Universidad Autónoma de Madrid
Escuela politécnica superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

MONITORIZACIÓN DE SISTEMAS CON BLUEMIX

Autor: Rodrigo Vicente Amaducci Szwarc
Tutor: Francisco Javier Gómez Arribas

Mayo 2016

MONITORIZACIÓN DE SISTEMAS CON BLUEMIX

Autor: Rodrigo Vicente Amaducci Szwarc
Tutor: Francisco Javier Gómez Arribas

Cátedra UAM/IBM
Dpto. de TEC
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Mayo 2016

Resumen

Resumen

Hoy en día todos los sistemas informáticos generan grandes cantidades de información en los llamados registros o logs de funcionamiento e incidencias, que debido al gran tráfico de datos y al siempre creciente número de dispositivos llegan a alcanzar tamaños de varios Gigabytes, o incluso Teras. El análisis de estos archivos resulta muy útil, y muchas veces esencial, de cara a detectar comportamientos anómalos en el sistema o accesos no autorizados, prever posibles fallos futuros o simplemente estudiar el rendimiento del equipo. Sin embargo, el abultado volumen de datos previamente mencionado, así como la falta de una estructura clara y común en estos registros, hacen del análisis de logs una tarea tediosa y complicada.

Entre las nuevas tecnologías que están creciendo últimamente se encuentran las herramientas Big Data y la computación en la nube. Las técnicas de Big Data son aquellas que se emplean en el análisis y tratamiento de grandes conjuntos de datos desestructurados que no puede ser manejados con herramientas convencionales, y son cada vez más relevantes debido al exponencial aumento de información digital. Por otra parte, la computación en la nube se encarga de ofrecer servicios a través de Internet, que pueden ser usados de manera fácil y transparente por los usuarios. Una de estas plataformas en la nube es IBM Bluemix, que ofrece herramientas y servicios para desarrolladores.

El objetivo de este TFG ha sido desarrollar una aplicación de análisis de logs que sea fácil e intuitiva de utilizar para un usuario sin conocimientos avanzados en informática, y que al estar basada en tecnologías en la nube de Bluemix no haga necesario poseer un equipo con unos requisitos especializados.

Para el desarrollo de la aplicación antes mencionada ha sido necesario tener conocimiento acerca del análisis de logs y las técnicas y herramientas ya existentes, así como sobre las posibilidades de la computación en la nube, especialmente del entorno IBM Bluemix, que es el que ha sido utilizado. Para las distintas partes de la aplicación, dedicadas a la recolección, parseado, almacenamiento, filtrado y visualización de la información, se han aprendido y utilizado diversas tecnologías (DB2, Spring, Hadoop, etc) y lenguajes (Java, JavaScript, HTML, etc).

Palabras Clave

Registros, Nube, Análisis, Prevención, Bluemix, Java, Logs, Big Data

Abstract

Nowadays every computer system generates huge information quantities as operations and incidences logs, that due to the great data traffic and the always growing number of devices can reach sizes of several Gigabytes, and even Terabytes. The analysis of these files is really useful, and sometimes even essential, when we want to detect anomalous behaviours in the system or unauthorized accesses, prevent possible future failures or simply study the performance of a machine. Nevertheless, the previously mentioned large volume of data, among the lack of a clear and common log structure, make log analysis a tedious and complicated task.

Among the numerous technologies emerging these days we can find Big Data and Cloud Computing tools. Big Data techniques are those used for the analysis and treatment of huge unestructured data sets that can not be handled by conventional tools, and are becoming more relevant every day as the amount of digital information increases. On the other hand, Cloud Computing offers services through the Internet, that can be used easily and transparently by the users. One of these cloud platforms is IBM Bluemix, that offers tools and services for developers.

The goal of this Bachelor Thesis was to develop a log analysis application, intuitive and easy to use for an user without advanced computer knowledge, and since it is based in Bluemix cloud technologies, it does not require to be run in a machine with any special requirements.

For the development of the aforementioned application, knowledge about Log Analysis, and the techniques and tools used for it, was needed, and also about the possibilities of Cloud Computing, specially the IBM Bluemix environment that was used. For the application different parts, dedicated to gather, parse, store, filter and show the information, multiple technologies were learnt and used (DB2, Spring, Hadoop, etc), as well as different programming languages (Java, JavaScript, HTML, etc).

Key words

Logs, Cloud, Analysis, Prevention, Bluemix, Java, Big Data

Agradecimientos

Me gustaría agradecer a la Cátedra UAM/IBM el haberme dado la oportunidad de trabajar con ellos, tanto a mi tutor Paco como a Estrella, así como a mis compañeros allí, por toda la ayuda brindada y por hacer mucho más amenas las largas horas de trabajo. Por supuesto dar las gracias a mi familia y amigos, por estar siempre a mi lado e incluso interesarse por mi trabajo, aunque no entendiesen nada de lo que les contaba.

Índice general

Índice de Figuras

1 Introducción

1.1. Motivación del proyecto

El análisis de logs es una de las técnicas más efectivas para la monitorización de sistemas y aplicaciones, la detección de anomalías e incluso la predicción de eventos futuros. Anteriormente una forma de estudiar el comportamiento de un dispositivo o un proceso consistía en observar el tráfico de red generado, sin embargo hoy en día este se encuentra codificado en gran medida, dejando los registros como casi única fuente de información. Puesto que existen cada vez más dispositivos y que el tráfico de información es mayor, también aumenta el tamaño y la cantidad de los logs generados, que de por sí ya contaban con una estructura muy heterogénea y poco clara, lo que hace necesario el uso de herramientas especializadas para su análisis.

En la *Cátedra UAM/IBM* se realizan trabajos sobre cuestiones de computación cognitiva y analítica Big Data, enfocados también en el uso de herramientas cloud. En este contexto se han estudiado las capacidades y servicios que ofrece la plataforma en la nube IBM Bluemix, especialmente en lo relacionado al Big Data, debido a que son tecnologías cada vez más extendidas y que previsiblemente seguirán creciendo en el futuro junto con los volúmenes de datos.

La motivación de este proyecto ha sido por tanto el relacionar ambos temas, desarrollando una aplicación para el análisis de logs en la plataforma cloud Bluemix y aprovechando las capacidades Big Data para el tratamiento de grandes volumenes de datos.

1.2. Objetivos y enfoque

El objetivo de este proyecto ha sido el desarrollo de una aplicación, denominada *LogBlueAnalyzer*, para el análisis de logs y que funcione sobre las tecnologías cloud ofrecidas por la plataforma IBM Bluemix. Esta debe encargarse de todas las fases del proceso para analizar un log:

- Debe **recolectar** los logs de las máquinas en tiempo real.
- Debe **parsear** la información leída de los logs.
- Debe **almacenar** la información parseada.

- Debe **mostrar** la información de los logs al usuario de manera clara, ofreciendo la posibilidad de **filtrar** según se considere oportuno.
- Debe **notificar** al usuario vía e-mail de incidencias que le hayan sido señaladas por el usuario.

Teniendo estas fases en cuenta se ha diseñado la aplicación de forma modular, de manera que para cada una de ellas existe un componente independiente que realizará su tarea a pesar de los cambios que sufran las demás. Todos estos módulos están a su vez implementadas en diferentes entornos y lenguajes. Las ideas básicas que cumple esta aplicación son:

- **Funcionamiento en la nube:** excepto la parte de recolección, que funciona sobre la máquina local, todos los demás componentes funcionan en la nube, liberando así de la necesidad de tener equipos especializados ejecutando el software.
- **Fácil de utilizar:** puede ser utilizada sin dificultad por usuarios sin grandes conocimientos técnicos gracias a su diseño intuitivo. Además, el hecho de funcionar en la nube le ahorra al usuario el tener que instalar y configurar la aplicación.
- **Multiplataforma:** se puede acceder a la interfaz web de la aplicación desde cualquier navegador y en cualquier dispositivo.
- **Concurrencia:** está diseñada para que varios usuarios la puedan utilizar a la vez.
- **Escalabilidad:** la implementación está pensada como una primera parte de un proceso de desarrollo más extenso, siendo el objetivo desarrollar una primera versión funcional de la aplicación a la que posteriormente se le añadan funcionalidades.

1.3. Organización de la memoria

Este documento consta de los siguientes apartados:

- **Estado del arte:** en este capítulo se estudiará el contexto actual de las técnicas y herramientas que se emplearán en el desarrollo del trabajo.
- **Análisis:** en este capítulo se plantearán los requisitos a cumplir por la aplicación, tanto a nivel de utilidades para el usuario como de características generales.
- **Diseño e implementación:** en este capítulo se explicará el desarrollo de la aplicación en cada una de sus fases, así como las diversas técnicas, herramientas y entornos empleados. En cada una de estas fases (recolección, parseado, almacenamiento, filtrado y visualización, notificación) se han utilizado herramientas y entornos diferentes, tanto en local como en la nube, cuyo uso se detallará y explicará.
- **Funcionamiento y utilización:** en este capítulo se presentará una guía para manejar la aplicación, con el objetivo también de detallar sus funcionalidades y casos de uso.
- **Conclusiones y trabajo futuro:** en este capítulo se sintetizarán los resultados del trabajo y se analizará si se han cumplido los objetivos propuestos. Asimismo, también se plantearán posibles cuestiones con las que continuar el trabajo en el futuro.

2

Estado del arte

En este capítulo se estudia el estado del arte de las tecnologías, herramientas y técnicas que se abordan a lo largo de este proyecto.

2.1. Computación en la Nube

La *computación en la nube*, también conocida como *computación bajo demanda* o *cloud computing* en inglés, se define según la Real Academia de Ingeniería como la "utilización de las instalaciones propias de un servidor web albergadas por un proveedor de Internet para almacenar, desplegar y ejecutar aplicaciones a petición de los usuarios demandantes de las mismas" [1]. En otras palabras, es un paradigma que permite el acceso a una serie de servicios informáticos de forma remota e inmediata, que son ofrecidos por un proveedor que es el que mantiene la infraestructura, de manera que el usuario no requiera conocimientos avanzados ni poseer equipos especializados. El acceso a los servicios se realiza a través de Internet y se consumen bajo demanda, es decir, cada usuario contrata los servicios y la cantidad de los mismos que requiere en un momento concreto, liberándolos cuando no le son necesarios, y pagando solo por el consumo realizado.

Cuando la nube es gestionada y ofrecida por terceros ajenos al usuario recibe el nombre de *nube pública*, y se accede a ellas de manera remota a través de Internet. Los datos y servicios consumidos por todos los usuarios se encuentran en los mismos servidores y sistemas de almacenamiento, pero cada uno de ellos solo puede acceder a su información y es ajeno a las actividades que están llevando a cabo el resto. En otros casos una determinada organización quiere ofrecer servicios en la nube pero solo a sus miembros, por lo general por motivos de privacidad y seguridad, por lo que montan una *nube privada*. Varias organizaciones pueden querer compartir una nube privada para realizar una tarea concreta, constituyendo una *nube comunitaria*. Un proveedor puede escoger crear una *nube híbrida*, en la que una parte de los servicios son privados y otra está abierta al público (pero con acceso controlado). Existen otros tipos de nubes como las *nubes distribuidas*, que tienen los recursos distribuidos en diferentes localizaciones, o las *Intercloud*, que consiste en una "nube de nubes", entre otras.

Según el National Institute of Standards and Technology, existen cinco aspectos esenciales que definen la computación en cloud [2]:

- **Servicio bajo demanda:** los usuarios pueden obtener servicios automáticamente, sin necesidad de interacción humana con el proveedor.
- **Facilidad de acceso:** los servicios pueden ser accedidos desde cualquier lugar con conexión a Internet, y con diversos dispositivos.
- **Agrupación de recursos:** los recursos ofrecidos son utilizados por varios usuarios con diferentes necesidades, por lo que deben asignarse de forma dinámica según las mismas.
- **Elasticidad:** los usuarios pueden obtener recursos y liberarlos rápidamente.
- **Servicio controlado:** el uso de los recursos es controlado y optimizado automáticamente, pudiendo ser monitorizado y medido para mayor transparencia del usuario y el proveedor.

2.1.1. Ventajas

- Un usuario pequeño puede contratar pocos servicios al principio, suficientes para cubrir sus necesidades, y escalar rápida y fácilmente si lo requiere. Esto lo puede hacer incluso temporalmente, para hacer frente a picos de trabajo, pagando por los recursos solo el tiempo que los usa.
- El usuario puede disponer de los recursos en tiempo real, sin necesidad de grandes tiempos de provisionamiento, y no tiene que preocuparse por la instalación y configuración del software, ya que de eso se encarga el proveedor.
- Consecuencia directa de los puntos antes mencionados es que los costes para comenzar son menores, puesto que se pueden obtener recursos cuyo montaje e instalación en un caso normal requerirían de una gran inversión inicial.
- El mantenimiento y actualización de los recursos es realizado por el proveedor, por lo que el usuario no tiene que preocuparse de realizar estas tareas en todas sus máquinas.
- Mayor eficiencia en la utilización de la infraestructura, ya que se usan técnicas de virtualización para que varios usuarios compartan los mismos recursos. Esto también deriva en un ahorro energético significativo.
- Al encontrarse todos los recursos centralizados se facilita su seguridad, al menos a nivel físico.

2.1.2. Desventajas

- Uno de los puntos más sensibles es la seguridad, puesto que un proveedor de servicios cloud almacena grandes cantidades de información, que en muchos casos puede ser de interés para agentes maliciosos. Un par de ejemplos de violaciones de seguridad en la nube son el hackeo en 2014 de Dropbox (7 millones de contraseñas robadas) e iCloud (documentos personales). Las mayores amenazas de seguridad en la nube según la Cloud Security Alliance son las APIs inseguras, la perdida y filtrado de datos y el uso malvado de las tecnologías Cloud (para infectar con malware a miles de ordenadores por ejemplo), entre otras [3]. Debido a esto muchos proveedores centran sus esfuerzos en ofrecer una seguridad fuerte para sus servicios, que en muchos casos es mejor de lo que usuarios pequeños podrían obtener por su cuenta.

- El hecho de que todos los recursos se encuentren centralizados generan una relación de dependencia de los proveedores de servicios: si estos deciden dejar de prestar un servicio o simplemente desaparecen el usuario no puede hacer nada. Además en muchos casos los Términos y Condiciones del servicio no aclaran que los datos del usuario no puedan ser utilizados con fines de lucro por el proveedor. Esta falta de libertad y privacidad de los usuarios ha sido fuertemente criticada por diversas personalidades, como por ejemplo el fundador de la Free Software Fundation, Richard Stallman [4].
- Como las aplicaciones se van actualizando automáticamente, así como sus interfaces, los usuarios con pocos conocimientos tecnológicos pueden tener dificultades para mantenerse al día. Además, la integración en aplicaciones de forma automatizada también se puede ver afectada.
- Para acceder a los servicios es necesario tener conexión a Internet.

2.1.3. Tipos de servicios

Se distinguen tres tipos de servicios en la nube, jerarquizados como se muestra en la figura 2.1:

- **Software as a Service (SaaS):** consiste en una aplicación concreta ofrecida como servicio, mientras que la plataforma y la infraestructura sobre las cuales funciona son gestionadas por el proveedor. De esta manera el usuario no tiene que preocuparse de la instalación y mantenimiento del software y carece de control sobre el mismo. La aplicación se ejecuta sobre máquinas virtuales y pueden soportar multitenencia, es decir, una sola instancia de la misma trabajando para múltiples usuarios. Algunos ejemplos de SaaS son Gmail y Microsoft Office 365.
- **Platform as a Service (PaaS):** el servicio que se proporciona es un entorno de desarrollo, con una serie de módulos y funcionalidades (tales como bases de datos, servidores web, compiladores, etc) que permiten al usuario desarrollar y desplegar sus aplicaciones en la nube sin preocuparse de las configuraciones software y hardware del sistema (ni de su coste). Ejemplos de PaaS son Microsoft Azure e IBM Bluemix.
- **Infrastructure as a Service (IaaS):** se proporcionan servicios hardware, tales como almacenamiento básico, capacidades de cómputo y firewalls, a través de la red. Usando generalmente máquinas virtuales se logra abstraer al usuario de detalles de la infraestructura como su localización, escalabilidad o seguridad. El usuario sin embargo debe hacerse cargo de la instalación y el mantenimiento de los sistemas operativos y el software que requiera. Ejemplos de IaaS son Amazon Web Services y Softlayer.

2.2. IBM Bluemix

Bluemix es un entorno *Platform as a Service* creado por IBM en 2014, que permite desarrollar y desplegar aplicaciones en diversos lenguajes de programación (Java, Node.js, Go, PHP, Python, Ruby, Scala) y utilizar diferentes recursos (bases de datos, entornos Big Data, computación cognitiva, etc). Está basado en la PaaS open source *Cloud Foundry* y funciona sobre una IaaS *SoftLayer*. Actualmente es posible usar el servicio de manera gratuita, con diferentes planes y funcionalidades gratuitas para cada servicio, con ciertas limitaciones de capacidad y uso en algunos casos. Se muestra un ejemplo del panel de control de Bluemix en la figura 2.2.

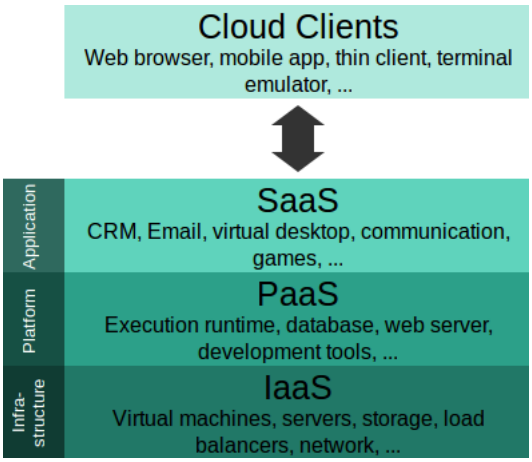

Figura 2.1: Jerarquía de servicios en la nube.

2.2.1. Buildpacks

Un *buildpack* se encarga de obtener los artefactos adecuados (scripts que implementan la lógica del buildpack), iniciar el entorno de ejecución correspondiente e instalar y desplegar la aplicación. Por ejemplo, en el caso de Bluemix, si queremos lanzar una aplicación contenida en un archivo WAR (archivo Java ejecutable en un servidor web) tendremos que utilizar el buildpack *Liberty for Java*, que desplegará un servidor de aplicaciones en el que ejecutar nuestro archivo.

Hay tres buildpacks oficiales, siendo uno de ellos el ya mencionado para aplicaciones en Java y los otros dos para Node.js y XPages. Existen también otros creados por la comunidad que soportan Go, PHP, Python, Ruby y aplicaciones Tomcat. Todos los buildpacks son gratuitos en el momento de redacción de este documento.

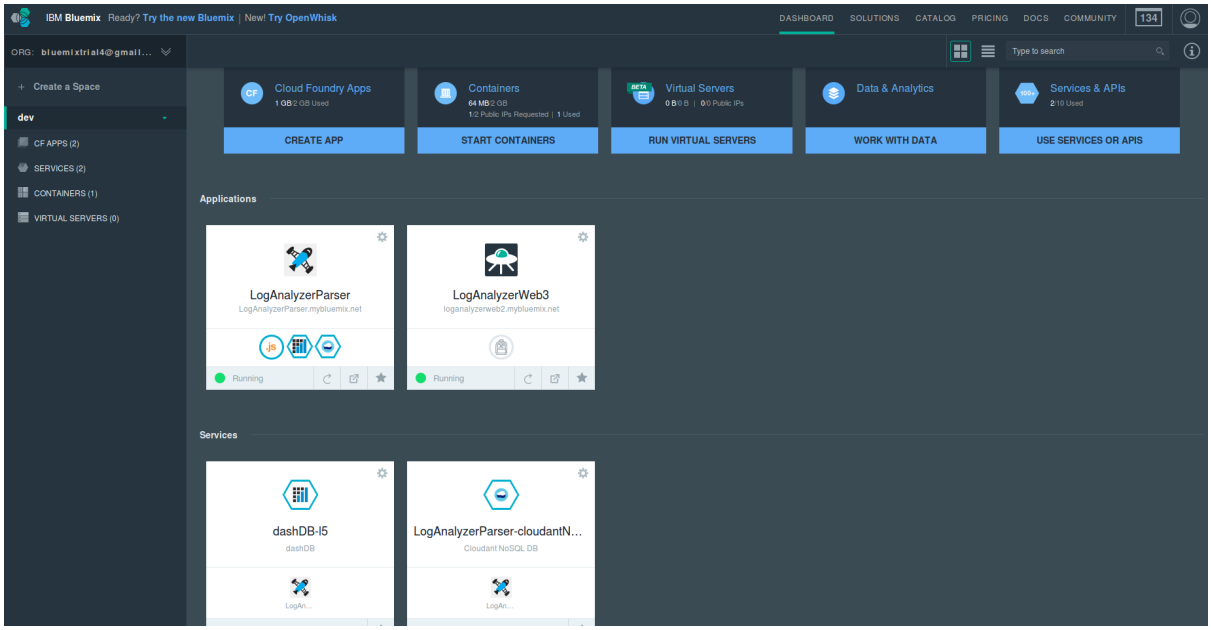

Figura 2.2: Ejemplo del Panel de Control de Bluemix.

2.2.2. Servicios

La mayor parte del catálogo de Bluemix lo ocupan los servicios, que consisten en variadas herramientas y funcionalidades (algunas proporcionadas por IBM y otras por empresas externas) que se pueden añadir a las aplicaciones creadas o usar de manera independiente. Existe un gran número de ellos, que varía frecuentemente debido a la inclusión de algunos nuevos y la eliminación de otros. Los servicios están agrupados en las siguientes categorías:

- **Watson:** herramientas de computación cognitiva basadas en el superordenador Watson, capaz de interpretar el lenguaje natural producido por los humanos. Entre las herramientas de este apartado encontramos un Clasificador de Lenguaje Natural y varias utilidades de interpretación de textos (traducción, extracción de relaciones, análisis de personalidad, etc).
- **Móvil:** conjunto de funcionalidades para el desarrollo de aplicaciones móviles, tales como envío de notificaciones y análisis de los desplazamientos físicos de un dispositivo.
- **DevOps:** utilidades destinadas a facilitar y acelerar la producción de software, como herramientas para ayudar en el despliegue o monitorización.
- **Web y aplicación:** variado conjunto de utilidades para el desarrollo de aplicaciones web, que van desde planificadores de tareas hasta aceleradores de búsqueda en caché.
- **Red:** herramientas para crear redes privadas como VPNs.
- **Integración:** utilidades para integrar aplicaciones en Bluemix con otras locales.
- **Datos y análisis:** gran conjunto de servicios de base de datos de diferente tipo (DB2, NoSQL, PostgreSQL, MongoDB, etc) y de herramientas para el análisis de grandes cantidades de información (Apache Hadoop, Apache Spark).
- **Seguridad:** funcionalidades para proteger las aplicaciones desplegadas en la plataforma.
- **Almacenamiento:** servicio de almacenamiento de objetos no estructurado en la nube.
- **Internet de las cosas:** herramientas enfocadas al llamado *Internet de las cosas*, que es como se conoce a la interconexión de objetos cotidianos, como un coche o una nevera por ejemplo, con Internet para transmitir información recopilada con sensores y otros mecanismos.

2.2.3. Contenedores

La última adición a la plataforma Bluemix han sido los *contenedores* de Docker, que consisten en un entorno virtualizado cerrado que contiene todo lo necesario para el funcionamiento de una aplicación: código, librerías y dependencias, entorno de ejecución, herramientas del sistema, etc. De esta manera se puede tener a la aplicación funcionando en un entorno controlado que no varía y es indiferente del lugar donde se este ejecutando.

La filosofía de los contenedores es la siguiente: la plataforma proporciona un entorno con sistema operativo sobre el cual el usuario despliega sus contenedores, los cuales toman del sistema los recursos que necesitan pero por lo demás funcionan de manera aislada unos de otros. De esta manera el usuario no tiene que preocuparse de configurar el sistema operativo y es más eficiente, puesto que no es necesario instalar sistemas completos en cada máquina virtual.

2.3. Logs: Registro de eventos y errores

Se emplea el término *registro* o *log* para referirse a un fichero de texto que recopila información sobre eventos sucedidos en un sistema o aplicación. Cada línea del archivo se corresponde con un evento y por lo general son de tipo semi-estructurado, puesto que suelen estar divididos en unos campos predefinidos, pero dentro de estos la información puede ser no estructurada. La información contenida en estos ficheros suele responder a lo que se llama las cinco W sobre un evento: who, why, when, where y what (quién, por qué, cuándo, dónde y qué). Se muestra un ejemplo de un log de errores de Apache en la figura 2.3.

```
[Sat May 09 22:47:15 2016] [error] [client 74.6.53.167] File does not exist: /opt/web-ii/old
[Sat May 09 22:47:21 2016] [error] [client 181.177.248.174] File does not exist: /opt/web-ii/favicon.ico, referer: http://arantxa.ii.uam
.es/~jmmartinez/Pseudocodigo.htm
[Sat May 09 22:47:42 2016] [error] [client 181.168.12.202] File does not exist: /opt/web-ii/favicon.ico, referer: http://arantxa.ii.uam.
es/~gdrivera/labetcii/curso0607/proyecto.htm
[Sat May 09 22:47:58 2016] [error] [client 157.55.39.99] File does not exist: /home/rc2lab/public_html/pr2009
[Sat May 09 22:48:05 2016] [error] [client 189.133.92.168] File does not exist: /opt/web-ii/favicon.ico, referer: http://arantxa.ii.uam.
es/~epulido/bdatos/php.pdf
[Sat May 09 22:48:11 2016] [error] [client 94.0.93.157] File does not exist: /opt/web-ii/favicon.ico, referer: http://arantxa.ii.uam.es/
[Sat May 09 22:48:12 2016] [error] [client 186.119.121.188] File does not exist: /opt/web-ii/favicon.ico, referer: http://arantxa.ii.uam
.es/~ig/practicas/enunciados/prac3/binario.pdf
[Sat May 09 22:48:37 2016] [error] [client 90.23.250.124] File does not exist: /opt/web-ii/favicon.ico, referer: http://arantxa.ii.uam.
es/~jms/pfcsteleco/lecturas/20070619JavierGarciaOcon.pdf
[Sat May 09 22:49:05 2016] [error] [client 37.237.148.88] File does not exist: /opt/web-ii/favicon.ico
[Sat May 09 22:49:06 2016] [error] [client 37.237.148.88] File does not exist: /opt/web-ii/favicon.ico
[Sat May 09 22:49:12 2016] [error] [client 189.144.47.239] File does not exist: /opt/web-ii/favicon.ico, referer: http://arantxa.ii.uam.
es/~eguerra/docencia/0708/09%20Command.pdf
[Sat May 09 22:49:15 2016] [error] [client 181.174.104.102] File does not exist: /opt/web-ii/favicon.ico, referer: http://arantxa.ii.uam
.es/~eguerra/docencia/0809/06%20Creacion.pdf
```

Figura 2.3: Ejemplo de log de errores de un servidor Apache.

El análisis de estos logs es una de las técnicas de seguridad informática más efectivas, y también es usado para estudiar cuestiones de rendimiento y uso de los equipos y aplicaciones. Esto se debe a que es una de las pocas fuentes de información sobre funcionamiento de sistemas y aplicaciones disponibles actualmente. Anteriormente era común el análisis del tráfico de red para estas tareas, sin embargo hoy en día la mayoría de este tráfico está cifrado por lo que no se puede analizar, y previsiblemente en pocos años se encriptará por completo.

Existen varios factores que dificultan considerablemente el análisis de registros:

- Debido al alto grado de uso de sistemas informáticos y al gran flujo de información que se produce hoy en día la cantidad de registros generados es enorme, alcanzando fácilmente tamaños del orden de Gigabytes, e incluso Terabytes en servidores grandes. Para poder utilizar toda esa información es necesario que al menos se presente de manera ordenada y visualmente interpretable al analista. Para el manejo de estos grandes volúmenes de datos existen las herramientas y técnicas *Big Data*, que se detallarán en la sección 2.4.

- Por lo general la mayor parte de esa información registrada es irrelevante para encontrar anomalías, intrusiones, fallos y demás incidencias de interés. Por ello se requiere que de alguna forma los datos se filtren y clasifiquen.

- Aunque por lo general suelen compartir ciertos elementos comunes, cada aplicación, proceso o sistema tiene su propio formato para escribir los logs, algunos de ellos más estructurados que otros, y con distintos campos y orden.

Para hacer frente a estas dificultades existen numerosas herramientas para el análisis de logs, con características bastante variadas. A continuación se ofrece una pequeña reseña de algunas de las más importantes.

2.3.1. Herramientas comerciales de análisis de logs

Splunk

Es la herramienta más importante y utilizada, contando entre sus clientes a varias de las multinacionales más grandes del mundo como Goldman Sachs y Telefónica. Es capaz de analizar casi cualquier tipo de log, ya sean sobre seguridad, monitorización de sistemas, lógica empresarial, etc. También sus herramientas de búsqueda y visualización son muy completas, pudiendo acceder a casi cualquier tipo de dato.

Un factor negativo sobre Splunk es que su versión convencional hay que instalarla en el lugar de uso, lo cual resulta muy costoso en dinero y complejidad, pudiendo necesitar su propio cluster. Actualmente cuentan con una versión en la nube, *Splunk Cloud*, para hacer frente a este inconveniente. El otro punto a tener a cuenta es su elevado precio, costando 1350 dólares por mes la versión SaaS. Hay un plan gratuito de la versión tradicional pero carece de sistema de alertas y está limitada a 500MB por día.

Sumo Logic

Esta herramienta surge como un intento de crear un Splunk en la nube, aunque actualmente se ha ganado un nombre como software de análisis por si mismo. Al igual que el anteriormente mencionado Splunk, posee varias herramientas para visualización de datos y notificaciones, siendo además bastante sencillo de usar. Señalar también que está enfocado al análisis de logs empresariales, pero aún así es interesante estudiarlo por ser un SaaS.

Existe una versión gratuita que permite hasta tres usuarios y 500MB por día y otra llamada Profesional que admite hasta 20 usuarios y cuesta 90 dólares por mes. En cuanto al plan completo no hay un precio específico y depende los servicios contratados.

ElasticSearch + Logstash + Kibana (ELK)

En este caso son tres herramientas funcionado conjuntamente para ofrecer una aplicación de análisis de logs completa: *Logstash* se encarga de la recolección y administración de los logs, *ElasticSearch* de la indexación y búsqueda y *Kibana* de la visualización, siguiendo la arquitectura mostrada en la figura 2.4. La particularidad de esta solución es que es código abierto, por lo que es generalmente más barata pero ofrece tantas posibilidades como las otras.

Sin embargo, el hecho de ser tres aplicaciones separadas tiene sus inconvenientes, ya que en entornos de producción deben ejecutarse en máquinas distintas y además cada una se maneja de forma diferente: Logstash funciona con *Ruby*, Kibana con *JavaScript* y ElasticSearch tiene su propia *API REST*.

Logstash es una herramienta diseñada en Ruby y que se ejecuta sobre una *Java Virtual Machine*, por lo que se puede usar en cualquier sistema operativo que corra JVM. Emplea una serie de plugins para tratar la información, que son de cuatro tipos:

- **Entradas:** fuentes de los datos.
- **Códecs:** conversores de formato de los datos.
- **Filtros:** modifican y eliminan datos.
- **Salidas:** destinos a los que los datos son enviados.

Figura 2.4: Flujo de datos de ELK.

ElasticSearch es un motor de búsqueda de texto completo, distribuido y que soporta multitenencia y basado en Lucene. Se utiliza mediante una API REST y usa documentos JSON. Almacena la información que recibe de Logstash ya procesada. Por último, *Kibana* es un plugin para la visualización de datos indexados en ElasticSearch.

2.4. Big Data

Los avances tecnológicos y la masificación del uso de dispositivos computacionales en los últimos años han provocado que en la actualidad se generen enormes cantidades de datos digitales, que alcanzan el orden de zettabytes (10^{21} bytes). El origen de estos datos es de los más variado, clasificándose por lo general dentro de los siguientes campos:

- **Web y redes sociales:** todo el contenido web y la información compartida por redes como Twitter o Facebook.
- **Grandes transacciones:** registros de operaciones bancarias, ventas, registros telefónicos, etc.
- **Máquina-a-máquina:** información compartida entre máquinas, como señales GPS por ejemplo.
- **Biométrica:** datos obtenidos a partir de escaneos faciales, de retina, de huella dactilar, etc.
- **Generados por humanos:** información variada generada por las personas, como documentos académicos o notas de voz.

Debido a esto no solo el volumen de la información es muy elevado, sino que su tipo y estructura también.

Los conjuntos de datos muy grandes pero muy estructurados son fáciles de manejar, así como los conjuntos pequeños de datos sin estructura. Es la combinación de una gran cantidad datos y sin estructura fija lo que da lugar al *Big Data* [5], que podemos definir como un conjunto de información cuyo análisis y procesamiento mediante técnicas convencionales llevaría más tiempo del razonable.

Entre todo ese amasijo enorme de información diferenciamos entre tres tipos de datos: los *estructurados*, que poseen un formato y longitud específicos y pueden almacenarse en tablas relacionales, como los timestamp; los *no estructurados*, los cuales carecen de formato y por tanto no se pueden almacenar en una base de datos relacionales, como los mensajes de un foro por ejemplo; y los *semi-estructurados*, que no poseen unos campos específicos pero si unos delimitadores claros, como los archivos CSV.

Una tecnología que ha surgido como respuesta a los retos del Big Data ha sido el almacenamiento *NoSQL* o *Not Only SQL*. Este tipo de bases de datos no siguen el esquema de entidad-relación y ofrecen un sistema mucho más flexible que las bases relacionales. Existen cuatro tipos de bases de datos NoSQL:

- **Clave-Valor:** se accede a los datos a partir de una clave y se guardan tal cual, sin ser necesario interpretación o formateo estricto, por lo que es un sistema muy flexible. *Cassandra* es la base de datos de este tipo más conocida.
- **Documental:** parecidas a las anteriores, con la diferencia de que los datos se guardan en formato semi-estructurado (JSON, XML, etc), por lo que pueden realizarse consultas sobre ellos. Un ejemplo de esta clase de base de datos es *MongoDB*.
- **En grafo:** en vez de basarse en el uso de tablas lo hace en la teoría de grafos, considerando los datos (de cualquier tipo) como nodos y las relaciones entre ellos como las aristas. *Facebook* usa este sistema, usando a los usuarios como nodos y sus amistades e intereses como aristas.
- **Orientado a columnas:** parecido al Documental, solo que en este caso se puede almacena más de un atributo con cada clave. Un ejemplo de esta clase de almacenamiento es *HBase*.

También ha sido necesario el desarrollo de nuevas técnicas para el análisis del Big Data:

- **Asociación:** se emplea para encontrar relaciones entre variables. Esta es la clase de tecnología que se utiliza para crear los anuncios personalizados que abundan en Internet.
- **Text Analytics:** gran parte del Big Data consiste en textos en lenguaje natural producido por personas, que son analizados para extraer sentimientos o predecir palabras, como hace un buscador web.
- **Data Mining:** conjunto de técnicas de aprendizaje automático y de estadística empleadas sobre grandes conjuntos de datos para predecir comportamientos.
- **Clustering:** consiste en la división de grandes conjuntos de datos en otros más pequeños con alguna similitud en común, que no se conocía al principio.

2.4.1. Apache Hadoop

Apache Hadoop [6] es un framework software de código abierto diseñado para trabajar con grandes volúmenes de datos de forma distribuida en varios nodos. Está basado en los trabajos de Google sobre *MapReduce* y *Google File System* y el máximo contribuyente a está tecnología ha sido Yahoo. Señalar que uno de los primeros usos de esta tecnología fue el de tratar con grandes cantidades de logs para su análisis. Sus componentes básicos son cuatro: *Hadoop Commons*, que consiste en el conjunto de librerías con funcionalidades básicas de la plataforma; *Hadoop Distributed File System*, que es el sistema de almacenamiento; *Hadoop YARN*, que se encarga de la planificación y administración del cluster; y *Hadoop MapReduce*, un sistema de procesamiento en paralelo de grandes conjuntos de datos.

El sistema de almacenamiento HDFS es distribuido, escalable y portátil, diseñado para funcionar sobre hardware convencional sin grandes especificaciones. Esto se debe a que se basa en la premisa de que ya que el hardware suele fallar es mejor tener muchos equipos de menor calidad pero con la información replicada, de manera que aunque fallen frecuentemente son fáciles de sustituir y los datos siguen disponibles, que tener pocos equipos de mucha calidad, que acabarán por fallar también, con consecuencias más graves. Los archivos que se guardan en este sistema suelen ser de gran tamaño (del orden de terabytes a gigabytes) que es fragmentado y distribuido por un gran número de nodos, siendo cada parte replicada (por defecto tres veces) para soportar fallos y pérdidas. Los procesos son más eficientes cuanto más cerca de la información a utilizar están, sobretodo en casos en los que hay tantos datos, y al mismo tiempo mover el proceso es más eficiente que mover los datos, por lo que HDFS posee herramientas para acercar los procesos a la información.

La arquitectura de HDFS (figura 2.5) es de tipo maestro/esclavo, con un nodo maestro *NameNode* que administra todo el sistema de almacenamiento y se encarga de tareas como el mapeo de los bloques y abrir, cerrar y renombrar ficheros o directorios, de manera que el usuario solo trata con un fichero único; y varios nodos esclavos *DataNodes* que se alojan en cada uno de los demás nodos del cluster y contienen los fragmentos de los ficheros, realizando las tareas de lectura, escritura, creación y borrado de bloques cuando el NameNode lo indica.

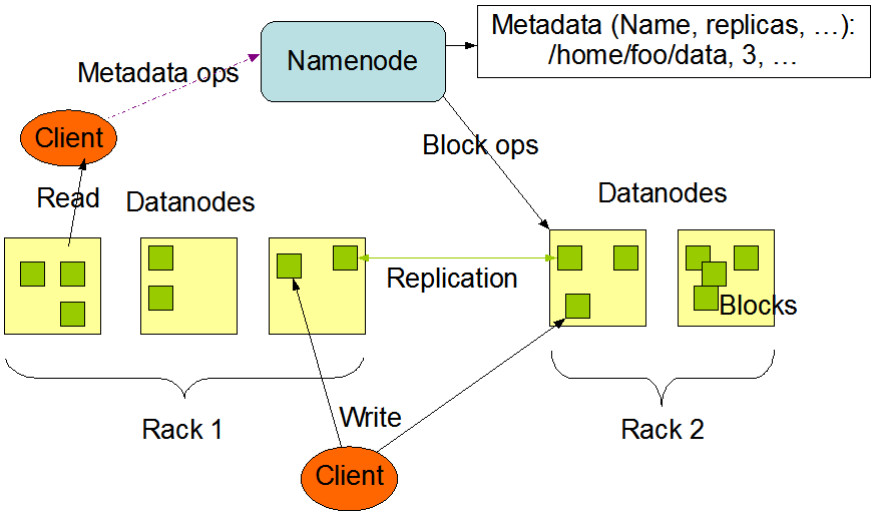

Figura 2.5: Arquitectura de HDFS.

Los programas diseñados para funcionar sobre Hadoop usan el framework *MapReduce*, que permite procesar de forma paralela varias tareas en un gran número de nodos. Este framework se basa de nuevo en la arquitectura maestro/esclavo, con un nodo *JobTracker* encargado de planificar y monitorizar el trabajo de los esclavos, y varios nodos *TaskTracker*, uno por cada otro nodo del cluster, que realizan los trabajos mandados por el maestro. Un programa MapReduce lee la información del disco, ejecuta una función *Map* sobre los datos de forma paralela y en el nodo donde se encuentren los datos, hace un *Reduce* sobre los resultados del mapeo y guarda los resultados del Reduce en disco.

2.4.2. Apache Spark

En 2013 surge una alternativa a Hadoop conocida como *Spark* [7], que posee una velocidad de procesamiento mucho mayor (hasta 100 veces). Este framework sigue funcionando sobre Hadoop YARN y HDFS, siendo una alternativa a MapReduce en realidad. La mejora en la velocidad se

debe a que Spark carga los datos de memoria, lo cual es mucho más rápido que cargarlos de disco como hace MapReduce. Esto sin embargo requiere de una mayor cantidad de memoria, lo cual es más caro y no siempre merece la pena: si los datos ocupan toda la memoria Spark es la mejor opción, pero si no es el caso y se quiere que otros programas se ejecuten al mismo tiempo MapReduce sigue siendo preferible. En la plataforma Bluemix existe un servicio de *Apache Spark*.

3 Análisis

En este capítulo se realiza una primera descripción de las funcionalidades que se quieren implementar en la aplicación, para después detallar los roles de usuario que existirán y desarrollar un análisis de los requisitos funcionales y no funcionales.

3.1. Descripción general

En primer lugar, la aplicación debe ser capaz de leer en tiempo real un fichero de log de una máquina y enviar su información a donde sea necesario. De esto se encargará el programa *LogBlueAnalyzer Retriever*, que funcionará de manera local en la misma máquina en la que se encuentre el fichero. Otro componente (ya en la nube) de la aplicación se encargará de recoger y parsear esta información leída e introducirla en la base de datos.

De manera independiente habrá una interfaz web, *LogBlueAnalyzer Web*, que permitirá al usuario acceder a la información leída de sus logs en cualquier momento. Desde esta interfaz el usuario será capaz de crear gráficas personalizadas para visualizar los datos de un fichero log más fácilmente, crear filtros personalizados para seleccionar la información que le parezca más útil y establecer alertas personalizadas para que la aplicación le notifique vía e-mail de los eventos que solicite. Esta aplicación web se encontrará desplegada en la nube, al igual que la funcionalidad encargada de comprobar las alertas y enviar las notificaciones por correo.

3.2. Roles de usuario

En la aplicación completa se distinguirán solo dos tipos de usuario:

- **Usuario no registrado:** usuario cuya información no se encuentra en la base de datos. En la aplicación *Retriever* solo puede acceder a la pantalla de Login, y en la aplicación *Web* a la de Login y la de registro.
- **Usuario registrado:** usuario cuya información está en la base de datos y ha accedido a la aplicación mediante la pantalla de Login (en cualquiera de los dos programas) introduciendo su e-mail y su contraseña. Tiene acceso a todas las funcionalidades de la aplicación y a su información.

3.3. Requisitos

Para este proyecto se usará la siguiente nomenclatura para los requisitos: se denominarán RFW si son funcionales de LogBlueAnalyzer Web, RFR si son funcionales de LogBlueAnalyzer Retriever y RNF si son no funcionales (se aplican al programa completo), seguidos de un número identificativo. Se incluirá también el nombre, el rol del usuario al que se aplica en el caso de los funcionales y su descripción.

Los requisitos funcionales son aquellos que se refieren a una funcionalidad de la aplicación que puede utilizar un usuario. Los requisitos no funcionales se refieren a otras características que determinan la calidad del producto.

3.3.1. Requisitos funcionales

Requisito funcional 1	
ID	RFW01
Nombre	Registrarse
Rol del usuario	Usuario no registrado
Descripción	
Un usuario no registrado puede registrarse en la aplicación accediendo a la pantalla de registro e introduciendo sus datos.	

Requisito funcional 2	
ID	RFW02
Nombre	Acceder
Rol del usuario	Usuario registrado
Descripción	
Un usuario registrado puede acceder a la aplicación accediendo a la pantalla de Login e introduciendo sus datos.	

Requisito funcional 3	
ID	RFW03
Nombre	Seleccionar fichero
Rol del usuario	Usuario registrado
Descripción	
Un usuario registrado que haya accedido a la aplicación puede seleccionar un fichero subido por él en la pantalla de selección de ficheros.	

Requisito funcional 4	
ID	RFW04
Nombre	Eliminar fichero
Rol del usuario	Usuario registrado
Descripción	
Un usuario registrado que haya accedido a la aplicación puede eliminar un fichero subido por él desde la pantalla de selección de ficheros.	

Requisito funcional 5	
ID	RFW05
Nombre	Crear filtro
Rol del usuario	Usuario registrado
Descripción	
Un usuario registrado que haya accedido a la aplicación y que haya seleccionado un fichero puede crear un filtro desde la pantalla de panel de control.	

Requisito funcional 6	
ID	RFW06
Nombre	Eliminar filtro
Rol del usuario	Usuario registrado
Descripción	
Un usuario registrado que haya accedido a la aplicación y que haya seleccionado un fichero puede eliminar un filtro creado por él desde la pantalla de panel de control.	

Requisito funcional 7	
ID	RFW07
Nombre	Crear gráfica
Rol del usuario	Usuario registrado
Descripción	
Un usuario registrado que haya accedido a la aplicación y que haya seleccionado un fichero puede crear una gráfica desde la pantalla de panel de control.	

Requisito funcional 8	
ID	RFW08
Nombre	Crear alerta
Rol del usuario	Usuario registrado
Descripción	
Un usuario registrado que haya accedido a la aplicación y que haya seleccionado un fichero puede crear una alerta desde la pantalla de panel de control.	

Requisito funcional 9	
ID	RFW09
Nombre	Eliminar alerta
Rol del usuario	Usuario registrado
Descripción	
Un usuario registrado que haya accedido a la aplicación puede eliminar una alerta creada por él desde la pantalla de alertas.	

Requisito funcional 10	
ID	RFW10
Nombre	Descargar Retriever
Rol del usuario	Usuario registrado
Descripción	
Un usuario registrado que haya accedido a la aplicación puede descargar la aplicación Retriever desde la pantalla de descargas.	

Requisito funcional 11	
ID	RFW11
Nombre	Salir
Rol del usuario	Usuario registrado
Descripción	
Un usuario registrado que haya accedido a la aplicación puede salir de la aplicación pulsando el boton de Log out en cualquiera de las pantallas.	

Requisito funcional 12	
ID	RFR01
Nombre	Acceder
Rol del usuario	Usuario registrado
Descripción	
Un usuario registrado puede acceder a la aplicación accediendo a la pantalla de Login e introduciendo sus datos.	

Requisito funcional 13	
ID	RFR02
Nombre	Añadir fichero
Rol del usuario	Usuario registrado
Descripción	
Un usuario registrado que haya accedido a la aplicación puede añadir un fichero para analizar introduciendo su directorio y nombre completos y el nombre de la máquina en la pantalla de añadir ficheros.	

Requisito funcional 14	
ID	RFR03
Nombre	Finalizar lectura
Rol del usuario	Usuario registrado
Descripción	
Un usuario registrado que haya accedido a la aplicación y este leyendo un fichero puede terminar la lectura pulsado el botón Finish.	

Requisito funcional 15	
ID	RFR04
Nombre	Salir
Rol del usuario	Usuario registrado y no registrado
Descripción	
Un usuario puede cerrar la aplicación pulsando la cruz de cierre en cualquier pantalla.	

3.3.2. Requisitos no funcionales

Requisito no funcional 1	
ID	RNF01
Nombre	Escalabilidad
Descripción	
El diseño de la aplicación debe estar pensado de forma que sea fácil realizar cambios y añadir elementos sin que esto supongo un coste elevado ni repercuta en el funcionamiento.	

Requisito no funcional 2	
ID	RNF02
Nombre	Seguridad
Descripción	
Se debe tratar de preservar la confidencialidad, integridad y disponibilidad de la información de los usuarios.	

Requisito no funcional 3	
ID	RNF03
Nombre	Concurrencia
Descripción	
La aplicación podrá ser utilizada por varios usuarios a la vez.	

Requisito no funcional 4	
ID	RNF04
Nombre	Multiplataforma
Descripción	
La aplicación podrá ser utilizada en cualquier dispositivo a través de un navegador web.	

4

Diseño e implementación

En este capítulo se detalla el proceso de desarrollo de la aplicación *LogBlueAnalyzer*, creada como primera implementación funcional de un programa de análisis de logs completo que funcione sobre el entorno Bluemix. Para cada una de las partes de la aplicación se han empleado herramientas, lenguajes y enfoques diferentes:

- **Recolección:** desarrollada en Java y ejecutada en el ordenador local donde se encuentre el log a analizar.
- **Parseado:** desarrollada en Node.js y ejecutada en Bluemix.
- **Almacenamiento:** utiliza una base de datos DB2 alojada en Bluemix.
- **Visualización y filtrado:** desarrollada como una aplicación J2EE desplegada en Bluemix.
- **Alertas:** desarrollada en Node.js y ejecutada en Bluemix.

En el resto del capítulo se detallarán el diseño e implementación de estos componentes.

4.1. Recolección

Lo primero que debe hacer una aplicación de análisis de logs es obtener información de estos. Para ello se puede seguir dos enfoques distintos: leer el fichero completo cuando el usuario lo requiera, lo cual es computacionalmente menos costoso pero no es automático, o tener un proceso siempre ejecutándose y leyendo el fichero en tiempo real. Para esta aplicación se ha seguido la segunda filosofía por considerarla más adecuada para cumplir los objetivos propuestos.

Por lo tanto se ha implementado una aplicación en *Java* llamada *LogBlueAnalyzer Retriever*, cuya función es la de leer línea a línea el fichero especificado por el usuario y enviar la información al siguiente componente de *LogBlueAnalyzer* que se encargará del parseado. Mientras no se cierre la aplicación esta seguirá leyendo cualquier nuevo dato escrito en el fichero.

El protocolo de transporte empleado para enviar la información al parseador es *MQTT* [8], mecanismo comúnmente usado para la comunicación máquina-a-máquina en el llamado *Internet de las Cosas*, pero que se ha usado en esta aplicación puesto que consume muy poco ancho

de banda y es muy fácil de implementar. Posee una arquitectura de nodos, en la que un nodo central o *broker* es quién recibe y distribuye la información, que se diferencia mediante un *topic* que define el usuario. El broker utilizado en este caso es *tcp://test.mosquitto.org* y el topic *bluemixTrial4/data*. Este protocolo funciona sobre TCP, por lo que también permite la encriptación de los datos usando TLS/SSL.

LogBlueAnalyzer Retriever está diseñada siguiendo el patrón Modelo-Vista-Controlador, que se utiliza para separar la lógica de la aplicación de la interfaz gráfica con la que interactúa el usuario, dividiéndose en tres partes:

- **Modelo:** se encarga de la lógica de la aplicación, accediendo a la información cuando recibe la petición del Controlador, y envía a la Vista los datos que se requieran.
- **Vista:** representa la información que le envía el Modelo de forma adecuada para que el usuario interactúe con ella.
- **Controlador:** recibe las acciones del usuario y envía peticiones al Modelo cuando se requiere información, o a la Vista cuando se quiere cambiar algo en la representación.

A continuación se detalla el código Java que conforma la aplicación separado por paquetes.

com.uam.logblueanalyzerretriever.model

En este paquete se encuentran las clases que manejan los datos, desde la lectura del fichero hasta su envío:

- **File, Machine y User:** tipos de datos básicos usados para almacenar la información de ficheros, máquinas y usuarios.
- **DbHandler:** contiene los métodos para conectarse y desconectarse de la base de datos, así como para realizar cualquier operación con ella usando JDBC.
- **Logger:** contiene el método para iniciar sesión en la aplicación.
- **SenderMQTT:** métodos para conectarse y desconectarse del broker MQTT, y enviarle los datos.
- **Reader y ReaderTailerListener:** se encargan de leer el fichero usando una implementación del *TailerListener* de Apache, y de llamar a los métodos de *SenderMQTT*.
- **Session:** guarda los datos de la sesión en curso, es decir, los IDs del usuario, la máquina y el fichero, y el tipo de este último.

com.uam.logblueanalyzerretriever.view

Contiene las clases que implementan la interfaz gráfica, desarrollada con *Swing*. Cada pantalla cuenta con una clase que detalla la interfaz (GUI), una interface con los métodos a implementar en el controlador (Listener), la clase controlador como tal (Controller) y un clase principal que une Modelo, Vista y Controlador (Main). La aplicación consta de tres pantallas:

- **Login:** pantalla inicial, donde el usuario tiene que introducir su e-mail y contraseña (debe estar previamente registrado).

- **Read:** el usuario introduce el nombre de la máquina y nombre completo del fichero (directorio incluido) a leer.
- **Finish:** aparece mientras se está leyendo un fichero y permite terminar la lectura y cerrar el programa.

4.2. Parseado

La información recopilada desde las máquinas es enviada al subsistema de parseado, que está implementado en la plataforma Node-RED [9] en Bluemix, es decir, en la nube. Node-RED es una herramienta creada por IBM cuya finalidad es permitir la interconexión de diferentes funciones, APIs y servicios de manera fácil e intuitiva, organizándolas en los llamados flows (flujos) de nodos, cada uno de los cuales realiza una tarea independiente, y mostrándolos con una interfaz gráfica sencilla. La plataforma funciona sobre Node.js y por lo tanto ha sido necesario aprender JavaScript para desarrollar esta parte de la aplicación. Los nodos intercambian información mediante el objeto *msg*, que por lo general transmite el mensaje deseado en su campo *msg.payload*, así como mediante variables globales tanto para un flujo en concreto como para el entorno completo.

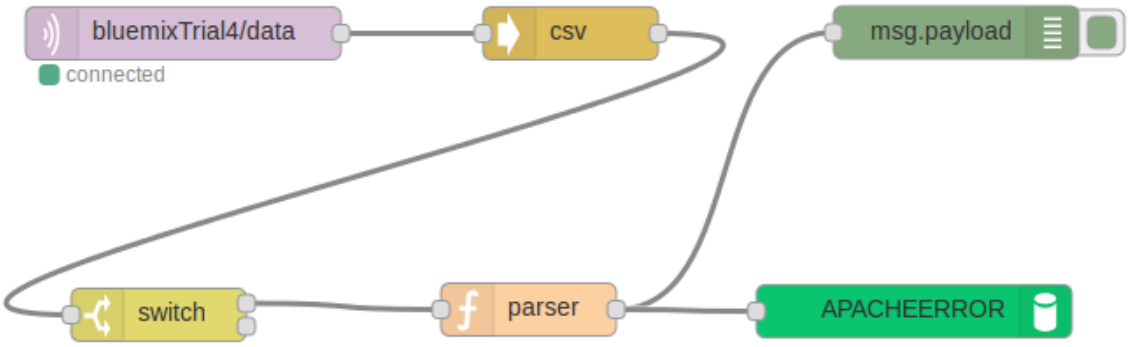

Figura 4.1: Flujo de nodos en Node-RED que realiza la tarea de parseado y almacenamiento.

En la figura 4.1 se puede ver el flujo de nodos que compone el subsistema, y que procede a explicarse detalladamente a continuación. El primer nodo que vemos (bluemixTrial4/data) es un receptor MQTT, que obtiene la información que va llegando al broker que se describió en la sección anterior y la envía al nodo siguiente en el msg.payload. Forma parte de los servicios ya existentes en la plataforma, solo teniendo que configurarlo añadiendo el servidor al que conectarse y el topic de MQTT.

La información enviada por MQTT consiste en una cadena que contiene un entero que identifica el tipo de fichero, el número identificador del fichero y la línea del log que se ha leído, los tres separados por comas. Debido a esto el siguiente nodo que encontramos es un parseador de formato CSV (Comma Separated Values), que nos genera un objeto de este tipo con elementos *type*, *file_id* y *data*, y que se envía como msg.payload al siguiente elemento del flujo. El siguiente nodo es un switch, que ramifica el flujo según el valor del msg.payload.type que se mencionaba antes. De esta manera, para cada tipo de fichero de log tenemos una función parseadora distinta, y se almacenan los datos en una tabla SQL diferente también, por lo que necesitamos que cada línea del mismo vaya a la rama que le corresponde, donde se realizarán las operaciones correspondientes a su tipo. Con este diseño es muy sencillo agregar y quitar tipos de ficheros que se pueden tratar según sea necesario: simplemente después del switch habría que

añadir o eliminar los nodos de *parser* y el de la base de datos para un tipo en concreto. En esta implementación se trabaja con un tipo de log, *ApacheError*, con identificador *1*.

La función de parseado en todos los casos realiza las mismas operaciones: primero parte la cadena de entrada en sus diferentes elementos usando una expresión regular, que será diferente según el tipo de log y su estructura; después transformará los datos que sean necesarios y por último generará un objeto JSON que contiene cada uno de estos elementos así como el nombre la columna SQL que ocuparán. Este JSON es enviado a un último nodo, de tipo *Input-dashDB* (también incluido por defecto en la plataforma) que se encarga de realizar una consulta de tipo INSERT de los datos incluidos en el JSON que le llega por msg.payload, en las columnas que se le indican en el mismo. Para configurar el nodo simplemente hay que seleccionar la base de datos que se quiera utilizar (debe a estar ligada a la aplicación Node-RED previamente) e indicar el nombre de la tabla donde se hará la inserción.

En el caso de los logs de tipo ApacheError cada línea contiene los siguientes elementos:

- **wday:** Tres letras que identifican el día de la semana en el que ocurrió la incidencia.
- **month:** Tres letras que identifican el mes en el que ocurrió la incidencia.
- **mday:** Día del mes en el que ocurrió la incidencia.
- **hour:** Hora (en formato 24h) en la que ocurrió la incidencia.
- **min:** Minuto en el que ocurrió la incidencia.
- **sec:** Segundo en el que ocurrió la incidencia.
- **year:** Año en el que ocurrió la incidencia.
- **severity:** Palabra que señala la gravedad de la incidencia.
- **id:** Texto y números que identifican al autor de la incidencia.
- **info:** Texto que aporta información no estructurada acerca de la incidencia.

Se utiliza la expresión regular */(\[(\w{3}) (\w{3}) (\d{2}) ((\d{2}):(\d{2}):(\d{2})) (\d{4})\]) (\[(\w+)\])((\[([0-9A-Za-z\:\._ \/\]*)\])*) ([0-9A-Za-z\:\._ \/\\(\)\-\,\;]+)/* para romper la línea en esos componentes (el mes se parsea a su equivalente en número). Además de ellos, en la tabla SQL también se incluirán el número identificador del fichero (id_file) y el timestamp en formato UNIX (en milisegundos) de la incidencia.

4.3. Almacenamiento

La parte más importante y que sostiene toda la aplicación es la de almacenamiento, siendo también el único cuello de botella de la misma debido a las limitaciones del servicio utilizado. También debido a que la plataforma se encuentra en desarrollo y sujeta a cambios ocasionales ha sido necesario migrar la base de datos empleada hasta en dos ocasiones.

En un primer momento la idea era aprovechar las herramientas para Big Data que ofrecía la PaaS, por lo que se decidió almacenar los datos en un HDFS, usando el servicio *IBM Analytics For Hadoop*, que proporcionaba una instancia de *IBM InfoSphere BigInsights Enterprise Edition 3.0*. Tras probar diversas formas de tratar y analizar los datos (JAQL, Hive) se optó por usar BigSQL [10] (figura 4.2), una funcionalidad ofrecida por BigInsights que permite crear tablas y realizar consultas como si de SQL normal se tratase, pero sobre Hadoop, por lo que emplea

MapReduce en las consultas cuando es apropiado. Además de la comodidad que ofrece el trabajar con SQL, el principal motivo para elegir esta opción es que se puede integrar en aplicaciones Java a través de JDBC muy fácilmente.

Para subir los archivos a HDFS se utilizaba una versión previa del *LogBlueAnalyzer Retriever* que hacía uso de la API REST de BigInsights, empleando un comando cURL (cliente para enviar o coger archivos mediante diversos protocolos) ejecutado en la terminal (a través de la aplicación Java). Esta no era la mejor forma de hacerlo puesto que requería que cURL estuviese instalado en el sistema. A continuación se muestra el método en Java para cargar el fichero:

```
public void loadFile (String filePath, String fileName) throws IOException,
    InterruptedException{

       //Elimina versiones antiguas del fichero
      String[] cmd = {"bash", "-c", "curl -X DELETE -L -b cookieJar
          \"https://bi-hadoop-prod-145.services.dal.bluemix.net:8443/data/
      controller/dfs/analyzer/files/" + fileName + "\""};
      Process p = new ProcessBuilder(cmd).redirectError(Redirect.INHERIT).
      redirectOutput(Redirect.INHERIT).start();
      p.waitFor();

      System.out.println("\n\nFile "+ fileName +" deleted.");

      //Carga el fichero
      String[] cmd2 = {"bash", "-c", "curl -X POST -L -b cookieJar
          \"https://bi-hadoop-prod-145.services.dal.bluemix.net:8443/data/controller/
      dfs/analyzer/files/" + fileName + "?op=CREATE&data=true\" --header
          \"Content-Type:application/octet-stream\" --header
          \"Transfer-Encoding:chunked\" -T " + filePath + fileName};
      Process p2 = new ProcessBuilder(cmd2).redirectError(Redirect.INHERIT).
      redirectOutput(Redirect.INHERIT).start();
      p2.waitFor();

      System.out.println("\n\nFile "+ fileName +" loaded.");
   }
```

Desgraciadamente, desde junio de 2015 este servicio se vio sometido a varios cambios, hasta su definitiva eliminación el 22 de febrero de 2016. Como reemplazo se procedió a usar el servicio SQLDB, que ofrecía una instancia de una base de datos SQL con motor DB2, sin funcionalidad Big Data esta vez pero con JDBC, por lo que a efectos prácticos solo hubo que cambiar credenciales y direcciones. En mayo de 2016 este servicio también fue removido de la plataforma, por lo que una vez más hubo que migrar la base de datos, esta vez al servicio dashDB, accesible también a través de JDBC y con motor DB2.

La base de datos está formada por cinco tablas básicas: *users*, *machines*, *files*, *filters* y *alerts*. Además, para cada tipo de fichero habrá una tabla separada, que en cada tupla guardará una línea de registro, con los campos que se han detallado en la sección 3.2 para cada caso.

La tabla *users* almacenará los usuarios registrados en la aplicación y consta de tres campos:

- **id:** Entero único que identifica a cada usuario.
- **name:** Cadena de 50 caracteres máximo con el e-mail del usuario. Es la clave primaria, por lo que solo puede haber un usuario por cada dirección de correo.
- **pass:** Cadena de 50 caracteres máximo con la contraseña del usuario.

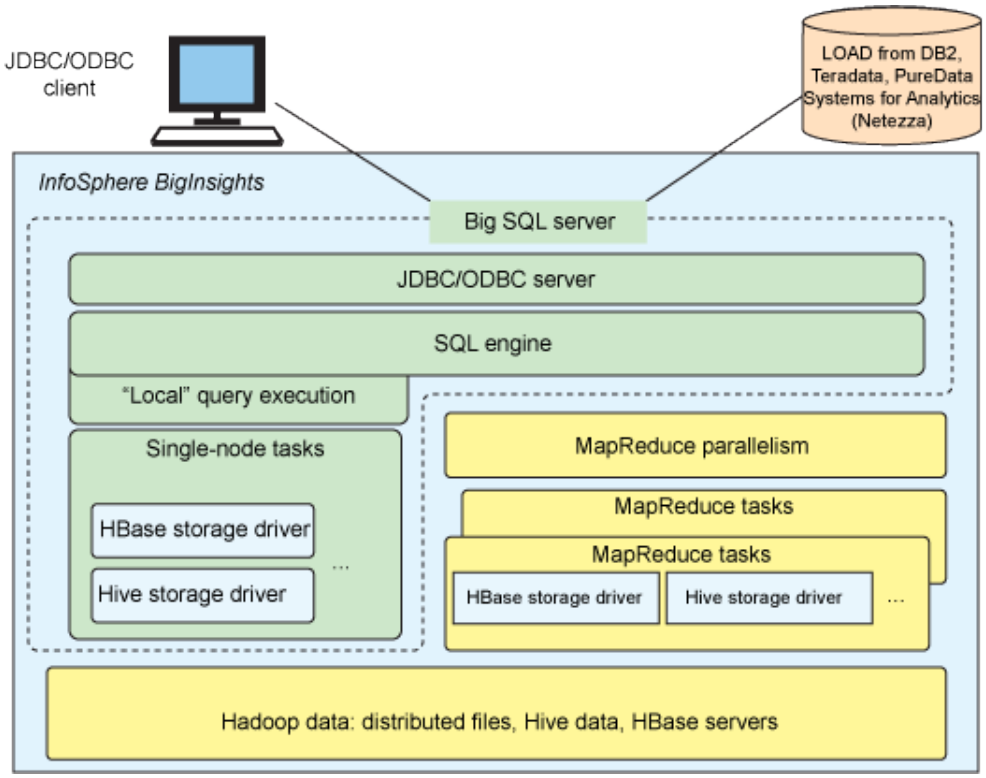

Figura 4.2: Arquitectura de BigSQL.

Cada usuario puede tener varias máquinas cuyos logs quiera analizar, y la información de estas será guardada en la tabla *machines*. Cada tupla de esta tabla tiene como clave primaria el identicador del usuario y el nombre de la máquina, es decir, solo se admite una máquina con el mismo nombre por cada usuario:

- **id:** Entero único que identifica a cada máquina.
- **name:** Cadena de 100 caracteres máximo con el nombre de la máquina.
- **id_user:** Entero con el identificador del usuario que es dueño de la máquina.

En cada máquina (entendiéndose "máquina" como el sistema cuyos logs se quieren analizar) puede haber varios ficheros de log que se están analizando, que se almacenarán en *files*, tabla que tiene como clave primaria el identificador de la máquina y el nombre de fichero, por lo que solo puede haber un fichero determinado por cada máquina:

- **id:** Entero único que identifica a cada fichero.
- **name:** Cadena de 500 caracteres máximo con el nombre completo del fichero (directorio y nombre como tal).
- **type:** Entero que señala el tipo del fichero.
- **id_machine:** Entero con el identificador de la máquina en la que se encuentra el fichero.

Para almacenar los filtros que se aplican a los ficheros se usa la tabla *filters*, que usa como clave primaria todos sus campos menos *id*, por lo que no puede haber dos filtros idénticos activos para el mismo fichero:

- **id:** Entero único que identifica a cada filtro.
- **id_file:** Entero con el identificador del fichero al que se aplica el filtro.
- **field:** Cadena de 50 caracteres con el nombre del campo a filtrar.
- **key:** Cadena de 50 caracteres con el valor que se usará para filtrar.
- **neg:** Entero que indica si el filtro es positivo (contiene/es igual) con valor 0, o negativo (no contiene/es distinto) con valor 1.
- **equals:** Entero que indica si la operación del filtro es de igualdad total (valor 1) o parcial (valor 0).

Para la gestión del sistema de alertas (ver sección 4.5) se cuenta con la tabla *alerts*, que cuenta con una clave primaria formada por *id_user*, *alert_query*, *time*, *comparator* y *threshold*, es decir, un usuario no podrá tener activa dos veces la misma alerta para el mismo intervalo de tiempo, pero puede tener todas las alarmas activas dsitintas que desee:

- **id:** Entero único que identifica a cada alerta.
- **id_user:** Entero con el identificador del usuario que activa la alerta.
- **alert_query:** Cadena de 200 caracteres con la consulta SQL de tipo COUNT que se usará para obtener el número de eventos en un fichero con unos filtros activos determinados.
- **time:** Entero que indica el intervalo de tiempo en el que comprobar las condiciones de la alerta (en minutos).
- **threshold:** Entero que indica el número de eventos límite.
- **comparator:** Entero que indica si se activa la alerta al no alcanzar el número de eventos un mínimo (valor -1), al ser igual a un número determinado (valor 0) o al ser mayor que un número máximo (valor 1).
- **msg:** Cadena de 700 caracteres con el texto que se enviará al usuario en caso de que se active la alerta.

Por último, la información leída y parseada de cada log se guardará en diferentes tablas según el tipo del fichero, cada una con los campos detallados en la sección anterior.

4.4. Filtrado y visualización

Con el fin de poder visualizar y manipular los datos almacenados en la base de datos de forma sencilla se dispone de una aplicación web, *LogBlueAnalyzer Web*, desarrollada sobre J2EE [11], que combina programación en Java con páginas JSP escritas en HTML y JavaScript. Se encuentra desplegada sobre Bluemix, usando el *Java Buildpack* que la plataforma proporciona, que permite desplegar aplicaciones web en Java compiladas en un archivo WAR, de la misma manera que se haría con un servidor Tomcat, con la ventaja de que dispone automáticamente de una dirección URL a través de la cual se puede acceder desde cualquier navegador en cualquier dispositivo.

En el desarrollo de esta aplicación se han utilizado los lenguajes de programación *Java*, *JavaScript* y *HTML*, así como otros tipos de lenguajes para la configuración (*XML*) y el diseño

de la interfaz (*CSS*). Además se ha empleado la herramienta *Maven* para la gestión de las dependencias, el framework *Spring Web MVC* para desarrollar la funcionalidad de la aplicación en si y las APIs *Google Charts* y *Datatables* para la visualización de los datos. En las siguientes secciones se explican en detalle.

4.4.1. Maven

Por lo general, en cualquier proyecto de programación tenemos la necesidad de usar ciertas librerías externas para importar alguna funcionalidad, que a su vez pueden depender de otras o puede que se requiera de una versión concreta de alguna de ellas. Si el proyecto es de cierta envergadura, el administrar estas librerías correctamente puede ser una tarea complicada.

Para solventar este problema se creó Maven [12] en 2002, una herramienta de gestión de software similar a Apache Ant, pero basado en XML y más simple de usar. Utiliza un archivo POM para definir la configuración del proyecto y sus dependencias a artefactos (objetos que contienen una librería determinada así como información adicional, como la versión o sus dependencias, entre otras cosas), que se descargarán de su repositorio. En este fichero también se puede especificar objetivos para la compilación y despliegue de la aplicación. Aunque no es obligatorio configurarlo así, existe un ciclo de vida específico para los proyectos Maven:

- **Compile:** se crean los ficheros *.class* a partir de los *.java*.
- **Test:** si existen, se ejecutan los test *jUnit* y en caso de fallo en alguno se termina el proceso.
- **Package:** se empaqueta la aplicación compilada en un archivo JAR o WAR.
- **Install:** se copia el paquete a un directorio donde Maven guarda todos los paquetes.
- **Deploy:** se copia el paquete a un servidor remoto donde otras aplicaciones Maven pueden tener acceso a él.

4.4.2. Spring Web MVC Framework

De la misma forma en la que se emplean numerosas librerías en un desarrollo, muchas veces también se usan varios frameworks, cada uno con objetos y gestiones del ciclo de vida de los mismos independientes. Spring Framework [13][14] sirve para liberar al desarrollador de tener que crear los objetos y gestionarlos, encargándose él de esta tarea según lo especificado en ficheros XML y anotaciones en el código, iniciándolos e integrándolos de manera correcta. Este patrón de diseño en el que se suministran los objetos a una clase en vez de que esta los cree se denomina *Inyección de Dependencias*.

Spring posee diversos módulos para ofrecer distintos servicios, y para este trabajo se ha usado en concreto Spring Web MVC Framework, un módulo basado en HTTP y Servlets y que, como el propio nombre indica, sirve para el desarrollo de aplicaciones web. De la misma manera, en el nombre también se indica que este framework sigue el patrón Modelo-Vista-Controlador.

El elemento principal de este framework es el *DispatcherServlet*, que se encarga de las peticiones y respuestas HTTP. En la figura 4.3 se puede observar el proceso de actuación de este componente: cuando llega una petición HTTP la envía al *HandlerMapping*, que busca al Controlador adecuado para atenderla; el Controlador llama a los métodos del Modelo correspondientes para atender la petición, y el Servlet busca la Vista adecuada para la operación; por último envía la información del Modelo a la Vista escogida, que se muestra en el navegador.

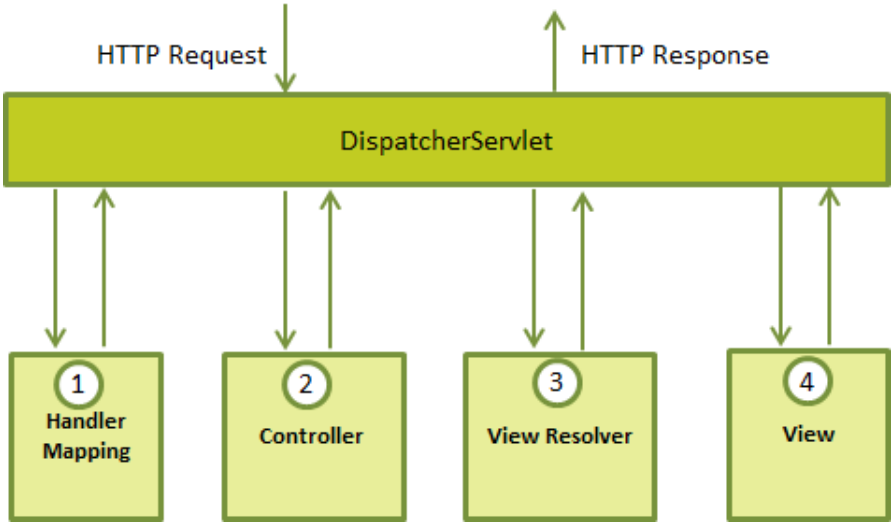

Figura 4.3: Diagrama con el funcionamiento del DispatcherServlet.

```
<display-name>LogAnalyzerWeb2</display-name>
<welcome-file-list>
  <welcome-file>index.html</welcome-file>
</welcome-file-list>

  <servlet>
      <servlet-name>analyzer</servlet-name>
      <servlet-class>
          org.springframework.web.servlet.DispatcherServlet
      </servlet-class>
      <load-on-startup>1</load-on-startup>
  </servlet>


  <servlet-mapping>
      <servlet-name>analyzer</servlet-name>
      <url-pattern>/login2</url-pattern>
      <url-pattern>/login</url-pattern>
```

Figura 4.4: Fragmento de *web.xml*.

```
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd">

<context:component-scan base-package="com.analyzer.controller" />

<bean id="viewResolver"
    class="org.springframework.web.servlet.view.UrlBasedViewResolver">
    <property name="viewClass"
        value="org.springframework.web.servlet.view.JstlView" />
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
</bean>



<mvc:resources mapping="/resources/**" location="/resources/" />
<mvc:annotation-driven />
```

Figura 4.5: Fragmento de *analyzer-servlet.xml*.

Dos ficheros importantes para la configuración de la aplicación usando este framework son el *web.xml* (figura 4.4), que indica que se usará el *DispatcherServlet* y las direcciones que puede mapear, y el *analyzer-servlet.xml* (figura 4.5), en el que se configura el *ViewResolver*, indicándole en que paquete debe buscar los Controladores y que tipo de ficheros pueden ser Vistas.

A continuación se detalla el código fuente de la aplicación Java divido por paquetes:

com.uam.logblueanalyzerweb.types

En este paquete se encuentran los tipos de datos básicos que usará la aplicación para manejar la información de los usuarios, ficheros, filtros y eventos de los logs.

com.uam.logblueanalyzerweb.model

Contiene la clase *DbHandler*, que se encarga de trabajar con la base de datos a petición de los controladores. Posee métodos para conectarse y desconectarse de la base de datos, así como para realizar consultas y operaciones de añadir y eliminar información.

Todos estos métodos implementan funcionalidades de la librería *java.sql* ofrecidos por la API *JDBC*, que permite realizar conexiones a bases de datos SQL en entornos Java. Para ello

se necesitan las credenciales del servicio (nombre de usuario, URL, contraseña y nombre de la base de datos) que nos proporciona Bluemix.

com.uam.logblueanalyzerweb.controller

Paquete que contiene las clases que actúan como controladores. Estas clases están marcadas con la etiqueta *@Controller* para que el ViewResolver antes mencionado sepa identificarlas. La mayoría de los métodos de estas clases están marcados con la etiqueta *@RequestMapping*, que incluye la dirección de mapeo a la que responden, y devuelven un objeto de tipo *ModelAndView*, que incluye la vista a utilizar y los parámetros que se le pasan.

Las vistas de la aplicación se corresponden con los ficheros JSP escritos en HTML y JavaScript.

4.4.3. Visualización de los datos

El objetivo principal de la aplicación es que el usuario pueda visualizar los datos leídos de los logs fácilmente, y que esto le permita localizar errores e incidencias en el funcionamiento de su sistema, o simplemente obtener un análisis del mismo. Para ello se utilizan dos APIs en la interfaz web, una que representará la información en tablas y la otra que hará lo propio pero en forma de gráfica.

La primera es *Datatables* [15] (figura 4.6), un plugin jQuery que genera tablas interactivas a partir de la información que se le pasa en un array de JSONs (cada uno de los cuales equivale a una tupla). Debido a que hay que indicar las columnas concretas que tendrá la tabla en el código JavaScript habrá un fichero JSP específico para generar la tabla correspondiente a cada tipo de fichero.

Show 10 entries Search:

Weekday	Day	Month	Year	Hour	Min	Sec	Severity	ID	Info
Sat	9	5	2016	23	15	42	error	client 179.44.108.71	File does not exist: /opt/web-ii/favicon.ico
Sat	9	5	2016	23	23	53	error	client 200.48.178.231	client denied by server configuration: /home/siguenza/public_html
Sat	9	5	2016	23	28	7	error	client 201.255.15.194	File does not exist: /opt/web-ii/favicon.ico
Sat	9	5	2016	23	36	36	error	client 89.248.160.217	File does not exist: /opt/web-ii/FCKeditor

Figura 4.6: Ejemplo de tabla creada con *Datatables*.

Para la representación de gráficas se usa *Google Charts* [16], que apenas requiere importar unas librerías y escribir unas pocas líneas en JavaScript. Los datos a representar le llegan desde el Modelo en formato JSON, por lo que serán parseados en un array, para que la API los pueda manejar correctamente. Podemos ver un ejemplo de gráfica en la figura 4.7.

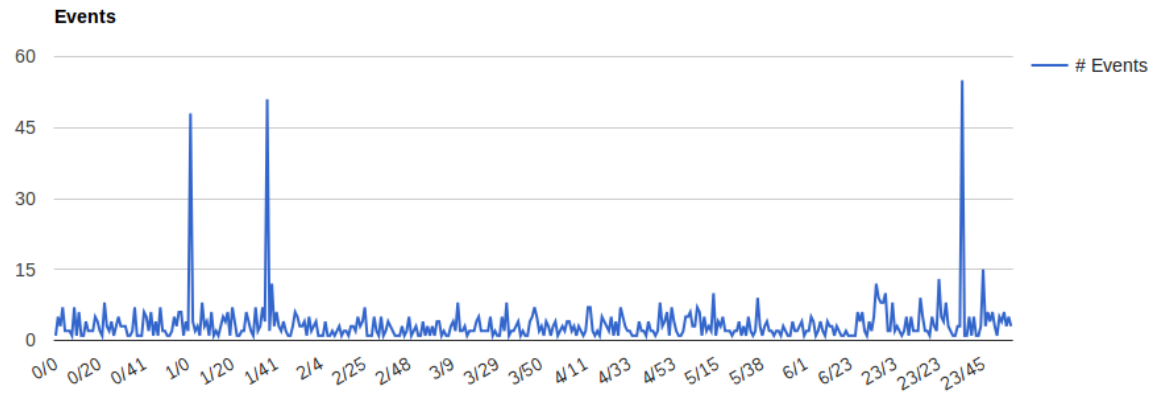

Figura 4.7: Ejemplo de gráfica creada con *Google Charts*. El eje Y representa el número de eventos mientras que el eje X los valores que haya seleccionado el usuario, en este caso horas/minutos.

4.5. Alertas

La aplicación incorpora un sistema de alertas que notificará al usuario en caso de que se registre un cierto número de incidencias de determinado tipo en un espacio de tiempo concreto. Desde el entorno web el usuario puede establecer una alerta para un intervalo de tiempo, señalando el número de eventos límite y si la cantidad de incidencias en el intervalo debe ser menor, igual o mayor para que se active. Se aplicarán los filtros activos en el momento de creación de la alerta. En el momento en el que es creada se guardarán en la base de datos los elementos que se especificaron en la sección 4.3 en la tabla *alerts*, incluyendo la consulta SQL con formato:

```
SELECT count(*) as n FROM errorTable WHERE filters AND (? - timestamp < ?)
```

La funcionalidad encargada de comprobar si se deben activar las alertas y notificar al usuario está implementada en Node-RED nuevamente, existiendo una rama diferente para cada intervalo de tiempo a comprobar. En la figura 4.8 se muestra la rama correspondiente a las alertas que se comprueban cada diez minutos.

Figura 4.8: Rama de Node-RED que comprueba las alertas cada 10 minutos.

El primer nodo se encarga de activar la rama en el intervalo de tiempo correspondiente, insertando el timestamp en milisegundos (que es guardado como variable del flujo en la función *storeTimestamp*) en los siguientes nodos. El siguiente nodo se encarga de buscar en la base de datos todas las alarmas que se tienen comprobar en ese tiempo con sus usuarios, y se envían como JSONs a la siguiente función, *parseData*, que las transformará en un array que mandará a *setParams*. Este nodo se encarga de, para cada alarma, guardar el límite de incidencias, la operación a realizar (menor que, igual, mayor que), el e-mail del usuario y el texto de notificación como variables del flujo; guardar la *alert_query* en msg.payload; recuperar el timestamp guardado como variable del flujo y meterlo en msg.param1 y guardar en msg.param2 el tiempo del intervalo de activación de la alerta en milisegundos.

El siguiente nodo vuelve a ser un nodo de base de datos, que esta vez ejecutará la consulta que

le llega como payload, sustituyendo las interrogaciones que se mencionaban anteriormente por param1 y param2. Esto significa que solo se tomarán los eventos que cumplan que la diferencia de tiempo entre el momento en el que se comprueba la alerta y en el que ocurrió el evento sea menor al intervalo que se está comprobando, es decir, solo se tienen en cuenta los eventos sucedidos tras la última comprobación.

Cuando ya se tiene el número de eventos, la función *prepareMail* recupera las variables almacenadas como globales y comprueba si la condición de activación se ha dado y, en caso afirmativo, establece el mensaje de notificación en msg.payload, el e-mail del usuario en msg.to y *LogAnalyzer alert activated* como msg.topic. El último nodo enviará un correo al destinatario que le llega por msg.to, con asunto msg.topic y cuerpo msg.payload, usando el server *smtp.gmail.com* y desde la dirección que le indiquemos.

5

Funcionamiento y utilización

En este capítulo se detalla un breve manual de uso de las dos partes de la aplicación general que son utilizadas interactivamente por el usuario, *LogBlueAnalyzer Retriever* y *LogBlueAnalyzer Web*.

A nivel de interacción usuario-aplicación existen tres partes bien diferenciadas en el programa. Por un lado, el usuario puede tener tantas instancias de *LogBlueAnalyzer Retriever* ejecutándose para leer ficheros como desee, en una máquina o en varias. La información leída se enviará al parseador en la nube, que la introducirá en la base de datos.

Asimismo, en cualquier momento el usuario puede acceder a *LogBlueAnalyzer Web* desde un navegador para consultar la información que tiene guardada en la base de datos. Si en ese momento se encuentra leyendo algún fichero con el Retriever su información estará disponible en tiempo real. Desde esta interfaz también se pueden definir las alertas a comprobar.

Por último, de manera independiente y automática se comprobarán las alertas definidas cada cierto tiempo, y se enviará un correo electrónico de notificación al usuario cuando proceda.

5.1. LogBlueAnalyzer Web

Para utilizar la aplicación web primero hay que realizar unos pasos de configuración (registrarse, acceder, seleccionar fichero) antes de poder usar las utilidades que ofrece (aplicar filtros, crear gráficas, activar alertas).

5.1.1. Configuración

Acceso a la aplicación

Para abrir la aplicación debemos acceder a la URL *http://loganalyzerweb2.mybluemix.net*, que nos llevará a la pantalla de *Login* (figura 5.1) si no hay ninguna sesión iniciada, o al *Panel de Control (Dashboard)* (figura 5.3) si la hay. Si es la primera vez que usamos la aplicación necesitaremos registrarnos en ella, para lo que debemos pulsar el enlace a *Register* (figura 5.2) y rellenar el formulario con nuestro usuario (dirección de e-mail) y contraseña (de más de ocho

caracteres). Si ya tenemos un usuario creado simplemente debemos introducir el e-mail y la contraseña en la pantalla de *Login* y pulsar el botón.

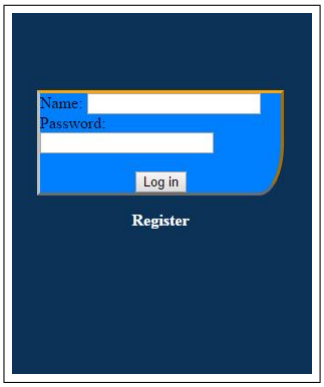

Figura 5.1: Pantalla de *Login* .

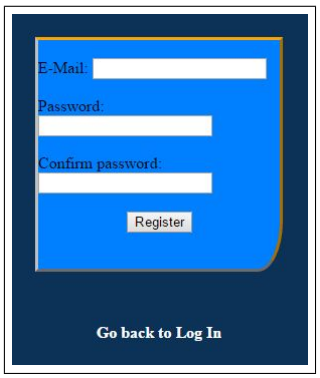

Figura 5.2: Pantalla de *Register*.

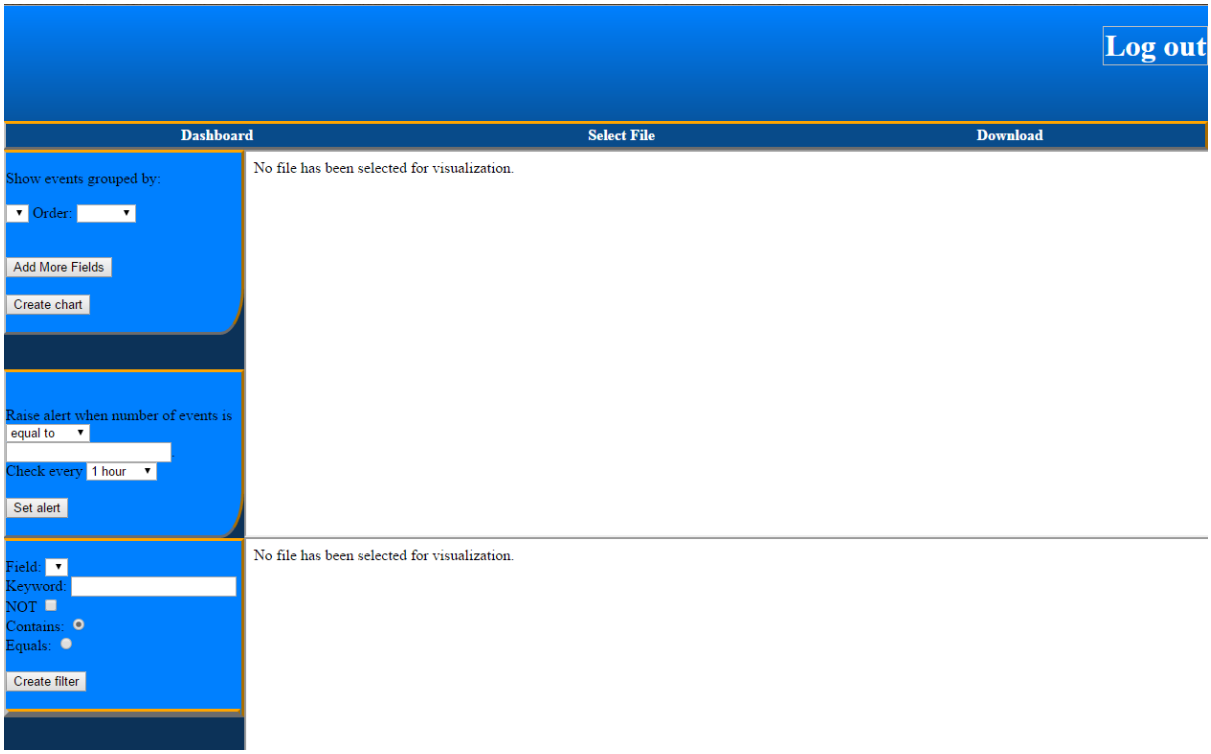

Figura 5.3: Pantalla de *Dashboard* inicial.

Seleccionar archivo

En la pantalla de *Dashboard* inicial no se muestra ninguna gráfica ni tabla, y los menús desplegables usados para crear gráficas y filtros se encuentran vacíos, puesto que primero hay que seleccionar el archivo a visualizar. Para hacer esto hay que ir a la pantalla de *Select File*, pulsando en su correspondiente enlace en el menú superior.

En esta ventana aparecerá un listado con todos los ficheros de log que tengamos cargados y el nombre de las máquinas en las que se encuentran. Para seleccionar un archivo solo hay que hacer click en su nombre. También podremos eliminar un fichero pulsando el botón *Remove*, borrando toda información del mismo de la base de datos. Una vez seleccionado un fichero podremos ver

una tabla con los datos en *Dashboard*, y habrán aparecido elementos en los menús desplegables para crear gráficas y filtros.

Cerrar sesión

Desde cualquiera de las pantallas se puede cerrar la sesión pulsando en *Log out*, en la esquina superior derecha.

5.1.2. Utilidades

Filtros

Una vez hayamos seleccionado un fichero podremos aplicar filtros para facilitarnos la visualización de los datos y quedarnos solo con aquello que consideremos relevante. Los filtros activos se aplican por fichero y se guardan en la base de datos, por lo que podremos cambiar de fichero o cerrar la aplicación sin perderlos, siendo necesario eliminarlos manualmente. En el lateral izquierdo de la pantalla de *Dashboard* existen tres menús: el inferior es el usado para crear filtros (figura 5.4). Para crear un filtro es necesario rellenar todos los campos y pulsar el botón *Create filter*:

- **Field:** menú desplegable que contiene todos los campos en los que se divide una entrada del log y que varían en función del tipo del mismo.
- **Keyword:** palabra, símbolo o número que se usará como clave para filtrar.
- **Not:** si se marca está casilla se aplicará el filtro de forma negativa, es decir, se buscarán eventos cuyos campos a filtrar *sean distintos* o *no contengan* la palabra clave.
- **Contains/Equals:** si se selecciona *Contains* el campo a filtrar debe contener el elemento clave, pero si se selecciona *Equals* este debe ser 100 % igual a la clave.

Por ejemplo, en la figura 5.4 podemos ver que hay dos filtros activos sobre los campos *hour* e *info*, y que se va a crear uno para que *Weekday* sea igual a "Sat". Para eliminar un filtro hay que pulsar el botón *Remove* junto a su descripción. Los filtros se aplican a las tablas y gráficas que se muestran, así como a las alarmas que se creen mientras esten activos. Podemos tener tantos filtros activos simultáneamente como queramos.

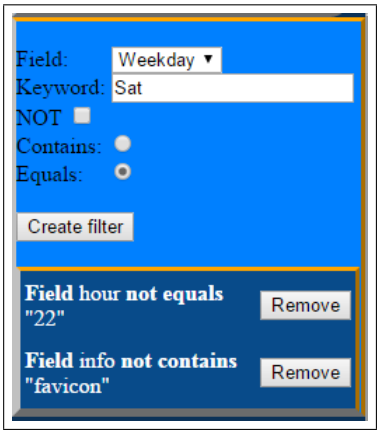

Figura 5.4: Menú de *Filtros*.

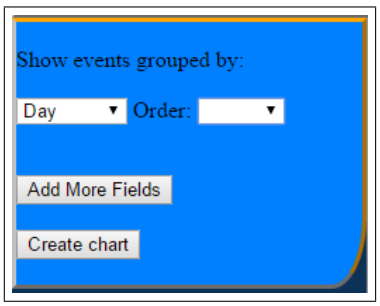

Figura 5.5: Menú de *Gráficas*.

Gráficas

Teniendo un archivo seleccionado podemos crear gráficas que se mostrarán en la pantalla de *Dashboard* (solo una simultanea). Para ello emplearemos el menú superior de los que aparecen en el margen izquierdo de esta pantalla (figura 5.5). Estas gráficas mostrarán el número de eventos ocurridos agrupados según los parámetros que le indiquemos, aplicándose los filtros activos en ese momento. Inicialmente solo habrá uno, pero se pueden añadir más pulsado el botón *Add More Fields*, y eliminarlos pulsando la *X* junto a ellos. Una vez seleccionados los campos y el orden con el que queremos que se aparezcan pulsamos el botón *Create*.

Cuando se realiza la agrupación por campos podemos indicarle si queremos algún orden en concreto (de menor a mayor o de mayor a menor) o dejar el campo en blanco para que los vaya agrupando según vayan apareciendo en la tabla. Por ejemplo, si por algún motivo en nuestro fichero aparecen primero eventos de 2015 y después eventos de 2014, pero queremos que en la gráfica aparezcan agrupados por año y en orden cronológico, debemos indicar que sea de menor a mayor, puesto que si lo dejamos en blanco saldrán primero los de 2015 y luego los de 2014.

El orden en el que se añaden los parámetros por los que agrupar es relevante, puesto que cada uno se va agrupando sobre las agrupaciones ya realizadas. Por ejemplo, si seleccionamos los campos *Day*, *Month* y *Year*, todos ellos ordenados de menor a mayor, primero se agruparán por día del mes (1, 2, 3, etc), luego sobre esos números se agruparán por meses (1/Jan, 1/Feb,..., 2/Jan, etc) y por último por año (1/Jan/2015, 1/Jan/2016, 2/Jan/2015, etc). Es decir, con esa distribución, en nuestro eje horizontal de la gráfica nos aparecería el 1/Jan/2016 y después el 2/Jan/2015. Si lo que queremos es ordenar cronológicamente por fecha habría que seleccionar primero el año (2015, 2016), luego el mes (2015/Jan, 2015/Feb, ..., 2016/Jan, etc) y por último el día (2015/Jan/1, 2015/Jan/2, etc). Un ejemplo de gráfica con esta última configuración se muestra en la figura 5.6.

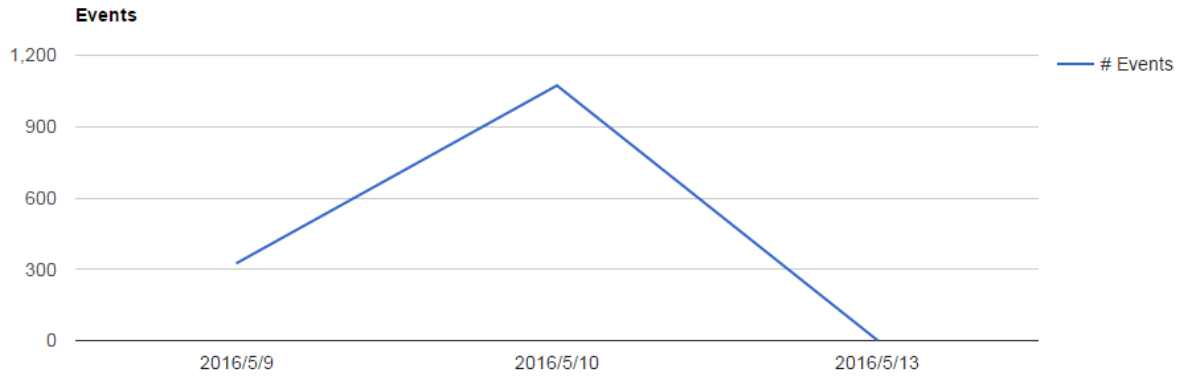

Figura 5.6: Ejemplo de gráfica. En el eje Y se muestra el número de eventos y en el eje X el Año/Mes/Día

Alertas

Se pueden establecer alertas sobre un fichero para que la aplicación nos avise vía e-mail de si ha ocurrido algo de interés. Estas alertas se activan usando el menú intermedio de los que aparecen en el lateral izquierdo de la pantalla de *Dashboard* (figura 5.7). Simplemente hay que indicar que queremos que se nos avise cuando el número de eventos sea mayor, igual o menor que el número que le especifiquemos, y cada cuanto tiempo debe comprobarlo. Los filtros activos en el momento de creación de la alerta se aplicarán, incluso cuando estos hayan sido eliminados.

Se puede comprobar que alertas se encuentran activas en la pantalla *Alerts*, a la cual se accede mediante su correspondiente enlace en el menú superior, y en la cual podemos eliminar las que ya no nos interesen.

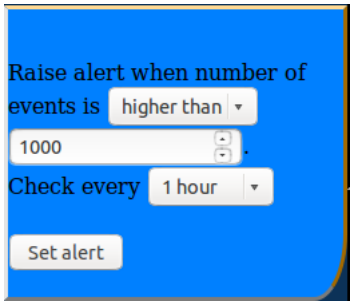

Figura 5.7: Menú de *Alertas*.

Descargar *LogBlueAnalyzer Retriever*

El usuario puede descargar la aplicación *LogBlueAnalyzer Retriever* desde la pantalla de *Download* pulsando en el correspondiente enlace.

5.2. LogBlueAnalyzer Retriever

Con *LogBlueAnalyzer Retriever* solo es necesario un paso de configuración (acceso) para utilizar su funcionalidad (leer fichero).

5.2.1. Configuración

Acceso a la aplicación

Primero hay que ejecutar la aplicación Java y una vez en la pantalla de *Login* (figura 5.8) introducir el e-mail y la contraseña (debe estar registrado previamente).

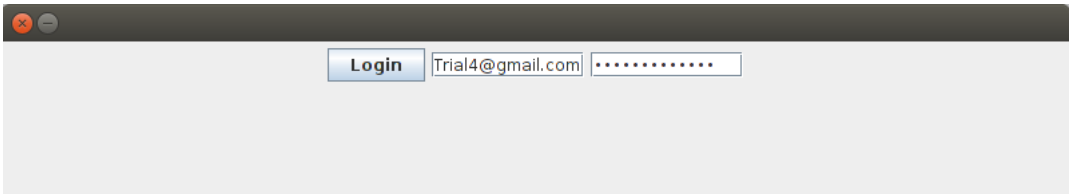

Figura 5.8: Pantalla de *Login* de *LogBlueAnalyzer Retriever*.

5.2.2. Utilidades

Leer fichero

Una vez se ha accedido a la aplicación aparece la pantalla de *Leer Fichero* (figura 5.9), en la que hay que introducir el nombre completo del archivo (directorio y nombre), el nombre de la máquina y seleccionar el tipo de fichero en el menú desplegable. Una vez hecho esto se pulsa en botón, se comenzará a leer el log y se mostrará la pantalla de *Finalizar*, que muestra un único botón que al ser pulsado termina de leer el archivo y cierra la aplicación.

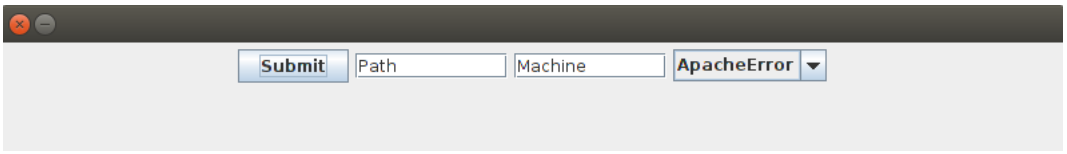

Figura 5.9: Pantalla de *Leer Fichero* de *LogBlueAnalyzer Retriever*.

6

Conclusiones y trabajo futuro

6.1. Conclusiones

El primer objetivo de este trabajo era desarrollar una aplicación, *LogBlueAnalyzer*, que permitiese el análisis de logs, implementando todas las funcionalidades necesarias, separadas en distintos módulos, desde la recopilación de datos hasta las notificaciones. Esto se ha realizado satisfactoriamente gracias al uso de diversas herramientas y lenguajes.

Se ha logrado también el otro objetivo principal, que era el de desplegar la aplicación en IBM Bluemix: todas las partes de la misma, menos el Retriever, se encuentran en la nube. Además de esto, también se cumplen los requisitos de que pueda ser utilizada por varios usuarios simultáneamente, que sea fácil de usar, que pueda ser accedida desde cualquier plataforma y que se puedan añadir funcionalidades fácilmente.

Señalar que esta implementación funciona sobre servicios gratuitos de Bluemix, aunque sea en sus versiones de prueba. La única limitación que esto supone es el tamaño de almacenamiento en la base de datos, de 1GB máximo, costando 50 dólares mensuales ampliar a 20GB.

Para la recolección de datos se ha implementado una aplicación Java que se encarga de leer un registro línea a línea en tiempo real y enviarlas, junto con la información del usuario, al componente encargado del parseado usando el protocolo de transporte MQTT. El parseado lo realiza una aplicación implementada en Node.js, utilizando el entorno Node-RED. A través de una serie de nodos y con la información contenida en el mensaje MQTT se determina el tipo de fichero que se ha leído y, mediante el uso de una expresión regular, se divide la línea del log en sus componentes básicos y se almacena en la base de datos.

El almacenamiento se realiza en una base de datos DB2 proporcionada por el servicio dashDB, tras una primera implementación sobre HDFS que fue descartada por la eliminación del servicio de Hadoop. Para poder visualizar y filtrar cómodamente la información obtenida de los logs se utiliza una aplicación web J2EE, que usa los frameworks Maven y Spring Web MVC para su gestión e implementación. Se emplea código Java para las clases que cumplen los roles de Modelos y Controladores y páginas JSP con código HTML y JavaScript para las Vistas. Se emplean las APIs de JavaScript Google Charts y Datatables para la creación de gráficas y tablas respectivamente. Desde la interfaz web anterior se pueden crear alertas que se comprueban cada un tiempo determinado; si durante ese tiempo se dan unas condiciones especificadas por el

usuario se le envía un correo electrónico para notificarle. Esta funcionalidad está implementada en Node.js usando Node-RED también.

6.2. Trabajo futuro

Debido a que esta es una primera versión de la aplicación existen varias funcionalidades que añadir en el futuro, como el hacer que *LogBlueAnalyzer Retriever* se ejecute en segundo plano mientras lee un fichero. Se podría mejorar la parte de visualización para que ofreciese una mayor variedad de gráficas (de barras, circulares, etc) y con más opciones.

Uno de los objetivos de la aplicación era que, aunque en esta primera versión solo se analizase un tipo de log, debía ser fácil añadir más posteriormente. Podrían añadirse más tipos predeterminados de logs (ApacheAccess, Syslog, errores de aplicaciones, etc) e incluso que el usuario cree sus tipos personalizados e indique que campos quiere extraer y almacenar.

Una vez desarrollada la aplicación hay que evaluar su utilidad en entornos reales, bajo los límites de no pago que marque la plataforma Bluemix.

El requisito no funcional de Seguridad queda pendiente para su implementación futura. El envío de datos mediante MQTT puede cifrarse mediante SSL/TLS, lo cual es sencillo pero también obligaría a instalar certificados. También deberían tomarse medidas para evitar ataques tipo SQLInjection.

Por último, un campo muy interesante pero bastante complejo es el de predicción de eventos a partir del análisis de logs. Para esta tarea se requieren técnicas de Aprendizaje Automático y se podría intentar aprovechar los servicios de Watson que ofrece Bluemix para la computación cognitiva.

Glosario de acrónimos

- **J2EE**: Java Enterprise Edition
- **JSP**: JavaScript Pages
- **POM**: Project Object Model
- **API**: Application Programming Interface
- **WAR**: Web Application Archive
- **JAR**: Java Archive
- **REST**: Representational State Transfer
- **JSON**: JavaScript Object Notation
- **SQL**: Structured Query Language
- **CSV**: Comma Separated Values
- **XML**: eXtensible Markup Language
- **HDFS**: Hadoop Distributed File System
- **MQTT**: MQ Telemetry Transport
- **TLS**: Transport Layer Security
- **SSL**: Secure Sockets Layer
- **TCP**: Transmission Control Protocol
- **JDBC**: Java Database Connectivity
- **GUI**: Graphical User Interface
- **HTML**: HyperText Markup Language
- **URL**: Uniform Resource Locator
- **CSS**: Cascading Style Sheets
- **JVM**: Java Virtual Machine

Bibliografía

[1] Real Academia de Ingeniería. Computación en la nube. *Diccionario Español de Ingeniería*, 2014.

[2] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. *NIST Special Publication 800-145*, 2011.

[3] Anju Chhibber and Dr. Sunil Batra. Security Analysis of Cloud Computing. *International Journal of Advanced Research in Engineering and Applied Sciences*, 2015.

[4] Bobbie Johnson. Cloud computing is a trap, warns GNU founder Richard Stallman. *Guardian*, 2008.

[5] José Francisco Aldana Montes. *Introducción al Big Data*. García Maroto, 2016.

[6] Tom White. *Hadoop: The definitive guide*. O'Reilly Media, 2011.

[7] Mario Pérez Esteso. Apache Spark: qué es y cómo funciona. *Geeky Theory*, 2015.

[8] Andy Piper. *MQTT Wiki*, 2015.

[9] IBM Emerging Technologies. *Node-RED Documentation*, 2016.

[10] Cynthia M. Saracco and Uttam Jain. What's the big deal about Big SQL? *IBM developerWorks*, 2013.

[11] Abraham Otero. *Tutorial básico de Java EE*, 2010.

[12] Sonatype Company. *Maven: The Definitive Guide*. O'Reilly Media, 2008.

[13] Cecilio Álvarez. ¿Qué es Spring Framework? *Genbeta:dev*, 2014.

[14] Craig Walls. *Spring in action*. Manning, 2008.

[15] SpryMedia. *Datatables Manual*, 2016.

[16] Google. *Google Charts Docs*, 2016.

A

Experiencia con IBM Bluemix

Parte importante del desarrollo de este proyecto ha consistido en estudiar las posibilidades que ofrece la plataforma cloud *IBM Bluemix* para así determinar la mejor manera de implementar la aplicación de análisis de logs. En este anexo se explicará la experiencia que se ha tenido con la PaaS y sus servicios, algunos de los cuales han sido utilizados en la aplicación final y otros no.

Uno de los servicios que más se utilizaron fue *Analytics for Apache Hadoop* (eliminado de la plataforma el 22 de febrero de 2016), que proporcionaba una instancia de *IBM InfoSphere BigInsights*, una plataforma para trabajar con Big Data. En ella podíamos acceder a tres servicios:

- **BigSheets:** permite manejar la información mediante los llamados *Workbooks*, pudiendo realizar consultas, filtrar información, etc.
- **Text Analytics:** permite realizar diversas operaciones sobre textos, ya sean estructurados, semi-estructurados o no estrucutrados, que van desde extraer componentes usando expresiones regulares a usar Watson para hacer análisis de sentimientos.
- **BigSQL:** ofrece un entorno para trabajar con bases de datos bastante similar al del servicio dashDB, pero enfocado al Big Data con BigSQL.

Un conjunto grande de los servicios de Bluemix son los basados en el superordenador Watson para tareas de computación cognitiva. Se exploraron las capacidades de estas herramientas para determinar su utilidad como utilidades de aprendizaje automático que pueden usarse en la predicción de eventos a partir de logs conocidos. La mayoría de ellos realizan tareas poco útiles para este objetivo, como la traducción de textos o el análisis de sentimientos, sin embargo el Clasificador de Lenguaje Natural si que podría ser útil. Este clasificador recibe un texto natural como entrada y lo clasifica en función de un entrenamiento previo, lo que podría ser utilizado con los campos de logs que incluyan información detallada sin estructura. El estudio de cuales podrían ser esas posibles clases y atributos con los que entrenar quedan pendientes como trabajo futuro.

Una herramienta muy interesante y que ha sido añadida recientemente a la plataforma es la de los contenedores. Para probar sus capacidades se realizó una pequeña prueba que consistía en desplegar un servidor del popular videojuego *Minecraft* en un contenedor. Este servidor viene

empaquetado en un ejecutable JAR y requiere de un par de archivos de configuración. Además de estos archivos solo hace falta un *Dockerfile*, un archivo de texto con cierta información de configuración y el script para cargar los archivos en una imagen virtual y ejecutarlos ya en el contenedor. A continuación se muestra el contenido del Dockerfile utilizado:

```
# Version 0.0.3
#
FROM ubuntu:14.04
RUN apt-get update
RUN apt-get install -y git
RUN apt-get install -y openjdk-7-jre-headless
RUN apt-get install -y wget
RUN mkdir minecraft
#ADD server.properties ./server.properties
ADD . ./
CMD java -Xmx1024M -Xms1024M -jar minecraft_server.1.9.2.jar nogui
EXPOSE 9085
```

Usando este fichero hay que crear primero el contenedor en nuestra máquina local, para lo que hay que tener instalado *Docker*, y después subirlo a Bluemix usando la consola de Cloud Foundry. Una vez en la nube se podía acceder al servidor desde el juego a través de la dirección IP y puerto especificados por la plataforma.

En definitiva, Bluemix ofrece un conjunto de servicios y herramientas que, incluso en sus versiones gratuitas, son de gran utilidad. La contraparte negativa que existe es que es una plataforma nueva (hasta hace poco en fase beta) y continuamente en cambio, por lo que a menudo se producen modificaciones en los servicios disponibles (como la eliminación de *Apache Analytics for Hadoop*) y diversos fallos temporales (en el momento de redacción de este documento los contenedores no están disponibles, por ejemplo).

B

Diagramas UML

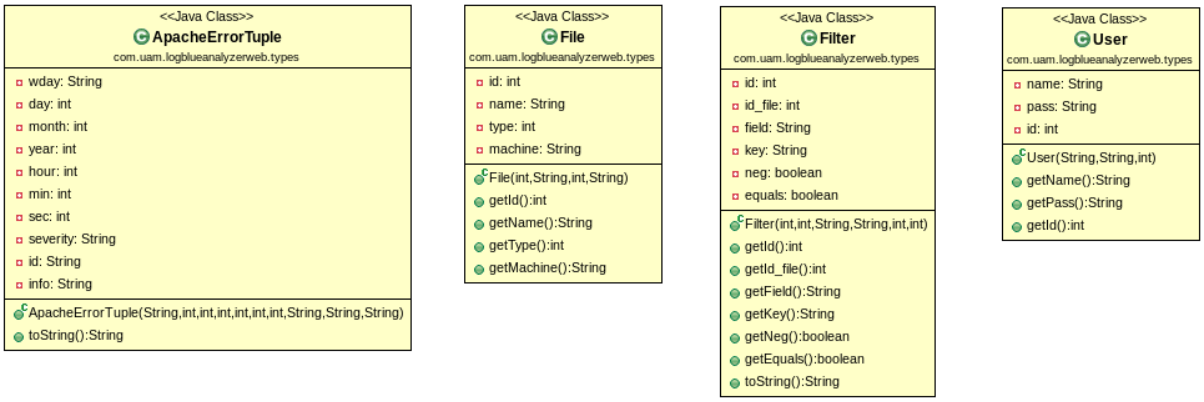

Figura B.1: Diagrama de clases de los Types de *LogBlueAnalyzer Web*.

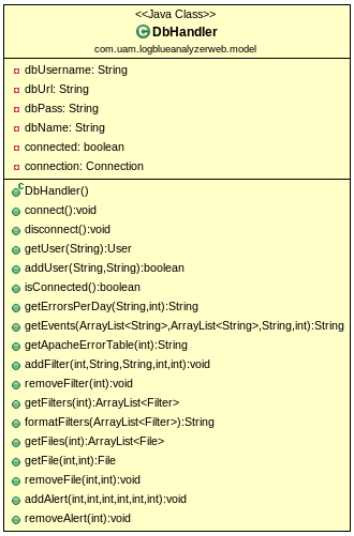

Figura B.2: Diagrama de clases de los Model de *LogBlueAnalyzer Web*.

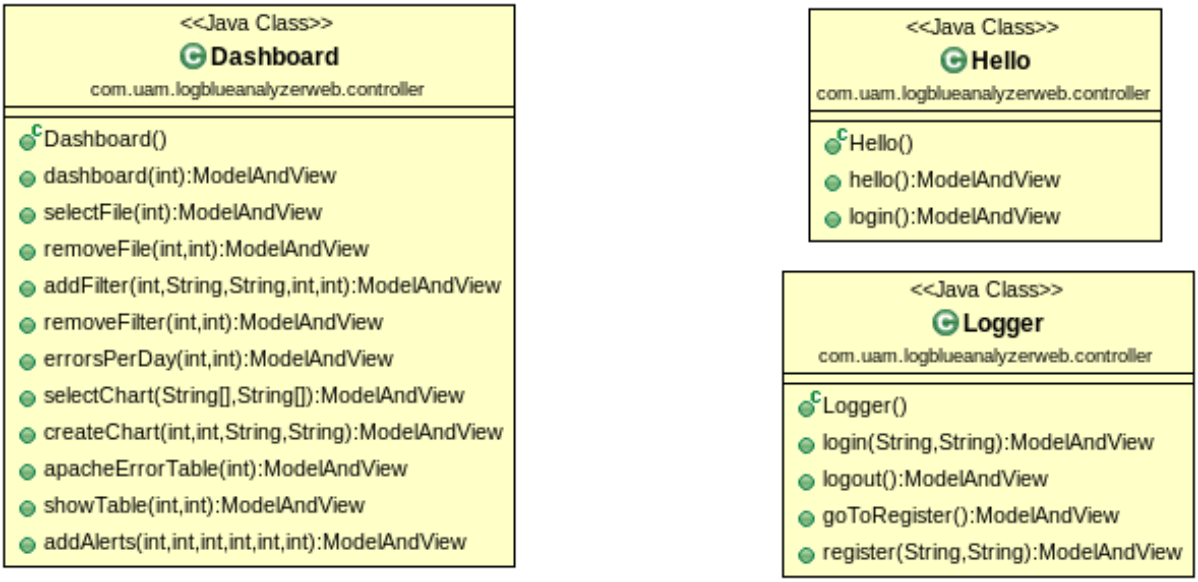

Figura B.3: Diagrama de clases de los Controllers de *LogBlueAnalyzer Web*.

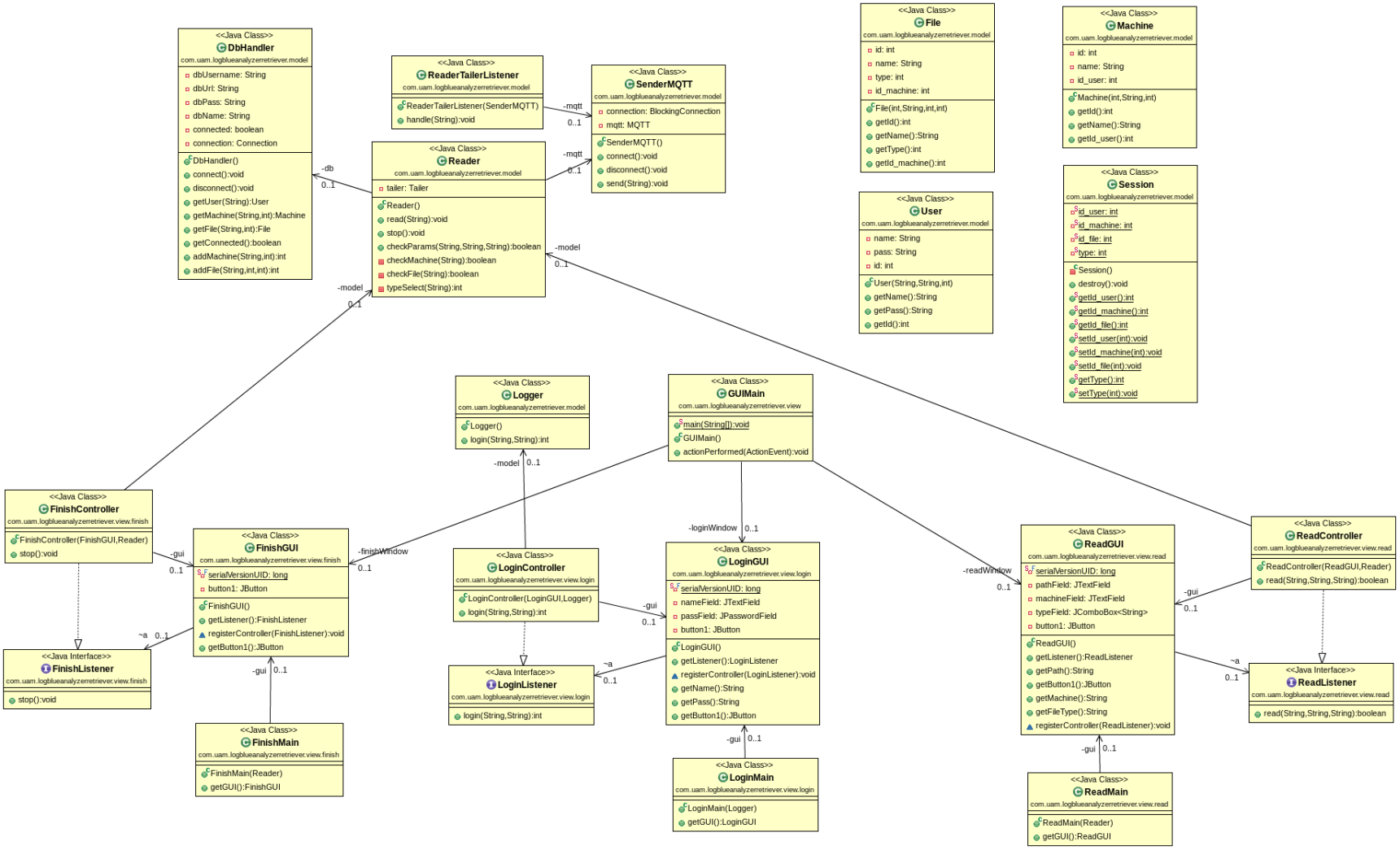

Figura B.4: Diagrama de clases de *LogBlueAnalyzer Retriever*.