



GRADO EN TECNOLOGÍAS DE LA TELECOMUNICACIÓN

Curso Académico 2018/2019

Trabajo Fin de Grado/Máster

REPRESENTACIÓN DE DATOS DE GRIMOIRE LAB CON GRAFANA

Autor : Jonathan Cano Picazo

Tutor : Dr. Jesús María González Barahona

Trabajo Fin de Grado/Máster

Representación de Datos de Grimoire Lab con Grafana

Autor : Jonathan Cano Picazo

Tutor : Dr. Jesús González Barahona

La defensa del presente Proyecto Fin de Carrera se realizó el día de
de 20XX, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 20XX

*Dedicado a
mi familia / mis amigos / mis abuelos,
pero sobre todo a mis padres, que me apoyaron desde el principio.*

Agradecimientos

Todo comenzó en el momento en el que recibí mi primer ordenador personal. Desde aquel instante la tecnología entró por mis ojos, y tuve bastante claro que quería dedicar mi vida entorno a ella. Han pasado ya unos 18 años desde entonces, y no me arrepiento de mi decisión.

Ha sido una travesía llevadera en algunos momentos, pero muy dura en otros tantos. Ha habido momentos de mucha diversión, desde el momento en el que conocí a mis primeros compañeros de clase, los cuales siguen siendo íntimos amigos. Pero también he pasado momentos difíciles, con esos temidos exámenes finales, y por desgracia bastantes recuperaciones...

Pero finalmente logré mi objetivo. Y esto no hubiese sido posible de no ser por el apoyo incondicional de mi familia: mis padres, M^a Ángeles y Francisco, y mis hermanos, Rubén e Ismael. No tengo palabras para agradecer toda la ayuda que me habéis proporcionado durante todos estos años. Aunque también recibí mucho apoyo por parte de mi abuelo, que deseaba que su nieto se labrase un futuro como ingeniero, y así sentirse completamente orgulloso al hablar de él y de sus éxitos. Y por suerte, va a vivir para contarlo...

Y por último, y no menos importante, querría darle las gracias también a mi gran amigo Javier Fernández Morata, quién ha sido quizás mi gran apoyo durante estos últimos y duros años de la carrera.

Resumen

Aquí viene un resumen del proyecto. Ha de constar de tres o cuatro párrafos, donde se presente de manera clara y concisa de qué va el proyecto. Han de quedar respondidas las siguientes preguntas:

- ¿De qué va este proyecto? ¿Cuál es su objetivo principal?
- ¿Cómo se ha realizado? ¿Qué tecnologías están involucradas?
- ¿En qué contexto se ha realizado el proyecto? ¿Es un proyecto dentro de un marco general?

Lo mejor es escribir el resumen al final.

Summary

Here comes a translation of the “Resumen” into English. Please, double check it for correct grammar and spelling. As it is the translation of the “Resumen”, which is supposed to be written at the end, this as well should be filled out just before submitting.

Índice general

1. Introducción	1
1.1. Contexto	1
1.2. Estructura de la memoria	2
2. Objetivos	5
2.1. Objetivo general	5
2.2. Objetivos secundarios	6
2.3. Planificación temporal	6
3. Estado del arte	7
3.1. Git	7
3.2. Github	8
3.3. Elasticsearch	10
3.4. Kibana	10
3.5. Grafana	11
3.6. GrimoireLab	12
3.7. Docker Hub	13
3.8. Overleaf	14
3.9. Cauldron	15
4. Diseño e implementación	17
4.1. SCRUM	17
4.2. Arquitectura general	18
4.3. Sprint 1 - Docker Hub y GrimoireLab	18
4.3.1. Preparando el contenedor	19

4.3.2.	Funcionamiento de GrimoireLab	20
4.3.3.	Exploración de dashboards con Kibiter/Kibana	21
4.4.	Sprint 2 - Grafana	22
4.4.1.	Instalación	22
4.4.2.	Enlazado	24
4.4.3.	Exploración de índices (ElasticSearch)	24
4.4.4.	Familiarización con la interfaz	26
4.5.	Sprint 3 - Dashboards y plugins de Grafana	30
4.5.1.	Plugins	33
4.6.	Sprint 4 - Dashboards funcionales y comparación	37
4.7.	Sprint 5 - Docker Hub y contenedor de GrimoireLab/Grafana	38
4.7.1.	Creando un nuevo Dockerfile	39
4.7.2.	Creación de una nueva <i>docker image</i>	40
4.8.	Sprint 6 - Actualización del contenedor de Grafana	41
4.8.1.	Exportación/importación disjunta	42
4.8.2.	Exportación/importación abrupta	44
4.9.	Sprint 7 - Imagen final y contenedores múltiples	45
5.	Resultados	47
6.	Conclusiones	49
6.1.	Consecución de objetivos	49
6.2.	Aplicación de lo aprendido	49
6.3.	Lecciones aprendidas	49
6.4.	Trabajos futuros	50
A.	Manual de usuario	51
	Lista de figuras	53

Capítulo 1

Introducción

1.1. Contexto

En la actualidad, gracias al auge de las tecnologías de la información y comunicación, un usuario es capaz de recibir continuos datos desde múltiples fuentes. Uno de los grandes usos de esta tecnología es en el ámbito de la publicidad. Los anuncios se pueden encontrar de múltiples formas: vallas publicitarias, vehículos públicos, pantallas LED... Es algo que impacta de manera muy directa en el consumidor, y puede decantar la decisión de, por ejemplo, comprar un producto o no.

Este pequeño ejemplo permite introducir un nuevo término, el cual está íntimamente relacionado. Hablamos de la visualización o representación de datos. Es una herramienta, que utilizada de la forma adecuada, permite al usuario explorar de un vistazo la parte de información que pueda interesarle más o menos. Al igual que en el caso de la publicidad, existen diversas maneras de mostrar dicha información: utilizando un aspecto más visual o llamativo, mediante gráficos o cuadros de mando; o algo más enfocado a observar un gran número de registros, datos estadísticos... Todo ello nos permite observar la correlación existente entre los diferentes datos mostrados.

La manera de representar los datos, y su aspecto visual final, son determinantes a la hora de presentarlo al usuario. Para ello existen múltiples herramientas que nos facilitan el uso y manejo de cada bit recolectado. Y esto es precisamente lo que se aborda en este proyecto. Explorar esas herramientas y explotar al máximo sus posibilidades.

Por ese mismo motivo, las grandes empresas dedican muchos recursos a este sector, ya que es una manera muy eficaz de, por ejemplo:

- Observar la escala evolutiva de un proyecto.
- Tener un seguimiento en tiempo real de diferentes infraestructuras (webs, BBDD, radio-enlaces...).
- Diagnosticar problemas (de rendimiento, uso, congestión...).
- Y por supuesto, atraer la atención de futuros clientes/patrocinadores.

Esta fue la visión general que mi tutor, el Dr. Barahona, me dio sobre el proyecto en el que me acabaría embarcando. No tardé mucho en decantarme por la idea ofertada, ya que me entró mucho por los ojos, me impactó. Tuvo un efecto similar al de un consumidor con un anuncio, como se ha comentado al principio.

1.2. Estructura de la memoria

Debido al gran volumen de datos del proyecto en el que nos encontramos, se ha de seguir una determinada organización, para proporcionar una correcta lectura y comprensión. Esta tesis, por lo tanto, se estructura de la siguiente manera:

- **Capítulo 1: Introducción.** Se presenta la idea inicial del proyecto y como surgió. Además, se proporciona una pequeña estructura de la tesis, explicando el contenido de cada sección.
- **Capítulo 2: Objetivos.** En este apartado se define más concretamente la finalidad de este proyecto y cuales son sus requisitos u objetivos finales.
- **Capítulo 3: Estado del arte.** Aquí nos encontraremos con una descripción general de todas las herramientas utilizadas en este proyecto (historia, características, uso...).
- **Capítulo 4: Diseño e implementación.** Se presentan y desarrollan las diferentes etapas que han llevado al proyecto hasta su estado final. Estas etapas están basadas en el algoritmo Scrum, para desarrollo ágil de software.

- **Capítulo 5: Resultados.** Análisis de los resultados finales obtenidos.
- **Capítulo 6: Conclusiones.** En esta sección se contrastarán los datos obtenidos con los objetivos iniciales del proyecto, para comprobar si se han cumplido las expectativas. Además, se proponen mejoras y posibles proyecciones futuras, así como una valoración general del proyecto y de los conocimientos aplicado y aprendidos.

Capítulo 2

Objetivos

2.1. Objetivo general

En este proyecto se compone de 4 objetivos principales:

- Representación de datos con Grafana¹ gracias al uso de Grimoire Lab², la herramienta que nos permite la recolección y almacenamiento de datos extraídos de Git³ y Github⁴.
- Unificación de ambas en un mismo *docker* o contenedor gracias a Docker Hub, automatizando así el despliegue de ambas aplicaciones en una misma imagen.
- Despliegue web de este software unificado, permitiendo un acceso más rápido e intuitivo al usuario.
- Comparación de las dos herramientas de representación utilizadas, Grafana y Kibana⁵. Para ello se exploran los límites de cada una de ellas, ahondando sobre todo en el apartado de plugins que ofrece la primera.

¹<http://www.grafana.com>

²<https://chaoss.github.io/grimoirelab/>

³<https://git-scm.com/>

⁴<https://github.com/>

⁵<https://www.elastic.co/es/products/kibana>

2.2. Objetivos secundarios

Para conseguir los resultados propuestos en el objetivo principal, ha sido necesario también manejar al completo distintas herramientas, como es el caso de Elasticsearch⁶, Kibana y Docker Hub⁷.

Primeramente, se abordó el tema de las bases de datos no relacionales. En este proyecto, la mayor parte de la información recolectada se almacena en Elasticsearch. Por lo tanto, ha sido necesaria la formación en este tipo de BBDD, ya que siempre había trabajado con relacionales. Además, enlaza directamente con la segunda herramienta, Kibana. Esta última es un complemento de visualización de datos para Elasticsearch, y es el software utilizado por defecto por Grimoire Lab. Una vez aprendí su total manejo, ha servido de guía para obtener resultados similares a los proporcionados por Grafana, y realizar así su posterior comparación.

2.3. Planificación temporal

La realización de este proyecto se ha llevado a cabo durante un periodo de 12 meses. La mayor parte del tiempo coincidió durante mi último curso académico, mientras que los últimos meses se solapó con mi trabajo actual. En este lapso, el proyecto ha seguido diferentes fases:

- Elección del proyecto. En primer lugar, me reuní con mi tutor del proyecto, Jesús, para que me ayudase a elegir un proyecto que se adaptase a lo que estaba buscando. Me presentó dos posibles ideas, quedándome finalmente con la presente, ya que me llamó mucho más la atención.
- Aprendizaje de las tecnologías a utilizar. Esta fue la parte más tediosa y larga del proyecto. Hizo falta leer mucha documentación e instalar bastantes dependencias, para adecuar el entorno sobre el que desarrollar la idea que Jesús y yo teníamos en mente.
- Implementación de los objetivos propuestos.
- Redacción de la memoria. Esta fase se fue solapando con la anterior, rellenando el documento presente a medida que se terminaban objetivos concretos del proyecto.

⁶<https://www.elastic.co/es/products/elasticsearch>

⁷<https://hub.docker.com/>

Capítulo 3

Estado del arte

Como se ha comentado en el capítulo anterior, existen infinidad de herramientas para representar visualmente la información. Estas dependen a su vez de muchas otras, que conectadas, nos permiten reproducir un resultado final muy llamativo. En este capítulo, por tanto, se abordarán todos los aspectos relacionados con el software utilizado para la elaboración de este proyecto. Para poder presentar todas las tecnologías, debemos empezar por lo básico, lo primero que se realiza, la obtención de la información. Para ello nos ayudamos de dos plataformas: Git y Github.

3.1. Git

Git es un software de control de versiones (VCS, *Version Control System*) originalmente diseñado por Linus Torvalds en 2005. Su propósito es llevar registro de los cambios en archivos, permitir recuperar versiones anteriores de estos o actualizar a versiones posteriores. Está pensado para coordinar el trabajo que varias personas realizan sobre archivos compartidos. Actualmente, Git es el VCS más usado en el mundo, por encima de sus competidores directos, como pueden ser CVS¹, SVN² o Mercurial³. Su funcionamiento es muy sencillo. Partimos de un espacio de trabajo personal o *repositorio*. A partir de aquí, procedemos a crear y/o modificar los archivos de nuestro proyecto. Para poder guardar el estado en el que se encuentra en ese momento nuestro repositorio, primero tenemos que indicarle a Git que archivos queremos que

¹<https://www.gnu.org/software/trans-coord/manual/cvs/cvs.html>

²<https://subversion.apache.org/>

³<https://www.mercurial-scm.org/>

preserve a lo largo del tiempo. Una vez añadidos, basta con “comprometer” los cambios (hacer un *commit*) para que Git haga una captura o *snapshot* del estado del repo. A partir de aquí, podemos crear nuevas ramas de trabajo paralelas a una versión dada, volver a versión más antigua del proyecto, borrar versiones...

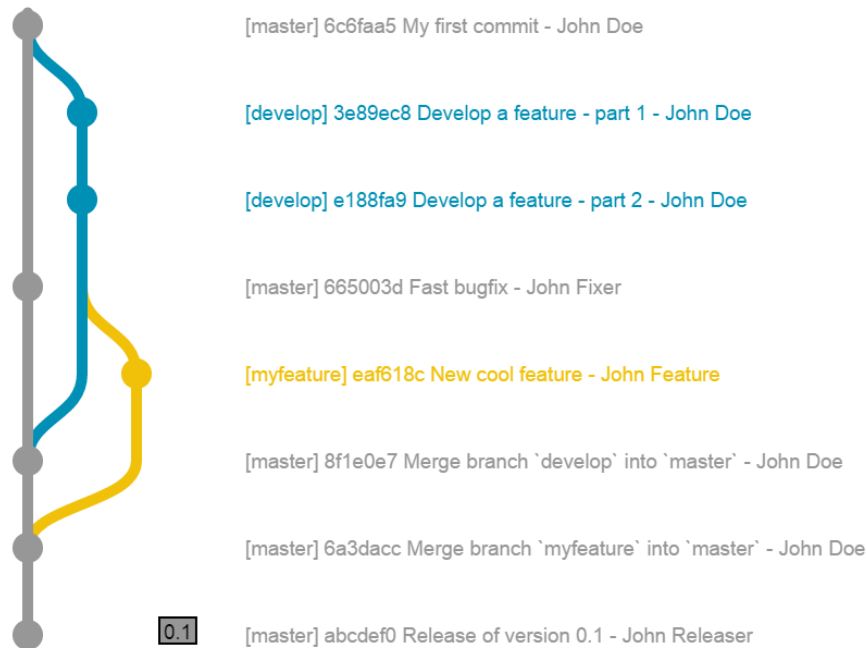


Figura 3.1: Arquitectura en árbol de Git

Este tipo de estructura es la que ha hecho a Git tan famoso hoy en día, y ha permitido tener grandes grupos de personas trabajando remota y paralelamente en un mismo proyecto. Para poder trabajar de manera remota y no local, surgió la herramienta que se presenta a continuación, Github.

3.2. Github

GitHub es un sitio web que permite alojar proyectos utilizando el sistema de control de versiones Git. Fue fundado en 2008, y se usa principalmente para proyectos de desarrollo de código fuente conjuntos. El software que opera GitHub fue escrito en Ruby on Rails. Actualmente, es una de las webs líder en su sector, muy por delante de sus competidoras directas,

como son Buddy⁴ o Bitbucket⁵. La principal diferencia entre una herramienta y otra, reside en que Git solo funciona mediante un terminal o línea de comandos, mientras que Github proporciona una interfaz gráfica muy vistosa. Esta tiene multitud de opciones, entre las que destacan, entre muchas otras:

- Cada usuario tiene su perfil único, donde se muestra en que proyectos está trabajando, todos sus contribuciones pasadas, etc.
- Posibilidad de contribuir en proyectos públicos, haciendo *fork* de estos (copia editable del diseño de otro usuario), y si se permite, un posterior *pull request* (acción de validar un código que se va a hacer *merge* de una rama a otra).
- Páginas web dedicadas a cada proyecto, contando con *wiki*, documentación, registro de *commits* anteriores, *issues*, etc.

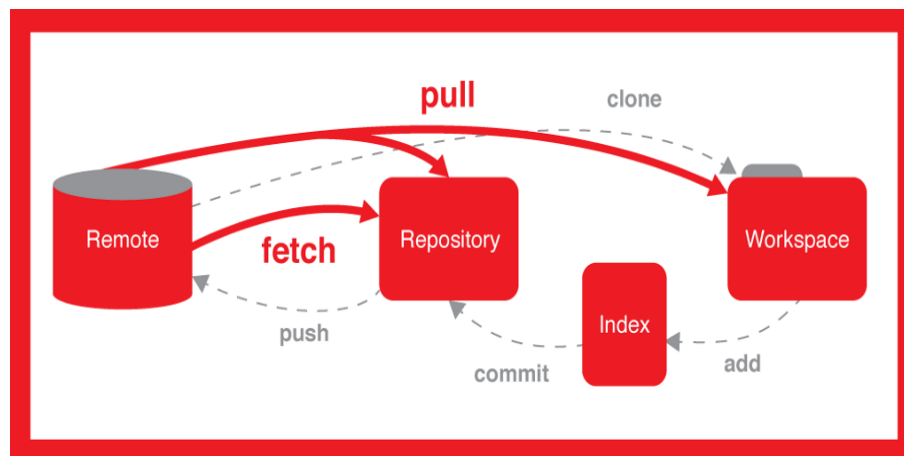


Figura 3.2: Funcionamiento básico de Github

⁴<https://buddy.works/>

⁵<https://bitbucket.org/>

3.3. Elasticsearch

Elasticsearch es un servidor de búsqueda y análisis basado en Lucene⁶. Es de código abierto, basado en Java⁷, y con licencia Apache⁸. Trabaja conjuntamente con un motor de recopilación de datos llamado Logstash⁹, y una plataforma de análisis y visualización llamada Kibana¹⁰. Juntos, forman el proyecto ELK (acrónimo de los nombres de las tres herramientas). Entre sus principales características, se puede destacar que:

- Posee una intuitiva API con interfaz web RESTful.
- Provee un motor de búsqueda de texto completo, distribuido y con capacidad de multi-tenencia con documentos JSON¹¹.
- Es un sistema de base de datos no relacional, por lo que tiene una gran velocidad de respuesta al trabajar con conjuntos muy grandes de datos, debido al uso de índices invertidos distribuidos.
- Viene completamente integrado con Logstash y Kibana, por lo que facilita el uso de las tres herramientas a la vez (recolección/tratado, guardado y visualización).

3.4. Kibana

Kibana es un *plugin* o complemento de Elasticsearch, de código abierto. Permite la visualización de datos previamente indexados por la herramienta anterior. Ofrece múltiples posibilidades para crear diferentes representaciones: gráficos de barras horizontales/verticales, lineales, *scatter*/dispersión, de tarta, circulares, etc. Además, permite diseñar tablas, listas, tops, medidores, mapas de calor... Al igual que Elasticsearch, Kibana es fácilmente accesible desde cualquier navegador de manera local. A partir de ahí, se pueden crear paneles desde cero, o utilizar alguna de las bibliotecas de la herramienta para importar paneles predeterminados, y hacerse una idea de un posible resultado final.

⁶<https://lucene.apache.org/>

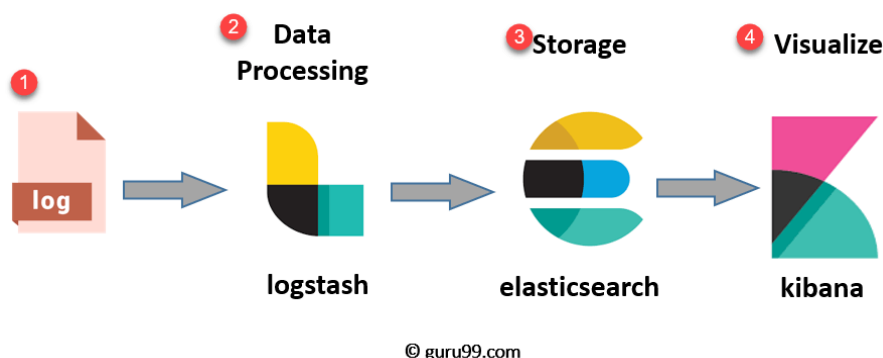
⁷<https://www.java.com/es/>

⁸<http://httpd.apache.org/>

⁹<https://www.elastic.co/es/products/logstash>

¹⁰<https://www.elastic.co/es/products/kibana>

¹¹<https://www.json.org/>

Figura 3.3: Esquema de funcionamiento de *ELK Stack*

3.5. Grafana

De manera similar a la herramienta anterior tenemos a Grafana, otro potente software de código abierto basado en licencia de Apache, escrito en lenguaje *Go*¹². Permite también la visualización de información almacenada en múltiples bases de datos. Trabaja especialmente bien con BBDD de series de tiempo, tales como Graphite, InfluxDB... Originalmente comenzó como un componente de Kibana, aunque en la actualidad se considera como proyecto independiente. Entre las características principales, se puede destacar:

- Es una herramienta RESTful. Se puede ejecutar desde cualquier terminal, y a partir de ahí es accesible desde cualquier navegador, al ser una API HTTP.
- Es multiplataforma, y no tiene ninguna dependencia con otros programas, al contrario que Kibana.
- Además de los cuadros de mandos básicos (gráficos, tablas, *heatmaps*...) permite añadir muchos más mediante *plugins* de fácil instalación.
- Tiene una gran comunidad detrás, la cual amplía drásticamente las maneras de representar los datos, mediante los citados complementos.
- Compatibilidad con una de bases de datos: Elasticsearch, MariaDB¹³, MySQL¹⁴, Post-

¹²<https://golang.org/>

¹³<https://mariadb.org/>

¹⁴<https://www.mysql.com/>

greSQL¹⁵, CloudWatch¹⁶...

- Gestión muy intuitiva de usuarios y *dashboards*¹⁷. Esto permite que cada persona posea unos cuadros de mandos propios, y que pueda realizar *snapshoots* (capturas estáticas) de estos, para poder compartirlos públicamente.

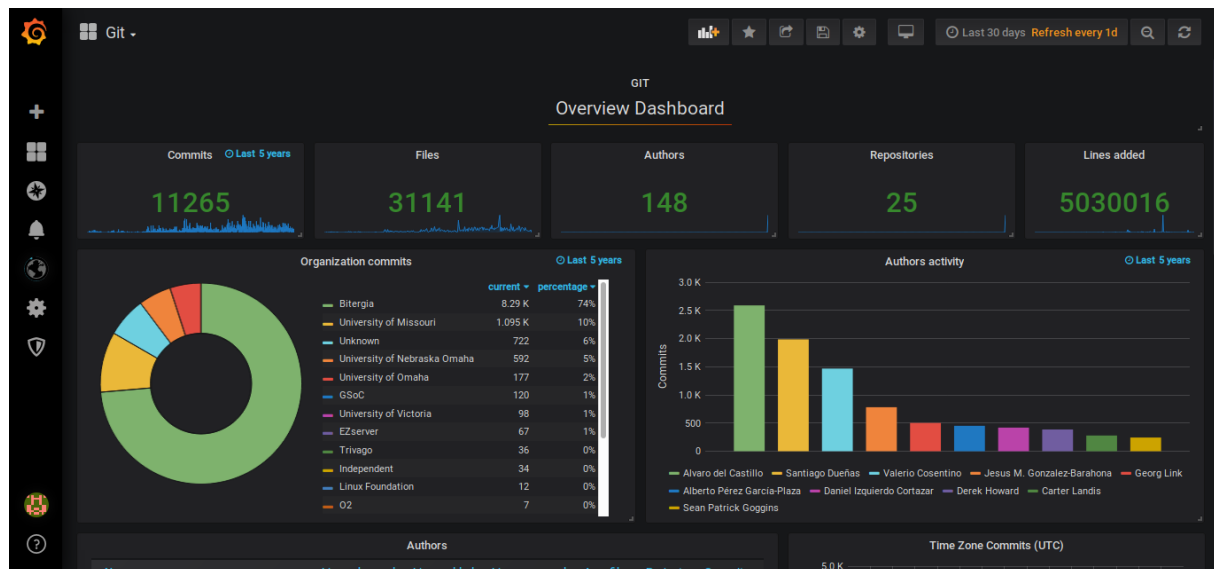


Figura 3.4: Ejemplo de dashboard

3.6. GrimoireLab

GrimoireLab es un conjunto de herramientas de código libre, orientada a la investigación analítica de datos. Ha sido desarrollada por la empresa española Bitergia¹⁸. Permite extraer datos desde muchas fuentes distintas, y producir análisis y visualizaciones de estos. Su uso ha sido crucial para el propósito de esta memoria, pues gracias a su herramienta Perceval¹⁹, nos permite recolectar información de más de 20 fuentes diferentes, pertenecientes a repositorios de Git y a proyectos de Github. Estos proyectos abarcan desde *issue trackers* (rastreadores de problemas)

¹⁵<https://www.postgresql.org/>

¹⁶<https://aws.amazon.com/es/cloudwatch/>

¹⁷<https://www.40defiebre.com/que-es/dashboard>

¹⁸<https://bitergia.com/>

¹⁹<https://chaoss.github.io/grimoirelab-tutorial/perceval/intro.html>

como Jira²⁰ o Bugzilla²¹, *mailing lists* o listas de correo, diferentes fuentes de Stackoverflow²², etc. Actualmente, GrimoireLab forma parte del proyecto CHAOSS, perteneciendo también a la Fundación Linux.

3.7. Docker Hub

Docker Hub es un conjunto de repositorios que permite alojar contenedores de imágenes de distintos tipos: proyectos de desarrollo de la comunidad, proyectos de código abierto o vendedores independientes de código (ISV), que construyen y distribuyen su código en contenedores. Los usuarios pueden tener acceso a repositorios públicos gratuitos para almacenar y compartir sus propias imágenes, u optar por un plan de pago, permitiendo tener repositorios privados. La empresa Docker fue creada en 2013 por Solomon Hykes. Está basada en el lenguaje de programación Go, y cuenta con licencia Apache. La idea inicial del proyecto fue poder crear un entorno virtual que permita empacar todas las dependencias, herramientas y código necesario para que un número determinado de aplicaciones se ejecute de manera rápida y segura. Este paquete ejecutable, en esencia, una imagen de contenedor. En la Figura 3.5 se muestra de manera global como funciona la herramienta. Entre sus características principales, destacan:

- Tiene la mayor comunidad de contenedores de imágenes del mundo. La mayoría son oficiales y gratuitas, por lo que podemos hacer *pull* de ellas directamente (ejemplos serían Elasticsearch, Grafana, GrimoireLab, Ubuntu, MongoDB, etc.).
- Permite a los desarrolladores resolver el problema “works on my machine”. Al tener entornos virtuales aislados, permite trabajar de manera simultánea en diferentes espacios específicos a cada situación.
- Tolera la construcción de imágenes de manera automática, desde contenedores alojados en Github o Bitbucket. Si enlazamos ambas plataformas, al hacer *push* se resubren las imágenes en Docker Hub.

²⁰<https://www.atlassian.com/es/software/jira>

²¹<https://www.bugzilla.org/>

²²<https://stackoverflow.com/>

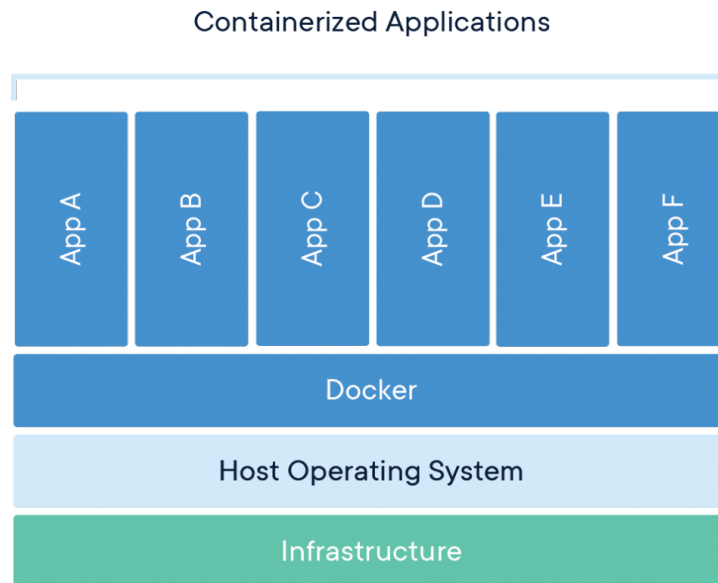


Figura 3.5: Esquema de funcionamiento de DockerHub

3.8. Overleaf

Overleaf es un editor de LaTeX²³ online. Permite usar de manera rápida, sencilla y colaborativa la herramienta más famosa de composición de textos profesionales. Fue fundado en 2017, fusionándose con la anterior herramienta de edición, ShareLatex²⁴. Existen dos versiones del editor, una gratuita y una de pago. En este proyecto hemos utilizado la licencia gratuita, la cual cuenta con todas las características de edición propias de otras herramientas, como son TeXmaker²⁵, LyX²⁶, etc. Aunque también tiene alguna restricción, como no poder trabajar con documentos muy grandes sin tener que dividirlos en diferentes ficheros o paquetes, o bien no tener sincronización automática con plataformas como Dropbox o Github.

²³<https://www.latex-project.org/>

²⁴<https://es.sharelatex.com/>

²⁵<https://www.xmlmath.net/texmaker/>

²⁶<https://www.lyx.org/>

3.9. Cauldron

Si se integra Grafana en esta aplicación, hablar de ello ²⁷

²⁷<https://gitlab.com/cauldron2/cauldron-deployment>

Capítulo 4

Diseño e implementación

En este capítulo abordaremos diferentes aspectos del proyecto, como son su estructuración, su desarrollo y su implementación. Primeramente, debemos mencionar la metodología utilizada en este proyecto, la cual es una modificación más simple del conocido **método SCRUM**¹.

4.1. SCRUM

Scrum es una metodología de trabajo enfocada al **desarrollo ágil de software**. Se basa en la aplicación de un conjunto de buenas prácticas de manera regular, para trabajar en equipo, y obtener el mejor resultado posible de proyectos. Su principal característica consiste en solapar las diferentes fases del desarrollo, en lugar de realizar una tras otra en un ciclo secuencial o en cascada. Para ello, el trabajo se subdivide en tareas más pequeñas, para así poder abordarlas concurrentemente. Estas subdivisiones se denominan *sprints*.

El *sprint* es el período de tiempo en el cual se lleva a cabo el trabajo en sí. En este proyecto su duración inicial era de una semana, aunque se fue ampliando debido al volumen de objetivos a realizar en cada *sprint*, los llamados *sprint backlogs*.

Debido a que nuestro equipo de desarrollo es muy pequeño (dos personas), nos vimos obligados a modificar la metodología, a simplificarla. Por tanto, no estarán presentes aspectos característicos de Scrum, como pueden ser reuniones diarias, duración mínima de sprints (dos semanas), flexibilidad a cambios (no tenemos cliente), predicciones de tiempos, etc.

¹<https://www.scrum.org/resources/blog/que-es-scrum>

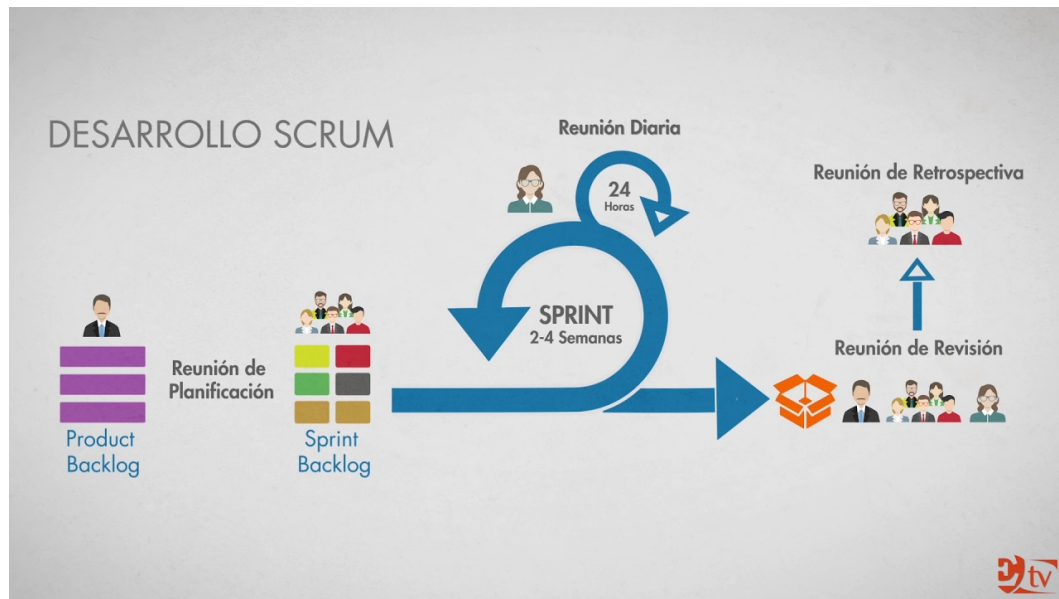


Figura 4.1: Esquema del entorno de trabajo SCRUM

4.2. Arquitectura general

A continuación, se presenta la estructura general del proyecto. Tal y como se ha comentado anteriormente, está dividida en los diferentes *scrums* realizados. En la Figura 4.2 se puede observar de un vistazo la totalidad del proyecto. Se estructura en cuatro partes:

- Bloques azules: DockerHub, GrimoireLab y Kibana.
- Bloques verdes: Grafana y plugins.
- Bloques morados: Creación de un contenedor de imagen que incluya todo.
- Bloque naranja: Comparación de las dos herramientas principales: Grafana y Kibana.

4.3. Sprint 1 - Docker Hub y GrimoireLab

Tal y como se ha mostrado en el esquema anterior, el primer paso del proyecto fue instalar los componentes y dependencias necesarias para poder visualizar nuestro primer dashboard. Para ello, utilizamos Docker Hub, herramienta mencionada en la sección 3.7, que nos permitirá utilizar un contenedor de imagen ya preparado para poder visualizar rápidamente diferentes

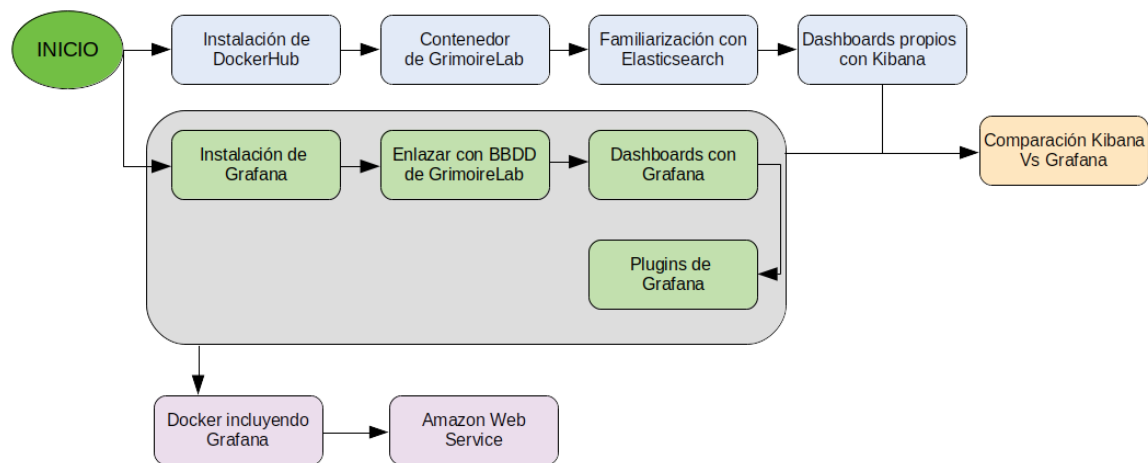


Figura 4.2: Diagrama de bloques del proyecto

paneles de control. Más adelante, en la sección 4.7, se profundiza en el funcionamiento de la herramienta.

4.3.1. Preparando el contenedor

El contenedor de imagen a utilizar es `grimoirelab/full`. Fue creado por el equipo CHAOSS, y contiene todos los elementos necesarios para producir dashboard completo y funcional: Elasticsearch, MariaDB, and Kibiter. El contenedor produce un dashboard del proyecto GrimoireLab, que nos servirá de guía para nuestros futuros paneles. Lo primero, por tanto, será la instalación de Docker Hub². También es recomendable seguir los pasos de la post-instalación³, para evitar tener que dar privilegios de acceso/administrador (usuario `root`), escribiendo ‘sudo’ al principio de cada sentencia).

Una vez instalada la herramienta siguiendo el manual (mediante repositorio o paquete de instalación), se procede a descargar el contenedor de imagen mencionado anteriormente. Basta

²<https://docs.docker.com/install/linux/docker-ce/ubuntu/>

³<https://docs.docker.com/install/linux/linux-postinstall/>

con introducir la siguiente sentencia:

```
1 docker pull grimoirelab/full
```

Antes de poder lanzar el contenedor debemos preparar un archivo de configuración, para que este pueda recolectar adecuadamente los datos necesarios. Para ello, debemos crear el fichero `credentials.cfg` en una ruta de fácil acceso (`/home/USER` en mi caso). El fichero debería ser un *token*⁴ de la API de Github, siguiendo el siguiente formato:

```
1 [github]
2 api-token = XXX
```

Ahora sí, se procede a lanzar el contenedor de imagen, utilizando el comando `docker run`⁵:

```
1 docker run -p 127.0.0.1:5601:5601\
2 -v $(pwd)/credentials.cfg:/override.cfg\
3 -t grimoirelab/full
```

Ahora, para poder visualizar nuestro primer dashboard, nos dirigimos a cualquier navegador e introducimos: `http://localhost:5601`. Tras un determinado tiempo de espera (hasta que la herramienta recolecte todos los índices), se nos mostrará algo similar a la Figura 4.5. Toda la información ampliada se encuentra disponible en el tutorial de la página del proyecto GrimoireLab⁶.

4.3.2. Funcionamiento de GrimoireLab

Antes de continuar explorando la herramienta de GrimoireLab, se decidió estudiar exhaustivamente su funcionamiento. Para ello, el esquema de la Figura 4.3 fue de gran ayuda:

En la Figura 4.3 se pueden apreciar todos los componentes de la herramienta, representados dentro del rectángulo verde. Las flechas sombreadas corresponden al flujo principal de los datos: comenzamos con Perceval (que extrae la información de las fuentes), siguiendo por Arthur (que programa lotes de recuperación y almacena resultados en Redis⁷), llegando a GrimoireELK (que almacena los datos extraídos como índices crudos, y los utiliza para producir

⁴<https://github.com/settings/tokens>

⁵<https://docs.docker.com/engine/reference/run/>

⁶<https://chaoss.github.io/grimoirelab-tutorial/basics/dockerhub.html>

⁷<https://redis.io/>

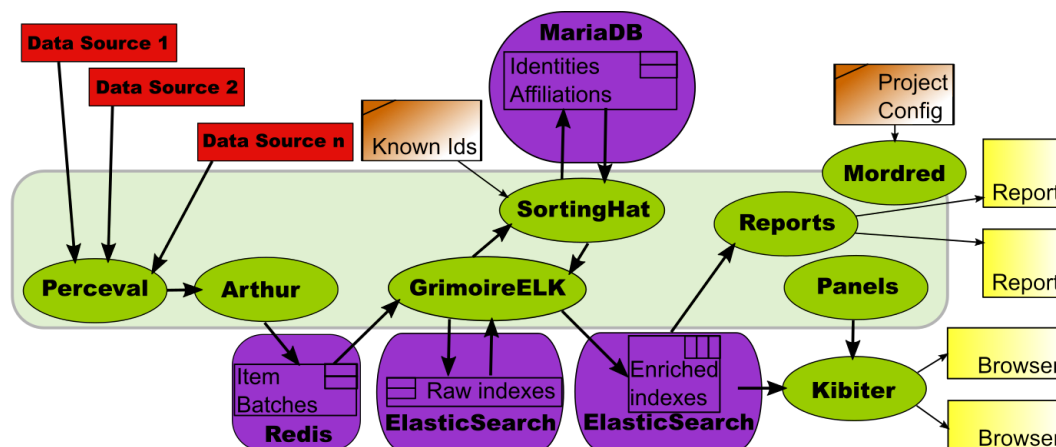


Figura 4.3: Esquema de funcionamiento de GrimoireLab

índices enriquecidos⁸, ambos en ElasticSearch), para finalmente llegar a Kibiter (herramienta *soft fork* de Kibana, para visualizar dashboards interactivables).

GrimoireELK (análogo al proyecto ELK visto en la Sección 3.3) utiliza SortingHat para almacenar todas las identidades que encuentra en una DB de MariaDB. SortingHat utiliza listas de identificadores conocidos (generalmente mantenidos en archivos de configuración) y heurísticas para fusionar identidades correspondientes a la misma persona e información relacionada (como afiliación).

Todo el proceso es configurado y orquestado por Mordred, el cual usa su propia configuración sobre, por ejemplo, qué fuentes de datos utilizar.

4.3.3. Exploración de dashboards con Kibiter/Kibana

Una vez aprendido el funcionamiento de GrimoireLab, comenzamos a explorar la cantidad de posibilidades que ofrece una de sus herramientas, Kibiter (que es un “soft fork” de Kibana, compartiendo todas las funciones principales de esta).

Siguiendo los pasos de la sección 4.3.1, tendremos inicializado el contenedor, el cual irá mostrando distintas líneas en el terminal a modo de *logs*, como se aprecia en la Figura 4.4. Una vez recolectada la información y enriquecidos los índices (puede demorar un tiempo), podremos acceder a la página desde el navegador (<http://localhost:5601>), mostrándose así nuestro primer dashboard. Todos los aspectos relacionados con la base de datos se profundizan en la Sección

⁸<https://chaoss.github.io/grimoirelab-tutorial/README.html#data-storage>

4.4.3

```

jonycp@jonycp:~$ sudo docker run -p 127.0.0.1:5601:5601 -p 127.0.0.1:9200:9200 -p 127.0.0.1:3306:3306 -v $(pwd)/credentials.cfg:/override.cfg -t grimoirelab/full
Starting container: 68c994101722
Starting Elasticsearch
[ OK ] Starting Elasticsearch Server:..
Waiting for Elasticsearch to start...
tcp        0      0 0.0.0.0:9200        0.0.0.0:*           LISTEN
Elasticsearch started
Starting MariaDB
[ OK ] Starting MariaDB database server: mysqld . . .
Waiting for MariaDB to start...
tcp6      0      0 :::3306            :::*                LISTEN
MariaDB started
Starting Kibiter
Waiting for Kibiter to start...
..Kibiter started
Starting Sirmordred to build a GrimoireLab dashboard
This will usually take a while...
/usr/local/lib/python3.5/dist-packages/sirmordred/task_panels.py:180: YAMLLoadWarning: calling yaml.load() without Loader=... is deprecated, as the default Loader
is unsafe. Please read https://msg.pyyaml.org/load for full details.
  self.panels.menu = yaml.load(f)
/usr/local/lib/python3.5/dist-packages/sirmordred/task_panels.py:437: YAMLLoadWarning: calling yaml.load() without Loader=... is deprecated, as the default Loader
is unsafe. Please read https://msg.pyyaml.org/load for full details.
  self.panels.menu = yaml.load(f)
Collection for pipemail: starting...
Collection for github: starting...
Collection for git: starting...
Loading blacklist...
0/0 blacklist entries loaded
Loading unique identities...

```

Figura 4.4: Logs mostrados tras el lanzamiento del contenedor de GrimoireLab

A continuación, si nos situamos en la barra superior de la página, podemos elegir entre las diferentes categorías de paneles a visualizar (todos pertenecientes al proyecto CHAOSS). Un ejemplo sería el de *Overview*, el cual mostrará algo similar a la Figura 4.5. Al ser un contenedor, podemos modificar a antojo los dashboards, ya que cada vez que lo lancemos, regresarán a su estado original.

Más adelante, en la Sección ?? se ahonda más en el uso de Kibana, y se compara con Grafana, la herramienta principal de este proyecto.

4.4. Sprint 2 - Grafana

4.4.1. Instalación

Tras seguir los pasos del manual de instalación de Grimoirelab y comprobar su correcto funcionamiento, la siguiente tarea fue intentar enlazar esta herramienta con Grafana. Para ello, se procedió a su instalación⁹, la cual es tan simple como ejecutar las siguientes sentencias:

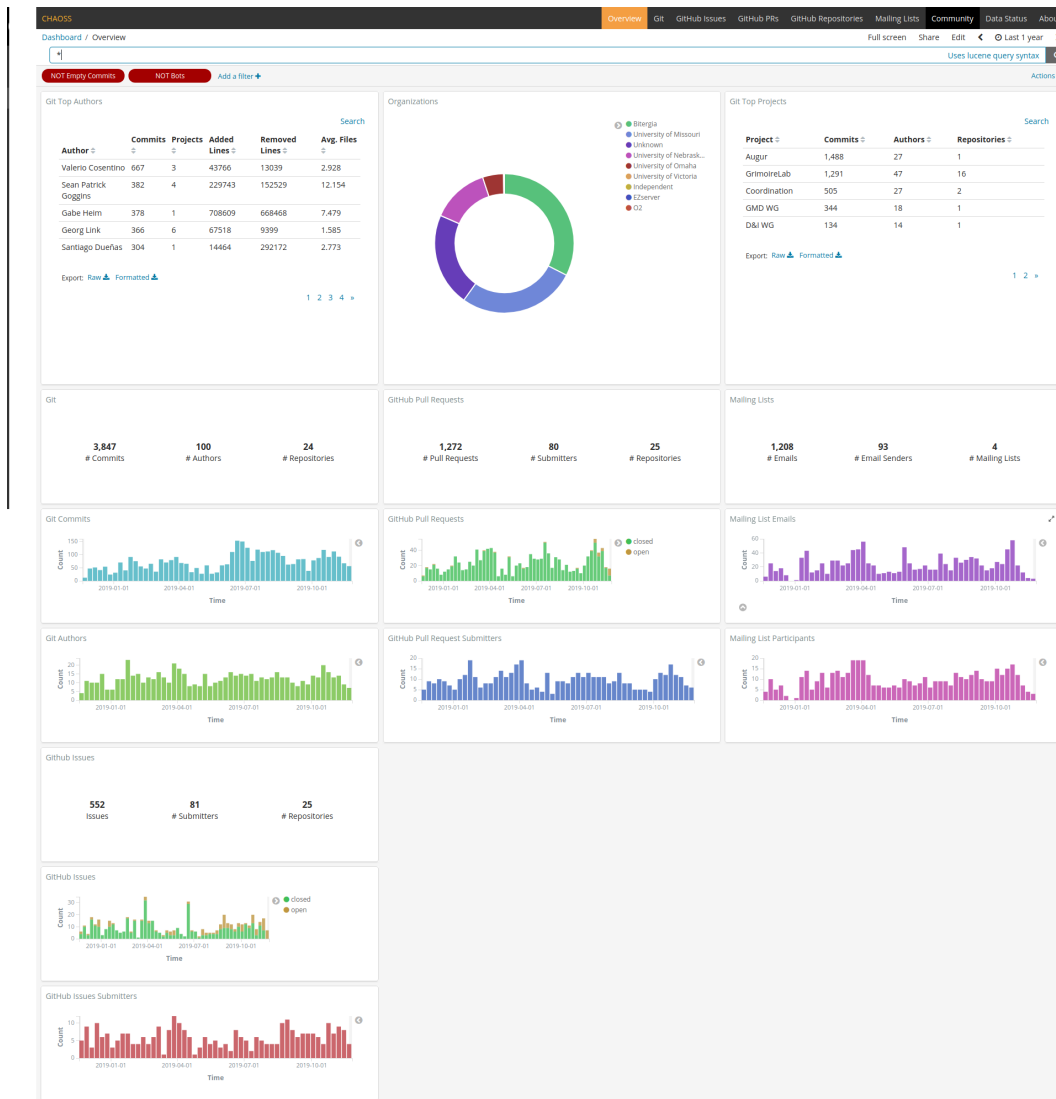
```

1 wget https://dl.grafana.com/oss/release/grafana_6.4.4_amd64.deb
2 sudo dpkg -i grafana_6.4.4_amd64.deb

```

En mi caso, se instaló primeramente la versión 6.1.3 sobre una distro basada en Debian. A medida que avanzó el proyecto se fue actualizando, ya que permitió solucionar algunos fallos de diseño.

⁹<https://grafana.com/grafana/download?platform=linux>

Figura 4.5: Ejemplo de dashboard ofrecido en *Overview*)

Ahora, para poder acceder a la herramienta, la iniciamos con la siguiente instrucción:

```
1 sudo service grafana-server start
```

Esto iniciará el proceso correspondiente al servidor de Grafana. Para acceder al GUI, basta con dirigirse al navegador e ir a <http://localhost:3000/>. Por defecto, el servidor se quedará “escuchando” en el puerto 3000, aunque se puede cambiar a cualquier otro puerto libre¹⁰. Además, la primera vez que entremos, el usuario y contraseña por defecto serán `admin/admin`.

¹⁰https://grafana.com/docs/guides/getting_started/

Nombre	Descripción
<i>Name</i>	Nombre que le daremos al índice en Grafana
<i>URL</i>	Dirección y puerto en el que escucha Elasticsearch
<i>Access</i>	Como se realizan las consultas a la BD (Server/Browser)
<i>Auth fields</i>	RELLENAR EN UN FUTURO acceso remoto a db
<i>Index name</i>	Nombre del índice real en Elasticsearch
<i>Time Field name</i>	Nombre del <i>document</i> de tipo tiempo (obligatorio)
<i>Version</i>	Versión de Elasticsearch (6.0+ para rango entre 6.0 - 7.0)
<i>Max concurrent Shard Request</i>	Número máximo de consultas a subíndices (<i>shards</i>)

Cuadro 4.1: Parámetros de configuración de la BD

4.4.2. Enlazado


Para poder visualizar los datos correctamente, es necesario ligar Grafana con las bases de datos de Elasticsearch. Siguiendo los pasos de la documentación¹¹, accedemos al menú *Configuration ->Data Sources*, ubicado en el menú lateral. A continuación, al pulsar sobre *Add data source*, tendremos que seleccionar/rellenar diversos campos (no todos son obligatorios). En la siguiente tabla se explican los campos principales:

Un ejemplo de configuración sería el mostrado en la Figura 4.6. Al pulsar sobre el botón *Save and Test*, Grafana intentará conectarse a la base de datos y explorar el índice. Si hemos puesto todo correctamente, se nos mostrará un mensaje en verde con *Index OK*. *Time field name OK* y la fuente se añadirá a las ya existentes. Si no, aparecerá un mensaje en rojo con *Invalid index*. Este último error suele deberse a que hemos introducido un nombre incorrecto de índice temporal, o bien que no tenemos conexión con Elasticsearch.


4.4.3. Exploración de índices (ElasticSearch)

Para escoger correctamente los *Data Sources* de Grafana, hemos de estudiar detenidamente los índices de la base de datos. Para ello contamos con una consola proporcionada por Kibana, que se accede desde el icono de la herramienta (*DevTool>Console*). Desde ahí, podemos

¹¹<https://grafana.com/docs/features/datasources/elasticsearch/>


 **Data Sources / git**
Type: Elasticsearch


Settings


Name  git

Default ☐


HTTP


URL  http://localhost:9200

Access Server (Default)  [Help ▶](#)


Whitelisted Cookies  Add Name

Auth

Basic Auth ☐ With Credentials  ☐


TLS Client Auth ☐ With CA Cert  ☐

Skip TLS Verify ☐


Forward OAuth Identity  ☐

Elasticsearch details


Index name git

Pattern No pattern 

Time field name metadata__timestamp

Version 6.0+ 

Max concurrent Shard Requests 256

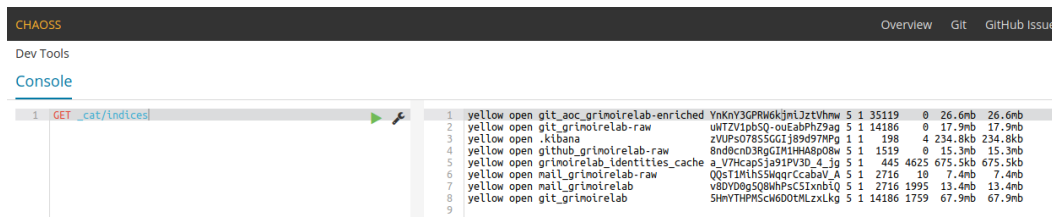
Min time interval 10s 

Save & Test

Delete

Back

Figura 4.6: Ejemplo de configuración de índice



Index	Type	Repository	Size
1	yellow open	git_aoc_grinoirelab-enriched	YnKnV3GPRW6k3nLJztVhnmw 5 1 35119 0 26.6mb 26.6mb
2	yellow open	git_grinoirelab-raw	uMTZV1pb5Q-ouEabphZ9ag 5 1 14186 0 17.9mb 17.9mb
3	yellow open	.kibana	zVUPs0785SGGIj89d97NPG 1 1 198 4 234.8kb 234.8kb
4	yellow open	github_grinoirelab-raw	8nd8cn03rgGIMH48g08w 5 1 1519 0 15.3mb 15.3mb
5	yellow open	grinoirelab_identities_cache	a_V7Hcap5ja91PV3D_A_jg 5 1 445 4625 675.5kb 675.5kb
6	yellow open	mail_grinoirelab-raw	Q0sT1Mihs55WqqrCcabaV_A 5 1 2716 10 7.4mb 7.4mb
7	yellow open	mail_grinoirelab	v80YD8g5Q8WHPsCS1xnbtQ 5 1 2716 1995 13.4mb 13.4mb
8	yellow open	git_grinoirelab	SHmYTHPMScW6D0UMLxLkg 5 1 14186 1759 67.9mb 67.9mb
9			

Figura 4.7: Listado de los índices recopilados hasta el momento

realizar *queries* directamente a los diferentes índices. En la Figura 4.7 se muestra un ejemplo de uso.

Uno de los aspectos más interesante es la diferenciación entre índices enriquecidos y no enriquecidos:

- **Índices crudos (no enriquecidos):** cada entrada corresponde con una entrada extraída de la API de Parceval, guardándose como un documento JSON con diferentes metadatos. En la mayoría de casos, pueden tener una estructura de datos demasiado compleja o incómoda para su análisis directo.
- **Índices enriquecidos:** cada ítem corresponde a una abstracción más útil de los anteriores, para un tipo concreto de análisis o visualización. Mientras que los índices crudos contienen toda la información de la fuente original (incluyendo su estructuración), los enriquecidos contienen una información mejor estructurada (simplicidad), más plana y más útil para la mayoría de los análisis de datos.

En este proyecto, por tanto, nos hemos centrado en analizar los índices enriquecidos, por dos principales motivos: son mucho más manejables (más sencillez y solo información más útil) y son extrapolables, es decir, se pueden utilizar otras bases de datos, ya que todos tienen la misma estructura, los mismos *documentos*.

En la Figura 4.8, se puede observar que mediante una *query*, podemos obtener todos los índices de la base de datos utilizados hasta el momento (tanto crudos como enriquecidos). Vienen separados además por repositorios de *git*, *github* y *pipermail* (o listas de correo).

4.4.4. Familiarización con la interfaz

Antes de continuar con la creación de dashboards, explicaremos de manera simple como se distribuye la interfaz de Grafana. Podremos diferenciar tres zonas: superior (diferenciada en dos

The screenshot shows the CHAOSS Dev Tools interface. The 'Console' tab is active, displaying a query and its results. The query is a GET request to the Elasticsearch index 'git-grinoirelab/items/_search' with a match query for 'Author_org_name: "Bitergia"'. The results show a single hit with a score of 0.46710923. The hit details include metadata, project information, commit details, and author information.

```

1 GET git-grinoirelab/items/_search
2 {
3   "query": {
4     "match": {
5       "Author_org_name": "Bitergia"
6     }
7   }
8 }
9
10
11 {
12   "took": 2,
13   "timed_out": false,
14   "_shards": {
15     "total": 5,
16     "successful": 5,
17     "skipped": 0,
18     "failed": 0
19   },
20   "hits": {
21     "total": 9052,
22     "max_score": 0.46710923,
23     "hits": [
24       {
25         "_index": "git-grinoirelab",
26         "_type": "items",
27         "_id": "c3d49f68804a50c3e6eae6f1c43285b06ccc787c",
28         "_score": 0.46710923,
29         "_source": {
30           "Author_gender": "Unknown",
31           "denography_min_date": "2016-01-20T18:04:30.000Z",
32           "metadata_gelk_backend_name": "GitEnrich",
33           "tz": 1,
34           "project": "GrinoireLab",
35           "metadata_timestamp": "2019-11-21T18:31:05.190620+00:00",
36           "uuid": "c3d49f68804a50c3e6eae6f1c43285b06ccc787c",
37           "Author_user_name": "",
38           "Commit_id": "c780b9e5b06174118a9dbc1b738e9cd0e5fa3566",
39           "Commit_user_name": "",
40           "author_date": "2017-02-20T17:43:48",
41           "metadata_enriched_on": "2019-11-21T18:57:33.719424+00:00",
42           "Author_domain": "gmail.com",
43           "author_bot": false,
44           "tag": "https://github.com/chaoss/grinoirelab-sigils",
45           "message_analyzed": "Add files via upload",
46           "author_org_name": "Bitergia",
47           "hash_short": "b4987d",
48           "commit_date": "2017-02-20T17:43:48",
49           "repository_labels": [],
50           "Author_org_name": "Bitergia",
51           "project_id": "GrinoireLab",
52           "Author_uuid": "081bbdada20a17b4130636e0c42204b9e302fd02",
53           "offset": null,
54           "lines_added": 0,
55           "committer_domain": "github.com",
56           "Commit_uuid": "8ab10d6173e6ca8ff0e03f8bf1dd70c25257ba43",
57           "branches": [],
58           "committer_name": "GitHub",
59           "Commit_gender": "Unknown",
60           "Author_gender_acc": 0,
61           "Commit_org_name": "Unknown",
62           "files": 1,
63           "metadata_gelk_version": "0.55.0",
64           "author_id": "a7c4988a7b78559f40ac99131a9329c33ac2ce70",
65           "hash": "b4987d4689bfcf9555ab58d34e15d5cda5af98ed",

```

Figura 4.8: Ejemplo de query en Elasticsearch

partes), lateral y central, y cada botón corresponde con una o más funcionalidades:



Figura 4.9: Interfaz de Grafana (menús superiores)

1. Botón Home de Grafana. Al pulsarlo, nos enviará de vuelta al menú principal de la aplicación, donde tenemos nuestro *Home Dashboard*.
2. Desplegable de dashboards. Cambiará su nombre dependiendo del dashboard en el que nos encontremos. Al seleccionarlo, nos encontramos con un menú desplegable donde están nuestros dashboards ordenados por carpetas, recientes y favoritos. Además, nos permite filtrar y buscar los ya existentes; o crear, buscar o importar nuevos.
3. Añade un nuevo panel justo al principio del dashboard.
4. Añade el dashboard con el que estamos trabajando a favoritos. Esto permite poder filtrarlos y agruparlos después por esta categoría, teniendo así los paneles más recurrentes a mano.
5. Comparte el dashboard en cuestión de tres formas diferentes: creando un link estático a tu aplicación (cuidado si se trabaja en local), un *snapshot* o captura plana del mismo, o bien exportándolo como archivo JSON para poder importarlo en un futuro (más información en la Sección 4.8.1).
6. Se guarda la información del dashboard con el que se está trabajando (un atajo es la combinación **CTRL+S**).
7. Menú de opciones de dashboard. Permite cambiar diferentes aspectos de ese conjunto de paneles: podemos cambiar su nombre, rango de tiempos, crear anotaciones¹², crear variables¹³, cambiar permisos¹⁴ de lectura/escritura por usuario/grupo, y un largo etc.

¹²<https://grafana.com/docs/reference/annotations/>

¹³<https://grafana.com/docs/reference/templating/>

¹⁴<https://grafana.com/docs/v5.1/administration/permissions/>

8. Menú de home (1) en barra lateral. Dependiendo de la versión podemos encontrarlo en el top o en el lateral.
9. Menú de creación. Permite tres opciones: crear un nuevo dashboard, crear una nueva carpeta para agruparlos, o bien importar otros dashboards desde url o documento JSON.
10. El desplegable de dashboards permite también tres opciones: **manipulación** (mover, agrupar y filtrar), crear **playlists**¹⁵ de los ya existentes (creando una presentación en bucle de varios grupos de paneles) o visionar los **snapshots** existentes en nuestra aplicación.
11. Permite explorar los diferentes *Data Sources* añadidos, pero sin crear ningún dashboard¹⁶.
12. El botón de manejo de **alertas** de Grafana nos permite visualizar y configurar las diferentes alertas creadas por el usuario. El apartado de notificaciones se aborda en profundidad en los siguientes sprints.
13. El menú de configuración general permite multitud de opciones: añadir nuevas fuentes de datos, añadir nuevos usuarios con diferentes roles (**se profundiza en Sección xx**), nuevos equipos (para gestionar permisos más fácilmente), administrar plugins, editar preferencias básicas o configurar *API Keys* para extraer información de otras aplicaciones.
14. Este es el menú de administración del servidor. Nos permite comprobar diversos aspectos como pueden ser el número de usuarios totales creados, la configuración inicial de arranque de la herramienta o las estadísticas generales de Grafana (nº total de dashboards, usuarios activos, alertas, etc.).
15. En el menú de usuario podemos configurar las preferencias del usuario actual, con el que hemos iniciado sesión (cambio de nombre, correo, tema, cambio de contraseña, *logout*, etc.).
16. Este es el botón de ayuda. Desde aquí podemos acceder a los atajos de la herramienta, la documentación de Grafana o al foro de la comunidad.
17. Panel de *stat* básico en Grafana. Se profundiza en la **Sección 5.xx**.

¹⁵<https://grafana.com/docs/reference/playlist/>

¹⁶<https://grafana.com/docs/features/explore/>

18. Herramienta de manejo temporal de un panel concreto. Se puede cambiar el rango de tiempo de cada panel de manera individual. Se analiza en profundidad en la **Sección 5.xx**.
19. Plugin de Grafana. En este caso es el complemento *Pie Chart*. Se profundiza en este aspecto en la **Sección 5.XX**.
20. Leyenda del panel. Hay dos aspectos relevantes: si se pulsa sobre el color, podemos cambiar la distribución de estos en el grafo, individualmente o de manera grupal. O bien, si se pulsa sobre el texto, podemos mostrar solo la serie cliqueada, o mostrar un conjunto de ellas.



Figura 4.10: Interfaz de Grafana (menús laterales)

4.5. Sprint 3 - Dashboards y plugins de Grafana

Una vez obtenida una idea de la estructuración de los índices, la siguiente tarea fue la de crear los primeros paneles. Primeramente, nos servimos de guía en los dashboards ofrecidos por el contenedor de GrimoireLab. Crearemos paneles similares a estos, para posteriormente comparar ambas herramientas.

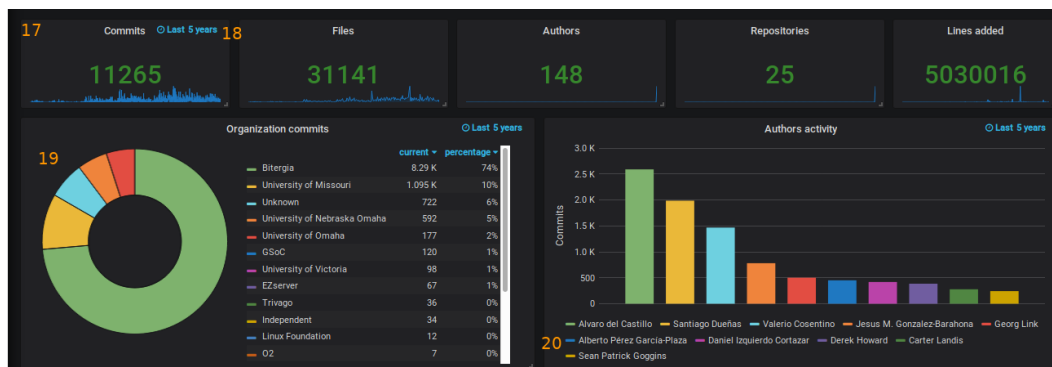


Figura 4.11: Interfaz de Grafana (central) y ejemplo de dashboard

En este primer acercamiento y en posteriores sprints se utilizaron los índices de *git*, *mailing list* y *areas of code*, que corresponden a `git-grimoirelab`, `mail-grimoirelab` y `git-areas-of-code` en la tabla de índices, respectivamente.

Para crear nuestro **primer dashboard**, pulsamos sobre el botón + ->New Dashboard ubicado en el menú lateral. A continuación, se nos mostrará un nuevo dashboard vacío, donde podremos elegir entre elegir la **query** o la **visualización** a utilizar, o bien convertir este nuevo panel en una fila clasificatoria (*row*). Escogemos por ejemplo la visualización, y se nos mostrará el abanico de posibilidades de las que dispone la herramienta. Para un primer ejemplo, podemos elegir un *Graph Panel*. A continuación, una vez elegido el grafo, procedemos a realizar la query, la cual tiene la siguiente estructuración:

- **Queries to:** Datasource a utilizar, el cual debe ser previamente añadido (ver Sección 4.4.3).
- **Query:** Se puede utilizar una query de tipo Lucene¹⁷ para filtrar más concretamente lo que se mostrará en el panel (por ejemplo utilizando RegEx¹⁸ al buscar por nombres).
- **Metric:** Aquí se elige la métrica a utilizar. La herramienta ofrece múltiples opciones: conteo de elementos individual, media de estos, diferenciación, suma total, min/max, etc.
- **Group by:** En este bloque se agruparán los elementos para su clasificación. Tenemos las siguientes opciones: por término (*field* en la base de datos), filtro (subquery para un

¹⁷<https://www.elastic.co/guide/en/kibana/current/lucene-query.html>

¹⁸https://en.wikipedia.org/wiki/Regular_expression

conjunto de *fields*), histograma de tiempo (siendo necesario un *field* de tipo *time*) o histograma estándar. Además, si se elige agrupación por término/filtro, se puede ordenar todo de manera ascendente/descendente, permite elegir un tamaño máximo de campos a mostrar, etc.

- **Then by:** Una segunda agrupación. Se puede utilizar por ejemplo cuando queremos dos filtros, uno para agrupar por términos y otro para redistribuir estos a lo largo del tiempo (ver gráfico de barras de la Figura 4.11). Además, si agrupamos por tiempo, permite fijar un intervalo de tiempo sobre el que filtrar.
- **Time:** Dado el *field* de tipo tiempo elegido al principio, aquí podemos filtrar su intervalo de agrupación (por minutos, horas, días...), su tiempo relativo de búsqueda (todos los campos del último día, mes, año...), si el eje de tiempo empezará desplazado o si se nos muestra la información temporal en la esquina del panel (ver Apartado 18 de la Figura 4.11).

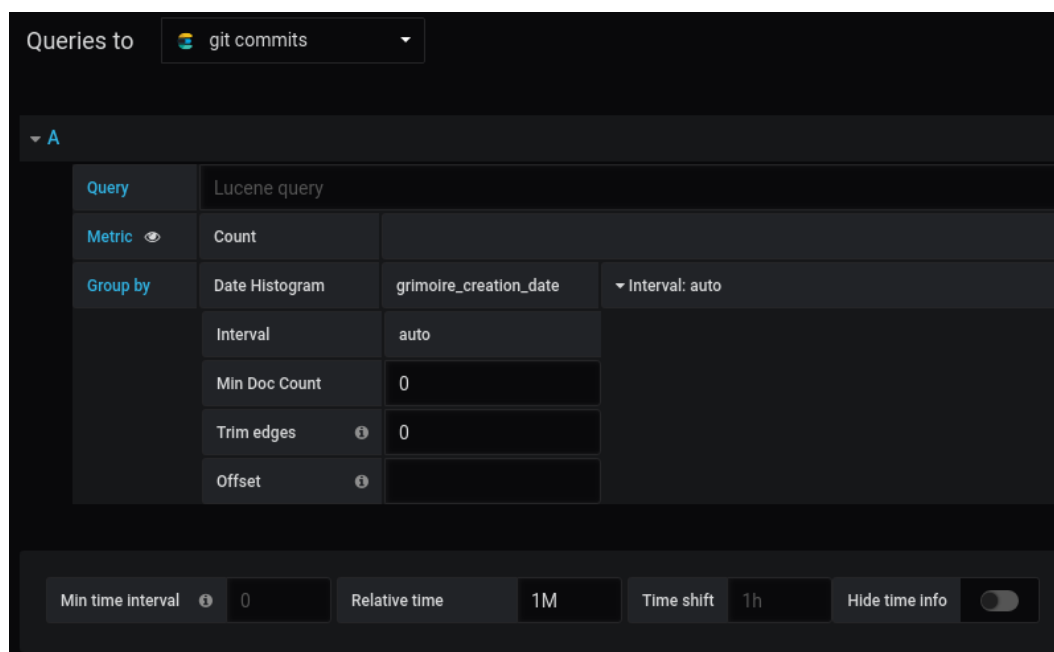


Figura 4.12: Ejemplo de query para panel de Grafana

En cuanto al apartado de *Visualization*, cada panel tiene decenas de posibilidades de cambios de diseño, destacando aspectos como:

- Uso de barras, líneas o puntos en gráficos.

- Personalización de los ejes X e Y (nombres, unidades, mínimos, máximos...).
- Personalización de la leyenda (agrupación, colocación, ocultación...).
- Uso de *Thresholds* o límites al sobrepasar un valor dado.

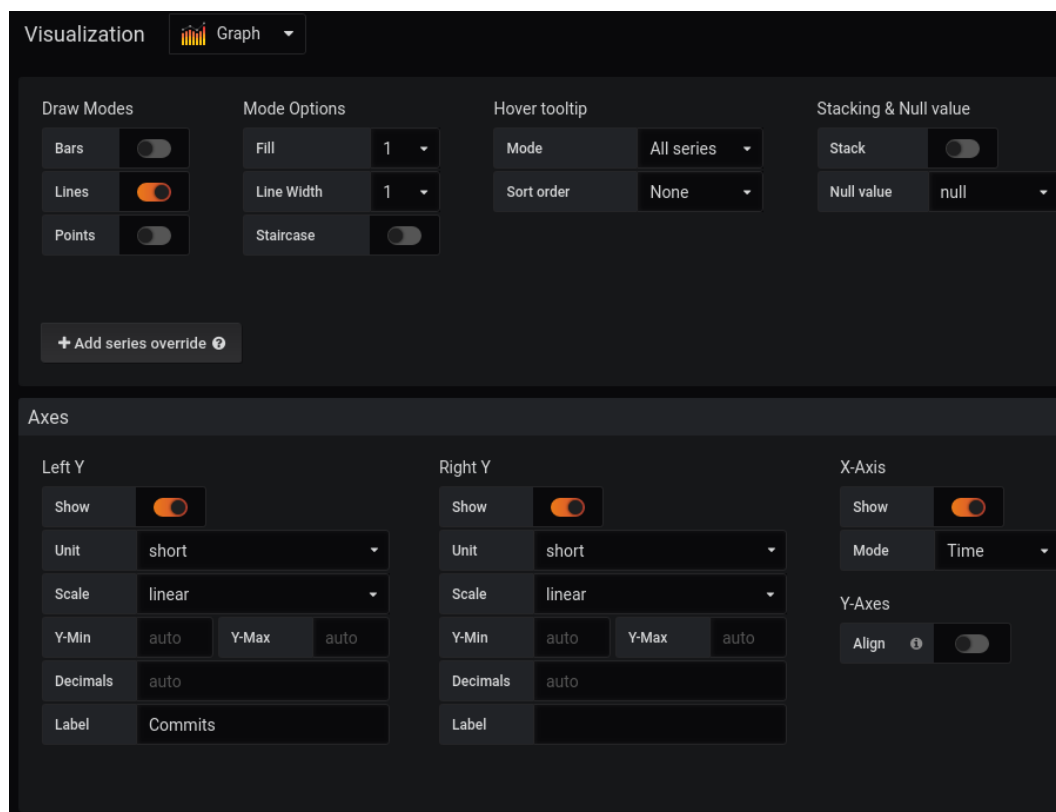


Figura 4.13: Ejemplo de configuración visual de panel tipo *Graph*

El resto de apartados permiten cambiar el título o la descripción del dashboard, o agregarle alertas para gráficos en tiempo real (aunque este no es nuestro caso).

Una vez configurado todo correctamente, regresamos a la ventana anterior y ya tendremos nuestro primer panel, que será algo similar a lo mostrado en la Figura 4.14.

4.5.1. Plugins

Uno de los aspectos más distintivos de Grafana es el uso de los plugins¹⁹ creados y mantenidos por la **comunidad**. Hay infinidad de complementos para la herramienta: la mayoría son **nuevos paneles**, otros son para la posibilidad de añadir **nuevas fuentes de datos** de BBDD

¹⁹<https://grafana.com/grafana/plugins?orderBy=weight&direction=asc>

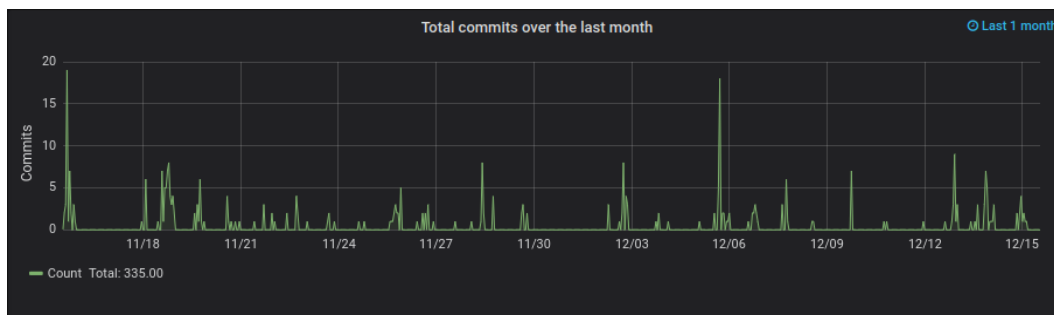


Figura 4.14: Ejemplo final de *Graph Panel*

no permitidas por defecto en la herramienta (Prometheus²⁰, Akumuli²¹, Stackdriver²²...) y otros son **apps complementarias** a Grafana, que añaden funcionalidad adicional.

En este proyecto nos centramos principalmente en la primera opción, ya que nuestra fuente de datos es Elasticsearch, y la herramienta por defecto la soporta. Se exploran también la tercera opción, incluyendo una app en la herramienta, la cual se aborda en la **Sección 5.xx (ALERTING)**.

Instalar un plugin en Grafana es muy sencillo. Basta con elegir uno de entre su catálogo y seguir las instrucciones en la pestaña con el correspondiente nombre. El método más sencillo de instalación es mediante el comando `grafana-cli`²³. Por poner un ejemplo, para añadir el plugin *Pie Chart*, basta con escribir en un terminal lo siguiente (con el servidor de Grafana ya lanzado):

```
grafana-cli plugins install grafana-piechart-panel
```

PENSAMIENTO DE MUDAR ESTE LISTADO A LA SECCIÓN 5 - ZONA PLUGINS Desde este sprint en adelante se han instalado, analizado y probado diversos complementos, para finalmente quedarnos con un total de **20 plugins**, que son los que nos han parecido más relevantes/útiles:

- **Pie Chart:** Típico gráfico tipo *tarta* o tipo *donut* (intercambiable). No venía por defecto en Grafana, como si viene en Kibana.
- **Worldmap Panel:** Permite representar datos de tipo serie temporal o *geohash* (coor-

²⁰<https://prometheus.io/>

²¹<https://akumuli.org/>

²²<https://cloud.google.com/stackdriver/>

²³<https://grafana.com/docs/grafana/latest/administration/cli/>

nadas) en un mapa del mundo. Es obligatorio que en cada documento de Elasticsearch se especifique su latitud y longitud para su posterior representación.

- **Annunciator:** Versión mejorada del panel `SingleStat` (por defecto en Grafana). Es especialmente útil para la monitorización y control, ya que puede representar thresholds, límites superiores/inferiores, variaciones de color...
- **Blendstat:** Muy similar de nuevo a `SingleStat`, diferenciándose en que permite representar múltiples series en el mismo panel.
- **Carpet plot:** Diagrama de alfombra o diagrama de calor (*heatmap*). El panel recibe una serie de datos temporal y la divide en diferentes fragmentos o *buckets*. Agrupa los datos día a día para el eje X, y después en fragmentos de día para el eje Y (horas / minutos / rango).
- **Bubble Chart:** Gráfico de burbujas. Los círculos se agrupan de manera descendente partiendo del centro, y su tamaño y color dependen del valor agregado de una métrica concreta dada una serie de datos temporal.
- **Datatable Panel:** Parte de la base de la tabla por defecto de Grafana, pero mejorando sus características. La principal diferencia es que posee un cuadro de búsqueda y permite filtrar la tabla, entre otras virtudes.
- **Diagram:** Permite crear diagramas de flujo, de secuencia y de *Gantt*²⁴, mediante el uso de la librería `mermaid.js`²⁵.
- **Epict Panel:** Este panel permite elegir una imagen especificando su URL y mostrarla con métricas sobre ella. Es útil para mostrar por ejemplo el logo de una empresa en cada dashboard, a modo de marca de agua.
- **PictureIt:** Al igual que el panel anterior, permite añadir métricas sobre una imagen de fondo. Permite además la personalización visual de la misma (recuadro, color de letra, de fondo...).

²⁴https://es.wikipedia.org/wiki/Diagrama_de_Gantt

²⁵<https://mermaidjs.github.io/>

- **Geoloop:** Este panel proporciona un mapa de calor sobre una zona o mapa real, en forma de bucle animado. Al igual que `Worldmap`, es necesario que en cada documento se especifique latitud/longitud y que la fuente de datos sea una serie de datos temporal (para la animación en bucle).
- **Progress List:** Gráfico de tipo barras laterales, a modo de lista de progreso. Es similar a un conjunto de simples paneles `Singlestat`, contando con múltiples métricas a la vez.
- **BreadCrumb:** Plugin útil para la interfaz gráfica de Grafana. Se trata de un típico *breadcrumb*²⁶ o hilo de Ariadna tan utilizado en páginas webs o exploradores de archivos. Es una línea de texto en la que se indica el recorrido seguido por el usuario y la forma de regresar.
- **Boom Theme:** De nuevo otro plugin gráfico. Nos permite cambiar el fondo de nuestro dashboard por cualquier imagen que le indiquemos mediante una URL, o bien usar algunos de los temas por defecto que posee.
- **Multistat:** De nuevo otro plugin que mejora el clásico panel `Singlestat`. Se utiliza para multimétricas. Además, permite bastante personalización gráfica: añadir alertas parpadeantes cuando se supera un umbral, agrupación de gráficos de barras verticales/horizontales en un mismo panel (agrupación en un atributo), etc.
- **Radar Graph:** Este panel muestra un típico gráfico de radar, usando la librería `Chart.js`²⁷. Al agrupar una métrica por término y agregación, se mostrará como de recurrente es un elemento en el índice dado.
- **Ploty:** Un típico gráfico de dispersión (*scatter plot*), con la particularidad de poder representar datos en 3 dimensiones, usando el *framework* `plot.ly`²⁸ de javascript²⁹.
- **Polistat Panel:** Nos permite asignar un poliedro (en este caso hexágonos) a cada métrica recibida. Permite agrupar métricas, y mostrar así una fusión de las mismas en un único

²⁶https://en.wikipedia.org/wiki/Breadcrumb_navigation

²⁷<https://www.chartjs.org/>

²⁸<https://plot.ly/>

²⁹<https://developer.mozilla.org/es/docs/Web/JavaScript>

hexágono. Permite además configurar la distribución de los hexágonos a mostrar, escalado según si un dato aparece más o menos, cambiar cada color por separado, etc.

- **Statusmap:** Panel que nos permite agrupar los valores de una o varias métricas en filas o contenedores, a modo de mapa de estado de la métrica. Dependiendo del valor de la métrica, se puede configurar el *bucket* para que muestre un color u otro.
- **Clock:** Plugin de tipo reloj digital. Es realmente útil si por ejemplo se quieren utilizar varios dashboards en tiempo real, que se correspondan con BBDD de diferentes países. Permite además ser utilizado como cuenta atrás, actualizándose a antojo del usuario (cada segundo, minuto, hora...).

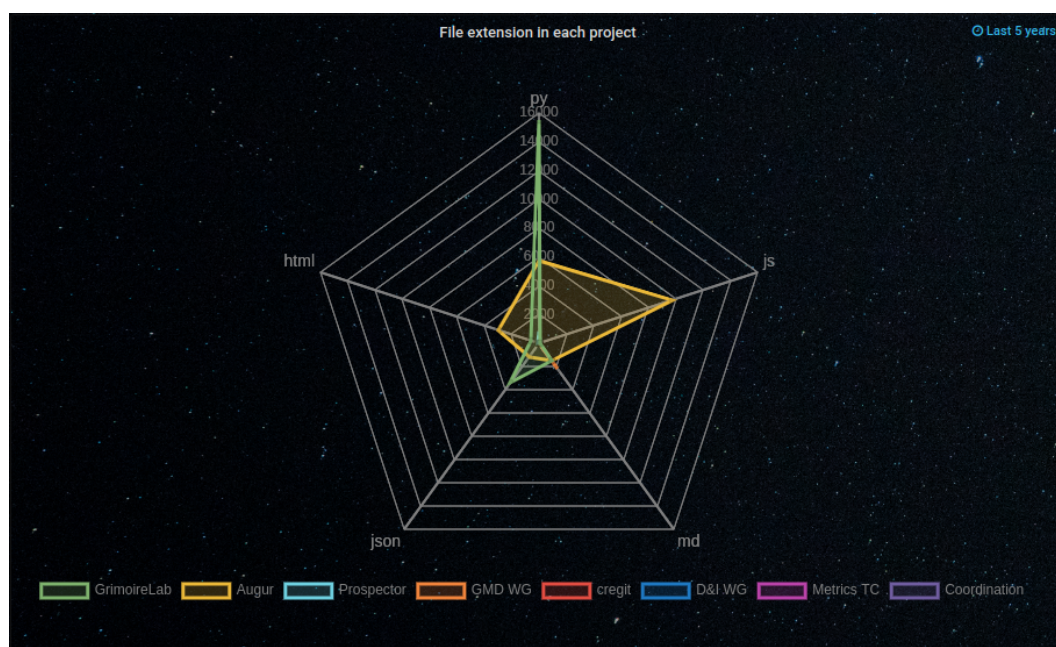


Figura 4.15: Ejemplo del plugin *Radar Graph*

4.6. Sprint 4 - Dashboards funcionales y comparación

En esta parte del proyecto se continuó el proceso explicado anteriormente. Se crearon dashboards completamente funcionales, tomando como ejemplo los proporcionados en Kibana. Para ello, se tomaron los índices de *git*, *mailing list*, *areas of code* y *github*, que corresponden a *git-grimoirelab*, *mail-grimoirelab*, *git-areas-of-code* y *github-grimoirelab*,

respectivamente. Esto sirvió además para la demo mostrada en la Sección ?? **REF HERE!!!!** y para poder realizar una comparación entre ambas herramientas (virtudes y defectos), la cual se aborda en la Sección ?? **REF HERE!!!!**.

Por otro lado, se siguió explorando el resto de plugins, sobre todo los más complicados a primera vista, o los de más difícil configuración (*Graph Radar*, *Multistat*, *Polistat*...). También se hizo hincapié en los filtros de Grafana:

- Selección específica en tablas (jugando con +/- , cuadros de búsqueda y *Regex*).
- Modificación del eje de tiempo en gráficas (arrastrando o seleccionando zona concreta en el eje X). Se comprueba así que cada panel puede ser independiente del resto.
- Manipulación del rango de tiempos de cada panel o de manera general (barra superior, ver Figura 4.9).

4.7. Sprint 5 - Docker Hub y contenedor de GrimoireLab/-Grafana

El objetivo en este sprint fue conseguir crear una nueva ***docker image*** incluyendo a Grafana, partiendo de la base de la imagen de contenedor de GrimoireLab. Este nuevo docker tendrá todas la configuración precargada, y contará con los dashboards hechos hasta el momento (aunque en un futuro se fue actualizando con los nuevos añadidos).

Pero para poder realizar todo esto, hubo que recurrir primeramente al manual de DockerHub, para familiarizarnos con los comandos y archivos necesarios para construir nuestra imagen. Se utilizaron principalmente dos fuentes: la proporcionada por la documentación oficial³⁰, y un pequeño manual en Github³¹. Leyendo ambos manuales queda claro que sí o sí necesitábamos usar el comando `docker build` y un nuevo archivo llamado ***Dockerfile***. Este fichero es la clave de todo, pues en él se encuentran todas las **instrucciones** necesarias para instalar e incluir todas las **dependencias** para que nuestro contenedor se despliegue de manera adecuada (Elasticsearch, Kibana, Grafana, configuración de GrimoireLab...).

³⁰<https://docs.docker.com/engine/reference/commandline/build/>

³¹[https://odewahn.github.io/docker-jumpstart/building-images-with-dockerfiles.](https://odewahn.github.io/docker-jumpstart/building-images-with-dockerfiles.html)

Pero tal y como se ha comentado al principio, partimos de una base, de la imagen de contenedor de GrimoireLab. Sus dependencias, por tanto, se encuentran ubicadas en el repositorio del proyecto en Github³². El siguiente paso fue clonar este repositorio y buscar e inspeccionar su Dockerfile. Los ficheros asociados al docker se encontraban en el directorio con el mismo nombre (/docker), en la raíz del repositorio. El fichero Dockerfile elegido como referencia fue `Dockerfile-full`, ya que es el utilizado para construir la imagen utilizada en nuestro proyecto (`grimoirelab-full`).

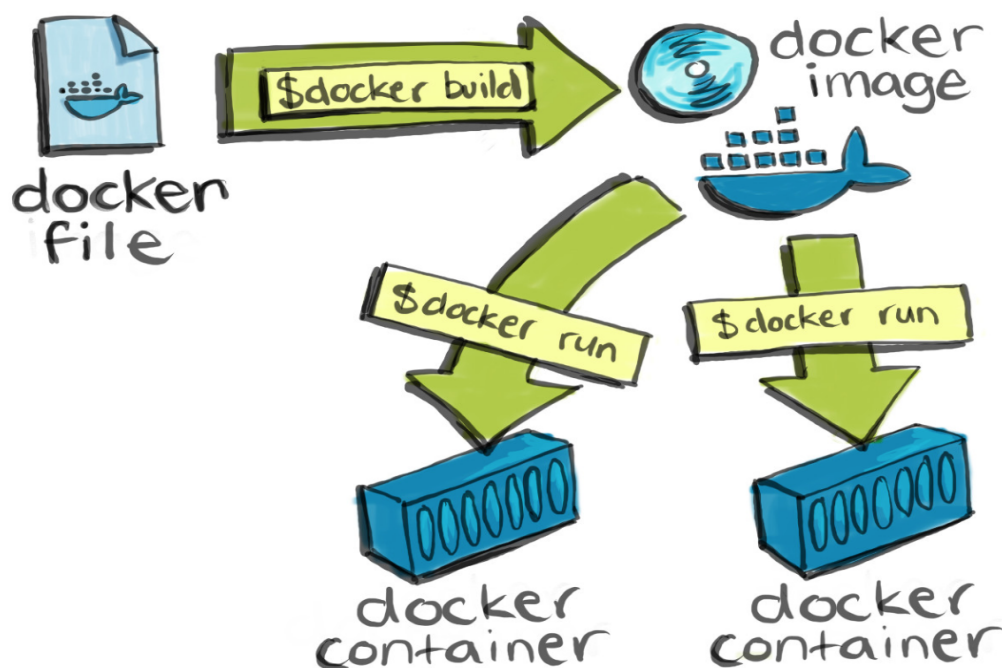


Figura 4.16: Funcionamiento de docker con Dockerfile

4.7.1. Creando un nuevo Dockerfile

Al abrir el fichero mencionado anteriormente, `Dockerfile-full`, nos encontramos con un conjunto de instrucciones y comandos familiares, explicados en el manual de la sección anterior. De todos los que se explican, nosotros utilizamos únicamente algunos de ellos:

- **RUN:** Ejecuta un nuevo comando y guarda el resultado en una nueva capa.
- **ADD:** Copia un fichero de la máquina *host* al contenedor (de la forma `ADD source destination`).

³²<https://github.com/chaoss/grimoirelab>

- **ENV**: Establece una nueva variable de entorno en el nuevo contenedor.
- **USER**: Establece el usuario por defecto dentro del contenedor (para uso de *user root* por ejemplo).
- **ENTRYPOINT**: Define el comando que se ejecutará por defecto cuando el contenedor se lance.
- **CMD**: Se usa para ejecutar comandos al construir un nuevo contenedor desde una *docker image*.

Para nuestro caso, era necesario instalar Grafana y conseguir que la herramienta se desplegara en el nuevo contenedor. A este nuevo fichero le llamamos **Dockerfile-Grafana**, y un ejemplo de su contenido añadido es el mostrado en la Figura 4.17.

```
82 #Install Grafana
83 RUN wget https://dl.grafana.com/oss/release/${GRAF}_amd64.deb && \
84     sudo dpkg -i ${GRAF}_amd64.deb && \
85     rm ${GRAF}_amd64.deb
```

Figura 4.17: Extracto de código agregado al nuevo Dockerfile

Una vez realizado el paso anterior, la tarea fue crear también un nuevo fichero **Entrypoint**. Tal y como se ha explicado antes, este archivo es el que se ejecutará al lanzar el contenedor. Este será un **archivo ejecutable**, que actuará a modo de *log file*, mostrando en un terminal los eventos que van ocurriendo en tiempo real con respecto a las aplicaciones que se van ejecutando. Un ejemplo de esto es el mostrado en la Figura 4.4 (aunque entonces no sabíamos que eso era nuestro entrypoint).

Por lo tanto, partimos también de un archivo base, llamado **entrypoint-full.sh**, ubicado también en *grimoirelab/docker*. El propósito era registrar los eventos que ocurrían al ejecutar las nuevas aplicaciones añadidas al contenedor. Un ejemplo del código es el mostrado por la Figura 4.18.

4.7.2. Creación de una nueva *docker image*

Tras realizar los pasos anteriores, ya tenemos todo listo para poder crear la nueva imagen. Para ello hay que hacer uso del comando `docker build`. Si se consulta el manual³³, pode-

³³<https://docs.docker.com/engine/reference/commandline/build/>

```

48 # Start Grafana
49 echo "Starting Grafana"
50 sudo chown -R grafana.grafana /etc/grafana
51 sudo service grafana-server start
52 until $(curl --output /dev/null --silent --head --fail http://127.0.0.1:3000); do
53     printf '.'
54     sleep 2
55 done
56 echo "Grafana started"

```

Figura 4.18: Extracto de código agregado al nuevo Entrypoint

mos observar que necesitamos al menos un parámetro: el *path* donde se ubica el Dockerfile a utilizar. En nuestro caso, además, añadimos un segundo parámetro opcional (*flag -t*), para dar nombre y tag a nuestra imagen y así tener un correcto nombre de repositorio:

```

1 docker build -f $(pwd)/docker/Dockerfile-grafana\
2 -t grimoirelab/grafana:latest .

```

Si todo está correcto, se nos irán mostrando diferentes logs según se van ejecutando los comandos del fichero, y al final un mensaje del estilo *Successfully built at ****, indicando que la imagen se creó satisfactoriamente. El siguiente paso, por tanto, es lanzar el contenedor y probar si Grafana inicia correctamente:

```

1 docker run -p 127.0.0.1:5601:5601 -p 127.0.0.1:9200:9200\
2 -p 127.0.0.1:3306:3306 -p 127.0.0.1:3000:3000 \
3 -v $(pwd)/credentials.cfg:/override.cfg \
4 -t grimoirelab/grafana

```

Gracias a los logs del *entrypoint* sabremos si el servicio de Grafana se habrá iniciado, tal y como se muestra en la Figura 4.19. Tendremos entonces disponible la herramienta donde siempre, en `localhost:3000`.

4.8. Sprint 6 - Actualización del contenedor de Grafana

Lo realizado en el sprint anterior nos permitía ejecutar un contenedor con el servicio de Grafana operativo. Aunque si nos dirigimos a `localhost:3000`, con `user/password = admin/admin`, podemos observar un menú completamente vacío. Es necesario, por tanto, **exportar la configuración** existente hasta el momento. El propósito durante este sprint es justamente eso, abordar las **dos posibles maneras** de realizar dicha exportación y su futura importación en la imagen.

```

jonycp@jonycp:~$ docker run -p 127.0.0.1:5601:5601 -p 127.0.0.1:9200:9200 -p 127.0.0.1:3306:3306 -p 127.0.0.1:3000:3000
-v $(pwd)/credentials.cfg:/override.cfg -v $(pwd)/es-data:/var/lib/elasticsearch -t grimoirelab/grafana
Starting container: 0aba9df5alf0
Starting Elasticsearch
[ ok ] Starting Elasticsearch Server:.
Waiting for Elasticsearch to start...
tcp        0      0 0.0.0.0:9200          0.0.0.0:*            LISTEN     -
Elasticsearch started
Starting MariaDB
[ ok ] Starting MariaDB database server: mysqld . . . . .
Waiting for MariaDB to start...
tcp6       0      0 :::3306              :::*                 LISTEN     -
MariaDB started
Starting Kibiter
Waiting for Kibiter to start...
tcp        0      0 0.0.0.0:5601         0.0.0.0:*            LISTEN     425/kibiter-6.1.4-1
Kibiter started
Starting Grafana
[ ok ] Starting Grafana Server:.
Grafana started

```

Figura 4.19: Logs del nuevo contenedor de Grafana

4.8.1. Exportación/importación disjunta

Exportación de dashboards

Todos los dashboards de Grafana están escritos en formato JSON. Esto es especialmente útil para su exportación, ya que en un único fichero se encuentra toda la información de cada uno de sus paneles (métricas, variables, estilos, queries, data sources utilizadas, regiones de tiempo...). La manera más sencilla de exportar uno o más dashboards es pulsando sobre el botón de *Compartir (Share)*, ubicado en la barra superior dentro de cada dashboard (ver Figura 4.9). Dentro del menú, solo queda dirigirse a *Exportar* y a *Guardar a fichero*.

Importación de dashboards

Aquí de nuevo nos encontramos con dos métodos, uno proporcionado por la interfaz de Grafana y otro por su API:

- **Interfaz** (manual): Es tan sencillo como dirigirse al menú lateral, situarse sobre + y pulsar sobre *Import*. En la nueva ventana, basta con seleccionar **Upload .json file** y elegirlo de nuestro directorio local.
- **API** (automático): Este método es más enrevesado de primeras, pero con la ventaja de que importará todos nuestros dashboards de una vez. Se hace uso del **aprovisionamiento**³⁴ que proporciona la API de la herramienta. Para ello, basta con crear un fichero `.yaml`, el cual contendrá una lista con los dashboards que Grafana cargará al iniciarse. En nuestro

³⁴<https://grafana.com/docs/administration/provisioning>

caso, esta lista se corresponde al directorio local donde exportamos previamente los dashboards, del cual daremos su *path* (ver Figura 4.20). Además, el fichero debe ir en una ruta específica (/etc/grafana/provisioning/dashboards), por lo que hemos de modificar el Dockerfile, sumando dos líneas: una para añadir el nuevo fichero y otra para agregar el directorio con los dashboards a la imagen (ver ejemplo de Figura 4.21).

```
apiVersion: 1

providers:
  # <string> an unique provider name
  - name: 'a unique provider name'
    # <int> org id. will default to orgId 1 if not specified
    orgId: 1
    # <string, required> name of the dashboard folder. Required
    folder: ''
    # <string> folder UID. will be automatically generated if not specified
    folderUid: ''
    # <string, required> provider type. Required
    type: file
    # <bool> disable dashboard deletion
    disableDeletion: false
    # <bool> enable dashboard editing
    editable: true
    # <int> how often Grafana will scan for changed dashboards
    updateIntervalSeconds: 10
    # <bool> allow updating provisioned dashboards from the UI
    allowUiUpdates: false
    options:
      # <string, required> path to dashboard files on disk. Required
      path: /var/lib/grafana/dashboards
```

Figura 4.20: Ejemplo de fichero de provisionamiento para importación de dashboards

Exportación de *Data Sources*

Para trasladar las fuentes de datos de Grafana necesitamos hacer uso de nuevo de su API. Por tanto, para exportar todo a un fichero llamado `data-sources` (por ejemplo), usaremos la siguiente sentencia:

```
1 mkdir -p $(pwd)/data-sources &&\
```

```
2 curl -s "http://localhost:3000/api/datasources"\
3 -u admin:12341829as|jq -c -M '.[]'|split -l 1 - $(pwd)/data-sources/
```

Donde `admin:12341829as` son `user:password` en mi caso. Con esto, se nos irán mostrando diversos mensajes en el terminal indicando que fuentes se están exportando, y un mensaje final si todo ha salido correctamente.

Importación de *Data Sources*

El proceso inverso al anterior se consigue utilizando también la API, y mediante un pequeño script³⁵, el cual debemos añadir a nuestro *Entrypoint* para su aplicación al inicio:

```
1 for i in /data-sources/*; do \
2     $(curl --output /dev/null\
3     -X "POST" "http://127.0.0.1:3000/api/datasources"\
4     -H "Content-Type: application/json"\
5     --user admin:admin --data-binary @$i)\
6     echo "Added database $i"
7 done
```

Además, debemos modificar también nuestro *Dockerfile*, para poder usar el directorio previamente creado (`data-sources`) en nuestra imagen (ver Figura 4.21).

4.8.2. Exportación/importación abrupta

Si nuestro propósito es realizar un volcado total de nuestra **configuración**, incluyendo usuarios, permisos, etc., lo ideal es utilizar este tipo de exportación-importación.

Grafana dispone de una base de datos propia donde guarda toda su estructuración, ubicada en `/var/lib/grafana/grafana.db`. Si copiamos este fichero y lo añadimos a nuestra imagen (ver Figura 4.21), al iniciar la herramienta en el contenedor todo estará tal cual lo dejamos. Es un método tosco pero efectivo, aunque puede no ser lo ideal, ya que no siempre se requiere un volcado total de la DB de Grafana. Para focalizar que exportar y que importar, se recurrirá siempre al primer método (disjunto).

³⁵<https://rmoff.net/2017/08/08/simple-export/import-of-data-sources-in-grafana/>

```

87 #Installing Grafana Plugins
88 RUN sudo grafana-cli plugins install grafana-piechart-panel
89 RUN sudo grafana-cli plugins install grafana-worldmap-panel
90 RUN sudo grafana-cli plugins install raintank-worldping-app
91 RUN sudo grafana-cli plugins install michaelmoore-annunciator-panel
92 RUN sudo grafana-cli plugins install farski-blendstat-panel
93 RUN sudo grafana-cli plugins install petrslavotinek-carpetplot-panel
94 RUN sudo grafana-cli plugins install digrich-bubblechart-panel
95 RUN sudo grafana-cli plugins install briangann-datatable-panel
96 RUN sudo grafana-cli plugins install jdbramham-diagram-panel
97 RUN sudo grafana-cli plugins install larona-epict-panel
98 RUN sudo grafana-cli plugins install citilogics-geoloop-panel
99 RUN sudo grafana-cli plugins install bessler-pictureit-panel
100 RUN sudo grafana-cli plugins install corpglory-progresslist-panel
101 RUN sudo grafana-cli plugins install digiapulssi-breadcrumb-panel
102 RUN sudo grafana-cli plugins install yesoreyeram-boomtheme-panel
103 RUN sudo grafana-cli plugins install michaelmoore-multistat-panel
104 RUN sudo grafana-cli plugins install snuids-radar-panel
105 RUN sudo grafana-cli plugins install natel-plotly-panel
106 RUN sudo grafana-cli plugins install grafana-polystat-panel
107 RUN sudo grafana-cli plugins install flant-statusmap-panel
108 RUN sudo grafana-cli plugins install grafana-clock-panel
109
110 #Adding files
111 ADD docker/entrypoint-full.sh /entrypoint.sh
112 ADD docker/grafana.db /var/lib/grafana/
113 #ADD docker/data-sources /data-sources
114 #ADD docker/dashboards /dashboards
115 #ADD docker/dashboards-conf.yaml /etc/grafana/provisioning/dashboards
116
117 USER root
118 RUN chmod 777 /entrypoint.sh
119 #RUN chmod -R 755 /data-sources
120 #RUN chmod -R 755 /dashboards
121 #VOLUME /var/lib/elasticsearch

```

Figura 4.21: Extracto de nuevo Dockerfile (utilizando método de exp/imp abrupta)

4.9. Sprint 7 - Imagen final y contenedores múltiples

EN DESARROLLO Analizar un proyecto específico (repos de ES, Grafana oficiales) y utilizar exactamente el mismo contenedor pero cambiando projects.json Posibilidad de desplegar varios contenedores con proyectos diferentes. Para ello, guardar en local lo que recogemos al desplegar.

DB guardada localmente (da ciertos errores en terminal y no se visualiza todo en grafana):

```

1 docker run -p 127.0.0.1:5601:5601\
2 -v $(pwd)/credentials.cfg:/override.cfg\
3 -v $(pwd)/es-data:/var/lib/elasticsearch
4 -t grimoirelab/full

```


Capítulo 5

Resultados

En este capítulo se incluyen los resultados de tu trabajo fin de grado.

Si es una herramienta de análisis lo que has realizado, aquí puedes poner ejemplos de haberla utilizado para que se vea su utilidad.

Capítulo 6

Conclusiones

6.1. Consecución de objetivos

Esta sección es la sección espejo de las dos primeras del capítulo de objetivos, donde se planteaba el objetivo general y se elaboraban los específicos.

Es aquí donde hay que debatir qué se ha conseguido y qué no. Cuando algo no se ha conseguido, se ha de justificar, en términos de qué problemas se han encontrado y qué medidas se han tomado para mitigar esos problemas.

6.2. Aplicación de lo aprendido

Aquí viene lo que has aprendido durante el Grado/Máster y que has aplicado en el TFG. Una buena idea es poner las asignaturas más relacionadas y comentar en un párrafo los conocimientos y habilidades puestos en práctica.

1. a

2. b

6.3. Lecciones aprendidas

Aquí viene lo que has aprendido en el Trabajo Fin de Grado/Máster.

1. a

2. b

6.4. Trabajos futuros

Ningún software se termina, así que aquí vienen ideas y funcionalidades que estaría bien tener implementadas en el futuro.

Es un apartado que sirve para dar ideas de cara a futuros TFGs/TFM.

Apéndice A

Manual de usuario

Índice de figuras

3.1. Arquitectura en árbol de Git	8
3.2. Funcionamiento básico de Github	9
3.3. Esquema de funcionamiento de <i>ELK Stack</i>	11
3.4. Ejemplo de dashboard	12
3.5. Esquema de funcionamiento de DockerHub	14
4.1. Esquema del entorno de trabajo SCRUM	18
4.2. Diagrama de bloques del proyecto	19
4.3. Esquema de funcionamiento de GrimoireLab	21
4.4. Logs mostrados tras el lanzamiento del contenedor de GrimoireLab	22
4.5. Ejemplo de dashboard ofrecido en <i>Overview</i>)	23
4.6. Ejemplo de configuración de índice	25
4.7. Listado de los índices recopilados hasta el momento	26
4.8. Ejemplo de query en Elasticsearch	27
4.9. Interfaz de Grafana (menús superiores)	28
4.10. Interfaz de Grafana (menús laterales)	30
4.11. Interfaz de Grafana (central) y ejemplo de dashboard	31
4.12. Ejemplo de query para panel de Grafana	32
4.13. Ejemplo de configuración visual de panel tipo <i>Graph</i>	33
4.14. Ejemplo final de <i>Graph Panel</i>	34
4.15. Ejemplo del plugin <i>Radar Graph</i>	37
4.16. Funcionamiento de docker con Dockerfile	39
4.17. Extracto de código agregado al nuevo Dockerfile	40
4.18. Extracto de código agregado al nuevo Entrypoint	41

4.19. Logs del nuevo contenedor de Grafana	42
4.20. Ejemplo de fichero de provisionamiento para importación de dashboards	43
4.21. Extracto de nuevo Dockerfile (utilizando método de exp/imp abrupta	45