



ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

GRADO EN INGENIERÍA INFORMÁTICA EN TECNOLOGÍAS DE LA INFORMACIÓN

Lenguajes y Sistemas Informáticos

TRABAJO FIN DE GRADO/MÁSTER N° ???

**Explotación, integración y visualización de múltiples fuentes de datos
mediante un Data Lake**

Mier Montoto, Juan Francisco

TUTORES:

**D. Augusto Alonso, Cristian
D. Morán Barbón, Jesús
D. Vázquez Faes, Eduardo**

FECHA: julio 2024

Índice de contenido

| | |
|---|-----------|
| Índice de contenido | 1 |
| Índice de figuras | 3 |
| Índice de tablas | 4 |
| Índice de listados | 5 |
| 1. Introducción | 6 |
| 1.1. Antecedentes | 6 |
| 1.2. Motivación | 8 |
| 1.3. La empresa | 9 |
| 1.4. Objetivos | 10 |
| 2. Fundamento teórico | 11 |
| 2.1. <i>Big data</i> | 11 |
| 2.2. Paradigmas de almacenamiento de datos | 13 |
| 2.2.1. Data warehouse | 13 |
| 2.2.2. Data lake | 13 |
| 2.2.3. Data lakehouse | 14 |
| 2.3. Procesos ETL | 15 |
| 2.3.1. Funcionamiento | 16 |
| 2.3.2. Alternativas | 18 |
| 2.4. Cuadros de mandos (<i>dashboards</i>) | 19 |
| 2.5. Infraestructura como código | 20 |
| 3. Descripción general del proyecto | 21 |
| 3.1. Partes interesadas (<i>stakeholders</i>) | 21 |
| 3.2. Alternativas existentes | 23 |
| 3.2.1. Criterios de evaluación | 23 |
| 3.2.2. Proceso de selección | 23 |
| 3.2.3. Alternativas consideradas | 24 |
| 3.2.4. Conclusiones | 30 |
| 3.3. Descripción del proyecto | 31 |
| 3.3.1. Dashboards planteados | 31 |
| 4. Planificación del proyecto | 32 |
| 4.1. Metodología | 32 |
| 4.1.1. Scrum | 33 |
| 4.1.2. Visualización | 35 |
| 4.1.3. Comunicación | 36 |
| 4.1.4. Herramientas | 36 |
| 4.2. Planificación inicial | 37 |
| 4.3. Presupuesto | 39 |
| 4.3.1. Presupuesto de material | 39 |
| 4.3.2. Presupuesto de personal | 41 |

| | |
|--|-----------|
| 4.3.3. Presupuesto total | 41 |
| 5. Diseño del sistema | 42 |
| 5.1. Estudio de alternativas | 42 |
| 5.1.1. Despliegue de infraestructura | 43 |
| 5.1.2. Ingesta de datos | 45 |
| 5.1.3. Proveedor de nube | 48 |
| 5.1.4. Sistemas de ejecución y servicios | 49 |
| 5.1.5. Tipos de máquinas | 50 |
| 5.2. Arquitectura del sistema | 51 |
| 5.2.1. Infraestructura | 53 |
| 5.2.2. Seguridad | 54 |
| 5.2.3. Redes | 55 |
| 5.3. Modelos de datos | 56 |
| 6. Implementación | 58 |
| 6.1. Despliegue local | 58 |
| 6.1.1. Explicación del código | 60 |
| 6.1.2. Uso del sistema | 67 |
| 6.1.3. Proceso de desarrollo | 69 |
| 6.2. Despliegue <i>cloud</i> | 70 |
| 6.2.1. Proceso de desarrollo | 70 |
| 6.2.2. Despliegue de la infraestructura | 72 |
| 6.2.3. Explicación del código | 73 |
| 6.3. Ingesta de datos | 74 |
| 6.4. Visualización de datos | 75 |
| 7. Manuales | 76 |
| 7.1. Manual de usuario | 76 |
| 7.2. Manual de despliegue | 77 |
| 7.2.1. Requisitos previos | 77 |
| 7.2.2. Despliegue | 79 |
| 8. Resultados y trabajo futuro | 81 |
| A. Código de despliegue local | 82 |
| B. Script de creación de certificados | 88 |
| C. Scripts de despliegue cloud | 89 |
| Bibliografía | 90 |

Índice de figuras

| | |
|---|----|
| 2.1. Fases de un proceso ETL | 16 |
| 2.2. Ejemplo de flujo con virtualización | 18 |
| 2.3. Diagrama de flujo de un proceso <i>ELT</i> | 18 |
| 3.1. Logo de Elasticsearch ® | 24 |
| 3.2. Logo de Logstash ® | 24 |
| 3.3. Logo de Kibana ® | 24 |
| 3.4. Logo de Fluentd ® | 25 |
| 3.5. Logo de Graylog ® | 26 |
| 3.6. Logo de Prometheus ® | 26 |
| 3.7. Logo de Loki ® | 27 |
| 3.8. Logo de Loggly ® | 29 |
| 3.9. Logo de Splunk ® | 29 |
| 4.1. Diagrama de la metodología <i>Scrum</i> | 33 |
| 4.2. Tablero <i>Kanban</i> del proyecto | 35 |
| 4.3. Roadmap de apartados de la memoria | 36 |
| 4.4. Planificación inicial del proyecto | 37 |
| 5.1. Logo de Terraform ® | 43 |
| 5.2. Logo de AWS CloudFormation ® | 43 |
| 5.3. Logo de Ansible ® | 44 |
| 5.4. Logo de Kafka ® | 45 |
| 5.5. Logo de Amazon EC2 ® | 50 |
| 5.6. Logo de AWS Fargate ® | 50 |
| 5.7. Diagrama inicial de la infraestructura en AWS | 53 |
| 5.8. Diagrama incial de la seguridad en AWS | 54 |
| 5.9. Diagrama incial de las redes en AWS | 55 |
| 5.10. Modelo de datos de los logs de un balanceador de carga de AWS | 56 |
| 5.11. Modelo de datos de los logs de una base de datos SQL | 57 |
| 5.12. Modelo de datos de los logs de una base de datos MongoDB | 57 |
| 6.1. Inicio de sesión en Kibana | 67 |
| 6.2. Página de inicio de Kibana | 68 |
| 6.3. Ejemplo de commits en el repositorio privado | 69 |
| 6.4. Ejemplo de definciones de tareas de ECS en AWS | 71 |
| 6.5. Ejemplo de definción de tarea (Kafka) | 71 |
| 7.1. Comprobación de la versión de Terraform | 77 |
| 7.2. Configuración de credenciales en AWS CLI | 78 |
| 7.3. Demostración de credenciales en AWS CLI | 78 |
| 7.4. Despliegue de la infraestructura base | 79 |
| 7.5. Despliegue exitoso de la infraestructura base | 80 |

Índice de tablas

| | | |
|------|--|----|
| 4.1. | Historias de usuario iniciales | 38 |
| 4.2. | Propuesta de presupuesto de materiales | 40 |
| 4.3. | Propuesta de presupuesto de personal | 41 |
| 4.4. | Costes combinados de presupuesto y materiales con beneficio industrial . . | 41 |
| 6.1. | Lista de HUs cumplimentadas con el despliegue local | 59 |

Índice de listados

| | | |
|------|---|----|
| 6.1. | Definición de volúmenes y redes en Docker Compose | 60 |
| 6.2. | Definición del servicio de preparación | 60 |
| 6.3. | Definición de los servicios de Kafka | 62 |
| 6.4. | Definición del servicio de Elasticsearch | 63 |
| 6.5. | Definición de los servicios de Kibana | 64 |
| 6.6. | Definición de los servicios de Logstash | 65 |
| A.1. | Fichero de despliegue local | 82 |
| A.2. | Fichero de ejemplo de variables de entorno | 87 |
| B.1. | Script de creación de credenciales | 88 |

1. Introducción

El proyecto que se presenta en este documento tiene como objetivo la automatización de despliegue de la infraestructura y procesos que permitan hacer un análisis masivo de datos. Para conseguir este objetivo, se hace un análisis del contexto y se realiza un diseño para, posteriormente, implementar una solución que permita la integración, almacenamiento y análisis de grandes volúmenes de datos, provenientes de múltiples fuentes y en diferentes formatos.

1.1. Antecedentes

Hoy en día, el crecimiento de la cantidad de dispositivos conectados a Internet (teléfonos móviles, dispositivos *IoT*...) ha provocado un aumento exponencial de la cantidad de datos que se manejan¹, un hecho que se ve reflejado en el ámbito empresarial. Dicha cantidad de datos genera una necesidad de análisis y tratamiento que las tecnologías tradicionales de datos (bases de datos) no pueden suplir. La diversidad de fuentes y formatos de estos datos introduce una complejidad significativa en su manejo, conocida como *heterogeneidad*², siendo las bases de datos, archivos de registros y APIs las fuentes más habituales.

El término *big data* describe este fenómeno de acumulación masiva de datos, cuya magnitud y complejidad sobrepasan las capacidades de los métodos de procesamiento convencionales. El *big data* se caracteriza por tres características principales: volumen, variedad y velocidad - su adecuada gestión y análisis pueden otorgar ventajas competitivas significativas a las empresas, tales como el descubrimiento de patrones ocultos, identificación de nuevas oportunidades de mercado y optimización de procesos de toma de decisiones.

Uno de los procesos que permite la extracción de esta información es la pirámide DIKW, [1] es un modelo que describe la relación entre los datos, la información, el conocimiento y la sabiduría. Según este modelo, los datos son la materia prima de la información, que a su vez es la materia prima del conocimiento, que a su vez es la materia prima de la sabiduría. Una organización sin los procesos adecuados para la gestión y análisis de estos datos, se enfrenta a importantes desafíos, como la dificultad para identificar patrones y tendencias, la toma de decisiones incorrectas y la pérdida de oportunidades de negocio. Por otro lado, una organización que logre extraer información valiosa de sus datos, podrá mejorar su eficiencia, aumentar su competitividad y adaptarse mejor a un entorno empresarial en constante cambio.

¹<https://www.statista.com/statistics/871513/worldwide-data-created/>

²<https://www.sciencedirect.com/topics/computer-science/data-heterogeneity>

La evolución tecnológica ha propiciado el desarrollo de innovadoras herramientas y metodologías diseñadas para enfrentar estos desafíos. Entre ellas, los *data lakes* (o *lagos de información*) se destacan por su capacidad para consolidar vastos volúmenes de datos heterogéneos, facilitando su posterior análisis y aprovechamiento de manera más efectiva.

Sin embargo, a pesar de que existen herramientas de almacenamiento, el proceso de integración, visualización y análisis de estos datos es una tarea desafiante, ya que requiere de una gran cantidad de recursos y de un tiempo de desarrollo considerable del que, normalmente, no se dispone en el ámbito empresarial.

Con la ingesta masiva de datos, se presentan nuevos problemas a la hora de analizar y obtener información de ellos:

- **Grandes cantidades de información:** la masificación de información impide el análisis manual de los mismos, requiriendo resúmenes estadísticos o representaciones gráficas como *dashboards* para su correcta interpretación. La visualización de datos es una técnica que permite representar la información de manera visual, para facilitar su análisis y comprensión, una parte vital del proceso de análisis de datos, ya que permite identificar patrones, tendencias y anomalías en los mismos de forma más rápida y sencilla.
- **Heterogeneidad de los datos:** la heterogeneidad de los datos, tanto en formato como en origen, dificulta su consolidación y análisis, ya que requiere de un proceso de integración y transformación previo para homogeneizarlos y poder analizarlos de forma conjunta.
- **Decisiones de negocio erróneas debidas a un mal tratamiento:** sin la necesaria automatización y correcta aplicación de los procesos ETL (ver 2.3), al tratarse de un crecimiento exponencial de los datos y, por lo tanto, de la fuerza de trabajo necesaria para manejarla, los resultados del análisis pueden ser incorrectos, lo que deriva en errores y decisiones de negocio equivocadas que impactan negativamente en la empresa.

1.2. Motivación

Actualmente, las empresas (especialmente aquellas en el sector IT), se enfrentan a la necesidad de unificar, gestionar y analizar grandes volúmenes de datos, provenientes de múltiples fuentes y en diferentes formatos. La correcta gestión y análisis de estos datos es fundamental para la toma de decisiones y para la mejora de los procesos internos de la empresa.

En la actualidad, Okticket (en adelante la empresa) dispone de una gran cantidad de datos que se encuentran en diferentes formatos y en diferentes ubicaciones, lo que dificulta su análisis y explotación. Por otra parte, se depende de la consulta manual o de servicios de terceros para poder analizar estos datos, lo que supone un coste adicional.

El proyecto surge de la necesidad de la empresa de extraer información y conocimiento de las múltiples y heterogéneas fuentes de datos de las que se disponen, tanto internas (e.g. bases de datos, archivos de registros, APIs, entre otros), como externas (e.g. APIs o datos de webs de terceros, datos de fuentes públicas...).

Además del uso interno, la empresa también quiere ofrecer a sus clientes la posibilidad de consultar estos datos de forma visual y sencilla, para que puedan analizarlos y explotarlos de forma autónoma, lo que supondría un valor añadido para los mismos.

1.3. La empresa

Okticket es una startup nacida en Gijón en 2017 cuyo producto principal es un servicio software que escanea automáticamente tickets y facilita su gestión usando conceptos contables como notas de gastos, anticipos y más. Esto permite reducir los costes y el tiempo que invierten las empresas en contabilizar y manejar los gastos de viaje profesionales.

La empresa tienen su sede principal en el Parque Tecnológico de Gijón, aunque cuenta con un número de sedes creciente en varios países, como Francia, Portugal o, más recientemente, México. En esta oficina principal se encuentran los departamentos de ventas y marketing, así como el equipo de desarrollo y consultoría.

Okticket es una de las empresas que más crecen tanto del sector como del propio Parque Tecnológico. Debido a este rápido crecimiento, el equipo está en constante desarrollo y cambio, tanto aquí en España como en el resto de sedes. Este crecimiento se refleja en la recepción de un gran número de galardones y reconocimientos.^{3 4 5 6}

La parte principal del negocio es el núcleo del software como servicio (Software as a Service en inglés, en adelante *SaaS*), es decir, la aplicación completa tanto para administradores como para empleados. Este *SaaS* se oferta a empresas de cualquier tamaño, cuyo precio final varía en función del número de usuarios, las características e integraciones que requiera la empresa cliente y el soporte que se ofrezca.

Recientemente se han añadido nuevas propuestas a la cartera de servicios ofertada por Okticket, como la OKTCard - una tarjeta inteligente que gestiona automáticamente los gastos, así como la inclusión de nuevos “módulos” de gestión de gastos y viajes.

Debido al crecimiento acelerado de Okticket, la empresa maneja una gran cantidad de datos de diversos tipos y almacenados en diferentes silos (programas de gestión contable, ventas, consultoría, así como los datos que genera el *SaaS*), que deben ser unificados para poder ser analizados y explotados de forma eficiente. Por otra parte, actualmente se depende de la consulta manual o de servicios de terceros para poder analizar estos datos, lo que es costoso, tedioso y muy poco eficiente.

³Okticket en el especial startups 2023 de Forbes (LinkedIn)

⁴Arcelor y Okticket, premios nacional de Ingeniería Informática (EL COMERCIO)

⁵Okticket recibe el sello Pyme Innovadora (okticket.es)

⁶Okticket, empresa emergente certificada (okticket.es)

1.4. Objetivos

El objetivo del proyecto es la creación de un proceso que permita el despliegue automático de una infraestructura de datos para la integración, almacenamiento y análisis de grandes volúmenes de datos, provenientes de múltiples fuentes y en diferentes formatos. La infraestructura de datos debe ser escalable, flexible y robusta, para poder adaptarse a las necesidades cambiantes de la empresa. La integración de datos debe ser automática y programable, para poder automatizar el proceso de ingestión de datos y reducir el tiempo y los costes asociados.

El resultado final del proyecto será la plataforma en sí, es decir, la infraestructura automatizada que integre y almacene los datos. El entregable final será la colección de ficheros y *scripts* necesarios para el despliegue de la infraestructura y el tratamiento de la información.

2. Fundamento teórico

En este capítulo se presentan los conceptos y términos fundamentales que se utilizan en el proyecto para proporcionar una base teórica sobre la que se desarrolle. Se discuten los conceptos fundamentales.

2.1. *Big data*

El término *big data* se refiere a la gestión y análisis de grandes volúmenes de datos que no pueden ser tratados de manera convencional. La evolución natural del progreso tecnológico, la digitalización de la sociedad y la aparición de nuevas tecnologías han propiciado la generación de grandes cantidades de datos en todo el mundo, lo que genera la necesidad de nuevas formas de gestionar y tratar estos datos.

El término *big data* no solo se refiere a la cantidad de datos que se generan, sino a también otras características, a las que se refieren como “las uves del big data”. La cantidad de *uves* depende del autor y de la fuente [2, 3], variando desde 3 hasta 7, pero las más comunes son las siguientes:

- **Volumen:** la cantidad de datos que se generan y almacenan en un determinado periodo de tiempo. El volumen de datos que se maneja en el *big data* es mucho mayor que el que se maneja en los sistemas tradicionales de gestión de datos, que además se encuentra en aumento constante.
- **Variedad:** se refiere a la diversidad de fuentes y formatos de los datos que se manejan. Los datos pueden provenir de diversas fuentes, como bases de datos, sensores o registros, pueden estar en diferentes formatos o tener diferentes estructura. Se pueden clasificar de la siguiente manera:
 - **Estructurados:** datos que se encuentran en un formato estructurado, como una base de datos relacional.
 - **Semi-estructurados:** datos que no se encuentran en un formato estructurado, pero que tienen una estructura interna que permite su análisis, como un archivo XML o JSON.
 - **No estructurados:** datos que no tienen una estructura definida, como un archivo de texto o una imagen.

- **Velocidad:** se refiere a la frecuencia con la que se generan y se procesan los datos. En este ámbito, los datos se tratan a una velocidad mucho mayor que en los sistemas tradicionales de gestión de datos. Dicha velocidad puede ser crítica en ámbitos como la bolsa, donde la velocidad de procesamiento de los datos puede ser la diferencia entre obtener beneficios o pérdidas. Según la frecuencia de procesamiento de los datos, los sistemas se pueden clasificar en:
 - **Batch (en lotes):** los datos se procesan en lotes, de manera periódica, como cada hora o cada día. Este tipo de datos, frecuente en aplicaciones que se tratan en Okticket como las notas de viajes o las nóminas, no requieren un procesamiento inmediato.
 - **Streaming (en tiempo real):** los datos se procesan en tiempo real, a medida que se generan. Este tipo de datos, que se puede encontrar en aplicaciones como los sensores o los logs, requieren un procesamiento inmediato si se quiere obtener información relevante sobre los mismos.
 - **Near real-time (casi en tiempo real):** los datos se procesan con un pequeño retraso, de manera que se obtiene información relevante sobre los mismos en un tiempo muy corto.
- **Veracidad:** se refiere a la calidad de los datos que se manejan. La veracidad de los datos es un factor crítico en el ámbito del *big data*, pues la calidad de la salida depende directamente de la calidad de la información de entrada. La veracidad de los datos se puede ver afectada por diferentes factores, como la calidad de los datos, la precisión de los mismos, la integridad de los datos, etc.

2.2. Paradigmas de almacenamiento de datos

En el ámbito del *big data*, existen diferentes paradigmas de almacenamiento de datos que se utilizan para almacenar y analizar grandes cantidades de información. Los tres paradigmas a considerar para este proyecto son los *data warehouses*, los *data lakes* y los *data lakehouses*.

2.2.1. Data warehouse

Un *data warehouse*¹, también conocido en español como almacén de datos, es una base de datos que se utiliza para almacenar y analizar grandes cantidades de datos de manera eficiente. Los almacenes de datos proporcionan acceso rápido y compatible con plataformas de consultas (como SQL) a grandes cantidades de datos, lo que permite a los analistas y a los científicos de datos realizar análisis complejos sobre los datos almacenados.

Todos los datos almacenados en un *data warehouse* se encuentran en un formato común, para lo que se aplican procesos ETL (extracción, transformación y carga) que transforman los datos de diferentes fuentes en un formato común. Esto significa que la información se encuentra en un formato o esquema optimizado y específico, lo que facilita su manipulación y análisis pero limita la flexibilidad al acceso de los datos y genera costes adicionales en el caso de tener que modificar o transferir los mismos para su uso.

2.2.2. Data lake

Los *data lakes*² son almacenes de datos que guardan grandes cantidades de datos de manera no estructurada [4]. En el ámbito de una empresa, un *data lake* contiene datos de diferentes fuentes de valor no considerado hasta su análisis, de manera que su explotación posterior y su análisis no depende de una estructuración y transformación compleja, reduciendo los costes de los procesos ETL derivados, una flujo de tareas que se aplican sobre la información para ingestarla. Esto no quiere decir que no se apliquen estos procesos a los datos, sino que se aplican de manera más flexible y básica que en otras estructuras de almacenamiento de datos con esquemas predefinidos, como los *data warehouses*. [5]

A diferencia de los *data warehouses*, los *data lakes* no tienen un esquema definido, lo que permite almacenar datos *heterogéneos*. Esto permite almacenar grandes cantidades de información sin tener que definir un esquema de antemano, lo que puede ser útil en aquellos casos en los que no se conoce la estructura de los datos que se van a almacenar.

¹<https://aws.amazon.com/es/data-warehouse/>

²<https://aws.amazon.com/es/what-is/data-lake/>

Estas características de los *data lakes* hacen que sean más atractivos en el sector empresarial, puesto que implica la gestión de un solo *stack* tecnológico que contiene toda la información, en contraste con las estructuras planteadas normalmente en el campo de la investigación académica.

Para consultar esta gran cantidad de datos almacenados, se suelen utilizar técnicas de visualización de datos, como los *dashboards*, herramientas de visualización que permiten observar los datos de manera sencilla y eficiente.

2.2.3. Data lakehouse

Los *data lakehouses* son una combinación funcional de los dos paradigmas vistos anteriormente, los *data lakes* y los *data warehouses*. Los *data lakehouses* permiten almacenar datos tanto de manera estructurada como no estructurada, lo que facilita aprovechar la información al contar con una única estructura de bajo coste que ofrece a los usuarios que lo necesiten explorar y analizar los datos según sus necesidades.

2.3. Procesos ETL

Si anteriormente se presentaban los distintos paradigmas de almacenamiento de datos, para su creación y mantenimiento se requieren aplicar unos ciertos procesos que permitan la correcta ingestión y almacenamiento de los datos. Estos procesos se conocen como *procesos ETL*.

Formalmente se definen los procesos ETL [4] como procesos que combinan datos de múltiples fuentes en un único destino, transformando los datos en un formato común. Estos procesos se utilizan para extraer datos de diferentes fuentes, transformarlos en un formato común y cargarlos en un destino común, como puede ser un *data lake*.

Los procesos ETL, fundamentales en el ámbito de la gestión de datos, presentan atributos distintivos que facilitan la integración eficaz de información procedente de diversas fuentes:

- **Adaptabilidad:** los procesos ETL deben de adaptarse a la estructura de los datos de la fuente de origen, ya que dichas fuentes pueden tener diferentes estructuras y tener tipos de datos diferentes (la característica de *heterogeneidad* de los datos que ya se ha mencionado).
- **Escalabilidad:** otra de las características clave de los procesos ETL es que sean escalables, ya que los datos que se muestran en los dashboards suelen ser datos que se generan de manera continua, y por lo tanto los procesos ETL deben ser capaces de procesar grandes cantidades de datos de manera eficiente. En ocasiones, los procesos ETL se pueden realizar en *streaming*, lo que significa que los datos se procesan en tiempo real a medida que se generan.
- **Eficiencia:** los procesos ETL deben ser eficientes, puesto que el tiempo de procesamiento de los datos es un factor vital en el ámbito del *big data*. Los procesos ETL deben ser capaces de procesar grandes cantidades de datos en un tiempo razonable para que los datos estén disponibles en el menor tiempo posible.
- **Fiabilidad:** la fiabilidad es un componente crítico de todo el flujo de datos, ya que estos se utilizan para la toma de decisiones importantes de cualquier empresa. Los procesos ETL deben ser capaces de procesar los datos de manera fiable y consistente, para que los datos que se visualicen y analicen posteriormente sean correctos y fiables.

2.3.1. Funcionamiento

Los procesos ETL se dividen en tres fases principales: (1) *Extraer*, (2) *Transformar* y (3) *Cargar*, como se muestra en el siguiente diagrama:

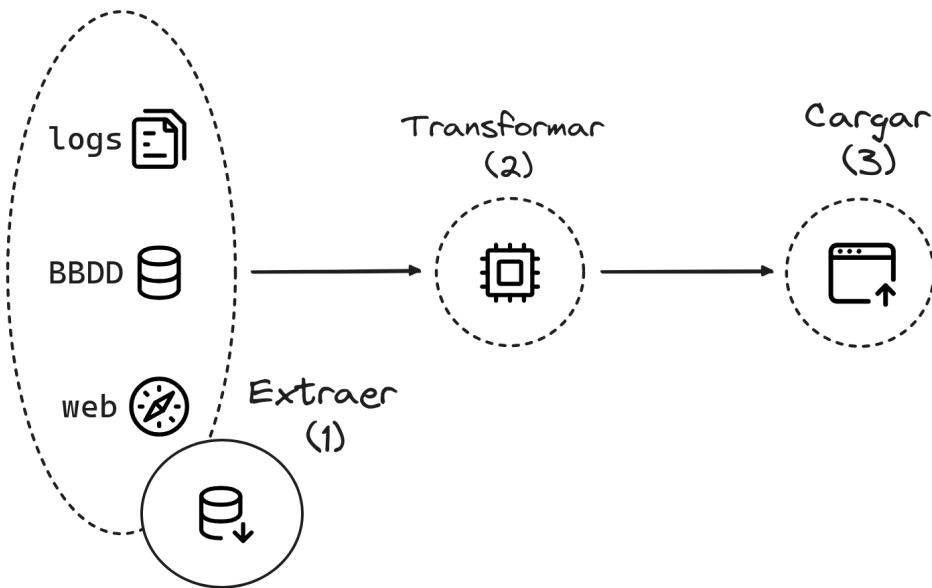


Figura 2.1: Fases de un proceso ETL

Como entrada, se tienen datos presuntamente heterogéneos que no se pueden analizar de manera eficiente. Tras aplicar todos los pasos de las fases anteriores, se obtiene como salida un conjunto de datos corregidos y listos para ser analizados en el destino indicado, sea cual sea el paradigma de almacenamiento de datos elegido.

Extracción (1) En este proceso se obtienen los datos de las fuentes de datos, que pueden ser bases de datos, logs, APIs, etc. En esta fase, se pueden aplicar filtros para extraer solo los datos que se necesiten, y se pueden extraer datos de múltiples fuentes *heterogéneas*.

La fase de extracción se puede realizar de dos formas: continua o incremental. Una extracción incremental se realiza de manera periódica, por ejemplo, cada hora, cada día o cada semana, y se extraen los datos que se han generado desde la última extracción. Esto es útil cuando los datos se generan de manera periódica y se necesita mantener actualizada la información. Por otro lado, en una extracción continua se extraen los datos en tiempo real según se van generando. Esto puede ser útil para procesar datos que se generan en tiempo real, como logs o datos de sensores.

Transformación (2) Durante esta fase, se transforman los datos extraídos en la fase anterior, normalmente aplicándoles un proceso de limpieza y transformación a un formato común. En este paso, se pueden aplicar diferentes operaciones a los datos, como la limpieza, la agregación, la normalización, la conversión de formatos, etc.

Uno de los tipos de transformaciones de datos más comunes es la limpieza, que consiste en la revisión y corrección de los datos extraídos, para asegurar que se almacena información correcta y consistente. Durante esta fase se contemplan operaciones más complejas, como pueden ser la agregación de datos, la conversión de formatos, la normalización de datos, el cifrado, etc. La limpieza de datos puede ser una tarea muy sencilla, como la eliminación de caracteres delimitadores, o muy compleja, como la corrección de errores en los datos, la detección de duplicados o la minimización [6] y/o compresión [7] de datos (eliminación de información no relevante o redundante).

Estos procesos de transformación son vitales cuando el sistema maneja una gran cantidad de datos heterogéneos de múltiples fuentes de manera simultánea, como puede ser el caso de un *data lake* o un *data warehouse*. En el caso del primero, no es necesaria la transformación de los datos a un formato común, pero si otros procesos clave como la limpieza y la normalización de los datos, entre otros.

Carga (3) En este proceso se vuelcan los datos transformados en el destino final. Frecuentemente, los datos se almacenan, dependiendo del paradigma de almacenamiento elegido, en una *data lake*, *data warehouse* o *data lakehouse* para su posterior análisis.

Las características de la carga de datos varían dependiendo de la arquitectura de datos que se esté utilizando. Por ejemplo, en ciertos sistemas puede ser necesario cifrar los datos antes de cargarlos en el destino, o puede ser necesario realizar una carga incremental para mantener actualizada la información en el destino. En otros casos, puede ser necesario realizar una carga masiva para cargar grandes cantidades de datos en el destino de una sola vez. La periodicidad de la carga es una característica clave del *big data*, como se ha mencionado anteriormente (ver 2.1 *Big data*).

2.3.2. Alternativas

Aunque lo más común es el flujo anteriormente explicado de *extracción, transformación y carga*, existen algunos flujos alternativos que son útiles para ciertos procesos diferentes:

- **Virtualización de datos:** capa virtual de abstracción que permite acceder a los datos de las fuentes sin necesidad de extraerlos. Esto permite ahorrar espacio de almacenamiento y tiempo de procesamiento, pero suele ser menos eficiente en términos de rendimiento y no es compatible con todas las arquitecturas de datos.

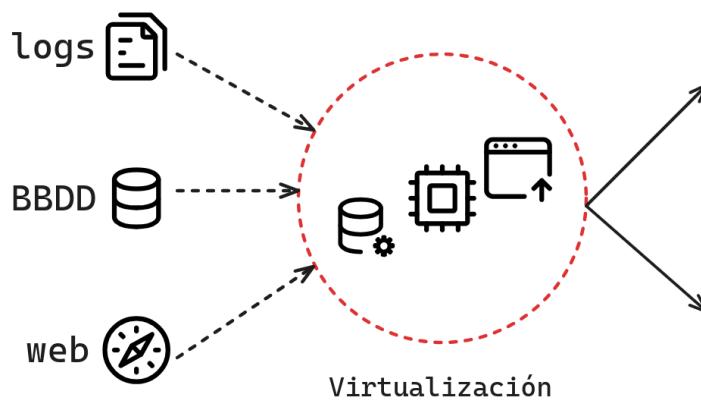


Figura 2.2: Ejemplo de flujo con virtualización

- **Proceso ELT³:** en lugar de transformar los datos antes de cargarlos en el destino, se cargan los datos en bruto y se transforman en el destino. Funciona bien para grandes conjuntos de datos sin estructura que requieran una carga (o recarga) continua, aunque, al igual que la virtualización, puede ser menos eficiente o incompatible con algunas arquitecturas de datos, como los *data warehouses*.

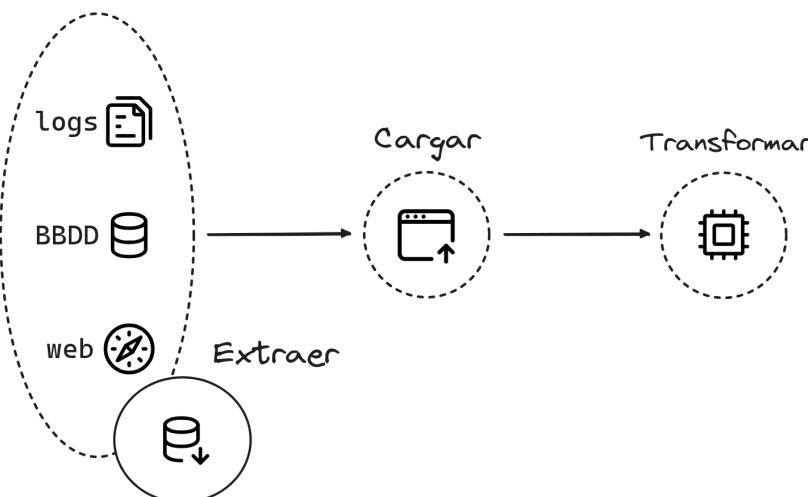


Figura 2.3: Diagrama de flujo de un proceso ELT

³<https://www.ibm.com/topics/elt>

2.4. Cuadros de mandos (*dashboards*)

Definición Los cuadros de mandos, en adelante *dashboards*, son soluciones en forma de interfaz gráfica que muestran información relevante de manera visual sobre un proceso o negocio. Aunque el término se utiliza en muchos ámbitos (indicadores comerciales, de producción, de marketing, de calidad, de recursos humanos...) en este proyecto se utilizará en el ámbito de la monitorización de sistemas y procesos de negocio.

En el ámbito de este proyecto, los dashboards reflejan en tiempo real el rendimiento de actividades o procesos de negocio, y se utilizan para tomar decisiones informadas basándose en los mismos. Por ejemplo, el dashboard de una empresa digital puede mostrar desde el rendimiento de la arquitectura en tiempo real hasta el número de ventas conseguidas, y permitir a los directivos tomar decisiones informadas sobre el futuro de la empresa (e.g. necesidad de aumentar la capacidad de los servidores, lanzar una campaña de marketing, etc.).

Características Los dashboards cuentan con una serie de características que los hacen útiles para la toma de decisiones: [4]

- **Visualización de datos:** es la característica fundamental de cualquier dashboard, y aquella que determina su utilidad. La visualización de datos es la ciencia de presentar los datos de manera que se pueda extraer información útil y realizar decisiones informadas sobre ellos. Un buen dashboard cuenta con gráficas, tablas, indicadores, etc. que permiten al usuario entender la información que se está presentando con un conocimiento técnico mínimo.
- **Interactividad y personalización:** un dashboard debe permitir al usuario interactuar con los datos (filtrarlos, ordenarlos, profundizar en ellos...) y ajustar la información que se muestra sobre cada proceso o negocio que se esté evaluando (granularidad de la información). Esta capacidad asegura que el dashboard se adapte tanto a las necesidades actuales como a las evoluciones futuras de lo que se esté analizando.
- **Accesibilidad y portabilidad:** un dashboard debe ser accesible desde una variedad de situaciones y dispositivos, manteniendo su funcionalidad y forma. Aunque normalmente los dashboards se analizan en pantallas grandes, es importante que también se puedan consultar en otras circunstancias, como dispositivos móviles.

2.5. Infraestructura como código

La infraestructura como código (o *IaC* por sus siglas en inglés) es una práctica que consiste en gestionar la infraestructura de un sistema de manera automática y programática mediante código, en lugar de configuraciones manuales.

La infraestructura como código permite gestionar la arquitectura global de un sistema de manera eficiente y escalable, y facilita la creación y el mantenimiento de entornos de desarrollo y producción, lo que elimina la necesidad de realizar tareas repetitivas [8] (*TOIL* por sus siglas en inglés), reduce la posibilidad de errores humanos y favorece la replicabilidad de los procesos.

En el ámbito de este proyecto, la infraestructura como código se utilizará para gestionar el despliegue y orquestación de los servicios requeridos para el paradigma de almacenamiento que se escoja, como los servicios de ingestión o de visualización de datos.

3. Descripción general del proyecto

Esta sección describe el proyecto en términos generales, incluyendo una descripción de los problemas que se pretenden resolver, las partes interesadas en el proyecto y una valoración de las alternativas consideradas.

3.1. Partes interesadas (*stakeholders*)

Las partes interesadas en el proyecto son aquellas personas o entidades que tienen un interés en el mismo, ya sea porque se ven afectadas por el resultado del proyecto, o porque tienen algún tipo de interés en el mismo. Las partes interesadas en este proyecto son las siguientes:

1. **Okticket:** la empresa es la principal parte interesada en el proyecto, ya que es la que se beneficiará directamente de los resultados del mismo, así como de las oportunidades de negocio que se abren con la explotación de los datos. Dentro de la empresa, se pueden identificar varias entidades afectadas:
 - **Equipo de desarrollo:** son los encargados de llevar a cabo la implementación del sistema y de garantizar su correcto funcionamiento, además de gestionar el soporte de servicio a nivel técnico.
 - **Equipo de soporte:** el sistema planteado ahorraría tiempo al equipo de soporte, ya que les permitiría analizar los datos de forma más eficiente e identificar problemas antes de que tener que resolver las peticiones de los clientes afectados a nivel básico.
 - **Equipo de negocio:** afectado porque permitirá tomar mejores decisiones estratégicas, definir y seguir métricas de seguimiento de los objetivos.
 - **Equipo de éxito de clientes:** afectado porque le permitirá evaluar mejor a qué clientes dar seguimiento, mejorar el seguimiento, identificar oportunidades de *upselling* ...
 - **Equipo de operaciones:** afectado porque les permitirá hacer un mejor seguimiento de las operaciones y procesos de la empresa.
2. **Clientes:** se beneficiarán de los nuevos servicios que se ofrecen, como los dashboards de negocio que se han descrito anteriormente. Estos clientes no son necesariamente los usuarios finales, sino los administradores y gestores de las empresas que utilizan Okticket como herramienta de gestión de gastos.

3. **Investigador y desarrollador (*Mier Montoto, Juan Francisco*):** el desarrollador del proyecto tiene la oportunidad de aplicar los conocimientos adquiridos en el desarrollo de un proyecto real, y de adquirir nuevos conocimientos en el proceso.

3.2. Alternativas existentes

Antes de comenzar la planificación y el desarrollo del proyecto, se consideran varias opciones ya existentes en el mercado que podrían resolver o adaptarse a las necesidades planteadas.

3.2.1. Criterios de evaluación

Los puntos claves a considerar de cada alternativa son los siguientes:

- **Coste:** se dará prioridad a alternativas gratuitas, siempre que su coste de operación y mantenimiento no sea excesivo.
- **Complejidad:** se priorizarán aquellas alternativas que sean sencillas y no requieran una curva de aprendizaje excesiva para su implementación y uso.
- **Rendimiento y escalabilidad:** a la hora de manejar los volúmenes de datos con los que se están tratando, el sistema debe responder positivamente tanto a la ingestión, tratamiento y visualización como a la escalabilidad del mismo.
- **Licencias:** se priorizará el software de código abierto con el objetivo de reducir el *vendor lock-in* y permitir una mayor flexibilidad y personalización del sistema sin coste de uso.

3.2.2. Proceso de selección

Para seleccionar las alternativas a considerar, se realiza un análisis de las herramientas y tecnologías más populares en el mercado para la gestión de datos, centrándose en aquellas que ofrecen capacidades de recopilación, almacenamiento, búsqueda y visualización de logs y métricas.^{1 2 3}

A la hora de seleccionar las alternativas, se tienen en cuenta los criterios de evaluación establecidos, además de preseleccionar el *stack ELK* como opción preferente, ya que es la sugerida por la empresa.

¹https://old.reddit.com/r/sysadmin/comments/v8oikw/elk_stack_or_an_alternative_in_2022/

²<https://www.perplexity.ai/search/elk-stack-alternatives-.33EfV0mR0qACcdj3B0J5w>

³https://www.reddit.com/r/selfhosted/comments/cgpyi9/selfhosted_lightweight_alternative_to_elk_stack/

3.2.3. Alternativas consideradas

Para este proyecto, el análisis de alternativas se realiza en torno a las herramientas y tecnologías disponibles en el mercado para la gestión de datos en base a los criterios establecidos anteriormente. A continuación, se describen las alternativas consideradas destacando puntos a favor, en contra y unas breves conclusiones, además de una valoración final de las mismas.

3.2.3.1. Elasticsearch, Logstash, Kibana (ELK)

La pila ELK es una solución de código abierto que combina tres herramientas diferentes: *Elasticsearch*, *Logstash* y *Kibana*. Este *stack* tecnológico es ampliamente utilizado en la industria para la gestión de logs y métricas, y ofrece una solución integral para la recopilación, almacenamiento, búsqueda y visualización de datos.

Elasticsearch es un motor de búsqueda y análisis de código abierto que permite almacenar, buscar y analizar grandes volúmenes de datos de forma rápida y eficiente.



Figura 3.1: Logo de Elasticsearch ®

Logstash es una herramienta de procesamiento de logs que permite recopilar, transformar y enviar logs de diferentes fuentes a Elasticsearch para su análisis.



Figura 3.2: Logo de Logstash ®

Kibana es una plataforma de visualización de datos que permite crear dashboards interactivos y visualizaciones de datos a partir de los datos almacenados en Elasticsearch.



Figura 3.3: Logo de Kibana ®

Juntas, estas herramientas forman una solución integral para la gestión de logs y métricas, que permite a las empresas consolidar, monitorizar y analizar datos de diferentes fuentes en tiempo real.

La principal ventaja de esta solución es su flexibilidad y capacidad de personalización, que permite adaptarla a las necesidades específicas de cada empresa. Además, al tratarse de herramientas de código abierto, la pila ELK es una opción asequible y escalable que se adapta a las necesidades de la empresa.

En cuestión a la complejidad, la pila ELK puede resultar más compleja de configurar y mantener en comparación con otras soluciones al requerir múltiples componentes y una configuración detallada. Sin embargo, la comunidad activa y los recursos disponibles pueden ayudar a superar estos desafíos.

Una desventaja de la pila ELK es su curva de aprendizaje, que puede ser pronunciada para aquellos que no están familiarizados con las herramientas. Sin embargo, una vez superada esta curva, la pila ELK ofrece una solución potente y flexible para la gestión de información.

3.2.3.2. Elasticsearch, Fluentd, Kibana (EFK)

La pila EFK es una alternativa a la pila ELK que sustituye Logstash por *Fluentd*, una herramienta de código abierto para la recopilación y procesamiento de logs. Al igual que Logstash, Fluentd permite recopilar logs de diferentes fuentes y enviarlos a Elasticsearch para su análisis.



Figura 3.4: Logo de Fluentd ®

Este *stack* de tecnologías es muy similar a la alternativa anterior con la diferencia de que está enfocado a su ejecución en entornos *Kubernetes*⁴ ⁵, lo que puede ser incompatible con las necesidades de este proyecto.

⁴Simplifying Kubernetes Logging with EFK Stack

⁵How To Set Up an Elasticsearch, Fluentd and Kibana (EFK) Logging Stack on Kubernetes

3.2.3.3. Graylog + Prometheus

Esta combinación de herramientas es una alternativa interesante para la gestión de datos en entornos de producción. Graylog es una plataforma de gestión de logs de código abierto que permite centralizar, monitorizar y analizar logs de diferentes fuentes. Por otro lado, Prometheus es un sistema de monitorización y alertas de código abierto que se centra en la recopilación de métricas de sistemas y aplicaciones.



Figura 3.5: Logo de Graylog ®

En este documento se tratan de manera combinada porque, aunque son herramientas independientes, su integración es una solución completa que se está volviendo popular en el sector. Graylog se encarga de la gestión de logs, mientras que Prometheus se encarga de la recopilación de métricas y alertas.



Figura 3.6: Logo de Prometheus ®

La combinación de Graylog y Prometheus ofrece varios beneficios, como la centralización de información que facilita un análisis más integral y eficiente de los datos. Además, la mejora en la monitorización a través de las capacidades avanzadas de Prometheus, integradas con Graylog, mejora la detección de problemas y la respuesta a incidentes en tiempo real. Esta integración también proporciona flexibilidad y escalabilidad, adaptándose a las necesidades específicas de cada entorno de producción. Sin embargo, existen desventajas como la complejidad de configuración al integrar dos sistemas independientes, lo que puede resultar en un mayor esfuerzo en mantenimiento y gestión. Además, al tratarse de herramientas novedosas, supondría una mayor curva de aprendizaje para el equipo de desarrollo a la hora de tratar con las herramientas.

3.2.3.4. Loki

Loki es una herramienta de gestión de logs desarrollada por Grafana Labs, diseñada específicamente para ser altamente eficiente y escalable. A diferencia de otras soluciones de gestión de datos, Loki no indexa el contenido completo de estos, sino que se centra en los metadatos, lo que reduce significativamente los requisitos de almacenamiento y mejora el rendimiento.

Al estar creado por Grafana Labs, Loki se integra de forma nativa con Grafana, lo que facilita la visualización y análisis de los datos de logs en tiempo real. Además, está inspirado en Prometheus y pensado para utilizar con *Kubernetes*.



Figura 3.7: Logo de Loki ®

Ventajas

- **Integración con Grafana:** Una de las principales ventajas de Loki es su integración nativa con Grafana, una popular plataforma de visualización de datos. Esto permite a los usuarios crear paneles de control y alertas basados en los datos de logs de manera sencilla y eficiente.
- **Eficiencia en el almacenamiento:** Al no indexar el contenido completo de los logs, Loki reduce significativamente los requisitos de almacenamiento. Esto lo hace una opción más económica y eficiente en comparación con otras soluciones.
- **Escalabilidad:** Loki está diseñado para ser altamente escalable, lo que permite manejar grandes volúmenes de datos de logs sin comprometer el rendimiento.
- **Simplicidad en la configuración:** La configuración de Loki es relativamente sencilla, especialmente para aquellos que ya están familiarizados con Grafana y Prometheus. Esto facilita su adopción y despliegue en entornos de producción.

Desventajas

- **Limitaciones en la búsqueda:** Al no indexar el contenido completo de los logs, las capacidades de búsqueda de Loki son más limitadas en comparación con otras herramientas como Elasticsearch. Esto puede ser una desventaja en caso de necesitar realizar búsquedas complejas y detalladas.
- **Ecosistema en desarrollo:** Aunque Loki ha ganado popularidad rápidamente, su ecosistema y comunidad de usuarios aún están en desarrollo. Esto puede limitar el acceso a recursos y soporte en comparación con soluciones más maduras.
- **Dependencia de Grafana:** Si bien la integración con Grafana es una ventaja, es un factor limitante para aquellos que no utilicen Grafana, como es el caso de Okticket, al tener que adaptarse a un ecosistema diferente.

En resumen, Loki es una solución eficiente y escalable para la gestión de datos, especialmente adecuada para aquellos que ya utilizan Grafana. Sin embargo, sus limitaciones en la búsqueda y su ecosistema en desarrollo son factores a considerar antes de su implementación.

3.2.3.5. Loggly, Splunk (SaaS de gestión de logs)

Ambas herramientas se centran en la gestión y análisis de logs basada en la nube, permitiendo centralizar, monitorizar y analizar datos de logs en tiempo real. Diseñadas para simplificar la gestión de logs, las dos alternativas ofrecen una interfaz intuitiva y potentes capacidades de búsqueda y visualización que facilitan la identificación y resolución de problemas en los sistemas y aplicaciones.



Figura 3.8: Logo de Loggly ®

Una de las principales ventajas de este tipo de herramientas es su capacidad para integrarse con una amplia variedad de servicios y plataformas, lo que permite a las empresas consolidar sus datos de logs en un único lugar. Además, suelen ofrecer algún sistema de alertas en tiempo real y paneles de control personalizables, lo que encaja muy bien con algunos de los requisitos de este proyecto.

Sin embargo, también presentan algunas desventajas que entran en conflicto con los intereses descritos anteriormente. En primer lugar, al tratarse de servicios enfocados en el tratamiento exclusivo de logs, podrían no acomodar algunos de los requisitos esenciales, como la ingesta de las bases de datos propias de la empresa.



Figura 3.9: Logo de Splunk ®

Otra posible limitación es su sistema de precios; estas herramientas cuentan con una jerarquía de suscripciones que limita mucho su escalabilidad y aumentaría rápidamente los costes de su uso en caso de llegar a depender de ellas, incumpliendo así los criterios de coste y licencias.

En resumen, aunque tanto Loggly como Splunk o cualquier otro SaaS similar sean soluciones robustas y eficientes para la gestión de logs, sus posibles costes adicionales, junto con sus limitaciones en el análisis de datos de inteligencia de negocio, las convierten en opciones menos atractivas para el desarrollo de este proyecto.

3.2.4. Conclusiones

Tras evaluar las alternativas consideradas, se concluye que **ninguna de las alternativas comerciales** se ajusta completamente a los requisitos y necesidades del proyecto. Si bien todas ellas ofrecen capacidades de gestión de logs y métricas, ninguna de ellas proporciona una solución integral que cumpla con los requisitos de integración, visualización y análisis de datos de negocio de Okticket.

Por lo tanto, se considera que la mejor opción es desarrollar una solución personalizada que se adapte a las necesidades específicas de la empresa. Esta solución permitirá a Okticket consolidar y analizar datos de múltiples fuentes, así como crear dashboards de negocio personalizados que faciliten la toma de decisiones y la identificación de oportunidades de negocio.

Desde un principio, la empresa considera que el desarrollo junto a un *stack ELK* es la mejor opción para el desarrollo de este proyecto. Aunque se considera que la integración de Prometheus y Grafana es una alternativa interesante, se opta por la solución de *stack ELK* debido a su mayor flexibilidad y capacidad de personalización.

Además de la pila de *Elastic*, se analizará más adelante la posibilidad de integrar otras herramientas y tecnologías que puedan mejorar la eficiencia y escalabilidad del sistema, como Kafka, Spark o Flink, entre otras.

3.3. Descripción del proyecto

3.3.1. Dashboards planteados

Para el sistema que se describe, se plantean dos tipos de dashboards diferentes:

- **Dashboards internos:** que reflejan el rendimiento de la plataforma en tiempo real. Estos dashboards están destinados al uso interno de la empresa, y permiten a los empleados monitorizar el rendimiento de la plataforma y tomar decisiones informadas sobre su mantenimiento y evolución.
- **Dashboards externos:** que reflejan el rendimiento de las ventas y permiten a los clientes tomar decisiones informadas sobre su negocio. Estos dashboards están enfocados a los clientes de la empresa, y permite a los mismos obtener información relevante sobre su negocio que tenga Okticket.

4. Planificación del proyecto

La planificación de un proyecto es fundamental para su correcto funcionamiento y desarrollo, dentro de los plazos y costes establecidos. Se presenta un primer apartado de metodología, un segundo apartado con la planificación inicial para posteriormente inferir en base a esta el presupuesto.

4.1. Metodología

En este capítulo se aborda la metodología adoptada para el desarrollo del proyecto, fundamentada en principios ágiles y enfocada en la entrega continua de valor. La elección de *Scrum*, una metodología que permite elaborar productos software de manera incremental, revisando el producto continuamente y adaptándolo a las necesidades del cliente, subraya el compromiso con la adaptabilidad y la mejora continua del producto.

La estructura de este capítulo se organiza en torno a la descripción detallada de la metodología *Scrum*, la visualización de la planificación y las estrategias de comunicación adoptadas. A través de esta metodología, se busca optimizar los recursos disponibles, ajustarse a los plazos establecidos y garantizar la calidad del producto final.

La implementación de *Scrum* se complementa con herramientas de visualización y gestión de proyectos, como los tableros *Kanban*, que facilitan la organización y seguimiento de las tareas. Además, se pone especial énfasis en la comunicación efectiva dentro del equipo de desarrollo y con los stakeholders, asegurando así una alineación constante con los objetivos del proyecto.

Existen otras variantes de los tableros *Kanban* que se pueden utilizar para visualizar el progreso de las tareas, pero en este proyecto se ha elegido esta alternativa para facilitar la visualización de las tareas y su estado (ver 4.1.2 Visualización). La visualización de la planificación es esencial para el seguimiento y control del proyecto, ya que permite identificar posibles desviaciones y tomar medidas correctivas de manera temprana.

Este enfoque metodológico no solo refleja la planificación y ejecución del proyecto, sino que también establece las bases para una gestión eficaz, adaptativa y orientada a resultados.

4.1.1. Scrum

Para la planificación del proyecto se ha escogido *Scrum*, una metodología “ágil” que se basa en la realización de iteraciones cortas y en la adaptación a los cambios. La metodología *Scrum* se estructura en *sprints* (iteraciones cortas de una duración fija), en las que se llevan a cabo una serie de tareas que se han planificado previamente.

El primer paso de la metodología *Scrum* es la creación de un *product backlog*, una lista ordenada de las tareas a realizar durante el desarrollo del producto, a partir de los requisitos del sistema, que a su vez son una versión refinada de los requisitos iniciales del proyecto. A partir de este *product backlog* se planifican las tareas que se llevarán a cabo en cada *sprint*, de manera que sea posible cumplir con los objetivos del proyecto en el tiempo establecido.

A diferencia de metodologías tradicionales o *en cascada*, *Scrum* permite la adaptación a los cambios y la mejora continua del producto, ya que se revisa y se adapta en cada *sprint* según las necesidades del cliente y del equipo de desarrollo. Por otro lado, *Scrum* se diferencia de otras metodologías ágiles como *XP* en que no se centra tanto en las prácticas de desarrollo, sino en la gestión del proyecto y en la entrega de valor al cliente.

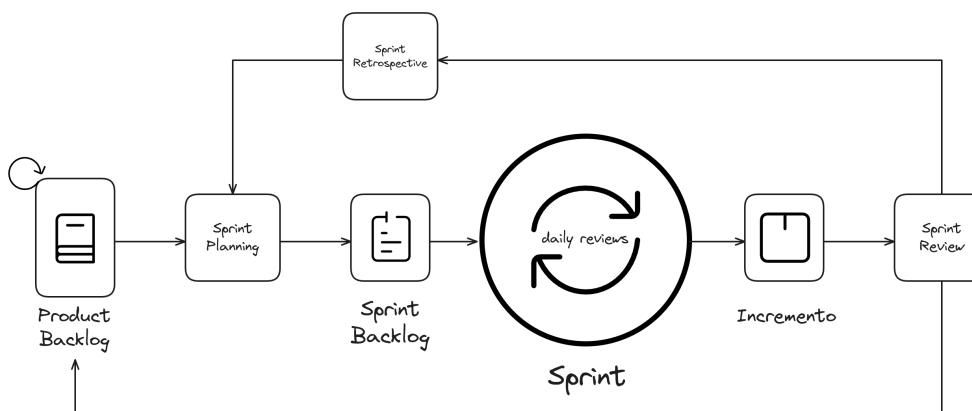


Figura 4.1: Diagrama de la metodología *Scrum*

Roles En *Scrum* se distinguen tres roles principales:

- **Product Owner:** es la persona responsable de definir los requisitos del producto y de priorizar las tareas del *product backlog*. Es el enlace entre el equipo de desarrollo y el cliente, y es el responsable de garantizar que el producto cumple con las expectativas del cliente. En el caso de este proyecto, el *Product Owner* es el director tecnológico de la empresa.
- **Scrum Master:** es la persona responsable de garantizar que el equipo de desarrollo

sigue la metodología *Scrum* y de eliminar los obstáculos que puedan surgir durante el desarrollo del proyecto. El *Scrum Master* es el encargado de organizar las reuniones diarias y de asegurar que el equipo de desarrollo cumple con los plazos y los objetivos del proyecto. En este proyecto, el *Scrum Master* son los tutores académicos del proyecto.

- **Equipo de desarrollo:** es el equipo encargado de llevar a cabo las tareas del *product backlog* y de entregar el producto final. El equipo de desarrollo es autoorganizado y multidisciplinario, y se organiza en torno a las tareas que se van a realizar en cada *sprint*. Para este proyecto, el “equipo” de desarrollo está constituido únicamente por el alumno, que se encarga de todas las tareas de desarrollo y documentación.
- **Stakeholders:** son las partes interesadas en el proyecto, como los clientes, los usuarios finales y los patrocinadores, que desconocen el proceso de desarrollo pero tienen un interés en el producto final y en su correcto funcionamiento.

Estimación En la metodología *Scrum* se pueden utilizar diferentes técnicas de estimación de tareas, como la estimación en puntos de historia, la estimación en horas o la estimación en tallas de camiseta. En este proyecto se ha optado por la estimación en tallas de camiseta, que consiste en asignar a cada tarea una talla que representa su complejidad y su duración. Las tallas de camiseta se suelen representar con letras (XS, S, M, L, XL), que se pueden traducir a puntos de historia siguiendo la secuencia de Fibonacci, es decir, $XS = 1$, $S = 2$, $M = 3$, $L = 5$, $XL = 8$.

La estimación en Scrum es esencial para la planificación de los *sprints* y para la asignación de tareas al equipo de desarrollo. La estimación en tallas se considera óptima para este proyecto, ya que permite una estimación rápida y sencilla de las tareas, al no necesitar una coordinación entre un equipo completo de desarrollo.

Además de la estimación del tamaño de las tareas, también se realiza una estimación sobre la *prioridad* de las mismas, que se representa siguiendo el equivalente de *GitHub* al sistema de colores de semáforo, donde el rojo (P0) es la máxima prioridad y el verde (P2) la mínima.

4.1.2. Visualización

Para la visualización de la planificación se ha utilizado la herramienta de gestión de proyecto de *GitHub*, que permite múltiples visualizaciones de tareas e *issues* en tableros separados.

- Se utiliza un tablero de *requisitos* al estilo *Kanban* para visualizar los requisitos del proyecto y su estado, siguiendo con la metodología *Scrum*. Un tablero *Kanban* es una herramienta visual que permite gestionar el flujo de trabajo de un proyecto por “sprints”, dividiendo las tareas en columnas y moviéndolas de una columna a otra según su estado.

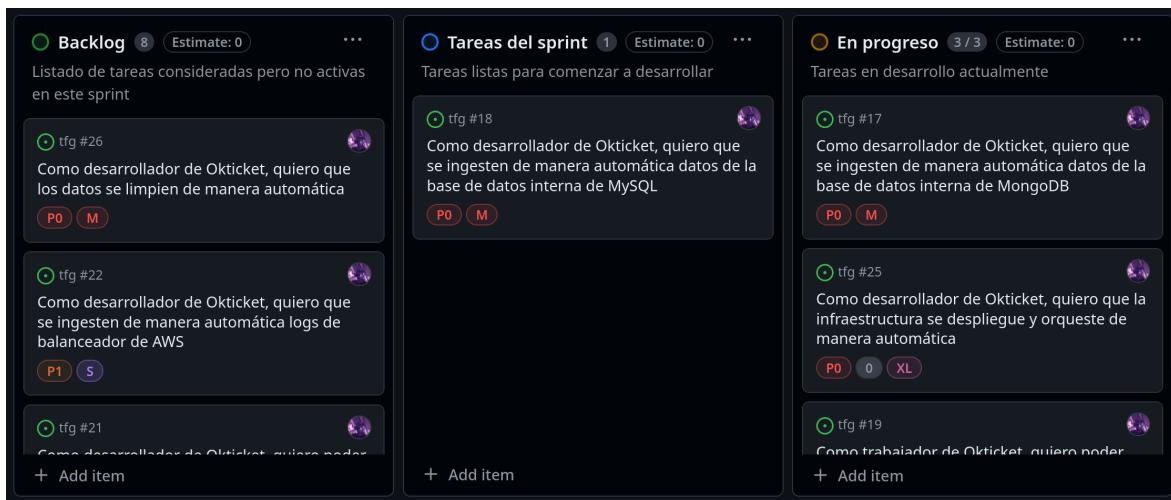


Figura 4.2: Tablero *Kanban* del proyecto

- Adicionalmente, se utiliza un *roadmap* de apartados de la memoria, separado del tablero de desarrollo normal, donde se visualiza su estado y sus fechas límite. Este *roadmap* no está relacionado con la metodología *Scrum*, sino que se ha creado para facilitar la visualización del progreso de cada sección y de la memoria en general.



Figura 4.3: Roadmap de apartados de la memoria

4.1.3. Comunicación

La comunicación con los tutores y con el equipo de desarrollo se considera fundamental para el correcto desarrollo del proyecto. Puesto que el trabajo se desarrolla de manera presencial en la oficina de la empresa, la comunicación con el equipo de desarrollo se realiza de manera frecuente y directa, mientras que la comunicación con los tutores se realiza de manera remota pero igual de frecuente, manteniendo el contacto mediante correo electrónico y Teams para pedir revisiones e informar sobre el estado del trabajo en todo momento.

4.1.4. Herramientas

Con el objetivo de facilitar las tareas de desarrollo y cumplimentar los requisitos por parte de la empresa, se utilizan las siguientes plataformas y herramientas de desarrollo para la fabricación del proyecto:

- **GitHub:** plataforma de desarrollo colaborativo para el desarrollo del proyecto. Se utiliza para la gestión de tareas, seguimiento del desarrollo y la documentación del proyecto.
- **Suite de Atlassian (*Jira, Bitbucket*):** Suite de herramientas de gestión de proyectos y desarrollo colaborativo. Se utiliza para el desarrollo y documentación del proyecto de parte de la empresa.
- **Suite de Microsoft (*Teams, Outlook*):** se utilizan las plataformas de comunicación puestas a disposición por la universidad.

4.2. Planificación inicial

Como se ha mencionado anteriormente, se utiliza la metodología *Scrum* para la planificación y desarrollo del proyecto. En la figura 4.4 se puede ver el *backlog* de tareas que se planifican en el proyecto.

| 12 Open ✓ 5 Closed | | Author ▾ | Label ▾ | Projects ▾ |
|-------------------------------------|--|--|-------------------|------------|
| <input type="checkbox"/> | Como desarrollador de Okticket, quiero poder ingestar información de páginas web externas (scrapping) enhancement | #29 opened on May 2 by miermontoto | ▷ Ingesta | |
| <input type="checkbox"/> | Como desarrollador de Okticket, quiero poder ingestar datos de APIs externas a la empresa enhancement | #28 opened on May 2 by miermontoto | ▷ Ingesta | |
| <input type="checkbox"/> | Como gestor de una empresa cliente, quiero poder ver información relevante sobre mi empresa que recoja Okticket enhancement | #27 opened on May 2 by miermontoto | ▷ Visualización | |
| <input type="checkbox"/> | Como desarrollador de Okticket, quiero que los datos se limpien de manera automática enhancement | #26 opened on May 2 by miermontoto | ▷ Software | |
| <input type="checkbox"/> | Desarrollo de un sistema de despliegue enhancement technical | #25 opened on May 2 by miermontoto | ▷ Infraestructura | |
| <input checked="" type="checkbox"/> | Creación de la infraestructura base enhancement technical | #24 by miermontoto was closed 5 days ago | ▷ Infraestructura | |
| <input type="checkbox"/> | Como desarrollador de Okticket, quiero que los datos contengan metadatos que faciliten su filtro enhancement | #23 opened on May 2 by miermontoto | ▷ Software | |
| <input type="checkbox"/> | Como desarrollador de Okticket, quiero que se ingesten de manera automática logs de balanceador de AWS enhancement | #22 opened on May 2 by miermontoto | ▷ Ingesta | |
| <input type="checkbox"/> | Como desarrollador de Okticket, quiero poder ver el estado general de la infraestructura enhancement | #21 opened on May 2 by miermontoto | ▷ Visualización | |
| <input type="checkbox"/> | Como trabajador de soporte de Okticket, quiero poder ver y consultar datos de empresas cliente enhancement | #20 opened on May 2 by miermontoto | ▷ Visualización | |
| <input type="checkbox"/> | Como trabajador de Okticket, quiero poder ver y consultar datos internos de la empresa enhancement | #19 opened on May 2 by miermontoto | ▷ Visualización | |
| <input type="checkbox"/> | Como desarrollador de Okticket, quiero que se ingesten de manera automática datos de la base de datos interna de MySQL enhancement | #18 opened on May 2 by miermontoto | ▷ Ingesta | |

Figura 4.4: Planificación inicial del proyecto

Las historias de usuario anteriores se clasifican y categorizan según su prioridad y tamaño, haciendo uso de la estrategia de tallas de camiseta como mencionado anteriormente. En el tablero *Kanban* (ver figura 4.2) se puede ver en todo momento el estado de las HU, su progreso y sus características. El listado inicial (ordenado según su prioridad) es el siguiente:

| Nombre | Prioridad | Tamaño |
|--|-----------|--------|
| Creación de la infraestructura base (técnica) | P0 | L |
| Como desarrollador de Okticket, quiero que la arquitectura se despliegue y orqueste de manera automática | P0 | XL |
| Como desarrollador de Okticket, quiero que se ingesten de manera automática datos de la base de datos interna de MongoDB | P0 | M |
| Como desarrollador de Okticket, quiero que se ingesten de manera automática datos de la base de datos interna de MySQL | P0 | M |
| Como desarrollador de Okticket, quiero que los datos se limpien de manera automática | P0 | M |
| Como trabajador de Okticket, quiero poder ver y consultar datos internos de la empresa | P1 | L |
| Como desarrollador de Okticket, quiero que se ingesten de manera automática logs de balanceador de AWS | P1 | S |
| Como desarrollador de Okticket, quiero poder ver el estado general de la infraestructura | P1 | L |
| Como desarrollador de Okticket, quiero que los datos contengan metadatos que faciliten su filtrado o búsqueda | P2 | S |
| Como trabajador de Okticket, quiero poder ver y consultar datos de empresas cliente | P2 | M |
| Como gestor de una empresa cliente, quiero poder ver información relevante sobre mi empresa que recoja Okticket | P2 | L |
| Como desarrollador de Okticket, quiero poder ingestar datos de APIs externas a la empresa | P2 | L |
| Como desarrollador de Okticket, quiero poder ingestar información de páginas web externas (<i>scraping</i>) | P2 | XL |

Tabla 4.1: Historias de usuario iniciales

Siguiendo la tabla anterior, se pueden planear los *sprints* y asignar las tareas a cada uno de ellos.

4.3. Presupuesto

Para poder llevar a cabo este proyecto, se realiza una estimación del coste total necesario para su desarrollo, que se divide en dos partes: el coste del material, que incluye el coste de los recursos necesarios para el desarrollo del proyecto, y el coste del personal, que incluye el coste de las horas de trabajo del desarrollador.

4.3.1. Presupuesto de material

Puesto que el proyecto se desarrolla en la empresa, se dispone de todos los recursos físicos necesarios para llevar a cabo el proyecto, es decir, que no se incluirá el coste del ordenador o de la conexión a internet en el presupuesto.

Sin embargo, se incluirá el coste de las herramientas y servicios utilizados durante el desarrollo del proyecto, como el coste de las licencias de software, el coste de los servicios en la nube, el coste de las herramientas de desarrollo, etc.

Es importante destacar de que los precios de los servicios en la nube son aproximados y pueden variar en función de la región, el tipo de instancia, el tipo de almacenamiento, etc. Por lo tanto, los precios presentados en este presupuesto son orientativos y pueden variar en función de las necesidades del proyecto. En este caso, se analizan los precios en junio de 2024 en la región de Amazon Web Services (AWS) de *eu-west-3* (París).

| Categoría | Ítem | Cantidad | Coste unitario | Coste total |
|-----------------|-----------------|-------------------|-------------------|-------------|
| AWS Compute | Fargate | 500 vCPU-h/mes | 0,04665€/vCPU-h | 23,33€ |
| | | 1000 GB-h/mes | 0,00511€/GB-h | 5,11€ |
| AWS Storage | EFS | 500 GB/mes | 0,36€/GB-mes | 180,00€ |
| | S3 | 250 GB/mes | 0,0245€/GB-mes | 6,13€ |
| AWS Network | VPC | 4 NAT Gateways | 0,052€/h | 149,76€ |
| | ELB | 4 ALBs | 0,0243€/h | 70,00€ |
| AWS Security | IAM | - | Sin cargo | 0,00€ |
| | KMS | 1 CMK | 1,00€/mes | 1,00€ |
| Stack KELK | Kafka | - | Licencia gratuita | 0,00€ |
| | Elasticsearch | - | Licencia gratuita | 0,00€ |
| | Logstash | - | Licencia gratuita | 0,00€ |
| | Kibana | - | Licencia gratuita | 0,00€ |
| AWS Container | ECS | - | Sin cargo | 0,00€ |
| AWS Monitor | CloudWatch | 5 métricas | 0,30€/métrica-mes | 1,50€ |
| | X-Ray | 50,000 trazas/mes | 4,60€/1M trazas | 0,23€ |
| Despliegue | Terraform | - | Licencia gratuita | 0,00€ |
| Capacitación | Cursillo online | 1 | 184,00€ | 184,00€ |
| Subtotal | | | | 620,06€ |
| Otros | Support | Plan Basic | Sin cargo | 0,00€ |
| | Optimización | - | 5 % del subtotal | 31,00€ |
| | Contingencia | - | 10 % del subtotal | 62,00€ |
| Total | | | | 713,06€ |

Tabla 4.2: Propuesta de presupuesto de materiales

4.3.2. Presupuesto de personal

A continuación, se presenta una propuesta de presupuesto de personal para el desarrollo del proyecto, que incluye el coste de las horas de trabajo según cada rol y el coste total del personal.

| Rol | Descripción | Horas/mes | CU (€/h) | CT (€/mes) |
|-----------------|---|-----------|----------|------------------------|
| Arquitecto | Diseño de la arquitectura y supervisión | 40 | 60 | 2.400,00€ |
| Desarrollador | Desarrollo y mantenimiento | 160 | 45 | 7.200,00€ |
| Administrador | Gestión de sistemas y seguridad | 160 | 50 | 8.000,00€ |
| DevOps | Infraestructuras y monitorización | 80 | 55 | 4.400,00€ |
| Subtotal | | | | 22.000,00€ |
| Otros | IVA (21 %) Margen (5 %) | | | 4.620,00€ 1.100,00€ |
| Total | | | | 27.720,00€ |

Tabla 4.3: Propuesta de presupuesto de personal

El coste del personal es ficticio, pero se ha calculado en base a experiencias previas de contratación y subcontratación de personal en la empresa, además de tener en cuenta el coste medio de los roles en Asturias.

4.3.3. Presupuesto total

Finalmente, se presenta el presupuesto total del proyecto, que incluye el coste del material y el coste del personal, así como el coste total del proyecto.

| Concepto | Coste (€/mes) |
|-----------------------------|-------------------|
| Presupuesto de materiales | 713,06€ |
| Presupuesto de personal | 27.720,00€ |
| Subtotal | 28.433,06€ |
| Beneficio Industrial (15 %) | 4.264,96€ |
| Total | 32.698,02€ |

Tabla 4.4: Costes combinados de presupuesto y materiales con beneficio industrial

El presupuesto total del proyecto asciende a 32.698,02€ (treinta y dos mil seiscientos noventa y ocho euros con dos céntimos), que incluye el coste del material, el coste del personal y el beneficio industrial, además de sus márgenes.

5. Diseño del sistema

Previo al desarrollo y la implementación del proyecto, es necesario realizar un diseño detallado del sistema que permita definir la arquitectura, los modelos de datos y las tecnologías a utilizar. En este capítulo se estudiarán las alternativas disponibles, se definirá la arquitectura del sistema en la nube y se establecerán los modelos de datos necesarios para el desarrollo del proyecto.

5.1. Estudio de alternativas

En este apartado se explorarán las diferentes alternativas disponibles para el diseño del sistema. Se analizarán las características, ventajas y desventajas de cada opción, con el objetivo de proporcionar una visión clara y fundamentada que permita seleccionar la alternativa más adecuada para el proyecto. Las áreas de estudio incluirán tanto el despliegue de infraestructura como otros aspectos críticos del diseño del sistema, asegurando una evaluación integral y detallada de las posibles soluciones.

Los criterios de evaluación en líneas generales son los ya vistos en la sección 3.2 *Alternativas existentes*, es decir: **coste, complejidad, rendimiento y escalabilidad y licencias** (o, más bien, la ausencia de ellas).

En esta sección se estudiarán las alternativas referentes a:

- Despliegue de infraestructura
- Ingesta de datos
- Proveedor de nube
- Sistemas de ejecución y servicios
- Tipos de máquinas

El orden de estudio de las alternativas no es casual, sino que sigue un orden lógico *top-down* en el desarrollo del proyecto, comenzando por la herramienta de despliegue y los componentes para centrarse posteriormente en proveedores, servicios, etc.

5.1.1. Despliegue de infraestructura

A la hora de desplegar la infraestructura de un proyecto, se consideran varias herramientas populares que permiten automatizar este proceso. Entre todas ellas, las más establecidas y atractivas son *Terraform*, *AWS CloudFormation* y *Ansible*.

A continuación, se describen brevemente estas herramientas y se comparan sus características.

Alternativas

Terraform es una herramienta de código abierto desarrollada por *HashiCorp* que permite definir y desplegar infraestructura de forma declarativa. *Terraform* permite definir la infraestructura en un archivo de configuración JSON, que describe los recursos que se desean crear y sus dependencias. A partir de este archivo, *Terraform* se encarga de desplegar los recursos en el proveedor de nube especificado, que en el caso de este proyecto es AWS.



Figura 5.1: Logo de Terraform ®

AWS CloudFormation es un servicio de *Amazon Web Services* similar a *Terraform* que permite definir y desplegar infraestructura en la nube de forma declarativa. *AWS CloudFormation* permite definir la infraestructura o bien mediante un archivo de configuración (en formato JSON o YAML), o bien gráficamente mediante diagramas, un punto muy fuerte a favor de esta alternativa.



Figura 5.2: Logo de AWS CloudFormation ®

Ansible es una herramienta multi-propósito de automatización de tareas entre las que se incluye el despliegue y orquestación de infraestructura. Se trata de una herramienta desarrollada por *Red Hat* que permite definir la infraestructura mediante *playbooks* escritos en YAML, que describen las tareas a realizar y los servidores en los que se deben ejecutar.



Figura 5.3: Logo de Ansible ®

Comparación Comenzando la comparativa por la facilidad de uso de cada herramienta, *Terraform* es la alternativa planteada más fácil de usar, ya que permite definir la infraestructura en un archivo con sintaxis sencilla y desplegarla con un solo comando. Por otro lado, *AWS CloudFormation* es un poco más complejo de usar, ya que requiere definir la infraestructura en un archivo de configuración o en un diagrama, y luego desplegarla mediante la consola de *AWS*. *Ansible* es más complejo de usar, ya que requiere definir la infraestructura mediante *playbooks* y ejecutarlos en los servidores, pero es más flexible y potente que las otras dos herramientas.

Mientras que *Terraform* y *Ansible* son herramientas *multi-cloud*, lo que significa que funcionan con cualquier proveedor de nube, *AWS CloudFormation* es una herramienta específica de *AWS* y solo funciona con sus servicios, lo que puede suponer tanto una ventaja como una desventaja, dependiendo de las necesidades del proyecto. En este caso, la solución se va a desplegar en la nube de *Amazon*, pero a la vez no se requiere el uso de servicios específicos de *AWS*, no se considera una ventaja significativa.

Decisión Ninguna de las alternativas consideradas es claramente superior a las demás, ya que se tratan de herramientas con características y funcionalidades similares y una gran popularidad en la industria. Sin embargo, se decide utilizar *Terraform* para el despliegue de la infraestructura de este proyecto, ya que es la herramienta más “sencilla”, la que mejor se podría adaptar a las necesidades del proyecto y la única que ya se ha usado en proyectos anteriores dentro de la empresa.

5.1.2. Ingesta de datos

A partir del conjunto de tecnologías seleccionadas en la descripción detallada del proyecto, se consideran diversas tecnologías, como *Redpanda*, *AWS Glue* y *Kafka* que permitan ingestar datos de todas las fuentes que requieren ser procesadas.

Alternativas

Kafka es una plataforma de transmisión de datos distribuida y de código abierto que se utiliza para construir pipelines de datos en tiempo real y aplicaciones de streaming. Desarrollada originalmente por LinkedIn y posteriormente donada a la Apache Software Foundation, *Kafka* se ha convertido en una de las tecnologías más populares para la gestión de flujos de datos en tiempo real.



Figura 5.4: Logo de Kafka ®

Una de las principales ventajas de *Kafka* es su capacidad para manejar grandes volúmenes de datos con alta eficiencia y baja latencia. *Kafka* utiliza un modelo de publicación-suscripción, donde los productores publican mensajes en temas y los consumidores se suscriben a estos temas para recibir los mensajes. Esta arquitectura permite una alta escalabilidad y flexibilidad en la gestión de datos.

Kafka se compone de varios componentes clave:

- **Tópico:** Un tópico es una categoría a la que se envían los mensajes y a la que los consumidores están *suscritos*. Los consumidores pueden estar suscritos a uno o varios tópicos, y los productores pueden enviar mensajes a uno o varios tópicos. Los tópicos son la unidad básica de organización de los mensajes en cualquier sistema de mensajería de publicación/suscripción.
- **Productor:** El productor es el componente responsable de crear y enviar mensajes al cluster de Kafka. Está separado del resto de los componentes y produce mensajes de manera asíncrona y rápida.

- **Consumidor:** El consumidor es el componente responsable de leer los mensajes producidos por el productor. Está suscrito a un tópico a través del broker y consume los mensajes.
- **Broker:** El broker es el componente responsable de recibir los mensajes producidos por el productor y enviarlos a los consumidores. Es el intermediario entre los productores y los consumidores.
- **Zookeeper:** Zookeeper es un servicio separado de coordinación distribuida que se utiliza para gestionar y coordinar los brokers de Kafka. Se encarga de mantener la información de los brokers y de los tópicos. Actualmente, este servicio es una dependencia obligatoria de Kafka.¹

A pesar de sus numerosas ventajas, *Kafka* también presenta algunos desafíos. La configuración y gestión de un clúster de *Kafka* puede ser compleja, especialmente en entornos de producción a gran escala. Además, *Kafka* depende de *Zookeeper* para la coordinación, lo que añade una capa adicional de complejidad en la administración del sistema.

En resumen, *Kafka* es una solución robusta y escalable para la transmisión de datos en tiempo real, ideal para aplicaciones que requieren alta disponibilidad y procesamiento eficiente de grandes volúmenes de datos. Sin embargo, su implementación y gestión requieren un conocimiento profundo de su arquitectura y componentes.

Redpanda es una plataforma de transmisión de datos en tiempo real que se destaca por su alto rendimiento y baja latencia. Diseñada como una alternativa moderna a *Kafka*, *Redpanda* ofrece una arquitectura simplificada que elimina la necesidad de dependencias externas como *Zookeeper*. Esto no solo reduce la complejidad operativa, sino que también mejora la eficiencia y la escalabilidad del sistema. *Redpanda* es compatible con la API de *Kafka*, lo que facilita la migración de aplicaciones existentes sin necesidad de cambios significativos en el código. Además, su diseño optimizado para hardware moderno permite un procesamiento más rápido y un uso más eficiente de los recursos, lo que la convierte en una opción ideal para aplicaciones que requieren una transmisión de datos rápida y confiable.

Sin embargo, *Redpanda* también presenta algunos puntos en contra. Al ser una tecnología relativamente nueva, su ecosistema y comunidad de usuarios no son tan amplios como los de *Kafka*, lo que puede limitar el acceso a recursos y soporte. Además, aunque la compatibilidad con la API de *Kafka* es una ventaja, puede haber ciertas características y extensiones

¹Dejará de ser necesario en la versión 4. <https://x.com/coltmcnealy/status/1801987159534264641>

específicas de *Kafka* que no estén completamente soportadas en *Redpanda*. Finalmente, la adopción de una nueva tecnología siempre conlleva riesgos asociados con la estabilidad y el soporte a largo plazo, aspectos que deben ser considerados cuidadosamente antes de su implementación.

AWS Glue es un servicio de integración de datos totalmente administrado que facilita la preparación y carga de datos para análisis. Diseñado para trabajar con grandes volúmenes de datos, este servicio automatiza las tareas de descubrimiento, catalogación, limpieza, enriquecimiento y movimiento de datos entre diferentes almacenes de datos.

Una de las principales ventajas de Glue es su capacidad para generar automáticamente el código necesario para realizar las transformaciones de datos, lo que reduce significativamente el tiempo y el esfuerzo requeridos. Además, es altamente escalable y puede manejar tanto cargas de trabajo por lotes como en tiempo real, lo que lo convierte en una opción muy versátil.

AWS Glue es un servicio administrado, por lo que su uso puede implicar costes adicionales en comparación con soluciones autogestionadas e introducir *vendor lock-in*. Además, aunque ofrece una gran flexibilidad y potencia, su configuración y optimización pueden requerir un conocimiento profundo de los servicios de la nube de Amazon y las correspondientes prácticas de integración de datos.

Comparación y decisión Desde el primer momento, en la empresa se considera Kafka como la opción más sólida junto con el *stack ELK* para desarrollar el proyecto, al tratarse de un estándar en la industria y una solución tanto rápida y escalable como asequible a nivel económico. Por eso, y pese a que las otras alternativas son atractivas para el desarrollo de este proyecto, se decide utilizar Kafka como servicio de ingestión de datos, en consonancia con *Logstash*.

5.1.3. Proveedor de nube

En cuanto a la elección del proveedor de nube, existen actualmente tres grandes alternativas en el mercado: *Amazon Web Services* (AWS), *Microsoft Azure* y *Google Cloud Platform* (GCP). Cada uno de estos proveedores ofrece una amplia gama de servicios y herramientas que permiten a las empresas construir, desplegar y escalar aplicaciones en la nube de forma rápida y eficiente.

- **Amazon Web Services (AWS)** es el proveedor de nube más grande y popular del mundo, con una amplia gama de servicios y herramientas que permiten a las empresas construir, desplegar y escalar aplicaciones en la nube de forma rápida y eficiente. AWS ofrece una infraestructura global con centros de datos en todo el mundo, lo que garantiza una alta disponibilidad y rendimiento de los servicios. Además, AWS cuenta con una amplia comunidad de usuarios y desarrolladores, lo que facilita la integración y el soporte de las aplicaciones en la nube.
- **Microsoft Azure** es otro proveedor de nube líder en el mercado conocido por su integración con las herramientas y servicios de Microsoft. El hecho de formar parte de las soluciones de Microsoft puede ser una ventaja para las empresas que ya utilizan sus productos, como es el caso de la Universidad de Oviedo.
- **Google Cloud Platform (GCP)** es el proveedor de nube de Google y, al igual que AWS y Azure, ofrece una amplia gama de servicios y herramientas para construir, desplegar y escalar aplicaciones en la nube. GCP es conocido por su enfoque en la innovación y la tecnología de vanguardia, lo que puede ser atractivo para empresas que buscan soluciones avanzadas y de alto rendimiento. Sin embargo, es la opción menos utilizada de las tres y ha tenido escándalos recientes de preservación de la información.²

Pese a que las tres alternativas son válidas y ofrecen una amplia gama de servicios y herramientas, la empresa ya utiliza la nube de Amazon para todas sus aplicaciones y servicios, por lo que es la opción más lógica y coherente para este proyecto. Además, AWS cuenta con servicios específicos referentes a *contenedores* que facilitarán el despliegue de la infraestructura y la gestión de los servicios en la nube.

²Google Cloud accidentally deletes \$1.25 billion Australian pension fund

5.1.4. Sistemas de ejecución y servicios

Okticket y el equipo de desarrollo utiliza *Amazon Web Services* (AWS) como proveedor de nube preferido para desplegar sus servicios y aplicaciones. AWS ofrece una amplia gama de servicios y herramientas que permiten a las empresas construir, desplegar y escalar aplicaciones en la nube de forma rápida y eficiente. Dentro de la plataforma de AWS, existen varios servicios que pueden ser utilizados para implementar las funcionalidades requeridas por el proyecto.

Amazon EC2 es el servicio de computación tradicional de AWS, que permite lanzar máquinas virtuales con arquitecturas comunes de manera rápida. Al tratarse de un sistema normal de máquinas virtuales, EC2 no es totalmente compatible con el modelo de microservicios y contenedores, lo que puede limitar su escalabilidad y flexibilidad en entornos de producción a gran escala.

Amazon ECS es un servicio de orquestación de contenedores que permite ejecutar y escalar contenedores de Docker en la nube de AWS. ECS es totalmente compatible con Docker y proporciona una interfaz sencilla para gestionar contenedores en entornos de producción. Sin embargo, ECS puede ser complicado de configurar y gestionar, especialmente en entornos de gran escala.

Amazon EKS es un servicio de orquestación de contenedores basado en Kubernetes que permite ejecutar y escalar contenedores de Docker en la nube de AWS. EKS es totalmente compatible con Kubernetes y proporciona una interfaz sencilla para gestionar clústeres de Kubernetes en entornos de producción. EKS es una opción popular para empresas que ya utilizan Kubernetes y desean aprovechar las ventajas de la nube de AWS. Sin embargo, esto supondría plantear todo el desarrollo desde cero en Kubernetes, un sistema que no se ha utilizado en la empresa hasta la fecha y que requeriría una curva de aprendizaje significativa.

Para este proyecto, se valoran positivamente las opciones más “novedosas” pero también se ha de tener en cuenta las limitaciones de tiempo y recursos, por lo que se decide utilizar *Amazon ECS* como servicio de orquestación de contenedores, ya que es el servicio más sencillo y fácil de configurar de los tres, y el que mejor se adapta a las necesidades del proyecto.

5.1.5. Tipos de máquinas

Una vez seleccionado el proveedor de nube y el servicio de orquestación, en este caso *Amazon ECS*, existen dos tipos de máquinas virtuales que se pueden utilizar para desplegar los contenedores: *EC2* y *Fargate*.

Amazon EC2 , como ya se ha comentado, es el servicio de computación tradicional de AWS que permite lanzar máquinas virtuales con arquitecturas comunes de manera rápida. Al escoger esta opción, se tendría que gestionar el *tipo de instancia*³ y la *capacidad* de las máquinas, lo que puede acarrear más tiempo de análisis y configuración.

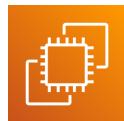


Figura 5.5: Logo de Amazon EC2 ®

AWS Fargate ⁴ es un servicio de contenedores de AWS con un enfoque más moderno que EC2, ya que permite ejecutar contenedores sin necesidad de gestionar las máquinas subyacentes. Fargate se encarga de aprovisionar y escalar la infraestructura necesaria para ejecutar los contenedores en base a los recursos definidos en la configuración. Aunque Fargate es más sencillo de usar y configurar que EC2, también puede ser más costoso y menos flexible, ya que no es posible acceder directamente a las máquinas subyacentes de manera sencilla o personalizar la configuración de las instancias.



Figura 5.6: Logo de AWS Fargate ®

Decisión Dado que el proyecto no requiere una configuración específica de las máquinas subyacentes y se busca una solución sencilla y rápida de desplegar, se decide utilizar *AWS Fargate* como servicio de contenedores para ejecutar los servicios del sistema. Fargate es una opción más moderna y sencilla que EC2, y permite centrarse en el desarrollo de las aplicaciones sin tener que preocuparse por la gestión de la infraestructura subyacente.

³<https://aws.amazon.com/es/ec2/instance-types/>

⁴<https://aws.amazon.com/es/fargate/>

5.2. Arquitectura del sistema

Tras la definición de los requisitos y la valoración de las alternativas disponibles, en este apartado se plantea la arquitectura completa del sistema en la nube, tomando como proveedor a *Amazon Web Services* (AWS), ya que es el proveedor de nube preferido por la empresa.

La arquitectura definida a continuación deberá ser definida y desplegada de manera automatizada mediante *Terraform* contando con la mínima intervención posible por parte de los operadores del sistema, y hará uso de *Amazon ECS* como servicio de orquestación de contenedores.

Además de *ECS*, se utilizarán otros servicios de AWS necesarios para el planteamiento de una arquitectura completa y funcional, como *VPC*, *IAM*, *EFS*, *S3*, *SG*, entre otros. A continuación, se detallan los servicios y componentes que formarán parte de la arquitectura del sistema.

- **IAM:** *Identity and Access Management* es un servicio que permite gestionar el acceso a los recursos de AWS de forma segura. En este caso, se crearán roles y políticas de IAM para controlar el acceso a los servicios del sistema y garantizar la seguridad de los datos.
- **ALB/NLB:** Los *Application* o *Network Load Balancers* son servicios de balanceo de carga que permiten distribuir el tráfico entre los contenedores del sistema. En este caso, se configurará un ALB o NLB para equilibrar la carga entre los contenedores del sistema y garantizar la disponibilidad y la escalabilidad de los servicios.
 - Los平衡adores de carga cuentan a su vez con varios componentes como *Target Groups* y *Listeners* que permiten configurar las reglas de enrutamiento y el tráfico de red.
 - La diferencia entre ALB y NLB⁵ radica en el nivel de la capa de red en la que operan, siendo ALB adecuado para aplicaciones web y NLB para aplicaciones de red de capa 4. Puesto que se usan servicios que operan mediante el protocolo TCP y no HTTP, se utilizará un NLB para garantizar la conectividad de dichos servicios. Sin embargo, el uso de un NLB conlleva mayor complejidad de configuración y mayores costes, además de limitar la capacidad de integración con otros servicios de AWS como el sistema de logs.

⁵<https://aws.amazon.com/es/compare/the-difference-between-the-difference-between-application-network-load-balancer-and-application-load-balancer/>

- **Componentes de red:** Se configurarán varios componentes de red, como *VPC*⁶, *Subnets*, *Route Tables*, *Internet Gateways*, *NAT Gateways*..., para garantizar la conectividad y la seguridad de los servicios del sistema.
- **EFS:** *Elastic File System* es un servicio de almacenamiento de archivos que permite compartir archivos entre los contenedores del sistema. EFS es vital para almacenar los datos y configuraciones de manera compartida entre los contenedores.
- **S3:** *Simple Storage Service* es un servicio de almacenamiento de objetos que permite almacenar y recuperar grandes volúmenes de datos de forma segura y escalable. Se usará S3 para almacenar los datos de los servicios del sistema.
- **CloudWatch:** *CloudWatch* es un servicio de monitorización y gestión de logs que permite supervisar y analizar los recursos de AWS en tiempo real. Este servicio se utilizará durante el periodo de desarrollo de la infraestructura, cuando la ingestión de datos no esté completamente implementada.
- **SG:** Los *Security Groups* son reglas de seguridad que definen qué tráfico está permitido o denegado en los recursos de AWS. Se tendrán que configurar grupos de seguridad para controlar el tráfico desde y hacia los servicios del sistema.
- **ECS:** Dentro de *Elastic Container Service*, se configurarán los clústeres, servicios, tareas y contenedores necesarios para ejecutar los servicios del sistema.
- **Route 53:** *Route 53* es un servicio de DNS que permite rastrear y redirigir el tráfico de red a los recursos de AWS. En este caso, se utilizará Route 53 para gestionar los nombres de dominio de la empresa y redirigir el tráfico a los servicios del sistema.
- **Secret Manager:** *Secrets Manager* es un servicio de gestión de secretos que permite almacenar y recuperar información sensible de forma segura. Se utilizará Secret Manager para gestionar las credenciales y claves de acceso de los servicios del sistema.
- **ACM:** *Certificate Manager* es un servicio de gestión de certificados SSL/TLS que permite proteger las conexiones seguras entre los servicios del sistema. ACM gestionará los certificados SSL/TLS de los recursos de AWS.

A continuación, se presentan los diagramas de la arquitectura del sistema en AWS para cada una de las áreas de estudio: infraestructura, seguridad y redes. En todos los diagramas se resaltan en morado (con líneas intermitentes) la región y en verde la nube virtual privada (*VPC*) en la que se desplegarán los servicios del sistema.

⁶https://docs.aws.amazon.com/es_es/vpc/latest/userguide/what-is-amazon-vpc.html

5.2.1. Infraestructura

Para la infraestructura del sistema, se utilizará un clúster de *ECS* con cuatro servicios en total: uno dedicado a *Elasticsearch*, otro para *Kibana*, un tercero para *Logstash* y por último un servicio que recoja *Kafka* y, su dependencia, *Zookeeper*. Cada uno de estos servicios estará compuesto por tantas tareas como se requieran para garantizar la disponibilidad y escalabilidad de los servicios, aunque inicialmente solo se desplegará una tarea por servicio. Dentro de cada tarea se podrán encontrar los correspondientes contenedores - imágenes de Docker que contienen el código y las dependencias necesarias para ejecutar los servicios.

Cada uno de los servicios estará detrás de un *ALB* o *NLB* que se encargará de distribuir el tráfico entre las tareas, garantizando la disponibilidad y escalabilidad de los servicios.

Los *ALB* están conectados directamente a un *bucket S3* que almacena los logs de los servicios (*NLB* no soporta esta funcionalidad). Además, los servicios que necesiten almacenar datos de forma persistente (todos menos Kafka) estarán conectados a un sistema de archivos *EFS* que permitirá compartir datos y configuraciones entre los contenedores del sistema.

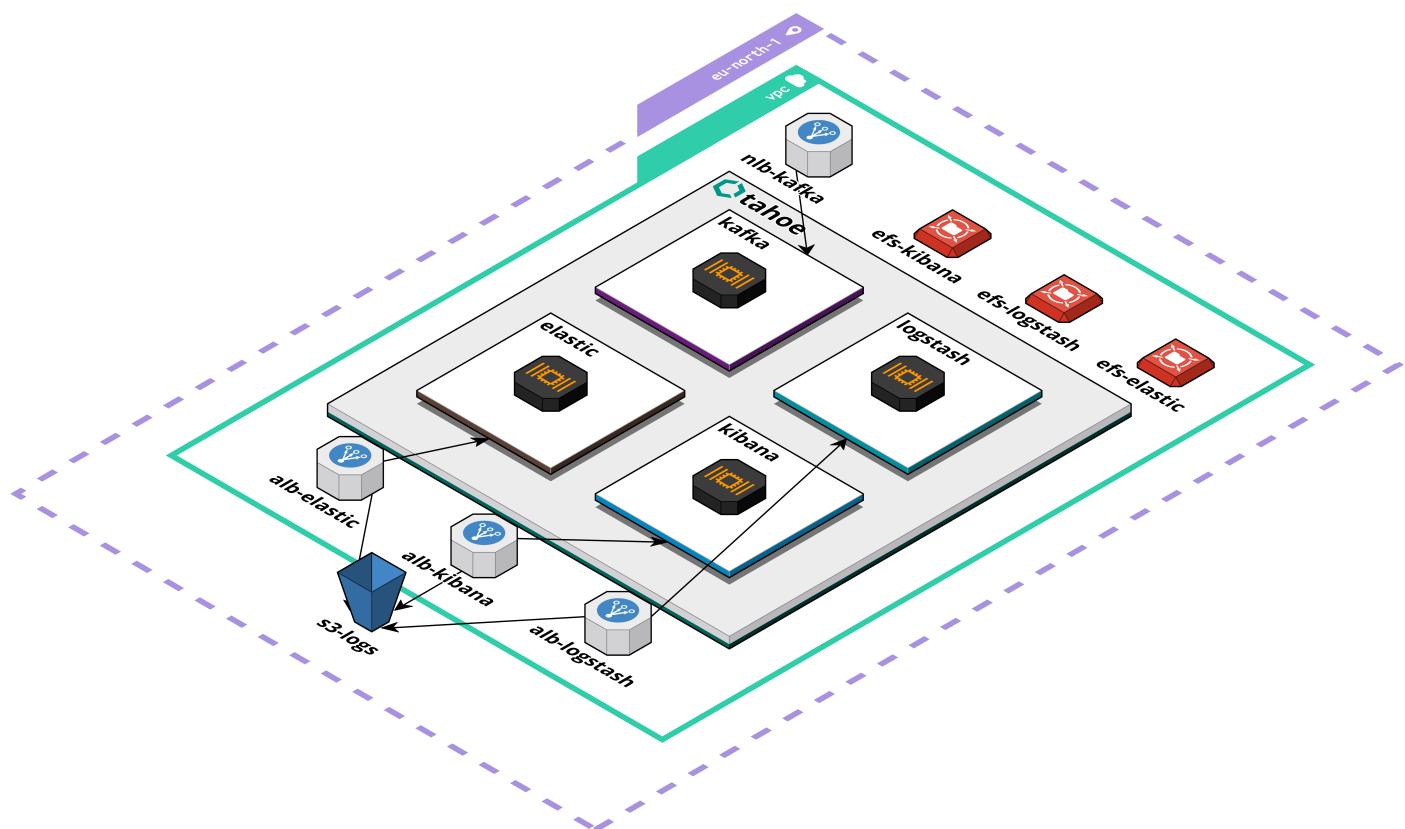


Figura 5.7: Diagrama inicial de la infraestructura en AWS

5.2.2. Seguridad

A nivel de seguridad, se debe garantizar la protección de los datos y la confidencialidad de la información. Para ello, se utilizarán varios servicios de AWS, como *IAM* para la gestión de roles y políticas de seguridad, o *grupos de seguridad* que permiten controlar el tráfico de red entre los contenedores del sistema.

Cada uno de los componentes del *clúster* de *ECS* tendrá su propio grupo de seguridad que permitirá controlar el tráfico de red según los puertos y protocolos permitidos. Además, se deberán utilizar tan solo las políticas y roles necesarios para garantizar la seguridad de los datos.

Además de los componentes de seguridad de AWS, se utilizarán protocolos seguros como *HTTPS* para la comunicación entre los servicios y se cifrará la información haciendo uso de *Secret Manager* para cargar y guardar las claves necesarias.

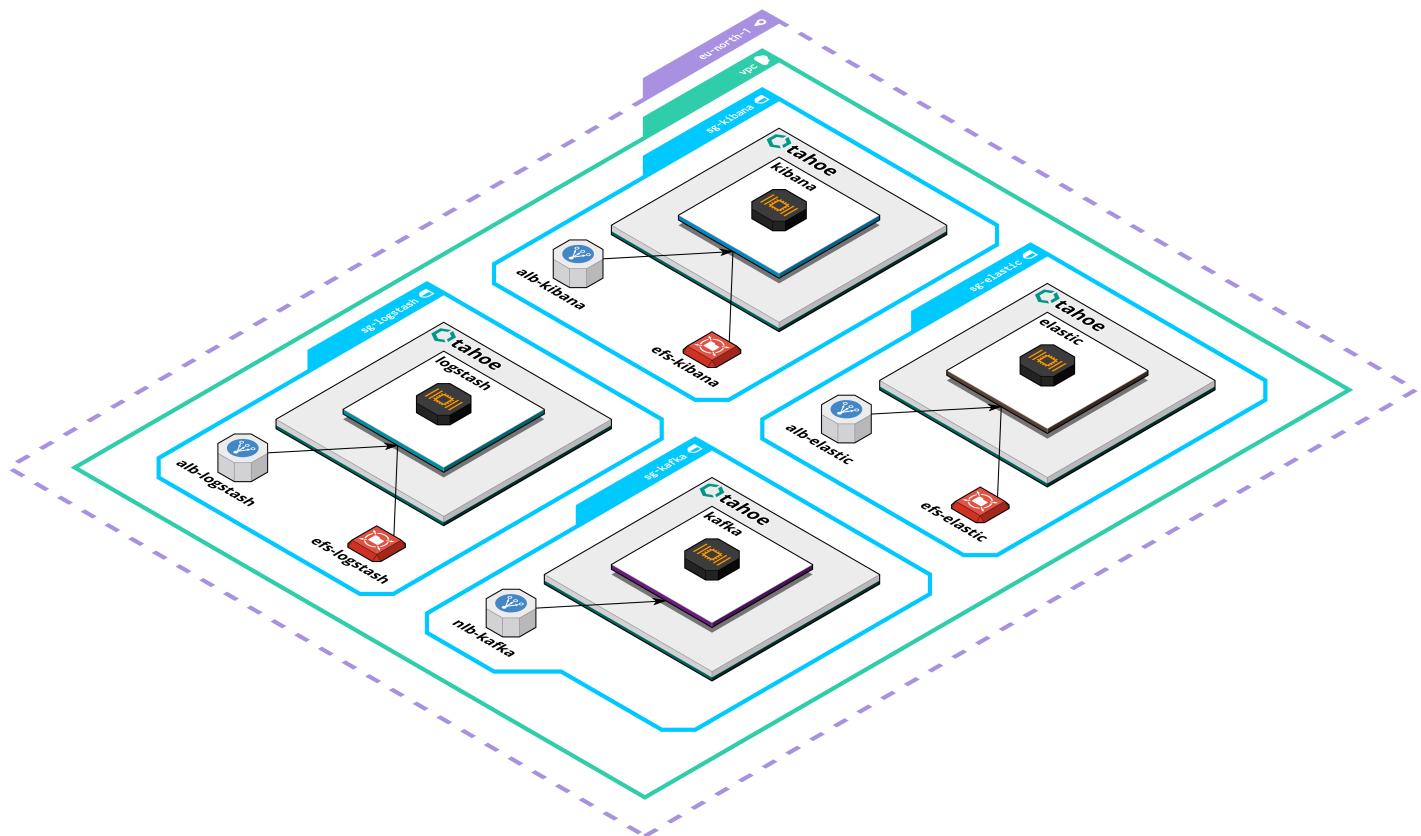


Figura 5.8: Diagrama inicial de la seguridad en AWS

En el diagrama anterior, se pueden observar los grupos de seguridad (resaltados en azul) que engloban a los componentes, mientras que todos ellos se encuentran dentro de una *VPC*.

5.2.3. Redes

A nivel de configuración de redes, se establecen tres subredes: dos públicas y otra privada, estando las públicas en zonas de disponibilidad diferentes. La subred pública estará conectada a internet a través de un *Internet Gateway*, mientras que la subred privada estará conectada a internet a través de un *NAT Gateway*. Ambas subredes estarán conectadas a una VPC que permitirá la comunicación entre los servicios del sistema.

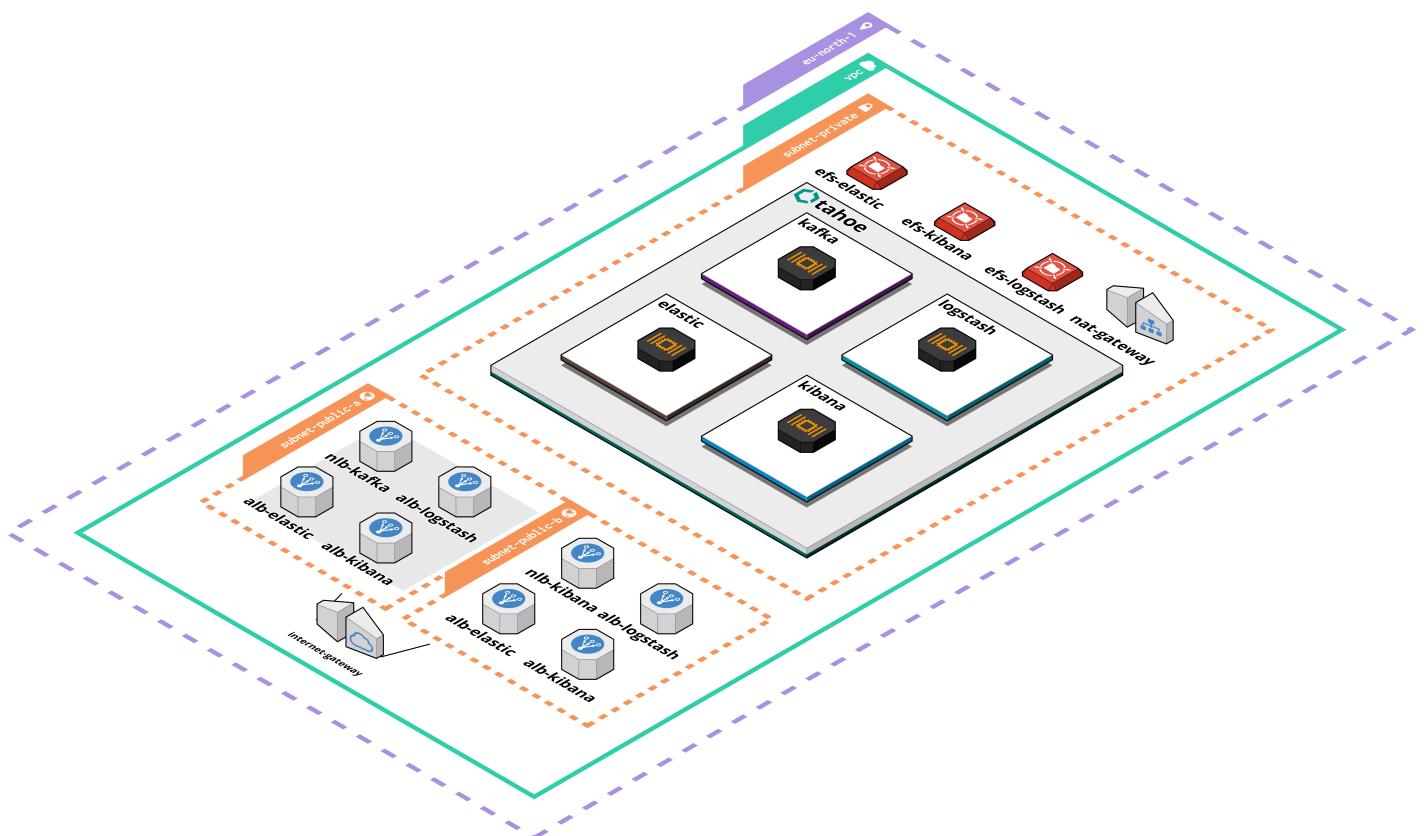


Figura 5.9: Diagrama inicial de las redes en AWS

En el diagrama se pueden observar las subredes (resaltadas en naranja con líneas intermitentes), siendo las dos inferiores subredes públicas en diferentes zonas de disponibilidad conectadas a un *Internet Gateway* y la subred superior una privada donde se encuentran los servicios junto con el *NAT Gateway* necesario para la conexión a Internet.

Además de los componentes planteados en el esquema anterior, se configurarán *rutas* y *tablas de rutas* para garantizar la conectividad y la seguridad de los servicios del sistema.

5.3. Modelos de datos

Los modelos de datos son una representación estructurada de la información almacenada que se utiliza para almacenar, recuperar y manipular los datos de forma eficiente. En este caso, se definirán los modelos de datos que se ingestarán en el sistema, así como las relaciones entre ellos y las características de cada uno.

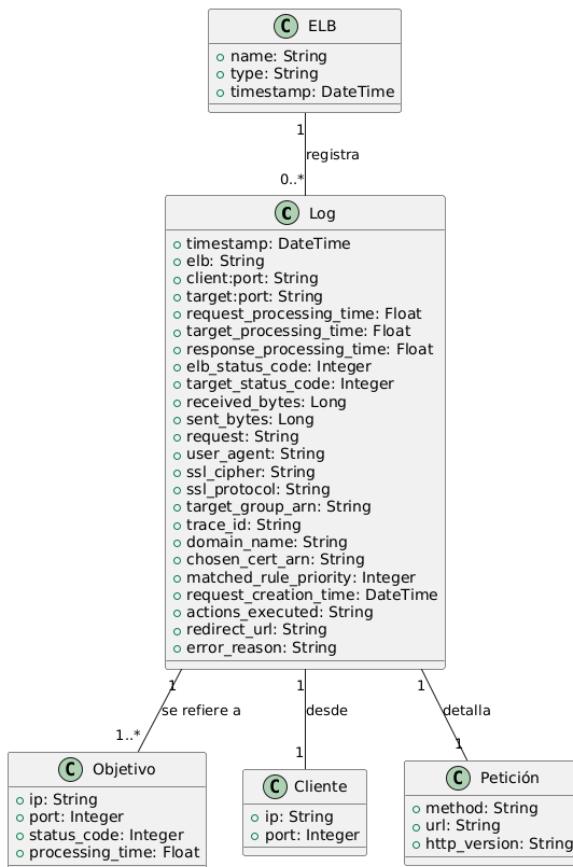


Figura 5.10: Modelo de datos de los logs de un balanceador de carga de AWS

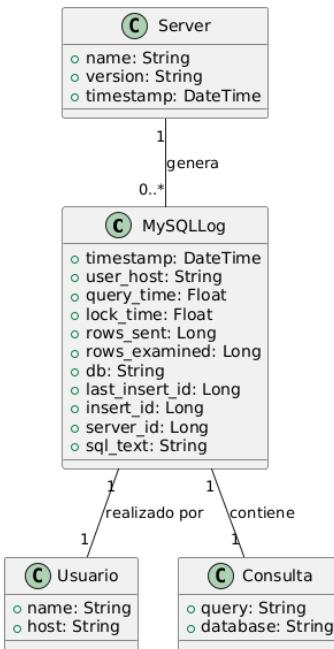


Figura 5.11: Modelo de datos de los logs de una base de datos SQL

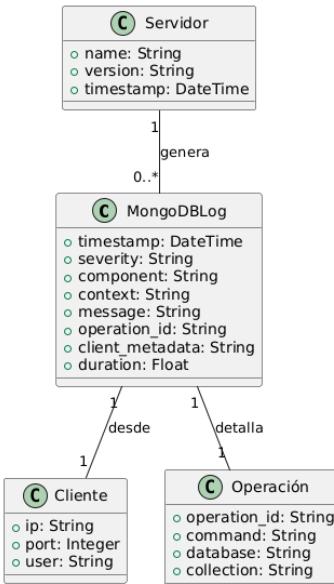


Figura 5.12: Modelo de datos de los logs de una base de datos MongoDB

6. Implementación

Durante la implementación, se ha seguido la planificación y metodologías anteriormente descritas, dividiendo el proyecto en tareas más pequeñas y manejables, para que se puedan realizar en un periodo de tiempo razonable.

Al seguir la prioridad de las tareas, se realizan primero las tareas más críticas, como la creación de la infraestructura y la ingesta de las fuentes esenciales, y se dejan para más adelante tareas como la visualización para clientes externos o fuentes menos críticas y más complejas, como las APIs de terceros o el *web scraping*.

6.1. Despliegue local

Para arrancar el desarrollo del proyecto, se decide construir una versión local del sistema, que permita trabajar en un entorno controlado y sin dependencias externas para comprobar su correcto funcionamiento y establecer las bases de configuración y del posterior despliegue en la nube.

Puesto que se ha decidido utilizar Docker para la gestión de contenedores, se ha creado un archivo `docker-compose.yml` que define los servicios necesarios para el proyecto, es decir, *Kafka*, *Zookeeper*, *Elasticsearch*, *Kibana* y *Logstash*, ignorando por el momento la ingesta de datos y la escalabilidad del sistema.

Para la configuración de los servicios, se ha creado un archivo `.env` que define las variables de entorno necesarias para el correcto funcionamiento de los servicios, como las contraseñas o la versión del *stack*.

Durante el arranque de los contenedores, se ejecuta un script de inicialización que se encarga de crear las credenciales y los usuarios necesarios para el funcionamiento de los servicios. Estas credenciales son necesarias ya que los contenedores funcionan mediante tráfico HTTPS.

Los contenedores cuentan con comprobaciones de salud (o *health-checks*) básicas para asegurar que los servicios se han arrancado correctamente y están funcionando.

Esta sección de la documentación documenta el desarrollo de la historia de usuario inicial, de acuerdo con lo establecido en la sección 4.2:

| Nombre | Prioridad | Tamaño |
|---|-----------|--------|
| Creación de la infraestructura base (técnica) | P0 | L |

Tabla 6.1: Lista de HUs cumplimentadas con el despliegue local

6.1.1. Explicación del código

El código de despliegue local se encuentra en *A Código de despliegue local.*

A nivel de configuración, se definen variables de entorno para cada contenedor mediante el uso de la palabra clave `environment` y las claves definidas por cada imagen. Para cada contenedor, se define además información adicional dependiendo de las características del servicio, como la dependencia en otras imágenes, los límites de recursos o los puertos de escucha.

Para evitar el ruido excesivo por consola una vez arrancado los servicios, se reduce el nivel de *logging* a `WARN` en los servicios que lo soporten.

Al comienzo del archivo `docker-compose.yml`, se definen los volúmenes y las redes necesarias para el correcto funcionamiento de los servicios.

```
1 volumes:
2   es01data:
3   kibanadata:
4   elasticdata:
5   logstashdata:
6   kafkaadata:
7   certs:
8
9 networks:
10  default:
11    driver: bridge
```

Listado 6.1: Definición de volúmenes y redes en Docker Compose

A continuación, se definen los servicios necesarios para el proyecto, comenzando por el contenedor de preparación de credenciales y usuarios. Se utiliza una imagen de Elasticsearch para la creación de las credenciales, y se monta un volumen para la persistencia de las mismas.

```
1 setup:
2   image:
3     ↳ docker.elastic.co/elasticsearch/elasticsearch:${STACK_VERSION}
4   volumes:
5     - certs:/usr/share/elasticsearch/config/certs
6   user: root
7   container_name: setup
8   command: >
9     bash -c '
```

```

10      echo "Set the ELASTIC_PASSWORD environment variable in the .env
11      ↪ file";
12      exit 1;
13      elif [ x${KIBANA_PASSWORD} == x ]; then
14          echo "Set the KIBANA_PASSWORD environment variable in the .env
15          ↪ file";
16          exit 1;
17          fi;
18          if [ ! -f config/certs/ca.zip ]; then
19              echo "Creating CA";
20              bin/elasticsearch-certutil ca --silent --pem -out
21              ↪ config/certs/ca.zip;
22              unzip config/certs/ca.zip -d config/certs;
23              fi;
24              if [ ! -f config/certs/certs.zip ]; then
25                  echo "Creating certs";
26                  echo -ne \
27                  "instances:\n\
28                  " - name: es01\n\
29                  "     dns:\n\
30                  "         - es01\n\
31                  "         - localhost\n\
32                  "     ip:\n\
33                  "         - 127.0.0.1\n\
34                  " - name: kibana\n\
35                  "     dns:\n\
36                  "         - kibana\n\
37                  "         - localhost\n\
38                  "     ip:\n\
39                  "         - 127.0.0.1\n\
40                  > config/certs/instances.yml;
41                  bin/elasticsearch-certutil cert --silent --pem -out
42                  ↪ config/certs/certs.zip --in config/certs/instances.yml
43                  ↪ --ca-cert config/certs/ca/ca.crt --ca-key
44                  ↪ config/certs/ca/ca.key;
45                  unzip config/certs/certs.zip -d config/certs;
46                  fi;
47                  echo "Setting file permissions"
48                  chown -R root:root config/certs;
49                  find . -type d -exec chmod 750 \{\}\ \;;
50                  find . -type f -exec chmod 640 \{\}\ \;;
51                  echo "Waiting for Elasticsearch availability";
52                  until curl -s --cacert config/certs/ca/ca.crt https://es01:9200
53                  ↪ | grep -q "missing authentication credentials"; do sleep 1;
54                  ↪ done;
55                  echo "Setting kibana_system password";

```

```

48     until curl -s -X POST --cacert config/certs/ca/ca.crt -u
49     ↳ "elastic:${ELASTIC_PASSWORD}" -H "Content-Type:
50     ↳ application/json"
51     ↳ https://es01:9200/_security/user/kibana_system/_password -d
52     ↳ "{\"password\":\"${KIBANA_PASSWORD}\")\"}" | grep -q "^{}"; do
53     ↳ sleep 10; done;
54     echo "All done!";
55
56   '
57
58 healthcheck:
59   test: [ "CMD-SHELL", "[ -f config/certs/es01/es01.crt ]" ]
60   interval: 5s
61   timeout: 10s
62   retries: 10

```

Listado 6.2: Definición del servicio de preparación

El servicio de preparación necesita tener una comprobación de salud, puesto que el resto de contenedores lo tienen marcado como dependencia para su arranque. En este caso, la comprobación consiste en la existencia de un archivo de certificado.

Una vez definido el servicio de preparación, se definen los servicios de *Kafka* y *Zookeeper*, que se basan en imágenes oficiales de *Confluent*.

```

1  zookeeper:
2    container_name: zookeeper
3    image: confluentinc/cp-zookeeper:latest
4    environment:
5      ZOOKEEPER_CLIENT_PORT: 2181
6      ZOOKEEPER_TICK_TIME: 2000
7      ZOO_LOG4J_PROP: WARN,CONSOLE
8    ports:
9      - 2181:2181
10
11 kafka:
12   container_name: kafka
13   image: confluentinc/cp-kafka:latest
14   depends_on:
15     - zookeeper
16     - es01
17   ports:
18     - 9092:9092
19     - 29092:29092
20   environment:
21     KAFKA_BROKER_ID: 1
22     KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181

```

```

23    KAFKA_ADVERTISED_LISTENERS:
24      ↳ LISTENER_DOCKER_INTERNAL://kafka:29092,LISTENER_DOCKER_EXTERNAL://localhost
25      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
26      ↳ LISTENER_DOCKER_INTERNAL:PLAINTEXT,LISTENER_DOCKER_EXTERNAL:PLAINTEXT
27      KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_DOCKER_INTERNAL
28      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
29      KAFKA_LOG4J_ROOT_LOGLEVEL: WARN
      KAFKA_TOOLS_LOG4J_LOGLEVEL: ERROR
      KAFKA_LOG4J_LOGGERS:
      ↳ 'kafka=WARN,kafka.controller=WARN,kafka.log.LogCleaner=WARN,state.change.log=WARN'

```

Listado 6.3: Definición de los servicios de Kafka

Ambos servicios cuentan con una serie de variables de entorno que definen su configuración, como el puerto de escucha, el *broker ID* o la dirección de *Zookeeper*.

Una vez definidos los servicios de Kafka, se define el servicio más crítico, el contenedor de Elasticsearch, del que depende el funcionamiento de todo el sistema.

```

1 es01:
2   image:
3     ↳ docker.elastic.co/elasticsearch/elasticsearch:${STACK_VERSION}
4   container_name: es01
5   restart: unless-stopped
6   depends_on:
7     - setup
8   environment:
9     - node.name=es01
10    - cluster.name=${CLUSTER_NAME}
11    - discovery.type=single-node
12    - bootstrap.memory_lock=true
13    - logger.level=WARN
14    - ELASTIC_PASSWORD=${ELASTIC_PASSWORD}
15    - xpack.security.enabled=true
16    - xpack.security.http.ssl.enabled=true
17    - xpack.security.http.ssl.key=certs/es01/es01.key
18    - xpack.security.http.ssl.certificate=certs/es01/es01.crt
19    -
20    ↳ xpack.security.http.ssl.certificateAuthorities=certs/ca/ca.crt
21    - xpack.security.transport.ssl.enabled=true
22    - xpack.security.transport.ssl.key=certs/es01/es01.key
23    - xpack.security.transport.ssl.certificate=certs/es01/es01.crt
24    -
25    ↳ xpack.security.transport.ssl.certificateAuthorities=certs/ca/ca.crt
26    - xpack.security.transport.ssl.verifyMode=certificate
27
28 ulimits:

```

```

25     memlock:
26         soft: -1
27         hard: -1
28     nofile:
29         soft: 65536
30         hard: 65536
31     cap_add:
32         - IPC_LOCK
33     labels:
34         co.elastic.logs/module: elasticsearch
35         co.elastic.metrics/module: elasticsearch
36     volumes:
37         - es01data:/usr/share/elasticsearch/data
38         - certs:/usr/share/elasticsearch/config/certs
39     ports:
40         - 9200:9200
41     healthcheck:
42         test:
43             [
44                 "CMD-SHELL",
45                 "curl -s --cacert config/certs/ca/ca.crt
46             ↳ https://localhost:9200 | grep -q 'missing authentication
47             ↳ credentials'"
48             ]
49         interval: 10s
50         timeout: 10s
51         retries: 10

```

Listado 6.4: Definición del servicio de Elasticsearch

Debido a que se trata del servicio más importante y grande, se requieren muchas opciones de configuración, como la limitación de recursos, la persistencia de datos o la configuración de seguridad. Como en el resto de servicios, se define una comprobación de salud que se encarga de comprobar que el servicio está disponible.

Para simplificar lo máximo posible la arquitectura de este prototipo, tan solo se define un nodo de Elasticsearch, aunque la configuración de escalabilidad sería sencilla gracias al diseño de Docker.

Por último, se definen los servicios de Kibana y Logstash, que dependen de Elastic.

```

1 kibana:
2     container_name: kibana
3     image: docker.elastic.co/kibana/kibana:${STACK_VERSION}
4     restart: unless-stopped

```

```

5   volumes:
6     - kibanadata:/usr/share/kibana/data
7     - certs:/usr/share/kibana/config/certs
8   environment:
9     SERVER_NAME: kibana
10    SERVER_PORT: 5601
11    SERVER_HOST: 0.0.0.0
12    ELASTICSEARCH_HOSTS: https://es01:9200
13    ELASTICSEARCH_USERNAME: kibana_system
14    ELASTICSEARCH_PASSWORD: ${KIBANA_PASSWORD}
15    ELASTICSEARCH_SSL_CERTIFICATEAUTHORITIES: config/certs/ca/ca.crt
16    LOGGING_ROOT_LEVEL: warn
17    XPACK_ENCRYPTEDSAVEDOBJECTS_ENCRYPTIONKEY: ${KEY}
18    XPACK_REPORTING_ENCRYPTIONKEY: ${KEY}
19    XPACK_SECURITY_ENCRYPTIONKEY: ${KEY}
20   links:
21     - es01
22   depends_on:
23     - setup
24     - es01
25   labels:
26     co.elastic.logs/module: kibana
27     co.elastic.metrics/module: kibana
28   ports:
29     - 5601:5601
30   healthcheck:
31     test:
32       [
33         "CMD-SHELL",
34         "curl -s -I http://localhost:5601 | grep -q 'HTTP/1.1 302"
35       ↳ Found!
36       ]
37     interval: 10s
38     timeout: 10s
39     retries: 10

```

Listado 6.5: Definición de los servicios de Kibana

```

1 logstash:
2   container_name: logstash
3   image: docker.elastic.co/logstash/logstash:${STACK_VERSION}
4   restart: unless-stopped
5   user: root
6   volumes:
7     - logstashdata:/usr/share/logstash/data
8     - certs:/usr/share/logstash/certs
9   environment:

```

```
10      - ELASTIC_HOSTS="https://es01:9200"
11      - ELASTIC_USER="elastic"
12      - ELASTIC_PASSWORD=${ELASTIC_PASSWORD}
13      - log.level=warn
14      - xpack.monitoring.enabled=false
15      command: >
16          bash -c "echo 'input { kafka { bootstrap_servers =>
17              \"kafka:29092\" topics => [\"laravel-logs\"] } } output {
18                  elasticsearch { hosts => [\"https://es01:9200\"] user =>
19                      \"elastic\" password => \"${ELASTIC_PASSWORD}\" ssl => true
20                      cacert => \"certs/ca/ca.crt\" } }' >
21          /usr/share/logstash/pipeline/logstash.conf; bin/logstash -f
22          /usr/share/logstash/pipeline/logstash.conf"
23      ports:
24          - 5000:5000
25      depends_on:
26          - es01
27          - kafka
28          - setup
29      labels:
30          co.elastic.logs/module: logstash
31          co.elastic.metrics/module: logstash
32      links:
33          - es01
34          - kibana
```

Listado 6.6: Definición de los servicios de Logstash

Se hace uso de la red interna de Docker para facilitar la comunicación entre los contenedores, y se definen comprobaciones de salud para asegurar que los servicios se han arrancado correctamente. Para Logstash, se define un script de lanzamiento que genera el archivo de configuración y ejecuta el servicio, aunque podría montarse un volumen con el archivo de configuración ya generado.

6.1.2. Uso del sistema

Una vez arrancados los contenedores mediante el comando docker compose up, se puede acceder a los servicios a través de la dirección local localhost. En el caso de Kibana, se puede acceder a la interfaz de usuario mediante la dirección localhost:5601. En el caso de Elasticsearch, se pueden hacer peticiones HTTPS a través de la dirección localhost:9200. Para Kafka, Zookeeper y Logstash, se pueden hacer peticiones a través de las direcciones localhost:9092, localhost:2181 y localhost:9600, respectivamente.

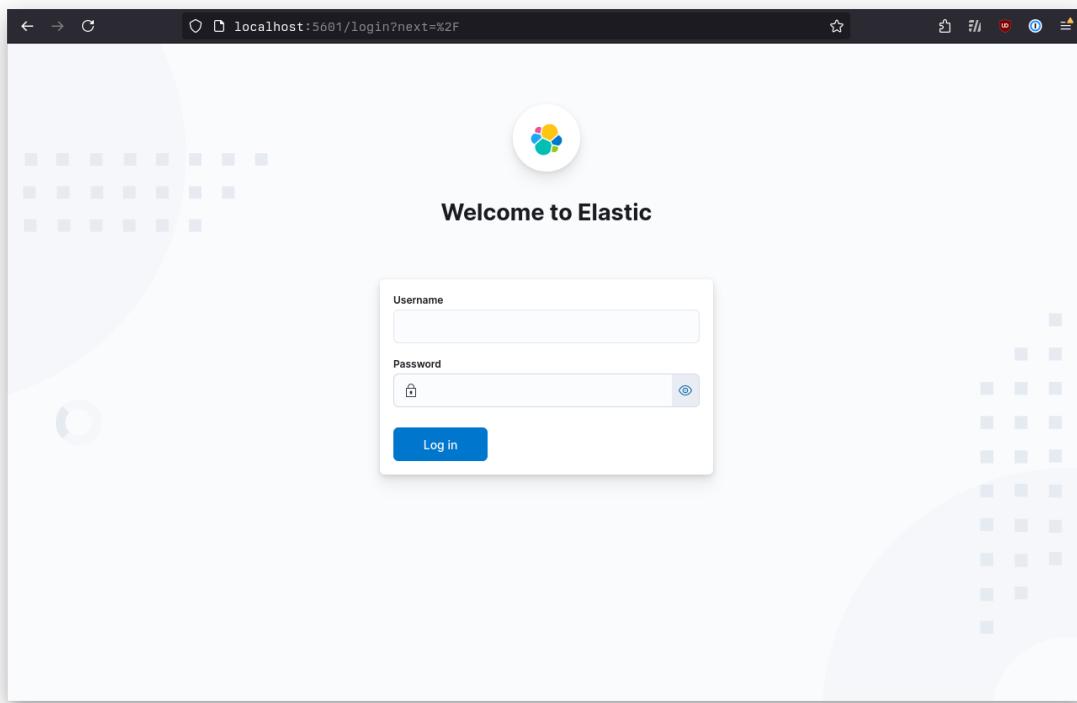


Figura 6.1: Inicio de sesión en Kibana

Una vez en la pantalla de inicio de sesión, se puede acceder con las credenciales definidas en el archivo .env para el usuario elastic.

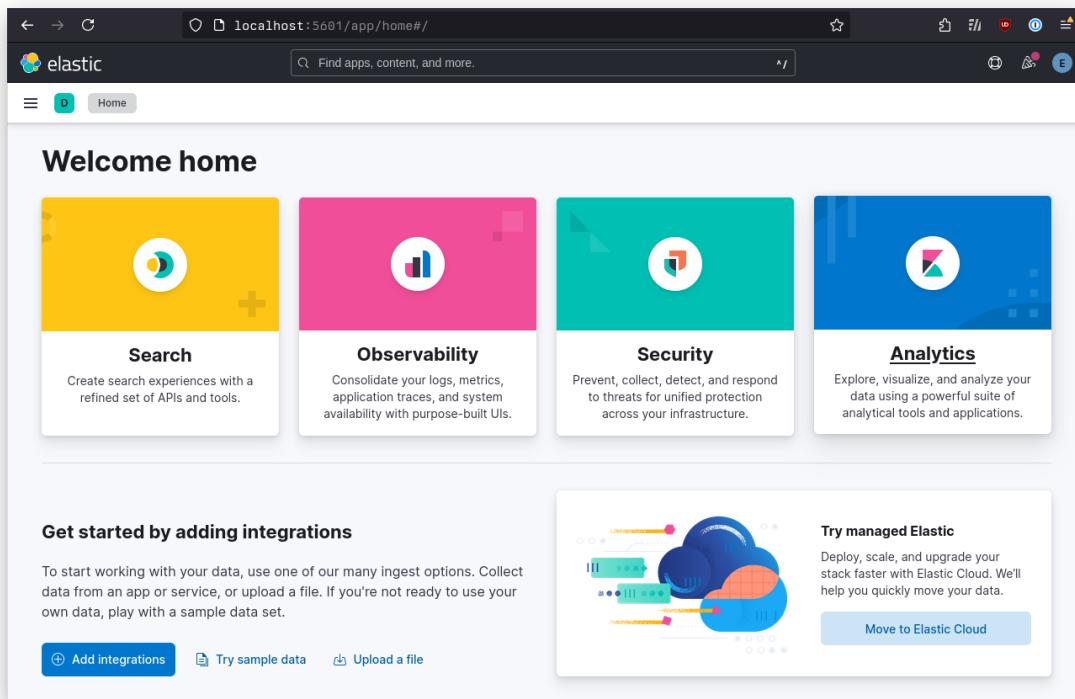


Figura 6.2: Página de inicio de Kibana

Una vez aquí, se puede hacer click en la opción Try sample data para cargar un conjunto de datos de ejemplo y probar la funcionalidad de Kibana con Elasticsearch. Por supuesto, también se pueden probar la ingestión de datos a través de Logstash y Kafka o, si así se desea, a través de los *Beats* especializados de ingestión directa de Elastic.

El manual de uso del sistema se encuentra en *7.1 Manual de usuario*.

6.1.3. Proceso de desarrollo

Para el desarrollo del sistema, se ha seguido un proceso iterativo, comenzando a partir del ejemplo oficial de Elastic para *Docker Compose*¹. A partir de este ejemplo, se añaden progresivamente configuraciones y servicios, y se prueban las funcionalidades de cada uno de ellos, actualizando con regularidad el código en el repositorio privado establecido.

| | | | | |
|---|--|-----------|-------------------------|--|
| • |  | Juan Mier | 8d5e5b7 | remove config files and embed configs in yaml |
| • |  | Juan Mier | 57a2b37 | fixes, getting ready for deploy |
| • |  | Juan Mier | 1d2e050 | init terraform |
| • |  | Juan Mier | 698754b | sample k8s files |
| • |  | Juan Mier | 331d187 | guardar certificados, reducir logging level |
| • |  | Juan Mier | 687052e | configure kafka |
| • |  | Juan Mier | 6bff095 | refinar docker-compose, setup container, https/ssl/certs |

Figura 6.3: Ejemplo de commits en el repositorio privado

Inicialmente, se prueban los servicios mínimos (Kibana y Elasticsearch) sin HTTPS para comprobar su correcto funcionamiento. Una vez se ha comprobado que los servicios funcionan correctamente, se añade la configuración de seguridad y se prueban los servicios con HTTPS, ajustando más configuraciones como el nivel de registro de logs o las comprobaciones de salud de los contenedores.

Una vez se ha comprobado que los servicios funcionan correctamente, se añaden los servicios de Kafka, Zookeeper y Logstash, y se prueban las conexiones entre los servicios, ajustando las configuraciones necesarias para que los servicios se comuniquen correctamente.

¹<https://www.elastic.co/blog/getting-started-with-the-elastic-stack-and-docker-compose>

6.2. Despliegue *cloud*

El desarrollo principal del despliegue en la nube se concentra en la creación de los scripts de *Terraform* necesarios para la implementación de la infraestructura planteada en el apartado 5.2 *Arquitectura del sistema*. Para ello, se divide el proyecto en scripts separados de manera que se puedan gestionar los recursos y los servicios de manera independiente.

El diseño de una infraestructura base y el desarrollo de un prototipo de manera local permiten tener una idea clara de los recursos necesarios y de las características específicas de cada servicio, facilitando la tarea de desarrollo.

Para el desarrollo, se hace uso de un repositorio privado en *Bitbucket* para el control de versiones y facilitar a la empresa la revisión y uso del código. El código completo se encuentra en *C Scripts de despliegue cloud*.

6.2.1. Proceso de desarrollo

El proceso de desarrollo de los scripts de *Terraform* parte de la implementación original de la infraestructura en local, y se va adaptando a las necesidades de la infraestructura en la nube, puesto que ambos comparten similaridades (como la mayoría de la configuración de los servicios, la estructura general de los mismos, las imágenes y versiones utilizadas, etc.).

Al igual que con el desarrollo local, se sigue un proceso iterativo, comenzando por la creación de un solo servicio, en este caso Kafka, y continuando con el resto de la arquitectura. Los primeros despliegues son tan solo pruebas de concepto, con el objetivo de adaptarse a la infraestructura de la nube, el funcionamiento de *Terraform* y la configuración de AWS.

Pese a que *Terraform* suele encargarse de la creación, modificación y destrucción de los recursos de manera automática, existen casos en los que es necesaria la intervención manual, como en la destrucción de los contenedores de *Secret Manager* o en la actualización de algunas configuraciones de los recursos. Estos casos ocurrirán solo durante la fase de desarrollo, puesto que se espera que, en la fase de producción, no sea necesario la reconfiguración de los recursos y servicios.

La definición de las tareas de ECS durante el desarrollo queda registrado en la sección correspondiente de AWS, cuyo código y configuraciones se puede consultar si así se desea.

The screenshot shows a list of task definitions for the 'elastic' service. There are 64 revisions listed, all of which are marked as 'INACTIVE'. The list includes entries like 'elastic:64', 'elastic:63', 'elastic:62', etc., up to 'elastic:58'. The interface includes a search bar, a filter for status (set to 'Inactive'), and buttons for Deploy and Actions.

| Task definition: revision | Status |
|---------------------------|----------|
| elastic:64 | INACTIVE |
| elastic:63 | INACTIVE |
| elastic:62 | INACTIVE |
| elastic:61 | INACTIVE |
| elastic:60 | INACTIVE |
| elastic:59 | INACTIVE |
| elastic:58 | INACTIVE |

Figura 6.4: Ejemplo de definiciones de tareas de ECS en AWS

The screenshot shows the details for a task definition named 'kafka:4'. It includes fields for ARN, Status (INACTIVE), Time created (July 02, 2024 at 11:57 (UTC+2:00)), App environment (FARGATE), Task role (ecsTaskExecutionRole), Task execution role (ecsTaskExecutionRole), Operating system/Architecture (-), and Network mode (awsvpc). Below this, there is a JSON tab showing the task definition configuration, which includes container definitions for a 'zookeeper' container with port mappings for port 2181.

```

1  {
2    "taskDefinitionArn": "arn:aws:ecs:eu-north-1:340917389686:task-definition/kafka:4",
3    "containerDefinitions": [
4      {
5        "name": "zookeeper",
6        "image": "confluentinc/cp-zookeeper:latest",
7        "cpu": 512,
8        "memory": 512,
9        "portMappings": [
10          {
11            "containerPort": 2181,
12            "hostPort": 2181,
13            "protocol": "tcp"
14          }
        ]
      }
    ]
  }

```

Figura 6.5: Ejemplo de definición de tarea (Kafka)

6.2.2. Despliegue de la infraestructura

6.2.3. Explicación del código

Los scripts de Terraform se dividen en varios archivos, cada uno de ellos con una función específica, con el objetivo de facilitar la gestión y configuración de los recursos y servicios. A continuación, se detallan dichos archivos y su función en el proyecto.

6.3. Ingesta de datos

Tras la creación y el despliegue de la infraestructura base, se procede a la creación de los scripts de ingestión de datos, de manera escalonada y siguiendo la prioridad de las fuentes de datos.

Puesto que se ha decidido utilizar Kafka como sistema de mensajería, se desarrollan los scripts de ingestión de datos para que los datos se envíen a Kafka y se procesen mediante Logstash y Elasticsearch.

Para la ingestión de datos, se han desarrollado scripts de Python que se encargan de la lectura de los datos de las fuentes, su transformación y su envío a Kafka. Estos scripts se ejecutan mediante un *cron* que se encarga de la ejecución periódica de los mismos.

6.4. Visualización de datos

Una vez se cuentan con datos en Elasticsearch, se puede comenzar el desarrollo de la visualización de los mismos mediante Kibana. Para ello, se han desarrollado paneles de visualización que permiten la monitorización de los datos en tiempo real y la creación de informes y *dashboards* personalizados para cada modelo de datos contemplado.

7. Manuales

7.1. Manual de usuario

7.2. Manual de despliegue

El sistema se ha diseñado para llevar la menor cantidad de pasos posible en su instalación. Debido a la colección de tecnologías utilizadas, (Terraform, Docker y ECS), el despliegue de la infraestructura base requiere la ejecución de tan solo dos comandos¹.

El sistema de despliegue está pensado para ser ejecutado en un sistema operativo UNIX, como Linux o macOS. Para su ejecución en Windows, se recomienda el uso de WSL2².

7.2.1. Requisitos previos

Previa al despliegue del sistema, se deben cumplir una serie de requisitos mínimos:

Terraform El proceso de instalación de Terraform es sencillo y se puede realizar siguiendo las instrucciones de la aplicación en su página web oficial³ para el sistema operativo que se desee. Una vez descargado e instalado, se puede comprobar que la instalación ha sido correcta ejecutando el comando `terraform -v` en la terminal de comandos del sistema.

```
mier@tnkpd okt-data-lake/tf 🐀 main ✘ % terraform -v
Terraform v1.9.1
on linux_amd64
+ provider registry.terraform.io/hashicorp/aws v5.56.1
+ provider registry.terraform.io/hashicorp/random v3.6.2
```

Figura 7.1: Comprobación de la versión de Terraform

¹Asumiendo que se cumplen los requisitos previos.

²<https://learn.microsoft.com/es-es/windows/wsl/about>

³<https://developer.hashicorp.com/terraform/install>

Usuario de AWS Para poder ejecutar los comandos de Terraform con los permisos necesarios, se debe disponer de un usuario de AWS con los permisos necesarios para la creación de los recursos. Se recomienda la creación de un usuario específico para el despliegue del sistema, con los roles y políticas estrictamente necesarias, para evitar posibles problemas de seguridad.

Para permitir la interacción con la consola de AWS, se debe instalar la herramienta de terminal de Amazon⁴.

Una vez creado o seleccionado el usuario deseado, se debe iniciar sesión en la consola a través del comando `aws configure` y seguir las instrucciones para introducir las credenciales de acceso al sistema.

```
mier@tnkpd okt-data-lake/tf ✘ main ✘ % aws configure
AWS Access Key ID [*****N3ER]:
AWS Secret Access Key [*****8oCb]:
Default region name [eu-west-3]:
Default output format [None]:
```

Figura 7.2: Configuración de credenciales en AWS CLI

AWS pone a disposición de los usuarios una guía de inicio rápido⁵ para la configuración de las credenciales.

```
mier@tnkpd okt-data-lake/tf ✘ main ✘ % aws sts get-caller-identity
{
    "UserId": "",
    "Account": "",
    "Arn": "arn:aws:iam:::user/juan.mier@okticket.es"
}
```

Figura 7.3: Demostración de credenciales en AWS CLI

⁴<https://aws.amazon.com/es/cli/>

⁵https://docs.aws.amazon.com/es_es/cli/latest/userguide/cli-configure-quickstart.html

Certificados Para que funcionen correctamente los servicios de la aplicación mediante HTTPS, se deben disponer de certificados separados tanto para el dominio y los recursos de AWS como para los servicios de la aplicación.

Los certificados de AWS se generan automáticamente con el script mediante el servicio de ACM, mientras que los certificados de la aplicación se deben generar previamente y almacenar en la carpeta certs del proyecto.

Para la generación de los certificados de la aplicación, se reutiliza los scripts de preparación de los contenedores del *6.1 Despliegue local*. Para la generación de los certificados de la aplicación, se debe ejecutar el script certs.sh que se encuentra en *B Script de creación de certificados* o bien se pueden reutilizar los certificados generados en el despliegue local.

Estos certificados deben ser almacenados en la carpeta certs junto a los scripts de despliegue de Terraform, y deben tener visibilidad suficiente para que Terraform pueda acceder a ellos.

7.2.2. Despliegue

Una vez cumplidos los requisitos previos, se puede proceder al despliegue de la infraestructura base. Para ello, se deben ejecutar los siguientes comandos desde la carpeta de scripts de Terraform a través de la terminal del sistema:

```
1 terraform init # Necesario solo la primera ejecucion  
2 terraform apply
```

```
mier@tnkpd okt-data-lake/tf [main] % terraform init && terraform apply -auto-approve  
Initializing the backend...  
Initializing provider plugins...
```

Figura 7.4: Despliegue de la infraestructura base

El comando terraform init se encarga de inicializar el directorio de trabajo de Terraform, descargando los módulos necesarios para el despliegue. El comando terraform apply se encarga de desplegar la infraestructura base en la cuenta de AWS asociada a las credenciales configuradas en la herramienta de AWS CLI.

Una vez ejecutado el comando, se mostrará un resumen de los recursos que se van a crear y se pedirá confirmación para proceder con el despliegue. Para confirmar, se debe introducir yes y pulsar la tecla Enter. Una vez confirmado, Terraform procederá a la creación de los recursos de manera automatizada.

Una vez concluido el despliegue, se mostrará un mensaje de éxito y se devolverán algunos datos relevantes sobre los recursos creados, como direcciones URL, identificadores o contraseñas de acceso creadas de manera aleatoria.

```
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```

Outputs:

```
hash = "6gpp"
passwords = "e:           , k:           , h:           "
url_elastic = "
url_kafka = "
url_kibana = "
```

Figura 7.5: Despliegue exitoso de la infraestructura base

8. Resultados y trabajo futuro

El propósito de este capítulo es presentar las conclusiones obtenidas a partir del desarrollo del proyecto, recopilar las dificultades encontradas y proponer líneas de trabajo futuro en vista a la ampliación y mejora del sistema.

A. Código de despliegue local

```
1 version: '3'

2

3 volumes:
4   es01data:
5   kibanadata:
6   elasticdata:
7   logstashdata:
8   kafkadata:
9   certs:

10

11 networks:
12   default:
13     driver: bridge

14

15 services:
16   setup:
17     image:
18       ↪ docker.elastic.co/elasticsearch/elasticsearch:${STACK_VERSION}
19     volumes:
20       - certs:/usr/share/elasticsearch/config/certs
21     user: root
22     container_name: setup
23     command: >
24       bash -c '
25         if [ x${ELASTIC_PASSWORD} == x ]; then
26           echo "Set the ELASTIC_PASSWORD environment variable in the
27           ↪ .env file";
28           exit 1;
29         elif [ x${KIBANA_PASSWORD} == x ]; then
30           echo "Set the KIBANA_PASSWORD environment variable in the
31           ↪ .env file";
32           exit 1;
33         fi;
34         if [ ! -f config/certs/ca.zip ]; then
35           echo "Creating CA";
36           bin/elasticsearch-certutil ca --silent --pem -out
37           ↪ config/certs/ca.zip;
38           unzip config/certs/ca.zip -d config/certs;
39         fi;
40         if [ ! -f config/certs/certs.zip ]; then
41           echo "Creating certs";
42           echo -ne \
43             "instances:\n"\`
```

```

40         " - name: es01\n" \
41         "   dns:\n" \
42         "     - es01\n" \
43         "     - localhost\n" \
44         "   ip:\n" \
45         "     - 127.0.0.1\n" \
46         " - name: kibana\n" \
47         "   dns:\n" \
48         "     - kibana\n" \
49         "     - localhost\n" \
50         "   ip:\n" \
51         "     - 127.0.0.1\n" \
52       > config/certs/instances.yml;
53       bin/elasticsearch-certutil cert --silent --pem -out
54   ↳ config/certs/certs.zip --in config/certs/instances.yml
55   ↳ --ca-cert config/certs/ca/ca.crt --ca-key
56   ↳ config/certs/ca/ca.key;
57       unzip config/certs/certs.zip -d config/certs;
58   fi;
59       echo "Setting file permissions"
60       chown -R root:root config/certs;
61       find . -type d -exec chmod 750 \{\} \;;
62       find . -type f -exec chmod 640 \{\} \;;
63       echo "Waiting for Elasticsearch availability";
64       until curl -s --cacert config/certs/ca/ca.crt
65   ↳ https://es01:9200 | grep -q "missing authentication
66   ↳ credentials"; do sleep 1; done;
67       echo "Setting kibana_system password";
68       until curl -s -X POST --cacert config/certs/ca/ca.crt -u
69   ↳ "elastic:${ELASTIC_PASSWORD}" -H "Content-Type:
70   ↳ application/json"
71   ↳ https://es01:9200/_security/user/kibana_system/_password -d
72   ↳ "{\"password\":\"${KIBANA_PASSWORD}\"]}" | grep -q "^{}"; do
73   ↳ sleep 10; done;
74       echo "All done!";
75   '''
76
77   healthcheck:
78     test: [ "CMD-SHELL", "[ -f config/certs/es01/es01.crt ]" ]
79     interval: 5s
80     timeout: 10s
81     retries: 10
82
83   zookeeper:
84     container_name: zookeeper
85     image: confluentinc/cp-zookeeper:latest
86     environment:

```

```

76     ZOOKEEPER_CLIENT_PORT: 2181
77     ZOOKEEPER_TICK_TIME: 2000
78     ZOO_LOG4J_PROP: WARN,CONSOLE
79   ports:
80     - 2181:2181
81
82   kafka:
83     container_name: kafka
84     image: confluentinc/cp-kafka:latest
85     depends_on:
86       - zookeeper
87       - es01
88     ports:
89       - 9092:9092
90       - 29092:29092
91     environment:
92       KAFKA_BROKER_ID: 1
93       KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
94       KAFKA_ADVERTISED_LISTENERS:
95         ↪ LISTENER_DOCKER_INTERNAL://kafka:29092,LISTENER_DOCKER_EXTERNAL://localhost
96         KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
97         ↪ LISTENER_DOCKER_INTERNAL:PLAINTEXT,LISTENER_DOCKER_EXTERNAL:PLAINTEXT
98         KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_DOCKER_INTERNAL
99         KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
100        KAFKA_LOG4J_ROOT_LOGLEVEL: WARN
101        KAFKA_TOOLS_LOG4J_LOGLEVEL: ERROR
102        KAFKA_LOG4J_LOGGERS:
103          ↪ 'kafka=WARN,kafka.controller=WARN,kafka.log.LogCleaner=WARN,state.change.lo
104
105   es01:
106     image:
107       ↪ docker.elastic.co/elasticsearch/elasticsearch:${STACK_VERSION}
108     container_name: es01
109     restart: unless-stopped
110     depends_on:
111       - setup
112     environment:
113       - node.name=es01
114       - cluster.name=${CLUSTER_NAME}
115       - discovery.type=single-node
116       - bootstrap.memory_lock=true
117       - logger.level=WARN
118       - ELASTIC_PASSWORD=${ELASTIC_PASSWORD}
119       - xpack.security.enabled=true
120       - xpack.security.http.ssl.enabled=true
121       - xpack.security.http.ssl.key=certs/es01/es01.key

```

```

118      - xpakc.security.http.ssl.certificate=certs/es01/es01.crt
119
120      -
121
122      ↳ xpakc.security.http.ssl.certificateAuthorities=certs/ca/ca.crt
123          - xpakc.security.transport.ssl.enabled=true
124          - xpakc.security.transport.ssl.key=certs/es01/es01.key
125          - xpakc.security.transport.ssl.certificate=certs/es01/es01.crt
126          -
127
128      ↳ xpakc.security.transport.ssl.certificateAuthorities=certs/ca/ca.crt
129          - xpakc.security.transport.ssl.verificationMode=certificate
130
131      ulimits:
132          memlock:
133              soft: -1
134              hard: -1
135
136          nofile:
137              soft: 65536
138              hard: 65536
139
140      cap_add:
141          - IPC_LOCK
142
143      labels:
144          co.elastic.logs/module: elasticsearch
145          co.elastic.metrics/module: elasticsearch
146
147      volumes:
148          - elastic-data:/usr/share/elasticsearch/data
149          - certs:/usr/share/elasticsearch/config/certs
150
151      ports:
152          - 9200:9200
153
154      healthcheck:
155          test:
156              [
157                  "CMD-SHELL",
158                  "curl -s --cacert config/certs/ca/ca.crt"
159                  ↳ https://localhost:9200 | grep -q 'missing authentication'
160                  ↳ credentials"
161              ]
162          interval: 10s
163          timeout: 10s
164          retries: 10
165
166
167      kibana:
168          container_name: kibana
169          image: docker.elastic.co/kibana/kibana:${STACK_VERSION}
170          restart: unless-stopped
171
172          volumes:
173              - kibana-data:/usr/share/kibana/data
174              - certs:/usr/share/kibana/config/certs
175
176          environment:
```

```

160     SERVER_NAME: kibana
161     SERVER_PORT: 5601
162     SERVER_HOST: 0.0.0.0
163     ELASTICSEARCH_HOSTS: https://es01:9200
164     ELASTICSEARCH_USERNAME: kibana_system
165     ELASTICSEARCH_PASSWORD: ${KIBANA_PASSWORD}
166     ELASTICSEARCH_SSL_CERTIFICATEAUTHORITIES: config/certs/ca/ca.crt
167     LOGGING_ROOT_LEVEL: warn
168     XPACK_ENCRYPTEDSAVEDOBJECTS_ENCRYPTIONKEY: ${KEY}
169     XPACK_REPORTING_ENCRYPTIONKEY: ${KEY}
170     XPACK_SECURITY_ENCRYPTIONKEY: ${KEY}
171     links:
172       - es01
173     depends_on:
174       - setup
175       - es01
176     labels:
177       co.elastic.logs/module: kibana
178       co.elastic.metrics/module: kibana
179     ports:
180       - 5601:5601
181     healthcheck:
182       test:
183         [
184           "CMD-SHELL",
185           "curl -s -I http://localhost:5601 | grep -q 'HTTP/1.1 302"
186         ↪ Found'"
187         ]
188       interval: 10s
189       timeout: 10s
190       retries: 10
191     logstash:
192       container_name: logstash
193       image: docker.elastic.co/logstash/logstash:${STACK_VERSION}
194       restart: unless-stopped
195       user: root
196       volumes:
197         - logstash-data:/usr/share/logstash/data
198         - certs:/usr/share/logstash/certs
199       environment:
200         - ELASTIC_HOSTS="https://es01:9200"
201         - ELASTIC_USER="elastic"
202         - ELASTIC_PASSWORD=${ELASTIC_PASSWORD}
203         - log.level=warn
204         - xpack.monitoring.enabled=false

```

```

205   command: >
206     bash -c "echo 'input { kafka { bootstrap_servers =>
207       \"kafka:29092\" topics => [\"laravel-logs\"] } } output {
208       elasticsearch { hosts => [\"https://es01:9200\"] user =>
209         \"elastic\" password => \">${ELASTIC_PASSWORD}\" ssl => true
210         cacert => \"certs/ca/ca.crt\" } }' >
211       /usr/share/logstash/pipeline/logstash.conf; bin/logstash -f
212       /usr/share/logstash/pipeline/logstash.conf"
213
214   ports:
215     - 5000:5000
216
217   depends_on:
218     - es01
219     - kafka
220     - setup
221
222   labels:
223     co.elastic.logs/module: logstash
224     co.elastic.metrics/module: logstash
225
226   links:
227     - es01
228     - kibana

```

Listado A.1: Fichero de despliegue local

```

1 # .env
2 ELASTIC_PASSWORD=changeme
3 KIBANA_PASSWORD=changeme
4 STACK_VERSION=8.12.2
5 CLUSTER_NAME=okt-tahoe-cluster
6 PROJECT_NAME=okt-tahoe
7 KEY=something_at_least_32_characters_long

```

Listado A.2: Fichero de ejemplo de variables de entorno

B. Script de creación de certificados

```
1 #!/bin/bash
2 echo "Creando certificado de autoridad";
3 bin/elasticsearch-certutil ca --silent --pem -out config/certs/ca.zip;
4 unzip config/certs/ca.zip -d config/certs;
5
6 echo "Creando certificados";
7 echo -ne \
8     "instances:\n" \
9     "  - name: es01\n" \
10    "    dns:\n" \
11      "      - es01\n" \
12      "      - localhost\n" \
13      "      ip:\n" \
14        "        - 127.0.0.1\n" \
15      "      - name: kibana\n" \
16      "        dns:\n" \
17        "          - kibana\n" \
18        "          - localhost\n" \
19        "        ip:\n" \
20          "          - 127.0.0.1\n" \
21 > config/certs/instances.yml;
22 bin/elasticsearch-certutil cert --silent --pem -out config/certs/certs
23   ↪ .zip --in config/certs/instances.yml --ca-cert config/certs/ca/
24   ↪ ca.crt --ca-key config/certs/ca/ca.key;
25 unzip config/certs/certs.zip -d config/certs;
```

Listado B.1: Script de creación de credenciales

C. Scripts de despliegue cloud

Bibliografía

- [1] Wikipedia contributors, “Dikw pyramid — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/w/index.php?title=DIKW_pyramid&oldid=1211227190, 2024. [Online; accessed 7-April-2024].
- [2] Ishwarappa and J. Anuradha, “A brief introduction on big data 5vs characteristics and hadoop technology,” *Procedia Computer Science*, vol. 48, pp. 319–324, 2015. International Conference on Computer, Communication and Convergence (ICCC 2015).
- [3] S. Sagiroglu and D. Sinanc, “Big data: A review,” in *2013 International Conference on Collaboration Technologies and Systems (CTS)*, pp. 42–47, 2013.
- [4] J. Mier, “Presentación de datos: dashboards y procesos ETL.” Primera entrega de teoría de la asignatura Inteligencia de Negocio, EPI Gijón, curso 23-24, 2023.
- [5] P. Khine and Z. Wang, “Data lake: a new ideology in big data era,” *ITM Web of Conferences*, vol. 17, p. 03025, 01 2018.
- [6] E. A. Mezmir, “Qualitative data analysis: An overview of data reduction, data display, and interpretation,” *Research on humanities and social sciences*, vol. 10, no. 21, pp. 15–27, 2020.
- [7] D. A. Lelewer and D. S. Hirschberg, “Data compression,” *ACM Computing Surveys (CSUR)*, vol. 19, no. 3, pp. 261–296, 1987.
- [8] B. Beyer, C. Jones, J. Petoff, and N. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly, 2016.
- [9] J. Mier, “latexTemplate.” <https://github.com/miermontoto/latexTemplate>, 2024. Plantilla de L^AT_EX personal para trabajos académicos.
- [10] J. Mier, “Minería de anomalías.” Segunda entrega de teoría de la asignatura Inteligencia de Negocio, EPI Gijón, curso 23-24, 2024.