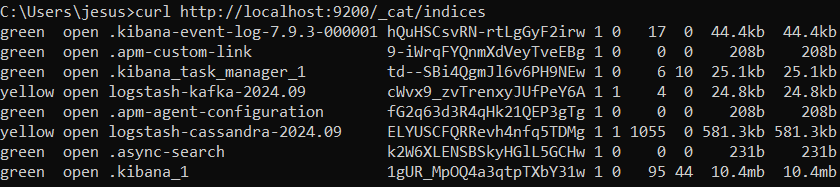
****

Imagen 9: Índices por defecto de Elasticsearch

Como se puede ver en la imagen superior, Elasticsearch por defecto tiene creados 6 índices con funciones predeterminadas:

* + **.kibana-event-log-7.9.3-000001**: es un índice utilizado por Kibana para registrar errores, ejecuciones en segundo plano, alertas, etc.
  + **.apm-custom-link**: es un índice utilizado por APM (Application Performance Monitoring) de Elasticsearch para almacenar enlaces personalizados.
  + **.kibana\_task\_manager\_1**: es un índice utilizado por Kibana para gestionar tareas en segundo plano.
  + **.apm-agent-configuration**: índice utilizado por APM (Application Performance Monitoring) para almacenar configuraciones personalizadas de los agentes APM, de forma que se pueda configurar los agentes APM desde Kibana directamente.
  + **.async-search**: índice utilizado por Elasticsearch para la función de búsqueda asíncrona (para consultas de larga duración).
  + **.kibana\_1**: es uno de los índices más importantes de Kibana, en el se almacenan los dashboards, visualizaciones, configuraciones, etc.

**TODO: Cambiar imagen a una sin un índice semanal, poner una actual con autenticación y autorización.**

**TODO: Reescribir la descripción de los índices**

## Configuración de Filebeat

Como dijimos previamente, se va a utilizar Filebeat en el proyecto, por lo que es necesario configurarlo para para recoger los logs de los servicios y enviárselos a Logstash, de forma que Logstash procese y filtre los logs para enviárselos a Elasticsearch.

A continuación, se deja un ejemplo de fichero “filebeat.yml” de configuración de Filebeat:

filebeat.inputs:

- type: log

  enabled: true

  paths:

    - /usr/share/filebeat/kafka\_logs/server.log # Logs de server.log (ahí están los logs de autenticación)

  fields:

    log\_source: kafka

    log\_type: autenticacion

  fields\_under\_root: true

- type: log

  enabled: true

  paths:

    - /usr/share/filebeat/kafka\_logs/kafka-authorizer.log # Logs de autorización

  fields:

    log\_source: kafka

    log\_type: autorizacion

  fields\_under\_root: true

output.logstash:

  hosts: ["logstash:5044"]

Hay que asegurarse que Logstash está configurado para recibir los logs desde Filebeat. A continuación, se muestra la sección de entradas necesaria en Logstash:

input {

beats {

port => 5044

}

}

En este proyecto Filebeat se ha empezado a utilizar después de probar el funcionamiento del sistema solamente con Logstash. Por esto, se ha comprobado que al integrar Filebeat surgen ciertos problemas. Al integrar Filebeat este es el primero en acceder a los logs en crudo, por lo que por defecto les da un formato de campos y lo convierte a un json, que es enviado a Logstash. Antes de integrar Filebeat, Logstash era el primero en acceder a los logs en crudo, por lo que los campos para realizar el filtrado eran simplemente los de Logstash.

Por ejemplo, al utilizar Filebeat, al llegar los logs a Logstash el campo “type” es siempre Filebeat, en vez de Kafka, Cassandra o el nombre del servicio determinado de donde provengan los logs, por lo que si en Logstash se utilizaba el campo “type” para filtrar los logs según su origen será necesario modificar este filtrado para identificar el origen de los logs de otra forma, por ejemplo, creando en Filebeat un nuevo campo “log\_source” y dándole un valor determinado en función de la ruta de donde procedan los logs, para en Logstash posteriormente poder filtrar por este campo.

Justo la estrategia mencionada de crear un campo “log\_source” se ha aplicado en la configuración mostrada previamente, además de añadir otro campo “log\_type” para identificar los logs de autenticación y autorización de Kafka.

Por lo tanto, al utilizar Filebeat el formato de campos que llega a Logstash es el siguiente:

{

"@timestamp" => 2024-04-06T14:21:30.826Z,

"@version" => "1",

"agent" => {

"ephemeral\_id" => "8ab4e6fa-4f73-4b51-bd19-a874b8c18226",

"hostname" => "cd954ed329da",

"id" => "9a51eb06-13c1-48d3-a185-41d449913adf",

"name" => "cd954ed329da",

"type" => "filebeat",

"version" => "7.9.3"

},

"ecs" => {

"version" => "1.5.0"

},

"host" => {

"name" => "cd954ed329da"

},

"input" => {

"type" => "log"

},

"log" => {

"file" => {

"path" => "/usr/share/filebeat/kafka\_logs/server.log"

},

"offset" => 79992

},

"log\_source" => "kafka",

“log\_type” => “autenticacion”,

"message" => "[2024-04-06 14:21:30,677] INFO [SocketServer listenerType=ZK\_BROKER, nodeId=1001] Failed authentication with /127.0.0.1 (Unexpected Kafka request of type METADATA during SASL handshake.) (org.apache.kafka.common.network.Selector)",

"tags" => [

"beats\_input\_codec\_plain\_applied"

]

}

# Visualización en Kibana

En este proyecto la única fuente de datos de la que se extraerá la información para mostrar en Kibana será Elasticsearch.

Kibana tiene múltiples funcionalidades, en este proyecto se explorarán unas cuantas.

## Observabilidad

# Despliegue local

## Limitaciones del despliegue local

Dado que estamos gestionando logs de una plataforma big data, el despliegue local del stack ELK (Elasticsearch, Logstash, Kibana) para la gestión, monitorización y análisis de logs presenta muchas limitaciones, principalmente relacionadas con el almacenamiento y la escalabilidad de la aplicación.

Durante el desarrollo de este proyecto, se ha identificado el almacenamiento como la principal limitación debido a la gran cantidad de logs que generan el conjunto de servicios de la plataforma big data. Como el encargado del almacenamiento es Elasticsearch, es en este dónde vemos reflejados los problemas del despliegue local.

## Explicación del problema de almacenamiento

Como se ha mencionado previamente, la principal limitación del despliegue local es la capacidad de almacenamiento, ya que los sistemas locales tienen una capacidad de almacenamiento fija, la cual puede ser sobrepasada rápidamente al tratarse de una plataforma big data.

Este problema se ve con facilidad en la gestión de logs, ya que los servicios generan una cantidad masiva de logs diariamente, lo cual unido a la capacidad máxima de almacenamiento acaba resultando en errores críticos como el siguiente:

"TOO\_MANY\_REQUESTS/12/disk usage exceeded flood-stage watermark, index has read-only-allow-delete block"

Cuando se da el problema superior, Elasticsearch marca los índices como solo lectura para prevenir la pérdida de datos debido a la falta de espacio, y esto inutiliza el sistema por completo, ya que en Kibana dejan de mostrarse los nuevos logs.

Viendo el problema anterior, podemos concluir que la escalabilidad en un despliegue local se convierte en todo un desafío. A diferencia de las soluciones en la nube, que permiten una escalabilidad casi infinita, en local la escalabilidad necesita de una intervención manual constante para actualizar el hardware y configurar el software de forma que se adapte al gran crecimiento de datos, lo cual seguramente acabaría desembocando en problemas similares al descrito previamente. Esto hace que acabe siendo mucho más costoso, lento y propenso a errores.

## Explicación de la solución local

Para eliminar este error en el despliegue local, se deben seguir los siguientes pasos:

1. Liberar espacio en disco: Esto puede implicar eliminar logs antiguos o no necesarios, mover datos a otro almacenamiento, o aumentar la capacidad de almacenamiento disponible para tu instancia de Elasticsearch.
2. Quitar el bloqueo de solo lectura del índice: Después de liberar espacio en disco, se debe eliminar el bloqueo de solo lectura con el siguiente comando:

curl -X PUT "http://localhost:9200/logstash-kafka-\*/\_settings" -H "Content-Type: application/json" -d "{\"index.blocks.read\_only\_allow\_delete\": null}"

Para ver el uso de memoria que está haciendo Elasticsearch se puede usar este comando:

curl -X GET http://localhost:9200/\_cat/nodes?v&h=ip,disk.total,disk.used,disk.avail,disk.used\_percent

Y para ver la asignación de disco a cada nodo de Elasticsearch:

curl -X GET http://localhost:9200/\_cat/allocation?v

Para este proyecto, la salida de este último comando sería normalmente similar a esta, como se puede ver la asignación de disco está casi llena por el problema descrito anteriormente:

shards disk.indices disk.used disk.avail disk.total disk.percent host ip

9 10.9mb 375.8gb 14.1gb 389.9gb 96 172.19.0.8 172.19.0.8

Por lo tanto, para solucionar este problema podríamos seguir estas estrategias:

1. Eliminar el contenedor y volumen periódicamente.
2. Aumentar la asignación de disco destinada a Elasticsearch, sin embargo, se acabaría llenando y habría que aplicar la solución anterior tarde o temprano.
3. Aplicando políticas ILM en Elasticsearch para eliminar índices cuando lleguen a una determinada capacidad de almacenamiento o según su antigüedad.

**TODO: ¿Las políticas ILM eliminan los datos o solo el índice? ¿Aplicar unas políticas ILM como ejemplo?**

# Explicación de la solución en la nube

Para solucionar los problemas previamente descritos, se pretende desplegar el sistema de gestión de logs en la nube, más específicamente en la nube de Google, Google Cloud, ya que nos permite usar la prueba gratuita.

# Objetivos de desarrollo futuro

# Bibliografía

1. *Analíticas de logs. [Consulta: 23 enero 2024]. Enlace:*

<https://www.elastic.co/es/what-is/log-analytics>

1. *Elasticsearch. [Consulta 25 enero 2024]. Enlace:*

<https://apiumhub.com/es/tech-blog-barcelona/usar-elasticsearch-ventajas-libros/>

1. *Apache Kafka. What is Apache Kafka. [Consulta: 5 febrero 2024]. Enlace:* <https://cloud.google.com/learn/what-is-apache-kafka?hl=es>
2. *Elasticsearch. What is Elasticsearch. [Consulta: 5 febrero 2024]. Enlace:* <https://aws.amazon.com/es/what-is/elasticsearch/>
3. *Elasticsearch. What is an Elasticsearch index. [Consulta 6 febrero 2024]. Enlace:* <https://www.elastic.co/es/blog/what-is-an-elasticsearch-index>
4. *Kibana. Que es Kibana. [Consulta 16 febrero 2024]. Enlace:* <https://openwebinars.net/blog/que-es-kibana/>
5. *Grok. Plugin filters Grok. [Consulta 16 febrero 2024]. Enlace:* <https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html>
6. *Elasticsearch. Search with Elasticsearch. [Consulta 19 febrero 2024]. Enlace:* [*https://www.elastic.co/guide/en/elasticsearch/reference/current/search-with-elasticsearch.html*](https://www.elastic.co/guide/en/elasticsearch/reference/current/search-with-elasticsearch.html)
7. *Docker. Que es Docker y cuáles son sus ventajas.[Consulta 23 febrero]. Enlace:* [*https://www.dimensiona.com/es/que-es-docker-y-cuales-son-sus-ventajas/#:~:text=1.-,Portabilidad,entre%20diferentes%20entornos%20sin%20problemas*](https://www.dimensiona.com/es/que-es-docker-y-cuales-son-sus-ventajas/#:~:text=1.-,Portabilidad,entre%20diferentes%20entornos%20sin%20problemas)*.*
8. *Elasticsearch. Mapping.[Consulta 15 marzo]. Enlace:* [*https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html*](https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html)
9. *Elasticsearch. Index Lifecycle. [Consulta 15 marzo]. Enlace:* [*https://www.elastic.co/guide/en/elasticsearch/reference/current/index-lifecycle-management.html*](https://www.elastic.co/guide/en/elasticsearch/reference/current/index-lifecycle-management.html)
10. *Kafka. Authorization. [Consulta 22 marzo]. Enlace:* [*https://developer.ibm.com/tutorials/kafka-authn-authz/*](https://developer.ibm.com/tutorials/kafka-authn-authz/)

COSAS QUE HACER:

-Explicar apartados Kibana

-Crear dashboards Kibana con autenticación-autorizacion

-Mover los indices antiguos a una bdd ¿?

-Añadir sección de despliegue en la nube.

Menos importante:

-Añadir índice de figuras y cuadros.

-Error autorización de lectura

MUY A FUTURO:

-Cambiar Spark por Storm y ver si funciona el programa java de Storm.

**RESUMEN:**

1. mvn clean package
2. Configurar fichero .yaml (volumenes)
3. Hacer deploy

docker volume create kafka\_logs

docker compose -f stack3ELK.yaml up

1. Crear tópico kafka

docker exec kafka /opt/kafka\_2.13-2.8.1/bin/kafka-topics.sh --create --topic topico1 --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1

CORRECTO:

docker exec kafka /opt/kafka/bin/kafka-topics.sh --create --topic topico1 --bootstrap-server kafka:9092 --replication-factor 1 --partitions 1 --command-config /admin.properties

INCORRECTO:

docker exec kafka /opt/kafka/bin/kafka-topics.sh --create --topic topico1 --bootstrap-server kafka:9092 --replication-factor 1 --partitions 1 --command-config /client.properties

1. Crear keyspace y tablas en Cassandra

cqlsh localhost 9042

create keyspace santanderPollution

with replication = {'class': 'SimpleStrategy', 'replication\_factor' : 1};

use santanderPollution;

create table lecturas (

    sensor\_id int,

    lat float,

lon float,

    no2 float,

    ozone float,

    temp float,

    co float,

    particles float,

    region int,

    generated bigint,

    timeframe bigint,

    PRIMARY KEY ((sensor\_id), generated, timeframe)

) WITH CLUSTERING ORDER BY (generated DESC, timeframe DESC);

create table pollutionAgg (

    region int,

    timeframe bigint,

    no2 float,

    ozone float,

    temp float,

    co float,

    particles float,

    PRIMARY KEY ((region), timeframe)

) WITH CLUSTERING ORDER BY (timeframe DESC);

1. Cambiar el jar de guava 14 por guava 19, en el master y worker

cd jars

rm guava-14.0.1.jar

docker cp C:\TFGLogs\guava-19.0.jar tfgbackup-spark-master-1:/opt/bitnami/spark/jars/

docker cp C:\TFGLogs\guava-19.0.jar tfgbackup-spark-worker-1:/opt/bitnami/spark/jars/

1. Mandar java a Spark

Con programa Spark:

docker exec tfg-spark-master-1 /opt/bitnami/spark/bin/spark-submit --class es.unican.dintenr.santanderPollution.SantanderPollutionMain --master spark://spark-master:7077 /santanderPollutionSpark-0.0.1-SNAPSHOT-jar-with-dependencies.jar "spark://spark-master:7077" "kafka:9092" "topico1" "cassandra"

Con programa Docker (necesito Apache Storm):

docker exec tfg-spark-master-1 /opt/bitnami/spark/bin/spark-submit --class es.unican.dintenr.santanderPollution.SantanderPollutionTopology --master spark://spark-master:7077 /santanderPollution-0.0.1-SNAPSHOT-jar-with-dependencies.jar

1. Ejecutar productor que envía datos a Kafka

Cada vez que lo ejecuto se envían 3 mensajes, ya que la variable ***LEC\_PER\_PRO = 3***

java -jar C:\TFGLogs\programas\productor\santanderPollutionProducer\target\santanderPollutionProducer-1.0-jar-with-dependencies.jar "host.docker.internal:9092" "topico1" "C:\TFGLogs\programas\productor\santanderPollutionProducer\target\sensores\_smart\_mobile.csv" "1"

1. Configurar logstash.conf y logstash.yml
2. Mapear los logs locales al contenedor de logstash

GENERAR LOGS DE AUTENTICACION/AUTORIZACION EN KAFKA:

AUTENTICACION - CREAR TOPICO - CORRECTO:

docker exec kafka /opt/kafka/bin/kafka-topics.sh --create --topic topico1 --bootstrap-server kafka:9092 --replication-factor 1 --partitions 1 --command-config /admin.properties

AUTENTICACION - CREAR TOPICO - INCORRECTO:

docker exec kafka /opt/kafka/bin/kafka-topics.sh --create --topic topico1 --bootstrap-server kafka:9092 --replication-factor 1 --partitions 1 --command-config /client.properties

AUTENTICACION - ENVIAR DATOS DEL SENSOR – CORRECTO:

java -jar C:\TFGLogs\programas\productor\santanderPollutionProducer\target\santanderPollutionProducer-1.0-jar-with-dependencies.jar "host.docker.internal:9092" "topico1" "C:\TFGLogs\programas\productor\santanderPollutionProducer\target\sensores\_smart\_mobile.csv" "1" --producer.config C:/TFGLogs/admin.properties

AUTENTICACION - ENVIAR DATOS DEL SENSOR – INCORRECTO:

java -jar C:\TFGLogs\programas\productor\santanderPollutionProducer\target\santanderPollutionProducer-1.0-jar-with-dependencies.jar "host.docker.internal:9092" "topico1" "C:\TFGLogs\programas\productor\santanderPollutionProducer\target\sensores\_smart\_mobile.csv" "1"

OTROS COMANDOS UTILIZADOS:

docker network ls -> Listado de redes

docker network inspect tfg\_storm -> Inspeccionar red tfg\_storm

docker exec -u root -it kafka sh -> Abrir shell root del contenedor

COMPROBAR QUE LOS DATOS ESTAN EN KAFKA:

docker exec -it kafka bash

kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic topico1 --from-beginning

COMPROBAR QUE ESTA CREADO EL TOPICO KAFKA:

docker exec kafka /opt/kafka\_2.13-2.8.1/bin/kafka-topics.sh --list --bootstrap-server localhost:9092

METER ESTO 1:

Sí, con esa configuración en tu archivo **docker-compose.yml**, has configurado la autenticación y autorización en Kafka utilizando SASL/PLAIN. Esta configuración define cómo Kafka debe manejar las conexiones entrantes, incluyendo la autenticación de los usuarios y la autorización de sus acciones mediante ACLs, siguiendo los detalles que has especificado. Aquí hay un resumen de lo que has configurado:

* **KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP:** Estableces el protocolo de seguridad para los listeners **INSIDE** y **OUTSIDE**, especificando que para **OUTSIDE** se use **SASL\_PLAINTEXT**, lo cual indica que se requiere autenticación SASL sobre conexiones no cifradas.
* **KAFKA\_INTER\_BROKER\_LISTENER\_NAME:** Indicas que el listener **INSIDE** es el que se usará para la comunicación entre brokers, lo cual es relevante para entornos donde tienes múltiples brokers de Kafka y necesitan comunicarse de manera segura y autenticada entre ellos.
* **KAFKA\_AUTHORIZER\_CLASS\_NAME:** Especificas el uso de **kafka.security.authorizer.AclAuthorizer** para la autorización, lo que permite controlar el acceso a los recursos de Kafka mediante ACLs.
* **KAFKA\_SUPER\_USERS:** Defines los usuarios superusuarios que tienen acceso total sin restricciones, lo cual es importante para realizar tareas administrativas sin limitaciones.
* **KAFKA\_SASL\_MECHANISM\_INTER\_BROKER\_PROTOCOL:** Estableces **PLAIN** como el mecanismo SASL para la comunicación entre brokers, permitiendo la autenticación basada en un simple nombre de usuario y contraseña.
* **KAFKA\_SASL\_ENABLED\_MECHANISMS:** Habilitas **PLAIN** como un mecanismo SASL disponible para la autenticación.
* **KAFKA\_OPTS:** Pasas opciones adicionales a la JVM de Kafka, en este caso, especificando el archivo de configuración JAAS (**/jaas.conf**) que contiene los detalles para la autenticación SASL.

Con esta configuración, tu cluster de Kafka debería requerir autenticación para los clientes que se conecten a través del listener **OUTSIDE** y utilizar ACLs para autorizar operaciones específicas. Sin embargo, recuerda que esta configuración solo habilita la autenticación y autorización en el lado de Kafka. Los clientes de Kafka que se conecten al cluster también deben configurarse adecuadamente para autenticarse y tener las ACLs necesarias establecidas para permitirles realizar las operaciones deseadas.

METER ESTO ¿? 2:

Kibana suele sugerir el uso de Filebeat porque es parte de la Elastic Stack y está diseñado para facilitar la recolección, el envío y la visualización centralizada de los logs. Filebeat es un agente ligero que se instala en tus servidores, captura los logs o eventos de los archivos de log y los envía a Logstash para su procesamiento o directamente a Elasticsearch para su indexación.

La recomendación de usar Filebeat en Kibana se debe a varias razones:

1. **Integración**: Filebeat viene con módulos preconfigurados para muchos servicios comunes (como Logstash, Elasticsearch, Apache, Nginx, entre otros), lo que simplifica la configuración inicial y la integración con Kibana.
2. **Gestión centralizada**: Puede ser administrado y monitoreado de manera centralizada a través de Kibana, permitiendo la configuración y administración de Filebeat desde una interfaz de usuario amigable.
3. **Eficiencia**: Filebeat está optimizado para ser ligero en uso de recursos y cuenta con funcionalidades como backpressure y gestión inteligente de red que pueden ser más eficientes que una configuración manual.
4. **Escalabilidad**: Al ser parte del Elastic Stack, está diseñado para trabajar de manera eficiente con Elasticsearch y escalar junto con tu infraestructura.
5. **Seguridad**: Filebeat soporta la encriptación SSL para los datos que se envían y puede integrarse fácilmente con las características de seguridad de Elasticsearch.

Aunque en tu caso estás mapeando los logs directamente a Logstash, Kibana puede sugerir Filebeat como una práctica recomendada para manejar logs de una manera más automatizada y gestionada. Sin embargo, usar Filebeat no es obligatorio y si tu sistema actual cumple con tus necesidades y estás cómodo con la configuración actual, puedes continuar sin implementar Filebeat.

Para dejar de ver la recomendación, podrías buscar opciones dentro de Kibana para desactivar las sugerencias o simplemente ignorarla si prefieres seguir con tu configuración actual.

NOTAS:

**Funcionalidad de Filebeat y Logstash**: Sí, tanto Filebeat como Logstash pueden recibir logs, procesarlos y enviarlos a un destino. Sin embargo, tienen roles y capacidades distintas dentro de la pila ELK (Elasticsearch, Logstash, Kibana):

* **Filebeat** está diseñado como un agente de envío ligero. Su principal función es recoger logs de archivos o entradas específicas y enviarlos a un destino como Logstash o directamente a Elasticsearch. Filebeat es excelente para la recolección y el envío eficiente de logs sin un procesamiento complejo.
* **Logstash** es una herramienta de procesamiento de datos más potente y flexible que Filebeat. Logstash puede recoger datos de muchas fuentes, transformarlos y enriquecerlos a través de su sistema de filtrado y luego enviarlos a varios destinos. Logstash es capaz de realizar transformaciones complejas y enriquecimiento de datos, algo que Filebeat no está diseñado para hacer.

**Uso Conjunto de Filebeat y Logstash**: Correcto, cuando utilizas Filebeat junto con Logstash, Filebeat actúa como el recolector y transportista de logs, enviando los datos a Logstash en forma de JSON. Luego, Logstash puede procesar estos datos (por ejemplo, filtrar, modificar, enriquecer) antes de enviarlos a Elasticsearch u otros destinos. Esta combinación es común para optimizar el rendimiento y la escalabilidad, utilizando Filebeat para la recolección ligera y distribuida y Logstash para el procesamiento más intensivo.

**Impacto de Introducir Filebeat en los Filtros de Logstash**: Es muy probable que al introducir Filebeat en tu arquitectura, necesites ajustar o rehacer tus filtros en Logstash. Esto se debe a que Filebeat agrega metadatos adicionales a los eventos de log, y la estructura del evento que llega a Logstash puede ser diferente a lo que tus filtros originales esperaban. Por ejemplo, campos como el camino del archivo (**file.path**), el tipo de entrada (**input.type**), y metadatos del agente (**agent.\***) son añadidos por Filebeat. Si tus filtros dependían de la estructura original de los logs sin estos campos adicionales, es necesario actualizarlos para que coincidan con el nuevo formato y aprovechar o excluir adecuadamente estos nuevos campos.

curl -X POST "localhost:9200/prueba/\_doc/1" -H "Content-Type: application/json" -d "{\"nombre\": \"Prueba\", \"fecha\": \"2024-04-07\", \"mensaje\": \"Esto es una prueba de inserción directa a Elasticsearch\"}"

curl -X POST "http://elasticsearch:9200/prueba/\_doc/1" -H "Content-Type: application/json" -d "{\"nombre\": \"Prueba\", \"fecha\": \"2024-04-07\", \"mensaje\": \"Esto es una prueba de inserción directa a Elasticsearch\"}"

¿?

Mover tu configuración de Docker Compose a Google Kubernetes Engine (GKE) puede ofrecer varios beneficios significativos, especialmente si estás buscando escalabilidad, manejo automatizado de fallas, y una gestión más eficiente de la infraestructura. Aquí te explico las diferencias clave entre usar Google Compute Engine (GCE) con Docker Compose y usar Google Kubernetes Engine:

* **Google Compute Engine con Docker Compose**
* **Configuración**: Configuración más directa si ya estás familiarizado con Docker Compose. Puedes levantar tu aplicación de manera similar a cómo funciona en un entorno local.
* **Gestión**: La gestión de la infraestructura es más manual. Tienes que manejar manualmente aspectos como las actualizaciones, el escalado, y la recuperación de fallos.
* **Escalabilidad**: El escalado puede ser manual o semi-automático, generalmente requiere intervención para optimizar los recursos o para manejar más carga.
* **Resiliencia**: Menos resistente comparado con GKE. Tienes que configurar y manejar la alta disponibilidad y la recuperación ante desastres.
* **Costo**: Puede ser menos costoso para cargas de trabajo más pequeñas o menos complejas, donde el overhead adicional de Kubernetes no se justifica.
* **Google Kubernetes Engine**
* **Orquestación**: Kubernetes es un orquestador de contenedores que gestiona automáticamente el despliegue, escalado y operaciones de aplicaciones contenedorizadas.
* **Gestión Automatizada**: GKE automatiza muchas tareas como el balanceo de carga, la auto-reparación, el escalado automático y la gestión de la configuración.
* **Escalabilidad**: Escalado automático basado en la carga de trabajo. Puede ajustar los recursos automáticamente para cumplir con la demanda sin intervención manual.
* **Resiliencia**: Alta disponibilidad integrada. Kubernetes intenta asegurar que el número deseado de instancias esté siempre corriendo y redistribuye instancias automáticamente en caso de fallo de nodo.
* **Costo**: Puede ser más costoso inicialmente debido a la sobrecarga de recursos que Kubernetes requiere. Sin embargo, puede ser más eficiente a escala debido a su gestión optimizada de recursos.
* **Consideraciones de Implementación**
* **Curva de Aprendizaje**: Kubernetes tiene una curva de aprendizaje más empinada comparada con Docker Compose. Requiere un entendimiento de conceptos como pods, servicios, volúmenes persistentes, y configuraciones.
* **Infraestructura como Código**: Kubernetes favorece la infraestructura como código, lo que puede mejorar la automatización y la reproducibilidad de los entornos de despliegue.
* **Ecosistema**: Kubernetes tiene un ecosistema extenso con numerosas herramientas y extensiones que pueden ampliar y mejorar su funcionalidad.
* **Conclusión**

Elegir entre GCE con Docker Compose y GKE depende de tus necesidades específicas de proyecto, experiencia técnica y expectativas de crecimiento. Si anticipas que tu aplicación crecerá significativamente y requerirá gestionar de manera eficiente múltiples contenedores y servicios, GKE ofrece una plataforma robusta y escalable. Para proyectos más pequeños o para aquellos donde el equipo tiene más experiencia con Docker Compose y no se necesitan las características avanzadas de Kubernetes, GCE con Docker Compose puede ser más adecuado y sencillo de manejar.