



ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

GRADO EN INGENIERÍA INFORMÁTICA EN TECNOLOGÍAS DE LA INFORMACIÓN

Lenguajes y Sistemas Informáticos

EXPLOTACIÓN, INTEGRACIÓN Y VISUALIZACIÓN DE MÚLTIPLES FUENTES DE DATOS MEDIANTE UN DATA LAKE

AUTOR

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	5
Índice de tablas	7
Índice de listados	8
1. Introducción	12
1.1. Antecedentes	12
1.2. Motivación	14
1.3. La empresa	15
1.4. Objetivo y alcance	16
2. Fundamento teórico	17
2.1. <i>Big data</i>	17
2.2. Paradigmas de almacenamiento de datos	19
2.2.1. Data warehouse	19
2.2.2. Data lake	19
2.2.3. Data lakehouse	20
2.3. Procesos ETL	21
2.3.1. Funcionamiento	22
2.3.2. Alternativas	24
2.4. Cuadros de mandos (<i>dashboards</i>)	25
2.5. Infraestructura como código	26
3. Descripción general del proyecto	27
3.1. Partes interesadas (<i>stakeholders</i>)	28
3.2. Alternativas existentes	29
3.2.1. Criterios de evaluación	29
3.2.2. Proceso de selección	29
3.2.3. Alternativas consideradas	30
3.2.4. Conclusiones	36
3.3. Descripción del proyecto	37
3.3.1. Roles y usuarios	37
3.3.2. Dashboards planteados	38
3.3.3. Proceso de despliegue	38
3.3.4. Características del sistema	39
4. Planificación del proyecto	40
4.1. Metodología	40
4.1.1. Scrum	41
4.1.2. Visualización	43
4.1.3. Comunicación	44
4.1.4. Herramientas	44
4.2. Planificación inicial	45

4.3. Presupuesto	47
4.3.1. Presupuesto de material	47
4.3.2. Presupuesto de personal	49
4.3.3. Presupuesto total	50
5. Análisis del sistema	51
5.1. Epics	51
5.2. Historias de usuario	52
5.2.1. Epic 1: Infraestructura y despliegue	52
5.2.2. Epic 2: Ingesta de datos	53
5.2.3. Epic 3: Procesamiento y almacenamiento de datos	54
5.2.4. Epic 4: Visualización y análisis	55
5.3. Story Mapping	56
6. Diseño del sistema	57
6.1. Estudio de alternativas	57
6.1.1. Despliegue de infraestructura	58
6.1.2. Ingesta de datos	60
6.1.3. Proveedor de nube	63
6.1.4. Sistemas de virtualización y servicios	65
6.1.5. Tipos de máquinas	66
6.2. Arquitectura del sistema	67
6.2.1. Servicios y componentes de AWS	67
6.2.2. Infraestructura	71
6.2.3. Seguridad	72
6.2.4. Redes	73
6.3. Modelos de datos	74
6.3.1. Logs de balanceadores (AWS)	75
6.3.2. Logs de bases de datos SQL	76
6.3.3. Logs de bases de datos MongoDB	77
6.3.4. Logs de Laravel (API)	78
7. Implementación	79
7.1. Despliegue local	81
7.1.1. Explicación del código	82
7.1.2. Uso del sistema	88
7.1.3. Proceso de desarrollo	90
7.2. Despliegue <i>cloud</i>	91
7.2.1. Proceso de desarrollo	92
7.2.2. Despliegue de la infraestructura	96
7.2.3. Explicación del código	98
7.3. Ingesta de datos	107
7.3.1. Ingesta de métricas con Kafka	108
7.3.2. Ingesta de logs de AWS	110
7.3.3. Otros métodos de ingesta	111
7.4. Visualización de datos	112
7.4.1. Desarrollo de dashboards	113
7.4.2. Ejemplos de dashboards	114

8. Pruebas	115
8.1. Plan de pruebas	116
8.1.1. Pruebas de sistema	117
8.1.2. Pruebas de carga	120
8.2. Ejecución de pruebas	122
8.2.1. Ejemplo de ejecución	123
8.2.2. Tabla de resultados	124
9. Manuales	125
9.1. Manual de usuario	125
9.1.1. Usuario normal	126
9.1.2. Usuario administrador	127
9.2. Manual de despliegue	128
9.2.1. Requisitos previos	128
9.2.2. Despliegue	131
10. Resultados y trabajo futuro	132
10.1. Resultados	132
10.2. Trabajo futuro	133
10.2.1. Historias de usuario restantes	133
10.2.2. Integración de lenguaje natural para búsqueda (DSL)	134
10.2.3. Aplicación de modelos de Lenguaje de Gran Escala (LLM)	134
10.2.4. Más perspectivas futuras	134
10.3. Conclusiones y retrospectiva	135
Anexos	136
A. Código de despliegue local	137
B. Script de creación de certificados	144
C. Scripts de despliegue cloud	145
C.1. Recursos de AWS	145
C.1.1. Fichero principal	145
C.1.2. Variables	147
C.1.3. Salidas	149
C.1.4. Volúmenes lógicos (EFS)	150
C.1.5. Roles, usuarios y políticas (IAM)	152
C.1.6. Secretos (Secret Manager)	154
C.1.7. Recursos de red	155
C.1.8. Grupos de seguridad (SG)	158
C.2. Servicios de ELK	160
C.2.1. Elasticsearch	160
C.2.2. Kibana	167
C.2.3. Logstash	173
C.2.4. Kafka	177

D. Scripts de ingestá de datos con Kafka	182
D.1. Ingestá de logs de Laravel	182
D.2. Ingestá de logs de MongoDB	184
D.3. Ingestá de logs de MySQL	186
E. Código de ingestá de logs de ELB (AWS)	188
E.1. Lambda de ingestá	188
E.2. Filtro de suscripción	189
E.3. Configuración de Logstash	189
E.4. Creación del índice	190
Bibliografía	191

Índice de figuras

2.1. Fases de un proceso ETL	22
2.2. Ejemplo de flujo con virtualización	24
2.3. Diagrama de flujo de un proceso <i>ELT</i>	24
3.1. Logo de Elasticsearch ®	30
3.2. Logo de Logstash ®	30
3.3. Logo de Kibana ®	30
3.4. Logo de Fluentd ®	31
3.5. Logo de Graylog ®	32
3.6. Logo de Prometheus ®	32
3.7. Logo de Loki ®	33
3.8. Logo de Loggly ®	35
3.9. Logo de Splunk ®	35
4.1. Diagrama de la metodología <i>Scrum</i>	41
4.2. Tablero <i>Kanban</i> del proyecto	43
4.3. Roadmap de apartados de la memoria	44
4.4. Planificación inicial del proyecto	45
5.1. Diagrama <i>Story Mapping</i> del proyecto	56
6.1. Logo de Terraform ®	58
6.2. Logo de AWS CloudFormation ®	58
6.3. Logo de Ansible ®	58
6.4. Logo de Kafka ®	60
6.5. Cuadrante mágico de Gartner para proveedores de nube	63
6.6. Logo de Amazon EC2 ®	66
6.7. Logo de AWS Fargate ®	66
6.8. Diagrama inicial de la infraestructura en AWS	71
6.9. Diagrama incial de la seguridad en AWS	72
6.10. Diagrama incial de las redes en AWS	73
6.11. Modelo de datos de los logs de un balanceador de carga de AWS	75
6.12. Modelo de datos de los logs de una base de datos SQL	76
6.13. Modelo de datos de los logs de una base de datos MongoDB	77
6.14. Modelo de datos de los logs de una aplicación Laravel	78
7.1. Diagrama Burndown que representa el progreso del proyecto	79
7.2. Inicio de sesión en Kibana	88
7.3. Página de inicio de Kibana	89
7.4. Ejemplo de commits en el repositorio privado	90
7.5. Ejemplo de definciones de tareas de ECS en AWS	92
7.6. Ejemplo de definición de tarea (Kafka)	93
7.7. Ejemplo de logs de una tarea de ECS	93
7.8. Estado de los balanceadores de carga	94
7.9. Métricas de estado del despliegue de un servicio	94
7.10. Estado de Kibana	95
7.11. Diagrama de despliegue de la infraestructura	97

7.12. Proceso de ingestión de logs de balanceadores de carga de AWS	110
7.13. Dashboard de estado general de infraestructura	114
7.14. Dashboard de métricas detalladas	114
8.1. Análisis estático de código con SonarQube	119
9.1. Pantalla de inicio de sesión de Kibana	125
9.2. Listado de dashboards de Kibana	126
9.3. Ejemplo de dashboard de Kibana	126
9.4. Opciones de fuentes de datos de Elasticsearch a través de Kibana	127
9.5. Ejemplo de alertas y métricas en Kibana	127
9.6. Comprobación de la versión de Terraform	128
9.7. Configuración de credenciales en AWS CLI	129
9.8. Demostración de credenciales en AWS CLI	129
9.9. Despliegue de la infraestructura base	131
9.10. Despliegue exitoso de la infraestructura base	131

Índice de tablas

4.1.	Historias de usuario iniciales	46
4.2.	Propuesta de presupuesto mensual de materiales (región eu-north-1)	48
4.3.	Propuesta de presupuesto de personal para el proyecto	49
4.4.	Costes combinados de presupuesto y materiales con beneficio industrial	50
7.1.	Lista de HUs cumplimentadas con el despliegue local	81
7.2.	Lista de HUs cumplimentadas con el despliegue en la nube	91
7.3.	Lista de HUs cumplimentadas con la ingesta de datos	107
7.4.	Lista de HUs cumplimentadas con la visualización de datos	112
8.1.	Ejecución de pruebas del sistema	124
10.1.	Historias de usuario restantes para futuras iteraciones	133

Índice de listados

7.1.	Definición de volúmenes y redes en Docker Compose	82
7.2.	Definición del servicio de preparación	83
7.3.	Definición de los servicios de Kafka	83
7.4.	Definición del servicio de Elasticsearch	84
7.5.	Definición de los servicios de Kibana	86
7.6.	Definición de los servicios de Logstash	87
7.7.	Definción de la tarea de Elastic	100
7.8.	Definción del servicio de Elastic	103
7.9.	Definción de recursos de Elastic	104
A.1.	Fichero de despliegue local	137
A.2.	Fichero de ejemplo de variables de entorno	141
A.3.	Fichero de configuración de Logstash	141
A.4.	Script shell de preparación de Elasticsearch	141
B.1.	Script de creación de credenciales	144
C.1.	Fichero <i>main.tf</i> de despliegue cloud	145
C.2.	Fichero <i>variables.tf</i> de despliegue cloud	147
C.3.	Fichero <i>outputs.tf</i> de despliegue cloud	149
C.4.	Fichero <i>efs.tf</i> de despliegue cloud	150
C.5.	Fichero <i>iam.tf</i> de despliegue cloud	152
C.6.	Fichero <i>secrets.tf</i> de despliegue cloud	154
C.7.	Fichero <i>network.tf</i> de despliegue cloud	155
C.8.	Fichero <i>secrets.tf</i> de despliegue cloud	158
C.9.	Fichero <i>elastic.tf</i> de despliegue cloud	160
C.10.	Fichero <i>kibana.tf</i> de despliegue cloud	167
C.11.	Fichero <i>logstash.tf</i> de despliegue cloud	173
C.12.	Fichero <i>kafka.tf</i> de despliegue cloud	177
D.1.	Script de ingestá de logs de Laravel	182
D.2.	Fichero <i>.env</i> de configuración de ingestá de logs de Laravel	183
D.3.	Script de ingestá de logs de MongoDB	184
D.4.	Fichero <i>.env</i> de configuración de ingestá de logs de MongoDB	185
D.5.	Script de ingestá de logs de MySQL	186
D.6.	Fichero <i>.env</i> de configuración de ingestá de logs de MySQL	187
E.1.	Lambda de ingestá de logs de ELB	188
E.2.	Comando para añadir el filtro de suscripción	189
E.3.	Configuración de Logstash para ingestá de logs de ELB	189
E.4.	Estructura del índice de Elasticsearch	190

Declaración de autoría

Yo, Juan Francisco Mier Montoto, con DNI 71777658V, declaro que este documento titulado EXPLORACIÓN, INTEGRACIÓN Y VISUALIZACIÓN DE MÚLTIPLES FUENTES DE DATOS MEDIANTE UN DATA LAKE y el trabajo presentado en él son de mi propiedad. Afirmo que:

- Este trabajo fue realizado completa y totalmente durante mi estancia en el Grado en Ingeniería Informática en Tecnologías de la Información en la Escuela Politécnica de Ingeniería de Gijón.
- Aquellas partes de este documento que hayan sido previamente publicadas se encuentran debidamente indicadas.
- Aquellas partes de este documento que se apoyen en trabajos previamente publicados se encuentran debidamente referenciadas.
- Todas las fuentes utilizadas para la realización de este documento han sido debidamente citadas.
- Durante el desarrollo de este trabajo, se han utilizado correctores generativos basados en inteligencia artificial (GPT-4o, Claude 3.5 Sonet) para la generación de texto, pero siempre bajo la supervisión y posterior revisión del autor. El autor toma la responsabilidad de cualquier error que pueda haber sido introducido durante el proceso de revisión.

Firmado: Juan Francisco Mier Montoto
Fecha: 18 de julio de 2024

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), 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. Objetivo y alcance

El objetivo del proyecto es la creación de un proceso que permita el despliegue automático de una infraestructura 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.

La plataforma que se desarrolle debe de ser capaz, además de manejar los datos comentados anteriormente, de ofrecer una interfaz visual de consulta y análisis de los mismos, para que los usuarios puedan explotar la información de forma sencilla y rápida.

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:

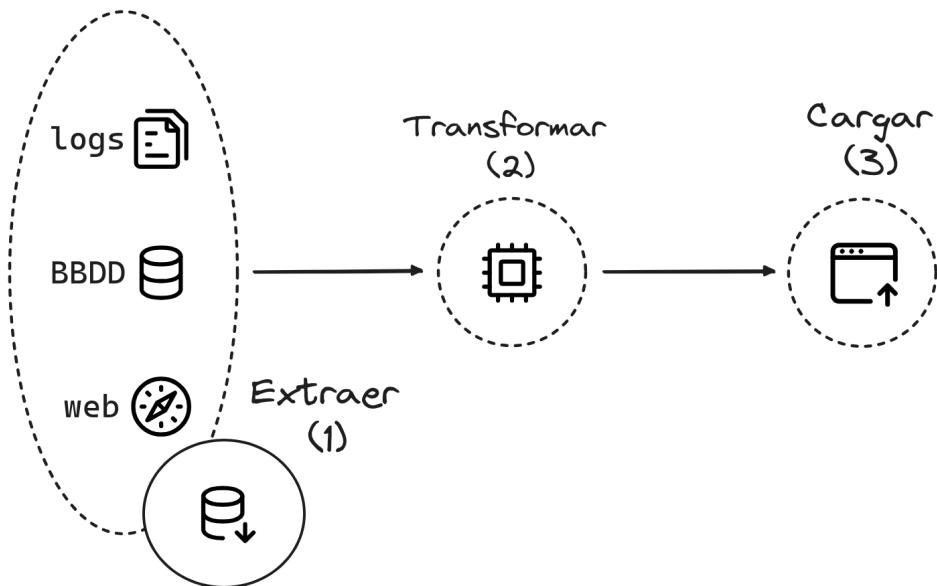


Figura 2.1: Fases de un proceso ETL

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.

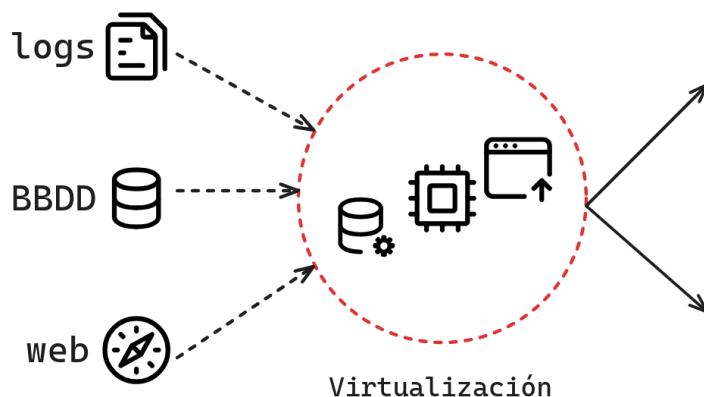


Figura 2.2: Ejemplo de flujo con virtualización

- **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*.

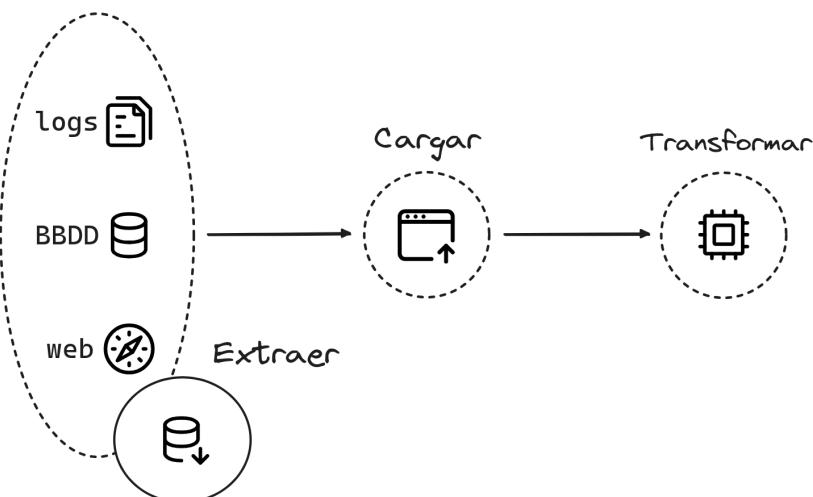


Figura 2.3: Diagrama de flujo de un proceso ELT

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.

El objetivo es describir el proyecto de manera general, sin entrar en detalles técnicos. Para ello, se describen los problemas que se pretenden resolver, las partes interesadas en el proyecto y las alternativas consideradas antes de comenzar el desarrollo del mismo. El estudio de tecnologías, proveedores y servicios se realiza en secciones posteriores (ver *6 Diseño del sistema*).

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 ingesta, 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, al obligar a utilizar un tipo de contenedores en específico.

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 datos de API o fuentes web.



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

El proyecto consiste en el desarrollo de una arquitectura de análisis de datos que permite obtener insights valiosos a partir de la información recopilada a partir de la extracción, transformación y carga de datos de diversas fuentes.

Además, el despliegue de la infraestructura se debe realizar de manera automática en la nube, de manera que se facilite la gestión sencilla y eficiente del sistema para los administradores.

El sistema es capaz de ingestar datos de diversas fuentes, como bases de datos internas, logs de aplicaciones y servicios externos, y de presentarlos de forma clara y útil a través de dashboards personalizados.

Los usuarios finales podrán acceder a los dashboards a través de un servicio web y visualizar los datos en tiempo real, así como filtrar la información mostrada según diferentes campos.

3.3.1. Roles y usuarios

En el proyecto se distinguen los siguientes roles:

- **Arquitecto:** responsable de diseñar la arquitectura de la plataforma y definir las tecnologías a utilizar.
- **Administrador/Desarrollador (DevOps):** responsable de desplegar y orquestar la infraestructura de manera automática.
- **Analista:** responsable de analizar los datos y sacar conclusiones aplicables al negocio a partir de los mismos.

A parte de estos roles, también se distinguen los siguientes usuarios:

- **Usuario interno:** empleado de la empresa que utiliza los dashboards para monitorizar el rendimiento de la plataforma.
- **Usuario externo:** cliente de la empresa que utiliza los dashboards para tomar decisiones informadas sobre su negocio.

3.3.2. 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.

3.3.3. Proceso de despliegue

El proceso de despliegue de la infraestructura se realizará de manera automática en la nube, utilizando herramientas de orquestación como Terraform y Ansible. Puesto que se trata de un *stack ELK*, se deberán desplegar los servicios, bien de manera separada mediante contenedores o clústeres, o bien de manera conjunta mediante una solución tradicional de máquinas virtuales.

El sistema se ha de poder desplegar rápida y sencillamente, de manera que se facilite la gestión y mantenimiento del mismo para los administradores en caso de necesitar actualizar, configurar o escalar los servicios.

Por requisitos de la empresa, la solución deberá estar desplegada en la nube, por lo que se deberá elegir un proveedor de servicios en la nube y desplegar la infraestructura en el mismo.

3.3.4. Características del sistema

El sistema debe cumplir con las siguientes características:

- **Escalabilidad:** el sistema debe ser capaz de escalar horizontalmente para soportar un gran volumen de datos.
- **Flexibilidad:** el sistema debe ser flexible y permitir la ingestión de datos de diversas fuentes.
- **Robustez:** el sistema debe ser robusto y tolerante a fallos, para garantizar la disponibilidad de los datos en todo momento.
- **Seguridad:** el sistema debe ser seguro y proteger la información sensible de los usuarios, al tratar información sensible de clientes de 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.

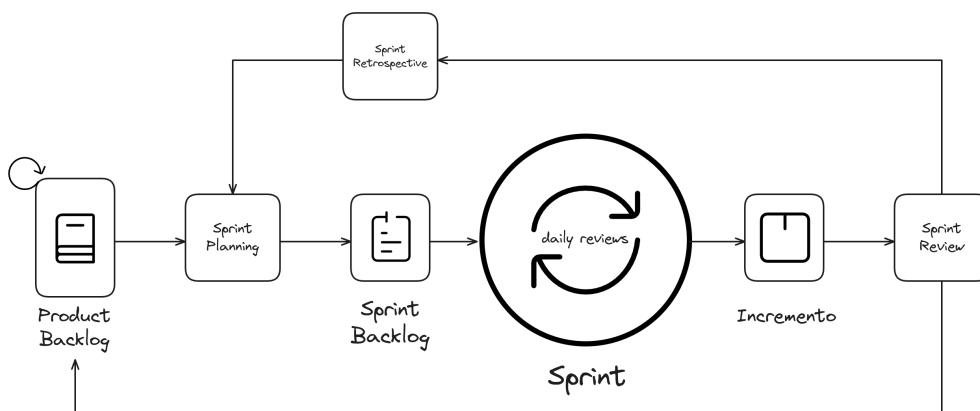


Figura 4.1: Diagrama de la metodología *Scrum*

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	S
Como trabajador de Okticket, quiero poder ver y consultar datos internos de la empresa	P1	M
Como desarrollador de Okticket, quiero que se ingesten de manera automática logs de balanceador de AWS	P1	M
Como desarrollador de Okticket, quiero poder ver el estado general de la infraestructura	P1	M
Como desarrollador de Okticket, quiero que los datos contengan metadatos que faciliten su filtrado o búsqueda	P2	XS
Como trabajador de Okticket, quiero poder ver y consultar datos de empresas cliente	P2	S
Como gestor de una empresa cliente, quiero poder ver información relevante sobre mi empresa que recoja Okticket	P2	M
Como desarrollador de Okticket, quiero poder ingestar datos de APIs externas a la empresa	P2	M
Como desarrollador de Okticket, quiero poder ingestar información de páginas web externas (<i>scraping</i>)	P2	S

Tabla 4.1: Historias de usuario iniciales

Los puntos de historia de usuario (PHU) se calculan aplicando la secuencia de Fibonacci como comentado anteriormente, donde $XS = 1$, $S = 2$, $M = 3$, $L = 5$ y $XL = 8$.

Debido a la estimación anterior, el valor total de los puntos de historia de usuario es de 41. El valor total de los puntos de historia necesarios para alcanzar el mínimo producto viable (MVP) es de 30, al no tener en cuenta las historias de usuario de prioridad 2.

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.

Se estima un horizonte de desarrollo de 3 meses, que es el tiempo estimado y disponible para el desarrollo del proyecto.

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 a junio de 2024 en la región de Amazon Web Services (AWS) de eu-north-1 (Estocolmo).

Categoría	Ítem	Cantidad	Coste unitario	Coste total
AWS Compute	Fargate	7168 vCPU-h/mes 18432 GB-h/mes	0,04928€/vCPU-h 0,00539€/GB-h	353,24€/mes 99,35€/mes
AWS Storage	EFS	512 GB/mes	0,38€/GB-mes	194,56€/mes
	S3	256 GB/mes	0,0255€/GB-mes	6,53€/mes
AWS Network	VPC	4 NAT Gateways	0,054€/h	155,52€/mes
	ELB	3 ALBs	0,012€/h	25,92€/mes
		1 NLB	0,027€/h	19,44€/mes
	Route 53	3 registros	0,50€/registro-mes	1,50€/mes
AWS Security	IAM	-	Sin cargo	0,00€
	KMS	1 CMK	1,00€/mes	1,00€/mes
	Secret Manager	12 secretos	0,40€/secreto-mes	4,80€/mes
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€/mes
	X-Ray	50,000 trazas/mes	4,60€/1M trazas	0,23€/mes
Despliegue	Terraform	-	Licencia gratuita	0,00€
Subtotal				863,59€/mes
Otros	Support	Plan Basic	Sin cargo	0,00€
	Optimización	-	5 % del subtotal	43,18€
	Contingencia	-	10 % del subtotal	86,36€
Total				993,13€/mes

Tabla 4.2: Propuesta de presupuesto mensual de materiales (región eu-north-1)

Asumiendo que el proyecto se desarrolla en la región de AWS de Estocolmo (eu-north-1), el coste total del material asciende a 993,13€ (novecientos noventa y tres euros con trece céntimos) al mes, que incluye el coste de los servicios en la nube, el coste de las licencias de software y el coste de las herramientas de desarrollo.

Suponiendo un horizonte de desarrollo de 3 meses, el coste total del material durante el desarrollo del proyecto asciende a 2.979,39€ (dos mil novecientos setenta y nueve euros con treinta y nueve céntimos).

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 totales	Coste unit.	Coste total
Arquitecto	Diseño de la arquitectura y supervisión	40 h	60 €/h	2.400,00 €
Desarrollador	Desarrollo y mantenimiento	160 h	45 €/h	7.200,00 €
Administrador	Gestión de sistemas y seguridad	160 h	50 €/h	8.000,00 €
DevOps	Infraestructuras y monitorización	80 h	55 €/h	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 para el proyecto

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.

El coste total del personal asciende a 27.720,00 € (veintisiete mil setecientos veinte euros), que incluye el coste de las horas de trabajo de cada rol, el IVA y el margen de beneficio industrial, todo ello a lo largo del horizonte de desarrollo establecido.

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, durante el horizonte de desarrollo establecido de 1 mes.

Concepto	Coste
Presupuesto de materiales	2.979,39 €
Presupuesto de personal	27.720,00 €
Subtotal	30.699,39 €
Beneficio industrial (15 %)	4.604,91 €
Total	35.304,30 €

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

El presupuesto total del proyecto asciende a 35.304,30 € (treinta y cinco mil trescientos cuatro euros con treinta céntimos), que incluye el coste del material, el coste del personal y el margen de beneficio industrial.

5. Análisis del sistema

En este capítulo se presenta un análisis detallado del sistema a desarrollar, desglosando las funcionalidades principales en epics y historias de usuario. Este enfoque permite una visión estructurada del proyecto, facilitando su planificación y desarrollo.

5.1. Epics

Los *epics* representan las grandes áreas funcionales del proyecto. Se han identificado las siguientes historias épicas:

1. Infraestructura y despliegue
2. Ingesta de datos
3. Procesamiento y almacenamiento de datos
4. Visualización y análisis

Cada uno de estos epics engloba un conjunto de funcionalidades relacionadas que, en conjunto, conforman el proyecto planteado.

5.2. Historias de usuario

Las historias de usuario describen las funcionalidades específicas desde la perspectiva del usuario final. A continuación, se detallan las historias de usuario para cada epic, incluyendo sus criterios de aceptación:

5.2.1. Epic 1: Infraestructura y despliegue

Este epic se centra en la creación y gestión de la infraestructura necesaria para el sistema.

- **HU1.1:** Como desarrollador de Okticket, quiero poder desplegar un prototipo del sistema en mi entorno local para facilitar el desarrollo y las pruebas iniciales.
 - El sistema se desplegará correctamente en un entorno local.
 - Todos los servicios estarán funcionando.
 - Se podrá acceder a las interfaces web de los servicios.
- **HU1.2:** Como desarrollador de Okticket, quiero que la arquitectura se despliegue y orqueste de manera automática en la nube para facilitar la gestión y el paso a producción del sistema.
 - El sistema se desplegará automáticamente con un solo comando.
 - Todos los recursos en la nube se crearán correctamente.
 - Los servicios serán accesibles a través de URLs públicas.
- **HU1.3:** Como administrador del sistema, quiero que la infraestructura sea capaz de escalar automáticamente en función de la demanda para optimizar el rendimiento y los costos.
 - Se habrán configurado políticas de auto-escalado para los servicios críticos.
 - El sistema aumentará los recursos cuando la carga aumente.
 - El sistema reducirá los recursos cuando la demanda disminuya.

5.2.2. Epic 2: Ingesta de datos

La ingestión de datos es fundamental para el funcionamiento del sistema, abarcando diversas fuentes de información.

- **HU2.1:** Como desarrollador de Okticket, quiero que se ingesten de manera automática datos de la base de datos interna de MongoDB para centralizar la información.
 - Los datos de MongoDB se ingestarán correctamente en el sistema.
 - La ingestión se realizará de forma periódica y automática.
 - Se podrá verificar la integridad de los datos ingestados.
- **HU2.2:** Como desarrollador de Okticket, quiero que se ingesten de manera automática datos de la base de datos interna de MySQL para tener una visión completa de los datos.
 - Los datos de MySQL se ingestarán correctamente en el sistema.
 - La ingestión se realizará de forma periódica y automática.
 - Se podrá verificar la integridad de los datos ingestados.
- **HU2.3:** Como desarrollador de Okticket, quiero que se ingesten de manera automática logs de balanceador de AWS para monitorear el rendimiento de la infraestructura.
 - Los logs del balanceador de AWS se ingestarán correctamente.
 - La ingestión se realizará en tiempo real o con un retraso mínimo.
 - Los logs ingestados incluirán toda la información relevante.
- **HU2.4:** Como desarrollador de Okticket, quiero poder ingestar datos de APIs externas a la empresa para enriquecer nuestros análisis.
 - Se podrán configurar conexiones a múltiples APIs externas.
 - Los datos de las APIs se ingestarán correctamente en el sistema.
 - La ingestión de APIs se realizará de forma programada o bajo demanda.
- **HU2.5:** Como desarrollador de Okticket, quiero poder ingestar información de páginas web externas (scraping) para obtener datos adicionales relevantes.
 - Se podrán configurar tareas de scraping para múltiples sitios web.
 - Los datos obtenidos por scraping se almacenarán correctamente.
 - El proceso de scraping respetará las políticas de los sitios web.

5.2.3. Epic 3: Procesamiento y almacenamiento de datos

Este epic se enfoca en la manipulación y organización eficiente de los datos ingresados.

- **HU3.1:** Como desarrollador de Okticket, quiero que los datos se limpien de manera automática para garantizar la calidad de la información.
 - Se implementarán procesos de limpieza de datos para cada fuente.
 - Los datos limpiados no contendrán valores nulos o incorrectos.
 - Se mantendrá un registro de las transformaciones aplicadas.
- **HU3.2:** Como desarrollador de Okticket, quiero que los datos contengan metadatos que faciliten su filtrado o búsqueda para mejorar la eficiencia en el análisis.
 - Cada registro de datos incluirá metadatos relevantes.
 - Los metadatos permitirán filtrar y buscar eficientemente.
 - Se podrán realizar búsquedas complejas utilizando los metadatos.

5.2.4. Epic 4: Visualización y análisis

La visualización y análisis de datos es crucial para extraer valor de la información reco-pilada.

- **HU4.1:** Como trabajador de Okticket, quiero poder ver y consultar datos internos de la empresa para tomar decisiones informadas.
 - Existirán paneles de control que mostrarán datos internos relevantes.
 - Los paneles se actualizarán en tiempo real o con una frecuencia adecuada.
 - Los usuarios podrán personalizar las visualizaciones según sus necesidades.
- **HU4.2:** Como desarrollador de Okticket, quiero poder ver el estado general de la infraestructura para monitorear su salud y rendimiento.
 - Existirá un panel que mostrará el estado de todos los servicios del sistema.
 - Se visualizarán métricas clave como CPU, memoria y uso de red.
 - El panel incluirá alertas visuales para problemas críticos.
- **HU4.3:** Como trabajador de Okticket, quiero poder ver y consultar datos de empresas cliente para ofrecer un mejor servicio y soporte.
 - Se podrán visualizar datos específicos de cada empresa cliente.
 - La información se presentará de forma clara y organizada.
 - Se podrán generar informes personalizados para cada cliente.
- **HU4.4:** Como gestor de una empresa cliente, quiero poder ver información relevante sobre mi empresa que recoja Okticket para optimizar mis procesos y tomar decisiones estratégicas.
 - Existirá un panel de control específico para cada empresa cliente.
 - Los datos se presentarán de forma comprensible para usuarios no técnicos.
 - Se incluirán métricas y KPIs relevantes para la gestión empresarial.

5.3. Story Mapping

El Story Mapping proporciona una visión de cómo las historias de usuario se traducen en tareas concretas de desarrollo. Esta estrategia permite una planificación más precisa y un seguimiento efectivo del progreso del proyecto.

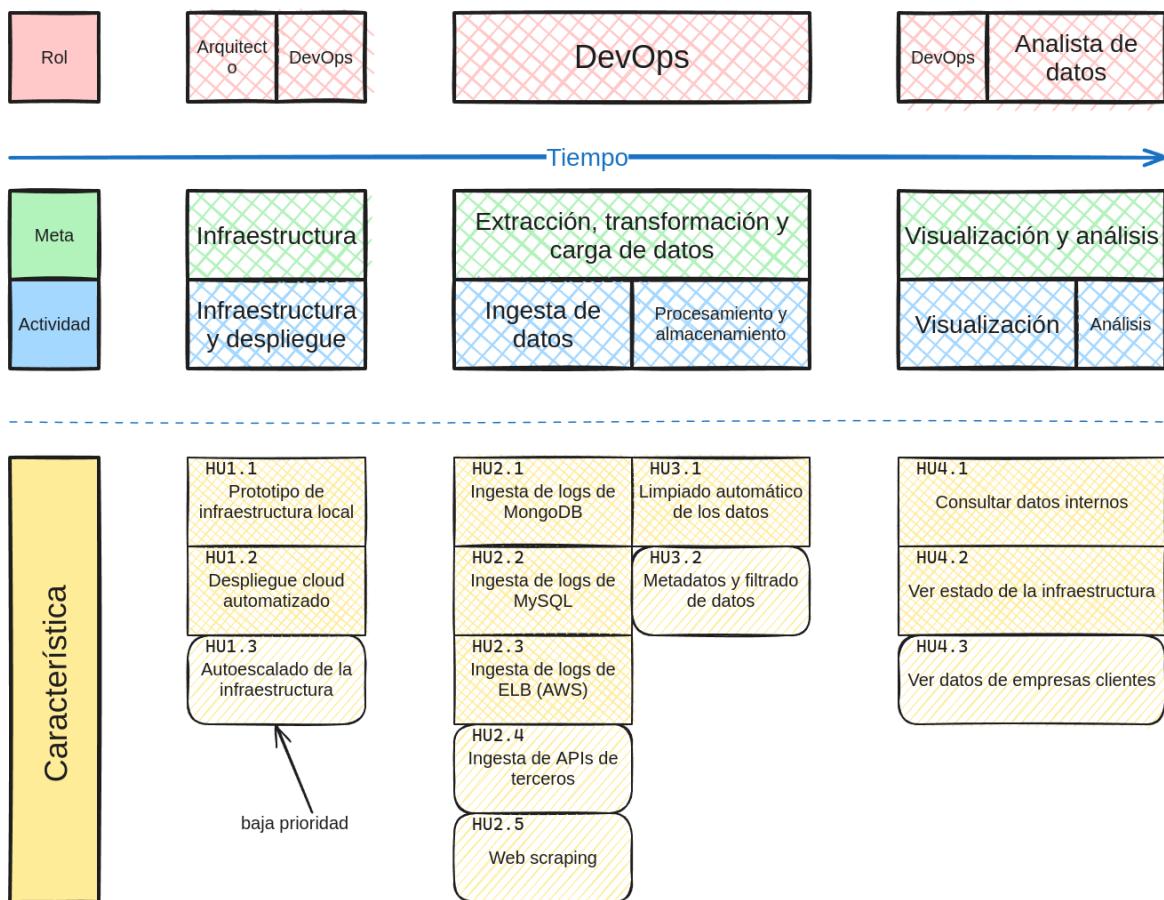


Figura 5.1: Diagrama *Story Mapping* del proyecto

El diagrama anterior muestra cómo las historias de usuario se organizan en epics y se desglosan en tareas específicas. Esta representación visual ayuda a comprender la estructura del proyecto y las dependencias entre las diferentes historias y tareas.

Este *story mapping* establece una hoja de ruta clara para el desarrollo del proyecto, asegurando que todas las historias de usuario se aborden de manera sistemática y eficiente.

6. 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.

6.1. 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 virtualización 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.

6.1.1. Despliegue de infraestructura

A la hora de desplegar la infraestructura de un proyecto, se consideran varias herramientas populares que permiten automatizar este proceso. Para este proyecto, se analizarán algunas de las más extendidas: *Terraform*, *AWS CloudFormation* y *Ansible*.

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 HCL, 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 6.1: 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 6.2: 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 6.3: 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, 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.

Como punto importante, Terraform es la única herramienta de las tres con la que se tiene experiencia previa dentro de la empresa, lo que facilitaría su adopción y uso en el proyecto.

Decisión Ninguna de las alternativas consideradas es funcionalmente superior a las demás, y se tratan de herramientas con características 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.

6.1.2. 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 la ingestión de datos de todas las fuentes que requieren ser procesadas.

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 integrada en 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 6.4: 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.

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 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*.

6.1.3. 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.

La compañía *Gartner* realiza un estudio anual sobre los proveedores de nube haciendo uso de una serie de criterios, como la *capacidad de ejecución* y la *visión completa*. En el último estudio, AWS lidera el mercado, seguido de Azure y el resto de proveedores.¹



Figura 6.5: Cuadrante mágico de Gartner para proveedores de nube

El resto de proveedores de nube, como *IBM Cloud*, *Oracle Cloud* o *Alibaba Cloud*, no se consideran en este estudio por su menor cuota de mercado y su menor presencia en la industria.

¹Enlace al estudio de Gartner

A continuación, se describen brevemente las características de cada uno de los tres proveedores de nube considerados:

- **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, gracias a su integración con flujos de *Office 365*, 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

6.1.4. Sistemas de virtualización 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 tipos de virtualización 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 requiere el mantenimiento de la máquina subyacente y la configuración e instalación del motor de contenedores si se desea utilizar este tipo de tecnologías.

Amazon ECS es un servicio de orquestación de contenedores que permite ejecutar y escalar contenedores de Docker en la nube de AWS. ECS está preparado para usar 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 en la nube. EKS 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 decide utilizar *Amazon ECS* como servicio de orquestación de contenedores, ya que es una opción más sencilla y rápida de desplegar que EKS y proporciona una interfaz intuitiva para gestionar contenedores en la nube de AWS. ECS está diseñado para trabajar con Docker y proporciona una solución completa para el despliegue y la gestión de contenedores en entornos de producción, lo que facilitará el desarrollo y la implementación de los servicios del sistema.

6.1.5. 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.

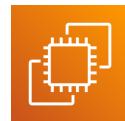


Figura 6.6: Logo de Amazon EC2 ®

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 6.7: 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 provee una capa de abstracción sobre la arquitectura subyacente, permitiendo centrarse el desarrollo de las aplicaciones sin tener que preocuparse por más gestiones.

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

⁴<https://aws.amazon.com/es/fargate/>

6.2. 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.

6.2.1. Servicios y componentes de AWS

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.

Componentes de seguridad

- **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.
- **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.

Componentes de red

- **VPC:** *Virtual Private Cloud* es un servicio que permite crear una red virtual privada en la nube de AWS. Se configurará una VPC para aislar los recursos del sistema y garantizar la seguridad de los datos.
- **Subnets:** Las *Subnets* son segmentos de red dentro de una VPC que permiten dividir la red en subredes más pequeñas. Se configurarán varias subredes para distribuir los recursos del sistema en diferentes zonas de disponibilidad.
- **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.
 - 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.
- **Route Tables:** Las *Route Tables* son tablas de rutas que permiten definir cómo se enruta el tráfico de red entre los recursos de AWS. Se configurarán varias tablas de rutas para garantizar la conectividad y la seguridad de los servicios del sistema.
- **Internet Gateway:** El *Internet Gateway* es un servicio que permite conectar una VPC a internet. Se configurará un Internet Gateway para permitir la comunicación entre los servicios del sistema y la red pública. Conceptualmente es equivalente a exponer un interfaz de red a internet.
- **NAT Gateway:** El *Network Address Translation Gateway* es un servicio que permite conectar una subred privada a internet. Se configurará un NAT Gateway para permitir la comunicación entre los servicios del sistema y la red pública.

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

Componentes de gestión

- **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.

Componentes de almacenamiento y monitorización

- **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 ingesta de datos no esté completamente implementada.

Componentes de ECS

- **Clústeres:** Los *clústeres* de ECS son grupos de servicios, tareas y contenedores que se ejecutan en una región de AWS. En el caso de este sistema, se configurará un clúster de ECS para agrupar los servicios del sistema.
- **Servicios:** Los *servicios* de ECS son conjuntos de tareas que se ejecutan en un clúster y que se encargan de ejecutar los contenedores del sistema.
- **Tareas:** Las *tareas* de ECS son instancias de contenedores que se ejecutan en un servicio y que se encargan de ejecutar el código de los servicios del sistema.
- **Contenedores:** Los *contenedores* de ECS son, en esencia, contenedores de Docker que contienen el código y las dependencias necesarias para ejecutar los servicios del sistema.

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.

6.2.2. 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 cada balanceador (*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.

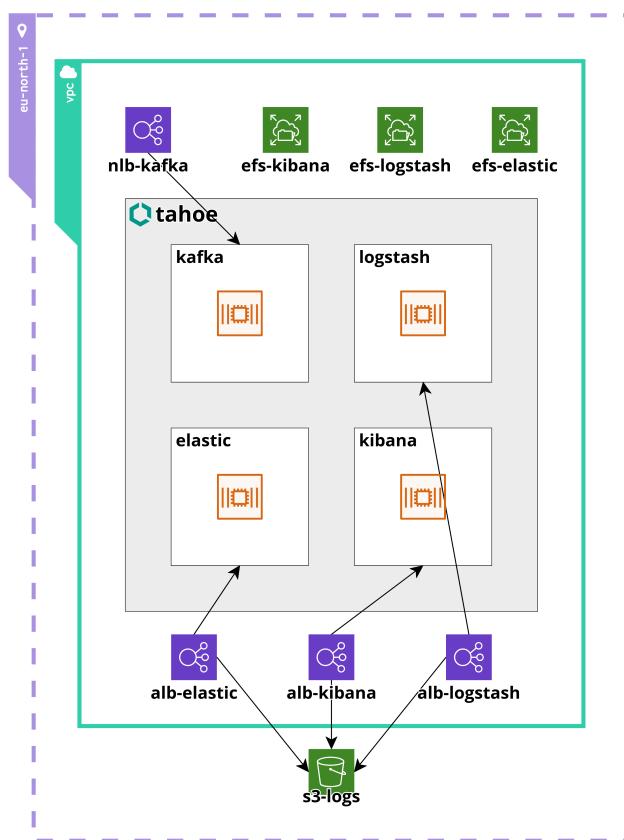


Figura 6.8: Diagrama inicial de la infraestructura en AWS

6.2.3. 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 establecidos. Por convenio, se minimizarán tanto las políticas como los roles necesarios para garantizar la seguridad de los datos.

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

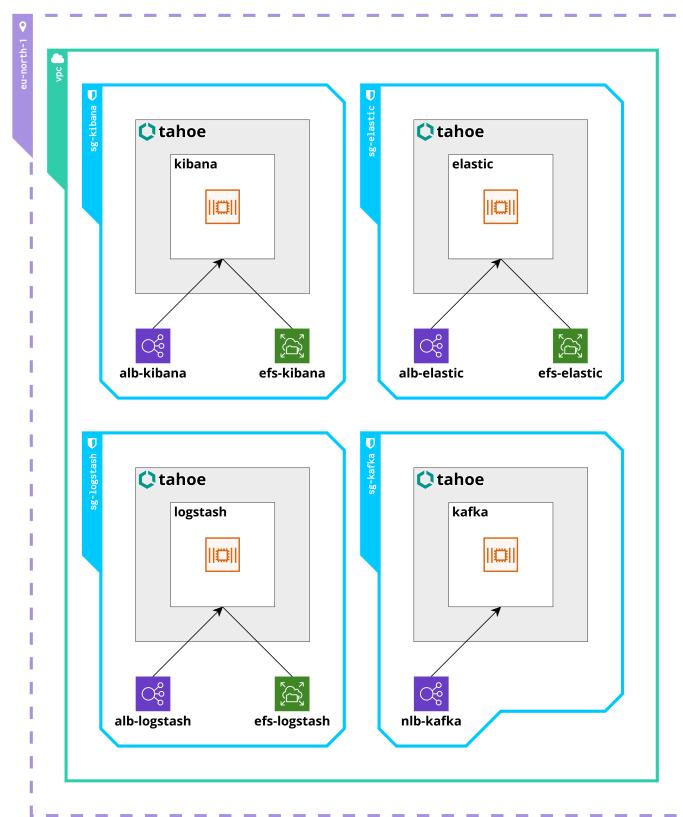


Figura 6.9: Diagrama incial de la seguridad en AWS

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.

6.2.4. 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.

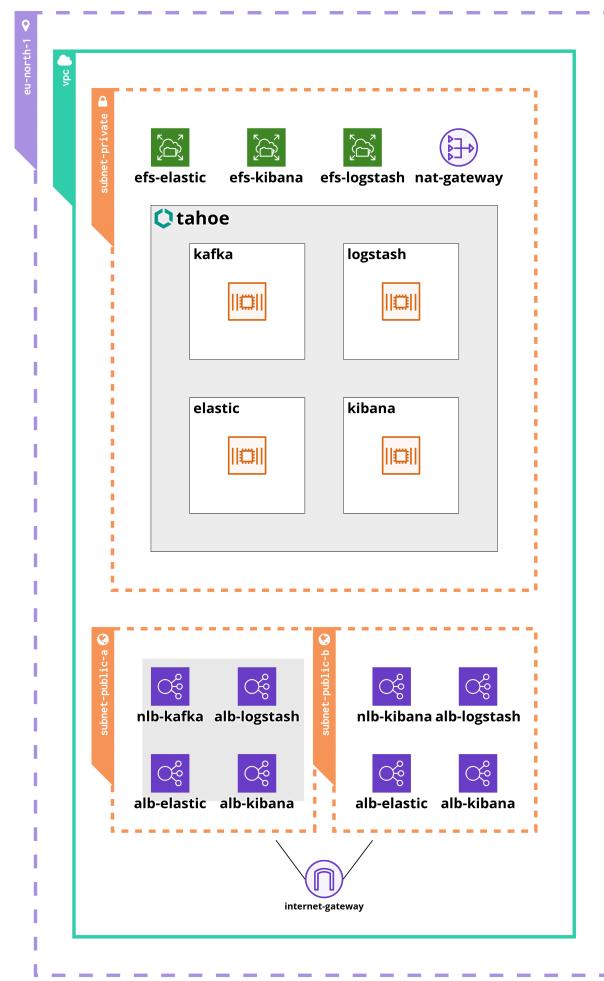


Figura 6.10: Diagrama incial de las redes en AWS

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.

6.3. 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.

Cada uno de los modelos de datos se corresponde con una fuente de datos diferente, y se utilizarán para analizar y visualizar la información de manera eficiente.

NOTA: por confidencialidad, no se mostrarán los modelos de datos internos de Okticket (hojas de gastos, reportes, usuarios, compañías, etc.).

6.3.1. Logs de balanceadores (AWS)

Los registros de los balanceadores de carga de AWS contienen información sobre las solicitudes que se realizan a los servicios de la aplicación. En la figura 6.11 se muestra el modelo de datos de estos registros.

Estos registros contienen información muy valiosa sobre el tráfico de la empresa, y permiten identificar métricas así como problemas de infraestructura de manera rápida y eficiente.

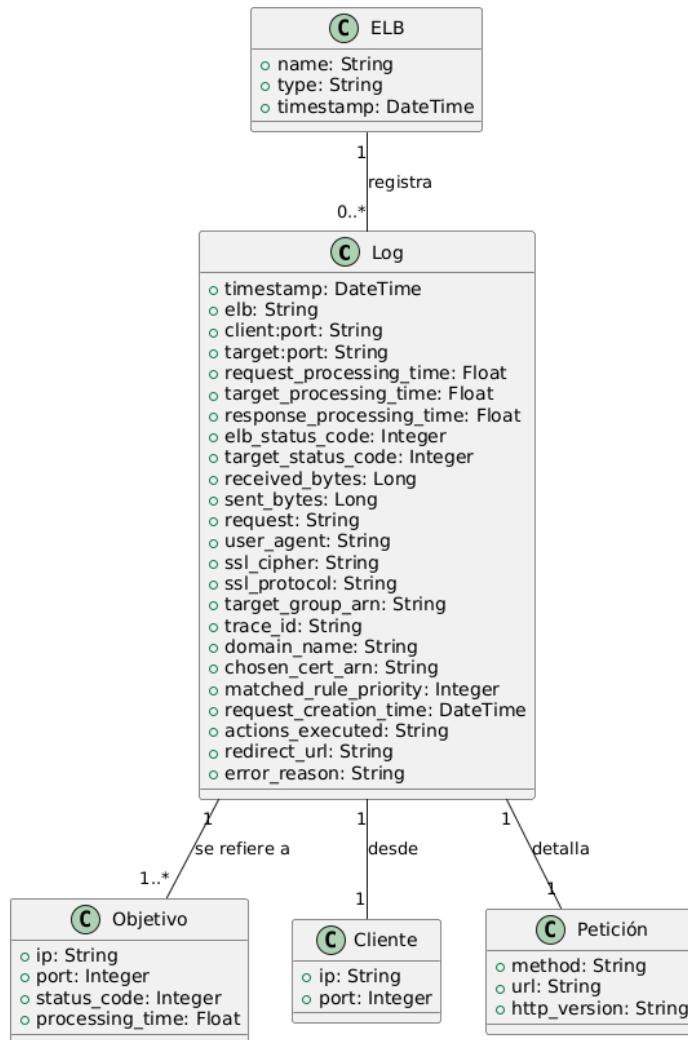


Figura 6.11: Modelo de datos de los logs de un balanceador de carga de AWS

En la figura 6.11, ELB es el balanceador de carga de AWS, Log es el registro de la solicitud, Cliente es el usuario que realiza la solicitud, Objetivo es el servicio al que se dirige la solicitud y Petición es la solicitud en sí.

6.3.2. Logs de bases de datos SQL

Los registros de las bases de datos SQL contienen información sobre las consultas que se realizan a la base de datos, así como errores y otros eventos importantes. En la figura 6.12 se muestra el modelo de datos de estos registros.

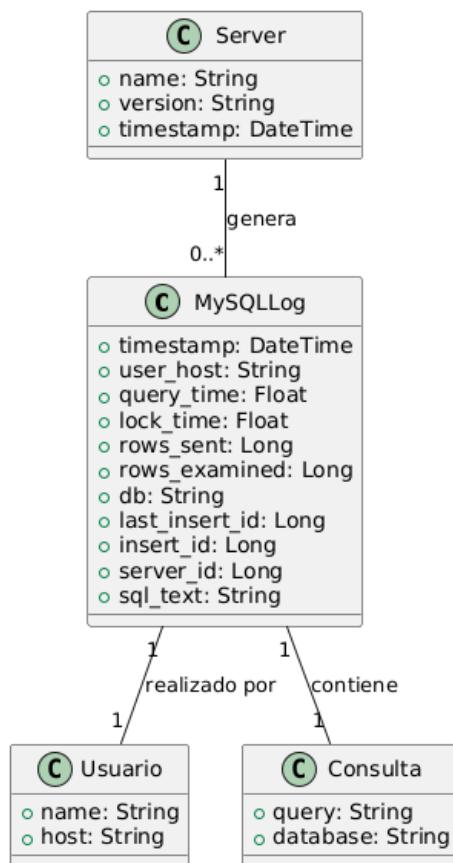


Figura 6.12: Modelo de datos de los logs de una base de datos SQL

En la figura 6.12, **Server** es el servidor de la base de datos, **MySQLLog** es el registro de la base de datos MySQL, **Usuario** es el usuario que realiza la consulta y **Consulta** es la consulta realizada.

6.3.3. Logs de bases de datos MongoDB

Los registros de las bases de datos MongoDB contienen información sobre las operaciones que se realizan en la base de datos, así como errores y otros eventos importantes. En la figura 6.13 se muestra el modelo de datos de estos registros.

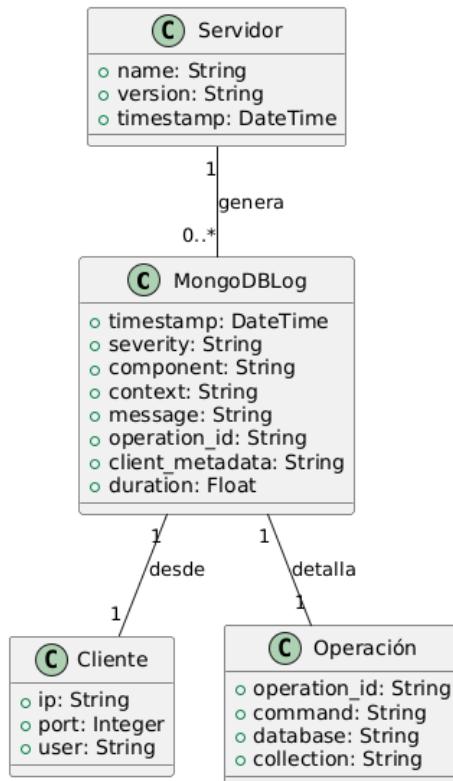


Figura 6.13: Modelo de datos de los logs de una base de datos MongoDB

En la figura 6.13, **Servidor** es el servidor de la base de datos, **MongoDBLog** es el registro de la base de datos MongoDB generado por el servidor, **Cliente** es el usuario que realiza la operación y **Operación** es la operación realizada.

6.3.4. Logs de Laravel (API)

Los logs de Laravel contienen, entre otras cosas, información sobre los errores y excepciones que se producen en el *backend* de la aplicación.

Para un análisis completo del estado de la aplicación (en el caso de Okticket), se deben analizar los logs de todos los despliegues de la API, ya que cada uno de ellos puede contener información relevante.

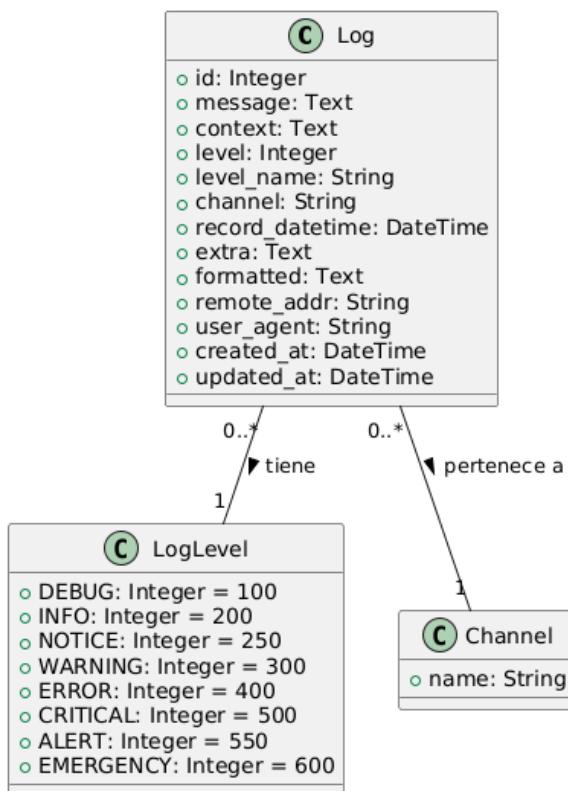


Figura 6.14: Modelo de datos de los logs de una aplicación Laravel

En la figura 6.14, Log es el registro de la aplicación en sí, LogLevel es el nivel de severidad del log y Channel es el canal de salida del log, un patrón de Laravel que permite filtrar los mensajes en función del nombre del canal (similar al concepto de tópico en Kafka).

7. 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 emplearse la metodología *Scrum*, se realizan primero las tareas más prioritarias, como la creación de la infraestructura y la ingestión 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*.

Con el objetivo de realizar un desarrollo iterativo y ágil, se desarrolla primero de todo un **prototipo de despliegue en local**, para posteriormente migrar este despliegue a la **nube**. A continuación, se desarrollan los scripts de ingestión de datos para las diferentes fuentes y se finaliza con la visualización de los datos en Kibana.

El burndown chart de la figura 7.1 muestra la evolución de las tareas a lo largo del tiempo, y se puede observar cómo se han ido completando las tareas planificadas.

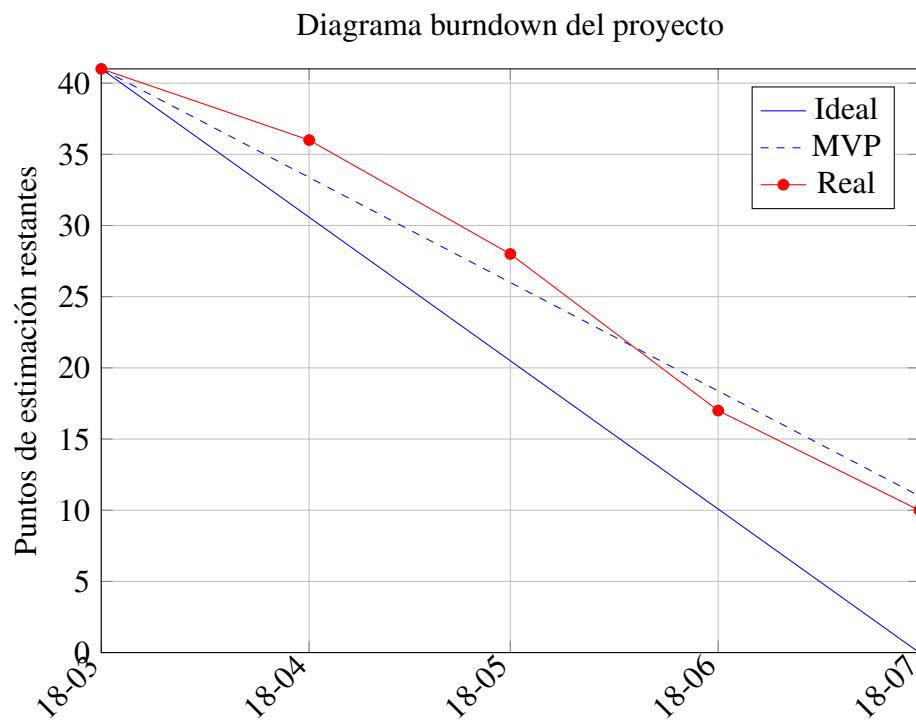


Figura 7.1: Diagrama Burndown que representa el progreso del proyecto

La estimación de las tareas se realizó en el apartado 4.2 *Planificación inicial*.

La línea azul representa el progreso ideal, mientras que la línea roja muestra el progreso real. Se toma el 18 de marzo como fecha de inicio del proyecto y el 18 de julio como fecha de finalización. Se puede observar cómo el progreso real sigue la línea ideal, aunque con algunas desviaciones debido a la naturaleza iterativa del desarrollo.

Pese a que el progreso del desarrollo no ha sido completamente óptimo, se ha logrado superar los objetivos principales del proyecto, entregando un producto mínimo viable (MVP) de calidad y sentando las bases para futuras iteraciones. Para una explicación más detallada sobre el progreso del proyecto contra las expectativas iniciales, se puede consultar el apartado *10 Resultados y trabajo futuro*.

7.1. Despliegue local

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

Puesto que se ha decidido utilizar Docker para la gestión de contenedores, se crea 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 ingestión 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*.

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

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

Tabla 7.1: Lista de HUs cumplimentadas con el despliegue local

7.1.1. Explicación del código

El código de despliegue local se encuentra en el anexo 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.

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. Los *health-checks* son normalmente comprobaciones de disponibilidad de un servicio, como la respuesta de un puerto o la existencia de un archivo.

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   kafkadata:
7   certs:
8
9 networks:
10  default:
11    driver: bridge
```

Listado 7.1: 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 Elastic-

search 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     - ./setup.sh:/usr/local/bin/setup.sh
7   user: root
8   container_name: setup
9   command: ["/bin/bash", "/usr/local/bin/setup.sh"]
10  healthcheck:
11    test: [ "CMD-SHELL", "[ -f config/certs/es01/es01.crt ]" ]
12    interval: 5s
13    timeout: 10s
14    retries: 10
```

Listado 7.2: 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
30   KAFKA_TOOLS_LOG4J_LOGLEVEL: ERROR
31   KAFKA_LOG4J_LOGGERS:
32     ↳ 'kafka=WARN,kafka.controller=WARN,kafka.log.LogCleaner=WARN,state.change.log=WARN'
```

Listado 7.3: 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    -
```

```
23      ↳ xpack.security.transport.ssl.certificateAuthorities=certs/ca/ca.crt
24          - xpack.security.transport.ssl.verificationMode=certificate
25
26      ulimits:
27          memlock:
28              soft: -1
29              hard: -1
30
31      nofile:
32          soft: 65536
33          hard: 65536
34
35      capAdd:
36          - IPC_LOCK
37
38      labels:
39          co.elastic.logs/module: elasticsearch
40          co.elastic.metrics/module: elasticsearch
41
42      volumes:
43          - es01data:/usr/share/elasticsearch/data
44          - certs:/usr/share/elasticsearch/config/certs
45
46      ports:
47          - 9200:9200
48
49      healthcheck:
50          test:
51              [
52                  "CMD-SHELL",
53                  "curl -s --cacert config/certs/ca/ca.crt"
54              ↳ https://localhost:9200 | grep -q 'missing authentication
55              ↳ credentials'"
56          ]
57          interval: 10s
58          timeout: 10s
59          retries: 10
```

Listado 7.4: 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 7.5: Definición de los servicios de Kibana

```
1 logstash:
2   container_name: logstash
3   image: docker.elastic.co/logstash/logstash:${STACK_VERSION}
4   restart: unless-stopped
5   user: root
6   volumes:
7     - logstash-data:/usr/share/logstash/data
8     - certs:/usr/share/logstash/certs
9     - ./logstash.conf:/usr/share/logstash/pipeline/logstash.conf
10  environment:
11    - ELASTIC_HOSTS=https://es01:9200
12    - ELASTIC_USER=elastic
13    - ELASTIC_PASSWORD=${ELASTIC_PASSWORD}
14    - log.level=warn
15    - xpack.monitoring.enabled=false
16  command: bin/logstash -f /usr/share/logstash/pipeline/logstash.conf
17  ports:
18    - 5000:5000
19  depends_on:
20    - es01
21    - kafka
22    - setup
23  labels:
24    co.elastic.logs/module: logstash
25    co.elastic.metrics/module: logstash
26  links:
27    - es01
28    - kibana
```

Listado 7.6: 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.

7.1.2. 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.

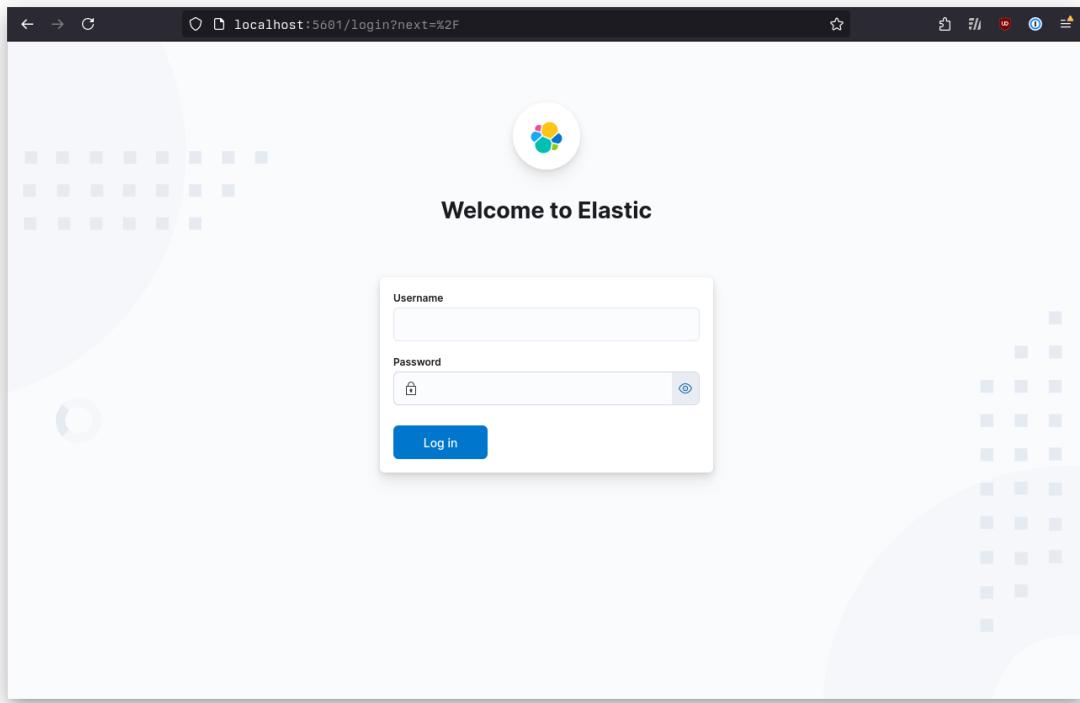


Figura 7.2: Inicio de sesión en Kibana

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

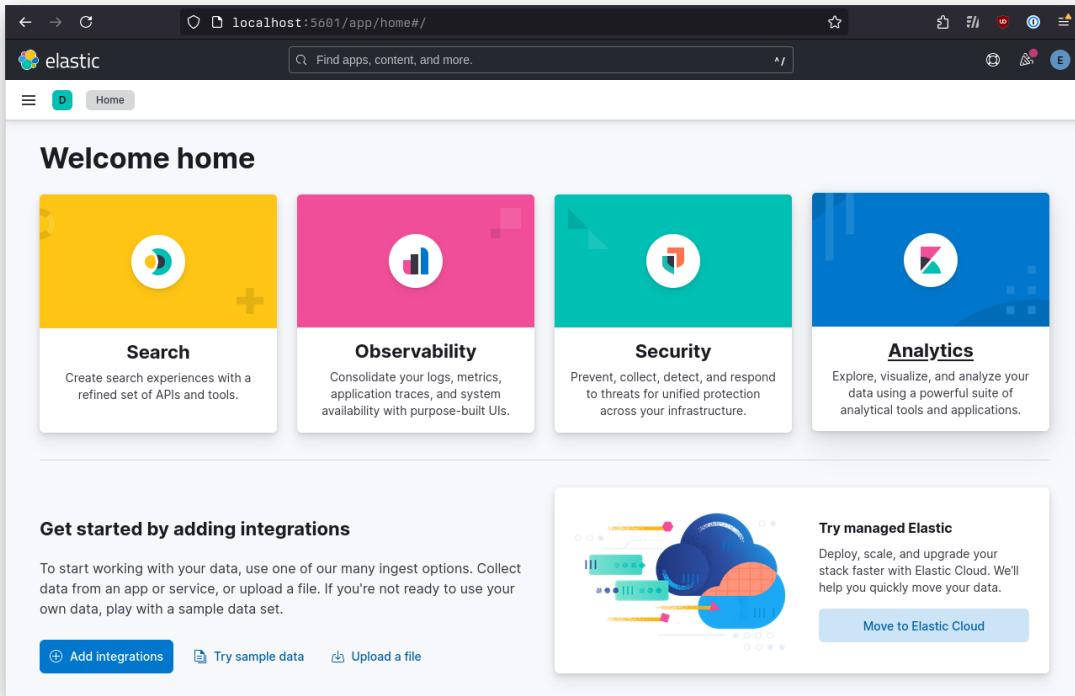


Figura 7.3: Página de inicio de Kibana

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.

Para más información, se puede consultar 9.1 *Manual de usuario*.

7.1.3. 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 7.4: 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>

7.2. 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 6.2 *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*.

Esta sección de la memoria documenta el desarrollo de las siguientes historias de usuario, siguiendo la planificación establecida en la sección 4.2 *Planificación inicial*:

Nombre	Prioridad	Tamaño
Como desarrollador de Okticket, quiero que la arquitectura se despliegue y orqueste de manera automática	P0	XL

Tabla 7.2: Lista de HUs cumplimentadas con el despliegue en la nube

7.2.1. 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.

elastic (64) Info		C	Deploy ▾	Actions ▾
<input type="checkbox"/>	Filter task definition revisions by value	Filter status	Inactive	▼
<input type="checkbox"/>	Task definition: revision	▼	Status	
<input type="checkbox"/>	elastic:64		INACTIVE	
<input type="checkbox"/>	elastic:63		INACTIVE	
<input type="checkbox"/>	elastic:62		INACTIVE	
<input type="checkbox"/>	elastic:61		INACTIVE	
<input type="checkbox"/>	elastic:60		INACTIVE	
<input type="checkbox"/>	elastic:59		INACTIVE	
<input type="checkbox"/>	elastic:58		INACTIVE	

Figura 7.5: Ejemplo de definiciones de tareas de ECS en AWS

The screenshot shows the AWS CloudWatch Task Definition console for a task named "kafka:4". The "Overview" tab is selected, displaying details such as 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 the overview, there are tabs for "Containers", "JSON" (which is selected), "Task placement", "Volumes (0)", and "Requires attributes". The "JSON" tab displays the task definition JSON code:

```

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                  }
15              ]
16          }
17      ]
18  }

```

Buttons for "Download JSON", "Download AWS CLI input", and "Copy to clipboard" are also visible.

Figura 7.6: Ejemplo de definición de tarea (Kafka)

Durante el desarrollo del despliegue, se utilizan las herramientas de monitorización de AWS para comprobar el estado de los recursos y servicios creados, y se realizan pruebas básicas de funcionamiento para asegurar que los servicios se han desplegado correctamente. En la figura 7.7, se muestran los logs de una tarea de ECS.

The screenshot shows the AWS CloudWatch Logs console for an ECS task. The "Logs" tab is selected. The log stream title is "Logs (175+)". A search bar at the top allows filtering log events with filter patterns. Below the search bar, there are filters for "Filter container" (set to "kibana") and "Filter date time range" (set to "Since 1 hour ago"). The log table has columns for "Timestamp (UTC+02:00)" and "Message". The log entries are:

- July 15, 2024 at 22:01 [2024-07-15T20:01:03.365+00:00] [INFO] [plugins.securitySolution.endpoint:user-artifactpackager:1.0.0] Last computed manifest not available yet
- July 15, 2024 at 22:01 [2024-07-15T20:01:03.366+00:00] [INFO] [plugins.securitySolution.endpoint:user-artifactpackager:1.0.0] Complete. Task run took 7ms [stated: 2024-07-15T20:01:03.359Z]
- July 15, 2024 at 22:01 [2024-07-15T20:01:03.359+00:00] [INFO] [plugins.securitySolution.endpoint:user-artifactpackager:1.0.0] Started. Checking for changes to endpoint artifacts

Figura 7.7: Ejemplo de logs de una tarea de ECS

En la figura 7.8, se muestra el estado actual de todos los balanceadores de carga, junto a más detalles como sus zonas de disponibilidad o el estado de los nodos.

Load balancers (4)											<input type="button" value="Create load balancer"/>
Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.											
	Name	DNS name	State	VPC ID	Availability Zones	Type	Date created				
<input type="checkbox"/>	tahoe-alb-kafka	tahoe-alb-kafka-Ocaf... 	Active 	vpc-08df523d430e9a5...	2 Availability Zones 	network	July 15, 2024, 13:14 (...)				
<input type="checkbox"/>	tahoe-alb-kibana	tahoe-alb-kibana-167786... 	Active 	vpc-08df523d430e9a5...	2 Availability Zones 	application	July 15, 2024, 13:14 (...)				
<input type="checkbox"/>	tahoe-alb-elastic	tahoe-alb-elastic-6930628... 	Active 	vpc-08df523d430e9a5...	2 Availability Zones 	application	July 15, 2024, 13:14 (...)				
<input type="checkbox"/>	tahoe-alb-logstash	tahoe-alb-logstash-11166... 	Active 	vpc-08df523d430e9a5...	2 Availability Zones 	application	July 15, 2024, 13:14 (...)				

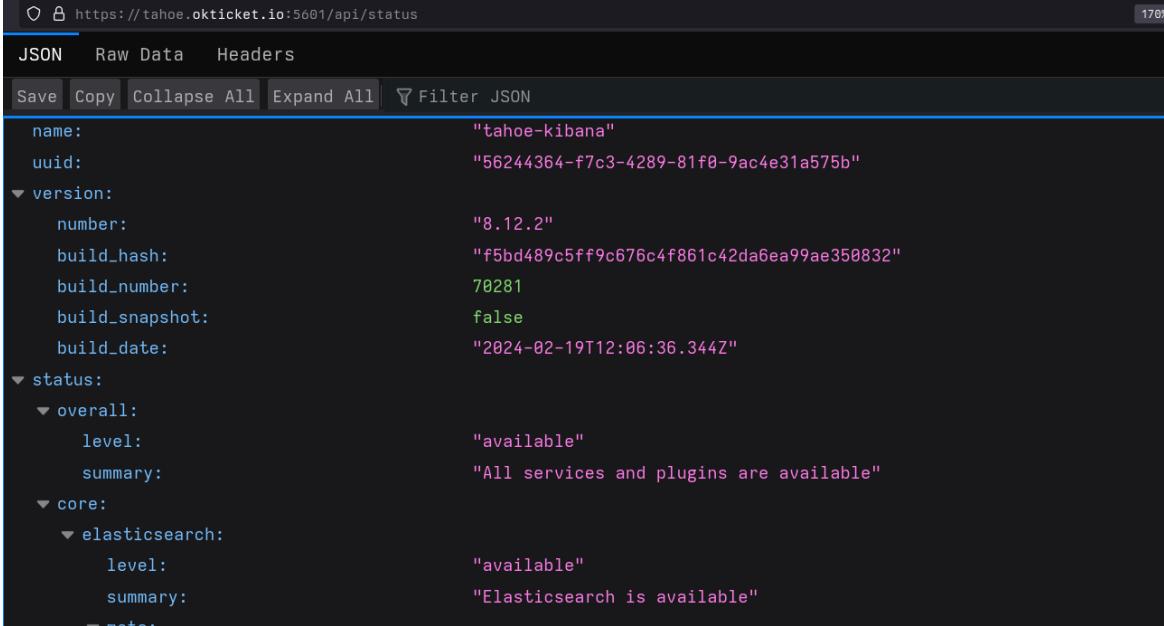
Figura 7.8: Estado de los平衡adores de carga

Por último, en la figura 7.9 se muestra el estado de un despliegue de un servicio, con información sobre el estado de los contenedores, la versión de la imagen, la cantidad de tareas en ejecución y la cantidad de tareas deseadas.

Deployment configuration <small>Info</small>								<small>View pipelines</small>							
Deployment status Completed		Deployment type Rolling update (ECS)		Platform version LATEST		Min and max running tasks 100% min and 200% max									
Deployment failure detection															
Task placement strategy and constraints															
Deployments (1) <small>Info</small>															
<input type="button" value="Filter deployments"/>															
Start date	Status	Failed tasks	Tasks	Version	Task definition	Revision	Last deployment								
July 15, 2024 at 14:51 (UTC+2:00)	Primary 100%	0	1 Running 0 Pending 1 Desired	1.4.0	kibana	45	Completed								
Events (45)															
<input type="button" value="Filter events by value"/>															
Started at	Message	Event ID													
July 15, 2024 at 21:00 (UTC+2:00)	service kibana has reached a steady state.	3ddfcba4-3195-4367-b029-660b3136ef64													
July 15, 2024 at 15:00 (UTC+2:00)	service kibana has reached a steady state.	284b7f5b-6b66-4c3f-af1e-7286c4bedf5e													

Figura 7.9: Métricas de estado del despliegue de un servicio

Por supuesto, los propios servicios cuentan con sus herramientas de monitorización básicas a las que se puede acceder a través del navegador (en caso de que funcionen correctamente).



The screenshot shows a browser window displaying a JSON API response from `https://tahoe.ockticket.io:5601/api/status`. The response is a nested JSON object with the following structure and values:

```
name: "tahoe-kibana"
uuid: "56244364-f7c3-4289-81f0-9ac4e31a575b"
version:
  number: "8.12.2"
  build_hash: "f5bd489c5ff9c676c4f861c42da6ea99ae350832"
  build_number: 78281
  build_snapshot: false
  build_date: "2024-02-19T12:06:36.344Z"
status:
  overall:
    level: "available"
    summary: "All services and plugins are available"
  core:
    elasticsearch:
      level: "available"
      summary: "Elasticsearch is available"
    meta:
```

Figura 7.10: Estado de Kibana

Estas herramientas se utilizan para el desarrollo incremental y la comprobación de que los servicios se despliegan correctamente, y se espera que, en la fase de producción, no sea necesario su uso, puesto que se cuenta con herramientas de monitorización más avanzadas y específicas para cada servicio.

7.2.2. Despliegue de la infraestructura

Para desplegar la infraestructura diseñada en AWS utilizando Terraform, se siguen los siguientes pasos:

1. **Preparación del entorno:** se asegura que se tienen las herramientas y configuraciones necesarias para llevar a cabo el despliegue.
2. **Inicialización:** se inicializa el directorio de trabajo con los módulos y configuraciones necesarios de la herramienta de despliegue.
3. **Planificación:** se planifican los cambios que se van a realizar en el proveedor en la nube.
4. **Aplicación:** se aplican los cambios planificados en el proveedor en la nube.
5. **Verificación:** una vez completado el despliegue, se verifican los recursos y servicios creados para asegurar que se han desplegado correctamente.
6. **Pruebas:** se realizan pruebas básicas de conectividad y funcionalidad para asegurar que la infraestructura está operativa y cumple con los requisitos establecidos.

Ver: 9.2 Manual de despliegue.

Como se ha mencionado anteriormente, durante cualquier fase del despliegue pueden ocurrir errores o problemas que requieran intervención manual. En caso de que ocurran, se debe revisar el estado de los recursos y servicios en la consola de AWS, y se debe corregir el problema manualmente si es necesario.

Se mantiene un control de versiones de los archivos de configuración de Terraform para facilitar el seguimiento de cambios y la posible futura colaboración en el desarrollo de la infraestructura.

Para visualizar el proceso de despliegue de la infraestructura, se ha creado un diagrama que ilustra los principales componentes y pasos. Este diagrama se muestra en la Figura 7.11.

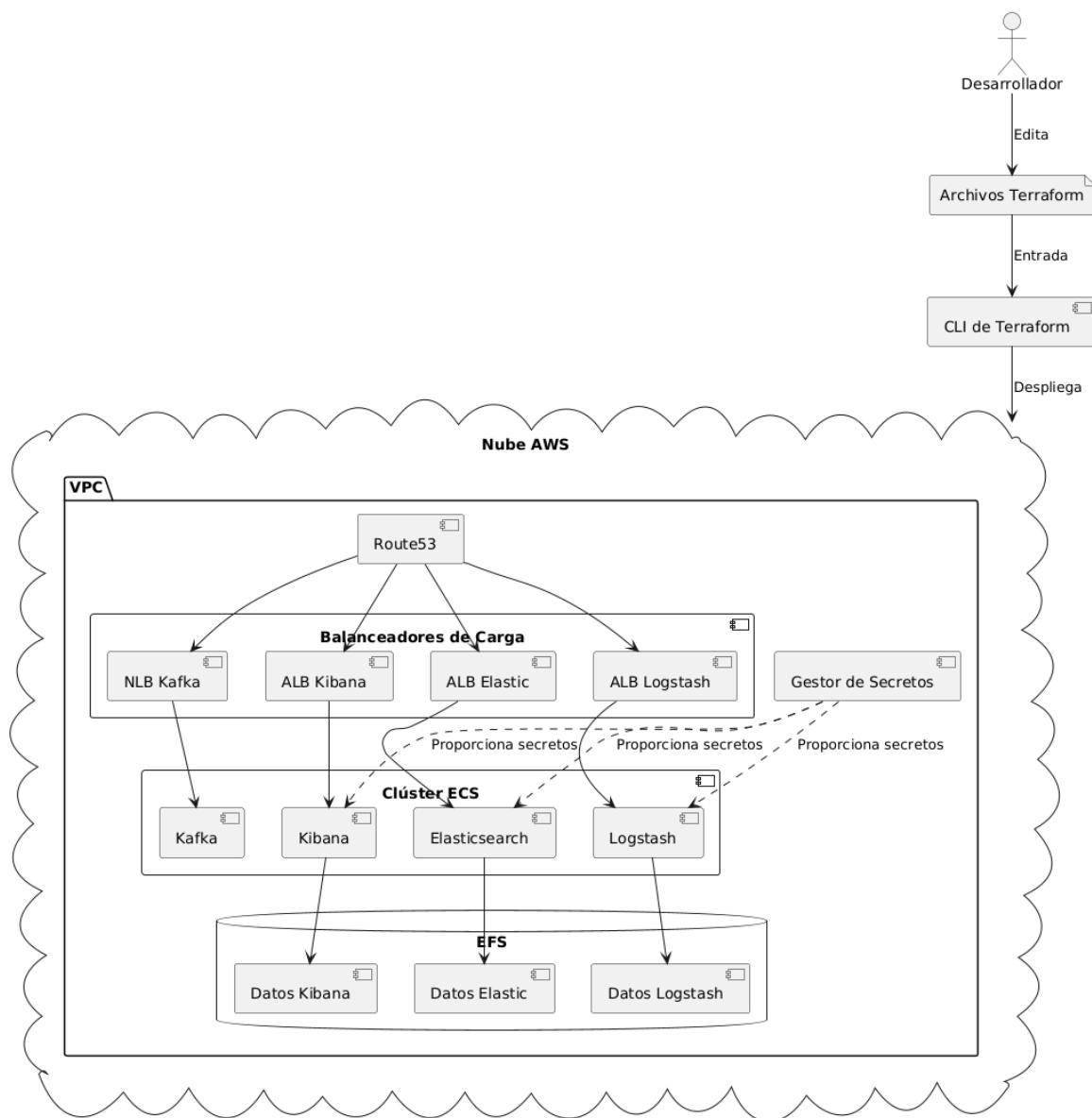


Figura 7.11: Diagrama de despliegue de la infraestructura

El diagrama 7.11 muestra el flujo desde los archivos de configuración de Terraform hasta los recursos desplegados en AWS, pasando por las diferentes etapas del proceso de despliegue.

7.2.3. Explicación del código

El código completo se encuentra en el anexo C Scripts de despliegue cloud.

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.

7.2.3.1. Recursos generales

Como previamente descrito, la definición de los recursos generales se divide en ficheros, cada uno de ellos con una función específica. A continuación, se detallan dichos ficheros y su función en el proyecto.

Fichero principal El fichero principal de Terraform, `main.tf`, se encarga de las configuraciones más esenciales o que no tienen cabida en otros ficheros, como la definición de la región, el cluster, el grupo de logs o el bucket S3 de logs.

Variables El fichero de variables `variables.tf` se encarga de la definición de todas las variables necesarias para el despliegue de la infraestructura, como nombres, regiones, versiones, puertos, etc., de manera que se puedan modificar y reutilizar de manera sencilla a lo largo del resto de ficheros de definición.

También se definen contraseñas de manera aleatoria, para mejorar la seguridad de los servicios.

Salidas El fichero de salidas `outputs.tf` se encarga de la definición de las salidas de Terraform, es decir, de las variables que se pueden consultar una vez que se ha desplegado la infraestructura, como las direcciones URL de los servicios o las contraseñas generadas aleatoriamente.

Un ejemplo de estas salidas se encuentra en la figura 9.10 *Despliegue exitoso de la infraestructura base*.

Volúmenes lógicos (EFS) El fichero de volúmenes lógicos `efs.tf` se encarga de la definición de los volúmenes lógicos necesarios para la persistencia de los datos de los servicios, como los datos de Elasticsearch o los logs de los servicios.

Roles, políticas y permisos (IAM) El fichero de roles, políticas y permisos `iam.tf` se encarga de la definición de los roles y políticas necesarios para el correcto funcionamiento de los servicios, como los roles de ejecución de tareas de ECS, los permisos de acceso a los servicios de AWS o las políticas de acceso a los recursos.

Secretos El fichero de secretos `secrets.tf` se encarga de la definición de las claves necesarias para el correcto funcionamiento de los servicios, como los certificados de Elasticsearch o las claves de acceso a los servicios.

Estos secretos se almacenan en *AWS Secrets Manager*, un servicio de AWS que permite el almacenamiento seguro de información sensible, como contraseñas, claves de acceso o certificados. Los certificados se almacenan en formato `base64` para facilitar su almacenamiento y evitar problemas de codificación en el paso de los mismos. Se utilizan cadenas generadas aleatoriamente para evitar la exposición de contraseñas y claves de acceso en el código.

Redes El fichero de redes `network.tf` se encarga de la definición de los elementos de red necesarios para la correcta comunicación entre los servicios y su seguridad.

La lógica y arquitectura de red se explica en el apartado 6.2.4 *Redes*, lo que facilita la definición de las redes en Terraform.

Grupos de seguridad El fichero de grupos de seguridad `security.tf` se encarga de la definición de los grupos de seguridad necesarios para la correcta comunicación entre los servicios y su seguridad.

La lógica y arquitectura de seguridad se explica en el apartado 6.2.3 *Seguridad*.

7.2.3.2. Servicios de ELK

Puesto que la estructura de definición de servicios en Terraform es similar para todos los servicios, se analiza el más completo, Elasticsearch, para explicar el funcionamiento y la lógica de los mismos.

Cada fichero de definición de servicios está separado en tres partes:

1. **Definición de la tarea**, que contiene la configuración del contenedor al estilo de *Docker Compose* (imagen, variables de entorno, volúmenes, etc.)
2. **Definición del servicio**, que asigna la tarea al resto de la configuración del servicio.
3. **Definición de recursos**, los necesarios para cada servicio:平衡adores de carga, *listeners*, configuraciones DNS, etc.

El código completo de los servicios se encuentra en el anexo *C Scripts de despliegue cloud*.

Definición de la tarea A continuación, se muestra un ejemplo de la definición de la tarea de Elastic:

```
1 resource "aws_ecs_task_definition" "elastic" {
2   family                  = "elastic"
3   network_mode            = "awsvpc"
4   requires_compatibility = [var.launch_type]
5   cpu                     = "2048"
6   memory                  = "4096"
7   execution_role_arn     = aws_iam_role.ecs_task_execution.arn
8   task_role_arn           = aws_iam_role.ecs_task_execution.arn
9
10  container_definitions = jsonencode([
11    {
12      name      = "es01"
13      image     = "docker.elastic.co/elasticsearch/elasticsearch:${var
14      ↪ .stack_version}"
15      cpu       = 2048
16      memory    = 4096
17      essential = true
18      environment = [
19        { name = "cluster.name", value = var.cluster_name },
20        { name = "xpack.security.enabled", value = "true" },
21        { name = "xpack.security.http.ssl.enabled", value = "true" },
```

```
21      <...>
22      ]
23      secrets = [
24      {
25          name      = "CA_CRT"
26          valueFrom = "${aws_secretsmanager_secret.es_certs.arn}:ca.
27      crt::"
28      },
29      {
30          name      = "ES01_KEY"
31          valueFrom = "${aws_secretsmanager_secret.es_certs.arn}:es01.
32      key::"
33      },
34      <...>
35      ]
36      entrypoint = [
37          "/bin/sh",
38          "-c",
39          <<-EOT
40          #!/bin/bash
41          set -e
42
43          echo "Configurando credenciales..."
44          mkdir -p /usr/share/elasticsearch/config/certs
45          echo $CA_CRT | base64 -d > /usr/share/elasticsearch/config/
46          certs/ca.crt
47          <...>
48          EOT
49      ]
50      # mountPoints = [
51      # {
52      #     sourceVolume  = "elastic-data"
53      #     containerPath = "/usr/share/elasticsearch/data"
54      #     readOnly      = false
55      # }
56      # ]
57      portMappings = [
58      {
59          containerPort = var.elastic_port,
60          hostPort      = var.elastic_port
61      }
62      ]
63      logConfiguration = {
64          logDriver = var.log_driver
65          options = {
66              "awslogs-group"      = var.log_group
67          }
```

```
64         "awslogs-region"          = var.region
65         "awslogs-stream-prefix" = "elastic"
66     }
67   }
68   ulimits = [
69   {
70     name      = "memlock"
71     softLimit = -1
72     hardLimit = -1
73   },
74   {
75     name      = "nofile"
76     softLimit = 65536
77     hardLimit = 65536
78   }
79 ]
80 }
81 ])
82
83 volume {
84   name = "elastic-data"
85   efs_volume_configuration {
86     file_system_id = aws_efs_file_system.elastic_data.id
87     root_directory = "/"
88   }
89 }
90 }
```

Listado 7.7: Definición de la tarea de Elastic

Como se puede comprobar, pese a que la sintaxis de configuración es distinta, la estructura de la misma es muy similar a aquella ya definida durante el 7.1 *Despliegue local*, con similares configuraciones de entorno, mapeos de puertos y volúmenes, y configuraciones de red. Ese es el objetivo del planteamiento iterativo y de desarrollo incremental, que permite reutilizar código y configuraciones ya definidas.

Al igual que en el desarrollo local, se obvia la configuración de los recursos necesaria para la escalabilidad y replicación de los servicios, puesto que de momento no se necesitan más que una instancia de cada servicio. Sin embargo, esto no significa que no se pueda añadir de manera sencilla en un futuro, como así lo demuestran, por ejemplo, la asignación de los servicios a volúmenes EFS.

A diferencia del desarrollo local, la preparación del servicio se realiza desde la declaración del endpoint, en lugar de requerir un segundo contenedor de configuración.

Definición del servicio A continuación, se detalla la definición del servicio de Elasticsearch:

```
1 resource "aws_ecs_service" "elastic" {
2   name           = "elastic"
3   cluster        = aws_ecs_cluster.cluster.id
4   task_definition = aws_ecs_task_definition.elastic.arn
5   desired_count  = 1
6   launch_type    = var.launch_type
7   enable_execute_command = true
8
9   network_configuration {
10     subnets      = [aws_subnet.private.id]
11     security_groups = [aws_security_group.elastic.id]
12     assign_public_ip = true
13   }
14
15   load_balancer {
16     target_group_arn = aws_lb_target_group.elastic.arn
17     container_name   = "es01"
18     container_port   = var.elasticsearch_port
19   }
20
21   depends_on = [
22     aws_lb_listener.elastic,
23     aws_ecs_task_definition.elastic
24   ]
25 }
```

Listado 7.8: Definción del servicio de Elastic

La definición del servicio es mucho más breve que el resto de definiciones, puesto que la mayoría de la configuración se realiza en la tarea. En este caso, se asigna la tarea a un grupo de seguridad y a una subred privada, y se asigna el servicio a un balanceador de carga, que se encargará de distribuir el tráfico entre los contenedores.

Definición de recursos Todos los recursos necesarios para el servicio, como el balanceador de carga, el grupo de seguridad, el *listener* o la configuración DNS, se definen a continuación.

```
1 resource "aws_lb" "elastic" {
2   name          = "tahoe-alb-elastic"
3   internal      = false
4   load_balancer_type = "application"
5   security_groups = [aws_security_group.elastic.id]
6   subnets        = [aws_subnet.public_a.id, aws_subnet.public_b.id
7                     ↵ ]
8
9   access_logs {
10     bucket    = aws_s3_bucket.access_logs.bucket
11     prefix    = "elastic"
12     enabled   = true
13   }
14
15  resource "aws_lb_target_group" "elastic" {
16    name          = "tahoe-tg-elastic"
17    port          = var.elastic_port
18    protocol     = "HTTPS"
19    vpc_id       = aws_vpc.main.id
20    target_type  = "ip"
21
22    health_check {
23      enabled      = true
24      path         = "/"
25      protocol    = "HTTPS"
26      matcher     = "200"
27      interval    = 300
28      timeout     = 60
29      healthy_threshold = 2
30      unhealthy_threshold = 5
31      port        = tostring(var.elastic_port)
32    }
33  }
34
35  resource "aws_lb_listener" "elastic" {
36    load_balancer_arn = aws_lb.elastic.arn
37    port            = var.elastic_port
38    protocol        = "HTTPS"
39    ssl_policy      = "ELBSecurityPolicy-2016-08"
40    certificate_arn = aws_acm_certificate.elastic.arn
41
42    default_action {
```

```
43     type          = "forward"
44     target_group_arn = aws_lb_target_group.elastic.arn
45   }
46 }
47
48 resource "aws_lb_listener" "elastic_https" {
49   load_balancer_arn = aws_lb.elastic.arn
50   port              = 443
51   protocol          = "HTTPS"
52   ssl_policy         = "ELBSecurityPolicy-2016-08"
53   certificate_arn   = aws_acm_certificate.elastic.arn
54
55   default_action {
56     type = "redirect"
57     redirect {
58       port      = tostring(var.elastic_port)
59       protocol  = "HTTPS"
60       status_code = "HTTP_301"
61     }
62   }
63 }
64
65 resource "aws_acm_certificate" "elastic" {
66   domain_name        = local.elastic_url
67   validation_method  = "DNS"
68
69   lifecycle {
70     create_before_destroy = true
71   }
72 }
73
74 resource "aws_route53_record" "elastic" {
75   zone_id    = var.route53_zone_id
76   name        = local.elastic_url
77   type        = "A"
78
79   alias {
80     name          = aws_lb.elastic.dns_name
81     zone_id      = aws_lb.elastic.zone_id
82     evaluate_target_health = false
83   }
84 }
85
86 resource "aws_route53_record" "elastic_validation" {
87   for_each = {
```

```
88     for dvo in aws_acm_certificate.elastic.domain_validation_options :
89         ↪   dvo.domain_name => {
90             name      = dvo.resource_record_name
91             record    = dvo.resource_record_value
92             type      = dvo.resource_record_type
93         }
94
95     allow_overwrite = true
96     name           = each.value.name
97     records        = [each.value.record]
98     ttl            = 60
99     type           = each.value.type
100    zone_id        = var.route53_zone_id
101 }
102
103 resource "aws_acm_certificate_validation" "elastic" {
104     certificate_arn          = aws_acm_certificate.elastic.arn
105     validation_record_fqdns = [for record in aws_route53_record.
106       ↪   elastic_validation : record.fqdn]
106 }
```

Listado 7.9: Definición de recursos de Elastic

Como para todos los servicios, se necesitan definir los recursos de red que permitan acceder a los mismos, es decir:

- Un balanceador de carga que distribuya el tráfico entre los contenedores de Elasticsearch.
- Un *target group* que asigne los contenedores al balanceador de carga y permita la comprobación de la salud de los contenedores.
- *Listeners* que permitan el acceso a los servicios desde el exterior o redirijan el tráfico a los contenedores dependiendo del puerto de conexión.

En el caso de Elastic (y el de otros servicios como Kibana), también se requiere la configuración de un subdominio DNS y sus certificados pertinentes, para facilitar el acceso a los servicios desde el exterior. Sin dichos subdominios, la dirección URL de los servicios cambiaría cada vez que se desplegaran, lo que dificultaría el acceso a los mismos. El tiempo de vida (*TTL*) de estas definiciones es de un minuto por defecto, lo que permite la actualización de los certificados y la dirección URL de manera rápida y sencilla.

7.3. 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.

Esta sección de la memoria documenta el desarrollo de las siguientes historias de usuario, siguiendo la planificación establecida en la sección 4.2 *Planificación inicial*:

Nombre	Prioridad	Tamaño
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	S
Como desarrollador de Okticket, quiero que se ingesten de manera automática logs de balanceador de AWS	P1	M

Tabla 7.3: Lista de HUs cumplimentadas con la ingestión de datos

7.3.1. Ingesta de métricas con Kafka

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 en bucle con una pausa de segundos entre ejecuciones, aunque se pueden adaptar para cualquier otra ejecución periódica como triggers, *cron jobs* o *webhooks*.

Algunos de los servidores que contienen la información a ingestar se encuentran detrás de subredes privadas, por lo que los scripts de ingestión se deben ejecutar a través de SSH en los servidores bastiones de dichas subredes. Estos servidores se encargarán de conectar con la infraestructura desplegada y enviar la información.

Ver código: D Scripts de ingestión de datos con Kafka

7.3.1.1. Componentes de Kafka

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.

7.3.1.2. Explicación del código

Todos los *productores* de datos siguen un patrón similar, ya que su función es la misma: leer datos de una fuente, transformarlos y enviarlos a Kafka.

Su código se divide en tres partes principales, siguiendo con el patrón de diseño de *Extract, Transform, Load* (ETL) definido anteriormente (ver 2.3):

- **Lectura de datos:** se encarga de la lectura de los datos de la fuente de datos. En el caso de las bases de datos, se conecta a la base de datos y ejecuta una consulta para obtener los datos. En el caso de los logs, se lee el fichero de logs y se extraen las líneas que contienen la información necesaria.
- **Transformación de datos:** se encarga de la transformación de los datos leídos a un formato adecuado para su envío a Kafka. En el caso de las bases de datos, se transforman los datos a un formato JSON. En el caso de los logs, se transforman las líneas de logs a un formato JSON.
- **Envío de datos:** se encarga del envío de los datos a Kafka. Para ello, se conecta a Kafka y envía los datos a un tópico concreto.

7.3.1.3. Dependencias

Para la ejecución de los scripts de Python, se necesitan las siguientes librerías, dependiendo del script a ejecutar:

- **kafka-python:** librería de Python que permite la conexión y el envío de datos a Kafka.
- **pymysql:** librería de Python que permite la conexión y la ejecución de consultas en bases de datos MySQL.
- **pymongo:** librería de Python que permite la conexión y la ejecución de consultas en bases de datos MongoDB.
- **python-dotenv:** librería de Python que permite la carga de variables de entorno desde un fichero .env.

7.3.2. Ingesta de logs de AWS

Para la ingestá de logs de balanceadores de carga de AWS, se desarrolla una lambda en Python que se encarga de la lectura de los logs de CloudWatch y su envío a Logstash. Se siguen cuatro pasos principales:

1. **Creación de la lambda:** se crea una lambda en Python que se ejecuta cada vez que se genera un log en CloudWatch (5 minutos)
2. **Filtro de suscripción:** se crea una suscripción en CloudWatch para que los logs generados se envíen a la lambda.
3. **Configuración de Logstash:** se configura Logstash para que reciba los logs de la lambda y los envíe a Elasticsearch.
4. **Creación del índice:** se crea un índice en Elasticsearch para almacenar los logs.

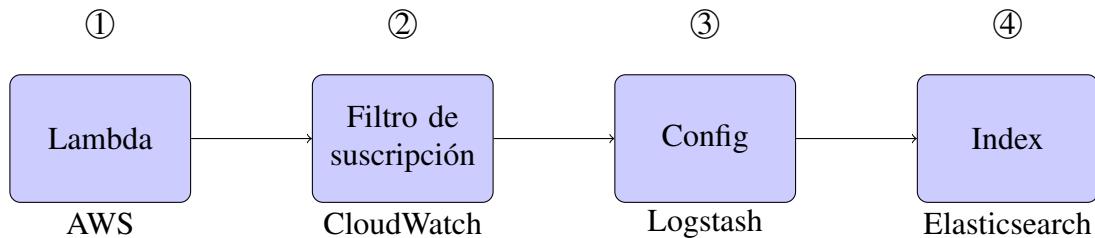


Figura 7.12: Proceso de ingestá de logs de balanceadores de carga de AWS

El código completo de las cuatro partes se encuentra en el anexo *E Código de ingestá de logs de ELB (AWS)*.

7.3.3. Otros métodos de ingesta

Pese a que el desarrollo y el alcance del proyecto se centra en la ingesta de datos a través de Kafka, existen otras formas de ingestar datos en el stack ELK, como por ejemplo:

- **Beats:** Beats es una familia de agentes que se encargan de la recolección de datos y su envío a Elasticsearch. Existen diferentes tipos de Beats, como Metricbeat, Filebeat o Heartbeat, que se adaptan a diferentes necesidades. Todos ellos son compatibles con la arquitectura planteada y son alternativas válidas y atractivas para el futuro de la solución.
- **Conectores:** Elastic cuenta con conectores oficiales para diferentes aplicaciones de terceros comunes, como Dropbox, Gmail o Jira. Estos conectores se encargan de la ingesta de datos de manera automática y son una opción a tener en cuenta.
- **Índices de API:** Elasticsearch permite la ingestión de datos a través de su API REST, lo que permite la integración con cualquier sistema que pueda enviar datos a través de HTTP.
- **Subida de ficheros:** A través de la interfaz de Kibana, es posible subir ficheros de datos para su ingestión en Elasticsearch. Esta opción es poco apropiada para el sistema planteado, pero puede resultar útil en caso de contar con información suelta de fuentes no habituales o para pruebas puntuales.
- **Web crawlers:** una de las historias de usuario planteadas en la sección 4.2 *Planificación inicial* es la ingesta de datos de webs de terceros. Elasticsearch cuenta con una funcionalidad diseñada con este objetivo, por lo que es una posibilidad clara.

7.4. 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.

Esta sección de la memoria documenta el desarrollo de las siguientes historias de usuario, siguiendo la planificación establecida en la sección 4.2 *Planificación inicial*:

Nombre	Prioridad	Tamaño
Como trabajador de Okticket, quiero poder ver y consultar datos internos de la empresa	P1	M
Como desarrollador de Okticket, quiero poder ver el estado general de la infraestructura	P1	M

Tabla 7.4: Lista de HUs cumplimentadas con la visualización de datos

7.4.1. Desarrollo de dashboards

Para el desarrollo de los dashboards, se han seguido los siguientes pasos:

1. **Identificación de métricas clave:** se identifican las métricas más relevantes incluyendo: número de solicitudes, latencia, códigos de estados, tipos de errores, etc.
2. **Diseño de visualizaciones:** Para cada métrica identificada, se decide el tipo de visualización más adecuada. Por ejemplo, gráficos temporales para métricas que varían con el tiempo, mapas para visualizar distribuciones geográficas, etc.
3. **Creación de paneles:** se crean dos paneles principales:
 - **Dashboard de estado general de infraestructura:** Incluye visualizaciones del tráfico total, distribución de carga, y estado de salud del servicio dependiendo de la infraestructura.
 - **Dashboard de métricas detalladas:** muestra información más específica como latencias por URL, análisis de errores, tendencias temporales, etc.
4. **Configuración de filtros y controles:** gracias al sistema de filtros y controles de Kibana, los usuarios pueden personalizar las visualizaciones para ver solo la información relevante, además de ajustar el rango temporal de los datos. (ver 9.1 *Manual de usuario*).

7.4.1.1. Personalización y acceso

Actualmente, todos los usuarios tienen acceso completo a los dashboards desarrollados. Se ha identificado la necesidad futura de implementar un sistema de gestión de usuarios y permisos en Kibana. Esto permitirá asegurar que cada usuario tenga acceso solo a los dashboards y datos relevantes para su función, especialmente importante para la visualización de métricas sensibles de los平衡adores de carga.

La implementación de este sistema de permisos, así como la capacidad de que los usuarios puedan personalizar sus propios dashboards, se ha planificado como una tarea para futuras iteraciones del proyecto.

7.4.2. Ejemplos de dashboards

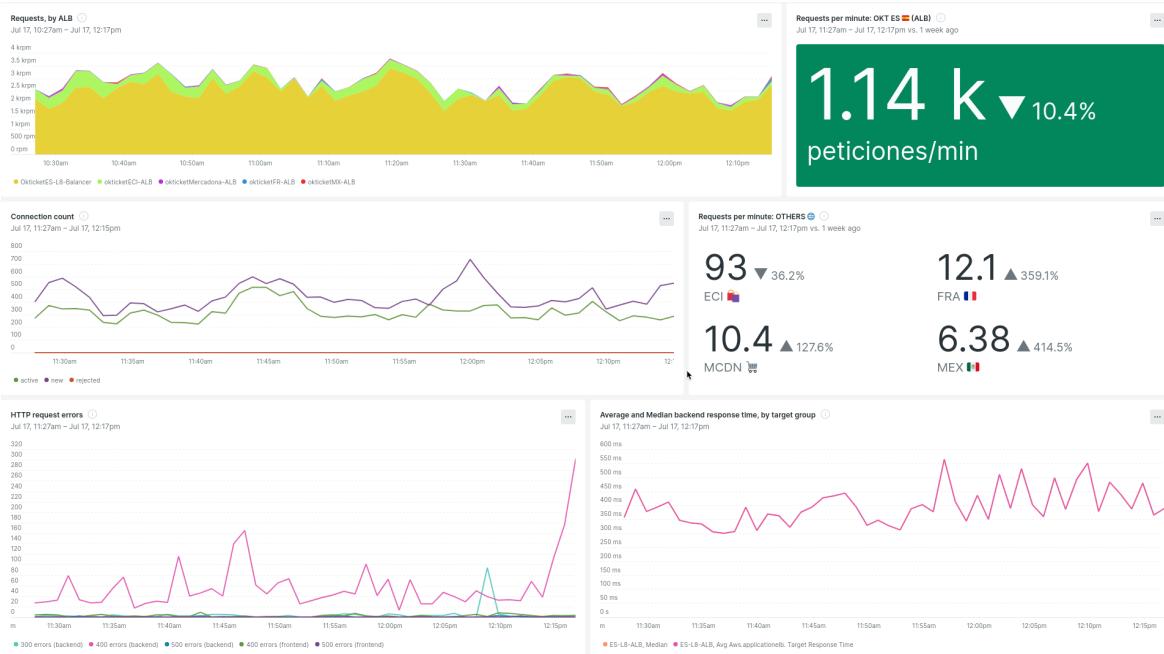


Figura 7.13: Dashboard de estado general de infraestructura

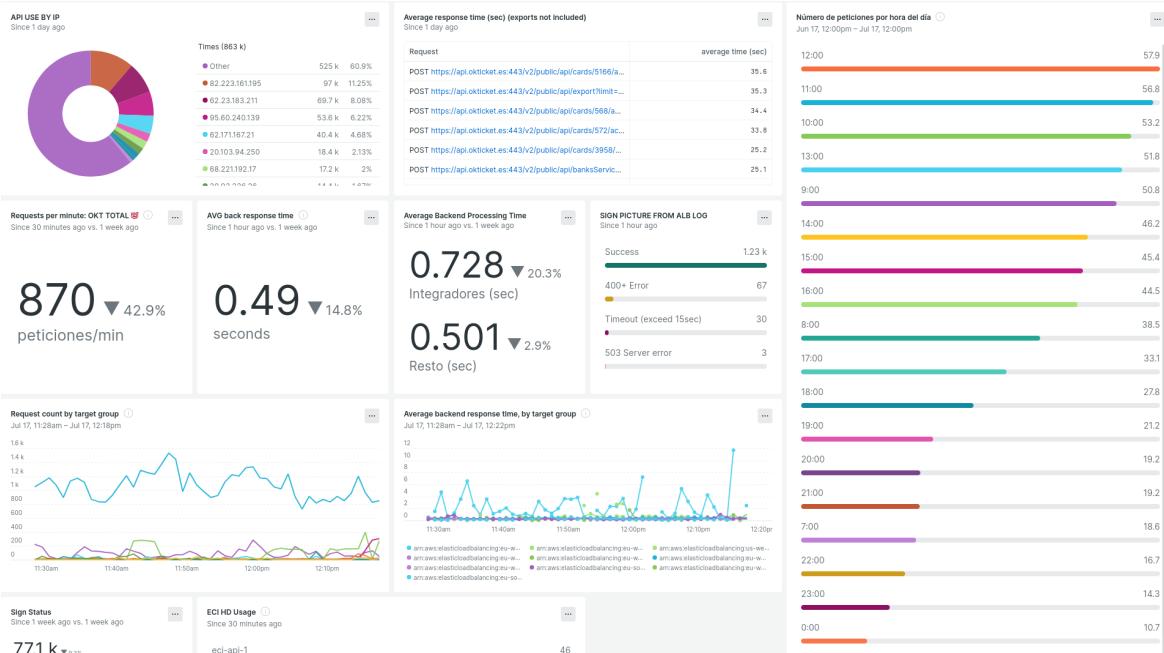


Figura 7.14: Dashboard de métricas detalladas

8. Pruebas

Las pruebas son una parte fundamental del desarrollo de software, ya que permiten verificar el correcto funcionamiento del sistema y detectar posibles errores antes de su implementación en un entorno de producción.

Con el objetivo de mejorar la calidad del sistema y tratar de encontrar posibles defectos, se ha diseñado un plan de pruebas que permite verificar y validar las diferentes funcionalidades del data lake. Este plan incluye pruebas de despliegue, ingestión de datos, visualización y rendimiento, abarcando los aspectos más críticos del sistema.

El plan de pruebas se ha diseñado siguiendo la técnica de clases de equivalencia, que permite reducir el número de casos de prueba necesarios mientras se mantiene una cobertura adecuada. Se han considerado diferentes aspectos del sistema, desde el despliegue inicial hasta el rendimiento bajo carga.

8.1. Plan de pruebas

El plan de pruebas evalúa diferentes aspectos del sistema, incluyendo el despliegue, la ingestión de datos, la visualización y el rendimiento. Cada aspecto se evalúa mediante pruebas de sistema y pruebas de carga, con el objetivo de verificar el correcto funcionamiento del sistema en diferentes condiciones.

El plan de pruebas se divide en dos categorías principales: pruebas de sistema y pruebas de carga. Cada categoría incluye diferentes tipos de pruebas, con objetivos específicos y casos de prueba asociados.

1. Pruebas de sistema

- a)* Pruebas de despliegue
- b)* Pruebas de visualización
- c)* Análisis estático de código

2. Pruebas de carga

- a)* Pruebas de ingestión de datos
- b)* Pruebas de rendimiento

A continuación, se detalla cada una de estas categorías, especificando los objetivos, las clases de equivalencia consideradas y los casos de prueba propuestos.

8.1.1. Pruebas de sistema

Las pruebas de sistema evalúan el correcto funcionamiento del sistema de principio a fin, desde la persistencia hasta la interacción del usuario con la UI, consiguiendo una cobertura completa del sistema.

8.1.1.1. Pruebas de despliegue

Las pruebas de despliegue son cruciales para asegurar que el sistema puede ser implementado correctamente en diferentes escenarios. Estas pruebas verifican la robustez del proceso de despliegue, su capacidad para manejar diversas condiciones iniciales y el manejo de errores durante el mismo.

Objetivo: Verificar que el sistema se despliega correctamente en diferentes escenarios.

Clases de equivalencia

- Estado del sistema: Sin desplegar, en progreso de despliegue, desplegado
- Existencia de volúmenes: Sin volúmenes virtuales, con volúmenes virtuales montados
- Estado de puertos: Libres, Ocupados
- Cuenta en AWS: Sin cuenta configurada, con cuenta configurada incorrectamente, con cuenta configurada correctamente.
- Servicios disponibles: servicios no disponibles, Elasticsearch disponible, ES+Kibana disponible, Kafka disponible, todos disponibles...

Casos de prueba

1. **DP-01:** Despliegue desde cero con todos los recursos libres
2. **DP-02:** Intento de despliegue con sistema ya desplegado
3. **DP-03:** Despliegue con volúmenes existentes y montados
4. **DP-04:** Despliegue con puertos ocupados (**Inválido**)
5. **DP-05:** Despliegue sin cuenta de AWS configurada (**Inválido**)
6. **DP-06:** Despliegue con cuenta de AWS mal configurada (**Inválido**)

8.1.1.2. Pruebas de visualización

La visualización efectiva de los datos es esencial para que los usuarios puedan obtener conocimientos valiosos. Estas pruebas evalúan la capacidad básica de Kibana para visualizar los datos ingestados.

Las pruebas de visualización son, conceptualmente, pruebas *end-to-end*, ya que validan tanto los procesos ETL, la persistencia de los datos, así como su visualización por parte del usuario.

Objetivo: Verificar que los datos ingestados se visualizan correctamente en Kibana.

Clases de equivalencia

- Tipo de visualización: Gráficos de barras, gráficos de líneas, tablas, paneles de control
- Complejidad de la consulta: Simple (una fuente), compleja (múltiples fuentes)
- Volumen de datos visualizados: Pocos (< 5 fuentes), muchos (≥ 5 fuentes)

Casos de prueba

1. **VIS-01:** Creación de un gráfico de barras simple con pocos datos
2. **VIS-02:** Generación de un panel de control complejo con datos de múltiples fuentes
3. **VIS-03:** Visualización de una tabla con gran volumen de datos

8.1.1.3. Análisis estático de código

El análisis estático de código es una técnica que permite detectar posibles errores y problemas en el código fuente sin necesidad de ejecutarlo. Estas pruebas evalúan la calidad del código y su cumplimiento de estándares y convenios, complejidad o la existencia de vulnerabilidades de seguridad.

En el proyecto se utiliza SonarQube, en la figura 8.1 se presenta el cuadro de mando donde se aprecia el cumplimiento de las reglas de calidad definidas por el equipo.

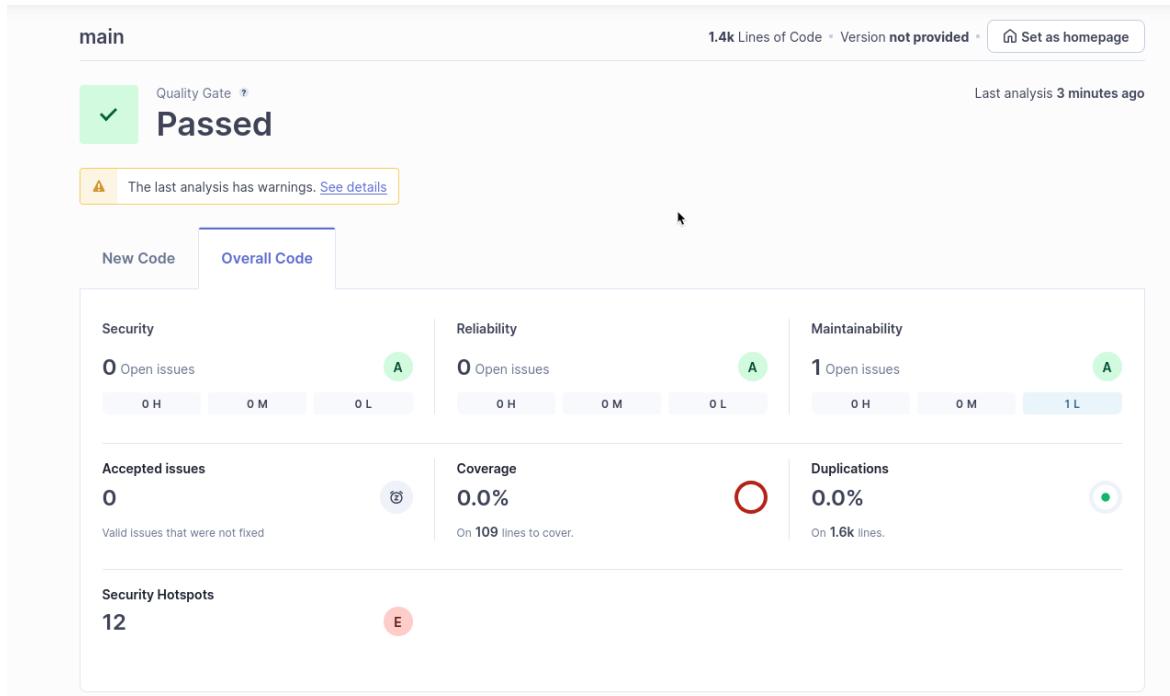


Figura 8.1: Análisis estático de código con SonarQube

SonarQube analiza la calidad de cada versión en base a un conjunto de reglas predefinidas (*quality gate*), y muestra un informe detallado con los problemas encontrados y sugerencias para su corrección.

Se utiliza la máquina especializada de SonarQube de la empresa, desplegada y mantenida por el alumno, para analizar el código de manera automatizada mediante las *pipelines* de Bitbucket cada vez que se publica un cambio al repositorio. Gracias a esta integración, se asegura la calidad del código y se podría configurar la cancelación del despliegue en caso de que se detecten problemas de seguridad o no se cumpla con la *quality gate*.

8.1.2. Pruebas de carga

Las pruebas de carga miden el rendimiento de manera anticipada mediante cargas simuladas. Esto permite anticipar posibles problemas de rendimiento bajo picos de tráfico y verificar las capacidades del sistema.

8.1.2.1. Pruebas de ingesta de datos

La ingesta de datos es una funcionalidad central del data lake. Estas pruebas están diseñadas para verificar la capacidad del sistema para procesar datos de diferentes fuentes, volúmenes y frecuencias.

Objetivo: Comprobar que el sistema ingesta correctamente datos de diferentes fuentes.

Clases de equivalencia

- Tipo de fuente: MySQL, MongoDB, Logs de Laravel, Logs de AWS ELB
- Volumen de datos: Pequeño (< 1000 registros), Mediano (1000 – 100000 registros), Grande (> 100000 registros)
- Frecuencia de ingesta: Baja (cada hora), Media (cada minuto), Alta (tiempo real)

Casos de prueba

1. **IN-01:** Ingesta de pequeño volumen de logs de MySQL
2. **IN-02:** Ingesta de gran volumen de datos de MongoDB
3. **IN-03:** Ingesta en tiempo real de logs de Laravel
4. **IN-04:** Ingesta periódica de logs de AWS ELB

8.1.2.2. Pruebas de rendimiento

El rendimiento del sistema es crítico, especialmente cuando se manejan grandes volúmenes de datos o múltiples consultas simultáneas. Estas pruebas evalúan cómo responde el sistema bajo diferentes condiciones de carga.

Para realizar estas pruebas de carga, se manipulan los scripts de ingesta de carga (ver *D Scripts de ingesta de datos con Kafka*) y se utiliza la herramienta de estrés de carga *Locust*, otra herramienta desplegada y mantenida por el alumno dentro de la nube de servicios de la empresa. Al crear picos de carga en el *backend*, se aumenta la cantidad de registros en AWS *CloudWatch*, que a su vez se ingestan a través del sistema establecido.

También se prueba la capacidad de respuesta del sistema mediante la ejecución de múltiples consultas simultáneas en Kibana mediante múltiples navegadores y usuarios.

Objetivo: Evaluar el rendimiento del sistema bajo diferentes cargas.

Clases de equivalencia

- Carga de ingesta: Baja (< 100 eventos/min), media (100 – 1000 eventos/min), alta (≥ 1000 eventos/min)
- Carga de consultas: Pocas consultas simultáneas (< 10), Muchas consultas simultáneas (≥ 10)

Casos de prueba

1. **PERF-01:** Ingesta de alta carga de eventos durante 1 hora
2. **PERF-02:** Ejecución de múltiples consultas complejas simultáneas
3. **PERF-03:** Combinación de alta carga de ingesta y consultas simultáneas

8.2. Ejecución de pruebas

La ejecución de las pruebas se lleva a cabo de manera sistemática, siguiendo el plan establecido. Para cada caso de prueba, se documentaron los siguientes aspectos:

1. Pasos de ejecución
2. Resultado esperado
3. Resultado obtenido
4. Estado (Pasado/Fallido)

A continuación, se presenta un ejemplo de ejecución de una prueba, incluyendo los pasos y resultados esperados y obtenidos, y posteriormente se presenta una tabla resumen con los resultados de todas las pruebas realizadas.

8.2.1. Ejemplo de ejecución

Despliegue desde cero con todos los recursos libres (DP-01) Esta prueba es la más básica de las pruebas de despliegue, y tiene como objetivo verificar que el sistema se puede desplegar correctamente en un entorno limpio, sin recursos previamente desplegados.

8.2.1.1. Pasos de ejecución

- Asegurar que no hay recursos desplegados previamente
- Configurar correctamente las credenciales de AWS
- Ejecutar el comando `terraform init`
- Ejecutar el comando `terraform apply`

8.2.1.2. Resultado esperado

- Terraform debe completar el despliegue sin errores
- Todos los recursos (ECS, Elasticsearch, Kibana, Logstash, Kafka) deben estar en estado "running"
- Se deben poder acceder a las interfaces web de Kibana y Elasticsearch

8.2.1.3. Resultado obtenido

- Terraform completó el despliegue sin errores
- Todos los recursos se desplegaron correctamente y están en estado "running"
- Se pudo acceder a Kibana y Elasticsearch a través de sus respectivas URLs

8.2.1.4. Estado

Pasado

Este resultado exitoso indica que el proceso de despliegue automatizado funciona correctamente y es capaz de configurar todos los componentes necesarios del sistema de manera eficiente.

8.2.2. Tabla de resultados

A continuación, se presenta una tabla resumen con los resultados de la ejecución de todas las pruebas planificadas, incluyendo las pruebas de despliegue, ingestión de datos, visualización y rendimiento.

ID	Salida esperada	Salida obtenida	Estado
DP-01	Recursos desplegados y accesibles	Recursos desplegados y accesibles	OK
DP-02	Sin cambios realizados	Sin cambios realizados	OK
DP-03	Volúmenes reutilizados	Volúmenes reutilizados	OK
DP-04	Error	Error de puertos mostrado	OK
DP-05	Error	Error de credenciales	OK
DP-06	Error	Error de credenciales	OK
IN-01	Logs en Elasticsearch	Logs en Elasticsearch	OK
IN-02	Datos ingestados sin pérdidas	Datos ingestados sin pérdidas	OK
IN-03	Logs visibles, retraso < 5s	Logs visibles, retraso < 5s	OK
IN-04	Logs cada 5min	Logs cada 5min	OK
VIS-01	Gráfico creado correctamente	Gráfico creado correctamente	OK
VIS-02	Panel con 4 fuentes de datos	Panel con 4 fuentes de datos	OK
VIS-03	1M registros en < 3s	1M registros en < 3s	OK
PERF-01	1M eventos/min sin pérdidas	1M eventos/min sin pérdidas	OK
PERF-02	20 consultas, respuesta < 2s	20 consultas, respuesta < 2s	OK
PERF-03	500k/min + 10 consultas, < 3s	500k/min + 10 consultas, < 3s	OK

Tabla 8.1: Ejecución de pruebas del sistema

La ejecución completa de las pruebas se realiza una vez completada la implementación del sistema, pero son pruebas similares a las que se han realizado durante el desarrollo del mismo, con el objetivo de asegurar que el sistema cumple con los requisitos y especificaciones establecidas.

9. Manuales

9.1. Manual de usuario

El usuario final, es decir, alguna de las partes interesadas (ver 3.1), interacciona con el sistema a través del servicio de Kibana, accesible a través de cualquier navegador web a través del subdominio DNS definido en la configuración de Terraform.

Puesto que el servicio está disponible en la red pública, es necesario autenticarse para poder acceder a él. Para ello, se debe introducir el usuario y contraseña generados durante el despliegue de la infraestructura.

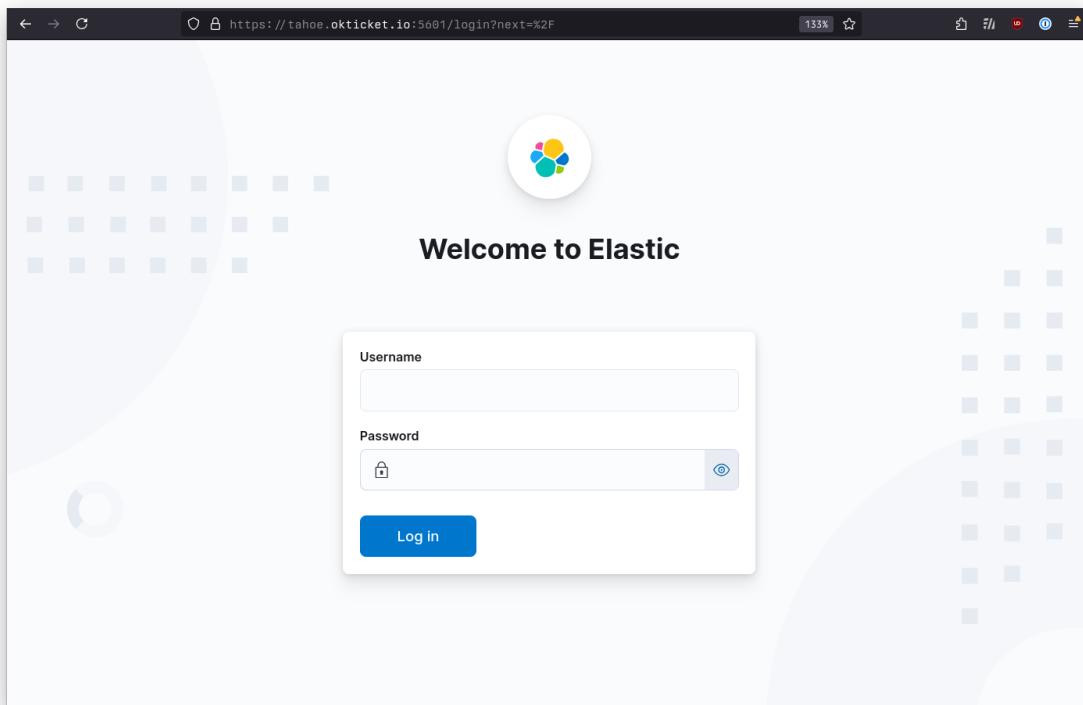


Figura 9.1: Pantalla de inicio de sesión de Kibana

NOTA: se utilizan datos y paneles de ejemplo para ilustrar el manual.

9.1.1. Usuario normal

Una vez introducidos los datos de acceso, Kibana redirige el usuario a la página por defecto configurada para su rol, que para el usuario final debería ser siempre el listado de *dashboards*.

Name, description, tags	Last updated	Actions
[Logs] Web Traffic Analyze mock web traffic log data for Elastic's website	June 17, 2024	
[System Windows Security] User Logons User logon activity dashboard.	June 12, 2024	
[System Windows Security] User Management Events User management activity.	June 12, 2024	
[System Windows Security] Group Management Events Group management activity.	June 12, 2024	
[Logs System] SSH login attempts SSH dashboard for the System integration in Logs	June 12, 2024	
[Logs System] Syslog dashboard Syslog dashboard from the Logs System integration	June 12, 2024	

Figura 9.2: Listado de dashboards de Kibana

Dentro de la lista de *dashboards*, el usuario puede seleccionar cualquiera a los que tenga acceso para visualizar los datos en tiempo real. En el caso del equipo de desarrollo, tendrán acceso a todos los paneles y podrán modificar y crear nuevos.

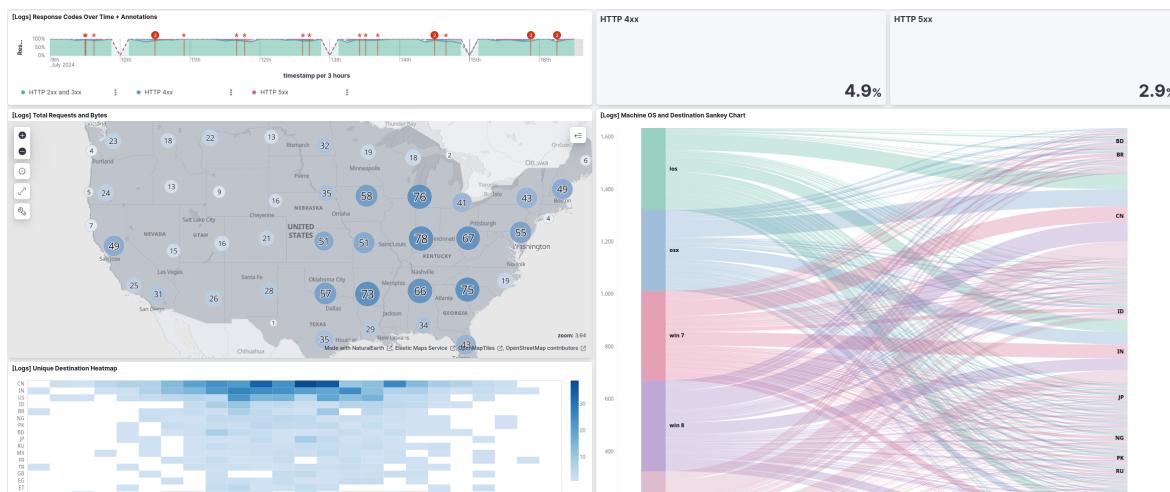


Figura 9.3: Ejemplo de dashboard de Kibana

En la parte superior de la pantalla, el usuario puede seleccionar el rango de tiempo que desea visualizar, así como el intervalo de actualización de los datos. Además, puede seleccionar el rango de tiempo de forma manual.

También existe la posibilidad de filtrar la información mostrada en el panel según diferentes campos, bien haciendo uso de los filtros predefinidos o haciendo consultas personalizadas en el campo de búsqueda con sintaxis KQL¹.

9.1.2. Usuario administrador

Aquellos usuarios con permisos de administración podrán además acceder a otras partes más avanzadas de Kibana, como la monitorización y creación de alertas o la gestión de usuarios y roles.

The screenshot shows the Kibana Settings interface. At the top, there's a header with the title 'Add data to Elasticsearch and then search, vectorize, or visualize' and a sub-instruction: 'There are endless ways to ingest and explore data with Elasticsearch, connect to your Elasticsearch instance and start indexing data'. Below this is a search bar and two buttons: 'New' and 'Manage'. The main area is titled 'Ingest your content' and contains several sections: 'API' (with a 'Create API index' button), 'Web Crawler' (with a 'Crawl URL' button), 'Connectors' (with a 'Create a connector' button), 'Language clients' (with a 'Browse clients' button), 'Upload a file' (with a 'Choose a file' button), and 'Sample data' (with an 'Import data' button). The entire interface has a light blue and white color scheme.

Figura 9.4: Opciones de fuentes de datos de Elasticsearch a través de Kibana

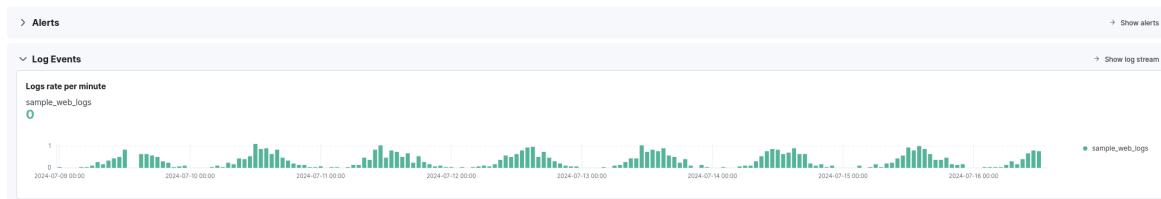


Figura 9.5: Ejemplo de alertas y métricas en Kibana

¹<https://www.elastic.co/guide/en/kibana/current/kuery-query.html>

9.2. 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 comandos².

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 WSL2³.

9.2.1. 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 oficial⁴ 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 9.6: Comprobación de la versión de Terraform

²Asumiendo que se cumplen los requisitos previos.

³<https://learn.microsoft.com/es-es/windows/wsl/about>

⁴<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 Amazon⁵.

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 9.7: Configuración de credenciales en AWS CLI

AWS pone a disposición de los usuarios una guía de inicio rápido⁶ 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 9.8: Demostración de credenciales en AWS CLI

⁵<https://aws.amazon.com/es/cli/>

⁶https://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 *7.1 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.

9.2.2. 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 9.9: 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 9.10: Despliegue exitoso de la infraestructura base

10. 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.

10.1. Resultados

El resultado del proyecto es un sistema de monitorización y análisis de datos funcional y escalable, que permite a los usuarios obtener insights valiosos a partir de la información recopilada.

Se ha logrado implementar una arquitectura robusta y flexible, basada en tecnologías modernas y en la nube, que facilita la ingesta, procesamiento y visualización de datos de manera eficiente y sencilla.

El sistema es capaz de ingestar datos de diversas fuentes, como bases de datos internas, logs de aplicaciones y servicios externos, y de presentarlos de forma clara y útil a través de Kibana.

La infraestructura se despliega y orquesta de manera automática en la nube, permitiendo una gestión sencilla y eficiente del sistema para los administradores.

Pese a que no se han completado todas las historias de usuario planificadas, se han logrado los objetivos principales del proyecto, sentando las bases para futuras iteraciones y logrando así entregar un producto mínimo viable (MVP).

El motivo de no haber completado todas las historias de usuario planificadas se debe al tamaño y naturaleza del proyecto, que ha resultado ser más complejo de lo esperado en un principio. Sin embargo, se ha logrado superar los objetivos principales y se ha entregado un producto funcional y de calidad.

10.2. Trabajo futuro

El proyecto ha sentado las bases para un sistema de monitorización y análisis de datos robusto y escalable. Sin embargo, existen áreas de mejora y ampliación que podrían ser abordadas en futuras iteraciones.

10.2.1. Historias de usuario restantes

Lo primero de todo sería completar las historias de usuario menos críticas que han quedado pendientes, como la ingesta de datos de APIs externas o el *web scraping*. Estas funcionalidades permitirían enriquecer los datos disponibles y ampliar las fuentes de información.

Nombre	Prioridad	Tamaño
Como desarrollador de Okticket, quiero que los datos contengan metadatos que faciliten su filtrado o búsqueda	P2	XS
Como trabajador de Okticket, quiero poder ver y consultar datos de empresas cliente	P2	S
Como gestor de una empresa cliente, quiero poder ver información relevante sobre mi empresa que recoja Okticket	P2	M
Como desarrollador de Okticket, quiero poder ingestar datos de APIs externas a la empresa	P2	M
Como desarrollador de Okticket, quiero poder ingestar información de páginas web externas (<i>scraping</i>)	P2	S

Tabla 10.1: Historias de usuario restantes para futuras iteraciones

Como se puede observar en la tabla 10.1, las historias de usuario restantes son las menos prioritarias (siguiendo con 4.2 *Planificación inicial*).

10.2.2. Integración de lenguaje natural para búsqueda (DSL)

Sería interesante explorar la posibilidad de integrar el sistema con un sistema de búsqueda y filtrado de datos a través de lenguaje natural, como *Natural Language Processing* (NLP)¹

Esto permitiría a los usuarios realizar consultas de manera más intuitiva y eficiente, sin necesidad de conocer la sintaxis de Kibana Query Language (KQL).

10.2.3. Aplicación de modelos de Lenguaje de Gran Escala (LLM)

Desde la empresa, se ha propuesto la posibilidad de aplicar modelos de LLM para la generación de texto automática, presumiblemente de manera agnóstica al proveedor (OpenAI, Anthropic, Google...).

Esto permitiría la generación de informes y análisis de manera automática a partir de los datos recopilados, facilitando la toma de decisiones y la comunicación de insights a los usuarios.

10.2.4. Más perspectivas futuras

Gracias a la flexibilidad del stack ELK, se podrían añadir nuevas fuentes de datos y visualizaciones, como logs de otras partes de la aplicación (gestor web, otros servicios internos como Hubspot o Holded, aplicación móvil...) o visualizaciones más avanzadas y personalizadas.

Lo bueno de haber diseñado la arquitectura de la manera que se ha hecho es que se pueden añadir nuevas funcionalidades sin necesidad de modificar la infraestructura existente, simplemente añadiendo nuevos servicios y configuraciones.

La escalabilidad del sistema también es un punto a tener en cuenta. En caso de necesitar más capacidad de procesamiento o almacenamiento, se podría establecer un sistema de autoescalabilidad en base a las definiciones ya realizadas.

Por último, se podría explotar la funcionalidad del stack de generar alertas en base a la información ingestada y procesada, para notificar a los usuarios de eventos críticos o anomalías detectadas en los datos.

¹<https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html>

10.3. Conclusiones y retrospectiva

El desarrollo de este proyecto me ha permitido, a nivel personal, adquirir conocimientos y habilidades muy útiles en áreas que tienen una gran demanda en el mercado laboral actual, como la ingesta y procesamiento de datos, la orquestación de servicios en la nube y la experiencia con proveedores cloud como AWS.

A nivel técnico, he aprendido a trabajar con tecnologías modernas y potentes, como Kafka, Logstash, Elasticsearch y Kibana, que me han permitido implementar una solución robusta y escalable para la monitorización y análisis de datos.

Además, he tenido la oportunidad de trabajar con metodologías ágiles y de gestión de proyectos, como Scrum, en proyectos reales con repercusiones y resultados tangibles.

En cuanto a la retrospectiva del proyecto, considero que el tiempo dedicado ha merecido la pena, pero podría haber sido gestionado u organizado diferente.

Anexos

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       - ./setup.sh:/usr/local/bin/setup.sh
22     user: root
23     container_name: setup
24     command: ["/bin/bash", "/usr/local/bin/setup.sh"]
25     healthcheck:
26       test: [ "CMD-SHELL", "[ -f config/certs/es01/es01.crt ]" ]
27       interval: 5s
28       timeout: 10s
29       retries: 10

30   zookeeper:
31     container_name: zookeeper
32     image: confluentinc/cp-zookeeper:latest
33     environment:
34       ZOOKEEPER_CLIENT_PORT: 2181
35       ZOOKEEPER_TICK_TIME: 2000
36       ZOO_LOG4J_PROP: WARN,CONSOLE
37     ports:
38       - 2181:2181

39

40   kafka:
41     container_name: kafka
42     image: confluentinc/cp-kafka:latest
```

```
43     depends_on:
44         - zookeeper
45         - es01
46     ports:
47         - 9092:9092
48         - 29092:29092
49     environment:
50         KAFKA_BROKER_ID: 1
51         KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
52         KAFKA_ADVERTISED_LISTENERS:
53             ↳ LISTENER_DOCKER_INTERNAL://kafka:29092,LISTENER_DOCKER_EXTERNAL://localhost:29092
54             KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
55             ↳ LISTENER_DOCKER_INTERNAL:PLAINTEXT,LISTENER_DOCKER_EXTERNAL:PLAINTEXT
56             KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_DOCKER_INTERNAL
57             KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
58             KAFKA_LOG4J_ROOT_LOGLEVEL: WARN
59             KAFKA_TOOLS_LOG4J_LOGLEVEL: ERROR
60             KAFKA_LOG4J_LOGGERS:
61                 ↳ 'kafka=WARN,kafka.controller=WARN,kafka.log.LogCleaner=WARN,state.change.log=WARN'
62
63     es01:
64         image:
65             ↳ docker.elastic.co/elasticsearch/elasticsearch:${STACK_VERSION}
66         container_name: es01
67         restart: unless-stopped
68         depends_on:
69             - setup
70         environment:
71             - node.name=es01
72             - cluster.name=${CLUSTER_NAME}
73             - discovery.type=single-node
74             - bootstrap.memory_lock=true
75             - logger.level=WARN
76             - ELASTIC_PASSWORD=${ELASTIC_PASSWORD}
77             - xpack.security.enabled=true
78             - xpack.security.http.ssl.enabled=true
79             - xpack.security.http.ssl.key=certs/es01/es01.key
80             - xpack.security.http.ssl.certificate=certs/es01/es01.crt
81             -
82             ↳ xpack.security.http.ssl.certificateAuthorities=certs/ca/ca.crt
83             - xpack.security.transport.ssl.enabled=true
84             - xpack.security.transport.ssl.key=certs/es01/es01.key
85             - xpack.security.transport.ssl.certificate=certs/es01/es01.crt
86             -
87             ↳ xpack.security.transport.ssl.certificateAuthorities=certs/ca/ca.crt
88             - xpack.security.transport.ssl.verifyMode=certificate
```

```
83      ulimits:
84        memlock:
85          soft: -1
86          hard: -1
87        nofile:
88          soft: 65536
89          hard: 65536
90      cap_add:
91        - IPC_LOCK
92      labels:
93        co.elastic.logs/module: elasticsearch
94        co.elastic.metrics/module: elasticsearch
95      volumes:
96        - elastic-data:/usr/share/elasticsearch/data
97        - certs:/usr/share/elasticsearch/config/certs
98      ports:
99        - 9200:9200
100     healthcheck:
101       test:
102         [
103           "CMD-SHELL",
104           "curl -s --cacert config/certs/ca/ca.crt"
105           ↪ https://localhost:9200 | grep -q 'missing authentication'
106           ↪ credentials"
107         ]
108       interval: 10s
109       timeout: 10s
110       retries: 10
111
112     kibana:
113       container_name: kibana
114       image: docker.elastic.co/kibana/kibana:${STACK_VERSION}
115       restart: unless-stopped
116       volumes:
117         - kibana-data:/usr/share/kibana/data
118         - certs:/usr/share/kibana/config/certs
119       environment:
120         SERVER_NAME: kibana
121         SERVER_PORT: 5601
122         SERVER_HOST: 0.0.0.0
123         ELASTICSEARCH_HOSTS: https://es01:9200
124         ELASTICSEARCH_USERNAME: kibana_system
125         ELASTICSEARCH_PASSWORD: ${KIBANA_PASSWORD}
126         ELASTICSEARCH_SSL_CERTIFICATEAUTHORITIES: config/certs/ca/ca.crt
127         LOGGING_ROOT_LEVEL: warn
128         XPACK_ENCRYPTEDSAVEDOBJECTS_ENCRYPTIONKEY: ${KEY}
```

```
127     XPACK_REPORTING_ENCRYPTIONKEY: ${KEY}
128     XPACK_SECURITY_ENCRYPTIONKEY: ${KEY}
129
130   links:
131     - es01
132
133   depends_on:
134     - setup
135     - es01
136
137   labels:
138     co.elastic.logs/module: kibana
139     co.elastic.metrics/module: kibana
140
141   ports:
142     - 5601:5601
143
144   healthcheck:
145     test:
146       [
147         "CMD-SHELL",
148         "curl -s -I http://localhost:5601 | grep -q 'HTTP/1.1 302"
149         ↪ Found"
150       ]
151     interval: 10s
152     timeout: 10s
153     retries: 10
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
```

```
171   labels:  
172     co.elastic.logs/module: logstash  
173     co.elastic.metrics/module: logstash  
174   links:  
175     - es01  
176     - 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=oxt-tahoe-cluster  
6 PROJECT_NAME=oxt-tahoe  
7 KEY=something_at_least_32_characters_long
```

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

```
1 input {  
2   kafka {  
3     bootstrap_servers => "kafka:29092"  
4     topics => ["laravel-logs"]  
5   }  
6 }  
7  
8 output {  
9   elasticsearch {  
10    hosts => ["https://es01:9200"]  
11    user => "elastic"  
12    password => "${ELASTIC_PASSWORD}"  
13    ssl => true  
14    cacert => "certs/ca/ca.crt"  
15  }  
16 }
```

Listado A.3: Fichero de configuración de Logstash

```
1 setup:  
2   image: docker.elastic.co/elasticsearch/elasticsearch:${  
3     ← STACK_VERSION}  
3   volumes:  
4     - certs:/usr/share/elasticsearch/config/certs  
5   user: root  
6   container_name: setup  
7   command: >  
8     bash -c '  
9       if [ x${ELASTIC_PASSWORD} == x ]; then
```

```
10         echo "Set the ELASTIC_PASSWORD environment variable in the .\n"
11         ↳ env file";
12         exit 1;
13     elif [ x${KIBANA_PASSWORD} == x ]; then
14         echo "Set the KIBANA_PASSWORD environment variable in the .\n"
15         ↳ env 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 config/
21         ↳ 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 config/
42         ↳ certs/certs.zip --in config/certs/instances.yml --ca-cert config/
43         ↳ /certs/ca/ca.crt --ca-key config/certs/ca/ca.key;
44         unzip config/certs/certs.zip -d config/certs;
45     fi;
46     echo "Setting file permissions"
47     chown -R root:root config/certs;
48     find . -type d -exec chmod 750 \{\}\ \;;
49     find . -type f -exec chmod 640 \{\}\ \;;
50     echo "Waiting for Elasticsearch availability";
51     until curl -s --cacert config/certs/ca/ca.crt https://es01
52         ↳ :9200 | grep -q "missing authentication credentials"; do sleep
53         ↳ 1; done;
54     echo "Setting kibana_system password";
55     until curl -s -X POST --cacert config/certs/ca/ca.crt -u "
```

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

Listado A.4: Script shell de preparación de Elasticsearch

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

C.1. Recursos de AWS

C.1.1. Fichero principal

```
1 provider "aws" {
2   region = var.region
3 }
4
5 resource "aws_ecs_cluster" "cluster" {
6   name = var.cluster_name
7 }
8
9 resource "aws_cloudwatch_log_group" "elastic_services" {
10  name = var.log_group
11  retention_in_days = 7
12 }
13
14 resource "aws_s3_bucket" "access_logs" {
15  bucket = "tahoe-access-logs-${local.hash}"
16 }
17
18 resource "aws_s3_bucket_policy" "access_logs" {
19  bucket = aws_s3_bucket.access_logs.id
20
21  policy = jsonencode({
22    Version = "2012-10-17"
23    Statement = [
24      {
25        Effect = "Allow"
26        Principal = {
27          AWS = "arn:aws:iam::${var.elb_account_id}:root"
28        }
29        Action = "s3:PutObject"
30        Resource = "${aws_s3_bucket.access_logs.arn}/*"
31      }
32    ]
33  })
34 }
35
36 resource "aws_s3_bucket_server_side_encryption_configuration" " "
37  ↢ access_logs" {
38  bucket = aws_s3_bucket.access_logs.id
```

```
38
39   rule {
40     apply_server_side_encryption_by_default {
41       sse_algorithm = "AES256"
42     }
43   }
44 }
```

Listado C.1: Fichero *main.tf* de despliegue cloud

C.1.2. Variables

```
1 variable "log_group" {default = "/ecs/elastic-services"}
2 variable "log_driver" {default = "awslogs"}
3
4 variable "launch_type" {default = "FARGATE"}
5 variable "cluster_name" {default = "tahoe"}
6 variable "region" {default = "eu-north-1"}
7 variable "domain_name" {default = "okticket.io"}
8 variable "route53_zone_id" {default = "Z01935951GG68PCUSU149"}
9 variable "elb_account_id" {default = "897822967062"}
10
11 variable "stack_version" {
12   type = string
13   default = "8.12.2"
14   description = "Version de la pila ELK"
15 }
16 variable "kibana_user" {
17   type = string
18   default = "kibana_user"
19   description = "Nombre de usuario de elasticsearch para Kibana"
20 }
21 variable "health_check_user" {
22   type = string
23   default = "aws_health_check_tahoe"
24   description = "Nombre del usuario de health-check"
25 }
26
27 locals {
28   health_check_password = random_string.random_service_password.result
29   kibana_password = random_string.random_service_password.result
30   random_key = random_string.random_key.result
31   elastic_password = random_string.random_password.result
32   hash = random_string.hash.result
33
34   kibana_url = "tahoe.${var.domain_name}"
35   elastic_url = "elastic.${var.domain_name}"
36   kafka_url = "kafka.${var.domain_name}"
37   logstash_url = "logstash.${var.domain_name}"
38 }
39
40 resource "random_string" "random_key" {
41   length  = 64
42   special = false
43 }
44 resource "random_string" "random_password" {
```

```
45   length  = 16
46   special = false
47 }
48 resource "random_string" "random_service_password" {
49   length  = 16
50   special = false
51 }
52 resource "random_string" "hash" {
53   length = 4
54   special = false
55   upper = false
56 }
57
58 variable "zookeeper_port" {default = 2181}
59 variable "kafka_port" {default = 9092}
60 variable "elastic_port" {default = 9200}
61 variable "kibana_port" {default = 5601}
62 variable "logstash_port" {default = 5044}
```

Listado C.2: Fichero *variables.tf* de despliegue cloud

C.1.3. Salidas

```
1 output "passwords" {
2   value = "e: ${local.elastic_password}, k: ${local.kibana_password},
3   ↪ h: ${local.health_check_password}"
4 }
5 output "url_elastic" {
6   value = "https://${aws_lb.elastic.dns_name}:${var.elastic_port} (
7   ↪ https://${local.elastic_url}:${var.elastic_port})"
8 }
9 output "url_kibana" {
10  value = "https://${aws_lb.kibana.dns_name}:${var.kibana_port} (https
11  ↪ ://${local.kibana_url}:${var.kibana_port})"
12 }
13 output "url_kafka" {
14  value = "http://${aws_lb.kafka.dns_name}:${var.kafka_port} (http://${
15  ↪ ${local.kafka_url}:${var.kafka_port})"
16 }
17 output "hash" {
18   value = local.hash
19 }
```

Listado C.3: Fichero *outputs.tf* de despliegue cloud

C.1.4. Volúmenes lógicos (EFS)

```
1 # EFS File Systems
2
3 resource "aws_efs_file_system" "elastic_data" {
4   creation_token = "elastic_data"
5
6   tags = {
7     Name = "tahoe-efs-elastic"
8   }
9 }
10
11 resource "aws_efs_file_system" "kibana_data" {
12   creation_token = "kibana_data"
13
14   tags = {
15     Name = "tahoe-efs-kibana"
16   }
17 }
18
19 resource "aws_efs_file_system" "logstash_data" {
20   creation_token = "logstash_data"
21
22   tags = {
23     Name = "tahoe-efs-logstash"
24   }
25 }
26
27 # EFS Policies
28
29 resource "aws_efs_file_system_policy" "elastic_data_policy" {
30   file_system_id = aws_efs_file_system.elastic_data.id
31   policy        = data.aws_iam_policy_document.efs.json
32 }
33
34 resource "aws_efs_file_system_policy" "kibana_data_policy" {
35   file_system_id = aws_efs_file_system.kibana_data.id
36   policy        = data.aws_iam_policy_document.efs.json
37 }
38
39 resource "aws_efs_file_system_policy" "logstash_data_policy" {
40   file_system_id = aws_efs_file_system.logstash_data.id
41   policy        = data.aws_iam_policy_document.efs.json
42 }
43
44 # EFS Mount Targets
```

```
45
46 resource "aws_efs_mount_target" "mount_elastic_data" {
47   file_system_id  = aws_efs_file_system.elastic_data.id
48   subnet_id       = aws_subnet.private.id
49   security_groups = [aws_security_group.elastic.id]
50 }
51
52 resource "aws_efs_mount_target" "mount_kibana_data" {
53   file_system_id  = aws_efs_file_system.kibana_data.id
54   subnet_id       = aws_subnet.private.id
55   security_groups = [aws_security_group.kibana.id]
56 }
57
58 resource "aws_efs_mount_target" "mount_logstash_data" {
59   file_system_id  = aws_efs_file_system.logstash_data.id
60   subnet_id       = aws_subnet.private.id
61   security_groups = [aws_security_group.logstash.id]
62 }
```

Listado C.4: Fichero *efs.tf* de despliegue cloud

C.1.5. Roles, usuarios y políticas (IAM)

```
1 resource "aws_iam_role" "ecs_task_execution" {
2   name = "ecsTaskExecutionRole"
3
4   assume_role_policy = jsonencode({
5     Version = "2012-10-17",
6     Statement = [
7       Action = "sts:AssumeRole",
8       Effect = "Allow",
9       Principal = {
10         Service = "ecs-tasks.amazonaws.com"
11       }
12     ]
13   })
14
15   managed_policy_arns = [
16     "arn:aws:iam::aws:policy/service-role/
17     ↪ AmazonECSTaskExecutionRolePolicy",
18     "arn:aws:iam::aws:policy/AmazonElasticFileSystemClientFullAccess",
19     "arn:aws:iam::aws:policy/SecretsManagerReadWrite"
20   ]
21
22 resource "aws_iam_role_policy" "ecs_task_secrets_access" {
23   name = "ecs_task_secrets_access"
24   role = aws_iam_role.ecs_task_execution.id
25
26   policy = jsonencode({
27     Version = "2012-10-17"
28     Statement = [
29       {
30         Effect = "Allow"
31         Action = [ "secretsmanager:GetSecretValue" ]
32         Resource = [ aws_secretsmanager_secret.es_certs.arn ]
33       }
34     ]
35   })
36 }
37
38 resource "aws_iam_role_policy" "ecs_exec_policy" {
39   name = "ecs_exec_policy"
40   role = aws_iam_role.ecs_task_execution.id
41
42   policy = jsonencode({
43     Version = "2012-10-17"
```

```
44 Statement = [
45   {
46     Effect = "Allow"
47     Action = [
48       "ssmmessages:CreateControlChannel",
49       "ssmmessages:CreateDataChannel",
50       "ssmmessages:OpenControlChannel",
51       "ssmmessages:OpenDataChannel"
52     ]
53     Resource = ["*"]
54   }
55 ]
56 }
57 ]
58
59 data "aws_iam_policy_document" "efs" {
60   statement {
61     sid      = "ExampleStatement01"
62     effect   = "Allow"
63
64     principals {
65       type        = "AWS"
66       identifiers = ["*"]
67     }
68
69     actions = [
70       "elasticfilesystem:ClientMount",
71       "elasticfilesystem:ClientWrite",
72     ]
73
74     resources = ["*"]
75
76     condition {
77       test      = "Bool"
78       variable = "aws:SecureTransport"
79       values    = ["true"]
80     }
81   }
82 }
```

Listado C.5: Fichero *iam.tf* de despliegue cloud

C.1.6. Secretos (Secret Manager)

```
1 resource "aws_secretsmanager_secret" "es_certs" {
2   name          = "tahoe-certs-${local.hash}"
3   description  = "Claves para el data lake de OKT (codename: tahoe)"
4 }
5
6 resource "aws_secretsmanager_secret_version" "es_certs" {
7   secret_id = aws_secretsmanager_secret.es_certs.id
8   secret_string = jsonencode({
9     "ca.crt"           = base64encode(file("certs/ca/ca.crt")),
10    "es01.key"         = base64encode(file("certs/es01/es01.key")),
11    "es01.crt"         = base64encode(file("certs/es01/es01.crt")),
12    "kibana.crt"       = base64encode(file("certs/kibana/kibana.crt"))
13    ↪ ),
14    "kibana.key"       = base64encode(file("certs/kibana/kibana.key"))
15    ↪ ),
16    "xpack_key"        = local.random_key,
17    "kibana_user"      = var.kibana_user,
18    "kibana_pass"       = local.kibana_password,
19    "elastic_pass"      = local.elastic_password,
20    "health_check_user" = var.health_check_user,
21    "health_check_pass" = local.health_check_password,
22  })
23 }
```

Listado C.6: Fichero *secrets.tf* de despliegue cloud

C.1.7. Recursos de red

```
1 resource "aws_vpc" "main" {
2   cidr_block          = "10.0.0.0/16"
3   enable_dns_support = true
4   enable_dns_hostnames = true
5
6   tags = {
7     Name = "tahoe-vpc"
8   }
9 }
10
11 resource "aws_subnet" "public_a" {
12   vpc_id              = aws_vpc.main.id
13   cidr_block          = "10.0.1.0/24"
14   map_public_ip_on_launch = true
15   availability_zone    = "${var.region}a"
16
17   tags = {
18     Name = "tahoe-subnet-public-a"
19   }
20 }
21
22 resource "aws_subnet" "public_b" {
23   vpc_id              = aws_vpc.main.id
24   cidr_block          = "10.0.3.0/24"
25   map_public_ip_on_launch = true
26   availability_zone    = "${var.region}b"
27
28   tags = {
29     Name = "tahoe-subnet-public-b"
30   }
31 }
32
33 resource "aws_subnet" "private" {
34   vpc_id              = aws_vpc.main.id
35   cidr_block          = "10.0.2.0/24"
36   availability_zone    = "${var.region}b"
37
38   tags = {
39     Name = "tahoe-subnet-private"
40   }
41 }
42
43 resource "aws_internet_gateway" "gw" {
44   vpc_id = aws_vpc.main.id
```

```
45
46   tags = {
47     Name = "tahoe-igw"
48   }
49 }
50
51 resource "aws_eip" "nat_eip" {
52   domain = "vpc"
53 }
54
55 resource "aws_nat_gateway" "nat" {
56   allocation_id = aws_eip.nat_eip.id
57   subnet_id      = aws_subnet.public_a.id
58
59   tags = {
60     Name = "tahoe-nat-gw"
61   }
62 }
63
64 resource "aws_route_table" "public_a" {
65   vpc_id = aws_vpc.main.id
66
67   route {
68     cidr_block = "0.0.0.0/0"
69     gateway_id = aws_internet_gateway.gw.id
70   }
71
72   tags = {
73     Name = "tahoe-rt-public-a"
74   }
75 }
76
77 resource "aws_route_table" "public_b" {
78   vpc_id = aws_vpc.main.id
79
80   route {
81     cidr_block = "0.0.0.0/0"
82     gateway_id = aws_internet_gateway.gw.id
83   }
84
85   tags = {
86     Name = "tahoe-rt-public-b"
87   }
88 }
89
90 resource "aws_route_table" "private" {
```

```
91 vpc_id = aws_vpc.main.id
92
93 route {
94     cidr_block      = "0.0.0.0/0"
95     nat_gateway_id = aws_nat_gateway.nat.id
96 }
97
98 tags = {
99     Name = "tahoe-rt-private"
100 }
101 }
102
103 resource "aws_route_table_association" "public_a_association" {
104     subnet_id      = aws_subnet.public_a.id
105     route_table_id = aws_route_table.public_a.id
106
107     depends_on = [aws_internet_gateway.gw]
108 }
109
110 resource "aws_route_table_association" "public_b_association" {
111     subnet_id      = aws_subnet.public_b.id
112     route_table_id = aws_route_table.public_b.id
113
114     depends_on = [aws_internet_gateway.gw]
115 }
116
117 resource "aws_route_table_association" "private_association" {
118     subnet_id      = aws_subnet.private.id
119     route_table_id = aws_route_table.private.id
120
121     depends_on = [aws_nat_gateway.nat]
122 }
```

Listado C.7: Fichero *network.tf* de despliegue cloud

C.1.8. Grupos de seguridad (SG)

```
1 resource "aws_security_group" "efs" {
2   name          = "tahoe-efs-sg"
3   description   = "Permitir trafico a los servicios de EFS"
4   vpc_id        = aws_vpc.main.id
5
6   ingress {
7     from_port    = 2049
8     to_port      = 2049
9     protocol     = "tcp"
10    cidr_blocks = ["0.0.0.0/0"]
11  }
12
13  egress {
14    from_port    = 0
15    to_port      = 0
16    protocol     = "-1"
17    cidr_blocks = ["0.0.0.0/0"]
18  }
19 }
20
21 resource "aws_security_group" "kafka" {
22   vpc_id = aws_vpc.main.id
23   name   = "tahoe-kafka-sg"
24   description = "Permitir trafico a los stacks Kafka/Zookeeper"
25
26   ingress {
27     from_port    = var.kafka_port
28     to_port      = var.kafka_port
29     protocol     = "tcp"
30     cidr_blocks = ["0.0.0.0/0"]
31   }
32
33   ingress {
34     from_port    = var.zookeeper_port
35     to_port      = var.zookeeper_port
36     protocol     = "tcp"
37     cidr_blocks = ["0.0.0.0/0"]
38   }
39
40   egress {
41     from_port    = 0
42     to_port      = 0
43     protocol     = "-1"
44     cidr_blocks = ["0.0.0.0/0"]
```

```
45     }
46 }
47
48 resource "aws_security_group" "elastic" {
49   vpc_id = aws_vpc.main.id
50   name   = "tahoe-elastic-sg"
51   description = "Permitir trafico a los nodos de Elasticsearch"
52
53   ingress {
54     from_port    = var.elastic_port
55     to_port      = var.elastic_port
56     protocol     = "-1"
57     cidr_blocks = ["0.0.0.0/0"]
58   }
59
60   egress {
61     from_port    = 0
62     to_port      = 0
63     protocol     = "-1"
64     cidr_blocks = ["0.0.0.0/0"]
65   }
66 }
67
68 resource "aws_security_group" "logstash" {
69   vpc_id = aws_vpc.main.id
70   name   = "tahoe-logstash-sg"
71   description = "Permitir trafico a los nodos de Logstash"
72
73   ingress {
74     from_port    = var.logstash_port
75     to_port      = var.logstash_port
76     protocol     = "tcp"
77     cidr_blocks = ["0.0.0.0/0"]
78   }
79
80   egress {
81     from_port    = 0
82     to_port      = 0
83     protocol     = "-1"
84     cidr_blocks = ["0.0.0.0/0"]
85   }
86 }
```

Listado C.8: Fichero *secrets.tf* de despliegue cloud

C.2. Servicios de ELK

C.2.1. Elasticsearch

```
1 resource "aws_ecs_task_definition" "elastic" {
2     family           = "elastic"
3     network_mode    = "awsvpc"
4     requires_compatibility = [var.launch_type]
5     cpu              = "2048"
6     memory           = "4096"
7     execution_role_arn = aws_iam_role.ecs_task_execution.arn
8     task_role_arn    = aws_iam_role.ecs_task_execution.arn
9
10    container_definitions = jsonencode([
11        {
12            name      = "es01"
13            image     = "docker.elastic.co/elasticsearch/elasticsearch:${var
14            .stack_version}"
15            cpu       = 2048
16            memory   = 4096
17            essential = true
18            environment = [
19                { name = "cluster.name", value = var.cluster_name },
20                { name = "xpack.security.enabled", value = "true" },
21                { name = "xpack.security.http.ssl.enabled", value = "true" },
22                { name = "xpack.security.transport.ssl.enabled", value = "true
23                " },
24                { name = "xpack.security.http.ssl.key", value = "/usr/share/
25                elasticsearch/config/certs/es01.key" },
26                { name = "xpack.security.http.ssl.certificate", value = "/usr/
27                share/elasticsearch/config/certs/es01.crt" },
28                { name = "xpack.security.http.ssl.certificateAuthorities",
29                value = "/usr/share/elasticsearch/config/certs/ca.crt" },
30                { name = "xpack.security.transport.ssl.key", value = "/usr/
31                share/elasticsearch/config/certs/es01.key" },
32                { name = "xpack.security.transport.ssl.certificate", value =
33                "/usr/share/elasticsearch/config/certs/es01.crt" },
34                { name = "xpack.security.transport.ssl.certificateAuthorities
35                ", value = "/usr/share/elasticsearch/config/certs/ca.crt" },
36                { name = "discovery.type", value = "single-node" },
37                # { name = "ES_JAVA_OPTS", value = "-Xms4g -Xmx4g" },
38                { name = "bootstrap.memory_lock", value = "true" },
39                # { name = "logger.level", value = "WARN" },
40                { name = "xpack.security.authc.anonymous.username", value =
41                var.health_check_user },
```

```
33      { name = "xpack.security.authc.anonymous.roles", value = "
34      ↳ monitoring_user" },
35      { name = "xpack.security.authc.anonymous.authz_exception",
36      ↳ value = "true" }
37    ]
38    secrets = [
39      {
40        name       = "CA_CRT"
41        valueFrom = "${aws_secretsmanager_secret.es_certs.arn}:ca.
42        ↳ crt::"
43      },
44      {
45        name       = "ES01_KEY"
46        valueFrom = "${aws_secretsmanager_secret.es_certs.arn}:es01.
47        ↳ key::"
48      },
49      {
50        name       = "ES01_CRT"
51        valueFrom = "${aws_secretsmanager_secret.es_certs.arn}:es01.
52        ↳ crt::"
53      },
54      {
55        name       = "ELASTIC_PASSWORD"
56        valueFrom = "${aws_secretsmanager_secret.es_certs.arn}:
57        ↳ elastic_pass::"
58      },
59      {
60        name       = "HEALTH_CHECK_PASSWORD"
61        valueFrom = "${aws_secretsmanager_secret.es_certs.arn}:
62        ↳ health_check_pass::"
63      },
64      {
65        name       = "KIBANA_PASSWORD"
66        valueFrom = "${aws_secretsmanager_secret.es_certs.arn}:
67        ↳ kibana_pass::"
68      }
```

```
69      ]
70      entrypoint = [
71          "/bin/sh",
72          "-c",
73          <<-EOT
74          #!/bin/bash
75          set -e
76
77          echo "Configurando credenciales..."
78          mkdir -p /usr/share/elasticsearch/config/certs
79          echo $CA_CRT | base64 -d > /usr/share/elasticsearch/config/
80          ↪ certs/ca.crt
81          echo $ES01_KEY | base64 -d > /usr/share/elasticsearch/config/
82          ↪ certs/es01.key
83          echo $ES01_CRT | base64 -d > /usr/share/elasticsearch/config/
84          ↪ certs/es01.crt
85          chmod 400 /usr/share/elasticsearch/config/certs/es01.key
86
87
88          # establecer vm.max_map_count
89          echo "Estableciendo vm.max_map_count..."
90          sysctl -w vm.max_map_count=262144
91
92
93          # iniciar elasticsearch
94          echo "Iniciando Elasticsearch..."
95          /usr/local/bin/docker-entrypoint.sh &
96
97
98          # esperar a que elasticsearch este listo
99          echo "Esperando a que Elasticsearch este listo..."
100         until curl -s -XGET https://localhost:9200 -u elastic:
101         ↪ $ELASTIC_PASSWORD -k > /dev/null; do
102             sleep 1
103             done
104
105             # crear usuarios
106             echo "Creando usuarios..."
107             curl -s -X POST -u "elastic:$ELASTIC_PASSWORD" -H "Content-
108             ↪ Type: application/json" https://localhost:9200/_security/user/
109             ↪ kibana_system/_password -d '{"password": "$KIBANA_PASSWORD"}'
110             curl -s -X POST -u elastic:$ELASTIC_PASSWORD -H "Content-Type:
111             ↪ application/json" https://localhost:9200/_security/user/
112             ↪ $HEALTH_CHECK_USER -d '{"password": "$HEALTH_CHECK_PASS", "roles
113             ↪ ": ["monitoring_user"]}"'
114
115             wait
116             EOT
117         ]
```

```
106      # mountPoints = [
107      #
108      #     sourceVolume  = "elastic-data"
109      #     containerPath = "/usr/share/elasticsearch/data"
110      #     readOnly      = false
111      #   }
112      #
113      portMappings = [
114      {
115          containerPort = var.elastic_port,
116          hostPort      = var.elastic_port
117      }
118  ]
119  logConfiguration = {
120      logDriver = var.log_driver
121      options = {
122          "awslogs-group"           = var.log_group
123          "awslogs-region"          = var.region
124          "awslogs-stream-prefix"  = "elastic"
125      }
126  }
127  ulimits = [
128  {
129      name      = "memlock"
130      softLimit = -1
131      hardLimit = -1
132  },
133  {
134      name      = "nofile"
135      softLimit = 65536
136      hardLimit = 65536
137  }
138 ]
139 }
140 ])
141
142 volume {
143     name = "elastic-data"
144     efs_volume_configuration {
145         file_system_id = aws_efs_file_system.elastic_data.id
146         root_directory = "/"
147     }
148 }
149 }
150
151 resource "aws_ecs_service" "elastic" {
```

```
152   name                  = "elastic"
153   cluster                = aws_ecs_cluster.cluster.id
154   task_definition         = aws_ecs_task_definition.elastic.arn
155   desired_count           = 1
156   launch_type             = var.launch_type
157   enable_execute_command  = true
158
159   network_configuration {
160     subnets               = [aws_subnet.private.id]
161     security_groups       = [aws_security_group.elastic.id]
162     assign_public_ip      = true
163   }
164
165   load_balancer {
166     target_group_arn      = aws_lb_target_group.elastic.arn
167     container_name        = "es01"
168     container_port         = var.elastic_port
169   }
170
171   depends_on = [
172     aws_lb_listener.elastic,
173     aws_ecs_task_definition.elastic
174   ]
175 }
176
177 resource "aws_lb" "elastic" {
178   name                  = "tahoe-alb-elastic"
179   internal              = false
180   load_balancer_type    = "application"
181   security_groups        = [aws_security_group.elastic.id]
182   subnets                = [aws_subnet.public_a.id, aws_subnet.public_b.id
183   ↵ ]
184
185   access_logs {
186     bucket    = aws_s3_bucket.access_logs.bucket
187     prefix    = "elastic"
188     enabled   = true
189   }
190 }
191
192 resource "aws_lb_target_group" "elastic" {
193   name      = "tahoe-tg-elastic"
194   port      = var.elastic_port
195   protocol  = "HTTPS"
196   vpc_id    = aws_vpc.main.id
197   target_type = "ip"
```

```
197
198     health_check {
199         enabled          = true
200         path             = "/"
201         protocol         = "HTTPS"
202         matcher          = "200"
203         interval         = 300
204         timeout          = 60
205         healthy_threshold = 2
206         unhealthy_threshold = 5
207         port              = tostring(var.elastic_port)
208     }
209 }
210
211 resource "aws_lb_listener" "elastic" {
212     load_balancer_arn = aws_lb.elastic.arn
213     port             = var.elastic_port
214     protocol         = "HTTPS"
215     ssl_policy       = "ELBSecurityPolicy-2016-08"
216     certificate_arn  = aws_acm_certificate.elastic.arn
217
218     default_action {
219         type          = "forward"
220         target_group_arn = aws_lb_target_group.elastic.arn
221     }
222 }
223
224 resource "aws_lb_listener" "elastic_https" {
225     load_balancer_arn = aws_lb.elastic.arn
226     port             = 443
227     protocol         = "HTTPS"
228     ssl_policy       = "ELBSecurityPolicy-2016-08"
229     certificate_arn  = aws_acm_certificate.elastic.arn
230
231     default_action {
232         type = "redirect"
233         redirect {
234             port          = tostring(var.elastic_port)
235             protocol      = "HTTPS"
236             status_code   = "HTTP_301"
237         }
238     }
239 }
240
241 resource "aws_acm_certificate" "elastic" {
242     domain_name      = local.elastic_url
```

```
243 validation_method = "DNS"
244
245 lifecycle {
246   create_before_destroy = true
247 }
248 }
249
250 resource "aws_route53_record" "elastic" {
251   zone_id = var.route53_zone_id
252   name    = local.elastic_url
253   type    = "A"
254
255   alias {
256     name          = aws_lb.elastic.dns_name
257     zone_id       = aws_lb.elastic.zone_id
258     evaluate_target_health = false
259   }
260 }
261
262 resource "aws_route53_record" "elastic_validation" {
263   for_each = {
264     for dvo in aws_acm_certificate.elastic.domain_validation_options :
265       ↪ dvo.domain_name => {
266         name    = dvo.resource_record_name
267         record = dvo.resource_record_value
268         type    = dvo.resource_record_type
269       }
270
271     allow_overwrite = true
272     name          = each.value.name
273     records        = [each.value.record]
274     ttl            = 60
275     type           = each.value.type
276     zone_id        = var.route53_zone_id
277   }
278
279 resource "aws_acm_certificate_validation" "elastic" {
280   certificate_arn      = aws_acm_certificate.elastic.arn
281   validation_record_fqdns = [for record in aws_route53_record.
282     ↪ elastic_validation : record.fqdn]
283 }
```

Listado C.9: Fichero *elastic.tf* de despliegue cloud

C.2.2. Kibana

```
1 resource "aws_ecs_task_definition" "kibana" {
2   family           = "kibana"
3   network_mode    = "awsvpc"
4   requires_compatibility = [var.launch_type]
5   cpu              = "2048"
6   memory           = "4096"
7   execution_role_arn = aws_iam_role.ecs_task_execution.arn
8   task_role_arn    = aws_iam_role.ecs_task_execution.arn
9
10  container_definitions = jsonencode([
11    {
12      name      = "kibana"
13      image     = "docker.elastic.co/kibana/kibana:${var.stack_version}
14      ↪ }"
15      cpu        = 2048
16      memory     = 4096
17      essential  = true
18      environment = [
19        { name = "SERVER_NAME", value = "tahoe-kibana" },
20        { name = "SERVER_PORT", value = tostring(var.kibana_port) },
21        { name = "ELASTICSEARCH_HOSTS", value = "https://${local.
22      ↪ elastic_url}:${var.elastic_port}" },
23        { name = "SERVER_PUBLICBASEURL", value = "https://${local.
24      ↪ kibana_url}:${var.kibana_port}" },
25        { name = "ELASTICSEARCH_SSL_VERIFICATIONMODE", value = "none"
26      ↪ },
27        { name = "SERVER_SSL_ENABLED", value = "true" },
28        { name = "ELASTICSEARCH_SSL_CERTIFICATEAUTHORITIES", value =
29      ↪ "/usr/share/kibana/config/certs/ca.crt" },
30        { name = "SERVER_SSL_CERTIFICATE", value = "/usr/share/kibana/
31      ↪ config/certs/kibana.crt" },
32        { name = "SERVER_SSL_KEY", value = "/usr/share/kibana/config/
33      ↪ certs/kibana.key" },
34        { name = "XPACK_SCREENSHOTTING_BROWSER_CHROMIUM_DISABLESDANBOX
35      ↪ ", value = "true" },
36        { name = "ELASTICSEARCH_USERNAME", value = "kibana_system" }
37      ]
38      secrets = [
39        {
40          name      = "CA_CRT"
41          valueFrom = "${aws_secretsmanager_secret.es_certs.arn}:ca.
42      ↪ crt::"
43        },
44        {
45
```

```
36         name      = "KIBANA_KEY"
37         valueFrom = "${aws_secretsmanager_secret.es_certs.arn}:
38     ↪ kibana.key::"
39     },
40     {
41         name      = "KIBANA_CRT"
42         valueFrom = "${aws_secretsmanager_secret.es_certs.arn}:
43     ↪ kibana.crt::"
44     },
45     {
46         name      = "ELASTICSEARCH_PASSWORD"
47         valueFrom = "${aws_secretsmanager_secret.es_certs.arn}:
48     ↪ kibana_pass::"
49     },
50     {
51         name      = "XPACK_ENCRYPTEDSAVEDOBJECTS_ENCRYPTIONKEY"
52         valueFrom = "${aws_secretsmanager_secret.es_certs.arn}:
53     ↪ xpack_key::"
54     },
55     {
56         name      = "XPACK_SECURITY_ENCRYPTIONKEY"
57         valueFrom = "${aws_secretsmanager_secret.es_certs.arn}:
58     ↪ xpack_key::"
59     ]
60     entrypoint = [
61         "/bin/sh",
62         "-c",
63         <<-EOT
64         mkdir -p /usr/share/kibana/config/certs
65
66         echo $CA_CRT | base64 -d > /usr/share/kibana/config/certs/ca.
67     ↪ crt
68         echo $KIBANA_KEY | base64 -d > /usr/share/kibana/config/certs/
69     ↪ kibana.key
70         echo $KIBANA_CRT | base64 -d > /usr/share/kibana/config/certs/
71     ↪ kibana.crt
72
73         chmod 400 /usr/share/kibana/config/certs/kibana.key
74         sleep 15
75         exec /usr/local/bin/kibana-docker
```

```
73     EOT
74   ]
75   # mountPoints = [
76   # {
77   #   sourceVolume = "kibana-data"
78   #   containerPath = "/usr/share/kibana/data"
79   #   readOnly = false
80   # }
81   # ]
82   portMappings = [
83   {
84     containerPort = var.kibana_port,
85     hostPort      = var.kibana_port
86   }
87   ]
88   logConfiguration = {
89     logDriver = var.log_driver
90     options = {
91       "awslogs-group"          = var.log_group
92       "awslogs-region"         = var.region
93       "awslogs-stream-prefix" = "kibana"
94     }
95   }
96 }
97 ])
98
99 volume {
100   name = "kibana-data"
101   efs_volume_configuration {
102     file_system_id = aws_efs_file_system.kibana_data.id
103     root_directory = "/"
104   }
105 }
106 }
107
108 resource "aws_ecs_service" "kibana" {
109   name           = "kibana"
110   cluster        = aws_ecs_cluster.cluster.id
111   task_definition = aws_ecs_task_definition.kibana.arn
112   desired_count  = 1
113   launch_type    = var.launch_type
114   enable_execute_command = true
115
116   network_configuration {
117     subnets      = [aws_subnet.private.id]
118     security_groups = [aws_security_group.kibana.id]
```

```
119    }
120
121  load_balancer {
122      target_group_arn = aws_lb_target_group.kibana.arn
123      container_name   = "kibana"
124      container_port   = var.kibana_port
125  }
126
127  depends_on = [aws_ecs_task_definition.kibana]
128 }
129
130 resource "aws_lb" "kibana" {
131     name          = "tahoe-alb-kibana"
132     internal      = false
133     load_balancer_type = "application"
134     security_groups = [aws_security_group.kibana.id]
135     subnets        = [aws_subnet.public_a.id, aws_subnet.public_b.id
136                      ↵ ]
137
138     access_logs {
139         bucket = aws_s3_bucket.access_logs.bucket
140         prefix = "kibana"
141         enabled = true
142     }
143
144 resource "aws_lb_target_group" "kibana" {
145     name          = "tahoe-tg-kibana"
146     port          = 5601
147     protocol      = "HTTPS"
148     vpc_id        = aws_vpc.main.id
149     target_type   = "ip"
150
151     health_check {
152         enabled      = true
153         port        = var.kibana_port
154         protocol    = "HTTPS"
155         path        = "/api/status"
156         interval    = 300
157         timeout     = 60
158         healthy_threshold = 2
159         unhealthy_threshold = 5
160         matcher     = "200-499"
161     }
162 }
163 }
```

```
164 resource "aws_lb_listener" "kibana_https" {
165   load_balancer_arn = aws_lb.kibana.arn
166   port              = 443
167   protocol          = "HTTPS"
168   ssl_policy        = "ELBSecurityPolicy-2016-08"
169   certificate_arn   = aws_acm_certificate.kibana.arn
170
171   default_action {
172     type = "redirect"
173     redirect {
174       port      = tostring(var.kibana_port)
175       protocol = "HTTPS"
176       status_code = "HTTP_301"
177     }
178   }
179 }
180
181 resource "aws_lb_listener" "kibana" {
182   load_balancer_arn = aws_lb.kibana.arn
183   port              = var.kibana_port
184   protocol          = "HTTPS"
185   ssl_policy        = "ELBSecurityPolicy-2016-08"
186   certificate_arn   = aws_acm_certificate.kibana.arn
187
188   default_action {
189     type          = "forward"
190     target_group_arn = aws_lb_target_group.kibana.arn
191   }
192
193   depends_on = [aws_acm_certificate_validation.kibana]
194 }
195
196 resource "aws_route53_record" "kibana" {
197   zone_id = var.route53_zone_id
198   name    = local.kibana_url
199   type    = "A"
200
201   alias {
202     name          = aws_lb.kibana.dns_name
203     zone_id       = aws_lb.kibana.zone_id
204     evaluate_target_health = false
205   }
206 }
207
208 resource "aws_acm_certificate" "kibana" {
209   domain_name    = local.kibana_url
```

```
210 validation_method = "DNS"
211
212 lifecycle {
213   create_before_destroy = true
214 }
215 }
216
217 resource "aws_route53_record" "kibana_acm_validation" {
218   for_each = {
219     for dvo in aws_acm_certificate.kibana.domain_validation_options :
220       ↪ dvo.domain_name => {
221         name      = dvo.resource_record_name
222         record    = dvo.resource_record_value
223         type      = dvo.resource_record_type
224       }
225     }
226   allow_overwrite = true
227   name           = each.value.name
228   records        = [each.value.record]
229   ttl            = 60
230   type           = each.value.type
231   zone_id        = var.route53_zone_id
232 }
233
234 resource "aws_acm_certificate_validation" "kibana" {
235   certificate_arn          = aws_acm_certificate.kibana.arn
236   validation_record_fqdns = [for record in aws_route53_record.
237     ↪ kibana_acm_validation : record.fqdn]
238 }
```

Listado C.10: Fichero *kibana.tf* de despliegue cloud

C.2.3. Logstash

```
1 resource "aws_ecs_task_definition" "logstash" {
2   family           = "logstash"
3   network_mode    = "awsvpc"
4   requires_compatibility = [var.launch_type]
5   cpu              = "1024"
6   memory           = "2048"
7   execution_role_arn = aws_iam_role.ecs_task_execution.arn
8   task_role_arn    = aws_iam_role.ecs_task_execution.arn
9
10  container_definitions = jsonencode([
11    {
12      name      = "logstash"
13      image     = "docker.elastic.co/logstash/logstash:${var.
14      ↪ stack_version}"
15      cpu       = 1024
16      memory    = 2048
17      essential = true
18      environment = [
19        { name = "XPACK_MONITORING_ELASTICSEARCH_HOSTS", value = "
19      ↪ https://${local.elastic_url}:${var.elastic_port}" },
20        { name = "XPACK_MONITORING_ENABLED", value = "true" },
21        { name = "CONFIG_RELOAD_AUTOMATIC", value = "true" },
22        { name = "CONFIG_RELOAD_INTERVAL", value = "60" },
23        { name = "ELASTICSEARCH_USERNAME", value = "elastic" },
24      ]
25      secrets = [
26        {
27          name      = "ELASTICSEARCH_PASSWORD"
28          valueFrom = "${aws_secretsmanager_secret.es_certs.arn}:
29          ↪ elastic_pass::"
30        },
31        {
32          name      = "CA_CERT"
33          valueFrom = "${aws_secretsmanager_secret.es_certs.arn}:ca.
34          ↪ crt::"
35        }
36      ]
37      entrypoint = [
38        "/bin/sh",
39        "-c",
40        <<-EOT
41        cat <<EOF > /usr/share/logstash/config/logstash.conf
42        input {
43          kafka {
```

```
41         bootstrap_servers => "http://${local.kafka_url}:${var.
42   ↪ kafka_port}"
43     topics => ["kafka"]
44   }
45 }
46 output {
47   elasticsearch {
48     hosts => ["https://${local.elastic_url}:${var.elastic_port
49   ↪ }"]
50     user => "elastic"
51     password => "$ELASTIC_PASSWORD"
52     ssl => true
53     cacert => "/usr/share/logstash/config/ca.crt"
54   }
55 }
56 EOF
57
58 echo $CA_CERT | base64 -d > /usr/share/logstash/config/ca.crt
59
60 exec /usr/local/bin/docker-entrypoint
61 EOT
62 ]
63 portMappings = [
64   {
65     containerPort = var.logstash_port,
66     hostPort      = var.logstash_port
67   }
68 ]
69 # mountPoints = [
70 #   {
71 #     sourceVolume  = "logstash-data"
72 #     containerPath = "/usr/share/logstash/data"
73 #     readOnly      = false
74 #   }
75 # ]
76 logConfiguration = {
77   logDriver = var.log_driver
78   options = {
79     "awslogs-group"          = var.log_group
80     "awslogs-region"         = var.region
81     "awslogs-stream-prefix" = "logstash"
82   }
83 }
84 ])
```

```
85   volume {
86     name = "logstash-data"
87     efs_volume_configuration {
88       file_system_id = aws_efs_file_system.logstash_data.id
89       root_directory = "/"
90     }
91   }
92 }
93
94 resource "aws_ecs_service" "logstash" {
95   name          = "logstash"
96   cluster        = aws_ecs_cluster.cluster.id
97   task_definition = aws_ecs_task_definition.logstash.arn
98   desired_count  = 1
99   launch_type    = var.launch_type
100
101  network_configuration {
102    subnets        = [aws_subnet.private.id]
103    security_groups = [aws_security_group.logstash.id]
104  }
105
106  load_balancer {
107    target_group_arn = aws_lb_target_group.logstash.arn
108    container_name   = "logstash"
109    container_port    = var.logstash_port
110  }
111
112  depends_on = [aws_ecs_task_definition.logstash]
113 }
114
115 resource "aws_lb" "logstash" {
116   name          = "tahoe-alb-logstash"
117   internal      = false
118   load_balancer_type = "application"
119   security_groups = [aws_security_group.logstash.id]
120   subnets        = [aws_subnet.public_a.id, aws_subnet.public_b.id
121   ↘ ]
122
123   access_logs {
124     bucket    = aws_s3_bucket.access_logs.bucket
125     prefix    = "logstash"
126     enabled   = true
127   }
128 }
129 resource "aws_lb_target_group" "logstash" {
```

```
130   name          = "tahoe-tg-logstash"
131   port           = var.logstash_port
132   protocol       = "HTTP"
133   vpc_id         = aws_vpc.main.id
134   target_type    = "ip"
135
136   health_check {
137     enabled        = true
138     path           = "/"
139     port           = var.logstash_port
140     protocol       = "HTTP"
141     interval       = 30
142     timeout        = 10
143     healthy_threshold = 3
144     unhealthy_threshold = 3
145     matcher        = "200-399"
146   }
147 }
148
149 resource "aws_lb_listener" "logstash" {
150   load_balancer_arn = aws_lb.logstash.arn
151   port             = var.logstash_port
152   protocol         = "HTTP"
153
154   default_action {
155     type            = "forward"
156     target_group_arn = aws_lb_target_group.logstash.arn
157   }
158 }
159
160 resource "aws_route53_record" "logstash" {
161   zone_id        = var.route53_zone_id
162   name           = local.logstash_url
163   type           = "A"
164
165   alias {
166     name           = aws_lb.logstash.dns_name
167     zone_id        = aws_lb.logstash.zone_id
168     evaluate_target_health = false
169   }
170 }
```

Listado C.11: Fichero *logstash.tf* de despliegue cloud

C.2.4. Kafka

```
1 resource "aws_ecs_task_definition" "kafka" {
2   family           = "kafka"
3   network_mode    = "awsvpc"
4   requires_compatibility = [var.launch_type]
5   cpu              = "2048"
6   memory           = "8192"
7   execution_role_arn = aws_iam_role.ecs_task_execution.arn
8   task_role_arn    = aws_iam_role.ecs_task_execution.arn
9
10  container_definitions = jsonencode([
11    {
12      name      = "zookeeper"
13      image     = "confluentinc/cp-zookeeper:latest"
14      cpu        = 1024
15      memory    = 2048
16      essential  = true
17      portMappings = [
18        {
19          containerPort = var.zookeeper_port
20          hostPort      = var.zookeeper_port
21        }
22      ]
23      environment = [
24        { name = "ZOOKEEPER_CLIENT_PORT", value = tostring(var.
25        ↪ zookeeper_port) },
26        { name = "ZOOKEEPER_TICK_TIME", value = "2000" },
27      ]
28      logConfiguration = {
29        logDriver = var.log_driver
30        options = {
31          "awslogs-group"      = var.log_group
32          "awslogs-region"     = var.region
33          "awslogs-stream-prefix" = "zookeeper"
34        }
35      },
36    {
37      name      = "kafka"
38      image     = "confluentinc/cp-kafka:latest"
39      cpu        = 1024
40      memory    = 6144
41      essential  = true
42      portMappings = [
43        {
```

```
44     containerPort = var.kafka_port
45     hostPort      = var.kafka_port
46   },
47   {
48     containerPort = 29092
49     hostPort      = 29092
50   }
51 ]
52 environment = [
53   { name = "KAFKA_BROKER_ID", value = "1" },
54   { name = "KAFKA_ZOOKEEPER_CONNECT", value = "127.0.0.1:${var.
55   ↪ zookeeper_port}" },
56   { name = "KAFKA_ADVERTISED_LISTENERS", value = "INTERNAL
57   ↪ ://127.0.0.1:29092,EXTERNAL://kafka.okticket.io:${var.kafka_port
58   ↪ }" },
59   { name = "KAFKA_LISTENER_SECURITY_PROTOCOL_MAP", value = "
59   ↪ INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT" },
60   { name = "KAFKA_INTER_BROKER_LISTENER_NAME", value = "INTERNAL
61   ↪ " },
62   { name = "KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR", value = "1"
63   ↪ },
64 ]
65 logConfiguration = {
66   logDriver = var.log_driver
67   options = [
68     "awslogs-group"          = var.log_group
69     "awslogs-region"         = var.region
70     "awslogs-stream-prefix" = "kafka"
71   ]
72 }
73 ]
74
75 resource "aws_ecs_service" "kafka" {
76   name           = "kafka"
77   cluster        = aws_ecs_cluster.cluster.id
78   task_definition = aws_ecs_task_definition.kafka.arn
79   desired_count  = 1
80   launch_type    = var.launch_type
81   enable_execute_command = true
82
83   network_configuration {
84     subnets      = [aws_subnet.private.id]
85     security_groups = [aws_security_group.kafka.id]
86     assign_public_ip = true
87 }
```

```
84    }
85
86    load_balancer {
87        target_group_arn = aws_lb_target_group.kafka.arn
88        container_name   = "kafka"
89        container_port   = var.kafka_port
90    }
91
92    load_balancer {
93        target_group_arn = aws_lb_target_group.zookeeper.arn
94        container_name   = "zookeeper"
95        container_port   = var.zookeeper_port
96    }
97
98    depends_on = [
99        aws_lb_listener.kafka,
100       aws_lb_listener.zookeeper
101    ]
102}
103
104 resource "aws_lb" "kafka" {
105     name          = "tahoe-alb-kafka"
106     internal      = false
107     load_balancer_type = "network"
108     security_groups = [aws_security_group.kafka.id]
109     subnets        = [aws_subnet.public_a.id, aws_subnet.public_b.id
110     ↵ ]
111
112     # access_logs {
113     #     bucket = aws_s3_bucket.access_logs.bucket
114     #     prefix = "kafka"
115     #     enabled = true
116     # }
117
118     tags = {
119         Name = "tahoe-nlb-kafka"
120     }
121
122 resource "aws_lb_target_group" "kafka" {
123     name          = "tahoe-tg-kafka"
124     port          = var.kafka_port
125     protocol      = "TCP"
126     vpc_id        = aws_vpc.main.id
127     target_type   = "ip"
128 }
```

```
129   health_check {
130     protocol          = "TCP"
131     port              = var.kafka_port
132     interval          = 30
133     healthy_threshold = 3
134     unhealthy_threshold = 3
135   }
136 }
137
138 resource "aws_lb_target_group" "zookeeper" {
139   name          = "tahoe-zookeeper-tg"
140   port          = var.zookeeper_port
141   protocol      = "TCP"
142   vpc_id        = aws_vpc.main.id
143   target_type   = "ip"
144
145   health_check {
146     protocol          = "TCP"
147     port              = var.zookeeper_port
148     interval          = 30
149     healthy_threshold = 3
150     unhealthy_threshold = 3
151   }
152 }
153
154 resource "aws_lb_listener" "kafka" {
155   load_balancer_arn = aws_lb.kafka.arn
156   port              = var.kafka_port
157   protocol          = "TCP"
158
159   default_action {
160     type          = "forward"
161     target_group_arn = aws_lb_target_group.kafka.arn
162   }
163 }
164
165 resource "aws_lb_listener" "zookeeper" {
166   load_balancer_arn = aws_lb.kafka.arn
167   port              = var.zookeeper_port
168   protocol          = "TCP"
169
170   default_action {
171     type          = "forward"
172     target_group_arn = aws_lb_target_group.zookeeper.arn
173   }
174 }
```

```
175
176 resource "aws_route53_record" "kafka" {
177   zone_id = var.route53_zone_id
178   name     = local.kafka_url
179   type     = "A"
180
181   alias {
182     name          = aws_lb.kafka.dns_name
183     zone_id       = aws_lb.kafka.zone_id
184     evaluate_target_health = false
185   }
186 }
```

Listado C.12: Fichero *kafka.tf* de despliegue cloud

D. Scripts de ingestá de datos con Kafka

D.1. Ingestá de logs de Laravel

```
1 import json
2 import logging
3 import os
4 import time
5
6 from dotenv import load_dotenv
7 from kafka import KafkaProducer
8
9 # Configurar logging
10 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
11
12 # Cargar variables de entorno
13 load_dotenv()
14
15 # Configuracion del productor de Kafka
16 producer = KafkaProducer(bootstrap_servers=[os.getenv(
17     'KAFKA_BOOTSTRAP_SERVERS')], 
18                             value_serializer=lambda x: json.dumps(x).
19                             encode('utf-8'))
20
21 # Funcion para enviar logs a Kafka
22 def send_to_kafka(log):
23     producer.send(os.getenv('KAFKA_TOPIC'), value=log)
24     logging.info(f"Enviado log a Kafka: {log['timestamp']}")
```



```
25
26 # Funcion para analizar las lineas de log de Laravel
27 def parse_laravel_log(line):
28     parts = line.split('] ')
29     if len(parts) >= 3:
30         timestamp = parts[0][1:]
31         level = parts[1][1:-1]
32         message = '] '.join(parts[2:])
33         return {
34             'timestamp': timestamp,
35             'level': level,
36             'message': message.strip()
37         }
38     return None
```

```
39
40
41 # Funcion principal para procesar logs de Laravel
42 def process_laravel_logs():
43     logging.info(f" Iniciando procesamiento de logs de Laravel desde {
44         ↪ os.getenv('LARAVEL_LOG_FILE')} ")
45     with open(os.getenv('LARAVEL_LOG_FILE'), 'r') as file:
46         file.seek(0, 2)
47         while True:
48             line = file.readline()
49             if not line:
50                 time.sleep(0.1)
51                 continue
52             log_entry = parse_laravel_log(line)
53             if log_entry:
54                 send_to_kafka(log_entry)
55             else:
56                 logging.warning(f"No se pudo analizar la linea: {line.
57         ↪ strip()}")
58
59 if __name__ == "__main__":
60     process_laravel_logs()
```

Listado D.1: Script de ingestá de logs de Laravel

```
1 LARAVEL_LOG_FILE=/var/www/okt-api/storage/logs/laravel.log
2 KAFKA_BOOTSTRAP_SERVERS=kafka.okticket.io:9092
3 KAFKA_TOPIC=laravel_logs
```

Listado D.2: Fichero .env de configuración de ingestá de logs de Laravel

D.2. Ingesta de logs de MongoDB

```
1 import json
2 import logging
3 import os
4 from datetime import datetime
5
6 import pymongo
7 from dotenv import load_dotenv
8
9 from kafka import KafkaProducer
10
11 # Configurar logging
12 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
13
14 # Cargar variables de entorno
15 load_dotenv()
16
17 # Conexion a MongoDB
18 mongo_client = pymongo.MongoClient(f"mongodb://{{os.getenv('MONGO_HOST'
19     '')}}:{{os.getenv('MONGO_PORT')}}/")
20 db = mongo_client[os.getenv('MONGO_DB')]
21 logs_collection = db[os.getenv('MONGO_COLLECTION')]
22
23 # Configuracion del productor de Kafka
24 producer = KafkaProducer(bootstrap_servers=[os.getenv(
25     'KAFKA_BOOTSTRAP_SERVERS')],
26                             value_serializer=lambda x: json.dumps(x).
27                             encode('utf-8'))
28
29 # Funcion para enviar logs a Kafka
30 def send_to_kafka(log):
31     producer.send(os.getenv('KAFKA_TOPIC'), value=log)
32     logging.info(f"Enviado log a Kafka: {log['_id']}")
33
34 # Bucle principal para obtener y enviar logs continuamente
35 def process_mongodb_logs():
36     last_processed_time = datetime.min
37
38     logging.info("Iniciando procesamiento de logs de MongoDB")
39     while True:
40         new_logs = logs_collection.find({"timestamp": {"$gt":
41             last_processed_time}})
```

```
40
41     for log in new_logs:
42         log['_id'] = str(log['_id'])
43         send_to_kafka(log)
44         last_processed_time = log['timestamp']
45
46         logging.info(f"Ultimo timestamp procesado: {
47             last_processed_time}")
48
49 if __name__ == "__main__":
50     process_mongodb_logs()
```

Listado D.3: Script de ingestá de logs de MongoDB

```
1 MONGO_HOST=redactado
2 MONGO_PORT=27017
3 MONGO_DB=redactado
4 MONGO_COLLECTION=logs
5 KAFKA_BOOTSTRAP_SERVERS=kafka.okticket.io:9092
6 KAFKA_TOPIC=mongodb_logs
```

Listado D.4: Fichero .env de configuración de ingestá de logs de MongoDB

D.3. Ingesta de logs de MySQL

```
1 import json
2 import logging
3 import os
4 import time
5
6 import pymysql
7 from dotenv import load_dotenv
8 from kafka import KafkaProducer
9
10 # Configurar logging
11 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
12
13 # Cargar variables de entorno
14 load_dotenv()
15
16 # Configuracion de conexion a MySQL
17 mysql_config = {
18     'host': os.getenv('MYSQL_HOST'),
19     'user': os.getenv('MYSQL_USER'),
20     'password': os.getenv('MYSQL_PASSWORD'),
21     'database': os.getenv('MYSQL_DATABASE')
22 }
23
24 # Configuracion del productor de Kafka
25 producer = KafkaProducer(bootstrap_servers=[os.getenv(
26     'KAFKA_BOOTSTRAP_SERVERS')], value_serializer=lambda x: json.dumps(x).
27     encode('utf-8'))
28
29 # Funcion para enviar logs a Kafka
30 def send_to_kafka(log):
31     producer.send(os.getenv('KAFKA_TOPIC'), value=log)
32     logging.info(f"Enviado log a Kafka: ID {log['id']}")
33
34
35 # Funcion para obtener logs de MySQL
36 def fetch_mysql_logs(cursor, last_id):
37     query = f"SELECT * FROM mysql_general_log WHERE id > {last_id}"
38     query += " ORDER BY id ASC"
39     cursor.execute(query)
40     return cursor.fetchall()
```

```
41
42 # Funcion principal para procesar logs de MySQL
43 def process_mysql_logs():
44     logging.info("Iniciando procesamiento de logs de MySQL")
45     connection = pymysql.connect(**mysql_config)
46     cursor = connection.cursor(pymysql.cursors.DictCursor)
47
48     last_processed_id = 0
49
50     while True:
51         new_logs = fetch_mysql_logs(cursor, last_processed_id)
52
53         for log in new_logs:
54             send_to_kafka(log)
55             last_processed_id = log['id']
56
57         logging.info(f"Ultimo ID procesado: {last_processed_id}")
58         time.sleep(1)
59
60
61 if __name__ == "__main__":
62     process_mysql_logs()
```

Listado D.5: Script de ingesta de logs de MySQL

```
1 MYSQL_HOST=redactado
2 MYSQL_USER=redactado
3 MYSQL_PASSWORD=redactado
4 MYSQL_DATABASE=redactado
5 KAFKA_BOOTSTRAP_SERVERS=kafka.okticket.io:9092
6 KAFKA_TOPIC=mysql_logs
```

Listado D.6: Fichero .env de configuración de ingesta de logs de MySQL

E. Código de ingestra de logs de ELB (AWS)

E.1. Lambda de ingestra

```
1 import json
2 import urllib.request
3 import gzip
4 import base64
5
6 LOGSTASH_URL = "http://logstash.okticket.io:8080"
7
8
9 def lambda_handler(event, context):
10     # Decodificar y descomprimir los datos de log
11     compressed_payload = base64.b64decode(event['awslogs']['data'])
12     uncompressed_payload = gzip.decompress(compressed_payload)
13     log_data = json.loads(uncompressed_payload)
14
15     # Extraer y procesar eventos de log
16     for log_event in log_data['logEvents']:
17         # Preparar el mensaje de log
18         log_message = {
19             'timestamp': log_event['timestamp'],
20             'message': log_event['message'],
21             'log_group': log_data['logGroup'],
22             'log_stream': log_data['logStream']
23         }
24
25         # Enviar log a Logstash
26         request = urllib.request.Request(LOGSTASH_URL)
27         request.add_header('Content-Type', 'application/json')
28         response = urllib.request.urlopen(request, json.dumps(
29             ↪ log_message).encode('utf-8'))
30
31         return {
32             'statusCode': 200,
33             'body': json.dumps('Logs enviados a Logstash exitosamente')
34         }
```

Listado E.1: Lambda de ingestra de logs de ELB

E.2. Filtro de suscripción

```
1 aws logs put-subscription-filter \  
2     --log-group-name "<log_group>" \  
3     --filter-name "LambdaFilter" \  
4     --filter-pattern "" \  
5     --destination-arn "<arn_lambda>"
```

Listado E.2: Comando para añadir el filtro de suscripción

E.3. Configuración de Logstash

```
1 input {  
2     http {  
3         port => 8080  
4     }  
5 }  
6  
7 filter {  
8     json {  
9         source => "message"  
10    }  
11    date {  
12        match => [ "timestamp", "UNIX_MS" ]  
13    }  
14 }  
15  
16 output {  
17     elasticsearch {  
18         hosts => ["https://elastic.okticket.io:9200"]  
19         index => "cloudwatch-logs-%{+YYYY.MM.dd}"  
20     }  
21 }
```

Listado E.3: Configuración de Logstash para ingesta de logs de ELB

E.4. Creación del índice

```
1 {
2     "index_patterns": [
3         "cloudwatch-logs-*"
4     ],
5     "mappings": {
6         "properties": {
7             "log_group": {
8                 "type": "keyword"
9             },
10            "log_stream": {
11                "type": "keyword"
12            },
13            "message": {
14                "type": "text"
15            },
16            "timestamp": {
17                "type": "date"
18            }
19        }
20    },
21    "settings": {
22        "number_of_shards": 1
23    }
24 }
```

Listado E.4: Estructura del índice de Elasticsearch

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.