UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Estudio y configuración de un sistema ELK
Documentación Técnica

Presentado por Hugo de la Cámara Saiz
en Universidad de Burgos — 8 de julio de 2024
Tutores: Jesús Manuel Maudes Raedo y
Antonio Canepa Oneto

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En este apartado de la documentación de los anexos se pretende exponer de manera cronológica como ha ido evolucionando el desarrollo del estudio realizado.

Durante la maduración del mismo se ha empleado la metodología *Scrum* para dividir cada fase del proyecto en *sprints*, dentro de cada cuál se incluye una planificación, un desarrollo y una revisión en forma de reunión del mismo.

A lo largo del desarrollo del estudio, se han ido anotando todas las tareas solicitadas para cada sprint en el repositorio GitHub del proyecto, por lo que en este apartado se hará un desglose de cada tarea involucrada.

Además de todo lo mencionado anteriormente, también se va a estudiar la viabilidad del proyecto, tanto económica, como legal.

A.2. Planificación temporal

Sprint 1 (15/02 - 17/03)

Planificación

Tras una primera toma de contacto entre los miembros del estudio, se acordó que inicialmente se comenzaría con un trabajo de investigación sobre el funcionamiento de cada componente de un sistema basado en ELK, así como la instalación de los mismos y prueba de que se ejecutaban correctamente.

También se encomendó la tarea de crear el repositorio GitHub del estudio, en el que se irían actualizando, a medida que se avanzaba las *issues* del proyecto.

Issues

- **Investigación e instalación de Kibana**
- **Investigación e instalación de ElasticSearch**
- **Investigación e instalación de Logstash**
- **Configuración del repositorio GitHub**
- **Inicio documentación de la memoria**

Desarrollo

En este primer mes se investigó a través de las fuentes principales de información de cada programa su instalación y uso básico de cara a probar con pruebas lo que ofrecían. En primer lugar ElasticSearch no opuso grandes dificultades, puesto que posee una extensa documentación, y su ejecución fue relativamente sencilla, cosa que no ocurrió con Logstash, ya que su comprensión era más compleja y requería de conocimientos más elevados que para las otros dos componentes. Por último, en Kibana fue en el que más profundizo en este sprint, puesto que era el programa más intuitivo que más posibilidades y complementos ofrecía.

También se hizo énfasis en analizar programas alternativos a estos comparando las ventajas e inconvenientes frente a Elastic, Logstash y Kibana.

Revisión

En la segunda reunión se compartieron los resultados encontrados y por parte del tutor el programa que más interés desperto fue Logstash, puesto que las opciones que ofrecía erán las más interesantes de estudiar, cimentando así las bases del segundo sprint que comenzaría inmediatamente después.

Sprint 2 (18/03 - 03/04)

Planificación

En esta segunda fase del proyecto, una vez conocidas las características básicas de los componentes y como se relacionan entre sí, se creyó conveniente profundizar en las posibilidades de filtrado que ofrece Logstash, sabiendo diferenciar qué se puede y qué no se puede hacer.

También se empezó a mencionar la idea de aplicar tanto Machine Learning como MapReduce en algún momento del proceso a los datos, por lo que se encomendó el comienzo de la investigación de ambos temas.

Issues

- **Comenzar con la documentación de la memoria en LaTeX**
- **Investigar sobre la funcionalidad de Machine Learning en Elastic**
- **Investigar funcionamiento filtrado Logstash**
- **Investigar desarrollo con plugins en ElasticSearch**
- **Investigar posible integración con Python para el tratamiento de datos**
- **Investigar desarrollo con Docker y como integrarlo**
- **Investigar funcionamiento MapReduce en Logstash**
- **Investigar funcionamiento MapReduce en ElasticSearch**
- **Investigar funcionamiento MapReduce en Kibana**
- **Probar filtrado en Logstash: sumas, agrupamientos, discretizar columna numérica**

- **Continuar con documentación en Overleaf**
- **Familiarización con librería SKLearn en cuanto a clasificación, regresión, clustering y PCA**

Desarrollo

Se investigó la posibilidad que ofrecía el plugin de Machine Learning que ofrece ElasticSearch, cuyas funciones gratuitas son limitadas, y las más interesantes son de pago. Por parte del tutor, se ofreció una fuente de información para poder aplicar funciones de Machine Learning de manera gratuita a fuentes de datos, *Scikit-Learn*, de manera que durante este tiempo el estudio se enfocó en comprender el funcionamiento de esta librería, y de qué manera la podíamos acoplar en el sistema ELK.

Por otro lado, indagar en las funciones avanzadas de Logstash, fue costoso, puesto que la información presente en internet era escasa y la mayoría antigua, por lo que las pruebas con este programa fueron un claro ensayo y error hasta dar con la tecla. Se profundizo en las operaciones de filtrado, agrupamientos básicos y discretizaciones en el apartado *filter* de el archivo de configuración.

Revisión

En la tercera y cuarta reunión entre los miembros se discutió sobre qué filtros erán más interesantes de aplicar, el orden que debían seguir los datos a lo largo de todo el proceso ETL, y cómo se podían integrar los resultados obtenidos por las funciones de *Scikit-Learn* en Elastic para su posterior exposición en Kibana. Concluyendo así el segundo sprint del proyecto y abriendo temas para la tercera fase.

Sprint 3 (04/04 - 20/04)

Planificación

Y así entramos en la tercera etapa del estudio, en la que se plantearon tareas a realizar como la carga con éxito de los datos del script del Iris en Kibana, continuar explorando los filtros de Logstash, y abordar el tema los *streamings* de datos, que aún no estaba desarrollado en este momento del estudio.

Otro punto a destacar como importante de cara a realizar en esta fase, es el de conseguir realizar de manera eficiente y útil MapReduce en Logstash.

Issues

- **Cargar datos Iris en Kibana**
- **Explorar más posibilidades de agrupamiento en Logstash con máximos, medias, etc.**
- **Probar Data Streams: los que se puede y no (funcionamiento, campos, etc.)**
- **Comparar MapReduce en Kibana con Logstash**
- **Probar Data Streams: los que se puede y no (funcionamiento, campos, etc.)**

Desarrollo

Por un lado, gracias a los foros presentes en las páginas oficiales tanto de Elastic como de *Scikit-Learn*, se consiguió combinar los dos, de manera que el tráfico de información entre ambos fuera fluido y exitoso. Esto se hizo tras realizarle unas modificaciones al script tanto en la forma en la que ingestaba los datos, como en la de como se exportaban los mismos, y hacia donde.

El tema de los *data streams* siguió sin avanzar, puesto que no se encontraba de qué manera se podía implementar esta forma de ingesta de datos en el sistema ELK.

Y trás darle unas cuantas vueltas, se llegó a la conclusión de que la manera de realizar el MapReduce en Logstash era con su función *aggregate*.

Revisión

Así se cubrió lo relacionado con *Scikit-Learn* de manera parcial y cubriendo la tarea encomendada, mientras que los *streamings* de datos en vivo seguía en el aire y sin saber muy bien hacia donde seguir tirando.

Sprint 4 (22/04 - 30/04)

Planificación

Para este cuarto sprint, se vió que el tema de los *data streams* había que terminarlo, por lo que se le dió máxima relevancia a este, de manera que se pudieran mandar datos en tiempo real a Elastic de manera que se les aplicara un filtrado con Logstash y pudieran ser expuestos en Kibana.

Issues

- **Probar funcionamiento Data Streams**
- **Investigar funcionamiento WebSocket en ELK**
- **Conectar WebSocket con Logstash**

Desarrollo

Esto se logró gracias a las herramientas *WebSocket*, la cuál permitio poder conectarnos a una fuente de datos, *Finnhub*, que mandara los datos tanto a Elastic como a Logstash, y esos datos fueran depurados para ser mostrados en un *dashboard* de Kibana. Esto se consiguió gracias a un repositorio público de GitHub que ofrecía diferentes maneras de interactuar con datos en vivo.

De manera paralela a los *WebSockets*, se trabajó con una herramienta del ecosistemas Elastic llamada *Filebeat*, la cuál ofrecía cubir la necesidad de poder mandar datos en *streaming* hacia Elastic o Logstash, pero al ver que la alternativa encontrada era más eficiente y ofrecía más posibilidades, se optó por los *WebSockets*.

Revisión

En la sexta reunión de los miembros se mostró tranquilidad al ver que gran parte de los problemas presentes habían sido solventados y que se comprendía de que manera se podían implementar *data streams* en el sistema ELK.

Con esto, se empezó a plantear la posibilidad de empezar a preparar los escenarios finales junto con la documentación de el funcionamiento de cada uno.

Sprint 5 (31/04 - 06/05)

Planificación

Por lo que en la séptima reunión se hizo retrospectiva de lo que se tenía hasta el momento, y se decidió que el tema de las transformaciones de datos en Logstash aún no estaba en un buen punto, por lo que en este sprint se centró el tiempo en cerrar todo esto.

También se dió enfásis a la idea del *Edge Computing*, y como podía ser relevante aplicarlo en el estudio.

Issues

- **Estudiar posibilidad hacer MapReduce del data stream en Logstash**
- **Investigar funcionamiento MapReduce en Kibana**
- **Investigar sobre el Edge Computing y como funciona**

Desarrollo

Finalmente se terminó de cerrar todo lo relacionado con los filtrados en Logstash al aplicar distintas funciones a una fuente de datos. Funciones como:

- Borrar campos vacíos
- Transformación de tipos
- Operaciones con distintos campos

Por otra parte, hubo un trabajo de investigación sobre *Edge Computing* de cara a poder documentarlo en la memoria, y se comprendió la idea que se le asociaba.

Revisión

Así se concluyó esta quinta fase, dando por sentados todos los escenarios que iban a estar presentes en la documentación técnica, y se comenzó a plantear la idea de la preparación e ilustración de los mismos.

Sprint 6 (07/05 - 16/05)

Planificación

Llegando a esta sexta etapa ya se empezaba a divisar el final en el horizonte y se encomendó la tarea de ilustrar y preparar minuciosamente cada escenario tratado de manera que la documentación de los mismos fuera más sencilla.

Issues

- **Estudiar caso importar directamente el fichero en Elastic**
- **Estudiar caso importar el fichero desde Logstash**
- **Estudiar caso importar un DataSet (Iris)**
- **Estudiar caso importar a través de WebSocket a Elastic**
- **Estudiar caso importar a través de un WebSocket con Logstash**

Desarrollo

Durante este tiempo se trabajó en preparar todos los directorios con la información relevante de cada escenario, así como ilustrar los *dashboards* finales obtenidos en cada uno.

Se empezó a documentar en *Overleaf* la memoria del estudio de manera básica para ir teniendo un punto de partida.

Revisión

Tras la novena reunión de miembros se empezaron a realizar correcciones a la memoria y reestructuraciones a los escenarios de manera que las ideas presentadas fueran más claras y concisas.

Sprint 7 (17/05 - 10/06)

Planificación

En esta séptima etapa se continuó con el plan pensado para la anterior de seguir documentando los escenarios y los diferentes apartados de la memoria técina del estudio.

Issues

- **Investigar sobre el caso de Edge Computing y cómo funciona**
- **Continuar desarrollo de escenarios**

Desarrollo

Por lo que todo este tiempo se dedicó a realizar modificaciones y continuar agregando información a la memoria corrigiendo la estructura y los contenidos de cada apartado de manera que fuerán más acordes a un trabajo científico.

Revisión

Siguiendo la pauta de la anterior reunión las reuniones octava y novena estuvieron enfocadas a seguir mejorando el trabajo realizado en al documentación.

Sprint 8 (11/06 - 10/07)

Planificación

Tras una serie de revisiones tanto de la documentación de los anexos como de la memoria, se han planteado diversas mejoras en ambos por parte del tutor. También se ha mencionado el modificar el script en el que se aplica Machine Learning, de manera que los datos se carguen desde Elastic y una vez procesados sean devueltos. Para terminar se encomendó la tarea de realizar videos explicativos del proyecto, el pulido final del repositorio y la creación de una máquina virtual que contenga todo lo relacionado con el proyecto.

Issues

- **Documentación de la memoria**
- **Documentación de los anexos**

Desarrollo

Se dedicó todo el último mes a pulir detalles en la documentación de manera satisfactoria mediante una serie de revisiones. Se consiguió modificar la fuente de ingesta del script de Machine Learning con éxito, y se trabajó en todo lo mencionado para que la entrega del proyecto se pudiera hacer en fecha.

Revisión

Tras una última revisión de los contenidos, ambas partes quedaron satisfechas con el resultado y se acordó que esa era el producto final a entregar.

A.3. Estudio de viabilidad

Viabilidad económica

A lo largo de este estudio se ha empleado una versión gratuita de Elastic, en la cual, por ejemplo, el plugin que habilita la función Machine Learning, está bloqueado. Elastic ofrece esta y más funciones en sus diferentes niveles de suscripción:

1. **Basic (Gratis)**
   - Ofrece un amplio mercado de plugins y de características limitadas, entre las cuales no se encuentra la de aplicar Machine Learning a los índices de datos. Se ofrecen integraciones para transmitir logs, métricas, rastreos, contenido y más desde diferentes fuentes.
2. **Gold (109 doláres al mes)**
   - Ofrece características adicionales de seguridad, monitoreo de datos y algunas funciones de Machine Learning.
3. **Platinum (125 doláres al mes)**
   - Ofrece todas las características del nivel Gold, además de acceso completo a las capacidades de Machine Learning, alertas avanzadas, y soporte premium.
4. **Enterprise (175 dólares al mes)**
   - Ofrece todas las características del nivel Platinum, con funcionalidades adicionales de administración de los datos y un mejor soporte.

Sklearn no supone coste alguno puesto que es una herramienta de código abierto. El *hardware* empleado ha sido un ordenador de unos 500€.

Suponiendo que en este trabajo se han invertido unas 600 horas de trabajo a lo largo de 5 meses, que son aproximadamente 21 semanas. Asignando unas 28 horas de carga de trabajo semanal, y teniendo en cuenta que el salario del alumno será de 18€/hora, se obtiene un salario bruto de entorno a 2000€/mes. Este salario una vez se le aplica las deducciones por impuestos indicadas en la página de la seguridad social (ver figuras A.1 y A.2).

TIPOS DE COTIZACIÓN (%)				
CONTINGENCIAS	**EMPRESA**	**TRABAJADORES**	**TOTAL**	**Accidentes de Trabajo y Enfermedades Profesionales**
Comunes	23,60	4,70	28,30	Tarifa Primas establecida en la disposición adicional cuarta Ley 42/2006, de 28 de diciembre, PGE 2007, en la redacción dada por la Disposición Final Quinta del RDL 28/2018 de 28 de diciembre (BOE del 29) siendo las primas resultantes a cargo exclusivo de la empresa
Horas Extraordinarias Fuerza Mayor	12,00	2,00	14,00	
Resto Horas Extraordinarias	23,60	4,70	28,30	
Mecanismo Equidad Intergeneracional (MEI)	0,58	0,12	0,7	

(1) Exoneración en la cotización por contingencias comunes (salvo IT), desempleo, fondo garantía salarial y formación profesional, prevista en el art. 152 del RD Legislativo 8/2015:

- Aplicable durante el año 2024 a trabajadores cuenta ajena o a socios trabajadores o de trabajo de cooperativas, que continúen trabajando tras haber alcanzado la edad de 65 años si se acreditan 38 o más años de cotización o 66 años y 6 meses cuando se acrediten menos de 38 años de cotización.
- Tipo de cotización aplicable durante el año 2024 por IT por contingencias comunes: 1,55 %, del que el 1,30% será a cargo de la empresa, y el 0,25 % a cargo del trabajador.

(2) Los contratos de duración determinada por tiempo inferior a 30 días tendrán una cotización adicional a cargo del empresario que se abonará a su finalización, y que durante el año 2024 tendrá un importe de 31,22 €. Esta cotización adicional no se aplicará a los contratos por sustitución, a los contratos para la formación y el aprendizaje ni a los contratos de formación en alternancia.

Figura A.1: Tipos de cotización en el régimen general de la Seguridad Social.

DESEMPLEO	**EMPRESA**	**TRABAJADORES**	**TOTAL**
Tipo General: Contratación indefinida, incluidos los contratos indefinidos a tiempo parcial y fijos discontinuos, contratación de duración determinada en las modalidades de contratos de formación en alternancia, para la formación y aprendizaje, formativo para la obtención de la práctica profesional adecuada al nivel de estudios, de relevo, interinidad y contratos realizados con trabajadores que tengan reconocido un grado de discapacidad no inferior al 33%	5,50	1,55	7,05
Contrato duración determinada a tiempo completo o a tiempo parcial	6,70	1,60	8,30

	EMPRESA	**TRABAJADORES**	**TOTAL**
FOGASA	0,20		0,20

	EMPRESA	**TRABAJADORES**	**TOTAL**
FORMACIÓN PROFESIONAL	0,60	0,10	0,70

Figura A.2: Más tipos de cotización en el régimen general de la Seguridad Social.

Los impuestos supondrán un 23,6 por ciento del sueldo en concepto de contingencias, un 5,5 por ciento en concepto de desempleo, un 0,2 en concepto de FOGASA y un 0,6 en concepto de formación profesional por parte de la empresa. Aplicando todos estos porcentajes el sueldo real del alumno alcanzaría la cifra de 2853€/mes.

Además del alumno también hay que tener en cuenta que se trabaja con dos profesores cumpliendo el rol de tutores. Siendo el salario mensual de 38€, y suponiendo que que trabajan 2 horas por semana, el sueldo bruto de los dos profesores será de 608€/mes, que con la aplicación de impuestos se queda en 867€/mes reales.

Una vez se tienen los datos sobre la mesa, el gasto mensual que supone el proyecto en total es de 3461€, que multiplicado por los 5 meses de duración del trabajo se quedan en 17305€ de coste total.

Viabilidad legal

Un sistema ELK usa principalmente dos tipos de licencias:

- La Licencia de Apache 2.0 para versiones anteriores a la 7.11. Esta licencia permite un uso comercial, de distribución, modificación y patente incluyendo un aviso de *copyright* y de licencia.
- La Licencia Elastic para versiones a partir de la 7.11. Esta también ofrece licencias básicas gratuitas y comerciales para acceso a funciones avanzadas y soporte técnico.
- Sklearn posee una licencia BSD 3-Clause que otorga permisos comerciales, de distribución y modificación incluyendo un aviso de *copyright* y de licencia.

Apéndice B

Especificación de Requisitos

B.1. Introducción

Este apéndice ha sido modificado de una escpecificación de requisitos convencial, en lugar de exponer los diferentes requisitos, estos van a ser sustituidos por los escenarios analizados en el proyecto, adaptando así este anexo a la forma del estudio.

B.2. Objetivos generales

El objetivo de este estudio ha sido el de hacer un estudio de diferentes escenarios modificando la fuente de ingesta de un sistema ELK. En este anexo se van a hacer un desglose de cada uno de esos escenarios.

B.3. Catálogo de requisitos

1. **Escenario 1: ingesta de fichero desde Elastic**
2. **Escenario 2: ingesta de fichero desde Logstash**
3. **Escenario 3: ingesta desde WebSocket a Elastic**
4. **Escenario 4: ingesta de WebSocket desde Logstash**
5. **Escenario 5: ingesta de data stream desde Logstash aplicando MapReduce**

6. **Escenario de aplicación de Machine Learning a un conjunto de datos**

B.4. Especificación de requisitos

Para adaptar este apartado al estudio, en esta sección se va a exponer cada escenario realizado como si fuera un caso de uso, adaptando la plantilla de manera que se adecue a la información que se quiere mostrar

CU-1	**Escenario 1: ingesta de fichero desde Elastic**
Versión	1.0
Autor	Hugo de la Cámara Saiz
Descripción	En este escenario se pretende ingestar un fichero de tipo CSV directamente desde Elastic sin intermediarios.
Precondición	Tener un fichero con datos en formato CSV en el sistema local
Acciones	1. Importar el archivo desde el menú de Elastic 2. Mostrar en un dashboard visualizaciones de los datos.
Postcondiciones	1. Comprobar en el Index Management que el índice se ha generado correctamente 2. Comprobar en el Discoverer que el contenido del archivo ha sido importado correctamente.
Importancia	Alta

Tabla B.1: CU-1 Escenario 1: ingesta de fichero desde Elastic

CU-2	**Escenario 2: ingesta de fichero desde Logstash**
Versión	1.0
Autor	Hugo de la Cámara Saiz
Descripción	En este escenario se pretende ingestar un fichero de tipo log desde Logstash aplicándole una serie de transformaciones y mandarlo a Elastic posteriormente.
Precondición	Tener el fichero .log en el sistema local
Acciones	1. Crear un archivo .conf para Logstash en el que se le indique la ruta al input de los datos. 2. Modificar en la sección filter las diferentes transformaciones que se quieren aplicar. 3. Indicar en la sección output que se quiere mandar a ElasticSearch los datos procesados. 4. Ejecutar Logstash con el archivo de configuración creado. 5. Mostrar en un dashboard visualizaciones de los datos.
Postcondiciones	1. Comprobar en el Index Management que el índice se ha generado correctamente 2. Comprobar en el Discoverer que el contenido del archivo ha sido importado correctamente.
Importancia	Alta

Tabla B.2: CU-2 Escenario 2: ingesta de fichero desde Logstash

CU-3	**Escenario 3: ingesta desde WebSocket a Elastic**
Versión	1.0
Autor	Hugo de la Cámara Saiz
Descripción	En este escenario se pretende ingestar un data stream desde un WebSocket directamente a Elastic, sin intermediarios.
Precondición	Disponer de una fuente de datos websocket
Acciones	1. Configurar la fuente de datos websocket para ingestar los datos directamente en ElasticSearch. 2. Mostrar en un dashboard visualizaciones de los datos.
Postcondiciones	1. Comprobar en el Index Management que el índice se ha generado correctamente 2. Comprobar en el Discoverer que el contenido del archivo ha sido importado correctamente.
Importancia	Alta

Tabla B.3: CU-3 Escenario 3: ingesta desde WebSocket a Elastic

CU-4	**Escenario 4: ingesta de WebSocket desde Logstash**
Versión	1.0
Autor	Hugo de la Cámara Saiz
Descripción	En este escenario se pretende ingestar un data stream de un WebSocket desde Logstash, procesar los datos y que se manden a Elastic.
Precondición	Disponer de una fuente de datos websocket
Acciones	1. Crear un script para estructurar la suscripción y los campos que se quieren mandar. 2. Configurar la fuente de datos WebSocket para ingestar los datos desde Logstash 3. Configurar Logstash para que transforme los datos y los mande a Elastic 4. Mostrar en un dashboard visualizaciones de los datos
Postcondiciones	1. Comprobar en el Index Management que el índice se ha generado correctamente 2. Comprobar en el Discoverer que el contenido del archivo ha sido importado correctamente.
Importancia	Alta

Tabla B.4: CU-4 Escenario 4: ingesta de WebSocket desde Logstash

CU-5	**Escenario 5: ingesta de data stream desde Logstash aplicando MapReduce**
Versión	1.0
Autor	Hugo de la Cámara Saiz
Descripción	En este escenario se pretende ingestar los datos de un data stream procesados aplicando MapReduce por Logstash hasta Elastic.
Precondición	Disponer de una fuente de streaming de datos
Acciones	1. Configurar la fuente de datos para ingestar los datos hacia Logstash 2. Configurar Logstash para que transforme los datos haciendo Map-Reduce y mandándolos a Elastic 3. Mostrar en un dashboard visualizaciones de los datos.
Postcondiciones	1. Comprobar en el Index Management que el índice se ha generado correctamente 2. Comprobar en el Discoverer que el contenido del archivo ha sido importado correctamente.
Importancia	Alta

Tabla B.5: CU-5 Escenario 5: ingesta de data stream desde Logstash aplicando MapReduce

CU-6	**Escenario de aplicación de Machine Learning a un conjunto de datos**
Versión	1.0
Autor	Hugo de la Cámara Saiz
Descripción	En este escenario se pretende aplicar funciones de Machine Learning a un conjunto de datos y mandar los resultados a Elastic.
Precondición	Disponer de un conjunto de datos
Acciones	1. Configurar un script que aplique algoritmos de clasificación, regresión, clustering y reducción de características a un conjunto de datos y que mande los resultados a Elastic. 2. Mostrar en un dashboard visualizaciones de los datos.
Postcondiciones	1. Comprobar en el Index Management que el índice se ha generado correctamente 2. Comprobar en el Discoverer que el contenido del archivo ha sido importado correctamente.
Importancia	Alta

Tabla B.6: CU-6 Escenario de aplicación de Machine Learning a un conjunto de datos

Apéndice C

Especificación de diseño

C.1. Introducción

En este apartado lo que se pretende es indagar en la estructura que se ha seguido para el desarrollo del estudio desde la perspectiva del diseño, tanto de los datos como procedimental y arquitectónico.

C.2. Diseño de datos

Una vez los datos son procesados por Logstash y previo a su exposición en un *dashboard* de Kibana, estos son almacenados en ElasticSearch en forma de indices que se pueden o generar en el momento de carga, o generar previamente por línea de comando en la terminal *Dev Tools* de Elastic. Esta sección se va a centrar en analizar como se gestionan y almacenan esos datos, y de que forma son estructurados en Elastic.

Un índice en ElasticSearch consiste en un almacen de documentos que contienen campos en forma de pares clave-valor que almacenan los datos [1]. En este estudio se han analizado cinco escenarios, conteniendo toda la información de cada escenario en índices aislados de manera que se siga una estructrua ordenada.

Para el primero de los escenarios, en el cuál se realiza la ingesta de un archivo de tipo CSV directamente desde Elastic, el índice que se generó recibe el nombre de *titanic*, puesto que el archivo original tiene este nombre. Contiene 891 documentos equivalente a las 891 líneas de registro que posee el archivo original, y el mapeo de este índice está estructurado de la siguiente forma:

{ "mappings": { "_meta": { çreated_by": "file-data-visualizer"}, "properties": { "**Age**": { "type": "double"}, "**Cabin**": { "type": "keyword"}, "**Embarked**": { "type": "keyword"}, "**Fare**": { "type": "double"}, "**Name**": { "type": "text"}, "**Parch**": { "type": "long"}, "**PassengerId**": { "type": "long"}, "**Pclass**": { "type": "long"}, "**Sex**": { "type": "keyword"}, "**SibSp**": { "type": "long"}, "**Survived**": { "type": "long"}, "**Ticket**": { "type": "keyword"} } } }

Incluyendo en el apartado *properties* los datos del nombre y el tipo de cada campo del archivo, de manera que una vez los datos seán cargados, Elastic comprenda de que tipo es cada valor cargado.

En el siguiente escenario, en el cuál se realiza la ingesta de archivo pasando por Logstash donde se le aplican una serie de modificaciones antes de llegar a Elastic, el índice creado recibe el nombre de *casas*, puesto que el archivo contiene información de distintas vivendas. Posee 80 documentos equivalente a las 80 casas en venta que tiene la inmobiliaria y el mapeo tiene la siguiente estructura:

{ "mappings": { "properties": { "@**timestamp**": { "type": "date"}, "**barrio**": { "type": "keyword"}, "**categoria**": { "type": "keyword"}, "**ciudad**": { "type": "keyword"}, "**metros_cuadrados**": { "type": "float"}, "**num_casa**": { "type": ïnteger"}, "**num_habitaciones**": { "type": ïnteger"}, "**precio**": { "type": "float"} } } }

Incluyendo en el apartado *properties* los datos del nombre y el tipo de cada campo del archivo, de manera que una vez los datos seán cargados, Elastic comprenda de que tipo es cada valor cargado.

En los siguientes dos escenarios la dificultad se volvió mayor puesto que se trabajó con streams de datos, por lo que los índices de estos dos escenarios son más complejos que los anteriores y ocupan un mayor tamaño. En el caso del tercer escenario, el índice recibe el nombre de *websockets-data*, y tiene un tamaño cambiante en función del tiempo que esté expuesto a la carga de datos por parte del script del WebSocket. Tiene el siguiente mapeo puesto que los datos que llegan son tal cuál los que se mandan por el WebSocket:

{ "mappings": { "properties": { "@**timestamp**": { "type": "date"}, "data": { "properties": { "**c**": { "type": "text", "fields": { "keyword": { "type": "keyword", ïgnore_above": 256 } } }, "**p**": { "type": "float"}, "**s**": { "type": "text", "fields": { "keyword": { "type": "keyword", ïgnore_above": 256 } } }, "**t**": { "type": "long"}, "**v**": { "type": "float"} } }, "type": { "type": "text", "fields": { "keyword": { "type": "keyword", ïgnore_above": 256 } } } } } }

En el cuál llegan cinco campos con nombres de letras los cuáles serán ignorados si su longitud máxima excede los 256 caracteres.

En el cuarto escenario la descripción del índice es la misma, salvando que en este caso recibe el nombre de *test-data-stream*. El mapeo del contenido tendrá la siguiente forma una vez los datos:

{ "mappings": { "dynamic": "true", "dynamic_date_formats": ["strict_date_optional_time", ɛyyy/MM/dd HH:mm:ss Z||yyyy/MM/dd Z"], "dynamic_templates": [], "date_detection": true, "numeric_detection": true, "properties": { "@**timestamp**": { "type": "date", "format": "strict_date_optional_time"}, "@**version**": { "type": "long"}, "data": { "properties": { "**LastPrice**": { "type": "float"}, "**Symbol**": { "type": "text", "fields": { "keyword": { "type": "keyword", ïgnore_above": 256 }}}, "**Timestamp**": { "type": "long"}, "**TotalPrice**": { "type": "float"}, "**Volume**": { "type": "float"}}}

Como se puede observar las variables que antes eran letras ahora tienen un nombre que describe la información que contiene, así como la inclusión de nuevos campos como son la versión y el precio total.

En el quinto y último escenario, en el cuál se ingesta un streaming de datos en vivo aplicando MapReduce en Logstash antes de llegar a Elastic, el índice que se ha creado recibe el nombre de *transactions aggregate*, y contiene tan solo 3 filas por los 3 métodos de pago existentes, que es la variable que se ha tomado como referencia para hacer el MapReduce. El mapeo de los campos cargados tiene la siguiente estructura:

{ "mappings": { "properties": { "@**timestamp**": { "type": "date"}, "@**version**": { "type": "text", "fields": { "keyword": { "type": "keyword", ïgnore_above": 256 } } }, "**amount**": { "type": "float"}, "**city**": { "type": "text", "fields": { "keyword": { "type": "keyword", ïgnore_above": 256 } } }, "**country**": { "type": "text", "fields": { "keyword": { "type": "keyword", ïgnore_above": 256 } } }, "**customer_id**": { "type": "text", "fields": { "keyword": { "type": "keyword", ïgnore_above": 256 } } }, .ᵉᵛent": { "properties": { .ᵒriginal": { "type": "text", "fields": { "keyword": { "type": "keyword", ïgnore_above": 256 } } } } }, "host": { "properties": { "name": { "type": "text", "fields": { "keyword": { "type": "keyword", ïgnore_above": 256 } } } } }, "log": { "properties":

{ "file": { "properties": { "path": { "type": "text", "fields": { "keyword": { "type": "keyword", ïgnore_above": 256 } } } } } }, "**payment_method**": { "type": "text", "fields": { "keyword": { "type": "keyword", ïgnore_above": 256 } } }, "**product_category**": { "type": "text", "fields": { "keyword": { "type": "keyword", ïgnore_above": 256 } } }, "**timestamp**": { "type": "date"}, "**total_amount**": { "type": "float"}, "**transaction_id**": { "type": "text", "fields": { "keyword": { "type": "keyword", ïgnore_above": 256 } } } } }

Almacenando todas los campos presentes en el streaming de datos de manera que se pueda completar los 3 documentos con información interesante recopilada de los datos.

Figura C.1: Arquitectura del sistema ELK.

C.3. Diseño arquitectónico

El estudio se ha estructurado en un sistema ELK, compuesto por ElasticSearch, Logstash y Kibana, así como de una fuente desde donde se ingestan los datos. Estos cuatro elementos conforman la arquitecturá básica del proyecto (ver ilustración C.1).

La fuente de ingesta de datos ha ido variando en función de la situación que se quería mostrar en cada escenario. Siendo un archivo de tipo CSV en el primero, un archivo de tipo .log en el segundo y streamings de datos en los demás. Lo que permanece inamovible son los otros tres componentes del proyecto, ElasticSearch, siendo empleado en todos los escenarios como base de almacenamiento y búsqueda de los datos, Logstash, como intermediario entre Elastic y la fuente de datos para procesar y transformar la información de estos antes de ser cargada, y por último pero no menos importante, Kibana, cumpliendo el papel de expositor para la información procesada y cargada previamente por los otros componentes.

Apéndice D

Documentación técnica de programación

D.1. Introducción

En este anexo se va a explicar todo lo necesario que tiene que saber un programador para poder replicar los experimentos de este estudio. Incluyendo versiones de los programas usados, origen de las fuentes de datos así como la estructura del código desarrollado.

D.2. Estructura de directorios

El proyecto se estructura en una serie de directorios que contienen tanto la documentación como el material empleado en cada escenario desarrollado.

- **docs**: contiene toda la información relacionada con la documentación del proyecto, como pueden ser PDFs, ficheros LaTeX o fuentes e imágenes empleadas.
- **Escenario 1: ingesta desde fichero en Elastic**: contiene tanto los archivos involucrados en este escenario como el *dashboard* final del mismo.
- **Escenario 2: ingesta desde fichero con Logstash**: contiene tanto los archivos involucrados en este escenario como el *dashboard* final del mismo.

- **Escenario 3: ingesta a través de WebSocket a Elastic**: contiene el script de ingesta de datos junto con el *dashboard* final obtenido.
- **Escenario 4: ingesta a través de WebSocket con Logstash**: contiene el script de ingesta de datos y el archivo de configuración de Logstash junto con el *dashboard* final obtenido.
- **Escenario 5: ingesta de data stream con MapReduce en el servidor mediante Logstash**: contiene los scripts de ingesta de datos y el archivo de configuración de Logstash junto con el *dashboard* final obtenido.
- **Machine Learning**: contiene los scripts de ingesta de datos junto con el *dashboard* final obtenido.

D.3. Manual del programador

Para el desarrollo del proyecto se han empleado una serie de programas a parte de los que conforman el ecosistema ELK. En este apartado se van a exponer las características y configuraciones empleadas en cada uno de ellos para el desarollo del proyecto.

ElasticSearch

Este programa se descargó desde la página oficial del software, instalando la versión 8.11.3, la cuál consiste en un archivo comprimido el cuál una vez descomprimido ya permite el uso mediante la ejecución del archivo *elasticsearch.bat* alojado en el directorio *bin*, el cuál si es ejecutado arrancará el servidor en el equipo, y tras unos segundos permitirá consultar su estado accediendo desde el navegador al puerto 9200, en el cuál estará corriendo el servidor de Elastic (ver ilustración D.1).

Logstash

Esta herramienta de administración de datos fue descargada desde la página oficial de Logstash, en la versión 8.13.2, que al igual que el anterior, consistirá en un comprimido que una vez extraído se tendrá que crear un archivo de tipo .conf que será la configuración que empleará Logstash en el momento de ejecución del archivo logstash.bat ubicado en la carpeta *bin*. Una vez tenemos el archivo de configuración listo ejecutaremos desde la terminal el archivo .bat mencionado anteriormente indicándole la configuración que seguirá.

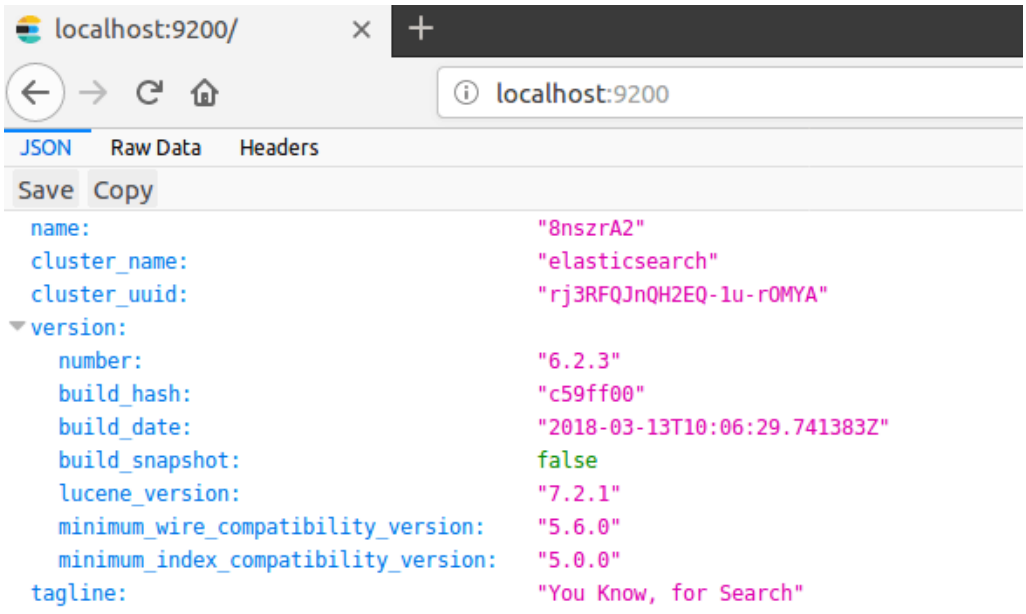

Figura D.1: ElasticSearch corriendo en el puerto local 9200

Kibana

El útimo componente del *stack ELK*, al igual que los otros dos, ha sido descargado desde la página oficial del programa, en la versión 8.11.3, y que una vez extraído el archivo comprimido, se ejecutará el archivo *kibana.bat* alojado en el directorio *bin*, el cuál al ser ejecutado arrancará el programa, y tras unos segundos permitirá acceder al mismo desde el navegador en el puerto 5601, en el cuál estará corriendo el servicio de Kibana.

Python

Fue instalado en su momento a través la página oficial con la versión 3.12.2, la cuál consiste en un ejecutable que una vez abierto instalará todos los archivo necesarios para poder escribir y ejecutar código en este lenguaje de programación.

Jupyter Notebook

Este programa fue descargado desde la página del proyecto Jupyter en la versión 7.1.2. Una vez instalado el programa, desde la terminal se ejecuta el comando *jupyter notebook* y en el puerto local 8888 se mostrará la estructura de directorios del servicio.

Visual Studio Code

El software que se usó para programar parte del código empleado en el proyecto fue descargado desde la página oficial del programa en la versión 1.90.2, y una vez ejecutado el instalador el programa estará listo para ser usado.

D.4. Ejecución de los escenarios experimentados

En esta sección se van a abordar los procesos seguidos para ejecutar el procesamiento de los datos en cada uno de los cinco escenarios con las herramientas ELK.

El caso del primer escenario es el más sencillo de todos, puesto que la ingesta del archivo de tipo CSV se hace directamente desde el menú interactivo de Elastic, en la opción *upload file* (ver ilustración D.2), donde mediante un mecanismo de *drag and drop* Elastic permite la importación de archivos CSV, TSV, o JSON entre otros.

En el segundo escenario, Logstash está de por medio entre la fuente de datos y ElasticSearch, por lo que es ahí donde ocurre todo el procesamiento de los datos. Primero, indicando en al apartado *input* del archivo de configuración para este escenario desde donde se van a ingestar los datos en Logstash (ver ilustración D.3), en este caso se introduce la ruta *path* hacia el archivo casas.log, del cuál se extrae la información, se indica la posición desde donde se quiere empezar a leer, si partimos de una base de datos guardada y la codificación del archivo.

Una vez el archivo es cargado en Logstash, ya se le pueden realizar todas las transformaciones y modificaciones oportunas en el apartado *filter*. Tras esto, los datos procesados deben ir al destino que se indique en el apartado *output* (ver ilustración D.4), en este caso primero se mandaran a Elastic indicando la ruta del puerto local, el nombre del índice donde se alojaran y que hay que crear, así como el usuario y contraseña de Elastic en caso de

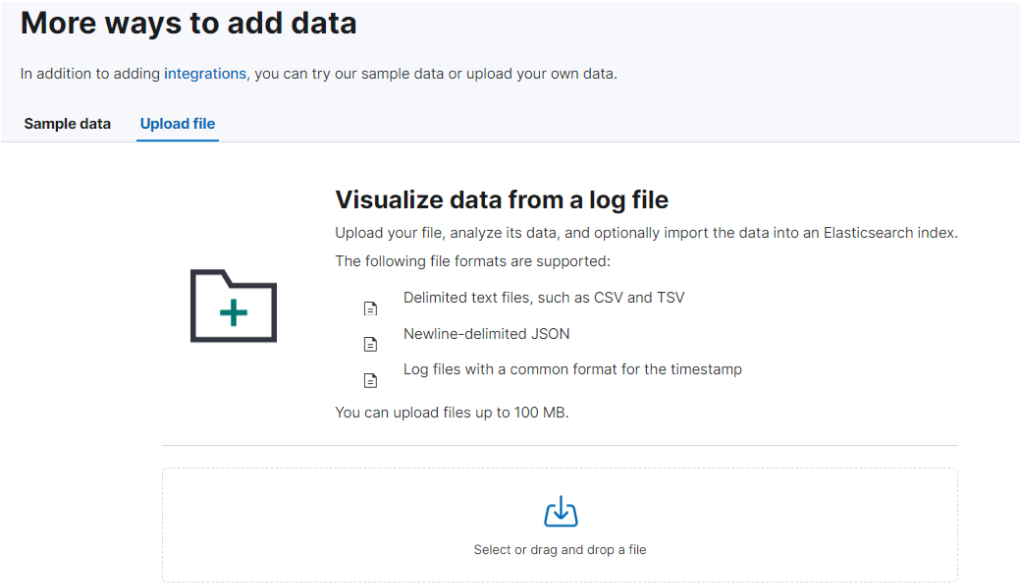

Figura D.2: Menú de importación de archivos de Elastic.

```
input {
  file {
    path => "C:/Users/hugod/Desktop/TFG/ficheros/casas.log"
    start_position => "beginning"
    sincedb_path => "NUL"
    codec => json
  }
}
```

Figura D.3: Apartado de ingesta del segundo escenario.

```
output {
  elasticsearch {
    hosts => ["http://localhost:9200"]
    index => "casas"
    user => elastic
    password => elastic
    action => "create"
  }
  stdout { codec => rubydebug }
}
```

Figura D.4: Apartado de exportación de los datos tratados.

que fueran necesarios. Tambien se indica que los datos deben ser mostrados por terminal para comprobar que el proceso a sido satisfactorio.

Para el tercer escenario, en el que se trabaja con un stream de datos directamente sobre Elastic, todo el proceso de ingesta se realiza en el script escrito en Python del mismo. Primeramente se indica en la variable *client* cuál es el puerto local destino de los datos, y en la función *message* se almacenará la fecha y los datos en el índice *websockets data* (ver ilustración D.5). En la parte final se encuentra la suscripción a diferentes divisas criptográficas, así como el token para utilizar la API de Finnhub, la cuál estará mandando datos hasta que se detenga el script (ver ilustración D.6).

El cuarto escenario, al ser una modificación del tercero, tendrá una estructura del procedimiento de ingesta de datos similar. La primera diferencia será en la parte inicial del script (ver ilustración D.7), donde se añadirá una nueva función que renombre los campos recibidos de manera que sea más entendible la información que proporcionan. También cambia la ruta del destino de los datos, puesto que a través del protocolo HTTP los datos serán enviados al puerto 8080 donde Logstash estará escuchando a la espera de información.

```
client = Elasticsearch("http://localhost:9200")

client.info()

def on_message(ws, message):
    message_json = json.loads(message)
    message_json["@timestamp"] = datetime.datetime.utcnow().isoformat()
    resp = client.index(index="websockets-data", body=message_json)
    print(message_json)
```

Figura D.5: Sección del script indicando datos de la ingesta del tercer escenario.

```
def on_open(ws):
    ws.send('{"type":"subscribe","symbol":"AAPL"}')
    ws.send('{"type":"subscribe","symbol":"AMZN"}')
    ws.send('{"type":"subscribe","symbol":"BINANCE:BTCUSDT"}')
    ws.send('{"type":"subscribe","symbol":"IC MARKETS:1"}')

if __name__ == "__main__":
    websocket.enableTrace(True)
    ws = websocket.WebSocketApp("wss://ws.finnhub.io?token=conrq51r01qm6hd153ogconrq51r01qm6hd153p0",
                                on_message=on_message,
                                on_error=on_error,
                                on_close=on_close)
    ws.on_open = on_open
    ws.run_forever()
```

Figura D.6: Sección del script indicando más datos de la ingesta del tercer escenario.

```python
def rename_fields(message):
    data = json.loads(message)
    if "data" in data:
        renamed_data = []
        for item in data["data"]:
            renamed_item = {
                "Symbol": item.get("s"),
                "LastPrice": item.get("p"),
                "Timestamp": item.get("t"),
                "Volume": item.get("v"),
                "TradeConditions": item.get("c")
            }
            renamed_data.append(renamed_item)
        data["data"] = renamed_data
    return json.dumps(data)

def on_message(ws, message):
    url = 'http://localhost:8080'
    headers = {'Content-Type': 'application/json'}
    modified_message = rename_fields(message)
    response = requests.post(url, headers=headers, data=modified_message)
    print(response.text)
```

Figura D.7: Primera parte de la ingesta del cuarto escenario.

El resto del script tiene una estructura idéntica al del tercer escenario. Una vez los datos son mandados al puerto 8080, Logstash tendrá especificado en el apartado *input* que tiene que estar escuchando de ese puerto a través del protocolo HTTP (ver ilustración D.8). Una vez cargados en tiempo real, la información es procesada con las transformaciones del apartado *filter* y mandadas a Elastic al índice *test data stream* como se especifica en el apartado *output* del archivo de configuración (ver ilustración D.9).

Para el último escenario en el que aplicamos MapReduce a una transmisión de datos en vivo, el proceso de ingesta tiene varias partes. La primera será el script que genera estos datos en vivo (ver ilustración D.10), en el cuál se especifica la ruta del archivo donde se almacenarán los registros, así como los campos sobre los que se van a ir generando valores aleatorios en función de los indicados.

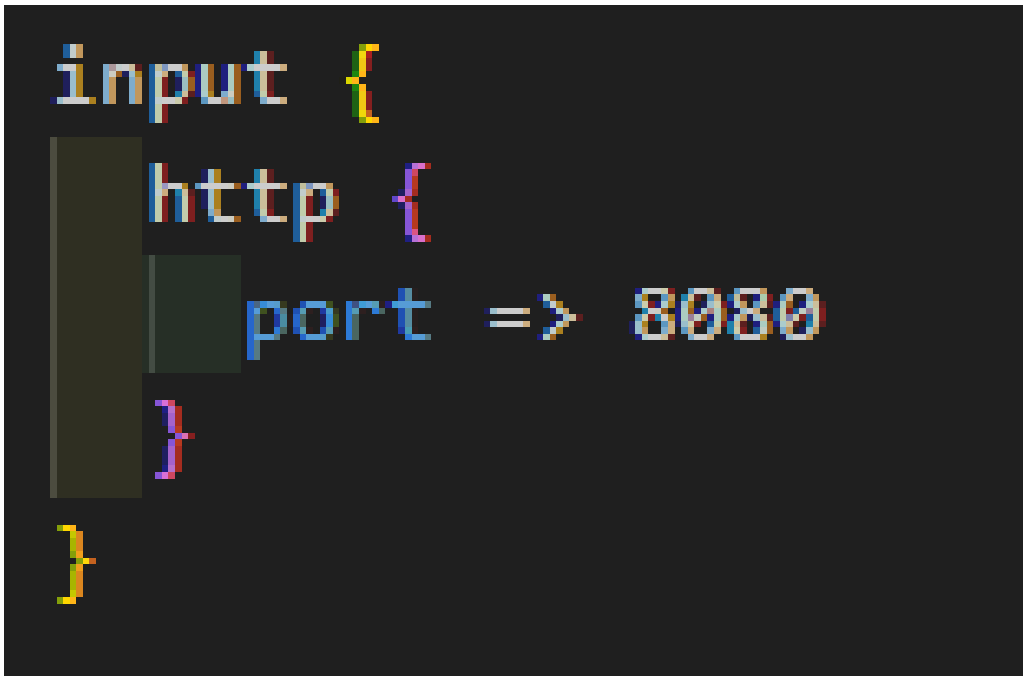

Figura D.8: Segunda parte de la ingesta de datos del cuarto escenario.

Una vez se ejecuta el script los registros se van escribiendo del archivo del cuál Logstash está informado para poder realizar el MapReduce (ver ilustración D.11). Una vez se detiene la carga de datos a Logstash por parte del script, este mandará la información procesada a Elastic dirigiéndose hacia el índice indicado (ver ilustración D.12).

```
output {
  stdout {}
  elasticsearch {
    hosts => ['localhost:9200']
    index => "test-data-stream"
    user => elastic
    password => elastic
    action => "create"
  }
}
```

Figura D.9: Úiltima parte del proceso de ingesta del cuarto escenario.

```python
# Path to save the log file
path = r'C:\Users\hugod\Desktop\TFG\ficheros\test.log'

# Data for generating transactions
customers = [123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 139]
payment_methods = ["Credit Card", "Debit Card", "PayPal"]
product_categories = ["Electronics", "Books", "Clothing"]

# Function to generate a random transaction
def generate_transaction(transaction_id, timestamp):
    customer_id = random.choice(customers)
    amount = round(random.uniform(20.0, 300.0), 2)
    payment_method = random.choice(payment_methods)
    product_category = random.choice(product_categories)

    transaction = {
        "transaction_id": str(transaction_id),
        "timestamp": timestamp.isoformat() + "Z",
        "customer_id": str(customer_id),
        "amount": amount,
        "payment_method": payment_method,
        "product_category": product_category,
    }
    return transaction
```

Figura D.10: Script de generación de datos del quinto escenario.

```
input {
  file {
    path => "C:/Users/hugod/Desktop/TFG/ficheros/test.log"
    start_position => "beginning"
    codec => "json"
  }
}
```

Figura D.11: Ingesta de datos del script a Logstash en el quinto escenario.

```
output {
  elasticsearch {
    hosts => ["http://localhost:9200"]
    index => "transactions_aggregate2"
    document_id => "%{payment_method}"
    action => "create"
  }

  stdout {
    codec => rubydebug
  }
}
```

Figura D.12: Destino final para los datos del quinto escenario.

D.5. Compilación, instalación y ejecución del proyecto

Una vez se tiene todo el software instalado y listo, es necesario descargar los datos utilizados en el proyecto. Para ello, se deja a disposición el repositorio del proyecto en el cuál se encuentra toda la información empleada con la estructura de directorios mencionada anteriormente.

En este apartado se va a describir escenario por escenario, el funcionamiento del mismo. Antes de comenzar, lo primero es tener corriendo tanto ElasticSearch como Kibana de la manera que se ha mencionado antes. Una vez la terminal muestra que ambos programas están en ejecución, se puede empezar a trabajar.

	A	B	C	D	E	F	G
1	PassengerId,Survived,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare,Cabin,Embarked						
2	1,0,3,"Braund, Mr. Owen Harris",male,22,1,0,A/5 21171,7.25,,S						
3	2,1,1,"Cumings, Mrs. John Bradley (Florence Briggs Thayer)",female,38,1,0,PC 17599,71.2833,C85,C						
4	3,1,3,"Heikkinen, Miss. Laina",female,26,0,0,STON/O2. 3101282,7.925,,S						
5	4,1,1,"Futrelle, Mrs. Jacques Heath (Lily May Peel)",female,35,1,0,113803,53.1,C123,S						
6	5,0,3,"Allen, Mr. William Henry",male,35,0,0,373450,8.05,,S						
7	6,0,3,"Moran, Mr. James",male,,0,0,330877,8.4583,,Q						
8	7,0,1,"McCarthy, Mr. Timothy J",male,54,0,0,17463,51.8625,E46,S						
9	8,0,3,"Palsson, Master. Gosta Leonard",male,2,3,1,349909,21.075,,S						
10	9,1,3,"Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)",female,27,0,2,347742,11.1333,,S						

Figura D.13: *Dataset* del titanic.

Escenario 1: ingesta desde fichero en Elastic

En este primer escenario, se empleó como origen de los datos un *dataset* conocido en el mundo de la ciencia de datos, como es el de los pasajeros del desastre del Titanic. Este fue obtenido desde el repositorio de datos *Data Science Dojo*, el cuál pone a disposición pública gran variedad de *datasets* para experimentar.

El *dataset* consiste en un archivo de valores separados por comas con información de los 891 pasajeros del Titanic (ver ilustración D.13), incluyendo campos como la clase del pasajero, su nombre o su edad entre otros.

Una vez se tiene en el sistema el archivo *titanic.csv*, el siguiente paso teniendo en movimiento tanto Elastic como Kibana, es importar el archivo a ElasticSearch a través de la opción *Upload a file* (ver ilustración D.14).

Cuando el fichero se haya cargado en ElasticSearch, se podrá comprobar desde Kibana en el apartado *Discover* que todos los datos han sido cargados correctamente. Esta parte del proceso queda explicada con detenimiento en el anexo *Manual de Kibana*.

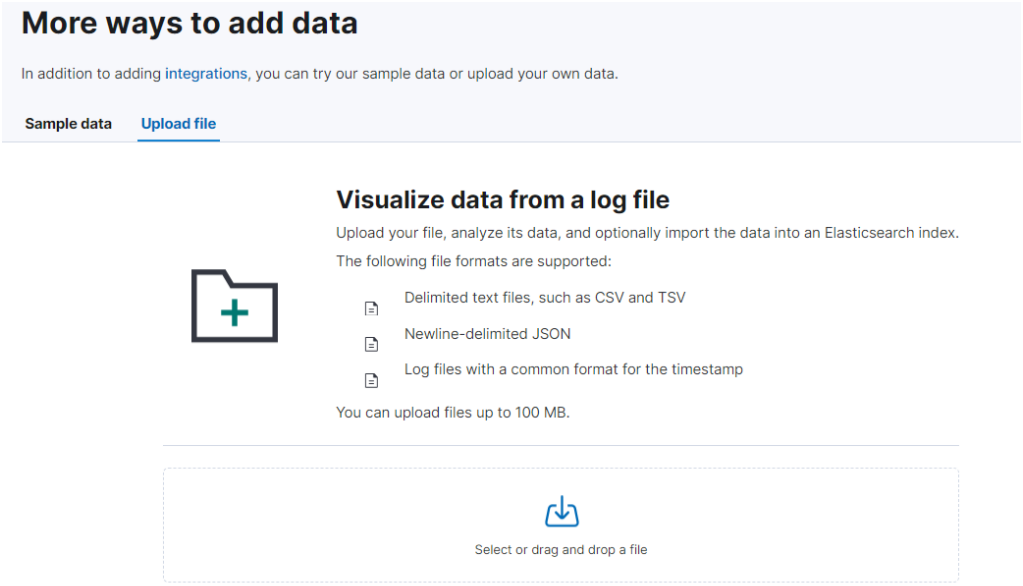

Figura D.14: Pantalla para importar datos del primer escenario.


```
{"numero": 1, "ciudad": "Madrid", "barrio": "Salamanca", "precio": 320000, "habitaciones": 3, "baños": 2,
"metros_cuadrados": 120}
{"numero": 2, "ciudad": "Barcelona", "barrio": "Gràcia", "precio": 290000, "habitaciones": 3, "baños": 2,
"metros_cuadrados": 130}
{"numero": 3, "ciudad": "Sevilla", "barrio": "Triana", "precio": 260000, "habitaciones": 2, "baños": 1,
"metros_cuadrados": 90}
{"numero": 4, "ciudad": "Valencia", "barrio": "Ruzafa", "precio": 190000, "habitaciones": 2, "baños": 1,
"metros_cuadrados": 75}
{"numero": 5, "ciudad": "Madrid", "barrio": "Chamberí", "precio": 280000, "habitaciones": 2, "baños": 1,
"metros_cuadrados": 100}
{"numero": 6, "ciudad": "Barcelona", "barrio": "Eixample", "precio": 350000, "habitaciones": 2, "baños": 1,
"metros_cuadrados": 110}
{"numero": 7, "ciudad": "Sevilla", "barrio": "Nervión", "precio": 240000, "habitaciones": 2, "baños": 1,
"metros_cuadrados": 100}
{"numero": 8, "ciudad": "Valencia", "barrio": "Ciutat Vella", "precio": 220000, "habitaciones": 2, "baños": 1,
"metros_cuadrados": 85}
```


Figura D.15: Estructura del fichero fuente de datos del segundo escenario.

Escenario 2: ingesta desde fichero con Logstash

Para este segundo escenario se requiere del uso de Logstash para transformar y cargar en Elastic un fichero presente en el sistema. El origen de la fuente de datos de este fichero va a ser el archivo *casas.log* (ver ilustración D.15), un conjunto de datos sobre los diferentes inmuebles que se ofertan en una inmobiliaria, con información como la ciudad, el barrio o el precio entre otros.

```
input {
  file {
    path => "C:/Users/hugod/Desktop/TFG/ficheros/casas.log"
    start_position => "beginning"
    sincedb_path => "NUL"
    codec => json
  }
}
```

Figura D.16: Apartado *input* del archivo de configuración del segundo escenario.

Una vez están en ejecución los otros dos componentes del *stack ELK*, hay que poner en funcionamiento Logstash, pero primero se necesita un archivo que indique la configuración a seguir en este escenario, y ese va a ser el archivo *escenario2.conf*, el cuál estará estructurado de manera que en el apartado *input* (ver ilustración D.16) se indique el *path* hacia el archivo *casas.log*.

En la sección de filtrado (ver ilustración D.17) del archivo de configuración se realizará una operación en la que se va a realizar es la modificación el nombre de los campos *numero* por *num casa* y de *habitaciones* por *num habitaciones* con la función *rename*, la cuál permite cumplir esta orden. La siguiente transformación que se ha aplicado es la de cambiar los tipos de los campos *precio* y *metros cuadrados* con la función *convert*, la cuál es de gran ayuda cuando los campos que se van a mandar a Elastic no poseen un tipado que permita el correcto procesamiento de la información. La tercera operación aplicada en este filtro va a consistir en la eliminación del campo *baños* con la función *remove field*, la cuál permite eliminar del mensaje final mandado a Elastic contendido que no se considere oportuno o que carece de relevancia. La última transformación es una discretización en función del campo *precio* de cada vivienda. En ella se indica la creación de un nuevo campo para cada registro llamado *categoria*, en el cuál si la vivienda tiene un precio menor de 100000 será considerada de categoría barata, si el precio oscila entre 100000 y 300000 será considerada de categoría intermedia y si excede la última cantidad será catalogada como cara.

Y en la parte de salida de los datos (ver ilustración D.18) se especifica el destino, que es Elastic.

```
filter {
  mutate {
    rename => { "numero" => "num_casa" }
    rename => { "habitaciones" => "num_habitaciones" }
    convert => { "precio" => "float" }
    convert => { "metros_cuadrados" => "float" }
    remove_field => ["baños"]
  }

  if [precio] < 200000 {
    mutate {
      add_field => { "categoria" => "Barata" }
    }
  } else if [precio] >= 200000 and [precio] < 300000 {
    mutate {
      add_field => { "categoria" => "Intermedia" }
    }
  } else {
    mutate {
      add_field => { "categoria" => "Cara" }
    }
  }
}
```

Figura D.17: Apartado *filter* del archivo de configuración del segundo escenario.

```
output {
  elasticsearch {
    hosts => ["http://localhost:9200"]
    index => "casas"
    user => elastic
    password => elastic
    action => "create"
  }
  stdout { codec => rubydebug }
}
```

Figura D.18: Apartado *output* del archivo de configuración del segundo escenario.

Cuando se tiene listo el archivo de configuración, desde la terminal habrá que dirigirse a la carpeta que contiene el archivo *logstash.bat*, que es el directorio *bin*, y desde ahi especifacar cuál es la configuración que se quiere usar en el momento de ejecución mediante el comando *logstash -f escenario2.conf*. Al ejecutar se mostrarán por pantalla los datos modificados y llegarán a Elastic en dirección al índice *casas*, el cuál podrá ser consultado más adelante como se indica en el *Manual de Kibana*.

```
client = Elasticsearch("http://localhost:9200")

client.info()

def on_message(ws, message):
    message_json = json.loads(message)
    message_json["@timestamp"] = datetime.datetime.utcnow().isoformat()
    resp = client.index(index="websockets-data", body=message_json)
    print(message_json)

def on_error(ws, error):
    print(error)

def on_close(ws):
    print("### closed ###")
```

Figura D.19: Primera parte del script de carga del tercer escenario.

Escenario 3: ingesta a través de WebSocket a Elastic

En este tercer escenario, el origen de la ingesta de datos será un script que se suscribirá a un envío de datos en vivo por parte de la plataforma de criptodivisas *Finnhub*. Esta suscripción se realiza a través de un WebSocket programado en Python.

Para la ejecución de este script será necesario la importación de las librerías *websocket*, *elasticsearch*, *datetime* y *json* al principio del código. Una vez importadas, en la primera parte del código (ver ilustración D.19) , se especifica en la variable *client* la dirección del puerto en el que se ejecuta el servidor de ElasticSearch. Tras esto, el método *on message* define que el contenido del mensaje que se va a mandar está en formato JSON e incluirá la fecha en la que fue mandado. También indica que el contenido será mandado al índice *websockets data*.

Comenzando con la segunda parte del código (ver ilustración D.20), en el método *on open* se definen las divisas cuyo contenido va a ser incluído en el mensaje mandado a Elastic. Tras esto se define el método *main* en el cuál se especificará la key de la API de *Finnhub* utilizada como suscripción. También se especifica que mientrás el script esté operativo se sigan mandando datos hasta que se detenga con la función *run forever()*.

```
def on_open(ws):
    ws.send('{"type":"subscribe","symbol":"AAPL"}')
    ws.send('{"type":"subscribe","symbol":"AMZN"}')
    ws.send('{"type":"subscribe","symbol":"BINANCE:BTCUSDT"}')
    ws.send('{"type":"subscribe","symbol":"IC MARKETS:1"}')

if __name__ == "__main__":
    websocket.enableTrace(True)
    ws = websocket.WebSocketApp("wss://ws.finnhub.io?token=conrq51r01qm6hd153ogconrq51r01qm6hd153p0",
                                on_message=on_message,
                                on_error=on_error,
                                on_close=on_close)
    ws.on_open = on_open
    ws.run_forever()
```

Figura D.20: Segunda parte del script de carga del tercer escenario.

Este script es la piedra angular de este escenario, ya que tras su ejecución los datos llegarán a Elastic y serán mostrados en el *dashboard* correspondiente tal y como se mencionó en la documentación de la memoria.

Escenario 4: ingesta a través de WebSocket con Logstash

Para este cuarto escenario, el origen de la fuente de datos será el mismo que en el tercero, peró habrá una serie de cambios en el script de carga, de manera que en lugar de los datos ser mandados directamente a un índice de ElasticSearch, serán mandados a Logstash donde se procesarán y cargarán en Elastic.

La primera parte del script de carga de datos (ver ilustración D.21) está formada por la importación de las librerías *requests*, *json* y *websocket*, y por el método de renombrado de campos mandados por *Finnhub*, de manera que el campo que llega como *s* sea mandado como *Symbol*, el que llega como *p* sea mandado como *LastPrice*, el que llega como *t* sea mandado como *Timestamp*, el que llega como *v* sea mandado como *Volume*, y el que llega como *c* sea mandado como *TradeConditions* a Logstash.

En la segunda parte del script de carga de datos (ver ilustración D.22), se define el método *on message* el cuál especifica que los datos serán mandados en formato JSON al puerto local 8080 en el cuál estará escuchando Logstash a la espera de información. También se definen los métodos *on error* y *on close* que son meramente informativos de cara la visualización del envío de datos en la terminal.

```python
def rename_fields(message):
    data = json.loads(message)
    if "data" in data:
        renamed_data = []
        for item in data["data"]:
            renamed_item = {
                "Symbol": item.get("s"),
                "LastPrice": item.get("p"),
                "Timestamp": item.get("t"),
                "Volume": item.get("v"),
                "TradeConditions": item.get("c")
            }
            renamed_data.append(renamed_item)
        data["data"] = renamed_data
    return json.dumps(data)
```

Figura D.21: Primera parte del script de carga del cuarto escenario.

```python
def on_message(ws, message):
    url = 'http://localhost:8080'
    headers = {'Content-Type': 'application/json'}
    modified_message = rename_fields(message)
    response = requests.post(url, headers=headers, data=modified_message)
    print(response.text)

def on_error(ws, error):
    print(error)

def on_close(ws):
    print("### closed ###")
```

Figura D.22: Segunda parte del script de carga del cuarto escenario.

```
def on_open(ws):
    ws.send('{"type":"subscribe","symbol":"BINANCE:ETHUSDT"}')
    ws.send('{"type":"subscribe","symbol":"BINANCE:XRPUSDT"}')
    ws.send('{"type":"subscribe","symbol":"BINANCE:BTCUSDT"}')
    ws.send('{"type":"subscribe","symbol":"BINANCE:LTCUSDT"}')
    ws.send('{"type":"subscribe","symbol":"BINANCE:BCHUSDT"}')

if __name__ == "__main__":
    websocket.enableTrace(True)
    ws = websocket.WebSocketApp("wss://ws.finnhub.io?token=cp0dqk9r01qnigejuh40cp0dqk9r01qnigejuh4g",
                                on_message=on_message,
                                on_error=on_error,
                                on_close=on_close)
    ws.on_open = on_open
    ws.run_forever()
```

Figura D.23: Tercera parte del script de carga del cuarto escenario.

```
++Rcv raw: b'\x81~\x02\xf9{"data":[{"c":null,"p":3391.28,"s":"BINANCE:ETHUSDT","t":1719337736428,"v":1.0279},{"c":null,"p":3391.28,"s":"BI
NANCE:ETHUSDT","t":1719337736428,"v":0.9721},{"c":null,"p":61580,"s":"BINANCE:BTCUSDT","t":1719337737036,"v":0.00315},{"c":null,"p":3391.2
7,"s":"BINANCE:ETHUSDT","t":1719337736814,"v":0.002},{"c":null,"p":3391.27,"s":"BINANCE:ETHUSDT","t":1719337736814,"v":0.002},{"c":null,"p
":3391.27,"s":"BINANCE:ETHUSDT","t":1719337736814,"v":0.0002},{"c":null,"p":3391.24,"s":"BINANCE:ETHUSDT","t":1719337736814,"v":0.1958},{"
c":null,"p":61580,"s":"BINANCE:BTCUSDT","t":1719337736829,"v":0.00027},{"c":null,"p":3391.24,"s":"BINANCE:ETHUSDT","t":1719337736877,"v":0
.0037},{"c":null,"p":3391.24,"s":"BINANCE:ETHUSDT","t":1719337736877,"v":0.1584}],"type":"trade"}'
++Rcv decoded: fin=1 opcode=1 data=b'{"data":[{"c":null,"p":3391.28,"s":"BINANCE:ETHUSDT","t":1719337736428,"v":1.0279},{"c":null,"p":3391
.28,"s":"BINANCE:ETHUSDT","t":1719337736428,"v":0.9721},{"c":null,"p":61580,"s":"BINANCE:BTCUSDT","t":1719337737036,"v":0.00315},{"c":null
,"p":3391.27,"s":"BINANCE:ETHUSDT","t":1719337736814,"v":0.002},{"c":null,"p":3391.27,"s":"BINANCE:ETHUSDT","t":1719337736814,"v":0.002},{
"c":null,"p":3391.27,"s":"BINANCE:ETHUSDT","t":1719337736814,"v":0.0002},{"c":null,"p":3391.24,"s":"BINANCE:ETHUSDT","t":1719337736814,"v"
:0.1958},{"c":null,"p":61580,"s":"BINANCE:BTCUSDT","t":1719337736829,"v":0.00027},{"c":null,"p":3391.24,"s":"BINANCE:ETHUSDT","t":17193377
36877,"v":0.0037},{"c":null,"p":3391.24,"s":"BINANCE:ETHUSDT","t":1719337736877,"v":0.1584}],"type":"trade"}'
ok
```

Figura D.24: Salida por pantalla de los datos del script del cuarto escenario.

Para la tercera y última parte del script de carga de datos (ver ilustración D.23), en el método *on open* se definen las divisas que se mandarán a Logstash, y en el método *main* se especifica la *key* empleada como suscripción al servicio de envío de datos que ofrece *Finnhub*.

Una vez se ejecuta el script los datos serán mandados en forma *raw* (ver ilustración D.24) al puerto local 8080, donde Logstash está a la espera de recibir datos. Una vez son mandados allí, pasan por el filtro especificado en el archivo de configuración de este cuarto escenario, el cuál esta conformado por dos partes. En la primera parte del apartado de filtrado del archivo de configuración (ver ilustración D.25), se especifica que la información que va a llegar va a estar en formato JSON, que el campo *TradeConditions* va a ser eliminado del mensaje final, y que el tipo del campo *LastPrice* será *float*, que *Timestamp* será *integer* y *Volume* será *float*.

En la parte final del filtrado del archivo de configuración (ver ilustración D.26), una vez se tienen modificados los tipos de los campos *LastPrice* y

```
filter {

  json {
    source => "message"
  }

  mutate {
    remove_field => ["data.TradeConditions"]
  }

  mutate {
    convert => {
      "LastPrice" => "float"
      "Timestamp" => "integer"
      "Volume" => "float"
    }
  }
```

Figura D.25: Primera parte del filtrado del archivo de configuración del cuarto escenario.

```
ruby {
  code => '
    data = event.get("data")
    puts "Original data: #{data}"
    data.each { |item|
      total_price = item["LastPrice"] * item["Volume"]
      puts "Calculando el precio total del item: #{item}"
      puts "LastPrice: #{item["LastPrice"]}, Volume: #{item["Volume"]}"
      item["TotalPrice"] = total_price
      puts "TotalPrice: #{total_price}"
    }
    event.set("data", data)
  '
}
```

Figura D.26: Segunda parte del filtrado del archivo de configuración del cuarto escenario.

Volume, en lenguaje *Ruby* se ha programado la inserción de un nuevo campo *TotalPrice* calculado a partir del producto de estos dos campos.

Habiendo finalizado el archivo de configuración, se tienen en ejecución tanto Logstash como el script, los datos se irán mandando a Elastic con una estructura más entendible (ver ilustración D.27).

Figura D.27: Salida por pantalla de los datos una vez salen de Logstash hacia Elastic.

```
path = r'C:\Users\hugod\Desktop\TFG\ficheros\test.log'

customers = [123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 139]
payment_methods = ["Credit Card", "Debit Card", "PayPal"]
product_categories = ["Electronics", "Books", "Clothing"]
```

Figura D.28: Primera parte del generador de datos del quinto escenario.

Escenario 5: ingesta de data stream con MapReduce en el servidor mediante Logstash

En este quinto escenario se pretende aplicar MapReduce a datos en vivo generados a través de un script que irá añadiendo líneas a un fichero de tipo log llamado *test.log* que Logstash irá leyendo y actualizando el envío de sus datos a Elastic, de manera que pueda contener en un gran número de registros para poder aplicarles MapReduce.

Este fichero contendrá información sobre las transacciones que se han hecho en un negocio, incluyendo información como el identificador del cliente, el método de pago o la categoría del producto comprado.

En la primera parte de este script generador de datos se especifica tanto la ruta del fichero destino de los datos generados, como los valores sobre los que irá operando el método generador de los datos (ver ilustración D.28) . Para la ejecución de este script hará falta la importación de las librerías *json*, *random*, *time* y *datetime*.

El método *generate transaction* es el encargado de generar elecciones aleatorias sobre las que se han especificado previamente para los campos *customer id*, *amount*, *payment method* y *product category* (ver ilustración D.29) . También indicará que cada línea escrita en el archivo contendrá estos campos junto a un identificador de la transacción y la fecha en la que fue generado.

En esta última parte del script se establece un bucle en el que comenzando por la transacción número 1 se irán generando transacciones cada segundo, y no se detendrá hasta que se pare el script (ver ilustración D.30) .

Con esto, una vez se ejecuta el script, resultará en que el archivo *test.log* irá almacenando las transacciones en un formato entendible tanto para el usuario como para Logstash.

```
def generate_transaction(transaction_id, timestamp):
    customer_id = random.choice(customers)
    amount = round(random.uniform(20.0, 300.0), 2)
    payment_method = random.choice(payment_methods)
    product_category = random.choice(product_categories)

    transaction = {
        "transaction_id": str(transaction_id),
        "timestamp": timestamp.isoformat() + "Z",
        "customer_id": str(customer_id),
        "amount": amount,
        "payment_method": payment_method,
        "product_category": product_category,
    }
    return transaction
```

Figura D.29: Segunda parte del generador de datos del quinto escenario.

```
with open(path, 'a') as log_file:
    transaction_id = 1
    current_time = datetime.utcnow()

    while True:
        transaction = generate_transaction(transaction_id, current_time)
        log_file.write(json.dumps(transaction) + '\n')
        log_file.flush()

        transaction_id += 1
        current_time += timedelta(minutes=5)

        time.sleep(1)
```

Figura D.30: Tercera parte del generador de datos del quinto escenario.

```
{"transaction_id": "1", "timestamp": "2024-06-18T16:01:23.220156Z", "customer_id": "136",
"amount": 71.0, "payment_method": "Debit Card", "product_category": "Books"}
{"transaction_id": "2", "timestamp": "2024-06-18T16:06:23.220156Z", "customer_id": "133",
"amount": 28.99, "payment_method": "Debit Card", "product_category": "Books"}
{"transaction_id": "3", "timestamp": "2024-06-18T16:11:23.220156Z", "customer_id": "125",
"amount": 73.38, "payment_method": "Credit Card", "product_category": "Electronics"}
{"transaction_id": "4", "timestamp": "2024-06-18T16:16:23.220156Z", "customer_id": "139",
"amount": 296.39, "payment_method": "Debit Card", "product_category": "Clothing"}
{"transaction_id": "5", "timestamp": "2024-06-18T16:21:23.220156Z", "customer_id": "123",
"amount": 256.04, "payment_method": "Debit Card", "product_category": "Books"}
```


Figura D.31: Estructura de la fuente de los datos del quinto escenario.

Cuando el archivo se va llenando de líneas, se tiene Logstash en ejecución simultáneamente, de manera que se vaya ejecutando el método *aggregate*, el cual Logstash ofrece para fusionar muchas líneas de información en una con un determinado criterio en el que se le especifica sobre que campo de los datos va a centrarse. Las agrupaciones se van a realizar en función del método de pago utilizado en cada transacción, y cada 20 segundos se refrescarán estas agrupaciones, y si tras 40 segundos no ha ocurrido ningún cambio, se detendrá el servicio.

Machine Learning

En este último caso planteado en el cuál se aplica una serie de algoritmos que ofrece la librería Sklearn sobre el popular *dataset* Iris, la manera de ingestar los datos se planteó de manera que los datos de Iris estuvieran presentes en un índice en Elastic, estos datos se cargarán en el script que aplica los algoritmos, y una vez cargados cada algoritmo devolverá los datos procesados a un índice individual para cada uno.

Para resolver esta estructura planteada, se generaron dos scripts. En el primero de ellos se crea un índice en Elastic que contiene el *dataset* Iris. Son necesarias las librerías *pandas*, *sklearn* y *elasticsearch*. De manera que inicialmente se cargan los datos de Iris tal cuál viene de SKLearn, con el mismo nombre de los campos (esto va a ser importante en el segundo script para realizar operaciones sobre ellos) (ver ilustración D.32). También se establece la conexión con ElasticSearch indicando el puerto local en el que se ubica el servidor.

En la segunda parte del primer script se va a definir tanto el nombre del índice que contendrá el *dataset* Iris como el nombre de las propiedas

```
# Cargar el dataset de Iris
iris = datasets.load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = iris.target

# Conexión a Elasticsearch
es = Elasticsearch([{'host': 'localhost', 'port': 9200, 'scheme': 'http'}])
```

Figura D.32: Parte inicial del primer script de Machine Learning.

```
# Definición del índice
index_name = 'irisdata'

mapping = {
    "mappings": {
        "properties": {
            "sepal length (cm)": {"type": "float"},
            "sepal width (cm)": {"type": "float"},
            "petal length (cm)": {"type": "float"},
            "petal width (cm)": {"type": "float"},
            "target": {"type": "integer"}
        }
    }
}

# Se crea el índice con el mapeo en caso de que no existiese
if not es.indices.exists(index=index_name):
    es.indices.create(index=index_name, body=mapping)
```

Figura D.33: Segunda parte del primer script de Machine Learning

del mismo junto con el tipo que son (ver ilustración D.33), de manera que coincidan con las del *dataset* original para que no haya errores.

Una vez se tiene toda la información del destino de los datos cargada, se procede al envío de los mismos (ver ilustración D.34) de manera que

```
# Se cargan los datos en Elasticsearch
for i, row in df.iterrows():
    # Se convierte cada fila en un diccionario
    doc = row.to_dict()
    res = es.index(index=index_name, id=i, body=doc)
    print(res['result'])
```

Figura D.34: Parte final del primer script de Machine Learning.

```
# Función para cargar datos desde Elasticsearch
def load_data_from_elasticsearch(url, index):
    query = {
        "query": {
            "match_all": {}
        }
    }
    response = requests.get(f'{url}/{index}/_search?size=10000', json=query, auth=('elastic', 'elastic'))
    hits = response.json()['hits']['hits']
    data = [hit['_source'] for hit in hits]
    df = pd.DataFrame(data)
    return df

# Función para crear un índice en Elasticsearch
def create_index(url, index):
    response = requests.put(f'{url}/{index}', auth=('elastic', 'elastic'))
    print(response.text)
```

Figura D.35: Definición de funciones auxiliares para el segundo script de Machine Learning.

se vayan mostrando por pantalla en la terminal a medida que van siendo cargados.

Tras ejecutar este primer script los datos ya están presentes en Elastic, y para dividir el trabajo se creó un segundo script en el que se trabajara sobre esos datos cargados. En este segundo script lo primero que se hacer es definir tanto la función *load data from elasticsearch* como *create index* que lo que van a hacer es facilitar la carga de datos del índice que se le indique pasado por valor, y crear nuevos índices con los nombres indicados para poder almacenar los datos que devuelvan los algoritmos de manera ordenada (ver ilustración D.35). Están son funciones auxiliares en futuros pasos del script.

```
# URL de Elasticsearch y nombre del índice
url = 'http://localhost:9200'
index = 'irisdata'

# Carga de datos desde Elasticsearch
df = load_data_from_elasticsearch(url, index)

# Separar características y etiquetas
X = df.drop(columns=['target']).values
y = df['target'].values

# Se crean los índices en Elasticsearch para almacenar los resultados
create_index(url, 'iris_clasificacion')
create_index(url, 'iris_regresion')
create_index(url, 'iris_clustering')
create_index(url, 'iris_pca')
```

Figura D.36: Definición de operaciones auxiliares y carga de datos para el segundo script de Machine Learning.

En la segunda parte del script se van a seguir realizando operaciones auxiliares para poder operar con los algoritmos. Se establece la URL y el nombre del índice a cargar, que es el mismo en el que se ingestaron los datos de Iris en el script previo, de manera que los datos sean cargados y separados para poder trabajar sobre ellos. También se generan los distintos índices que contendrán los datos obtenidos como resultado de la ejecución de cada algoritmo.

Una vez está toda la información cargada y lista para ser usada, se comienza por el primer algoritmo utilizado, que va a ser uno de clasificación, concretamente *KNN* o K vecinos más cercanos, en el cuál mediante los métodos proporcionados por Sklearn como *train test split* o *fit*, sobre el se creará y entrenará un clasificador en función de los tres vecinos más cercanos (ver ilustración D.37), obteniendo así el rendimiento de este clasificador, información la cual será exportada a Elastic así como todos los datos que se han procesado en la ejecución del algoritmo, que irán destinados al índice *iris clasificacion* para su posterior uso y exposición en Kibana (ver ilustración D.38).

```
# Clasificación: K vecinos más cercanos (KNN)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Se divide el conjunto de datos en conjuntos de entrenamiento (80%) y prueba (20%).
scaler = StandardScaler()
# Se estandarizan las características para que todas tengan la misma escala
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

knn = KNeighborsClassifier(n_neighbors=3)
# Se crea un clasificador KNN utilizando los 3 vecinos más cercanos para la clasificación.

knn.fit(X_train_scaled, y_train)
# Se entrena el clasificador KNN utilizando el conjunto de datos de entrenamiento escalado
```

Figura D.37: Generación del clasificador y entrenamiento.

```
accuracy = knn.score(X_test_scaled, y_test)
# Se evalúa el rendimiento del clasificador utilizando el conjunto de datos de prueba es

print("Accuracy of KNN Classifier:", accuracy)

# Se guardan los resultados de la clasificación en Elasticsearch
url_clasificacion = f'{url}/iris_clasificacion/_doc'
y_test_serializable = y_test.astype(int).tolist()

for i in range(len(X_test)):
    etiqueta_predicha = int(knn.predict(X_test_scaled[i].reshape(1, -1))[0])
    data = {
        "caracteristica_1": X_test[i, 0],
        "caracteristica_2": X_test[i, 1],
        "caracteristica_3": X_test[i, 2],
        "caracteristica_4": X_test[i, 3],
        "etiqueta_real": y_test_serializable[i],
        "etiqueta_predicha": etiqueta_predicha,
        "precision_knn": accuracy
    }
    response = requests.post(url_clasificacion, json=data, auth=('elastic', 'elastic'))
    print(response.text)
```

Figura D.38: Cálculo de la precisión del clasificador y envío de datos a Elastic.

```
# Regresión lineal
print("Columnas del DataFrame para regresión:", df.columns)  # Añadir esta línea para ver

X = df[['petal length (cm)']].values   # Tomamos solo la tercera característica (longitud
y = df['petal width (cm)'].values      # La cuarta característica será nuestro objetivo (a

# Se divide el conjunto de datos en conjuntos de entrenamiento y prueba igual que antes
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Escalado de características
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Se crea y entrena el modelo de regresión lineal
lr = LinearRegression()
lr.fit(X_train_scaled, y_train)
```

Figura D.39: Generación del modelo de regresión y los conjuntos de entrenamiento.

El siguiente algoritmo al que se le ha dado uso es el de regresión lineal, en el cuál tomando como referencia la tercera característica (longitud del pétalo) para el eje X, y la cuarta característica (anchura del pétalo) para el eje Y, se cree un modelo que sea capaz de predecir un comportamiento típico de los datos. Para ello se le entrena mediante el método *fit* con un conjunto de datos de entrenamiento obtenido de los totales (ver ilustración D.39). Posteriormente se evalúa su rendimiento y se manda a Elastic junto con el resto de datos empleados al índice *iris regresion* r (ver ilustración D.40).

En este tercer algoritmo se va a emplear uno de *clustering*, el *K Means*, en el cuál se van a generar tres clusters de manera que se cree un modelo que devuelva una visualización de los tres en forma de gráfico de dispersión de la longitud y anchura de los sépalos (ver código fuente en D.41). Una vez se obtiene el gráfico, los datos son mandados a Elastic para poder visualizarlos en un *dashboard*.

Por último se va a hacer uso de otro algoritmo visual de reducción de dimensionalidad, *PCA*. Lo que se quiere, es agrupar las instancias por su similitud, generando un gráfico bidimensional. Se toman los dos primeros componentes equivalentes a una combinación lineal de los atributos originales (ver ilustración D.42). Se entrena el modelo y se genera un gráfico de dispersión con estos dos componentes el cuál será enviado a Elastic junto con toda la información involucrada.

```
# Se hacen predicciones
y_pred = lr.predict(X_test_scaled)

# Se evalua el rendimiento del modelo
r2_score = lr.score(X_test_scaled, y_test)
print("R^2 Score of Linear Regression:", r2_score)

# Los resultados de la regresión son mandados a Elasticsearch
url_regresion = f'{url}/iris_regresion/_doc'

for i in range(len(X_test)):
    data = {
        "longitud_petalo": X_test[i, 0],
        "anchura_petalo_real": y_test[i],
        "anchura_petalo_predicha": y_pred[i],
        "r2_score": r2_score
    }
    response = requests.post(url_regresion, json=data, auth=('elastic', 'elastic'))
    print(response.text)
```

Figura D.40: Cáclulo del rendimiento del modelo y envío de datos a Elastic.

```
# Clustering: K-Means
X = df.drop(columns=['target']).values

# Se crea el modelo de clustering K-Means
kmeans = KMeans(n_clusters=3)  # 3 clusters porque hay 3 tipos en el conjunto de datos Ir

# Se entrena el modelo de clustering
kmeans.fit(X)

# Visualización de los clústeres
plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red')
plt.title("Clustering K-Means en el conjunto de datos Iris")
plt.xlabel("Longitud del Sépalo (cm)")
plt.ylabel("Anchura del Sépalo (cm)")
plt.show()
```

Figura D.41: Estructura del algoritmo *K Means* del segundo script de Machine Learning.

```
# Reducción de dimensionalidad: PCA
X = df.drop(columns=['target']).values
y = df['target'].values

# Se crea una instancia de PCA reduciendo la dimensionalidad a dos
pca = PCA(n_components=2)

# Se entrena el modelo PCA con las características de entrada
X_pca = pca.fit_transform(X)

# Se muestran los datos reducidos en un gráfico de dispersión
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis')
plt.title("PCA del Iris Dataset")
plt.xlabel("Componente Principal 1")
plt.ylabel("Componente Principal 2")
plt.show()
```

Figura D.42: Estructura del algoritmo de reducción de la dimensionalidad del segundo script de Machine Learning.

Tras ejecutar los dos scripts los datos llegarán los diferentes índices que se han ido indicando en ambos. El *dataset* Iris estará almacenado en el índice *irisdata* (ver ilustración D.43), los datos del algoritmo de clasificiación en el índice *iris clasificacion* (ver ilustración D.44), los datos del algoritmo de regresión en el índice *iris regresion* (ver ilustración D.45), los datos del algoritmo de clustering en el índice *iris clustering* (ver ilustración D.46), y por último los datos del algoritmo de reducción de dimensionalidad en el índice *iris pca* (ver ilustración D.47).

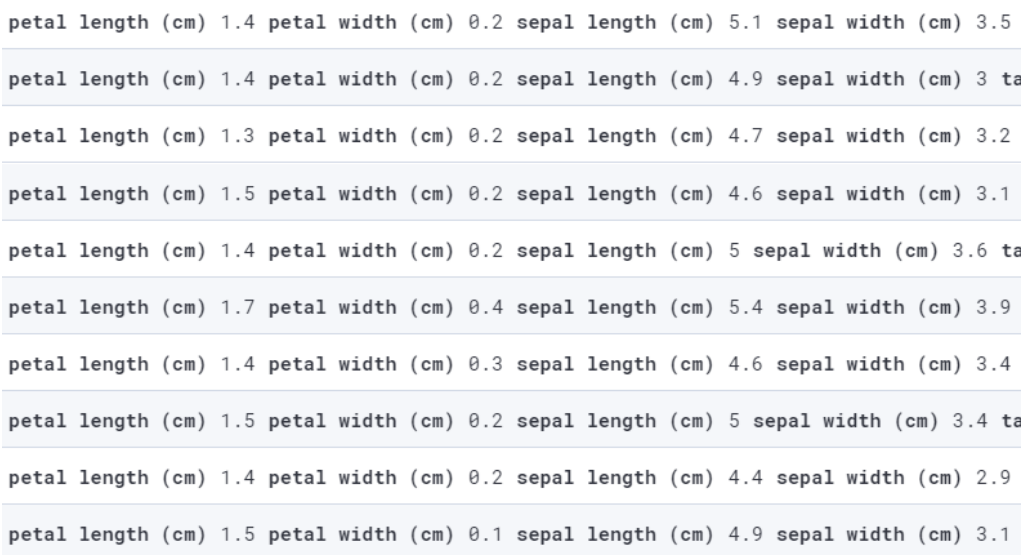

Figura D.43: Índice *irisdata* con información de los datos de Iris.

Figura D.44: Índice *iris clasificacion* con información de los datos procesados por el algoritmo de clasificación.

Figura D.45: Índice *iris regresion* con información de los datos procesados por el algoritmo de regresión.

Figura D.46: Índice *iris clustering* con información de los datos procesados por el algoritmo de clustering.

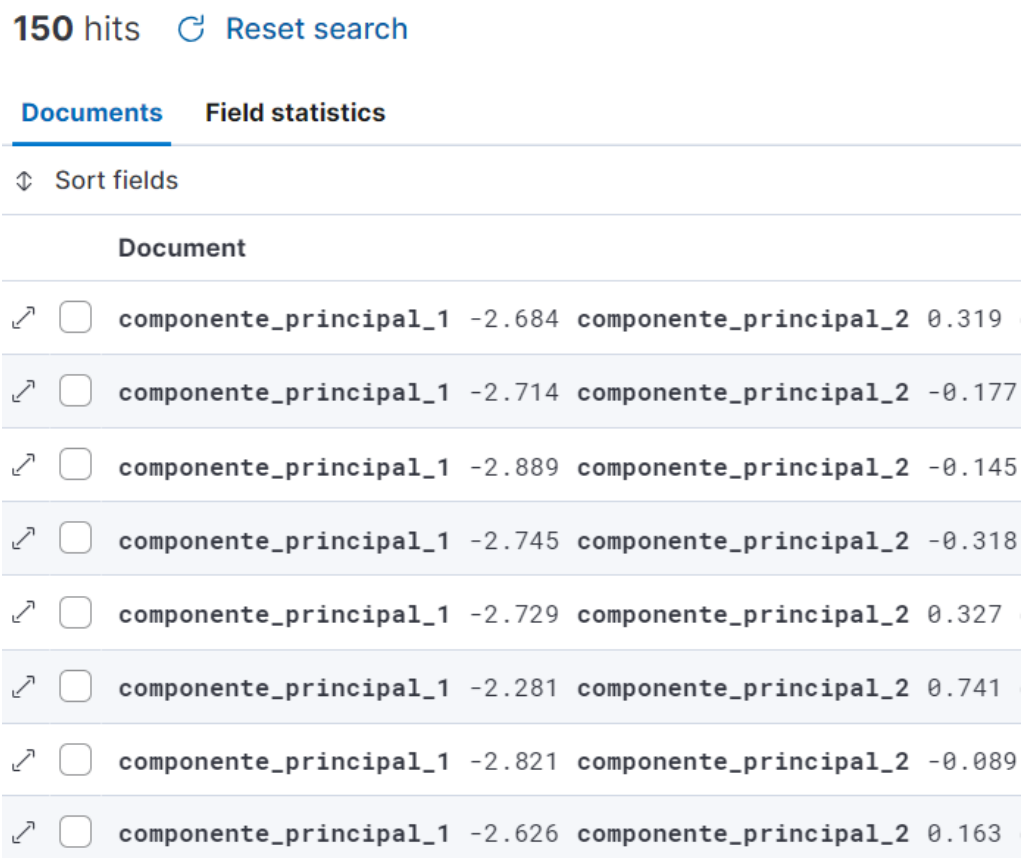

Figura D.47: Índice *iris pca* con información de los datos procesados por el algoritmo de reducción de dimensionalidad.

Apéndice E

Manual de Kibana

E.1. Introducción

Este anexo en un TFG convencional recibiría el nombre de *Manual de usuario*, pero al consistir este TFG de un estudio, se ha decidido modificarlo a *Manual de Kibana*, y se indagará en todas las posibilidades y caracterísiticas avanzas que ofrece el programa más visual e intuitivo de los tres componentes del stack ELK.

E.2. Requisitos de usuarios

Este *software* no requiere de gran cantidad de requisitos ni es muy demandante en recursos, puesto que trabaja sobre los datos procesados y optimizados previamente. Para su uso se requieren los siguientes requisitos:

- Acceso a Internet
- Acceso a un navegador convencional como pueden ser Google Chrome, Microsoft Edge o Brave entre otros.
- Acceso a un stack ELK.
- Que el sistema disponga de espacio un espacio de procesamiento suficiente para cargar y analizar las visualizaciones de datos.
- Habilidad para comprender visualizaciones de datos.

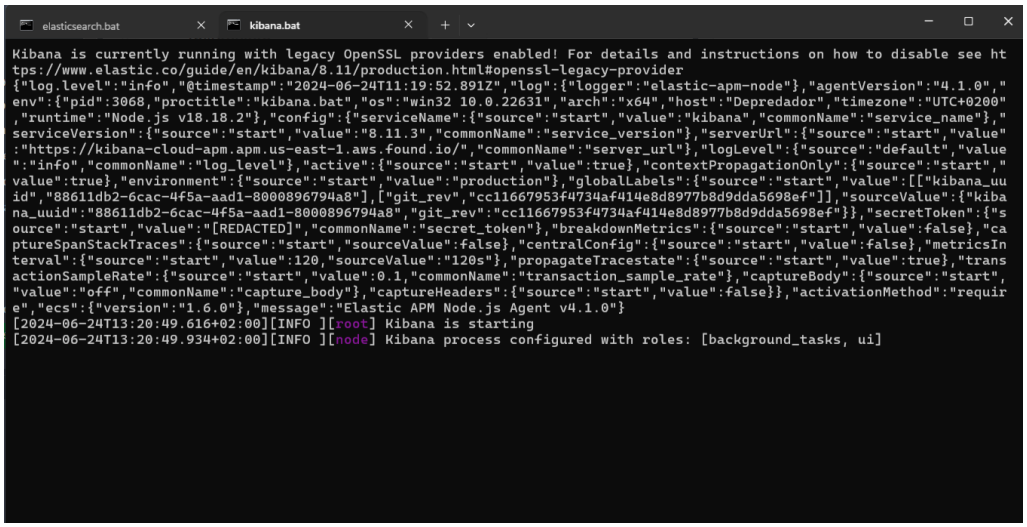

Figura E.1: Visualización de terminal una vez se ejecute kibana.bat

E.3. Instalación

El programa dispone de un sitio web oficial donde su descarga e instalación es sencilla e intuitiva. También se incluyen guías y videotutoriales en caso de complicaciones en la instalación o ejecución.

E.4. Manual del usuario

Este software de visualización de datos, una vez sea descargado de su página oficial, se tendrá que ejecutar el fichero *kibana.bat* en el directorio *bin* como se muestra en la ilustración E.1.

Pasados unos segundos, la terminal indicará que el servicio de Kibana está activo y para acceder a el nos iremos al navegador y nos dirigemos al puerto local 5601, en el cual se nos mostrará la interfaz gráfica de Kibana (ver ilustración E.2).

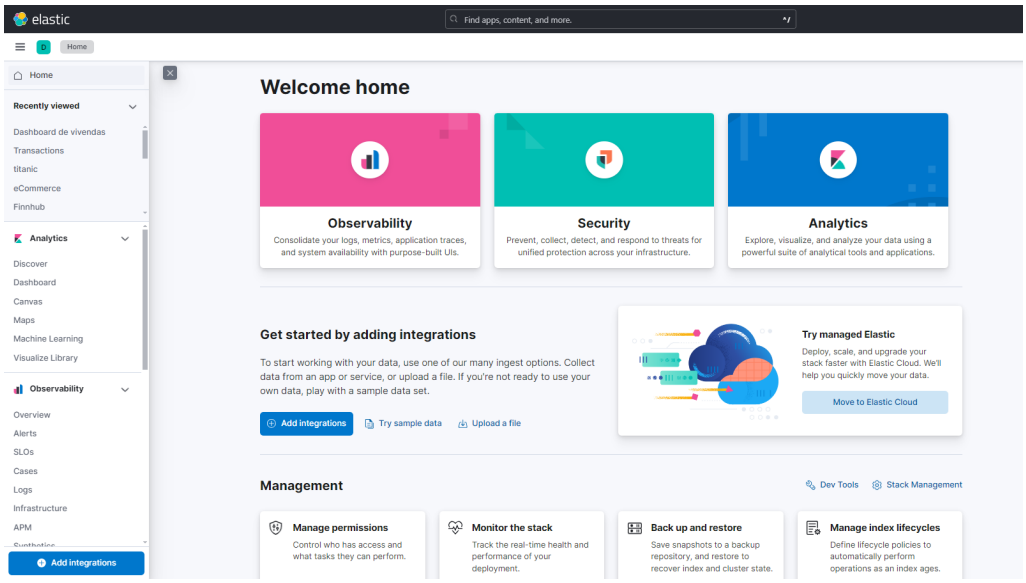

Figura E.2: Interfaz gráfica inicial de Kibana.

Una vez se llega a esta pantalla, se pueden observar las diferentes funcionalidades que ofrece Kibana. La primera en la que se va a indagar, y que es importante de cara a comprobar que los datos de los índices de ElasticSearch son los deseados, es la funcionalidad *Analytics* (ver ilustración E.3). La parte que más interesa es la de *Discover*, en la cuál se van a mostrar todos los registros presentes en el índice que se indique. En este ejemplo se van a visualizar los datos del índice *titanic* (ver ilustración E.4), como se puede observar en la columna de la izquierda se muestran los campos disponibles junto al tipo al que pertenecen, pudiendo realizar filtrados personalizados pero no modificaciones. En la sección principal llamada *Documents*, se muestran en detalle todos los datos de los registros presentes en el índice, permitiendo indagar más profundamente por si interesa alguno en concreto (ver ilustración E.5).

Un apartado que resulta interesante de la funcionalidad *Discover* es el de *Field statistics* (ver ilustración E.6), en el cuál se muestran datos estadísticos útiles y precisos de cada campo presente en el índice, como pueden ser los valores únicos o la distribución de los valores.

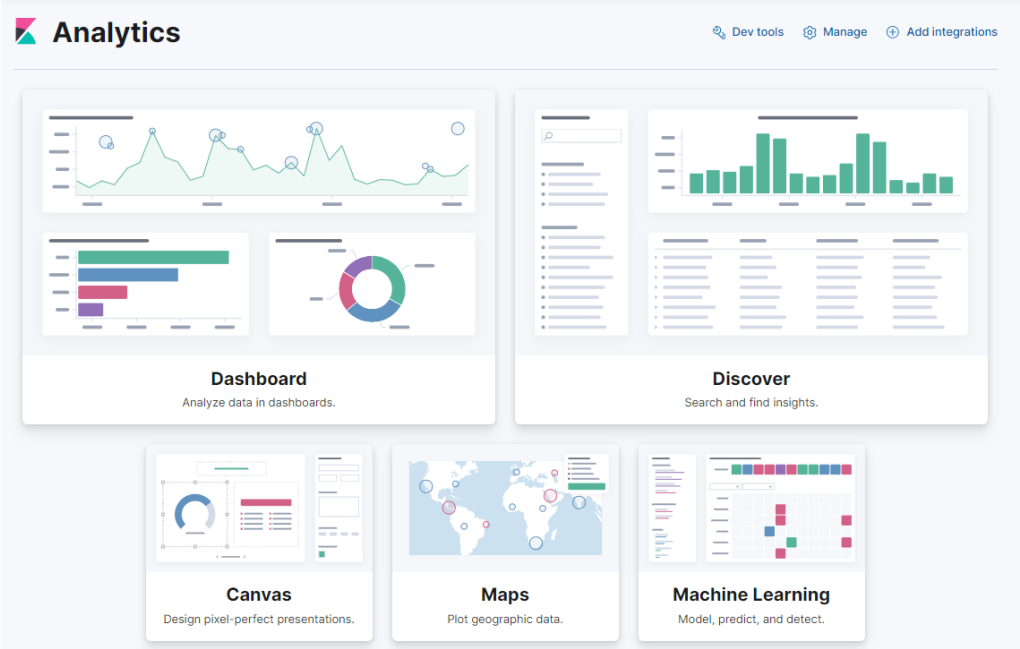

Figura E.3: Visión general de la funcionalidad *Analytics*.

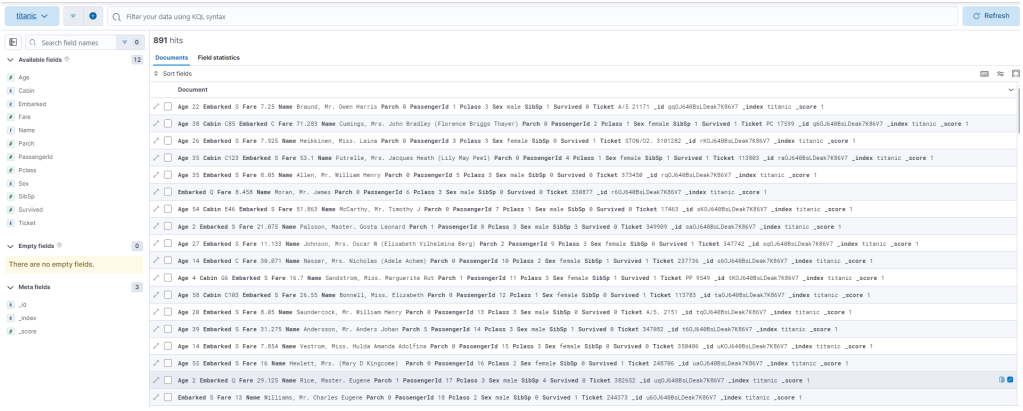

Figura E.4: Visualización del *Discover* de los datos del índice *titanic*.

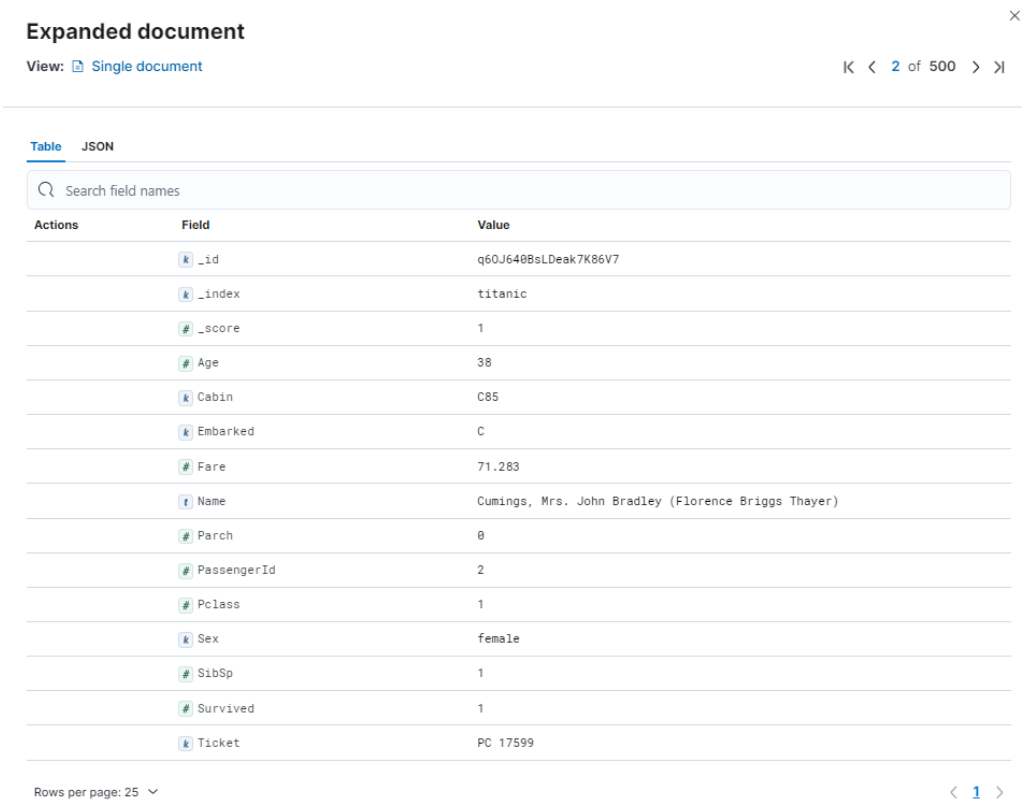

Figura E.5: Visualización de un registro en concreto.

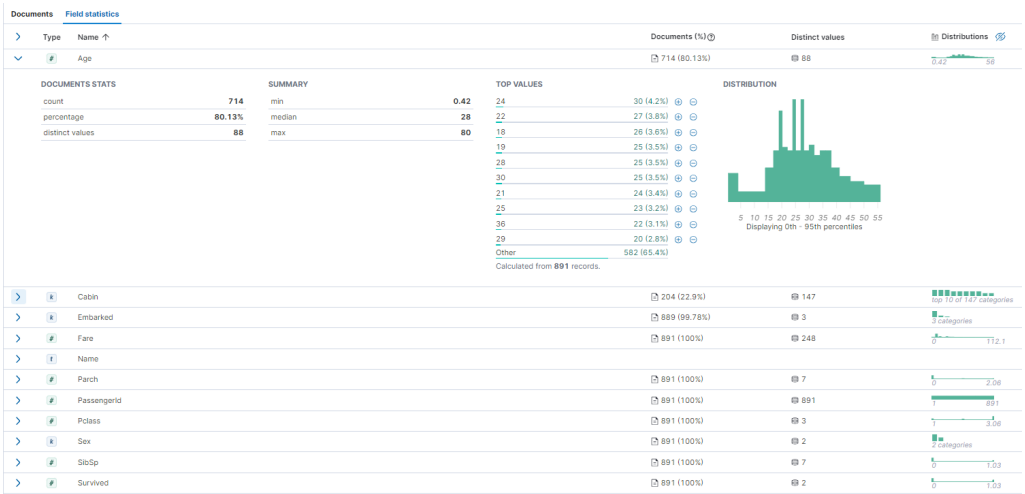

Figura E.6: Visualización de *Field Statistics*.

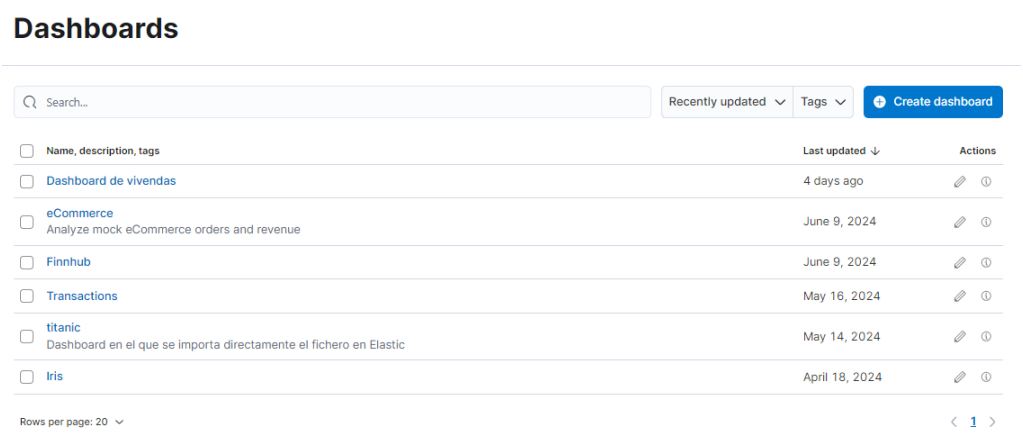

Figura E.7: Visualización inicial de la funcionalidad *Dashboards*.

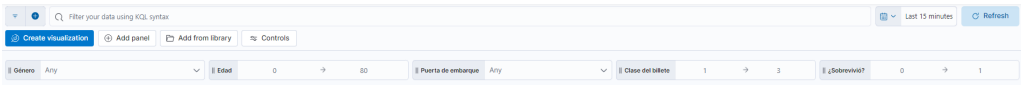

Figura E.8: Apartado de filtrado del *dashboard*.

Cuando se tiene comprendido la funcionalidad *Discover*, se puede pasar a la parte más interesante, que es la funcionalidad *Dashboards* (ver ilustración E.7) , en la cuál se realiza la parte más visual e intuitiva de todo el sistema ELK.

En esta sección, al igual que en la anterior, se va a explicar en el *dashboard* proveniente del índice *titanic*, equivalente al primero de los escenarios tratados. La primera función a destacar es la de la posibilidad de filtrar los campos en función de los valores que se quieran (ver ilustración E.8). Los filtros se añaden desde la opción *Controls*, pudiendo elegir si el filtro va a ser temporal o en función de los campos presentes. En este ejemplo se ha establecido un filtrado en función de la edad del pasajero (ver ilustración E.9), pudiendo configurar la etiqueta mostrada, el campo sobre el que se aplica el filtro o el tamaño que tendrá en el *dashboard*.

Antes de entrar en las visualizaciones, se quiere destacar la opción de permitir añadir un panel con toda la información de los datos sobre los que está trabajando el *dashboard*, a modo de atajo sin entrar al *Discover*.

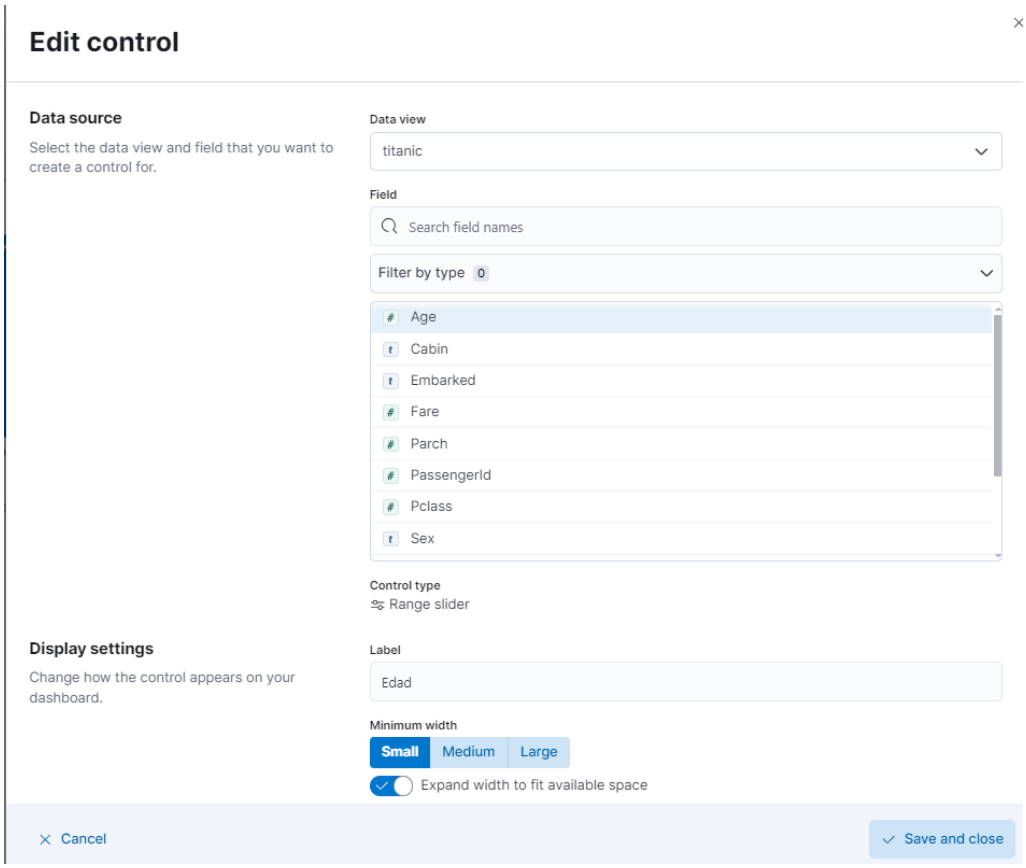

Figura E.9: Edición de un filtro del *dashboard*.

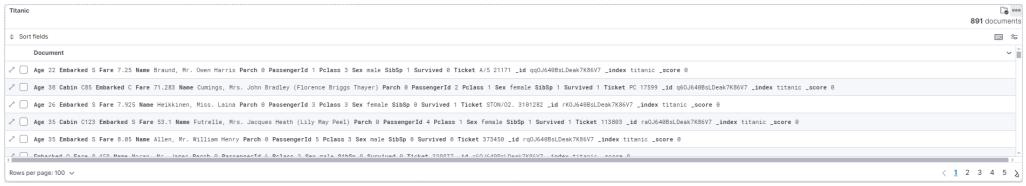

Figura E.10: Panel mostrando los datos del índice del *dashboard*.

Esto se puede realizar desde la opción *Add Library* (ver ilustración E.10), especificando el índice que se quiere mostrar.

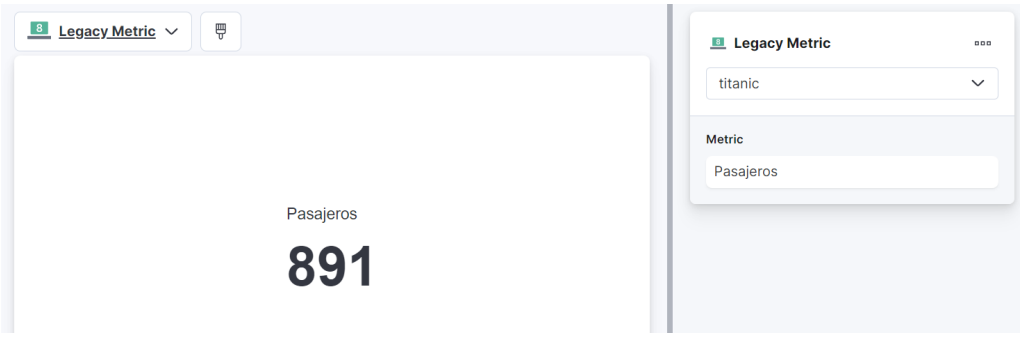

Figura E.11: Métrica del conteo del número de parajeros.

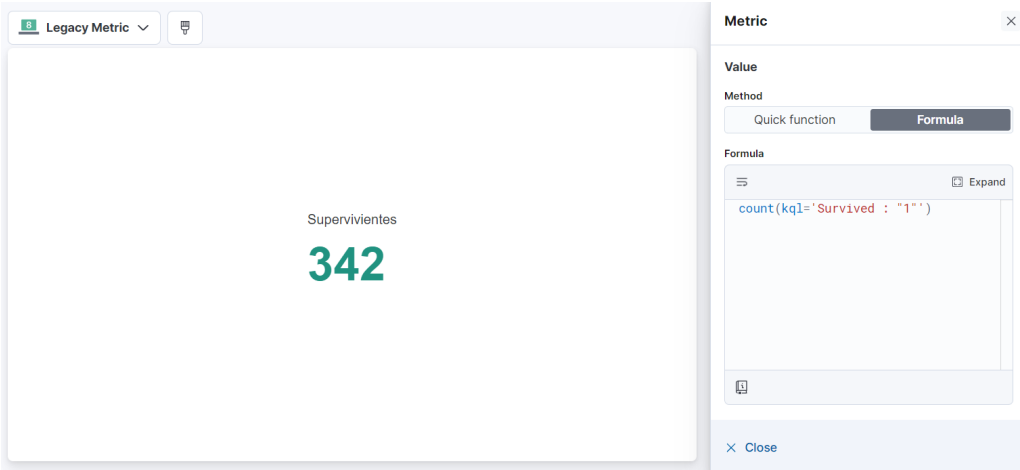

Figura E.12: Métrica de conteo del número de supervivientes.

Habiendo explicado estas funciones secundarias que ofrece la funcionalidad *Dashboards* de Kibana, es momento de exponer la función que permite crear visualizaciones con los datos del índice. Esto se realiza desde la opción *Create visualization*. La visualización más sencilla es la de *Metric*, en la cuál se puede especificar que se le aplique una función sencilla a un campo en concreto, en este ejemplo se ha hecho un conteo de todos los registros (ver ilustración E.11), y en este otro un conteo solo de los supervivientes con una fórmula usando sintaxis KQL (ver ilustración E.12).

Otra visualización interesante es la de los gráficos de barras, en los cuáles hay que especificar que campo se quiere usar para cada eje, y si se quiere aplicar alguna función sobre los mismos.

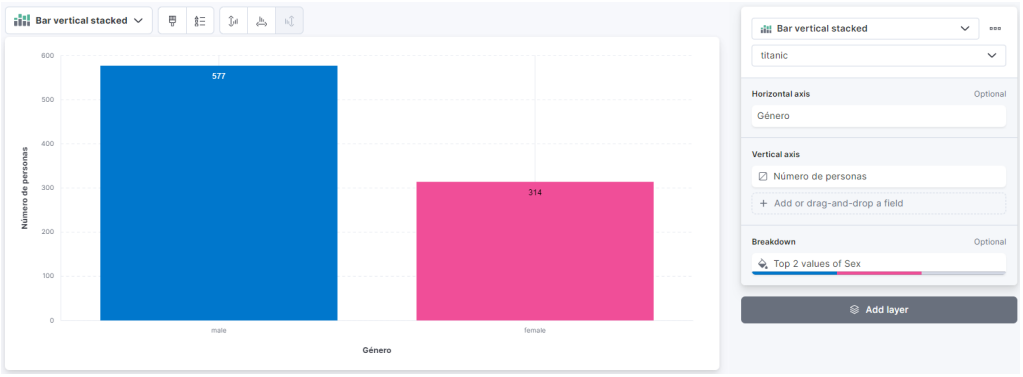

Figura E.13: Gráfico de barras mostrando el número de pasajeros por género.

En este ejemplo se está realizando un conteo del número de pasajeros clasificados por género (ver ilustración E.13, tomando como referencia el campo *Género* para el eje horizontal representando en color azul los hombres y en rosa las mujeres, y el campo *Records* para el eje vertical especificando que se realize un conteo de los valores.

Otro tipo de visualización que resulta intersante es la del gráfico en forma de donut, en el cuál en este ejemplo se muestra el porcentaje de pasajeros por clase de billete (ver ilustración E.14, especificando que se va a realizar un filtrado por intervalos del campo *Pclass* que es el que contiene la información pudiendo modificar las formas y colores del gráfico.

Un gráfico presente en este *dashboard* y que resulta interesante visualmente es el de un gráfico de barras apiladas (ver ilustración E.15), en este ejemplo se muestra en el eje horizontal la edad de los pasajeros y en el eje vertical el número de pasajeros estableciendo como diferencidador para el apilamiento el género de los mismos, esto se hace desde la función *Breakdown* tomando como referencia los valores del campo *Sex*.

Además de las visualizaciones mencionadas, Kibana ofrece visualizaciones similares en forma gráficos de barra horizontales, gráficos de area, mapas de calor o mosaicos. También incluye paneles más avanzados que requieren de datos más detallados como pueden ser mapas o streams de un log (ver ilustración E.16, en este *dashboard* de la página oficial de Kibana se pueden observar algunos de estos paneles (ver ilustración E.17.

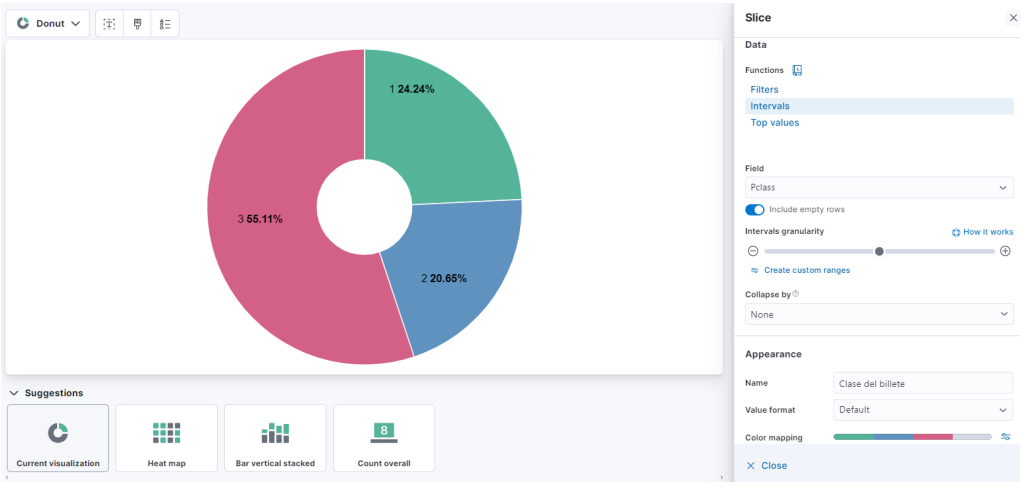

Figura E.14: Gráfico de donut mostrando el porcentaje de pasajeros por clase de billete.

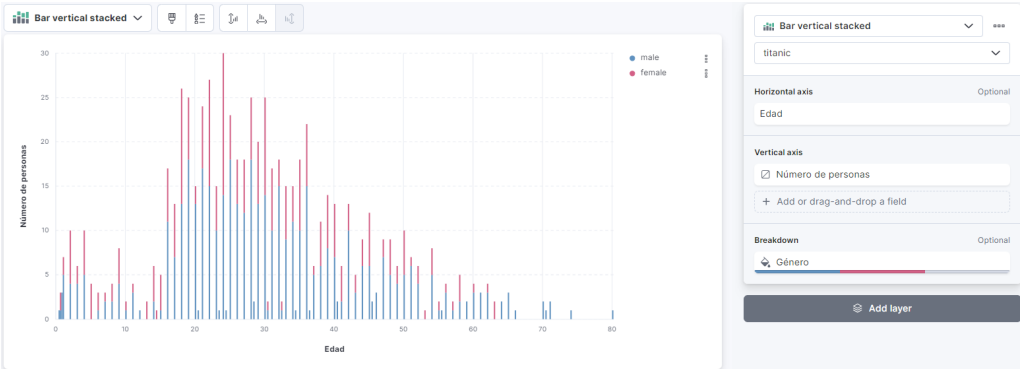

Figura E.15: Gráfico de barras apiladas mostrando la edad del número de pasajeros en función del género.

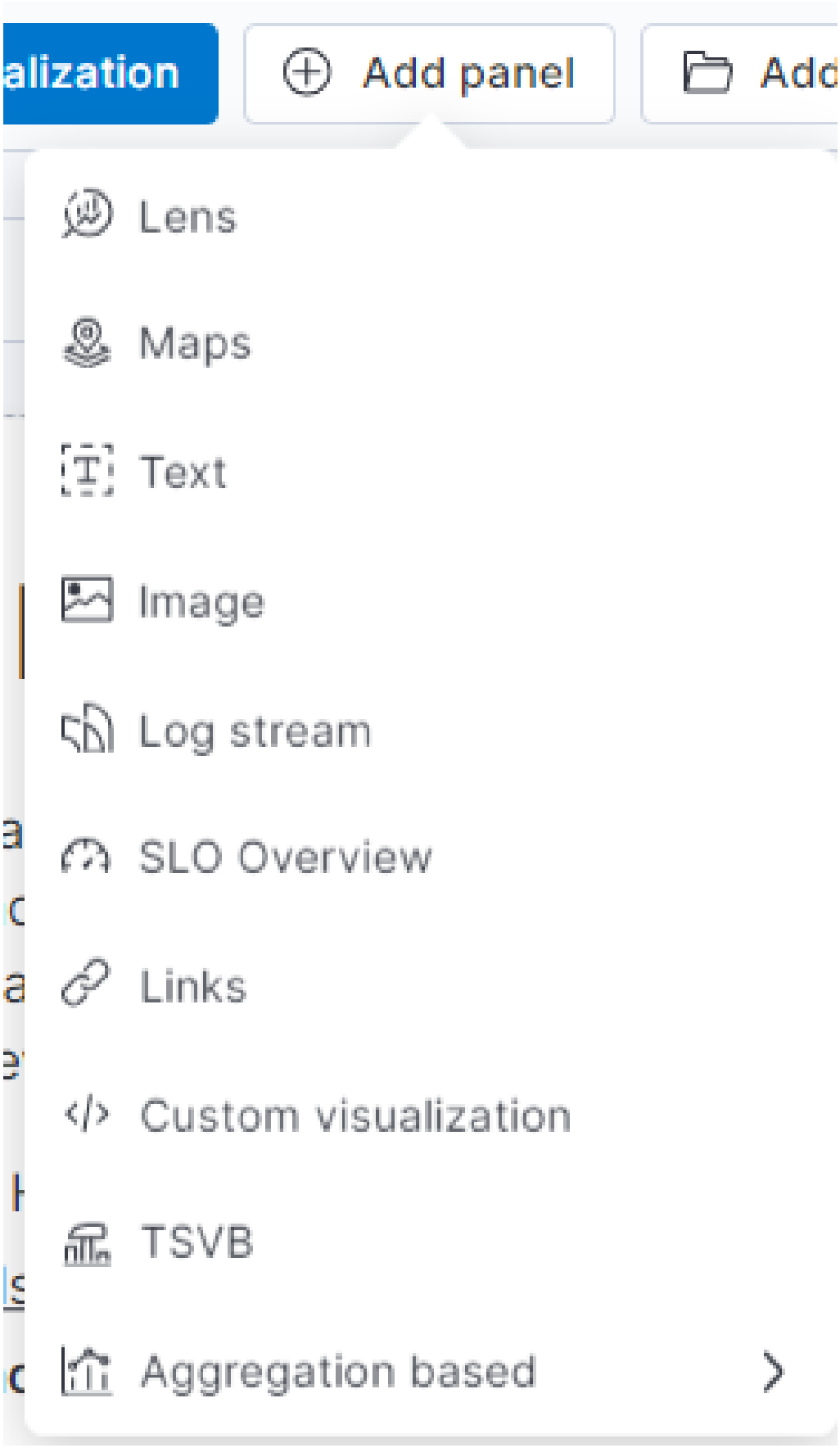

Figura E.16: Paneles más avanzados que ofrece Kibana.

Figura E.17: *Dashboard* con paneles avanzados de Kibana.

Una vez se da por finalizado la adición de visualizaciones y paneles al gráfico, este se puede guardar en el estado actual para poder ser modificado o exportado posteriormente desde la opción *Save* en la esquina superior derecha. También se incluye una opción para visualizar el *dashboard* como si se fuera un espectador desde *Switch to view mode* (ver ilustración E.18, permitiéndo comprobar que todo está correcto.

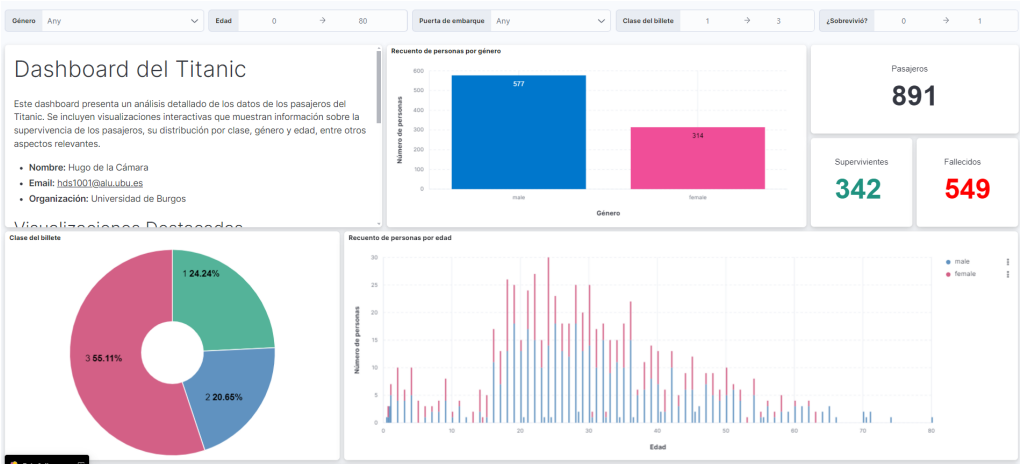

Figura E.18: Visualización general del *dashboard* desde el modo espectador de Kibana.

Apéndice F

Anexo de sostenibilización curricular

F.1. Introducción

En los tiempos actuales, la sostenibilidad y el bueno uso de los recursos que se tienen son temas que han cobrado especial relevancia en todos los ámbitos de la vida. Por lo que en este anexo lo que se pretende es hacer una reflexión personal sobre cómo se han tratado estos temas a lo largo del estudio en los sistemas ELK.

F.2. Impacto ambiental

El impacto ambiental y social de las tecnologías que usamos en nuestro dia a dia cada vez es mayor, y una de las motivaciones de este TFG era comprender de que manera el uso de un sistema ELK puede ayudar a reducir el impacto ambiental de los sistemas de la información. Mediante la monitorización de procesos se reduce el uso de energía, ayudando a reducir el impacto en la huella de carbono, así como identificar ineficiencias operativas.

F.3. Correcto uso de los recursos

Para poder mejorar la sostenibilidad es clave saber gestionar correctamente los recursos disponibles. En este estudio se han aplicado configuraciones para optimizar el rendimiento y reducir el consumo de estos recursos. Kibana permite identificar de manera más rápida y eficiente que áreas se pueden

mejorar del proyecto ETL, así como ayudar a la toma de decisiones de cara a poder optimizar el uso de los recursos.

F.4. Buenas prácticas

Cabe destacar que a lo largo de este TFG se han hecho uso de prácticas responsables y éticas a la hora del manejos de los datos y la información, respetando tanto la privacidad como la seguridad de los datos procesados. Elastic a su vez permite usar medidas de seguridad para garantizar que solo los usuarios autorizados acceden y manejan la información sensible presente en el sistema.

Bibliografía

[1] David Brimley. Latex, wikipedia, la enciclopedia libre. https://https://www.elastic.co/es/blog/what-is-an-elasticsearch-index. [Consultado el 21 de junio de 2024].