A black background with blue text

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

para acceder al

**GRADO EN INGENIERÍA INFORMÁTICA**

Autor: Jesús Saiz Gutiérrez

Director: Marta Zorrilla Pantaleón

Junio - 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.

TODO: Escribir al final

**Palabras clave**

Logs, Monitorización, Cuadro de mandos, Big Data, Stack ELK

**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.

TODO: Escribir al final

**Keywords**

Logs, Monitoring, Dashboard, Big Data, ELK Stack

# Índice

[Índice 5](#_Toc166410471)

[1. Introducción 9](#_Toc166410472)

[1.1. Importancia de la gestión de logs 9](#_Toc166410473)

[1.2. Objetivos 9](#_Toc166410474)

[1.3. Arquitectura y caso de uso 10](#_Toc166410475)

[1.4. Elección de tecnologías para la gestión y análisis de logs 10](#_Toc166410476)

[1.5. Resumen de los capítulos 11](#_Toc166410477)

[2. Stack ELK 12](#_Toc166410478)

[2.1. Logstash 13](#_Toc166410479)

[2.2. Elasticsearch 14](#_Toc166410480)

[2.3. Kibana 15](#_Toc166410481)

[2.4. Filebeat 15](#_Toc166410482)

[3. Despliegue de la plataforma y Stack ELK 16](#_Toc166410483)

[3.1. Configuración del fichero .yaml 16](#_Toc166410484)

[3.2. Logs de Kafka 20](#_Toc166410485)

[3.2.1. Ficheros de logs de Kafka 21](#_Toc166410486)

[3.2.2. Autenticación vs Autorización en Kafka 21](#_Toc166410487)

[3.2.3. Configuración de la autenticación/autorización de acceso de Kafka 22](#_Toc166410488)

[4. Diseño de la solución 24](#_Toc166410489)

[5. Configuración del Stack ELK 25](#_Toc166410490)

[5.1. Filebeat 25](#_Toc166410491)

[5.2. Logstash 27](#_Toc166410492)

[5.2.1. Ficheros de configuración 27](#_Toc166410493)

[5.2.2. Explicación del plugin Grok con ejemplos 29](#_Toc166410494)

[5.3. Elasticsearch 33](#_Toc166410495)

[6. Cuadro de mandos - Kibana 34](#_Toc166410496)

[6.1. Observabilidad 34](#_Toc166410497)

[6.2. Análisis 35](#_Toc166410498)

[6.3. Dashboards y Lens 35](#_Toc166410499)

[7. Limitaciones del despliegue local 39](#_Toc166410500)

[7.1. Explicación del problema de almacenamiento 40](#_Toc166410501)

[7.2. Explicación de la solución local 40](#_Toc166410502)

[8. Explicación de la solución en la nube 41](#_Toc166410503)

[9. Objetivos de desarrollo futuro 41](#_Toc166410504)

[10. Bibliografía 42](#_Toc166410505)

**Índice de imágenes**

[Imagen 1: Arquitectura general del sistema (sin Filebeat). 12](#_Toc166410506)

[Imagen 2: Formato de logs proporcionado por Logstash por defecto. 13](#_Toc166410507)

[Imagen 3: Arquitectura de la solución final usando Filebeat. 15](#_Toc166410508)

[Imagen 4: Formato de logging de 5 ficheros de Kafka 31](#_Toc166410509)

[Imagen 5: Formato de logging de 1 fichero de Kafka 31](#_Toc166410510)

[Imagen 6: Visualización de campos en Kibana para los 5 ficheros 32](#_Toc166410511)

[Imagen 7: Visualización de campos en Kibana para el fichero independiente 32](#_Toc166410512)

[Imagen 8: Índices por defecto de Elasticsearch 33](#_Toc166410513)

[Imagen 9: Menú principal de Observability de Kibana 34](#_Toc166410514)

[Imagen 10: Otras opciones del menú Observability de Kibana 35](#_Toc166410515)

[Imagen 11: Filtrado errores de acceso del usuario2 35](#_Toc166410516)

[Imagen 12: Creación de una visualización con Lens 36](#_Toc166410517)

[Imagen 13: Dashboard para la autenticación y autorización de Kafka 36](#_Toc166410518)

[Imagen 14: Número de fallos de autorización por usuario 36](#_Toc166410519)

[Imagen 15: Número de fallos de autenticación por ip 37](#_Toc166410520)

[Imagen 16: Fallos de autorización por usuario en el dominio del tiempo 37](#_Toc166410521)

[Imagen 17: Fallos de autenticación por ip en el dominio del tiempo 38](#_Toc166410522)

[Imagen 18: Tipos de operaciones de autorización denegadas 38](#_Toc166410523)

[Imagen 19: Tipos de peticiones de autorización denegadas 38](#_Toc166410524)

[Imagen 20: Tipo de recursos a los que se intenta acceder 39](#_Toc166410525)

[Imagen 21: Recursos específicos a los que se intenta acceder 39](#_Toc166410526)

[Imagen 22: Dashboard filtrado por la actividad del usuario usuario1 39](#_Toc166410527)

TODO: ¿Índice de cuadros?

# Introducción

Este Trabajo de Fin de Grado se ha realizado en relación con el proyecto de investigación sobre la monitorización y el análisis de datos en streaming en sistemas distribuidos y de gran escalabilidad. La meta era crear un sistema de seguimiento y análisis de datos operando en una infraestructura de big data, con Kappa como arquitectura y alineada con el enfoque del metamodelo RAI4.0.

Este metamodelo busca ofrecer una estrategia para el despliegue de aplicaciones que requieren un uso intensivo de datos en contextos distribuidos que emplean, entre otros, sistemas de IoT o computación en la nube.

Por consiguiente, el proyecto tiene como finalidad presentar un ejemplo práctico originado del IoT industrial utilizando Kafka como base de la plataforma big data, además de otras herramientas que permiten la visualización de datos y supervisión de los servicios de la plataforma big data.

TODO: MEJORAR

## Importancia de la gestión de logs

La gestión de logs es crucial porque permite a las organizaciones monitorizar y analizar el comportamiento de sus servicios.

Los logs actúan como un registro detallado de eventos, errores y transacciones, lo que ayuda a identificar problemas, a entender el funcionamiento del servicio y a detectar posibles brechas de seguridad. Una gestión efectiva de logs facilita la resolución proactiva de incidencias, mejora el rendimiento de los sistemas y asegura el cumplimiento de las normativas de auditoría y seguridad.

En esencia, los logs son una pieza clave para mantener la integridad y la operatividad óptima de los servicios tecnológicos utilizados por la organización.

TODO: REFERENCIAS A BIBLIOGRAFIA

## Objetivos

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

Para su consecución se realizarán las siguientes fases:

* **Desplegar y configurar la plataforma de servicios big data que será auditada y monitorizada por el sistema.**
* **Definir e implementar un pipeline para el procesamiento de logs**: El objetivo es desarrollar un flujo de datos con Filebeat, Logstash, Elasticsearch y Kibana para facilitar la captura, procesamiento y visualización de los registros.
* **Determinar y establecer estrategias para el filtrado y procesamiento de logs**: Es esencial diseñar criterios para estructurar los logs filtrándolos y procesándolos adecuadamente, de acuerdo con los requisitos de análisis que se especifiquen.
* **Crear cuadros de mando o dashboards**: Esto es, crear interfaces visuales que faciliten y ayuden a la interpretación de los logs y su análisis para la toma de decisiones.
* **Despliegue en la nube**: La última fase consistirá en el despliegue del sistema de gestión de logs 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.

## Arquitectura y caso de uso

El sistema bajo desarrollo se ha de implantar sobre una plataforma digital conforme al metamodelo RAI4.0. Esto es un sistema distribuido soportado sobre un bus de datos en el que están desplegados distintos servicios. Las tecnologías que dan soporte a estos servicios son las que deben estar monitorizadas.

Estas tecnologías son:

* **Apache Kafka:** Gestor 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. Esta tecnología es la que soporta el bus de datos y por ello es el servicio principal bajo supervisión.
* **Zookeeper:** Facilita la coordinación entre procesos distribuidos mediante un espacio de nombres jerárquico común de registros de datos, conocidos como znodes. Es un servicio necesario para el funcionamiento de la plataforma big data.
* **Apache Spark:** Es un framework de programación para procesamiento de datos distribuidos diseñado para ser rápido y de propósito general.
* **Cassandra:** Sistema de gestión de bases de datos NoSQL.

TODO: Imagen de la plataforma digital con sus tecnologías?

El caso de uso específico abordado se enfoca en la gestión de logs relacionados con la autenticación y autorización en Kafka. Este enfoque implica la supervisión y el análisis de los eventos de seguridad dentro de la plataforma, lo cual es crucial para garantizar un acceso controlado a los recursos y operaciones. La gestión adecuada de estos logs es esencial para detectar, investigar y mitigar posibles incidentes de seguridad, asegurando la integridad y confidencialidad de los datos gestionados por Kafka

## 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 como son:

* Splunk
* Graylog
* Fluentd
* Prometheus y Grafana
* Loki
* Datadog
* Sumo Logic
* Stack ELK
* Herramientas nativas de Google Cloud

Para el desarrollo del proyecto era necesario que la tecnología fuese open source y gratuita, por lo que se descartan opciones como Splunk. Además, sería recomendable que la tecnología empleada tuviese una gran comunidad de usuarios. También sería recomendable que su uso estuviera extendido para la función que vamos a desarrollar en el proyecto, es decir, gestión de logs. Además, se prefirió crear un sistema personalizado en lugar de implementar herramientas nativas de Google Cloud como Google Cloud Logging y sus agentes. Esta decisión se basa en varios factores clave como la flexibilidad y control del sistema, la personalización de logs que permiten Filebeat y Logstash, y la interoperabilidad que permite el stack ELK, evitando atarte con un solo proveedor de nube.

Teniendo en cuenta todos estos parámetros se ha decidido emplear el stack ELK para este proyecto.

TODO: REFERENCIAS A BIBLIOGRAFIA

## Resumen de los capítulos

**Capítulo 1: Introducción**

Este capítulo introduce la temática del proyecto, destacando la importancia de la gestión de logs en sistemas distribuidos. Se detallan los objetivos principales del proyecto y se describe brevemente la arquitectura y el caso de uso del sistema, que se centra en la gestión de logs de autenticación y autorización de Kafka. Finalmente, se justifica la elección de las tecnologías para el análisis y gestión de logs y se ofrece un resumen de los capítulos del documento.

**Capítulo 2: Stack ELK**

Este capítulo explora el Stack ELK (Elasticsearch, Logstash, Kibana) y Filebeat, detallando el funcionamiento de cada herramienta.

**Capítulo 3: Despliegue de la plataforma y Stack ELK**

En este capítulo se describen los motivos por los que se despliega con Docker y se incluye la configuración del fichero .yaml para orquestar los contenedores Docker. Además, se detallan los logs de Kafka, sus formatos y localizaciones, y la configuración de autenticación y autorización en Kafka.

**Capítulo 4: Diseño de la solución**

Este capítulo organiza el diseño de la solución global para la gestión de logs. Describe qué logs se monitorizan y para qué objetivos, cómo se organizan en índices en Elasticsearch, y cómo se visualizan en Kibana.

**Capítulo 5: Configuración del Stack ELK**

Se profundiza en la configuración del Stack ELK para el caso de uso de autenticación y autorización de Kafka.

**Capítulo 6: Cuadro de mandos – Kibana**

Se explica el desarrollo del cuadro de mandos en Kibana, incluyendo las diferentes funcionalidades de Kibana.

**Capítulo 7: Limitaciones del despliegue local**

Este capítulo aborda las limitaciones del despliegue local, como problemas de almacenamiento y escalabilidad. Se presenta la solución local propuesta para mitigar estos problemas, detallando las estrategias implementadas.

**Capítulo 8: Explicación de la solución en la nube**

Se describe la transición a una solución en la nube utilizando Google Cloud, explicando cómo se implementa y se mejora la gestión de logs en un entorno escalable y flexible utilizando Kubernetes.

**Capítulo 9: Objetivos de desarrollo futuro**

Este apartado detalla los objetivos para el desarrollo futuro, como la integración de nuevas fuentes de logs.

**Capítulo 10: Bibliografía**

Se proporciona una lista de todas las fuentes y referencias utilizadas en la elaboración del TFG.

# 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, además de otras tecnologías secundarias como Filebeat.

Antes de explicar las herramientas, se muestra a continuación un esquema con la representación gráfica general de las tecnologías utilizadas para el análisis de logs.

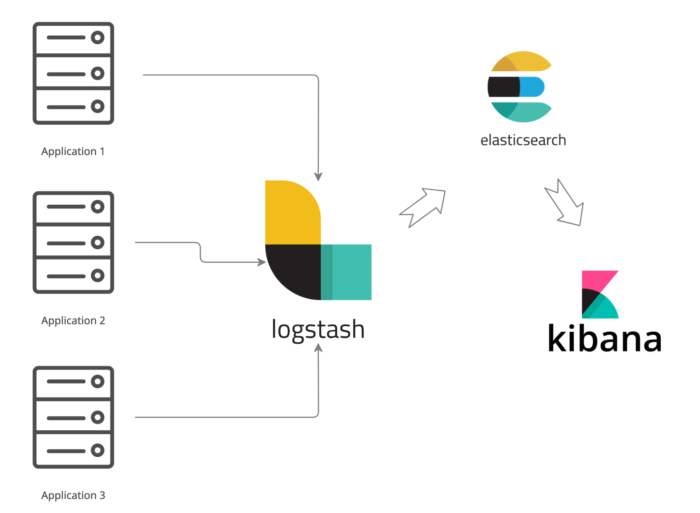


Imagen 1: Arquitectura general del sistema (sin Filebeat).

* **Filebeat:** Son agentes ligeros que se pueden instalar en los servicios para enviar directamente los logs a Elasticsearch o Logstash.
* **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.
* **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.

## 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 distintos servicios, organizándolos e incorporando metadatos como tipo, fecha, hora, host, y convertirlos en formato JSON para su almacenamiento en Elasticsearch.

Logstash además proporciona filtros para extraer campos específicos de los logs o realizar otras tareas de procesado.

En la imagen 2 se puede ver desde Kibana el formato estándar que Logstash aplica a los logs en crudo que recibe.A group of text boxes

Description automatically generated

Imagen 2: Formato de logs proporcionado por Logstash por defecto.

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

* + **type:** Indica el servicio del que proviene 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 donde se encuentra el fichero del cual se obtuvo el log.
  + **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 log.
  + **message:** Indica el mensaje principal del log, suele contener gran parte de este.
  + **\_id:** Este campo sirve para identificar cada log (documento json) dentro de cada colección de documentos de Elasticsearch, obviamente tiene un valor único.
  + **\_type:** Este campo ya no se utiliza (deprecated).
  + **\_index:** Nombre del índice de Elasticsearch donde se ha almacenado este log.
  + **\_score:** Indica como de importante es un documento en referencia a una consulta de búsqueda determinada. En este proyecto no se ha utilizado.

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 el formato de campos determinado por Filebeat.

## 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 log, se encuentra 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 búsquedas.

Una característica de Elasticsearch utilizada en este proyecto y propia de la tecnología NoSQL es que no es necesario definir la estructura del fichero JSON a indexar previamente (schemaless), es decir, puede 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 para distribuir los documentos y ser indexados en diferentes nodos de forma redundante para permitir consultas concurrentes 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 (JSON schema). Es útil en casos en los que se quieran restringir los campos de los documentos que formarán parte del índice. Permite definir tanto los campos que existirán como cuales de ellos contendrán texto, números, fechas.
  + **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 adecuado.

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.

## 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 las funciones principales, que se explicarán con un caso de uso más adelante.

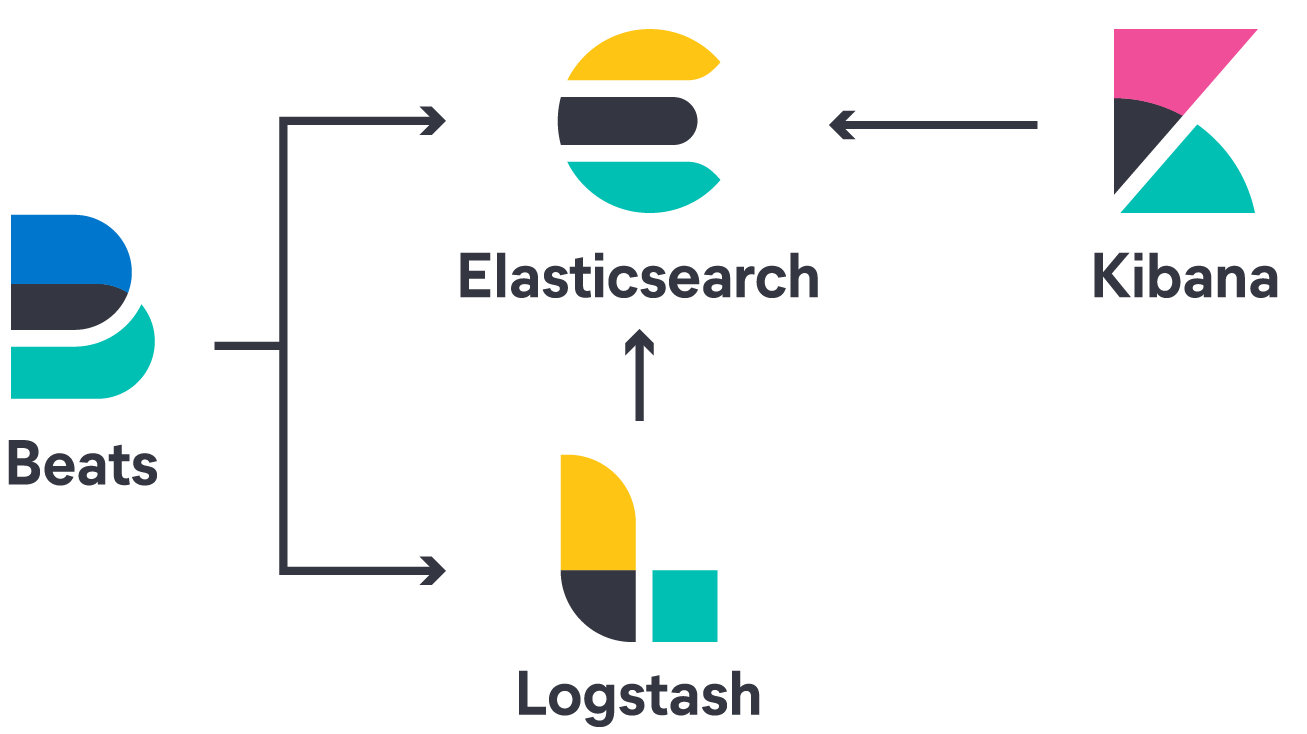
## Filebeat

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 *bind mounts* o volúmenes. Sin embargo, para darle un contexto más realista y poder utilizar una función de Kibana que necesita Filebeat se empleará este en el proyecto.

En la imagen 3 se muestra como quedaría la arquitectura de la solución para la gestión de logs usando los agentes (Beats) de Filebeat:

Imagen 3*: Arquitectura de la solución final usando Filebeat.*

# Despliegue de la plataforma y Stack ELK

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. Esta es la estrategia habitual en aplicaciones big data desplegadas en la nube.
  + **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.

## 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, tanto de la plataforma big data como de las herramientas del stack ELK.

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

## Logs 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.

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.

### 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 número 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 puede deducir, 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 autenticación).

**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 de Kafka

Como se ha mencionado previamente, se va a activar la autenticación y autorización de Kafka poder visualizar esta información en 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.

Se muestra a continuación la configuración de SASL\_PLAINTEXT en Kafka:

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"

Para definir las credenciales válidas es necesario crear un archivo como el que se muestra a continuación:

KafkaServer {

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

   username="admin"

   password="admin"

   user\_admin="admin"

   user\_usuario1="usuario1"

   user\_usuario2="usuario2";

};

En el cliente, para utilizar SASL\_PLAINTEXT, se referencia en los comandos de producción, consumición o configuración de Kafka a un fichero donde se definen las credenciales que se están usando, como el siguiente:

security.protocol=SASL\_PLAINTEXT

sasl.mechanism=PLAIN

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

De forma que para crear un tópico utilizando SASL\_PLAINTEXT se referencia al fichero con las credenciales de la siguiente manera:

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

Ya está configurada la autenticación de Kafka, ahora es necesario configurar la autorización, es decir, crear ACL’s para definir los permisos de acceso.

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

Necesitaremos generar logs por consola para probar el caso de uso, para ello se utilizarán los siguientes comandos:

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

# Diseño de la solución

En este apartado se describe el diseño global de la solución para la gestión de logs, enfocándose en el caso de uso de autenticación y autorización en Kafka. A continuación, se detalla la estrategia para la monitorización de logs, su organización en Elasticsearch y su visualización en Kibana.

El objetivo principal de la monitorización de logs en este proyecto es detectar y registrar todas las actividades de acceso no autorizado.

Para cumplir con el objetivo establecido, se han identificado y seleccionado los siguientes logs de Kafka:

* **Logs de autenticación:** Registran los intentos de conexión fallidos. Estos logs se encuentran en el fichero “server.log”. A continuación, se muestra el formato de estos:

[2024-04-22 15:23:34,795] INFO [SocketServer listenerType=ZK\_BROKER, nodeId=1001] Failed authentication with /172.18.0.1 (Unexpected Kafka request of type METADATA during SASL handshake.) (org.apache.kafka.common.network.Selector)

* **Logs de autorización:** Registran los intentos de acceso por parte de usuarios autenticados a recursos sobre los que no tienen permisos. Estos logs se encuentran en el fichero “kafka-authorizer.log”. A continuación, se muestra el formato de estos:

[2024-04-27 12:36:55,192] INFO Principal = User:usuario2 is Denied Operation = Create from host = 127.0.0.1 on resource = Topic:LITERAL:topico1 for request = CreateTopics with resourceRefCount = 1 (kafka.authorizer.logger)

Los logs se organizan en Elasticsearch en dos índices separados para optimizar su gestión y consulta, estos índices se crearán automáticamente de forma mensual.

Los campos clave de los logs de autenticación y autorización son los siguientes:

@timestamp Apr 22, 2024 @ 17:23:35.715

\_index logstash-kafka-autenticacion-2024.04

ip 172.18.0.1

log\_source kafka

log\_type autenticación

message

[2024-04-22 15:23:34,795] INFO [SocketServer listenerType=ZK\_BROKER, nodeId=1001] Failed authentication with /172.18.0.1 (Unexpected Kafka request of type METADATA during SASL handshake.) (org.apache.kafka.common.network.Selector)

@timestamp Apr 22, 2024 @ 17:23:35.715

\_index logstash-kafka-autorizacion-2024.04

log\_source kafka

log\_type autorizacion

message

[2024-04-22 15:23:34,795] INFO [SocketServer listenerType=ZK\_BROKER, nodeId=1001] Failed authentication with /172.18.0.1 (Unexpected Kafka request of type METADATA during SASL handshake.) (org.apache.kafka.common.network.Selector)

operation Create

request\_type CreateTopics

resource\_name topico1

resource\_type Topic

usuario usuario2

Para la visualización de la información en Kibana se creará un dashboard específico con visualizaciones para mostrar:

* + Número de fallos de autorización por usuario del sistema
  + Número de fallos de autenticación por ip
  + Fallos de autorización por usuario en el dominio del tiempo
  + Fallos de autenticación por ip en el dominio del tiempo
  + Tipos de operaciones de autorización denegadas
  + Tipos de peticiones de autorización denegadas
  + Tipo de recursos a los que se intenta acceder
  + Recursos específicos a los que se intenta acceder

Esta organización asegura una monitorización eficiente y una gestión estructurada de los logs, permitiendo una rápida identificación y resolución de problemas relacionados con la seguridad y el acceso en a recursos de Kafka.

# 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.

## Filebeat

Como se dijo previamente, se utiliza 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 muestra el 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"]

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 pueden surgir 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, por lo que hay que tener en cuenta el formato de campos de Filebeat al realizar el filtrado en Logstash.

Por ejemplo, usando 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.

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.

Por lo tanto, al utilizar Filebeat con la configuración anterior 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

},

#Campos añadidos manualmente

"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"

]

}

## 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 la configuración utilizada para recibir los logs de autenticación y autorización desde Filebeat, filtrarlos y enviarlos a su índice correspondiente en Elasticsearch.

input {

  beats {

    port => 5044

  }

}

filter {

  if [log\_type] == "autenticacion" {

    # Filtrar solo los mensajes de autenticación fallida

    if [message] =~ /Failed authentication/ {

      grok {

        match => { "message" => "Failed authentication with /%{IP:ip}" }

      }

    } else {

      drop { }

    }

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

    grok {

      match => {

        "message" => [

          "\[.\*\] INFO Principal = User:%{USERNAME:usuario} is Denied Operation = %{WORD:operation} from host = %{IP:host\_ip} on resource = %{WORD:resource\_type}:%{WORD:literal}:%{NOTSPACE:resource\_name} for request = %{WORD:request\_type}"

        ]

      }

    }

  }

}

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, que en este caso es la configuración por defecto.

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

### Explicación del plugin Grok con ejemplos

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 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. En este caso estamos tratando los logs de Kafka, más específicamente este log del fichero 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} \(%{JAVACLASS:java\_class}\)

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 sigue un patrón de FECHA-NIVEL-MENSAJE (aunque dentro del mensaje se podrían especificar más campos).

En la imagen 4 se ve la estructura de 5 de los 6 ficheros de logs de Kafka:

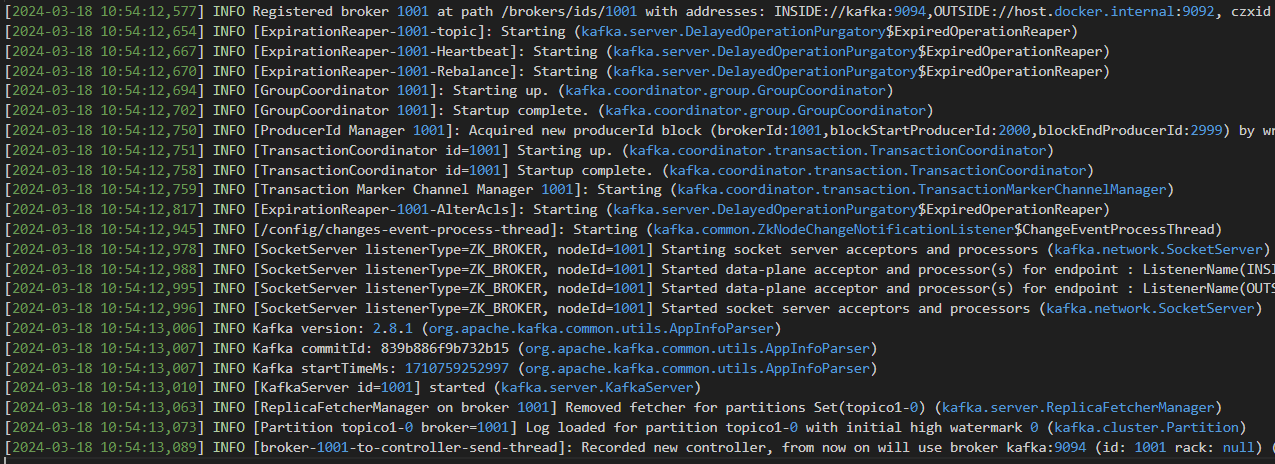


Imagen 4: Formato de logging de 5 ficheros de Kafka

En la imagen 5 se ve la estructura del único fichero de Kafka que no cumple la estructura anterior:

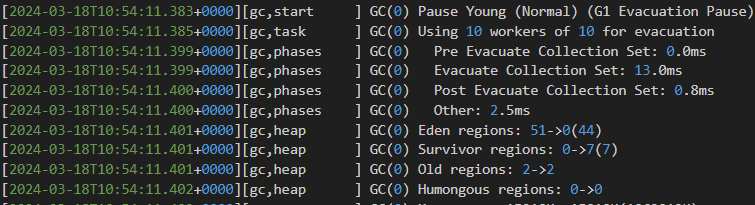


Imagen 5: Formato de logging de 1 fichero de Kafka

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 en la imagen 6, se ha generado el campo “loglevel”:

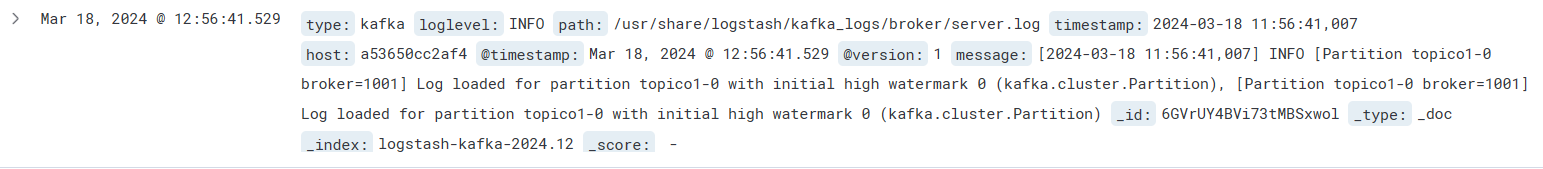


Imagen 6: Visualización de campos en Kibana para los 5 ficheros

Y los logs del fichero tratado de forma diferente quedan con los siguientes campos, podemos ver en la imagen 7 que se ha añadido el campo “gc\_tags”:

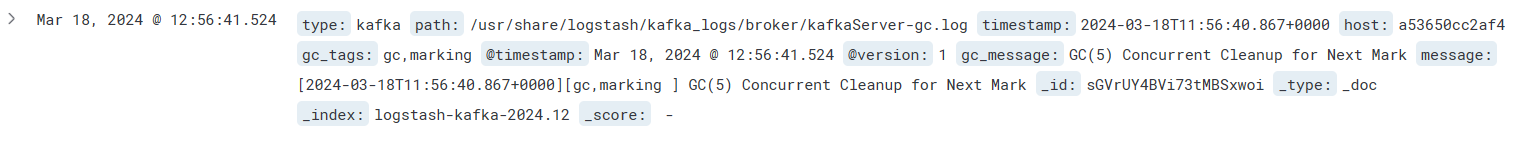


Imagen 7: Visualización de campos en Kibana para el fichero independiente

Como vemos en las imágenes 6 y 7, 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"]

}

Se ha explicado el funcionamiento del plugin Grok con diferentes logs de Kafka, aunque no sean los de autenticación y autorización, cuyos filtros se muestran en la configuración del stack ELK.

TODO: ¿Tiene sentido explicar el plugin Grok con logs de ejemplo que no son del caso de uso especifico de autenticación y autorización?

## Elasticsearch

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

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.

Elasticsearch permite configurar los campos que contendrán los logs de ese índice en concreto de forma estática, sin embargo, en este proyecto se usará el mapeo dinámico.

Con Elasticsearch desplegado y configurado, es importante monitorizar el estado del cluster. Este comando sirve para verificar el estado de salud del cluster de Elasticsearch:

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, es muy útil para solucionar errores:

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 imagen 8 con la salida al comando de consulta de los índices, de forma que se observe los índices creados por defecto por Elasticsearch.

A computer screen with white text

Description automatically generated

Imagen 8: Índices por defecto de Elasticsearch

Como se puede ver en la imagen 8, Elasticsearch por defecto tiene creados 6 índices con funciones predeterminadas:

* + **.kibana-event-log-7.9.3-000001**
  + **.apm-custom-link**
  + **.kibana\_task\_manager\_1**
  + **.apm-agent-configuration**
  + **.async-search**
  + **.kibana\_1**

# Cuadro de mandos - 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 las más relevantes.

## Observabilidad

El menú de observabilidad actúa como un panel de control general, proporcionando una visión resumida de métricas esenciales creadas por defecto por Kibana, como el volumen de logs recibidos. Para un análisis más detallado, por ejemplo, para examinar detalladamente los logs recibidos o para ver métricas más desglosadas acompañadas de representaciones gráficas, es necesario navegar a otras secciones más especializadas. Mientras que el menú de observabilidad es útil para monitorizar métricas generales, el menú de analytics es el que se utiliza para analizar los logs en profundidad, debido a que ofrece capacidades más avanzadas para este fin. A continuación, se muestra la imagen 9 con el menú de observabilidad.

A screenshot of a computer

Description automatically generated

Imagen 9: Menú principal de Observability de Kibana

Para hacer uso del menú de pbservabilidad es necesario utilizar Filebeat y, además, este menú nos permite añadir más visualizaciones en caso de estar utilizando otras herramientas como las que se ven en la imagen 10.

A screenshot of a computer screen

Description automatically generated

Imagen 10: Otras opciones del menú Observability de Kibana

## Análisis

Kibana tiene funcionalidades de análisis que permiten al usuario visualizar los logs de diferentes maneras. El menú principal de análisis es “Discover”, en este menú se muestra toda la información de los logs recibidos, y permite al usuario crear filtros a su gusto para modificar la visualización.

En este caso, a continuación, se mostrará en la imagen 11 el filtro de visualización para un contexto en el que se quiera ver los intentos fallidos de acceso (autorización) del usuario usuario2.

A screenshot of a computer

Description automatically generated

*Imagen 11: Filtrado errores de acceso del usuario2*

Este menú creado previamente se puede guardar con un nombre determinado para observarlo cuando sea necesario, esto puede ser útil para tener un panel con cada usuario, de forma que podemos vigilar su actividad y enterarnos en caso de que intenten acceder a algún tipo de información a la que no tienen acceso.

## Dashboards y Lens

Dashboard es una función que depende de Lens. Lens facilita la creación de gráficos utilizando datos extraídos de los logs, permitiendo elegir entre varios tipos de gráficos y aplicar filtros como hemos hecho previamente en análisis. Dashboard, por su parte, recopila estos gráficos creados con Lens y los presenta en una interfaz unificada para su visualización. Esta capacidad de visualizar datos gráficamente es fundamental para detectar problemas de manera rápida y eficiente.

A continuación, se muestra en la imagen 12 como ejemplo como crear un gráfico de líneas para visualizar por cada usuario, cuantos intentos de acceso denegados ha realizado diariamente.

A screenshot of a computer

Description automatically generated

Imagen 12: Creación de una visualización con Lens

Tras crear las visualizaciones con Lens, podemos añadir estas a un dashboard. A continuación, se muestra en la imagen 13 el dashboard con las visualizaciones requeridas.

A screenshot of a computer

Description automatically generated

Imagen 13: Dashboard para la autenticación y autorización de Kafka

A continuación, se mostrarán de forma individual las visualizaciones creadas y especificadas en el diseño de la solución:

* + Número de fallos de autorización por usuario del sistema

A screenshot of a graph

Description automatically generated

*Imagen 14: Número de fallos de autorización por usuario*

* + Número de fallos de autenticación por ip

A screenshot of a graph

Description automatically generated

Imagen 15: Número de fallos de autenticación por ip

* + Fallos de autorización por usuario en el dominio del tiempo

A graph with blue and green lines

Description automatically generated

Imagen 16: Fallos de autorización por usuario en el dominio del tiempo

* + Fallos de autenticación por ip en el dominio del tiempo

A white rectangular object with blue and green dots

Description automatically generated

Imagen 17: Fallos de autenticación por ip en el dominio del tiempo

* + Tipos de operaciones de autorización denegadas

A pie chart with numbers and text

Description automatically generated

Imagen 18: Tipos de operaciones de autorización denegadas

* + Tipos de peticiones de autorización denegadas

A screenshot of a computer screen

Description automatically generated

Imagen 19: Tipos de peticiones de autorización denegadas

* + Tipo de recursos a los que se intenta acceder

A pie chart with numbers and a few percentages

Description automatically generated

Imagen 20: Tipo de recursos a los que se intenta acceder

* + Recursos específicos a los que se intenta acceder

A graph with numbers and a bar

Description automatically generated with medium confidence

Imagen 21: Recursos específicos a los que se intenta acceder

Además, es un dashboard interactivo que permite filtrar sobre él, de forma que todas las visualizaciones del dashboard se adaptan a ese filtro.

En la imagen 22 se muestra el dashboard filtrado únicamente por la actividad del usuario usuario1.

A screenshot of a computer

Description automatically generated

Imagen 22: Dashboard filtrado por la actividad del usuario usuario1

# 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.

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, es necesario liberar espacio en disco y posteriormente quitar el bloqueo de solo lectura del incide a través del 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 monitorizar estos errores puede ser útil ver el uso de memoria que está haciendo Elasticsearch con el siguiente comando:

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

O ver la asignación de disco a cada nodo de Elasticsearch:

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

En caso de tener este problema, la salida de este último comando sería 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 eliminar el contenedor y volumen periódicamente, aumentar la asignación de disco destinada a Elasticsearch o aplicar políticas ILM, que permiten eliminar colecciones de logs cuando lleguen a una determinada capacidad de almacenamiento o según su antigüedad.

# 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.

Para llevar a cabo el despliegue en la nube, se hará una transición de la infraestructura local basada en Docker y ELK Stack a una solución más escalable y gestionada utilizando Kubernetes.

TODO: Pendiente

NOTAS:

- Crear proyecto

- Habilitar “Kubernetes Engine API”

- Crear un Cluster (privado) de Kubernetes

- Instalar chocolatey

- Instalar helm

- Utiliza Helm para desplegar Kafka y ELK en el cluster.

# 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/)