Monitorización de datos en tiempo real usando ELK y desarrollo de una web de visualización y gestión

UNIVERSITAT DE BARCELONA

Oscar Emanuel Lima Torrico

Facultad de Matemáticas e Ingeniería Informática

Universidad de Barcelona

Supervisor

Dr. Sergio Escalera Guerrero

En cumplimiento parcial de los requisitos para el grado de

Ingeniería Informática

Junio, 2019

Agradecimientos

Me gustaría agradecer a mis compañeros de trabajo quienes me han ayudado y facilitado el aprendizaje de todo el sistema y tecnologías que envolvían este proyecto, así como a mi tutor de la universidad, el Dr. Sergio Escalera, por animarme en todo momento y la empatía que ha mostrado con las decisiones que he tomado, sus correcciones de este proyecto, y su ayuda en cuanto la estructura de esta memoria. Por último y no menos importante querría agradecer a mis familiares y amigos por todo el apoyo y ánimo que me han brindado.

Este trabajo está dedicado a mis queridos padres, quienes siempre han sido una fuente de fuerza e inspiración y me han apoyado moral, emocional y financieramente a lo largo de toda mi vida.

Abstract

In medium to large projects, monitoring and organizing services begins to be important, since it increases the scalability and contributes to the detection of issues and the taking of measures quickly. My work has consisted, overall, in developing a monitoring project. This implies, on the one hand, monitoring different aspects of the logs that are generated and sent in the calls to services that my company offers to the client, such as the total volume, the volume of errors, the average time, or the timeouts. We also had to create alerts based on this monitoring, whose purpose is to send a webhook to a website every time an anomaly is detected or a desired condition is met. On the other hand, I have also contributed to the development of this website, which aims to allow users to quickly view key aspects of the health of the services and notify them by SMS or email when the anomalies are generated.

The monitoring part has been done using an ELK cluster (acronym for Elasticsearch, Logstash and Kibana), technologies that allow us to search and analyze our logs easily (Elasticsearch), ingest data from different sources simultaneously, transform it and send it to a warehouse (Logstash), visualize and manage the cluster from a GUI (Kibana). Once we have the data in indexs, we can create machine learning jobs, which will be fed of selected and transformed data from these indexs, and will learn to detect and generate anomalies using, among others, unsupervised learning. In addition, ELK allows the creation of watches, which are scheduled querys which verify if their result meets a condition, and if so, perform an action (in our case send a webhook).

On the other hand, the website is being developed using Spring for the backend and Angular for the frontend, as well as an Oracle database. This web renovates and unifies the two websites that were used in this project before we began to use ML jobs of ELK: one web of monitoring, which created graphs of the data of ELK (obtained using its API) and associate them with a service; the other one was an administration website, which allowed grouping users, grouping services, linking groups of users with a service, creating rules that a service must meet, notify users associated to the service in case that a rule is not meeting, etc.

Resumen

En proyectos medianamente grandes, la monitorización y organización de servicios comienza a ser importante, puesto que ayuda a la escalabidad del proyecto, a la detección de fallos y a la toma de medidas al respecto con rapidez. Mi trabajo ha consistido, globalmente, en desarrollar un proyecto de monitorización. Esto implica, por una parte, monitorizar diversos aspectos, tales como la volumetría, el volumen de errores, el tiempo medio, o los *timeouts* de los *logs* que se generan y envían en las llamadas de los servicios que mi empresa ofrece al cliente; y crear alertas en base a esta monitorización, cuyo fin es generar un *webhook* a una web de administración cada vez que se detecte una anomalía o no se cumpla una condición deseada. Por otra parte, también he contribuido al desarrollo de dicha de web de administración, cuyo fin es permitir a los usuarios visualizar de forma rápida aspectos claves de la salud de los servicios y notificarles vía SMS o email cuando se generen las anomalías comentadas anteriormente.

La parte de monitorización se ha llevado a cabo utilizando un *clúster* de ELK (acrónimo de Elasticsearch, Logstash y Kibana), tecnologías que permiten buscar y analizar fácilmente (Elasticsearch), ingestar datos de múltiples fuentes de forma paralela, transformarlos y enviarlos a un un almacén (Logstash), y manejar el clúster desde una interfaz gráfica (Kibana). Una vez tenemos los datos en *índices*, podemos crear *jobs* de machine learning, los cuales se alimentarán de la selección y transformación de datos obtenidos de dichos *índices*, e irán aprendiendo a detectar y generar anomalías mediante, entre otros, *unsupervised learning*. Además, ELK permite la creación de *watches*, que son querys programadas para ser lanzadas cada cierto tiempo que verifican que su resultado cumpla una condición, y de ser así realizan una acción, en nuestro caso lanzar un *webhook*.

Por otro lado, la web de administración se está desarrollando usando Spring para el *backend* y Angular para el *frontend*, así como una base de datos de Oracle. Esta web renueva y unifica dos webs que se usaban en este proyecto antes de empezar a usar los *jobs* de ML de ELK: una de monitorización, la cual permite crear gráficas a partir de los datos en ELK mediante su API y asociarlas con un servicio; y otra de administración, que permite agrupar usuarios, agrupar servicios, relacionar agrupaciones de usuarios con un servicio, crear reglas que un servicio debe cumplir, las cuales, al recibir la alerta del *webhook* notifican a los usuarios asociados a dicho servicio que no se están cumpliendo, etc.

Resum

En projectes mitjanament grans, el monitoratge i organització de serveis comença a ser important, donat que ajuda a l'escalabilitat del

projecte i contribueix a la detecció d'errades i a prendre mesures al respecte amb rapidesa. El meu treball ha consistit en, globalment, desenvolupar un projecte de monitoratge. Això implica, per una banda, monitorar diversos aspectes, tals com el volum total, el volum d'errors, el temps mitjà o els *timeouts* dels *logs* que es generen i envien a les crides a serveis que la meva empresa ofereix al client, a més de crear alertes basades en aquest monitoratge, les quals tenen la finalitat de generar un *webhook* a una web d'administració cada cop que es detecti una anomalia o no es compleixi alguna condició desitjada pel client. Per altra banda, també he contribuït al desenvolupament de la web d'administració esmentada, que té com a finalitat permetre als seus usuaris visualitzar de forma ràpida els aspectes claus de la salut dels serveis i notificar-los via SMS o email quan es generin les anomalies esmentades anteriorment.

La part de monitoratge s'ha dut a terme fent ús d'un *clúster* de ELK (acrònim de Elasticsearch, Logstash i Kibana), tecnologies que permeten buscar i analitzar fàcilment (Elasticsearch), ingerir dades de múltiples fonts de forma paral·lela, transformar-les i enviar-les a un magatzem (Logstash) i finalment visualitzar-les (Kibana). Un cop tenim les dades a *índexs*, podem crear *jobs* de machine learning, els quals s'alimentaran de la selecció i transformació de dades obtingudes dels esmentat *índexs*, i aniran aprenent a detectar i generar anomalíes mitjançant *unsupervised learning*. A més a més, ELK permet la creació de *watches*, que són *querys* que es programen per ser llençades cada cert temps, verifiquen que el ser resultat compleixi una condició, i si ho fa, realitza una acció determinada, en el nostre cas llençar un *webhook*.

Per altra banda, la web d'administració s'està desenvolupant utilitzant Spring pel *backend* i Angular pel *frontend*, així com una base de dades d'Oracle. Aquesta web renova i unifica dues webs que s'utilitzaven en aquest projecte abans de començar a utilitzar els *jobs* de ML de ELK: una de monitoratge, la qual permet obtenir gràfiques de les dades en ELK mitjançant la seva API i associar-les amb un servei; i l'altra d'administració, que permet agrupar usuaris, agrupar serveis, relacionar agrupacions d'usuaris amb un servei, crear regles que un servei ha de complir, les quals, en rebre una alerta del *webhook* notifiquen als usuaris asociats al servei esmentat que no s'estàn complint, etc.

Índice

Lista de figuras

Lista de tablas

Introducción

1.1 Contexto

Este proyecto forma parte de una serie de servicios que mi empresa ofrece a sus clientes, y no es un trabajo de final de grado hecho por cuenta propia. La empresa en la que he trabajado durante el desarrollo de este proyecto, de ahora en adelante "mi Empresa", es una consultora tecnológica de gran tamaño, que presta servicios de Consultoría, Tecnología, Subcontratación y Servicios Profesionales Locales.

Este proyecto de monitorización se encuentra dentro de varios proyectos de carácter tecnológico que mi Empresa ofrece a una entidad financiera, de ahora en adelante "el Cliente".

El proyecto global, que agrupa tanto el de monitorización de servicios como otros, trata de prestar todos los servicios que necesita el Cliente para ofrecer una serie de tarjetas prepago compatibles con Visa.

1.2 Motivaciones

Empecé haciendo prácticas en mi Empresa en Julio de 2018, con la intención de obtener un primer contacto con el mundo laboral y convalidar una asignatura de prácticas, en vistas a tener 30 créditos restantes para mi graduación en en-

ero de 2019, y acabar mi carrera haciendo esos 30 créditos en el extranjero de Erasmus. Acabé las prácticas en diciembre de 2018, con un buen *feedback* de la empresa, y habiéndome llevado una buena experiencia en ella. Tras una serie de complicaciones en el trámite del Erasmus que empezaron en septiembre de 2018 y fueron desgastando mis ganas de Erasmus hasta enero de 2019, decidí renunciar a la plaza del mismo y acabar mi carrera en Barcelona. Sabiendo que la empresa quería que volviese, y ofreciéndome trabajar en un proyecto que me parecía interesante puesto que me serviría también para ganar experiencia laboral y conocimiento más profundo en diversas áreas de la informática que tocaba el proyecto, decidí hacer una segunda asignatura de prácticas y el TFG en la empresa en este semestre de primavera de 2019, quedándome así solo una optativa para acabar mi carrera, la cual podría compaginar fácilmente con el proyecto en mi Empresa.

1.3 Metodología

En el proyecto global trabajamos más de 10 personas, habiendo un lider de proyecto, y varios desarrolladores experimentados que gestionan al resto del equipo dependiendo del proyecto o rama en la que estén. En cuanto al proyecto de monitorización éramos 2 personas hasta mediados de Abril, cuando se unió un tercero. Cristian, el compañero que me guiaba al principio y el responsable del proyecto de monitorización, también era el que me asignaba tareas. Las tareas tenían unas fechas de inicio y de fin y unas horas teóricas definidas. Para distribuirlas y ver el estado de ellas usamos una pizarra de Kanban. También hacemos meetings 3 veces por semanas con todos los integrantes del proyecto para ver cómo van las tareas, resolver dudas, avisar de cualquier imprevisto, etc.

1.4 Estructura del *backend*

El backend del proyecto global se divide en 4 principales bloques:

- SN: Es el nivel más bajo del backend, el cual hace las conexiones con la BBDD, valida la existencia de datos, y conecta con servicios de terceros que también trabajan con el Cliente. Un ejemplo sería validar la existencia de un DNI.

- SI: Es la capa de abstracción que se utiliza para operar con los SN. Los SI preparan los datos para llamar a los SN, además de comprobar los outputs de los SN y tomar una u otra decisión. Siguiendo con el ejemplo anterior, el SI 'registrar usuario' llamaría, entre otros al SN 'validar DNI'.

- SE: Es la capa del backend a la que llaman terceros, ya sea el Cliente u otras entidades que necesiten de nuestro servicio. Esta capa acaba llamando a los SI.

- Canal: Es el canal de entrada de datos, por ejemplo una web interna, una tablet o un cajero. Si un cliente quiere dar de alta una tarjeta a través de la web, esa web será el Canal, y llamará al SI que necesite.

Los distintos niveles del backend se comunican mediante objetos que fomatean el input y el output de dicho nivel. Por ejemplo, al crear un SE 'X' , se crean también objetos que indican la forma de su input, 'X.input', y su output, 'X.output'. Este SE puede llamar a un SI 'Y', del cual obtiene su 'Y.input' antes de llamarlo, y lo rellena con los datos pertinentes, sabiendo que los datos que recibirá tendrán la forma de 'Y.output'. De esta forma, el SI obtiene los parámetros necesarios para operar, y éste hará lo mismo con los SN a los que le toque llamar.

En cada ejecución de un servicio se escriben y guardan logs de dicha ejecución para tener un control constante del flujo de datos y así facilitar la detección de

fallos. Estos logs son los que acaban en Elasticsearch y sirven como fuente de datos para la monitorización.

1.5 Relación del backend, Elasticsearch y la Web

Como acabo de comentar, los logs generados en el backend acaban en Elasticsearch, y sirven como fuente de datos para la monitorización. Estos logs alimentan los *jobs* de machine learning del clúster de Elasticsearch, los cuales generan anomalías cuando el compartamiento de los datos es suficientemente distinto al esperado.

La web de monitorización tiene 2 partes: la parte de gestión y la parte de visualización.

La parte de gestión permite modificar los permisos de la visualización de la monitorización, así como la organización de los usuarios y jobs. Cada job del clúster es asociado con lo que llamamos una 'operativa'. Estas operativas pueden formar parte de grupos y de dimensiones puesto que así el cliente se organiza mejor. Los usuarios pueden pertenecer a múltiples grupos (los asociados con operativas); de esta manera, controlamos el acceso a las operativas según el usuario. Además, cada operativa puede tener una o más reglas, las cuales son asociadas con dos watches del clúster, y pueden tener asociada una lista de usuarios y un mensaje para alertar cuando se incumplan. Por último, también se guardan y listan las anomalías que generan los jobs de Elasticsearch en la web, y estas, siguiendo un análisis aún en desarrollo generan o alimentan incidencias, las cuales también sirven para alertar a los usuarios de que algo no está bien.

Por otro lado, la parte de visualización es a la que también tienen acceso los usuarios no administradores, y permite ver la monitorización (volumen, volumen

de error, volumen de timeouts y tiempo medio) de las operativas dentro del grupo del usuario, filtrar las operativas por grupo o dimensión, buscar una operativa, ver las reglas vigentes asociadas a la operativa, y ver un historial de anomalías e incidencias así como su estado y comentarios o archivos adjuntos que otros usuarios pueden haber hecho intentando solventar el problema.

Objetivos

El principal objetivo de este proyecto es monitorizar en 'tiempo real' los servicios que el Cliente desea, así como alertar a una serie de personas cuando un servicio dado falle o no cumpla una regla impuesta. Además, también se pretende desarrollar una página web que se encargue de la visualización de la monitorización de los servicios en detalle, la gestión de usuarios, grupos de usuarios, operativas, reglas que una operativa/servicio debe cumplir, notificaciones, etc.

Personalmente, mi objetivo es aprender lo máximo posible, tanto a nivel técnico en desarrollo web, gestión de bases de datos o uso de tecnologías punteras como Elasticsearch o Angular; como en el ámbito de gestión y negocio de las empresas.

Planificación

Se ha separado en 2 bloques el proyecto en general: la parte de ELK que se resume en la creación de jobs y watches de Elasticsearch, y la parte del desarrollo de la nueva web, la cual a su mismo tiempo se podría subdividir en home/detalle de operativa, gestión de usuarios y grupos, gestión de dimensiones y operativas (funcionalidades que ya se cumplían en las 2 webs antigüas de monitorización), y en la obtención y clasificación de anomalías desde ELK (parte nueva y aún en desarrollo que consiste en clasificar las anomalías que generan los jobs como incidencia o no incidencia).

En el transcurso de este proyecto he tocado, en más o menos medida, todas las tareas que se definen en el diagrama de Gantt [figura 3.1], excepto la parte de configuración/conectividad de ELK.

3.1 Análisis de tecnologías usadas

En este proyecto hemos utilizado principalmente ELK, Angular y Spring, además de software interno para el despliegue y conectividad con servicios de 3ros, y Figma para el diseño de la interfaz.

Se ha optado por ELK, y no por una base de datos relacional convencional por distintos motivos: es rápido; distribuido; escalable; RESTful; open source; permite el análisis de los datos; permite una alta personalización de las búsquedas; tiene una gran comunidad detrás, tanto en foros como en github creando dis-

tintas versiones y software basado en ELK; incluye muchas herramientas útiles como la visualización de datos, gráficas, herramientas de desarrollo para llamar a su API, etc.;y algunas esenciales para el proyecto, como las de X-Pack para usar machine learning para la generación de anomalías basada en los datos guardados en Elasticsearch, o generar alarmas cuando un evento suceda de forma eficaz. Existen muchas alternativas, como por ejemplo usar Apache Solr como motor de búsqueda y desarollar nosotros los algoritmos de machine learning, y un motor de gráficas para las visualizaciones, pero esto supondría un proyecto mucho más complejo, con más trabajadores y dificultades para la escalabilidad.

Se escogió Angular y Spring debido a que la mayoría de trabajadores de mi equipo que hacen web tienen experiencia con estos framework; también porque son de los frameworks más versatiles para el desarollo Web, reusables, amigables con los test unitarios, eficientes, permite el uso de *Material Design* y tiene soporte de Google en el caso de Angular, además de implementar las funciones CRUD (*create, read, update and delete*) fácilmente en el caso de Spring. Otras alternativas podrían ser Django, React o Flask, aunque se prefirió Angular y Spring por todo lo mencionado anteriormente.

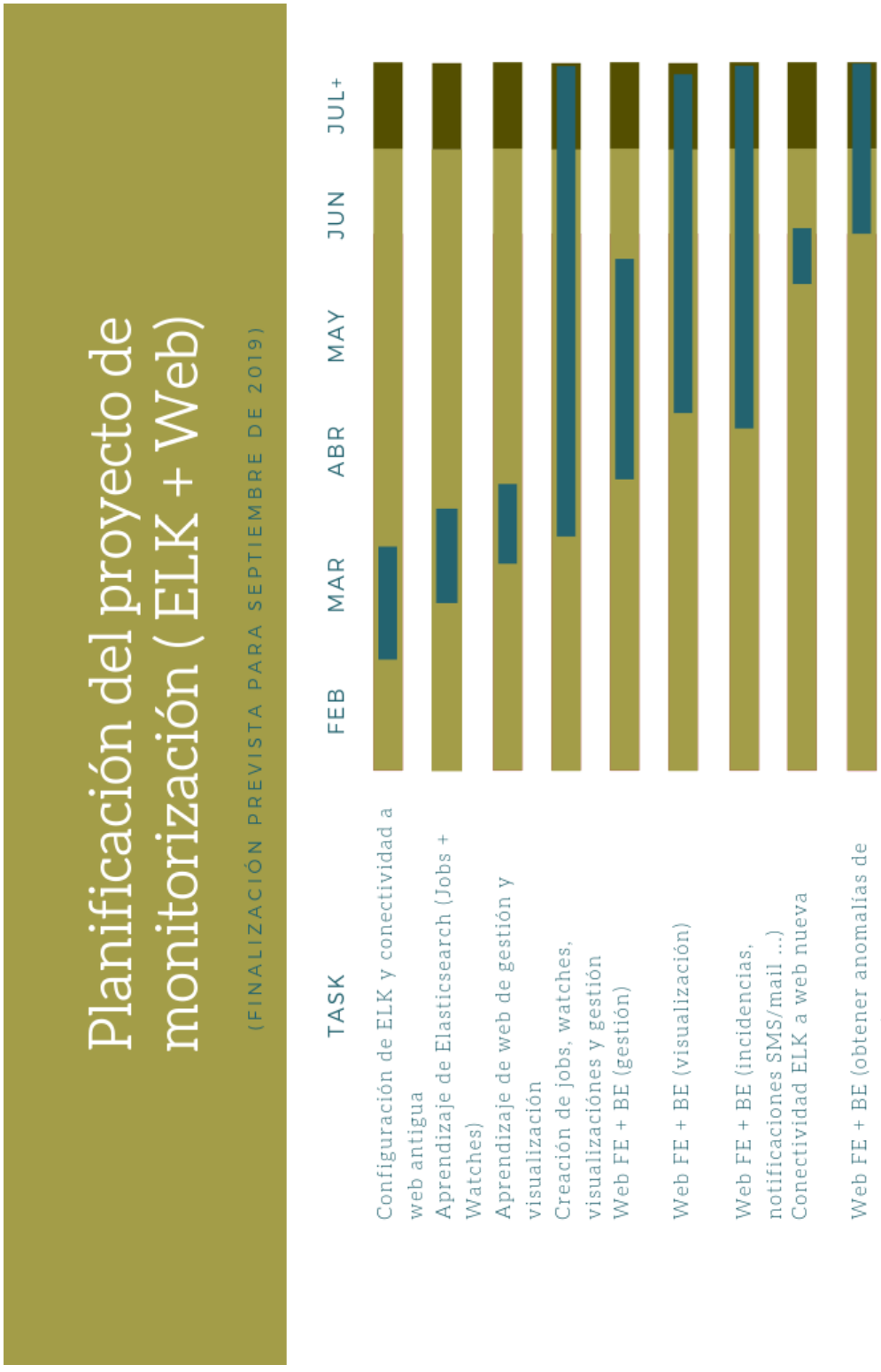

Figure 3.1: Diagrama de Gantt con la planificación del proyecto

Costes

Mi Empresa, a nivel de proveedor, estimó el coste de este proyecto en aproximadamente 2200 horas, lo que sumaría, a un coste de 15€/h, un total de 33,000€. No obstante, también hay que contar las licencias de X-Pack de Elasticsearch, que en el caso de este proyecto es la Platinum [referencia [1]] y contamos con un clúster de 10 nodos, precio del cual variará dependiendo de la empresa y el uso del mismo, pero se podría estimar en (mínimo) 50,000€/año. Además habría que pagar el ordenador a, en este caso, 3 desarrolladores (coste orientativo de 2,000€) y el alquiler de las oficinas durante al menos 7 meses (coste orientativo de 10,000€).

Por lo tanto, sin incluir el servicio de 3ros ni el coste de todo el backend (debido a que lleva años y no es pertinente a este proyecto), una cota mínima del coste de este proyecto sería 45,000€(desarrollo) + 50,000€/año (X-Pack Platinum) + mantenimiento/modificaciones futuras.

Elasticsearch

¿Qué es Elasticsearch?

Elasticsearch es un motor de análisis y búsqueda distribuido y RESTful construido sobre Apache Lucene [referencia [2]], que está publicado como código abierto bajo las condiciones de la licencia Apache. Es el *core* del clúster de ELK, ya que es donde se almacenan los datos del clúster, y donde se realizan todas las operaciones y análisis del mismo. Sus principales características son:

- Rapidez: Elasticsearch utiliza varias estrategias para generar y guardar sus *índices* de manera que la búsquedas se puedan realizar de forma eficiente.
  - Sus índices son *inverted indexes*, es decir, se generan mapeando los términos a documentos que contengan el término. Dichos términos se guardan en orden, haciendo que las búsquedas de términos sean muy rápidas, y por ende también encontrar el documento al que pertenecen [figura 5.1].
  - Utiliza distintos algoritmos dependiendo de la query que hagamos. Viendo los *inverted indexes* que usa, es bastante obvio que buscar un término concreto o términos que empiecen por el prefijo X se consigue en tiempo récord. Para otros casos, como buscar términos que acaben en X, se revierten los términos y se buscan aquellos que empiecen por el prefijo X. También hashea los datos geográficos a términos haciendo fácil su búsqueda. Se generan términos al guardar datos numéricos para

acotar el rango de búsqueda de un número (por ejemplo, el número 123 se puede guardar como “1”-hundreds, “12”-tens y “123”, y buscar en el rango [100,199] es buscar las coincidencias con los términos “1”-hundreds, que hace más eficiente la búsqueda en comparación con buscar todos los número que empiecen por 1), etc.

Figure 5.1: Inverted Index

- Flexibilidad: Elasticsearch soporta datos de todo tipo: números, texto, geográficos, fechas, estructurados y no estructurados. Además Elasticsearch ofrece API’s que permiten comunicarse con el clúster en Java, Python, .NET, PHP o C# entre otros, dando más cabida aún a su uso.

- Escalabilidad: El escalado horizontal de Elasticsearch permite manejar millones de eventos por segundos mientras que maneja cómo los índices y querys se distribuyen por el cluster. Además el hecho que te comuniques con la interfaz de la misma forma operando en 1 solo ordenador que en un clúster de n ordenadores hace más atractivo este software para usuarios que buscan

una solución escalable y fácil de usar.

Por estas características y muchas otras, la popularidad de ELK ha aumentado drásticamente en los últimos años, convirtiéndose en menos de una década en el motor de búsqueda empresarial más popular según DB-Engines [referencia [3]].

5.1 Estructura

Un clúster de Elasticsearch puede contener múltiples *indexes*, que a su vez contienen múltiples *types*. Estos *types* guardan multiples *documents*, y cada *document* tiene múltiples *fields*. Elasticsearch está orientado a los documentos, lo que significa que almacena documentos enteros, entendiendo documento como la raiz de un JSON que puede tener números, listas, geo localizaciones, fechas u otros objetos JSON dentro.

Los índices son un *namespace lógico* que apunta a uno o más *shards*. Las shards són motores de búsqueda en sí (instancias de Lucene) que mantienen una parte de los datos de un índice. Podemos imaginar los shards como contenedores de datos que Elasticsearch distribuye a través del clúster. Cuando el cúster crece o decrece, Elasticsearch migrará shards entre nodos para mantener el clúster balanceado. Las shards pueden ser primarias (definidas en la creación del índice) o replicas (las cuales pueden ser creadas dinámicamente). Cada documento pertenece a una shard primaria, mientras que las replicas son copias redundantes que protegen nuestros datos contra fallos de hardware, y atienden a llamadas de lecturas, como búsquedas.

En la práctica, el clúster con el que he trabajado tenía múltiples índices, aunque los más usados eran 3 y se caracterizaban por su procedencia y porque solían contener logs distintos (por lo general uno contenía logs de los SI, otro de los SE, y

otro de lo demás).

5.1.1 Símil y diferencias con bases de datos SQL

A pesar de que el propósito final sea el mismo, la terminología es distinta. Aún así podemos hacer un *mapping* fácilmente:

SQL	Elasticsearch	Descipción
columna	campo	En última instancia se guardan datos de un cierto tipo en una *entry*, llamada columna en SQL, y campo en Elasticsearch. En Elasticsearch se pueden guardar múltiples valores del mismo tipo en un campo, esencialmente listas, mientras que en SQL sólo se puede guardar un valor.
fila	documento	Las columnas y filas no existen por si solas, si no que forman parte de una fila o documento. Se diferencian en que un documento tiende a ser un poco más flexible a pesar de que sigue teniendo una estructura.
tabla	índice	Objetivo a quien se hacen las consultas.
esquema	*implícito*	En los RDBMS, los esquemas suele ser un *namespace* de tablas usado como segunda restricción, el cual no tiene equivalente en Elasticsearch. Aún así, si activamos la seguridad en elasticsearch, un rol sólo vera datos de los cuales tenga permiso (lo que se considera esquema en jerga de SQL).

base de datos	clúster	Set de esquemas; conjunto de tablas o índices agrupados. Se diferencia en que mientras en SQL es otro *namespace*, en Elasticsearch es una instancia de Elasticsearch en ejecución, por lo que potencialmente SQL puede tener varias bases de datos en una instancia, mientras que Elasticsearch solo puede tener una instancia al mismo tiempo.

Table 5.1: Equivalencias entre SQL y Elasticsearch

5.2 Búsqueda

Las búsquedas se realizan llamando a la API, a la cual enviamos parámetros a través de la URI o a través del *request body*. Esta segunda opción nos permite definir las búsquedas en un formato JSON de lectura más fácil. La API REST para búsquedas es accesible desde el *endpoint* "_search", siendo el ejemplo más sencillo "GET /{índice}/_search?q={parámetros}", o de una forma más legible, "GET /{índice}/_search { 'query' : { ... }, ... }".

Cada búsqueda nos retorna un JSON con *metadata*, como por ejemplo el tiempo que ha tardado, los *hits* de la búsqueda, las shards a las que ha buscado, así como las que han respondido de forma exitosa o fallida, el tiempo que ha tardado, un *score* que nos indica cuán preciso ha sido el resultado dependiendo de la búsqueda, y finalmente la lista de elementos de la búsqueda en si.

5.2.1 Tipos de búsqueda

Existen distintos tipos de *querys* o búsquedas que podemos usar en función de los datos que usamos y del fin de la búsqueda. Dichas querys aceptan parámetros

para mayor o menor tolerancia, o usar una u otra operación a la hora de hacer la búsqueda. En este apartado comentaré las querys simples que he usado más en el proyecto:

- Match: Esta query es la más básica cuando se quiere buscar por campo. Podemos pasarle un string, un número o una frase. En este último caso, devolverá documentos que contentan una o más palabra de la frase en el campo indicado, con mayor score mientras más palabras contenga.

- Match_phrase: Como match, pero busca frases exactas o palabras con cierto grado de libertad.

- Term: Retorna los documentos que contentan un termino exacto en el campo indicado. Se diferencia de match, en que match hace querys internas y podría encontrar "foo" si buscamos "FOO", mientras que term sólo buscaria "FOO".

- Range: Devuelven los documentos con campos que tengan términos dentro de un rango concreto. Se suele usar para filtrar por campos numéricos o por fechas.

- Bool: Query que permite combinar múltiples querys "hojas" mediante cláusulas "must", "should", "must not" o "filter".

Gracias a estas y varios tipos de querys más, podemos extraer los datos que queremos de los índices de Elasticsearch.

El uso de querys es básico a la hora de operar con las distintas herramientas que nos proporciona Elasticsearch. Se utiliza para crear *jobs* de machine learning (que ingestan datos a partir de querys), para crear *watches* que pueden utilizar querys para comprobar que un índice cumple una condición, y en general para

recuperar datos que nos interesen para el propósito que nos haga falta.

5.3 Agregaciones

Las agragaciones proporcionan la habilidad de agrupar y extraer estadísticas de nuestros datos, de forma similar al "GROUP BY" y "AGGREGATE" de SQL. Elasticsearch permite ejecutar querys, que además de retornarnos el listado de hits, agreguen otros resultados en una sola respuesta. En estas agregaciones hacemos referencia a los datos que obtenemos de la query, pudiendo así calcular, por ejemplo, la media de un campo "edad" que obtenemos de la query. Además, se permiten anidar agregaciones dentro de otras agregaciones, permitiendo por ejemplo, agrupar los resultados por país, y calcular la media de ingresos de cada país.

Para crear agregaciones basta con poner un nombre al campo agregado, y usar una query que haga referencia a un campo que nos retorne la búsqueda.

En este proyecto he usado agregaciones para calcular los campos que queremos monitorizar: volumen de errores (se contabilizan los distintos resultados del campo "RESULT", y se suman los que no sean correctos), tiempo medio (se hace la media del campo "TIME" ya sea directamente o dividiéndola por el factor oportuno para obtenerla en milisegundos), *timeouts* (se filtran los documentos cuyo campo "TIME" sea mayor que X), errores por volumen (dividiendo el volumen de errores entre el total), y timeout por volumen (dividiendo el volumen de timeouts entre el total).

5.4 X-Pack

X-Pack es una extensión de Elasticsearch que proporciona herramientas de seguridad, alertas, monitorización y *machine learning* entre otras. X-Pack viene con la instalación de Elasticsearch, aunque esta extensión es de pago, y sólo se puede probar de manera gratuita durante 30 días.

En este proyecto, el Cliente, dueño del clúster, tenía la subscripción completa, puesto que usamos el varias funcionalidades de X-Pack. De hecho, la monitorización, generación de anomalías (auto-generadas gracias al módulo de machine learning de X-Pack), y generación de alertas (generadas por watches) se realizan usando herramientas de X-Pack.

Jobs de Machine Learning en Elasticsearch

¿Qué son?

Como introduje en el anterior punto, X-Pack, una extensión de pago de Elasticsearch, añade diversos servicios para el clúster, incluyendo un módulo de *machine learning*. En concreto, este módulo proporciona detecciones de anomalías en series temporales, modelando el comportamiento normal de nuestro datos (aprendiendo tendencias, periodicidad, y más) en tiempo real para identificarlas.

Cada *job* de ML contiene la configuración y los metadatos necesarios para realizar una tarea de análisis. Todos los jobs necesitan un intervalo de tiempo para "resumir" y "modelar" los datos llamado *bucket span*, y uno o más *detectors*, los cuales aplican una función analítica a un campo de nuestros datos, cuyo resultado puede ser visto en gráficas para hacernos una idea del estado de nuestros datos.

Se pueden configurar propiedades del job que afectan en la elección de entidades que son consideradas anómalas. Por ejemplo podemos especificar si queremos que las entidades sean analizadas viendo su comportamiento anterior, o relativo a otras entidades de su población. También podemos dividir los datos en categorías o particiones, para, por ejemplo monitorizar el volumen de datos un job de 'tráfico de datos' por un campo 'IP'.

También se puede de manera opcional agrupar jobs. De esta forma podemos

ver el resultadao de múltiples jobs de manera sencilla.

Tanto los jobs sueltos como agrupados se pueden suscribir a un calendario de fechas (con precisión de hasta segundos) en las cuales no generamos anomalías. Esto sirve para evitar falsos positivos o negativos en periodos de tiempos anormales, como por ejemplo el "Black Friday" si monitorizamos números de ventas.

6.1 Estructura

Un job es un JSON que contiene su configuración y metadatos. Sus principales campos son:

- analysis_config: Es un objeto JSON que contiene la configuración de análisis. Sus puntos claves son:
  - bucket_span: Intervalo de tiempo en que se recopila información para analizar. Si por ejemplo queremos comparar la media de resultados de forma horaria, pondremos este valor a 1h.
  - detectors: Array de objetos detectores. Dichos objetos contienen, entre otros, una descripción, una función analítica que determinará cuándo queremos que se detecten anomalías (por ejemplo usaremos las funciones *count, mean, min, low_count, sum etc.* para generar anomalías cuando el volumen, media, mínimo, volumen sólo si es menor de lo habitual o suma son atípicas) y el campo al cual se aplicará dicha función. También se le pueden añadir reglas que permiten personalizar cómo trabaja el detector (por ejemplo, omitir la generación de anomalía cuando el resultado actual sólo difiera en X del típico), o partir los datos por un campo.

  - influencers: Array de strings (nombre de campos) que creemos que influyen particularmente en el resultado.

- analysis_limits: Especifica los límites de los recursos para el job.
- job_id: Identificador.

Una vez especificado el job e inciada la ingesta de datos del mismo, el job usará los algoritmos internos de machine learning de Elasticsearch para aprender y modelar el comportamiento normal de nuestros datos según la configuración que le hemos dado, y así generar anomalías cuando las considere necesarias.

6.2 Fuente de datos

Los jobs de machine learning pueden analizar datos almacenados en Elasticsearch o datos enviados desde otra fuente mediante su API. Los *datafeeds* recuperan datos de Elasticsearch para su análisis, y es la manera más simple y común de enviar datos a los jobs.

Los datos de otra fuente se pueden enviar al job mediante batches usando la API 'POST _ml/anomaly_detectors/<job_id>/_data'; pasando documentos JSON que contengan los datos a analizar.

Puesto que en este proyecto he pasado datos a todos los jobs realizados usando datafeeds, explicaré su estructura principal.

- indices: Array de índices de los que se obtendrán los datos.
- job_id: Nombre del job asociado al datafeed.
- query: Query que se lanzará a los índices indicados para obtener los datos que usará el job para su análisis.

- aggregations: Agregaciones que podemos hacer sobre los datos de la query, las cuales también estarán disponibles para ser analizadas por el job.
- query_delay: Margen de tiempo tras el cual se lanza la query. Si los datos que se ingestan a las 10:00 a.m. no están disponibles hasta las 10:05 a.m., se debería poner este campo a 5m.
- frequency: Intervalo en el cual las querys programadas se lanzan mientras el datafeed corre en tiempo real. Normalmente suele ser el bucket_span, o una fracción del mismo en casos en que sea muy grande.
- delayed_data_check_config: Este campo indica si el datafeed debe verificar que no se hayan perdido datos, y la ventana de tiempo en la cual verifica. Sirve para los casos en que la ingesta de datos es lenta y en el momento de lanzar una query aún no están todos los datos que deberían estar.

Cuando creamos un job avanzado desde kibana, el datafeed viene dentro del JSON del job, y es un JSON con clave 'datafeed_config', permitiéndonos modificarlo para personalizar los datos del job como queramos. También existe la opción de crear un job sencillo, y aunque te autogenera el JSON del job y del datafeed, te limita bastante las opciones, ya que no puedes hacer agregaciones y varios campos de configuración vienen por defecto.

6.3 Anomalías

Cuando hay una o varias ocurrencias raras dentro de nuestros datos analizados en un job, Elasticsearch genera anomalías y les pone una puntuación. Existen 3 tipos de anomalías distintas: por registro (o documento, es decir una anomalía individual), por *influencer* (cuando una entidad contribuye o es culpada de la

generación de anomalías, por ejemplo un usuario o una IP) y por *bucket* (anomalías generadas en una ventana de tiempo).

Las anomalías por registro se puntúan basandose especificamente en una instancia de algo que ha ocurrido, por ejemplo, el valor de una respuesta ha pasado a ser 300% más alto de lo habitual, o el número de respuestas procesadas hoy es mucho menor que el del mismo día de la semana pasada. Cuando se detecta una anomalía, Elasticsearch calcula la probabilidad de la misma basada en un modelo construído a partir del comportamiento pasado de nustros datos. Una vez calculada esta probabilidad, ML la normaliza para que en Kibana podamos ver una puntuación de 0-100.

Todas las anomalías se guardan en un índice llamado '.ml-anomalies-*', permitiendo así su búsqueda desde jobs, watches o un programa externo mediante la API de Elasticsearch, haciendo posible su análisis. Gracias a ello, desde la web que estamos creando, podemos obtener las anomalías que se generan en un job, analizarlas, y en un futuro crear incidencias según veamos conveniente.

6.4 Jobs realizados y ejemplos

Durante este proyecto he creado más de 50 jobs que monitorizan distintas operaciones realizadas por los 4 bloques del backend (SN, SI, SE y CA).

La gran mayoría de los jobs siguen el mismo patrón: en el apartado **'analysis_config'** defino los 6 detectores mencionados: volumetría (con función 'low_count ya que nos interesa generar anomalías sólo cuando se cuente menos registros de lo normal), errores (con función 'high_mean' del campo 'mean_time', puesto que nos interesa generar anomalías sólo cuando el tiempo medio sea mayor al normal), tiempo medio ('high_mean(total_errors)'), timeouts ('high_mean(timeouts_script)'),

y errores/timeouts por volumen (también usando high_mean). Además suele estar puesto como influencer el campo 'RESULT', ya que condiciona a la generación de anomalías.

Todos los campos que hace referencia el job están descritos en el **'datafeed_config'**.

En él defino el índice del cual obtendremos los datos, la frecuencia, la query (que suele ser un match de un método del campo 'METHODNAME', el cual normalmente viene dado como [CA|SE|SI|SI].package.nombreMétodo), y las agregaciones, en las cuales calculo, mediante filtros y scripts, todos los campos que utilizan los detectors del job excepto la volumetría.

Watches

¿Qué son?

Los *watches* son una herramienta de X-Pack que permite estar pendientes de cambios o anomalías en nuestros datos y realizar una acción como respuesta. Algunos ejemplos de usos podrían ser *trackear* los tiempos de respuesta, y si exceden los SLAs durante 5min poner un ticket de *helpdesk*; enviar un email cuando el espacio libre de una infraestructura sea menor que X, o enviar una notificación cuando detectemos que un nodo se cae o una query excede un rango esperado.

X-Pack proporciona un API para crear, testear y gestionar watches. Cada watch describe una alerta y puede contener múltiples acciones de notificación. Además, Elasticsearch mantiene un historial de todos los watches en un índice, donde graba si se ha cumplido la condición, las acciones que se tomaron y los resultados de las querys de los watches cada vez que se activan.

7.1 Estructura

Los watches vienen definidos en un JSON con 4 bloques principales:

- trigger: Se encarga de programar la periodicidad en la que se activa el watcher. Puede ser definida de forma diaria, horaria, semanal, mensual, o cron [referencia [4]] entre otras.

- input: Define la forma en que se obtendrán los datos que serán evaluados para tomar o no una acción. Este input puede ser simple (cargamos datos estáticos que definimos en el watch); una búsqueda (a la cual le pasamos los índices en que queremos buscar y la query que queremos hacer contra ellos); una request HTTP para obtener datos de otro clúster de Elasticsearch mediante su API o de un *web service* externo; o en cadena, usando uno o más inputs de los mencionados anteriormente.

- condition: La condición que se evalúa una vez se cargan los datos, y determina si es requerido actuar. La condición puede ser 'always', 'never', 'compare', 'array compare' o un *script*. Cuando se evalúa una condición se tiene acceso a todo el contexto y el payload de los datos. Además también se pueden añadir condiciones a nivel de acción.

- actions: Especifica las acciones que se ejecutan una vez se cumple condition. Las acciones tienen acceso al payload de los datos, pudiéndolo usar de la forma que deseemos. Las acciones pueden ser el envío de un email, el indexado de datos, enviar un mensaje por Slack, crear un evento en PagerDuty, o enviar un *webhook*.

7.2 Watches realizados y ejemplo

Se crean 2 watches por cada regla que el cliente quiera asignar a una operativa. Hay más de 200 watches en el clúster que saltan cuando, por ejemplo, no se cumplen un mínimo de operaciones en una operativa, o cuando se genera una anomalía con score mayor a X.

La mayoría de watches siguen esta estructura: obtenemos los datos en el input haciendo una query a nuestro clúster, comprobamos que estén en el rango de

tiempo y que hay más o menos hits que los especificados en **condition**, y si se cumple se envía un webhook al servidor, que avisará por mail/SMS a las personas que lo deseen y estén en el grupo al que pertenece dicha operativa.

Se crean 2 watcher por regla, uno de inicio y otro de fin con condiciones opuestas, ya que cuando uno cumple su condición se desactiva y activa el otro. Así alertamos cuando una regla se incumple y cuando vuelve a la 'normalidad'.

Web

Resumen

Antes de empezar a usar jobs de machine learning de Elasticsearch, ya se monitorizaban servicios. Para ello existían 2 webs, una de monitorización y otra de gestión.

En la web de monitorización se podía (y se puede) ver las gráficas que se obtienen de hacer querys sencillas a Elasticsearch. Cada gráfica va asociada a una operativa, y los umbrales de las regla de la operativa se dibujaban como una línea horizontal en la gráfica, ya que por lo general sólo se miraba que haya más de X resultados en un rango de tiempo concreto. Los administradores pueden crear visualizaciones sencillas indicando un índice y algunos aspectos de la query que se hace a Elasticsearch, y asociarla con una regla de un listado que se obtiene desde la web de gestión.

En la web de gestión, se crean, listan, modifican y borran diversas entidades relacionadas entre ellas. Por una parte, los usuarios se asocian con empresas, las cuales tienen varios grupos. Se puede asociar a cada usuario con uno o más grupos de las empresas a las que pertenece. Por otro lado, están las operativas, las cuales pertenecen a un grupo, y se asocian con una visualización de la web de monitorización (actualmente con un job). Estas operativas pueden tener reglas, las cuales a su vez tienen umbrales que se asocian con 2 watches (uno de inicio y uno de fin) que avisan cuando se incumplen para notificar por mail o SMS a los

usuarios que pertenezcan al grupo de la operativa de la regla que lo deseen.

En el proyecto de monitorización de mi empresa decidió renovar también la parte web, creando una nueva que unificase las mencionadas anteriormente. Esta nueva web, aún en desarrollo, permite a los usuarios hacer las mismas acciones que en las 2 anteriores, con un diseño más dinámico, y algunas mejoras como que las gráficas vienen generadas a través de los jobs y de una query (teniendo así las gráficas de volumetría, tiempo medio, errores y timeouts de cada operativa, en vez de tener que crear una regla por cada gráfica que queramos); las anomalías que se generen por machine learning crearán incidencias las cuales se podrán seguir en un apartado específico, generar un PDF con su detalle, comentar el estado entre distintos usuarios sobre la incidencia, etc. Por otro lado, la web de gestión también se integra en esta nueva web, siendo posible acceder a ella sólo si el usuario está marcado como administrador

8.1 Front end

Para el frontend usamos el framework Angular, el cual usa TypeScript como lenguaje de programación, y HTML/CSS para la parte visual. Este framework divide el código en componentes y los va cargando cuando son necesarios, haciendo sus webs rápidas y eficientes. Además decidimos usar también Angular Material, debido a la popularidad y sencillez de implementar componentes de *Material Design* con Angular, creando una web más bonita y dándo una mejor experiencia de usuario al Cliente.

En nuestro caso dividimos la web en 4 bloques principales:

- Home/Detalle Operativa: En la home se mostrarán las agrupaciones de operativas por dimensión. Estando en la dimensión X se mostrarán las

operativas (agrupadas) de dicha dimensión que tienen los usuarios, y un conjunto de 'otras', que al clickarlo cambiará de dimensión en busca de otras operativas. Una vez se consigue que hayan menos de 5 operativas, se muestran en un listado y se puede acceder a su detalle clickando en su nombre (éste listado también se muestra si buscamos las operativas por nombre en la barra de búsqueda). En el detalle se ven la gráficas de los principales detectores de los jobs (volumetría, tiempo medio, volumen de errores y volumen de timeouts) creadas con datos que vienen del job de dicha operativa Elasticsearch en un intervalo de tiempo que puede elegir el usuario, así como un breve resumen de su estado, una lista de reglas que tiene asignada dicha operativa con como su estado (listado que pueden ver todos pero solo modificar los usuarios admin) el cual cambia automáticamente según los datos que llegan del backend, y una lista de incidencias abiertas que tiene dicha operativa.

Esta sección del FE vendría a resumir la antigüa web de 'visualización'.

El FE de la home fue hecho por un compañero, mientras que el FE del detalle de la operativa lo hice yo. En ambos casos usamos HTML, CSS, Angular y componentes de Angular Material.

- Gestión de usuarios y grupos: Esta sección, sólo disponible para administradores, lista los usuarios creados (los cuales tendrán acceso a esta web) y permite su filatrado por nombre, apellido o ID. Al clickar en el ID de uno de ellos se muestran sus datos personales (nombre completo, id, mail, telefono), así como los grupos a los que pertenece, clasificados en 'Grupos', 'Empresas', 'Departamentos, 'Proveedores'. También permite administrar dichos grupos, listando los grupos totales (no solo a los que pertenece el usuario en cuestión), permitiendo asignar o desasignar al usuario a un grupo, borrar un grupo, y crear un grupo nuevo.

Mediante estos grupos, agrupamos a los usuarios (y las operativas), permitiendo filtrar el acceso de un grupo de usuarios a una serie de operativas concretas, y facilitando la administración para el Cliente. También cabe destacar que ya no existe el concepto de empresa en esta web, y el acceso de un grupo de usuarios a una serie de operativas se rige únicamente por el grupo.

Esta sección fue hecha totalmente por un compañero.

- Gestión de dimensiones y operativas: Esta sección, sólo disponible para administradores, permite listar, crear, editar y borrar Operativas. Dichas operativas tienen un job de machine learning de Elasticsearch previamente creado el cual no es posible editarlo. También se le puede asignar, desasignar (y editar) a cada operativa varios grupos y dimensiones. Los grupos y dimensiones asignados a una operativa deben ser creados previamente, y sirven para filtrar el acceso de un grupo de usuarios a una serie de operativas en el caso de los grupos, y para filtrar las operativas por 'agrupaciones' (las cuales se crean al asignar una dimensión a una operativa), i.e.: Tenemos un proyecto grande cuyo personal está dividido en distintos equipos. El proyecto clasifica las tareas por distintas Áreas que a su vez están divididas Departamentos. Cada grupo puede estar en distintas Áreas/Departamentos. En nuestra web, 'Área' y 'Departamento' serían dimensiones, mientras que los equipos serían grupos. De esta manera un usuario de un equipo podría ver la monitorización de las operativas con las que trabaja navegando por área o departamento desde la home.

  Esta parte del FE fue hecha en su totalidad por mi. Para su creación se usó HTML, CSS, Angular y varios componentes de Angular Material.

- Incidencias y Anomalías: En este apartado, nuevamente solo disponible por

administradores, se listan todas las incidencias (generadas por nosotros a partir de un análisis, aún en desarrollo, de las anomalías), así como todas las anomalías (generadas por los jobs) de todas las operativas que se monitorizan. Por una parte, se listan las anomalías que generan los jobs en un tabla junto con datos sobre las mismas (nombre del job, detector, etc.), las cuales pueden ser clickadas para acceder a ellas 'en detalle' y ver más información. Por otro lado, está la tabla de incidencias, las cuales són anomalías o grupos de anomalías que se consideran (mediante filtros estáticos y en un futuro mediante ML) lo suficientemente grave como para necesitar que se actúe en consecuencia con cierta rapidez. Esta tabla de incidencias también contiene datos como el estado (Abierta, En revisión, Finalizada) o la fecha de inicio. Al clickar en una nos vamos al detalle de la misma, en la cual hay más información, así como apartados para manejarla y ver cómo se actúa (comentarios, documentos adjuntos y un listado de personas a las que se le notifica).

Este apartado está hecho en su totalidad por un compañero, usando las tecnologías mencionadas anteriormente.

8.2 Back End

Para el backend hemos usado Spring, un framework para aplicaciones Java. La web ofrece una API REST, la cual se puede implementar fácilmente siguiendo los decoradores que ofrece Spring. Para todas las entidades de la base de datos usamos JpaRepository, una interfaz que implementa un repositorio CRUD y que facilita las acciones de creación, listado, modificación y borrado de entidades. La mayoría de los servicios se dividen en 3 capas:

- Controladores: Esta capa define los métodos a los que se pueden llamar.

Reciben llamadas de métodos HTTP (GET y POST) y se comunican con los Servicios para satisfacer la petición del cliente. Por ejemplo, dentro del endpoint /api/operativa, podemos tener un método que sea getById el cual devuelva una operativa mediante un GET al endpoint /api/operativa/id. Para devolver dicha operativa el controlador llamará al Servicio de operativas y hará las operaciones pertinentes de control.

- Servicios: Esta capa se comunica con los Repositorios y prepara los objetos y su contenido para pasárselo a los Controladores.

- Repositorios: Esta capa se comunica directamente con la base de datos. Un repositorio se define con una tupla (objeto, objeto identificador (long u otra clase)), y permite hacer las querys necesarias para obtener los datos que nos hagan falta. Viene con algunos métodos implementados, como getOne(id), pero también podemos definir una query 'personalizada' y pasarle parámetros para recuperar datos más complejos.

Por otra parte tenemos el modelo de datos que se relacionan con las entidades de la base de datos y tablas intermedias necesarias para su funcionamiento.

En cuanto a las tareas y al reparto de las mismas, por lo general son tareas CRUD y querys a Elasticsearch ya sea para obtener un job, su datafeed, sus anomalías, etc,; y nos la hemos repartido siguiendo casi por completo los 4 bloques del front end.

Como se puede ver en la planificacion [3.1], la web no está completa, y aún falta acabar tanto parte de front como de back end, así como perfilar cosas ya hechas o arreglar algún fallo.

Pruebas y resultados

Puesto que trabajamos para terceros (el Cliente), obtenemos feedback y hacemos cambios/añadimos nuevas características constantemente. En cuanto a los jobs, se han realizado más de 50, los cuales son revisados por el Cliente y comparados con su antiguo sistema de monitorización para ver que el resultado sea el mismo. Se han creado más de 150 watches, los cuales se prueban en el momento que salta (o no) una anomalía o un suceso digno de notificar; por lo general funcionan bien, y si falla algo el Cliente nos pide que lo cambiemos. En cuanto a la web, el front end está basado en una diseño hecha por mi Empresa y aprobado por el Cliente previamente, por lo que intentamos ceñirnos a él lo máximo posible ya que así habíamos acordado con el Cliente que quedaría. Si bien es cierto que el Cliente no está presente en el desarrollo del back end, permitimos que lo testee y aceptamos sus quejas/críticas para mejorar la parte funcional de la web, además que nosotros internamente ejecutamos tests de CI/CD para verificar que el código sea funcional y realizamos tests unitarios. Dicho todo esto, y con el buen feedback que nos da el Cliente, creo que los resultados son positivos y el trabajo se está haciendo bien y a buen ritmo.

Conclusiones y trabajo futuro

En resumen, se ha configurado un clúster de ELK que monitoriza logs del back-end global que mi Empresa ofrece al Cliente; con este clúster se crean jobs que mediante técnicas de machine learning, datos que le indicamos y funciones, es capaz de generar anomalías cuando considere que el comportamiento de lo que monitoriza no es el normal; se han creado watches que van lanzado querys a Elasticsearch y generan un webhook si los resultados cumplen X condición. Las gráficas que se generan con estos jobs son recuperadas en la web que estamos desarrollando para su visualización en detalle gracias a la API de Elasticsearch, y los watches envían un webhook para la notificación de usuarios sobre dicho incidente. También se está re-modelando y unificando las webs de monitorizacion/gestión en una sola.

En cuanto trabajo futuro, creo que tengo la capacidad de crear cualquier job o watch que el cliente necesite. Nos falta terminar la web, cuya finalización está prevista para septiembre (aunque ya la estamos subiendo a pre-producción para que el cliente pueda testear y ver su estado). También estamos pensando en cómo aplicar machine learning con los datos de las anomalías que se generan en los jobs para clasificarlas en incidencia o no incidencia (cosa que actualmente se hace con reglas estáticas), y en un futuro poder relacionar incidencias de jobs distintos para evitar abrir incidencias cuyo origen sea el mismo o tengan relación suficiente entre sí.

Anexo: Screenshots

Aquí muestro screenshots que muestran el trabajo realizado hasta la fecha. Se puede observar el progreso de la nueva página web (aún en desarrollo), los jobs realizados, los watches creados, y cómo estos se relacionan con la (antigua) web para notificar a usuarios de los sucesos (función aún en desarrollo en la nueva web).

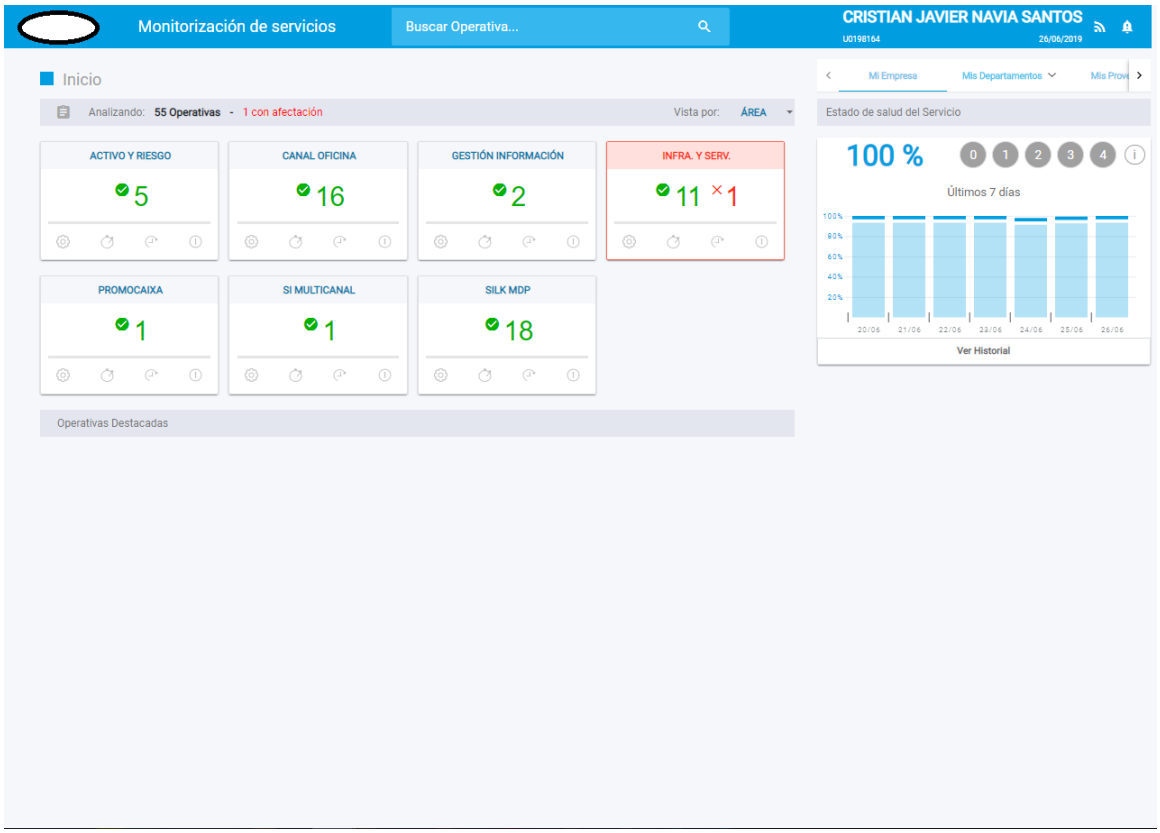

Figure A.1: Home de la nueva página web

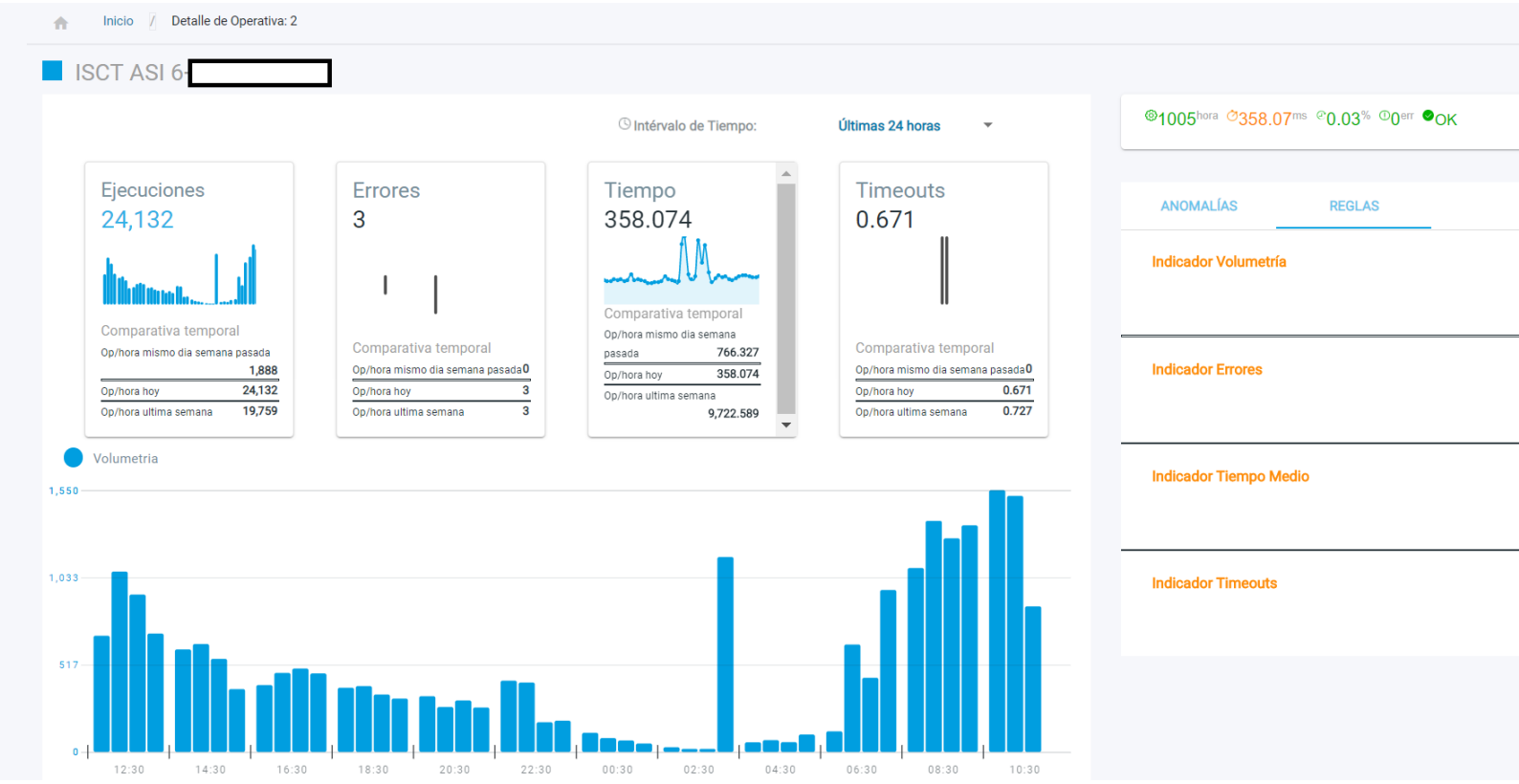

Figure A.2: Detalle de operativa de la nueva página web

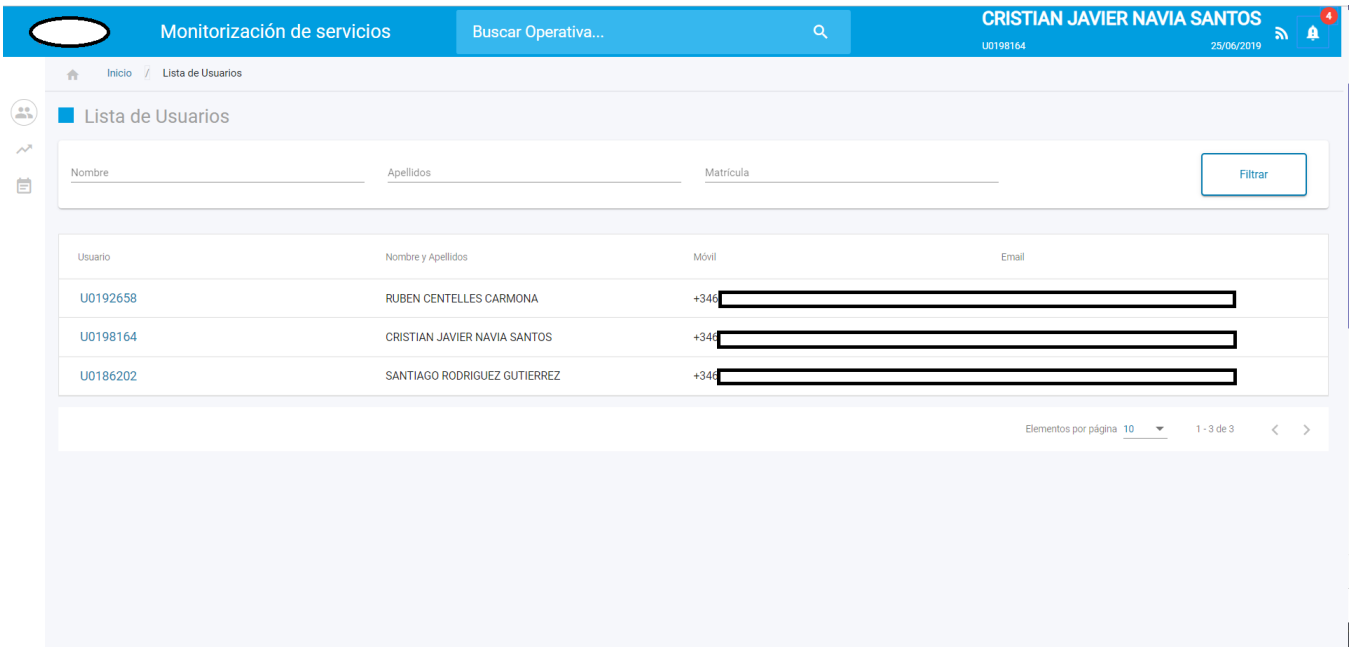

Figure A.3: Apartado de gestión de usuarios y grupos (Lista de usuarios)

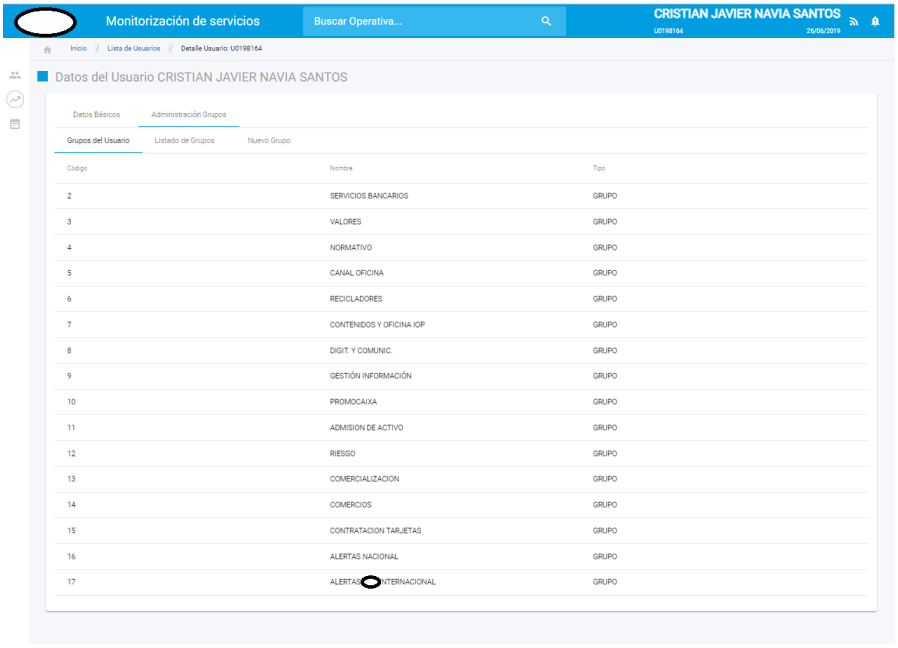

Figure A.4: Apartado de gestión de usuarios y grupos (Administración de grupos)

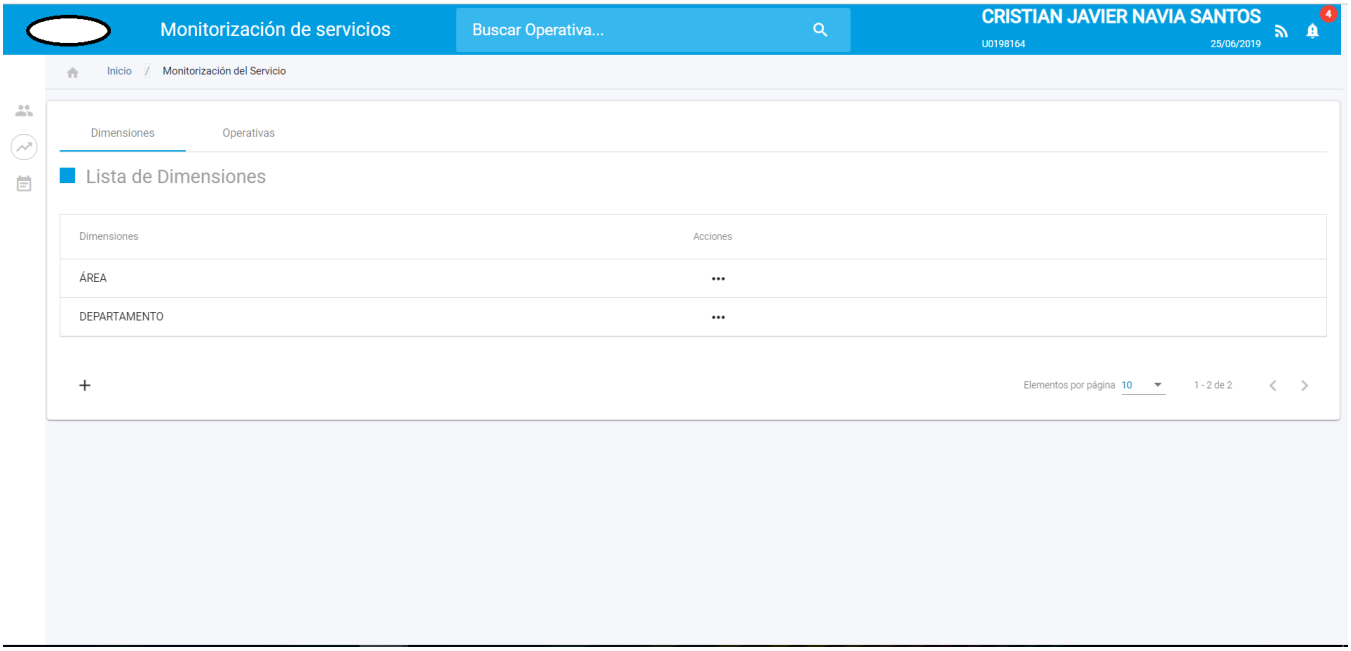

Figure A.5: Apartado de monitorización: Listado de dimensiones

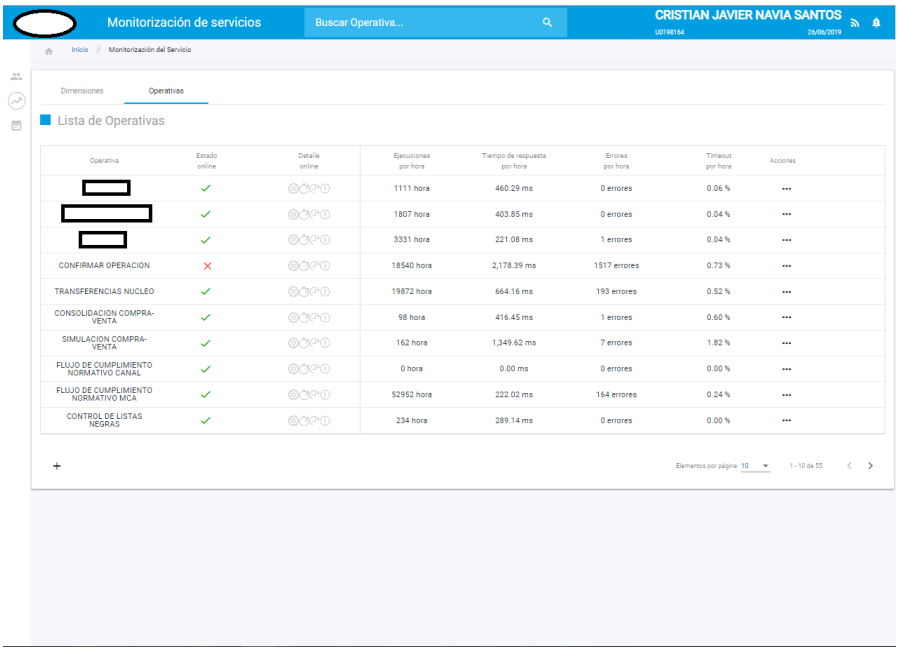

Figure A.6: Apartado de monitorización: Listado de operativas

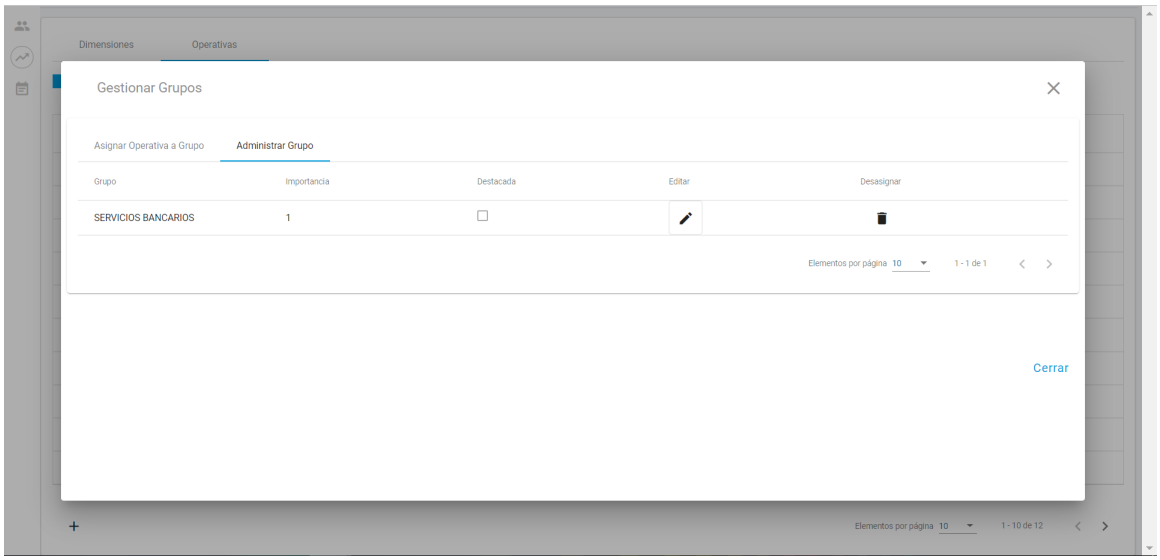

Figure A.7: Apartado de monitorización: Gestión de relación operativa - grupo

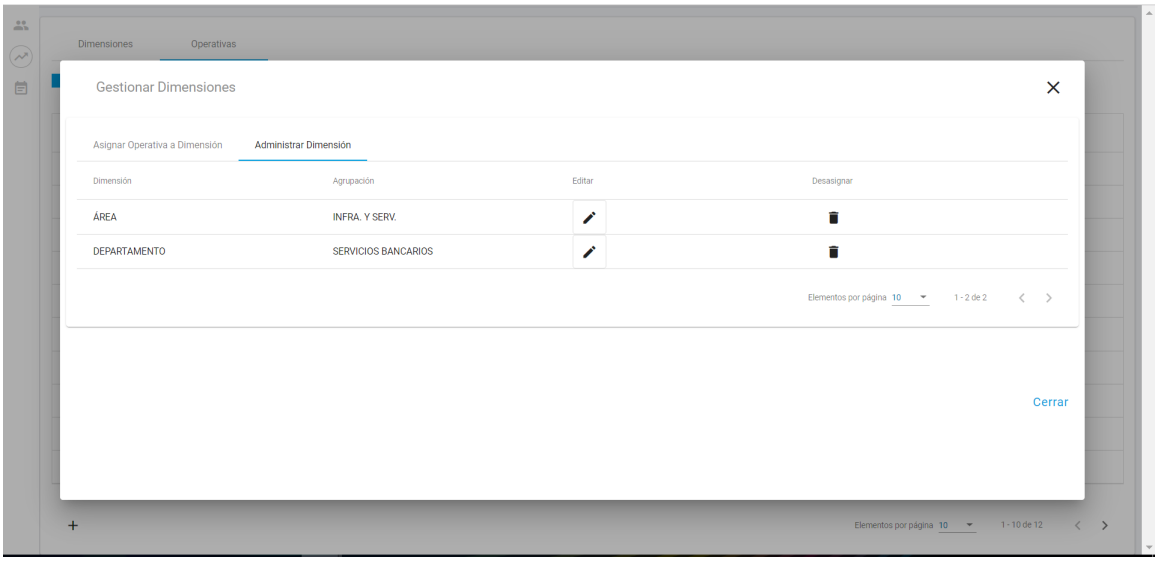

Figure A.8: Apartado de monitorización: Gestión de relación operativa - dimensión

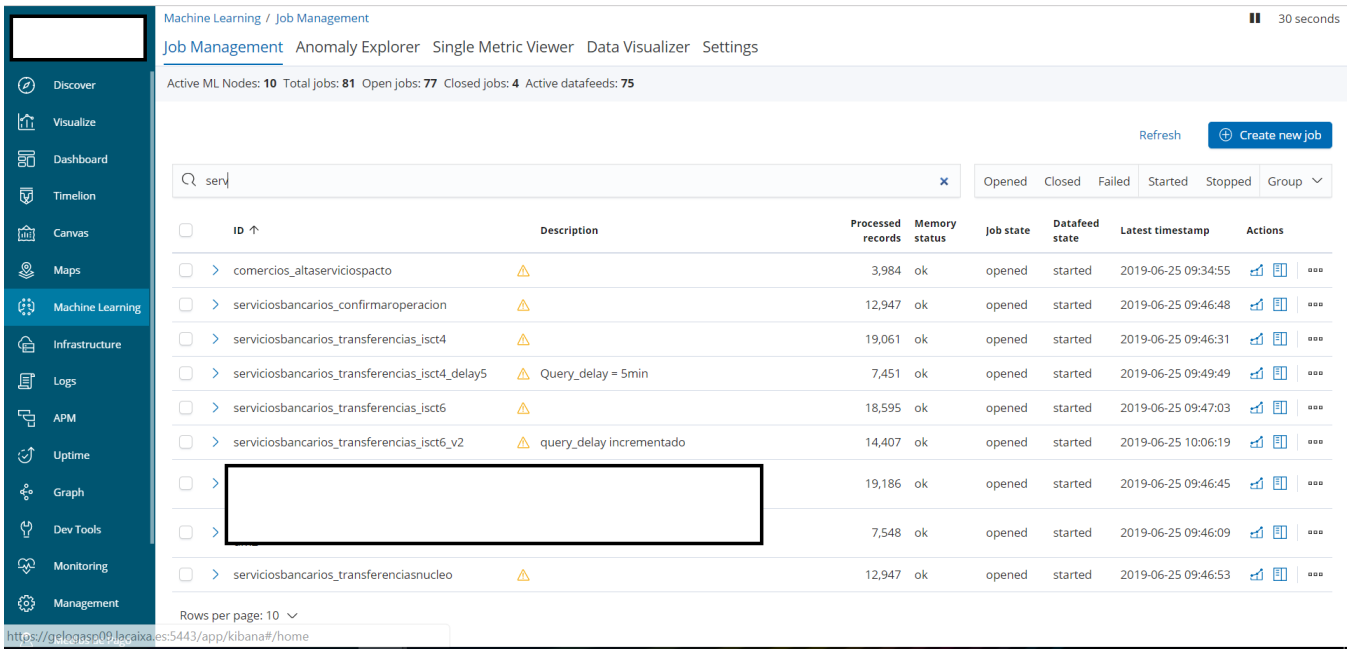

Figure A.9: Clúster de ELK: Listado de jobs

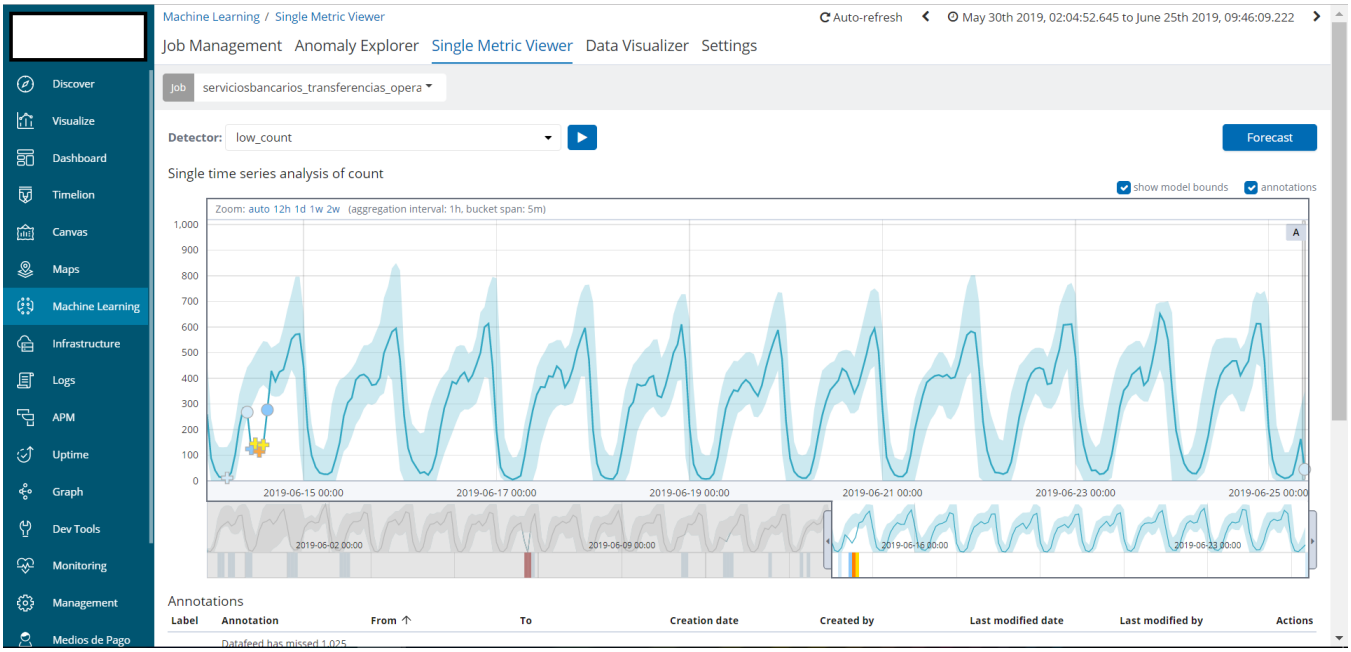

Figure A.10: Clúster de ELK: Gráfica de volumetría de un job

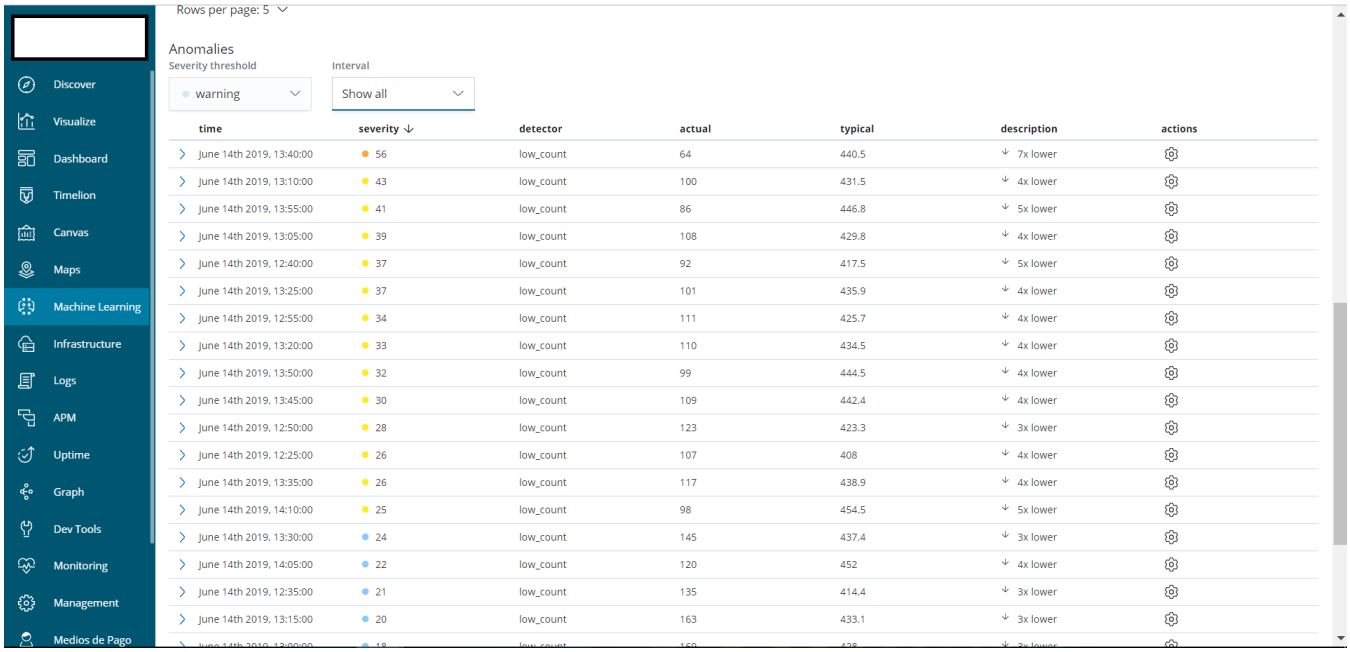

Figure A.11: Clúster de ELK: Listado de anomalías de un job

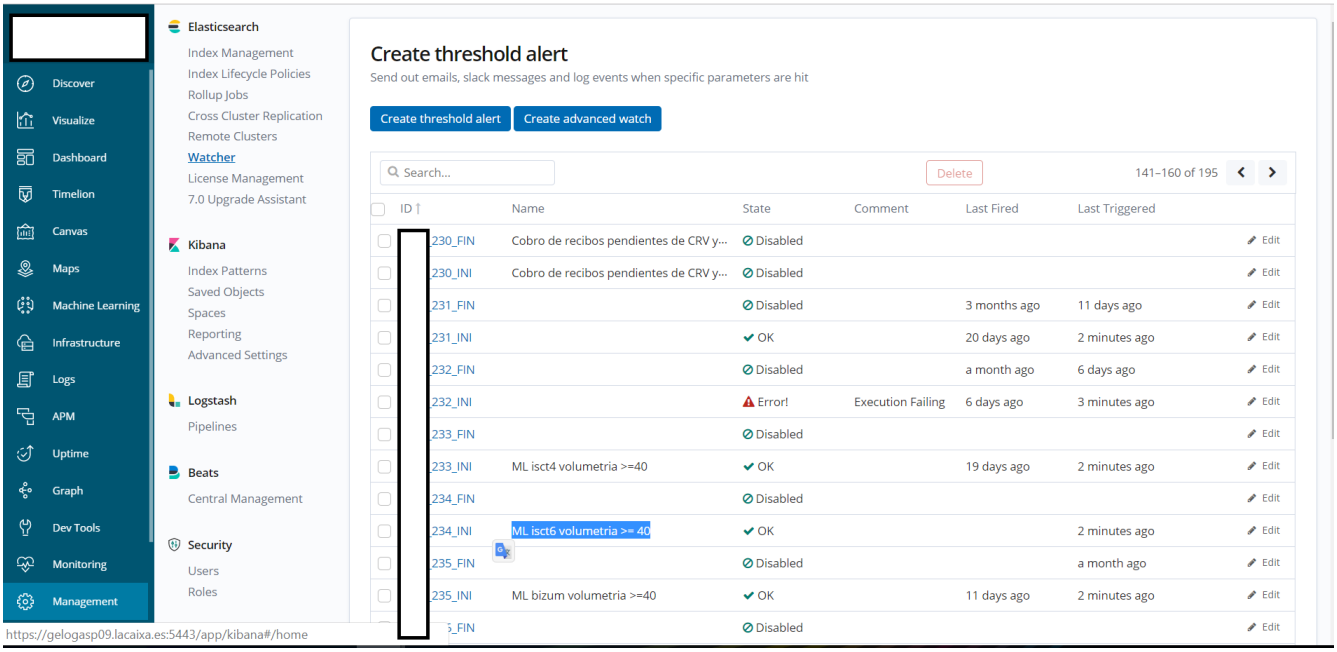

Figure A.12: Clúster de ELK: Listado de watches

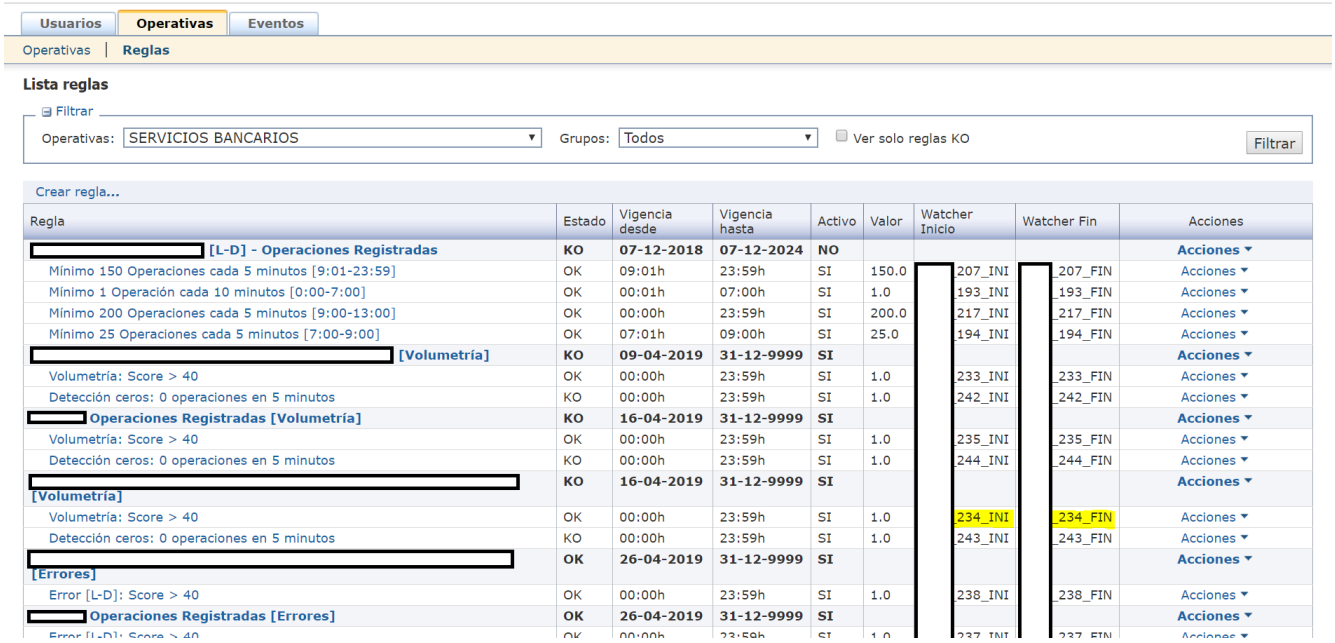

Figure A.13: Antigua web de gestión: Listado de reglas de una operativa

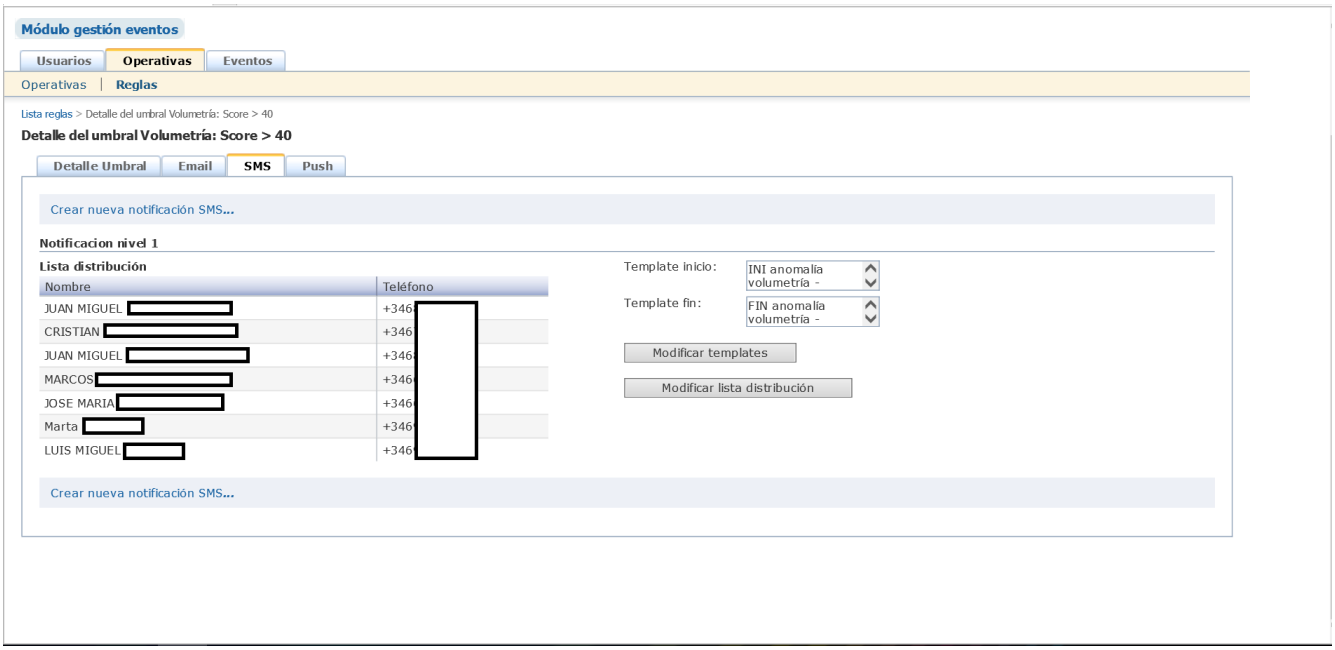

Figure A.14: Antigua web de gestión: Listado de personas a notificar y mensaje.

Referencias y bibliografía

[1] "Subscriptions | elastic stack products & support." https://www.elastic.co/es/subscriptions. Accessed: 2019-04-18. 10

[2] "Apache lucene." https://lucene.apache.org/. Accessed: 2019-02-20 - 2019-03-03. 11

[3] "Db-engines ranking - popularity ranking of search engines." https://db-engines.com/en/ranking/search+engine. Accessed: 2019-06-22. 13

[4] "Cron." https://en.wikipedia.org/wiki/Cron. Accessed: 2019-04-20. 25

[5] "Elasticsearch reference." https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html. Accessed: 2019-02-20 - 2019-06-20.

[6] "Cron." https://www.elastic.co/guide/en/elastic-stack-overview/current/xpack-alerting.html. Accessed: 2019-04-20.

[7] "Elasticsearch from the bottom up." https://www.elastic.co/es/blog/found-elasticsearch-from-the-bottom-up. Accessed: 2019-02-25.

[8] "Scoring de anomalías en el aprendizaje automático y elasticsearch: Cómo funciona." https://www.elastic.co/es/blog/machine-learning-anomaly-scoring-elasticsearch-how-it-works. Accessed: 2019-03-15.

[9] "Building a restful web service." https://spring.io/guides/gs/rest-service/. Accessed: 2019-05-09.